Visual Studio 2005 Visual Basic

Visual Basic

Visual Basic 言語から発展した Microsoft Visual Basic 2005 は、型セーフなオブジェクト指向アプリケーションを生産的に構築できるよう作成されています。Visual Basic では、Windows アプリケーション、Web アプリケーション、およびモバイル デバイス アプリケーションを開発できます。Microsoft .NET Framework 向けのすべての言語と同様、Visual Basic で書かれたプログラムもセキュリティ機能および言語の相互運用性機能を利用できます。

この世代の Visual Basic でも、これまでに続いてすばやく簡単な方法で .NET Framework ベースのアプリケーションを作成できます。

このバージョンの Visual Basic では、エディット コンティニュのサポートが復活しています。また、アプリケーションを短時間で開発するための新しい機能が用意されています。このうちの 1 つに **My** と呼ばれる機能があります。この機能を使用することによって、.NET Framework の頻繁に使用されるタスクにすばやくアクセスできるだけでなく、アプリケーションおよびその実行時環境に関連する情報や既定のオブジェクトのインスタンスへのすばやいアクセスも可能になります。言語の新機能にはループの継続、保証されたリソースの破棄、演算子のオーバーロード、ジェネリック型、およびカスタム イベントなどがあります。また、Visual Basic では .NET Framework と共通言語ランタイム (CLR: Common Language Runtime) を完全に統合することによって、言語の相互運用性、ガベージ コレクション、セキュリティ機能拡張、およびバージョン管理のサポートを実現しています。

このセクションの内容

Visual Basic の概要

この製品の新機能および各種エディションで使用できる機能の一覧を示し、製品について簡単に説明します。

Visual Basic ガイド ツアー

Visual Basic を使用したプログラミングのさまざまな側面を順をおって説明します。

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic のバージョン 6.0 と現在のバージョンの違いについて説明します。

Visual Basic アプリケーションのアップグレード

Visual Basic の各種コードのアップグレード方法を示します。

Visual Basic でのアプリケーションの開発

Visual Basic での開発におけるさまざまな側面について、コード エディタ、セキュリティ、例外処理、デバッグ、および .NET Framework クラス ライブラリの利用などを説明します。

Visual Basic のプログラミング ガイド

オブジェクト指向プログラミング言語である Visual Basic に不可欠なコンポーネントを紹介します。

リファレンス (Visual Basic)

Visual Basic 言語およびコンパイラの情報について説明します。

Visual Basic のサンプル アプリケーション

サンプルに関する情報を提供します。

関連するセクション

Visual Studio の紹介

新機能、Visual Studio および .NET Framework の概要、および導入のヒントについて説明します。

Visual Studio の統合開発環境

アプリケーションのデザイン、開発、デバッグ、および配置に使用する共有ツールについて説明します。

Windows ベースのアプリケーション、コンポーネント、サービス

Windows アプリケーション、各種コンポーネント、XML Web サービス、Windows サービス、コンソール アプリケーション、および 64 ビットアプリケーションについて説明します。

Visual Studio での .NET Framework プログラミング

.NET Framework によるアプリケーション開発について説明します。

Visual Web Developer

Web アプリケーションの作成方法について説明します。

.NET Framework クラス ライブラリ リファレンス

Microsoft .NET Framework SDK に含まれるクラス、インターフェイス、および値型のライブラリを紹介します。

Visual Studio Tools for Office

Microsoft .NET Framework version 2.0 の生産性強化を活用して、Microsoft Office Word 2003、Microsoft Office Excel 2003、および Microsoft Office Outlook 2003 を拡張する方法について説明します。

スマート デバイス開発

Pocket PC や Smartphone など、Windows CE ベースのスマート デバイスで実行されるソフトウェアを開発する方法について説明します。

Visual Basic の概要

ここでは、Visual Basic によるアプリケーション開発を開始するための情報を提供します。

このセクションの内容

Visual Basic の新機能

Visual Basic の新機能を説明します。

Visual Basic のエディション

Visual Studio および .NET Framework の機能を説明します。

Visual Basic 言語のチュートリアル

Visual Basic 言語のさまざまな面を説明するヘルプ ページの一覧を示します。

Visual Basic プログラマのための追加リソース

一般的な問題を解決するのに役立つ Web サイトおよびニュースグループの一覧を示します。

Visual Basic での操作方法

重要なプログラミング テーマについての "方法" のヘルプ ページへのリンクをカテゴリに分類して示します。

関連するセクション

使用する技術とツールの決定

使用する方法やテクノロジを対話形式で選択できます。また、詳細情報へのリンクも用意されています。

Visual Basic におけるオブジェクト指向プログラミング

オブジェクト指向プログラミングの概要、独自のオブジェクトの作成方法、およびオブジェクトを使用してコーディングを簡潔にする方法を説明するページへのリンクを示します。

Visual Basic のサンプル アプリケーション

Visual Basic によるサンプル コードへのリンクを示します。

Visual Studio のヘルプの使い方

フィルタの使用方法について説明し、ヘルプの検索、キーワードの使用、および MSDN ライブラリの情報の検索方法に関するページへのリンクを示します。

開発環境のカスタマイズ

ショートカット キー、デバッグの環境設定など、統合開発環境 (IDE: Ingegrated Development Environment) のオプションを設定する方法について説明します。

製品のサポート

サポートの利用方法、およびフィードバックの送付方法について説明します。

Visual Studio

Visual Studio のドキュメントへのリンクを示します。

Visual Studio のエディション

Visual Studio の 3 種類のエディションについて、機能の一覧を示します。

Visual C#

Visual C# を使用したアプリケーション開発についてのドキュメントへのリンクを示します。

Visual C++

Visual C++ のドキュメントへのリンクを示します。

Visual J#

Visual J# を使用したアプリケーション開発についてのドキュメントへのリンクを示します。

Visual Studio Tools for Office

Microsoft Office と Visual Studio をビジネス アプリケーションの一部として使用する方法について説明します。

Visual Basic の新機能

このページでは、Visual Studio 2005 のこのリリースの Visual Basic で使用できる新機能と強化された機能を紹介します。各機能の詳細を参照するには、下のリンクをクリックしてください。

メモ：

Visual Basic 6.0 から Visual Basic 2005 に移行する場合は、新しいバージョンへの移行に役立つトピックがいくつかあります。詳細については、「Visual Basic 6.0 ユーザー向けの新機能」を参照してください。

Visual Basic のコンパイラと言語

Visual Basic 2005 の言語が改良された結果、ソース コードが簡略化されました。また、高度な機能を使用するコンポーネントと対話できるようになりました。詳細については、「Visual Basic 言語の新機能」および「Visual Basic コンパイラの新機能」を参照してください。

My を使用した Visual Basic での開発

Visual Basic 2005 には、生産性の向上を目的とする RAD (Rapid Application Development) のための新機能や、機能提供時の操作性のための新機能が用意されています。このうちの 1 つは **My** と呼ばれる機能であり、これは、一般的に使用されている .NET Framework の機能へのアクセスだけでなく、アプリケーションおよびその実行時環境に関連する情報や既定のオブジェクトのインスタンスへのアクセスも提供します。この情報は、IntelliSense を介して検出できる形式で構成されており、用途に応じて論理的に記述されます。詳細については、「My による開発」を参照してください。

アプリケーションの開発

My.Application オブジェクトはアプリケーションへのアクセスを提供します。これを使用して、アプリケーションを簡単にアップデートしたり、アプリケーションの情報をログに記録したりできます。また、**My.Application** を使用して、アプリケーションの開始時や終了時に、そのコマンド ライン引数や実行コードにアクセスすることもできます。詳細については、「アプリケーションの更新」、「アプリケーションからの情報のログ記録」、および「Visual Basic での実行中のアプリケーションへのアクセス」を参照してください。

クリップボードのトピック

My.Computer.Clipboard オブジェクトを使用すると、クリップボードからの読み取りやクリップボードへの書き込みを容易に行うことができます。クリップボードの中身を削除したり、クリップボードに格納されているデータの型を確認するためのメソッドもあります。詳細については、「クリップボードのデータの格納と読み込み」を参照してください。

コンピュータ リソース

コンピュータの時計、キーボード、マウス、ポート、およびレジストリなどの多くのコンピュータ リソースに、**My** オブジェクトを介してアクセスできます。詳細については、「コンピュータ リソースへのアクセス」を参照してください。

ファイルの入出力

My.Computer.FileSystem オブジェクトには、ファイル アクセスのためのメソッドおよびプロパティが用意されており、ファイル I/O が簡単になります。TextFieldParser オブジェクトを使用すると、固定幅またはコンマやタブで区切られた大きなテキスト ファイルを解析できます。**WriteAllText**、**WriteAllBytes**、**ReadAllText**、および **ReadAllBytes** などのメソッドを使用すると、ファイルの読み取りや書き取りを直観的に行うことができるうえ、パフォーマンスも向上します。また、**GetFileInfo**、**GetDirectoryInfo**、および **GetDriveInfo** の各メソッドを使用すると、ファイル、ディレクトリ、およびドライブに関する情報を入手できます。詳細については、「TextFieldParser オブジェクトによるテキスト ファイルの解析」を参照してください。

ネットワーク操作

My.Computer.Network オブジェクトには、ファイルのアップロードとダウンロード、接続状態の確認、およびリモート コンピュータが使用できるかどうかの確認のためのメソッドとプロパティが用意されています。詳細については、「ネットワーク操作の実行」を参照してください。

リソース

My.Resources オブジェクトを使用すると、オーディオ リソース、アイコン リソース、およびローカライズされたリソースなどにアクセスできます。詳細については、「アプリケーション リソースへのアクセス」を参照してください。

リソースは、プロジェクト デザイナの [リソース] ページを使用して追加および管理できます。詳細については、「[リソース] ページ (プロジェクト デザイナ)」を参照してください。

ユーザー設定とアプリケーション設定

My.Settings オブジェクトを使用すると、ユーザー設定の保持と変更、およびアプリケーション設定の読み取りを行うことができます。詳細については、「アプリケーション設定へのアクセス」を参照してください。

また、アプリケーション設定はプロジェクト デザイナの [設定] ページを使用して管理することもできます。詳細については、「[設定] ページ (プロジェクト デザイナ)」を参照してください。

データ アクセス

Visual Basic 2005 には、データにアクセスするアプリケーションを開発する際に役立ついくつかの新機能が用意されています。データ ソース構成ウィザードを使用すると、データベース、Web サービス、およびユーザーが作成したオブジェクトのデータにアプリケーションを簡単に接続できます。

新しい[データ ソース] ウィンドウでは、プロジェクトで利用できる関連データが 1 つの画面に表示され、ウィンドウからフォームに項目をドラッグしてデータ バインド コントロールを作成できるため、複雑なデータ バインド操作をより簡単に実行できます。

データセットへのデータの読み込み、クエリの実行、およびストアド プロシージャの実行は、新しい Visual Studio で生成された TableAdapter オブジェクトを使用して行うことができるようになりました。新しいローカル データ機能により、Microsoft Access データベース ファイルおよび Microsoft SQL Server Express データベース ファイルをアプリケーションに直接組み込むことができるようになりました。新機能の一覧については、「データの新機能」を参照してください。

デザイン時拡張機能

エディット コンティニュ

エディット コンティニュを使用すると、デバッガ環境でアプリケーションを実行しているとき、アプリケーションの停止や再起動を行わずに、アプリケーションに対する変更を行うことができます。中断モード時に行った変更は、アプリケーションの実行を継続すると即時に適用されます。

詳細については、「エディット コンティニュ (Visual Basic)」を参照してください。

[プロパティ] ウィンドウでの属性の編集

クラスおよびメソッドに適用される共通の属性は、コードに適用する代わりに、[プロパティ] ウィンドウで編集できるようになりました。

詳細については、「方法 : コード属性を編集する」を参照してください。

IntelliSense でのフィルタ処理

コード エディタで IntelliSense により表示される詳細レベルを調整できるようになりました。[IntelliSense] ウィンドウが表示されているときに、[よく使われる候補] タブをクリックすると、あまり一般的に使用されていないメンバが表示されないようにフィルタをかけることができます。[すべての候補] タブをクリックすると、利用できるすべてのメンバが表示されます。詳細レベルを調整することにより、必要に応じて、一般的でないまたは難解なメンバをフィルタにかけたり、表示したりできます。

詳細については、「Visual Basic でのフィルタ処理されたコンプリート リスト」を参照してください。

ゾーン内 IntelliSense

ゾーン内 IntelliSense は Visual Basic IntelliSense 機能の 1 つであり、指定されたセキュリティ ゾーンで実行するためのアクセス許可を持っていない項目があると、その項目がステートメント入力候補一覧内で別の色で表示されます。デザイン時サポートにより、部分信頼で実行されるアプリケーションの開発が容易になります。

詳細については、「Visual Basic 固有の IntelliSense」を参照してください。
「方法 : ClickOnce アプリケーションのセキュリティ ゾーンを設定する」も参照してください。

IntelliSense コード スニペット

IntelliSense コード スニペット ライブラリは、380 個の作成済みコードで構成されます。再利用できるルーチンであるこれらのコード スニペットは、自分のコードに追加し、コードに含まれているポインタを使用して編集できます。コード スニペットを使用するには、コード エディタの任意の場所を右クリックしてから、[スニペットの挿入] をクリックします。ショートカット キーを使用することもできます。

コード スニペットのタスクは、カスタム例外の作成から電子メール メッセージの送信、円の描画に至るまで多岐にわたっています。一般的なプログラミング構造の簡単な拡張であるコード スニペットもあります。このライブラリは拡張可能です。ユーザーのビジネス ニーズに合ったコード タスクを作成して、ライブラリに追加できます。また、サードパーティのソフトウェア ベンダや Visual Basic コミュニティ サイトからライブラリ タスクをダウンロードすることもできます。

詳細については、「Visual Basic の IntelliSense コード スニペット」および「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

エラー修正と警告

コード エディタのスマート コンパイル エラー修正を使用すると、一般的なエラーや警告に対する解決策が表示されるので、適切な修正を選択して自分のコードに適用できます。エラーの発生時、波線の右側の下に記号が表示されている場合は、マウス カーソルを波線の上に置くと記号がスマート タグ パネルに変わります。スマート タグ パネルをクリックすると、[エラー修正のオプション] ヘルパー ウィンドウが表示され、エラーに関する説明や、エラーの修正方法の候補が示されます。また、場合によってはこのウィンドウで修正のプレビューを行うこともできます。

詳細については、「スマート コンパイル自動修正」および「Visual Basic での警告の構成」を参照してください。

例外処理アシスタント

アプリケーションのデバッグ時に、既定では例外処理アシスタントが起動し、未処理の例外を処理する手助けをします。必要な場合は、自分のコード内でなんらかの例外が発生したときにアシスタントが表示されるように、環境を変更できます。例外処理アシスタント リストには、例外の種類、スローされた例外に固有のメッセージ、トラブルシューティングのためのヒントの一覧、および適用される操作が表示されます。

詳細については、「方法 : 例外処理アシスタント を使用してランタイム エラーを修正する」を参照してください。

XML ドキュメント

XML コード ドキュメントを使用すると、コード内のクラスを文書化し、そのドキュメントを XML として公開できます。結果として生成された XML ドキュメントはさまざまな方法で使用したり表示したりでき、コードを記述しながらドキュメントを作成できます。

詳細については、「方法 : Visual Basic で XML ドキュメントを作成する」を参照してください。

[ドキュメント アウトライン] ウィンドウ

[ドキュメント アウトライン] ウィンドウでは、ASP.NET Web ページおよび HTML ページに加えて、Windows フォームのアウトライン ビューもサポートされるようになりました。[ドキュメント アウトライン] ウィンドウを使用して、エディタの [デザイン] ビューにいながら、Windows フォームのコントロールの間を移動できます。このウィンドウにアクセスするには、[表示] メニューの [その他のウィンドウ] をクリックし、[ドキュメント アウトライン] をクリックします。詳細については、「[ドキュメント アウトライン] ウィンドウ」を参照してください。

プロジェクト デザイナによるプロジェクト、設定、およびリソースの管理

プロジェクト デザイナは、プロジェクトのプロパティ、設定、およびリソースを管理するための中心点として機能します。これには、[プロジェクト] メニューの [プロパティ] からアクセスできます。

詳細については、「プロジェクト デザイナの概要」を参照してください。

デザイナの設定

プロジェクト デザイナの [設定] ページを使用すると、プロジェクトのアプリケーション設定を指定できます。アプリケーション設定はユーザー スコープでもアプリケーション スコープでもかまいません。この機能は、動的プロパティに置き換わるものです。詳細については、「[設定] ページ (プロジェクト デザイナ)」を参照してください。

リソース デザイナ

リソース デザイナは、プロジェクトが使用するリソース (文字列、イメージ、アイコン、オーディオ、ファイルなど) を管理するためのユーザー インターフェイス ツールです。実行時に My.Resources オブジェクトを使用してアクセスできる、厳密に型指定されたリソースを作成します。**My.Resources** オブジェクトを使用したリソースへのアクセスの詳細については、「アプリケーション リソースへのアクセス」を参照してください。

プロジェクト デザイナの [リソース] ページは、リソースを 1 つの場所 (Resources.resx) に格納して管理するリソース デザイナのインスタンスをホストします。詳細については、「[リソース] ページ (プロジェクト デザイナ)」を参照してください。

ClickOnce の配置

ClickOnce の配置により、自動的に更新される Windows ベース アプリケーションおよびコンソール アプリケーションを発行できます。このアプリケーションは、Web アプリケーションと同じように簡単にインストール、更新、および実行できます。詳細については、「ClickOnce の配置」を参照してください。

プロジェクト デザイナに新たに追加された [セキュリティ]、[署名]、および [発行] の各タブを使用すると、ClickOnce の配置をカスタマイズできます。[ビルド] に新たに追加された [発行] をクリックするか、ソリューション エクスプローラのショートカット メニューを使用して、発行ウィザードにアクセスできます。このウィザードでは、アプリケーションを発行するために必要な手順を実行します。詳細については、「[発行] ページ (プロジェクト デザイナ)」、「[署名] ページ (プロジェクト デザイナ)」、および「[セキュリティ] ページ (プロジェクト デザイナ)」を参照してください。

64 ビット アプリケーションの配置

ClickOnce と Windows インストーラの配置は、どちらも 64 ビット プラットフォームへのインストールをサポートします。詳細については、「64 ビット アプリケーションの配置」を参照してください。

サンプルの追加

Visual Basic 2005 には、60 種以上のサンプル アプリケーションが同梱されています。扱っている新機能には、次の機能が含まれます。

- ClickOnce の配置
- **My** オブジェクトを使用したプログラミング
- 言語拡張 (ジェネリックおよび XML コメントを含む)

- データ アクセス

詳細については、「Visual Basic のサンプル アプリケーション」を参照してください。

Visual Basic 6.0 からのアップグレード

Visual Basic 2005 のアップグレード ツールにさまざまな拡張が行われました。これには、多くの Visual Basic 6.0 ActiveX コントロールを Visual Basic 2005 の対応する機能にアップグレードする機能が含まれます。

詳細については、「アップグレードの新機能」を参照してください。

参照

概念

Visual Studio 2005 の新機能
Windows フォームと Windows フォーム コントロールの新機能
ASP.NET の新機能
Web プロジェクトの新機能
Visual Studio の Web 開発の新機能
配置の新機能

Visual Basic 言語の新機能

Visual Basic 2005 で導入された新しい言語機能には、ループの継続、リソース破棄の保証、アクセス プロパティの混在、符号なしデータ型と null 許容型、演算子のオーバーロード、部分型とジェネリック型、カスタム イベント、共通言語仕様 (CLS) 準拠のチェックなどがあります。

ここでは、Visual Basic 2005 で Visual Basic 言語に追加された機能について説明します。以前のバージョンから変更された特徴や機能の説明については、「言語の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Continue ステートメント

Visual Basic に追加された **Continue** ステートメントは、**Do** ループ、**For** ループ、または **While** ループから次の反復にすぐにスキップできます。詳細については、「方法 : ループの次の反復処理にスキップする」および「Continue ステートメント (Visual Basic)」を参照してください。

Visual Basic 6.0 フォーム アクセス

Visual Basic では、`Form1` などの定義済みのフォームをクラス名で参照できるようになり、クラスのインスタンスを明示的に作成する必要はなくなりました。詳細については、「方法 : フォームにアクセスする」を参照してください。

IsNot 演算子

Visual Basic で追加された **IsNot** 演算子を使用すると、**Not** 演算子と **Is** 演算子を不恰好に並べて使用する必要はありません。詳細については、「方法 : 2 つのオブジェクトが等しいかどうかをテストする」および「IsNot 演算子」を参照してください。

TryCast 演算子

Visual Basic に **TryCast** 型変換演算子が新たに追加されました。InvalidCastException エラーをスローする **CType** や **DirectCast** とは異なり、この演算子は変換に失敗した場合、**Nothing** を返します。詳細については、「TryCast」を参照してください。

using ステートメント

Visual Basic では、新しい **Using...End Using** ブロックを使って、コードの実行がなんらかの理由でそのブロックから出るときにシステム リソースが確実に破棄されるようにすることができます。詳細については、「方法 : システム リソースを破棄する」および「Using ステートメント (Visual Basic)」を参照してください。

配列の下限値 0 の明示

Visual Basic では、配列の宣言において各次元の上限値と共に下限値 (0) を指定できるようになりました。詳細については、「方法 : 配列の下限に 0 を指定する」を参照してください。

アクセス レベルの混在するプロパティ

Visual Basic では、プロパティを宣言するときに、そのプロパティとは異なるアクセス レベルを **Get** プロシージャおよび **Set** プロシージャに指定できるようになりました。詳細については、「方法 : 複数のアクセス レベルを持つプロパティを宣言する」を参照してください。

unsigned 型

Visual Basic では、符号なし整数データ型 (**UShort**、**UInteger**、および **ULong**)、および signed 型 (**SByte**) をサポートしています。詳細については、「方法 : 符号なしの型を使用する Windows の機能を呼び出す」、「方法 : unsigned 型を使用して正の整数のストレージを最適化する」、「UInteger データ型」、「ULong データ型 (Visual Basic)」、「UShort 型 (Visual Basic)」、および「SByte 型 (Visual Basic)」を参照してください。

null 許容型

Visual Basic では、通常値または null 値を設定できるように値型を拡張できるようになりました。null 値は、情報が現在使用できないため、定義済みの値が変数に格納されていないことを示すには便利です。詳細については、「値型と参照型」および「定義された値を持たない可能性のある値型」を参照してください。

演算子のオーバーロード

Visual Basic では、独自に定義したクラスまたは構造体に標準の演算子 (**+**、**&**、**Not**、または **Mod** など) を定義することができます。詳細については、「方法 : 演算子を定義するクラスを使用する」、「方法 : 演算子を定義する」、および「Operator ステートメント」を参照してください。

部分型を使ったコードの分割

Visual Basic は、作成したコードから生成されたコードを複数のソース ファイルに分割する機能を統合開発環境 (IDE) に提供します。作業のほとんどでは、自分で作成したコードだけを扱います。詳細については、「Partial (Visual Basic)」を参照してください。

ジェネリック型

Visual Basic では、ジェネリックなクラス、構造体、インターフェイス、プロシージャ、およびデリゲートで型パラメータがサポートされています。対応する型引数が、ジェネリック型の 1 つの要素のデータ型をコンパイル時に指定します。詳細については、「Visual Basic におけるジェネリック型」を参照してください。

カスタム イベント

Visual Basic では、イベントのさまざまな動作を細かく制御できるようになりました。**Custom** キーワードを **Event** ステートメントの修飾子として使用して、カスタム イベントを宣言できます。カスタム イベントでは、コードでイベント ハンドラを追加または削除するときに、つまりコードでイベントを生成するときに、何が起こるかを正確に指定します。「Event ステートメント」、「方法 : メモリの使用量を節約するイベントを宣言する」、および「方法 : ブロックを回避するイベントを宣言する」の例を参照してください。既存のコードとの下位互換性を維持するため、**Custom** キーワードは予約されていません。

コンパイラ チェック オプション

Visual Basic 2005 では、新しいコンパイラ チェック オプションが導入されています。/nowarn オプションおよび /warnaserror (Visual Basic) オプションを使って、警告の扱い方を細かく制御できます。これらのコンパイラ オプションでは、警告 ID のリストを省略可能なパラメータとして渡して、どの警告をオプションに適用するかを指定できます。

- CLS 準拠チェック

Visual Basic では、共通言語仕様 (CLS) でサポートされない仕様または操作が含まれるコード行に警告が生成されます。

- 初期化されない変数のチェック

Visual Basic では、初期化されない可能性がある変数に警告が生成されます。値を代入しないで変数を使用する可能性がある実行パスが少なくとも 1 つある場合、その変数にこのステータスが与えられます。

詳細については、「Visual Basic での警告の構成」を参照してください。

参照

関連項目

データ型の概要

概念

Visual Studio 2005 の新機能
CLS 準拠コードの記述
オブジェクトの有効期間 : オブジェクトの作成と破棄

その他の技術情報

Visual Basic リファレンス
Visual Basic での例外およびエラー処理

Visual Basic コンパイラの新機能

Visual Basic 2005 には、新しいコンパイラ オプションといくつかの変更が加えられています。

条件付きコンパイル

Visual Basic では、すべてのプロジェクトに対して次の新しい条件付きコンパイル定数が追加されています。

- **TRACE**。ブール値の定数です。この定数を指定すると、**Trace** クラスのメソッドが出力ウィンドウに出力を生成します。
- **VBC_VER**。Visual Basic のバージョンを *major.minor* の形式で表します。Visual Basic 2005 の場合は、**VBC_VER** の値は 8.0 になります。

詳細については、「条件付きコンパイル定数」を参照してください。

Vbc.rsp 内の既定のコマンド ライン オプション

Visual Basic のコマンド ライン コンパイラは、既定のコマンド ライン オプションを、コマンド ライン コンパイラ (Vbc.exe) と同じディレクトリ内にある Vbc.rsp ファイル内に保存しています。Vbc.rsp を編集すると、既定のコマンド ライン オプションを再構成できます。

新しいコマンド ライン オプション

次の 7 つのコマンド ライン コンパイラ オプションが新たに追加されました。

- /codepage オプションは、ソース ファイルを開くときにどのコード ページを使用するかを指定します。
- /doc オプションを指定すると、コード内のコメントに基づいて XML ドキュメント ファイルが生成されます。
- /errorreport オプションは、Visual Basic 内部コンパイル エラーを Microsoft に報告するときの便利な方法として機能します。
- /filealign オプションは、出力ファイル内のセクションのサイズを指定します。
- /noconfig オプションは、よく使用される .NET Framework アセンブリを自動的に参照したり、**System** および **Microsoft.VisualBasic** 名前空間をインポートしたりしないようコンパイラに指示します。
- /nostdlib オプションは、System.dll アセンブリとその他のよく使用されるアセンブリに対する自動参照を削除します。
- /platform オプションは、出力ファイルの対象となるプロセッサを指定します。対象プラットフォームを明示的に指定する必要がある場合に使用します。

/nowarn および /warnaserror

/nowarn および /warnaserror オプションを使用すると、警告の処理方法を細かく制御できます。これらのコンパイラ オプションでは、警告 ID の一覧を省略可能なパラメータとして指定することで、どの警告に対してオプションを提供するかを指示します。

参照

概念

Visual Basic 言語の新機能

Visual Basic のエディション

ここでは、Visual Basic に含まれる機能の概要を示します。

Visual Basic プロジェクト テンプレート

新しいプロジェクトの作成に最適な方法は、Visual Basic のテンプレート (アプリケーション ウィザード) を使用する方法です。プロジェクト テンプレートは、アプリケーション フレームワークやライブラリと連携して機能し、初期プログラムを作成します。

Visual Studio のさまざまなエディションで使用できる Visual Basic プロジェクト テンプレートを次の表に示します。

テンプレート	Microsoft Visual Basic 2005 Express Edition	Visual Studio 2005 Standard	Visual Studio 2005 Professional 以上
ASP.NET Web サイト		X	X
ASP.NET Web サービス		X	X
ASP.NET Crystal Reports Web サイト			X
クラス ライブラリ	X	X	X
コンソール アプリケーション	X	X	X
Crystal Reports アプリケーション			X
デバイス アプリケーション		X	X
空のプロジェクト	X	X	X
空の Web サイト		X	X
Excel テンプレート			X
Excel ブック			X
ムービー コレクション スタート キット	X	X	X
Outlook アドイン			X
パーソナル Web サイト スタート キット		X	X
Pocket PC 2003: クラス ライブラリ		X	X
Pocket PC 2003: クラス ライブラリ (1.0)		X	X
Pocket PC 2003: コンソール アプリケーション		X	X
Pocket PC 2003: コンソール アプリケーション (1.0)		X	X
Pocket PC 2003: コントロール ライブラリ		X	X

Pocket PC 2003: デバイス アプリケーション		X	X
Pocket PC 2003: デバイス アプリケーション (1.0)		X	X
Pocket PC 2003: 空のプロジェクト		X	X
Pocket PC 2003: 空のプロジェクト (1.0)		X	X
スクリーン セーバー スタート キット	X	X	X
SQL Server プロジェクト			X
Smartphone 2003: クラス ライブラリ (1.0)		X	X
Smartphone 2003: コンソール アプリケーション (1.0)		X	X
Smartphone 2003: デバイス アプリケーション (1.0)		X	X
Smartphone 2003: 空のプロジェクト (1.0)		X	X
テスト プロジェクト			X
Web コントロール ライブラリ		X	X
Windows アプリケーション	X	X	X
Windows CE 5.0: クラス ライブラリ		X	X
Windows CE 5.0: コンソール アプリケーション		X	X
Windows CE 5.0: コントロール ライブラリ		X	X
Windows CE 5.0: デバイス アプリケーション		X	X
Windows CE 5.0: 空のプロジェクト		X	X
Windows コントロール ライブラリ		X	X
Windows サービス			X
Word ドキュメント			X
Word テンプレート			X

使用可能なプロジェクト テンプレートとアイテムの詳細については、「Visual Studio の既定のプロジェクト テンプレート」を参照してください。ファイ

ルの種類の詳細については、「Visual Basic、Visual C#、および Visual J# のファイルの種類とファイル名の拡張子」を参照してください。

ビジュアル ツール

Visual Basic には、次のデザイナが組み込まれています。

機能	説明	Microsoft Visual Basic 2005 Express Edition	Visual Studio 2005 Standard	Visual Studio 2005 Professional 以上
Web フォーム デザイナ	ASP.NET の Web フォーム ページとコントロールを処理するためのビジュアル ツールを提供します。			X
Windows フォーム デザイナ	Windows のフォームとコントロールを処理するためのビジュアル ツールを提供します。	X	X	X
コンポーネント デザイナ	ビジュアルでないコンポーネントを処理するためのビジュアル ツールを提供します。			X
XML スキーマ デザイナ (XML デザイナ)	XML スキーマ、ADO.NET データセット、および XML ドキュメントを使用するためのビジュアルなツールのセットが用意されています。			X
Visual Database Tools	データベースを操作するためのツール セットを提供します。	X	X	X
クラス ビュー	開発中のアプリケーションで定義、参照、または呼び出されるシンボルを表示します。		X	X
Crystal Reports	Windows アプリケーションと Web アプリケーション用の、レポートを作成するためのツールを提供します。			X
レポート ビューア	カスタム アプリケーション用のレポート ツールを提供します。		X	

Visual Studio の各種エディションに含まれるすべての機能の一覧については、「Visual Studio のエディション」を参照してください。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

関連項目

Visual Basic の設定

概念

インストールとセットアップの基本事項

Microsoft Visual Database Tools と Visual Studio のエディション

Visual Basic Express のプロジェクトの種類

その他の技術情報

Visual Studio のエディション

Visual Basic 言語のチュートリアル

チュートリアルでは、一般的なシナリオについて手順に沿った指示が用意されており、製品や特定の機能を知るための糸口となります。

このセクションの内容

チュートリアル : イベントの宣言と発生

Visual Basic におけるイベントの宣言方法および生成方法について説明します。

イベントの処理

標準の **WithEvents** キーワードまたは新規の **AddHandler**/**RemoveHandler** キーワードを使用してイベントを処理する方法を示します。

チュートリアル : クラスの定義

クラスとそのフィールド、プロパティ、メソッド、およびイベントを宣言する方法について説明します。

チュートリアル : Windows API の呼び出し

Declare ステートメントの使用方法と Windows API の呼び出し方法について説明します。属性を使って API 呼び出しのマーシャリングを制御する方法と、API 呼び出しをクラスのメソッドとして公開する方法についても説明します。

チュートリアル : インターフェイスの作成と実装

Visual Basic におけるインターフェイスの宣言方法および実装方法について説明します。

マルチスレッド

ある語が出現するかどうかについてテキスト ファイルを検索するマルチスレッド アプリケーションの作成方法を示します。

構造化例外処理

包括的なエラー処理を含むプログラムの作成および保守の方法と、**Try...Catch...Finally** 構文の使い方を紹介します。

チュートリアル : Visual Basic 2005 での COM オブジェクトの作成

Visual Basic での COM オブジェクトの作成について、COM クラス テンプレートを使用する方法と使用しない方法を紹介します。

チュートリアル : COM オブジェクトによる継承の実装

Visual Basic 6.0 でクラスを含む COM オブジェクトを作成してから、Visual Basic でそのオブジェクトを基本クラスとして使用する方法について説明します。

チュートリアル : Visual Basic によるファイルとディレクトリの操作

Visual Basic の関数を使用して、ファイルに関する情報を確認する方法、ファイル内で文字列を検索する方法、およびファイルへの書き込みを行う方法について説明します。

チュートリアル : .NET Framework のメソッドによるファイル操作

.NET Framework のメソッドを使用して、ファイルに関する情報を確認する方法、ファイル内で文字列を検索する方法、およびファイルへの書き込みを行う方法について説明します。

関連するセクション

Visual Basic のチュートリアル

分散アプリケーション、データ、Web サービス、Windows フォーム、Web フォーム、コンポーネントとコントロールの作成、フレームワーク サービス、アップグレード、国際化、およびユーザー補助について説明する、Visual Basic の各チュートリアルへのリンクを示します。

Visual Basic プログラマのための追加リソース

一般的な問題に対する回答を見つけるのに役立つサイトやニュースグループを次に示します。

Microsoft のリソース

Web 上

Microsoft Help and Support

サポート技術情報 (KB: Knowledge Base) の文書、ダウンロードや更新用のファイル、Support Webcast、およびその他のサービスへのアクセスを提供します。

Microsoft Visual Basic ディベロッパー センター

コード サンプル、アップグレード情報、無償のダウンロード、および技術的な情報が用意されています。

Microsoft Visual Basic Team Blog

Visual Basic チーム ブログへのアクセスを提供します。

Microsoft ASP.NET

Visual Basic での Web 開発のための文書、デモンストレーション、ツールのプレビュー、およびその他の情報を提供します。

GotDotNet for Visual Basic

Visual Basic の開発者に役立つ文書、サンプル、およびその他の情報が含まれています。

チャット、フォーラム、ディスカッション グループ

MSDN Forums

Visual Basic、.NET Framework など多数の Microsoft テクノロジに関する Web ベースのディスカッション フォーラムを提供します。

MSDN Discussion Groups

ニュースグループを提供します。世界中の技術者と情報をやり取りできます。

MSDN Chats

Microsoft 製品やテクノロジに関するディスカッションを提供します。各チャットは 1 名以上の Microsoft エキスパートがホストしています。既に終了しているチャットについてはトランスクリプトを利用できます。

ビデオとウェブキャスト

Visual Basic at the Movies

Visual Basic の初心者から経験豊富な開発者まですべてのユーザーを対象に、Visual Basic の新しいスキルを紹介する 101 個のショートフィルムを公開しています。

Channel 9

ビデオ、Wikis、およびフォーラムを通じて継続的なコミュニティを提供します。

サードパーティのリソース

MSDN の Web サイトでは、Visual Basic に関する最新のサードパーティのサイトやニュースグループの情報が提供されています。使用できるリソースの最新の一覧については、「MSDN Visual Basic Community」を参照してください。

Web 上

DevX Visual Basic Zone

Visual Basic 2005 に移行する今日の Visual Basic 開発者のために、詳細な技術文書を提供します。

devCity.NET

devCity.NET の会員になると、Microsoft Visual Basic および ASP (Active Server Pages) 関連の文書に加えて、ソース コード、チュートリアル、およびユーザーから提供されたコンテンツなど、5,000 を超えるコンテンツにアクセスできます。

参照

その他の技術情報

Visual Basic の概要
他の開発者との連携
製品のサポートとユーザー補助

Visual Basic での操作方法

「操作方法」のページには、Visual Basic のプログラミングおよびアプリケーション開発に関連した主要なタスク ベースのトピックが紹介されています。ここでは、Visual Basic を使用して実行できる機能の主なカテゴリを紹介します。該当するリンクをクリックすると、重要な手順について解説されたヘルプ ページに移動できます。

Visual Basic 言語の習得 (Visual Basic での操作方法)

オブジェクト指向プログラミング、制御フローの管理、変数の操作、エラーと例外の処理などについて説明します。

Visual Basic 6 のユーザー向けアップグレード (Visual Basic の操作方法)

Visual Basic 6 アプリケーションのアップグレード、.NET Framework などについて説明します。

データ アクセス (Visual Basic での操作方法)

概要、データの読み込み、データの検証などについて説明します。

配置 (Visual Basic での操作方法)

ClickOnce による配置、セットアップ プロジェクトの使用、その他の配置プロジェクトの使用などについて説明します。

Windows アプリケーション (Visual Basic での操作方法)

フォームとコントロール、データ、印刷などについて説明します。

Web アプリケーション (Visual Basic での操作方法)

Web サイトの開発、ユーザーの Web フォームとコントロール、Web サイトの配置などについて説明します。

Web サービス (Visual Basic での操作方法)

Web サービスの作成、Web サービスの呼び出し、Web サービスのデバッグなどについて説明します。

Office プログラミング (Visual Basic での操作方法)

Excel アプリケーション、Word アプリケーションなどについて説明します。

スマート デバイス アプリケーション (Visual Basic での操作方法)

データの操作、コントロールの操作、スマート デバイス アプリケーションの配置などについて説明します。

セキュリティ (Visual Basic での操作方法)

Web アプリケーション、Windows アプリケーション、コンポーネントなどについて説明します。

デバッグとテスト (Visual Basic での操作方法)

アプリケーションのデバッグ、アプリケーションの監視などについて説明します。

Visual Basic 6 のユーザー向けアップグレード (Visual Basic の操作方法)

ここでは、Visual Basic 6 を使用していたユーザーが Visual Basic .NET を使い始める際に役立つヘルプへのリンクを示します。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic 6.0 からの変更点を説明します。

Visual Basic 言語の新機能

新しい言語機能と言語機能 (継承、インターフェイス、オーバーライド、共有メンバ、オーバーロードなど) について説明します。

チュートリアル : Visual Basic 6.0 アプリケーションから現在のバージョンの Visual Basic へのアップグレード

Visual Basic のグラフィックス サンプル アプリケーションを Visual Basic .NET にアップグレードする方法について説明します。アップグレード プロセスの説明に加えて、グラフィック アーキテクチャにおける Visual Basic 6.0 と Visual Basic .NET の違いについて説明します。

Visual Basic 6.0 アプリケーションのアップグレードの準備

多少の準備を行うことにより、アップグレードを最適化するためにできるいくつかのことについて、詳しく説明します。

ここに示す推奨事項に従うことにより、既存のプロジェクトを Visual Basic .NET にアップグレードした後に必要な変更作業を最小限に抑えることできます。また、変更作業を完全に回避できる場合もあります。

アップグレードを行うにあたっての注意事項

旧バージョンとの互換性が維持されていない部分について説明します。

My による開発

Visual Basic の機能の 1 つである、アプリケーションおよびその実行時環境に関連する情報や既定のオブジェクトのインスタンスへのアクセスを提供する機能について説明します。

Visual Basic プログラマのための追加リソース

Visual Basic に関するさまざまな Microsoft およびサードパーティのサイトに関する情報を提供します。

Visual Basic 6 アプリケーションのアップグレード

Visual Basic 6.0 と現在のバージョンの Visual Basic の両方を使用する

Visual Basic 6.0 からの変更点を説明します。

チュートリアル : Visual Basic 6.0 アプリケーションから現在のバージョンの Visual Basic へのアップグレード

Visual Basic のグラフィックス サンプル アプリケーションを Visual Basic .NET にアップグレードする方法について説明します。アップグレード プロセスの説明に加えて、グラフィック アーキテクチャにおける Visual Basic 6.0 と Visual Basic .NET の違いについて説明します。

Visual Basic 6.0 からのアップグレード

アプリケーションをアップグレードする方法と、Visual Basic 2005 の変更点をすばやく理解する方法を示す、ヘルプ トピックへのリンクを提供します。

方法 : Visual Basic アップグレード ウィザードを使ってプロジェクトをアップグレードする

使いやすいウィザードを用いた Visual Basic 6 プロジェクトのアップグレードについて説明します。

Visual Basic 6.0 アプリケーションのアップグレードの準備

多少の準備を行うことにより、アップグレードを最適化するためにできるいくつかのことについて、詳しく説明します。

ここに示す推奨事項に従うことにより、既存のプロジェクトを Visual Basic .NET にアップグレードした後に必要な変更作業を最小限に抑えることできます。また、変更作業を完全に回避できる場合もあります。

アップグレードを行うにあたっての注意事項

旧バージョンとの互換性が維持されていない部分について説明します。

COM 相互運用

Visual Basic アプリケーションにおける COM 相互運用について説明します。

Visual Basic プログラマのための追加リソース

Visual Basic に関するさまざまな Microsoft およびサードパーティのサイトに関する情報を提供します。

データの処理 (Visual Basic 6 ユーザー向け)
データ アクセス (Visual Basic 6.0 ユーザー向け)

ADO および ADO.NET 間の、概念的およびタスクの観点におけるいくつかの相違点について説明します。

ADO.NET と ADO の比較

ADO の知識があるユーザー向けに ADO.NET について説明します。

.NET Framework について
.NET Framework 概念の概要

共通言語ランタイムおよび .NET Framework クラス ライブラリの機能について説明します。

Visual Basic と .NET Framework

Visual Basic の観点からの .NET プログラミングの基礎について説明します。

Visual Studio の .NET Framework クラス ライブラリの概要

名前空間とアセンブリの使用方法を説明します。

VBRun

Visual Basic 6.0 Resource Centerダウンロード、文書、ニュースグループ、ユーザー グループ、その他のリソースへのリンクを提供します。
http://msdn.microsoft.com/vbrun/

Visual Basic プログラマのための追加リソース

Visual Basic に関するさまざまな Microsoft およびサードパーティのサイトに関する情報を提供します。

参照
概念
Visual Basic での操作方法

Visual Basic 言語の習得 (Visual Basic での操作方法)

ここでは、Visual Basic のプログラミングに関連するタスク全般について扱ったヘルプへのリンクを紹介しています。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

Visual Basic の新機能

Visual Studio 2005 のこのリリースの Visual Basic で使用できる新機能と強化された機能を紹介します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 以降、Visual Basic 言語がどのように変更されたかを、詳細へのリンクと共に示します。

Visual Basic 2005 におけるオブジェクト指向プログラミング

継承を使用する状況

インターフェイスではなく継承を使用する場合について説明します。

継承の基本

継承の修飾子、メソッドとプロパティのオーバーライド、**MyClass**、および **MyBase** について説明します。

方法 : 演算子を定義する

オペランドの 1 つまたは両方がユーザーのクラスまたは構造体の型である場合の標準の演算子 (*、<>、**And** など) の動作を定義する方法について説明します。

方法 : オブジェクトのメンバにアクセスする

作成したオブジェクトのメンバの読み取り、書き込み、呼び出し方法について説明します。

方法 : プロシージャの複数のバージョンを定義する

オーバーロードを使用して、1 つのプロシージャを複数のバージョンで定義する方法について説明します。各バージョンは同じ名前を使用しますが、パラメータ リストが異なります。

プロパティの操作

方法 : プロパティに値を格納する

プロパティへの値の格納方法について説明します。

方法 : プロパティから値を取得する

プロパティに格納された値の取得方法について説明します。

方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)

名前を使わずにアクセスできるプロパティの、宣言および呼び出しの方法について説明します。

方法 : 複数のアクセス レベルを持つプロパティを宣言する

格納と取得に関して異なるアクセス レベルを持つプロパティの宣言について説明します。

方法 : プロパティを作成する

プロパティの作成方法について説明します。

継承の使用

方法 : 既存クラスのメンバを使用するクラスを定義する.

別のクラスから派生するクラスを作成するコード例を紹介します。

方法 : 継承された変数を隠す

Shadows キーワードを使用して、継承した変数を隠ぺいする方法について説明します。

方法 : 派生クラスによって非表示になっている変数にアクセスする

派生クラスで隠ぺいされた基本クラスの変数にアクセスする方法を説明します。

プロシージャの操作

方法 : プロシージャを作成する

コード内の複数の場所で必要になるタスクを実行するプロシージャの宣言の方法について説明します。

方法 : 値を返すプロシージャを呼び出す

Function プロシージャを呼び出して、そこから返される値を使用するための 2 つの方法を説明します。

方法 : 値を返さないプロシージャを呼び出す

タスクを実行するが、呼び出し元のコードに値を返さない **Sub** プロシージャを呼び出す方法について説明します。

方法 : プロシージャにパラメータを定義する

呼び出し元のコードによってプロシージャに渡された値を受け取るための、パラメータ リストの定義の方法について説明します。

方法 : プロシージャに引数を渡す

プロシージャの各パラメータに引数を指定して、プロシージャに値を渡す方法を説明します。

プログラム制御フローの管理

方法 : 長いコードを複数のコードに分割する.

プロシージャを使用して、構造化プログラミングを実現したコードを作成する方法について説明します。

方法 : コード内でステートメントを分割および連結する

1 つのステートメントを複数のソース コード行に分割したり、複数のステートメントを組み合わせて 1 つのソース行にしたりする方法を説明します。

方法 : コードにコメントを追加する (Visual Basic)

ソース コードに説明のコメントを置くための 2 つの方法について説明します。

条件付きステートメント

方法 : 1 つ以上の条件に基づいて、ステートメントを実行する

If...**Then**...**Else** の構文を使用し、条件の変化に応じて異なるステートメント ブロックを実行する方法について説明します。

関数とサブルーチン

方法 : プロシージャを作成する

コード内の複数の場所で必要になるタスクを実行するプロシージャの宣言の方法について説明します。

方法 : 値を返すプロシージャを呼び出す

Function プロシージャを呼び出して、そこから返される値を使用するための 2 つの方法を説明します。

方法 : 値を返さないプロシージャを呼び出す

タスクを実行するが、呼び出し元のコードに値を返さない **Sub** プロシージャを呼び出す方法について説明します。

方法 : プロシージャにパラメータを定義する

呼び出し元のコードによってプロシージャに渡された値を受け取るための、パラメータ リストの定義の方法について説明します。

方法 : プロシージャに引数を渡す

プロシージャの各パラメータに引数を指定して、プロシージャに値を渡す方法を説明します。

ループ ステートメント

方法 : ループの次の反復処理にスキップする

ループの現在の反復処理をスキップして、次の反復処理に進む方法について説明します。

方法 : 複数のステートメントを繰り返し実行する

ステートメント ブロックを設定された回数だけ実行する方法、または条件に一致するまで実行する方法を説明します。

方法 : Visual Basic でコレクションを反復処理する

コレクションの各要素に対して同じステートメントを実行する **For Each**...**Next** ループの使用方法を説明します。

方法 : ループのパフォーマンスを改善する

ループの制御変数に、最も効果的なデータ型を選択することについて説明します。

コレクション、ジェネリック、および配列の操作

方法 : クラス内にコレクションを定義する

Visual Basic の簡単な **Collection** オブジェクトを定義および使用する方法を説明します。

コレクションによるオブジェクトの管理

関連するオブジェクトをグループ化するための、コレクションの型を選択する際に考慮する基準について説明します。

方法 : コレクションの項目を追加、削除、および取得する

新しい要素の追加、既存の要素の削除、および要素の取得という基本的な操作について説明します。

方法 : オブジェクトのコレクションを作成する

Visual Basic のコレクションとジェネリック コレクションについて、それらの作成方法と反復処理の方法を説明します。

コレクション

方法 : オブジェクトの配列を作成する

関連するオブジェクトをグループ化するためのもう 1 つの方法である配列について説明します。

コレクションのトラブルシューティング

コレクションに伴う一般的な問題を示し、それらの対処方法を説明します。

ジェネリック

方法 : ジェネリック クラスを使用する

1 つ以上の型パラメータを受け取るクラスを使用する方法を示します。

方法 : プロシージャまたはプロパティに配列を渡す

引数リストを使用して、値の配列をプロシージャまたはプロパティに渡す方法を説明します。

方法 : 複数のデータ型に同一の機能を提供できるクラスを定義する

クラスを 1 つ定義して、データ型が異なっても同じ機能を実行できるようにする方法を説明します。

配列

方法 : 配列を作成する

配列オブジェクトを作成して、それを配列変数に代入するための 2 つの方法を説明します。

方法 : 配列変数を初期化する

配列を配列変数に格納する方法と、配列の長さおよび要素の値を設定する方法を説明します。

方法 : ジャグ配列を初期化する

配列の配列、つまりジャグ配列を変数に格納するための 4 つの方法と、配列の長さおよび要素の値を設定する方法を説明します。

方法 : 多次元配列を初期化する

2 次元以上の配列を変数に格納するための 4 つの方法と、配列の長さおよび要素の値を設定する方法を説明します。

方法 : 配列に値を格納する

配列の要素に値を格納する方法を説明します。

方法 : Visual Basic で配列を並べ替える

配列の要素をアルファベット順に並べ替える方法について説明します。

方法 : Visual Basic で配列内の内容を反転させる

配列の要素を逆順に並べ替える方法について説明します。

方法 : 配列の下限に 0 を指定する

配列の下限にゼロを宣言することによって、コードの可読性を高める方法について説明します。

変数の操作

方法 : 新しい変数を作成する

変数の作成に使用できる、**Dim** ステートメントとさまざまなキーワードおよび句について説明します。

方法 : オブジェクトを作成する

オブジェクト変数、およびクラスのインスタンスを作成する方法を説明します。

方法 : 変数に値を格納する、および変数から値を取得する

変数の値を格納および取得する方法について説明します。

変数の宣言と初期化

方法 : Visual Basic でオブジェクト変数を宣言し、オブジェクト変数にオブジェクトを代入する

Object データ型について、またここに任意の型のオブジェクトを代入する方法について説明します。

方法 : 変数内で複数の値を保持する

構造体、配列、およびクラスなど、複数の値を保持できる複合データ型について説明します。

方法 : True および False の値を変数に保持する

Boolean データ型について、および論理値を保持する変数の宣言方法について説明します。

方法 : 変数の有効期間を延長する

Static キーワードについて解説し、このキーワードを使用して、変数のコンテナ要素が存在しなくなった後も変数が削除されないようにする方法を説明します。

方法 : 2 つのオブジェクトが関連しているかどうかを決める

GetType メソッドを使用して、あるオブジェクトが別のオブジェクトを継承したかどうかを調べる方法を説明します。

変数のスコープの制御

方法 : 変数のスコープを制御する

スコープのレベルについて説明し、それらを使用して、どのコードが変数への参照を作成できるかを制御する方法について説明します。

方法 : 変数の可用性を制御する

アクセス レベルの違いについて説明し、その 1 つを変数に割り当てて、どのコードがその変数に対して読み取りまたは書き込みできるかを制御する方法を説明します。

方法 : 自分で宣言した変数と同じ名前の変数を隠す

シャドウを使用して、ある変数を同じ名前の別の変数で隠ぺいするための方法を 2 つ説明します。

データ型の使用

Visual Basic におけるデータ型

データ型をさまざまな種類のプログラミング要素に割り当てる方法を説明します。

文字と文字列

方法 : 変数内に文字を保持する

Char データ型と **String** データ型について、および文字の値を保持する変数の宣言方法について説明します。

方法 : 文字列がパターンに一致するかを調べる

Like 演算子を使用して、文字列内の文字をさまざまな文字セットと一致させる方法を説明します。

方法 : 文字列の一部を削除する (Visual Basic)

部分文字列が文字列に出現する場合にすべて削除する方法を説明します。

数値

方法 : 変数に整数を保持する

SByte、**Short**、**Integer**、および **Long** のデータ型について、および符号付き整数値を保持する変数の宣言方法について説明します。

方法 : 変数で最大有効桁数を保持する

Decimal データ型について、および最大 29 桁を保持する変数の宣言方法について説明します。

方法 : 変数に可能な最大値を保持する

ULong、**Decimal**、**Single**、および **Double** のデータ型について、および非常に大きな値を保持する変数の宣言方法について説明します。

方法 : 変数で小数桁を保持する

Single データ型と **Double** データ型について、および小数の値を保持する変数の宣言方法について説明します。

データ型の変換

方法 : Visual Basic でオブジェクトを別の型に変換する

Object データ型から、より特別なデータ型に型変換する方法について説明します。

日付と時刻
方法 : 日付および時刻の値を変数に保持する

Date データ型について、および日付と時刻の値を保持する変数の宣言方法について説明します。

unsigned 型
方法 : unsigned 型を使用して正の整数のストレージを最適化する

UInteger データ型について、および最も効率的に正の整数値を保持する変数の宣言方法について説明します。

方法 : 符号なしの型を使用する Windows の機能を呼び出す

Byte、**UShort**、**UInteger**、および **ULong** のデータ型について、および unsigned 型を使う関数とやり取りする方法を説明します。

通貨
方法 : 変数で通貨値を保持する.

Decimal データ型について、および通貨の値を保持する変数の宣言方法について説明します。

イベントを使用するプログラム
方法 : Visual Basic でイベント ハンドラを呼び出す

イベントおよびイベント ハンドラの定義方法について説明し、**AddHandler** ステートメントを使用してイベントとイベント ハンドラを関連付けます。

方法 : Visual Basic コード エディタでのイベント ハンドラの作成

Visual Basic コード エディタを使用して、イベントに応答するコードを作成する手順を示します。

方法 : イベントを発生させる (Visual Basic)

イベントの定義方法について説明し、**RaiseEvent** ステートメントを使用してイベントを発生させます。

チュートリアル : イベントの宣言と発生

クラスのイベントの宣言および発生のプロセスを順をおって説明します。

チュートリアル : イベントの処理

イベント ハンドラ プロシージャの記述方法を示します。

エラーおよび例外の処理
方法 : エラー オブジェクトから情報を取得する

Err オブジェクトのプロパティから情報を取得する方法について説明します。

方法 : エラー発生時に制御を継続する

コードの実行中に特定の例外が発生した場合に特定のステートメント ブロックを実行する方法について説明します。

チュートリアル : 構造化例外処理

簡単なアプリケーションの作成方法と例外処理コードの挿入方法に関するチュートリアルです。

方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する

Try...**Catch** ブロックを使用してコード セクションをテストする方法について説明します。

サンプル
Visual Basic 言語のサンプル

これらのサンプルでは、Visual Basic 言語の概念を示します。

参照
概念
Visual Basic での操作方法

データ アクセス (Visual Basic での操作方法)

ここでは、Visual Basic のデータに関連するタスク全般について扱ったヘルプへのリンクを紹介しています。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

データの新機能

クライアント データ アプリケーションの新しいデータ機能に関する情報およびリンクを提供します。

データ アクセスを使用した作業の開始

Visual Studio を使用して、データを操作するアプリケーションを作成する方法に関するトピックへのリンクを示します。

Visual Studio でのデータ アプリケーションの作成

Visual Studio でデータ アプリケーションを作成するプロセスについて説明します。

チュートリアル : 単純なデータ アプリケーションの作成

基本的なデータ アクセス アプリケーションの作成方法を、段階ごとに説明します。

データ アクセス (Visual Basic 6.0 ユーザー向け)

データ アクセスおよびデータ ツールに加えられた変更点について説明します。

ADO.NET

.NET Framework のデータ アクセス機能について説明しているトピックへのリンクを示します。

ローカル データの概要

SQL Server Express データベース、および Access データベースを操作するための、Visual Studio の機能について説明します。

Visual Studio におけるオブジェクトのバインド

データをオブジェクトにバインドするための、Visual Studio の機能について説明します。

ASP.NET データ アクセス (Visual Studio)

ASP.NET Web ページでのデータ操作方法に関する情報へのリンクを示します。

Office ソリューションにおけるデータ

Office ソリューションで、データが機能するしくみについて説明したトピックへのリンクを示します。

マネージ デバイス プロジェクトのデータ

Visual Studio でデバイス用のデータを管理する方法について説明します。

Windows フォームでのデータの表示

データの表示の概要

Windows フォームでデータを表示するプロセスについて説明します。

[データ ソース] ウィンドウ

[データ ソース] ウィンドウから項目をドラッグすることによって、データ バインド フォームを作成する方法を説明します。

方法 : 既存のコントロールにデータをバインドする

既存のコントロールを特定のデータ フィールドにバインドする方法を説明します。

方法 : Windows フォーム DataGridView コントロールにデータを表示する

Windows フォームの DataGridView にデータを表示する方法を説明します。

方法 : 個々の Windows フォーム コントロールにデータを表示する

Windows フォームの各コントロールにデータを表示する方法を説明します。

方法 : 関連するデータを Windows アプリケーションに表示する

データセットの関連テーブルのデータを Windows フォームに表示する方法について説明します。

方法 : パラメータ クエリを Windows アプリケーションのフォームに追加する

既存の Windows フォームを変更して、パラメータの入力およびクエリの実行を行うコントロールを追加する方法を説明します。

方法 : [データ ソース] ウィンドウからドラッグしたときに作成されるコントロールを設定する

[データ ソース] ウィンドウから Windows フォームにアイテムをドラッグしたときに作成されるコントロールの設定方法を説明します。

方法: [データ ソース] ウィンドウにカスタム コントロールを追加する

[データ ソース] ウィンドウでアイテムに対して利用できるコントロールの一覧を変更する方法、およびこの一覧にカスタム コントロールを追加する方法について説明します。

方法 : [データ ソース] ウィンドウを開く

Visual Studio IDE で [データ ソース] ウィンドウを開く方法について説明します。

チュートリアル : Windows アプリケーションのフォームでのデータの表示

データベースからデータを照会する方法、およびそのデータを Windows フォームに表示する方法を、段階ごとに説明します。

チュートリアル : Windows アプリケーションのフォームでの関連データの表示

2 つの関連するテーブルからデータを表示する方法、およびそのデータを Windows フォームに表示する方法を、段階ごとに説明します。

チュートリアル : Windows アプリケーションのデータ検索フォームの作成

ユーザーが入力した値に基づいてデータベース内のレコードを検索する Windows フォームを作成する方法を、段階ごとに説明します。

チュートリアル : ルックアップ テーブルの作成

あるテーブルで選択されたデータに基づいて別のテーブルからデータを表示する方法を、段階ごとに説明します。

チュートリアル : Windows アプリケーションのフォーム間でのデータの受け渡し

アプリケーション内のあるフォームから別のフォームへ値を受け渡す方法を、段階ごとに説明します。

チュートリアル : 単純データ バインディングをサポートするユーザー コントロールの作成

Windows フォームのデータ バインディングを処理するための専用の属性を持つカスタム コントロールを作成する方法を、段階ごとに説明します。

データへのアプリケーションの接続

Visual Studio でのデータへの接続の概要

Visual Studio で、デザイン時ツールを使用してアプリケーションをデータに接続する方法と、ADO.NET 接続オブジェクトを使用してアプリケーションをデータに接続する方法に関する情報を示します。

データ ソース構成ウィザード

データ ソース構成ウィザードを実行して、データへの接続とデータ ソースの作成を行う方法について説明します。

方法 : 接続文字列を保存する

アプリケーションに接続情報を保存する方法について説明します。

方法 : 接続文字列を編集する

以前にアプリケーションに保存した既存の接続情報を編集する方法について説明します。

方法 : オブジェクトのデータに接続する

データ ソース構成ウィザードを使用して、オブジェクトのプロパティ値にデータ バインドする方法について説明します。

方法 : データベース内のデータに接続する

データ ソース構成ウィザードを使用して、アプリケーションとデータベース間の接続を作成する方法について説明します。

方法 : Web サービスのデータに接続する

データ ソース構成ウィザードを使用して、Web サービスから返されたデータとアプリケーションとの間に接続を作成する方法を説明します。

方法 : Access データベース内のデータに接続する

データ ソース構成ウィザードを使用して、アプリケーションと Access データベース間の接続を作成する方法について説明します。

方法 : SQL Express Server データベース内のデータに接続する

データ ソース構成ウィザードを使用して、アプリケーションと SQL Express データベース間の接続を作成する方法について説明します。

方法 : SQL Server データベースへの接続を作成する

SQL Server データベースへの接続文字列を作成する方法について説明します。

方法 : Access データベースへの接続を作成する

Access データベースへの接続文字列を作成する方法について説明します。

方法 : Oracle データベースへの接続を作成する

Oracle データベースへの接続文字列を作成する方法について説明します。

チュートリアル : データベース内のデータへの接続

アプリケーションとデータベース間の接続を作成する方法の詳細な手順について説明します。

チュートリアル : SQL Server Express データベース内のデータへの接続

アプリケーションと SQL Express データベースの間に接続を作成する方法を、段階ごとに説明します。

チュートリアル : Web サービスのデータへの接続

アプリケーションと Web サービスの間に接続を作成する方法を、段階ごとに説明します。

チュートリアル : オブジェクトのデータへの接続

アプリケーションとオブジェクトのプロパティの間に接続を作成する方法を、段階ごとに説明します。

チュートリアル : Access データベース内のデータへの接続

アプリケーションと Access データベース ファイルの間に接続を作成する方法を、段階ごとに説明します。

データ ソースへの接続

ADO.NET の Connection オブジェクトと、それを使用してデータ ソースに接続する方法について説明します。

データの変更

方法 : DataTable に行を追加する

DataRow オブジェクトを作成し、これをデータ テーブルに追加する手順について説明します。

方法 : DataTable の行を編集する

データ テーブルの既存のデータ行を編集する手順について説明します。

方法 : DataTable の行を削除する

データ テーブルの特定のデータ行を削除する手順について説明します。

方法 : データセットの読み込み中に制約をオフにする

データセットで制約チェックを一時的に無効にする方法について説明します。

方法 : データセットの変更をコミットする

データセット内のデータに対して行った変更を受け入れる方法について詳しく説明します。

方法 : 変更された行をチェックする

データセット内のデータが変更されたかどうかを確認する手順について説明します。

方法 : 変更された行を取得する

最後に変更を受け入れた後で変更されたデータ行を返す方法について説明します。

方法 : DataTable の特定の行を探す

主キー値または列の値を使用して、データ テーブル内のデータを検索する方法について説明します。

方法 : 特定のバージョンの DataRow を取得する

DataRowVersion 列挙体を使用して、特定のデータ行を返す方法について説明します。

方法 : エラーが発生している行を探す

データ テーブル内でエラーのあるデータを検索する方法について説明します。

データの保存

データ保存の概要

更新済みのデータをアプリケーションからデータベースに送信する方法について説明しているトピックへのリンクを示します。

方法 : データセットの変更をデータベースに保存する

更新したデータを、TableAdapter と DataAdapter を使用してデータベースに返送する方法について詳細に説明します。

方法 : TableAdapter を使用してデータを更新する

更新したデータを、TableAdapter を使用してデータベースに返送する方法について詳細に説明します。

方法 : データベースに新しいレコードを挿入する

データベースでの新規レコードの作成について詳しく説明します。

方法 : データベースのレコードを更新する

更新済みのレコードをデータベースに返送する方法について詳しく説明します。

方法 : データベースのレコードを削除する

データベースからのレコードの削除について詳しく説明します。

チュートリアル : データベースへのデータの保存 (単一テーブル)

データ テーブルの変更済みデータを基になるデータベースに保存する方法の詳細な手順について説明します。

チュートリアル : データベースへのデータの保存 (複数テーブル)

2 つの関連するデータ テーブルの変更済みデータを基になるデータベースに保存する方法の詳細な手順について説明します。

チュートリアル : トランザクションのデータの保存

System.Transactions 名前空間を使用してデータを保存する方法を、段階ごとに詳しく説明します。

TableAdapter の操作

TableAdapter の概要

TableAdapter の概要と、これを作成するために利用できるツールについて説明します。

TableAdapter クエリの構成ウィザード

ウィザードを実行する方法と、各ウィザード画面の詳細について説明します。

方法 : TableAdapter を作成する

新しい TableAdapter を作成する手順について説明します。

方法 : TableAdapter を編集する

既存の TableAdapter を編集する手順について説明します。

方法 : TableAdapter クエリを作成する

既存の TableAdapter にクエリを追加する手順について説明します。

方法 : TableAdapter クエリを編集する

既存の TableAdapter クエリを編集する手順について説明します。

TableAdapter での NULL 値の使用

TableAdapter パラメータで null 値を使用可能にするための手順について説明します。

方法 : TableAdapter のクエリを表示する

TableAdapter の既存のクエリを表示する方法について説明します。

方法 : パラメータ付きの TableAdapter クエリを作成する

パラメータを受け入れる TableAdapter クエリを作成する手順について説明します。

方法 : データセットにグローバル クエリを追加する

1 つの値を返すクエリを持つ TableAdapter を作成する手順について説明します。

チュートリアル : 複数のクエリによる TableAdapter の作成

TableAdapter を作成してクエリを追加する方法の詳細な手順について説明します。

レポートの設計と表示

ReportViewer Web Server and Windows Forms Controls

Windows アプリケーションまたは Web ベースのアプリケーションでレポートを作成するために使用できる 2 つの新しい ReportViewer コントロールについて説明します。

Creating Reports (Crystal Reports を使用)

Visual Studio で、埋め込まれた Crystal Reports Designer を使用する方法について説明します。

データ サンプル

データのサンプル

データ アクセスを示すサンプルです。

データのフェッチ

データセットへの読み込みとデータのクエリの概要

データセットへのデータの読み込み、ストアド プロシージャの実行、およびデータベースに対するクエリの実行方法について説明します。

方法 : データセットにデータを読み込む

TableAdapter および DataAdapter を使用して、データセットにデータを読み込む方法について説明します。

方法 : 行を返す SQL ステートメントを作成および実行する

TableAdapter クエリおよび Command オブジェクトを使用して、行を返す SQL ステートメントを作成および実行する方法について説明します。

方法 : 単一の値を返す SQL ステートメントを作成および実行する

TableAdapter クエリおよび Command オブジェクトを使用して、単一の値を返す SQL ステートメントを作成および実行する方法について説明します。

方法 : 値を返さない SQL ステートメントを作成および実行する

TableAdapter クエリおよび Command オブジェクトを使用して、値を返さない SQL ステートメントを作成および実行する方法について説明します。

方法 : 行を返すストアド プロシージャを実行する

TableAdapter クエリおよび Command オブジェクトを使用して、行を返すストアド プロシージャを実行する方法について説明します。

方法 : 単一の値を返すストアド プロシージャを実行する

TableAdapter クエリおよび Command オブジェクトを使用して、単一の値を返すストアド プロシージャを実行する方法について説明します。

方法 : 値を返さないストアド プロシージャを実行する

TableAdapter クエリおよび Command オブジェクトを使用して、値を返さないストアド プロシージャを実行する方法について説明します。

方法 : コマンド オブジェクトのパラメータを設定および取得する

クエリおよびストアド プロシージャのパラメータへの値の代入、および実行されたコマンドから返されたパラメータの値の読み込み方法について説明します。

データの検証

データの妥当性検査の概要

データがデータセットにコミットされる前にデータの解析に使用できるイベントの概要を説明します。

方法 : 列の変更時にデータを検証する

ColumnChanging イベント時にデータを検証する方法について説明します。

方法 : 行の変更時にデータを検証する

RowChanging イベント時にデータを検証する方法について説明します。

チュートリアル : データセットへの検証の追加

データセットに検証コードを追加する方法の詳細な手順について説明します。

データセットの操作

方法 : 型指定されたデータセットを作成する

Visual Studio のデザイン ツールを使用して、型指定されたデータセットを作成する方法について説明します。

方法 : データセットの機能を拡張する

型指定されたデータセットの機能を拡張する際に、コードを配置する方法および場所について説明します。

方法 : DataTable を作成する

データセット デザイナを使用して新しい DataTable を作成する手順について説明します。

方法 : DataTable に列を追加する

既存の **DataTable** に新しい DataColumn を作成する手順について説明します。

方法 : DataColumn のデータ型を設定する

DataColumn のデータ型プロパティを設定または変更する方法について説明します。

方法 : DataColumn のキャプションを変更する

DataColumn のキャプションを列名または別の名前に設定する方法について説明します。

方法 : DataColumn の既定値を設定する

新しい **DataColumn** の初期値を設定する方法について説明します。

方法 : 一意の値が格納されるようにデータ列を制限する

DataColumn に重複する値が含まれないように設定する方法について説明します。

方法 : 主キーとしてデータ列を設定する

DataColumn が **DataTable** の主キーになるように設定する方法について説明します。

方法 : 式を表示するデータ列を作成する

計算結果の値が表示されるように **DataColumn** を設定する方法について説明します。

方法 : データセット デザイナで DataRelation を作成する

データセット デザイナを使用してデータセットに DataRelation オブジェクトを追加する方法について説明します。

チュートリアル : データセット デザイナでの DataTable の作成

DataTable を作成し、これを構成する **DataColumn** を定義する方法の詳細な手順について説明します。

チュートリアル : データセット デザイナでのデータセットの作成

データ ソース構成ウィザードを使用しないで、型指定されたデータセットを作成する方法の詳細な手順について説明します。

複数のユーザーと競合の管理

ADO.NET におけるデータ同時実行制御の概要

同時実行制御のさまざまな方法の概要を示します。

方法 : 同時実行エラーを処理する

DBConcurrencyException オブジェクトを使って、同時実行例外や、エラーの原因となったレコードを識別する方法について説明します。

チュートリアル : 同時実行例外の処理

同時実行エラーの識別および解決の手順を提供します。

SQL Server 2005 の使用

SQL Server プロジェクト

SQL Server プロジェクトの概要を示し、SQL Server プロジェクト内に作成できるさまざまな項目について説明します。

マネージ コードを使用したデータベース オブジェクトの作成の利点

.NET 言語を使用して SQL Server データベース オブジェクトを作成することの利点について簡単に説明します。

SQL Server プロジェクトおよびデータベース オブジェクトの属性

SQL Server プロジェクトで使用されるさまざまな属性について説明します。

方法 : SQL Server プロジェクトを作成する

新しい SQL Server プロジェクトを作成する方法の詳細な手順について説明します。

方法 : SQL Server のプロジェクト項目を SQL Server に配置する

SQL Server プロジェクトをデータベースに配置する方法の詳細な手順について説明します。

方法 : CLR の SQL Server ストアド プロシージャを作成および実行する

マネージ コードでストアド プロシージャを作成する方法の詳細な手順について説明します。

方法 : CLR の SQL Server の集計を作成および実行する

マネージ コードで集計を作成する方法の詳細な手順について説明します。

方法 : CLR の SQL Server トリガを作成および実行する

マネージ コードでトリガを作成する方法の詳細な手順について説明します。

方法 : CLR の SQL Server ユーザー定義関数を作成および実行する

マネージ コードでユーザー定義関数を作成する方法の詳細な手順について説明します。

方法 : CLR の SQL Server ユーザー定義型を作成および実行する

マネージ コードでユーザー定義型を作成する方法の詳細な手順について説明します。

SQL Server 2005 のデバッグ

方法 : SQL CLR のストアド プロシージャをデバッグする

SQL Server のストアド プロシージャのデバッグ方法を示します。

チュートリアル : SQL CLR トリガのデバッグ

トリガが起動したときに、ストアド プロシージャからトリガに移動する方法を示します。

チュートリアル : SQL CLR のユーザー定義集計のデバッグ

ユーザー定義集計にステップ インする方法を示します。

チュートリアル : SQL CLR のユーザー定義スカラ関数のデバッグ

ユーザー定義スカラ関数にステップ インする方法を示します。

チュートリアル : SQL CLR のユーザー定義のテーブル値関数のデバッグ

ユーザー定義テーブル値関数にステップ インする方法を示します。

チュートリアル : SQL CLR のユーザー定義型のデバッグ

ユーザー定義型にステップ インする方法を示します。

SQL CLR データベースのデバッグ

CLR データベース オブジェクトのデバッグ方法を示します。

チュートリアル : T-SQL ストアド プロシージャのデバッグ

SQL Server のストアド プロシージャのデバッグ方法を示します。

チュートリアル : T-SQL トリガのデバッグ

トリガが起動したときに、ストアド プロシージャからトリガに移動する方法を示します。

チュートリアル : T-SQL のユーザー定義関数のデバッグ

ストアド プロシージャからユーザー定義関数にステップ インする方法を示します。

T-SQL データベースのデバッグ

T-SQL データベース オブジェクトのデバッグ方法を示します。

データに関するチュートリアル

チュートリアル : データセットへの検証の追加

データセットに検証コードを追加する方法の詳細な手順について説明します。

チュートリアル : データベース内のデータへの接続

アプリケーションとデータベース間の接続を作成する方法の詳細な手順について説明します。

チュートリアル : SQL Server Express データベース内のデータへの接続

アプリケーションと SQL Express データベース間の接続を作成する方法の詳細な手順について説明します。

チュートリアル : Web サービスのデータへの接続

アプリケーションと Web サービス間の接続を作成する方法の詳細な手順について説明します。

チュートリアル : Access データベース内のデータへの接続

アプリケーションと Access データベース ファイル間の接続を作成する方法の詳細な手順について説明します。

チュートリアル : オブジェクトのデータへの接続

アプリケーションとオブジェクトのプロパティ間の接続を作成する方法の詳細な手順について説明します。

チュートリアル : データセット デザイナでのデータセットの作成

データ ソース構成ウィザードを使用しないで、型指定されたデータセットを作成する方法の詳細な手順について説明します。

チュートリアル : データセット デザイナでの DataTable の作成

T:System.Data.DataTable を作成し、T:System.Data.DataTable を構成する T:System.Data.DataColumn を定義する方法の詳細な手順について説明します。

チュートリアル : ルックアップ テーブルの作成

別のテーブルで選択したデータに基づいて 1 つのテーブルからデータを表示する方法の詳細な手順について説明します。

チュートリアル : データ テーブル間のリレーションシップの作成

データセット デザイナを使用して 2 つのデータ テーブルを作成し、これらの間のリレーションシップを追加する方法の詳細な手順について説明します。

チュートリアル : 複数のクエリによる TableAdapter の作成

TableAdapter を作成してクエリを追加する方法の詳細な手順について説明します。

チュートリアル : Windows アプリケーションのデータ検索フォームの作成

ユーザーが入力した値に基づいてデータベース内のレコードを検索する Windows フォームを作成する方法の詳細な手順について説明します。

チュートリアル : 単純データ バインディングをサポートするユーザー コントロールの作成

Windows フォームのデータ バインディングを使用する際に固有の属性を持つカスタム コントロールを作成する方法の詳細な手順について説明します。

チュートリアル : Windows アプリケーションのフォームでのデータの表示

データベースからデータを照会する方法、およびそのデータを Windows フォームに表示する方法を、段階ごとに説明します。

チュートリアル : Windows アプリケーションのフォームでの関連データの表示

2 つの関連するテーブルからデータを表示する方法とそのデータを Windows フォームに表示する方法の詳細な手順について説明します。

チュートリアルのトピック - ASP.NET のデータ アクセス (Visual Studio)

Web ページのデータ アクセスについて説明しているチュートリアル トピックへのリンクを示します。

サンプル

データのサンプル

Visual Basic アプリケーションでのデータの使用を示したサンプルです。

参照

概念

Visual Basic での操作方法

配置 (Visual Basic での操作方法)

ここでは、Visual Basic の配置に関連するタスク全般について扱ったヘルプへのリンクを紹介しています。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

配置ストラテジの選択

ClickOnce と Windows インストーラのテクノロジを比較しています。

64 ビット アプリケーションの配置

64 ビット アプリケーションを配置する際の、一般的な注意事項について説明しています。

カスタムの必須コンポーネントの追加

アプリケーション実行の必要条件である共有コンポーネント、またはシステム コンポーネントのパッケージのインストール方法について説明しています。

Windows インストーラ配置のタスク

共通の Windows インストーラ配置タスクについて説明しています。

ClickOnce による配置

方法 : ClickOnce アプリケーションを発行する

Web サーバー、ファイル共有、またはリムーバブル メディアに ClickOnce アプリケーションを発行する方法について説明します。

方法 : ClickOnce で発行されるファイルを指定する

特定のファイルを発行する方法、または条件に基づいて特定のファイルをインストールする方法を説明します。

方法 : ClickOnce アプリケーションの発行言語を変更する

開発コンピュータの言語とカルチャを設定する方法について説明します。ローカライズされたアプリケーションを発行する場合、ローカライズされたバージョンに合わせて言語とカルチャを指定する必要があります。

方法 : ClickOnce アプリケーションと共に必須コンポーネントをインストールする

アプリケーションと共にパッケージ化される必須コンポーネントを選択する方法について説明します。

方法 : ClickOnce アプリケーションの更新を管理する

更新チェックを実行する時期と方法、更新が必須かどうか、およびアプリケーションの更新をチェックする場所の指定について説明します。

方法 : ClickOnce のセキュリティ設定を有効にする

アプリケーションを発行するためにセキュリティ設定を有効にする方法について説明します。

方法 : アプリケーション マニフェストおよび配置マニフェストに署名する

マニフェストに厳密な名前を付ける方法について説明します。

方法 : 64 ビット アプリケーションを ClickOnce で配置する

64 ビット アプリケーションを ClickOnce で配置する方法について説明しています。

アプリケーションの発行

方法 : ClickOnce アプリケーションを発行する

Web サーバー、ファイル共有、またはリムーバブル メディアに ClickOnce アプリケーションを発行する方法について説明します。

方法 : ClickOnce で発行されるファイルを指定する

特定のファイルを発行する方法、または条件に基づいて特定のファイルをインストールする方法を説明します。

方法 : ClickOnce アプリケーションの発行言語を変更する

開発コンピュータの言語とカルチャを設定する方法について説明します。ローカライズされたアプリケーションを発行する場合、ローカライズされたバージョンに合わせて言語とカルチャを指定する必要があります。

方法 : ClickOnce アプリケーションと共に必須コンポーネントをインストールする

アプリケーションと共にパッケージ化される必須コンポーネントを選択する方法について説明します。

方法 : ClickOnce アプリケーションの更新を管理する

更新チェックを実行する時期と方法、更新が必須かどうか、およびアプリケーションの更新をチェックする場所の指定について説明します。

方法 : ClickOnce の発行バージョンを自動的にインクリメントする

アプリケーションを発行するときに、発行バージョンのリビジョン番号をインクリメントする方法について説明します。

方法 : ClickOnce のインストール モードを指定する

アプリケーションをオフラインまたはオンラインで使用できるかどうかを判断する方法について説明します。

方法 : 配置プロジェクトのプロパティを設定する

配置プロパティのカテゴリ (共通プロジェクト プロパティと構成依存プロパティ) について説明します。

方法 : ユーザー単位のインストールまたはコンピュータ単位のインストールを指定する

コンピュータのすべてのユーザーがアプリケーションを使用できるか、インストールを実行するユーザーだけが使用できるかを指定する方法について説明します。

マニフェストへの署名

方法 : アプリケーション マニフェストおよび配置マニフェストに署名する

マニフェストに厳密な名前を付ける方法について説明します。

方法 : アセンブリに遅延署名する (Visual Studio)

遅延または部分署名を使用して公開キーを指定し、アセンブリが引き渡されるまで秘密キーの追加を遅延する方法について説明します。

安全なアプリケーションの配置

方法 : ClickOnce アプリケーションのセキュリティ ゾーンを設定する

開始時にはアクセス許可の基本セットを指定し、アプリケーションに必要なアクセス許可を 1 つずつ追加していく方法について説明します。

セットアップ プロジェクトによる配置 (Windows インストーラ)

方法 : セットアップ プロジェクトを作成または登録する

セットアップ プロジェクトを作成する方法について説明します。セットアップ プロジェクトは、Windows インストーラ (.msi) ファイルを作成するために使用されます。セットアップ プロジェクトには、標準と Web の 2 種類があります。

チュートリアル : Windows ベースのアプリケーションの配置

ショートカットとファイルの関連付けのセットアップ、レジストリへのエントリの追加、カスタム ダイアログの表示、Internet Explorer のバージョン確認をインストール時に行う Windows アプリケーション用のインストーラを作成する方法について説明します。

チュートリアル : カスタム動作の作成

インストール終了時にユーザーを Web ページへ導く DLL カスタム動作の作成手順について説明します。

方法 : 64 ビットプラットフォーム用の Windows インストーラを作成する

64 ビット アプリケーションおよびコンポーネントのインストーラを作成する方法について説明します。

方法 : Windows インストーラ配置で必須コンポーネントをインストールする

アプリケーションと共にパッケージ化される必須コンポーネントを選択する方法について説明します。

その他の配置プロジェクト

方法 : マージ モジュール プロジェクトの作成または登録を行う

コンポーネントのすべてのファイル、リソース、レジストリ エントリ、およびセットアップ ロジックが含まれるマージ モジュール ファイル (.msm) を作成します。

方法 : Cab プロジェクトを作成または登録する

Web ブラウザにコンポーネントをダウンロードするときに使用するキャビネット ファイル (.cab) を作成します。サーバーではなくクライアントのコンピュータでコードを実行する場合は、このオプションを使用する必要があります。

方法 : 配置プロジェクトにマージ モジュールを登録する

Visual Studio を使用して独自のマージ モジュールを作成します。また、Microsoft やサードパーティ ベンダから提供されている、標準コンポーネント対応のマージ モジュールを使用することもできます。

参照
概念
Visual Basic での操作方法

スマート デバイス アプリケーション (Visual Basic での操作方法)

ここでは、スマート デバイス アプリケーションを作成する際に多く使用される、Visual Basic のタスクに関するヘルプへのリンクを示します。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

方法 : Visual Basic または Visual C# を使用してデバイス アプリケーションを作成する

デバイスのマネージ プロジェクトの作成方法について説明します。これは、デスクトップのプロジェクトを作成する場合と同じプロセスに従います。

チュートリアル : デバイス対応の Windows フォーム アプリケーションの作成

Visual Basic .NET または Visual C# を使用して作成した単純な Windows フォーム アプリケーションを示し、Pocket PC エミュレータでそのアプリケーションを実行します。

チュートリアル : Hello World: スマート デバイスの COM 相互運用の例

簡単な COM オブジェクトとマネージ アプリケーションを 1 つのソリューションとして結合する方法を示します。

方法 : フォームの向きおよび解像度を変更する (デバイス)

既定の方向 (回転) のプロパティを変更する方法について説明します。この説明は、これらのプロパティがない、または無効な SDK をインストールした場合にも適用されます。

方法 : ビジネス オブジェクトをデータ ソースとして追加する (デバイス)

スマート デバイス プロジェクトでビジネス オブジェクトをデータ ソースとして追加する方法を示します。

スマート デバイスでのデータの操作

方法 : パラメータ付きクエリを作成する (デバイス)

パラメータ クエリを作成する方法について説明します。[データ ソース] ウィンドウで SQL Server Mobile データベースが利用できると仮定しています。

方法 : マスター/詳細アプリケーションを作成する (デバイス)

マスター/詳細アプリケーションを作成する方法について説明します。[データ ソース] ウィンドウでテーブルのリレーションシップを持つ SQL Server Mobile データベースが利用できると仮定しています。

方法 : Web サービスをデータ ソースとして追加する (デバイス)

スマート デバイス プロジェクトで Web サービスをデータ ソースとして追加する方法を示します。

方法 : SQL Server データベースをデータ ソースとして追加する (デバイス)

マネージ デバイス プロジェクトで SQL Server データベースをデータ ソースとして使用する方法を示します。

スマート デバイス アプリケーションの配置

チュートリアル : 配置用のスマート デバイス ソリューションのパッケージ化

Visual Studio 2005 を使用してアプリケーションとそのリソースを 1 つの CAB ファイルにパッケージ化し、エンド ユーザーのスマート デバイスに配置できるようにする方法を示します。

方法 : Visual Basic アプリケーションまたは Visual C# アプリケーションに署名する (デバイス)

マネージ デバイス アプリケーションに署名する手順を示します。

コントロールの操作

チュートリアル : デバイスのユーザー コントロールの作成

コントロール ライブラリを作成し、これに格納するユーザー コントロールを作成します。1 つの再利用可能単位としての複数の Windows フォームの組み合わせであるユーザー コントロールと、標準のコントロールからは利用できない UI または機能を必要とするコントロールであるカスタム コントロールとの違いについて説明します。

アプリケーションのデバッグとテスト

デバイスのデバッグとデスクトップのデバッグとの違い

デスクトップ デバッガでサポートされる機能の大部分をサポートするデバイス デバッガと、デスクトップ デバッガの違いについて説明します。例外も示します。

チュートリアル : Windows フォームのデバッグ

Windows フォームをデバッグする方法について説明します。

チュートリアル : Web フォームのデバッグ

Web フォームをデバッグする方法について説明します。

チュートリアル : XML Web サービスのデバッグ

XML Web サービスをデバッグする方法について説明します。

サンプル
サンプルとチュートリアル (スマート デバイス プロジェクト)

デバイス プログラミングに関するさまざまな問題を解決するためのテクニックを示すサンプルです。

参照
概念
Visual Basic での操作方法
スマート デバイス開発での操作方法

Office プログラミング (Visual Basic での操作方法)

このページでは、よく使用する Office プログラミングのタスクに関するヘルプへのリンクを紹介します。各ヘルプには、Visual Studio Tools for Office のコンテキストでの Visual Basic のサンプル コードが示されています。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

方法 : ドキュメント レベルのプロジェクトをアップグレードする

最新バージョンの Microsoft Visual Studio Tools for the Microsoft Office System で提供される新しいプロジェクト システムおよびツールを利用するために、実行しなければならない手順について説明します。

方法 : Office のプライマリ相互運用機能アセンブリをインストールする

必要な Microsoft Office 2003 プライマリ相互運用機能アセンブリをエンド ユーザーのコンピュータにインストールする方法について説明します。

方法 : Office ソリューションの構成情報を設定する

Office プロジェクトに .config ファイルを手動で追加する方法について説明します。

方法 : Visual Studio Tools for Office プロジェクトを作成する

Microsoft Office プロジェクト ウィザードにより、新規または既存の文書やブックを利用して Word プロジェクトまたは Excel プロジェクトを作成する方法について説明します。

方法 : プライマリ相互運用機能アセンブリを使用して Office アプリケーションを自動化する

マネージ コードを使用して Visual Studio で他の Office アプリケーション (Microsoft Office PowerPoint 2003 など) を自動化する方法について説明します。

方法 : Office ドキュメントにコントロールを追加する

デザイン時または実行時に、Excel 2003 および Word 2003 の文書に Windows フォーム コントロールを追加する方法について説明します。

チュートリアル : Office ソリューションのアクセス許可の付与および削除

Visual Studio Tools for Office ソリューションのセキュリティについて基本手順を説明します。

方法 : Office ソリューションを配置する

Office ソリューションを配置する 3 つの方法を説明します。ドキュメントとアセンブリをネットワークに配置する方法、それらをローカルに配置する方法、アセンブリをネットワークにドキュメントをローカルに配置する方法です。

Excel アプリケーション

方法 : Excel の計算をプログラムで実行する

Calculate メソッドを使用して、特定の範囲で、または開いているすべてのブックで、すべての計算を実行する方法について説明します。

方法 : Office プロジェクトのエラーを処理する

マネージ コードとアンマネージ コードの操作が、Microsoft Office Word 2003 プロジェクトおよび Microsoft Office Excel 2003 プロジェクトでのエラー処理にどのように影響するかについて説明します。

方法 : ワークシートのセルに文字列を表示する

プログラムを使用してセルにテキストを表示する方法を示します。名前付き範囲を作成してセルを参照できます。

Word アプリケーション

方法 : Word の組み込みダイアログ ボックスを使用する

ユーザーからの入力を、Word の組み込みダイアログ ボックスを使って表示する例を紹介します。

方法 : Office ツール バーをプログラムで作成する

プログラムを使用して 2 つのボタンが設定されたツール バーを Word 2003 に作成する方法の例を示します。

Outlook アプリケーション

方法 : 電子メール アイテムを作成する

Microsoft Outlook 2003 で電子メール アイテムを作成する方法を示す例を紹介します。

方法 : カスタム アイコンをツール バーおよびメニュー項目に追加する

カスタム メニューのコマンドにアイコンを適用する例を紹介します。

Office ソリューションにおけるデータ

方法 : Office ドキュメント内のデータ ソースをプログラムでキャッシュする

プログラムで **StartCaching** メソッドを呼び出し、データ ソースを渡して、文書キャッシュにデータ ソースを追加する方法について説明します。

方法 : Visual Studio 内で Word 文書にスキーマを割り当てる

Visual Studio Tools for Office プロジェクトでドキュメントを開いているときに、ドキュメントが Visual Studio の外部で開かれている場合に使用するのと同じ Word 2003 ツールを使用して XML スキーマをドキュメントに割り当てる方法について説明します。

チュートリアル : ワークシートのセルのデータベース フィールドへの連結

SQL Server データベース内の単一のデータ フィールドを Excel 2003 内の名前付き範囲にバインドする際の基本事項について説明します。

方法 : オブジェクトのデータをドキュメントに読み込む

Word 2003 文書内のオブジェクトに格納されたデータにアクセスする方法と、Windows フォーム プロジェクトの場合の類似点について説明します。

スマート タグと操作ウィンドウ

チュートリアル : 操作ウィンドウから文書へのテキストの挿入

ユーザーが操作ウィンドウで入力したテキストを使って Word 2003 文書にデータを入力する場合の、操作ウィンドウの作成手順について説明します。

方法 : カスタム ドキュメント プロパティを作成および変更する

文書またはブックに保存する追加情報がある場合に、カスタム ドキュメント プロパティを作成および変更する方法について説明します。

サンプル

Office 開発のサンプル

Microsoft Office System 用の Microsoft Visual Studio 2005 Tools を使用して、Microsoft Office Word 2003 および Microsoft Office Excel 2003 に含まれる機能を利用する方法を示すサンプルです。

参照

概念

Visual Basic での操作方法

Visual Studio Tools for Office での操作方法

Web アプリケーション (Visual Basic での操作方法)

ここでは、Web アプリケーションを作成する際に多く使用される、Visual Basic のタスクに関するヘルプへのリンクを示します。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

Visual Studio の Web 開発の新機能

ASP.NET Web アプリケーションを作成するためのツールとユーティリティのセットである Visual Web Developer について説明します。

チュートリアル : Visual Web Developer での基本的な Web ページの作成

Visual Studio で簡単な ASP.NET Web ページを作成したり編集したりする方法を紹介します。Web 開発環境の概要についても説明します。

チュートリアル : Visual Web Developer での基本的な HTML 編集

Visual Web Developer の HTML 編集機能について紹介します。

チュートリアル : Visual Web Developer での Web ページのデバッグ

Web ページでのデバッガの使用方法の概要を示します。

Web フォームおよびコントロールの使用

方法 : ASP.NET サーバー コントロールに必要なエントリを検証する

RequiredFieldValidator コントロールをページに追加し、それを必要なコントロールにリンクさせることにより、特定のコントロールにおいて、ユーザーがなんらかの情報を必ず入力しなければならないように指定する方法について説明します。

方法 : CheckBox Web サーバー コントロールにおけるユーザー選択に応答する

ユーザーが各 CheckBox コントロールをクリックしたときに、直接応答するイベント ハンドラを作成する手順を示します。

方法 : リスト Web サーバー コントロールの選択項目を確認する

リスト選択項目に関連する 3 つの値を取得する方法を示します。3 つの値とは、選択項目のインデックス値、テキスト (表示値)、およびその値 (存在する場合) です。

方法 : ユーザーが DataList Web サーバー コントロール内の項目を選択できるようにする

ユーザーが項目の強調表示に使用できる選択ボタンを用意する手順を示します。

方法 : Table Web サーバー コントロールに行およびセルを動的に追加する

実行時に、Table コントロールに行やセルを作成する手順を示します。

方法 : ASP.NET Web ページにコントロールをプログラムによって追加する

ドロップダウン リストの変更イベントを使用して、実行時に Web フォーム ページにラベルを追加します。

方法 : ASP.NET Web ページにクライアント スクリプトを動的に追加する

サーバー コードを使用してプログラムでページにクライアント スクリプトを追加する方法について説明します。

Web サイトの配置

チュートリアル : Web サイトの公開

Web サイトをコンパイルしてアセンブリを作成し、Web サーバーに配置できるようにする方法を紹介します。

チュートリアル : Copy Web Site ツールを使用した Web サイトのコピー

Web サイトを構築し、実行用の Web サーバーにファイルをコピーする方法を紹介します。

Web サイトの開発

方法 : ASP.NET マスタ ページ用のコンテンツ ページを作成する (Visual Studio)

マスタ ページに関連付けられている ASP.NET Web ページであるコンテンツ ページの作成方法について説明します。

方法 : ASP.NET Web ページ間で値をやり取りする

1 つの ASP.NET Web ページから別の Web ページにリダイレクト (移動) するアプリケーションで、元のページから移動先のページに情報を渡す必要がある場合に、値の受け渡し方法について説明します。

方法 : 他のページにユーザーをリダイレクトする

1 つの ASP.NET Web ページから別の Web ページにユーザーをリダイレクトする方法について説明します。

方法 : ASP.NET マスター ページのコンテンツを参照する

コンテンツ ページでコードを使用してマスタ ページのプロパティ、メソッド、およびコントロールを参照する方法について説明します。いくつかの制限も示します。

方法 : ASP.NET のテーマを適用する

アプリケーション レベルおよびページ レベルでテーマを設定する方法について説明します。

ASP.NET サーバー コントロールの検証の種類

使用できる検証コントロールの種類とその使用方法の一覧を示します。

チュートリアル : Visual Studio でのマスター/詳細 Web ページの作成

複数のコントロールおよび複数のテーブル (マスター/詳細リレーションシップを持つものを含む) のデータを操作する方法について説明します。

チュートリアル : ASP.NET ユーザー コントロールによる再利用可能な要素の作成

ピッカー コントロールとして機能する ASP.NET ユーザー コントロールの作成方法について説明します。ピッカー コントロールは 2 つのリスト ボックスで構成され、そのうちの 1 つ (ソース) に選択肢のセットが格納されています。

チュートリアル : モバイル デバイス用 Web ページの作成

MobilePage クラスから継承する、モバイル デバイス向けに設計された 2 つの Web ページを作成します。

チュートリアル : Visual Studio でのテーマを使用した Web サイトのカスタマイズ

テーマを使用して、Web サイトのページおよびコントロールに一貫した表示形式を適用する方法について説明します。

チュートリアル : ASP.NET トレースと System.Diagnostics トレースの統合

ASP.NET のトレース機能を System.Diagnostics トレース機能と統合し、すべてのトレース メッセージを 1 つのトレース出力に書き込む方法について説明します。

チュートリアル : ASP.NET でのローカリゼーションのためのリソースの使用

ローカライズされた Web ページを効率的に作成する方法について説明します。ユーザーの言語とカルチャに基づいて、ページのテキストやコントロールのリソースを使用します。

チュートリアル : Visual Web Developer での Web パーツ ページの作成

Visual Studio などのビジュアルなデザイナで Web パーツを操作するための基本的なコンポーネントとタスクを示します。

ASP からのアップグレード
Visual Studio .NET からの Web プロジェクト変換

Visual Studio 2003 .NET プロジェクトから Visual Studio 2005 プロジェクトへの変換に関するさまざまな局面について説明します。

Visual Web Developer への変換のトラブルシューティング

Web アプリケーションの動作が変換前と変換後で異なる場合の状況および解決策を示します。

参照
概念
Visual Basic での操作方法
Visual Web Developer での操作方法

Web サービス (Visual Basic での操作方法)

ここでは、Web サービス アプリケーションを作成する際に多く使用される、Visual Basic のタスクに関するヘルプへのリンクを示します。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

チュートリアル : Visual Web Developer での ASP.NET Web サービスの作成と使用

Visual Web Developer で単純な XML Web サービスを作成する方法について説明します。

チュートリアル : Visual Basic または Visual C# を使った XML Web サービスの作成

華氏で測定された温度を摂氏に変換する XML Web サービスを作成する方法について説明します。

方法 : ASP.NET Web サービス プロジェクトを作成する

ASP.NET Webサービス プロジェクトのテンプレートを使用して XML Web サービスを作成する方法について説明します。

方法 : Web サービスのデータに接続する

データ ソース構成ウィザードを使用して、アプリケーションと Web サービスから返されたデータとの間の接続を作成する方法について説明します。

Web サービスの作成

チュートリアル : Visual Web Developer での ASP.NET Web サービスの作成と使用

Visual Web Developer で単純な XML Web サービスを作成する方法について説明します。

チュートリアル : Visual Basic または Visual C# を使った XML Web サービスの作成

華氏で測定された温度を摂氏に変換する XML Web サービスを作成する方法について説明します。

方法 : ASP.NET Web サービス プロジェクトを作成する

ASP.NET Webサービス プロジェクトのテンプレートを使用して XML Web サービスを作成する方法について説明します。

方法 : Web サービスのデータに接続する

データ ソース構成ウィザードを使用して、アプリケーションと Web サービスから返されたデータとの間の接続を作成する方法について説明します。

Web サービスのデバッグと配置

XML Web サービスへのステップ イン

クライアント アプリケーションと Web サービスの間を行き来する方法について説明します。

方法 : マネージ コードを使用して XML Web サービスを配置する

複数の方法を使用して Web サービスを配置する方法について説明します。

方法 : マネージ コードを使用して XML Web サービスをデバッグする

複数の方法を使用して Web サービスをデバッグする方法について説明します。

Web サービスの呼び出し

チュートリアル : Visual Basic または Visual C# を使った XML Web サービスへのアクセス

Visual Basic または Visual C# を使って作成されたアプリケーションから XML Web サービスにアクセスする手順について説明します。

方法 : Web サービス クライアントから非同期呼び出しを実行する

クライアント アプリケーションから Web サービスに対して非同期呼び出しを行う方法について説明します。

方法 : ASP.NET を使用して作成した既存の XML Web サービスを調査する

Web サービスが実装している Web サービス メソッド、そのパラメータ、戻り値の型など、Web サービスの機能に関する情報を提供します。

方法 : Web サービスを呼び出す

Web サービス メソッドを呼び出して文字列値を定義する方法について説明します。

方法 : マネージ コードを使用して既存の Web プロジェクトに XML Web サービスを追加する

既存の Web プロジェクトに Web サービスを追加する方法について説明します。

参照

概念

Visual Basic での操作方法

Visual Web Developer での操作方法

Windows アプリケーション (Visual Basic での操作方法)

ここでは、Windows アプリケーションを作成する際に多く使用される、Visual Basic のタスクに関するヘルプへのリンクを示します。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

Visual Basic プログラムの構造

単純な Visual Basic プログラムの一般的なアウトラインを説明します。

チュートリアル : 簡単な Windows フォームの作成

簡単な "Hello, World" アプリケーションの作成方法を示します。

Windows フォームおよびコントロールの使用

Windows フォームと Windows フォーム コントロールの新機能

このリリースの Visual Studio で使用できる新機能と強化された機能を紹介します。

Windows フォームの新機能 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6 ユーザーに関連する重要な変更点について説明します。

フォームの操作

デザイン時

方法 : Windows アプリケーションでスタートアップ フォームを選択する

アプリケーションを起動すると最初に表示されるフォームを設定する方法について説明します。

方法 : Windows フォームの 1 つのイベント ハンドラに複数のイベントを関連付ける

イベントを使用して、複数のコントロールに同じ機能を割り当てる手順を示します。

方法 : Windows フォームでマルチペイン ユーザー インターフェイスを作成する

Microsoft Outlook で使われているようなマルチペインのユーザー インターフェイスを作成する方法について説明します。

方法 : Windows フォームに背景イメージを追加する

コントロールまたはフォーム自体に背景イメージを配置する方法について説明します。

方法 : デザイン時に Windows フォームのコントロールにツールヒントを設定する

コードまたはデザイナでツールヒントを設定する方法について説明します。

方法 : 既存のコントロールを別の親に再配置する

新しい親コンテナに既存のコントロールを割り当てる方法について説明します。

方法 : Windows フォームに ActiveX コントロールを追加する

レガシ ActiveX コントロールを使用する手順を示します。

方法 : Windows アプリケーションにヘルプを提供する

HelpProvider コンポーネントを使用してヘルプ システムのファイルにコントロールを関連付ける方法について説明します。

方法 : Windows フォーム コントロールのアクセス キーを作成する

定義済みショートカット キーの作成について説明します。

方法 : Windows フォームを継承する

コードを記述して、継承フォームを作成する方法について説明します。

方法 : 既存の Windows フォーム コントロールから継承する

継承されたコントロールを作成する方法について説明します。

実行時

方法 : コントロールのコレクションに対して実行時にコントロールを追加または削除する

実行時にパネルにコントロールを追加したり、パネルからコントロールを削除したりする方法について説明します。

方法 : Windows XP の Visual スタイルを有効にする

EnableVisualStyles プロパティを使用して、Windows XP の標準の外観をフォームおよびコントロールに設定する方法について説明します。

方法 : スタートアップ Windows フォームを非表示にする

フォームの初期の参照可能範囲を実行時に設定する方法について説明します。

方法 : Windows フォームを常に一番手前に表示する

実行時に、常にほかのフォームの手前に Windows フォームを表示する方法について説明します。

方法 : Windows フォームをモーダルおよびモードレスで表示する

ダイアログ ボックスをモーダルに表示する方法とモードレスに表示する方法について説明します。

方法 : 実行時にコントロールを非表示にする

実行時に表示されないユーザー コントロールを作成する方法について説明します。

コモン コントロール

TextBox コントロール

方法 : Windows フォーム TextBox コントロールでテキストを選択する

テキスト ボックスのテキストを強調表示する方法について説明します。

方法 : 文字列に引用符を挿入する (Windows フォーム)

テキスト ボックスの文字列に引用符を追加する方法について説明します。

方法 : Windows フォームの RichTextBox コントロールにスクロール バーを表示する

RichTextBox コントロールのスクロール バーに関するさまざまな選択肢について説明します。

方法 : 読み取り専用テキスト ボックスを作成する (Windows フォーム)

テキスト ボックスの内容が変更されないようにする方法について説明します。

方法 : Windows フォームの TextBox コントロールを使用してパスワード テキスト ボックスを作成する

テキスト ボックスに入力するデータを非表示にする方法について説明します。

方法 : Windows フォーム TextBox コントロールでのカーソル位置を制御する

エディット コントロールに最初にフォーカスが設定されたときに、カーソルを表示する位置を指定する手順について説明します。

方法 : クリップボードからデータを取得する

クリップボードに格納されているデータにアクセスする方法について説明します。

方法 : クリップボードにデータを追加する

プログラムでクリップボード上に情報を挿入する方法について説明します。

方法 : MaskedTextBox コントロールにデータをバインドする

データベースのデータがマスク定義で予期される書式に一致しない場合に、データの書式を再設定する方法について説明します。

チュートリアル : MaskedTextBox コントロールの使用

MaskedTextBox コントロールの主な機能について説明します。

RichTextBox コントロール

方法 : Windows フォームの RichTextBox コントロールにファイルを読み込む

RichTextBox コントロールに既存のファイルを読み込む方法について説明します。

方法 : Windows フォームの RichTextBox コントロールにスクロール バーを表示する

RichTextBox コントロールのスクロール バーに関するさまざまな選択肢について説明します。

方法 : Windows フォームの RichTextBox コントロールのフォント属性を設定する

RichTextBox コントロール内のテキストのフォント ファミリ、サイズ、スタイル、および色を設定する方法について説明します。

方法 : Windows フォームの RichTextBox コントロールを使用してインデント、ぶら下げインデント、および箇条書き段落を設定する

RichTextBox コントロール内の段落に書式を設定する方法について説明します。

方法 : Windows フォームの RichTextBox コントロールにおけるドラッグ アンド ドロップ操作を有効にする

RichTextBox コントロールにデータをドラッグする方法について説明します。

方法 : Windows フォームの RichTextBox コントロールを使用して Web スタイルのリンクを表示する

RichTextBox コントロールから Web サイトにリンクする方法について説明します。

Button コントロール

方法 : Windows フォームのボタンのクリックに応答する

Windows アプリケーションのフォームでのボタンの基本的な使用法について説明します。

方法 : デザイナを使用して Windows フォームの Button コントロールを承認ボタンとして指定する

ボタンを Accept ボタンとして設定する方法について説明します。

方法 : デザイナを使用して Windows フォームの Button コントロールをキャンセル ボタンとして指定する

ボタンを Cancel ボタンとして設定する方法について説明します。

CheckBox および RadioButton コントロール

CheckBox コントロール

方法 : Windows フォームの CheckBox コントロールでオプションを設定する

チェック ボックスを使用して、オブジェクトのプロパティなどのオプションを設定する方法について説明します。

方法 : Windows フォーム CheckBox のクリックに応答する

チェック ボックスを使用して、アプリケーションのアクションを決定する方法について説明します。

RadioButton コントロール

方法 : セットとして機能する Windows フォーム RadioButton コントロールをグループ化する

1 つのボタンだけをオンにできるセットとして、オプション ボタンをグループ化する方法について説明します。

ListBox、ComboBox、および CheckedListBox コントロール

方法 : Windows フォームの ComboBox または ListBox コントロールをデータにバインドする

リスト ベースのコントロールをデータ ソースに連結する方法について説明します。

方法 : Windows フォーム ComboBox、ListBox、または CheckedListBox コントロールのルックアップ テーブルを作成する

フォーム データを使いやすい形式で表示および保持する方法について説明します。

方法 : Windows フォームの ComboBox、ListBox、または CheckedListBox コントロールに項目を追加または削除する

コントロールの項目のリスト内の項目を追加または削除する方法について説明します。

方法 : Windows フォーム ComboBox、ListBox、または CheckedListBox コントロールの特定の項目にアクセスする

リスト内のどの項目が指定された位置に表示されるかをプログラミングによって判断する方法について説明します。

方法 : Windows フォーム ComboBox、ListBox、または CheckedListBox コントロールを並べ替える

データ ソースにあるリスト データを並べ替える方法について説明します。

CheckedListBox コントロール

方法 : Windows フォーム CheckedListBox コントロールでオンになっている項目を判断する.

どの項目のチェック ボックスがオンにされているかを確認するためのリストの操作手順について説明します。

DataGridView コントロール

方法 : デザイナを使用してデータを Windows フォーム DataGridView コントロールにバインドする

コントロールのスマート タグで [データ ソースの選択] オプションを使用して、データに接続する方法について説明します。

方法 : Windows フォームの 2 つの DataGridView コントロールを使用してマスター/詳細形式のフォームを作成する

2 つの関連するデータベース テーブルのデータを表示する際に、一方の DataGridView コントロールに表示されている値が別のコントロールで現在選択されている行に依存するようにする方法を、コードと共に示します。

方法 : Windows フォーム DataGridView コントロールのデータを検証する

ユーザー入力を検証してデータ入力書式エラーの発生を防ぐ方法を、コードと共に示します。

方法 : Windows フォーム DataGridView コントロールでのデータ入力中に発生したエラーを処理する

ユーザーが新しい値をコミットしようとしたときに、データ ソースから生じたデータ入力エラーを処理する方法を、コードと共に示します。

方法 : デザイナを使用して Windows フォーム DataGridView コントロールで行が追加および削除されないようにする

コントロールのスマート タグを使用して、ユーザーがデータを追加したり削除したりできないようにする方法について説明します。

方法 : Windows フォーム DataGridView コントロールの新しい行に既定値を指定する

新しいレコードの行をあらかじめ読み込んで、データ入力時間を節約する方法について説明します。

方法 : 連結されていない Windows フォーム DataGridView コントロールを作成する

手動でコントロールにデータを読み込む方法を、コードと共に示します。

方法 : データバインドされた Windows フォーム DataGridView コントロールに非バインド列を追加する

バインドされていない追加の列を表示することにより、バインドされたデータ ソースのデータを補完する方法について説明します。

方法 : Windows フォーム DataGridView コントロールのセルにイメージを表示する

各セルにアイコンを表示するイメージ列を作成する方法について説明します。

方法 : Windows フォーム DataGridView Cells でコントロールをホストする

IDataGridViewEditingControl インターフェイスを実装し、DataGridViewCell および DataGridViewColumn から派生したカスタム型を作成して、セルが編集モードのときに DateTimePicker コントロールが表示されるようにする方法について説明します。

チュートリアル : Windows フォーム DataGridView コントロールのデータの妥当性検査

ユーザー入力を検証してデータ入力書式エラーの発生を防ぐ方法について説明します。

チュートリアル : Windows フォーム DataGridView コントロールでのデータ入力中に発生したエラーの処理

ユーザーが新しい値をコミットしようとしたときに、データ ソースから生じたデータ入力エラーを処理する方法について説明します。

チュートリアル : バインドされていない Windows フォーム DataGridView コントロールの作成

手動でコントロールにデータを読み込む方法について説明します。

DataGridView レイアウトと書式

方法 : デザイナを使用して Windows フォーム DataGridView コントロールの列を固定する

[列の編集] ダイアログ ボックスを使用して、特定の列がスクロールしないようにする方法について説明します。

方法 : デザイナを使用して Windows フォームの DataGridView コントロールで列を読み取り専用にする

[列の編集] ダイアログ ボックスを使用して、特定の列の値をユーザーが編集できないようにする方法について説明します。

方法 : デザイナを使用して Windows フォーム DataGridView コントロールの列の並べ替えを有効にする

コントロールのスマート タグを使用して、ユーザーが列を並べ替えることができるようにする方法について説明します。

方法 : デザイナを使用して Windows フォーム DataGridView コントロールの列の順序を変更する

[列の編集] ダイアログ ボックスを使用して、列を並べ替える方法について説明します。

方法 : デザイナを使用して Windows フォーム DataGridView コントロールの列を追加および削除する

[列の追加] ダイアログ ボックスおよび [列の編集] ダイアログ ボックスを使用して、列コレクションへのデータの読み込みや列コレクションの編集を行う方法について説明します。

ListView および TreeView コントロール

チュートリアル : デザイナを使用した、ListView コントロールと TreeView コントロールを含むエクスプローラ スタイルのインターフェイスの作成

2 つのコモン コントロールを使用して、本格的な外観の Windows アプリケーションを作成する方法について説明します。

ListView コントロール

方法 : Windows フォーム ListView コントロールで項目を追加および削除する

リスト ビューの項目を追加または削除する方法について説明します。

方法 : ListView コントロールに検索機能を追加する

ユーザーに対して検索機能を提供する、テキスト一致および位置検索という 2 つの方法について説明します。

方法 : Windows フォーム ListView コントロール内の項目を選択する

Windows フォームの ListView コントロール内の項目をプログラムで選択する方法について説明します。

方法 : Windows フォーム ListView コントロールのアイコンを表示する

リスト ビューを適切なイメージ リストに関連付け、大きいアイコンまたは小さいアイコンを表示する方法について説明します。

方法 : Windows フォーム ListView コントロールの列にサブ項目を表示する

リストの各項目に関する情報を列に表示する方法について説明します。

TreeView コントロール

方法 : Windows フォーム TreeView コントロールのアイコンを設定する

ツリー ビューのノードに対してアイコンを表示する方法について説明します。

方法 : Windows フォーム TreeView コントロールでノードを追加および削除する

ツリー ビューのノードを追加または削除する方法について説明します。

方法 : クリックされた TreeView ノード (Windows フォーム) を判別する

アプリケーションが適切に応答できるように、ツリー ビューのどのノードがクリックされたかを判断する方法について説明します。

コンテナ コントロール

方法 : ウィンドウを水平方向に分割する

SplitContainer コントロール内で分割線の方向を制御する方法について説明します。

方法 : Windows フォームでマルチペイン ユーザー インターフェイスを作成する

Microsoft Outlook で使用されるようなマルチペイン ユーザー インターフェイスを作成します。

方法 : TableLayoutPanel コントロールの行と列を拡大する

TableLayoutPanel で隣接する行や列にコントロールがまたがるようにする方法について説明します。

チュートリアル : TableLayoutPanel を使用した Windows フォーム上のコントロールの配置

フォームのサイズが変更されたり、コンテンツのサイズが変更されたときに、それに応じて自身でレイアウトを調整するフォームを作成する方法について説明します。

チュートリアル : FlowLayoutPanel を使用した Windows フォーム上のコントロールの配置

フォームのサイズが変更されたり、コンテンツのサイズが変更されたときに、それに応じて自身でレイアウトを調整するフォームを作成する方法について説明します。

ピクチャおよび Image コントロール

方法 : デザイナを使用してピクチャを読み込む (Windows フォーム)

プロパティを設定することにより、デザイン時にピクチャを読み込んでフォームに表示する方法について説明します。

方法 : 実行時にピクチャを設定する (Windows フォーム)

実行時にピクチャを表示およびクリアする手順について説明します。

方法 : 実行時にピクチャのサイズまたは配置を変更する (Windows フォーム)

SizeMode プロパティの実行内容と設定方法について説明します。

方法 : イメージをトリミングおよびスケーリングする

ベクタおよびラスタ イメージのトリミングとスケール調整をプログラムで行う方法について説明します。

日付設定コントロール

DateTimePicker コントロール

方法 : Windows フォームの DateTimePicker コントロールを使用して日付を設定および取得する

コントロールで日付を設定する手順、およびユーザーによって選択された日付にアクセスする手順を説明します。

方法 : Windows フォームの DateTimePicker コントロールを使用してカスタム形式で日付を表示する

書式指定文字列を使用して日付を任意の書式で表示する方法について説明します。

MonthCalendar コントロール

方法 : Windows フォームの MonthCalendar コントロールで日付の範囲を選択する

MonthCalendar コントロールから特定範囲の日付をプログラムで選択する方法について説明します。

方法 : Windows フォームの MonthCalendar コントロールを使用して特定の日付を太字で表示する

特定の日付を太字で表示させる方法を説明します。

方法 : Windows フォームの MonthCalendar コントロールにおいて複数の月を表示する

同時に 2 か月以上を表示させるように **MonthCalendar** コントロールを設定する方法について説明します。

方法 : Windows フォームの MonthCalendar コントロールの外観を変更する

MonthCalendar コントロールの表示形式をカスタマイズする方法について説明します。

データ アクセス (Windows フォームの場合)

データの新機能

クライアントおよびデータ層アプリケーションの新しいデータ機能に関する情報およびリンクを提供します。

方法 : Windows フォーム DataGridView コントロールにデータを表示する

DataGridView コントロールにデータを表示する方法について説明します。

チュートリアル : Windows アプリケーションのフォーム間でのデータの受け渡し

1 つのアプリケーション内のあるフォームから別のフォームに値を渡す方法の詳細な手順について説明します。

チュートリアル : Windows アプリケーションのフォームでのデータの表示

データベースのデータに対するクエリを実行し、そのデータをフォームに表示する方法の詳細な手順について説明します。

チュートリアル : 単純データ バインディングをサポートするユーザー コントロールの作成

Windows フォームのデータ バインディングを処理するための専用の属性を持つカスタム コントロールを作成する方法を、段階ごとに説明します。

チュートリアル : Windows アプリケーションのデータ検索フォームの作成

ユーザーが入力した値に基づいてデータベース内のレコードを検索するフォームを作成する方法の詳細な手順について説明します。

チュートリアル : オブジェクトのデータへの接続

アプリケーションとオブジェクトのプロパティ間の接続を作成する方法の詳細な手順について説明します。

メニューとツール バー

ToolStrip コントロール

方法 : デザイナを使用して標準アイテムで基本的な Windows フォーム ToolStrip を作成する

ToolStrip を作成し、標準的なタスクを表す 7 つの ToolStripButton コントロールを追加する方法について説明します。

方法 : ToolStrip に ToolStripItem を配置する

ToolStripButton を **ToolStrip** の左端または右端に配置する方法について説明します。

方法 : デザイナを使用して ToolStripMenuItems を無効にする

メニュー全体および個々のメニュー コマンドの両方を無効にする方法について説明します。

方法 : デザイナを使用して ToolStripMenuItems を非表示にする

メニュー全体および個々のメニュー コマンドの両方を非表示にする方法について説明します。

方法 : ToolStripMenuItems を移動する

トップレベル メニュー全体およびそのメニュー項目を MenuStrip 上の別の位置に移動します。

方法 : Windows フォーム内の ToolStrip テキストとイメージの外観を変更する

ToolStripItem コントロールでのテキストおよびイメージの配置を定義および変更する方法について説明します。

コンテキスト メニュー

方法 : ショートカット メニューを Windows フォーム NotifyIcon コンポーネントに関連付ける

ユーザーが右クリックしたときにコマンドのメニューが表示されるように、NotifyIcon コンポーネントにコンテキスト メニューを追加する手順について説明します。

方法 : Windows フォーム ContextMenu コンポーネントのメニュー項目を追加および削除する

コンテキスト メニューの項目を追加する方法および削除する方法を説明します。

コントロールでのデータ バインディング

方法 : データ バインドで発生するエラーと例外を処理する

BindingSource コンポーネントを使用して、データ バインディング操作で発生したエラーを適切に処理する方法について説明します。

BindingSource コントロール

方法 : デザイナを使用して Windows フォーム コントロールを BindingSource コンポーネントにバインドする

アプリケーションに関連するデータを実行時にユーザーが変更したり保存したりできるように、データ ソースにコントロールをバインドする方法について説明します。

方法 : Windows フォーム BindingSource コンポーネントを使用してルックアップ テーブルを作成する

コンボ ボックスを使用して、親テーブルから子テーブルへの外部キー リレーションシップを持つフィールドを表示する方法について説明します。

方法 : BindingSource を使用して Windows フォーム コントロール内にデータ ソースの更新を反映させる

BindingSource コンポーネントを使用して、データ ソースの変更に対応する方法について説明します。

方法 : Windows フォーム BindingSource コンポーネントで ADO.NET データを並べ替える/フィルタ処理する

BindingSource コンポーネントを使用して、表示されたデータに並べ替えおよびフィルタを適用する方法について説明します。

方法 : Windows フォーム BindingSource を使用して Web サービスにバインドする

BindingSource コンポーネントを使用して、Web サービスにバインドする方法について説明します。

BindingNavigator コントロール

方法 : Windows フォーム BindingNavigator コントロールを使用してデータ間を移動する

BindingNavigator コントロールをデータ ソースにバインドする手順について説明します。

方法 : Windows フォームの BindingNavigator コントロールを使用して DataSet を移動する

BindingNavigator コントロールを使用して、DataSet でレコード間を移動する方法について説明します。

方法 : Windows フォーム BindingNavigator コントロールに [Load]、[Save]、[Cancel] の各ボタンを追加する

データにテキスト ボックス コントロールをバインドし、次に、フォームに追加された **ToolStrip** コントロールを編集して読み込み、保存、およびキャンセルのためのボタンを追加します。

印刷

方法 : 標準の Windows フォーム印刷ジョブを作成する

PrintDocument コンポーネントを使用してフォームで印刷を行う方法について説明します。

方法 : Windows フォームの印刷ジョブを完了する

ユーザーに印刷ジョブの完了を通知する方法について説明します。

方法 : Windows フォームで複数ページのテキスト ファイルを印刷する

プリンタにテキストを送る方法について説明します。

方法 : PrintDialog コンポーネントを表示する

ダイアログ ボックスを表示する方法、およびプロパティが保存される場所について説明します。

方法 : Windows フォーム アプリケーションに印刷プレビューを表示する

PrintPreviewDialog コントロールを表示する方法について説明します。

方法 : Windows フォームでユーザーのコンピュータに接続されたプリンタを選択する

実行時に PrintDialog コンポーネントを使用してプリンタを変更する方法について説明します。

方法 : 実行時に PrintDialog のユーザー入力をキャプチャする

選択した印刷オプションを **PrintDialog** コンポーネントを使用してプログラムで変更する方法について説明します。

ユーザー コントロールとカスタム コントロール

チュートリアル : Visual Basic による複合コントロールの作成

単純なユーザー コントロールを作成し、継承を使用してその機能を拡張します。

方法 : UserControl の実行時の動作をテストする

Visual Studio に用意されている UserControl テスト コンテナの使用法を示します。

Visual Studio に用意されている UserControl クラスの使用法を説明します。

方法 : UserControl クラスを継承する

UserControl クラスを使用した継承を示します。

方法 : 複合コントロールを作成する

ユーザー コントロールおよびコントロール クラス ライブラリの作成方法と、ユーザー コントロールからの継承方法を示します。

ユーザー コントロールへのコントロールの追加

ユーザー コントロールの基本的な操作方法について説明します。

ユーザー コントロールへのコードの追加

ユーザー コントロールの基本的な操作方法について説明します。

マルチ ドキュメント インターフェイス (MDI) アプリケーション

方法 : MDI 親フォームを作成する

MDI アプリケーション内に複数のドキュメントのコンテナを作成する方法について説明します。

方法 : MDI 子フォームを作成する

MDI 親フォーム内で動作するウィンドウを 1 つ以上作成する方法について説明します。

方法 : MDI 子フォームを配置する

MDI アプリケーションの子ウィンドウを並べて表示したり、重ねて表示したり、並べ替えたりする方法について説明します。

方法 : アクティブな MDI 子フォームを特定する

フォーカスを持つ子ウィンドウを確認する方法、およびそのウィンドウの内容をクリップボードに送る方法について説明します。

方法 : アクティブな MDI 子フォームにデータを送信する

アクティブな子ウィンドウに情報を転送する方法について説明します。

方法 : MenuStrip を使用して MDI ウィンドウの一覧を作成する (Windows フォーム)

MDI を使用して、親フォームの [ウィンドウ] メニューに、すべてのアクティブな子フォームの一覧を作成する方法について説明します。

グラフィックス

方法 : 形状のアウトラインを描画する

図形を描画する方法について説明します。

方法 : 線形グラデーションを作成する

LinearGradientBrush クラスを使用して線形グラデーションを作成する方法を示します。

方法 : パス グラデーションを作成する

PathGradientBrush クラスを使用してパス グラデーションを作成する方法を説明します。

方法 : 直線、曲線、および形状から図形を作成する

GraphicsPath を使用して図形を作成する方法を示します。

方法 : 描画する Graphics オブジェクトを作成する

Graphics オブジェクトを作成して描画を行う方法について説明します。

方法 : サムネイル イメージを作成する

サムネイル イメージの作成方法について説明します。

方法 : 垂直方向のテキストを作成する

GDI+ でテキストを垂直方向に配置して描画する方法について説明します。

方法 : 描画テキストを配置する

GDI+ でのテキストの書式設定の方法について説明します。

方法 : Windows フォームに直線を描画する

線を描画する方法について説明します。

方法 : イメージを回転、反転、および傾斜させる

回転、反転、および傾斜させたイメージを描画する方法について説明します。

方法 : Windows フォームにテキストを描画する

テキストを描画する方法について説明します。

方法 : ビットマップを読み込んで表示する

ビットマップを読み込む方法と描画する方法について説明します。

方法 : メタファイルを読み込んで表示する

メタファイルを読み込む方法と描画する方法について説明します。

方法 : イメージをトリミングおよびスケーリングする

ベクタ イメージおよびラスタ イメージのトリミングとスケール調整を行う方法について説明します。

Windows フォームのローカライズとグローバライズ

チュートリアル : Windows フォームのローカリゼーション

Windows Application プロジェクトのリソース ファイルを作成および使用する方法について説明します。

チュートリアル : ローカライズの際に均等に調整されるレイアウトの作成

表示される文字列値を他の言語に翻訳するときに均等に調整されるレイアウトを作成する方法について説明します。

方法 : AutoSize と TableLayoutPanel コントロールを使用して Windows フォームのローカリゼーションをサポートする

デザイン時に予想できないテキスト文字列の長さの違いに対応するレイアウトを作成する方法について説明します。

方法 : Windows フォームのグローバリゼーション用のカルチャおよび UI カルチャを設定する

CurrentCulture プロパティと **CurrentUICulture** プロパティを設定して、どのリソースをアプリケーションに読み込むか、また情報をどのように書式設定するかを設定する方法を説明します。

方法 : グローバリゼーション用に Windows フォームで右から左の方向でテキストを表示する

Windows フォーム上で右から左に読むテキストを表示する方法について説明します。

方法 : ローカリゼーションに対応した Windows フォーム レイアウトをデザインする

TableLayoutPanel コントロールを使用して、ローカリゼーションに適切に対応できるフォームを作成する例を示します。

アプリケーション リソースの管理

方法 : Visual Basic で文字列リソースを取得する

My.Resources オブジェクトを使用した、文字列リソースのアクセスおよび取得の方法について説明します。

方法 : Visual Basic でクリップボードからイメージを取得する

クリップボードからイメージを取得する方法を説明します。

方法 : リソースを追加または削除する

リソース デザイナを使用して、プロジェクトにリソースを追加または削除する方法について説明します。

方法 : 文字列リソースを追加または削除する

リソース デザイナの文字列ペインを使用して、プロジェクトに文字列リソースを追加または削除する方法について説明します。

ファイル、フォルダ、およびドライブの操作

方法 : Visual Basic でファイルを作成する

ファイルを作成する方法を説明します。

方法 : Visual Basic でファイルを移動する

ファイルを別のディレクトリに移動する方法を説明します。

方法 : Visual Basic でファイルの名前を変更する

ファイルの名前を変更する方法を説明します。

方法 : Visual Basic でファイルを削除する

ファイルを削除する方法を説明します。

方法 : Visual Basic でファイルのコピーを同じディレクトリに作成する

ファイルのコピーを同じディレクトリに作成する方法を説明します。

ファイルからの読み取り

方法 : StreamReader を使用してファイルからテキストを読み取る (Visual Basic)

StreamReader を使用してファイルを読み取る方法をデモンストレーションします。

方法 : Visual Basic でテキスト ファイルを読み取る

テキスト ファイルからの読み取り方法をデモンストレーションします。

方法 : My Documents の既存のテキスト ファイルを読み取る (Visual Basic)

My Documents ディレクトリのテキスト ファイルを読み取る方法をデモンストレーションします。

方法 : Visual Basic でバイナリ ファイルを読み取る

バイナリ ファイルからの読み取り方法をデモンストレーションします。

ファイルの操作

方法 : Visual Basic でファイルの拡張子を確認する

ファイルの拡張子を確認する方法を説明します。

方法 : Visual Basic でファイルの作成時刻を確認する

ファイルの作成時刻を確認する方法を説明します。

方法 : Visual Basic でファイルの絶対パスを確認する

ファイルの絶対パスを確認する方法を説明します。

方法 : Visual Basic でファイルが隠しファイルかどうかを調べる

ファイルが隠しファイルかどうかを確認する方法を説明します。

方法 : Visual Basic でファイルが存在するかどうかを確認する

ファイルが存在するかどうかを確認する方法を説明します。

方法 : Visual Basic でファイル パスを解析する

My メソッドを使用してファイル パスを結合する方法を説明します。

方法 : Visual Basic でファイル名とパスを検証する

文字列がファイル名とパスのどちらを表すのかを確認する方法を説明します。

ドライブの操作

方法 : Visual Basic でドライブのボリューム ラベルを確認する

ドライブのボリューム ラベルを確認する方法を説明します。

方法 : Visual Basic でドライブの種類を確認する

ドライブの種類を確認する方法を説明します。

方法 : Visual Basic でドライブの合計領域を確認する

ドライブの合計領域を確認する方法を説明します。

方法 : Visual Basic でドライブのルート ディレクトリを確認する

ドライブのルート ディレクトリを確認する方法を説明します。

ファイルへの書き込み

方法 : Visual Basic でバイナリ ファイルに書き込む

バイナリ ファイルへの書き込み方法をデモンストレーションします。

方法 : Visual Basic で My Documents ディレクトリのファイルにテキストを書き込む

My Documents ディレクトリに新しいテキスト ファイルを作成して書き込む方法をデモンストレーションします。

方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

StreamWriter オブジェクトでのファイルへの書き込み方法をデモンストレーションします。

フォルダの操作

方法 : Visual Basic でディレクトリを作成する

ディレクトリを作成する方法を説明します。

方法 : Visual Basic でディレクトリの属性を確認する

ディレクトリの属性を確認する方法を説明します。

方法 : Visual Basic でディレクトリを移動する

ディレクトリを移動する方法を説明します。

方法 : Visual Basic でディレクトリが存在するかどうかを確認する

ディレクトリが存在するかどうかを確認する方法を説明します。

方法 : Visual Basic でディレクトリを削除する

ディレクトリを削除する方法を説明します。

イベントのログ記録とトレース

チュートリアル : My.Application.Log の出力のフィルタ処理

My.Application.Log ログの出力をフィルタ処理する方法について説明します。

チュートリアル : My.Application.Log による情報の書き込み先の確認

My.Application.Log が情報を書き込む場所を判断する方法を説明します。

チュートリアル : カスタム ログ リスナの作成

My.Application.Log 用のカスタム ログ リスナを作成する方法について説明します。

方法 : ログ メッセージを書き込む

イベント情報をアプリケーションのログに書き込む方法について説明します。

方法 : アプリケーション イベント ログに書き込む

イベント ログへ情報を書き込むように **My.Application.Log** を構成する方法について説明します。

方法 : Visual Basic で例外をログに記録する

例外情報をアプリケーションのログに書き込む方法について説明します。

アプリケーション テンプレートの使用

方法 : Windows アプリケーション プロジェクトを作成する

統合開発環境 (IDE: Integrated Development Environment) を使用して Windows アプリケーション プロジェクトを作成する方法について説明します。

COM との相互運用

チュートリアル : COM オブジェクトによる継承の実装

既存の COM オブジェクトを新しいオブジェクトの基礎として使用する方法について説明します。

チュートリアル : Visual Basic 2005 での COM オブジェクトの作成

COM クラス テンプレートを使用する場合と使用しない場合の COM オブジェクトの作成手順を説明します。

方法 : Visual Basic から COM オブジェクトを参照する

タイプ ライブラリのある COM オブジェクトに参照を追加する方法を説明します。

方法 : オブジェクトの現在のインスタンスを参照する

Me キーワードを使用して、コードが現在実行しているインスタンスを参照する方法について説明します。

アプリケーションの設定の管理

方法 : Visual Basic でアプリケーション設定を読み取る

アプリケーションのフォームにアクセスして、ユーザー設定の値を調べる方法を説明します。

方法 : Visual Basic でユーザー設定を永続化する

アプリケーションのフォームにアクセスして、更新後のユーザー設定の値を保存する方法を説明します。

方法 : アプリケーション設定を追加または削除する

プロジェクト デザイナの [設定] ページを使用して、アプリケーション設定を追加または削除する方法について説明します。

コンピュータ リソースへのアクセス

方法 : Visual Basic でシリアル ポートから文字列を受信する

コンピュータのシリアル ポートから文字列を受信する方法を説明します。

方法 : Visual Basic で利用可能なシリアル ポートを表示する

利用可能なシリアル ポートを示す方法を説明します。

方法 : アプリケーションを起動してキーストロークを送る (Visual Basic)

アプリケーションを起動して、キーストロークをアプリケーションに送信する方法について説明します。

クリップボードの使用

方法 : Visual Basic でクリップボードに書き込む

クリップボードにデータを書き込む方法を説明します。

方法 : Visual Basic でクリップボードからイメージを取得する

クリップボードからイメージを取得する方法を説明します。

方法 : Visual Basic でクリップボードから読み込む

クリップボードからデータを読み込む方法を説明します。

サウンドの再生

方法 : Visual Basic でオーディオ リソースを取得する

My.Resources オブジェクトを使用して、オーディオ リソースを取得する方法を説明します。

方法 : Visual Basic でシステム サウンドを再生する

My.Computer.Audio オブジェクトを使用して、システム サウンドを再生する方法を説明します。

方法 : Visual Basic でサウンドを再生する

My.Computer.Audio オブジェクトを使用して、サウンド ファイルまたはアプリケーション リソースのサウンドをバックグラウンドで再生する方法を説明します。

方法 : Visual Basic でループ サウンドを再生する

My.Computer.Audio オブジェクトを使用して、サウンド ファイルまたはアプリケーション リソースのサウンドをループで継続して再生する方法を説明します。

方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する

My.Computer.Audio オブジェクトを使用して、バックグラウンドで再生しているサウンドを停止する方法を説明します。

レジストリの操作

チュートリアル : レジストリ キーの作成と値の変更

My.Computer.Registry オブジェクトを使用して、レジストリ キーを作成し、その値を設定する方法を説明します。

方法 : Visual Basic でレジストリ キーの値を設定する

My.Computer.Audio オブジェクトを使用して、レジストリ キーに値を設定する方法を説明します。

方法 : Visual Basic で、レジストリ キーから値を読み取る

My.Computer.Audio オブジェクトを使用して、レジストリ キーから値を読み取る方法を説明します。

方法 : Visual Basic で、レジストリ キーを削除する

My.Computer.Audio オブジェクトを使用して、レジストリ キーを削除する方法を説明します。

方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する

My.Computer.Audio オブジェクトを使用して、特定のレジストリ キーに値が存在するかどうかを判断する方法を説明します。

イベント ログの使用

チュートリアル : イベント ログ、イベント ソース、およびエントリの基礎

Visual Studio アプリケーションにおけるイベント ログの主要な機能を見ていきます。

方法 : イベント ログを消去する

イベント ログからエントリを削除する方法を説明します。

方法 : カスタム イベント ログを作成または削除する

ローカル コンピュータでカスタム イベント ログを作成する方法について説明します。

方法 : イベント ログを削除する

既存のログとそのすべてのエントリを削除する方法を説明します。

方法 : イベント ログ エントリを読み込む

イベント ログのエントリを読み取る方法を説明します。

方法 : イベント ソースの有無を確認する

特定のログが存在するかどうかを問い合わせる方法を説明します。

ネットワークの使用

方法 : Visual Basic でファイルをアップロードする

My.Computer.Network を使用してファイルをアップロードし、リモートの場所に格納する方法を説明します。

方法 : Visual Basic で接続ステータスをチェックする

コンピュータが有効なネットワーク接続を持つかどうかを判断する方法を示します。

方法 : Visual Basic でリモート コンピュータが利用可能かどうかを確認する

Ping メソッドを使用して、リモート コンピュータまたはホストが利用可能かどうかを判断する方法を示します。

ソース コード管理

方法 : ソース管理からプロジェクトまたはソリューションを開く

ソース管理からプロジェクトまたはソリューションを開く方法を説明します。

方法 : プロジェクトをソース管理に追加する

ソース管理にプロジェクトを追加する手順について説明します。

安全なアプリケーションの作成

チュートリアル : パスワードの複雑さの検証 (Visual Basic).

文字列が強力なパスワードの特徴を備えているかどうかを確認する方法を説明します。

チュートリアル : カスタムの認証および承認の実装

カスタムの認証および承認を実装する方法、およびアプリケーション スレッドの既定の ID をオーバーライドする方法を説明します。

チュートリアル : Visual Basic での文字列の暗号化と複号化

DES (Data Encryption Standard) アルゴリズムを使用した、文字列の暗号化と復号化の方法について説明します。

方法 : ユーザーがグループに属しているかどうかを確認する

My.User オブジェクトを使用して、ユーザーのロールを確認する方法を説明します。

方法 : ユーザーのログイン名を確認する

My.User オブジェクトを使用して、ユーザーのログイン名を取得する方法を説明します。

サンプル

Visual Basic Windows フォームのサンプル

これらのサンプルでは、Windows フォーム アプリケーションの例を示します。

参照

概念

Visual Basic での操作方法

セキュリティ (Visual Basic での操作方法)

ここでは、Visual Basic のセキュリティに関連するタスク全般について扱ったヘルプへのリンクを紹介しています。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

セキュリティと Visual Basic 開発

セキュリティ上の懸念への対処が必要な、よくある 3 つの状況について説明します。

Visual Studio におけるセキュリティ

安全なコーディング技法など、セキュリティの基本概念に関する情報へのリンクを示します。

コード アクセス セキュリティの基礎

コード アクセス セキュリティとその最も一般的な使用方法について説明します。

ロール ベース セキュリティ

.NET Framework のロール ベース セキュリティの概要を示します。

セキュリティ ポリシーの実施

Windows 開発者を対象に、安全なソフトウェアを開発するための推奨される手順を紹介します。

Microsoft Security Developer Center

デスクトップ レベルからネットワーク レベルまでのシステムを対象に、アプリケーションやシステムを安全な状態に保つための Microsoft の取り組みについて紹介しています。IT 専門家、開発者、およびホーム ユーザー向けのセキュリティ リソースへのリンクも用意されています。

方法 : データベース接続を変更する

データベース接続文字列の格納方法などの問題について説明します。

最適な使用方法

セキュリティ ポリシーの実施

Windows 開発者を対象に、安全なソフトウェアを開発するための推奨される手順を紹介します。

Web セキュリティ

ASP.NET を使用して作成した XML Web サービスのセキュリティ

ASP.NET Web サービスの認証と承認のしくみについて説明します。

ASP.NET Web サイトのセキュリティ (Visual Studio)

ここでは、さまざまな種類のセキュリティを組み込んだ ASP.NET アプリケーションの作成方法の例を示します。

ASP.NET Web サイトへのアクセスの制限

Web プロジェクト ファイルにアクセスできるユーザー、およびそれぞれのユーザーに許可するアプリケーションへのアクセス レベルを制御する方法について説明します。

Windows のセキュリティ

Windows フォームのセキュリティの概要

.NET Framework のセキュリティ モデルについて簡単に説明し、アプリケーション内の Windows フォームの安全性を確保するための基本的な手順を示します。

Windows フォームでのより安全な印刷

やや信頼度の低い環境で印刷機能にアクセスする方法について説明します。

Windows フォームにおけるファイルおよびデータへのより安全なアクセス

やや信頼度の低い環境でファイルとデータにアクセスする方法について説明します。

ClickOnce の配置とセキュリティ

ClickOnce 技術を使用して配置されたアプリケーションのアクセス許可に関する問題について説明します。このようなアプリケーションは、セキュリティ ゾーンに基づく制限付きのアクセス権で "サンドボックス" の中で実行されます。

コンポーネントのセキュリティ
コード アクセス セキュリティ

コード セキュリティの概念を示し、コンポーネントのセキュリティに対する .NET Framework のアプローチについて説明します。

強制セキュリティ

強制セキュリティ チェックを使用して権限のない用途から機密性の高いコードを保護する方法について説明します。

データベース セキュリティ
方法 : データベース接続を変更する

データベース接続文字列の格納方法などの問題について説明します。

サンプル
Visual Basic のセキュリティのサンプル

これらのサンプルでは、セキュリティ タスクの例を示します。

参照
概念
Visual Basic での操作方法

デバッグとテスト (Visual Basic での操作方法)

ここでは、Visual Basic のビルド、デバッグ、およびテストに関連するタスク全般について扱ったヘルプへのリンクを紹介しています。その他、ヘルプでカバーされている一般的なタスク カテゴリについては、「Visual Basic での操作方法」を参照してください。

全般

チュートリアル : Windows フォームのデバッグ

Windows フォーム アプリケーションをデバッグする方法について説明します。

チュートリアル : Web フォームのデバッグ

Web フォームをデバッグする方法について説明します。

チュートリアル : XML Web サービスのデバッグ

XML Web サービスをデバッグする方法について説明します。

アプリケーションの構築

方法 : ビルドの準備および管理

ビルド順序と保存オプションの設定方法、さらにソリューションまたはプロジェクトのビルドとリビルドの方法について説明します。

方法 : Visual Basic 開発者設定が適用されたビルド構成を管理する

Visual Basic のビルド構成情報を作成および編集する方法について説明します。

方法 : プロジェクトを構成して対象プラットフォームを設定する

指定のプラットフォームを対象とするプロジェクトを構成する方法を説明します。

アプリケーションのデバッグ

Visual Basic アプリケーションのデバッグ

Visual Studio に組み込まれているデバッグ機能に関するドキュメントへのリンクを示します。

方法 : 実行を開始する

デバッグの起動方法について説明します。

方法 : コードにステップ インする

コードを 1 行ずつ実行する方法について説明します。

方法 : [プロセス] ウィンドウを使用する

[プロセス] ウィンドウを使用する方法について説明します。

方法 : 単純なブレークポイントを設定する

アプリケーションを特定のポイントで停止させる方法について説明します。

方法 : [クイック ウォッチ] ダイアログ ボックスを使用する

変数と式をすばやく調べたり評価したりする方法を説明します。

方法 : エディット コンティニュの中断モード時に編集を適用する

コードを中断モードで編集し、実行を停止および再開することなく継続させる方法について説明します。

方法 : 例外処理アシスタント を使用してランタイム エラーを修正する

コードが例外を受け取った場合の、トラブルシューティングのためのヒントを得る方法について説明します。

方法 : 部分信頼アプリケーションをデバッグする

ClickOnce を使用して配置する予定の、部分信頼アプリケーションのデバッグ方法について説明します。

アプリケーションのテスト

方法 : Visual Basic でプロジェクトをコンパイルして実行する

開発環境内からのコンパイルおよび実行に使用するオプションについて説明します。

マネージ コード障害の検出と修正

Microsoft Visual Studio 2005 Team Edition for Software Developers を使用して、コードの欠陥を検出および修正するための方法を示します。

監視アプリケーション

チュートリアル : イベント ログ、イベント ソース、およびエントリの基礎

重要なイベントに関する情報をアプリケーションおよびコンポーネントで記録する方法について説明します。これらの記録を使用して、システムへのアクセスの監査、問題のトラブルシューティング、および使用法の再現を行うことができます。

方法 : アプリケーションのコードをトレースする

実装およびトレースにおける主要な手順について説明します。

方法 : トレースとデバッグを指定して条件付きコンパイルを実行する

コンパイル済みのアプリケーションにトレース オプションを組み込む方法について説明します。

参照

概念

Visual Basic での操作方法

Visual Basic ガイド ツアー

Visual Basic 2005 を初めて使用する人も、コンピュータ プログラミング自体が初体験の人も、ここがスタートラインです。Visual Basic ガイド ツアーは、Visual Basic 2005 によるプログラミングの基本について説明する一連のレッスンです。

このセクションの内容

Visual Basic Express の概要

Microsoft Visual Basic 2005 Express Edition と Visual Basic 開発環境について説明する一連のトピックです。

初めての Visual Basic プログラムの作成

このセクションでは、Web ページを表示するプログラムの作成がいかに簡単かを示します。

Visual Basic プログラミング言語の概要

このセクションでは、Visual Basic 言語の基本について説明します。

プログラムの外観の作成 : Windows フォームの概要

このセクションでは、フォームおよびコントロールを使用してユーザー インターフェイスを作成する方法について説明します。

問題の確認 : デバッグを行ってエラーを検出し、修正する

このセクションでは、Visual Basic 2005 のデバッグ ツールの使用方法について説明します。

レコードの管理 : プログラムでデータを使用する

このセクションでは、データベースの操作方法について説明します。

オブジェクトによるプログラミング : クラスを使用する

このセクションでは、オブジェクト指向プログラミングの基本について説明します。

表示されるオブジェクト : 初めてのユーザー コントロールの作成

このセクションでは、再利用可能な独自のコントロールを作成する方法について説明します。

ピクチャの描画 : グラフィックスを使用する

このセクションでは、Visual Basic 2005 のグラフィックス機能について説明します。

プログラムの配布

このセクションでは、作成したプログラムを他の人と共有する方法について説明します。

さらに上のステップへ : 次の学習

Visual Basic 2005 を習得するための追加リソースにリンクする一連のトピックです。

Visual Basic Express での操作方法

タスク固有の情報にリンクする一連のトピックです。

参照

Visual Basic リファレンス

Visual Basic のすべてのプログラミング可能な要素について説明します。

Visual Basic Express の概要

Microsoft Visual Basic 2005 Express Edition は、アップグレードを考えている Visual Basic 6.0 上級者、Visual Basic を初めて習得しようと考えているベテラン プログラマ、コンピュータ プログラミングが初めてのユーザーなど、経験度合いに関係なくすべてのユーザーを対象にデザインされています。Microsoft Visual Basic 2005 Express Edition は、Visual Basic の必要最低限の機能を装備するだけでなく、プログラミングをこれまでになく簡単で楽しいものにするために作成されました。

Visual Basic ガイド ツアーには、Visual Basic 言語と Visual Basic Express Edition ツールの両方を迅速に習得するための一連のレッスンが含まれています。開始する前に、このセクションのトピックを参照すると、Visual Basic Express Edition の機能に慣れておくことができます。プログラミングが初めての場合は、「初めての Visual Basic プログラムの作成」に進んでから、後でこれらのトピックに戻ることもできます。

このセクションの内容

Visual Basic Express の概要

Visual Basic Express Edition は、単なる学習ツールにとどまらず、Windows フォーム アプリケーション、コンソール アプリケーション、およびクラス ライブラリの作成に関心がある初心者のプログラマとプログラム愛好家に、本格的な開発環境を用意しています。

Visual Basic Express ツールの概要

Visual Basic 言語を使用するアプリケーションを作成するためのプログラムである Visual Basic Express Edition の概要を説明します。

方法 : 既存のアプリケーションを操作する

既に Visual Basic に関する知識があり、操作対象のアプリケーションがある場合は、既存のアプリケーションを開くことは簡単です。

スタート キット : いますぐ始めよう

Visual Basic Express Edition には、プログラムをすぐに作成できるスタート キットが用意されています。

Visual Basic Express のプロジェクトの種類

Visual Basic Express Edition を使用すると、Windows フォーム アプリケーション、コンソール アプリケーション、クラス ライブラリなど、複数の種類のアプリケーションを作成できます。

Visual Basic Express のトラブルシューティング

Visual Basic Express Edition によってプログラミングは楽になりますが、必ずしも期待どおりに動作しない場合があります。

関連するセクション

Visual Basic ガイド ツアー

Visual Basic ガイド ツアーは、Visual Basic 2005 によるプログラミングの基本について説明する一連のレッスンです。

初めての Visual Basic プログラムの作成

Visual Basic を使用したプログラミングを習得する最適な方法は、実際にプログラムを作成することです。

さらに上のステップへ : 次の学習

これで、「Visual Basic ガイド ツアー」のレッスンは完了しました。先に進みましょう。

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic 6.0 の知識を活用することにより、Visual Basic 2005 を使用してすぐに成果を上げることができます。

Visual Basic 6.0 ユーザー向けの新機能

Visual Basic 6.0 を使い慣れているユーザーにとって、Visual Basic 2005 は多くの新機能や大幅に改良された機能を備えているため、以前のバージョンを使用するのに比べて、Visual Basic での開発はより簡単かつ高性能になっています。

Visual Basic Express の概要

想像しているコンピュータ プログラムがあるならば、そのほとんどは Microsoft Visual Basic 2005 Express Edition を使用して作成できます。メッセージを表示する単純なプログラムから、データベースや Web ページにアクセスする本格的なアプリケーションに至るまで、Visual Basic Express Edition には、その作成に必要なツールが用意されています。

Visual Basic Express Edition は、学習ツールというだけでなく、初めてプログラミングを行う人にも、Windows フォーム アプリケーションやコンソール アプリケーション、クラス ライブラリのビルドに興味を持つ愛好家にも、完成度の高い機能的な開発環境を提供します。スタート キットやその他の入門用機能により、初めてプログラミングを行う人でも、Visual Basic の多くの機能を利用できるようになっています。

Visual Basic Express について

Visual Basic Express Edition は Visual Basic の低価格バージョンです。Visual Basic でのプログラミングを学ぶためのツールであると同時に、完全バージョンの Visual Basic は必要ないプログラマ向けの、十分な機能を備えた開発ツールでもあります。しかし、Visual Basic Express Edition は、Visual Basic のサブセットというだけではありません。Visual Basic によるプログラミングをこれまでになく簡単にする多くの機能を備えています。

Visual Basic Express Edition で実行できる処理について理解するための最良の方法は、「Visual Basic ガイド ツアー」の一連のレッスンを終了することです。これらのレッスンを終了すると、Visual Basic のツールと概念について十分な知識を得ることができ、独自のプログラムを記述できるようになります。

Visual Basic Express の対象ユーザー

Visual Basic Express Edition は、完全に機能するアプリケーションやコンポーネントを作成して他の人と共有できる、強力なツールです。ただし、プロの開発者やチーム環境で作業するプログラマのためのものではありません。他のバージョンの Visual Basic には、プロフェッショナルな開発やチーム開発の高度なニーズに対応する機能が備わっています。

作成するアプリケーションで、ネットワーク データベースへの接続、Microsoft Office とのやり取り、モバイル デバイスや 64 ビット オペレーティング システムのサポート、またはリモート デバッグが必要な場合には、より高度なバージョンの Visual Basic が必要です。

メモ：
Visual Basic Express Edition は Web アプリケーションの開発をサポートしません。Web 開発を行う場合には、Visual Web Developer Express Edition をダウンロードできます。

ヘルプの表示

Visual Basic Express Edition に含まれているヘルプ ファイルは、Visual Studio 2005 Express Edition の MSDN ライブラリのサブセットであり、これはさらに、完全版の MSDN ライブラリのサブセットです。インターネットに接続している場合、完全版のライブラリのすべてのヘルプ トピックにアクセスできます。オンライン アクセスがない場合や、Visual Studio 2005 Express Edition の MSDN ライブラリをインストールしなかった場合には、一部のヘルプ トピックを利用できないことがあります。詳細については、「Visual Basic Express のトラブルシューティング」を参照してください。

参照

概念

Visual Basic Express のプロジェクトの種類

その他の技術情報

Visual Basic Express の概要

Visual Basic ガイド ツアー

Visual Basic Express のプロジェクトの種類

Microsoft Visual Basic 2005 Express Edition では、Windows フォーム アプリケーション、コンソール アプリケーション、クラス ライブラリなど、数種類のアプリケーションを作成できます。作成する各アプリケーションは、それぞれのプロジェクトに格納されます。またプロジェクトを作成しやすくするため、プロジェクト テンプレートが用意されています。

新しいプロジェクトを作成するときに、[新しいプロジェクト] ダイアログ ボックスおよび [プロジェクトの追加] ダイアログ ボックスに表示されるアイコンは、使用できるプロジェクトの種類とそのテンプレートを表しています。Visual Basic Express Edition で新しいプロジェクトを開くときに使用できるプロジェクト テンプレートを、次に示します。

プロジェクトの種類

プロジェクト テンプレート	使用に適した状況
Windows アプリケーション テンプレート	ユーザーのマシンでローカルに実行される Windows ベースのアプリケーションを作成するために使用します。Windows 電卓のような単純な単一ウィンドウのツールから、複数のウィンドウと高度な機能を備えた本格的なアプリケーションまで、任意のプログラムをビルドできます。
クラス ライブラリ テンプレート	他のプロジェクトと共有できる再利用可能なクラスやコンポーネントを作成するために使用します。
コンソール アプリケーション テンプレート	Windows コマンド プロンプトから実行され、ビジュアル インターフェイスを持たないプログラムであるコマンド ライン アプリケーションを作成するために使用します。
ムービー コレクション スタート キット テンプレート	あらかじめビルドされた My Movie Collection サンプル アプリケーションを作成するために使用します。このアプリケーションは必要に応じてカスタマイズできます。詳細については、「スタート キット : いますぐ始めよう」を参照してください。
スクリーン セーバー スタート キット テンプレート	あらかじめビルドされた Screen Saver サンプル アプリケーションを作成するために使用します。このアプリケーションは必要に応じてカスタマイズできます。詳細については、「スタート キット : いますぐ始めよう」を参照してください。

メモ :

Windows コントロール ライブラリのプロジェクト テンプレートはありませんが、クラス ライブラリ テンプレートを使用して独自のコントロールを作成できます。詳細については、「表示されるオブジェクト : 初めてのユーザー コントロールの作成」を参照してください。

参照

概念

Visual Basic Express の概要

その他の技術情報

Visual Basic ガイド ツアー

Visual Basic Express ツールの概要

このトピックでは、Visual Basic 言語を使用してアプリケーションを作成するためのプログラムである Visual Basic Express Edition の概要を説明します。Microsoft Outlook のようなプログラムが、電子メールを扱うためのさまざまなツールを備えているのと同様に、Visual Basic Express Edition は、さまざまなプログラミング タスクを実現するツールキットです。

ヒント

プログラミングに取り組むのが始めての場合は、まず Visual Basic ガイド ツアー (基礎について説明することを目的とした一連のレッスン) を完了してから、このトピックに戻ることもできます。ツアーを開始するには、「初めての Visual Basic プログラムの作成」を参照してください。

開発プロセス

Visual Basic Express Edition では、アプリケーション開発のプロセスは簡単です。大半の場合、開発プロセスは以下の手順で構成されます。

- **プロジェクトを作成します。** プロジェクトには、アプリケーションに必要なすべてのファイルが含まれ、アプリケーションについての情報を保持します。1 つのアプリケーションに、複数のプロジェクトが含まれることもあります。たとえば、1 つの Windows アプリケーション プロジェクトと、1 つまたは複数のクラス ライブラリ プロジェクトで構成される場合です。こうしたアプリケーションはソリューションと呼ばれます。これは、プロジェクトのグループの別の呼び名です。
- **ユーザー インターフェイスをデザインします。** 具体的には、ボタンやテキスト ボックスなどのさまざまなコントロールを、フォームというデザイン サーフェイスにドラッグします。その後で、フォームおよびそのコントロールの外観や動作を定義するプロパティを設定します。

  メモ：

  クラス ライブラリやコンソール アプリケーションなど、ユーザー インターフェイスを持たないアプリケーションの場合は、この手順は必要ありません。

- **コードを記述します。** 続いて、アプリケーションの動作やユーザーとのやり取りの仕方を定義する Visual Basic コードを記述する必要があります。Visual Basic Express Edition では、IntelliSense、オート コンプリート、コード スニペットなどの機能を使用して、コードを簡単に記述できます。
- **コードをテストします。** アプリケーションは、意図したとおりに動作することを確認するよう、必ずテストする必要があります。このプロセスはデバッグと呼ばれます。Visual Basic Express Edition には、コード内のエラーを対話形式で簡単に見つけて修復できるデバッグ ツールが備わっています。
- **アプリケーションを配布します。** アプリケーションが完成したら、できあがったプログラムを自分のコンピュータにインストールしたり、他の人に配布して共有できます。Visual Basic Express Edition では、ClickOnce 発行という新しい技術を使用します。この技術では、ウィザードによってアプリケーションを簡単に配置でき、また、後でアプリケーションに変更を加えたときに、更新バージョンを自動的に提供できます。

各部分の説明

Visual Basic Express Edition のユーザー インターフェイスは統合開発環境 (IDE) とも呼ばれます。このユーザー インターフェイスを一見すると、なじみがないように感じられるかもしれませんが、いったん使い方に慣れれば、簡単に使用できることがわかります。以下のセクションでは、IDE の中で特によく利用する各部分について説明します。

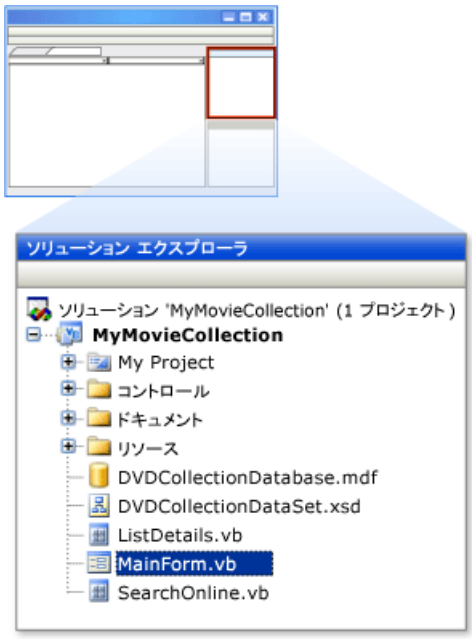
起動時

Visual Basic Express Edition を最初に開くと、IDE の大半を占める形で [スタート ページ] ウィンドウが表示されます。スタート ページには、最近使用したプロジェクトのクリック可能な一覧、重要なヘルプ トピックへのリンクを示す作業の開始領域、およびオンライン文書やその他のリソースへのリンクの一覧が含まれます。インターネットに接続している場合、この一覧は定期的に更新されます。

スタート ページに表示する内容は、自分の好みに合わせて変更できます。詳細については、
「方法 : スタート ページのニュース セクションをカスタマイズする」を参照してください。

IDE の右側には、[ソリューション エクスプローラ] ウィンドウが表示されます。最初は空ですが、プロジェクトやソリューション (プロジェクトのグループ) についての情報はここに表示されます。詳細については、「ソリューション エクスプローラの使用」を参照してください。

図 1 : ソリューション エクスプローラ



IDE の左側には、[ツールボックス] という垂直のタブが表示されます。こちらも最初は空ですが、操作を進めていくと、現在扱っているタスクに使用できる項目が表示されます。詳細については、「ツールボックスの使用」を参照してください。

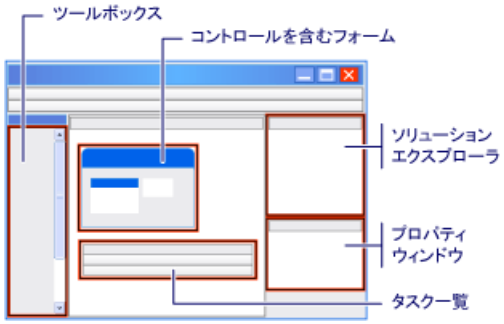
IDE の上部には、メニュー バーとツール バーが表示されます。利用できるメニューとツール バー ボタンは、現在のタスクに応じて変化します。いろいろ試してみて、どんな選択肢があるか確認してみてください。メニューやツール バーも、自分の好みに合わせてカスタマイズできます。詳細については、「方法 : ツール バーをカスタマイズする (Visual Studio)」を参照してください。

IDE の最下部にはステータス バーがあり、[コマンド] と表示されています。IDE で操作を進めると、ステータス バーは変化し、現在のタスクに関連するメッセージが表示されます。たとえば、プロジェクトのビルドの進行状況の情報などです。

デザイン モード

プロジェクトを開くかまたは作成すると、IDE の外観はデザイン モードに変わります。これは Visual Basic の視覚的な部分であり、アプリケーションの外観はここでデザインします。

図 2 : デザイン モードの IDE



デザイン モードでは、スタート ページの上に [フォーム デザイナ] という別のウィンドウが表示されます。これは基本的に、アプリケーションのユーザー インターフェイスを表す空のキャンバスです。スタート ページは、フォーム デザイナの上のタブをクリックすれば引き続き利用できます。

フォーム デザイナが表示されているときには、ツールボックスには、ボタン、テキスト フィールド、グリッドなどを表すいくつかのコントロールが含まれています。これらのコントロールは、フォーム上に配置して好きなように並べ替えることができます。詳細については、「Windows フォーム デザイナ」を参照してください。

また、[ソリューション エクスプローラ] ウィンドウの下に、[プロパティ] ウィンドウという新しいウィンドウが表示されています。ここで、フォームおよびそのコントロールの外観や動作を定義するプロパティを設定します。詳細については、「[プロパティ] ウィンドウ」を参照してください。

既定では表示されていませんが、IDE の下部の [タスク一覧] ウィンドウでは、完了する必要があるタスクを把握したり、プログラミングの際にメモを取ることができます。詳細については、「タスク一覧 (Visual Studio)」を参照してください。

フォームまたはコントロールをダブルクリックすると、コード エディタという新しいウィンドウが開きます。このウィンドウを使用して、アプリケーションの実際のコードを記述します。コード エディタは、単なるテキスト エディタではありません。IntelliSense という技術を使用して、入力に関連する情報を提供し、コードの記述を支援します。詳細については、「Visual Basic 固有の IntelliSense」を参照してください。

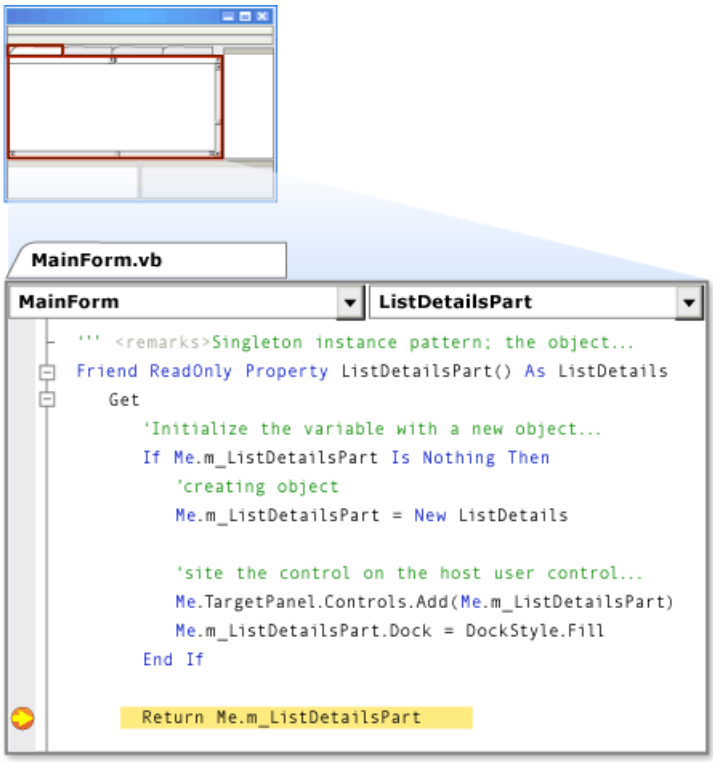
メモ :

ユーザー インターフェイスがないクラス ライブラリ プロジェクトなど、プロジェクトの種類によっては、フォーム デザイナではなくコード エディタが表示されます。

実行モード

アプリケーションを実行またはデバッグするときには、IDE は実行モードに変化します。アプリケーション自体が起動され、デバッグに関連する追加的なウィンドウが表示されます。実行モードでは、フォーム デザイナ、[プロパティ] ウィンドウ、またはソリューション エクスプローラで変更を加えることはできませんが、コード エディタでコードを変更することはできます。

図 3 : 中断モードの Visual Basic フォーム



実行モードでは、[イミディエイト] ウィンドウという新しいウィンドウが IDE の下部に表示されます。アプリケーションを中断モードにすると、[イミディエイト] ウィンドウで値を照会したり、コードをテストしたりできます。詳細については、「[イミディエイト ウィンドウ]」を参照してください。

実行時には、変数の値のウォッチ、出力の表示、およびその他のデバッグ タスクのための追加的なウィンドウを、[デバッグ] メニューから選択して表示できます。

その他の重要なウィンドウ

IDE で使用するウィンドウは他にもたくさんあり、それぞれ固有のプログラミング タスクが割り当てられています。その中で、特に一般的なものを以下に示します。

- [エラー一覧] ウィンドウは、不正なコードを入力したり、デザイン時にその他のエラーが発生したときに、IDE の下部に表示されるウィンドウです。詳細については、「[エラー一覧] ウィンドウ」を参照してください。
- [オブジェクト ブラウザ] ウィンドウは、アプリケーションで使用できる任意のオブジェクトのプロパティ、メソッド、およびイベントをチェックするために使用します。詳細については、「オブジェクト ブラウザ」を参照してください。
- [プロジェクト デザイナ] は、アプリケーションのプロパティ (リソース、デバッグの動作、配置設定など) の構成に使用します。詳細については、「プロジェクト デザイナの概要」を参照してください。
- [データベース エクスプローラ] では、既存のデータベースを参照および使用したり、新しいデータベースを作成および設計できます。詳細については、「サーバー エクスプローラ/データベース エクスプローラ」を参照してください。

カスタマイズ

Visual Basic Express Edition では、IDE をカスタマイズできます。ウィンドウのレイアウトの並べ替え、表示するウィンドウの選択、メニュー コマンドやツール バー ボタンの追加または削除など、さまざまなカスタマイズが可能です。詳細については、「開発環境のカスタマイズ」を参照してください。

ヘルプの表示

Visual Basic Express Edition を操作するときには、キーを 1 つ押すだけでいつでもヘルプを表示できます。コード エディタの使用中でも、他のウィンドウの使用中でも、F1 キーを押すと、その時点での操作に最も関連性が高いヘルプ トピックが表示されます。たとえば、コード エディタを使用していて、**Inherits** というキーワードの上にカーソルがあるときには、ヘルプ ブラウザが起動され、**Inherits** ステートメントの使用方法を説明するトピックが表示されます。

メモ：
Visual Basic Express Edition に含まれているヘルプ ファイルは、Visual Studio 2005 Express Edition の MSDN ライブラリのサブセットであり、これはさらに、完全版の MSDN ライブラリのサブセットです。インターネットに接続している場合、完全版のライブラリのすべてのヘルプ トピックにアクセスできます。オンライン アクセスがない場合や、Visual Studio 2005 Express Edition の MSDN ライブラリをインストールしなかった場合には、一部のヘルプ トピックを利用できないことがあります。

ヘルプは [ヘルプ] メニューからも起動でき、[目次] ウィンドウ、[キーワード] ウィンドウ、または [検索] タブを使用して、探しているトピックを検索できます。詳細については、「ヘルプに関するヘルプ (Microsoft Document Explorer ヘルプ)」を参照してください。

参照

その他の技術情報

初めての Visual Basic プログラムの作成
Visual Basic ガイド ツアー
Visual Basic 6.0 ユーザー向けのヘルプ
Visual Basic のプログラミング ガイド
Windows ベースのアプリケーションの作成

方法：既存のアプリケーションを操作する

既に Visual Basic に関する知識があり、操作対象のアプリケーションがある場合は、既存のアプリケーションを開くことは簡単です。

既存のアプリケーションを開くには

1. [ファイル] メニューの [プロジェクトを開く] をクリックします。
2. [プロジェクトを開く] ダイアログ ボックスで、開くプロジェクトを選択し、[開く] をクリックします。

参照

概念

ソリューション エクスプローラの使用

その他の技術情報

プロジェクト デザイナを使用したプロジェクトのプロパティの管理

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

スタート キット : いますぐ始めよう

スタート キットは、新しいツールやプログラミング技法を習得し、正常に動作するプログラムを簡単にビルドするために使用できる、テンプレートまたはモデル プログラムです。スタート キットには、コード例、ドキュメント、およびその他の役立つリソースが含まれます。

スタート キットは、そのまま使用することも、カスタマイズして使用することもできます。スタート キットを友人や同僚、または Visual Basic コミュニティのその他のメンバと簡単に共有できます。Microsoft Visual Basic 2005 Express Edition のすべてのスタート キットにチュートリアルがあり、キットをカスタマイズするのに役立ちます。また、すべてのコードには、コードの動作を理解するのに役立つコメントが含まれています。

Visual Basic スタート キット

Visual Basic Express Edition には、2 つのスタート キットが含まれています。Visual Basic スタート キットは、[ファイル] メニューの [新しいプロジェクト] ダイアログ ボックスからアクセスできます。必要なスタート キットを選択して、始めましょう。

スタート キット : マイ ムービー コレクション

マイ ムービー コレクション スタート キットは、動画コレクションを整理でき、また情報をオンラインで検索できるようにデザインされたサンプル アプリケーションです。アプリケーションは、自由にカスタマイズできます。アプリケーションをパーソナル化して、収集した他のものも整理するように変更できます。

マイ ムービー コレクションを使用すると、作成時に次の処理を実行できます。

- 動画コレクションに関する説明情報を追加します。
- 情報をデータベースに格納して保存します。
- 動画に関する情報をオンラインで検索します。

スタート キット : スクリーン セーバー

スクリーン セーバー スタート キットを使用すると、コンピュータのモニタにライブ インターネット RSS フィードを表示する Windows スクリーン セーバーを作成できます。プロジェクトは、すぐにコンパイルして実行できるようになっていますが、カスタマイズすることも簡単です。

簡単なカスタマイズには、次のようなものがあります。

- 独自に組み込んだ既定のイメージの選択
- 複数の背景イメージの設定
- RSS フィードの一覧の変更
- 独自のスクリーン セーバーの作成
- 気象情報など、XML Web サービスによる情報の表示

はじめに - スタート キットの起動

スタート キットを起動して実行するには、まずキットを読み込みます。次に、アプリケーションを実行して、実行できる単純なタスクのいくつかを確認できます。

アプリケーションを読み込んで実行するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [Visual Studio にインストールされたテンプレート] の一覧から使用するスタート キットをクリックして [OK] をクリックします。

   スタート キット プロジェクトが読み込まれます。

メモ :
[プロジェクト名] ボックスでアプリケーションのバージョンの名前を変更できます。

3. F5 キーを押してアプリケーションを実行します。
4. スタート キットのチュートリアルの手順に従って、アプリケーションを理解し、カスタマイズを行います。

参照

関連項目

[新しいプロジェクト] ダイアログ ボックス

概念

方法 : スタート キットを作成する

その他の技術情報

スタート キット

Visual Basic Express のトラブルシューティング

Visual Basic Express Edition によってプログラミングは楽になりますが、必ずしも期待どおりに動作しない場合があります。Visual Basic ガイド ツアーで学習しているとき、またはプログラムの記述に進んだときに問題が発生したら、このトピックで解決の手掛かりを得られる場合があります。

ガイド ツアーのレッスンが機能しない

ガイド ツアーのレッスンを進めるときに、プログラムが動作しないことや、意図した動作と異なることがあります。ほとんどの場合、エラーの原因は、手順を抜かしたことか、またはコードの入力ミスによるものです。

ヒント
ガイド ツアーのレッスンのコードを入力するときには、ヘルプ トピックの [コードのコピー] ボタンを使用して、コード エディタにコードを貼り付けます。その方が、自分でキー入力するより簡単で、ミスの可能性も少なくなります。

また、互換性に影響する変更が後から製品に加わったことが原因で、レッスンの手順またはコードが不正確になっている可能性もあります。こうしたトピックのオンライン バージョンは随時更新されているので、同じトピックの修正バージョンをオンラインで参照できる場合もあります。

レッスンのエラーを修正するには

1. プログラムがきちんと動作した最後の手順に戻り、それ以降の各手順に注意深く従っていきます。
2. 自分で入力したコードを見直し、レッスンのコードと完全に一致していることを確認します。
3. F5 キーを押してプログラムを実行します。

   それでもプログラムが動作しない場合や、意図した動作と異なる場合は、トピックの更新バージョンが公開されていないかどうかをチェックします。詳細については、後述の「ヘルプ トピックにアクセスできない」を参照してください。

コマンドまたはオプションが見つからない

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ヘルプ トピックにアクセスできない

Visual Basic Express Edition に含まれているヘルプ ファイルは、Visual Studio 2005 Express Edition 用の MSDN ライブラリのサブセットであり、これはさらに、完全版の MSDN オンライン ライブラリのサブセットです。インストール時に、Visual Studio 2005 Express Edition 用の MSDN ライブラリをインストールするというオプションがありました。そのオプションを選択しなかった場合は、ここでインストールすることを検討してください。

リンクが機能しない

Visual Studio 2005 Express Edition 用の MSDN ライブラリに含まれていないトピックへのリンクでは、[情報が見つかりません] ページが表示されますが、MSDN オンライン ライブラリを検索すると、リンクの参照先のトピックにアクセスできます。

リンクが機能しない場合にトピックにアクセスするには

1. [ヘルプ] メニューの [検索] をクリックします。

   [検索] ページが表示されます。

2. リンクに表示されていたテキストを検索ボックスに入力し、[検索] をクリックします。

   検索が実行され、結果が表示されます。

3. [検索] ページの右側の [MSDN オンライン] タブをクリックします。

メモ :
[MSDN オンライン] が表示されていない場合は、ヘルプ システムがオンライン アクセス用に設定されていない可能性があります。詳細については、後述の「方法 : オンライン アクセスを有効にする」を参照してください。

4. 元のリンクのテキストに最も近いトピック タイトルをクリックします。

トピックがダウンロードされ、表示されます。

F1 キーを押しても機能しない

IDE の任意の場所で F1 キーを押すと、状況依存のヘルプが表示されます。Visual Studio 2005 Express Edition 用の MSDN ライブラリには、一部含まれていないトピックがありますが、インターネットに接続している場合には、F1 キーを使用して、完全版のライブラリのヘルプ トピックにアクセスできます。インターネットにアクセスできない場合や、オンライン ヘルプを有効にしないよう選択した場合には、一部のヘルプ トピックを F1 キーで利用できません。

ヘルプ オプションは、次の手順に従って、いつでも変更できます。

方法 : オンライン アクセスを有効にする

Visual Basic Express Edition からヘルプに初めてアクセスすると、オンライン ヘルプの設定を選択するよう求められます。選択できるのは、オンライン バージョンのトピックを先に検索する、ローカルにインストールされたヘルプを先に検索して見つからない場合はオンラインを検索する、ローカルのヘルプのみを使用する、のいずれかです。

ヒント
インターネットにアクセスできる場合には、オンライン ヘルプを先にチェックするようにヘルプを設定します。新しいバージョンのトピックが用意されている場合に、常に最新の情報を参照できるからです。

ヘルプ オプションは、次の手順に従って、いつでも変更できます。

メモ :
オンライン ヘルプにアクセスするには、アクティブなインターネット接続が必要です。また、プロキシ サーバーやファイアウォールを使用している場合には、MSDN オンライン ライブラリへのアクセスが可能となるように設定する必要があります。

オンライン アクセスを有効にするには

1. Visual Basic をまだ開いていない場合は、Windows の [スタート] メニューから開きます。
2. [ツール] メニューの [オプション] をクリックします。
3. [オプション] ダイアログ ボックスで、[すべての設定を表示] チェック ボックスをオンにします。
4. [環境] の下の [ヘルプ] を展開し、[オンライン] をクリックします。
5. [ヘルプ コンテンツを読み込むとき] ボックスで、[最初にオンライン、次にローカルで実行] または [最初にローカル、次にオンラインで実行] をクリックします。

   これでオンライン ヘルプにアクセスできます。

データベースに変更内容が反映されない

ローカル データベース ファイルは、プロジェクトのファイルとして含めることができます。初めてアプリケーションをローカル データベース ファイルに接続するときに、データベースのコピーをプロジェクトに作成するか、現在の場所にある既存のデータベース ファイルに接続するかを選択できます。

データベースのコピーを作成するように選択した場合、ファイル設定によっては、プロジェクトを実行するたびにデータベースの新しいコピーが作成され、変更を格納したバージョンが上書きされている可能性があります。詳細については、
「方法 : プロジェクトでローカル データ ファイルを管理する」を参照してください。

参照

処理手順

Visual Studio におけるデータ アクセスのトラブルシューティング
データ型のトラブルシューティング

概念

ClickOnce の配置のトラブルシューティング
開発環境に関するエラーのトラブルシューティング情報

初めての Visual Basic プログラムの作成

Visual Basic を使用したプログラミングを習得する最良の方法は、実際にプログラムを作成することです。次に示す演習では、Web ページを表示するためのプログラムを作成するプロセスを詳しく説明します。

すべてのことをすぐに理解できなくても、気にする必要はありません。ここで紹介する概念については、「Visual Basic ガイド ツアー」の他のセクションで詳しく説明します。

このセクションの内容

手順 1: Visual Basic でのプロジェクトの作成

手順 2: ユーザー インターフェイスの作成

手順 3: 外観と動作のカスタマイズ

手順 4: Visual Basic コードの追加

手順 5: プログラムの実行とテスト

関連するセクション

詳細情報 : プロパティ、メソッド、イベントについて

Visual Basic Express のプロジェクトの種類

Visual Basic プログラミング言語の概要

プログラムの外観の作成 : Windows フォームの概要

Visual Basic 2005 Express Edition (http://www.microsoft.com/japan/msdn/vstudio/2005/express/vbasic)

手順 1: Visual Basic でのプロジェクトの作成

Visual Basic プログラムを作成する最初の手順は、Visual Studio を開いてプロジェクトを作成することです。これは、すべての Visual Basic プログラムを作成するときに実行します。

プログラムのプロジェクトを作成するには

1. Windows の [スタート] メニューの Microsoft Visual Basic 2005 Express Edition をクリックします。

   [Visual Basic Express の開始] 画面が表示されます。これは Visual Basic 2005 Express Edition のインターフェイスであり、統合開発環境または IDE とも呼ばれます。

2. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。

   [新しいプロジェクト] ダイアログ ボックスが表示されます。

3. [Windows アプリケーション] をクリックし、[OK] をクリックします。

   IDE に新しいフォームが表示され、プロジェクトに必要なファイルがソリューション エクスプローラのウィンドウに追加されます。これが初めて作成した Windows アプリケーション プロジェクトの場合は、"WindowsApplication1" という名前が付けられます。

詳細情報

Web 参照プログラムのプロジェクトを作成しました。Visual Basic のプロジェクトには、プログラムが整理された状態で格納されます。

新しいプロジェクトを初めて作成する場合、プロジェクトはメモリ内にのみ存在します。Visual Basic IDE を閉じるときに、プロジェクトを保存するか破棄するかを確認するためのプロンプトが表示されます。プロジェクトを保存する場合、わかりやすい名前を付けることができます。

[新しいプロジェクト] ダイアログ ボックスを開いたときに、選択可能な複数のタイプのプロジェクトが表示されていました。Web 参照プログラムは、標準的な Windows ベースのアプリケーションです。つまり、プログラムを [スタート] メニューから実行できます。

プロジェクトを作成したときに、IDE にフォーム (フォーム デザイナとも呼ばれる) が表示されました。このフォームは、プログラムの実行時に表示されるウィンドウを表します。多くのプログラムが 1 つ以上のウィンドウを表示するため、プロジェクトには複数のフォームを含めることができます。

次の手順

次のレッスンでは、フォームにコントロールを追加して Web 参照アプリケーションを作成する方法について説明します。

次のレッスン : 「手順 2: ユーザー インターフェイスの作成」

参照

概念

Visual Basic Express のプロジェクトの種類

その他の技術情報

初めての Visual Basic プログラムの作成

手順 2: ユーザー インターフェイスの作成

Web ブラウザの作成を開始します。ツールボックスからフォームにコントロールを追加し、Microsoft Visual Basic 2005 Express Edition を使用してユーザー インターフェイス (ユーザーが対話する表示部分) を作成します。

ツールボックスは Visual Studio の左側にあり、[データ]、[コンポーネント]、[Windows フォーム] などの複数のタブで構成されます。各タブ内には、アプリケーションに追加できるコントロールまたはコンポーネントを表す一連のエントリがあります。たとえば、[すべての Windows フォーム] タブには、TextBox、Button、および CheckBox という名前の、コントロールを表すエントリがあり、フォームにドラッグしてアプリケーションに追加できます。

アプリケーションにコントロールを追加するには

1. [ツールボックス] パネルをクリックします。

   ツールボックスが表示されます。

ヒント
ウィンドウを開いたままにしている場合は、ツールボックスを使用する方が簡単です。これは、押しピンのように表示されている [自動的に隠す] アイコンをクリックすることで実行できます。

2. ツールボックスの [すべての Windows フォーム] タブをクリックして、Panel コントロールを選択し、パネルをフォームの左上隅にドラッグします。

ヒント
適切なコントロールが見つからない場合は、ツールボックスを右クリックして、[アイテムをアルファベット順に並べ替え] をクリックします。

3. 同じタブから Button コントロールをドラッグして、Panel の上部に配置します。

ヒント
ドラッグ アンド ドロップ操作でコントロールの位置を変更することもできます。コントロールの端または隅をクリックしてドラッグすることにより、コントロールのサイズ変更もできます。

4. 同じタブから TextBox コントロールをドラッグして、Panel の上部に配置します。
5. 最後に、[Windows フォーム] タブで WebBrowser コントロールを選択して、Panel の下に配置します。

ヒント
ツールボックス ウィンドウを開いたままにしている場合は、ウィンドウを閉じて作業領域を広げることもできます。これは、[自動的に隠す] アイコンを再度クリックすることで実行できます。

詳細情報

フォームに 4 つのコントロールを追加しました。コントロールには、コントロールの外観および実行できるタスクを定義するコードが含まれています。

たとえば、Button コントロールの例で説明します。ほとんどすべてのプログラムに [OK] ボタンまたは [終了] ボタンがあります。独自のコードを記述して、画面にボタンを描画し、ボタンが押されたときに外観を変更したり、ボタンがクリックされたときにタスクを実行したりすることもできますが、すべてのプログラムについて同じことを繰り返すのは面倒な作業になります。Button コントロールには、これらの作業を実行するために必要なコードが既に含まれているため、不要な作業を行わなくても済みます。

このように、ツールボックスにはさまざまなコントロールがあり、各コントロールに固有の用途があります。Panel コントロールは、追加したコントロールなど、その他のコントロールを保持するために使用します。Button コントロールは通常、プログラムの終了など、ユーザーがボタンをクリックしたときにタスクを実行するために使用します。TextBox コントロールは、キーボードを使用して画面にテキストを入力するために使用します。WebBrowser コントロールは、Internet Explorer に類似した組み込み Web 参照機能を提供します。そのためのコードすべてを記述する手間が省けます。

以降のレッスンでは、このようなコントロールの外観をカスタマイズする方法、およびコントロールの動作を定義するコードを記述する方法について説明します。ツールボックスのコントロールを使用する以外に、ユーザー コントロールと呼ばれる独自のコントロールを作成することもできます。これ

についても次のレッスンで説明します。

次の手順

アプリケーションに必要なコントロールをすべて追加しました。このアプリケーションは、荒削りで未完成のように見えるかもしれません。実際、そのとおりです。次のレッスンでは、[プロパティ] ウィンドウを使用して、アプリケーションの外観と操作性を制御するプロパティを設定します。

次のレッスン : 「手順 3: 外観と動作のカスタマイズ」

参照

処理手順

手順 1: Visual Basic でのプロジェクトの作成

その他の技術情報

初めての Visual Basic プログラムの作成

Windows フォームで使用するコントロール

手順 3: 外観と動作のカスタマイズ

前のレッスンでは、アプリケーションにコントロールを追加して、ユーザー インターフェイスを作成しました。ただし、この時点では、概観も機能も完成したアプリケーションらしくありません。このレッスンでは、[プロパティ] ウィンドウを使用して、コントロールの外観を制御するプロパティを設定します。

コントロールのプロパティを設定するには

1. フォーム デザイナの Panel コントロールをクリックします。

   IDE の右下隅にある [プロパティ] ウィンドウに、Panel1 という名前の **Panel** コントロールのプロパティがすべて表示されます。

2. [プロパティ] ウィンドウで、Dock プロパティを選択し、右側の矢印をクリックします。複数のボックスがある小さいプロパティ選択ウィンドウが表示されます。

ヒント
Dock プロパティは、[配置] カテゴリにあります。[プロパティ] ウィンドウの [az] ボタンをクリックすると、プロパティをアルファベット順に並べ替えることができます。

3. **Dock** プロパティを **Top** に設定するには、プロパティ選択ウィンドウの上部のボックスをクリックします。**Panel** コントロールは、フォームの一番上まで埋まるように拡大されます。

4. フォーム デザイナの WebBrowser コントロールをクリックします。[プロパティ] ウィンドウで、**Dock** プロパティを **Fill** に設定します。それには、**Dock** プロパティを選択し、右側の矢印をクリックして、プロパティ選択ウィンドウの中央のボックスを選択します。

5. フォーム デザイナの Button コントロールをクリックします。

6. [プロパティ] ウィンドウで、**Button** コントロールの Text プロパティを選択します。右側の列で、「`Button1`」を削除し、「`Go!`」に置き換えます。

7. コントロールのサイズを変更するか、再配置してから、フォームを任意のサイズに変更します。

メモ :
TextBox コントロールおよび **Button** コントロールは、**Panel** 上にあることが必要です。外に出すと、アプリケーションを実行したときにコントロールが表示されません。

詳細情報

このレッスンでは、アプリケーションのコントロールの外観を変更する複数のプロパティを設定しました。Visual Basic のプロパティは、オブジェクト (この場合はコントロール) の属性を表します。たとえば、Button コントロールの属性の 1 つに、コントロールに表示されるテキストがあります。この例では、"Go!" と表示されるように Text プロパティを設定しました。プロパティの詳細については、
「詳細情報 : プロパティ、メソッド、イベントについて」を参照してください。

プロパティは、テキスト以外にもさまざまな種類の値を取ることができます。たとえば、**Dock** プロパティでは、使用可能なオプションがプロパティ選択ウィンドウで表示されました。その他のプロパティ値には、数値、一覧から選択したオプション、または True/False オプションがあります。

コントロールのサイズを変更、または再配置すると、プロパティも設定されています。Size プロパティおよび Location プロパティは、コントロールのサイズとフォーム上の位置を決定します。これを行うには、[プロパティ] ウィンドウの **Size** プロパティをクリックし、マウスを使用してコントロールのサイズを変更します。マウス ボタンを離すと、[プロパティ] ウィンドウに新しい **Size** 値が表示されます。

[プロパティ] ウィンドウでプロパティを設定する以外に、ほとんどのプロパティはコードを記述することでも設定できます。次のレッスンでは、コードを記述してプロパティを設定する方法の詳細について説明します。

次の手順

これで、ユーザー インターフェイスが完成しました。後は、プログラムを機能させるために必要な命令 (コードとも呼ばれます) を追加するだけです。次のレッスンで、プログラムにコード行を追加します。

次のレッスン : 「手順 4: Visual Basic コードの追加」

参照

処理手順

手順 1: Visual Basic でのプロジェクトの作成

手順 2: ユーザー インターフェイスの作成

概念

詳細情報 : プロパティ、メソッド、イベントについて

その他の技術情報

初めての Visual Basic プログラムの作成

詳細情報：プロパティ、メソッド、イベントについて

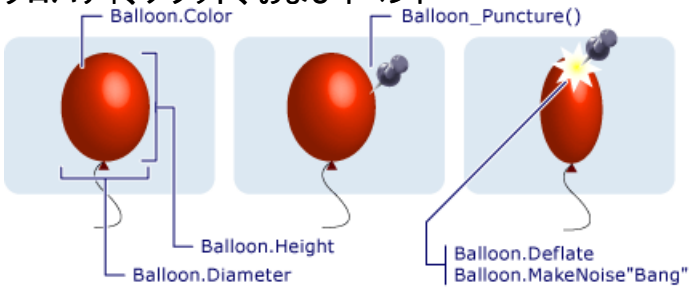
フォームやコントロールを含む、Visual Basic 言語内のオブジェクトはすべて、独自のプロパティ、メソッド、およびイベントを持っています。プロパティは、オブジェクトの属性、メソッドはオブジェクトのアクション、イベントはオブジェクトの応答と考えることができます。

ヘリウム気球などの通常のオブジェクトにも、プロパティ、メソッド、およびイベントがあります。気球のプロパティには、高さ、直径、色などの可視属性があります。その他、気球の状態 (ふくらませてあるか、しぼんでいるか)、または寿命などの目に見えない属性を表すプロパティもあります。すべての気球にこれらのプロパティがありますが、プロパティの値は気球ごとに異なる可能性があります。

気球には、実行可能な既知のメソッドまたはアクションもあります。気球には、膨張メソッド (ヘリウムを入れる)、収縮メソッド (ヘリウムを抜く)、および上昇メソッド (手を放す) があります。すべての気球は、これらのメソッドを実行できます。

気球には、特定の外部イベントに対する応答もあります。たとえば気球は、穴を開けられるイベントに対しては収縮し、解放されるイベントに対しては上昇することにより応答します。

プロパティ、メソッド、およびイベント



気球はプロパティ (Color、Height、Diameter) を持ち、イベント (Puncture) に応答し、メソッド (Deflate、MakeNoise) を実行できます。

プロパティ

気球のプログラムを作成する場合、Visual Basic コードは次のようになり、気球のプロパティが設定されます。

```
Balloon.Color = Red

Balloon.Diameter = 10

Balloon.Inflated = True
```

コードの順番は、オブジェクト (Balloon) の後にプロパティ (Color) が続き、プロパティの後に値の代入 (= Red) が続きます。別の値を代入して、気球の色を変更することもできます。

メソッド

気球のメソッドは、次のように呼び出されます。

```
Balloon.Inflate

Balloon.Deflate

Balloon.Rise(5)
```

順序は、プロパティの場合と同様に、オブジェクト (名詞) の後にメソッド (動詞) が続きます。3 番目のメソッドには、引数と呼ばれる項目があり、気球が上昇する高さが指定されます。実行されるアクションを記述するために、1 つ以上の引数を持つメソッドもあります。

イベント

気球は、次のようにイベントに応答します。

```
Sub Balloon_Puncture()
    Balloon.MakeNoise("Bang")
    Balloon.Deflate
    Balloon.Inflated = False
End Sub
```

この場合、コードは Puncture イベントが発生したときの気球の動作を記述します。引数 "Bang" (発生する音の種類) を使用して MakeNoise メソッドを呼び出した後、Deflate メソッドを呼び出します。気球はふくらまなくなったため、Inflated プロパティは **False** に設定されます。

実際に気球のプログラムを作成することはできませんが、Visual Basic フォームまたはコントロールをプログラムできます。目的の外観と動作を実現するために、変更するプロパティ、呼び出すメソッド、または応答するイベントを決定します。

次に行う作業

次のレッスンで、プログラムにコード行を追加します。

次のレッスン : 「手順 4: Visual Basic コードの追加」

参照

その他の技術情報

Visual Basic のプログラミング ガイド

手順 4: Visual Basic コードの追加

前のレッスンでは、[プロパティ] ウィンドウを使用して、フォーム上のコントロールのプロパティを設定しました。このレッスンでは、プログラムの機能を制御するコードを追加します。

プログラムにコードと機能を追加するには

1. フォーム デザイナで Button コントロールをダブルクリックします。

   コード エディタと呼ばれる新しいウィンドウが表示されます。このウィンドウを使用して、プログラムのすべてのコードを追加します。

2. コード エディタで次のように入力します。

   VB

   ```
   WebBrowser1.Navigate(Textbox1.Text)
   ```

   このコードは、ユーザーがボタンをクリックしたときに実行されます。

   ヒント

   コード エディタが開くと、カーソルが Button のプロシージャに自動的に移動して、そのまま入力を始めることができます。

詳細情報

コード エディタが開くと、次のようなコードが既に表示されています。

```
Private Sub Button1_Click(ByVal sender As System.Object...

|

End Sub
```

このコードはイベント ハンドラです。サブ プロシージャと呼ばれることもあります。ボタンをクリックするたびに、このプロシージャ内 (**Sub** から **End Sub** まで) のコードが実行されます。カーソルがイベント プロシージャ内にあるため、入力するだけで済みました。

入力したコード (`WebBrowser1.Navigate(TextBox1.Text)`) は、**TextBox1.Text** (**TextBox** コントロールの **Text** プロパティに含まれる値) を引数に指定して、**WebBrowser** コントロール (名前 WebBrowser1) の **Navigate** メソッドを使用するようにプログラムに通知します。プロパティ、メソッド、およびイベントの詳細については、「詳細情報 : プロパティ、メソッド、イベントについて」を参照してください。

コードがわからない場合でも心配は無用です。以降のレッスンで、コードを記述する方法について詳細に説明します。

次の手順

これで、アプリケーションが完成しました。次のレッスンでは、初めて自分で作成した Visual Basic アプリケーションを実行します。

次のレッスン : 「手順 5: プログラムの実行とテスト」

参照

処理手順

手順 1: Visual Basic でのプロジェクトの作成
手順 2: ユーザー インターフェイスの作成
手順 3: 外観と動作のカスタマイズ

概念

詳細情報 : プロパティ、メソッド、イベントについて

その他の技術情報

初めての Visual Basic プログラムの作成

手順 5: プログラムの実行とテスト

これでプログラムが完成しました。次は、これを実行してテストします。複雑なプログラムのテストは、時間がかかるうえに困難なプロセスです。これについては、後のレッスンで詳しく説明します。このプログラムについては、実行するだけで十分です。

プログラムを実行するには

1. コンピュータをインターネットに接続します。
2. Visual Basic IDE で、[デバッグ] メニューの [デバッグ開始] をクリックします。

   このコマンドにより、プログラムが実行されます。

ヒント
プログラムを実行するためのショートカット キーは F5 です。

3. テキスト ボックスに「`http://www.microsoft.com/japan`」と入力し、[Go!] をクリックします。

   プログラム内の WebBrowser コントロールが、Microsoft のホーム ページに移動します。このページから関連リンクを通じて移動できます。別の Web ページを表示するには、テキスト ボックスにアドレスを入力して [Go!] をクリックします。

4. プログラムを終了するには、[デバッグ] メニューの [デバッグの停止] をクリックします。

ヒント
フォームの右上端にある [閉じる] をクリックしても、プログラムを終了できます。

詳細

このレッスンでは、作成したプログラムが機能するかどうか確かめるために、プログラムを実行しました。ほとんどの Visual Basic プログラムでは、このプロセスを何回も繰り返すことになります。通常、新しいコードを追加した後は、コードが期待した動作をするかどうか確認するために、プログラムを実行します。コードが期待した動作をしない場合は、修復する必要があります。このプロセスをデバッグと呼びます。これについては、後のレッスンで詳しく説明します。

コード行を 1 行記述するだけで、プログラムが Web ページに移動して表示するようになるのは驚きかもしれません。これが Visual Basic の魅力です。必要なコードはすべて **WebBrowser** コントロールに組み込まれているため、時間と手間を省くことができます。これらすべてを自分で行おうとする場合、何百行、何千行というコードを記述する必要があります。

トラブルシューティング

プログラムが動作しない場合や、Web ページが表示されない場合は、次に示すいくつかの項目をチェックします。

- インターネットに接続していることを確認します。Internet Explorer を開き、Microsoft のホーム ページに移動してみてください。Internet Explorer で表示できる場合は、プログラムでも表示できます。
- アドレス (`http://www.microsoft.com/japan`) を正確に入力したことを確認します。
- 「手順 2: ユーザー インターフェイスの作成」に戻って内容を確認し、正しいコントロールをフォーム上に配置していることを確認します。
- 「手順 4: Visual Basic コードの追加」に戻り、コードを正しく入力したことを確認します。

次の手順

これで処理は完了しました。初めての Visual Basic プログラムの完成です。Visual Basic を使用すると、強力なプログラムを迅速かつ容易に開発できることが実証されました。次からのレッスンでは、Visual Basic プログラミング言語のさらなる機能について説明します。

次のレッスン : 「Visual Basic プログラミング言語の概要」

参照

処理手順

手順 1: Visual Basic でのプロジェクトの作成
手順 2: ユーザー インターフェイスの作成
手順 3: 外観と動作のカスタマイズ
手順 4: Visual Basic コードの追加

その他の技術情報

初めての Visual Basic プログラムの作成

問題の確認 : デバッグを行ってエラーを検出し、修正する

Visual Basic プログラミング言語の概要

Microsoft Visual Basic 2005 Express Edition は、Microsoft Windows 向けプログラムを迅速かつ簡単に作成するための方法です。Visual Basic では、Windows プログラミングを初めて行う人でも、開発を簡素化するための完成度の高いツール セットを使用できます。

まず Visual Basic について説明します。"Visual" とは、ユーザーの目に見えるもの、つまりグラフィカル ユーザー インターフェイス (GUI) を作成するために使用する方法を指します。"Basic" とは、コンピュータ史上最も多くのプログラマが使用したプログラミング言語である BASIC (Beginners All-Purpose Symbolic Instruction Code) 言語を指します。Visual Basic では、その機能の一部を学習するだけで、役に立つプログラムを作成できます。以降では、Visual Basic プログラミングを始めるために役立つリンクを示します。各リンクには、コード例の他に、追加情報へのアクセスが含まれています。

プログラミングの概念

プログラミング言語とは、厳密に言うと何でしょうか。次のリンクでは、言語とは何か、また言語が異なる種類の情報を格納するしくみについて、その背景が示されています。

用語	定義
基本 : プログラミングの機能について	プログラミング言語が機能するしくみと共に、基本的な用語について説明します。
変数による単語、数字、および値の表現	変数が値を格納して情報を表現するしくみと共に、変数の使用方法を説明します。
単語およびテキスト : 文字列変数を使用して単語を編成する	**String** 変数を使用して単語やテキストを表現する方法を説明します。
配列 : 複数の値を表す変数	**Array** 変数を使用して同じ型の複数の値を表現する方法を説明します。
演算 : 変数および演算子を使用した式の作成	演算を実行するコードを記述する方法を説明します。
比較 : 値を比較するための式の使用	数値を比較するコードを記述する方法を説明します。

初めてのプログラム

実際のプログラミングを体験する準備はできましたか。次のリンクでは、単純なプログラムを作成する手順と、作成したプログラムにエラーがないかどうかチェックする方法を説明します。

用語	定義
コンピュータが行う処理の記述 : 初めてのプロシージャの作成	プログラムに特定の処理を実行するよう指示するコードを記述する方法を説明します。
プログラムでの処理の繰り返し : For...Next ループによるループ処理	プログラム内で処理を繰り返して、その処理が実行された回数をカウントするコードを記述する方法を説明します。
二者択一のプログラム : If...Then ステートメント	異なる条件に応じて異なる処理を実行するコードを記述する方法を説明します。
問題が発生した場合の対処 : エラー処理	プログラムのエラーを処理するコードを記述する方法を説明します。各種のエラーについても説明します。

Visual Basic の詳細情報

次のリンクは、プログラミングおよび Visual Basic 2005 についてさらに知識を深めるために役立ちます。

用語	定義
詳細情報 : プロパティ、メソッド、イベントについて	プロパティ、メソッド、およびイベントのしくみについて説明します。

詳細情報 : データ型	さまざまな変数を使用してデータを格納する方法を説明します。
詳細情報 : ある変数の型を別の型に変換する	データの型を変換する方法を説明すると共に、このプロセスに共通の落とし穴について説明します。
詳細情報 : Do...While と Do...Until を使用して条件が成立するまで処理を繰り返す	**Do...While** ステートメントおよび **Do...Until** ステートメントを使用して、特定の条件に基づいてコードを繰り返す方法を説明します。
Select Case を使用して複数の選択肢から処理を決定する	多くの選択項目がある場合に、複数の条件に基づいてコードを実行する方法を説明します。
Visual Basic ガイド ツアー	Visual Basic 2005 プログラミング言語で実行できるその他の処理について説明します。

基本：プログラミングの機能について

Visual Basic プログラミング言語の学習を開始する前に、プログラミング言語の概念と機能およびプログラミング用語などを理解しておくと便利です。基本から始めるのが重要です。

プログラミングの機能

コンピュータは、単独ではそれほど高性能ではありません。

コンピュータは基本的に、オンまたはオフになる、小さい電子スイッチが多数集まったものです。これらのスイッチのさまざまな組み合わせにより、画面に何か表示する、音を出すなどの処理をコンピュータで実行できます。それがプログラミングの最も基本的な機能、つまりコンピュータに処理を通知することです。

スイッチをどう組み合わせると、コンピュータがどのような処理を実行するかを理解するのは時間がかかるため、プログラミング言語が使用されます。

プログラミング言語とは

人間が自分の意思を表現するときには、多数の単語を組み合わせた言語を使用します。コンピュータは、1 と 0 のみで構成される単純な言語を使用します。1 は "オン"、0 は "オフ" を意味します。コンピュータにマシン語で話しかけるのは、モールス信号を使用して友人に話しかけるようなものであり、可能ではありますがその必要はありません。

プログラミング言語は、人間とコンピュータとの間の翻訳者として機能します。コンピュータの母語 (マシン語とも呼ばれます) を習得せずに、プログラミング言語を使用してコンピュータに命令できます。プログラミング言語の方が、理解と習得が簡単です。

コンパイラと呼ばれる特別なプログラムは、プログラミング言語で記述された命令を受け取り、マシン語に変換します。つまり、Visual Basic プログラマとしては、コンピュータの処理内容やその方法を理解する必要はなく、Visual Basic プログラミング言語の機能を理解するだけで済みます。

Visual Basic 言語の内部

Visual Basic は人間が普段使用する言語に多くの点で似ています。人間が話すときや書くときには、名詞や動詞などさまざまな種類の単語を使用します。種類によって、単語の使用方法が定義されています。Visual Basic も、プログラムを記述する際の単語の使用方法を定義したプログラミング要素と呼ばれるさまざまな種類の単語を使用します。

Visual Basic のプログラミング要素には、ステートメント、宣言、メソッド、演算子、およびキーワードがあります。以降のレッスンを完了すると、これらの要素の詳細と使用方法を理解できるようになります。

書き言葉および話し言葉にも、文での単語の順序を定義する規則、つまり構文があります。Visual Basic にも構文があります。最初は奇妙に思えることもありますが、実際には非常に単純です。たとえば、"車の最高速度は 55 です" と述べるには、次のように記述します。

```
Car.Speed.Maximum = 55
```

構文の詳細については、後で説明します。IntelliSense などの Visual Basic のツールは、プログラムを記述するときに正しい構文を使用するために役立ちます。

人間が書いたり話したりする言語にも構造があります。たとえば、本は章で構成され、章には段落が含まれ、段落には文が含まれます。Visual Basic で記述されたプログラムにも構造があります。モジュールは章、プロシージャは段落、コード行は文に似ています。

次の手順

このレッスンでは、プログラミング言語の概念と機能について説明しました。次のレッスンでは、Visual Basic プログラミング言語の使用方法について説明します。心配は無用です。すぐに Visual Basic を使うことができるようになります。

次のレッスン：「変数による単語、数字、および値の表現」

参照

処理手順

変数による単語、数字、および値の表現

概念

詳細情報：プロパティ、メソッド、イベントについて

Visual Basic プログラミング言語の概要

その他の技術情報

初めての Visual Basic プログラムの作成

Visual Basic のプログラミング ガイド

変数による単語、数字、および値の表現

変数は、コンピュータ プログラミングにおける重要な概念です。変数は、値を格納できる文字または名前です。コンピュータ プログラムを作成する場合、変数を使用して、数値 (建物の高さなど) や単語 (人の名前など) を格納できます。簡単に言えば、変数を使用すると、プログラムが必要とするすべての種類の情報を表すことができます。

"情報を使用すれば済む場合に、変数を使用するのはなぜか" と思うかもしれません。名前が示すように、変数は、表す値をプログラムの実行中に変更できます。たとえば、机の上の瓶に入っているゼリー ビーンの数を追跡するプログラムを記述するとします。キャンディーは食べられることになっているため、瓶の中のゼリー ビーンの数は時間と共に変わります。甘いものが欲しくなるたびにプログラムを書き直さなくても、時間と共に変更できる変数を使用して、ゼリー ビーンの数を表すことができます。

変数への情報の格納

変数を使用するには、次の 3 つの手順を実行します。

1. **変数を宣言します。** 使用する変数の名前と種類をプログラムに通知します。
2. **変数を割り当てます。** 変数が保持する値を指定します。
3. **変数を使用します。** 変数が保持している値を取得し、プログラムで使用します。

変数の宣言

変数を宣言するときに、変数に割り当てる名前とデータ型を決定する必要があります。

次のように、**Dim** キーワードおよび **As** キーワードを使用して変数を宣言します。

VB

```
Dim aNumber As Integer
```

このコード行は、`aNumber` という名前の変数を、整数 (**Integer** データ型) を格納する変数として使用することをプログラムに通知します。

`aNumber` は **Integer** であるため、整数のみを格納できます。たとえば、42.5 を格納する場合は、**Double** データ型を使用します。単語を格納する場合は、**String** というデータ型を使用します。ここで説明するもう 1 つのデータ型は **Boolean** です。これは、`True` 値、または `False` 値を格納できます。

変数を宣言する方法の例を次に示します。

VB

```
Dim aDouble As Double
Dim aName As String
Dim YesOrNo As Boolean
```

その他の変数の型の詳細については、「詳細情報 : データ型」を参照してください。

変数の割り当て

変数に値を割り当てるには、次の例に示すように = 記号を使用します。代入演算子と呼ばれることもあります。

VB

```
aNumber = 42
```

このコード行は値 `42` を取得して、前に宣言した `aNumber` という名前の変数に格納します。

既定値を使用した変数の宣言と割り当て

上記のように、あるコード行で変数を宣言してから、別の行で値を割り当てることもできます。値を割り当てる前に変数を使用しようとすると、エラーが発生する可能性があります。

そのため、変数の宣言と割り当ては 1 行で行うことをお勧めします。変数が保持する値の種類がわからない場合でも、既定値を割り当てることができます。前に示したのと同じ変数を宣言して割り当てるコードは、次のようになります。

VB

```
Dim aDouble As Double = 0
Dim aName As String = "default string"
Dim YesOrNo As Boolean = True
```

変数の宣言と既定値の割り当てを 1 行で行えば、エラーの発生を予防できます。その場合でも、後で代入を使用し、変数に異なる値を設定できます。

やってみよう

この練習では、4 つの変数を作成して値を割り当て、メッセージ ボックスと呼ばれるウィンドウに各値を表示する短いプログラムを記述します。まず、コードが格納されるプロジェクトを作成しましょう。

プロジェクトを作成するには

1. Visual Basic をまだ開いていない場合は、Windows の [スタート] メニューから開きます。
2. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
3. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
4. [プロジェクト名] ボックスに「`Variables`」と入力し、[OK] をクリックします。

   Visual Basic によりプログラムのファイルが作成され、フォーム デザイナが開きます。

次に、変数を作成します。

変数を作成して値を表示するには

1. フォームをダブルクリックします。

   コード エディタに、`Form1_Load` というコード セクションが表示されます。このコード セクションは、プロシージャと呼ばれ、フォームがメモリに最初に読み込まれたときに実行される命令が含まれています。

2. `Form1_Load` プロシージャに次のコードを追加します。

   VB

   ```
   Dim anInteger As Integer = 42
   Dim aSingle As Single = 39.345677653
   Dim aString As String = "I like candy"
   Dim aBoolean As Boolean = True
   ```

   このコードは、**Integer**、**Single**、**String**、および **Boolean**— の 4 つの変数を宣言して、既定値を割り当てます。

   ヒント

   コードの入力時に、「`As`」と入力すると、単語の一覧がカーソルの下に表示されます。この機能は IntelliSense と呼ばれています。この機能により、単語の最初の数文字を入力するだけで、その単語を一覧から選択できます。単語を選択した後、Tab キーを押して単語を完成させます。

   メモ :

   プログラムで実際のテキストを表す場合は、常に引用符 ("") で囲む必要があります。これは、テキストを変数名ではなく実際のテキストとして解釈するようにプログラムに通知します。**Boolean** 変数に値 **True** または **False** を割り当てる場合は、単語を引用符で囲む必要はありません。これは、**True** および **False** は、独自の特別な意味を持つ Visual Basic キーワードであるためです。

3. 前の手順で記述したコードの後に、次のように入力します。

   VB

   ```
   MsgBox(anInteger)
   MsgBox(aSingle)
   ```

```
MsgBox(aString)
MsgBox(aBoolean)
```

VB

```
End
```

コードの最初の 4 行は、**MsgBox** 関数を使用して、前の手順で割り当てた各値を新しいウィンドウに表示するようにプログラムに通知します。最後の行は、このプロシージャを実行した後に終了するようにプログラムに通知します。これには、**End** ステートメントを使用します。

4. F5 キーを押してプログラムを実行します。

   ウィンドウが表示されるたびに [OK] をクリックします。各変数の値が順に表示され、プログラムが終了します。プログラムが完了した後、戻ってコードに割り当てられた値を変更し、アプリケーションを再実行することもできます。その場合は、新しい値が表示されます。

次の手順

このレッスンでは、変数の基本について説明しました。次のレッスンでは、**String** 変数の詳細について説明します。

次のレッスン : 「単語およびテキスト : 文字列変数を使用して単語を編成する」

参照

処理手順

単語およびテキスト : 文字列変数を使用して単語を編成する

詳細情報 : ある変数の型を別の型に変換する

概念

基本 : プログラミングの機能について

詳細情報 : データ型

詳細情報：データ型

Visual Basic におけるデータ型は、変数に格納できる値またはデータの種類、さらにはそのデータの格納方法も決定します。さまざまなデータ型があるのはなぜでしょうか。次のように考えてみてください。3 つの変数があるとして、そのうちの 2 つに数値を格納し、もう 1 つに名前を格納する場合、前の 2 つを使用すると演算を実行できますが、名前には演算を実行できません。変数にデータ型を割り当てることで、その変数を使用できる場合と使用できない場合を、より簡単に決定できます。

メモ：

データ型は、定数、プロパティ、および関数など、他のプログラミング要素でも使われます。データ型の他の使用方法については、次のレッスンで説明します。

数値のデータ型

ほとんどのコンピュータ プログラムでは、何らかの形式の数値を扱います。数値にはさまざまな表現方法がありますが、Visual Basic では、数値をより効率的に扱うためのいくつかの数値データ型が用意されています。

最も頻繁に使用する数値データ型は **Integer** です。これは整数 (小数部のない数値) を表すために使用します。整数を表すデータ型を選択する場合、およそ 20 億を超える数値を変数に格納するときは **Long** データ型を使用する必要があります。それ以外の場合は **Integer** を使用する方が効率的です。

すべての数値が整数というわけではありません。たとえば、9 を 2 で割ると 4.5 になるように、2 つの整数を除算すると、整数に小数部が付いた数値になる場合がよくあります。**Double** データ型は、小数部を持つ数値を表すために使用します。

メモ：

Decimal、**Short**、**SByte**、**UInteger** などの数値データ型もあります。これらのデータ型は、通常、メモリの使用や処理速度が重視される非常に大規模なプログラムで使われます。しかし、現時点では、基本的な数値データ型を知っていれば十分です。高度なデータ型についてさらに学習する場合は、「数値データ型」を参照してください。

テキストのデータ型

ほとんどのプログラムでは、ユーザーに情報を表示したり、ユーザーが入力したテキストを取得したりするために、テキストも扱います。テキストは、通常 **String** データ型に格納されます。このデータ型は、一連の文字、数値、空白、およびその他の文字を格納できます。**String** には、文章や段落から、単一の文字、何もない状態 (null 文字列) まで、長さに制限がありません。

常に 1 つの文字だけを表す変数には、**Char** データ型も使用できます。単一の変数に 1 つの文字を格納するだけでよい場合は、**String** ではなく **Char** データ型を使用できます。

その他のデータ型

プログラムでは、テキストや数値に加えて、他の種類の情報を格納する必要が生じる場合があります。つまり、true または false の値、日付、またはプログラムにとって特別な意味を持つデータなどです。

Visual Basic では、true または false、はいまたはいいえ、オンまたはオフで表現できる値のために、**Boolean** データ型が用意されています。**Boolean** 変数には、**True** および **False** という 2 つの選択肢のうち 1 つを格納できます。

日付や時間は、数値で表すこともできますが、**Date** データ型を使用すると、日付や時間の計算がより簡単になります。たとえば、誕生日までの日数や、昼食までの分数を計算する場合などに使用できます。

単一の変数に複数の型のデータを格納する必要がある場合は、複合データ型を使用できます。複合データ型には、配列、構造体、およびクラスがあります。これらのデータ型については後のレッスンで説明します。

最後に、格納する必要のあるデータの型が、そのときによって異なる場合があります。**Object** データ型を使用すると、まず変数を宣言して、後からデータ型を定義できます。**Object** データ型についても後のレッスンで説明します。

次の手順

次のレッスン (「単語およびテキスト : 文字列変数を使用して単語を編成する」) では、文字列変数を使用して文章を作成する方法を詳しく説明します。

次のレッスン : 「単語およびテキスト : 文字列変数を使用して単語を編成する」

参照

処理手順

単語およびテキスト : 文字列変数を使用して単語を編成する

変数による単語、数字、および値の表現

概念

Visual Basic におけるデータ型

単語およびテキスト : 文字列変数を使用して単語を編成する

このレッスンでは、**String** データ型を使用して、単語やテキストを表現する方法を説明します。

前のレッスンでは、プログラムで変数を使用してデータを格納する方法を説明し、各変数が、格納するデータに適した型である必要があることを示しました。このレッスンでは、テキストを格納するために使用する **String** データ型について詳しく説明します。

文字列とは

文字列とは、文字、数値、特殊文字、空白文字などで構成される一連のテキスト文字を指します。文字列は、人間が判読できる句または文章 ("The quick brown fox jumps over the lazy dog" など) である場合と、一見理解不能なテキスト文字の組み合わせ ("@#fTWRE^3 35Gert" など) である場合があります。

String 変数の作成方法は、他の変数とまったく同じです。つまり、次のように、まず変数を宣言してから値を代入します。

VB

```
Dim aString As String = "This is a string"
```

実際のテキスト (リテラル文字列とも呼ばれる) を **String** 変数に代入するときは、テキストを引用符 ("") で囲む必要があります。また、ある **String** 変数を別の **String** 変数に代入するときは、次の例に示すように = 文字を使用します。

VB

```
    Dim aString As String = "This is a string"
...
    Dim bString As String = ""
    bString = aString
```

このコードは、`bString` の値を `aString` と同じ値 (`This is a string`) に設定します。

アンパサンド (`&`) 文字を使用すると、複数の文字列を連続的に組み合わせて新しい文字列を作成できます。次に例を示します。

VB

```
Dim aString As String = "Across the Wide"
Dim bString As String = "Missouri"
Dim cString As String = ""
cString = aString & bString
```

この例では、3 つの **String** 変数を宣言して、前の 2 つにそれぞれ "Across the Wide" と "Missouri" を代入し、これら 2 つを組み合わせた値を 3 つ目の変数に代入しています。この場合、`cString` の値はどうなるでしょうか。意外に思われるかもしれませんが、この値は `Across the WideMissouri` となります。`aString` の末尾や `bString` の先頭に空白文字がないためです。2 つの文字列は、単純に結合されるだけです。2 つの文字列の間に空白などを追加する場合には、`" "` などのリテラル文字列を使用する必要があります。次に例を示します。

```
Dim aString As String = "Across the Wide"
Dim bString As String = "Missouri"
Dim cString As String = ""
cString = aString & " " & bString
```

このように記述すると、`cString` に格納されるテキストは `Across the Wide Missouri` となります。

やってみよう

文字列を結合するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスで、次の手順を実行します。
    a. [テンプレート] ペインの [Windows アプリケーション] をクリックします。

b. [ファイル名] ボックスに「**Concatenation**」と入力します。

c. [OK] をクリックします。

新しい Windows フォーム プロジェクトが開きます。

3. フォームをダブルクリックしてコード エディタを開きます。

4. **Form1.Load** イベント プロシージャで、次のように 4 つの変数を宣言して文字列値を代入します。

VB

```
Dim aString As String = "Concatenating"
Dim bString As String = "Without"
Dim cString As String = "With"
Dim dString As String = "Spaces"
```

5. 次のコードを追加して、文字列を連結し、その結果を表示します。

VB

```
    ' Displays "ConcatenatingWithoutSpaces".
    MsgBox(aString & bString & dString)

...    ' Displays "Concatenating With Spaces".
    MsgBox(aString & " " & cString & " " & dString)
```

メッセージ ボックスには、前の手順で代入した文字列変数を結合したテキストが表示されます。最初のボックスでは、文字列が空白文字なしで結合されています。2 番目のボックスでは、各文字列の間に空白文字が明示的に挿入されています。

次の手順

このレッスンでは、文字列を宣言して代入する方法と、連結演算子の & を使用して文字列を結合する方法を説明しました。次のレッスン (「配列 : 複数の値を表す変数」) では、類似したアイテムのグループを格納するための変数を作成する方法を説明します。

次のレッスン :「配列 : 複数の値を表す変数」

参照

処理手順

配列 : 複数の値を表す変数

変数による単語、数字、および値の表現

概念

Visual Basic プログラミング言語の概要

配列：複数の値を表す変数

このレッスンでは、配列を使用して値のグループを格納する方法について説明します。

前のレッスンで説明したように、変数は、プログラムで使用するさまざまな種類のデータを格納するために使用されます。配列と呼ばれる別の種類の変数は、同じ型の複数の値を格納するのに便利です。

たとえば、野球チームのプログラムを記述していて、守備側のすべての選手の名前を格納するとします。選手ごとに 9 つの別個の文字列変数を作成することも、次のコードのような配列変数を宣言することもできます。

VB

```
Dim players() As String
```

配列変数を宣言するには、変数名の後にかっこを付けます。格納する必要がある値の数がわかっている場合は、次のように、宣言内に配列のサイズを指定することもできます。

VB

```
Dim players(8) As String
```

野球チームの選手は `9` 人なのに、配列のサイズが `8` になっているのは奇妙に見えるかもしれません。これは、配列の値 (要素) が要素 0 で始まり、宣言で指定された数で終了するためです。つまり、配列には要素 0 から 8 が含まれ、合計で 9 つの要素になります。

配列への値の割り当て

他の値の型と同様に、配列に値を割り当てる必要があります。配列に値を割り当てるには、次に示すように割り当ての一部として要素番号を参照します。

VB

```
players(0) = "John"
players(3) = "Bart"
```

上記のコードでは、値 `John` が配列の最初の要素 (要素 0) に割り当てられ、値 `Bart` が 4 番目の要素 (要素 3) に割り当てられています。配列の要素は、順に割り当てる必要はありません。割り当てられていない要素には、既定値 (この場合は空の文字列) が割り当てられます。

他の値の型と同様に、配列も、次のように 1 行で値の宣言と割り当てを行うことができます。

VB

```
Dim players() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

この場合、中かっこは値のリストを示します。値は、表示されている順序で要素に割り当てられます。配列のサイズが指定されていないことに注意してください。配列のサイズは、指定した項目の数によって決定されます。

配列からの値の取得

番号を使用して配列内の項目の位置を指定するように、要素番号を使用して取得する値を指定します。

VB

```
Dim AtBat As String
AtBat = players(3)
```

上記のコードは、配列の 4 番目の要素を取得して、文字列変数 `AtBat` に割り当てます。

やってみよう

値を配列に格納するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`MyFirstArray`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、フォームに Textbox コントロールをドラッグします。
5. ツールボックスから、フォームに Button コントロールをドラッグします。
6. Buttonをダブルクリックしてコード エディタを開きます。
7. **Button1_Click** イベント プロシージャに次のコードを追加します。

   VB

```
Dim players() As String = {"Dan", "Fred", "Bart", "Carlos", _
  "Ty", "Juan", "Jay", "Sam", "Pedro"}
Dim i As Integer = CInt(Textbox1.Text)
MsgBox(players(i) & " is on first base.")
```

   上のコードでは、**CInt** 関数を使用して、**String** 値 (`TextBox1.Text`) を **Integer** (`i`) に変換しています。変換の詳細については、「詳細情報 : ある変数の型を別の型に変換する」で説明します。
8. F5 キーを押してプログラムを実行します。
9. テキスト ボックスに 0 から 8 までの数値を入力して、ボタンをクリックします。その要素に対応する名前がメッセージ ボックスに表示されます。

次の手順

このレッスンでは、配列を使用して、類似した値のグループの格納および取得を行う方法を説明しました。

次のレッスンでは、算術演算子を使用して式を作成する方法について説明します。

次のレッスン : 「演算 : 変数および演算子を使用した式の作成」

参照

処理手順

演算 : 変数および演算子を使用した式の作成

単語およびテキスト : 文字列変数を使用して単語を編成する

概念

Visual Basic の配列の概要

演算：変数および演算子を使用した式の作成

このレッスンでは、演算を実行して値を返すための式を作成する方法を説明します。

式は、演算を実行して値を返すコードのセグメントです。たとえば、単純な加算式は次のようになります。

5 + 4

式 `5 + 4` は、計算が実行されると、値 `9` を返します。この式は、操作の対象であるオペランド (`5` および `4`) と、実行する操作を指定する演算子 (`+`) の 2 つの部分から構成されます。

式によって返された値を使用する

式を役立てるには、返される値を使用して何らかの処理を行う必要があります。最も一般的な処理は、次に示すように、戻り値を変数に代入することです。

VB

```
Dim anInteger As Integer = 5 + 4
```

この例では、`anInteger` という新しい **Integer** 変数を宣言して、`5 + 4` から返される値をその変数に代入しています。

算術演算子

式の一般的な用途は、変数に対して演算 (加算、減算、乗算、除算) を実行することです。次の表は、演算に使用される一般的な演算子を示しています。

演算子	説明	例
+ (加算)	2 つのオペランドの和を返す	5 + 4
- (減算)	2 つのオペランドの差を返す	5 - 4
* (乗算)	2 つのオペランドの積を返す	5 * 4
/ (除算)	2 つのオペランドの商を返す	5 / 4

演算を実行するときに使用する変数の種類が、その結果に影響を及ぼす場合があります。2 つの数値を除算すると、戻り値が整数でなくなることがしばしばあります。たとえば、3 を 2 で割ると 1.5 になります。この式の戻り値を **Integer** 変数に代入すると、最も近い整数である 2 に端数処理されます。このため、除算を実行するときは、戻り値を格納する変数として **Double** 変数を使用する必要があります。

メモ：

また、Visual Basic の変換関数を使用することにより、変数のデータ型を別のデータ型に変換できます。詳細については、「詳細情報 : ある変数の型を別の型に変換する」を参照してください。

やってみよう

数値を加算するには

1. [ファイル] メニューの [新しいプロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Arithmetic`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、フォームに 2 つの Textbox コントロールをドラッグします。
5. ツールボックスから、フォームに Button コントロールをドラッグします。
6. Buttonをダブルクリックしてコード エディタを開きます。
7. **Button1_Click** イベント プロシージャに次のコードを入力します。

VB

```
Dim A As Double = Textbox1.Text
Dim B As Double = Textbox2.Text

MsgBox(A + B)
MsgBox(A - B)
MsgBox(A * B)
MsgBox(A / B)
```

最初の 2 行は、このプログラムで使用する数値を格納する変数 `A` と変数 `B` を宣言し、2 つの **TextBox** コントロールの値 (テキスト) を、変数 `A` と変数 `B` に代入します。

後半の 4 行は、2 つの変数と基本的な算術演算子を 1 つずつ使用した式を作成し、その式の結果をメッセージ ボックスに表示します。

8. F5 キーを押してアプリケーションを実行します。
9. 各テキスト ボックスに数値を入力し、Button1 をクリックします。

メモ :
数値以外の文字をテキスト ボックスに入力すると、エラーが発生します。

入力した 2 つの数値と、4 つの基本的な算術演算子 (加算、減算、乗算、除算) を 1 つずつ使用した式が作成されます。各式の結果が、メッセージ ボックスに表示されます。

次の手順

このトピックでは、式を作成して使用する方法を説明しました。また、オペランドと演算子、および式の構築方法についても説明しました。この時点で、次のレッスン (「比較 : 値を比較するための式の使用」) に進むか、変数の型を変換する方法について「詳細情報 : ある変数の型を別の型に変換する」でより詳細に学習するかのどちらかを選択できます。「詳細情報」を選択する場合は、上記のレッスンを完了してから、次のレッスンに進んでください。

参照

処理手順

詳細情報 : ある変数の型を別の型に変換する

概念

Visual Basic における算術演算子

詳細情報：ある変数の型を別の型に変換する

既に説明したように、変数にはさまざまな型を割り当てることができます。型によって、変数に格納できるデータの種類が決定されます。**Integer** 変数は小数点のない数値のみ格納できます。**String** 変数はテキストのみ格納できます。

では、**String** を入力する必要のある TextBox コントロールに、**Integer** を表示する場合はどうなるでしょうか。その場合は、データをある型から別の型に変換する必要があります。このトピックでは、データをある型から別の型に変換する方法を説明し、データ変換に使用するいくつかのテクニックと、その一般的な落とし穴について説明します。

変数をテキストに変換する

Visual Basic のすべての変数は、**CStr** (**C**onvert to **Str**ing の略) と呼ばれる特殊な関数を使用することにより、テキストに変換できます。この関数は、その名前が示すとおり、変数によって表されたデータを **String** として返します。次の手順は、**Integer** をテキストに変換する簡単な例を示しています。

やってみよう

変数をテキストに変換するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Conversion`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開きます。
5. Form1_Load イベント ハンドラに次のコードを追加します。

   VB

   ```
   Dim anInteger As Integer = 54
   MsgBox(CStr(anInteger))
   ```

   このコードは、まず `anInteger` という Integer 変数を宣言し、これに 54 という値を代入します。その後、**CStr** 関数を呼び出すことにより、その値をテキストに変換し、メッセージ ボックスに表示します。
6. `F5` キーを押して、アプリケーションをビルドおよび実行します。`54` という文字を含むメッセージ ボックスが表示されます。

   次の手順も試してみてください。コード エディタで、`MsgBox(CStr(anInteger))` という行を `MsgBox(anInteger)` に変更し、`F5` キーを押して実行します。どうなりましたか。プログラムは前の手順とまったく同様に動作します。`Visual Basic` は非常に高機能であるため、**Integer** をメッセージ ボックスに表示するためには、これをテキストに変換する必要があることを判断できるのです。しかし、自動的に変換できない変数の型も多いため、すべての状況でこの機能に依存できるわけではありません。このため、変数が自動的にテキストに変換される場合でも、常に **CStr** 関数を使用することをお勧めします。

CStr 関数は、**Integer** 変数をテキストに変換するだけでなく、**Double** や **Long** など、任意の数値データ型に使用できます。また、**Date** データ型や **Boolean** データ型をテキストに変換するときにも使用できます。データ型の詳細については、「詳細情報 : データ型」を参照してください。

数値データ型間で変換する

演算に関するレッスンで説明したように、算術演算の結果を **Integer** で表現できないことがあります。`Visual Basic` には、数値をテキストに変換する関数と同様に、変数をある数値データ型から別の数値データ型に変換する関数もあります。たとえば、**CDbl** (`Convert to Double` の略) 関数を算術演算で使用すると、処理対象が **Integer** 変数である場合に小数の値を返すことができます。次の手順は、`2` つの整数を除算するときに **CDbl** 関数を使用する方法を示しています。

やってみよう

数値データ型間で変換するには

1. コード エディタで、前の手順で入力したコードを削除し、次のコードを入力します。

   VB

   ```
   Dim A As Integer = 1
   Dim B As Integer = 2
   MsgBox(CDbl(A / B))
   ```

このコードは、まず 2 つの **Integer** 変数 (A および B) を宣言し、これらに 1 および 2 という値を代入します。その後、**CDbl** 関数を使用して、除算操作の結果 (A / B) を変換し、メッセージ ボックスに表示します。

2. F5 キーを押して、アプリケーションをビルドおよび実行します。0.5 という文字を含むメッセージ ボックスが表示されます。

Visual Basic には、他の型の数値変数に使用する関数もあります。たとえば、**Double** 型の 2 つの変数を加算して、その結果を最も近い整数に丸める場合は、**CInt** 関数を使用します。その他の数値変換関数には、**CByte**、**CDec**、**CLng**、および **CShort** があります。Visual Basic のすべての変換関数を記載した一覧については、「データ型変換関数」を参照してください。

次の手順

このレッスンでは、数値変数をテキストに変換する方法と、数値変数の型を別の型に変換する方法を説明しました。次のレッスン (「比較 : 値を比較するための式の使用」) では、式を評価する方法を説明します。

参照

処理手順

演算 : 変数および演算子を使用した式の作成

概念

詳細情報 : データ型

その他の技術情報

Visual Basic における型変換

比較：値を比較するための式の使用

このレッスンでは、比較演算子を使用して値を比較する式を作成する方法について説明します。

前回のレッスンでは、算術演算子を使用して数式を作成し、数値を返す方法について説明しました。もう 1 つの演算子である比較演算子を使用すると、数値を比較して **Boolean** (**True** または **False**) 値を返すことができます。

比較演算子は、値を比較してその比較に基づいて決定を行う場合に最も頻繁に使用されます。プログラムでの決定の詳細については、「二者択一のプログラム : If...Then ステートメント」を参照してください。

比較演算子を次の表にまとめて示します。

演算子	説明	例
= (等号) (等しい)	左側の数値が右側の数値と等しい場合に **True** を返します。	5 = 4 (false) 4 = 5 (false) 4 = 4 (true)
<> (等しくない)	左側の数値が右側の数値と等しくない場合に **True** を返します。	5 <> 4 (true) 4 <> 5 (true) 4 <> 4 (false)
> (不等号) (より大)	左側の数値が右側の数値より大きい場合に **True** を返します。	5 > 4 (true) 4 > 5 (false) 4 > 4 (false)
< (不等号) (より小)	左側の数値が右側の数値より小さい場合に **True** を返します。	5 < 4 (false) 4 < 5 (true) 4 < 4 (false)
>= (以上)	左側の数値が右側の数値以上の場合に **True** を返します。	5 >= 4 (true) 4 >= 5 (false) 4 >= 4 (true)
<= (以下)	左側の数値が右側の数値以下の場合に **True** を返します。	5 <= 4 (false) 4 <= 5 (true) 4 <= 4 (true)

やってみよう

式を比較するには次の操作を実行します。

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Comparison`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックス から、フォームに 2 つの Textbox コントロールをドラッグします。
5. ツールボックスから、フォームに Button コントロールをドラッグします。
6. Button をダブルクリックしてコード エディタを開きます。

7. **Button1_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Dim A As Double = CDbl(Textbox1.Text)
   Dim B As Double = CDbl(Textbox2.Text)
   MsgBox(A > B)
   MsgBox(A < B)
   MsgBox(A = B)
   ```

   最初の 2 行は、このプログラムで使用する数値を格納する変数 `A` と変数 `B` を宣言します。`CDbl` ステートメントを使用して、`Textbox1` と `Textbox2` のテキストを数値に変換します。最後に、最後の 3 行で 3 つの基本的な比較演算子を使用して 2 つの変数を比較する式を作成し、その式の結果を 3 つのメッセージ ボックスに表示します。

8. F5 キーを押してアプリケーションを実行します。

9. 各テキスト ボックスに数値を入力して、[Button1] をクリックします。

   `A` (最初のテキスト ボックスに入力した数値) が `B` (2 番目のテキスト ボックスに入力した数値) より大きい場合、最初のメッセージ ボックスに **True** が表示されます。それ以外の場合は **False** が表示されます。`A` が `B` より小さい場合、2 番目のメッセージ ボックスに **True** が表示されます。両方の数値が同じ場合、3 番目のメッセージ ボックスに **True** が表示されます。

   テキスト ボックスに異なる数値を入力して、結果が変わるのを確認してください。

次の手順

このレッスンでは、比較演算子を使用して数値を比較する方法について説明しました。次のレッスンでは、アクションを実行するプロシージャ コードを作成して呼び出す方法について説明します。

次のレッスン : 「コンピュータが行う処理の記述 : 初めてのプロシージャの作成」

参照

処理手順

演算 : 変数および演算子を使用した式の作成

詳細情報 : ある変数の型を別の型に変換する

二者択一のプログラム : If...Then ステートメント

概念

Visual Basic における比較演算子

コンピュータが行う処理の記述：初めてのプロシージャの作成

このレッスンでは、他のコード ブロックから実行できる独立したコード ブロックであるプロシージャの作成方法を説明した後、プロシージャのパラメータを作成する方法を説明します。

プロシージャは、プログラムに対して処理の実行を指示するコードの一部分です。既に前のレッスンでプロシージャを使用しています。たとえば、**MsgBox** 関数は、ダイアログ ボックスの表示という処理を実行する組み込みプロシージャです。

Visual Basic には、一般的な処理を実行するための組み込みプロシージャが数多く用意されていますが、組み込みプロシージャでは不可能な処理をプログラムで実行する必要のある状況が必ず発生します。たとえば **MsgBox** 関数は、画像付きのダイアログ ボックスを表示できません。このタスクを実行するには、独自のプロシージャを記述する必要があります。

プロシージャとは何か

プロシージャは、他のコード ブロックから実行できる独立したコード ブロックです。一般に各プロシージャには、1 つのタスクを実行するために必要なコードが含まれています。たとえば、wave ファイルを実行するために必要なコードを含む **PlaySound** というプロシージャを定義できます。プログラムで音を鳴らす必要が生じるたびに、サウンドを再生するためのコードを記述することもできますが、プログラムのどの場所からも呼び出すことができる単一のプロシージャを作成する方が、より効率的です。

プロシージャの起動、つまり実行は、コード内でそのプロシージャを呼び出すことによって行われます。たとえば、**PlaySound** プロシージャを実行するには、次のように、プロシージャの名前のコード行をプログラムに追加するだけです。

```
PlaySound
```

ただこれだけです。プログラムは、この行に到達すると、**PlaySound** プロシージャに移動し、そのプロシージャに含まれるコードを実行します。その後、プログラムは、**PlaySound** の呼び出しの次に記述されているコード行に戻ります。

プロシージャは、数の制限なく呼び出すことができます。プロシージャの実行は、呼び出された順序で行われます。たとえば、**DisplayResults** というプロシージャも定義している場合、**PlaySounds** プロシージャを実行した後にこのプロシージャを実行するには、次のようにプロシージャを呼び出します。

```
PlaySounds

DisplayResults
```

関数とサブ

プロシージャには、関数とサブルーチン (サブとも呼ばれる) の 2 種類があります。関数が呼び出し元プロシージャに値を返すのに対し、サブはコードを実行するのみです。サブは、次の例のように、サブの名前を含むコード行がプログラムに追加されたときに呼び出されます。

```
DisplayResults
```

これに対して関数は、コードを実行するだけでなく値を返すため、異なる方法で呼び出されます。たとえば、曜日を示す **Integer** を返す **GetDayOfWeek** という関数があるとします。この関数を呼び出すには、まず戻り値を格納する変数を宣言してから、後で利用できるように戻り値をその変数に代入します。コードは次のようになります。

```
Dim Today As Integer

Today = GetDayOfWeek
```

この例では、関数によって返された値は、後で利用できるよう、`Today` という名前の関数にコピーおよび格納されます。

プロシージャの記述

プロシージャの記述は、プロシージャの宣言を記述することから始まります。プロシージャの宣言には、複数の役割があります。つまり、プロシージャが関数とサブのどちらであるのかの宣言、プロシージャ名の指定、およびプロシージャが受け取るパラメータの指定です (パラメータについては、このレッスンの後半で詳しく説明します)。簡単なプロシージャの宣言の例を次に示します。

VB

```
Sub MyFirstSub()
End Sub
```

`Sub` キーワードにより、このプロシージャがサブであり、値を返さないことが、プログラムに対して宣言されます。次に、サブの名前 (`MyFirstSub`) が宣言され、空のかっこにより、このプロシージャにはパラメータがないことが示されます。最後に、**End Sub** キーワードによってサブルーチンの終了が示されます。このサブによって実行されるすべてのコードは、これら 2 行の間に記述されます。

関数の宣言もこれと似ていますが、戻り値の型の指定 (**Integer**、**String** など) という手順が追加されます。たとえば、**Integer** を返す関数は次のようになります。

VB

```
Function MyFirstFunction() As Integer
End Function
```

`As Integer` キーワードにより、この関数が **Integer** 値を返すことが示されます。関数から値を返すには、次の例に示すように **Return** キーワードを使用します。

VB

```
Function GetTheNumberOne() As Integer
    Return 1
End Function
```

このプロシージャは数値 1 を返します。

やってみよう

プロシージャを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`MyFirstProcedure`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開きます。
5. コード エディタで `End Class` という行を確認します。これは、フォームを構成するコード セクションの末尾です。この行のすぐ前に、次のプロシージャを追加します。

   VB

   ```
   Function GetTime() As String
     Return CStr(Now)
   End Function
   ```

   この関数は、**Now** 組み込みプロシージャを使用して現在時刻を取得し、さらに **CStr** 関数を使用して、**Now** から返された値を人間が判読できる **String** に変換します。最後に、その **String** 値が関数の結果として返されます。
6. 前の手順で追加した関数の上に、次の **Sub** を追加します。

   VB

   ```
   Sub DisplayTime()
     MsgBox(GetTime)
   End Sub
   ```

   このサブは、関数 `GetTime` を呼び出して、これによって返された値をメッセージ ボックスに表示します。
7. 最後に、**Form1_Load** イベント ハンドラに、次に示す **DisplayTime** サブを呼び出す行を追加します。

   VB

   ```
   DisplayTime()
   ```
8. F5 キーを押してプログラムを実行します。

   プログラムが起動すると、`Form1_Load` イベント プロシージャが実行されます。このプロシージャは **DisplayTime** サブを呼び出します。し

たがって、プログラムの実行は **DisplayTime** サブ プロシージャに移動します。さらにこのサブは **GetTime** 関数を呼び出すため、プログラムの実行は **GetTime** 関数に移動します。この関数は、時間を表す **String** を **DisplayTime** サブ プロシージャに返し、このサブ プロシージャが、その文字列をメッセージ ボックスに表示します。サブの実行が終了すると、プログラムは通常どおり次へ進み、フォームが表示されます。

関数およびサブのパラメータ

状況によって、プロシージャに追加情報を提供する必要が生じる場合があります。たとえば、**PlaySound** プロシージャでは、複数の異なるサウンドのうちいずれかを再生する必要がある場合があります。どのサウンドを再生するのかに関する情報を提供するには、パラメータを使用します。

パラメータは、変数に非常に良く似ています。パラメータは、変数と同様、型と名前を持ち、情報を格納します。またプロシージャ内で変数と同様に使用できます。パラメータと変数の主な相違点は、次の 2 つです。

- パラメータは、個別のコード行ではなく、プロシージャの宣言の中で宣言されます。
- パラメータは、宣言されたプロシージャ内でのみ使用できます。

パラメータの宣言は、プロシージャ宣言の中の、プロシージャ名の後ろに続くかっこ内で行われます。**As** キーワードを使用して型が宣言されます。また、各パラメータの前には、通常 **ByVal** キーワードが付加されます。このキーワードは、ユーザーが追加しない場合、Visual Basic によって自動的に追加されます。このキーワードを使用すると、このレッスンでは示さない非常に高度な機能を実行できます。

パラメータを持つサブの例を次に示します。

VB

```
Sub PlaySound(ByVal SoundFile As String, ByVal Volume As Integer)
  My.Computer.Audio.Play(SoundFile, Volume)
End Sub
```

パラメータの値を指定してサブを呼び出すには、次のように記述します。

VB

```
PlaySound("Startup.wav", 1)
```

関数のパラメータも、サブとまったく同じ方法で宣言できます。

やってみよう

パラメータを取る関数を作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`parameters`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックス から、フォームに 2 つの Textbox コントロールをドラッグします。
5. ツールボックスから、フォームに Button コントロールをドラッグします。
6. Buttonをダブルクリックしてコード エディタを開きます。
7. `Button1_Click` イベント ハンドラの **End Sub** 行の直後に、次のプロシージャを追加します。

   VB

   ```
   Function AddTwoNumbers(ByVal N1 As Integer, ByVal N2 As Integer) _
     As Integer
     Return N1 + N2
   End Function
   ```
8. `Button1_Click` プロシージャに次のコードを追加します。

VB

```
Dim aNumber As Integer = CInt(Textbox1.Text)
Dim bNumber As Integer = CInt(Textbox2.Text)
MsgBox(AddTwoNumbers(aNumber, bNumber))
```

このコードは、2 つの integer を宣言し、2 つのテキスト ボックスに入力されたテキストを integer 値に変換します。その後、これらの値を **AddTwoNumbers** 関数に渡し、戻り値をメッセージ ボックスに表示します。

9. F5 キーを押してプログラムを実行します。
10. 各テキスト ボックスに数値を入力し、ボタンをクリックします。2 つの数値が加算され、その結果がメッセージ ボックスに表示されます。

次の手順

このレッスンでは、関数とサブの相違点、およびこれら 2 種類のプロシージャを作成する方法を説明しました。また、プロシージャの呼び出し方法、およびパラメータを受け取るプロシージャの作成方法も説明しました。

次のレッスンでは、**For...Next** ステートメントを使用して処理を繰り返す方法を説明します。

次のレッスン : 「プログラムでの処理の繰り返し : For...Next ループによるループ処理」

参照
処理手順
比較 : 値を比較するための式の使用
概念
Visual Basic におけるプロシージャ

プログラムでの処理の繰り返し：For...Next ループによるループ処理

このレッスンでは、**For...Next** ステートメントを使用してプログラムで処理を繰り返し、この処理が何回実行されたかをカウントする方法について説明します。

プログラムを記述する場合、処理の繰り返しが必要なことがしばしばあります。たとえば、画面に一連の数字を表示するメソッドを記述するとします。必要に応じて何度でも数字を表示するコード行を繰り返すことができます。

For...Next ループを使用すると、数値を指定して、ループ内のコードを指定された回数だけ繰り返すことができます。**For...Next** ループがコード内で使用されている例を次に示します。

VB

```
Dim i As Integer = 0
For i = 1 To 10
  DisplayNumber(i)
Next
```

For...Next ループは、カウンタ変数 `i` で始まります。これは、ループを実行した回数をカウントするために使用される変数です。次の行 (`For i = 1 to 10`) は、ループを繰り返す回数と、 `i` に設定される値をプログラムに通知します。

コードが **For...Next** ループに入ると、まず `i` に最初の値 (この場合は 1) が設定されて開始されます。次に、`For` 行と `Next` 行の間のコード行が実行されます。この場合、パラメータに `i` (この場合も 1) を指定して `DisplayNumber` メソッドが呼び出されます。

`Next` 行に達すると、`i` に `1` が加算され、プログラムの実行は `For` 行に戻ります。この処理は、`i` の値が `For` 行の 2 番目の数値 (この場合は 10) より大きくなるまで繰り返されます。2 番目の数値を超えると、プログラムは `Next` 行の次のコードに進みます。

やってみよう

For...Next ステートメントを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`ForNext`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックス から、1 つの TextBox コントロールと 1 つの Button コントロールをフォームにドラッグします。
5. Buttonをダブルクリックしてコード エディタを開きます。
6. **Button1_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Dim i As Integer = 0
   Dim NumberOfRepetitions As Integer = CInt(Textbox1.Text)
   For i = 1 To NumberOfRepetitions
     MsgBox("This line has been repeated " & i & " times")
   Next
   ```
7. F5 キーを押してプログラムを実行します。
8. テキスト ボックスに数値を入力し、ボタンをクリックします。

   テキスト ボックスに指定した回数だけメッセージ ボックスが表示されます。

次の手順

このトピックでは、**For...Next** ループを使用して、指定された回数だけコードを繰り返す方法について説明しました。ここで、このシリーズの次の「二者択一のプログラム : If...Then ステートメント」に進むか、
「詳細情報 : Do...While と Do...Until を使用して条件が成立するまで処理を繰り返す」で別の種類のループについて参照するかを選択できま

す。

参照

処理手順

コンピュータが行う処理の記述 : 初めてのプロシージャの作成

関連項目

For...Next ステートメント (Visual Basic)

概念

条件判断構造

詳細情報：Do...While と Do...Until を使用して条件が成立するまで処理を繰り返す

このレッスンでは、**Do...While** ステートメントおよび **Do...Until** ステートメントを使用して、特定の条件に基づいてコードを繰り返す方法を説明します。

前のレッスンでは、**For...Next** ステートメントを使用して、コードのブロックを指定された回数だけループする方法を説明しました。では、コードを繰り返す必要がある回数が、特定の条件によって異なる場合はどうなるでしょうか。**Do...While** ステートメントおよび **Do...Until** ステートメントを使用すると、特定の条件が **True** の間、または特定の条件が **True** になるまでブロック内のコードを繰り返し実行できます。

たとえば、一連の数値を加算するプログラムがあり、合計が 100 を超えないようにするとします。この場合、**Do...While** ステートメントを使用して、次のように加算を実行できます。

VB

```
Dim sum As Integer = 0
Do While sum < 100
  sum = sum + 10
Loop
```

上記のコードでは、`Do While` 行は変数 `sum` を評価して、100 未満かどうかを確認します。100 未満の場合は、次のコード行が実行されます。100 以上の場合は、`Loop` に続く次のコード行に移動します。`Loop` キーワードは、`DoWhile` 行に戻り、`sum` の新しい値を評価するようにコードに通知します。

やってみよう

Do...While ステートメントを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`DoWhile`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックス から、1 つの TextBox コントロールと 1 つの Button コントロールをフォームにドラッグします。
5. Buttonをダブルクリックしてコード エディタを開きます。
6. **Button1_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Dim sum As Integer = 0
   Dim counter As Integer = 0
   Do While sum < 100
     sum = sum + CInt(Textbox1.Text)
     counter = counter + 1
   Loop
   MsgBox("The loop has run " & CStr(counter) & " times!")
   ```
7. F5 キーを押してプログラムを実行します。
8. テキスト ボックスに数値を入力し、ボタンをクリックします。

   100 になるまでに数値が何回加算されたかを示すメッセージ ボックスが表示されます。
9. [デバッグ] メニューの [デバッグの停止] をクリックしてプログラムを終了します。このプロジェクトは、開いたままにしておいてください。次の手順でプロジェクトに追加するものがあります。

Do...Until ステートメント

Do...While ステートメントは、条件が **True** の間ループを繰り返しますが、条件が **True** になるまでコードを繰り返す必要がある場合もあります。次のように **Do...Until** ステートメントを使用できます。

VB

```
Dim sum As Integer = 0
Do Until sum >= 100
  sum = sum + 10
Loop
```

このコードは、**Do...While** ステートメントのコードと似ていますが、このコードでは `sum` 変数を評価して、100 以上かどうかを確認する点が異なります。

やってみよう

Do...Until ステートメントを使用するには

1. `MsgBox` 行の下に次のコードを追加します。

   VB

   ```
   Dim sum2 As Integer = 0
   Dim counter2 As Integer = 0
   Do Until sum2 >= 100
     sum2 = sum2 + CInt(Textbox1.Text)
     counter2 = counter2 + 1
   Loop
   MsgBox("The loop has run " & CStr(counter2) & " times!")
   ```

2. F5 キーを押してプログラムを実行します。

3. テキスト ボックスに数値を入力し、ボタンをクリックします。

   100 以上になるまでに数値が何回加算されたかを示す 2 つ目のメッセージ ボックスが表示されます。

次の手順

このトピックでは、**Do...While** ループおよび **Do...Until** ループを使用して、条件に従ってコードを繰り返す方法を説明しました。ここで、次のレッスン「二者択一のプログラム : If...Then ステートメント」に進むことができます。

参照

処理手順

プログラムでの処理の繰り返し : For...Next ループによるループ処理

関連項目

Do...Loop ステートメント (Visual Basic)

二者択一のプログラム : If...Then ステートメント

このレッスンでは、**If...Then** ステートメントを使用して、条件に基づいてコードを実行する方法を説明します。

プログラムは、さまざまな条件に応じてさまざまな処理を実行する必要があります。たとえば、プログラムで曜日を確認した後、曜日によって異なる処理を実行できます。**If...Then** ステートメントを使用すると、条件を評価した後、その条件の結果に基づいて、異なるコード セクションを実行できます。

If...Then ステートメントの使用例を次に示します。

VB

```
If My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Monday Then
  MsgBox("Today is Monday!")
End If
```

このコードを実行すると、条件 (**If** から **Then** までの部分) が評価されます。条件が true の場合、次のコード行が実行され、メッセージ ボックスが表示されます。条件が false の場合、コードは **End If** 行にスキップします。つまり、このコードは、"今日が月曜日の場合にメッセージを表示する" ように指定しています。

やってみよう

If...Then ステートメントを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`IfThen`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開きます。
5. **Form1_Load** イベント ハンドラに次のコードを追加します。

   VB

   ```
   If My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Saturday Or _
     My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Sunday Then
     MsgBox("Happy Weekend!")
   End If
   ```
6. F5 キーを押してプログラムを実行します。

   今日が土曜日または日曜日の場合は、`Happy Weekend!` というメッセージ ボックスが表示されます。その他の場合は、メッセージ ボックスは表示されません。
7. [デバッグ] メニューの [デバッグの停止] をクリックしてプログラムを終了します。このプロジェクトは、開いたままにしておいてください。次の手順「**Else** 句を使用するには」で追加するものがあります。

上記の例では、**If...Then** ステートメントで **Or** 演算子を使用して複数の条件 ("今日は土曜日 **Or** 今日は日曜日") を評価しました。**Or** 演算子および **And** 演算子を使用して、1 つの **If...Then** ステートメントで必要な数だけ条件を評価できます。

Else 句

If...Then ステートメントを使用して、条件が true の場合にコードを実行する方法を説明しましたが、条件が true の場合と false の場合で異なる一連のコードを実行するにはどうするとよいでしょうか。この場合は、**Else** 句を使用します。**Else** 句を使用すると、条件が false の場合に実行されるコードのブロックを指定できます。**Else** 句の使用例を次に示します。

VB

```
If My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Friday Then
  MsgBox("Today is Friday!")
```

```
Else
  MsgBox("It isn't Friday yet!")
End If
```

この例では、式が評価されます。true の場合、次のコード行が実行され、最初のメッセージ ボックスが表示されます。false の場合、コードは **Else** 句に飛び、**Else** の次の行が実行されて、2 番目のメッセージ ボックスが表示されます。

やってみよう

Else 句を使用するには

1. **If...Then** ステートメントのコードを次のように変更します。

   VB

   ```
   If My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Saturday Or _
     My.Computer.Clock.LocalTime.DayOfWeek = DayOfWeek.Sunday Then
     MsgBox("Happy Weekend!")
   Else
     MsgBox("Happy Weekday! Don't work too hard!")
   End If
   ```

2. F5 キーを押してプログラムを実行します。プログラムは、対応する内容と共に週末か平日かを示すメッセージ ボックスを表示します。

   メモ：

   両方のコード ブロックの実行をテストするには、Windows タスク バーの時刻をダブルクリックして、曜日を変更します (タスク バーとは、Windows の [スタート] ボタンがあるバーです。既定ではデスクトップの下部に表示され、時刻はその右隅に表示されています)。

次の手順

このレッスンでは、**If...Then** ステートメントと **Else** 句を使用して、実行時に、条件に基づいてコードのブロックを選択的に実行する方法を説明しました。次のレッスンについては、「Select Case を使用して複数の選択肢から処理を決定する」を参照して実行するコードを選択する方法を調べるか、「問題が発生した場合の対処 : エラー処理」に進むかを選択できます。

参照

処理手順

プログラムでの処理の繰り返し : For...Next ループによるループ処理

比較 : 値を比較するための式の使用

関連項目

If...Then...Else ステートメント (Visual Basic)

Select Case を使用して複数の選択肢から処理を決定する

このレッスンでは、**Select Case** ステートメントを使用して、複数の条件に基づいてコードを実行する方法を説明します。

前のレッスンでは、**If...Then** ステートメントを使用して、異なる条件ごとに異なるコード ブロックを実行する方法を説明しました。**If...Then** ステートメントでも、**ElseIf** キーワードを使用すると、複数の条件を評価できますが、**Select Case** ステートメントでは、はるかに効果的な方法で複数の条件を評価できます。

Select Case ステートメントでは、必要に応じて数の制限なく条件 (ケース) を設定できるため、多くの選択肢がある状況のコードを記述するのに便利です。たとえば、プログラムで色の選択を格納する **String** 変数を使用しており、色の値を取得する必要がある場合を考えてみましょう。**Select Case** ステートメントのコードは次のようになります。

VB

```
Select Case Color
  Case "red"
      MsgBox("You selected red")
  Case "blue"
      MsgBox("You selected blue")
  Case "green"
      MsgBox("You selected green")
End Select
```

このコードを実行すると、**Select Case** 行で式の値 (`Color`) が決定されます。このとき、`Color` が **String** 変数で、この変数が **Select Case** ステートメントを含むメソッドのパラメータであると仮定します。次に、`Color` の値が最初の **Case** ステートメントの値と比較されます。値が同じであれば、次のコード行が実行され、**End Select** 行にスキップします。値が同じでなければ、次の **Case** 行が評価されます。

Case ステートメントでは、さまざまな形式を使用できます。上の例では、**String** を使用しています。この他にも、任意のデータ型や式を使用できます。

To キーワードを次のように使用すると、一定の範囲の数値を評価できます。

VB

```
Case 1 To 10
```

この例では、1 から 10 までの数値であれば一致していると見なされます。

また、単一の **Case** ステートメント内で、次のように値をコンマで区切ることによって、複数の値を評価できます。

VB

```
Case "red", "white", "blue"
```

この例では、3 つの値のうちいずれかであれば一致していると見なされます。

さらに、次のように比較演算子と **Is** キーワードを使用することによって値を評価できます。

VB

```
Case Is > 9
```

この例では、9 よりも大きい数値であれば一致していると見なされます。

Case Else

上記の例は、可能性のある条件がすべてわかっている場合には役立ちますが、想定しなかった条件が発生した場合はどうなるでしょうか。たとえば、`Color` の値が `yellow` である場合、コードで評価されるのは上記の 3 つのケースのみであるため、一致していると見なされず、メッセージ ボックスが表示されません。

Case Else ステートメントを使用すると、一致が見つからなかった場合にコードを実行できます。次に例を示します。

VB

```
Select Case Color
  Case "red"
      MsgBox("You selected red")
  Case "blue"
      MsgBox("You selected blue")
  Case "green"
      MsgBox("You selected green")
  Case Else
      MsgBox("Please choose red, blue, or green")
End Select
```

上記のコードでは、`Color` の値が `yellow` である場合、この値と最初の 3 つの **Case** 行が比較されますが、一致が見つかりません。**Case Else** 行に到達すると、**End Select** に移動する前にコードの次の行が実行されます。

Select Case ステートメントを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`SelectCase`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. [ツールボックス] から、1 つの TextBox コントロールと 1 つの Button コントロールをフォームにドラッグします。
5. ボタンをダブルクリックしてコード エディタを開きます。
6. Button1_Click イベント ハンドラに次のコードを入力します。

   VB

   ```
   Dim Number As Integer = CInt(Textbox1.Text)
   Select Case Number
     Case 1
         MsgBox("Less than 2")
     Case 2 To 5
         MsgBox("Between 2 and 5")
     Case 6, 7, 8
         MsgBox("Between 6 and 8")
     Case 9 To 10
         MsgBox("Greater than 8")
     Case Else
         MsgBox("Not between 1 and 10")
   End Select
   ```
7. F5 キーを押してプログラムを実行します。
8. テキスト ボックスに名前を入力し、ボタンをクリックします。

   入力した数値に対応する **Case** ステートメントのメッセージを含むメッセージ ボックスが表示されます。

次の手順

このトピックでは、**Select Case** ステートメントを使用して、複数の条件から選択する方法を説明しました。これで、次のレッスン (「問題が発生した場合の対処 : エラー処理」) に進むことができます。

次のレッスン : 「問題が発生した場合の対処 : エラー処理」

参照

処理手順

問題が発生した場合の対処 : エラー処理

二者択一のプログラム : If...Then ステートメント

関連項目

Select...Case ステートメント (Visual Basic)

問題が発生した場合の対処：エラー処理

このレッスンでは、プログラムの基本的なエラー処理コードを作成する方法について説明します。

最適にデザインされたプログラムでも、エラーが発生することがあります。コード内の欠陥で、検出して修正できるエラーもあれば、プログラムの当然の結果であるエラーもあります。たとえば、プログラムが既に使用中のファイルを開こうとしているとします。このような場合、エラーの発生が予測されますが、回避できません。プログラマは、このようなエラーを予測して、プログラムでエラーを処理できるようにする必要があります。

ランタイム エラー

プログラムの実行中に発生するエラーは、ランタイム エラーと呼ばれます。ランタイム エラーは、プログラムがデザインで意図したことと異なる処理を実行しようとしたときに発生します。たとえば、プログラムが数値以外の文字列を数値に変換するなど、無効な操作を実行しようとすると、ランタイム エラーが発生します。

ランタイム エラーが生じると、プログラムは例外を発行します。例外は、エラーを処理するコードをプログラム内から検出することによって、そのエラーを処理します。そうしたコードが見つからない場合、プログラムは中断され、再開する必要があります。この場合、データが損失する可能性があるため、エラーの発生が予測されるすべての場所にエラー処理コードを作成することをお勧めします。

Try...Catch...Finally ブロック

Try...Catch...Finally ブロックを使用してコード内のランタイム エラーを処理できます。コードのセグメントに対して **Try** を実行できます。そのコードによって例外が発行された場合、**Catch** ブロックにジャンプし、**Catch** ブロック内のコードが実行されます。そのコードが完了すると、**Finally** ブロック内のコードが実行されます。**Try...Catch...Finally** ブロック全体は、**End Try** ステートメントによって閉じられます。次のコードは、各ブロックの使用例です。

VB

```
Try
  ' Code here attempts to do something.
Catch
  ' If an error occurs, code here will be run.
Finally
  ' Code in this block will always be run.
End Try
```

まず、**Try** ブロック内のコードが実行されます。エラーなしで実行された場合、プログラムは **Catch** ブロックをスキップして、**Finally** ブロック内のコードを実行します。**Try** ブロックでエラーが発生した場合、実行は直ちに **Catch** ブロックにジャンプし、そのブロックのコードが実行された後、**Finally** ブロック内のコードが実行されます。

やってみよう

Try...Catch ブロックを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`TryCatch`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックス から、1 つの TextBox コントロールと 1 つの Button コントロールをフォームにドラッグします。
5. Buttonをダブルクリックしてコード エディタを開きます。
6. **Button1_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Try
     Dim aNumber As Double = CDbl(Textbox1.Text)
     MsgBox("You entered the number " & aNumber)
   Catch
     MsgBox("Please enter a number.")
   Finally
     MsgBox("Why not try it again?")
   ```

```
    End Try
```

7. F5 キーを押してプログラムを実行します。
8. テキスト ボックスに数値を入力し、ボタンをクリックします。入力した数値を表示するメッセージ ボックスが表示された後、再試行を勧めるメッセージが表示されます。
9. 次に、単語など、数値以外の値をテキスト ボックスに入力して、ボタンをクリックします。今度は、プログラムがテキスト ボックス内のテキストを数値に変換しようとすると、変換できずにエラーが発生します。**Try** ブロック内のコードを完了せずに、**Catch** ブロックが実行され、数値の入力を要求するメッセージ ボックスが表示されます。その後、**Finally** ブロックが実行され、再試行を勧められます。

次の手順

このレッスンでは、**Try...Catch...Finally** ブロックを使用して基本的なエラー処理構造を作成する方法を説明しました。このレッスンで、Visual Basic 言語の概要は終了です。次の一連のレッスンでは、プログラムのユーザー インターフェイスを作成する方法について説明します。

次のレッスン : 「プログラムの外観の作成 : Windows フォームの概要」

参照

処理手順

二者択一のプログラム : If...Then ステートメント

バグの理解 : 3 種類のプログラミング エラー

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

その他の技術情報

Visual Basic での構造化例外処理

プログラムの外観の作成：Windows フォームの概要

ユーザー インターフェイスは、プログラムを実行したときにユーザーが目にする部分です。通常、ユーザー インターフェイスは、メイン ウィンドウまたはフォームと、いくつかのコントロール (ボタンやテキスト入力フィールドなど) で構成されます。コンピュータ上で実行される Visual Basic プログラムは Windows フォーム アプリケーションと呼ばれ、ユーザー インターフェイスは Windows フォーム コントロールを使用して作成されます。

このセクションのレッスンでは、最もよく使用されるいくつかの Windows フォーム コントロールを使用して、ユーザー インターフェイスを作成する方法を示します。

このセクションの内容

関連するセクション

プログラムのユーザーとの通信：ユーザー インターフェイス

このレッスンでは、ユーザー インターフェイス (UI) とコントロールについて説明すると共に、コントロールを UI に追加する方法を説明します。

初期のパーソナル コンピュータでは、ユーザーは主にコマンド ラインを通じてプログラムと対話していました。プログラムは、起動してから一時停止して、ユーザーの入力を受け取ってから次の処理を実行していました。しかし、今日使用されているほとんどのプログラムは、テキストの入力や、ボタンのクリック、事前設定されたメニューからの項目選択などの方法で、ユーザーがプログラムとやり取りできるようにする (つまりインターフェイスとなる) 1 つ以上のウィンドウで実行されています。これ以降のいくつかのレッスンでは、独自の Windows ベースの UI をビルドする方法を説明します。

フォームを使用する

フォームは、UI の基本的なビルド ブロックです。プログラムの各フォームは、ユーザーに表示されるウィンドウを表します。Visual Basic IDE (統合開発環境) の作業において、フォームは、UI の設計に使用するデザイナであり、画像の描写に使用する Windows Paint のようなものです。

コントロールは、デザイナで UI の外観を作成するために使用します。コントロールは、定義済みの外観と動作を持つオブジェクトです。たとえば、Button コントロールの外観と動作はプッシュ ボタンに似ています。クリックすると、ボタンが押し下げられたような状態の外観に変化します。

Visual Basic の各コントロールには、それぞれ固有の用途があります。たとえば、TextBox コントロールはテキストの入力に使用し、PictureBox コントロールは画像の表示に使用します。Visual Basic には、50 以上のさまざまなコントロールが用意されています。また、ユーザー コントロールと呼ばれる独自のコントロールを作成することもできます。各種コントロールについては、以降のレッスンで詳しく説明します。

UI を設計するときは、コントロールをツールボックスからドラッグしてフォームにドロップし、コントロールの位置を決定して、好みの外観になるようコントロールのサイズを変更します。外観をさらに変更するには、[プロパティ] ウィンドウでフォームとコントロールのプロパティを設定します。たとえば、フォームとほとんどのコントロールには、背景色を設定するための **BackColor** プロパティがあります。

プロパティを使用すると、フォームやコントロールの動作も定義できます。たとえば、フォームの ShowInTaskbar プロパティは、プログラムの実行中にフォームが Windows タスク バーに表示されるかどうかを決定します。プロパティを使用すると、UI の外観と動作をカスタマイズできます。

やってみよう

フォームのプロパティを変更するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`FirstForm`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが作成されます。新しいフォームがメイン ウィンドウに表示され、Visual Basic IDE の右下端にある [プロパティ] ウィンドウに、そのフォームのプロパティが表示されます。
4. フォームを 1 回クリックして選択します。
5. [プロパティ] ウィンドウで、Text プロパティを「My First Form」に変更し、Enter キーを押します。

   新しい値を入力すると、フォームの上端に表示されたテキストが変更されます。
6. [プロパティ] ウィンドウで、BackColor プロパティを別の色に変更します。これには、ドロップダウン リストから色を選択します。

   BackColor プロパティは、特殊なインターフェイスを通じて変更できます。このインターフェイスを使用すると、色を選択する前にその色を見ることができ、現在システムで使用されている色や、標準の Web 色、または独自に選択した色から選択できます。また、[プロパティ] ウィンドウのボックスに色の名前 (`Red` など) を入力するだけでも変更できます。

   フォームの他のプロパティを変更してみてください。準備ができたら、次の手順に進みます。

フォームにコントロールを追加する

この手順では、通常 Visual Basic IDE の左側にあるツールボックス ウィンドウでコントロールをクリックし、それをフォームにドラッグすることによって、コントロールをフォームに追加します。その後、そのコントロールのプロパティを操作します。

フォームにコントロールを追加するには

1. ツールボックスから、**Button** コントロール、**TextBox** コントロール、Label コントロール、CheckBox コントロールの順に、フォームにドラッグします。
2. **Button** コントロールをクリックしてフォーム上をドラッグし、位置を変更します。

他のコントロールの近くにコントロールをドラッグしたときに、ガイドラインがどのように表示されるかを確認してください。これらのガイドラインは、コントロールの位置を正確に決定するために役立ちます。

3. UI が好みの外観になるまで、他のコントロールにもこのプロセスを繰り返します。
4. **Button** コントロールをクリックして、右下端に表示される白い四角形をクリックしてドラッグし、サイズを変更します。
5. しばらくの間、コントロールのプロパティを操作してみてください。フォーム上の各コントロールをクリックして選択し、[プロパティ] ウィンドウでそのコントロールのプロパティをいくつか変更します。**Font**、**BackColor**、**ForeColor**、**Text** などのプロパティを変更してみてください。
6. F5 キーを押してプログラムを実行します。先ほど追加したコントロールを含むウィンドウが表示されます。ボタンをクリックしたり、チェック ボックスをオンおよびオフにしたり、テキスト ボックスに入力したりできます。

次の手順

このレッスンでは、フォームを作成してコントロールを追加する方法を説明しました。また、[プロパティ] ウィンドウで、フォームとコントロールのプロパティを変更する方法も説明しました。以降のいくつかのレッスンでは、一部のコントロールについてより詳しく説明します。

次のレッスン : 「ユーザーとの対話 : ボタンを使用する」

参照
概念
詳細情報 : プロパティ、メソッド、イベントについて
その他の技術情報
プログラムの外観の作成 : Windows フォームの概要

ユーザーとの対話：ボタンを使用する

このレッスンでは、フォームに Button を追加する方法、ボタンの外観を変更する方法、およびボタンがクリックされたときに実行するコードを記述する方法を説明します。

ユーザーがプログラムと対話するための最も簡単な方法は、ボタンを使用することです。たとえば、多くのプログラムには終了ボタンがあります。前のレッスンで述べたとおり、Visual Basic の **Button** コントロールは、プッシュ ボタンのような外観を持ち、それに似た動作を行います。また **Button** コントロールには、プログラムの終了などの処理を実行するために使用する定義済みのイベントがあります。

ボタンを使用する

ボタンは、一般に、フォーム上にある浮き出た外観を持つ四角形のコントロールです。ただし、この外観を変更するために設定できるプロパティは数多くあります。最もわかりやすい例は、表示されるテキストを決定する Text プロパティです。このテキストは、Font プロパティによって決定されるフォントまたはタイプフェイスで表示されます。BackColor プロパティはボタンの色を決定し、ForeColor プロパティはテキストの色を決定します。

実行時にボタンがクリックされると、**Button** は Click イベントを発生させます。イベントが発生すると、そのイベントを受けてコントロールがコードを実行します。実行するコードを記述するには、イベント ハンドラを作成します。

イベント ハンドラは、イベントが発生したときに実行されるメソッドです。ボタンがクリックされると、ボタンの **Click** イベントのイベント ハンドラが実行されます。実際はそれほど難しいことではありません。次の例を見ればイベント ハンドラの記述方法が理解できます。イベントおよびイベント ハンドラについては、「ユーザーに反応するプログラム : イベント ハンドラを作成する」で詳しく説明します。

やってみよう

ボタンを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`ButtonExample`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、フォームに **Button** をドラッグします。
5. [プロパティ] ウィンドウで、**Text** プロパティを「`What time is it?`」に変更し、Enter キーを押します。

   このテキストはボタンの中に収まりません。
6. [プロパティ] ウィンドウで、AutoSize プロパティを選択して `True` に設定します。

   テキストに合わせてボタンのサイズが自動的に変更されます。
7. フォームでボタンをダブルクリックしてコード エディタを開きます。

   コード エディタが開き、**Button1_Click** というメソッドが表示されます。これが **Button1.Click** イベント ハンドラです。ここに記述するコードは、ボタンがクリックされたときに実行されます。
8. **Button1_Click** イベント ハンドラに次のコード行を追加します。

   VB

   ```
   MsgBox("The current time is " & Now.ToShortTimeString)
   ```
9. F5 キーを押してプログラムを実行します。

   プログラムが起動し、フォームが表示されます。**Button** をクリックすると、現在時刻を含むメッセージ ボックスが表示されます。

次の手順

このレッスンでは、フォームにボタンを追加する方法、およびユーザーがマウスでボタンをクリックしたときに実行されるコードを追加する方法を説明しました。次のレッスンでは、テキストの表示および取得を目的とするコントロール (Label および TextBox) を操作する方法を説明します。

次のレッスン : 「テキストの表示と受信 : ラベルとテキスト ボックスを使用する」

参照

処理手順
プログラムのユーザーとの通信 : ユーザー インターフェイス
関連項目
Button コントロールの概要 (Windows フォーム)
概念
詳細情報 : プロパティ、メソッド、イベントについて
その他の技術情報
プログラムの外観の作成 : Windows フォームの概要

テキストの表示と受信：ラベルとテキスト ボックスを使用する

ここでは、Label コントロールおよび TextBox コントロールを使用してテキストを表示し、ユーザーからのテキスト入力を受け入れる方法について説明します。

ユーザー情報を伝達して、ユーザーから情報を受け取る最も簡単な方法の 1 つは、テキストを使用することです。プログラムの機能に関するテキストを表示して、ユーザーからデータをテキストとして受信し、プログラムで使用できます。Visual Basic には、テキストの表示および受信用にデザインされた 2 つのコントロールが用意されています。それは、**Label** コントロールと **TextBox** コントロールです。

Label コントロールを使用したテキストの表示

Label コントロールは、テキストを表示するためのプライマリ コントロールです。このコントロールは、四角形で囲まれたテキストとしてフォームに表示されます。通常、この領域はフォームと同じ色であるため、フォーム上では単なるテキストのように見えます。

Label は主にテキストを表示することを意図しているため、**Label** コントロールの重要なプロパティのほとんどは、外観を制御するためのプロパティです。Text プロパティには、**Label** コントロールに表示されるテキストが含まれます。Font プロパティは、**Text** プロパティ内のテキストの表示フォントを決定します。ForeColor プロパティはテキスト自体の色を決定し、BackColor プロパティはテキストを囲む領域の色を決定します。

TextBox コントロールを使用したテキストの受信

テキストの表示および受信の両方が必要な場合、**TextBox** コントロールはこのジョブを処理するようにデザインされています。テキストを表示できるだけでなく、**TextBox** コントロールを使用すると、ユーザーは実行時に **TextBox** にテキストを入力し、プログラムはそのテキストを取得できます。

Label コントロールと同様に、**TextBox** コントロールの最も重要なプロパティは、外観に関連するプロパティです。重要なプロパティは、**Text** プロパティです。これは、**TextBox** コントロール内のテキストを表します。ユーザーが **TextBox** コントロールにテキストを入力すると、**Text** プロパティが更新され、変更が反映されます。したがって、**TextBox** コントロールに表示されるテキストは常に **Text** プロパティの値を反映します。

TextBox コントロールの動作に影響するプロパティもあります。Multiline プロパティは、**TextBox** コントロールが複数の行を許可するかどうかを決定します。このプロパティが **False** に設定されている場合、**TextBox** コントロールは常に 1 行分の高さになり、垂直方向に拡大することはできません。**True** に設定されている場合、**TextBox** コントロールは複数の行を許可し、任意の高さに設定できます。

やってみよう

Label コントロールおよび Textbox コントロールを使用してユーザー インターフェイスを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`TextBoxExample`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、**TextBox**、**Label**、および Button のコントロールをフォームにドラッグします。
5. **Label** を選択して、**TextBox** コントロールの上にドラッグします。
6. [プロパティ] ウィンドウで、**Label** コントロールの Text プロパティを次のコードに変更します。

   ```
   Enter your name and click the button.
   ```

これで、基本的なユーザー インターフェイスが作成されました。プログラムに数行のコードを追加すると、プログラムをテストできるようになります。

コードを追加してプログラムをテストするには

1. Button コントロールをダブルクリックしてコード エディタを開きます。

   コード エディタ内に **Button1_Click** イベント ハンドラが表示されます。
2. **Button1_Click** イベント ハンドラに次のコード行を追加します。

   VB

   ```
   MsgBox("Your Name is " & Textbox1.Text)
   ```
3. F5 キーを押してプログラムを実行します。

4. フォームが表示されたら、**TextBox** コントロールに名前を入力して、ボタンをクリックします。メッセージ ボックスが表示され、**TextBox** コントロールにテキストが表示されます。テキストを変更して、ボタンを再度クリックします。ボタンをクリックするたびに、更新されたテキストが表示されます。

次の手順

このトピックでは、**Label** コントロールおよび **TextBox** コントロールを使用してテキストの表示および受信を行う方法について説明しました。次のトピックでは、コントロール イベントを処理するメソッドを作成する方法について説明します。**Button_Click** イベント ハンドラなど、いくつかの基本的なイベント ハンドラの作成方法については既に説明しましたが、コントロールに発生するその他のイベントを処理するメソッドの作成方法について説明します。

次のレッスン : 「ユーザーに反応するプログラム : イベント ハンドラを作成する」

参照

処理手順

ユーザーとの対話 : ボタンを使用する

関連項目

Label コントロールの概要 (Windows フォーム)

TextBox コントロールの概要 (Windows フォーム)

ユーザーに反応するプログラム : イベント ハンドラを作成する

このレッスンでは、イベント ハンドラの作成方法について説明します。

これまでのレッスンで説明したように、コントロールにはプロパティ、メソッド、およびイベントがあり、ユーザー インターフェイスの作成に使用されます。イベントは、コントロールに対して発生する、関連のある事象です。たとえば、コントロールをクリックする、コントロールにテキストを入力する、コントロールの上にマウス ポインタを移動するなどです。

目的の事象が発生すると、コントロールはイベントを発生させ、何かが発生したことを通知するために、プログラムにシグナルが送信されることを示します。プログラムは、そのイベントを処理するメソッドがあるかどうかを確認します。このようなメソッドは、イベント ハンドラと呼ばれます。たとえば、「ユーザーとの対話 : ボタンを使用する」で作成したメソッドなど、ボタンをクリックしたときに実行されるメソッドがあります。

さまざまなコントロール イベントに対してイベント ハンドラを作成できます。このレッスンでは、ボタンの MouseEnter イベントおよび MouseLeave イベントを処理するイベント ハンドラを作成します。これらのイベントは、マウスをコントロールの上に移動したときに発生します。

やってみよう

MouseEnter イベントを処理するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`EventHandler`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、フォームに Button コントロールをドラッグします。
5. [プロパティ] ウィンドウで、AutoSize プロパティを **True** に設定します。
6. [表示] メニューの [コード] をクリックしてコード エディタを開きます。

   コード エディタのすぐ上に、2 つのドロップダウン ボックスが表示されます。左側のボックスには、フォーム上のすべてのコントロールの一覧、[Form1]、[(全般)]、および [(Form1 イベント)] が表示されます。右側のボックスには、左側のボックスに表示された項目に使用できる各イベントが表示されます。
7. 左側のボックスで Button1 を選択します。
8. 右側のボックスで MouseEnter を選択します。

   Button1_MouseEnter という名前の新しいイベント ハンドラが、コード エディタに表示されます。
9. **Button1_MouseEnter** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Button1.Text = "The Mouse has entered"
   ```
10. F5 キーを押して、アプリケーションを実行します。マウス ポインタをボタンの上に移動します。マウス ポインタが Button1 の上を通過すると、ボタンのテキストが変わります。

別のイベント ハンドラの追加

前の例では、マウス ポインタが Button1 の上を通過すると、ボタンのテキストが変わりましたが、マウス ポインタが離れてもテキストは元に戻りませんでした。マウスがボタンの上から離れたときにもテキストが変わるようにするには、**MouseEnter** イベント以外に **MouseLeave** イベントも処理する必要があります。

MouseLeave イベントを処理するには

1. コード エディタの左側のドロップダウン ボックスで Button1 が選択されていることを確認してから、右側のドロップダウン ボックスで **MouseLeave** を選択します。

   Button1_MouseLeave という名前の新しいイベント ハンドラがコード エディタに表示されます。
2. Button1_MouseLeave イベント ハンドラに次のコードを追加します。

VB

```
Button1.Text = "The mouse has left"
```

3. F5 キーを押して、アプリケーションを実行します。

   これで、マウス ポインタがボタンの上を通過すると、テキストが `The mouse has entered` に変わりますが、マウスがボタンの上を離れると、テキストが `The mouse has left` に変わります。

次の手順

このレッスンでは、コード エディタを使用してイベント ハンドラを作成する方法について説明しました。この時点で、順序どおり次のレッスン「ユーザー選択の取得 : チェック ボックスとオプション ボタンを使用する」に進むか、「詳細情報 : イベント ハンドラを共有する」でイベント ハンドラの詳細を参照するかを選択できます。2 番目のオプションを選択する場合は、そのレッスンで使用するために EventHandler プロジェクトを保存します。

参照

処理手順

テキストの表示と受信 : ラベルとテキスト ボックスを使用する

関連項目

Label コントロールの概要 (Windows フォーム)

TextBox コントロールの概要 (Windows フォーム)

詳細情報：イベント ハンドラを共有する

このレッスンでは、複数のコントロールのイベントを処理する共有イベント ハンドラを作成する方法について説明します。

前のレッスン「ユーザーに反応するプログラム：イベント ハンドラを作成する」では、Button コントロールの MouseEnter イベントおよび MouseLeave イベントに応答するコードを記述する方法について説明しました。では、2 つ以上の **Button** コントロールがあり、すべてのコントロールに同じメッセージを表示する場合はどうなるでしょうか。コントロールごとにイベント ハンドラにコードを記述することもできますが、それよりも簡単な方法があります。

MouseEnter イベントのイベント ハンドラ メソッドをよく見ると、**Method** 宣言 (`Private Sub Button1_MouseEnter(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.MouseEnter`) に **Handles** 句 (`Handles Button1.MouseEnter`) が含まれています。当然のことながら、**Handles** キーワードは、処理するイベントをイベント ハンドラに通知します。

複数のコントロール間でイベント ハンドラを共有するには、追加のコントロール名、および処理するイベントの名前を追加するだけです。それらのコントロールのいずれかに対してイベントが発生すると、イベント ハンドラに通知されます。たとえば、2 つの **Button** コントロールがあり、両方に同じイベント ハンドラを使用する場合、**Handles** 句は次のようになります。

```
Handles Button1.MouseEnter, Button2.MouseEnter.
```

これで、両方のコントロールの **MouseEnter** イベントを単一のメソッドで処理できますが、イベント ハンドラは、イベントを呼び出したコントロールをどのようにして知るのでしょうか。**Method** 宣言をもう一度見ると、`ByVal sender As Object` という句があります。**Sender** キーワードは、イベントを呼び出したオブジェクト (この場合はコントロール) をイベント ハンドラに通知します。

やってみよう

イベント ハンドラを共有するには

1. 前のレッスンで作成した EventHandler プロジェクトを開きます。プロジェクトを保存しなかった場合は、まず前のレッスン「ユーザーに反応するプログラム：イベント ハンドラを作成する」に戻って、そのレッスンの手順を完了する必要があります。
2. ソリューション エクスプローラで Form1.vb を選択して、[表示] メニューの [デザイナ] をクリックします。
3. ツールボックスから、フォームに別の **Button** コントロールをドラッグします。
4. [プロパティ] ウィンドウで、AutoSize プロパティを **True** に設定します。
5. [表示] メニューの [コード] をクリックしてコード エディタを開きます。
6. **Button1_MouseEnter** メソッドの宣言 (`Private Sub Button1_MouseEnter(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.MouseEnter`) で、**Handles** 句を `Handles Button1.MouseEnter, Button2.MouseEnter` と変更します。
7. イベント宣言の本体で、コードを次のように置換します。

   VB

   ```
   If sender.Equals(Button1) Then
     Button1.Text = "The mouse has entered Button1"
   Else
     Button2.Text = "The mouse has entered Button2"
   End If
   ```

   このコードは、送信元が Button1 かどうかを確認します。送信元が Button1 の場合は、Button1 の **Text** プロパティが更新されます。送信元が Button1 以外の場合は、Button2 の **Text** プロパティが更新されます。
8. **Button1_MouseLeave** メソッドの宣言で、**Handles** 句を次のように変更します。

   ```
   Handles Button1.MouseLeave, Button2.MouseLeave.
   ```
9. イベント宣言の本体で、コードを次のように置換します。

   VB

   ```
   sender.Text = "The mouse has left"
   ```

この場合、コードは送信元 (Button1 または Button2) の **Text** プロパティを同じ文字列に設定します。

10. F5 キーを押して、アプリケーションを実行します。

    これで、マウス ポインタがボタンの上を通過すると、テキストが `The mouse has entered` とボタンの名前に変わります。マウスがボタンの上を離れると、テキストが `The mouse has left` に戻ります。

    フォームにコントロールを追加して、コントロールを含むように **Handles** 句を変更してみましょう。同じ種類のコントロールでなくてもかまいません。

次の手順

このレッスンでは、複数のコントロール間で単一のイベント ハンドラを共有する方法について説明しました。次のレッスンでは、2 種類のコントロール (CheckBox コントロールおよび RadioButton コントロール) を使用して、ユーザーに選択肢を提示する方法について説明します。

次のレッスン : 「ユーザー選択の取得 : チェック ボックスとオプション ボタンを使用する」

参照

処理手順

ユーザーに反応するプログラム : イベント ハンドラを作成する

ユーザー選択の取得：チェック ボックスとオプション ボタンを使用する

このレッスンでは、チェック ボックスとオプション ボタンを使用して、ユーザーの選択項目を提示および取得する方法を説明します。

プログラムのユーザー インターフェイスを作成するときに、選択項目を提示する必要が生じることがよくあります。たとえば、ピザ レストランの注文を取るアプリケーションを記述する場合を考えてみましょう。この場合、ユーザーが、さまざまなトッピングのうちいずれか、またはすべてを選ぶことができるようにする必要があると考えられます。CheckBox コントロールは、このオプションを簡単に作成できるようにするビジュアル表示を提供します。

CheckBox コントロールは、テキスト ラベルと、ユーザーが選択できるボックスで構成されます。ユーザーがボックスをクリックすると、ボックス内にチェック マークが表示されます。ボックスを再度クリックすると、チェック マークは消えます。チェック ボックスの状態を取得するには、**CheckBox.Checked** プロパティを使用します。ボックスにチェック マークが表示されている場合は、このプロパティは **True** を返します。チェック マークが表示されていない場合は、このプロパティは **False** を返します。

やってみよう

チェック ボックスを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`UserChoices`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、フォームに 1 つの Button コントロールと 3 つの **CheckBox** コントロールをドラッグします。
5. [プロパティ] ウィンドウで、Checkbox1、Checkbox2、および Checkbox3 の Text プロパティを、それぞれ「`Pepperoni`」、「`Sausage`」、「`Mushrooms`」に変更します。
6. [プロパティ] ウィンドウで、Button1 の **Text** プロパティを「`Order Pizza`」に変更します。
7. フォームのボタンをダブルクリックします。**Button1_Click** イベント ハンドラがコード エディタで開きます。
8. **Button1_Click** イベント ハンドラに次のコードを追加します。

   VB

```
Dim toppings As String = ""
If CheckBox1.Checked = True Then
    toppings &= "Cheese "
End If
If CheckBox2.Checked = True Then
    toppings &= "Peppers "
End If
If CheckBox3.Checked = True Then
    toppings &= "Mushrooms"
End If
If toppings <> "" Then
    MsgBox("Your pizza has the following toppings: " & toppings)
End If
```

9. F5 キーを押してプログラムを実行します。フォームが表示されたら、いくつかのトッピングを選択してボタンをクリックします。選択したピザのトッピングを示すメッセージ ボックスが表示されます。

オプション ボタンを使用して排他的選択を行う

ここまでは、ユーザーが複数の選択項目のうちのいずれか、またはすべてを選ぶことができるようにする方法を説明しました。しかし、複数の選択肢のうち 1 つだけを選んでもらう必要がある場合はどうすればよいでしょうか。この場合は、RadioButton コントロールを使用します。

オプション ボタンは、チェック ボックスと異なり、常にグループの一部として機能します。1 つのオプション ボタンを選択すると、グループに含まれる他のすべてのオプション ボタンは直ちにオフになります。オプション ボタンのグループは、"内部で 1 つだけ選択できる選択項目のセット" と定義されま

す。

RadioButton コントロールのグループを使用すると、ユーザーに排他的オプションから選択させることができるようになります。たとえば、ピザのソースとして、通常のソースかスパイシー ソースのどちらかを選択できるが、両方は選択できないようにする場合が考えられます。**CheckBox** コントロールと同様、**RadioButton** コントロールの状態に関する情報も、**RadioButton.Checked** プロパティから取得できます。

オプション ボタンを使用するには

1. ツールボックスから、フォームに 2 つの **RadioButton** コントロールをドラッグします。
2. [プロパティ] ウィンドウで、RadioButton1 の **Text** プロパティを「`Regular Sauce`」に変更します。
3. RadioButton1 の Checked プロパティを **True** に設定します。

ヒント
選択項目のグループを定義するときは、常に 1 つの選択項目を既定として設定する必要があります。

4. [プロパティ] ウィンドウで、RadioButton2 の **Text** プロパティを「`Spicy Sauce`」に変更します。
5. フォームのボタンをダブルクリックして、**Button1_Click** イベント ハンドラをコード エディタで開きます。
6. **Button1_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   If RadioButton1.Checked = True Then
       MsgBox("You chose regular sauce")
   Else
       MsgBox("You chose spicy sauce")
   End If
   ```

7. F5 キーを押してプログラムを実行します。オプション ボタンのうち 1 つをオンにし、[Order Pizza] をクリックします。選択した項目を示すメッセージ ボックスが表示されます。

   両方のオプション ボタンを同時にオンにしようとしてみてください。オプション ボタンは排他的です。一方をオンにすると、もう一方は自動的にオフになります。

次の手順

このレッスンでは、**CheckBox** コントロールと **RadioButton** コントロールを使用して、ユーザー インターフェイスでユーザーに選択項目を提示する方法を説明しました。この時点で、次のレッスン (「ピクチャ描画 : イメージを表示する」) に進むか、
「詳細情報 : 複数グループのオプション ボタンを使用する」でオプション ボタンの複数のグループを作成する方法を学習するかのいずれかを選択できます。後者を選択する場合は、次のレッスンで使用するために UserChoices プロジェクトを保存する必要があります。

参照
関連項目
CheckBox コントロールの概要 (Windows フォーム)
RadioButton コントロールの概要 (Windows フォーム)

詳細情報：複数グループのオプション ボタンを使用する

このレッスンでは、一度に 1 つしか選択できないオプション ボタンのグループを、1 つのフォーム上に複数作成する方法を説明します。

前のレッスンでは、オプション ボタンのグループを作成して、一度に 1 つしか選択できない選択項目のセットをユーザーに提示する方法を説明しました。しかし、複数の選択項目のセットを提示した場合はどうなるでしょうか。この場合、フォーム上のすべての RadioButton コントロールは 1 つのグループとして扱われるため、選択できるオプション ボタンは 1 つのみとなります。

しかし、Visual Basic には、コンテナ コントロールという的確な名前を持つ、他のコントロールを格納できるコントロールがいくつかあります。コンテナ コントロールをフォーム上に配置し、**RadioButton** コントロールをそのコンテナ コントロール内に配置すると、同じフォーム上にオプション ボタンの複数のグループを含めることができます。

最もよく使用されるコンテナ コントロールは、GroupBox コントロールと Panel コントロールです。これら 2 つの主な相違点は、**GroupBox** コントロールの周囲には境界線が表示されるのに対し、**Panel** の周囲には境界線がないことです。コンテナ コントロールを使用してオプション ボタンをグループ化する場合は、境界線によって選択項目のグループをわかりやすい方法で区別できるため、**GroupBox** コントロールを使用することをお勧めします。

やってみよう

GroupBox をコンテナとして使用するには

1. 前のレッスンで作成した UserChoices プロジェクトを開きます。このプロジェクトを保存していない場合は、先に前のレッスン (「ユーザー選択の取得 : チェック ボックスとオプション ボタンを使用する」) に戻って、手順を実行する必要があります。
2. ソリューション エクスプローラで Form1.vb を選択して、[表示] メニューの [デザイナ] をクリックします。
3. ツールボックスから、フォームに **GroupBox** コントロールをドラッグします。
4. [プロパティ] ウィンドウで、**GroupBox** コントロールの Text プロパティを `Select a crust` に変更します。
5. **GroupBox** コントロールを選択したままの状態で、2 つの **RadioButton** コントロールをツールボックスからドラッグし、**GroupBox** コントロールにドロップします。
6. [プロパティ] ウィンドウで、RadioButton3 と RadioButton4 の Text プロパティを、それぞれ `Thin crust` と `Thick crust` に変更します。
7. フォームの [Order Pizza] をダブルクリックして、**Button1_Click** イベント ハンドラをコード エディタで開きます。
8. **Button1_Click** イベント ハンドラに次のコードを追加します。

   VB

```
If RadioButton3.Checked = True Then
    MsgBox("You chose a thin crust")
Else
    MsgBox("You chose a thick crust")
End If
```

9. F5 キーを押してプログラムを実行します。オプション ボタンのうち 1 つをオンにし、[Order Pizza] をクリックします。選択した項目を示すメッセージ ボックスが表示されます。このとき、選択したソースも保存されています。

次の手順

このレッスンでは、コンテナ コントロールを使用して、**RadioButton** コントロールをグループ化する方法を説明しました。次のレッスンでは、画像を表示する方法を説明します。

次のレッスン : 「ピクチャ描画 : イメージを表示する」

参照

処理手順

ユーザー選択の取得 : チェック ボックスとオプション ボタンを使用する

方法 : Windows フォーム GroupBox コントロールを使用してコントロールをグループ化する

関連項目

Panel コントロールの概要 (Windows フォーム)

ピクチャ描画：イメージを表示する

このレッスンでは、PictureBox コントロールを使用してイメージを表示する方法、およびピクチャをフォームの背景イメージとして表示する方法について説明します。

1 枚の絵は 1000 の言葉に値する (百聞は一見にしかず) と言われるように、多くのプログラムがピクチャを使用して情報を伝達しています。Visual Basic でピクチャを表示するには、複数の方法がありますが、最も一般的な方法は、**PictureBox** コントロールを使用することです。

PictureBox コントロールは、ピクチャのコンテナとして機能します。Image プロパティを設定して、表示するピクチャを選択します。**Image** プロパティは、[プロパティ] ウィンドウで設定するか、表示するピクチャをプログラムに通知するコードを記述します。

PictureBox コントロールのその他の便利なプロパティとしては、**PictureBox** がピクチャに合わせて伸縮するかどうかを決定する AutoSize プロパティ、**PictureBox** 内でイメージを拡大、中央揃え、またはズームするために使用される SizeMode プロパティがあります。

PictureBox コントロールにピクチャを追加する前に、通常はピクチャ ファイルをリソースとしてプロジェクトに追加します。プロジェクトにリソースを追加すると、必要に応じて何回でもリソースを再利用できます。たとえば、同じピクチャを複数の場所に表示できます。

やってみよう

ピクチャをリソースとして追加するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Pictures`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. [ソリューション エクスプローラ] ウィンドウで、[My Project] ノードを右クリックしてプロジェクト デザイナを開きます。
5. プロジェクト デザイナで、[リソース] タブをクリックします。
6. [リソースの追加] をクリックして、ドロップダウン リストの [既存のファイルの追加] をクリックします。

   [既存のファイルをリソースに追加] ダイアログ ボックスが開きます。ピクチャ ファイルが表示されない場合は、ピクチャを含むフォルダを参照します。
7. イメージ ファイル (ファイル拡張子が .bmp、.gif、または .jpg のファイル) を選択して、[開く] をクリックします。この例では、小さいピクチャを選択することをお勧めします。

   ピクチャがプロジェクトに追加され、[リソース マネージャ] ウィンドウに表示されます。
8. 2 番目のピクチャをプロジェクトに追加するには、前の 2 つの手順を繰り返します。
9. [ファイル] メニューの [閉じる] をクリックします。変更を保存するかどうかを確認するメッセージが表示された場合は [はい] をクリックします。

PictureBox コントロールを使用してピクチャを表示するには

1. ソリューション エクスプローラで Form1.vb を選択して、[表示] メニューの [デザイナ] をクリックします。
2. ツールボックスから、フォームに **PictureBox** コントロールをドラッグします。
3. [プロパティ] ウィンドウで **Image** プロパティの隣にある 省略記号 (...) ボタンをクリックして、[リソースの選択] ダイアログ ボックスを開きます。
4. [入力] ボックスで追加したピクチャを選択して、[OK] をクリックします。
5. **SizeMode** プロパティを選択して、**AutoSize** に設定します。

   ピクチャに合わせて **PictureBox** コントロールのサイズが自動的に変更されることに注意してください。
6. フォームの **PictureBox** コントロールをダブルクリックして、**PictureBox1_Click** イベント ハンドラをコード エディタで開きます。
7. **PictureBox1_Click** イベント ハンドラに次のコードを追加します。

メモ：
"`MyPictureName2`" を、前に追加した 2 番目のピクチャの実際の名前に置き換える必要があります。

VB

```
PictureBox1.Image = My.Resources.MyPictureName2
```

8. F5 キーを押してプログラムを実行します。フォームが表示されたら、ピクチャをクリックして、2 番目のピクチャを表示させます。

フォームでの背景イメージの表示

PictureBox コントロールでピクチャを表示する以外に、ピクチャをフォームの背景として表示することもできます。フォームの BackgroundImage プロパティは、Windows デスクトップの壁紙と同じように、フォームのその他のコントロールの背後に表示されるピクチャを表示するために使用されます。

Windows で、壁紙を中央に表示するか、並べて表示するか、拡大して表示するかを選択できるのと同じように、BackgroundImageLayout プロパティを使用すると、フォームに対して同じことを実行できます。

ヒント

Panel、GroupBox などのその他のコントロールの多く、および Button コントロールにも **BackgroundImage** プロパティがあります。試して確認してみてください。

やってみよう

フォームに背景イメージを表示するには

1. ソリューション エクスプローラで Form1.vb を選択して、[表示] メニューの [デザイナ] をクリックします。
2. **PictureBox** の外部をクリックして、フォームを選択します。
3. [プロパティ] ウィンドウで **BackgroundImage** プロパティの隣にある 省略記号 (...) ボタンをクリックして、[リソースの選択] ダイアログ ボックスを開きます。
4. [入力] ボックスで前に追加したピクチャを選択して、[OK] をクリックします。

   フォームの **PictureBox** の背後にピクチャが表示されます。既定では並べて表示されています。

   メモ：

   PictureBox コントロール内のピクチャが大きすぎる場合、背景ピクチャが見えないことがあります。この場合は、**PictureBox** コントロールを選択してフォームの下部にドラッグし、背景が見えるようにします。

5. **BackgroundImageLayout** プロパティを選択して、Stretch に設定します。

   フォーム全体を埋めるようにピクチャが拡大します。

6. フォームをダブルクリックしてコード エディタを開きます。
7. 左側のドロップダウン ボックスで Form1 Events が選択されていることを確認してから、右側のドロップダウン ボックスの [クリック] をクリックします。
8. **Form1_Click** イベント ハンドラに次のコードを追加します。

   VB

```
If Me.BackgroundImageLayout = ImageLayout.Stretch Then
  Me.BackgroundImageLayout = ImageLayout.Center
Else
  Me.BackgroundImageLayout = ImageLayout.Stretch
End If
```

9. F5 キーを押してプログラムを実行します。フォームが表示されたら、クリックしてレイアウトを変更します。

次の手順

このレッスンでは、**PictureBox** コントロールを使用してイメージを表示する方法、およびフォームの **BackgroundImage** プロパティを使用する

方法について説明しました。次のレッスンでは、ユーザーに選択肢を提示するメニューを作成する方法について説明します。

次のレッスン : 「ユーザーへの選択肢の提示 : デザイン時にメニューを作成する」

参照

処理手順

ユーザー選択の取得 : チェック ボックスとオプション ボタンを使用する

方法 : Windows フォーム パネルの背景を設定する

関連項目

PictureBox コントロールの概要 (Windows フォーム)

ユーザーへの選択肢の提示：デザイン時にメニューを作成する

このレッスンでは、メニューの作成方法について説明するほか、メニュー項目の選択時に実行するコードの記述方法を示します。

メニューは、ユーザーがプログラムに関する選択を行う場合によく使用される、簡単な方法です。メニューの一般的な用途には、プログラムのオプションの提示、切り取りや貼り付けなど一般的なタスクのショートカットの追加、ファイルの読み込みおよび保存などがあります。

Visual Basic では、メニューを簡単に実装できます。MenuStrip コントロールを使用すると、メニューをグラフィカルに作成できます。**MenuStrip** コントロールをフォーム上にドラッグすると、フォームの上部にボックスが表示されます。このボックスの中には、"ここへ入力" という言葉が表示されています。このボックスをクリックして中にテキストを入力すると、メニュー タイトルを作成できます。

1 つのメニュー項目のタイトルを設定すると、その下側と右側に追加のメニュー項目を作成できます。これにより、メニューに追加の項目やサブ項目を必要な数だけ設定して、メニューを拡張できます。メニューの外観が完成したら、各項目の Click イベントを処理するためのイベント ハンドラを作成できます。

やってみよう

メニューを追加するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Menus`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、フォームに **MenuStrip** コントロールをドラッグします。

   ドロップする場所に関係なく、**MenuStrip** コントロールは、フォームの最上端に追加されます。

   また、フォームの下にある灰色の領域に、[MenuStrip1] アイコンが追加されます。この領域のことを、コンポーネント トレイと呼びます。**MenuStrip** コントロールは、別の場所をクリックすると表示されなくなりますが、[MenuStrip1] アイコンをクリックすると再度表示できます。
5. フォームで、**MenuStrip** コントロールをクリックして「`File`」と入力し、Enter キーを押します。

   追加のメニューを入力するための新しいボックスが、最初のメニュー項目の下側と右側に表示されます。これらのボックスは、追加のメニュー項目のための領域です。いずれかのボックスにメニュー項目を追加し、メニューが完成するまでこれを繰り返します。
6. 最初のボックスの下側にあるボックスに「`Exit`」と入力し、Enter キーを押します。
7. [終了] メニューをダブルクリックしてコード エディタを開きます。
8. **ExitToolStripMenuItem_Click** イベント ハンドラに次のコードを追加します。

   VB

```
Application.Exit()
```

9. F5 キーを押してプログラムを実行します。マウスを使用して [ファイル] メニューをクリックし、[終了] をクリックします。アプリケーションが終了します。

このレッスンでは、**MenuStrip** コントロールを使用してメニューを設計する方法について説明しました。この時点で、タイマに関する次のレッスンに進むか、「詳細情報：メニューの詳細」でメニューのより高度な使用方法を学習してからタイマに関するレッスンに進むかを選択できます。

次のレッスン：「スケジュールの設定：タイマを使用して定期的に処理を実行する」

参照

処理手順

ピクチャ描画：イメージを表示する

詳細情報：メニューの詳細

このレッスンでは、実行時にメニューを有効または無効にする方法と、ショートカット メニューを作成する方法を説明します。

前のレッスンでは、MenuStrip コントロールを使用して、ユーザーがプログラムに関する選択を実行できるようにするメニューの作成方法を説明しました。しかし、選択項目を特定の状況でしか使用できない場合があります。たとえば、[コピー] メニュー コマンドは、コピーできる対象がある場合にのみ使用できます。

ほとんどのプログラムでは、メニュー コマンドを使用できない場合、それを非表示にするのではなく無効にします。メニュー項目が無効になると、メニュー テキストの色が灰色に変わり、そのメニュー項目をクリックしても何も起こらなくなります。**MenuStrip** コントロールを使用するときに、MenuItem の Enabled プロパティを使用すると、メニュー項目を有効または無効にできます。

やってみよう

メニュー項目を無効または有効にするには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「Menus2」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、フォームに **MenuStrip** コントロールと TextBox コントロールをドラッグします。
5. フォームで、**MenuStrip** コントロールをクリックして「Edit」と入力し、Enter キーを押します。
6. 最初のボックスの下側にあるボックスに「Copy」と入力し、Enter キーを押します。
7. [プロパティ] ウィンドウで、copyToolStripMenuItem の **Enabled** プロパティを **False** に設定します。
8. **TextBox** コントロールをダブルクリックしてコード エディタを開きます。
9. **TextBox1_TextChanged** イベント ハンドラに次のコードを追加します。

   VB

```
If Textbox1.Text <> "" Then
  CopyToolStripMenuItem.Enabled = True
Else
  CopyToolStripMenuItem.Enabled = False
End If
```

10. F5 キーを押してプログラムを実行します。[編集] メニューをクリックします。[コピー] メニュー項目が無効になっています。**TextBox** コントロールにテキストを入力し、[編集] メニューをクリックします。今度は [コピー] メニュー項目が有効になっています。

ショートカット メニューの作成

多くのプログラムでは、ショートカット メニュー (コンテキスト メニューとも呼ばれる) を使用して、一般的に使用されるコマンドに簡単にアクセスできるようになっています。コンテキスト メニューは、実行時にフォームまたはコントロールを右クリックすると表示されます。Visual Basic で独自のコンテキスト メニューを作成するには、ContextMenuStrip コントロールを使用します。

MenuStrip コントロールと同様、**ContextMenuStrip** コントロールをフォーム上にドラッグすると、内側に "ここへ入力" という単語が表示されたボックスの形で **ContextMenuStrip** コントロールがフォームの上部に表示され、コンポーネント トレイにアイコンが追加されます。ただし、**MenuStrip** と異なり、追加のメニュー項目を作成できるのは、最初のメニュー項目の下側のみです。これにより、縦方向のメニューが作成されます。

さらに、**ContextMenuStrip** は、表示するフォームまたはコントロールと関連付ける必要があります。これには、フォームまたはコントロールの **ContextMenuStrip** プロパティを、**ContextMenuStrip** コントロールの名前に設定します。1 つの **ContextMenuStrip** に関連付けることができるコントロールの数に、制限はありません。

やってみよう

コンテキスト メニューを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「ContextMenus」と入力し、[OK] をクリックします。

新しい Windows フォーム プロジェクトが開きます。

4. ツールボックスから、フォームに **ContextMenuStrip** コントロールをドラッグします。

5. [プロパティ] ウィンドウで、フォームの **ContextMenuStrip** プロパティをクリックし、ボックスの一覧の [ContextMenuStrip1] をクリックします。

6. フォームで、**ContextMenuStrip** コントロールをクリックして「Option1」と入力し、Enter キーを押します。

7. 最初のボックスの下側にあるボックスに「Option2」と入力し、Enter キーを押します。

8. [option1] メニュー項目をダブルクリックしてコード エディタを開きます。

9. **Option1ToolStripMenuItem_Click** イベント ハンドラに次のコードを追加します。

```
VB
MsgBox("You chose Option 1")
```

10. コード エディタで、左側のドロップダウン ボックスの [Option2ToolStripMenuItem] をクリックし、右側のドロップダウン ボックスの [クリック] をクリックします。

    Option2ToolStripMenuItem _Click という新しいイベント ハンドラがコード エディタに表示されます。

11. Option2ToolStripMenuItem _Click イベント ハンドラに次のコードを入力します。

```
VB
MsgBox("You chose Option 2")
```

12. F5 キーを押してプログラムを実行します。フォームを右クリックしてコンテキスト メニューのいずれかの項目をクリックします。クリックした選択項目を通知するメッセージ ボックスが表示されます。

次の手順

このレッスンでは、メニューを有効および無効にする方法と、コンテキスト メニューを作成する方法を説明しました。次のトピックでは、異なる種類のコントロール (Timer) を使用して処理を実行する方法を説明します。

次のレッスン : 「スケジュールの設定 : タイマを使用して定期的に処理を実行する」

参照

処理手順

ユーザーへの選択肢の提示 : デザイン時にメニューを作成する
方法 : ContextMenuStrip をコントロールに関連付ける

関連項目

MenuStrip コントロールの概要 (Windows フォーム)

スケジュールの設定：タイマを使用して定期的に処理を実行する

このレッスンでは、Timer コンポーネントを使用して、ユーザー入力を要求することなく処理を実行する方法を説明します。

プログラムで処理を繰り返し実行すると、ファイルを数分おきに保存したり、ユーザー インターフェイスを更新したりする場合などに、役に立つことがあります。**Timer** コンポーネントを使用すると、ユーザーからの入力がなくても、あらかじめ設定した処理を定期的に実行できます。

Timer コンポーネントは、実行時のビジュアル表示を持たないという点で、これまで使用したコントロールとは異なります。ビジュアル表示を持たないコントロールは、コンポーネントと呼ばれます。ユーザーには **Timer** コンポーネントと直接対話する手段がないため、このコンポーネントはプログラムの実行時にバックグラウンドで実行されます。

Timer コンポーネントには、最もよく使用されるメンバとして、2 つのプロパティと 1 つのイベントがあります。Enabled プロパティは、**Timer** コンポーネントが機能するかどうかを決定します。**Enabled** が **True** に設定されている場合、**Timer** はアクティブです。**Enabled** が **False** に設定されている場合、**Timer** はアクティブではありません。

Interval プロパティは、**Timer** のタイマ刻み間隔であるミリ秒数を決定します。たとえば、**Interval** プロパティが `1000` に設定されている場合、**Timer** コンポーネントは Tick イベントを 1000 ミリ秒、つまり 1 秒ごとに発生させます。

Tick イベントは、**Timer** コンポーネントによって定期的に発生します。この間隔は、**Interval** プロパティの値によって決定されます。**Timer.Tick** イベント ハンドラにコードを追加すると、**Tick** イベントが発生するたびにそのコードが実行されます。

Enabled プロパティと **Interval** プロパティを設定して、**Tick** イベント ハンドラにコードを追加することによって、ユーザーの処理を待たずに定期的に実行されるコードを作成できます。

やってみよう

Timer コンポーネントを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Timer`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、フォームに Label コントロールと **Timer** コンポーネントをドラッグします。

   Timer コンポーネントは、フォーム自体には表示されず、その下のコンポーネント トレイに表示されます。これは、Timer がビジュアル表示を持たないためです。
5. **Timer** コンポーネントを選択し、[プロパティ] ウィンドウで、**Enabled** プロパティを **True** に、**Interval** プロパティを `1000` に、それぞれ設定します。
6. **Timer** コンポーネントをダブルクリックしてコード エディタを表示します。
7. **Timer1_Tick** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Label1.Text = My.Computer.Clock.LocalTime.ToLongTimeString
   ```
8. F5 キーを押して、アプリケーションを実行します。

   ラベル内のテキストが、正確に 1 秒ごとに更新されます。

次の手順

このレッスンでは、**Timer** コンポーネントを使用して、定期的にコードを実行する方法を説明しました。**Timer** コンポーネントを使用すると、コードにスケジュールを設定して、任意の間隔で定期的に実行できます。これで、ガイド ツアーのこのセクションのレッスンは終了です。

次のセクションでは、Visual Basic のツールを使用して、プログラムのエラー (一般にバグと呼ばれる) を追跡および修復する方法を説明します。

次のレッスン：「問題の確認：デバッグを行ってエラーを検出し、修正する」

参照
処理手順

ユーザーへの選択肢の提示 : デザイン時にメニューを作成する

関連項目

Timer コンポーネントの概要 (Windows フォーム)

問題の確認：デバッグを行ってエラーを検出し、修正する

コンピュータ プログラムを記述していると、必ずといってよいほどエラーが発生します。タイポグラフィのエラーを犯す場合もあれば、プログラムが期待どおりに動作しない場合や、まったく動作しない場合もあります。プログラムでエラーが発生した場合は、それを発見して修復する必要があります。このエラーの発見および修復プロセスのことを、デバッグと呼びます。

次に示すレッスンでは、Visual Basic プログラムをデバッグするためのいくつかのテクニックについて説明します。

このセクションの内容

エラーの検出 : Visual Basic のデバッグの概要

バグの理解 : 3 種類のプログラミング エラー

ミスのチェック : コンパイラ エラーを検出して修正する

ランタイム エラーの検索と除去

想定外の現象 : 論理エラーを突き止める

プログラムの注釈 : コメントを使用する

関連するセクション

詳細情報 :イミディエイト ウィンドウでコードをテストする

追加レッスン : 他のエラーを発見する

プログラムの外観の作成 : Windows フォームの概要

レコードの管理 : プログラムでデータを使用する

Visual Basic ガイド ツアー

エラーの検出：Visual Basic のデバッグの概要

このレッスンでは、デバッグを通じたプログラムのエラー修復について説明します。

プログラムの設計やコードの記述をどれほど慎重に行っても、エラーは必ず発生します。エラーが原因で、プログラムが起動すらしない場合や、プログラムが実行を中止したりクラッシュしたりする場合、さらにはプログラムが動作しても期待した結果が得られない場合があります。

もちろん、エラーが発生したら、それを発見して修復する必要があります。プログラム内のエラーは一般にバグと呼ばれ、バグを発見して修復するプロセスをデバッグと呼びます。

デバッグのプロセスは、反復処理、つまり何度も繰り返すプロセスです。通常、コードを記述したら、エラーが発生するまでプログラムを実行し、バグを発見して修復してから、再度プログラムを実行します。

ほとんどの場合、エラーを修復するためにプログラムを停止させる必要はありません。エラーが発生した場所でコードを修復して、中断したところからプログラムの実行を再開できます。このプロセスをエディット コンティニュと呼びます。

デバッグは Visual Basic IDE (統合開発環境) 内で行われます。Visual Basic IDE には、バグの発見に役立ついくつかの特殊なコマンドとウィンドウが用意されています。これらのコマンドやウィンドウについては、この後のレッスンで詳しく説明します。

やってみよう

メモ：

ここでは、例外に関連する例を示します。例外とは、エラーが発生したことをプログラムが認識したときに作成 (およびスロー) されるオブジェクトです。発生したエラーの種類に応じて、異なる型の例外が作成されます。既定のユーザー設定では、Visual Basic プログラムの実行中に例外が発生した場合、エラーの説明を示すダイアログ ボックスが表示され、エラー修復の手助けとなります。

エディット コンティニュを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Edit`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開きます。
5. **Form_Load** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Dim number As Integer = 1
   Dim numbers As String = ""
   MsgBox(numbers + 1)
   ```
6. F5 キーを押してプログラムを実行します。プログラムは中断され、"InvalidCastException はハンドルされませんでした。" というメッセージを示す例外のダイアログ ボックスが表示されます。

   この例外は、コードにタイポグラフィのエラーがあるために発生しました。`number` (**Integer** 変数) を使用するところを、`numbers` (**String** 変数) という間違った変数を使用したためです。

   この時点で、プログラムはまだ実行中です。デバッグの中断モードにあるだけです。エディット コンティニュを使用すると、プログラムを停止することなく (および再実行してチェックすることなく)、エラーを修復できます。
7. コード エディタで、`numbers + 1` を `number + 1` に変更します。
8. F5 キーを押して続行します。数値 2 を含むメッセージ ボックスが表示されます。

次の手順

このレッスンでは、デバッグについて説明すると共に、バグを修復してプログラムの実行を再開する方法を説明しました。次のレッスンでは、さまざまな種類のエラーについて説明します。

次のレッスン：「バグの理解：3 種類のプログラミング エラー」

参照

その他の技術情報

問題の確認 : デバッグを行ってエラーを検出し、修正する
プログラムの外観の作成 : Windows フォームの概要
Visual Basic ガイド ツアー

バグの理解 : 3 種類のプログラミング エラー

このレッスンでは、プログラムを記述しているときに発生する可能性のあるさまざまな種類のエラーについて説明します。

多くの経験を積んだプログラマですら間違いを犯します。このため、アプリケーションをデバッグして、そうした間違いを発見する方法を知ることは、プログラミングの重要な要素です。しかし、デバッグのプロセスについて学習する前に、発見して修復する必要のあるバグの種類を知っておくと役に立ちます。

プログラミング エラーは、コンパイル エラー、ランタイム エラー、および論理エラーの 3 種類に分類されます。これらの各エラーをデバッグするためのテクニックについては、次の 3 つのレッスンで説明します。

コンパイル エラー

コンパイル エラーは、コンパイラ エラーとも呼ばれ、プログラムの実行を妨げるエラーです。F5 キーを押してプログラムを実行すると、Visual Basic はコードをコンパイルして、コンピュータが理解できるバイナリ言語にコンパイルします。Visual Basic コンパイラは、理解できないコードを検出すると、コンパイラ エラーを発行します。

ほとんどのコンパイラ エラーは、コード入力時の間違いが原因で発生します。たとえば、キーワードのスペルを間違えたり、必要な区切り記号を挿入しなかったり、先に **If** ステートメントを使用せずに **End If** ステートメントを使用したりした場合などです。

しかし、Visual Basic のコード エディタは、プログラムを実行しようとする前にこうした間違いを識別するよう設計されています。コンパイル エラーの発見および修復方法については、次のレッスン (「ミスのチェック : コンパイラ エラーを検出して修正する」) で説明します。

ランタイム エラー

ランタイム エラーは、プログラムの実行時に発生するエラーです。このエラーは、通常、実行できない操作をプログラムが試行したときに発生します。

その 1 つの例として、ゼロによる除算が挙げられます。次のステートメントがあるとします。

```
Speed = Miles / Hours
```

変数 `Hours` の値が 0 の場合、除算は失敗し、ランタイム エラーが発生します。このエラーはプログラムの実行時でないと検出できず、また `Hours` に有効値が格納されている場合には、エラーはまったく発生しません。

ランタイム エラーが実際に発生した場合は、Visual Basic のデバッグ ツールを使用することにより原因を特定できます。ランタイム エラーの発見および修復方法については、「ランタイム エラーの検索と除去」のレッスンで説明します。

論理エラー

論理エラーは、プログラムが目的の動作を行うことを妨げるエラーです。コードのコンパイルと実行でエラーが発生しなくても、操作の結果が予想と異なる場合があります。

たとえば、初期値として空白の文字列が設定されている `FirstName` という変数があるとします。プログラムの後半で、`FirstName` を `LastName` という別の変数と連結して、フル ネームを表示します。しかし、`FirstName` に値を代入し忘れた場合、意図していたフル ネームではなく、姓だけが表示されます。

論理エラーは、発見および修復が最も難しいエラーですが、Visual Basic には、このジョブをより簡単にするデバッグ ツールも用意されています。論理エラーの発見および修復方法については、「想定外の現象 : 論理エラーを突き止める」で説明します。

次の手順

このレッスンでは、プログラミング エラーの 3 つのカテゴリについて説明しました。次のレッスンでは、コンパイラ エラーのデバッグについて説明します。

次のレッスン : 「ミスのチェック : コンパイラ エラーを検出して修正する」

参照

処理手順

エラーの検出 : Visual Basic のデバッグの概要

その他の技術情報

問題の確認 : デバッグを行ってエラーを検出し、修正する

ミスのチェック：コンパイラ エラーを検出して修正する

このレッスンでは、コンパイラ エラーを検出して修正する方法について説明します。

前のレッスンで説明したように、コンパイラ エラーは、通常は入力時の間違いにより、Visual Basic コンパイラが認識できないコードを検出したときに発生します。コンパイラ エラーがあるとプログラムを実行できないため、プログラムを実行する前に、エラーを検出して修正、つまりデバッグする必要があります。

コンパイラ エラーの検出と修正

エラーを修正するまでプログラムを実行できないため、コンパイラ エラーの検出は簡単です。F5 キーを押すと、コンパイラ エラーがあった場合、"ビルド エラーが発生しました。続けますか ?" というダイアログ ボックスが表示されます。[はい] を選択すると、エラーのない最新バージョンのプログラムが実行されます。[いいえ] を選択すると、プログラムは停止し、[エラー一覧] ウィンドウが表示されます。

[エラー一覧] ウィンドウには、エラーの説明とコード内のエラーの場所など、コンパイラ エラーに関する情報が表示されます。[エラー一覧] 内のエラーをダブルクリックすると、エラーの原因となったコード行がコード エディタで強調表示されます。F1 キーを押してヘルプを表示し、エラーに関する詳細と修正方法を確認することもできます。

Visual Basic のコード エディタを使用すると、プログラムを実行しようとする前でも、コンパイラ エラーの検出と修正に役立ちます。Visual Basic は、IntelliSense と呼ばれる機能を使用して入力時にコードを検査します。コンパイラ エラーの原因となるコードが見つかった場合、そのコードの下に青色の波線が引かれます。マウスをその行の上に置くと、エラーを説明するメッセージが表示されます。[エラー一覧] ウィンドウが表示される場合は、エラー メッセージもそのウィンドウに表示されます。

やってみよう

コンパイラ エラーを検出して修正するには

1. [ファイル] メニューの [新しいプロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`CompilerErrors`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開きます。
5. **Form_Load** イベント ハンドラに次のコードを追加します。

   VB

   ```
   End If
   ```

6. Enter キーを押します。`End If` の下に青色の波線が表示されます。

   マウスをその行の上に置くと、"'End If' の前には、対応する 'If' を指定しなければなりません。" というメッセージが表示されます。
7. コードを次のように変更します。

   VB

   ```
   If 1 < 2 Then
   End If
   ```

   青色の波線が消えたことを確認します。
8. If...Then ステートメントの後に、次の新しいコード行を追加します。

   VB

   ```
   MgBox("Hello")
   ```

9. F5 キーを押してプログラムを実行します。ダイアログ ボックスに "ビルド エラーが発生しました。続行して、最後に成功したビルドを実行し

ますか?" というメッセージが表示されます。

10. [いいえ] をクリックします。[エラー一覧] ウィンドウに "名前 'MgBox' は宣言されていません。" というメッセージが表示されます。
11. [エラー一覧] でエラー メッセージをダブルクリックし、コードを `MsgBox("Hello")` に変更します。
12. 再度 F5 キーを押します。今回はプログラムが実行され、メッセージ ボックスが表示されます。

次の手順

このレッスンでは、コンパイラ エラーを検出して修正する方法について説明しました。次のレッスンでは、別の種類のエラーとして、ランタイム エラーを修正する方法について説明します。次のレッスン :「ランタイム エラーの検索と除去」

参照

処理手順

バグの理解 : 3 種類のプログラミング エラー

エラーの検出 : Visual Basic のデバッグの概要

ランタイム エラーの検索と除去

このレッスンでは、プログラムをデバッグしてランタイム エラーを修正する方法について説明します。

以前に説明したように、ランタイム エラーは、プログラムが完了できない操作を実行しようとした場合に発生します。ランタイム エラーが発生すると、プログラムが停止してエラー メッセージが表示されます。プログラムを続行するには、エラーをデバッグして修正する必要があります。

ランタイム エラーの検索と修正

ほとんどのランタイム エラーは、変数を使用する前に変数に値を割り当てていないなどの、コード内の間違いによって発生します。プログラムを実行して間違いが検出されると、プログラムは停止し、コード エディタ ウィンドウに [例外処理アシスタント] ダイアログ ボックスが表示されます。このとき、プログラムは中断モードになります。これは、デバッグが実行されるモードです。

[例外処理アシスタント] ダイアログ ボックスには、エラーの説明以外に、エラーの原因を解明するためのトラブルシューティングのヒントが表示されます。トラブルシューティングのヒントをクリックすると、詳細なヘルプ トピックが表示されます。

プログラムを続行するには、エラーを修正する必要があります。それにはコードを調べてエラーの原因を探すことが必要です。たとえば、変数に間違った値が含まれているためにエラーが発生したと思われる場合は、中断モードに入ったままで、IntelliSense を使用して変数の値を表示できます。コード エディタで変数の上にマウスを置くと、ツールヒントに変数の値が表示されます。値が予期したものと異なる場合は、それより前にあるコードでその値が設定されている場所を探し、そのコードを修正して続行します。

やってみよう

変数の値を調べるには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`RunTimeErrors`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開きます。
5. **Form_Load** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Dim miles As Integer = 0
   Dim hours As Integer = 0
   Dim speed As Integer = 0
   ```

   VB

   ```
   miles = 55
   speed = miles / hours
   MsgBox(CStr(speed) & " miles per hour")
   ```
6. F5 キーを押してプログラムを実行します。[例外処理アシスタント] ダイアログ ボックスが表示され、メッセージ "OverflowException はハンドルされませんでした" が表示されます。

   ダイアログ ボックスからコード ファイルへ引かれた点線から、コードのどの行にエラーの原因があるかがわかります。

   [例外処理アシスタント] に最初に表示されるトラブルシューティングのヒントは、ゼロで割っていないことを確認する必要があることを示しています。
7. 変数 `miles` の上にマウスを移動して、数秒間そのままにします。"miles 55" というツールヒントが表示されます。
8. 次に、変数 `hours` の上にマウスを移動すると、ツールヒントが "hours 0" になります。

   ゼロで割ることはできないにもかかわらず、`hours` の値がゼロになっています。これがエラーの原因です。`hours` の値が更新されていません。

9. `miles = 55` の行の上に次のコード行を追加します。

   VB

   ```
   hours = 2
   ```

10. コードの左側のマージンにある黄色の矢印をクリックして、`hours = 2` の行までドラッグします。

    これにより、エラーを含む行ではなくその行からプログラムを続行できます。エラーの修正を認識させるためには、このように新しく追加したコード行を実行することが必要です。

11. F5 キーを押してプログラムを続行します。"28 miles per hour" を表示するダイアログ ボックスが表示されます。

次の手順

このレッスンでは、ランタイム エラーを検出して修正する方法について説明しました。次のレッスンでは、3 種類目のプログラミング エラーとして、論理エラーについて説明します。高度なデバッグ手法については、「詳細情報 :イミディエイト ウィンドウでコードをテストする」を参照してから、論理エラーのレッスンに進んでください。

次のレッスン :「想定外の現象 : 論理エラーを突き止める」

参照

処理手順

ミスのチェック : コンパイラ エラーを検出して修正する
バグの理解 : 3 種類のプログラミング エラー
エラーの検出 : Visual Basic のデバッグの概要

詳細情報 :イミディエイト ウィンドウでコードをテストする

このレッスンでは、イミディエイト ウィンドウを使用してコードを評価および実行する方法について説明します。

前のレッスンでは、[例外処理アシスタント] を使用してランタイム エラーを修正する方法について説明しました。しかし、エラーの修正方法が明確でない場合、実際のコードを変更せずに、修正をテストすると便利なこともあります。イミディエイト ウィンドウと呼ばれる特別なデバッグ ウィンドウを使用すると、このようなテストを実行できます。

イミディエイト ウィンドウ

プログラムが中断モードの場合、[イミディエイト] ウィンドウを使用してコードの一部を実行し、変数や式を評価できます。たとえば、空の変数が原因でランタイム エラーが発生する場合、変数の値をチェックできます。また、[イミディエイト] ウィンドウを使用してその変数に値を代入し、その場合にプログラムの残りの部分がどのように動作するかをチェックすることもできます。

ヒント
プログラムをデバッグ モードで実行している場合、[デバッグ] メニューの [中断] をクリックすると、いつでもプログラムを中断モードにできます。

コード エディタの場合と同じように [イミディエイト] ウィンドウにコードを入力し、Enter キーを押すと、コードを実行できます。変数または式を評価するには、評価する変数または式を疑問符の後に入力して、Enter キーを押します。結果は次の行に表示されます。

やってみよう

イミディエイト ウィンドウでコードをテストするには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Immediate`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、2 つの TextBox コントロールと Button コントロールをフォームにドラッグします。
5. ボタンをダブルクリックしてコード エディタを開きます。
6. **Button_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Dim miles As Integer = 0
   Dim hours As Integer = 0
   Dim speed As Integer = 0
   ```

   VB

   ```
   miles = CInt(Textbox1.Text)
   hours = CInt(Textbox2.Text)
   speed = miles / hours
   MsgBox(CStr(speed) & " miles per hour")
   ```
7. F5 キーを押してプログラムを実行します。最初のテキスト ボックスに「`100`」、2 番目のテキスト ボックスに「`0`」と入力します。
8. Button1 をクリックします。プログラムが停止し、[例外処理アシスタント] ダイアログ ボックスに "OverflowException はハンドルされませんでした" というメッセージが表示されます。
9. IDE の下部にある [イミディエイト] ウィンドウに「`?miles`」と入力して、Enter キーを押します。

   値 `100` が次の行に表示されます。

ヒント

[イミディエイト] ウィンドウは、[デバッグ] メニューの [ウィンドウ] をクリックし、[イミディエイト] をクリックすると、いつでも開けます。

10. 「`?hours`」と入力して、Enter キーを押します。

    値 0 が次の行に表示されます。

11. 「`hours = 4`」と入力し、Enter キーを押します。次に、「`?hours`」と入力し、Enter キーを押します。

    `hours` の値が `4` (前の行で入力した値) になりました。プログラムのコードを変更せずに、[イミディエイト] ウィンドウで `hours` の値を変更できました。

12. F5 キーを押して続行します。結果を示すメッセージ ボックスが表示されます。

ヒント

このランタイム エラーが発生しないようにするには、**Try** ブロックで有効な数値を確認するエラー ハンドラを追加して、**Catch** ブロックでユーザーにメッセージを表示します。エラー ハンドラの詳細については、「問題が発生した場合の対処 : エラー処理」を参照してください。

次の手順

このレッスンでは、イミディエイト ウィンドウを使用して値を確認し、コードを実行する方法について説明しました。次のレッスンでは、論理エラーを検出して修正する方法について説明します。

次のレッスン : 「想定外の現象 : 論理エラーを突き止める」

参照

処理手順

ランタイム エラーの検索と除去

問題が発生した場合の対処 : エラー処理

想定外の現象：論理エラーを突き止める

このレッスンでは、プログラム内の論理エラーを検出する方法について説明します。

前のレッスンでは、コンパイラ エラーおよびランタイム エラーを検出して修正する方法について説明しました。3 種類目のプログラミング エラーは、論理エラーで、検出が最も困難なエラーです。論理エラーの場合、警告は一切出ません。プログラムは動作しますが、結果は間違っています。それからコードを調べて、問題が発生した理由を調べる必要があります。

コードを調べる際には、Visual Basic のデバッグ ツールが役に立ちます。ブレークポイントの設定とコードのステップ実行の 2 つのデバッグ手法により、実行中にコードを 1 行ずつ調べて、エラーを検出できます。

実行できるコード行いずれに対しても、コード エディタでブレークポイントを設定できます。プログラムを実行してブレークポイントのコード行に達すると、実行が強制的に停止され、中断モードに入ります。そして、その時点でのプログラムの状態について、必要な情報を取得できます。変数の値を確認したり、[イミディエイト] ウィンドウで式をテストしたり、エディット コンティニュを使用してコードを変更したりできます。

中断モードに入ると、コードをステップ実行、つまり 1 行ずつ実行して、コードがどのように動作するかを確認できます。F8 キーを押すと、現在のコード行が実行され、次の行で停止します。その時点で、変数の値を調べて、次の行に移動する間に値がどのように変化するかを確認できます。

現在のコード行が、コード内の他の場所の関数または **Sub** プロシージャを呼び出している場合、F8 キーを押すと、実行はそのプロシージャにステップ インします。プロシージャのステップ実行が終わると、そのプロシージャを呼び出した行の次の行に戻ります。プロシージャをステップ実行しない場合は、Shift キーを押しながら F8 キーを押して、プロシージャをステップ オーバーします。

やってみよう

論理エラーを検証するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`LogicErrors`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ツールボックスから、フォームに 2 つの TextBox コントロールと 1 つの Button コントロールをドラッグします。
5. Button1 をダブルクリックしてコード エディタを開きます。
6. **Button1_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   Dim minutes As Integer = CInt(Textbox1.Text)
   Dim miles As Double = CDbl(Textbox2.Text)
   Dim hours As Double = 0
   hours = minutes / 60
   MsgBox("Average speed " & GetMPH(hours, miles))
   ```

7. `End Sub` 行の下に次の関数を追加します。

   VB

   ```
   Function GetMPH(ByVal miles As Double, ByVal hours As Double) _
   As String
       GetMPH = CStr(miles / hours)
   End Function
   ```

8. F5 キーを押してプログラムを実行します。最初のテキスト ボックスに「`10`」(10 分を表す) と入力し、2 番目のテキスト ボックスに「`5`」(5 マイルを表す) と入力して、Button1 をクリックします。

   メッセージ ボックスに "Average speed 0.03333334" というメッセージが表示されます。しかし、5 マイルを 10 分で移動する場合、正しい答えは 30 mph です。

プロジェクトは開いたままにしておいてください。次の手順で、論理エラーを検出する方法について説明します。

論理エラーの検出

前の例では、プログラムのロジックに明白な間違いがあります。1 時間あたり 30 マイル移動したはずが、この結果では、1 時間あたり 1 マイル未満しか移動していないことになります。どこにエラーがあるのでしょうか。

次の手順では、ブレークポイントを設定してコードをステップ実行し、エラーを検出します。

やってみよう

ブレークポイントを設定してコードをステップ実行するには

1. コード エディタで行 `hours = minutes / 60` を探し、そのコード行の左マージンをクリックします。

   マージンに赤い点が表示され、コードが赤で強調表示されます。これはブレークポイントを示します。

2. F5 キーを押して再度プログラムを実行します。最初のテキスト ボックスに「`10`」、2 番目のテキスト ボックスに「`5`」と入力します。次に Button1 をクリックします。

   ブレークポイントに達すると、プログラムが停止します。行 `hours = minutes / 60` が黄色で強調表示されます。

   変数の値の上にマウスを置いて、値を調べます。`hours` の値は `0`、`minutes` の値は `10` になります。

3. F8 キーを押して行 `hours = minutes / 60` を実行し、次の行に進みます。

   行 `MsgBox("Average speed " & GetMPH(hours, miles))` の変数の値を調べます。`hours` の値は `0.166666672`、`miles` の値は `5.0` になります。

4. F8 キーを再度押して、現在の行を実行します。

   実行が行 `Function GetMPH` に進みます。

   この行で変数の値を調べます。前の行とは逆に、`miles` の値が `0.166666672` になり、`hours` の値が `5.0` になっています。エラーが検出されました。

   プロジェクトは開いたままにしておいてください。次の手順で、論理エラーを修正する方法について説明します。

論理エラーの修正

前の手順では、変数 `miles` と `hours` の値の位置が入れ替わっていました。原因がわかりますか。

行 `MsgBox("Average speed " & GetMPH(hours, miles))` を見ると、`GetMPH` 関数に 2 つの引数 `hours` および `miles` がこの順序で渡されています。関数宣言 `Function GetMPH(ByVal miles As Double, ByVal hours As Double)...` を見ると、引数は `miles`、`hours` の順になっています。

引数が間違った順序で渡されたため、論理エラーが発生し、間違った計算が行われました。引数の型が異なっていれば、ランタイム エラーが発生したと思われますが、引数の型が同じだったため、ランタイム エラーは発生しませんでした。単純な間違いでしたが、その結果発生したバグは、検出するのが困難でした。

次の手順では、ブレークポイントを設定してコードをステップ実行し、エラーを検出します。

やってみよう

論理エラーを修正するには

1. コード エディタで、行 `MsgBox("Average speed " & GetMPH(hours, miles))` を次のように変更します。

   VB

   ```
   MsgBox("Average speed " & GetMPH(miles, hours))
   ```

2. 左マージンの赤い点をクリックして、ブレークポイントを解除します。

3. F5 キーを押してプログラムを実行します。最初のテキスト ボックスに「`10`」と入力し、2 番目のテキスト ボックスに「`5`」と入力して、Button1 をクリックします。

   今度は、正しい結果である "Average speed 30" を表示するメッセージ ボックスが表示されます。

プログラムは修正されたように見えますが、検出がさらに困難な論理エラーが存在します。この論理エラーの検出を試みる場合は、プロジェクトを開いたままにしておいてください。レッスン「追加レッスン : 他のエラーを発見する」でプロジェクトを再び使用します。

次の手順

このレッスンでは、論理エラーを検出して修正する方法について説明しました。ここで、コメントの使用に関する次のレッスンに進むか、「追加レッスン : 他のエラーを発見する」で別の論理エラーを検出してみるかを選択できます。

次のレッスン : 「プログラムの注釈 : コメントを使用する」

参照

処理手順

ランタイム エラーの検索と除去
バグの理解 : 3 種類のプログラミング エラー
エラーの検出 : Visual Basic のデバッグの概要

追加レッスン：他のエラーを発見する

このレッスンでは、特定の状況でのみ発生する論理エラーを突き止める方法について説明します。

前のレッスン「想定外の現象 : 論理エラーを突き止める」では、論理上発生したエラーを検出して修正する方法について説明しました。前のレッスンのプログラム例には、重大なバグが隠れています。これは、特定の状況でのみ発生するため、検出が困難なバグです。

プログラムのテスト

プログラマは、プログラムをテストして、プログラムが希望どおりに動作するかどうかを確認するのには適していません。プログラムがどのように動作するかを知っているため、論理エラーが検出されるような間違いをする可能性はほとんどありません。しかし、プログラムに慣れていないユーザーは、予想外の処理を実行する可能性があります。

たとえば、旅行したマイル数をかかった時間で割って、1 時間あたりのマイル数を計算するプログラムでは、ユーザーが時間数やマイル数にゼロを入力した場合、どうなるでしょうか。では、実際に試して確認してみましょう。

やってみよう

プログラムをテストするには

1. 前のレッスン「想定外の現象 : 論理エラーを突き止める」で作成した `LogicErrors` プロジェクトを開きます。

メモ：
前のプロジェクトを完了または保存していない場合は、続行する前に、前のプロジェクトに戻って完了する必要があります。

2. F5 キーを押してプログラムを実行します。最初のテキスト ボックスに「`0`」(分を表す) と入力し、2 番目のテキスト ボックスに「`5`」(マイルを表す) と入力して、Button1 をクリックします。

   "Average speed Infinity" というメッセージを含むメッセージ ボックスが表示されます。

   プロジェクトは開いたままにしておいてください。次の手順で、論理エラーを検出する方法について説明します。

5 を 0 で割ると無限大か

前の手順で、"無限大" を予期していたわけではありませんが、これは数学的には正確です。5 を 0 で割ると商は無限大です。しかし、これはプログラムのユーザーに対して表示したい結果ではありません。これを回避するには、どのような方法があるでしょうか。

「問題が発生した場合の対処 : エラー処理」のレッスンで説明した手順に従って、エラー ハンドラを追加するという方法も考えられます。しかしこの場合にはうまくいきません。"無限大" という結果はエラーではなく、単に、表示したい結果と異なるというだけだからです。

速度をゼロと表示しても役に立たないため、問題を修正する 1 つの方法は、値がゼロでないかチェックして、ゼロより大きい値を入力する必要があることをユーザーに警告することです。ユーザーが負数を入力できないようにすることもできます。負数を使用した場合も、誤った結果が生成されるためです。

次の手順では、値がゼロより大きい場合のみ `GetMPH` 関数を呼び出すように、**Button1_Click** イベント ハンドラのコードを変更します。

やってみよう

バグを修正するには

1. コード エディタで、**Button1_Click** イベント ハンドラのコードを次のように変更します。

   VB

```
Dim minutes As Integer = CInt(Textbox1.Text)
Dim miles As Double = CDbl(Textbox2.Text)
Dim hours As Double = 0
If minutes <= 0 Or miles <= 0 Then
  MsgBox("Please enter a number greater than zero")
Else
  hours = minutes / 60
  MsgBox("Average speed " & GetMPH(miles, hours))
End If
```

2. F5 キーを押して再度プログラムを実行します。最初のテキスト ボックスに「0」、2 番目のテキスト ボックスに「5」と入力します。次に Button1 をクリックします。

   0 より大きな値を入力するよう指示するメッセージ ボックスが表示されます。他のさまざまな数値の組み合わせでプログラムをテストし、バグが修正されたことを確認します。

次の手順

このレッスンでは、予期しない動作の原因になった論理エラーを検出して修正する方法について説明しました。次のレッスンでは、コード内でコメントを使用する方法について説明します。

次のレッスン : 「プログラムの注釈 : コメントを使用する」

参照

処理手順

想定外の現象 : 論理エラーを突き止める
問題が発生した場合の対処 : エラー処理
バグの理解 : 3 種類のプログラミング エラー
エラーの検出 : Visual Basic のデバッグの概要

関連項目

/ 演算子 (Visual Basic)

プログラムの注釈：コメントを使用する

このレッスンでは、プログラムのコード内にコメントを作成する方法を説明します。

プログラムを構成するコードは、読んで理解するのが難しいことがあります。自分で記述したコードでない場合には特にそうです。コメントを使用すると、自分自身、またはコードを使用する他の人を対象に、メモを作成できます。

コメントは、プログラム実行時に Visual Basic コンパイラによって無視される、コード エディタで入力されたテキストです。このため、プログラムの特定のセクションが果たす役割を説明したり、完了していないプログラミング タスクの実行を促したりするために、自分自身に対してメモを記述できます。

コメントを作成するには、行の先頭に ' 文字を入力します。コメントの作成方法を次の例に示します。

VB

```
' This is a comment. WOW!
```

同様に ' 文字を使用して、コメントを行末に追加することもできます。この方法は、各コード行に関するコメントを付加する場合によく使用されます。次に例を示します。

VB

```
MsgBox("Hello World!") ' This line causes a message box to appear.
```

単一行コメントと同様、行内で ' 文字の後に続くすべてのテキストは、プログラムによって無視されます。

デバッグのためにコメントを使用する

コメントのもう 1 つの一般的な用途は、プログラムのデバッグ時に、一時的にコード行が実行されないようにすることです。たとえば、メッセージ ボックスを表示する次のようなコード行があるとします。

VB

```
MsgBox("Hello World!")
```

メッセージ ボックスを表示せずにプログラムを実行するが、メッセージ ボックスを完全に削除しない場合は、次に示すようにコメント文字 (') を使用して、メッセージ ボックスを一時的にプログラムから隠すことができます。

VB

```
' MsgBox("Hello World!")
```

' 文字の後にあるすべてのテキストは無視されるため、プログラムはこの行を飛ばして実行されます。後で ' 文字を削除すると、メッセージ ボックスが表示されるようになります。

やってみよう

コメントを挿入するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Comments`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開きます。
5. **Form1_Load** イベント ハンドラに次のコードを追加します。

   VB

```
' This code will cause two message boxes to appear
MsgBox("This is Message Box 1") ' Display Message Box 1
MsgBox("This is Message Box 2") ' Display Message Box 2
```

6. F5 キーを押してプログラムを実行します。

   プログラムが起動し、2 つのメッセージ ボックスが順に表示されます。

7. [デバッグ] メニューの [デバッグの停止] をクリックしてプログラムを終了します。

8. コード エディタで、1 つ目のメッセージ ボックスの行に、コメント文字 (') を次のように追加します。

   VB

```
' MsgBox("This is MessageBox 1") ' Ignore Message Box 1
```

9. F5 キーを押してプログラムを実行します。

   今度は、1 つ目のメッセージ ボックスの行が無視され、2 つ目のメッセージ ボックスのみが表示されます。

次の手順

このレッスンでは、コード内にコメントを作成する方法と、各コード行を一時的に無効にする方法を説明しました。これで、デバッグに関するレッスンは終了です。次からのレッスンでは、データベースを使用してプログラムの情報を格納および取得する方法を説明します。

次のレッスン : 「レコードの管理 : プログラムでデータを使用する」

参照

概念

コード内のコメント

その他の技術情報

問題の確認 : デバッグを行ってエラーを検出し、修正する

レコードの管理 : プログラムでデータを使用する

ほとんどのプログラムは、データを何らかの方法で使用します。たとえば、先のレッスンでは、数字のフォームにデータを入力し、そのデータを使用して計算を行って、結果をメッセージ ボックスに返しました。

非常に単純なプログラムでは、データはプログラム自体のフィールドとして表されます。より複雑なプログラムでは、データはデータベースと呼ばれる、プログラムとは別の構造に格納されます。

この一連のレッスンでは、データベースを作成し、データベースを使用してプログラム内のデータを表示および更新する方法について説明します。

このセクションの内容

関連するセクション

Visual Basic ガイド ツアー

データの格納とアクセス

このレッスンでは、データベースを使用してデータを格納し、データにアクセスする方法について説明します。

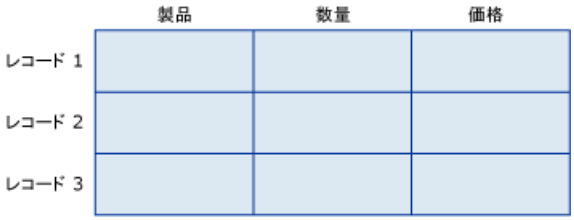
データは、コンピュータ プログラミングの中心となる概念です。ほとんどのプログラムは、データを何らかの方法で使用します。たとえば、先のレッスンでは、数字のフォームにデータを入力し、そのデータを使用して計算を行った後、メッセージ ボックスに返しました。

非常に単純なプログラムでは、データはプログラム内のフィールドとして表されます。より複雑なプログラムでは、データはプログラムとは別の構造に格納されます。この構造をデータベースと呼びます。

データベースとは

データベースはデータの集合であり、プログラムから独立したファイルに格納されます。データベースに格納されるデータは、テキスト、数値、画像など、さまざまな種類があります。さまざまなプログラムが同じデータベースに接続して、データベース内のデータを表示および更新できます。

通常、データベースは 1 つ以上のテーブルに分かれています。テーブルは、関連するレコードの集合です。たとえば、中小企業に関するデータを保持したデータベースを使用する場合、製品を表すテーブル、注文を表すテーブル、および顧客を表すテーブルを使用できます。



各テーブルは、列と行のグリッドとして構成されます。列は、レコード内のデータのカテゴリを表し、行は個別のレコードを表します。たとえば、上記の図では、各注文を表す個別の行またはレコード、および注文された製品を表す列が、数量および価格と共に Orders テーブルに表示されます。

データを使用した作業の開始

プログラムからデータベース内のデータにアクセスするには、まずアクセスするデータベースを用意する必要があります。Visual Basic を使用すると、独自のデータベースの作成や、他の人が作成したデータベースの利用が簡単にできます。

Visual Basic Express Edition は、Microsoft SQL Server データベースまたは Microsoft Access データベースの 2 種類のデータベースにアクセスできます。このレッスンでは、SQL Server データベースを使用します。

データベースを準備した後、DataSet と呼ばれるオブジェクトを使用してデータベースをプログラムに関連付け、データ バインディング と呼ばれる方法を使用してフォーム上のフィールドまたはコントロールをデータに関連付けます。

TextBox コントロールなど、プログラム内のフィールドがデータベース テーブルの列にバインドされている場合、テキスト ボックス内のその列でデータを表示し、テキスト ボックスのデータを変更してデータベースに保存するか、新しいレコードのデータを入力してデータベースに追加できます。

これは複雑に思われそうですが、実際には難しくありません。以降のレッスンで説明するように、Visual Basic のデータベース ツールにより、データを使用した作業が簡単になります。

次の手順

このレッスンでは、データベースの機能、およびデータベースをプログラムに関連付ける方法について説明しました。次のレッスンでは、プログラムで使用できるデータベースを作成する方法について説明します。

メモ :
Visual Basic Express Edition を使用して SQL Server データベースを作成してアクセスするには、SQL Server Express Edition もインストールする必要があります。既定では、Visual Basic Express Edition のインストール時に SQL Server Express Edition もインストールされますが、その際にインストールしなかった場合は、次のレッスンに進む前にインストールしておく必要があります。

次のレッスン : 「初めてのデータベースの作成」

参照

その他の技術情報

レコードの管理 : プログラムでデータを使用する

Visual Basic ガイド ツアー

初めてのデータベースの作成

このレッスンでは、データベースを作成する方法を説明します。このデータベースは、アドレス帳プログラムを作成する、後続のレッスンで使用します。

前のレッスンでは、データベースを使用して Visual Basic プログラムのデータの格納や取得ができることを説明しました。まず、アクセスするデータベースを作成する必要があります。既存のデータベースを使用することもできますが、このレッスンでは、Visual Basic に含まれる Visual Database Tools を使用して新しいデータベースを作成する方法について説明します。

前提条件

Visual Basic Express Edition を使用して SQL Server データベースを作成してアクセスするには、SQL Server Express Edition もインストールする必要があります。既定では、Visual Basic Express Edition のインストール時に SQL Server Express Edition もインストールされますが、インストールしない選択をした場合、続行する前にインストールを実行する必要があります。

やってみよう

データベースを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`FirstDatabase`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. [プロジェクト] メニューの [新しい項目の追加] をクリックします。
5. [新しい項目の追加] ダイアログ ボックスで、[SQL データベース] をクリックします。
6. [ファイル名] ボックスに「`FirstDatabase`」と入力し、[追加] をクリックします。

   データ ソース構成ウィザードが起動します。
7. データ ソース構成ウィザードで、[キャンセル] をクリックします。

   `FirstDatabase.mdf` という新しいデータベースがプロジェクトに追加され、ソリューション エクスプローラに表示されます。

テーブルの追加

上記の手順のように、データベースの作成は簡単です。この時点では、データベースにはデータが含まれていないため役に立ちません。次の手順では、データベースにテーブルを追加します。ここでは、アドレス情報を格納するテーブルを追加します。

やってみよう

データベースにテーブルを追加するには

1. [表示] メニューの [データベース エクスプローラ] をクリックします。
2. [データベース エクスプローラ] の [FirstDatabase.mdf] ノードを展開 (正符号をクリック) し、[テーブル] ノードを選択します。
3. [データ] メニューの [新規追加] をポイントし、[テーブル] をクリックします。

   [テーブル デザイナ] ウィンドウが開きます。
4. [プロパティ] ウィンドウで、[オブジェクト名] をクリックして「`Addresses`」と入力します。
5. [テーブル デザイナ] ウィンドウで、[列名] フィールドを選択して「`FirstName`」と入力します。
6. [データ型] フィールドを選択して、ドロップダウン リストから nvarchar(50) を選択します。[Null を許容] 列が自動的にチェックされます。

   これで、新しいテーブルの最初の列を定義しました。

   前の 2 つの手順を繰り返して、次の値を持つ 4 つの列を追加します。

   a. [列名] : `LastName`、[データ型] : nvarchar(50)

   b. [列名] : `StreetAddress`、[データ型] : nvarchar(50)

c. [列名] : `City`、[データ型] : nvarchar(50)

d. [列名] : `Phone`、[データ型] : nvarchar(50)

7. [ファイル] メニューの [Addresses を保存] をクリックします。

キーの追加

これで、アドレス帳用に名前、住所、および電話番号データを格納するために使用する、データベース内のテーブルが作成されました。さらに、キーを追加して、レコードが重複しないようにします。

キー列は、主キーとも呼ばれ、テーブル内の 1 つまたは複数の列の値が一意となるように指定します。ある値を含む行は、テーブル内に 1 つのみ存在します。同じ値の 2 つ目の行を入力しようとすると、エラーが発生します。

`Addresses` テーブルの場合、`FirstName` 列と `LastName` 列の両方を主キーとして指定します。名または姓が同じ人が複数いる可能性はありますが、同姓同名の知人が 2 人いる可能性は低いためです。

やってみよう

テーブルにキーを追加するには

1. テーブル デザイナで、[FirstName] 行および [LastName] 行の [Null を許容] チェック ボックスをオフにします。
2. [FirstName] 行と [LastName] 行の両方を選択します。

ヒント
[FirstName] フィールドの左側にあるグレーの四角形をクリックして、Ctrl キーを押したまま [LastName] 行をクリックすると両方の行を選択できます。

3. テーブル デザイナ メニューの [主キーの設定] をクリックします。

   各行の左側に小さい鍵の記号が表示されます。

4. [ファイル] メニューの [Addresses を保存] をクリックします。

データの追加

これで、1 つのテーブル `Addresses` を含むデータベースが作成されました。データベースは、データが含まれていなければ、あまり役に立ちません。次の手順では、`Addresses` テーブルにデータを追加します。必要に応じて、例の名前と住所を、知っている人の名前と住所で置き換えることもできます。

やってみよう

テーブルにデータを追加するには

1. データベース エクスプローラで [テーブル] ノードを展開し、[Addresses] ノードを選択して、[データ] メニューの [テーブル データの表示] をクリックします。

   データ テーブル ウィンドウが開きます。

2. データ テーブル ウィンドウで、[FirstName] フィールドを選択して「`Samantha`」と入力します。

メモ :
最初にフィールドをクリックすると、値 NULL が各フィールドに表示されます。null は、フィールドが空であることを示すデータベース用語です。

3. [LastName] フィールドを選択して、「`Smith`」と入力します。
4. [StreetAddress] フィールドを選択して、「`123 45th Ave. E`」と入力します。
5. [City] フィールドを選択して、「`Seattle`」と入力します。
6. [Phone] フィールドを選択して「`2065550100`」と入力し、Tab キーを押します。

   これで、`Addresses` テーブルの最初のレコードを定義しました。

前の 5 つの手順を繰り返して、次の値を持つ 2 つのレコードを追加します。

a. [FirstName] : `Michael`、[LastName] : `Alexander`、[StreetAddress] : `789 W. Capital Way`、[City] : `Tacoma`、[Phone] : `2065550101`

b. [FirstName] : `Andrea`、[LastName] : `Dunker`、[StreetAddress] : `722 Moss Bay Blvd`、[City] : `Kirkland`、[Phone] : `2065550102`

7. [ファイル] メニューの [すべてを保存] をクリックして、プロジェクトとデータベースを保存します。

データを入力している間は、小さな鉛筆のアイコンがデータの横に表示されます。Tab キーを押して次の行に移動すると、アイコンは消えます。この鉛筆のアイコンは、そのデータがデータベースに保存されていないことを示します。データの入力行から移動すると、行全体のデータがデータベースに自動的に保存されます。

次の手順

このレッスンでは、データベースを作成して、データベース テーブルを追加し、IDE のテーブルにレコードを追加しました。次のレッスンでは、プログラム内でデータベースを使用する方法について説明します。

次のレッスン : 「必要な情報の入手 : 既存のデータベースに接続する」

参照

処理手順

データの格納とアクセス

その他の技術情報

レコードの管理 : プログラムでデータを使用する

Visual Basic ガイド ツアー

必要な情報の入手：既存のデータベースに接続する

このレッスンでは、プログラムを既存のデータベースに接続する方法を説明します。

既存のデータベースへの接続は簡単です。Visual Basic Express Edition のビジュアル ツールを使用すると、データベースを参照してローカル コピーをプロジェクトに追加できます。このレッスンでは、新しいプロジェクトを作成して、前のレッスンで作成した `Addresses` データベースにこのプロジェクトを接続します。

やってみよう

既存のデータベースに接続するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Addresses`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ソリューション エクスプローラで `FirstDatabase.mdf` ノードを選択します。
5. [プロパティ] ウィンドウの [出力ディレクトリにコピー] プロパティを選択し、その値を [新しい場合はコピーする] に変更します。
6. ソリューション エクスプローラの [データ ソース] タブをクリックします。
7. [データ ソース] ウィンドウで、[新しいデータ ソースの追加] をクリックします。

   データ ソース構成ウィザードが起動します。
8. [データベース] をクリックし、[次へ] をクリックします。
9. [新しい接続] をクリックします。

   [接続の追加] ダイアログ ボックスが表示されます。
10. [接続の追加] ダイアログ ボックスで、[データ ソース] が [Microsoft SQL Server データベース ファイル (SqlClient)] でない場合には、[変更] をクリックし、[データ ソースの変更] ダイアログ ボックスで [Microsoft SQL Server データベース ファイル] を選択します。[OK] をクリックします。
11. [参照] をクリックして、`FirstDatabase.mdf` データベースを保存した場所に移動し、[開く] をクリックします。
12. [OK] をクリックしてダイアログ ボックスを閉じ、データ ソース構成ウィザードで [次へ] をクリックします。データ ファイルをプロジェクトにコピーするかどうか確認するメッセージが表示されたら、[はい] をクリックします。
13. ウィザードの次のページで、[次の名前で接続を保存する] チェック ボックスをオンにします。[次へ] をクリックして続行します。
14. [データベース オブジェクトの選択] ページで、[テーブル] ノードを展開し、[Addresses] テーブルのチェック ボックスをオンにします。
15. [完了] をクリックして終了します。

    ローカル データベース ファイルがプロジェクトに追加されています。また `FirstDatabaseDataSet` オブジェクトが [データ ソース] ウィンドウに追加されています。
16. [ファイル] メニューの [すべてを保存] をクリックして、プロジェクトを保存します。

次の手順

このレッスンでは、ローカル データベースをプロジェクトに追加する方法を説明しました。次のレッスンでは、ユーザー インターフェイスを作成してデータベース内のレコードを表示する方法を説明します。

次のレッスン：「ユーザーへの情報表示：ユーザー インターフェイスにデータを表示する」

参照

処理手順

初めてのデータベースの作成

その他の技術情報

レコードの管理：プログラムでデータを使用する

Visual Basic ガイド ツアー

ユーザーへの情報表示：ユーザー インターフェイスにデータを表示する

このレッスンでは、ローカル データベース内のデータを表示するための基本的なユーザー インターフェイスを作成する方法を説明します。

ここまでの手順で、ローカル データベース ファイルへの接続が作成されました。次の手順では、データを表示するユーザー インターフェイスを作成します。データベースからデータを取得し、そのデータをユーザー インターフェイスに表示する処理の背後にあるプログラミングは、非常に複雑です。しかし、Visual Basic では、必要なデータ オブジェクトが自動的に作成および構成されるため、ユーザーに必要な操作は、オブジェクトの選択と配置のみです。このレッスンでは、簡単なデータ表示フォームを作成する方法を説明します。

やってみよう

データ アクセス フォームを作成するには

1. 前のレッスンで使用した `Addresses` プロジェクトを開きます。このプロジェクトを作成していない場合は、「必要な情報の入手 : 既存のデータベースに接続する」に戻ってレッスンを完了してから次に進んでください。
2. ソリューション エクスプローラで Form1.vb を選択して、[表示] メニューの [デザイナ] をクリックします。
3. ソリューション エクスプローラの [データ ソース] タブをクリックします。

   [データ ソース] ウィンドウで、しばらくの間 [FirstDatabaseDataSet] ノードと [Addresses] ノードを調べます。[Addresses] ノードを展開すると、テーブル内の個別のフィールドがすべて表示されます。
4. [Addresses] ノードを [データ ソース] ウィンドウからフォームにドラッグします。

   メモ：

   いくつかのコントロールが自動的にフォームに追加されます。またいくつかのコンポーネントが作成され、フォームの下にあるコンポーネント トレイに追加されます。この中には、テーブルの行と列を表示する **DataGridView** コントロールと、移動のためのコントロール (AddressesBindingNavigator) が含まれます。さらに、データベースへの接続、データの取得と更新の管理、およびローカル **DataSet** へのデータの格納を行う各コンポーネント (それぞれ AddressesBindingSource、AddressesTableAdapter、および FirstDatabaseDataSet) が、Visual Basic によって作成されます。
5. AddressesDataGridView コントロールを選択し、[プロパティ] ウィンドウで **Dock** プロパティを **Fill** に設定します (中央のボタンをクリックします)。

   こうすると、グリッドがフォーム全体に拡張されます。
6. F5 キーを押してプログラムを実行します。

   `Addresses` テーブルのデータが、フォーム上の **DataGridView** コントロールに表示されます。BindingNavigator 内のコントロールを使用すると、行間の移動や、レコードの追加または削除も実行できます。また、グリッドに表示されたデータを変更することでレコードを変更できますが、これらの変更は、[データの保存] アイコンをクリックしないと保存されません。データに対する変更を自動的に保存する方法については、次のトピックで説明します。

次の手順

このレッスンでは、Visual Basic のビジュアル ツールを使用して、基本的なデータ指向ユーザー インターフェイスを作成する方法を説明しました。次のレッスンでは、データ エントリ フォームを作成して、データベースに対する変更を保存する方法を説明します。

次のレッスン :「レコードの追加または変更 : データを更新する」

参照

処理手順

必要な情報の入手 : 既存のデータベースに接続する

その他の技術情報

レコードの管理 : プログラムでデータを使用する

Visual Basic ガイド ツアー

レコードの追加または変更：データを更新する

このレッスンでは、データ エントリ フォームを作成して、ローカル データベースのデータを更新する方法を説明します。

前の 3 つのレッスンでは、データベースの作成、データベース ファイルのプロジェクトへの追加、および基本的なユーザー インターフェイスの作成を行いました。これらの時点でも、アドレス データの変更や、さらには新しいレコードの追加が可能でした。しかし、プログラムを終了して再起動すると、これらの変更は失われていました。

このデータは、実際にはローカルの **DataSet** に格納された、データベース内のデータのコピーでした。プログラムが起動するたびに、**DataSet** はデータをデータベースから取得します。**DataSet** に変更が加えられても、その変更はデータベースには反映されません。

AddressesBindingNavigator コントロールの [データの保存] をクリックすると、すべての変更が **DataSet** からデータベースにコピーされます。ユーザーが必ず作業内容を保存するとは考えられないため、プログラム終了時に自動的に変更をデータベースに保存するコードを追加します。同時に、ユーザー インターフェイスも変更して、データを簡単に入力できるようにします。

やってみよう

ローカル データベース ファイルを更新するには

1. 前のレッスンで使用した `Addresses` プロジェクトを開きます。前のレッスンをまだ完了していない場合は、「ユーザーへの情報表示 : ユーザー インターフェイスにデータを表示する」に戻って手順を完了します。
2. ソリューション エクスプローラで Form1 を選択して、[表示] メニューの [デザイナ] をクリックします。
3. フォームで AddressesDatGridView コントロールをクリックして削除します。
4. ソリューション エクスプローラの [データ ソース] タブをクリックします。
5. [データ ソース] ウィンドウで [Addresses] テーブルをクリックし、ボックスの一覧の [詳細] をクリックします。
6. [Addresses] ノードを [データ ソース] ウィンドウから新しいフォームにドラッグします。

   テーブルの各フィールドに、そのフィールドの内容を記述する **Label** コントロールと共に、**TextBox** コントロールが追加されます。
7. フォームをダブルクリックしてコード エディタを開きます。
8. [イベント] ボックスの [FormClosing] をクリックします。
9. Form1_FormClosing イベント ハンドラに次のコードを追加します。

   ```
   Me.AddressesBindingSource.EndEdit()
   Me.AddressesTableAdapter.Update(Me.FirstDatabaseDataSet.Addresses)
   ```

   このコードにより、AddressesTableAdapter がデータセット内の変更をローカル データベースにコピーします。
10. F5 キーを押してプログラムを実行します。

    一部のデータに変更を加えるか、新しいレコードを追加して、フォームを終了します。
11. 再度 F5 キーを押します。変更が保存されています。

ここまでの一連のレッスンでは、データベースとプログラムを作成してデータベースにアクセスする方法を説明しました。次からのレッスンでは、プログラムで再利用可能なオブジェクトの設計図であるクラスについて説明します。

次のレッスン :「オブジェクトによるプログラミング : クラスを使用する」

参照

処理手順

必要な情報の入手 : 既存のデータベースに接続する

その他の技術情報

レコードの管理 : プログラムでデータを使用する

Visual Basic ガイド ツアー

オブジェクトによるプログラミング : クラスを使用する

前のレッスンで説明したように、Visual Basic プログラムは、フォーム、コントロールなどのオブジェクトで作成されます。オブジェクトは、人、コンピュータなどの現実世界のもの、さらに抽象的な銀行口座などのものを表すこともできます。

クラスは、オブジェクトの種類を単に表したものです。クラスは、オブジェクトの設計図と見なされます。1 枚の設計図から複数の建築物を建造できるように、1 つのクラスからオブジェクトの複数のコピーを作成できます。

以降のレッスンでは、Visual Basic プログラムでクラスを使用する方法について説明します。

このセクションの内容

関連するセクション

クラスとは

このレッスンでは、クラスを使用してプログラム内でオブジェクトを表現する方法を説明します。

前のレッスンで説明したように、Visual Basic プログラムは、フォームやコントロールなどのオブジェクトで構成されています。またオブジェクトを使用すると、現実世界のもの (人やコンピュータなど) や、もっと抽象的なもの (銀行口座など) を表現することもできます。

クラスとは、オブジェクトの種類を表すものに過ぎません。オブジェクトを表現する設計図であるとも言えます。1 枚の設計図から複数の建築物を建造できるように、1 つのクラスからオブジェクトの複数のコピーを作成できます。

実は、これまでのレッスンで既にクラスを使用しています。たとえば、**TextBox** コントロールは、その外観と機能を定義する **TextBox** クラスによって定義されます。**TextBox** コントロールをフォームにドラッグするたびに、**TextBox** クラスの新しいインスタンスが実際に作成されます。

各 **TextBox** コントロールは、それを定義するクラスである TextBox クラスの完全なコピーですが、それぞれ独立しています。各オブジェクトはクラスの個別の "インスタンス" であるため、クラスを作成する操作はインスタンス化と呼ばれます。

これまでのレッスンでは、ツールボックスから **TextBox** コントロールをドラッグすることによってフォームに追加しましたが、次に示すように、コードで **New** キーワードと共に **TextBox** オブジェクトを宣言することでも、これをインスタンス化できます。

VB

```
Dim Textbox1 As New TextBox
```

クラスの作成および使用については、次からの一連のレッスンで詳しく説明します。

クラスに含まれるもの

前のレッスン (「詳細情報 : プロパティ、メソッド、イベントについて」) で、すべてのオブジェクトには、属性を示すプロパティ、操作を示すメソッド、応答を示すイベントがあることを説明しました。同様に、オブジェクトを定義するクラスにも、そのクラスのすべてのインスタンスに受け継がれる独自のプロパティ、メソッド、およびイベント (メンバとも呼ばれる) があります。

たとえば、銀行口座を表すクラスには、`AccountNumber` や `AccountBalance` などのプロパティ、`CalculateInterest` などのメソッド、`BalanceChanged` などのイベントがあることが考えられます。銀行口座オブジェクトをインスタンス化すると、**TextBox** などのオブジェクトと同様に、そのプロパティ、メソッド、およびイベントにアクセスできます。

クラスの一部のメンバはプライベートです。つまり、クラス内のコードからのみアクセスできます。たとえば、銀行口座クラスには、残高を計算するメソッドがあることが考えられます。この場合、プログラムからその残高を読み込むことができるようにはしても、残高を直接変更できるようにはしない方がよいと考えられます。

クラスのメンバを隠ぺいするには、そのメンバを **Private** として宣言し、公開するには、**Public** として宣言します。プロパティへのアクセスは許可するが、値の変更を許可しない場合は、そのプロパティを **ReadOnly** として宣言します。次のコードは、`BankAccount` クラスの例を示しています。

VB

```
Class BankAccount
    Private AccountNumber As String
    Private AccountBalance As Decimal
    Public Sub UpdateBalance()
        ' add code to recalculate balance.
    End Sub
    ReadOnly Property Balance() As Decimal
        Get
            Return AccountBalance
        End Get
    End Property
End Class
```

次の手順

このトピックでは、クラスの基本と共に、いくつかの新しい技術について説明しました。次のレッスンでは、クラスを作成する方法を説明します。

次のレッスン : 「現実世界のオブジェクトのモデリング : 初めてのクラスの作成」

参照

概念

詳細情報 : プロパティ、メソッド、イベントについて

クラス : オブジェクトの設計図

その他の技術情報

オブジェクトによるプログラミング : クラスを使用する

現実世界のオブジェクトのモデリング：初めてのクラスの作成

このレッスンでは、クラス ライブラリ プロジェクトを使用してクラスを作成する方法を説明します。

前のレッスンでは、現実世界の物をモデル化したオブジェクトの設計図として、クラスを使用できることを説明しました。クラスを使用する最大の理由の 1 つとして、ある種類のオブジェクトのクラスを作成すると、あらゆるプロジェクトでそのクラスを再利用できる点が挙げられます。

たとえば、記述する多くのプログラムに、人が関係している場合があります。つまり、友人に関する記録を保持するアドレス帳プログラム、ビジネス上の連絡を行うコンタクト マネージャ プログラム、従業員の記録を取るプログラムなどです。プログラムはそれぞれ大きく異なりますが、人に適用される属性は同じです。すべての人には、名前、年齢、住所、電話番号があります。

これ以降のいくつかのレッスンでは、人を表すクラスを作成します。このクラスを保存しておくと、今後記述する他のプログラムに使用できます。

クラスの作成方法は 3 つあります。Windows アプリケーション プロジェクトのフォーム モジュールに含まれるコードの一部として作成する方法、Windows アプリケーション プロジェクトに追加される個別のクラス モジュールとして作成する方法、スタンドアロンのクラス ライブラリ プロジェクトとして作成する方法です。

クラスを作成する

前のいくつかのレッスンにもありましたが、フォームをダブルクリックしてコード エディタを開くと、次のようなコードが表示されます。

```
Public Class Form1
    Private Sub Form1_Load...

    End Sub
End Class
```

フォームは、実際には **Class** ステートメントと **End Class** ステートメントで示されたクラスであり、これら 2 つのステートメントの間に入力するコードは、クラスの一部となります。既定ではフォーム モジュールに含まれるクラスは 1 つだけですが、次のように **End Class** ステートメントの下にコードを追加することで、追加のモジュールを作成できます。

```
Public Class Form1
    ' Form1 code here
End Class
Public Class MyFirstClass
    ' Your class code here
End Class
```

この方法でクラスを作成することの欠点は、クラスを作成したプロジェクトでしかそのクラスを使用できないことです。他のプロジェクトとクラスを共有する場合は、そのクラスをクラス モジュールに追加します。

クラス モジュール

クラス モジュールは、1 つ以上のクラスを含む個別のコード ファイルです。個別のファイルであるため、他のプロジェクトで再利用できます。クラス モジュールの作成方法は 2 つあります。Windows アプリケーション プロジェクトに追加するモジュールとして作成する方法とスタンドアロンのクラス ライブラリ プロジェクトとして作成する方法です。

新しいクラス モジュールを既存のプロジェクトに追加するには、[プロジェクト] メニューの [新しい項目の追加] ダイアログ ボックスを開き、[クラス] をクリックします。ここでは、以降のいくつかのレッスンで使用するために、スタンドアロンのクラス ライブラリ プロジェクトを作成します。

やってみよう

クラス ライブラリ プロジェクトを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[クラス ライブラリ] をクリックします。
3. [プロジェクト名] ボックスに「`Persons`」と入力し、[OK] をクリックします。

   新しいクラス ライブラリ プロジェクトが開き、コード エディタにクラス モジュール `Class1.vb` が表示されます。
4. ソリューション エクスプローラで、`Class1.vb` を右クリックして [名前の変更] をクリックし、名前を `Persons.vb` に変更します。

   コード エディタの名前も `Persons.vb` に変わります。

5. [ファイル] メニューの [すべてを保存] をクリックします。
6. [プロジェクトの保存] ダイアログ ボックスの [上書き保存] をクリックします。

ヒント
プロジェクトを既定の場所に保存せずに、再利用するすべてのクラスを格納するためのディレクトリを作成する方が都合が良い場合もあります。この場合、プロジェクトを保存する前に、[プロジェクトの保存] ダイアログ ボックスの [場所] フィールドに保存先フォルダを入力します。

まだプロジェクトは開いたままにしておいてください。次のレッスンで内容を追加します。

次の手順

このレッスンでは、クラス モジュールの作成方法を説明しました。しかし、空のクラスはほとんど役に立ちません。次のレッスンでは、このクラスにプロパティを追加する方法を説明します。

次のレッスン : 「クラスへのプロパティの追加」

参照

処理手順

方法 : 新しいプロジェクト項目を追加する

概念

クラスとは

その他の技術情報

オブジェクトによるプログラミング : クラスを使用する

クラスへのプロパティの追加

このレッスンでは、前のレッスンで作成したクラスにプロパティを追加する方法について説明します。

前のレッスン「詳細情報 : プロパティ、メソッド、イベントについて」では、すべてのオブジェクトに属性があり、プロパティは属性を表すことについて説明しました。前のレッスンでは、人を表す `Persons` クラスを作成しました。人には名前や年齢などの属性があるため、`Persons` クラスにはそれらの属性を表すプロパティが必要です。

プロパティは、フィールドまたはプロパティ プロシージャのいずれかとしてクラスに追加できます。プロパティの **Public**、**ReadOnly**、または **WriteOnly** の各修飾子を使用して、プロパティの動作を決定することもできます。

フィールドとプロパティ プロシージャ

フィールドは、クラス内の単なるパブリック変数であり、クラスの外部に設定するか、クラスの外部から読み込むことができます。フィールドは、**Boolean** (**True** または **False**) 値など、検証の必要がないプロパティに便利です。`Persons` クラスの場合、人が死んでいるか生きているかを指定する `Alive` という名前の **Boolean** プロパティの設定が考えられます。設定できる値は 2 つしかないため、フィールドはこのプロパティに対して正しく機能します。

フィールドをクラスに追加する場合、コードは次のようになります。

VB

```
Public Alive As Boolean
```

ただし、ほとんどのプロパティはフィールドよりも複雑です。プロパティをクラスに追加するには、ほとんどの場合、プロパティ プロシージャを使用します。プロパティ プロシージャには、プロパティ値を格納するためのプライベート変数の宣言、値を公開する **Get** プロシージャ、および値を設定する **Set** プロシージャの 3 つの部分があります。

たとえば、`Persons` クラスの `Name` プロパティのプロパティ プロシージャは、次のようになります。

VB

```
Private nameValue As String
Public Property Name() As String
    Get
        Name = nameValue
    End Get
    Set(ByVal value As String)
        nameValue = value
    End Set
End Property
```

最初のコード行は、プロパティの値を格納する **String** プライベート変数 `nameValue` を宣言します。プロパティ プロシージャ自体は `Public Property` で始まり、`End Property` で終了します。

Get プロシージャには、値を読み取るときに実行されるコードが含まれます。たとえば、`Persons.Name` プロパティを読み取る場合、コードは `nameValue` 変数に格納された値を返します。

Set プロシージャには、`value` 引数として渡される値を使用して、`nameValue` 変数に新しい値を割り当てるために使用されるコードが含まれます。たとえば、コード `Persons.Name = "John"` を記述すると、**String** 値 `John` が `value` 引数として渡されます。その後、**Set** プロシージャのコードは、この値を `NameValue` 変数に割り当てて格納します。

`Name` プロパティを表すためにフィールドを使用するのではなく、手間のかかる方法を取るのはなぜでしょうか。実際のシナリオでは、名前に対して特定の規則があります。たとえば、通常は名前には数値は含まれません。**Set** プロシージャにコードを追加すると、`value` 引数を確認して、数値が含まれる場合にはエラーを返すことができます。

次のプロシージャでは、`Persons` クラスに 1 つのフィールドと 3 つのプロパティを追加します。

やってみよう

プロパティをクラスに追加するには

1. 前のレッスンで作成した `Persons` プロジェクトを開きます。プロジェクトを保存しなかった場合は、まず前のレッスン「現実世界のオブジェクトのモデリング : 初めてのクラスの作成」に戻って、そのレッスンの手順を完了する必要があります。

2. ソリューション エクスプローラの Persons.vb をクリックし、[表示] メニューの [コード] をクリックします。

3. `Public Class Persons` 行の下に次の宣言コードを追加します。

VB

```
Private firstNameValue As String
Private middleNameValue As String
Private lastNameValue As String
Public Alive As Boolean
```

4. 宣言コードの下に次のプロパティ プロシージャを追加します。

VB

```
Public Property FirstName() As String
    Get
        FirstName = firstNameValue
    End Get
    Set(ByVal value As String)
        firstNameValue = value
    End Set
End Property

Public Property MiddleName() As String
    Get
        MiddleName = middleNameValue
    End Get
    Set(ByVal value As String)
        middleNameValue = value
    End Set
End Property

Public Property LastName() As String
    Get
        LastName = lastNameValue
    End Get
    Set(ByVal value As String)
        lastNameValue = value
    End Set
End Property
```

5. [ファイル] メニューの [すべてを保存] をクリックして作業を保存します。

読み取り専用プロパティと書き込み専用プロパティ

プロパティは、一度設定されると、プログラムの実行中に変更できないようになっている場合もあります。たとえば、従業員番号を表すプロパティは変更できないため、別のプログラムで読み取ることはできますが、そのプログラムで値を変更することはできません。

ReadOnly キーワードは、プロパティ値が読み取り可能で変更不可であることを指定するために使用されます。**ReadOnly** プロパティに値を割り当てようとすると、コード エディタでエラーが発生します。

読み取り専用プロパティを作成するには、次のように **Set** プロシージャを含めずに **Get** プロシージャを使用してプロパティ プロシージャを作成します。

VB

```
Private IDValue As Integer
ReadOnly Property ID() As Integer
    Get
        ID = IDValue
    End Get
```

```
End Property
```

同様に、**WriteOnly** キーワードは、プロパティ値を設定することはできますが、読み取ることはできません。たとえば、パスワード プロパティを他のプログラムで読み取ることはできません。クラス内で処理を実行する場合にプロパティ値を使用することはできますが、プライベートにしておきます。

書き込み専用プロパティを作成するには、次のように **Get** プロシージャを含めずに **Set** プロシージャを使用してプロパティを作成します。

VB

```
Private passwordValue As String
WriteOnly Property Password() As String
    Set(ByVal value As String)
        passwordValue = value
    End Set
End Property
```

ReadOnly プロパティ プロシージャおよび **WriteOnly** プロパティ プロシージャは、あるプロパティ値を取得して別の値に変換する場合にも便利です。たとえば、人の年齢について考えてみましょう。名前とは違い、年齢は時がたつにつれて変わります。年齢をクラスに代入すると、1 年後にそこから読み取ったとき、その値は正しくありません。

`Persons` クラスでは、誕生年を表す **WriteOnly** `BirthYear` プロパティ (変化しない) と、現在の年と誕生年の差を計算して値を返す **ReadOnly** `Age` プロパティの 2 つのプロパティを追加することでこの問題を回避できます。

やってみよう

ReadOnly プロパティと WriteOnly プロパティをクラスに追加するには

1. クラス モジュールの上部にあるその他の宣言の下に、次の宣言コードを追加します。

   VB

   ```
   Private birthYearValue As Integer
   ```

2. 宣言コードの下に次のプロパティ プロシージャを追加します。

   VB

   ```
   WriteOnly Property BirthYear() As Integer
       Set(ByVal value As Integer)
           birthYearValue = value
       End Set
   End Property

   ReadOnly Property Age() As String
       Get
           Age = My.Computer.Clock.LocalTime.Year - birthYearValue
       End Get
   End Property
   ```

3. [ファイル] メニューの [すべてを保存] をクリックして作業を保存します。

次の手順

このレッスンでは、プロパティと、プロパティをクラスに追加するさまざまな方法について説明しました。次のレッスンでは、メソッドをクラスに追加してアクションを実行できるようにする方法について説明します。

次のレッスン : 「クラスへのメソッドの追加」

参照
処理手順
現実世界のオブジェクトのモデリング : 初めてのクラスの作成

概念

プロパティ プロシージャとフィールド

その他の技術情報

オブジェクトによるプログラミング : クラスを使用する

クラスへのメソッドの追加

このレッスンでは、クラスにメソッドを追加して、処理を実行できるようにする方法を説明します。

前のレッスン (「詳細情報 : プロパティ、メソッド、イベントについて」) では、ほとんどのオブジェクトには実行できる処理があり、これらの処理はメソッドと呼ばれることを説明しました。「現実世界のオブジェクトのモデリング : 初めてのクラスの作成」のレッスンで作成した `Persons` クラスは人を表します。人間は、さまざまな処理を実行できます。`Persons` クラスでは、それらの処理がクラス メソッドとして表現されます。

クラス メソッド

クラスのメソッドは、クラスで宣言されている **Sub** プロシージャまたは **Function** プロシージャです。たとえば、`Account` クラスには、残高を更新する `Recalculate` という名前の **Sub** プロシージャや、最新の残高を返す `CurrentBalance` という **Function** プロシージャがあることが考えられます。これらのメソッドを宣言するコードは、次のようになります。

VB

```
Public Sub Recalculate()
    ' add code to recalculate the account.
End Sub
Public Function CurrentBalance(ByVal AccountNumber As Integer) As Double
    ' add code to return a balance.
End Function
```

ほとんどのクラス メソッドはパブリックですが、そのクラス自身のみが使用できるメソッドが必要な場合もあります。たとえば、`Persons` クラスには、その人の年齢を計算する固有の関数がある場合があります。関数を **Private** として宣言することにより、クラスの外側からその関数を認識したり呼び出したりできなくなります。

プライベート関数のコードは次のようになります。

VB

```
Private Function CalcAge(ByVal year As Integer) As Integer
    CalcAge = My.Computer.Clock.LocalTime.Year - year
End Function
```

`CalcAge` の値を計算するコードは後から変更できます。その場合、メソッドを使用する側のコードに変更を加えなくても、メソッドの動作に影響はありません。メソッドを実行するコードを隠蔽することを、カプセル化と呼びます。

`Persons` クラスで、フル ネームを返すパブリック メソッドと、年齢を計算するプライベート関数を作成してみましょう。

やってみよう

クラスにメソッドを追加するには

1. 前のレッスンで作成した `Persons` プロジェクトを開きます。このプロジェクトを保存していない場合は、先に前のレッスン (「クラスへのプロパティの追加」) に戻って、手順を実行する必要があります。
2. ソリューション エクスプローラの Persons.vb をクリックし、[表示] メニューの [コード] をクリックします。
3. プロパティ プロシージャの下に次のコードを追加します。

   VB

   ```
   Public Function FullName() As String
       If middleNameValue <> "" Then
           FullName = firstNameValue & " " & middleNameValue & " " _
                                    & lastNameValue
       Else
           FullName = firstNameValue & " " & lastNameValue
       End If
   End Function
   ```

VB

```
Private Function CalcAge(ByVal year As Integer) As Integer
    CalcAge = My.Computer.Clock.LocalTime.Year - year
End Function
```

4. `Age` プロパティ プロシージャ内のコードを次のように変更して、このプライベート関数を使用するようにします。

VB

```
ReadOnly Property Age() As String
    Get
        ' Age = My.Computer.Clock.LocalTime.Year - birthDateValue
        Age = CalcAge(birthYearValue)
    End Get
End Property
```

5. [ファイル] メニューの [すべてを保存] をクリックして変更を保存します。

次の手順

このレッスンでは、クラスにパブリック メソッドとプライベート メソッドを追加する方法を説明しました。メソッドの詳細について学習するには、「詳細情報 : オーバーロードを使用して同じメソッドの複数バージョンを作成する」に進んでください。また、次のレッスンでは、作成したクラスを使用およびテストする方法を説明します。

次のレッスン : 「クラスのテスト」

参照

処理手順

クラスへのプロパティの追加

その他の技術情報

オブジェクトによるプログラミング : クラスを使用する

詳細情報：オーバーロードを使用して同じメソッドの複数バージョンを作成する

このレッスンでは、メソッドの複数バージョンをクラスに追加する方法について説明します。

前のレッスンでは、メソッドを `Persons` クラスに追加する方法について説明しました。単一のメソッドでは不十分な場合もあります。たとえば、さまざまな状況で異なるデータ型をメソッドに渡す必要がある場合や、結果として別の形式を返す場合などです。

オーバーロードと呼ばれる技法を使用して、複数のバージョンのメソッドを作成できます。1 つのクラスに、名前が同じで引数のセットが異なる複数のメソッドがある場合、メソッドはオーバーロードされます。

オーバーロード

オーバーロードされたメソッドを作成するには、2 つ以上の **Sub** プロシージャまたは **Function** プロシージャをクラスに追加して、それぞれに同じ名前を付けます。プロシージャ宣言では、各プロシージャの引数のセットが異なっている必要があります。引数のセットが同じ場合、エラーが発生します。

2 つのオーバーロードがあるメソッドを次に示します。一方は **String**、もう一方は **Integer** を引数として使用します。

VB

```
Public Sub TestFunction(ByVal input As String)
    MsgBox(input)
End Sub
Public Sub TestFunction(ByVal input As Integer)
    MsgBox(CStr(input))
End Sub
```

このメソッドをコードから呼び出して文字列を渡すと、1 つ目のオーバーロードが実行され、メッセージ ボックスに文字列が表示されます。メソッドに数値を渡すと、2 つ目のオーバーロードが実行され、数値が文字列に変換された後、メッセージ ボックスに表示されます。

必要な数だけオーバーロードを作成して、各オーバーロードに異なる数の引数を含めることができます。

`Persons` クラスで、任意の人のミドル ネームの頭文字を返すには、2 つのオーバーロードがあるメソッドを追加します。一方のオーバーロードは頭文字のみ、もう一方は頭文字の後にピリオドが付きます。

やってみよう

オーバーロードされたメソッドを作成するには

1. 前のレッスンで作成した `Persons` プロジェクトを開きます。プロジェクトを保存しなかった場合は、前のレッスン「クラスへのメソッドの追加」に戻り、手順を完了します。
2. ソリューション エクスプローラの Persons.vb をクリックし、[表示] メニューの [コード] をクリックします。
3. 既存のメソッドの下に次のコードを追加します。

   VB

   ```
   Public Function MiddleInitial() As String
       MiddleInitial = Left$(middleNameValue, 1)
   End Function

   Public Function MiddleInitial(ByVal period As Boolean) As String
       MiddleInitial = Left$(middleNameValue, 1) & "."
   End Function
   ```

4. [ファイル] メニューの [すべてを保存] をクリックして変更を保存します。

次の手順

このレッスンでは、オーバーロードされたメソッドを作成する方法について説明しました。次のレッスンでは、作成したクラスをテスト プロジェクトで使用する方法について説明します。

次のレッスン : 「クラスのテスト」

参照

処理手順

クラスへのメソッドの追加

概念

プロシージャのオーバーロードに関する注意事項

その他の技術情報

オブジェクトによるプログラミング : クラスを使用する

Visual Basic ガイド ツアー

クラスのテスト

このレッスンでは、クラスをテストするために、クラスのインスタンスを作成する方法について説明します。

これまでのレッスンでは、Persons クラスを作成して、プロパティとメソッドを定義しました。これで、コードの追加は完了したため、Persons クラスを使用して、期待どおりに動作することを確認します。

クラスのインスタンスの作成

気付いていないかもしれませんが、以前の多くのレッスンで既にクラスを使用しています。フォームとコントロールは、実際にはクラスです。Button コントロールをフォームにドラッグすると、実際には Button クラスのインスタンスを作成していることになります。

どのクラスも、**New** キーワードによる宣言を使用して、コード内でインスタンスを生成できます。たとえば、**Button** クラスの新しいインスタンスを作成するには、次のコードを追加します。

VB

```
Dim aButton As New Button
```

Persons クラスを使用してテストするには、テスト プロジェクトを作成してから、クラス モジュールに参照を追加する必要があります。

やってみよう

クラスのテスト プロジェクトを作成するには

1. 前のレッスンで作成した Persons プロジェクトを開きます。このプロジェクトを保存していない場合は、先に前のレッスン (「クラスへのメソッドの追加」) に戻って、手順を実行する必要があります。
2. [ファイル] メニューの [追加] をポイントし、[新しいプロジェクト] をクリックします。
3. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
4. [プロジェクト名] ボックスに「PersonsTest」と入力し、[OK] をクリックします。
5. ソリューション エクスプローラで PersonsTest プロジェクトをクリックし、[プロジェクト] メニューの [スタートアップ プロジェクトに設定] をクリックします。
6. ソリューション エクスプローラで PersonsTest プロジェクトをクリックし、[プロジェクト] メニューの [参照の追加] をクリックします。

   [参照の追加] ダイアログ ボックスが表示されます。
7. [プロジェクト] タブをクリックして [Persons] をクリックし、[OK] をクリックします。
8. フォームをダブルクリックしてコード エディタを開き、Public Class Form1 行のすぐ後に次の宣言を入力します。

   VB

   ```
   Dim person1 As New Persons.Persons
   ```

   これは、Persons クラスの新しいインスタンスを宣言します。Persons を 2 回入力する必要があるのは、1 つ目のインスタンスは Persons.vb クラス モジュール、2 つ目のインスタンスはそのモジュール内の Persons クラスであるためです。
9. [ファイル] メニューの [すべてを保存] をクリックします。

クラスのテスト

次の手順では、Persons クラスを使用するユーザー インターフェイスとコードを追加します。各プロパティ (読み取り専用の Age プロパティを除く) の値をユーザーが入力するためのテキスト ボックス、Alive フィールドのチェック ボックス、および各パブリック メソッドをテストするためのボタンを追加します。

やってみよう

クラスをテストするには

1. ソリューション エクスプローラで Form1 を選択して、[表示] メニューの [デザイナ] をクリックします。

2. ツールボックス から、4 つの **TextBox** コントロール、1 つの **CheckBox**、および 2 つの **Button** コントロールをフォームにドラッグします。
3. 最初の **Button** コントロールを選択し、[プロパティ] ウィンドウで **Text** プロパティを `Update` に設定します。
4. 2 つ目の **Button** コントロールを選択し、[プロパティ] ウィンドウで Text プロパティを `Full Name` に設定します。
5. 1 つ目のボタン (Update) をダブルクリックしてコード エディタを開き、**Button1_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   With person1
       .FirstName = Textbox1.Text
       .MiddleName = Textbox2.Text
       .LastName = Textbox3.Text
       .BirthYear = Textbox4.Text
       .Alive = CheckBox1.Checked
   End With
   ```

   入力すると、`Persons` クラスのすべてのメンバを含む一覧が表示されます。クラスは参照として追加されたため、IntelliSense は、その他のクラスの場合と同じように、クラスに関する情報を表示します。

6. **Button2_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   ' Test the FullName method.
   MsgBox(person1.FullName)

   ' test the Age property and CalcAge method.
   MsgBox(CStr(person1.Age) & " years old")

   ' Test the Alive property.
   If person1.Alive = True Then
       MsgBox(person1.FirstName & " is alive")
   Else
       MsgBox(person1.FirstName & " is no longer with us")
   End If
   ```

7. F5 キーを押してプロジェクトを実行し、フォームを表示します。
   a. 1 つ目のテキスト ボックスに名を入力します。
   b. 2 つ目のテキスト ボックスにミドル ネームを入力します。
   c. 3 つ目のテキスト ボックスに姓を入力します。
   d. 4 つ目のテキスト ボックスに 4 桁の誕生年 (つまり 1983) を入力します。
   e. まだ生きている場合は、チェック ボックスをオンにします。
8. [Update] ボタンをクリックしてクラスのプロパティを設定し、[Full Name] ボタンをクリックします。

   3 つのメッセージ ボックスが表示され、フル ネーム、年齢、およびステータスが表示されます。
9. [ファイル] メニューの [すべてを保存] をクリックします。

オーバーロードされたメソッドのテスト

オプションのレッスン「詳細情報 : オーバーロードを使用して同じメソッドの複数バージョンを作成する」を完了した場合、`Persons` クラスに追加した、オーバーロードされたメソッドをテストすることもできます。レッスンを完了していない場合は、戻って完了することも、次の手順に進むこともできます。

やってみよう

オーバーロードされたメソッドをテストするには

1. ソリューション エクスプローラで `Form1` を選択して、[表示] メニューの [デザイナ] をクリックします。
2. ツールボックスから、フォームにさらに 2 つの **Button** コントロールをドラッグします。
3. 3 つ目の **Button** コントロールを選択し、[プロパティ] ウィンドウで **Text** プロパティを `With` に設定します。
4. 4 つ目の **Button** コントロールを選択し、[プロパティ] ウィンドウで **Text** プロパティを `Without` に設定します。
5. 1 つ目のボタン (With) をダブルクリックしてコード エディタを開き、**Button3_Click** イベント ハンドラに次のコードを入力します。

   VB

   ```
   MsgBox(person1.FirstName & " " & person1.MiddleInitial(True) & _
                              " " & person1.LastName)
   ```

   入力すると、`Persons` クラスのすべてのメンバを含む一覧が表示されます。クラスは参照として追加されたため、IntelliSense は、その他のクラスの場合と同じように、クラスに関する情報を表示します。

6. **Button4_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   MsgBox(person1.FirstName & " " & person1.MiddleInitial & _
                              " " & person1.LastName)
   ```

7. F5 キーを押してプロジェクトを実行し、フォームを表示します。
   a. 1 つ目のテキスト ボックスに名を入力します。
   b. 2 つ目のテキスト ボックスにミドル ネームを入力します。
   c. 3 つ目のテキスト ボックスに姓を入力します。
   d. 4 つ目のテキスト ボックスに 4 桁の誕生年 (つまり 1983) を入力します。
   e. まだ生きている場合は、チェック ボックスをオンにします。
8. [Update] ボタンをクリックしてクラスのプロパティを設定し、[With] ボタンをクリックします。

   メッセージ ボックスが表示され、ミドル ネームの頭文字の後にピリオドを付けた名前が表示されます。

9. [Without] ボタンをクリックします。

   メッセージ ボックスが表示され、ミドル ネームの頭文字の後にピリオドを付けない名前が表示されます。

10. [ファイル] メニューの [すべてを保存] をクリックします。

次の手順

このレッスンでは、テスト プロジェクトを作成し、そのテスト プロジェクトを使用してクラスのプロパティおよびメソッドをテストする方法について説明しました。次のレッスンでは、継承を使用して、既存のクラスを基にクラスを作成する方法を説明します。

次のレッスン : 「既存のクラスを基にしたクラスの構築 : 継承を使用する」

参照
処理手順
クラスへのメソッドの追加
詳細情報 : オーバーロードを使用して同じメソッドの複数バージョンを作成する

既存のクラスを基にしたクラスの構築：継承を使用する

このレッスンでは、継承を使用して、既存のクラスを基にクラスを作成する方法について説明します。

現実に存在する多くのオブジェクトには、共通する属性や動作があります。たとえば、すべての自動車は車輪とエンジンを持ち、走行と (願わくば) 停止ができます。しかし、一部の自動車には他とは異なる属性があります。たとえば、コンバーチブル車には開放できる屋根があり、その屋根は電子制御または手動によって下げることができます。

自動車を表すオブジェクトを作成した場合、すべての共通する属性や動作を表すプロパティやメソッドは定義する必要がありますが、開放可能な屋根などの属性は、すべての自動車に当てはまる属性ではないため、追加する必要はありません。

継承を使用すると、自動車クラスから派生した "convertible" クラスを作成できます。このクラスは、自動車クラスが持つすべての属性を継承し、さらにコンバーチブル車に特有の属性や動作も備えることができます。

既存のクラスから継承する

Inherits ステートメントは、既存のクラス (基本クラス) に基づいて新しいクラス (派生クラス) を宣言するときに使用します。派生クラスは、基本クラスで定義されているすべてのプロパティ、メソッド、イベント、フィールド、および定数を継承します。次のコードは、派生クラスの宣言を示しています。

VB

```
Class DerivedClass
    Inherits BaseClass
End Class
```

新しいクラスである `DerivedClass` は、インスタンス化できるようになり、そのプロパティやメソッドへは `BaseClass` と同様にアクセスできます。また、新しいクラスに特有の新しいプロパティやメソッドを追加することもできます。たとえば、前のレッスンで作成した `Persons` クラスを見てみましょう。

野球選手を表すクラスを作成すると仮定します。野球選手は、`Persons` クラスで定義されたすべての属性を備えていますが、背番号やポジションなど固有の属性も持っています。この場合、そうしたプロパティを `Persons` クラスに追加するのではなく、`Persons` から継承する新しい派生クラスを作成して、そのクラスに新しいプロパティを追加します。

やってみよう

派生クラスを作成するには

1. 前のレッスンで作成した `Persons` プロジェクトを開きます。このプロジェクトを保存していない場合は、「クラスのテスト」に戻って手順を実行します。
2. ソリューション エクスプローラの Persons プロジェクトのノードを選択します。
3. [プロジェクト] メニューの [クラスの追加] をクリックします。
4. [新しい項目の追加] ダイアログ ボックスの [ファイル名] ボックスに「`Players`」と入力し、[追加] をクリックします。

   新しいクラス モジュールがプロジェクトに追加されます。
5. コード エディタで、`Public Class Players` 行のすぐ後に次のコードを追加します。

   VB

   ```
   Inherits Persons
   ```
6. 次のコードを追加して、`2` つの新しいプロパティを定義します。

   VB

   ```
   Private numberValue As Integer
   Private positionValue As String
   Public Property Number() As Integer
       Get
           Number = numberValue
   ```

```
        End Get
        Set(ByVal value As Integer)
            numberValue = value
        End Set
    End Property
    Public Property Position() As String
        Get
            Position = positionValue
        End Get
        Set(ByVal value As String)
            positionValue = value
        End Set
    End Property
```

7. [ファイル] メニューの [すべてを保存] をクリックします。

Players クラスをテストする

これで、Persons クラスから派生した Players クラスが作成されました。次の手順では、Players クラスをテストするための新しいプログラムを作成します。

クラスのテスト プロジェクトを作成するには

1. [ファイル] メニューの [追加] をポイントし、[新しいプロジェクト] をクリックします。
2. [新しいプロジェクトの追加] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「PlayerTest」と入力し、[OK] をクリックします。
4. 新しい Windows フォーム プロジェクトがソリューション エクスプローラに追加され、新しいフォームが表示されます。
5. ソリューション エクスプローラで PlayerTest プロジェクトをクリックし、[プロジェクト] メニューの [スタートアップ プロジェクトに設定] をクリックします。
6. ソリューション エクスプローラで PlayerTest プロジェクトをクリックし、[プロジェクト] メニューの [参照の追加] をクリックします。

   [参照の追加] ダイアログ ボックスが開きます。
7. [プロジェクト] タブをクリックして [Persons] をクリックし、[OK] をクリックします。
8. フォームをダブルクリックしてコード エディタを開き、Public Class Form1 行のすぐ後に次の宣言を入力します。

   VB

```
Dim player1 As New Persons.Players
Dim player2 As New Persons.Players
```

9. これにより、Players クラスの 2 つのインスタンスが宣言されます。
10. [ファイル] メニューの [すべてを保存] をクリックします。

派生クラスをテストするには

1. ソリューション エクスプローラで PlayerTest プロジェクトの Form1 をクリックし、[表示] メニューの [コード] をクリックします。
2. コード エディタで、**Form1_Load** イベント プロシージャに次のコードを追加します。

   VB

```
With player1
    .FirstName = "Andrew"
    .LastName = "Cencini"
    .Number = 43
    .Position = "Shortstop"
End With
With player2
    .FirstName = "Robert"
    .LastName = "Lyon"
    .Number = 11
    .Position = "Catcher"
End With
```

3. ソリューション エクスプローラで PlayerTest プロジェクトの Form1 をクリックし、[表示] メニューの [デザイナ] をクリックします。
4. ツールボックスから、フォームに 2 つの Button コントロールをドラッグします。
5. 1 つ目の Button コントロールを選択し、[プロパティ] ウィンドウで **Text** プロパティを At Bat に設定します。
6. 2 つ目の Button コントロールを選択し、[プロパティ] ウィンドウで **Text** プロパティを On Deck に設定します。
7. 1 つ目のボタン (At Bat) をダブルクリックしてコード エディタを開き、**Button1_Click** イベント ハンドラに次のコードを入力します。

   VB

   ```
   MsgBox(player1.Position & " " & player1.FullName & ", #" & _
     CStr(player1.Number) & " is now at bat.")
   ```

   ここでは、基本クラスである Persons から継承した FullName メソッドを使用しています。
8. **Button2_Click** イベント ハンドラに次のコードを追加します。

   VB

   ```
   MsgBox(player2.Position & " " & player2.FullName & ", #" & _
     CStr(player2.Number) & " is on deck.")
   ```
9. F5 キーを押してプログラムを実行します。各ボタンをクリックして結果を確認します。
10. [ファイル] メニューの [すべてを保存] をクリックします。

次の手順

このレッスンでは、既存のクラスから継承する方法、および派生クラスを拡張する方法を説明しました。継承の詳細について学習するには、「詳細情報 : メンバをオーバーライドする」に進んでください。また、次のレッスンでは、コレクションについて学びます。

次のレッスン :「オブジェクトの追跡 : コレクションを使用して複数のオブジェクトを管理する」

参照

概念

継承の基本

詳細情報：メンバをオーバーライドする

このレッスンでは、派生クラスのメンバをオーバーライドする方法について説明します。

前のレッスンでは、基本クラスから継承し、新しいプロパティを使用して派生クラスを拡張する方法について説明しました。派生クラスに新しいプロパティまたはメソッドを追加する以外に、既存のプロパティまたはメソッドの動作を変更、つまりオーバーライドすることもできます。

たとえば、`StartEngine` メソッドを含む `Car` クラスから派生した `Truck` クラスを作成できます。Truck がディーゼル エンジンを使用する場合、エンジンをかけるプロセスは、Car の場合とは異なります。この場合、Truck に合わせて `StartEngine` メソッドをオーバーライドすることになります。

プロパティとメソッドのオーバーライド

既定では、クラスのプロパティとメソッドはオーバーライドできません。派生クラスでプロパティまたはメソッドをオーバーライドできるようにするには、**Overridable** キーワードを使用してプロパティまたはメソッドを宣言して、オーバーライド可能としてマークする必要があります。

```
Public Overridable Property EngineType As String

Public Overridable Sub StartEngine(ByVal EngineType As String)
```

基本クラスから継承する場合、**Overridable** としてマークされたプロパティとメソッドはそのまま使用することも、**Overrides** キーワードを使用して宣言することで要件に合わせて変更することもできます。

```
Public Overrides Property EngineType As String

Public Overrides Sub StartEngine(ByVal EngineType As String)
```

前のレッスンで作成した `Players` クラスでは、`FullName` メソッドをオーバーライドして、選手の背番号も名前に含め、さらにミドル ネームを返すコードを削除できます。

やってみよう

FullName メソッドをオーバーライドするには

1. 前のレッスンで作成した `Persons` プロジェクトを開きます。プロジェクトを保存しなかった場合は、前のレッスン「既存のクラスを基にしたクラスの構築：継承を使用する」に戻り、手順を完了します。
2. ソリューション エクスプローラで Persons.vb ノードをクリックし、[表示] メニューの [コード] をクリックします。
3. コード エディタで、`FullName` メソッドの宣言を次のように変更します。

   VB

   ```
   Public Overridable Function FullName() As String
   ```

4. ソリューション エクスプローラで、Players.vb ノードをクリックし、[表示] メニューの [コード] をクリックします。
5. コード エディタで、次のコードをクラスに追加します。

   VB

   ```
   Public Overrides Function FullName() As String
       FullName = FirstName & " " & LastName & ", #" & numberValue
   End Function
   ```

6. ソリューション エクスプローラで、PlayerTest プロジェクトの Form1.vb ノードを選択して、[表示] メニューの [コード] をクリックします。
7. コード エディタで、`Button1_Click` イベント コードを次のように変更します。

   VB

   ```
   MsgBox(player1.Position & " " & player1.FullName & _
         " is now at bat.")
   ```

8. `Button2_Click` イベント コードを次のように変更します。

VB

```
MsgBox(player2.Position & " " & player2.FullName & _
       " is on deck.")
```

9. F5 キーを押してプログラムを実行し、各ボタンをクリックして結果を確認します。

   オーバーライドされた `FullName` メソッドを使用していても、結果は前と同じです。

10. [ファイル] メニューの [すべてを保存] をクリックします。

次の手順

このレッスンでは、メソッドをオーバーライドする方法について説明しました。次のレッスンでは、コレクションを使用して類似したオブジェクトのグループを管理する方法について説明します。

次のレッスン : 「オブジェクトの追跡 : コレクションを使用して複数のオブジェクトを管理する」

参照

処理手順

既存のクラスを基にしたクラスの構築 : 継承を使用する

関連項目

Overridable

Overrides

オブジェクトの追跡：コレクションを使用して複数のオブジェクトを管理する

このレッスンでは、コレクションを使用してオブジェクトのグループを管理する方法について説明します。

前のレッスンでは、配列を使用して変数のグループを管理する方法を説明しました。配列を使用してオブジェクトのグループを管理することもできますが、Visual Basic には、類似したオブジェクトのグループの格納や取得に使用できる、コレクションと呼ばれる特別な種類のオブジェクトもあります。

配列と同様に、Collection オブジェクト内の各項目には、項目の識別に使用できるインデックスがあります。また、**Collection** オブジェクト内の各項目には、項目の識別に使用できるキーと呼ばれる **String** 値もあります。キーを使用する利点は、項目のインデックスを覚える必要がないという点です。インデックスを覚えなくても、わかりやすい名前を使用して項目を参照できます。

コレクションの作成

コレクションは、プログラムが同じクラスの複数のインスタンスを使用する場合に便利です。たとえば、前のレッスンで作成した `Players` クラスを見てみましょう。野球チームを表すためには、複数の `Players` オブジェクトが必要となる可能性が大です。

コレクションを作成するには、次の宣言に示すように、まず **Collection** オブジェクトのインスタンスを作成します。

VB

```
Dim baseballTeam As New Collection
```

Collection オブジェクトを作成した後、そこに項目を追加する場合には **Add** メソッド、項目を削除する場合には **Remove** メソッドを使用します。項目を追加する場合は、まず追加する項目を指定して、次にキーとして使用する **String** 値を指定します。

VB

```
baseballTeam.Add(playerObject, "Player's Name")
```

項目を削除する場合は、キーを使用して削除する項目を指定します。

VB

```
baseballTeam.Remove("Player's Name")
```

次の手順では、2 つの新しい `Players` オブジェクトを追加して `team` コレクションを作成し、`Position` プロパティをキーとして使用して、`Players` オブジェクトをコレクションに追加します。

やってみよう

オブジェクトのコレクションを作成するには

1. 前のレッスンで作成した `Persons` プロジェクトを開きます。プロジェクトを保存しなかった場合は、前のレッスン「既存のクラスを基にしたクラスの構築：継承を使用する」に戻り、手順を完了します。
2. ソリューション エクスプローラで、[PlayerTest] プロジェクトの [Form1.vb] ノードを選択して、[表示] メニューの [コード] をクリックします。
3. コード エディタで、宣言セクション (`player2` の宣言の下) に次のコードを追加します。

   VB

   ```
   Dim player3 As New Persons.Players
   Dim player4 As New Persons.Players
   Dim team As New Collection
   ```

4. `Form1_Load` イベント プロシージャに次のコードを追加します。

   VB

   ```
   With player3
   ```

```
        .FirstName = "Eduardo"
        .LastName = "Saavedra"
        .Number = 52
        .Position = "First Base"
    End With

    With player4
        .FirstName = "Karl"
        .LastName = "Jablonski"
        .Number = 22
        .Position = "Pitcher"
    End With

    team.Add(player1, player1.Position)
    team.Add(player2, player2.Position)
    team.Add(player3, player3.Position)
    team.Add(player4, player4.Position)
```

5. ソリューション エクスプローラの [PlayerTest] プロジェクトで、[Form1.vb] ノードを選択します。次に、[表示] メニューの [デザイナ] をクリックします。
6. ツールボックスから、フォームに ComboBox コントロールをドラッグします。
7. [プロパティ] ウィンドウで、Items プロパティを選択し、[...] ボタンをクリックします。
8. 文字列コレクション エディタで次のように入力して、[OK] をクリックします。

   ```
   Catcher

   First Base

   Pitcher

   Shortstop
   ```

9. ComboBox コントロールをダブルクリックしてコード エディタを開き、`ComboBox1_SelectedIndexChanged` イベント ハンドラに次のコードを入力します。

   ```
   Dim SelectedPlayer As Persons.Players
   SelectedPlayer = team(ComboBox1.SelectedItem)
   MsgBox("Playing " & ComboBox1.SelectedItem & " is " & _
   SelectedPlayer.FullName & "!")
   ```

10. F5 キーを押してプログラムを実行します。ドロップダウン リストからポジションを選択します。そのポジションの選手がメッセージ ボックスに表示されます。

次の手順

このレッスンでは、**Collection** オブジェクトを使用してオブジェクトのグループを管理する方法について説明しました。ここで、「詳細情報 : コレクションで For Each...Next ループを使用する」に進んでコレクションの詳細を参照するか、次の一連のレッスンに進んで独自のコントロールを作成する方法を参照するかを選択できます。

次のレッスン : 「表示されるオブジェクト : 初めてのユーザー コントロールの作成」

参照

処理手順

詳細情報 : コレクションで For Each...Next ループを使用する
詳細情報 : メンバをオーバーライドする
既存のクラスを基にしたクラスの構築 : 継承を使用する

概念

Visual Basic のコレクション クラス

その他の技術情報

表示されるオブジェクト : 初めてのユーザー コントロールの作成

詳細情報：コレクションで For Each...Next ループを使用する

このレッスンでは、**For Each...Next** ループを使用して、コレクションをループ処理する方法を説明します。

前のレッスンでは、**For...Next** ループを使用して、コード ブロックを指定した回数だけ実行する方法を説明しました。Visual Basic のコレクション オブジェクトは、**For Each...Next** ループという特殊なループをサポートしています。これは、あらかじめ設定した回数だけブロックを実行するのではなく、コレクション内の各要素に対してコード ブロックを実行するループです。

For Each...Next ループを追加する

前のレッスンでは、チーム コレクションに含まれる `Players` オブジェクトの `Position` プロパティの値を、手動で ComboBox コントロールに追加しました。この方法も 1 つの例ではありますが、最良の方法ではありません。新しい選手を追加するたびに、**ComboBox** コントロールの **Items** コレクションも更新する必要があるからです。

これを処理するためのより良い方法は、**For Each...Next** ループを使用して `team` コレクションをループ処理することにより、`Position` の値を **Items** コレクションに追加することです。

For...Next ループでは、まずカウンタ変数を宣言する必要がありました。これに対して **For Each...Next** ループでは、まずオブジェクト変数を宣言する必要があります。**For Each...Next** ループのコード例を次に示します。

VB

```
Dim player As Persons.Players
For Each player In team
    ComboBox1.Items.Add(player.Position)
Next
```

この場合、`Players` の数がどれだけ多くても、チーム コレクションに含まれる各 `Players` について **ComboBox1.Items.Add** メソッドが 1 回ずつ実行され、`Position` の値が一覧に追加されます。

やってみよう

コレクションをループ処理するには

1. 前のレッスンで使用した `Persons` プロジェクトを開きます。前のレッスンを終了していない場合は、「オブジェクトの追跡 : コレクションを使用して複数のオブジェクトを管理する」に戻って手順を完了します。
2. ソリューション エクスプローラで [PlayerTest] プロジェクトの [Form1.vb] ノードをクリックし、[表示] メニューの [デザイナ] をクリックします。
3. **ComboBox** コントロールを選択します。次に、[プロパティ] ウィンドウで、**Items** プロパティを選択し、[...] ボタンをクリックします。
4. 文字列コレクション エディタ1 で、既存の 4 つのエントリを削除して [OK] をクリックします。
5. フォームをダブルクリックしてコード エディタを開きます。
6. コード エディタで、宣言セクション (`team` の宣言の下) に次のコードを追加します。

   VB

   ```
   Dim player As Persons.Players
   For Each player In team
       ComboBox1.Items.Add(player.Position)
   Next
   ```

7. F5 キーを押してプログラムを実行します。ドロップダウン リストでポジションを選択します。そのポジションの選手がメッセージ ボックスに表示されます。

次の手順

このレッスンでは、コレクションで **For Each...Next** ループを使用しました。次のレッスンからは、別の種類のオブジェクトであるユーザー コントロールを作成します。

次のレッスン : 「表示されるオブジェクト : 初めてのユーザー コントロールの作成」

参照

処理手順

オブジェクトの追跡 : コレクションを使用して複数のオブジェクトを管理する

関連項目

For Each...Next ステートメント (Visual Basic)

その他の技術情報

オブジェクトによるプログラミング : クラスを使用する

表示されるオブジェクト : 初めてのユーザー コントロールの作成

ここまでの一連のレッスンでは、オブジェクトの青写真であるクラスを扱う方法について説明しました。作成したクラスは他のプログラムで再利用できるため、同じコードを何度も繰り返し作成する必要がありません。

コントロールも、複数のプロジェクトで再利用できるクラスです。同じユーザー インターフェイスを何度も繰り返してデザインするという経験は、おそらくだれでもあります。たとえば、姓と名を入力するための **TextBox** コントロールを追加した後で、両方を組み合わせてフル ネームを作成するコードを追加するなどの作業です。そうした余分な手間を省くことができれば楽になります。

そのような場合には、ユーザー コントロールを利用できます。ユーザー コントロールとは、可視のオブジェクトを作成するためのクラスだと考えることができます。つまり、Visual Basic 2005 に標準装備されているコントロールと同様に再利用できる、独自のカスタム コントロールです。大半のユーザー コントロールは複合コントロールです。つまり、1 つまたは複数の Visual Basic 2005 標準コントロールで構成されるコントロールです。

以降のレッスンでは、他のプログラムで再利用できる複合ユーザー コントロールを作成する方法を説明します。

このセクションの内容

関連するセクション

詳細 : 名前付き値を持つプロパティの追加

詳細情報 : ユーザー コントロールのカスタマイズ

ユーザー コントロール デザイナについて

このレッスンでは、ユーザー コントロール デザイナを使用して独自のコントロールを作成する方法について説明します。

ここまでの一連のレッスンでは、クラス ライブラリ プロジェクトを使用してクラスを作成する方法について説明しました。ユーザー コントロールとは、画面に表示されるクラスにすぎません。ユーザー コントロールは、Visual Basic 2005 に用意されている標準コントロールとまったく同様に、デザイン時にフォーム上に配置でき、プログラムの実行時に表示されます。

プログラムをデザインするときに、コントロールを配置してその見た目を決定する場所がフォーム デザイナです。また、ユーザー コントロール用のデザイナもあります。それがユーザー コントロール デザイナです。開発者はこれを使用して、コントロールの見た目を決定できます。

ユーザー コントロールの作成

ユーザー コントロールは、他のクラスと似ていますが、ツールボックスに配置でき、フォーム上に表示できるという追加的な機能を持ちます。クラス モジュールにはコードしかないのに対し、ユーザー コントロール モジュールにはコードとデザイナの両方があります。ユーザー コントロール デザイナはフォーム デザイナと似ており、ユーザー コントロールの外観と動作を制御するためのプロパティを持ちます。

ユーザー コントロールを作成する方法は、使用している Visual Basic のバージョンによって多少異なります。Visual Basic 2005 の場合は Windows コントロール ライブラリというプロジェクトの種類があるのに対し、Visual Basic Express Edition の場合は、まずクラス ライブラリ プロジェクトを作成してから、ユーザー コントロール テンプレートを追加する必要があります。

やってみよう

Visual Basic Express Edition を使用してユーザー コントロールを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[クラス ライブラリ] をクリックし、[OK] をクリックします。
3. [プロジェクト] メニューの [ユーザー コントロールの追加] をクリックします。
4. [新しい項目の追加] ダイアログ ボックスで、[ユーザー コントロール] をクリックします。
5. [ファイル名] ボックスに「`NamesControl`」と入力し、[追加] をクリックします。

   新しいユーザー コントロール テンプレートがプロジェクトに追加され、ユーザー コントロール デザイナが開きます。
6. ソリューション エクスプローラで、`Class1.vb` を右クリックし、[削除] をクリックして、[OK] をクリックします。
7. [ファイル] メニューの [すべてを保存] をクリックします。
8. [プロジェクトの保存] ダイアログ ボックスで、「`NamesUserControl`」と入力し、[保存] をクリックします。

Visual Studio 2005 を使用してユーザー コントロールを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows コントロール ライブラリ] をクリックします。
3. [プロジェクト名] ボックスに「`NamesControl`」と入力し、[OK] をクリックします。

   新しいユーザー コントロール テンプレートがプロジェクトに追加され、ユーザー コントロール デザイナが開きます。
4. [ファイル] メニューの [すべてを保存] をクリックします。
5. [プロジェクトの保存] ダイアログ ボックスで、「`NamesUserControl`」と入力し、[保存] をクリックします。

次の手順

このレッスンでは、ユーザー コントロールを持つプロジェクトを作成する方法と、ユーザー コントロール デザイナを表示する方法について説明しました。ただし、空のコントロールはほとんど役に立ちません。次のレッスンでは、ユーザー コントロールにコントロールを追加してそのユーザー インターフェイスを作成する方法を説明します。

次のレッスン : 「ユーザー コントロールへのコントロールの追加」

参照

その他の技術情報

表示されるオブジェクト : 初めてのユーザー コントロールの作成
オブジェクトによるプログラミング : クラスを使用する

ユーザー コントロールへのコントロールの追加

このレッスンでは、コントロールを追加して複合ユーザー コントロールを作成する方法を説明します。

ユーザー コントロールのデザイン

先にも述べたように、ユーザー コントロールの大多数が複合コントロール、つまり 1 つ以上の標準 Windows フォーム コントロールを組み合わせたコントロールです。コントロールは、フォームをデザインする場合と同様、ツールボックスからユーザー コントロール デザイナにドラッグすることによって、ユーザー コントロール テンプレートに追加できます。

コントロールを追加したら、デザイナでサイズ変更や移動を行ったり、[プロパティ] ウィンドウでプロパティを設定したりできます。

たとえば、Label コントロールを追加してフル ネームを表示し、3 つの TextBox コントロールを追加してファースト ネーム、ラスト ネーム、およびミドル ネームを表示することが可能です。

やってみよう

ユーザー コントロールにコントロールを追加するには

1. 前のレッスンで作成した `NamesUserControl` プロジェクトを開きます。プロジェクトを保存しなかった場合は、まず前のレッスン「ユーザー コントロール デザイナについて」に戻って、そのレッスンの手順を完了する必要があります。
2. ソリューション エクスプローラの NamesControl.vb をクリックし、[表示] メニューの [デザイナ] をクリックします。
3. ツールボックス から、デザイナに Label コントロールをドラッグします。

ヒント
ウィンドウを開いたままにしている場合は、ツールボックスを使用する方が簡単です。これは、押しピンのように表示されている [自動的に隠す] アイコンをクリックすることで実行できます。

4. [プロパティ] ウィンドウで、**Name** プロパティを `FullName` に変更します。
5. ツールボックス から、デザイナに 3 つの Textbox コントロールをドラッグします。コントロールを好きな位置に配置してください。
6. [プロパティ] ウィンドウで、**Name** プロパティを `FirstName`、`MiddleName`、および `LastName` に変更します。
7. [ファイル] メニューの [すべてを保存] をクリックして変更を保存します。

次の手順

このレッスンでは、コントロールをユーザー コントロール デザイナに追加して、プロパティを設定する方法について説明しました。これで、実際に有用なタスクを実行するように、コントロールを追加できます。次のレッスンでは、ユーザー コントロールがファースト ネーム、ミドル ネーム、ラスト ネームを表示し、また、新しいプロパティを公開するようにコードを作成する方法を説明します。

次のレッスン : 「ユーザー コントロールへのコードの追加」

参照

処理手順

ユーザー コントロール デザイナについて

その他の技術情報

表示されるオブジェクト : 初めてのユーザー コントロールの作成

オブジェクトによるプログラミング : クラスを使用する

Visual Basic ガイド ツアー

ユーザー コントロールへのコードの追加

このレッスンでは、ユーザー コントロールにコードを追加して、フル ネームの表示と新しいプロパティの公開を行う方法について説明します。

ユーザー コントロールにも、標準コントロールと同じように、プロパティ、メソッド、およびイベントがあります。コントロールのイベントを処理するコードの記述や、コントロールのユーザーにどのプロパティを公開するかの判断は、コントロールの開発者が行います。

ユーザー コントロールのイベント処理

ユーザー コントロールの利便性を高めるためには、コントロールのイベントを処理するコードを記述する必要があります。ユーザー コントロールのイベント処理プロシージャを記述するのは、フォームまたはコントロールのイベント処理プロシージャを記述するのと変わりません。

この例では、**TextChanged** イベント ハンドラを使用して、`FirstName`、`MiddleName`、および `LastName` の各ボックスへの入力時に、その内容で `FullName` ラベルを更新するイベント プロシージャを記述します。

やってみよう

ユーザー コントロールにコードを追加するには

1. 前のレッスンで作成した `NamesUserControl` プロジェクトを開きます。プロジェクトを保存しなかった場合は、まず前のレッスン「ユーザー コントロール デザイナについて」に戻って、そのレッスンの手順を完了する必要があります。
2. ソリューション エクスプローラの NamesControl.vb をクリックし、[表示] メニューの [コード] をクリックします。
3. コード エディタで、**FirstName_TextChanged** イベント ハンドラに次のコードを追加します。

```
Private Sub FirstName_TextChanged(ByVal sender As System.Object, ByVal e As System.Eve
ntArgs) Handles FirstName.TextChanged, MiddleName.TextChanged, LastName.TextChanged
    ' Display the contents of the three text boxes in the label.
    FullName.Text = FirstName.Text & " " & MiddleName.Text & " " & LastName.Text
End Sub
```

4. F5 キーを押してプログラムを実行します。[ユーザー コントロール テスト コンテナ] が開き、ユーザー コントロールが表示されます。
5. 名、ミドル ネーム、および姓を 3 つのテキスト ボックスに入力します。入力時に、`FullName` ラベルに名前が表示されます。

上で入力したコードを見ると、宣言の **Handles** 句によって、3 つの **TextBox** コントロールすべての **TextChanged** イベントが処理されることがわかります。どのテキスト ボックスを最初に入力した場合でも、入力に合わせて `FullName` ラベルが常に更新されます。

ユーザー コントロールのプロパティの公開

標準コントロールでは、コントロールのデザイン時と実行時にプロパティの値を設定および取得できます。ユーザー コントロールでも、プロパティを利用できるようにする必要があります。そうすると、デザイン時に [プロパティ] ウィンドウで設定でき、コードでも参照できるようになります。

ユーザー コントロールのプロパティを公開するのは、クラスのプロパティを公開するのと非常によく似ています。大きな違いは、ユーザー コントロールに含まれているコントロールのプロパティも公開できるという点です。クラスと同様に、プロパティを宣言し、**Get** プロシージャと **Set** プロシージャにコードを追加します。ユーザー コントロールの中に含まれているコントロールのプロパティを公開する場合、その値を格納するためのプライベート変数を宣言する必要はありません。そのコントロールのプロパティが格納します。

現時点では、`FirstName`、`MiddleName`、および `LastName` の各コントロールに入力されているテキストや `FullName` ラベルの値を取得する方法は用意していません。コントロールの利便性を高めるためには、これらの値をプロパティとして公開する必要があります。`FullName` ラベルの値は、外部のコードから変更できないよう、読み取り専用のプロパティとして公開する必要があります。

やってみよう

プロパティを追加するには

1. コード エディタで、`FirstName`、`MiddleName`、および `LastName` の各値をプロパティとして公開する次のコードを追加します。

```
Property FirstNameText() As String
    Get
        Return FirstName.Text
    End Get
    Set(ByVal value As String)
```

```
            FirstName.Text = value
        End Set
    End Property
    Property MiddleNameText() As String
        Get
            Return MiddleName.Text
        End Get
        Set(ByVal value As String)
            MiddleName.Text = value
        End Set
    End Property
    Property LastNameText() As String
        Get
            Return LastName.Text
        End Get
        Set(ByVal value As String)
            LastName.Text = value
        End Set
    End Property
```

2. `FullName` ラベルの値を読み取り専用プロパティとして公開する次のコードを追加します。

```
    ReadOnly Property FullNameText() As String
        Get
            Return FullName.Text
        End Get
    End Property
```

3. F5 キーを押してプログラムを実行します。
4. [ユーザー コントロール テスト コンテナ] で、[プロパティ] グリッドの一番下にスクロールし、FirstNameText プロパティを選択します。名前を入力し、FullNameText プロパティを選択します。FirstName テキスト ボックスに名前が表示され、FullNameText プロパティと一致します。

   他のプロパティのいくつかを、[プロパティ] グリッドとコントロール自体の両方で変更してみて、両者の関係を確認します。これは、このコントロールのユーザーがデザイン時に経験することと同じです。
5. [ファイル] メニューの [すべてを保存] をクリックして変更を保存します。

次の手順

このレッスンでは、ユーザー コントロールでイベントを処理する方法と、いくつかのプロパティを公開する方法について説明しました。プロパティの詳細について学習するには、「詳細 : 名前付き値を持つプロパティの追加」に進んでください。また、次のレッスンでは、作成したコントロールを使用する方法を説明します。

次のレッスン : 「ユーザー コントロールのテスト」

参照

処理手順

ユーザー コントロールへのコントロールの追加

その他の技術情報

表示されるオブジェクト : 初めてのユーザー コントロールの作成

オブジェクトによるプログラミング : クラスを使用する

Visual Basic ガイド ツアー

ユーザー コントロールのテスト

このレッスンでは、ユーザー コントロールを他のプロジェクトでテストして実行時の動作を確認する方法について説明します。

実行時の動作

ユーザー コントロールが完成し、テスト コンテナ でデザイン時の動作をテストしたら、プログラムで使用したときの動作を確認する必要があります。Visual Basic 2005 では、Windows アプリケーション プロジェクトを追加することにより、ユーザー コントロールを簡単にテストできます。

ユーザー コントロールは、ツールボックスに自動的に表示されます。そして、他のコントロールの場合と同様にフォームに追加して、プロパティを設定できます。

やってみよう

ユーザー コントロールをテストするには

1. 前のレッスンで作成した `NamesUserControl` プロジェクトを開きます。プロジェクトを保存しなかった場合は、まず前のレッスン「ユーザー コントロールへのコードの追加」に戻って、そのレッスンの手順を完了する必要があります。
2. [ファイル] メニューの [追加] をポイントし、[新しいプロジェクト] をクリックします。
3. [新しいプロジェクトの追加] ダイアログ ボックスで、[Windows アプリケーション] をクリックします。
4. [プロジェクト名] ボックスに「`UserControlTest`」と入力し、[OK] をクリックします。

   新しいプロジェクトがソリューション エクスプローラに追加され、新しいフォームが表示されます。
5. ソリューション エクスプローラの UserControlTest プロジェクトをクリックし、[プロジェクト] メニューの [スタートアップ プロジェクトに設定] をクリックします。
6. ツールボックスで、NamesControl をクリックし、フォームにドラッグします。
7. [プロパティ] ウィンドウで、FirstNameText、MiddleNameText、および LastNameText の各プロパティを、自分の名前に合わせて設定します。
8. F5 キーを押してプログラムを実行します。テキスト ボックスで名前を変更して、ラベルが正しく更新されることを確認します。
9. [ファイル] メニューの [すべてを保存] をクリックして変更を保存します。

次の手順

このレッスンでは、ユーザー コントロールをフォームに追加して実行時の動作を確認する方法について説明しました。ここで、「詳細情報 : ユーザー コントロールのカスタマイズ」に進んでユーザー コントロールの詳細を参照するか、次の一連のレッスンに進んでグラフィックスについて参照するかを選択できます。

次のレッスン : 「ピクチャの描画 : グラフィックスを使用する」

参照

処理手順

ユーザー コントロールへのコードの追加

その他の技術情報

表示されるオブジェクト : 初めてのユーザー コントロールの作成

オブジェクトによるプログラミング : クラスを使用する

Visual Basic ガイド ツアー

詳細情報：ユーザー コントロールのカスタマイズ

このレッスンでは、ユーザー コントロールの利便性を高めるようにカスタマイズする方法について説明します。

ラベルの追加

前のレッスンで、NamesControl ユーザー コントロールをテストし、正しく動作することを確認しました。このコントロールには、改良の余地があります。たとえば、どのテキスト ボックスにどの名前を入力するのかが明確でない点や、ユーザーに 3 つの名前すべてを確実に入力してもらえるとは限らない点です。

このユーザー コントロールの利便性を高めるために、各テキスト ボックスを識別するラベルを追加できます。ラベルのテキストを、"First Name"、"Middle Name"、および "Last Name" と設定してもよいですが、もし後で "Middle Initial" が必要となった場合はどうすればよいでしょうか。テキストをデザイン時に変更できるようにし、また各プロパティに既定値を用意できるよう、ラベル テキストを表すプロパティを作成する方が適切です。

やってみよう

ユーザー コントロールをカスタマイズするには

1. 前のレッスンで作成した `NamesUserControl` プロジェクトを開きます。プロジェクトを保存しなかった場合は、まず前のレッスン「ユーザー コントロールのテスト」に戻って、そのレッスンの手順を完了する必要があります。
2. ソリューション エクスプローラの NamesControl.vb をクリックし、[表示] メニューの [デザイナ] をクリックします。
3. ツールボックスから、3 つの Label コントロールをデザイナにドラッグし、各 TextBox の上に配置します。
4. ソリューション エクスプローラの NamesControl.vb をクリックし、[表示] メニューの [コード] をクリックします。
5. コード エディタで、ラベル テキストのプロパティを作成する次のコードをクラスに追加します。

```
Private text1 As String = "First Name"
Property Label1Text() As String
    Get
        Return text1
    End Get
    Set(ByVal value As String)
        text1 = value
        Label1.Text = text1
    End Set
End Property
Private text2 As String = "Middle Name"
Property Label2Text() As String
    Get
        Return text2
    End Get
    Set(ByVal value As String)
        text2 = value
        Label2.Text = text2
    End Set
End Property
Private text3 As String = "Last Name"
Property Label3Text() As String
    Get
        Return text3
    End Get
    Set(ByVal value As String)
        text3 = value
        Label3.Text = text3
    End Set
End Property
```

このコードでは、ラベル テキストを表す 3 つの **Private** 変数を宣言しており、表示する既定値を指定しています。

6. コード エディタで、左のドロップダウン ボックスの [(NamesControl イベント)] を選択し、右のドロップダウン ボックスの [Load] イベントを選択します。

7. **NamesControl_Load** イベント ハンドラに次のコードを追加します。

```
' Initialize the three labels
Me.Label1.Text = Label1Text
Me.Label2.Text = Label2Text
Me.Label3.Text = Label3Text
```

8. [ビルド] メニューの [ソリューションのビルド] をクリックします。

9. ソリューション エクスプローラで Form1.vb を選択して、[表示] メニューの [デザイナ] をクリックします。

   ラベルに既定のテキストが含まれていることを確認します。[プロパティ] ウィンドウで Label1Text プロパティを変更し、コントロールでも同様に変更されることを確認します。

10. [ファイル] メニューの [閉じる] をクリックして、フォーム デザイナを閉じます。

検証の追加

利便性を高めるためのもう 1 つのカスタマイズとしては、入力内容を検証して正しいかどうかを確認するコードの追加があります。各 TextBox コントロールを個別に検証しなくても済む、ユーザー コントロール全体用の検証コードを作成できます。

大半のコントロールには、フォーカスが他に移動したときに発生する **Validating** イベントがあります。検証コードはそこに入力します。ここでは、各テキスト ボックスに名前が入力されていることを確認するコードを記述する必要があります。

空のテキスト ボックスがある場合、名前を入力するようユーザーに通知するメッセージ ボックスを表示します。既定のメッセージを格納するプロパティを用意すると、コントロールのユーザーがメッセージを変更して、必要な内容を通知できます。

また、コントロールのユーザーによっては、ミドル ネームを必要としない可能性もあるので、MiddleName テキスト ボックスの検証をオフにするための **Boolean** プロパティも追加する必要があります。

やってみよう

検証を追加するには

1. コード エディタで、検証に関連する 2 つのプロパティのコードを追加します。1 つは、ミドル ネームが必要かどうかを指定するプロパティ、もう 1 つは、検証が失敗した場合に表示するメッセージを指定するプロパティです。

```
Private required As Boolean = True
Property MiddleNameRequired() As Boolean
    Get
        Return required
    End Get
    Set(ByVal value As Boolean)
        required = value
    End Set
End Property
Private errormessage As String = "Please enter your name."
Property ValidationErrorMessage() As String
    Get
        Return errormessage
    End Get
    Set(ByVal value As String)
        errormessage = value
    End Set
End Property
```

2. コード エディタで、左のドロップダウン ボックスの [(NamesControl イベント)] を選択し、右のドロップダウン ボックスの [Validating] イベントを選択します。

3. **NamesControl_Validating** イベント ハンドラに次のコードを追加します。

```
If MiddleNameRequired = True Then
    If FirstName.Text = "" Or MiddleName.Text = "" Or _
LastName.Text = "" Then
        MsgBox(ValidationErrorMessage)
    End If
Else
    ' Middle name isn't required.
    If FirstName.Text = "" Or LastName.Text = "" Then
        MsgBox(ValidationErrorMessage)
    End If
End If
```

4. [ビルド] メニューの [ソリューションのビルド] をクリックします。
5. ソリューション エクスプローラで Form1.vb を選択して、[表示] メニューの [デザイナ] をクリックします。

   フォーム上のユーザー コントロールを選択し、2 つの新しいプロパティが [プロパティ] ウィンドウに表示されることを確認します。
6. ツールボックスから、フォームに Button コントロールをドラッグします。
7. F5 キーを押してプログラムを実行します。

   姓と名を入力します。ミドル ネームは入力せずにおきます。ボタンをクリックすると、**ValidationErrorMessage** の内容を伝えるメッセージ ボックスが表示されます。
8. [ファイル] メニューの [すべてを保存] をクリックして変更を保存します。

次の手順

このレッスンでは、ユーザー コントロールをカスタマイズして利便性を高める方法について説明しました。次のレッスンでは、グラフィックスを使用して画像やテキストを描画する方法について説明します。

次のレッスン : 「ピクチャの描画 : グラフィックスを使用する」

参照

処理手順

ユーザー コントロールのテスト

その他の技術情報

表示されるオブジェクト : 初めてのユーザー コントロールの作成

オブジェクトによるプログラミング : クラスを使用する

Visual Basic ガイド ツアー

ピクチャの描画：グラフィックスを使用する

前のレッスンでは、フォームおよびコントロールを使用してユーザー インターフェイスを作成する方法について説明しました。コントロールでは作成できないイメージまたは効果を使用して、プログラムの外観をカスタマイズする場合もあります。

Visual Basic では、グラフィックス メソッドを使用して、フォームまたはコントロールにほとんどすべてのものを描画できます。このレッスンでは、Visual Basic のグラフィックス機能について説明します。

このセクションの内容

関連するセクション

ピクチャ描画 : イメージを表示する

Visual Basic ガイド ツアー

グラフィックスの表示

このレッスンでは、Visual Basic のグラフィックス メソッドを使用して、フォーム上に描画する方法を説明します。

前のレッスンでは、**PictureBox** コントロールを使用して、フォーム上に画像を表示する方法を説明しました。既に画像がある場合はそれで十分ですが、フォーム上に直接何かを描画する必要が生じる場合があります。たとえば、2 つのフィールドを区切る線や、重要なラベルを強調する円を描画する必要がある場合などが考えられます。

Visual Basic では、グラフィックス メソッドを使用することで、フォームやコントロールに事実上何でも描画できます。

グラフィックスの基本

描画を始める前に、いくつかのことを知っておく必要があります。コンピュータの画面は、何千個ものピクセルと呼ばれる小さなドットで構成されています。プログラムは、各ピクセルの色を定義することによって、画面に何を表示するかを制御します。もちろん、その処理の大半は、フォームやコントロールを定義するコードによって自動的に行われます。

フォームを、描画やペイントを行うキャンバスと想定してください。実際のキャンバスと同様、フォームにも寸法があります。実際のキャンバスはインチやセンチメートルなどの単位で計測しますが、フォームはピクセル単位で計測します。座標というシステムによって各ピクセルの位置が決定されます。つまり、X 座標で左から右方向の寸法を計測し、Y 座標で上から下方向の寸法を計測します。

座標の始点は、フォームの左上端です。このため、左から 10 ピクセル、上から 10 ピクセルの位置に 1 つのドットを描画する場合は、X 座標と Y 座標を `10, 10` と表します。

ピクセルは、グラフィックスの幅と高さを表現するときにも使用します。幅 100 ピクセル、高さ 100 ピクセルで、左上端が左から 10 ピクセル、上から 10 ピクセルの位置にある四角形を定義するには、座標を `10, 10, 100, 100` と表します。

画面上に描画する操作は、ペイントと呼ばれます。フォームおよびコントロールには、再描画の必要が生じたときに発生する **Paint** イベントが定義されています。たとえば、フォームが最初に表示されるときや、他のウィンドウの下に隠れてしまったときなどです。グラフィックスを表示するためのコードは、通常 **Paint** イベント ハンドラに記述します。

直線の描画

フォーム上に線を描画するには、座標および色という 2 つの事項を定義する必要があります。前述したとおり、X 座標と Y 座標はピクセルで表されます。線には、座標の組み合わせが 2 つあります。つまり、始点と、その後に記述される終点です。

紙に線を描くときにペンを使用するように、Visual Basic でも **Pen** オブジェクトを使用してフォーム上に描画します。**Pen** オブジェクトは、線の外観 (この場合は色) を定義します。次の手順では、フォーム上に横線、縦線、および斜めの線を描画します。

やってみよう

線を描画するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Lines`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開き、[イベント] ボックスの一覧の [Paint] をクリックします。
5. Form1_Paint イベント ハンドラに次のコードを追加します。

   VB

   ```
   ' Draw a 400 pixel black line 25 pixels from the top of the form.
   e.Graphics.DrawLine(Pens.Black, 0, 25, 400, 25)
   ' Draw a 500 pixel red line 100 pixels from the left of the form.
   e.Graphics.DrawLine(Pens.Red, 100, 0, 100, 500)
   ' Draw a diagonal blue line from the upper left to the lower right.
   e.Graphics.DrawLine(Pens.Blue, 0, 0, Me.Width, Me.Height)
   ```
6. F5 キーを押してプログラムを実行します。フォームに 3 本の線が表示されます。

次の手順

このレッスンでは、グラフィックスの基本と、線を描画する方法を説明しました。次のレッスンでは、四角形や円などの形を描画する方法を説明します。

次のレッスン : 「フォームへの図形の描画」

参照

処理手順

ピクチャ描画 : イメージを表示する

その他の技術情報

ピクチャの描画 : グラフィックスを使用する

Visual Basic ガイド ツアー

フォームへの図形の描画

このレッスンでは、四角形や円などの図形をフォームに描画する方法について説明します。

前のレッスンでは、**DrawLine** グラフィックス メソッドおよび **Pen** オブジェクトを使用してフォームに線を描画する方法について説明しました。**DrawLine** メソッド以外に、Visual Basic には、図形を描画するためのグラフィックス メソッド、および図形を塗りつぶすためのブラシと呼ばれるグラフィックス オブジェクトがあります。

単純な図形の描画

図形の描画は、線の描画に似ています。描画に使用する座標と色を定義する必要があります。線では、座標で開始点と終了点を定義しましたが、正方形や四角形などの図形では、座標で左上隅、幅、および高さを表します。

円および楕円には左上隅はないため、座標は外接する四角形の左上隅を表します。これは、円または楕円と同じ幅および高さを持つ架空の四角形です。

やってみよう

図形を描画するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Shapes`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開き、[イベント] ボックスの一覧の [描画] をクリックします。
5. Form1_Paint イベント ハンドラに次のコードを追加します。

   VB

```
' Draw a 200 by 150 pixel green rectangle.
e.Graphics.DrawRectangle(Pens.Green, 10, 10, 200, 150)
' Draw a blue square
e.Graphics.DrawRectangle(Pens.Blue, 30, 30, 150, 150)
' Draw a 150 pixel diameter red circle.
e.Graphics.DrawEllipse(Pens.Red, 0, 0, 150, 150)
' Draw a 250 by 125 pixel yellow oval.
e.Graphics.DrawEllipse(Pens.Yellow, 20, 20, 250, 125)
```

6. F5 キーを押してプログラムを実行します。フォームに 4 つの図形が表示されます。

   プロジェクトは開いたままにしておいてください。次の手順でプロジェクトに追加します。

塗りつぶされた図形の描画

描画した図形は外枠だけです。純色を使用して図形を描画するには、**FillRectangle** や **FillEllipse** などの塗りつぶしメソッドの 1 つを使用する必要があります。塗りつぶしメソッドは、別の種類の塗りつぶしグラフィックス オブジェクトである **Brush** オブジェクトを使用します。

別の色を使用して図形を塗りつぶす場合、図形より小さい座標を定義する必要があります。図形より小さくしておかないと、境界線まで塗りつぶされます。たとえば、座標 `0, 0, 150, 150` の正方形を塗りつぶすには、座標 `1, 1, 148, 148` で塗りつぶしを指定します。こうすると、境界線の太さが 1 ピクセルになります。

塗りつぶされた図形を描画するには

1. Form1_Paint イベント ハンドラで、以前に入力したコードの下に次のコードを追加します。

   VB

```
' Fill the circle with the same color as its border.
e.Graphics.FillEllipse(Brushes.Red, 0, 0, 150, 150)
' Fill the square with a different color.
```

```
e.Graphics.FillRectangle(Brushes.Aquamarine, 31, 31, 148, 148)
```

2. F5 キーを押してプログラムを実行します。

   塗りつぶされた正方形が塗りつぶされた円の上に表示されますが、重なった境界線の部分が見えなくなっています。グラフィックス メソッドを呼び出す順序によって、図形が描画される順序が決まります。この場合、塗りつぶされた円は境界線が青い四角形の後に描画されました。

   メソッドの順序を変更して、どのような結果になるか確認してください。

次の手順

このレッスンでは、図形を描画して塗りつぶす方法について説明しました。次のレッスンでは、グラフィックス メソッドを使用してテキストを描画する方法について説明します。

次のレッスン : 「フォームへのテキストの描画」

参照

処理手順

グラフィックスの表示

その他の技術情報

ピクチャの描画 : グラフィックスを使用する

Visual Basic ガイド ツアー

フォームへのテキストの描画

ここでは、グラフィックス メソッドを使用してフォームにテキストを描画する方法について説明します。

前のレッスンでは、**Label** コントロールを使用してテキストを表示する方法について説明しました。しかし、グラフィックス メソッドを使用してテキストを自分で描画することが必要な場合もあります。たとえば、斜めになっているテキストが必要な場合、**Label** コントロールは使用できませんが、グラフィックス メソッドを使用すると、テキストを任意の角度で描画できます。

テキストの描画

フォームまたはコントロールにテキストを描画するには、**DrawString** グラフィックス メソッドを使用します。その他の描画メソッドと同様に、**DrawString** は、色を決定する **Brush** オブジェクト、およびテキストを描画する場所を指定する座標 (この場合はテキストに外接する四角形の左上隅の X 座標と Y 座標) を使用します。

DrawString メソッドには追加の引数として、描画する文字列、およびテキストの外観を決定するフォントの 2 つの引数があります。フォントを指定するには、**Font** オブジェクトを作成してから、**DrawString** メソッドへの引数としてそのオブジェクトを使用する必要があります。

やってみよう

テキストを描画するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。

   a. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。

   b. [プロジェクト名] ボックスに「`DrawText`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。

2. フォームをダブルクリックしてコード エディタを開き、[イベント] ボックスの [Paint] をクリックします。

3. Form1_Paint イベント ハンドラに次のコードを追加します。

   VB

```
' Create a font object.
Dim aFont As New System.Drawing.Font("Arial", 22, FontStyle.Bold)
' Display the text with the DrawString method.
e.Graphics.DrawString("Graphics are fun!", aFont, Brushes.Black, _
    20, 10)
```

4. F5 キーを押してプログラムを実行します。テキストがフォームに表示されます。

   プロジェクトは開いたままにしておいてください。次の手順でプロジェクトに追加するものがあります。

回転したテキストの描画

テキストを任意の角度で描画するには、変形と呼ばれる別の種類のグラフィックス メソッドを使用する必要があります。さまざまなグラフィックス効果に使用できる変形には、いくつかの種類があります。この場合は **RotateTransform** メソッドを使用します。

RotateTransform メソッドは、テキストを回転する角度を指定する 1 つの引数を使用します。変形は、**RotateTransform** メソッドの次のコード行で実行されます。このメソッドを使用して、その他の描画メソッドを使用して描画される図形または線を回転することもできます。

やってみよう

回転したテキストを描画するには

1. Form1_Paint イベント ハンドラで、以前に入力したコードの下に次のコードを追加します。

   VB

```
' Rotate the text 45 degrees.
e.Graphics.RotateTransform(45)
e.Graphics.DrawString("And exciting too!", aFont, Brushes.Red, _
    100, 0)
```

2. F5 キーを押してプログラムを実行します。回転したテキストがフォームに表示されます。

次の手順

このレッスンでは、**DrawString** メソッドを使用してテキストを表示する方法について説明しました。次のレッスンでは、グラフィックス メソッドを使用してイメージを表示する方法について説明します。

次のレッスン : 「フォームへのイメージの描画」

参照

処理手順

フォームへの図形の描画

その他の技術情報

ピクチャの描画 : グラフィックスを使用する

Visual Basic ガイド ツアー

フォームへのイメージの描画

このレッスンでは、グラフィックスの呼び出しを使用してイメージを表示する方法について説明します。

以前のレッスンで、**PictureBox** コントロールを使用してイメージを表示する方法について説明しました。Visual Basic のグラフィックス メソッドを使用してファイルからイメージを表示することもできます。イメージの回転など、特別な処理を実行する必要がある場合は、前のレッスンと同様に、**PictureBox** コントロールではなくグラフィックス メソッドを使用する必要があります。

イメージの表示

フォームまたはコントロールにイメージを表示するには、**DrawImage** グラフィックス メソッドを使用します。**DrawImage** メソッドは、ビットマップ イメージ、およびイメージの左上隅を定義する X 座標と Y 座標を引数とします。

やってみよう

回転するイメージを表示する

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`DrawImage` 」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. ソリューション エクスプローラで、[My Project] ノードをダブルクリックしてプロジェクト デザイナを開きます。
5. プロジェクト デザイナの [リソース] タブをクリックし、[リソースの追加] をクリックして [既存のファイルの追加] をクリックします。
6. [既存のファイルをリソースに追加] ダイアログ ボックスで、任意のイメージ ファイルを表示して選択し、[開く] をクリックします。
7. ソリューション エクスプローラで、Form1 ノードを選択して、[表示] メニューの [コード] をクリックし、コード エディタを開きます。
8. コード エディタで、[イベント] ボックスの [Paint] をクリックします。
9. Form1_Paint イベント ハンドラに次のコードを追加します。

   VB

```
e.Graphics.RotateTransform(45)
e.Graphics.DrawImage(My.Resources.picture, 50, 0)
```

   メモ：

   `picture` を前の手順で追加したリソースの名前に置き換えます。
10. F5 キーを押してプログラムを実行します。回転するイメージがフォームに表示されます。

次の手順

このレッスンでは、回転するイメージをフォームに表示する方法について説明しました。これで、グラフィックスに関するレッスンは終了です。次の一連のレッスンでは、作成したプログラムを共有する方法について説明します。

次のレッスン：「プログラムの配布」

参照

処理手順

フォームへのテキストの描画

その他の技術情報

ピクチャの描画：グラフィックスを使用する

Visual Basic ガイド ツアー

プログラムの配布

他のユーザーとプログラムを共有するには、複数の方法があります。ClickOnce の発行機能を使用すると、プログラムを CD-ROM または DVD-ROM で使用できるようにしたり、他のユーザーがダウンロードして実行できるように、プログラムを Web サイトに発行したりできます。プログラムを電子メールで送信したり、ディスクにコピーしたりすることもできます。

このセクションの内容

プログラムの共有 : 配置の概要

CD でのプログラムの配布 : ClickOnce で発行する

関連するセクション

詳細情報 : プログラムに必須コンポーネントを含める

詳細情報 : インターネットでプログラムを配布する

電子メール経由でのプログラムの送信 : 圧縮 (ZIP 形式) ファイルを作成する

電子メール経由でのプログラムの取得 : 圧縮された (ZIP 形式) プログラムを抽出する

プログラムの共有：配置の概要

このレッスンでは、他のユーザーとプログラムを共有するためのさまざまな方法について説明します。

プログラムの記述、テスト、およびデバッグが完了した後、そのプログラムを他のユーザーと共有することができます。プログラムのコピーを作成して配布するプロセスは、配置と呼ばれます。

プログラム ファイルを他のコンピュータに単純にコピーするとプログラムを実行できると思っている人もいます。しかし、多くの場合、それではプログラムは動作しません。これは、ほとんどのプログラムが、コンポーネントと呼ばれる他のソフトウェアに依存しているためです。これを別のコンピュータにもインストールする必要があります。プログラムを実行しようとしたときに、コンポーネントが存在しない場合、プログラムは実行されません。

ClickOnce の発行機能

Visual Basic には、ClickOnce の発行機能という手法を使用してプログラムを配置するためのツールが備わっています。これを使用すると、必要なコンポーネントすべてをプログラムに自動的に組み合わせてインストールできるため、配置が容易になります。ClickOnce では、CD-ROM または DVD にプログラムを発行して、他の人と共有できます。

Web サーバーにアクセスできる場合、ClickOnce を使用してプログラムを Web サイトに発行することもできます。これにより、プログラムをインターネットでダウンロードできます。後でプログラムを変更した場合は、新しいバージョンを発行すると、プログラムをダウンロードしたユーザーは、次にプログラムを実行するときに新しいバージョンを自動的にダウンロードできます。

ClickOnce を使用してプログラムを発行するには、CD または DVD への書き込み機能または Web サーバーを使用する必要があります。これらのいずれも使用できない場合でも、プログラムを圧縮して、フロッピー ディスクにコピーするか、電子メールで送信することで、プログラムを共有できます。

次の手順

このトピックでは、配置と ClickOnce の発行機能について説明しました。次のレッスンでは、ClickOnce を使用してプログラムを発行する方法について説明します。

次のレッスン：「CD でのプログラムの配布：ClickOnce で発行する」

参照

処理手順

電子メール経由でのプログラムの送信：圧縮 (ZIP 形式) ファイルを作成する

その他の技術情報

プログラムの配布

CD でのプログラムの配布：ClickOnce で発行する

このレッスンでは、作成したプログラムを他の人々と共有するために、プログラムを CD-ROM または DVD に発行する方法について説明します。

作成したプログラムを他の人々と共有するには、まずセットアップ プログラムを作成して、CD-ROM やその他のメディアにコピーする必要があります。セットアップ プログラムには、プログラム自体の他に、プログラムが動作するために必要なその他のコンポーネントやファイルを含める必要があります。これらのコンポーネントやファイルは、必要条件と呼ばれます。

しかし、どのような必要条件を含めるのかを正確に決定するには、困難が伴う場合があることは想像できます。こうした場合、Visual Basic の ClickOnce 発行ツールを使用すると、これらすべての作業だけでなく、それ以上のことも実行できます。また、ClickOnce の発行機能を使用して、インターネットやローカル ネットワークにプログラムを発行することもできます。ただし、このレッスンでは、プログラムを CD に発行します。

メモ：

プログラムを CD-ROM や DVD に発行するには、プログラムを書き込むために使用する適切なハードウェアとソフトウェアが、コンピュータにインストールされている必要があります。音楽 CD に書き込みができる場合は、プログラムを CD に発行することもできます。この機能が備わっていない場合は、他の方法でプログラムを共有することもできます。詳細については、
「電子メール経由でのプログラムの送信 : 圧縮 (ZIP 形式) ファイルを作成する」を参照してください。

ClickOnce による発行

ClickOnce を使用したプログラムの発行は、非常に簡単なプロセスです。つまり、発行ウィザードでいくつかの選択を行って、作成されたファイルを CD に書き込むだけです。

プログラムを発行する前に、必ずそのプログラムをテストし、エラーが発生しないことを確認します。発行の準備が整ったら、[ビルド] メニューの [発行] をクリックして、発行ウィザードを起動します。

発行ウィザードは、3 つの手順で構成されます。最初の手順は、セットアップ プログラムと関連ファイルを配置する場所の選択です。CD に発行する場合は、ローカル ディスク上のフォルダを選択します。この場所は、後でプログラムを CD に書き込むときに、再度選択することになります。2 番目の手順は、ユーザーがプログラムをインストールする方法の指定です。この例では CD-ROM からのインストールです。

最後の手順は、プログラムが起動するたびに、新しいバージョンを自動的にチェックするかどうかの指定です。Web サーバーにアクセスできる場合は、プログラムの更新バージョンを Web サーバー上に発行できます。詳細については、「詳細情報 : インターネットでプログラムを配布する」を参照してください。ただし、この例では CD に発行するため、プログラムに更新バージョンのチェック機能は実装しません。

発行ウィザードを実行すると、プログラムの必要条件が自動的に決定されます。既定では、.NET Framework などの必要条件がセットアップ プログラムと共にパッケージ化されることはありません。セットアップ プログラムは、インストール時に必要条件をチェックし、必要に応じてインターネットからこれらの必要条件をダウンロードし、インストールします。

CD のセットアップ プログラムに必要条件を含めるには、プロジェクト デザイナでプロパティを設定できます。ただし、必要条件用の再頒布可能ファイルをローカル コンピュータにまずダウンロードする必要があります。詳細については、「詳細情報 : プログラムに必須コンポーネントを含める」を参照してください。

やってみよう

CD に発行するには

1. [ファイル] メニューの [プロジェクトを開く] をクリックします。
2. [プロジェクトを開く] ダイアログ ボックスで、前のレッスンで作成した [Windows アプリケーション] プロジェクトを参照し、[開く] をクリックします。
3. F5 キーを押してプロジェクトを実行します。エラーが発生したら、先に進む前にそのエラーを修復する必要があります。
4. [デバッグ] メニューの [デバッグの停止] をクリックします。
5. [ビルド] メニューの [<Projectname> の発行] をクリックします。Projectname は、プロジェクトの名前です。

   発行ウィザードが起動します。
6. 発行ウィザードの [アプリケーションをどこに発行しますか?] ページで、プログラムを発行する場所のパス (`C:\My Programs` など) を入力します。フォルダが存在しない場合は、作成するよう要求されます。

   [次へ] をクリックして、ウィザードの次のページに進みます。
7. [ユーザーはアプリケーションをどのようにインストールするのですか?] ページで、[CD-ROM または DVD-ROM から] をクリックし、[次へ] をクリックします。

8. [アプリケーションの更新はどこで確認するのですか?] ページで、[アプリケーションの更新を確認しない] をクリックします。
9. [完了] をクリックします。ウィザードの最初のページで指定した場所にプログラムが発行されます。

   次に、CD または DVD の書き込みアプリケーションを使用して、プログラムの CD または DVD を作成します。このとき、プログラムを発行したフォルダにあるすべてのファイルを含める必要があります。
10. 作成した CD または DVD を他のコンピュータに挿入し、Setup.exe ファイルを実行します。.NET Framework などの必要条件をインストールする必要がある場合は、ダウンロードしてインストールするよう要求されます。
11. インストールが完了すると、[スタート] メニューに作成されたショートカットからプログラムを起動できるようになります。

    必要条件をプログラムと共にパッケージ化する方法を学習する場合は、プロジェクトを開いたままにしておいてください。
    「詳細情報 : プログラムに必須コンポーネントを含める」のレッスンで使用します。

次の手順

このレッスンでは、ClickOnce の発行機能を使用して、プログラムを CD-ROM に発行する方法を説明しました。この時点で、次の学習事項には複数の選択肢があります。

Web サーバーにアクセスできる場合で、作成したプログラムをインターネットに公開する方法を学習する場合は、
「詳細情報 : インターネットでプログラムを配布する」に進んでください。

CD-ROM または DVD-ROM への書き込みができない場合は、「電子メール経由でのプログラムの送信 : 圧縮 (ZIP 形式) ファイルを作成する」に進んでください。

必要条件をプログラムと共にパッケージ化する方法を学習する場合は、「詳細情報 : プログラムに必須コンポーネントを含める」に進んでください。

上記以外の場合は、ガイド ツアーのメインの部分は終了です。次のトピックには、リソースに関するいくつかの提案があり、Visual Basic についてさらに学習できます。

次のレッスン : 「さらに上のステップへ : 次の学習」

参照
概念
プログラムの共有 : 配置の概要

詳細情報：プログラムに必須コンポーネントを含める

このレッスンでは、ClickOnce の発行機能を使用して、必要なコンポーネントをプログラムと共にパッケージ化する方法を説明します。

既定では、ClickOnce 技術を使用して公開されたプログラムは、インストール時に必要条件をインターネットからダウンロードします。このため、インターネットにアクセスできないコンピュータにプログラムをインストールする場合は、インストールが失敗する可能性があります。

これを防ぐため、.NET Framework の再頒布可能ファイルなどの必要条件を、プログラムと共にパッケージ化できます。

メモ：
プログラムをインストールする人すべてがインターネットにアクセスできることがわかっている場合は、既定の方法を使用する必要があります。これにより、必要条件の新バージョンがリリースされている場合に、ユーザーが最新のバージョンを入手できるからです。

必要条件をパッケージ化する

必要条件をプログラムと共にパッケージ化するには、プロジェクト デザイナの発行プロパティを変更します。1 つの必要条件をパッケージ化する場合、それ以外のすべての必要条件もパッケージ化する必要があります。つまり、1 つの必要条件のみをパッケージ化して、他の必要条件をインターネットからダウンロードすることはできません。

やってみよう

必要条件をパッケージ化するには

1. 前のレッスン (「CD でのプログラムの配布 : ClickOnce で発行する」) で使用したプロジェクトを開きます。
2. ソリューション エクスプローラで、プロジェクト ノードをクリックし、[プロジェクト] メニューの [プロパティ] をクリックします。

   プロジェクト デザイナが開きます。
3. プロジェクト デザイナで、[発行] タブをクリックします。
4. [必須コンポーネント] をクリックして [必須コンポーネント] ダイアログ ボックスを開きます。
5. [必須コンポーネント] ダイアログ ボックスで、[アプリケーションと同じ場所から必須コンポーネントをダウンロードする] チェック ボックスをオンにし、[OK] をクリックします。
6. [ビルド] メニューの [<ProjectName> の発行] をクリックします。ProjectName は、プロジェクトの名前です。

   発行ウィザードが起動します。
7. [完了] をクリックしてプログラムを公開します。

メモ：
.NET Framework の再頒布可能ファイルをダウンロードしていない場合は、ダウンロードするよう要求される場合があります。

8. Windows エクスプローラで、プログラムを公開した場所を参照し、Setup.exe ファイルと Dotnetfx.exe ファイルの両方があることを確認します。

次の手順

このレッスンでは、必要条件をプログラムと共に公開する方法を説明しました。この時点で、次の学習事項には複数の選択肢があります。

Web サーバーにアクセスできる場合で、作成したプログラムをインターネットに公開する方法を学習する場合は、「詳細情報 : インターネットでプログラムを配布する」に進んでください。

CD-ROM または DVD-ROM への書き込みができない場合は、「電子メール経由でのプログラムの送信 : 圧縮 (ZIP 形式) ファイルを作成する」に進んでください。

上記以外の場合は、ガイド ツアーのレッスンは終了です。次のトピックには他のリソースに関するいくつかの提案があり、Visual Basic についてさらに学習できます。

次のレッスン : 「さらに上のステップへ : 次の学習」

参照

処理手順

CD でのプログラムの配布 : ClickOnce で発行する

概念

プログラムの共有 : 配置の概要

詳細情報：インターネットでプログラムを配布する

このレッスンでは、ClickOnce の発行機能を使用してプログラムを Web サーバーに配置する方法について説明します。

前のレッスンでは、プログラムを CD-ROM または DVD-ROM に発行する方法について説明しました。Web サーバーへのアクセス権がある場合、ClickOnce を使用して、プログラムをインターネットで使用できるように発行することもできます。

Web サーバーに発行すると、プログラムを幅広く使用できるようにする以外に、ClickOnce の自動更新機能を利用できます。後で新しいバージョンのプログラムを発行すると、次にユーザーがそのプログラムを実行しようとするときに、新しいバージョンをダウンロードしてインストールするようにプロンプトが表示されます。

メモ：
Web サーバーに発行するには、Web サーバーがインターネット インフォメーション サービス (IIS: Internet Information Services) を実行しており、FrontPage Extensions がインストールされている必要があります。また、ユーザーには IIS の管理者特権が必要です。

Web サーバーへの発行

Web サーバーへの発行は、CD-ROM または DVD-ROM への発行に非常によく似ています。発行ウィザードの実行中に、いくつかの項目を選択するだけで済みます。

その 1 つが、プログラムがオフライン、つまりコンピュータがインターネットに接続されていない場合に使用できるかどうかの指定です。プログラムをオンラインとオフラインの両方で使用できるようにすると、ユーザーがプログラムを起動できるように、Windows の [スタート] メニューにエントリが追加されます。プログラムをオンラインでのみ使用できるようにすると、ユーザーがプログラムを実行しようとするたびにプログラムがダウンロードされますが、[スタート] メニューには追加されません。

やってみよう

Web サーバーに発行するには

1. [ファイル] メニューの [プロジェクトを開く] をクリックします。
2. [プロジェクトを開く] ダイアログ ボックスで [Windows アプリケーション] プロジェクトを参照して、[開く] をクリックします。
3. F5 キーを押してプロジェクトを実行します。エラーが発生した場合は、続行する前にエラーを修正する必要があります。
4. [デバッグ] メニューの [デバッグの停止] をクリックします。
5. [ビルド] メニューの [<ProjectName> の発行] をクリックします。ProjectName は、プロジェクトの名前です。

   発行ウィザードが起動します。
6. 発行ウィザードの [アプリケーションをどこに発行しますか?] ページで、「`http://www.microsoft.com/myprogram`」など、プログラムを発行する Web サイトの URL を入力します。

メモ：
Web サーバーに発行するには、Web サーバーがインターネット インフォメーション サービス (IIS: Internet Information Services) を実行しており、FrontPage Extensions がインストールされている必要があります。また、ユーザーには IIS の管理者特権が必要です。

   [次へ] をクリックして、ウィザードの次のページに進みます。
7. [アプリケーションはオフラインでも利用できますか?] ページで、既定の [はい、このアプリケーションはオンラインでもオフラインでも利用できます] をクリックします。
8. [完了] をクリックしてプログラムを発行します。

   プログラムが指定された Web サイトに発行され、HTML ページが作成されます。
9. 別のコンピュータで Internet Explorer を開き、手順 6 で入力したのと同じ URL に移動し、[インストール] リンクをクリックしてプログラムをインストールします。

次の手順

このレッスンでは、ClickOnce の発行機能を使用してプログラムを Web サイトに発行する方法について説明しました。ここで、次のレッスンについて複数の選択肢があります。

CD-ROM または DVD-ROM への書き込みができない場合は、「電子メール経由でのプログラムの送信 : 圧縮 (ZIP 形式) ファイルを作成する」に進んでください。

プログラムに必須コンポーネントを共にパッケージ化する方法については、「詳細情報 : プログラムに必須コンポーネントを含める」を参照してください。

上記以外の場合、ガイド ツアーのレッスンは完了です。Visual Basic の詳細については、次のトピックに進み、その他のリソースについて記載されている情報を参照してください。

次のレッスン : 「さらに上のステップへ : 次の学習」

参照
概念
プログラムの共有 : 配置の概要

電子メール経由でのプログラムの送信 : 圧縮 (ZIP 形式) ファイルを作成する

このレッスンでは、プログラムを圧縮 (ZIP 形式) ファイルとしてパッケージ化して、電子メールで送信する方法を説明します。

プログラムを作成すると、その作品を友人や同僚たちと共有しようと考える場合があります。最良の方法は、ClickOnce の発行機能を使用して、CD-ROM や DVD にプログラムを公開することです。しかし、CD または DVD への書き込み機能を使用できない場合は、プログラムを圧縮ファイルにパッケージ化して電子メールで送信することによって配布できる場合があります。

メモ :

信頼できない送信者から受信した添付ファイルは、.zip ファイルをはじめ、どのような種類でも決して開かないでください。予想外の添付ファイルには、コンピュータをクラッシュまたはハイジャックしたり、個人情報を盗んだりするウイルスが含まれている可能性があります。同様の理由から、要求していない相手に、作成したアプリケーションを送信しないようにする必要があります。

ほとんどの単純なプログラムは、実行可能ファイル (*.exe*) と構成ファイル (*.config*) を圧縮ファイルにパッケージ化して配布できます。たとえば、*Calculator* という単純なプログラムの場合、*your Visual Studio Projects/Calculator/Calculator/bin* ディレクトリ (*your Visual Studio Projects* は Visual Studio プロジェクトが保存されているディレクトリ。通常は *My Documents/Visual Studio/Projects*) にある *Calculator.exe* ファイルと *Calculator.exe.config* ファイルが必要です。

メモ :

Visual Basic で作成されたすべてのプログラムには Microsoft .NET Framework version 2.0 が必要です。プログラムを電子メールで配布する場合は、すべての受信者のコンピュータに .NET Framework が既にインストールされていることを確認します。

ヒント

.NET Framework は、他の多くのプログラムと共に配布されていますが、Microsoft の Web サイトからダウンロードすることもできます。.NET Framework がインストールされているかどうかチェックするには、[コントロール パネル] ウィンドウの [プログラムの追加と削除] をクリックし、Microsoft .NET Framework version 2.0 があるかどうか確認します。

ローカル データベースを含むプログラムなど、より複雑なプログラムの場合は、上記以外のファイルを含めることが必要な場合もあります。その場合、電子メールでプログラムを送信するのではなく、ClickOnce の発行機能を使用してプログラムを公開することをお勧めします。

やってみよう

次の手順は、Windows の既定の圧縮 (ZIP 形式) フォルダ機能を使用して、ファイルを圧縮する方法を示しています。.zip ファイルを操作するための別のプログラムをインストールしている場合は、ファイル圧縮に必要な手順が異なる場合があります。

単純なプログラムを電子メールで送信するには

1. 作成して実行したプロジェクトを選択し、次に Windows エクスプローラでプロジェクト フォルダを表示します。
2. プロジェクト フォルダで、プロジェクトの名前を持つフォルダを開き、Bin という名前のフォルダを開いて、Debug フォルダを選択します。
3. *ProjectName*.exe ファイル (*ProjectName* はプロジェクト名) を選択し、ファイルを右クリックします。次に、[送る] をポイントし、[圧縮 (zip 形式) フォルダ] をクリックします。

   ProjectName.zip という名前の新しいファイルが作成され、ディレクトリに追加されます。このファイルを、電子メール メッセージの添付ファイルとして送信できます。

次の手順

このレッスンでは、プログラムをパッケージ化して電子メールで送信する方法を説明しました。次のレッスンでは、受信したプログラムを抽出して実行する方法を説明します。

次のレッスン :「電子メール経由でのプログラムの取得 : 圧縮された (ZIP 形式) プログラムを抽出する」

参照

処理手順

CD でのプログラムの配布 : ClickOnce で発行する

概念

プログラムの共有 : 配置の概要
その他の技術情報
プログラムの配布

電子メール経由でのプログラムの取得 : 圧縮された (ZIP 形式) プログラムを抽出する

このレッスンでは、電子メール経由で友人または同僚から受信したプログラムを抽出して実行する方法について説明します。

友人や同僚と困難な仕事を共有する場合があります。前のレッスンでは、プログラムを電子メール経由で送信できるように圧縮された (ZIP 形式) ファイルに追加する方法を説明しました。このレッスンでは、電子メール経由で受信したプログラムを開いて実行する方法を説明します。

メモ :

信頼できない送信者から受信した添付ファイルは、.zip ファイルをはじめ、どのような種類でも、決して開かないでください。予想外の添付ファイルには、コンピュータをクラッシュまたはハイジャックしたり、個人情報を盗んだりするウイルスが含まれている可能性があります。同じ理由から、予期していないユーザーにアプリケーションを送信しないでください。

やってみよう

圧縮された (ZIP 形式) プログラムを抽出するには

1. 前のレッスンで作成した .zip ファイルに移動します。レッスンを完了していない場合は、「電子メール経由でのプログラムの送信 : 圧縮 (ZIP 形式) ファイルを作成する」を完了します。
2. .zip ファイルを右クリックして、[すべて展開] をクリックします。

   抽出ウィザードが開きます。
3. ウィザードの [次へ] ボタンをクリックします。

   [展開先の選択] ペインが表示されます。
4. プログラムをインストールするフォルダのパスを入力するか、[参照] をクリックしてフォルダに移動し、[次へ] をクリックします。

   ファイルが抽出され、[展開の完了] ペインが表示されます。
5. [展開されたファイルを表示する] チェック ボックスをオンにして、[完了] をクリックします。

   Windows エクスプローラの新しいウィンドウが開き、前の手順で選択したフォルダが表示されます。
6. *ProgramName*.exe ファイル (*ProgramName* はプログラムの名前) をダブルクリックして、プログラムを実行します。

   メモ :

   Visual Basic で作成されたすべてのプログラムには Microsoft .NET Framework version 2.0 が必要です。電子メール経由でプログラムを配布する場合は、すべての受信者のコンピュータに .NET Framework がインストールされていることを確認します。

   ヒント

   .NET Framework は他の数多くのプログラムに付随して配布されています。また、Microsoft Web サイトからもダウンロードできます。.NET Framework がインストールされているかどうかチェックするには、[コントロール パネル] ウィンドウの [プログラムの追加と削除] をクリックし、Microsoft .NET Framework version 2.0 があるかどうか確認します。

次の手順

このレッスンでは、信頼できる送信元から電子メール経由で受信したプログラムを抽出して実行する方法について説明しました。信頼できない送信元から受信したファイルを実行するのは危険であることも説明しました。これで、基本的な説明は完了しました。Visual Basic を独自に調べていくつかのプログラムを記述することも、Visual Basic を詳細に学習するための情報を参照することもできます。

次のレッスン : 「さらに上のステップへ : 次の学習」

参照

処理手順

電子メール経由でのプログラムの送信 : 圧縮 (ZIP 形式) ファイルを作成する

その他の技術情報

プログラムの配布

Visual Basic ガイド ツアー

さらに上のステップへ：次の学習

Visual Basic ガイド ツアー のレッスンは終了しました。Visual Basic の上級者とはまだ言えませんが、独自のプログラムを記述するための十分な知識を習得できました。

作成するプログラムの案がある場合は、このまま終了してプログラムの記述を始めてください。その前に Visual Basic についてさらに学習する場合は、次を参照してください。

さらに学習するには

生産性の向上 : Rapid Application Development

初期のコンピュータ プログラミングでは、単純なプログラムであっても、完成までに数日または数週間を要することがありました。

ヒント : 知られていない方法

Visual Basic ガイド ツアーでは、さまざまな手法について説明してきましたが、Visual Basic 2005 で実現できることは他にもたくさんあります。

コミュニティ リソース : 友人からのちょっとした手助け

プログラミングを習得するのは必ずしも簡単ではありません。たとえ経験豊富なプログラマでも、自力では解決できない問題に直面します。

次のステップへ : Visual Basic の学習リソース

Visual Basic ガイド ツアーのすべてのレッスンを完了すると、自作プログラムの開発を始めることができる程度の習熟度に達します。しかし、当然ながらエキスパートとは呼べません。

流用可能 : サンプルとスタート キット

コードを作成するよりも効率的なのは、コードを作成しないで済むことです。Visual Basic 2005 ではヘルプ トピック、サンプル アプリケーション、およびスタート キットに含まれるコードを簡単に再利用できます。

詳細解説 : より詳しい情報の参照

Visual Basic ガイド ツアーでは、Visual Basic Express Edition および Visual Basic 2005 で実現できることをいくつか説明しましたが、これはほんの一部にすぎません。ガイド ツアーに登場した話題を深く掘り下げるには、ここから始めます。

参照

処理手順

Visual Basic Express のトラブルシューティング

概念

Visual Basic Express での操作方法

その他の技術情報

プログラムの配布

Visual Basic ガイド ツアー

生産性の向上：Rapid Application Development

初期のコンピュータ プログラミングでは、単純なプログラムであっても、完成までに数日または数週間を要することがありました。1991 年に Visual Basic が登場すると、プログラミングの世界に大変革がもたらされました。ユーザー インターフェイスを作成するためのコードを書いたり、メモリ管理に頭を悩ませたりする必要がなくなったのです。この新しいプログラミング方式は Rapid Application Development (RAD) と呼ばれました。

RAD プログラミングの最大のメリットは生産性の向上です。Visual Basic 2005 には、より優れたアプリケーションを従来より短時間で作成するのに役立つ機能が数多く備わっています。以下に示すのは、そうした機能の一部です。

メモ：

Visual Basic Express Edition を使用している場合、インストール時に選択したオプションによっては、このページのヘルプ リンクの一部を利用できないことがあります。詳細については、「Visual Basic Express のトラブルシューティング」を参照してください。

コード スニペット

生産性を向上させる方法の 1 つが、同じコードを何回も書くのを避けることです。Visual Basic 2005 には、IntelliSense コード スニペットという、約 500 個のコードから成るコード ライブラリが用意されており、自作のアプリケーションに挿入できるようになっています。各スニペットは、ファイルの作成、電子メール メッセージの送信、円の描画など、完全なプログラミング タスクを実行します。スニペットをソース コードに挿入するには、マウスを数回クリックするだけです。

スニペットを挿入すると、コードの中で置換が必要な部分が強調表示され、開発者は必要に応じて独自の値を入力できます。たとえば、フォーム上に線を描画するコード スニペットでは、色、位置、および線の長さを表す値を使用します。これらの値は、必要に応じて変更することもできるし、特に変更を加えずに既定値で線を描画することもできます。

また、必要に応じて独自のスニペットを作成してライブラリに追加し、後で必要なときにそれらを使用することもできます。独自のスニペットを作成するときには、コードのどの部分を強調表示するか、および既定値をどうするかを決定します。詳細については、「IntelliSense コード スニペットの作成と使用」を参照してください。

コード スニペットを使用して実現できる一般的なタスクの 1 つに、ファイルからのテキストの読み取りと書き込みがあります。次に示す手順は、コード スニペットで生産性を高める方法の例です。

やってみよう

コード スニペットを使用するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「`Snippets`」と入力し、[OK] をクリックします。

   新しい Windows フォーム プロジェクトが開きます。
4. フォームをダブルクリックしてコード エディタを開きます。
5. コード エディタで、`Form1_Load` イベント ハンドラを右クリックし、ドロップダウン メニューの [スニペットの挿入] をクリックします。

   スニペットのカテゴリの一覧が表示されます。
6. [ディスク、フォルダおよびファイルを処理中] をダブルクリックします。

   コード スニペットの一覧が表示されます。
7. [テキストのファイルへの書き込み] をダブルクリックします。

   次のコードが挿入され、`"C\Test.txt"` と `"Text"` が強調表示されます。

   ```
   My.Computer.FileSystem.WriteAllText("C:\Test.txt", "Text", True)
   ```

   メモ：

   ファイルが存在しない場合、**WriteAllText** メソッドによって作成されます。ファイルが既に存在する場合、その末尾にテキストが追加されます。

8. "C\Test.txt" を "C\MySnippetTest.txt" に置き換え、"Text" を "This is really fast!" に置き換えます。
9. 2 つ目のスニペットを追加します。右クリックして、メニューの [スニペットの挿入] をクリックします。
10. [ディスク、フォルダおよびファイルを処理中] をダブルクリックします。
11. [ファイルからのテキストの読み取り] をダブルクリックします。

    次のコードが挿入され、"C\Test.txt" が強調表示されます。

```
Dim fileContents As String
fileContents = My.Computer.FileSystem.ReadAllText("C:\Test.txt")
```

12. "C\Test.txt" を "C\MySnippetTest.txt" に置き換えます。
13. 最後のスニペットの下に次のコードを追加します。結果を表示するためのコードです。

```
MsgBox(fileContents)
```

14. F5 キーを押してプログラムを実行します。

    指定したテキストが記述されたファイルが作成され、ファイルの内容を示すメッセージ ボックスが表示されます。

Visual Basic 2005 に含まれているコード スニペットには、ある程度の時間をかけて慣れてください。大きな時間の節約になり、コードを作成するときの無駄な努力をなくすことができます。詳細については、「方法 : コード スニペットを管理する」を参照してください。

My による開発

Visual Basic 2005 で導入されたもう 1 つの RAD 機能に、**My** というものがあります。**My** は、コンピュータ、アプリケーション、ユーザーなどに関連して一般的に使用される機能を持つオブジェクトの集まりです。**My** は、他の方法では余分なコードが必要な機能を簡単に利用するための、短縮ダイヤルのようなものです。

たとえば、自作のアプリケーションのバージョン番号を確認する必要があるとします。以前のバージョンの Visual Basic では、次のようなコードを使用します。

```
Dim VersionNumber As String
VersionNumber = System.Diagnostics.FileVersionInfo.GetVersionInfo _ (System.Reflection.Asse
mbly.GetExecutingAssembly.Location).FileVersion
```

新しい **My.Application** オブジェクトを使用すると、次のようなコードになります。

```
Dim VersionNumber As String
VersionNumber = My.Application.Info.Version.ToString
```

これを見るとわかるように、**My** を使用する方が、はるかに簡単で見つけやすいため、時間と手間を節約できます。以前の方法でバージョン番号を確認することもできますが、わざわざそうする必要はありません。

気付いているかもしれませんが、ここまでのいくつかのレッスンでも **My** を既に使用しています。次回、アプリケーション用のコードを入力するときには、「My」と入力し、表示される項目の一覧をドリル ダウンして、**My** オブジェクトについて調べてみてください。詳細については、「My による開発」を参照してください。

IntelliSense

レッスンの手順に従ってコードを入力するときにおそらく気付いたことと思いますが、入力時には、選択肢のドロップダウン リストがコード エディタに表示されます。これは、IntelliSense という機能の一例です。

IntelliSense には、言語リファレンスを容易に利用できるようにする、さまざまな機能が用意されています。コードを作成するときに、コード エディタを離れて言語要素についての情報を得る必要はありません。元の場所にいる状態で、必要な情報を探し、言語要素を直接コードに挿入して、IntelliSense で入力を補完できます。

IntelliSense はデバッグのときにも役立ちます。コード エディタで、コード内の変数の上にポインタを移動すると、その変数の現在の値がツールヒントで表示されます。IntelliSense は、イミディエイト ウィンドウでコードを入力するときにも利用できます。詳細については、「IntelliSense の使用方法」を参照してください。

次の手順

このレッスンでは、Visual Basic 2005 の生産性に関連する機能について説明しました。Visual Basic 2005 には、他にも数多くの機能が備わっており、Rapid Application Development に最適なツールです。下に、そうしたいくつかの機能へのリンクを示します。

次のレッスンでは、Visual Basic 2005 で実現できる意外な機能のいくつかについて説明します。

次のレッスン : 「ヒント : 知られていない方法」

参照

概念

さらに上のステップへ : 次の学習

Visual Basic Express での操作方法

その他の技術情報

プログラムの配布

Visual Basic ガイド ツアー

ヒント：知られていない方法

Visual Basic ガイド ツアーでは、さまざまな手法について説明してきましたが、これらは Visual Basic 2005 で実現できることの一部にすぎません。Visual Basic のエキスパートと呼ばれる人たちでも、新しいヒントや手法の習得に常に励んでいるのです。ここでは、あまり知られていない手法について、いくつか取り上げます。

メモ：
Visual Basic Express Edition を使用している場合、インストール時に選択したオプションによっては、このページのヘルプ リンクの一部を利用できないことがあります。詳細については、「Visual Basic Express のトラブルシューティング」を参照してください。

形状を指定した Windows フォームの作成

おなじみの四角形のフォームでは飽き足らない場合や、Windows Media Player のような "スキン" を使用した外観を自作のアプリケーションで実現する場合、Visual Basic 2005 なら簡単に対応できます。目的の形状のビットマップ イメージを作成してフォームとして使用でき、フォームを移動および閉じることができるようにするコードを追加できます。詳細については、「方法 : 四角形以外の Windows フォームを作成する」を参照してください。

分割ウィンドウの作成

現在使用しているこの Document Explorer のように、ユーザーがサイズを変更できる複数の領域に分かれたフォームを作成する場合、Windows フォームの **SplitContainer** コントロールを使用すると、コードなしで作成できます。SplitContainer コントロールをフォーム上にドロップし、他のコントロールをその上に追加します。サイズ変更の動作は、アプリケーションを実行すると自動的に実行されます。

複数の **SplitContainer** コントロールをフォーム上に追加して、サイズを変更できる領域を他の領域内に作成できます。こうすると Microsoft Outlook のような外観のアプリケーションを作成できます。詳細については、「SplitContainer コントロール (Windows フォーム)」を参照してください。

サウンドの再生

通常、ゲームを作成する場合には、さまざまなイベントに対する反応として、プログラムでサウンドを再生する必要があります。それには **My.Computer.Audio** オブジェクトを使用します。これを使用すると、アプリケーションに組み込んだ wave ファイルを再生したり、インターネットからファイルを直接再生したりできます。詳細については、「My.Computer.Audio オブジェクト」を参照してください。

ユーザー設定の保存

Windows ベースのアプリケーションの多くでは、ウィンドウの配置や、そのアプリケーションを前回使用したときに表示していたツール バーなど、ユーザーの設定が記憶されています。アプリケーション設定を作成および使用して、情報を格納し、アプリケーションの次回実行時にその情報を取得するようにすると、自作のプログラムでも同じことを実現できます。詳細については、「アプリケーション設定の概要」を参照してください。

パーソナル化の実現

Web ページがユーザーの名前を覚えていて、"Welcome back (*insert your name here*)" のようなメッセージを表示するのは、いったいどのようなしくみなのかと、不思議に思ったことはないでしょうか。自作のアプリケーションでも、My.User.Name プロパティ プロパティを使用して、コンピュータに現在ログオンしているユーザーの名前を取得すると、似たような処理を実現できます。詳細については、「My.User オブジェクト」を参照してください。

Visual Basic 6.0 のコードの使用

Visual Basic 6.0 のコード例を Visual Basic 2005 で使用する場合には、Visual Basic 6 コードのアップグレード ツールを使用すると、Visual Basic 6.0 のコードを変換して Visual Basic 2005 コードに挿入できます。コードの中に変換できなかった部分がある場合には、コメントが追加され、コードを動作させるために必要な対処を説明するヘルプ トピックへのリンクが示されます。詳細については、「方法 : [Visual Basic 6 コードのアップグレード] ダイアログ ボックスを使って Visual Basic 6.0 のコードをアップグレードする」を参照してください。

マルチスレッドを使用したパフォーマンスの向上

Visual Basic アプリケーションでは、マルチスレッドという手法を使用して、複数のタスクを同時に実行できます。マルチスレッドとは、あるタスクが別の実行スレッドで実行される処理のことで、プログラムのパフォーマンスや反応が向上します。

たとえば、インターネットからファイルをダウンロードするプログラムの場合、ダウンロードには長い時間がかかることがあるため、ダウンロードが完了するまでユーザーが他の操作を実行できなくなるおそれがあります。ダウンロードを別個のスレッドで実行すると、ファイルのダウンロードをバックグラウンドで進めたままで、ユーザーが他の操作を自由に実行できます。詳細については、「Visual Basic におけるマルチスレッド」を参照してください。

Visual Basic 2005 には **BackgroundWorker** というコンポーネントもあります。これを使用すると、タスクをバックグラウンドで簡単に実行できます。詳細については、「チュートリアル : バックグラウンド操作を使用するフォームの実装」を参照してください。

XML ドキュメントの作成

XML ドキュメントを使用すると、クラスまたはユーザー コントロールにコメントを追加でき、他のプログラマがその使用方法を把握できます。たとえば、自作のユーザー コントロールに、"Stretch" という名前のプロパティがあるとします。これが何をするプロパティなのかは、名前からは判断できません。XML ドキュメントを使用すると、"見出しいっぱいにテキストが広がるかどうかを決定する" のような説明を追加できます。この説明は [プロパティ] ウィンドウと IntelliSense に表示されます。詳細については、「XML の使用によるコードのドキュメントの作成 (Visual Basic)」を参照してください。

自作のプログラムと .NET Framework のインストール

Visual Basic 2005 で作成したプログラムを実行するコンピュータには、.NET Framework のランタイムがインストールされている必要があります。また、プログラムによっては、他のファイルや必要条件が併せて必要な場合もあります。ClickOnce の発行機能を使用してプログラムを共有する場合には、ブートストラップという機能を使用してこれらのファイルを含めることができ、必要に応じて自動的にインストールできます。詳細については、「方法 : ClickOnce アプリケーションと共に必須コンポーネントをインストールする」を参照してください。

次の手順

このレッスンでは、Visual Basic 2005 で実行できることをいくつか説明しました。もちろん、ここで取り上げたのはごく一部にすぎません。Visual Basic 2005 を使い続けていくうちに、それまで知らなかった手法をいくつも発見できます。

次のレッスンでは、Visual Basic 2005 の他のユーザーに接続する方法について説明します。

次のレッスン : 「コミュニティ リソース : 友人からのちょっとした手助け」

参照

処理手順

生産性の向上 : Rapid Application Development

概念

さらに上のステップへ : 次の学習

Visual Basic Express での操作方法

その他の技術情報

Visual Basic ガイド ツアー

コミュニティ リソース：友人からのちょっとした手助け

プログラミングを習得するのは必ずしも簡単ではありません。たとえ経験豊富なプログラマでも、自力では解決できない問題に直面します。さいわい、Visual Basic を使用しているプログラマは数百万人にのぼり、その中には、他のプログラマの役に立ちたいという意欲に満ちている人が大勢います。

メモ：
Visual Basic Express Edition を使用している場合、インストール時に選択したオプションによっては、このページのヘルプリンクの一部を利用できないことがあります。詳細については、「Visual Basic Express のトラブルシューティング」を参照してください。

フォーラムを利用した情報交換

Visual Basic で助けを得る最適な方法の 1 つが、オンライン フォーラムを利用する方法です。オンライン フォーラムとは、ユーザーが互いの質問に回答し合う形の仮想コミュニティです。[ヘルプ] ツール バーには [質問] ボタンがあります。このボタンをクリックすると、MSDN コミュニティ フォーラムの検索ページが表示されます。質問を入力すると検索が実行され、関連する質問に対する過去の回答が表示されます。

メモ：
アクティブなインターネット接続が必要です。また、プロキシ サーバーやファイアウォールを使用している場合には、MSDN コミュニティ フォーラムへのアクセスが可能となるように設定する必要があります。

フォーラム内での個々の質問や回答のことをポストと呼び、単一のトピックに関する一連のポストのことをスレッドと呼びます。自分の質問に対する回答が過去のスレッドに見つからない場合は、自らの質問をメッセージとしてフォーラムにポストし、後でそのスレッドを確認して、回答を寄せてくれた人がいるかどうかをチェックできます。

フォーラムを使用するうえでのいくつかのヒントを以下に示します。

- まずは Visual Basic ドキュメントを読みます。トピックがヘルプで説明されている場合には、他のユーザーがそれについて代わりに調べてくれることを期待しないでください。一方、ヘルプにある説明が理解できない場合は、質問してみましょう。
- フォーラムを検索し、同じ質問やその回答が既に掲載されていないかどうかを確認します。よくある質問と同じ内容を投稿した場合、ポストが無視される場合があります。
- 適切なフォーラムを選択します。Visual Basic 言語についての質問を Windows フォーム フォーラムにポストすると、Visual Basic 言語のフォーラムにポストした場合に比べて、回答を得られる可能性が低くなります。
- ポストには、内容がよく伝わるタイトルを付けます。"教えてください" というタイトルよりも、"背景が透明なユーザー コントロールの作成方法を教えてください" というタイトルの方が、反応を得られる可能性が高くなります。
- 質問の内容を明確に表現します。何をしようとしていて、何がうまくいっていないかを説明します。また、既に試した方法とその結果も記述します。
- 必要に応じてコードを含めます。特定のコード行でエラーが生じる場合は、その行で何を実現しようとしているのかが明確になるような、十分なコードを含めます。ただし、アプリケーション全体をポストして、だれかがデバッグしてくれることを期待しないでください。
- 自分が質問するのであれば、他の人の質問にも回答するよう心がけます。コミュニティ内で他の人を進んで助けようとしていれば、他の人から助けてもらえる可能性も高くなります。
- 常に礼儀正しくふるまいます。自分の質問に回答をもらった場合は、お礼を述べます。フォーラムに回答を寄せている人は、みな自発的にそうしているのだということを忘れないでください。

MSDN コミュニティ フォーラム以外にも、Visual Basic、Windows フォーム、およびその他の技術に関連するフォーラムはたくさんあります。そのうちのいくつかは、Visual Basic 2005 で [コミュニティ] メニューの [Codezone コミュニティ] をクリックすると利用でき、またオンラインで検索することもできます。

やってみよう

フォーラムを利用するには

1. Visual Basic 2005 で [コミュニティ] メニューの [質問] をクリックします。

   MSDN コミュニティ フォーラムの検索ページが表示されます。

2. [検索] フィールドに「`VB Transparent User Control`」と入力し、[実行] をクリックします。

以前のポストの中に、キーワードに一致するものがある場合は表示されます。

3. ポストを確認して、透明なユーザー コントロールの作成方法を説明しているものがあるかどうか調べます。
4. 自分の質問に対して回答が得られたかどうかを確認するには、[コミュニティ] メニューの [質問の状態確認] をクリックします。

ヒント
スレッドに応答があったときに、電子メール、インスタント メッセンジャー、または携帯電話に対するテキスト メッセージで通知を受けることも選択できます。フォーラムに対して、ポストまたはポストに対する返信を入力するときには、[返信があったら通知する] ボタンをクリックするだけです。

その他のコミュニティ リソース

Visual Basic 2005 の使用方法を習得するうえで助けが得られるコミュニティ リソースは、オンライン フォーラム以外にも数多くあります。

Visual Basic Express Edition を起動したときに表示される [スタート ページ] には [ニュース] セクションがあり、オンライン チャットや Web キャストなど、だれでも参加できるコミュニティ イベントに対するリンクが表示されます。これらのリンクは定期的に更新されます。

ヒント
[スタート ページ] のリンクは RSS フィードによって実現されており、表示するリンクの種類を選択できます。詳細については、「方法 : スタート ページのニュース セクションをカスタマイズする」を参照してください。

ヘルプの検索機能では、ローカルのヘルプ システムに加えて、Visual Basic 向けのオンライン リソースのコレクションである MSDN オンライン ライブラリおよび Codezone コミュニティも検索できます。MSDN オンライン ライブラリには、新しいバージョンのヘルプ トピックや、ドキュメントに含まれていない文書が含まれている場合があります。Codezone コミュニティを構成する Web サイトには、多数の Visual Basic のエキスパートによる有用な情報が掲載されています。

メモ :
MSDN オンライン ライブラリや Codezone コミュニティにアクセスするには、アクティブなインターネット接続が必要です。また、プロキシ サーバーやファイアウォールを使用している場合には、アクセスが可能となるように設定する必要があります。加えて、ヘルプ設定で、オンライン アクセスに合わせた構成に設定します。詳細については、「方法 : 検索結果と F1 ヘルプの結果にオンライン コンテンツを含める」を参照してください。

Microsoft の Visual Basic チームは、Visual Basic 2005 の改良案、製品のバグ報告、ドキュメントに関する提案など、利用者からの意見をいつでも歓迎しています。

Product Feedback Center では、製品やドキュメントについての提案やバグ報告を行ったり、他の人が行った提案や報告を参照したりできます。また、自分が行った提案やバグ報告の進行状況の追跡や、提案された機能に対する投票なども可能です。Product Feedback Center にアクセスするには、[コミュニティ] メニューの [フィードバックの送信] をクリックします。

Microsoft Visual Basic デベロッパー センターには、Visual Basic に関する文書やその他の情報があり、頻繁に更新されています。デベロッパー センターにアクセスするには、[コミュニティ] メニューの [ディベロッパー センター] をクリックします。

CodeWise コミュニティ サイトは、Microsoft .NET Framework および Microsoft Visual Studio 2005 の独立したエキスパートによる、サードパーティのヒント、サンプル プログラム、アドバイス、およびニュースのリソースです。CodeWise コミュニティ サイトにアクセスするには、[コミュニティ] メニューの [Codezone コミュニティ] をクリックします。

ユーザー グループは、似たような興味を持つプログラマどうしのローカルな集まりです。Visual Basic および Visual Basic 2005 をテーマとするユーザー グループは世界中に数多くあります。大半のユーザー グループは、定期的に顔を合わせ、コーディング手法の発表、情報交換、他のプログラマとの交流などを行っています。大都市またはその近郊に住んでいる場合には、Visual Basic のユーザー グループが近くにある可能性があります。

次の手順

このレッスンでは、世界中の Visual Basic コミュニティで利用できるリソースの一部について説明しました。次のレッスンでは、Visual Basic 2005 についての理解を深めるためのリソースについて説明します。

次のレッスン : 「次のステップへ : Visual Basic の学習リソース」

参照
処理手順
ヒント : 知られていない方法
概念
さらに上のステップへ : 次の学習

Visual Basic Express での操作方法
その他の技術情報
Visual Basic ガイド ツアー

次のステップへ：Visual Basic の学習リソース

Visual Basic ガイド ツアーのすべてのレッスンを完了すると、自作プログラムの開発を始めることができる程度の習熟度に達します。しかし、当然ながらエキスパートとは呼べません。経験豊富な Visual Basic の専門家ですら、学習に終わりはありません。Visual Basic 2005 についての知識を完全に備えている人はいないのです。以下に、Visual Basic について理解を深めるためのいくつかの提案を示します。

メモ：

Visual Basic Express Edition を使用している場合、インストール時に選択したオプションによっては、このページのヘルプ リンクの一部を利用できないことがあります。詳細については、「Visual Basic Express のトラブルシューティング」を参照してください。

操作中に手助けを得る方法

多くの人にとって、実践してみることが最良の習得方法です。Visual Basic Express Edition IDE (統合開発環境) を操作しているときに手助けを得る方法として最も簡単なのは、F1 キーを押して状況依存のヘルプを表示する方法です。

- IDE のいずれかのウィンドウを選択しているときに F1 キーを押すと、ウィンドウとその使用方法を説明するヘルプ トピックが表示されます。
- デザイン モードでコントロールを選択しているときに F1 キーを押すと、そのコントロールについてのヘルプ トピックが表示されます。
- [プロパティ] ウィンドウで F1 キーを押すと、選択しているプロパティについてのヘルプが表示されます。
- コード エディタで F1 キーを押すと、現在のカーソル位置に最も近い言語キーワードについての言語リファレンスのトピックが表示されます。

やってみよう

状況依存のヘルプを表示するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [テンプレート] ペインで、[Windows アプリケーション] をクリックします。
3. フォームを選択し、[プロパティ] ウィンドウで **KeyPreview** プロパティを選択します。
4. F1 キーを押します。

   KeyPreview プロパティについて説明するヘルプ トピックが表示されます。

   ヒント

   ヘルプ トピックによっては、Visual Basic 以外の言語の構文およびコード例が含まれていることがあります。Visual Basic のみの構文およびコードを表示するには、ヘルプ トピックの最上部の [言語のフィルタ] をポイントし、ドロップダウン リストで、[Visual Basic (使用法)] 以外のすべてのエントリについて、チェック ボックスをオフにします。設定は、次に変更するまで保持されます。

5. フォームをダブルクリックしてコード エディタを開き、**Form1_Load** イベント ハンドラに次のコードを入力します。

   ```
   Dim CurrentUser As String
   CurrentUser = My.User.Name
   ```

6. `Name` という語の中にカーソルを配置し、F1 キーを押します。

   My.User.Name プロパティについてのヘルプ トピックが表示されます。

7. `User` という語の中にカーソルを配置し、再度 F1 キーを押します。

   My.User オブジェクトについてのヘルプ トピックが表示されます。

製品のドキュメント

Visual Basic 2005 の製品ドキュメントには、Visual Basic の習得に役立つ膨大な情報が含まれています。"方法" のトピック、チュートリアル、および詳細な概念情報が多数あるのに加え、.NET Framework のすべてのメンバについての参照トピックが含まれています。多少の時間をかけてドキュメントに目を通すのは、Visual Basic 2005 の未知の機能について学習するのによい方法です。

Visual Basic Express Edition のインストール時に行った選択によっては、一部のトピックを利用できない場合があります。Visual Basic Express Edition に含まれているヘルプ ファイルは、Visual Studio 2005 Express Edition の MSDN ライブラリのサブセットであり、これはさら

に、完全版の MSDN オンライン ライブラリのサブセットです。インストール時に、Visual Studio 2005 Express Edition の MSDN ライブラリをインストールするというオプションがありました。そのオプションを選択しなかった場合は、ここでインストールすることを検討してください。

MSDN オンライン ライブラリでは、追加的な製品ドキュメントおよび Visual Studio 2005 Express Edition の MSDN ライブラリに含まれているトピックの更新バージョンを参照できます。MSDN オンライン ライブラリにアクセスするには、[コミュニティ] メニューの [ディベロッパー センター] をクリックし、Microsoft Visual Basic デベロッパー センターのホーム ページで [ライブラリ] をクリックします。

メモ :

MSDN オンライン ライブラリにアクセスするには、アクティブなインターネット接続が必要です。また、プロキシ サーバーやファイアウォールを使用している場合には、アクセスが可能となるように構成する必要があります。

オンライン情報源

Visual Basic を使用しているプログラマは世界中で数百万人にのぼります。そのため、Visual Basic の習得を深めるためのオンライン情報源が膨大にあるというのは驚くことではありません。Visual Basic というキーワードを MSN で検索してみると 4900 万以上の結果が返り、Visual Basic 2005 というキーワードでは 200 万以上の結果が返ります。それだけ多くのオンライン情報源がある中で、どこから始めればよいでしょうか。

習得のためのオンライン リソースを最初に探す場所としては、[コミュニティ] メニューから利用できる Microsoft Visual Basic デベロッパー センターが最適です。デベロッパー センターでは、Visual Basic についての新しい文書や、オンラインの書籍、マルチメディアおよび Web キャスト、チュートリアルなどへのリンクが常に更新されています。

最初に参照する場所としては、CodeWise Community も適しており、こちらも [コミュニティ] メニューから利用できます。CodeWise Community のサイトは、Visual Basic および Visual Basic 2005 の独立したエキスパートによる、サードパーティのヒント、サンプル プログラム、アドバイス、およびニュースが掲載されているオンライン リソースです。

CodeWise Community のメンバは、Microsoft の開発者向けツールや技術に対し、確固たる専門知識を持っています。そうしたメンバからは、オンライン コミュニティを通じて広範なコンテンツが提供され、また自分の意見を述べたり他の人から学んだりする機会が得られます。

その他のリソース

従来からある方法で習得する場合には、そのための方法も数多くあります。

Visual Basic 2005 についての書籍は多数出版されています。その種類は、初心者レベルの書籍から、Visual Basic 2005 の特定の用途 (ゲーム プログラミングなど) についての書籍に至るまで多岐にわたります。地元の図書館や書店、またはオンライン書店でコンピュータ プログラミングのコーナーを眺めると、現在入手可能な書籍を把握できます。

ヒント

書籍を選択するときには、その中で取り上げられているのが Visual Basic 2005 であり、旧バージョンの Visual Basic ではないことを確認してください。旧バージョン向けに書かれた記事の大部分は今でも当てはまりますが、場合によっては、Visual Basic 2005 では、新しい手法やより優れた手法を利用できることがあります。

各種大学や専門学校の中には、Visual Basic 2005 についてのコースを持つところが数多くあり、Visual Basic Express Edition 専門のコースを持つところもあります。地元でどのようなコースが開講されているかを確認してみてください。

次の手順

このレッスンでは、Visual Basic の習得を継続していくためのいくつかの方法について説明しました。次のレッスンでは、サンプル コードを検索および使用する方法について説明します。

次のレッスン :「流用可能 : サンプルとスタート キット」

参照

処理手順

コミュニティ リソース : 友人からのちょっとした手助け

概念

さらに上のステップへ : 次の学習

Visual Basic Express での操作方法

その他の技術情報

Visual Basic ガイド ツアー

詳細解説：より詳しい情報の参照

Visual Basic ガイド ツアーでは、Visual Basic Express Edition および Visual Basic 2005 で実現できることをいくつか説明しましたが、これはほんの一部にすぎません。ガイド ツアーに登場した話題を深く掘り下げるには、ここから始めます。

メモ：

Visual Basic Express Edition を使用している場合、インストール時に選択したオプションによっては、このページのヘルプ リンクの一部を利用できないことがあります。詳細については、「Visual Basic Express のトラブルシューティング」を参照してください。

Visual Basic 言語についての理解を深めるには

プログラム構造とコード規則

Visual Basic の基本的な構造およびコード規則 (名前付け規則、コード内のコメント、Visual Basic の制限など) についてのドキュメントが含まれています。

Visual Basic 言語の機能

Visual Basic の基本的なコンポーネントの概要を示します。

Visual Basic における宣言された要素

変数、定数、列挙型、構造体、プロパティ、メソッド、プロシージャ、プロシージャの引数、関数の戻り値、イベント、デリゲート、インターフェイス、およびクラスについて説明します。

Visual Basic におけるプロシージャ

Sub、**Function**、**Property**、および **Operator** の各プロシージャについて説明します。また、再帰プロシージャやオーバーロードされたプロシージャなどの高度なトピックについても説明します。

Visual Basic における制御フロー

プログラムの実行フローを制御する方法について説明します。

My による開発

新しい機能 **My** について説明します。これは、アプリケーションとその実行時環境に関連する情報および既定のオブジェクト インスタンスへのアクセスを提供します。

Windows フォームおよびコントロールについての理解を深めるには

Windows フォームについて

Windows フォームのパワーを利用してデータの表示やユーザー入力の処理を行う方法、およびアプリケーションを簡単に、またセキュリティを強力に保護しながら配置する方法を詳細に説明します。

Windows フォーム アプリケーションの拡張

ワールドクラスの Windows ベースのアプリケーションを作成できる拡張機能について説明します。

Windows アプリケーション (Visual Basic での操作方法)

Windows ベースのアプリケーションの開発に関する Visual Basic の一般的なタスク全般を扱ったヘルプへのリンクを紹介しています。

Windows フォームで使用するコントロール

Windows フォームで使用できるコントロールおよびコンポーネントの一覧をアルファベット順で示します。

Windows フォーム コントロールの機能別一覧

Windows フォーム コントロールとコンポーネントを機能ごとにまとめて示します。

Windows フォームと Windows フォーム コントロールの新機能

このリリースの Visual Basic 2005 で使用できる新機能と強化された機能を紹介します。

デバッグとエラー処理についての理解を深めるには

Visual Basic アプリケーションのデバッグ

このページでは、Visual Basic 2005 に内蔵されているデバッグ機能を説明するドキュメントへのリンクを示します。

方法 : 例外処理アシスタント を使用してランタイム エラーを修正する

例外処理アシスタント は、実行時に例外がデバッガで発生するたびに表示され、例外の種類、トラブルシューティングのヒント、例外処理アシスタント で適用できる修正操作が示されます。

方法 : エディット コンティニュの中断モード時に編集を適用する

エディット コンティニュを使用すると、中断モードでコードを編集した後、コードを停止したり再起動したりせずにデバッグを継続できます。

方法 : 単純なブレークポイントを設定する

Visual Studio デバッガには、ブレークポイントを設定する方法が数多くあります。ここでは、単純なブレークポイントを設定する簡単な方法を 2 つ紹介します。

方法 : 自動修正でコンパイラ エラーを修正する

スマート コンパイル自動修正機能を使用して、コード エディタでコンパイラ エラーを修正する方法を示します。

Visual Basic の構造化例外処理の概要

Visual Basic の構造化例外処理の説明と例を示します。

データベースについての理解を深めるには

Visual Basic アプリケーションにおけるデータ アクセス

Visual Basic には、データにアクセスするアプリケーションを開発する際に役立ついくつかの新機能が用意されています。

データの表示の概要

フォーム上のコントロールにデータをバインドすることにより、アプリケーションのユーザーに対してデータを表示します。[データ ソース] ウィンドウから Windows フォームにアイテムをドラッグして、データ バインド コントロールを作成します。

データ ソースの概要

データ ソースは、アプリケーションが利用できるデータを表し、[データ ソース] ウィンドウに表示されます。

TableAdapter の概要

TableAdapters を使用すると、アプリケーションとデータベース間で通信できます。

チュートリアル : Windows アプリケーションのフォームでの関連データの表示

2 つの関連するテーブルからデータを表示する方法、およびそのデータを Windows フォームに表示する方法を、段階ごとに説明します。

データの検証

データセットは、データ行の変更時に値として生成される特定のイベントを用意します。特定のアプリケーション要件に基づいてこれらの変更を検証するために、イベント ハンドラを作成できます。

オブジェクト指向プログラミングについての理解を深めるには

Visual Basic のオブジェクトの概要

オブジェクト指向プログラミングで使用される用語や概念を紹介します。

オブジェクトの作成と使用

クラスのインスタンスの作成方法および使用方法について説明します。

オブジェクトのグループの管理

オブジェクトの配列やコレクションを使用する方法を紹介します。

クラスについて

オブジェクトの作成と有効期間について、詳しい手順を交えて説明します。

Visual Basic におけるインターフェイス

インターフェイスの概要とアプリケーションでの使用方法を説明します。

Visual Basic の継承

他のクラスの基本クラスとして機能するクラスの定義方法について説明します。

ユーザー コントロールとコンポーネントについての理解を深めるには

Property ステートメント

プロパティの値を設定および取得するためのプロパティ プロシージャ、およびプロパティの名前を宣言します。

Visual Basic でのコンポーネントの作成および使用

"コンポーネント" という用語を定義し、コンポーネントをいつどのように作成するかについて説明します。

方法 : ActiveX コントロールを操作する

既存の ActiveX コントロールを使用して Visual Studio ツールボックスに機能を追加する方法を説明します。

Visual Basic の定数と列挙体

変化しない値を繰り返し使用するために格納する方法について説明します。

列挙型の概要

関連する定数値のセットを格納して名前を付ける方法について説明します。

グラフィックスについての理解を深めるには
Windows フォームにおけるグラフィックスと描画

Windows フォーム内でグラフィックス インターフェイスを使用する方法について説明するトピックへのリンクを示します。

グラフィックス プログラミングについて

GDI+ を使用して基本的なタスクを実行する方法について説明します。

フォントとテキストの使用

テキストの描画と、フォントとフォント ファミリを使用する方法について説明します。

グラフィックス コンテナの使用

グラフィックス オブジェクトの状態と、入れ子になったグラフィックス コンテナを管理する方法について説明します。

ダブル バッファリングされたグラフィックス

ダブル バッファリングを使用してちらつきを抑える方法について説明します。

組み込みのオーナー描画サポートを備えたコントロール

Windows フォームのオーナー描画とは、特定のコントロールの外観を変更するためのテクニックであり、カスタム描画とも呼ばれます。

配置についての理解を深めるには
ClickOnce の配置の概要

ClickOnce の目的と、Windows フォーム アプリケーションをクライアント コンピュータに配置する際の重要な問題がどのように解決されるかについて説明します。

ClickOnce 配置ストラテジの選択

ClickOnce アプリケーションをクライアント コンピュータに配置するさまざまな方法を、それぞれの利点と制限事項を含めて説明します。

ClickOnce の更新方法の選択

インストール後の ClickOnce アプリケーションを更新するために使用できるさまざまなオプションについて、自動更新サブスクリプションやオンデマンド更新を含めて説明します。

ClickOnce 配置のトラブルシューティング

ClickOnce アプリケーションを配置するときに発生する一般的な問題とその解決方法を説明します。

参照
処理手順
流用可能 : サンプルとスタート キット
概念
さらに上のステップへ : 次の学習
Visual Basic Express での操作方法
その他の技術情報
Visual Basic ガイド ツアー

流用可能：サンプルとスタート キット

コードを作成するよりも効率的なのは、コードを作成しないで済むことです。Visual Basic 2005 ではヘルプ トピック、サンプル アプリケーション、およびスタート キットに含まれるコードを簡単に再利用できます。

メモ：
Visual Basic Express Edition を使用する場合、インストールの際に選択したオプションによっては、このページにあるヘルプへのリンクが一部使用できない場合があります。詳細については、「Visual Basic Express のトラブルシューティング」を参照してください。

コードのコピー

レッスンを行っている間に気付いたかもしれませんが、ヘルプ トピックにコード ブロックが紹介されている所には必ず、"コードのコピー" という名前の付いたリンクがあります。ヘルプ トピック内の [コードのコピー] リンクをクリックすると、そのブロックのコードがクリップボードにコピーされるので、それをコード エディタに入力する代わりに、直接貼り付けることが可能です。

多くのヘルプ トピックに、特定の言語の要素、プロパティ、または関数の使い方を示すコード例が紹介されています。それらのコード例が、ユーザーの要求に完全に一致するとは限りませんが、まずはコード例を自分のコードにコピーすることから始めて、変数名や参照を変更しながら各自の要件に合わせてコードを修正することができます。

サンプル アプリケーションの使用

Visual Basic 2005 には、プログラムの作成時に有利なスタートを切ることができるサンプル アプリケーションが多数用意されています。サンプル アプリケーションは、Visual Basic 2005 に読み込んで実行できる、修正の必要のない完成されたプロジェクトです。それほど有用なプログラムではないことも多いかもしれませんが、便利なコードを多く含むプログラムも少なくありません。

たとえば、Game サンプル アプリケーションは、GDI+ グラフィックス、タイマ機能、ユーザー構成、およびハイ スコア保存の機能を備えた、Windows フォームの簡単なゲームを紹介しています。カード ゲームを作成する場合、この Game サンプルのユーザー インターフェイスのコードはおそらくあまり役に立ちません。しかし、高得点を保存および表示するコードは非常に役に立つ可能性があるので、そのコードをそのまま作成中のプログラムにコピーして使用できます。

または、サンプル アプリケーションを出発点として利用し、各自の要件に合わせて修正および追加を行うという方法もあります。サンプルが要求に完全には一致しなくても、多くの場合、コードの作成を一から始めるよりは簡単です。詳細については、「Visual Basic Express のサンプル アプリケーション」を参照してください。

スタート キット

スタート キットはサンプル アプリケーションと似ていますが、大半のサンプルと異なり、それ自体で使用可能な完成されたアプリケーションです。たとえば、ムービー コレクション スタート キットは DVD の映画コレクションを継続的に管理するための完成されたアプリケーションです。

サンプル アプリケーションと同様、スタート キットもプログラム作成のスタート地点として使用できます。映画ではなく音楽のファイルを継続的に管理するプログラムを作成する場合、ムービー コレクションを用途に合わせて簡単にカスタマイズできます。スタート キットにはサンプル コードの他、カスタマイズに役立つアドバイスも紹介されています。詳細については、「スタート キット : いますぐ始めよう」を参照してください。

自作のコードの再利用

プログラムの作成をしばらく続けていると、同じコードをおそらく何度も作成することになります。たとえば、数字のみを入力できるように、**TextBox** コントロールのコードを作成することがあります。すべてのプログラムに対して同じコードを記述するのではなく、コードを一度だけ作成し、それをツールボックスにドラッグして保存します。次回、そのコードが必要になったとき、コード エディタにドラッグすることで作成中のコードに挿入できます。

ヒント
ツールボックス に保存したコードが多くなると、必要なコードを探し出すのが難しくなる場合があります。コードをコード スニペットとして保存すると、Visual Basic 2005 に付属のコード スニペットとまったく同じ方法で利用できます。

コードの検索

Visual Basic Express Edition に含まれるコードに加え、Microsoft およびその他の企業など、コードを入手できるソースは多数存在します。コード スニペット、テンプレートやスタート キット、サンプル、およびコントロールを検索するには、[コミュニティ] メニューの [コミュニティ検索] を選択します。詳細については、「方法 : コード スニペットをオンラインで検索する」を参照してください。

また、ヘルプを検索して、コードを含むトピックを探すこともできます。詳細については、「方法 : サンプル コードを含むトピックを検索する」を参照してください。

次の手順

このレッスンでは、コードを自分で作成しなくても済むように、コードを見つけるためのいくつかの選択肢について説明しました。次のレッスンでは、Visual Basic Express Edition のさまざまな機能に関する詳細な情報を、どこで入手すればよいかについて説明します。

次のレッスン : 「詳細解説 : より詳しい情報の参照」

参照

処理手順

次のステップへ : Visual Basic の学習リソース

概念

さらに上のステップへ : 次の学習

Visual Basic Express での操作方法

その他の技術情報

Visual Basic ガイド ツアー

Visual Basic 6.0 ユーザー向けのヘルプ

上級の開発者は、これまで Visual Basic 6.0 の習得に多くの時間と努力を割いてきました。Visual Basic 2005 は前バージョンから根本的に変更されているため、一見すると最初から学習し直す必要があるように思われます。しかし、数多くの変更点はありますが、実質的には Visual Basic 6.0 での開発経験を活かすことができます。Visual Basic 6.0 の知識を活用することにより、Visual Basic 2005 を使用してすぐに成果を上げることができます。

これには、Visual Basic 6.0 に加えられたいくつかの変更点について理解し、一部の古い習慣を放棄する必要があります。言語規則の一部、デバッグや配置などの一部のタスク、さらに用語の一部も変更されていますが、その多くはわずかな変更です。最終的には、これらの変更により、Visual Basic 2005 はより使いやすく高性能になっています。

次に挙げるトピックは、Visual Basic 2005 の変更点を特定するのに役立ちます。

このセクションの内容

統合開発環境 (Visual Basic 6.0 ユーザー向け)

開発環境の変更点

プロジェクトの管理 (Visual Basic 6.0 ユーザー向け)

プロジェクト システムの変更点

Web プログラミング (Visual Basic 6.0 ユーザー向け)

Web 関連の開発の変更点

WebClass (Visual Basic 6.0 ユーザー向け)

WebClass プロジェクト (IIS アプリケーション プロジェクトとも呼ばれる) の変更点

データ アクセス (Visual Basic 6.0 ユーザー向け)

データ アクセスおよびデータ ツールの変更点

コンポーネントの作成 (Visual Basic 6.0 ユーザー向け)

コンポーネント作成の変更点

User コントロール (Visual Basic 6.0 ユーザー向け)

UserControl プロジェクト (ActiveX コントロール プロジェクトとも呼ばれる) の変更点。**PropertyBag** オブジェクトに関する情報へのリンクを示します。

Windows フォーム (Visual Basic 6.0 ユーザー向け)

フォームの変更点

デバッグ (Visual Basic 6.0 ユーザー向け)

デバッグの変更点

セットアップと配置 (Visual Basic 6.0 ユーザー向け)

セットアップおよび配置の変更点

ローカリゼーションとグローバリゼーション (Visual Basic 6.0 ユーザー向け)

ローカライゼーションの変更点

Windows API プログラミング (Visual Basic 6.0 ユーザー向け)

Windows API. へのアクセス方法の変更点

レジストリ アクセス (Visual Basic 6.0 ユーザー向け)

レジストリ値の設定および取得の変更点

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 で変更された定数の変更点

オブジェクト ライブラリと名前空間 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 のオブジェクト ライブラリと Visual Basic 2005 の名前空間の相違点と類似点

ランタイム ライブラリ (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 のランタイム ファイルと .NET Framework の共通言語ランタイムの相違点

関連するセクション

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 言語の変更点

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

コントロールの変更点

Visual Basic 6.0 からのアップグレード

アプリケーションの Visual Basic 2005 への変換

Visual Basic の新機能

Visual Basic 2005 の新機能

Visual Basic 6.0 ユーザー向けの新機能

Visual Basic 6.0 を使い慣れているユーザーにとって、Visual Basic 2005 は多くの新機能や大幅に改良された機能を備えているため、以前のバージョンに比べて Visual Basic での開発がより簡単かつ高性能になっています。次の一覧では、最大の変更点の一部を重点的に示し、詳細情報へのリンクを示します。

Visual Basic 言語の新機能

Visual Basic では、継承、インターフェイス、オーバーロードなどの言語機能の追加と改良を行い、高性能なオブジェクト指向プログラミング言語を実現しています。Visual Basic を使用して開発を行うことにより、真のマルチスレッド アプリケーションを作成できるようになりました。この他に Visual Basic 2005 に追加された言語機能には、構造化例外処理、カスタム属性、共通言語仕様 (CLS: Common Language Specification) への準拠などがあります。

Visual Basic 2005 では、継承、オーバーロード、**Overrides** キーワード、インターフェイス、共有メンバ、コンストラクタなど、多くの新しいまたは改良されたオブジェクト指向言語機能をサポートしています。

Visual Basic 言語の新機能の詳細については、「Visual Basic 言語の新機能 (Visual Basic 6.0 ユーザー向け)」を参照してください。

以前のバージョンから変更された特徴や機能の説明については、「言語の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。

My を使用した Visual Basic 開発

Visual Basic 2005 には、高性能を実現するだけでなく、生産性と使いやすさを向上させることを目的とした Rapid Application Development のための新機能が用意されています。こうした機能の 1 つに、**My** と呼ばれる機能があります。これにより、.NET Framework が備える機能のうちよく使用されるものにアクセスできるだけでなく、アプリケーションやその実行時環境に関連する情報や既定のオブジェクト インスタンスにもアクセスできます。この情報は、IntelliSense を通じて検出できる形式で編成され、使用に応じて論理的に表現されます。詳細については、「My による開発」を参照してください。

フォームとコントロールの新機能

Windows フォームは .NET Framework を実装する新しいオブジェクト指向のフレームワークです。Windows フォームは、Windows フォーム コントロールと共に、Visual Basic を使用した Windows ベースのアプリケーション開発のための堅牢なアーキテクチャを実現します。

フォームとコントロールの新機能の詳細については、「Windows フォームの新機能 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Visual Basic 6.0 のフォームに関する知識がある場合は、「フォームのタスク (Visual Basic 6.0 ユーザー向け)」で新概念に関する説明を参照してください。

Visual Basic IDE の新機能

Visual Basic 2005 の統合開発環境は、一見難しそうですが、使い始めてみると、生産性を高める多くの新しい機能を備えていることがわかります。

詳細については、「Visual Studio IDE の新機能 (Visual Basic 6.0 ユーザー向け)」を参照してください。

データの新機能

Visual Basic 2005 には、データにアクセスするアプリケーションの開発を支援するための新機能がいくつか用意されています。データ ソース構成ウィザード は、アプリケーションから、データベース、Web サービス、およびユーザーが作成したオブジェクトに含まれるデータへの接続を簡素化します。新しい [データ ソース] ウィンドウ は、プロジェクトで使用できるデータ、およびプロジェクトに関連するデータを表示する中心的な場所です。このウィンドウからフォームに項目をドラッグすることによってデータ バインド コントロールを作成できるため、データ バインディングの複雑性が緩和されます。新しい Visual Studio で生成された TableAdapter の概要 オブジェクトを使用して、データセットへのデータの読み込み、クエリの実行、およびストアド プロシージャの実行ができるようになりました。新しいローカル データ機能により、Microsoft Access データベース ファイルおよび Microsoft SQL Server Express データベース ファイルをアプリケーションに直接組み込むことができるようになりました。新機能の一覧については、「データの新機能」を参照してください。

セットアップと配置の新機能

ClickOnce による配置などの新しい技術により、Visual Basic 2005 で作成されたアプリケーションの配置が、従来に比べより簡単かつ高性能になりました。

ClickOnce の配置により、自動的に更新される Windows ベースのアプリケーションおよびコンソール アプリケーションを発行できます。このアプリケーションは、Web アプリケーションと同じように簡単にインストール、更新、および実行できます。プロジェクト デザイナの新しいタブの [セキュリティ]、[署名]、および [発行] を使用すると、ClickOnce による配置をカスタマイズできます。[ビルド] に新たに追加された [発行] をクリックするか、ソリューション エクスプローラのショートカット メニューを使用して、発行ウィザードにアクセスできます。このウィザードでは、アプリケーションを発行するために必要な手順を実行します。詳細については、「ClickOnce の配置」を参照してください。

また、Visual Basic 6.0 のセットアップ ウィザードとディストリビューション ウィザードは、Visual Basic 2005 では、Windows インストーラ ベースの [セットアップと配置] プロジェクトに置き換えられました。詳細については、「セットアップと配置 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Visual Basic 6.0 で作成したアプリケーションのアップグレード

Visual Studio 2005 では、.NET Framework の利点を活かして開発を続けるために、Visual Basic 6.0 で作成されたアプリケーションをアップグレードできます。Visual Basic 6.0 のプロジェクト ファイル (.vbp) を初めて開くと、Visual Basic アップグレード ウィザードが表示されます。開発環境以外でプロジェクトをアップグレードするためのコマンド ライン ツールも用意されています。詳細については、「以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード」を参照してください。

参照

概念

Visual Basic の新機能

その他の技術情報

Visual Basic ガイド ツアー

Visual Basic 言語の新機能 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、継承、インターフェイス、オーバーロードなどの言語機能の追加と改良を行い、高性能なオブジェクト指向プログラミング言語を実現しています。このページでは、最大の変更点の一部を重点的に示し、詳細情報へのリンクを示します。

メモ :

Visual Basic 6.0 に関する知識がある場合は、「言語の変更点 (Visual Basic 6.0 ユーザー向け)」で新概念に関する説明を参照してください。

ヒント

Visual Basic 2005 の新しい言語機能に関する実践的な説明については、「Visual Basic ガイド ツアー」を参照してください。

新機能

以降では、Visual Basic 2005 の新しい言語機能について説明します。

継承

Visual Basic 2005 は、派生クラスの派生元クラスを定義できるようにすることで、継承をサポートしています。派生クラスは、基本クラスのプロパティとメソッドを継承します。また、それらを拡張することもできます。さらに、基本クラスから継承したメソッドを、新しい実装でオーバーライドすることもできます。既定では、Visual Basic 2005 で作成するクラスはすべて継承できます。デザインするフォームは実際にはクラスであるため、継承を使用して、既存のフォームに基づいて新しいフォームを定義できます。詳細については、「Visual Basic の継承」を参照してください。

例外処理

Visual Basic 2005 は、C++ などの他の言語によってサポートされている **Try...Catch...Finally** 構文の拡張バージョンを使用することで、構造化例外処理をサポートしています。構造化例外処理では、最新の制御構造 (**Select Case** や **While** に似た制御構造) と、例外、コードの保護ブロック、およびフィルタが組み合わされています。構造化例外処理により、堅牢かつ包括的なエラー ハンドラを備えたプログラムを、より簡単に作成および管理できるようになります。詳細については、「例外処理の概要」を参照してください。

オーバーロード

オーバーロードは、同じ名前を持ちながら異なったデータ型を使用するプロパティ、メソッド、プロシージャ、または演算子を定義する機能です。オーバーロードされたプロシージャを使用すると、必要な数の実装を提供して各種のデータを処理できる一方で、単一の多機能プロシージャとしての外観を持たせることができます。詳細については、「オーバーロードされたプロパティとメソッド」を参照してください。

プロパティとメソッドのオーバーライド

Overrides キーワードを使用すると、派生オブジェクトは親オブジェクトから継承した特性をオーバーライドできます。オーバーライドしたメンバは、基本クラスから継承したメンバと同じ引数と持っていますが、実装は異なります。メンバの新しい実装から親クラスの元の実装を呼び出すには、メンバ名の前に **MyBase** を付加します。詳細については、「プロパティとメソッドのオーバーライド」を参照してください。

コンストラクタとデストラクタ

"コンストラクタ" は、クラスの新しいインスタンスの初期化を制御するプロシージャです。反対にデストラクタは、クラスがスコープから外れたとき、または **Nothing** に設定されたときに、システム リソースを解放するメソッドです。Visual Basic 2005 では、**Sub New** プロシージャと **Sub Finalize** プロシージャを使用して、コンストラクタとデストラクタをサポートしています。詳細については、「オブジェクトの有効期間 : オブジェクトの作成と破棄」を参照してください。

データ型

Visual Basic 2005 では、3 つの新しいデータ型が導入されています。**Char** データ型は、符号なし 16 ビット値で、Unicode 文字を格納します。これは、.NET Framework の **System.Char** データ型に相当します。**Short** データ型は、符号付き 16 ビットの整数です。以前のバージョンの Visual Basic では、**Integer** と呼ばれていた型です。**Decimal** データ型は、符号付き 96 ビットの整数で、10 の累乗によって大きさが変化します。以前のバージョンの Visual Basic では、**Variant** 内でのみ使用できました。さらに Visual Basic では、符号なし整数データ型 (**UShort**、**UInteger**、および **ULong**)、および signed 型 (**SByte**) をサポートしています。詳細については、「Visual Basic におけるデータ型」を参照してください。

インターフェイス

インターフェイスは、クラスのプロパティとメソッドを表しますが、クラスとは異なり、実装が存在しません。インターフェイスを宣言するには、**Interface** ステートメントを使用します。また **Implements** ステートメントを使用すると、インターフェイスに記述された項目を実行するコードを記述できます。詳細については、「Visual Basic におけるインターフェイス」を参照してください。

デリゲート

デリゲートは、オブジェクトのメソッドを代理で呼び出すことができるオブジェクトで、タイプ セーフなオブジェクト指向の関数ポインタと表現されることもあります。デリゲートを使用すると、イベント発生時に実行されるイベント ハンドラ メソッドをプロシージャに指定できます。またデリゲートは、マルチスレッド アプリケーションでも使用できます。詳細については、「デリゲートと AddressOf 演算子」を参照してください。

共有メンバ

"共有メンバ" は、1 つのクラスのすべてのインスタンスで共有されるプロパティ、プロシージャ、およびフィールドです。共有データ メンバは、複数のオブジェクトで共通の情報を使用する必要がある場合に役立ちます。共有クラス メソッドは、先にクラスからオブジェクトを作成しなくても使用できます。詳細については、「Visual Basic の共有メンバ」を参照してください。

参照

参照を使用すると、他のアセンブリで定義されたオブジェクトを使用できます。Visual Basic 2005 では、参照はタイプ ライブラリではなくアセンブリを指します。詳細については、「参照と Imports ステートメント」を参照してください。

名前空間

名前空間を使用すると、クラス、インターフェイス、およびメソッドが階層で整理され、名前の競合が起こらなくなります。詳細については、「Visual Basic における名前空間」を参照してください。

アセンブリ

アセンブリは、特定のコンポーネントまたはアプリケーションに必要なすべてのファイルを定義することで、タイプ ライブラリに置き換わり、機能を拡張します。アセンブリには、1 つ以上の名前空間を含めることができます。詳細については、「アセンブリ」を参照してください。

属性

属性を使用すると、プログラム要素に関する追加情報を指定できます。たとえば、属性を使用すると、クラスが XML Web サービスとして使用される場合に、クラス内のどのメソッドを公開するかを指定できます。詳細については、「Visual Basic における属性」を参照してください。

マルチスレッド

Visual Basic 2005 を使用すると、複数のタスクを独立して実行できるアプリケーションを記述できます。他のタスクを停止させる可能性のあるタスクを、別のスレッドで実行できます。このプロセスをマルチスレッドと呼びます。マルチスレッドによって、複雑なタスクをユーザー インターフェイスとは別のスレッドで実行することにより、アプリケーションのユーザー入力に対する応答性が向上します。詳細については、「マルチスレッド アプリケーション」を参照してください。

ビット シフト演算子

Visual Basic 2005 は、整数データ型 (**Byte**、**Short**、**Integer**、および **Long**)、および unsigned 型 (**UShort**、**UInteger**、および **ULong**) について、左シフト演算子と右シフト演算子をサポートしています。数値のシフトは、循環的には行われません。つまり、一方の端からはみ出したビットが、もう一方の端に補われることはありません。また、対応する代入演算子も用意されています。詳細については、「ビット シフト演算子」および「代入演算子」を参照してください。

ループ変数の宣言

Visual Basic 2005 を使用すると、**For** ループまたは **For Each** ループの一部としてループ変数を宣言できます。同じ名前の変数がループの外で宣言されていなければ、**For** ステートメントまたは **For Each** ステートメントに、その変数の **As** 句を含めることができます。この方法で宣言されたループ変数のスコープは、そのループ本体です。詳細については、「For...Next ステートメント (Visual Basic)」および「For Each...Next ステートメント (Visual Basic)」を参照してください。

Visual Basic 2005 における Visual Basic 言語の新機能

Visual Basic 2005 における Visual Studio の新しい言語機能には、ループの継続、保証されたリソースの廃棄、混合アクセスに関するプロパティ、unsigned データ型、演算子のオーバーロード、ジェネリック型などがあります。詳細については、「Visual Basic 言語の新機能」を参照してください。

参照

概念

Visual Basic 6.0 ユーザー向けの新機能
Windows フォームの新機能 (Visual Basic 6.0 ユーザー向け)
Visual Studio IDE の新機能 (Visual Basic 6.0 ユーザー向け)

Windows フォームの新機能 (Visual Basic 6.0 ユーザー向け)

Windows フォームは .NET Framework を実装する新しいオブジェクト指向のフレームワークです。Windows フォームと Windows フォーム コントロールは、Visual Basic を使用した Windows ベースのアプリケーション開発のための堅牢なアーキテクチャを実現します。

メモ：
Visual Basic 6.0 のフォームに関する知識がある場合は、「フォームのタスク (Visual Basic 6.0 ユーザー向け)」で新概念に関する説明を参照してください。

新機能

次に示す内容はすべて、Windows フォームを使用したクライアント アプリケーション開発にかかわるものです。

Windows フォーム

Windows フォームは、.NET Framework に基づく、Microsoft Windows ベースのアプリケーション開発用の新しいプラットフォームです。このフレームワークでは、オブジェクト指向のわかりやすく拡張性の高いクラスのセットを使用することにより、高性能で多機能な Windows ベースのアプリケーションを作成できます。詳細については、「Windows フォームの概要」を参照してください。

Windows フォーム コントロール

Visual Basic 2005 には、Visual Basic 6.0 で使用されていたすべてのコントロールが含まれています。その大部分は、新しいプロパティ、メソッド、およびイベントによって拡張され、性能の向上を実現しています。また、ユーザー インターフェイス作成のための新しいコントロールとコンポーネントがいくつか追加されています。詳細については、「Windows フォームで使用するコントロール」を参照してください。

3 階層アプリケーションのプレゼンテーション層としての Windows フォーム

Windows フォームは、データベースに接続されたサーバーへの HTTP 呼び出しを使用することにより、クライアント アプリケーションのユーザー インターフェイスとアプリケーション サーバーの高度な処理をつなぐユーザー インターフェイスとして機能します。詳細については、「チュートリアル : 分散アプリケーションの作成」を参照してください。

クライアント アプリケーションの作成

Windows フォームでは、Visual Basic を使用して、リッチ クライアント アプリケーションを作成できます。作成されたアプリケーションでは、広範なデータ ソースへのアクセスを実現し、Windows フォーム コントロールによるデータ表示機能とデータ編集機能を提供できます。詳細については、「チュートリアル : 簡単な Windows フォームの作成」を参照してください。

Windows フォームのセキュリティ モデル

Windows フォームのセキュリティは、.NET Framework 内で確立されたセキュリティ ポリシーに基づいています。System.Security 名前空間は、アクセス許可の基本クラスなど、.NET Framework のセキュリティ システムの基底となる構造を定義します。詳細については、「Windows フォームのセキュリティ」を参照してください。

Tag プロパティと Name プロパティ

Tag プロパティと **Name** プロパティは、Control のメンバとして追加されました。**Tag** プロパティは、コントロールに関するデータを保持するメモリ内のリポジトリです。**Name** プロパティには、コントロールの名前が保持されます。**Name** プロパティを実行時に使用すると、オブジェクトを型やプログラム的な名前で評価するのではなく、このプロパティに設定された名前で評価できます。**Tag** プロパティの詳細については、「Control.Tag Property」を参照してください。**Name** プロパティの詳細については、「Control.Name Property」を参照してください。

四角形以外の Windows フォーム

Windows フォームの形状を簡単にカスタマイズできます。イメージ ファイルといくつかのプロパティを設定するだけで、四角形以外のフォームを作成できます。詳細については、「方法 : 成型された Windows フォームを作成する」を参照してください。

スマート デバイスのプログラミング

Visual Basic を使用して、リソースが限られているデバイスに対して多機能なアプリケーションを開発できるようになりました。Visual Studio 2005 および .NET Framework のサブセットである .NET Compact Framework に用意されているツールを使用して、スマート デバイスで動作するアプリケーションを作成、ビルド、およびデバッグできます。

Visual Basic 言語はほぼ全体がデバイス開発に対応していますが、Visual Basic 2005 の一部の機能は、デバイス アプリケーションとの本質的な違いに対応するために変更または削除されています。詳細については、「.NET Compact Framework 開発デスクトップとの違い」を参照してください。

Visual Studio 2005 の Windows フォームおよびコントロールの新機能

Visual Basic 2005 には、上記の機能以外にも、いくつかの新しいコントロールや、バックグラウンド処理、一部コントロールのカスタム描画など、

新しい機能や拡張がいくつか追加されています。詳細については、「Windows フォームと Windows フォーム コントロールの新機能」を参照してください。

参照

概念

Visual Basic 6.0 ユーザー向けの新機能
Visual Basic 言語の新機能 (Visual Basic 6.0 ユーザー向け)
Visual Studio IDE の新機能 (Visual Basic 6.0 ユーザー向け)

Visual Studio IDE の新機能 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の統合開発環境 (IDE) は、一見難しそうですが、使い始めてみると、生産性を高める多くの新しい機能を備えていることがわかります。このページでは、最大の変更点の一部を重点的に示し、詳細情報へのリンクを示します。

メモ :

Visual Basic 6.0 を使用した経験がある場合は、「統合開発環境 (Visual Basic 6.0 ユーザー向け)」を参照してください。

ヒント

Visual Basic 2005 の新機能に関する実践的な説明については、「Visual Basic ガイド ツアー」を参照してください。

新機能

Visual Basic 2005 の新しい IDE 機能を次に示します。

共有 IDE

Visual Basic、Visual C#、Visual C++ を含むすべての .NET 言語は、Visual Studio IDE の中でホストされます。単一の IDE を共有すると、さまざまな製品の類似ツールが、Visual Studio 全体で使われる共有ツール セットに統合されるなど、多くの利点があります。

ウィンドウの管理

Visual Studio では、多くのコードを一度に画面に表示することが以前よりも簡単にできるようになりました。

タブ付きドキュメントは自動的にドキュメント ウィンドウを IDE 内でタブ化します。たとえば、複数のドキュメントをエディタやデザイナで編集するとき、すべてのドキュメントは、マルチ ドキュメント インターフェイス (MDI: Multiple-Document Interface) 領域の上部にタブとして表示されます。

[自動的に隠す] 機能を使用すると、ウィンドウによって画面領域が無駄に使用されないように、ソリューション エクスプローラやツールボックスなどのツール ウィンドウを IDE の端に最小化できます。ツール ウィンドウを最小化すると、エディタが表示される領域を増やすことができます。

詳細については、「ウィンドウの管理」を参照してください。

編集機能

Visual Studio には、IDE 内のすべての言語で使用する統合されたコード エディタが用意されています。このコード エディタには各言語に特化した機能も含まれています。コード エディタには、ワード ラップ、インクリメンタル検索、コードのアウトライン表示、定義の折りたたみ、行の番号付け、カラー印刷、ショートカットなど、拡張した点がいくつかあります。これらの機能には、[編集] メニューやコンテキスト メニューからアクセスできます。詳細については、「コード編集の新機能」を参照してください。

コード スニペットは、Visual Basic プロジェクトに挿入できるサンプル コードの断片です。利用できるコード スニペットの一覧を表示するには、コード エディタでアクティブ ドキュメントを右クリックし、ショートカット メニューの [スニペットの挿入] をクリックします。必要なスニペットの名前をクリックすると、エディタにコードが挿入されます。このコードは必要に応じて変更できます。詳細については、「方法 : コード スニペットを管理する」を参照してください。

Office スマート タグと同様に、Visual Studio スマート タグを使用すると、作業の状況に応じて一般的なタスクを利用できます。たとえば、スマート タグを使用することで、Visual Basic の一般的なエラーの一部をボタンを 1 回クリックするだけで修正できます。

配置

ClickOnce の配置を使用すると、Web アプリケーションと同様に容易にインストールし、実行できる自己更新型の Windows ベースのアプリケーションを配置できます。Windows クライアント アプリケーションの他に、コマンド ライン アプリケーションも配置できます。[プロジェクト] メニューには、[プロジェクトの発行] コマンドが新しく用意されました。詳細については、「ClickOnce の配置」を参照してください。

.NET Framework ランタイムなどの必須システム コンポーネントを、配置プロジェクトや ClickOnce の配置に含めることができるようになりました。詳細については、「配置の必要条件 (Visual Studio)」を参照してください。

[セットアップと配置] プロジェクトを使用すると、Microsoft Windows インストーラのテクノロジを使用したアプリケーションの配布、製品サーバーやステージング サーバーへの配置、アプリケーションの複数の階層のテスト コンピュータへの配置、および ASP.NET Web アプリケーションの Web サーバーへの配置を実現できます。詳細については、「Windows インストーラ配置」を参照してください。

Microsoft Build Engine

Microsoft Build Engine (MSBuild) は、Microsoft および Visual Studio の新しいビルド プラットフォームです。MSBuild には、理解しやすく、拡張が容易で、Microsoft によって完全サポートされた XML ベースのプロジェクト ファイル形式が新たに採用されています。MSBuild のプロジェクト ファイル形式では、どの項目をどのような方法でビルドする必要があるかについて、プラットフォームや構成の違いまで含めて、開発者が

的確に定義できるようになっています。また、異なるファイルに適用できる規則を記述しておき、製品を構成するさまざまなプロジェクトで再利用することにより、一貫したビルド作業を行うことができます。詳細については、「MSBuild」を参照してください。

Visual Studio IDE の新機能 (Visual Basic 2005 ユーザー向け)

このバージョンの Visual Studio では、定義済みの設定、タスク一覧とエラー リストの強化、ドッキング動作の向上、IDE ナビゲータ ウィンドウなど、Visual Basic 2005 の新しい IDE 機能が多く用意されています。詳細については、「開発環境の新機能」を参照してください。

参照

概念

Visual Basic 6.0 ユーザー向けの新機能
Visual Basic 言語の新機能 (Visual Basic 6.0 ユーザー向け)
Windows フォームの新機能 (Visual Basic 6.0 ユーザー向け)
Visual Basic の新機能

イベントおよびイベント処理 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 のイベントやイベント処理に慣れていると、最初は Visual Basic 2005 のイベント モデルに戸惑うかもしれませんが、Visual Basic 6.0 よりも機能は簡単になり、ずっと強力になっています。

概念の違い

Visual Basic 6.0 では、イベントが特定のオブジェクトに結び付けられ、個別のオブジェクトにイベント処理コードが用意されています。たとえば、ボタンとメニューがあるフォーム上では、それぞれが個別の **Click** イベントを保持しています。このため、両者がまったく同じ機能を実行する場合でも、イベント ハンドラに別個にコードを記述する必要があります。

```
' Visual Basic 6.0
Private Sub HelpButton_Click()
    HelpButton.Caption = "Help me!"
End Sub
Private Sub HelpMenu_Click()
    HelpMenu.Caption = "Help me!"
End Sub
```

Visual Basic 2005 では、イベントはデリゲートを介してイベント ハンドラに結び付けられます。したがって、複数のオブジェクトがある場合でも 1 つのイベント ハンドラを作成するだけで済みます。

VB

```
Private Sub HelpButton_Click(ByVal sender As Object, ByVal e As _
System.EventArgs) Handles HelpButton.Click, HelpMenu.Click
    sender.Text = "Help me!"
End Sub
```

上記の例では、処理するイベント (この例では **HelpButton** オブジェクトと **HelpMenu** オブジェクトの **Click** イベント) をイベント宣言内の `Handles` 句で定義しています。オブジェクトとイベントが同じ型である必要はありません。たとえば、ボタンの **Click** イベント、テキスト ボックスの **DoubleClick** イベント、およびタイマの **Tick** イベントを 1 つのイベント ハンドラで処理できます。

また、イベント宣言には `ByVal sender As Object` パラメータおよび `ByVal e As System.EventArgs` パラメータを指定することもできます。最初のパラメータ `sender` は、イベントを発生させたオブジェクトへの参照を示します。2 番目のパラメータ `e` では、処理されるイベントに応じた特定のオブジェクトを渡します。このオブジェクトのプロパティ (場合によってはメソッド) を参照すると、マウス イベントのマウスの位置や、ドラッグ アンド ドロップ イベントで転送されるデータなどの情報を取得できます。

以下のコード例では、**MouseDown** イベント ハンドラで `sender` パラメータを使用して、イベント生成元のオブジェクトの型を確認しています。オブジェクトが **PictureBox** の場合、`e` パラメータを使用して、クリックされた位置にラベルを移動します。このコードを複製するには、2 つの **PictureBox** コントロールと 1 つの **Label** コントロールをフォームに追加します。

VB

```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As _
System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown, _
PictureBox1.MouseDown, PictureBox2.MouseDown

    If TypeOf sender Is PictureBox Then
        Label1.Location = sender.Location + e.Location
    Else
        MsgBox("Please click a picture")
    End If
End Sub
```

通常、イベントごとに生成されるイベント ハンドラでは、2 番目に指定されるイベント オブジェクトの種類がそれぞれ異なります。**MouseDown** イベントや **MouseUp** イベントなどのイベント ハンドラでは、2 番目のパラメータは同じ **MouseEventArgs** オブジェクト型です。この種類のイベントでは、同じイベント ハンドラを使用して両方のイベントを処理できます。

複数のイベント オブジェクト型を渡すイベントの場合は、複数のイベント ハンドラを作成する必要があります。たとえば、**TextBox** コントロールの **TextChanged** イベントは、汎用の **EventArgs** イベント オブジェクトを渡しますが、**MouseDown** イベントは、それよりも特殊化された **MouseEventArgs** イベント オブジェクトを渡します。**MouseEventArgs** オブジェクトには、**Button** などのマウス イベント固有プロパティがあ

り、これらを使用してどのマウス ボタンが押されたのかを確認します。このようなプロパティは **TextBox** コントロールには適用されないので、参照しようとするとエラーが発生します。

イベント処理の概念上の違いに加え、Visual Basic 2005 では複数のオブジェクトで一部のイベントの名前と動作が異なります。詳細については、「Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)」を参照してください。

参照

概念

イベント ハンドラの概要 (Windows フォーム)

その他の技術情報

Windows フォーム内でのイベント ハンドラの作成

統合開発環境 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 は、Microsoft Visual Studio 2005 統合開発環境 (IDE: Integrated Development Environment) に完全に統合されています。この開発環境は、Visual Basic 6.0 IDE とはいくつかの点で異なります。

ウィンドウとレイアウトの変更点

Visual Basic 2005 におけるウィンドウの標準的な配置は、Visual Basic 6.0 とは異なります。Visual Basic 6.0 の配置を使用する場合は、ウィンドウを手動で配置することによって、Visual Basic 6.0 の配置を設定できます。次に IDE を開くと、配置が保持されます。

Visual Basic 6.0 では、IDE が既定で MDI (マルチ ドキュメント インターフェイス) レイアウトに設定されており、必要に応じて SDI (シングル ドキュメント インターフェイス) レイアウトに設定できました。Visual Basic 2005 では、IDE の既定のレイアウトがタブ付きドキュメント レイアウトになっています。SDI レイアウトはサポートされていません。MDI レイアウトに変更するには、[ツール] メニューの [オプション] を選択し、[環境] の [全般] タブの [マルチ ドキュメント] を選択します。

Visual Basic 6.0 では、[ツール] メニューの [オプション] を選択することによって、一部のツール ウィンドウのドッキング動作を制御できます。Visual Basic 2005 では、すべてのウィンドウに既定でドッキング機能が備わっています。この動作は、[ウィンドウ] メニューの [ドッキング可能] を使用して制御できます。

また Visual Basic 6.0 と Visual Basic 2005 では、使用できるウィンドウにもいくつかの相違点があります。

メモ :
Visual Basic 2005 では、一部のデバッグ ウィンドウは実行モードだけで使用できます。デザイン モードに切り替えると、これらのウィンドウは表示されなくなります。

Visual Basic 6.0 のいくつかのウィンドウとそれに相当する Visual Basic 2005 のウィンドウを次の表に示します。

Visual Basic 6.0	**Visual Basic 2005**
[カラー パレット]	新規に実装されました。
[データ ビュー]	新規に実装されました。サーバー エクスプローラに置き換えられています。
[フォーム レイアウト]	新規に実装されました。
[プロジェクト エクスプローラ]	[ソリューション エクスプローラ]
[ウォッチ]	[ウォッチ 1]、[ウォッチ 2]、[ウォッチ 3]、[ウォッチ 4]

メニューの変更点

Microsoft Visual Studio 2005 のメニュー コマンドは、すべての言語に対して標準化されています。Visual Basic 6.0 の一部のメニュー コマンドは、Microsoft Visual Studio 2005 では別のコマンドに置き換えられます。

Visual Basic 6.0 では、すべてのメニュー コマンドが常に表示され、コンテキストから外れたときには淡色表示になります。Visual Basic 2005 では、利用できるコマンドがコンテキストに応じて変化します。

Visual Basic 6.0 の一般的なコマンドと、それに相当する Visual Basic 2005 のコマンドを次の表に示します。

Visual Basic 6.0	**Visual Basic 2005**
[アドイン] - [アドイン マネージャ]	[ツール] - [アドイン マネージャ]
[デバッグ] - [ウォッチ式の追加]	[ウォッチ式の追加] (コンテキスト メニューのみ)
[デバッグ] - [ウォッチ式の編集]	新規に実装されました。ウォッチは [ウォッチ] ウィンドウで編集します。
[デバッグ] - [カーソル行の前まで実行]	[カーソル行の前まで実行] (コンテキスト メニューのみ)

[デバッグ] - [次のステートメントの設定]	[次のステートメントの設定] (コンテキスト メニューのみ)
[デバッグ] - [次のステートメントの表示]	[次のステートメントの表示] (コンテキスト メニューのみ)
[編集] - [入力候補]	[編集] - [IntelliSense] - [入力候補]
[編集] - [検索]	[編集] - [クイック検索]
[編集] - [次を検索]	[編集] - [クイック検索]
[編集] - [インデント]	[編集] - [詳細] - [行インデント]
[編集] - [ファイルの挿入]	[編集] - [テキストとしてファイルを挿入]
[編集] - [定数の一覧]	[編集] - [IntelliSense] - [メンバの一覧]
[編集] - [プロパティ/メソッドの一覧]	[編集] - [IntelliSense] - [メンバの一覧]
[編集] - [インデントを戻す]	[編集] - [詳細] - [行インデント解除]
[編集] - [パラメータ ヒント]	[編集] - [IntelliSense] - [パラメータ ヒント]
[編集] - [クイック ヒント]	[編集] - [IntelliSense] - [クイック ヒント]
[編集] - [置換]	[編集] - [クイック置換]
[ファイル] - [<プロジェクト名> の作成]	[ビルド] - [<プロジェクト名> のビルド]
[ファイル] - [プロジェクト グループの作成]	[ビルド] - [ソリューションのビルド]
[ファイル] - [プリンタの設定]	[ファイル] - [ページ設定]
[ファイル] - [最近使ったファイル]	[ファイル] - [最近使ったプロジェクト]
[ファイル] - [プロジェクトの解放]	[編集] - [削除]
[ファイル] - [プロジェクト グループの上書き保存]	[ファイル] - [<プロジェクト名> の保存] または [ファイル] - [すべてを保存]
[ファイル] - [名前を付けてプロジェクト グループの保存]	[ファイル] - [名前を付けて <ソリューション名> を保存]
[ファイル] - [選択項目の上書き保存]	[ファイル] - [選択されたファイルを上書き保存]
[書式] - [グリッドに合わせる]	新規に実装されました。[グリッドに合わせる] 環境オプションによって制御されます。
[ヘルプ] - [Microsoft Web ページ]	[コミュニティ] - [ディベロッパー センター]
[プロジェクト] - [クラス モジュールの追加]	[プロジェクト] - [クラスの追加]

[プロジェクト] - [Data Environment の追加]	新規に実装されました。データ環境はサポートされていません。
[プロジェクト] - [Data Report の追加]	新規に実装されました。データ レポートはサポートされていません。
[プロジェクト] - [DHTML Page の追加]	新規に実装されました。DHTML ページはサポートされていません。
[プロジェクト] - [ファイルの追加]	[プロジェクト] - [新しい項目の追加] または [プロジェクト] - [既存項目の追加] メモ　Visual Basic 6.0 ではファイルが直接プロジェクトに追加されますが、Visual Basic 2005 ではファイルのコピーが追加されます。
[プロジェクト] - [フォーム モジュールの追加]	[プロジェクト] - [Windows フォームの追加] または [プロジェクト] - [継承フォームの追加]
[プロジェクト] - [MDI フォーム モジュールの追加]	[プロジェクト] - [Windows フォームの追加] または [プロジェクト] - [継承フォームの追加]
[プロジェクト] - [プロパティ ページの追加]	新規に実装されました。プロパティ ページはサポートされていません。
[プロジェクト] - [ユーザー ドキュメントの追加]	新規に実装されました。ActiveX ドキュメントはサポートされていません。
[プロジェクト] - [WebClass の追加]	新規に実装されました。Web クラスはサポートされていません。
[プロジェクト] - [コンポーネント]	新規に実装されました。コンポーネントはツールボックスで管理されます。
[プロジェクト] - [参照設定]	新規に実装されました。参照はソリューション エクスプローラで管理されます。
[プロジェクト] - [item の解放]	[編集] - [削除] **メモ：** "item の解放" では、項目がプロジェクトからは解放されますが、ディスクからは削除されませんでした。"削除" では項目が削除されます。
[実行] - [中断]	[デバッグ] - [すべて中断]
[実行] - [終了]	[デバッグ] - [デバッグの停止]
[実行] - [再実行]	[デバッグ] - [再起動]
[実行] - [開始]	[デバッグ] - [デバッグ開始]
[実行] - [完全コンパイル後に開始]	新規に実装されました。CTRL+F5 キーを押すと、デバッグせずに引き続き実行できます。
[ツール] - [プロシージャの追加]	新規に実装されました。プロシージャはコード エディタで追加されます。
[ツール] - [メニュー エディタ]	新規に実装されました。メニュー エディタの代わりに、MainMenu コンポーネントと ContextMenu コンポーネントが用意されています。
[ツール] - [プロシージャ属性]	新規に実装されました。
[ツール] - [Visual SourceSafe]	[ファイル] - [ソース管理]

[表示] - [呼び出し履歴]	[デバッグ] - [ウィンドウ] - [呼び出し履歴]
[表示] - [カラー パレット]	新規に実装されました。[カラー パレット] ウィンドウはありません。
[表示] - [データ ビュー ウィンドウ]	新規に実装されました。[データ ビュー ウィンドウ] はありません。
[表示] - [定義]	[定義へ移動] (コンテキスト メニューのみ)
[表示] - [フォーム レイアウト ウィンドウ]	新規に実装されました。[フォーム レイアウト ウィンドウ] はありません。
[表示] - [イミディエイト ウィンドウ]	[デバッグ] - [ウィンドウ] - [イミディエイト]
[表示] - [元の位置へ移動]	新規に実装されました。ブックマークを使用してください。
[表示] - [ローカル ウィンドウ]	[デバッグ] - [ウィンドウ] - [ローカル]
[表示] - [オブジェクト]	[表示] - [デザイナ]
[表示] - [プロジェクト エクスプローラ]	[表示] - [ソリューション エクスプローラ]
[表示] - [ウォッチ ウィンドウ]	[デバッグ] - [ウィンドウ] - [ウォッチ]
[ウィンドウ] - [アイコンの整列]	新規に実装されました。
[ウィンドウ] - [重ねて表示]	MDI レイアウト モードだけで使用できます。
[ウィンドウ] - [上下に並べて表示]	MDI レイアウト モードだけで使用できます。
[ウィンドウ] - [左右に並べて表示]	MDI レイアウト モードだけで使用できます。
[ウィンドウ] - [ウィンドウの一覧]	[ウィンドウ]

キーボード マップの変更点

Microsoft Visual Studio 2005 におけるキーボード マップは、IDE を初めて読み込むときに選択する設定プロファイルに基づきます。Visual Basic プロファイルを選択すると、キーボード マップは Visual Basic 6.0 のキーボード マップと同じになります。[ツール] メニューの [オプション] ダイアログ ボックスで、キーボード マップを変更したり、元の設定に戻したりできます。詳細については、「Visual Studio の設定」を参照してください。

参照

その他の技術情報

開発環境のカスタマイズ
デバッガのロードマップ
Visual Basic 6.0 ユーザー向けのヘルプ

プロジェクトの管理 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 のプロジェクトと Visual Basic 6.0 のプロジェクトは似ていますが、注意する必要のある相違点がいくつかあります。

このセクションの内容

プロジェクトの動作 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 のプロジェクトの動作の全体的な相違点について説明します。

プロジェクト プロパティ (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 のプロジェクト プロパティの相違点について説明します。

プロジェクト参照 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 の参照処理の相違点について説明します。

プロジェクトの種類 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 のプロジェクトの種類の相違点について説明します。

関連するセクション

プロジェクト プロパティ (Visual Studio)

アプリケーションの作成時にアプリケーションを管理するプロジェクト システムについて説明します。

プロジェクトの動作 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 のプロジェクトは、Visual Basic 6.0 のプロジェクトに似ています。しかし、動作にいくつか違いがあります。このトピックでは、その違いについて説明します。

プロジェクト モデル

Visual Basic 6.0 のプロジェクトでは参照をベースとしたモデルが使用され、プロジェクト ファイルにはプロジェクト項目へのパスを指定する参照が格納されています。たとえば、テキスト ファイルをプロジェクトに追加すると、そのファイルの場所をプロジェクト ファイルがポイントします。プロジェクトがビルドされるときに、テキスト ファイルはその場所から読み込まれます。

Visual Basic 2005 ではフォルダをベースとしたモデルが使用され、すべてのプロジェクト項目がプロジェクト フォルダの階層に配置されます。テキスト ファイルを追加すると、ファイルのコピーがプロジェクト フォルダに配置されます。プロジェクトがビルドされるときに、テキスト ファイルはそのファイルのコピーから読み込まれます。

[解放] コマンド

Visual Basic 6.0 の [解放] コマンドでは、項目をプロジェクトから解除するものの、ファイル自体はディスクに残されます。

Visual Basic 2005 では、[解放] コマンドの代わりに [削除] コマンドが用意されています。このコマンドは、ファイルの解除と削除の両方を行います。ファイルを削除せずにプロジェクトから解除する場合は、[プロジェクトから削除] コマンドを使用します。

プロジェクト グループ

Visual Basic 6.0 では、複数のプロジェクトをプロジェクト エクスプローラに追加でき、これらの複数のプロジェクトはプロジェクト グループと呼ばれています。

Visual Basic 2005 では、プロジェクト エクスプローラがソリューション エクスプローラに変わり、プロジェクト グループがソリューションに変わります。プロジェクト グループが Visual Basic プロジェクトだけを格納するのに対し、ソリューションは Visual Basic 2005 言語の任意の組み合わせから成るプロジェクトを格納できます。

プロジェクト ファイル

Visual Basic 6.0 のプロジェクト ファイル (.vbp) およびグループ ファイル (.vbg) は、テキスト エディタで直接編集できるテキスト ファイルです。

Visual Basic 2005 のプロジェクト ファイルおよびソリューション ファイルは XML フォーマットであるため、直接編集するには不向きです。

参照

概念

プロジェクト プロパティ (Visual Basic 6.0 ユーザー向け)

プロジェクトの種類 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

プロジェクトの管理 (Visual Basic 6.0 ユーザー向け)

プロジェクト プロパティ (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 のプロジェクト プロパティによって、プロジェクトのさまざまなグローバル設定が指定されます。Visual Basic 6.0 と Visual Basic 2005 では、多くのプロジェクト プロパティが異なります。

概念の違い

Visual Basic 6.0 では、プロジェクト プロパティを [プロジェクト プロパティ] ダイアログ ボックスで設定します。

Visual Basic 2005 では、プロジェクト プロパティをプロジェクト デザイナで設定します。詳細については、「プロジェクト デザイナの概要」を参照してください。

ActiveX コンポーネントのプロパティ

Visual Basic 6.0 には、[対話型インターフェイスの抑制]、[スレッド モデル]、[ライセンス キーを要求する]、[メモリに保持]、[スタート モード]、[リモート サーバー]、[バージョン間の互換性] など、ActiveX コンポーネントに関する多くのプロパティがあります。

Visual Basic 2005 では、コンポーネントを作成するモデルが大きく異なるため、これらのプロジェクト プロパティに直接相当するものはありません。詳細については、「コンポーネントの作成 (Visual Basic 6.0 ユーザー向け)」を参照してください。

ActiveX コントロール プロパティのアップグレード

Visual Basic 6.0 では、[ActiveX コントロールのアップグレード] プロジェクト プロパティによって、プロジェクトを最後に開いた後で ActiveX コントロールの新バージョンをインストールした場合には、ActiveX コントロールが自動的に更新されます。

Visual Basic 2005 では、ActiveX コントロールは使用できますが、このプロパティに相当するものはありません。ActiveX コントロールを参照すると、それをプロジェクトに追加したときのバージョンが使用されます。

使用しない ActiveX コントロール プロパティについての情報の削除

Visual Basic 6.0 では、[使用しない ActiveX コントロールについての情報を削除する] プロジェクト プロパティによって、使用しないコントロールへの参照をコンパイル前に削除できます。

Visual Basic 2005 では、これに相当するものはありません。使用しないコントロールはコンパイルされません。

コンパイラの最適化

Visual Basic 6.0 と Visual Basic 2005 は両方ともコンパイラの最適化をサポートしますが、コンパイラと、使用できる最適化はそれぞれ異なります。詳細については、「Visual Basic コンパイラ」を参照してください。

Visual Basic 2005 コンパイラの最適化は、[ビルドの詳細設定] ダイアログ ボックスで設定します。このダイアログ ボックスには、プロジェクト デザイナの [コンパイル] タブからアクセスできます。

プロジェクト プロパティの対応関係

Visual Basic 6.0 と Visual Basic 2005 のプロジェクト プロパティを次の一覧で比較します。Visual Basic 2005 のプロジェクト プロパティの場所に関する情報は、各プロパティ名の下に記載されています。なお、すべての種類のプロジェクトですべてのプロパティを使用できるわけではありません。

Visual Basic 6.0	Visual Basic 2005
[自動インクリメント]	[発行ごとにリビジョンを自動的にインクリメントする] プロジェクト デザイナの [発行] タブ
[コマンド ライン引数]	[コマンドライン引数] プロジェクト デザイナの [デバッグ] タブ
[コンパイル]	新規に実装されました。詳細については、「Visual Basic コンパイラ」を参照してください。
[条件付きコンパイル引数]	[コンパイル定数] プロジェクト デザイナの [コンパイル] タブからアクセスできる [ビルドの詳細設定] ダイアログ ボックス

[DLL ベース アドレス]	[DLL ベース アドレス] プロジェクト デザイナの [コンパイル] タブからアクセスできる [ビルドの詳細設定] ダイアログ ボックス
[ヘルプ ファイル名]	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
[アイコン]	[アイコン] プロジェクト デザイナの [アプリケーション] タブ
[プロジェクトの説明]	[説明] - プロジェクト デザイナの [アプリケーション] タブからアクセスできる [アセンブリ情報] ダイアログ ボックス
[プロジェクト ヘルプ コンテキスト番号]	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
[プロジェクト名]	[アセンブリ名] プロジェクト デザイナの [アプリケーション] タブ
[プロジェクトの種類]	[アプリケーションの種類] プロジェクト デザイナの [アプリケーション] タブ
[リモート サーバー]	対応する項目はありません。
[使用しない ActiveX コントロールについての情報を削除する]	対応する項目はありません。
[ライセンス キーを要求する]	対応する項目はありません。
[メモリに保持]	対応する項目はありません。
[スタート モード] (コンポーネント)	対応する項目はありません。
[スタートアップ オブジェクト]	[スタートアップ フォーム]、[スタートアップ オブジェクト] プロジェクト デザイナの [アプリケーション] タブ
[スレッド モデル]	対応する項目はありません。
[タイトル]	[タイトル] プロジェクト デザイナの [アプリケーション] タブからアクセスできる [アセンブリ情報] ダイアログ ボックス
[対話型インターフェイスの抑制]	対応する項目はありません。
[ActiveX コントロールのアップグレード]	対応する項目はありません。
[バージョン間の互換性]	対応する項目はありません。
[バージョン情報]	[会社]、[製品]、[著作権]、[商標] プロジェクト デザイナの [アプリケーション] タブからアクセスできる [アセンブリ情報] ダイアログ ボックス

[バージョン番号]	[アセンブリ バージョン] プロジェクト デザイナの [アプリケーション] タブからアクセスできる [アセンブリ情報] ダイアログ ボックス
[このプロジェクトの実行を開始するときの動作]	[開始動作] プロジェクト デザイナの [デバッグ] タブ

参照

概念

コンポーネントの作成 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

プロジェクト プロパティ (Visual Studio)

プロジェクトの管理 (Visual Basic 6.0 ユーザー向け)

プロジェクト参照 (Visual Basic 6.0 ユーザー向け)

プロジェクト参照を追加する方法は、Visual Basic 2005 と Visual Basic 6.0 で大きく異なります。

概念の違い

参照の追加

Visual Basic 6.0 では、タイプ ライブラリへのプロジェクト参照は、[参照設定] ダイアログ ボックスで管理されます。参照を追加すると、コードからオブジェクトにアクセスできるようになります。

Visual Basic 2005 では、参照はアセンブリに対して設定し、プロジェクト デザイナで追加および削除します。参照を COM コンポーネントに対して設定することもできます。また、プロジェクト間参照、および ASP.NET Web サービスへの Web 参照もサポートされます。詳細については、「参照の管理」を参照してください。

コントロールの追加

Visual Basic 6.0 では、[コンポーネント] ダイアログ ボックスでコントロールをプロジェクトに追加します。このダイアログ ボックスでコントロールを追加すると、対応するコントロールがツールボックスに表示されます。

Visual Basic 2005 では、[ツールボックスのカスタマイズ] ダイアログ ボックスを使用して、コントロールをツールボックスに追加します。詳細については、「方法 : [ツールボックス] に項目を追加する」を参照してください。

参照

その他の技術情報

名前空間およびコンポーネントの参照
プロジェクトの管理 (Visual Basic 6.0 ユーザー向け)

プロジェクトの種類 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 と Visual Basic 6.0 では、作成できるプロジェクトの種類に違いがあります。プロジェクトの種類は本質的に同じでも、名前が変更されている場合があります。また、Visual Basic 6.0 プロジェクトの種類に直接対応するものがない場合もあります。

Visual Basic 6.0 のプロジェクトの種類と、それに対応する Visual Basic 2005 のプロジェクトの種類を次の一覧で比較します。

Visual Basic 6.0	**Visual Basic 2005**
標準の EXE	Windows アプリケーション
ActiveX DLL	クラス ライブラリ
ActiveX EXE	クラス ライブラリ
ActiveX コントロール	Windows コントロール ライブラリ **メモ：** Windows コントロール ライブラリ プロジェクトは Windows フォーム コントロールを作成します。ActiveX コントロールの作成はサポートされなくなりました。
ActiveX ドキュメント	対応するプロジェクトの種類はありません。Visual Basic 2005 は ActiveX ドキュメントと相互運用できます。
DHTML アプリケーション	対応するプロジェクトの種類はありません。Visual Web Developer の ASP.NET Web サイト アプリケーションを使用します。
IIS アプリケーション (Web クラス)	対応するプロジェクトの種類はありません。Visual Web Developer の ASP.NET Web サイト アプリケーションを使用します。

参照

概念

Visual Studio の既定のプロジェクト テンプレート

Web プログラミング (Visual Basic 6.0 ユーザー向け)

その他の技術情報

プロジェクトの管理 (Visual Basic 6.0 ユーザー向け)

言語の変更点 (Visual Basic 6.0 ユーザー向け)

以前のバージョンの Visual Basic は Microsoft Windows クライアント アプリケーションを開発の対象としていましたが、Visual Basic .NET 2002 以降のバージョンは XML Web サービス アプリケーションの作成も考慮してデザインされています。このため、Visual Basic は共通言語ランタイムのマネージ コードを生成します。これに伴って、言語自体に変更が加えられています。

Visual Basic の変更には、次の目的があります。

- 言語を簡略化し、より一貫性のある言語にする。
- ユーザーから要望された機能を新しく追加する。
- コードの読みやすさと保守性を向上させる。
- プログラマによるコーディング エラーを防ぐ。
- アプリケーションの信頼性を向上させ、デバッグを簡単にする。

以下のトピックでは、Visual Basic の前のバージョンからの言語の変更点について説明しています。Visual Basic 言語に新しく導入された機能については、「Visual Basic 言語の新機能」を参照してください。

このセクションの内容

プログラミング要素のサポートに関する変更の概要

Visual Basic で変更されたプログラミング要素やサポートされなくなったプログラミング要素をアルファベット順に示します。また、サポートされなくなったプログラミング要素に代わる要素も紹介します。

配列 (Visual Basic 6.0 ユーザー向け)

配列の境界、サイズの宣言、および ReDim ステートメント (Visual Basic) ステートメントの変更点に関するトピックへのリンクを提供します。

データ型 (Visual Basic 6.0 ユーザー向け)

データ型の宣言、使用方法、および変換の変更点に関するトピックや、整数型 (Integer) (Visual Basic) 型と汎用データ型に関するトピックへのリンクを提供します。

宣言の変更点 (Visual Basic 6.0 ユーザー向け)

宣言の構文、文字列長、構造体、およびブロック内変数のスコープの変更点に関するトピックへのリンクを提供します。

関数の変更点 (Visual Basic 6.0 ユーザー向け)

Format 関数 関数、日付と時刻に関連する関数、および 文字列型 (String) (Visual Basic) (**$**) サフィックスを含む関数の変更点に関するトピックへのリンクを提供します。

その他の言語の変更点 (Visual Basic 6.0 ユーザー向け)

演算子およびファイル処理の変更点に関するトピックへのリンクを提供します。

プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

パラメータの引き渡し、プロシージャ呼び出しのシーケンス、およびプロシージャの宣言の変更点に関するトピックへのリンクを提供します。

プロパティの変更点 (Visual Basic 6.0 ユーザー向け)

既定のプロパティ、プロパティ プロシージャ、およびプロパティ引数の変更点に関するトピックへのリンクを提供します。

制御フローの変更点 (Visual Basic 6.0 ユーザー向け)

実行フローの変更点、および構造化例外処理のサポートに関するトピックへのリンクを提供します。

オブジェクト指向プログラミング (Visual Basic 6.0 ユーザー向け)

オブジェクトの作成、バイナリ互換性、およびインターフェイスとクラスの変更点に関するトピックへのリンクを提供します。

関連するセクション

Visual Basic 言語の新機能

ループの継続、リソース破棄の保証、アクセス プロパティの混在、符号なしデータ型、演算子のオーバーロード、ジェネリック型、共通言語仕様 (CLS) への準拠などについて説明します。

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic の変更点、特に、Web 機能、プロジェクト、フォーム、定数、**Circle** メソッド、**Line** メソッド、**Pset** メソッド、および統合開発環境 (IDE: Integrated Development Environment) について説明します。

Visual Basic リファレンス

Visual Basic 言語と実行時の要素に関するリファレンスです。

プログラミング要素のサポートに関する変更の概要

Visual Basic 2005 では、主に共通言語ランタイムとの相互運用性のために、さまざまなプログラミング要素のサポート方法が変更されています。Visual Basic 6.0 の多くの要素が、名前変更されたり、再分類されたりしています。また、Visual Basic 2005 のその他のプログラミング要素と組み合わさている場合もあります。Visual Basic 6.0 の一部の要素はサポートされなくなりました。これは、共通言語ランタイム (CLR) に含まれる機能により、それらの要素が不要になったためです。詳細については、「共通言語ランタイム」を参照してください。

統合開発環境 (IDE: Integrated Development Environment)、Web 機能、プロジェクト、フォーム、定数、および **Circle**、**Line**、**Pset** の各メソッドを含む、Visual Basic の変更点の詳細については、「Visual Basic 6.0 ユーザー向けのヘルプ」を参照してください。

変更された要素

次の表は、変更されたプログラミング要素、およびその代替要素を示しています。

Visual Basic 6.0 のプログラミング要素	**Visual Basic 2005 で相当する要素**	**名前空間、クラス、またはランタイム ライブラリの場所**
Abs 関数	Abs メソッド	System 名前空間、Math クラス
AscB 関数	Asc 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
As Any キーワード	Visual Basic 2005 ではサポートされません。宣言の構文 (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
Atn 関数	Atan メソッド	System 名前空間、Math クラス
Calendar プロパティ	CurrentCulture プロパティ	System.Globalization 名前空間、Cultur
ChDir ステートメント	ChDir 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
ChDrive ステートメント	ChDrive 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Chr$, ChrB 関数	Chr 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
Close ステートメント	FileClose 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Cos 関数	Cos メソッド	System 名前空間、Math クラス
Currency 型	10 進型 (Decimal)。「データ型の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。	Visual Basic ランタイム ライブラリのメンバ、VariantType 列挙型
CVDate 関数	DateValue 関数	Visual Basic ランタイム ライブラリのメンバ、**DateAndTime** モジュール
CVErr 関数	Error ステートメント	該当なし
Date 関数、**Date** ステートメント	Now プロパティ、Today プロパティ	Visual Basic ランタイム ライブラリのメンバ、**DateAndTime** モジュール
Date$ 関数	DateString プロパティ	Visual Basic ランタイム ライブラリのメンバ、**DateAndTime** モジュール
Debug.Assert メソッド	Assert メソッド、Fail メソッド	System.Diagnostics 名前空間、Debug
Debug.Print メソッド	Write、Writelf、WriteLine、WriteLinelf の各メソッド	System.Diagnostics 名前空間、Debug
Deftype ステートメント	Visual Basic 2005 ではサポートされません。データ型の変更点 (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし

DeleteSetting ステートメント	DeleteSetting 関数	Visual Basic ランタイム ライブラリのメンバ、**Interaction** モジュール
DoEvents 関数	DoEvents メソッド	System.Windows.Forms 名前空間、Ap
Empty キーワード	なし	該当なし
Eqv 演算子	= 演算子、「Boolean 演算子 (Visual Basic 6.0 ユーザー向け)」を参照してください。	該当なし
Exp 関数	Exp メソッド	System 名前空間、Math クラス
FileCopy ステートメント	FileCopy 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Get ステートメント	FileGet 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
GoSub ステートメント	Visual Basic 2005 ではサポートされません。Return ステートメントを使用してください。制御ステートメント (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
Imp 演算子	Visual Basic 2005 ではサポートされません。Not 演算子と Or 演算子および「Boolean 演算子 (Visual Basic 6.0 ユーザー向け)」を参照してください。	該当なし
Initialize イベント	Visual Basic 2005 ではサポートされません。**Sub New** を使用してください。「Class_Initialize の変更点 (Visual Basic 6.0 ユーザー向け)」と「コンストラクタとデストラクタの使用方法」を参照してください。	該当なし
Input # ステートメント、**Input$** ステートメント、**Input$** 関数、**InputB** 関数、**InputB$** 関数	Input 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Instancing プロパティ	Visual Basic 2005 ではサポートされません。クラス レベルの宣言の詳細については、「Private (Visual Basic)」と「Public (Visual Basic)」を参照してください。プロシージャ レベルの宣言の詳細については、「Shared (Visual Basic)」を参照してください。	該当なし
InStrB 関数	InStr 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
IsEmpty 関数	IsNothing 関数	Visual Basic ランタイム ライブラリのメンバ、**Information** モジュール
IsMissing 関数	Visual Basic 2005 ではサポートされません。プロシージャ宣言 (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
IsNull 関数	IsDbNull 関数	Visual Basic ランタイム ライブラリのメンバ、**Information** モジュール
IsObject 関数	IsReference 関数	Visual Basic ランタイム ライブラリのメンバ、**Information** モジュール
Kill ステートメント	Kill 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
LCase$ 関数	LCase 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
Left$ 関数、**LeftB** 関数、**LeftB$** 関数	Left 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール

LenB 関数	Len 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
Let、**Set** 代入ステートメント	Visual Basic 2005 ではサポートされません。新しい **Set** ステートメントは古い **Set** ステートメントと関係ありません。既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
Line Input # ステートメント	LineInput 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Load ステートメント	New (Visual Basic) キーワード、Show メソッド、Load イベント	Visual Basic ランタイム ライブラリのメンバ、System.Windows.Forms 名前空間
Lock ステートメント	Lock 関数、Unlock 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Log 関数	Log メソッド	System 名前空間、Math クラス
LSet ステートメント、**RSet** ステートメント	LSet 関数、PadRight、PadLeft。「データ型の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール、System 名前空間、および String クラス
LTrim$ 関数	LTrim 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
MidB 関数	Mid 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
MidB ステートメント	Mid ステートメント	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
MkDir ステートメント	MkDir 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Name ステートメント	Rename 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Now 関数	Now プロパティ	Visual Basic ランタイム ライブラリのメンバ、**DateAndTime** モジュール
Null キーワード	なし	該当なし
Oct$ 関数	Oct 関数	Visual Basic ランタイム ライブラリのメンバ、**Conversion** モジュール
On ... GoSub 構造	Visual Basic 2005 ではサポートされません。Select...Case ステートメントを使用してください。制御ステートメント (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
On ... GoTo 構造	Visual Basic 2005 ではサポートされません。Select...Case ステートメントを使用してください。制御ステートメント (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
Open ステートメント	FileOpen 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Option Base ステートメント	Visual Basic 2005 ではサポートされません。配列境界 (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
Option PrivateModule ステートメント	Visual Basic 2005 ではサポートされません。Module ステートメントを使用してください。	該当なし

Print # ステートメント	Print 関数、PrintLine 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Property Get ステートメント、**Property Let** ステートメント、**Property Set** ステートメント	Visual Basic 2005 ではサポートされません。プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
Put ステートメント	FilePut 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Reset ステートメント	Reset 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Right$ 関数、**RightB** 関数	Right 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
RmDir ステートメント	RmDir 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Round 関数	Round メソッド	System 名前空間、Math クラス
RSet ステートメント、**LSet** ステートメント	RSet 関数。「データ型の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
RTrim$ 関数	RTrim 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
SaveSetting ステートメント	SaveSetting 関数	Visual Basic ランタイム ライブラリのメンバ、**Interaction** モジュール
Scale メソッド	Visual Basic 2005 ではサポートされません。	該当なし
Set、**Let** 代入ステートメント	Visual Basic 2005 ではサポートされません。新しい **Set** ステートメントは古い **Set** ステートメントと関係ありません。既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
SetAttr ステートメント	SetAttr 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Sgn 関数	Sign 関数	System 名前空間、Math クラス
Sin 関数	Sin メソッド	System 名前空間、Math クラス
Sqr 関数	Sqrt 関数	System 名前空間、Math クラス
String 関数	String コンストラクタ。「String ($) 関数 (Visual Basic 6.0 ユーザー向け)」を参照してください。	System 名前空間、String クラス
String ($) 関数	Visual Basic 2005 ではサポートされません。String ($) 関数 (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
Terminate イベント	Visual Basic 2005 ではサポートされません。**Sub Dispose** と **Sub Finalize** を使用してください。コンストラクタとデストラクタの使用方法 を参照してください。	該当なし
Time 関数、**Time** ステートメント	TimeOfDay プロパティ。「DateTime 構造体」および「日付型 (Date)」を参照してください。	Visual Basic ランタイム ライブラリのメンバ、**DateAndTime** モジュール
Time$ 関数	TimeString プロパティ	Visual Basic ランタイム ライブラリのメンバ、**DateAndTime** モジュール

Timer 関数	Timer プロパティ	Visual Basic ランタイム ライブラリのメンバ、**DateAndTime** モジュール
Trim$ 関数	LTrim 関数、RTrim 関数、および Trim 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
Type ステートメント	Visual Basic 2005 ではサポートされません。Structure ステートメントを使用してください。構造体宣言 (Visual Basic 6.0 ユーザー向け) を参照してください。	該当なし
UCase$ 関数	UCase 関数	Visual Basic ランタイム ライブラリのメンバ、**Strings** モジュール
Unlock ステートメント	Lock 関数、Unlock 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Variant 型	オブジェクト型 (Object)。「汎用データ型の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。	該当なし
Wend キーワード	While...End While ステートメントと End ステートメント。「制御ステートメント (Visual Basic 6.0 ユーザー向け)」を参照してください。	該当なし
Width # ステートメント	FileWidth 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール
Write # ステートメント	Write 関数、WriteLine 関数	Visual Basic ランタイム ライブラリのメンバ、**FileSystem** モジュール

参照

その他の技術情報

言語の変更点 (Visual Basic 6.0 ユーザー向け)
Visual Basic 6.0 ユーザー向けのヘルプ
共通言語ランタイム

配列 (Visual Basic 6.0 ユーザー向け)

次のトピックでは、Visual Basic 6.0 と Visual Basic 2005 における配列に関する変更点について説明します。

このセクションの内容

配列境界 (Visual Basic 6.0 ユーザー向け)

配列の境界および既定の境界に対する変更点について説明します。

配列サイズの宣言 (Visual Basic 6.0 ユーザー向け)

配列サイズの指定方法に対する変更点について説明します。

ReDim ステートメント (Visual Basic 6.0 ユーザー向け)

ReDim ステートメントを含む宣言に対しての変更点について説明します。

関連するセクション

プログラミング要素のサポートに関する変更の概要

Visual Basic 2005 で変更されたプログラミング要素やサポートされなくなったプログラミング要素をアルファベット順に示します。また、サポートされなくなったプログラミング要素に代わる要素も紹介します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の新機能および追加機能の概要を示します。

配列境界 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、他のプログラミング言語の配列との相互運用性を提供するために、配列境界の宣言が更新されています。

Visual Basic 6.0

Visual Basic 6.0 では、配列の各次元の下限は既定で 0 ですが、**Option Base** ステートメントを使用してこの値を 1 に変更できます。また、既定の下限を配列の宣言ごとにオーバーライドすることもできます。

既定値の 0 の場合、配列の要素数は上限に 1 を加えた数になります。次の宣言では、配列 `Weight` に 21 の要素が予約されます。

VB

```
Dim Weight(20) As Single
```

Visual Basic 2005

Visual Basic 2005 では、配列の各次元の下限は 0 です。他の値に宣言することはできません。**Option Base** ステートメントはサポートされません。

宣言で各次元について指定する数は上限であり、初期の要素数は上限に 1 を足した数になります。前の例の宣言では、`Weight` に 21 の要素が予約され、インデックスは 0 ～ 20 になります。

また、上限のいずれかを -1 と宣言することにより、要素を持たない長さ 0 の配列も指定できます。

参照

関連項目

Dim ステートメント (Visual Basic)
UBound 関数 (Visual Basic)

概念

宣言の構文 (Visual Basic 6.0 ユーザー向け)
配列サイズの宣言 (Visual Basic 6.0 ユーザー向け)
プログラミング要素のサポートに関する変更の概要

その他の技術情報

Visual Basic における配列

配列サイズの宣言 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、共通言語ランタイムとの相互運用性のために、配列サイズの宣言が更新されています。

Visual Basic 6.0

Visual Basic 6.0 では、次の例に示すように、配列のサイズを宣言で指定できます。

```
Dim Month(0 To 11) As Integer
```

この場合、配列は固定サイズとなり、**ReDim** ステートメントで変更できません。

Visual Basic 2005

Visual Basic 2005 では、配列のサイズは固定ではありません。上の例は、次のいずれかの宣言で書き換えることができます。

VB

```
Dim Month(11) As Integer    ' Reserves 12 elements -- (0) through (11).
```

VB

```
Dim Month() As Integer = New Integer(11) {}
```

これらの宣言は等価です。どちらも初期サイズを指定していますが、指定した初期サイズは実行中に **ReDim** ステートメントで変更できます。要素を初期化するには、次の構文を使用します。

VB

```
Dim Month() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

いずれかの次元を –1 と指定した場合、配列は要素を持ちません。**ReDim** ステートメントを使用すると、配列を空にしたり空以外に変更したりできます。

次元の数

Visual Basic 2005 では配列のサイズを変更できますが、次元数は固定する必要があります。次の例では 3 次元配列を宣言しています。

VB

```
Dim Point(,,) As Double
```

ReDim ステートメントを使用して各次元のサイズを設定または変更できますが、配列は常に 3 次元です。

参照

関連項目

Dim ステートメント (Visual Basic)
New (Visual Basic)
ReDim ステートメント (Visual Basic)

概念

プログラミング要素のサポートに関する変更の概要

その他の技術情報

Visual Basic における配列

ReDim ステートメント (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、簡略化と読みやすさの向上を実現するために、**ReDim** ステートメントによる宣言が更新されています。

Visual Basic 6.0

Visual Basic 6.0 では、**ReDim** ステートメントを使用して動的配列の初期宣言を指定できます。配列を他の場所で宣言する必要はありません。

Visual Basic 6.0 では、次の例に示すように、**ReDim** ステートメントを使用して配列のランクを変更することもできます。

```
Dim X(10) As Single
ReDim X(10, 10)
```

Visual Basic 2005

Visual Basic 2005 では、**ReDim** を宣言として使用できません。配列は、**ReDim** ステートメントに指定する前に、**Dim** または同等のステートメントを使用して、他の変数と同じように宣言する必要があります。

Visual Basic 2005 では配列のランクを変更できません。前の例で示したコードはエラーになります。

参照

関連項目

ReDim ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)

概念

プログラミング要素のサポートに関する変更の概要

データ型 (Visual Basic 6.0 ユーザー向け)

次のトピックでは、Visual Basic 6.0 と Visual Basic 2005 におけるデータ型に関する変更点について説明します。

このセクションの内容

データ型の変更点 (Visual Basic 6.0 ユーザー向け)

データ型の宣言、使用方法、および変換について説明します。

整数型 (Integer) (Visual Basic 6.0 ユーザー向け)

整数型について更新された点を説明します。

汎用データ型の変更点 (Visual Basic 6.0 ユーザー向け)

汎用データ型について更新された点を説明します。

関連するセクション

プログラミング要素のサポートに関する変更の概要

Visual Basic 2005 で変更されたプログラミング要素やサポートされなくなったプログラミング要素をアルファベット順に示します。また、サポートされなくなったプログラミング要素に代わる要素も紹介します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の新機能および追加機能の概要を示します。

データ型の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、他のプログラミング言語や共通言語ランタイムとの相互運用性のために、データ型が更新されています。この変更によって、データ型の宣言、使用方法、および変換に影響があります。

Visual Basic 6.0

Visual Basic 6.0 では、**Deftype** ステートメント (**DefBool**、**DefByte**、**DefCur**、**DefDate**、**DefDbl**、**DefDec**、**DefInt**、**DefLng**、**DefObj**、**DefSng**、**DefStr**、および **DefVar**) を使用して、変数の既定のデータ型を設定します。

通貨に関係する計算や固定小数点を含む計算では、**Currency** 型を使用します。

Date 型は、8 バイトを使用して **Double** 形式で格納されます。

整数を保持する 2 つの **Variant** 変数を乗算したときにオーバーフロー条件が発生した場合、結果のデータ型は **Double** に変更されます。

あるユーザー定義型の変数を別のユーザー定義型の変数にコピーするには、**LSet** ステートメントおよび **RSet** ステートメントを使用します。

Visual Basic 2005

Deftype ステートメントは Visual Basic 2005 ではサポートされません。**Currency** データ型もサポートされません。代わりに、新しい **Decimal** 型を使用します。この型は、通貨の変数および計算に関して、整数および小数の両方でより多くの桁を処理できます。**Decimal** は、共通言語ランタイムでも直接サポートされます。

Visual Basic 2005 では、**Date** 型は共通言語ランタイムの DateTime 型を使用します。この型は、8 バイトの整数値です。これらの表現の相違のため、**Date** 型と **Double** 型の間で暗黙の型変換は行われません。**Double** と Visual Basic 6.0 表記の **Date** の間で型変換を実行するには、System 名前空間の **DateTime** 構造体の ToOADate メソッドと FromOADate メソッドを使用します。

整数を含む 2 つの **Object** 変数を乗算してオーバーフローが発生した場合、結果は 64 ビットの **Long** 型になります。

LSet および **RSet** を使用して、あるデータ型を別のデータ型には代入することはできません。このような操作は、タイプ セーフではない必要があります。特に構造体では、検証不能なコードになる可能性があります。

参照

関連項目

10 進型 (Decimal) (Visual Basic)
日付型 (Date) (Visual Basic)
Val 関数
データ型変換関数
データ型の概要 (Visual Basic)
倍精度浮動小数点数型 (Double) (Visual Basic)
Long データ型 (Visual Basic)
DateTime
TimeSpan

概念

プログラミング要素のサポートに関する変更の概要

整数型 (Integer) (Visual Basic 6.0 ユーザー向け)

Visual Basic では、他のプログラミング言語や共通言語ランタイムとの相互運用性のために、整数型が更新されています。

次の表は、Visual Basic 6.0 と Visual Basic 2005 の整数型の対応を示しています。

整数型のサイズ	Visual Basic 6.0 の型と識別子の型文字	Visual Basic 2005 の型と識別子の型文字	共通言語ランタイム (CLR) の型
8 ビット、符号付き	(なし)	SByte (なし)	System.SByte
8 ビット、符号なし	**Byte** (なし)	Byte (なし)	System.Byte
16 ビット、符号付き	**Integer** (**%**)	Short (なし)	System.Int16
16 ビット、符号なし	(なし)	UShort (なし)	System.UInt16
32 ビット、符号付き	**Long** (**&**)	Integer (**%**)	System.Int32
32 ビット、符号なし	(なし)	UInteger (なし)	System.UInt32
64 ビット、符号付き	(なし)	Long (**&**)	System.Int64
64 ビット、符号なし	(なし)	ULong (なし)	System.UInt64

32 ビット システムでは、32 ビット整数の演算の方が、16 ビットまたは 64 ビットの整数演算よりも高速です。つまり、Visual Basic 2005 では、**Integer** と **UInteger** は、効率の優れた基本的な数値型です。Visual Basic 2005 に移行するときに **Long** 宣言を **Integer** に変更すると、アプリケーションのパフォーマンスを向上させることができます。

メモ :

Visual Basic 2005 以外のプラットフォームで作成されたコンポーネントとインターフェイスする場合は、使用するデータ型とその他のコンポーネントのデータ型が対応するように注意する必要があります。たとえば、Visual Basic 6.0 で作成された外部プロシージャを **Declare** ステートメントを使用して参照するときに、プロシージャによって **Integer** 引数 (Visual Basic 6.0 では 2 バイト) が定義されている場合は、**Declare** ステートメントでその引数を **Short** として指定する必要があります。これは、Short が Visual Basic 2005 で 2 バイトの整数型であるためです。

参照

関連項目

データ型の概要 (Visual Basic)
Declare ステートメント

概念

型文字
プログラミング要素のサポートに関する変更の概要

汎用データ型の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、共通言語ランタイムとの相互運用性のために、汎用データ型が更新されています。

Visual Basic 6.0

Visual Basic 6.0 では、**Variant** が汎用データ型として機能します。つまり、**Variant** 型の変数には、どのような型のデータでも格納できます。

Visual Basic 2005

Visual Basic 2005 では、**Object** が汎用データ型です。**Object** 型の変数は、どのような型のデータでも保持できます。**Variant** 型はサポートされません。Variant 型の機能はすべて **Object** 型で提供されます。

Variant は、構文では使用されませんが、Visual Basic 2005 でも予約語です。このため、以前のバージョンでの意味と混同されることはありません。

VarType 関数は、オブジェクト変数のデータ型の分類を提供する **VariantType** 列挙型のメンバを返します。次の例に示すように、System 名前空間のクラスを使用して、**Object** インスタンスの数値データ型情報を取得することもできます。

VB

```
' Visual Basic 2005
Dim SomeObj As New Object
' ... ... ... ...  SomeObj is assigned some value during processing.
' ... ... ... ...  Now we want to find out the data type of SomeObj.
Dim Dtype As Integer      ' To hold numeric data type result.
Dtype = Type.GetTypeCode(SomeObj.GetType())
```

参照

関連項目

オブジェクト型 (Object)
VarType 関数 (Visual Basic)
VariantType 列挙型
System
Object
Type

概念

Visual Basic におけるデータ型
値型と参照型
プログラミング要素のサポートに関する変更の概要

宣言の変更点 (Visual Basic 6.0 ユーザー向け)

次のトピックでは、Visual Basic 6.0 と Visual Basic 2005 における宣言に関する変更点について説明します。

このセクションの内容

宣言の構文 (Visual Basic 6.0 ユーザー向け)

複数の変数宣言と戻り値の型の宣言について説明します。

文字列長の宣言 (Visual Basic 6.0 ユーザー向け)

文字列長の指定について説明します。

構造体宣言 (Visual Basic 6.0 ユーザー向け)

構造体、つまりユーザー定義型 (UDT: user-defined type) について説明します。

変数のスコープ (Visual Basic 6.0 ユーザー向け)

ブロック内の変数のスコープについて説明します。

関連するセクション

プログラミング要素のサポートに関する変更の概要

Visual Basic 2005 で変更されたプログラミング要素やサポートされなくなったプログラミング要素をアルファベット順に示します。また、サポートされなくなったプログラミング要素に代わる要素も紹介します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の新機能および追加機能の概要を示します。

宣言の構文 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、プログラミング要素の宣言に関していくつかの点が変更されています。

複数の変数の宣言

Visual Basic 2005 では、複数の変数の同時宣言が簡略化されています。

Visual Basic 6.0

Visual Basic 6.0 では、異なる型の変数を同じステートメントで宣言できますが、各変数にデータ型を指定する必要があります。データ型を指定しない場合は、既定で **Variant** 型になります。次の例は、複数宣言とそのデータ型を示しています。

```
Dim I, J As Integer             ' I is Variant, J is Integer.
Dim L As Integer, M As Integer  ' L is Integer, M is Integer.
Dim N As Integer, X As Double   ' N is Integer, X is Double.
```

Visual Basic 2005

Visual Basic 2005 では、同じデータ型の複数の変数は、キーワードを繰り返さずに宣言できます。前の例で示した宣言は、次の宣言と等価になります。

VB

```
Dim I                           ' I is Object.
Dim J As Integer                ' J is Integer.
```

または

VB

```
Dim I As Object, J As Integer   ' I is Object, J is Integer.
Dim L, M As Integer             ' L is Integer, M is Integer.
Dim N As Integer, X As Double   ' N is Integer, X is Double.
```

外部プロシージャの宣言

Visual Basic 6.0

Visual Basic 6.0 では、外部プロシージャへの参照を **Declare** ステートメントで宣言するときに、引数および戻り値のデータ型に **As Any** を指定できます。**As Any** キーワードは型チェックを無効にするため、どのようなデータ型でも渡したり返したりできます。

Visual Basic 2005

Visual Basic 2005 では、**Any** キーワードはサポートされていません。**Option Strict** が **On** の場合は、**Declare** ステートメントで、引数と戻り値のデータ型を明確に宣言する必要があります。これによりタイプ セーフが向上します。プロシージャ宣言をオーバーロードして、さまざまな引数のデータ型を適用できます。戻り値の型だけをオーバーロードすることはできませんが、引数型のオーバーロードを使用して戻り値の型を変更したり、**Option Strict Off** にしたりできます。

行ラベルの宣言

Visual Basic 6.0

Visual Basic 6.0 では、区切り文字を使用せずに、行番号を同一行にあるステートメントの前に直接記述できます。

Visual Basic 2005

Visual Basic 2005 では、行ラベルの後にコロン (:) を付ける必要があります。同じ行でコロンの後にステートメントを続けたり、行ラベルとコロンを行に単独で記述したりできます。

参照

関連項目

Declare ステートメント

Option Strict ステートメント

概念

汎用データ型の変更点 (Visual Basic 6.0 ユーザー向け)

プロシージャ呼び出しシーケンス (Visual Basic 6.0 ユーザー向け)
Visual Basic におけるデータ型
プログラミング要素のサポートに関する変更の概要

文字列長の宣言 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、共通言語ランタイムとの相互運用性のために、文字列長の宣言が更新されています。

Visual Basic 6.0

Visual Basic 6.0 では、文字列の長さを宣言で指定できます。これによって、次の例に示すように、文字列は固定長となります。

```
Dim Name As String * 30
```

Visual Basic 2005

Visual Basic 2005 では、宣言で VBFixedStringAttribute クラス属性を使用しない限り、固定長の文字列を宣言できません。前の例で示したコードはエラーになります。

文字列は、長さを指定せずに宣言します。次の例に示すように、文字列に値を代入するときに、値の長さによって文字列の長さが決定されます。

VB

```
Dim Name As String
' ...
Name = "Name is now 30 characters long" ' Length can be changed later.
```

参照

関連項目

文字列型 (String) (Visual Basic)
Dim ステートメント (Visual Basic)

概念

Visual Basic で使用される属性
プログラミング要素のサポートに関する変更の概要

構造体宣言 (Visual Basic 6.0 ユーザー向け)

Visual Basic では、構造体とユーザー定義型 (UDT) を同じ型のプログラミング要素として扱います。Visual Basic 2005 では、構造体宣言を統合と読みやすさ向上のために更新します。

Visual Basic 6.0

Visual Basic 6.0 では、**Type ... End Type** 構造を使って構造体を宣言します。構造体とそのメンバはすべて、既定でパブリック アクセスになります。明示的なアクセス宣言はオプションです。次の例は、有効な構造体の宣言を示しています。

```
Type Employee

EmpNumber As Integer ' Defaults to Public access.

EmpOfficePhone As String

EmpHomePhone As String ' Cannot be declared Private inside Type.

End Type
```

Visual Basic 2005

Visual Basic 2005 では、**Type** ステートメントはサポートされていません。構造体は、**Structure** ステートメントを使用して、**Structure ... End Structure** 構造の中で宣言する必要があります。構造体のすべてのメンバには、**Public**、**Friend**、または **Private** のアクセス修飾子を指定する必要があります。**Dim** ステートメントを使用して、既定のパブリック アクセスにすることもできます。上の例の構造体は、次のように宣言できます。

VB

```
Structure Employee
  Public EmpNumber As Integer    'Must declare access, even if Public.
  Dim EmpOfficePhone As String   'Still defaults to Public access.
  Private EmpHomePhone As String 'Can be made Private inside Structure.
End Structure
```

Visual Basic 2005 では、構造体とクラスの構文が統一されています。構造体は、メソッドを含み、クラスの機能のほとんどをサポートします。

参照

関連項目

Structure ステートメント
Dim ステートメント (Visual Basic)
Public (Visual Basic)
Private (Visual Basic)

概念

プログラミング要素のサポートに関する変更の概要

変数のスコープ (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、ブロック スコープのサポートと構造化プログラミングの強化のために、ローカル変数のスコープが更新されています。

Visual Basic 6.0

Visual Basic 6.0 では、プロシージャ内で宣言された変数はプロシージャ スコープを持ちます。したがって、同じプロシージャ内のどこからでもアクセスできます。変数がブロック内で宣言されている場合、つまり **End**、**Loop**、または **Next** ステートメントで終了するステートメント セット内で宣言されている場合、変数はブロック外部からでもアクセスできます。

次のコード例では、プロシージャのスコープを示しています。ここでは、ブロックは **For** ループです。

```
For I = 1 To 10

Dim N As Long = 0

' N has procedure scope although it was declared within a block.

N = N + Incr(I)

Next I

W = Base ^ N

' N is still visible outside the block it is declared in.
```

Visual Basic 2005

Visual Basic 2005 では、ブロック内で宣言された変数はブロック スコープを持ち、ブロックの外部からはアクセスできません。上の例は次のように書き換えることができます。

VB

```
Dim N As Long = 0
' N is declared outside the block and has procedure scope.
For I As Integer = 1 To 10
    ' I is declared by the For statement and therefore within the block.
    N = N + Incr(I)
Next I
w = Base ^ N
' N is visible outside the block but I is not.
```

For ステートメントは `I` を **For** ブロックの一部として宣言するため、`I` はブロック スコープだけを持ちます。

ただし、変数の有効期間は、プロシージャ全体の有効期間のままになります。変数がブロック スコープまたはプロシージャ スコープのどちらの場合でも同様です。ブロック内で変数を宣言し、プロシージャの有効期間中に何度もブロックに入る場合は、変数を初期化して、変数の値が予測不可能になることを避けます。

参照

概念

プログラミング要素のサポートに関する変更の概要

関数の変更点 (Visual Basic 6.0 ユーザー向け)

次のトピックでは、Visual Basic 6.0 と Visual Basic 2005 における関数に関する変更点について説明します。

このセクションの内容

Format 関数 (Visual Basic 6.0 ユーザー向け)

Format 関数に対する日付/時刻、数値、文字列の各書式の変更点について説明します。

日付と時刻 (Visual Basic 6.0 ユーザー向け)

Date、**Date$**、**Time**、および **Time$** 関数の変更点について説明します。

String ($) 関数 (Visual Basic 6.0 ユーザー向け)

String ($) サフィックスを持つ関数について説明します。

関連するセクション

プログラミング要素のサポートに関する変更の概要

Visual Basic 2005 で変更されたプログラミング要素やサポートされなくなったプログラミング要素をアルファベット順に示します。また、サポートされなくなったプログラミング要素に代わる要素も紹介します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の新機能および追加機能の概要を示します。

Format 関数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の **Format** 関数は、共通言語ランタイム (CLR) の仕様に従ってデータの書式を設定するようになりました。.NET Framework でのデータ書式設定の詳細については、「型の書式設定」を参照してください。

Visual Basic 2005 の日付/時刻、数値、文字列の各書式に対して加えられた変更点を次に示します。

ユーザー定義の日付/時刻書式

Visual Basic 6.0

Visual Basic 6.0 では、短い形式または長い形式の日付表示に、それぞれ "ddddd" と "dddddd" の書式指定子を使用します。**DayOfWeek** ("w") 指定子および **WeekOfYear** ("ww") 指定子では、一週間の最初の日として認識された日、および一年の最初の週として認識された週がそれぞれ表示されます。小文字の "m" という文字は、先行するゼロが付かない数字で月を表示します。**Quarter** 指定子 ("q") は、1 ～ 4 の数字で四半期を表示します。

分を数字で表示するとき、先行するゼロを付ける場合は "Nn"、付けない場合は "N" という書式指定子を使用します。"Hh" という文字は、先行するゼロの付いた数字で時間を表示します。また、"ttttt" は絶対時間を表示します。午前と午後を示すために、大文字または小文字の "A" または "P" を表示するには、"AM/PM"、"am/pm"、"A/P"、"a/p"、または "AMPM" のいずれかを使用します。

短い形式の日付/時刻の指定子 ("c") を使うと、日付と時刻は "ddddd ttttt" 形式で表示されます。

Visual Basic 2005

Visual Basic 2005 では、"ddddd" と "dddddd" は、"dddd" と見なされ、日付の完全名が表示されます。短い形式や長い形式の日付は表示されません。**DayOfWeek** ("w") と **WeekOfYear** ("ww") はサポートされていません。次のコード例に示したように、**DatePart** 関数を代わりに使用できます。

VB

```
     Format(DatePart(DateInterval.Weekday, Now))
...     Format(DatePart(DateInterval.WeekOfYear, Now))
```

"M" と "m" には、それぞれ別の意味が割り当てられているため、大文字と小文字は区別されます。日付/時刻の書式の日付部分において月を示す場合にだけ "M" を使用します。"m" は、時刻部分において分を示す場合にだけ使用します。

Quarter 書式の指定子はサポートされていません。次のコード例に示したように、**DatePart** 関数を代わりに使用できます。

VB

```
Format(DatePart(DateInterval.Quarter, Now))
```

分を数値で表示するには、先行するゼロを付ける場合は "m"、付けない場合は "mm" を使用します。"ttttt" 形式はサポートされなくなりました。"H" と "h" には、それぞれ別の意味が割り当てられているため、大文字と小文字は区別されます。24 時間制の時刻形式では "H"、12 時間制の時刻形式では "h" を使用します。AM/PM 形式は、"t" および "tt" で置き換えられます。

"c" 指定子は、通貨の書式設定に使用します。日付/時刻の書式を設定する場合は、短い形式を指定するときに "g" を使用し、通常の形式を指定するときに "G" を使用します。"g" および "G" では、現在のロケールの設定を使って日付と時刻の適切な書式が決定されます。

ユーザー定義の数値書式

Visual Basic 6.0

Visual Basic 6.0 では、書式を設定する前に文字列を数値に変換する必要がある場合、**Format** 関数を使って変換を行います。小数部がない場合、**Format** 関数は後続の小数点を表示します。

Visual Basic 6.0 では、書式文字列は 4 つのセクションに分かれています。各セクションはセミコロン (;) で区切られ、それぞれは正、負、ゼロ、および null の各値の書式を設定する方法を示します。書式文字列の負のセクションが空になっている負の数値は、空の文字列を表示します。

指数表記の書式では、指数部の後に "0" と "#" 桁プレースホルダの両方がサポートされています。

Visual Basic 2005

Visual Basic 2005 では、**Format** 関数は、書式を設定する前に文字列を数値に変換しません。最初の引数には、文字列ではなく数値を渡す必要があります。次のコード例を Visual Basic 2005 で実行すると、最初のコード行は意図した結果になりませんが、2 番目のコード行は意図した結果になります。

VB

```
    MsgBox(Format("1.234", "#.#"))   ' Displays "#.#".
...    MsgBox(Format(CSng("1.234"), "#.#"))   ' Displays "1.2".
```

次の例に示したように、後続の小数点は表示されません。

VB

```
    MsgBox(Format(123, "###."))   ' Displays "123"
...    MsgBox(Format(123, "###.#"))   ' Displays "123"
```

Visual Basic 2005 では、書式文字列は、正、負、およびゼロの各値の書式を設定するための 3 つのセクションから構成されています。1 番目または 2 番目の書式セクションによってゼロ以外の値がゼロに丸められた場合でも、丸められた値は 3 番目のセクションによって書式指定されます。次の例に示すように、書式文字列の負のセクションが空になっている負の数値は、マイナス記号を表示します。

VB

```
MsgBox(Format(-1, ";"))   ' Displays "-".
```

指数表記の書式では、"0" 桁プレースホルダだけがサポートされ、"#" はサポートされません。次のコード例を Visual Basic 2005 で実行すると、最初のコード行は意図した結果になりませんが、2 番目のコード行は意図した結果になります。

VB

```
    MsgBox(Format(123, "#e+#"))   ' Displays "12e+3".
...    MsgBox(Format(123, "#e+0"))   ' Displays "1e+2".
```

文字列の書式

Visual Basic 6.0

Visual Basic 6.0 では、ユーザー定義の書式指定文字列の式を **@**、**&**、**<**、**>**、**!** という書式指定子を使って作成できます。

Visual Basic 2005

Visual Basic 2005 では、ユーザー定義の書式指定文字列は一切サポートされません。したがって、**@**、**&**、**<**、**>**、および **!** は意味を持たず、サポートもされていません。

参照

関連項目

Format 関数
DatePart 関数 (Visual Basic)

概念

日付と時刻 (Visual Basic 6.0 ユーザー向け)
プログラミング要素のサポートに関する変更の概要

日付と時刻 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、**Date**、**Date$**、**Time**、および **Time$** の各関数が更新され、**Now** 関数および **Timer** 関数はプロパティに置き換えられています。

Visual Basic 6.0

Visual Basic 6.0 では、**Date** 関数および **Time** 関数は、4 バイトの **Date** 形式でシステムの日付および時刻を返します。同様に、**Date** ステートメントと **Time** ステートメントは、Visual Basic 6.0 の形式を使用して、システムの日付と時刻を設定します。

Date$** および **Time$ 関数は、システムの日付と時刻を **String** 形式で返します。

Now 関数および **Timer** 関数は、現在の日付と時刻、および午前 0 時からの経過時間を秒数で指定します。

Visual Basic 2005

Visual Basic 2005 では、**Date** および **Time** は **Today** プロパティおよび **TimeOfDay** プロパティに置き換えられています。これらのプロパティでは、8 バイトの共通言語ランタイム (CLR) DateTime 構造が使用されます。これは、Visual Basic 2005 の **Date** 型に相当します。**Today** および **TimeOfDay** を使用すると、システムの日付と時刻を設定したり取得したりできます。

Visual Basic 2005 では、**Date$** 関数および **Time$** 関数は、**DateString** プロパティおよび **TimeString** プロパティに置き換えられています。**DateString** および **TimeString** を使用すると、システムの日付と時刻を設定したり取得したりできます。

Now 関数および **Timer** 関数は、同じ名前の読み取り専用プロパティに置き換えられています。呼び出しの順序については変更されていません。

参照

関連項目

Format 関数
Today プロパティ
TimeOfDay プロパティ
日付型 (Date) (Visual Basic)
DateString プロパティ
TimeString プロパティ
Now プロパティ
Timer プロパティ

概念

Format 関数 (Visual Basic 6.0 ユーザー向け)
プログラミング要素のサポートに関する変更の概要

String ($) 関数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、一部の関数に **String** バージョンと **Variant** バージョンを用意する必要はなくなりました。

Visual Basic 6.0

Visual Basic 6.0 では、一部の関数に、**String** 値を返すバージョンと **Variant** 値を返すバージョンの 2 つがあります。これらの関数のペアは、**String** バージョンにドル記号 ($) のサフィックスを付けることによって区別されます。たとえば、**Chr** と **Chr$** と表記されます。

Visual Basic 2005

Visual Basic 2005 では、各関数のペアが 1 つの関数に置き換えられます。**Variant** バージョンはなくなり、**String** バージョンは $ サフィックスを付けても付けなくても呼び出すことができます。

Trim だけが定義されているものの、Visual Basic 2005 では **Trim$** も受け入れます。これは、**$** が **String** 型の識別子型文字として機能するためです。

参照

関連項目

Trim、LTrim、RTrim 関数

概念

Format 関数 (Visual Basic 6.0 ユーザー向け)
日付と時刻 (Visual Basic 6.0 ユーザー向け)
プログラミング要素のサポートに関する変更の概要

その他の言語の変更点 (Visual Basic 6.0 ユーザー向け)

次のトピックでは、Visual Basic 6.0 と Visual Basic 2005 における演算子およびファイル処理に関する変更点について説明します。

このセクションの内容

Boolean 演算子 (Visual Basic 6.0 ユーザー向け)

Boolean 演算子と論理式について説明します。

ファイル処理 (Visual Basic 6.0 ユーザー向け)

ファイル処理機能の拡張の他、以前のバージョンの Visual Basic のファイル入出力関数との互換性について説明します。

関連するセクション

プログラミング要素のサポートに関する変更の概要

Visual Basic 2005 で変更されたプログラミング要素やサポートされなくなったプログラミング要素をアルファベット順に示します。また、サポートされなくなったプログラミング要素に代わる要素も紹介します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の新機能および追加機能の概要を示します。

Boolean 演算子 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、2 つの Boolean 演算子が削除され、パフォーマンスを向上させる別の 2 つの演算子が追加されています。

Visual Basic 6.0

Visual Basic 6.0 では、Boolean 演算子 (**And**、**Or**、**Not**、および **Xor**) は、オペランドのすべての式を常に評価します。

2 つの式の論理等値演算および論理包含演算を実行するには、**Eqv** 演算子と **Imp** 演算子を使用します。

Visual Basic 2005

Visual Basic 2005 でも、**And**、**Or**、**Not**、および **Xor** の各演算子は、オペランドに提供されたすべての式を評価します。また、Visual Basic 2005 では、**AndAlso** および **OrElse** という新しい演算子が導入されています。この演算子は、"ショートサーキット" 論理評価によって実行時間を短縮できます。**AndAlso** 演算子の 1 番目のオペランドが **False** に評価されると、2 番目のオペランドは評価されません。同様に、**OrElse** 演算子の 1 番目のオペランドが **True** に評価されると、2 番目のオペランドは評価されません。

メモ :

ショートサーキット演算子の使用には注意が必要です。2 番目のオペランドにプロシージャ呼び出しが含まれている場合、演算子が実行されるたびにプロシージャが呼び出されるとは限りません。

Eqv 演算子と **Imp** 演算子はサポートされません。論理評価とビットごとの評価のいずれの場合も、**Eqv** の代わりに等値演算子 (=) を使用します。論理 **Imp** 演算子は、次に示すように、**Not** 演算子および **Or** 演算子を使用した式で置き換えることができます。

VB

```
Result = (Not A) Or B  'Same as A Imp B
                       '(True unless A is True and B is False.)
```

ビットごとの **Imp** 演算子も、数値オペランドに **Not** 演算子および **Or** 演算子を使用して、同じ方法で置き換えることができます。

参照

関連項目

And 演算子 (Visual Basic)
AndAlso 演算子
Or 演算子 (Visual Basic)
OrElse 演算子
Not 演算子 (Visual Basic)
Xor 演算子 (Visual Basic)
TimeSpan

概念

プログラミング要素のサポートに関する変更の概要

ファイル処理 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、以前のバージョンの Visual Basic のファイル入出力関数との互換性を保ちつつ、ファイル処理機能が拡張されています。

Visual Basic 6.0

Visual Basic 6.0 では、**Open**、**Input**、**Output**、**Append** などのさまざまなファイル入出力関数を使用してファイル処理を行います。**FileSystemObject** オブジェクトにより、オブジェクト指向的な方法でファイルを扱うことができます。

Visual Basic 2005

Visual Basic 2005 では、My.Computer.FileSystem オブジェクトを通じてファイル処理を行います。さらに、**System.IO.File** クラスには、従来の Visual Basic のファイル入出力関数との互換性を持つ関数が用意されています。

FileStream クラスを使うと、標準の入出力ファイル、およびエラー デバイスにアクセスできます。

次の表に、Visual Basic 2005 で使用できる **My.Computer.FileSystem** オブジェクトのメンバの一覧を示します。

メンバ	説明
CombinePath	適切な形式の組み合わせたパスを **String** として返します。
CopyDirectory	ディレクトリをコピーします。
CopyFile	ファイルをコピーします。
CurrentDirectory	現在のディレクトリを取得または設定します。
CreateDirectory	ディレクトリを作成します。
DeleteDirectory	ディレクトリを削除します。
DeleteFile	ファイルを削除します。
DirectoryExists	ディレクトリが存在するかどうかを示す **Boolean** を返します。
Drives	すべての使用可能なドライブの名前を読み取り専用コレクションとして返します。
FileExists	ファイルが存在するかどうかを表す **Boolean** を返します。
FindInFiles	指定されたテキストが含まれるファイル名を読み取り専用の文字列コレクションとして返します。
GetDirectories	ディレクトリ内にあるサブディレクトリのパス名を **String** コレクションとして返します。
GetDirectoryInfo	指定されたパスの DirectoryInfo オブジェクトを返します。
GetDriveInfo	指定されたパスの DriveInfo オブジェクトを返します。
GetFileInfo	指定されたパスの FileInfo オブジェクトを返します。
GetFiles	ディレクトリ内にあるファイルの名前を読み取り専用の **String** コレクションとして返します。
GetParentPath	指定されたパスの親の絶対パスを **String** として返します。
GetTempFileName	一意の名前を持つ 0 バイトの一時ファイルをディスク上に作成し、そのファイルの完全パスを返します。

MoveDirectory	指定された位置にディレクトリを移動します。
MoveFile	指定された位置にファイルを移動します。
OpenTextFieldParser	TextFieldParser を開きます。
OpenTextFileReader	TextReader を開きます。
OpenTextFileWriter	TextWriter を開きます。
ReadAllBytes	バイナリ ファイルからデータを読み取ります。
ReadAllText	テキスト ファイルからデータを読み取ります。
RenameDirectory	ディレクトリの名前を変更します。
RenameFile	ファイルの名前を変更します。
SpecialDirectories	頻繁に参照されるディレクトリにアクセスするためのプロパティを備えたオブジェクトを取得します。
WriteAllBytes	バイナリ ファイルに書き込みます。
WriteAllText	テキスト ファイルに書き込みます。

参照

関連項目

System.IO Namespace
My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.SpecialDirectories オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析
ファイル エンコーディング
プログラミング要素のサポートに関する変更の概要

その他の技術情報

Visual Basic でのファイルの読み取り
Visual Basic でのファイルへの書き込み
Visual Basic でのファイルおよびディレクトリの作成、削除、および移動
Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

モジュールの変更点 (Visual Basic 6.0 ユーザー向け)

構文が変更されているため、プライベート モジュールの宣言に影響があります。

Option Private Module ステートメント

Visual Basic 6.0

Visual Basic 6.0 では、**Option Private Module** ステートメントは、モジュール全体がプライベートで、そのコンポーネントにプロジェクト外部からはアクセスできないことを示します。

Visual Basic 2005

Visual Basic 2005 では、**Option Private Module** ステートメントはサポートされていません。モジュールをプライベートとして宣言できません。ただし、モジュールの一部またはすべてのメンバをプライベートとして宣言することはできます。

モジュールとは異なり、クラスはプライベートにすることができます。モジュールそのものをプライベートとして動作させる必要がある場合は、代わりにクラスを宣言します。

参照

関連項目

Module ステートメント
Class ステートメント (Visual Basic)
Private (Visual Basic)

プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

次のトピックでは、Visual Basic 6.0 と Visual Basic 2005 におけるプロシージャに関する変更点について説明します。

このセクションの内容

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

プロシージャへの引数の引き渡しについて説明します。

プロシージャ呼び出しシーケンス (Visual Basic 6.0 ユーザー向け)

プロシージャの呼び出しと戻りについて説明します。

プロシージャ宣言 (Visual Basic 6.0 ユーザー向け)

省略できるプロシージャの引数と静的ローカル変数について説明します。

関連するセクション

プログラミング要素のサポートに関する変更の概要

Visual Basic 2005 で変更されたプログラミング要素やサポートされなくなったプログラミング要素をアルファベット順に示します。また、サポートされなくなったプログラミング要素に代わる要素も紹介します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の新機能および追加機能の概要を示します。

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、引数をプロシージャに渡す方法に影響する変更点があります。

既定の引き渡しの方法

Visual Basic 6.0

Visual Basic 6.0 では、プロシージャのパラーメータに **ByVal** または **ByRef** を指定しない場合、既定の引き渡し方法は **ByRef** になります。これにより、プロシージャに渡される変数を呼び出し元のプログラムで変更できます。

変数を変更できるように公開することで、危険が生じる可能性があります。次の例では、引き渡し方法は既定で **ByRef** になります。`elapsedSeconds` の値は `minutesPastHour` で変更され、**MsgBox** で `elapsedSeconds` が正しく表示されません。

```
Function minutesPastHour(seconds As Integer) As Integer
    Dim hours As Integer = seconds \ 3600
    seconds = seconds Mod 3600
    Return seconds \ 60
End Function
Sub showSecondsAndMinutes()
    Dim elapsedSeconds, extraMinutes As Integer
    elapsedSeconds = CInt(Timer()) ' Integer seconds since midnight.
    extraMinutes = minutesPastHour(elapsedSeconds)
    MsgBox "Total seconds: " & elapsedSeconds & _
        "; minutes past hour: " & extraMinutes
End Sub
```

引数を **ByRef** で渡すことにより、プロシージャは呼び出し元のプログラムで引数を変更できます。このため、予測できない動作が発生する可能性があります。また、プロシージャが別のプロシージャを呼び出し、同じ引数を **ByRef** で渡した場合は、意図せずに元の変数が変更される可能性が高くなります。

Visual Basic 2005

Visual Basic 2005 でプロシージャを宣言すると、各パラメータの既定の引き渡し方法は **ByVal** になります。これにより、引数が変更されるのを防ぎます。上の例の宣言は次のように書き換えることができます。

VB

```
Function MinutesPastHour(ByVal Seconds As Integer) As Integer
```

`seconds` を値によって引き渡すことで、プロシージャは呼び出し元のプログラムで変数にアクセスできません。また、上で説明したような危険性も回避できます。

特に指定しなくても既定の方法は **ByVal** ですが、各パラメータで明示的に指定する方が明確で読みやすいコードになります。

ByRef Property 引数

Visual Basic 6.0

Visual Basic 6.0 では、プロシージャに **ByRef** 引数として渡されるプロパティは、プロシージャにコピー インされますが、コピー アウトはされません。つまり、**ByRef** で渡された場合でも、これらのプロパティ引数に対する変更は、呼び出し元プログラムの元のプロパティには反映されません。

Visual Basic 2005

Visual Basic 2005 では、**ByRef** で渡されたプロパティ引数は、プロシージャにコピー インおよびコピー アウトされます。次の例は、プロシージャによってプロパティが変更されるしくみを示しています。

VB

```
Sub Reduce(ByRef Height As Single)
' ... ... ... ... ... ... ...  ' Code to modify Height argument.
End Sub
```

VB

```
Dim Sq As Square = New Square  ' Square has property Side of type Single.
```

```
Reduce(Sq.Side)                ' Side is changed when Reduce returns.
```

プロシージャがプロパティ引数を変更するときに、元のプロパティの値は、呼び出し元プログラムですぐには変更されません。元のプロパティの値は、プロシージャから制御が戻るときにコピー アウトされます。

ParamArray 引数

Visual Basic 6.0

Visual Basic 6.0 では、プロシージャの最後の引数に **ParamArray** キーワードを指定して、**Variant** 引数の配列を受け取ることができます。これらの引数の引き渡し方法は宣言できません。常に **ByRef** で渡されます。

Visual Basic 2005

Visual Basic 2005 では、**ParamArray** 引数は常に **ByVal** で渡されます。配列の引数は、すべて **ParamArray** 引数のデータ型である必要があります。

参照

関連項目

Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)
ByVal
ByRef
ParamArray

概念

プロシージャ宣言 (Visual Basic 6.0 ユーザー向け)
プロシージャ呼び出しシーケンス (Visual Basic 6.0 ユーザー向け)
パラメータ配列
プログラミング要素のサポートに関する変更の概要

プロシージャ呼び出しシーケンス (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、プロシージャ呼び出しシーケンスに関していくつかの点が変更されています。これらの変更により、構文の一貫性が強化されます。

プロシージャ呼び出しのかっこ

Visual Basic 6.0

Visual Basic 6.0 では、**Function** 呼び出しの引数リストをかっこで囲む必要があります。**Sub** 呼び出しでは、**Call** ステートメントを使用する場合はかっこが必要ですが、そうでない場合はかっこを使用しません。次の例は、有効な呼び出しステートメントを示します。

```
y = Sqrt(x)

Call displayCell(2, 14, cellValue)

displayCell 2, 14, cellValue ' Variation on previous statement.
```

Visual Basic 2005

Visual Basic 2005 では、プロシージャ呼び出しの空でない引数リストは、常にかっこで囲む必要があります。**Sub** 呼び出しでは、**Call** ステートメントは省略できます。上の例は次のように書き換えることができます。

VB

```
    Y = Math.Sqrt(X)
...    DisplayCell(2, 14, Value)
```

引数を指定せずにプロシージャを呼び出す場合は、空のかっこを指定するか、かっこを省略します。

Return ステートメント

Visual Basic 6.0

Visual Basic 6.0 では、**GoSub** ステートメントに続くコードに戻るためだけに **Return** ステートメントを使用します。どちらのステートメントも、同じプロシージャ内にある必要があります。

Visual Basic 2005

Visual Basic 2005 では、**GoSub** ステートメントはサポートされません。**Return** ステートメントを使用して、**Function** プロシージャまたは **Sub** プロシージャから呼び出し元のプログラムに制御を返すことができます。

参照

関連項目

Return ステートメント (Visual Basic)
Call ステートメント (Visual Basic)

概念

宣言の構文 (Visual Basic 6.0 ユーザー向け)
プロシージャ宣言 (Visual Basic 6.0 ユーザー向け)
パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)
制御ステートメント (Visual Basic 6.0 ユーザー向け)
プログラミング要素のサポートに関する変更の概要

プロシージャ宣言 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、プロシージャの宣言に関していくつかの点が変更されています。

省略可能な引数

Visual Basic 6.0

Visual Basic 6.0 では、既定値を指定せずに、プロシージャ パラメータを **Optional** として宣言できます。省略可能なパラメータが **Variant** 型の場合は、呼び出し元コードからパラメータに引数が渡されたかどうかをプロシージャ コードで **IsMissing** 関数を使って確認できます。

Visual Basic 2005

Visual Basic 2005 では、省略可能な引数に既定値を宣言する必要があります。呼び出し元のプログラムで引数が指定されない場合は、既定値がプロシージャに渡されます。**IsMissing** 関数を使用して、引数があるかどうかを確認する必要はありません。また、**IsMissing** 関数はサポートされません。次の例は、省略可能な引数の宣言を示しています。

VB

```
Sub Calculate(Optional ByVal Switch As Boolean = False)
```

静的なローカル変数

Visual Basic 6.0

Visual Basic 6.0 では、**Static** 修飾子を指定してプロシージャを宣言できます。これにより、プロシージャ内のすべてのローカル変数は静的となり、呼び出し間で値が保持されます。

Visual Basic 2005

Visual Basic 2005 では、**Function** ステートメントまたは **Sub** ステートメントで **Static** キーワードはサポートされません。**Static** に指定するローカル変数を個別に宣言する必要があります。

参照

関連項目

Optional (Visual Basic)
Static (Visual Basic)

概念

プロシージャ呼び出しシーケンス (Visual Basic 6.0 ユーザー向け)
パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)
プログラミング要素のサポートに関する変更の概要

プロパティの変更点 (Visual Basic 6.0 ユーザー向け)

次のトピックでは、Visual Basic 6.0 と Visual Basic 2005 におけるプロパティに関する変更点について説明します。

このセクションの内容

既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)

既定のプロパティ サポートの変更について説明します。

プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

プロパティ プロシージャおよび引数に対する変更点について説明します。

関連するセクション

プログラミング要素のサポートに関する変更の概要

Visual Basic 2005 で変更されたプログラミング要素やサポートされなくなったプログラミング要素をアルファベット順に示します。また、サポートされなくなったプログラミング要素に代わる要素も紹介します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の新機能および追加機能の概要を示します。

既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、簡略化と読みやすさの向上を実現するために、既定のプロパティ サポートが更新されています。

Visual Basic 6.0

Visual Basic 6.0 では、既定のプロパティはオブジェクトでサポートされます。たとえば、**Label** コントロールでは、**Caption** が既定のプロパティとなり、次の例の 2 つの代入は等価になります。

```
Dim lbl As Label
lbl = "Important"
lbl.Caption = "Important"
```

既定のプロパティを使用すると、Visual Basic コードを簡略に記述できますが、いくつかの欠点もあります。

- コードが読みにくくなる場合があります。**Label** コントロールに慣れていない場合、前の例の 1 つ目の代入で、文字列 `"Important"` が変数 `lbl` に直接格納されているのか、既定のプロパティに格納されているのかを判断できません。
- コードで使用するオブジェクトに既定のプロパティがあるかどうか、またどれが既定のプロパティであるかが、わかりにくい場合もあります。
- Visual Basic 言語では、既定のプロパティがあるために **Set** ステートメントが必要になります。次の例は、既定のプロパティではなく、オブジェクト参照の代入を示すために **Set** が必要であることを示しています。

  ```
  Dim lbl1 As Label, lbl2 As Label
  lbl1 = "Saving" ' Assign a value to lbl1's Caption property.
  lbl2 = lbl1       ' Replace lbl2's Caption property with lbl1's.
  Set lbl2 = lbl1   ' Replace lbl2 with an object reference to lbl1.
  ```

Visual Basic 2005

Visual Basic 2005 では、引数を受け取らない既定のプロパティはサポートされません。この構文変更により、**Let** ステートメントおよび **Set** ステートメントで代入の内容を指定する必要はなく、これらのステートメントは代入のステートメントで使用されません。Label コントロールの **Caption** プロパティは、Text プロパティに置き換えられています。前の例は、次のように書き換えることができます。

VB

```
Dim L1, L2 As New Label    ' Both become type Label
                           ' in the new version of Visual Basic.
L1.Text = "Saving"         ' Assign Text property.
L2.Text = L1.Text          ' Copy Text property.
L2 = L1                    ' Copy object reference.
```

Let は、構文では使用されませんが、Visual Basic 2005 でも予約語です。これにより、以前の意味との混同が避けられます。Visual Basic 2005 では、プロパティの値を設定するプロパティ プロシージャに **Set** ステートメントを使用します。

パラメータ化したプロパティ

引数を受け取る既定のプロパティはあいまいではなく、Visual Basic 2005 でサポートされます。多くの場合、既定のプロパティはコレクション クラスで使用されます。たとえば、System.Windows.Forms 名前空間では、Form クラスは以下の階層をサポートします。

Form オブジェクト

Controls プロパティ (このフォームの ControlCollection オブジェクトを返します)

ControlCollection オブジェクト (既定のプロパティは Item です)

Item プロパティ (コレクション内の 1 つの項目の Control オブジェクトを返します)

Control オブジェクト

Controls プロパティは、**ControlCollection** オブジェクトを返します。**Item** プロパティは、**Control** オブジェクトを返します。次の例は、Visual Basic 2005 の既定のプロパティの、有効な例と無効な例を示しています。

VB

```
Dim F As New Form   ' Assume F has been created and initialized.
```

VB

```
F.Controls.Item(0).Text = "Stop"  ' Valid -- no default properties used.
F.Controls(0).Text = "Stop"  ' Valid -- Item is parameterized.
```

VB

```
'F(0).Text = "Stop"  ' INVALID -- Form does not have a default property.
'F.Controls(0) = "Stop"  ' INVALID -- No default property on Control.
```

既定のプロパティの宣言

Visual Basic 2005 では、宣言の最初に **Default** キーワードを指定することで、プロパティを既定のプロパティとして指定します。プロパティ名をオーバーロードする場合は、オーバーロード宣言ごとに **Default** を指定する必要があります。既定のプロパティは、**Shared** または **Private** では宣言できません。

参照

関連項目

Set ステートメント (Visual Basic)
Default (Visual Basic)
Text
Label
System.Windows.Forms
Form
ControlCollection
Control

概念

プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)
プログラミング要素のサポートに関する変更の概要

プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、簡略化と他のプログラミング言語との相互運用性のために、プロパティのプロシージャとパラメータが更新されています。

Visual Basic 6.0

Visual Basic 6.0 では、**Property Get**、**Property Let**、および **Property Set** の各ステートメントを使ってプロパティ値の取得と設定を行います。

プロパティ パラメータを **ByRef** で宣言できます。このように変数をパラメータに渡すことにより、プロシージャは呼び出し元のコードにある変数を変更できます。

Visual Basic 2005

Visual Basic 2005 では、プロパティの値を取得および設定するプロシージャを含む、統一されたプロパティ宣言の構文が導入されています。これにより、アクセス レベルやオーバーロードなどのプロパティ属性の一貫性が保証されます。次の例は、パラメータを受け取らないプロパティの宣言を示しています。

VB

```
Private monthNum As Integer = 1
Property month() As Integer
    Get
        Return monthNum
    End Get
    Set(ByVal Value As Integer)
        If Value < 1 Or Value > 12 Then
            ' Error processing for invalid value.
        Else
            monthNum = Value
        End If
    End Set
End Property
```

Set 用のパラメータには希望の名前を付けることができます。引数を指定しない場合は、`Value` が自動的に生成されます。

このように構文が変更されたことにより、**Property Get** ステートメントと **Property Set** ステートメントは不要になり、サポートされなくなりました。Visual Basic 2005 では、パラメータのない既定のプロパティは認められないので、オブジェクト参照の割り当てか既定のプロパティの割り当てかを区別するための **Property Let** および **Property Set** は不要になりました。したがって、**Property Let** ステートメントもサポートされません。

Visual Basic 2005 では、**ByRef** プロパティ パラメータはサポートされません。プロパティ プロシージャに **ByRef** パラメータがあった場合、引数の基になる変数が変更されると、プロパティが予測不可能な動作をすることがあります。したがって、パラメータ化したプロパティの宣言では、パラメータに必ず **ByVal** を指定する必要があります。

参照

関連項目

ByVal

概念

既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)

プログラミング要素のサポートに関する変更の概要

制御フローの変更点 (Visual Basic 6.0 ユーザー向け)

次のトピックでは、Visual Basic 6.0 と Visual Basic 2005 における制御フローに関する変更点について説明します。

このセクションの内容

制御ステートメント (Visual Basic 6.0 ユーザー向け)

実行フローを制御するステートメントの変更について説明します。

例外処理 (Visual Basic 6.0 ユーザー向け)

新しくサポートされた構造化例外処理の他、引き続きサポートされる非構造化例外処理についても説明します。

関連するセクション

プログラミング要素のサポートに関する変更の概要

Visual Basic 2005 で変更されたプログラミング要素やサポートされなくなったプログラミング要素をアルファベット順に示します。また、サポートされなくなったプログラミング要素に代わる要素も紹介します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の新機能および追加機能の概要を示します。

制御ステートメント (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、実行のフローを制御するステートメントが更新されています。

Visual Basic 6.0

Visual Basic 6.0 では、**GoSub** ステートメントはプロシージャ内のサブプロシージャを呼び出します。**On ... GoSub** 構造および **On ... GoTo** 構造は、計算後の **GoSub** ステートメントおよび計算後の **GoTo** ステートメントと呼ばれ、初期の BASIC との互換性を維持するために用意されています。

While ... Wend 構造は、指定した条件が true の間、コード内をループします。

Visual Basic 2005

Visual Basic 2005 では、**Call** ステートメントを使用してプロシージャを呼び出すことができます。**GoSub** ステートメントはサポートされていません。**Select...Case** ステートメントを使用して複数分岐を実行できます。**On ... GoSub** 構造と **On ... GoTo** 構造は、サポートされていません。ただし、Visual Basic 2005 では、**On Error** ステートメントはまだサポートされています。

Visual Basic 2005 では、**While ... Wend** 構造は保持されていますが、**Wend** キーワードは **End** ステートメントに置き換えられています。**Wend** キーワードはサポートされません。

参照

関連項目

Call ステートメント (Visual Basic)
Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)
Return ステートメント (Visual Basic)
Select...Case ステートメント (Visual Basic)
On Error ステートメント (Visual Basic)
End ステートメント

概念

プロシージャ呼び出しシーケンス (Visual Basic 6.0 ユーザー向け)
プログラミング要素のサポートに関する変更の概要

例外処理 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、構造化例外処理のサポートが追加されています。非構造化例外処理も引き続きサポートされています。

Visual Basic 6.0

Visual Basic 6.0 では、コードによってエラーを処理するには、非構造化例外処理を使用します。コード ブロックの先頭に **On Error** ステートメントを配置することで、そのブロック内で発生するエラーをすべて処理できます。非構造化例外処理でも、**Error** ステートメントおよび **Resume** ステートメントを使用できます。

Visual Basic 2005

Visual Basic 2005 で実行時のエラーを検出して応答するには、構造化例外処理コードで、コントロール構造体を例外、保護されたコード ブロック、フィルタなどと組み合わせます。構造化例外処理は、**Try**、**Catch**、**Finally** という 3 つのブロックから成る **Try** ステートメントによって実現されます。**Try** ブロックは、実行されるステートメントを含むステートメント ブロックです。**Catch** ブロックは、例外を処理するステートメント ブロックです。**Finally** ブロックには、例外が発生したかどうかにかかわらず、**Try** ステートメントが終了するときに実行されるステートメントを記述します。**Catch** ブロックと組み合わせて使われる **Throw** ステートメントは、Exception クラスの派生クラスのインスタンスによって表される例外を発生させます。

参照

関連項目

Throw ステートメント (Visual Basic)
On Error ステートメント (Visual Basic)
Error ステートメント
Resume ステートメント
Exception

概念

Visual Basic の構造化例外処理の概要
非構造化例外処理の概要
プログラミング要素のサポートに関する変更の概要

オブジェクト指向プログラミング (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、オブジェクト指向言語要素がサポートされ、オブジェクトのサポートがライブラリに分散されています。Visual Basic 2005 では、オブジェクト指向プログラミングのサポートが拡張されて、すべての言語プロパティがサポートされています。

概念の違い

以下のセクションでは、Visual Basic 2005 のオブジェクト指向機能を、Visual Basic 6.0 で実装されている該当の機能と比較して一覧しています。各機能ごとに詳細なヘルプ ページへのリンクが用意されており、各機能について理解を深めることができます。

アクセス レベル

Visual Basic 6.0 では、**Private**、**Friend**、**Public**、および **Static** の各キーワードを使用して、宣言された要素のアクセス レベルを設定します。

Visual Basic 2005 では、**Private**、**Friend**、**Public**、および **Static** の各キーワードに加えて、**Protected** および **Protected Friend** という新しいキーワードを使用して、宣言された要素のアクセス レベルを設定します。宣言された要素のアクセス レベルとは、その要素にアクセスするために必要な権限の程度、つまり、その要素に対する読み取りと書き込みの許可がどのようなコードに与えられるかを示します。

詳細については、「Visual Basic でのアクセス レベル」および「Visual Basic におけるスコープ」を参照してください。

属性

Visual Basic 6.0 では、Visual Basic IDE のプロシージャ属性などのツールを通じて、埋め込み属性が限定的にサポートされます。

Visual Basic 2005 では、**Attribute** は説明的なタグであり、型と型のメンバの宣言ステートメントに注釈を付けることができます。このため、その意味を変更したり、その動作をカスタマイズしたりできます。たとえば、クラス ステートメントとクラス メソッド ステートメントに属性で注釈を付けることができます。

Visual Basic コンパイラなど、ユーザーのアプリケーションや他のアプリケーションは、リフレクションを使用して属性にアクセスし、型と型のメンバの使用方法を判断できます。

属性を使用すると、Visual Basic でアスペクト指向プログラミング (AOP: Aspect Oriented Programming) を実行できます。アスペクトとは、ビジネス ロジックを横断するプログラムの要素です。つまり、プログラムを完成するために必要ですが、必ずしも、プログラムの作成対象であるドメイン固有ではありません。ログや永続性などのアスペクトをビジネス ロジックから分離することが、アスクペクト指向プログラミング パラダイムの目的です。

詳細については、「Visual Basic における属性」、「Visual Basic で使用される属性」、および「属性の一般的な使用方法」を参照してください。

バイナリ互換性

Visual Basic 6.0 では、新規バージョンをコンパイルするときに、**Binary Compatibility** オプションを使用することによって、以前のバージョンのコンポーネントのクラス識別子とインターフェイス識別子を自動的に保持できます。

Visual Basic 2005 では、**Binary Compatibility** オプションはサポートされず、バイナリ互換性は属性で実現されます。これにより、クラス識別子とインターフェイス識別子、仮想テーブル オフセット、適切な COM 属性など、コンパイル済みのコンポーネントに格納される情報を直接制御できます。

詳細については、「バイナリ互換性の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。

クラス モジュール

Visual Basic 6.0 では、クラスはクラス モジュールで定義されます。1 つのクラス モジュールは、ファイル拡張子が .cls の、特別な種類のファイルに格納されます。

Visual Basic 2005 では、クラスは、クラスの名前とメンバを指定する **Class** ステートメントで定義されます。**Class** ステートメントは、ソース ファイルに格納されます。ソース ファイル全体は、プレーンテキストとして表示できます。

複数の **Class** ステートメントと他の型宣言ステートメントを、1 つのソース ファイルに格納できます。Visual Basic では、ソース ファイルの名前が、ソース ファイルで定義されている **Class** や型に一致している必要はありません。

詳細については、「Class ステートメント (Visual Basic)」を参照してください。

クラス コンストラクタ メソッド

Visual Basic 6.0 では、**Class_Initialize** という名前のクラスの **Initialize** イベント ハンドラを使用して、オブジェクト作成時に実行する必要があるコードを実行します。

Visual Basic 2005 では、1 つまたは複数のコンストラクタをクラスに追加して、コードを実行し、変数を初期化します。コンストラクタは、**New** という名前で、クラス内のメソッドです。**New** メソッドをオーバーロードして、同じクラス ステートメント内で **New** という名前の複数のコンストラクタを提供できます。

詳細については、「New (Visual Basic)」または「コンストラクタとデストラクタの使用方法」を参照してください。

クラス デストラクタ メソッド

Visual Basic 6.0 では、オブジェクト変数と実際のオブジェクトの関連付けを解除するのに、**Nothing** キーワードを使用します。オブジェクト変数に **Nothing** 値を割り当てるには、**Set** ステートメントを使用します。

Visual Basic 2005 では、アプリケーションで作成されるオブジェクトの大部分については、ガベージ コレクタによって、必要なメモリ管理タスクを自動的に実行できます。しかし、アンマネージ リソースでは、明示的なクリーンアップが必要です。最も一般的な種類のアンマネージ リソースは、ファイル ハンドル、ウィンドウ ハンドル、ネットワーク接続などのオペレーティング システム リソースをラップしたオブジェクトです。

アンマネージ リソースをカプセル化するオブジェクトを作成する場合は、そのアンマネージ リソースをクリーンアップするために必要なコードをパブリックな `Dispose` メソッドという形で提供することをお勧めします。`Dispose` メソッドを提供すると、ユーザーがオブジェクトを使い終わったときに、そのオブジェクトのメモリを明示的に解放できます。

アンマネージ リソースをカプセル化するオブジェクトを使用する場合は、`Dispose` メソッドの存在を念頭に置き、必要に応じて呼び出すようにしてください。

詳細については、「アンマネージ リソースのクリーンアップ」または「自動メモリ管理」を参照してください。

Class_Initialize

Visual Basic 6.0 では、**Class_Initialize** メソッドによってクラスの **Initialize** イベントを処理し、オブジェクト作成時に実行する必要があるコードを実行します。たとえば、クラス データ変数の値を初期化します。

Visual Basic 2005 では、**Initialize** イベントと **Class_Initialize** ハンドラはサポートされていません。クラス初期化を実行するには、1 つまたは複数のコンストラクタ メソッドを、定義するクラスまたは構造体に追加します。

詳細については、「Class_Initialize の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。

データ クラス

Visual Basic 6.0 では、データ ソース クラスと複合データ コンシューマ クラスを使用して、Microsoft SQL Server データベースなどの外部データを操作します。データ ソース クラスは、外部ソースのデータを提供します。データ コンシューマ クラスは、**Data Source** クラスなど、外部ソース データにバインドできます。

Visual Basic 2005 では、データ ソース、単純データ コンシューマ、複合データ コンシューマ、およびバインディング クラスを使用して、外部データと内部データを操作します。

詳細については、「データ連結と Windows フォーム」を参照してください。

Default プロパティ

Visual Basic 6.0 では、任意のクラス プロパティを、クラスの既定プロパティとして定義できます。

Visual Basic 2005 では、クラスまたは構造体の既定のメンバは、引数を受け取るプロパティだけです。既定プロパティ メンバは、クラスまたは構造体の 1 つのプロパティ宣言ステートメントに **Default** キーワードを指定することによって定義します。

詳細については、「既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。

デリゲート

Visual Basic 6.0 では、デリゲート型はサポートされていません。

Visual Basic 2005 では、デリゲート型は、メソッドへの参照を通じてメソッドを間接的に呼び出すことを可能にする、オブジェクト指向メソッド ポインタの形式をとります。デリゲートを使用すると、イベント ハンドラをフックして、あるメソッドから別のメソッドへメソッドを渡すことができます。

デリゲートを使用すると、非同期のデザイン パターンを実装できます。たとえば、非同期のデリゲートを使用すると、メイン プログラムの実行中に、大きなリストを列挙するメソッドをプログラムから呼び出すことができます。列挙が完了するとコールバックが作成され、プログラムがコールバックを処理します。

詳細については、「Visual Basic でのデリゲート」または「方法 : Visual Basic でプロシージャを別のプロシージャに渡す」を参照してください。

エラー処理

Visual Basic 6.0 では、**On Error** ステートメントを使用して、非構造化例外処理を実行します。

Visual Basic 2005 では、非構造化例外処理と構造化例外処理の両方がサポートされています。構造化例外処理では、例外、分離されたコード ブロック、およびフィルタを含む制御構造を使用して、例外処理機構を作成します。

詳細については、「Visual Basic での構造化例外処理」または「構造化例外処理と非構造化例外処理に適した状況」を参照してください。

イベント

Visual Basic 6.0 では、**Event**、**RaiseEvent**、および **WithEvents** の各キーワードを使用して、イベントを宣言、発生、および処理します。

Visual Basic 2005 では、**Event**、**RaiseEvent**、および **WithEvents** の各キーワードに加えて、**AddHandler**、**RemoveHandler**、および **Handles** という新しいキーワードを使用して、イベントを宣言、発生、および処理します。

AddHandler キーワードと **RemoveHandler** キーワードを使用すると、イベントに関連付けられているイベント ハンドラを動的に追加、削除、および変更できます。

Handles キーワードを使用すると、メソッドで **Handles** 句を定義できます。**Handles** は、プロシージャが指定のイベントを処理することを宣言します。

.NET Framework でのイベントは、デリゲート モデルに基づいています。デリゲート モデルは、オブジェクト指向のオブザーバ デザイン パターンに基づいています。

詳細については、「方法 : イベント ハンドラを記述する」を参照してください。

ジェネリック

Visual Basic 6.0 では、ジェネリック型はサポートされていません。

Visual Basic 2005 では、ジェネリックを使用して、Visual Basic プログラムにおけるパラメータ ポリモーフィズムを実装しています。ジェネリック コードは特定の型を記載せずに記述されるため、多くの新しい型で透過的に使用できます。

Visual Basic 2005 では、ジェネリックなクラス、構造体、インターフェイス、プロシージャ、およびデリゲートで型パラメータがサポートされています。対応する型引数が、ジェネリック型の 1 つの要素のデータ型をコンパイル時に指定します。

詳細については、「Visual Basic におけるジェネリック型」を参照してください。

グローバル クラス

Visual Basic 6.0 では、クラスの **Instancing** プロパティの値が、クラスがプライベートかどうか、つまり、1 つのコンポーネント内でのみ使用されるのか、他のアプリケーションでも使用できるのかを決定します。また、他のアプリケーションがクラスのインスタンスを作成する方法と、クラス メンバを呼び出す方法も決定します。

Visual Basic 2005 では、**Instancing** プロパティはサポートされていません。

アセンブリ内のクラス ステートメントに **Public** キーワードを適用すると、そのアセンブリ内のクラスを他のアセンブリに公開できます。

クラス ライブラリなどの外部アセンブリを参照すると、アプリケーション内のコードが、そのクラス ライブラリの **Public** クラスを使用できるようになります。

Imports キーワードを使用すると、参照先のプロジェクトとアセンブリから名前空間の名前をインポートできます。また、**Imports** キーワードを使用すると、ステートメントを使用するファイルと同じプロジェクト内に定義された、名前空間の名前をインポートできます。

クラスおよび構造体のフィールド、プロパティ、およびメソッド メンバに **Shared** キーワードを適用すると、共有メンバを実装できます。共有メンバは、1 つのクラスまたは構造体のすべてのインスタンスで共有されるプロパティ、プロシージャ、およびフィールドです。クラスの共有メンバには、クラスをインスタンス化せずにアクセスできます。

詳細については、「Visual Basic でのアクセス レベル」、「参照と Imports ステートメント」、または「Visual Basic の共有メンバ」を参照してください。

実装の継承

Visual Basic 6.0 では、実装の継承はサポートされていません。

Visual Basic 2005 では、実装の継承を通じてアドホックなポリモーフィズムを実装できます。これによって、実行時にクライアント コードが相互に使用できるクラスを定義できます。このクラスは、名前は同じでも機能的に異なるメソッドやプロパティを備えています。

派生クラスの元となる基本クラスを定義できます。派生クラスは、基本クラスのプロパティ、メソッド、およびイベントを継承します。また、派生クラスで、基本クラスのプロパティ、メソッド、およびイベントを拡張することもできます。さらに、継承したメソッドを新しい実装でオーバーライドすることもできます。

詳細については、「継承の基本」または「継承を使用する状況」を参照してください。

インターフェイスの継承

Visual Basic 6.0 は、複数の ActiveX インターフェイスを通じてポリモーフィズムを提供します。ActiveX 仕様のインフラストラクチャを形成するコンポーネント オブジェクト モデル (COM) では、複数のインターフェイスを使用することによって、既存のコードを壊すことなく、ソフトウェア コンポーネントのシステムを発展させていくことができます。

Visual Basic 2005 では、.NET Framework の **Interface** キーワードによってアドホックなポリモーフィズムを実装できます。

複数のクラスは同じ **Interface** を実装でき、1 つのクラスは 1 つ以上のインターフェイスを実装できます。インターフェイスは本来、クラスがどのように応答するかを定義したものです。インターフェイスは、クラスが実装する必要のあるメソッド、プロパティ、イベント、および各メンバが受け取り、返す必要のあるパラメータの型を定義しますが、これらのメンバをどのように実装するかは、インターフェイスを実装するクラスによって異なります。

複数のインターフェイスを使用すると、既存のコードを壊すことなく、ソフトウェア コンポーネントのシステムを発展させていくことができます。

詳細については、「インターフェイス ベースのポリモーフィズム」または「方法 : インターフェイスを作成および実装する」を参照してください。

メソッド: ByVal パラメータと ByRef パラメータ

Visual Basic 6.0 では、引数は値渡しまたは参照渡しでメソッドに渡されます。メソッド パラメータが **ByVal** キーワードまたは **ByRef** キーワードで明示的に指定されない場合、引数は参照渡し (**ByRef**) で暗黙に渡されます。

Visual Basic 2005 では、引数は値渡しまたは参照渡しでメソッドに渡されます。メソッド パラメータが **ByVal** キーワードまたは **ByRef** キーワードで明示的に指定されない場合、引数は値渡し (**ByVal**) で暗黙に渡されます。

詳細については、「引数の値渡しおよび参照渡し」を参照してください。

メソッド: オーバーロード

Visual Basic 6.0 では、メソッドのオーバーロードはサポートされていません。

Visual Basic 2005 では、**Method** オーバーロードを使用して、Visual Basic プログラムにおけるアドホックなポリモーフィズムを実装しています。複数のバージョンのメソッドをクラスに定義すると、メソッドはオーバーロードされます。オーバーロードしたバージョンは、パラメータと戻り値の型が異なります。

詳細については、「オーバーロードされたプロパティとメソッド」および「Visual Basic のポリモーフィズムのしくみ」を参照してください。

メソッド: オーバーライド

Visual Basic 6.0 では、メソッドをオーバーライドできません。

Visual Basic 2005 では、**Overrides** キーワードを使用して派生クラス内のメソッドをオーバーライドすることで、派生クラスの基本クラスとは異なるメソッドの実装を提供できます。

詳細については、「プロパティとメソッドのオーバーライド」を参照してください。

メソッド: 結果を返す

Visual Basic 6.0 では、**Function** メソッドの名前を変数名として使用して、**Function** メソッドの結果を返します。

Visual Basic 2005 では、**Return** キーワードを使用して、**Function** メソッドの結果を返します。

Sub、**Function**、または **Property** の各メソッドを呼び出したコードに制御を戻すには、**Return** を使用します。

詳細については、「Return ステートメント (Visual Basic)」を参照してください。

My

Visual Basic 2005 では、新しい **My** 機能が導入され、頻繁に使用する .NET Framework のクラスと関数にエントリ ポイントを提供することによって、Visual Basic での高速なオブジェクト指向プログラミングを実現しています。**My** は、関連機能がタスク ベースの API にまとめられている新しい高度な .NET Framework クラスを備えています。

詳細については、「My による開発」を参照してください。

MyBase、MyClass、および Me キーワード

Visual Basic 6.0 では、**Me** キーワードは、暗黙に宣言された変数と同じように動作します。クラスが複数のインスタンスを持つ場合、**Me** は、コードが実行されている特定のインスタンスのクラスを参照する手段を提供します。

Visual Basic 2005 では、**MyBase** キーワードは、現在のクラス インスタンスの基本クラスを参照する手段を提供します。

MyClass キーワードは、オーバーライドなしで現在のクラス インスタンスを参照する手段を提供します。

Me キーワードを使用すると、コードが実行されているクラスまたは構造体の特定のインスタンスを参照できます。

詳細については、「Visual Basic における Me、My、MyBase、MyClass」を参照してください。

New キーワード

Visual Basic 6.0 では、**New** キーワードを不適切に使用すると、異常な再初期化動作が発生し、これが原因でエラーとなり、メモリが過剰に使用されます。

Visual Basic 2005 では、**New** キーワードはオブジェクトに領域を割り当てるだけであり、異常な再初期化は発生しません。

New 句は、宣言ステートメントまたは代入ステートメントの中で使用します。ステートメントが実行されると、指定したクラスのコンストラクタが呼び出されて、**New** 句に指定した引数が渡されます。したがって、オブジェクトをインスタンス化する時点でデータの制約を強制できます。

クラスには、1 つ以上のコンストラクタがあります。コンストラクタは、**New** という名前の、メソッド内のメソッドです。クラス内で **New** メソッドをオーバーロードすることによって、複数のコンストラクタが定義されます。オーバーロードされたコンストラクタ メソッドは、パラメータ付きのコンストラクタとして記述されます。

詳細については、「New (Visual Basic)」を参照してください。

オブジェクトの有効期間

Visual Basic 6.0 では、オブジェクトの有効期間は確定的であり、すべてのオブジェクト インスタンスが参照カウントを保持します。インスタンスへの最後の参照が解放されると、カウントが 0 になり、オブジェクトはすぐに終了します。

Visual Basic 2005 では、オブジェクトの有効期間は確定的ではなく、最後の参照が解放された後すぐにデストラクタが呼び出されるとは限りません。これは、個別の参照カウントの代わりに、共通言語ランタイムが参照ツリーを保持しているためです。

ガベージ コレクタは、バックグラウンドで参照ツリーをトレースします。現在実行されているコードから参照されていないオブジェクトまたはオブジェクトのグループが見つかると、これらのオブジェクトのデストラクタが呼び出されます。

破棄の順序や、ガベージ コレクタが参照ツリーをトレースするのにかかる時間を予測することは不可能です。

詳細については、「Visual Basic における有効期間」または「オブジェクトの有効期間 : オブジェクトの作成と破棄」を参照してください。

演算子のオーバーロード

Visual Basic 6.0 では、演算子のオーバーロードはサポートされていません。

Visual Basic 2005 では、演算子のオーバーロードによって、標準の演算子 (*、<>、**And** など) の動作を、ユーザー定義のクラスまたは構造体に対して定義できます。

詳細については、「方法 : 変換演算子を定義する」を参照してください。

Option Strict

Visual Basic 6.0 では、実行時に寛容な型チェックが使用されます。

Visual Basic 2005 では、**OPTION STRICT ON** ステートメントを使用して、実行時の寛容な型チェックではなく、デザイン時の厳密な型チェックを有効にできます。既定では **OPTION STRICT OFF** が使用されます。

Visual Basic コンパイラは、データ型を変換する場合に厳密な型指定規則または寛容な型指定規則に基づいて動作します。厳密な型指定規則が有効な場合 (**OPTION STRICT ON**)、拡大変換だけが暗黙に許可されますが、縮小変換は明示的に実行する必要があります。寛容な型指定規則 (**OPTION STRICT OFF**) では、拡大変換と縮小変換をすべて暗黙に試行できます。型変換規則は、オブジェクト型も含めて、すべてのデータ型の変換に適用されます。

詳細については、「Visual Basic における型チェック」を参照してください。

プロパティの構文

Visual Basic 6.0 では、**Property Get**、**Property Let**、および **Property Set** の各ステートメントを使用してプロパティ値を取得および設定します。

Visual Basic 2005 では、プロパティの値を取得および設定するプロシージャを含む、統一されたプロパティ宣言の構文が使用されます。これにより、アクセス レベルやオーバーロードなどのプロパティ属性の一貫性が保証されます。

詳細については、「プロパティの変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。

パブリック インターフェイス

Visual Basic 6.0 では、クラスのパブリック インターフェイスは、そのパブリック メンバのセットです。

Visual Basic 2005 では、クラスのパブリック インターフェイスは、そのパブリック メンバのセットですが、抽象クラスを使用して、Visual Basic プログラムでサブタイプのポリモーフィズムを実装できます。

抽象クラスは、.NET Framework の **Interface** に似ています。抽象クラスは、派生クラスの基本クラスとして動作するようにのみ作成されていま

す。抽象クラスは、抽象的な概念やエンティティを表すためにしばしば使用されます。

抽象クラス自体をインスタンス化することはできません。抽象クラスは継承される必要があります。クラスのメンバの一部または全部を実装しないこともできます。実装は継承したクラスで行われます。実装されたメンバはオーバーライドできます。また、継承したクラスは追加インターフェイスやその他の機能も実装できます。

詳細については、「MustInherit」を参照してください。

読み取り専用プロパティ

Visual Basic 6.0 では、**Property** プロシージャで **Set** メソッドを除外すると、読み取り専用プロパティが作成されます。

Visual Basic 2005 では、プロパティに **ReadOnly** キーワードを適用し、プロパティの **Set** 句を除外して、プロパティを読み取り専用にする必要があります。

フィールドに **ReadOnly** キーワードを適用して、フィールドを読み取り専用にすることもできます。

詳細については、「ReadOnly (Visual Basic)」を参照してください。

リフレクション

Visual Basic 6.0 では、リフレクションはサポートされていません。

Visual Basic 2005 では、.NET Framework クラス ライブラリの System.Reflection 名前空間内のクラスを使用して、クラス、インターフェイス、値型など、型に関する情報を実行時に取得し、型インスタンスを作成して、起動およびアクセスできます。

詳細については、「Visual Studio のリフレクションの名前空間」を参照してください。

共有クラスと構造体メンバ

Visual Basic 2005 では、新しい **Shared** キーワードを **Property**、**Sub**、**Dim**、**Function**、**Operator**、および **Event** の各ステートメントに適用して、クラスまたは構造体のすべてのインスタンス間でこれを共有できます。クラスの共有メンバには、クラスをインスタンス化せずにアクセスできます。

共有メンバには、クラスのインスタンスではなく、クラスの名前を通じてアクセスします。

詳細については、「Visual Basic の共有メンバ」を参照してください。

ソース ファイル

Visual Basic 2005 では、プログラムは 1 つ以上のソース ファイルで構成されています。ソース ファイルは、.vb というファイル拡張子を持つ特別な種類のファイルに格納されます。クラス、構造体、インターフェイス、列挙体、デリゲートなどの型の型宣言ステートメントと、標準モジュールのステートメントは、プログラム ソース コード ファイルに格納されます。Visual Basic では、1 つのソース ファイルで 1 つの型を定義するという制限はなく、ソース ファイルの名前が、ソース ファイル内で定義される型の名前に一致する必要もありません。

部分型宣言ステートメントを使用すると、複数のソース ファイルにまたがってクラスや構造体を定義できます。マルチ ソース ファイル プログラムをコンパイルすると、すべてのソース ファイルが 1 つのファイルに連結されて処理されるかのように、すべてのソース ファイルがまとめて処理されます。

詳細については、「Visual Basic プログラムの構造」または「Partial (Visual Basic)」を参照してください。

ユーザー定義型

Visual Basic 6.0 では、**Type** ステートメントを使用して、さまざまな種類の型の変数を、1 つのユーザー定義型 (UDT) に結合します。

Visual Basic 2005 では、**Structure** ステートメントと **Enum** ステートメントを使用して、ユーザー定義の値型を定義します。

ユーザー定義の参照型を定義するには、**Class**、**Interface**、および **Delegate** の各ステートメントを使用します。

詳細については、「構造体 : 独自のデータ型」を参照してください。

変数の宣言

Visual Basic 6.0 では、同じステートメントで異なる型の変数を宣言できます。ステートメントに変数のデータ型が指定されていない場合、既定で **Variant** が使用されます。

Visual Basic 2005 では、同じデータ型の複数の変数は、ステートメント内で各変数のデータ型を指定せずに宣言できます。

詳細については、「Visual Basic での変数宣言」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、オブジェクト指向のコード コンストラクタは、最も便宜的な方法

で、Visual Basic 2005 の最も等価的な機能に変換されます。得られるコードを、Visual Basic 2005 におけるオブジェクト指向プログラミングの実行例と見なさないでください。

アップグレードしたアプリケーションを拡張して Visual Basic 2005 のオブジェクト指向コンストラクタを使用するには、まず、アップグレードしたコードを変更する必要があります。次のページを参考にしてください。

クラス (Visual Basic 6.0 ユーザー向け)

クラスは、オブジェクト指向アプリケーションを作成するためのビルド ブロックです。

方法 : カスタム コレクションを Visual Basic の型指定されたコレクションに変換する

Visual Basic 2005 には、型指定されたクラスを Visual Basic 6.0 カスタム コレクションから作成するためのオプションがあります。

方法 : ユーザー定義型を Visual Basic の構造体に変換する

Visual Basic 6.0 の **Type** ステートメントは、Visual Basic 2005 では **Structure** ステートメントにアップグレードされました。

方法 : Implements コードを新しい形式の継承に変換する

新しいバージョンの Visual Basic には、**Implements** ステートメントを使用する継承と **Inherits** ステートメントを使用する継承の 2 種類があります。

参照

処理手順

オブジェクト指向プログラミングのサンプル

概念

クラス (Visual Basic 6.0 ユーザー向け)
Visual Basic でのモジュールのオブジェクト指向の変更
インターフェイスの変更点 (Visual Basic 6.0 ユーザー向け)
継承 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

言語の変更点 (Visual Basic 6.0 ユーザー向け)
Visual Basic 6.0 ユーザー向けのヘルプ

クラス (Visual Basic 6.0 ユーザー向け)

クラスは、オブジェクト指向アプリケーションを作るビルド ブロックです。クラスは、システムにおいてオブジェクトを表すプログラミング構造です。アプリケーションを設計するには、アプリケーションを構成するオブジェクト、オブジェクトに格納する情報、およびオブジェクトで実行できる処理を設計することが必要です。クラスとオブジェクト指向プログラミングの詳細については、「Visual Basic におけるオブジェクト指向プログラミング」を参照してください。

クラス モジュール

Visual Basic 6.0

Visual Basic 6.0 では、クラス モジュールを使ってクラスを定義できます。クラスの定義は、.cls ファイルという特別な種類のファイルに保存されます。クラス モジュールごとに 1 つのクラスを定義します。クラス定義の一部は、.cls ファイルに埋め込まれ、プロジェクト システムを使用する方法でしか編集できません。

Visual Basic 2005

Visual Basic 2005 では、クラスはファイル名ではなく Class ステートメント (Visual Basic) を使って定義します。Visual Basic 2005 のクラスは、.vb ファイルというソース コード ファイルに定義します。複数のクラスを 1 つのファイルに定義できます。すべてのクラス定義は、ソース コード内にプレーンテキストとして現れます。

グローバル クラス

Visual Basic 6.0

Visual Basic 6.0 で新しいクラスを作成するときに、**Instancing** プロパティに設定可能な値には **GlobalSingleUse** および **GlobalMultiUse** が含まれます。これらの値は、他のコンポーネントが、新しいクラスのプロパティやメソッドを共有メンバであるかのように呼び出すことができることを示します。クラスのインスタンスは、メンバが最初に呼び出されるときに暗黙的に作成されます。

Visual Basic 2005

Visual Basic 2005 では、**Instancing** プロパティはサポートされていません。それと同じ機能を提供するために、Visual Basic 2005 では標準モジュール メンバを公開できるようになっています。クラスの共有プロパティおよび共有メソッドにアクセスするには、**Imports** ステートメントを使用します。**GlobalMultiUse** の機能は、**Public** クラス アクセスと適切なコンストラクタ アクセスで実行できます。

データ クラス

Visual Basic 6.0

Visual Basic 6.0 では、Data Source クラスと Complex Data Consumer クラスもサポートされています。

Visual Basic 2005

Visual Basic 2005 には、これらに直接相当するものはありません。データを操作するクラスの詳細については、「方法 : オブジェクトのデータに接続する」を参照してください。

既定のメンバ

Visual Basic 6.0

Visual Basic 6.0 では、特定のメソッドまたはデータ メンバをクラスの既定のメンバに指定できます。

Visual Basic 2005

Visual Basic 2005 では、クラスまたは構造体の既定のメンバは、引数を受け取るプロパティだけです。これにより、他のプログラミング言語との相互運用性が実現されます。詳細については、「方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)」を参照してください。

オブジェクトの有効期間

Visual Basic 6.0

オブジェクトの有効期間は、オブジェクト インスタンスの作成および終了によって決定されます。オブジェクトの作成時間は宣言元のプログラムによって決定されますが、終了についてはより複雑な機構がかかわってきます。

Visual Basic 6.0 では、すべてのオブジェクト インスタンスが参照カウントを保持します。インスタンスへの最後の参照が解放されると、カウントが 0 になり、オブジェクトはすぐに終了します。

Visual Basic 2005

Visual Basic 2005 では、最後の参照が解放された後すぐにデストラクタが呼び出されるとは限りません。これは、個別の参照カウントの代わりに、共通言語ランタイムが参照ツリーを保持しているためです。ガベージ コレクタは、バックグラウンドで参照ツリーをトレースします。現在実行されているコードから参照されていないオブジェクトまたはオブジェクトのグループが見つかると、これらのオブジェクトのデストラクタが呼び出されます。破棄の順序や、ガベージ コレクタが参照ツリーをトレースするのにかかる時間を予測することは不可能です。したがって、オブジェクトの有効期間

は不確定になります。詳細については、「オブジェクトの有効期間 : オブジェクトの作成と破棄」および「ガベージ コレクション」を参照してください。

アップグレード オプション

アップグレード ウィザードは、最小限のコードを変更して単純なクラス モジュールを作成します。Visual Basic のプログラマは、上に示した変更点のほかにも、特にオブジェクト指向プログラミングと関連のある変更点から新しい機構を選択できます。その一部を以下に示します。

- **コンストラクタ**　コンストラクタは、Visual Basic 6.0 の **Class_Initialize** に代えて使用できます。詳細については、「Class_Initialize の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。
- **既定のインデックス付きのプロパティ**　既定のインデックス付きのプロパティは、コレクションのアイテムにアクセスする場合に .NET Framework で多用されます。詳細については、「既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。
- **読み取り専用**　Visual Basic 6.0 では、**Get** プロシージャがないプロパティは、暗黙的に読み取り専用になります。Visual Basic 2005 では、**ReadOnly** キーワードを使って、読み取り専用であることを明示的に宣言する必要があります。詳細については、「Property ステートメント」を参照してください。
- **遅延バインディング**　Visual Basic 6.0 は、既定で遅延バインディングを備えています。**Option Strict** を **Off** に設定すると、コードのアップグレード時にエラーが発生しない可能性が高くなります。**Option Strict** を **On** に変更すると、タイプ セーフなコンパイルが実行され、実行時ではなくコンパイル時に微妙なエラーが顕在化することがあります。詳細については、「Option Strict ステートメント」を参照してください。
- **メソッドのオーバーロード**　メソッドのオーバーロードを使うと、名前付きのクラス メンバの数が減ります。それにより、クラスは読みやすくなり、再プログラミングが簡単になります。詳細については、「プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。
- **ファイル名**　Visual Basic 6.0 では、クラス名とファイル名が一致する必要があります。Visual Basic 2005 では、両方の名前をプログラマが自由に決めることや、複数のクラスを同じソース ファイルに定義することができます。1 つのファイルにいくつのクラスを定義するかは、通常はコーディング標準で定めます。
- **ByRef**　Visual Basic 6.0 では、パラメータの既定は **ByRef** なので、アップグレード時にも **ByRef** を使うのが安全です。詳細については、「ByVal」を参照してください。
- **Return キーワード**　Visual Basic 6.0 では、関数名が、関数の値を返すための変数として使用されます。Visual Basic 2005 では、関数の値を返すための明示的な **Return** キーワードが用意されています。詳細については、「Return ステートメント (Visual Basic)」を参照してください。
- **演算子のオーバーロード**　独自に定義したクラスのために、加算 (+)、減算 (-)、およびその他の演算子の意味を定義できます。詳細については、「Operator ステートメント」を参照してください。
- **新しい演算子**　+= 演算子を使用して、コード行を短く簡略化して書くことができます。詳細については、「算術演算子 (Visual Basic)」を参照してください。
- **新しい値のインライン定義**　Visual Basic 2005 の構文では、新しい変数の値の宣言と設定を 1 行のコードで記述できます。したがって、オブジェクトをインスタンス化する時点でデータの制約を強制できます。詳細については、「Dim ステートメント (Visual Basic)」を参照してください。

危険を冒さずにコードを書き直すことはできません。コードを変更した場合は、その部分をテストする必要があります。また、Visual Basic 2005 では、一部のキーワードは Visual Basic 6.0 で使用した場合と正確に同じ動作はしません。詳細については、「以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード」を参照してください。

参照

関連項目

Structure ステートメント
Property ステートメント
Option Strict ステートメント
Return ステートメント (Visual Basic)
'ByVal' および 'ByRef' を組み合わせて使用することはできません。
ByRef

概念

事前バインディングと遅延バインディング
Class_Initialize の変更点 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

オブジェクトの作成と使用
クラスについて

Class_Initialize の変更点 (Visual Basic 6.0 ユーザー向け)

コンストラクタは、オブジェクトを作成するときに実行するメソッドです。コンストラクタは、これまで Visual Basic にありませんでした。コンストラクタは、**New** ステートメントと Visual Basic 6.0 の **Class_Initialize** メソッドの動作を併せ持ちますが、クラス インスタンスを作成するための柔軟性と制御の機能をさらに高いレベルで備えています。オブジェクトの作成の詳細については、「オブジェクトの作成と使用」を参照してください。

Class_Initialize

Visual Basic 6.0

Visual Basic 6.0 では、コンストラクタの概念は **Class_Initialize** メソッドをとおしてサポートされます。このメソッドはプライベートであり、パラメータは指定できません。このメソッドは、クラスの新しいインスタンスを作成するときに自動的に呼び出されます。**New** キーワードを呼び出すと、**Class_Initialize** メソッドが呼び出されます (このメソッドが存在しない場合は呼び出されません)。

たとえば、既定を 5 年間として、樹木の年間成長率を指定するとします。クラス コードは次のようになります。

```
' Visual Basic 6.0
Private mvarYearlyGrowth As Integer

Public Property Get YearlyGrowth() As Integer
    YearlyGrowth = mvarYearlyGrowth
End Property

Public Property Let YearlyGrowth(ByVal newValue As Integer)
    mvarYearlyGrowth = newValue
End Property

Private Sub Class_Initialize()
    mvarHeight = 5
End Sub
```

樹木を作成し、プロパティを設定するコードは、次のようになります。

```
Dim growingTree As New Tree
growingTree.YearlyGrowth = 10
```

Visual Basic 2005

Visual Basic 2005 では、コンストラクタは **New** キーワードをとおしてサポートされます。この場合、`YearlyGrowth` プロパティを設定するコードは不要です。**New** キーワードを呼び出すと、クラスのオーバーロードされた **New** メソッドの 1 つが呼び出されます。クラスに **New** メソッドが存在しない場合は、このメソッドがパラメータなしでコンパイラによって作成されます。

VB

```
Option Strict On
```

VB

```
Public Class Tree
  Private yearlyGrowthValue As Integer = 5

  Public Property YearlyGrowth() As Integer
      Get
          Return yearlyGrowthValue
      End Get
      Set(ByVal Value As Integer)
          yearlyGrowthValue = Value
      End Set
  End Property


  Public Sub New(ByVal newYearlyGrowth As Integer)
      Me.YearlyGrowth = newYearlyGrowth
  End Sub
```

```
End Class
```

樹木を作成し、プロパティを設定するコードは、次のようになります。

VB

```
Dim growingTree As New Tree(10)
```

この例では、年間成長率を指定しないで `Tree` を作成することはできません。これをできるようにするには、パラメータのない別の `New` メソッドをクラスに追加します。これは、コンストラクタを "オーバーロードする" と呼ばれます。

New キーワード

Visual Basic 6.0

Visual Basic 6.0 では、次のようなコード行は一般に推奨されません。

```
Dim growingTree As New Tree
```

その理由は、`growingTree` 変数にアクセスするたびに、コンパイラでは値が **Nothing** かどうかがチェックされるからです。この値であれば、新しいインスタンスが作成され、`growingTree` に割り当てられます。これは効率が悪いだけでなく、プログラミング エラーにもつながります。

Visual Basic 2005

Visual Basic 2005 では、インスタンスはチェックされず、新しいインスタンスは作成されません。新しいインスタンスを作成する唯一の方法は、**New** または **As New** キーワードを使ってコード行を実行することです。事実上、このようなコード行は、新しいインスタンスの作成を扱う際に現在推奨される唯一の方法です。

アップグレードのヒント

アップグレード ウィザードは、**Class_Initialize** メソッドを次のコードにアップグレードします。

VB

```
'UPGRADE_NOTE: Class_Initialize was upgraded to Class_Initialize_Renamed. Click for more: m
s-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1061"'

Private mvarHeight As Short
Private Sub Class_Initialize_Renamed()
    mvarHeight = 0
End Sub

Public Sub New()
    MyBase.New()
    Class_Initialize_Renamed()
End Sub
```

`Class_Initialize_Renamed` の呼び出しを削除し、このコードをコンストラクタに直接移動してもかまいません。不要なプロシージャ呼び出しが省かれるので、コードが読みやすくなります。ほとんどの **New** 呼び出しの後で特定のプロパティを設定している場合は、それらのプロパティを設定するコンストラクタを追加することをお勧めします。Visual Basic 6.0 コードは、Visual Basic 2005 では次のコードで置き換えることができます。

VB

```
Public Sub New(ByVal newYearlyGrowth As Integer)
    Me.YearlyGrowth = newYearlyGrowth
End Sub
```

参照

処理手順

方法 : New キーワードを使用する

その他の技術情報

オブジェクトの作成と使用

Visual Basic のプロパティに関する変更点

フィールドとプロパティを組み合わせて使うことにより、クラスのデータにアクセスできます。フィールドは、クラス定義における変数であり、オブジェクトのデータを格納します。プロパティは、クラス インスタンス データにアクセスできるように **Get** メソッドと **Set** メソッドを提供します。詳細については、「プロパティとプロパティ プロシージャ」を参照してください。

プロパティの構文

Visual Basic 6.0

Property 構造は、**Get** メソッドと **Set** メソッドを組み合わせたものです。Visual Basic 6.0 では、これらの 2 つのメソッドは分離されています。

```
' Visual Basic 6.0
Private mvarYearlyGrowth As Integer
Public Property Get YearlyGrowth() As Integer
    YearlyGrowth = mvarYearlyGrowth
End Property

Public Property Let YearlyGrowth(ByVal newValue As Integer)
    mvarYearlyGrowth = newValue
End Property
```

Visual Basic 2005

Visual Basic 6.0 と Visual Basic 2005 におけるプロパティの変更は、主に構文の変更です。Visual Basic 2005 の構文では、**Get** メソッドと **Set** メソッドは、独立した 2 つのメソッドとするのではなく、プロパティの 1 つのコード ブロックに移動されています。Visual Basic 2005 では、参照型であるプロパティと値型であるプロパティは区別されないので、**Set** メソッドは 1 つしかなく、**Let** メソッドはなくなりました。`YearlyGrowth` のフィールドとプロパティは、Visual Basic 2005 ではこのようなものになります。

VB

```
Private yearlyGrowthValue As Integer = 5

Public Property YearlyGrowth() As Integer
    Get
        Return yearlyGrowthValue
    End Get
    Set(ByVal Value As Integer)
        yearlyGrowthValue = Value
    End Set
End Property
```

Visual Basic 6.0 と Visual Basic 2005 の両方で、このクラスには、データを格納する `mvarYearlyGrowth` フィールドまたは `yearlyGrowthValue` フィールドがあります。

アップグレードのヒント

アップグレード ウィザードは、プロパティを次のようにアップグレードします。

VB

```
Private mvarYearlyGrowth As Short

Public Property YearlyGrowth() As Short
    Get
        YearlyGrowth = mvarYearlyGrowth
    End Get
    Set(ByVal Value As Short)
        mvarYearlyGrowth = Value
    End Set
End Property
```

Visual Basic 6.0 の **Integer** は、Visual Basic 2005 では **Short** にアップグレードされます。これが原因で、Visual Basic 2005 の他のコード

で **Integer** をこのプロパティに割り当てようとすると、型の相互運用の問題が起こることがあります。変換したフィールドやプロパティの型を調べて、他のタイプ セーフの競合を避ける必要があります。

参照

関連項目

Dim ステートメント (Visual Basic)

Property ステートメント

その他の技術情報

プロパティの変更点 (Visual Basic 6.0 ユーザー向け)

メソッドの変更点 (Visual Basic 6.0 ユーザー向け)

メソッドは、**Sub** プロシージャ宣言または **Function** プロシージャ宣言です。メソッドによって、クラスの動作が提供されます。プロパティとの組み合わせにより、クラスの動作とデータが公開されます。

メソッドの宣言

Visual Basic 6.0

Visual Basic 6.0 のクラス定義では、**Sub** メソッドと **Function** メソッドを使用できます。これらのメソッドは、**Public**、**Private**、または **Friend** として宣言できます。2 つの例を次に示します。

```
Public Sub Draw(theForm As Form)
    ' Add code here to draw a tree on the form.
End Sub

Public Function GrowOneYear() As Integer
    mvarHeight = mvarHeight + 10
    GrowOneYear = mvarHeight
End Function
```

Visual Basic 2005

Visual Basic 2005 では、**Public**、**Private**、および **Friend** である **Sub** メソッドと **Function** メソッドもサポートされます。メソッドのすべてのパラメータには、**ByRef** または **ByVal** を指定する必要があります。明示的に追加しない場合、コード エディタによって **ByVal** が追加されます。

Visual Basic 6.0 からアップグレードされたコードでは、Visual Basic 6.0 での既定である **ByRef** にアップグレードされます。

VB

```
Public Sub Draw(ByRef theForm As System.Windows.Forms.Form)
    ' Add code here to draw a tree on the form.
End Sub
```

メソッドのオーバーロード

Visual Basic 6.0

Visual Basic 6.0 では、1 つのクラスに同じ名前のメソッドは複数存在できません。

Visual Basic 2005

Visual Basic 2005 では、メソッドはオーバーロードできます。複数のバージョンのメソッドをクラスに定義すると、メソッドはオーバーロードされます。オーバーロードしたバージョンは、パラメータと戻り値の型が異なります。これにより、オブジェクトのインターフェイスは小さく、単純になります。次のコード例は、オーバーロードした 2 つの `Draw` メソッドを示しています。詳細については、「オーバーロードされたプロパティとメソッド」を参照してください。

VB

```
Public Sub Draw(ByRef theForm As System.Windows.Forms.Form)
    ' Add code here to draw a tree on the form.
End Sub
```

VB

```
Public Sub Draw(ByVal surface As System.Drawing.Graphics)
    ' Add code here to draw a tree on the graphics surface.
End Sub
```

Return キーワード

Visual Basic 6.0

Visual Basic 6.0 では、関数の値を返す変数として関数の名前が使用されます。

```
Public Function GrowOneYear() As Integer
    heightValue = heightValue + 10
    GrowOneYear = heightValue
End Function
```

Visual Basic 2005

Visual Basic 2005 では、関数の値を返すときには明示的に **Return** ステートメントを指定します。Visual Basic 2005 では、`GrowOneYear` クラスは次のようになります。

VB

```
Public Function GrowOneYear() As Integer
    heightValue += 10
    Return heightValue
End Function
```

詳細については、「Return ステートメント (Visual Basic)」を参照してください。

アップグレードのヒント

アップグレード ウィザードによるアップグレードで、上記の `Draw` メソッドから次のコードが生成されます。

VB

```
Public Sub Draw(ByRef theForm As System.Windows.Forms.Form)
    ' Add code here to draw a tree on the form.
End Sub
```

アップグレードしたメソッドには、次のテクニックを使用できます。

- 名前と動作は似ているがパラメータと戻り値が異なるメソッドに、メソッドのオーバーロードを使用します。
- メソッドのスコープ (**Public**、**Private**、**Friend**) と戻り値の型 (**Short**、**Integer**、など) を確認します。
- 関数の値を返すために関数名を使う代わりに **Return** ステートメントを使用します。

参照

関連項目

Sub ステートメント (Visual Basic)
Function ステートメント (Visual Basic)
Return ステートメント (Visual Basic)

継承 (Visual Basic 6.0 ユーザー向け)

継承を使うと、既存のクラスから新しいクラスを作成できます。継承は、複数のクラスを関連付ける構造を提供することにより、アプリケーションの設計を単純化できます。また、新しいクラスの動作またはそれまでとは異なるクラスの動作だけを記述すればよいので、コードを再利用することもできます。

インターフェイスの継承

Visual Basic 6.0

Visual Basic 6.0 は、インターフェイスの継承をサポートします。詳細については、「インターフェイスの変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Visual Basic 2005

Visual Basic 2005 は、インターフェイスの継承を Interface ステートメント (Visual Basic) および Implements ステートメント でサポートします。インターフェイスの詳細については、「Visual Basic におけるインターフェイス」を参照してください。

実装の継承

実装の継承を使って新しいクラスを作成すると、基本クラスのすべてのメンバと実装が自動的に新しいクラスに与えられます。既存のクラスは、"基本クラス" と呼ばれます。新しいクラスは、"派生クラス" です。このクラスには、さらにメンバを追加できます。新しいクラスでコードを書くことにより、基本クラスの動作を変更できます。このテクニックは、オーバーライドと呼ばれています。

クラスを作成するには、以前に作成したクラス、プロジェクトに追加した参照内のクラス、または .NET Framework のオブジェクトを継承します。.NET Framework の多くのクラスは、継承によって関連付けられています。たとえば、TextBox クラスは、System.Windows.Forms.TextBoxBase クラスを継承します。

Visual Basic 6.0

Visual Basic 6.0 は、実装の継承をサポートしません。

Visual Basic 2005

Visual Basic 2005 では、**Inherits** ステートメントで継承を宣言します。次の例では、`Oak` 派生クラスは基本クラス `Tree` を継承します。

VB

```
Public Class Oak
    Inherits Tree
    ' Add code here to extend or
    ' modify the behavior of the Tree class.
End Class
```

MustInherit および NotInheritable

Visual Basic 6.0

Visual Basic 6.0 のインターフェイスの継承では、どのクラスもインターフェイス基本クラスとして動作できます。インターフェイス基本クラスとして動作しないようにクラスを設定する構文はありません。同様に、インターフェイス基本クラスとしてのみ動作するようにクラスを設定する構文もありません。

Visual Basic 2005

Visual Basic 2005 では、実装の継承と共に、MustInherit および NotInheritable という 2 つのクラス修飾子を定義します。これらの修飾子は、アプリケーション内の継承関係を制御するために使用できます。

クラス宣言の **MustInherit** 修飾子は、インスタンス化できないクラスであることを示します。

VB

```
Public MustInherit Class BaseClass
End Class
```

つまり、**New** キーワードの後に `BaseClass` を記述できません。インスタンス化できるのは、`BaseClass` を継承し、**MustInherit** 修飾子のないクラスだけです。オブジェクト指向の専門書や他のオブジェクト指向言語では、**MustInherit** クラスは "抽象" クラスと呼ばれています。**MustInherit** クラスではないクラス (インスタンス化できるクラス) は、"具象" クラスと呼ばれています。

これと関連する概念として、シール クラス (基本クラスとして使用できないクラス) があります。このステータスを示すクラス定義上のキーワードは **NotInheritable** です。

VB

```
Public NotInheritable Class DerivedClass
End Class
```

NotInheritable クラスは、継承の階層の末端に位置します。

詳細については、「Visual Basic の継承」を参照してください。

アップグレードのヒント

インターフェイスとして動作する Visual Basic 6.0 のクラスは、Visual Basic 2005 ではインターフェイスにアップグレードされます。次のような Visual Basic 6.0 の基本クラスと派生クラスがあるとします。

```
' Contents of class BaseClass
Public Sub BaseMethod()
End Sub

' Contents of class DerivedClass
Implements BaseClass
Private Sub BaseClass_BaseMethod()
End Sub
```

アップグレード ウィザードによるアップグレードで、次のコードが生成されます。

VB

```
Option Strict Off
Option Explicit On
Interface _BaseClass
    Sub BaseMethod()
End Interface

Friend Class BaseClass
    Implements _BaseClass
    Public Sub BaseMethod() Implements _BaseClass.BaseMethod
    End Sub
End Class

Friend Class DerivedClass
    Implements _BaseClass
    Private Sub BaseClass_BaseMethod() Implements _BaseClass.BaseMethod
    End Sub
End Class
```

インターフェイスの代わりに継承を使って、アップグレードしたコードを次のように変更できます。

VB

```
Friend Class BaseClass
    Public Sub BaseMethod()
        ' Add code here to define BaseMethod.
    End Sub
End Class

Friend Class DerivedClass
    Inherits BaseClass
End Class
```

これで、1 レベルの間接参照 (`_BaseClass`) が省かれます。また、1 つのメソッド定義 (`BaseClass_BaseMethod`) も省かれます。このメソッドは、基本クラスから継承されるので、コードを改めて記述する必要はありません。派生クラスの動作を変更する必要がある場合は、`BaseMethod`

を次のようにオーバーライドできます。

VB

```
Friend Class BaseClass
    Public Overridable Sub BaseMethod()
        ' Add code here to define BaseMethod.
    End Sub
End Class

Friend Class DerivedClass
    Inherits BaseClass
    Public Overrides Sub BaseMethod()
        ' Add code here to define behavior for DerivedClass.
    End Sub
End Class
```

アップグレードしたインターフェイスには、次のテクニックを使用できます。

- **Interface** ステートメントを **Inherits** ステートメントで置き換えます。
- 基本クラスを、インターフェイスを実装するクラスとして実装するのではなく、インターフェイスとして実装します。詳細については、「インターフェイスの変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。
- 基本クラスを **MustInherit** クラスとして実装します。
- 派生クラスを **NotInheritable** として実装します。

参照

その他の技術情報

Visual Basic の継承

Visual Basic におけるインターフェイス

イベントおよび例外の変更点 (Visual Basic 6.0 ユーザー向け)

イベントと例外は、クラス内での変更をクライアント コードに通知するために使用するクラス メンバです。通常、イベントは予期された変更を通知し、クライアント コードではそれに応答するかどうかを選択できます。例外は、エラーの状態を通知し、クライアント コードでは自身の責任においてエラーを無視できます。詳細については、「Visual Basic での構造化例外処理」を参照してください。

イベント

Visual Basic 6.0

イベントを宣言する構文は、Visual Basic 6.0 から変更されていません。

```
Public Event StateChanged()
```

Visual Basic 2005

.NET Framework のクラス ライブラリでは、イベントのコーディングに新しい規則が追加されています。通常、イベントには、*sender* という名前の Object と *e* という名前の EventArgs の 2 つのパラメータがあります。イベントが発生すると、イベントの発生元のオブジェクトは、それ自身への参照を *sender* パラメータとして渡します。**EventArgs** パラメータは、**EventArgs** オブジェクトか、**EventArgs** から派生したクラスのインスタンスです。派生クラスは、クライアント コードの役に立つイベント情報を保持します。以下に例を示します。

VB

```
Public Event StateChanged(ByVal sender As Object, ByVal e As EventArgs)
```

例外

Visual Basic 6.0

Visual Basic 6.0 では、例外はサポートしていませんが、エラーは発生します。エラーを発生させるには、**On Error GoTo** ステートメントとラベル付きステートメントを使用します。

Visual Basic 2005

エラーを発生する機能は、Visual Basic 2005 でサポートされています。詳細については、「非構造化例外処理の概要」を参照してください。

例外は、Visual Basic 2005 で新しくサポートされた機能です。Exception クラスは、すべての例外の基本クラスとなります。詳細については、「Exception クラスとプロパティ」を参照してください。Visual Basic 6.0 の場合のようにエラー番号を作成するのではなく、派生クラスを作成します。これにより、コードが読みやすくなります。**GoTo** ステートメントとラベル付きステートメントの代わりに Try...Catch...Finally ステートメント (Visual Basic) ブロック構造を使って、例外に応答します。以下に例を示します。

VB

```
Dim big As Integer = Integer.MaxValue
Dim bigger As Integer = Integer.MaxValue
Try
    Dim biggest As Integer = big + bigger
Catch ex As Exception
    MsgBox("Addition did not succeed.")
End Try
```

詳細については、「Visual Basic の構造化例外処理の概要」を参照してください。

アップグレードのヒント

Visual Basic 6.0 でエラーを発生させるコードは、Visual Basic 2005 でもエラーを発生させるコードに変換されます。エラーを発生させるコードの変換例を以下に示します。

VB

```
Private mvarAge As Short
Public Property Age() As Short
    Get
        Age = mvarAge
    End Get
```

```
    Set(ByVal Value As Short)
        If Value >= 0 Then
            mvarAge = Value
        Else
            Err.Raise(vbObjectError + 2, Me, _
                "Age must be greater than or equal to zero.")
        End If
    End Set
End Property
```

エラーの代わりに例外を使うには、`Err.Raise` 呼び出しを Throw ステートメント (Visual Basic) で置き換えます。エラー番号を使わずに済むという利点もあります。

イベントと例外を Visual Basic 2005 にアップグレードするときには、次のことを検討してください。

- **On Error GoTo** ステートメントを **Try** ステートメントに置き換えます。
- **Err.Raise** を **Throw** に置き換えます。
- イベントにパラメータ規則を使用します (`Public Event StateChanged(ByVal sender As Object, ByVal e As EventArgs)`)。
- 継承を使用して、カスタムの例外や **EventArgs** クラスを作成します。

参照

関連項目

Throw ステートメント (Visual Basic)

EventArgs

概念

Visual Basic の構造化例外処理の概要

その他の技術情報

Visual Basic での例外およびエラー処理

インターフェイスの変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定義では、インターフェイスとは、パブリック メンバのリストを定義する型を指です。**Interface** 型は、インスタンス化も実装もできません。

インターフェイスとクラス

クラスのインターフェイスは、そのクラスのパブリック メンバのリストです。Visual Basic 2005 で **Interface** ステートメントを使うと、パブリック メンバのリストを定義する型を宣言できます。**Implements** ステートメントを記述したクラスでは、インターフェイスの各メンバを実装することにより、そのインターフェイスのメンバを追加します。これは、**Inherits** キーワードを使った継承の実装と対照的です。継承の実装では、メンバは基本クラスから継承されるので、派生クラスにメンバを実装する必要はありません。

多くの場合に .NET Framework では、継承は、なんらかのサービスをアプリケーションでサポートするためにクラスで使用されます。たとえば、.NET Framework では、クラスが System.IComparable インターフェイスを実装している場合、ArrayList に格納されたそのクラスのインスタンスをランタイムで並べ替えることができます。

Visual Basic 2005 でのインターフェイスの使い方の詳細については、「Visual Basic におけるインターフェイス」を参照してください。

Visual Basic 6.0

Visual Basic 6.0 では、どのクラスもインターフェイスとして動作できます。Implements ステートメント を次のように使用して、別のクラスのインターフェイスをクラスに実装できます。

```
' Contents of class BaseClass
Public Sub BaseMethod()
End Sub

' Contents of class DerivedClass
Implements BaseClass
Private Sub BaseClass_BaseMethod()
End Sub
```

クラスの個々のメンバにコードを記述する必要はありませんが、記述してもかまいません。

Visual Basic 2005

Visual Basic 2005 では、クラスとインターフェイスは明確に区別されます。クラスの宣言には **Class** ステートメントを使用し、インターフェイスの宣言には **Interface** ステートメントを使用します。クラスは、インターフェイスとして動作できません。インターフェイスとして動作するには、型の宣言に **Interface** キーワードを使う必要があります。インターフェイスのメンバは、実装されません。実際、`End Sub` などのコードをインターフェイス定義で使うことは、構文で認められません。インターフェイスのメンバは、**Implements** ステートメントを使ってインターフェイスを宣言したクラスに実装されます。

VB

```
Interface Printable
    Sub Print()
    Property Mode() As Integer
End Interface

Public Class Tree
    Implements Printable

    Private modeValue As Integer
    Public Property Mode() As Integer Implements Printable.Mode
        Get
            Return modeValue
        End Get
        Set(ByVal Value As Integer)
            modeValue = Value
        End Set
    End Property

    Public Sub Print() Implements Printable.Print
        ' Add code to print a tree.
    End Sub
End Class
```

アップグレードのヒント

Visual Basic 6.0 で継承を実装する唯一の方法は、**Implements** ステートメントと基本クラスを使うことです。新しいバージョンの Visual Basic では、実装の継承とインターフェイスの継承という 2 つの種類の継承があります。どの種類を選択するかは、アプリケーションによります。次のことを考慮してください。

- Inherits ステートメント による実装の継承を使うと、コードを書き加えなくても新しいクラスを作成できます。基本クラスの動作を変更する必要がある場合にだけ、コードを追加します。インターフェイスの継承を使う場合は、基本クラスの各メンバにコードを追加する必要があります。
- インターフェイスは、複数のクラスから継承する必要がある場合に使用できます。Visual Basic 2005 では、1 つのクラスからの継承しかサポートされませんが、1 つのクラスに複数のインターフェイスを実装できます。
- .NET Framework には、共通のプログラミング タスクを単純化する複数のインターフェイスが定義されています。たとえば、**IComparable**、ISerializable、IFormattable などがあります。
- 多くの .NET Framework インターフェイスには、インターフェイスを実装するクラスも用意されています。たとえば、Component クラスは、IComponent インターフェイスを実装するクラスです。Inherits ステートメント を使って **Component** から継承することにより、クラスは、コードを書くことなく **IComponent** インターフェイスのすべての機能を取得します。

参照

関連項目

Interface ステートメント (Visual Basic)
Class ステートメント (Visual Basic)
Inherits ステートメント

Visual Basic でのモジュールのオブジェクト指向の変更

共有メンバは、クラスのすべてのメンバで使用できるが、クラスの特定のインスタンスには関連付けられていないクラス メンバ (プロパティやプロシージャなど) です。共有メンバの詳細については、「Visual Basic の共有メンバ」を参照してください。

インスタンス データと共有データ

インスタンス データは、**New** キーワードの実行時に作成される一連のデータです。それに対して、クラスの共有データは **New** キーワードの実行とは無関係に作成されます。

Visual Basic 6.0

Visual Basic 6.0 では、クラス定義で共有データはサポートされません。データをクラス インスタンス間で共有するには、モジュールでグローバル変数を使用します。

Visual Basic 2005

共有データを宣言するには、次のように Shared (Visual Basic) 修飾子を変数宣言のステートメントに追加します。

VB

```
Private Shared totalInstances As Integer
```

共有メソッド

Visual Basic 6.0

Visual Basic 6.0 では、共有クラス メンバはサポートされていません。同じような機能は、モジュールで使用できます。ランタイムは、モジュール内にデータのインスタンスを 1 つだけ作成します。モジュールのデータとメソッドのスコープは、プロジェクト全体です。ライブラリをモジュールに実装した例を以下に示します。

```
Public Sub GetTitle(wordXml As String)
    ' Add code here to find title in Xml string.
End Sub

Public Sub ReplaceTitle(wordXml As String, newTitle As String)
    ' Add code here to replace title.
End Sub
```

Visual Basic 2005

メンバがライブラリとして互いに関連する場合や、メンバが既存のクラスに関連する場合、よりオブジェクト指向性の強い方法でモジュールは共有クラス メンバを使用します。この設計では、すべてのメンバが共有されるのか、共有メンバとインスタンス メンバがクラスに混在するのかが確認されます。クラスに共有メンバだけが含まれる場合は、プライベートなコンストラクタだけを追加して、クラスがクライアント コードでインスタンス化されないようにします。次のコード例は、クライアント コードでインスタンス化できないクラスと、メンバの 1 つを呼び出すクライアント コードを示します。

VB

```
Public Class WordLibrary
    Public Shared Sub GetTitle(ByVal wordXml As String)
        ' Add code here to find title in Xml string.
    End Sub

    Public Shared Sub ReplaceTitle(ByVal wordXml As String, _
        ByVal newTitle As String)
        ' Add code here to replace title.
    End Sub

    Private Sub New()
        ' This prevents instantiation of the class in the client.
    End Sub
End Class
```

VB

```
' Code that calls the library method.
Sub ChangeTheTitle()
    WordLibrary.ReplaceTitle("Old Title", "New Title")
End Sub
```

このライブラリにモジュールではなくクラスを使う利点は、以下のとおりです。

- ライブラリに名前 (`WordLibrary`) があり、コード内に明示されます。
- メソッドが 1 つのクラスにまとめられ、互いに関連性があることが示されます。
- プライベートなコンストラクタを使うことで、このクラスが一連のユーティリティ メソッドを提供することが明確になります。
- クライアント コードでライブラリ名を使用すると、コードを読むときにプロジェクトの編成がわかります。

.NET Framework では、共有メソッドは、クラスのインスタンスでは必要とされないがクラスには関連する機能を提供するために主に使用されます。たとえば、**Parse** メソッドは、.NET Framework の多くのデータ型でサポートされています。整数型 (Integer) (Visual Basic) は、文字列をパラメータとして受け取る **Parse** メソッドをサポートしています。この **Parse** メソッドは、文字列で表現された整数を返します。**Parse** メソッドをメンバ メソッドにするのは不合理です。整数ではなく、文字列しか扱わないからです。

共有メンバへのアクセス

Visual Basic 6.0

Visual Basic 6.0 では、共有クラス メンバはサポートされていません。

Visual Basic 2005

共有メンバには、次のようにクラス名とインスタンス名を使ってアクセスできます。

VB

```
WordLibrary.ReplaceTitle("Old Title", "New Title")
```

クラスのインスタンスへの参照を受け取った場合を除き、共有メソッドからはインスタンス データにアクセスできません。

アップグレードのヒント

Visual Basic 6.0 の `WordLibrary` モジュールは、次のコードにアップグレードされます。

VB

```
Module WordLibrary
    Public Sub GetTitle(ByRef wordXml As String)
        ' Add code here to find title in Xml string.
    End Sub

    Public Sub ReplaceTitle(ByRef wordXml As String, _
        ByRef newTitle As String)
        ' Add code here to replace title.
    End Sub
End Module
```

モジュールのコードを Visual Basic 6.0 から Visual Basic 2005 にアップグレードする場合は、次の点に注意してください。

- モジュールのメソッドとデータは、共有データと共有メソッドとしてクラスに編成します。
- 既存のクラスに関連するモジュール メソッドは、共有メソッドとしてそのクラスに移動します。

参照

概念

Visual Basic の共有メンバ

バイナリ互換性の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、以前のバージョンのコンポーネントとの互換性を維持する機構が更新されています。

バイナリ互換性

Visual Basic 6.0

Visual Basic 6.0 では、新規バージョンをコンパイルするときに、[バイナリ互換] オプションを使用することによって、以前のバージョンのコンポーネントのクラス識別子とインターフェイス識別子を自動的に保持できます。新規バージョンには古い識別子と新しい識別子の両方が含まれるため、クライアント アプリケーションでは、コンポーネントへの古いインターフェイスを更新することなく使用できます。

Visual Basic 2005

Visual Basic 2005 では、バイナリ互換性は属性を使用して実現されます。これにより、クラス識別子とインターフェイス識別子、仮想テーブル オフセット、適切な COM 属性など、コンパイル済みのコンポーネントに格納される情報を直接制御できます。維持する属性を明示的に選択できるため、Visual Basic 6.0 の [バイナリ互換] オプションはサポートされません。

参照

関連項目

ComClassAttribute クラス

概念

Visual Basic における属性
Visual Basic で使用される属性
Visual Basic におけるグローバル属性
プログラミング要素のサポートに関する変更の概要

方法：カスタム コレクションを Visual Basic の型指定されたコレクションに変換する

Visual Basic 6.0 のカスタム コレクションでは、コレクションのコンテンツは特定の 1 つのクラスに制限されます。これは、型指定されたコレクションとも呼ばれます。Visual Basic 2005 では、Visual Basic 6.0 のカスタム コレクションから型指定されたクラスを作成する方法がいくつかあります。ここでは、3 つのプロシージャを示してそれぞれの方法を説明します。

説明では、Visual Basic 6.0 のカスタム コレクションがクラス ビルダ ユーティリティを使って作成されていると仮定します。このようなコレクションには、次のメンバがあります。

- **Add** カスタム クラスの新しいインスタンスをコレクションに追加します。
- **Item** コレクションのインデックスに基づいてコレクションからインスタンスを返します。
- **Count** コレクション内のインスタンスの数を返します。
- **Remove** コレクションのインデックスに基づいてコレクションからインスタンスを削除します。
- **Enumeration For Each** 構文を使う列挙をサポートします。

アップグレード ウィザードを使ってコレクションを作成するには

- Visual Basic 6.0 プロジェクトを Visual Basic 2005 で開きます。`Forest` という名前のコレクション クラスに `Tree` クラスのインスタンスが含まれているとすると、コードは次のようにアップグレードされます。この方法の利点は、コードを変更する必要がないことです。

VB

```
' For this example, the Tree class has no members.
Option Strict Off
Option Explicit On
Friend Class Tree
End Class

Friend Class Forest
  Implements System.Collections.IEnumerable
  'local variable to hold collection
  Private mCol As Collection

  Public Function Add(Optional ByRef sKey As String = "") As Tree
    'create a new object
    Dim objNewMember As Tree
    objNewMember = New Tree
    'set the properties passed into the method
    If Len(sKey) = 0 Then
      mCol.Add(objNewMember)
    Else
      mCol.Add(objNewMember, sKey)
    End If
    'return the object created
    Add = objNewMember
    'UPGRADE_NOTE: Object objNewMember may not be destroyed until it is garbage collec
ted. Click for more: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyword="
vbup1029"'

    objNewMember = Nothing
  End Function

  Default Public ReadOnly Property Item(ByVal vntIndexKey _
    As Object) As Tree
    Get
```

```
      'used when referencing an element in the collection
      'vntIndexKey contains either the Index or Key to the collection,
      'this is why it is declared as a Variant
      'Syntax: Set foo = x.Item(xyz) or Set foo = x.Item(5)
      Item = mCol.Item(vntIndexKey)
    End Get
  End Property

  Public ReadOnly Property Count() As Integer
    Get
      'used when retrieving the number of elements in the
      'collection. Syntax: Debug.Print x.Count
      Count = mCol.Count()
    End Get
  End Property

  'UPGRADE_NOTE: NewEnum property was commented out. Click for more: 'ms-help://MS.MSD
NQTR.80.en/commoner/redir/redirect.htm?keyword="vbup1054"'

  'Public ReadOnly Property NewEnum() As stdole.IUnknown
    'Get
      'this property allows you to enumerate
      'this collection with the For...Each syntax
      'NewEnum = mCol._NewEnum
    'End Get
  'End Property

  Public Function GetEnumerator() As System.Collections.IEnumerator _
    Implements System.Collections.IEnumerable.GetEnumerator
    'UPGRADE_TODO: Uncomment and change the following line to return the collection en
umerator. Click for more: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyw
ord="vbup1055"'

    'GetEnumerator = mCol.GetEnumerator
  End Function

  Public Sub Remove(ByRef vntIndexKey As Object)
    'used when removing an element from the collection
    'vntIndexKey contains either the Index or Key, which is why
    'it is declared as a Variant
    'Syntax: x.Remove(xyz)
    mCol.Remove(vntIndexKey)
  End Sub

  'UPGRADE_NOTE: Class_Initialize was upgraded to Class_Initialize_Renamed. Click for
more: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyword="vbup1061"'

  Private Sub Class_Initialize_Renamed()
    'creates the collection when this class is created
    mCol = New Collection
  End Sub
  Public Sub New()
    MyBase.New()
    Class_Initialize_Renamed()
  End Sub

  'UPGRADE_NOTE: Class_Terminate was upgraded to Class_Terminate_Renamed. Click for mo
re: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyword="vbup1061"'
```

```
  Private Sub Class_Terminate_Renamed()
    'destroys collection when this class is terminated
    'UPGRADE_NOTE: Object mCol may not be destroyed until it is garbage collected. Cli
ck for more: 'ms-help://MS.MSDNQTR.80.en/commoner/redir/redirect.htm?keyword="vbup1029
"'

    mCol = Nothing
  End Sub
  Protected Overrides Sub Finalize()
    Class_Terminate_Renamed()
    MyBase.Finalize()
  End Sub
End Class
```

.NET Framework と Visual Basic 2005 には、ジェネリック コレクションが用意されています。ジェネリック クラスを使う利点は、クラスを実装するために必要なコードが非常に少ないことです。

ジェネリック コレクションを使ってコレクションを作成するには

1. クラス定義を作成します。`Tree` クラスの例を次に示します。

   VB

   ```
   Public Class Tree
       Public Species As String
   End Class
   ```

2. .NET Framework からジェネリック リスト クラスを作成します。この宣言は、「アップグレード ウィザードを使ってコレクションを作成するには」で作成手順を説明した `Forest` クラス全体に基本的に置き換わります。

   VB

   ```
   Dim forest As New System.Collections.Generic.List(Of Tree)
   ```

3. リスト オブジェクトにアクセスするコードを作成します。次のコードでは、`Tree` クラスの 5 つのインスタンスをコレクションに追加し、それらを出力します。

   VB

   ```
   For count As Integer = 1 To 5
       Dim sapling As New Tree
       sapling.Species = "oak"
       Forest.Add(sapling)
   Next

   For Each sapling As Tree In Forest
       MsgBox(sapling.Species)
   Next
   ```

.NET Framework には、CollectionBase クラスがあります。**CollectionBase** を継承することにより、型指定されたコレクションを作成できます。この方法では、使用するコードはアップグレード ウィザードの場合よりも少なくなりますが、ジェネリックを使用するソリューションよりは多くなります。メンバをコレクション クラスに追加できるので、ジェネリックのソリューションより柔軟性があります。

コレクションを CollectionBase クラスから作成するには

1. クラス定義を作成します。`Tree` クラスの例を次に示します。

VB

```
Public Class Tree
    Public Species As String
End Class
```

2. **CollectionBase** クラスを継承するクラスを作成します。少なくとも `Add` メソッドと `Item` プロパティを追加します。これにより、コレクション クラスは厳密に型指定されます。

VB

```
Class TreeCollection
  Inherits System.Collections.CollectionBase

  Public Sub Add(ByVal value As Tree)
    Me.List.Add(value)
  End Sub

  Default Public Property Item(ByVal index As Integer) As Tree
      Get
          Return CType(Me.List(index), Tree)
      End Get
      Set(ByVal value As Tree)
          If index <= Me.Count - 1 Then
              Me.List(index) = value
          Else
              Throw New IndexOutOfRangeException()
          End If
      End Set
  End Property
End Class
```

3. リスト オブジェクトにアクセスするコードを作成します。次のコードでは、`Tree` クラスの 5 つのインスタンスをコレクションに追加し、それらを出力します。

VB

```
Dim forest As New TreeCollection
```

VB

```
For count As Integer = 1 To 5
    Dim sapling As New Tree
    sapling.Species = "oak"
    Forest.Add(sapling)
Next

For Each sapling As Tree In Forest
    MsgBox(sapling.Species)
Next
```

参照

関連項目

List
CollectionBase
Ienumerable

方法：ユーザー定義型を Visual Basic の構造体に変換する

Visual Basic 6.0 の **Type** ステートメントは、Visual Basic 2005 では **Structure** ステートメントにアップグレードされます。また、型をアップグレードする他の 2 つのオプションである、クラスへの変換およびインターフェイスへの変換についても説明します。

以下では、次の Visual Basic 6.0 型を例として説明を進めます。

```
Type Customer
    Dim FirstName As String
    Dim LastName As String
    Dim Id As Long
End Type

Private Function FormatFullName(aCustomer As Customer) As String
    FormatFullName = aCustomer.FirstName & " " & aCustomer.LastName
End Function
```

アップグレード ウィザードは、Visual Basic 6.0 の型を自動的に Visual Basic 2005 の構造体にアップグレードします。**Structure** は、メソッド、プロパティなどのメンバをサポートするデータ型です。

型を構造体にアップグレードするには

1. アップグレード ウィザードを実行します。

   メモ：

   アップグレード ウィザードは、Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードするためのツールです。Visual Basic 2005 で Visual Basic 6.0 プロジェクトを開くと、このウィザードが自動的に起動します。詳細については、「方法 : Visual Basic アップグレード ウィザードを使ってプロジェクトをアップグレードする」を参照してください。

2. **Structure** 型の他の機能を利用する場合は、構造体に含めるのが適切なメソッドを探します。次のコード例では、`FormatFullName` メソッドが構造体内の `FormatFullName` メソッドに変換されます。

   VB

   ```
   Structure Customer
       Dim FirstName As String
       Dim LastName As String
       Dim Id As Integer

       Public Function FormatFullName() As String
           Return FirstName & " " & LastName
       End Function
   End Structure
   ```

アップグレード ウィザードは、Visual Basic 6.0 の型を自動的に Visual Basic 2005 の構造体にアップグレードします。**Class** 型は、サポートしているメンバは **Structure** と同じですが、参照型です。クラスは基本クラスになることができますが、構造体は基本クラスになれません。

クラスへアップグレードするには

1. アップグレード ウィザードを実行します。
2. `Structure` を `Class` に置き換えます。
3. **Class** 型の他の機能を利用する場合は、構造体に含めるのが適切なメソッドを探します。次のコード例は、クラスへの `FormatFullName` の組み込みと `Id` プロパティを示しています。

   VB

   ```
   Class Customer
   ```

```
    Dim FirstName As String
    Dim LastName As String
    Private idValue As Integer

    Public Property Id() As Integer
        Get
            Return idValue
        End Get
        Set(ByVal Value As Integer)
            idValue = Value
        End Set
    End Property

    Public Function FormatFullName() As String
        Return FirstName & " " & LastName
    End Function
End Class
```

構造体をインターフェイスに変換するという、3 番目のオプションがあります。

インターフェイスへアップグレードするには

1. アップグレード ウィザードを実行します。
2. `Structure` を `Interface` に置き換えます。
3. 変数をプロパティ宣言に置き換えます。実装は含めません。
4. メソッドは追加しますが、実装は削除します。
5. コードは次のようになります。

   VB

```
Interface Customer
    Property FirstName() As String
    Property LastName() As String
    Property Id() As String

    Function FormatFullName() As String
End Interface
```

6. このインターフェイスは、別のクラスに実装できます。定義を以下に示します。

   VB

```
Class NewCustomer
    Implements Customer

    Public Property FirstName() As String Implements Customer.FirstName
        Get
            ' Add code here.
        End Get
        Set(ByVal Value As String)
            ' Add code here.
        End Set
    End Property

    Public Property Id() As String Implements Customer.Id
        Get
```

```
            ' Add code here.
        End Get
        Set(ByVal Value As String)
            ' Add code here.
        End Set
    End Property

    Public Property LastName() As String Implements Customer.LastName
        Get
            ' Add code here.
        End Get
        Set(ByVal Value As String)
            ' Add code here.
        End Set
    End Property

    Public Function FormatFullName() As String _
        Implements Customer.FormatFullName
        ' Add code here.
    End Function
End Class
```

参照

処理手順

方法 : Visual Basic アップグレード ウィザードを使ってプロジェクトをアップグレードする

関連項目

Structure ステートメント

Class ステートメント (Visual Basic)

Interface ステートメント (Visual Basic)

方法：Implements コードを新しい形式の継承に変換する

Visual Basic 6.0 では、**Implements** ステートメントをインターフェイスの継承に使用します。新しいバージョンの Visual Basic には、Implements ステートメント を使用する継承と Inherits ステートメント を使用する継承の 2 種類が用意されています。ここでは、このような 2 つのオプションを使用するコードを変換する方法を示します。

以下の Visual Basic 6.0 コードがこのページにあるとします。

```
' Account class.
Private mvarBalance As Currency
Private mvarOwner As String

Public Property Let Owner(ByVal vData As String)
    mvarOwner = vData
End Property

Public Property Get Owner() As String
    Owner = mvarOwner
End Property

Public Function Deposit(ByVal amount As Currency) As Currency
End Function

' Savings class.
Private mvarBalance As Currency
Private mvarOwner As String 'local copy

Public Property Let Owner(ByVal vData As String)
    mvarOwner = vData
End Property

Public Property Get Owner() As String
    Owner = mvarOwner
End Property

Public Function Deposit(ByVal amount As Currency) As Currency
End Function
```

共通のインターフェイスを使用して変換するには

1. アップグレード ウィザードを実行します。

   メモ：

   アップグレード ウィザードは、Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードするためのツールです。Visual Basic 2005 で Visual Basic 6.0 プロジェクトを開くと、このウィザードが自動的に起動します。詳細については、「方法 : Visual Basic アップグレード ウィザードを使ってプロジェクトをアップグレードする」を参照してください。

   ウィザードでは、`Account` クラスがインターフェイス (`_Account`) に変換され、このインターフェイスを実装するクラスが 2 つ (`Account` および `Savings`) 作成されます。

   VB

   ```
   Interface _Account
       Property Owner() As String
       Function Deposit(ByVal amount As Decimal) As Decimal
   End Interface
   Friend Class Account
       Implements _Account
       Private mvarBalance As Decimal
       Private mvarOwner As String
   ```

```
    Public Property Owner() As String Implements _Account.Owner
        Get
            Owner = mvarOwner
        End Get
        Set(ByVal Value As String)
            mvarOwner = Value
        End Set
    End Property

    Public Function Deposit(ByVal amount As Decimal) _
        As Decimal Implements _Account.Deposit
    End Function
End Class

Friend Class Savings
    Implements _Account

    Private Function Account_Deposit(ByVal amount As Decimal) _
        As Decimal Implements _Account.Deposit
    End Function

    Private Property Account_Owner() As String _
        Implements _Account.Owner
        Get
        End Get
        Set(ByVal Value As String)
        End Set
    End Property

    Public Function AddInterest() As Object
    End Function
End Class
```

2. `Savings` アカウントのインスタンスを作成し、使用するコードは、次のようになります。

   VB

```
Dim savings As _Account
savings = New Savings
savings.Owner = "Jeff Pike"
```

`Account` を基本クラスにする必要がある場合は、ウィザードで作成されるコードを変更できます。

Interface および Implements を使用して変換するには

1. アップグレード ウィザードを実行します。
2. `Account` クラスを削除します。
3. `_Account` クラスの名前を `Account` に変更します。
4. `Savings` クラスで、`_Account` を `Account` に置き換えます。

   VB

```
Friend Class Savings
    Implements Account

    Private Function Account_Deposit(ByVal amount As Decimal) _
```

```
        As Decimal Implements Account.Deposit
    End Function

    Private Property Account_Owner() As String Implements Account.Owner
        Get
        End Get
        Set(ByVal Value As String)
        End Set
    End Property

    Public Function AddInterest() As Object
    End Function
End Class
```

5. Savings アカウントのインスタンスを作成し、使用するコードは、次のようになります。

VB

```
Dim savings As Account
savings = New Savings
savings.Owner = "Jeff Pike"
```

`Account` を基本クラスにする必要がある場合は、ウィザードで作成されるコードを変更できます。

Class および Inherits を使用して変換するには

1. アップグレード ウィザードを実行します。
2. `_Account` インターフェイスを削除します。
3. `Account` クラスで、**Implements** ステートメント内の `_Account` への参照を削除します。

VB

```
Friend Class Account
    Private mvarBalance As Decimal
    Private mvarOwner As String

    Public Property Owner() As String
        Get
            Owner = mvarOwner
        End Get
        Set(ByVal Value As String)
            mvarOwner = Value
        End Set
    End Property

    Public Function Deposit(ByVal amount As Decimal) As Decimal
    End Function
End Class
```

4. `Savings` クラスで、`Implements _Account` を `Inherits Account` に置き換えます。
5. `Savings` クラスで、`Owner` メンバと `Deposit` メンバを削除します。

VB

```
Friend Class Savings
    Inherits Account
```

```
    Public Function AddInterest() As Object
    End Function
End Class
```

6. `Savings` アカウントのインスタンスを作成し、使用するコードは、次のようになります。

VB

```
Dim savings As Account
savings = New Savings
savings.Owner = "Jeff Pike"
```

参照

処理手順

方法 : Visual Basic アップグレード ウィザードを使ってプロジェクトをアップグレードする

関連項目

Class ステートメント (Visual Basic)
Inherits ステートメント
Interface ステートメント (Visual Basic)
Implements ステートメント

その他の技術情報

Visual Basic の継承
Visual Basic におけるインターフェイス

Web プログラミング (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、Web プログラミングは直接サポートされます。Visual Basic 2005 では、Web プログラミングは Visual Web Developer と Visual Basic 言語を使用して行われます。

概念の違い

Visual Basic 6.0 には、Web 用途のプログラミングをサポートする機能として、IIS (インターネット インフォメーション サービス) アプリケーション (Web クラス)、DHTML アプリケーション、ActiveX ドキュメント、および Web ページにダウンロードできる ActiveX コントロールがあります。

Visual Basic 2005 では、Web プログラミングは直接サポートされていません。代わりに、Visual Web Developer と Visual Basic 言語を使用して、ASP.NET Web サイトや ASP.NET Web サービスなどを作成できます。Visual Basic 2005 では、Visual Web Developer で使用できる Web コントロールを作成できます。また、Visual Basic 2005 アプリケーションまたはコンポーネントから ASP.NET Web サービスを利用できます。

ASP と ASP.NET

Visual Basic 6.0 では、IIS (インターネット インフォメーション サービス) アプリケーションは、Active Server Pages (ASP) モデルを使用して、IIS 上で動作するアプリケーションを作成します。ASP.NET のテクノロジを使用すると、Web フォーム ページを使用して Web サイトを作成できます。ASP.NET Web サイト アプリケーションも、IIS 上でホストされます。

DHTML アプリケーション

Visual Basic 6.0 では、ブラウザ上のユーザー アクションに応答する DHTML アプリケーションの作成には、ダイナミック HTML オブジェクト モデルおよび Visual Basic コードが使用されます。Visual Web Developer の Web フォームは、DHTML モデルを強化したものであり、クライアント側の検証に加え、より豊富なダイナミック ユーザー インターフェイスを実現しています。

ActiveX ドキュメントと ActiveX コントロール

Visual Basic 6.0 の ActiveX ドキュメントは Visual Basic 2005 ではサポートされません。ActiveX ドキュメントを Visual Web Developer Web サイト アプリケーションと相互運用することはできますが、開発自体は Visual Basic 6.0 で行う必要があります。

Visual Basic 2005 では、Web ページにダウンロードして利用される ActiveX コントロールを作成したり、Visual Basic 2005 アプリケーションで既存の ActiveX コントロールを使用したりすることもできます。

参照

概念

Visual Basic 6.0 と現在のバージョンの Visual Basic の両方を使用する

その他の技術情報

Visual Web Developer
Visual Basic 6.0 ユーザー向けのヘルプ
Visual Studio の Web の名前空間
XML Web サービスを利用した Web のプログラミング

WebClass (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 における ASP.NET Web サイトのプログラミングの概念と方法は、ほとんどの部分に関して、Visual Basic 6.0 で Web クラスを作成する概念と方法とは大きく異なります。

概念の違い

Visual Basic 6.0 では、WebClass プロジェクト (IIS アプリケーション プロジェクトとも呼ばれます) を使用して、ASP ページ テクノロジに基づく Web アプリケーションを作成します。

Visual Basic 2005 では、ASP.NET Web サイト プロジェクトを使用して、最新の ASP.NET テクノロジに基づく Web アプリケーションを作成します。詳細については、「Visual Web Developer で Web サイトを作成するためのガイド ツアー」を参照してください。

状態管理

Visual Basic 6.0 では、WebClass プロジェクトの **StateManagement** プロパティを使用して、要求間で Web クラスのインスタンスを保持できます。これを行うには、デザイン時に **StateManagement** プロパティを **2** または **wcRetainInstance** に設定します。また、**ReleaseInstance** メソッドを使用してインスタンスを終了することもできます。

Visual Basic 2005 では、ASP.NET Web アプリケーションには **StateManagement** プロパティはありません。アプリケーション状態を管理するモデルは大きく異なり、状態管理に関係するコードを置き換える必要があります。詳細については、「ASP.NET の状態管理」を参照してください。

アップグレード メモ

Visual Basic 6.0 の WebClass プロジェクトを Visual Basic 2005 にアップグレードすると、ASP.NET Web サイト プロジェクトに変換されます。

プロジェクトには、WebClass に 1 つ、WebClass プロジェクトの WebItems と Templates にそれぞれ 1 つ宣言が追加されます。プロジェクトには **Page_Load** イベント プロシージャが追加され、まず WebClass オブジェクトが作成されます。次に、Visual Basic 6.0 の WebClass プロジェクトに関連付けられている WebItems と Templates のそれぞれについて WebItem オブジェクトが作成されます。最後に、**Page_Load** イベント プロシージャに、WebClass 互換性ランタイム (**WebClass.ProcessEvents**) への呼び出しが追加されます。これにより、要求 URL で指定された WebItem が、ランタイムによって表示されます。このコードは、アップグレードされたプロジェクトに新しく追加される唯一のコードであり、Visual Basic 6.0 の WebClass ランタイムの基底の動作をエミュレートするだけです。

Visual Basic 6.0 のコードの **Function** プロシージャと **Sub** プロシージャ (**ProcessTags** や **Respond** など) は、WebClass 互換性ランタイムがこれらのプロシージャを実行できるように、スコープが **Private** から **Public** に変更されています。

Visual Basic 6.0 の WebClass イベントの中には、ASP.NET でサポートされていないイベントがあります。たとえば、**Initialize**、**BeginRequest**、**EndRequest**、**Terminate** などです。これらのイベント プロシージャは、アップグレードはされるものの、実行時に呼び出されることはありません。アップグレードした後で、これらのイベントのコードを、対応する ASP.NET イベントに移動する必要があります。

メモ :

ASP オブジェクトと ASP.NET オブジェクトでは、動作の異なるプロパティ、メソッド、およびイベントがいくつかあります。WebClass を ASP.NET にアップグレードする前に、そうした動作の違いについて把握し、コードをどのように修正する必要があるかを判断できるようにしておく必要があります。詳細については、「Visual Web Developer で Web サイトを作成するためのガイド ツアー」を参照してください。

参照

関連項目

WebClass イベントがサポートされていない

概念

Visual Basic 2005 への WebClass プロジェクトのアップグレード

その他の技術情報

ASP.NET Web ページの概要 (Visual Studio)

データ アクセス (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、データ アクセスに ActiveX データ オブジェクト (ADO: ActiveX Data Objects) を使用しています。Visual Basic 2005 では、データ アクセスは .NET Framework の構成要素である ADO.NET を使用して実現されます。2 つのテクノロジの間には、概念的にもタスクの観点からも、いくつかの相違点があります。ADO と ADO.NET の概念的な相違の詳細については、「ADO.NET と ADO の比較」を参照してください。

メモ：

Visual Basic 6.0 は、リモート データ オブジェクト (RDO: Remote Data Object) およびデータ アクセス オブジェクト (DAO: Data Access Object) に対して下位互換性をサポートしています。Visual Basic 2005 では、これらのテクノロジが COM オブジェクトとしてのみサポートされます。RDO または DAO のデータ バインディングを使用したアプリケーションは、Visual Basic 2005 にアップグレードできません。詳細については、「以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード」を参照してください。

Visual Basic 6.0 では、アプリケーションにデータ アクセスを実装する方法として、一般に 2 つの方法を使用しています。1 つはデザイン時に ADODC (ADO データ コントロール) をバインドするかデータ環境を使用する方法であり、もう 1 つはプログラムによって実行時に **Recordset** オブジェクトを作成することにより対話する方法です。

Visual Basic 2005 では、データ アクセスを実装する方法として、類似した 2 つの方法があります。1 つはデザイン時にデータ アダプタとデータセットを使用する方法であり、もう 1 つはデータ アダプタとデータセットを作成するコードをプログラムに追加することにより実行時に実装する方法です。詳細については、「データアダプタの作成」を参照してください。

Visual Basic 6.0 では、**DataChanged**、**DataField**、**DataFormat**、**DataMember**、および **DataSource** という、コントロールのバインディング関連のプロパティを設定することによって、データ バインディングが実現されます。ほとんどの場合、コントロールの表示プロパティ (**TextBox** コントロールの **Text** プロパティなど) がデータ ソースのフィールドにバインドされています。

Visual Basic 2005 では、データ バインディングがより広範に定義されています。コントロールの種類にかかわらず、すべてのプロパティをデータの格納先となるあらゆる構造体にバインドできます。Windows フォームにおけるバインディングは、コントロールの **Binding** オブジェクトのコレクションが格納される **DataBindings** プロパティを通じて実現されます。詳細については、「データ連結と Windows フォーム」を参照してください。

メモ：

ADO データ バインディングの下位互換性は、Microsoft Visual Basic 2005 の Compatibility Data Runtime を通じてサポートされます。

Visual Basic 6.0 では、データ環境からフォームにデータベース要素をドラッグすると、データ バインド フォームが自動的に作成されます。

参照

その他の技術情報

クライアント データ アプリケーションの作成
サーバー エクスプローラまたはデータベース エクスプローラによるデータへの接続
Visual Database Tools
Visual Basic 6.0 ユーザー向けのヘルプ
Visual Studio のデータの名前空間

コンポーネントの作成 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 のどちらにもコンポーネント作成機能が備わっていますが、コンポーネントの作成方法にはいくつかの重要な相違があります。

概念の違い

Visual Basic 6.0 でのコンポーネントの作成とは、COM アプリケーションで使用できる ActiveX コントロール、ActiveX DLL、ActiveX EXE など、COM コンポーネントを作成することです。

Visual Basic 2005 では、コンポーネントは .NET Framework に基づいています。つまり、.NET Framework を使用して作成されたアプリケーションで使用できるコンポーネントを作成します。Visual Basic 2005 を使用して作成されるコンポーネントは、継承に基づいています。つまり、すべてのコンポーネントは、**Component** または **Control** のいずれかの基本クラスから派生します。詳細については、「コンポーネントのクラス」を参照してください。

マルチスレッド コンポーネント

Visual Basic 6.0 ではスレッド処理に 2 つのモデルが用意されており、コンポーネントはシングルスレッドとアパートメント スレッドのいずれかです。

Visual Basic 2005 では、本当のマルチスレッド コンポーネントがサポートされます。詳細については、「コンポーネントのマルチスレッド」を参照してください。

コンポーネントのインスタンス化

Visual Basic 6.0 では、アクセス レベルおよびコンポーネントの作成方法が、クラスの **Instancing** プロパティによって制御されます。

Visual Basic 2005 では、コンポーネントのクラスごとに、コンストラクタでアクセス修飾子とアクセス レベルを設定することによりインスタンス化を制御します。詳細については、「Visual Basic .NET におけるコンポーネントのインスタンス化の変更点」を参照してください。

バイナリ互換性とバージョン管理

Visual Basic 6.0 では、バージョンの競合を避けるためにコンポーネントのバイナリ互換性が重要であり、コンポーネント プロジェクトの **Version Compatibility** プロパティによって制御されています。

Visual Basic 2005 の場合、バージョン管理がコンポーネントのアセンブリに組み込まれているため、バイナリ互換性を設定する煩わしさから解放されます。詳細については、「コンポーネントによるプログラミング」を参照してください。

ActiveX コントロール

Visual Basic 6.0 では、Windows ベースのアプリケーション用の ActiveX コントロール (ユーザー コントロール) を作成できます。

Visual Basic 2005 では、Windows フォーム用のユーザー コントロールを作成するか、または既存のコントロールを継承して機能を追加できます。詳細については、「デザイン時の Windows フォーム コントロールの開発」を参照してください。

メッセージング

Visual Basic 6.0 では、アプリケーション間で情報をやり取りするために Microsoft メッセージ キュー (MSMQ) を使用するコンポーネントを作成できます。

Visual Basic 2005 では、メッセージング機能が .NET Framework に組み込まれています。詳細については、「メッセージング コンポーネントの使用」を参照してください。

トランザクション

Visual Basic 6.0 では、Microsoft Transaction Server (MTS) と連携する MTS コンポーネントを作成することによって、トランザクションを処理できます。

Visual Basic 2005 では、.NET Framework によってトランザクションが処理されます。これは、トランザクション コンポーネントをプロジェクトに追加するか、または既存のクラスに **Transaction** 属性を追加することによって行われます。

参照

概念

User コントロール (Visual Basic 6.0 ユーザー向け)
継承 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

コンポーネントの作成
Visual Basic 6.0 ユーザー向けのヘルプ
Visual Studio のコンポーネント モデルの名前空間

User コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 も Visual Basic 2005 もコントロールを作成できますが、注意が必要ないくつかの違いがあります。

概念の違い

Visual Basic 6.0 では、**UserControl** プロジェクト (ActiveX コントロール プロジェクトとも呼ばれます) を使用して ActiveX コントロールを作成します。ActiveX コントロールのコンパイル後には、Visual Basic 6.0 フォームや Internet Explorer などの ActiveX をサポートするすべてのコンテナで UserControl をホストできます。

Visual Basic 2005 では、Windows コントロール ライブラリ プロジェクトを使用して、Windows フォーム アプリケーションでホストできるコントロールを作成します。

デザイン時の動作

Visual Basic 6.0 では、デザイン時に **UserControl** オブジェクトをフォームに追加する前に UserControl デザイナを閉じる必要があり、それによってコントロールが自動的にコンパイル (ビルド) されます。それ以降に **UserControl** オブジェクト を変更しても、アプリケーションを実行するまでフォーム上の **UserControl** のインスタンスには変更内容が反映されず、UserControl デザイナを開くと自動的に閉じられます。

Visual Basic 2005 では、UserControl デザイナを閉じずに、フォームに **UserControl** オブジェクトを追加できます。ただし、事前にユーザー コントロールをビルドしておく必要があります。UserControl デザイナを閉じただけでは、コントロールはビルドされません。それ以降にユーザー コントロールを変更しても、**UserControl** オブジェクトを再度ビルドするまでフォーム デザイナには変更内容が反映されません。テスト アプリケーションをデバッグするたびに、**UserControl** オブジェクトは 自動的に再度ビルドされます。

値の永続化

Visual Basic 6.0 では、**ReadProperties** イベントと **WriteProperties** イベントを使用して、**UserControl** オブジェクトの値を取得したり、値を **PropertyBag** オブジェクトに保存したりします。

Visual Basic 2005 では、**PropertyBag** オブジェクトはサポートされていません。また、**ReadProperties** イベントおよび **WriteProperties** イベントはありません。代わりに、シリアル化を使用して、プロパティをバイナリ形式または SOAP 形式で保存および取得します。詳細については、「チュートリアル : Visual Basic でのオブジェクトの永続化」を参照してください。

アップグレード メモ

Visual Basic 6.0 の **UserControl** プロジェクトを Visual Basic 2005 にアップグレードすると、Windows コントロール ライブラリにアップグレードされるため、Internet Explorer ではコントロールをホストできなくなります。

参照

処理手順

チュートリアル : Visual Basic でのオブジェクトの永続化

概念

PropertyBag の対応関係 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

デザイン時の Windows フォーム コントロールの開発

PropertyBag の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **PropertyBag** オブジェクトは、Visual Basic 2005 では **My.Settings** オブジェクトに置き換えられています。

概念の違い

Visual Basic 6.0 では、複数インスタンス間で使用するオブジェクトのデータを永続化する場合、**PropertyBag** オブジェクトを使用して値を保存し、次にそのオブジェクトのインスタンスを生成するときにその値を取得できます。デザイン時にオブジェクトのプロパティを既定値に設定できますが、実行時に入力した値はオブジェクトが破棄されるとすべて失われます。Visual Basic 6.0 の **PropertyBag** オブジェクトを使用すると、オブジェクトのコピーをバイナリ形式で永続化し、後で取得して再利用できます。たとえば、ローン金額の計算に使用するアプリケーションで **PropertyBag** オブジェクトを使用して、インスタンス間で利率を永続化すると、アプリケーションを実行するたびに利率を入力する必要がなくなります。

Visual Basic 2005 では、**PropertyBag** オブジェクトはなくなりましたが、**My.Settings** オブジェクトを使用すると、オブジェクトのデータを永続化できます。**My.Settings** オブジェクトを使用することで、コンポーネントまたはアプリケーションの設定にアクセスすることと、デザイン時および実行時にプロパティ設定やその他の設定を動的に格納および取得することができます。詳細については、「アプリケーションの設定の管理」を参照してください。

メモ :

My.Settings オブジェクトは、.NET Framework で定義されている型のデータを永続化する場合のみ使用できます。カスタム データ型の場合は、シリアル化を使用してデータを永続化します。詳細については、「チュートリアル : Visual Basic でのオブジェクトの永続化」を参照してください。

参照

概念

User コントロール (Visual Basic 6.0 ユーザー向け)

Windows フォーム (Visual Basic 6.0 ユーザー向け)

Visual Basic の 6.0 以前のバージョンでは、Windows ベースのアプリケーションの作成に、Visual Basic ランタイムの中心的な構成要素であるフォームが使用されていました。Visual Basic 2005 では、Windows ベースのアプリケーションの作成に、.NET Framework によって提供される Windows フォームを使用します。

Windows フォームおよび Windows フォーム コントロールには、Windows ベースのアプリケーションを開発するための、より豊富で拡張性のあるアーキテクチャが用意されています。ただし、Windows フォームとそのコントロールは、Visual Basic 6.0 のフォームとコントロールと多くの点で異なります。これらの相違点が説明されているトピックを次に示します。

このセクションの内容

フォームのタスク (Visual Basic 6.0 ユーザー向け)

フォームを使用した作業の一般的なタスクにおける違いについて説明します。

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Windows フォームでの、コントロールおよびオブジェクトの違いを説明するトピックへのリンクを提供します。

コントロールのプロパティ、メソッド、およびイベントの変更点 (Visual Basic 6.0 ユーザー向け)

フォームやコントロールなどのオブジェクトに関して、プロパティ、メソッド、およびイベントの違いを説明するトピックへのリンクを提供します。

コントロール配列 (Visual Basic 6.0 ユーザー向け)

コントロール配列について、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

フォームの配置 (Visual Basic 6.0 ユーザー向け)

フォームの画面位置の設定について、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

Menu オブジェクト (Visual Basic 6.0 ユーザー向け)

メニューの作成について、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

MDI (Visual Basic 6.0 ユーザー向け)

マルチ ドキュメント インターフェイス (MDI: Multiple-Document Interface) アプリケーションについて、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

ダイアログ ボックスの変更点 (Visual Basic 6.0 ユーザー向け)

ダイアログ ボックスの作成について、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

色の動作 (Visual Basic 6.0 ユーザー向け)

色の処理について、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

ヘルプ サポート (Visual Basic 6.0 ユーザー向け)

ヘルプの実装について、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

印刷の変更点 (Visual Basic 6.0 ユーザー向け)

アプリケーションの印刷サポートについて、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

関連項目

イベントおよびイベント処理 (Visual Basic 6.0 ユーザー向け)

イベント ハンドラの作成について、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)

ドラッグ アンド ドロップ操作の実装について、Visual Basic 6.0 と Visual Basic 2005 の違いを説明します。

グラフィックス (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 のグラフィックス メソッドの変更点について説明します。

Windows フォームの概要

フォームに関する基本概念を紹介します。

Windows フォームと Windows フォーム コントロールの新機能

Windows フォームの新機能について説明します。

Visual Studio の Windows アプリケーションの名前空間

Windows フォームに関連する名前空間を紹介します。

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic 6.0 と Visual Basic 2005 の違いをより詳細に説明するトピックへのリンクを示します。

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の組み込みコントロール、オブジェクト、およびコレクションは、Visual Basic 2005 の新しいコントロールおよびオブジェクトによって置き換えられています。名前が変更された場合もあり、多くの場合、プロパティ、メソッド、およびイベントが異なります。

このセクションの内容

次に挙げるトピックでは、Visual Basic 6.0 のコントロールおよびオブジェクトに相当する Visual Basic 2005 のコントロールおよびオブジェクトを示します。

RichTextBox コントロール (Visual Basic 6.0 ユーザー向け)

Screen オブジェクト (Visual Basic 6.0 ユーザー向け)

Shape コントロール (Visual Basic 6.0 ユーザー向け)

StatusBar コントロール (Visual Basic 6.0 ユーザー向け)

TextBox コントロール (Visual Basic 6.0 ユーザー向け)

ToolBar コントロール (Visual Basic 6.0 ユーザー向け)

Timer コントロール (Visual Basic 6.0 ユーザー向け)

TreeView コントロール (Visual Basic 6.0 ユーザー向け)

User コントロール (Visual Basic 6.0 ユーザー向け)

VBControlExtender オブジェクト (Visual Basic 6.0 ユーザー向け)

VScrollBar コントロール (Visual Basic 6.0 ユーザー向け)

WebBrowser コントロール (Visual Basic 6.0 ユーザー向け)

関連するセクション

Windows フォーム コントロール

Windows フォーム専用にデザインされたコントロールに関するトピックへのリンクを示します。

コントロールのプロパティ、メソッド、およびイベントの変更点 (Visual Basic 6.0 ユーザー向け)

フォームやコントロールなどのオブジェクトに関して、プロパティ、メソッド、およびイベントの違いを説明するトピックへのリンクを提供します。

各言語およびライブラリにおける、コントロールとプログラミング可能オブジェクトの比較

さまざまなコントロールおよびオブジェクトを比較します。

Windows フォームと Windows フォーム コントロールの新機能

Windows フォームの新機能について説明します。

App オブジェクト (Visual Basic 6.0 ユーザー向け)

ここでは、Visual Basic 6.0 の **App** オブジェクトと Visual Basic 2005 でそれらに相当するものを比較します。

Visual Basic 6.0 の **App** オブジェクトは、アプリケーションに関する情報を設定または取得するためのグローバル オブジェクトです。Visual Basic 2005 には、**App** オブジェクトに直接相当するものはありません。ただし、ほとんどのプロパティ、メソッド、およびイベントは、.NET Framework の対応するメンバで置き換えることができます。

概念の違い

Version Information プロパティ

App オブジェクトの **Version Information** プロパティは、Visual Basic 2005 では **Assembly** 属性で置き換えられます。**Version Information** プロパティは、[プロジェクトのプロパティ] ダイアログ ボックスで設定します。**Assembly** 属性は、[アセンブリ情報] ダイアログ ボックスで設定します。このダイアログ ボックスを表示するには、プロジェクト デザイナの [アプリケーション] タブの [アセンブリ情報] ボタンをクリックします。

メモ：
Version Information プロパティは、プロジェクト (.vbp) ファイルに保存されます。このファイルは、メモ帳などのテキスト エディタを使って編集できます。**Assembly** 属性は、AssemblyInfo.vb ファイルに保存され、コード エディタで編集できます。

File Description プロパティ

The Visual Basic 6.0 の **App** オブジェクトには、**FileDescription** プロパティと **Title** プロパティがあります。**FileDescription** は、Windows エクスプローラに表示される説明のテキストを指定します。

Visual Basic 2005 には、**FileDescription** 属性はありません。代わりに **Title** 属性が、Windows エクスプローラに表示される説明のテキストを指定します。

HelpFile プロパティ

Visual Basic 6.0 では、**HelpFile** プロパティがアプリケーション レベルでヘルプ ファイルを指定します。

Visual Basic 2005 では、ヘルプ ファイルは、フォームごとに HelpProvider コンポーネントを使って指定します。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。

Version プロパティ

Visual Basic 6.0 の **Major**、**Minor**、および **Revision** の各プロパティは、3 つの部分で構成されるアプリケーションのバージョン番号を取得するために使用します。

Visual Basic 2005 では、Build プロパティもバージョン情報に含まれます。4 つの部分で構成される完全なバージョン番号を取得するには、**Version** プロパティを使用します。詳細については、「バージョン番号 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Title プロパティ

Visual Basic 6.0 の **App** オブジェクトの **Title** プロパティは、プログラムで変更できます。そうすることにより、Windows のタスク マネージャに表示される名前を指定できます。

Visual Basic 2005 では、**Title** 属性は読み取り専用です。アプリケーションのメイン フォームの **Text** プロパティによって、タスク マネージャに表示される名前が決定されます。**Text** プロパティをプログラムで設定することはできますが、フォームのタイトル バーに表示されるテキストも変更されます。

TaskVisible プロパティ

Visual Basic 6.0 では、**App** オブジェクトの **TaskVisible** プロパティによって、Windows のタスク一覧 (Windows 9*x*) またはタスク マネージャの [アプリケーション] タブ (Windows 2000 以降) にアプリケーションを表示するかどうかが決定されます。一般に、このプロパティは、バックグラウンド タスクとして実行されるように設計したアプリケーションをユーザーが終了できないようにするために使用します。ほとんどの場合、**TaskVisible** プロパティは、ユーザー インターフェイスを表示しないアプリケーションで使用します。

Visual Basic 2005 には **TaskVisible** プロパティに相当するプロパティはありませんが、タスク一覧に表示されない Windows サービスまたはコンソール アプリケーションを作成することはできます。

PrevInstance プロパティ

Visual Basic 6.0 では、**PrevInstance** プロパティは、アプリケーションのインスタンスが既に実行中かどうかを表します。通常、アプリケーションの起動時に **PrevInstance** を呼び出し、**true** が返された場合にアプリケーションの実行を中止します。

Visual Basic 2005 には、**PrevInstance** オブジェクトに直接相当するものはありません。アプリケーションの複数のインスタンスが存在できるかどうかを制御するには、プロジェクト デザイナの [アプリケーション] タブの [複数インスタンス] プロパティを設定します。実行時には、ユーザーがアプ

リケーションの別のインスタンスを起動しようとすると、**NextInstanceStarted** イベントが発生します。このイベント ハンドラには、アプリケーションの最初のインスタンスをアクティブにするコードを追加できます。

他の違い

Visual Basic 6.0 には、ActiveX コンポーネント、OLE オートメーション、およびその他の技術に関連する多数の **App** オブジェクト プロパティが互換性のために残されていますが、これらは Visual Basic 2005 ではサポートされません。このようなプロパティは、このトピックの後の部分にある「App オブジェクト プロパティで対応するもの」という表に示します。

App オブジェクトを扱うコードの変更

次のコード例では、**App** オブジェクトの一般的な使い方における Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

アプリケーションのバージョン番号を表示するコードの変更

次のコード例は、アプリケーションのバージョン番号を取得し、ラベルに表示する方法を示しています。

```
' Visual Basic 6.0
Label1.Caption = "Version: " & App.Major & "." & App.Minor & "." _
& App.Revision
```

VB

```
' Visual Basic 2005
Label1.Text = My.Application.Info.Version.ToString()
```

アプリケーションのパスを確認するコードの変更

次のコード例は、実行中のアプリケーションのパスを取得することにより、アプリケーションのフォルダに格納されている画像ファイルを表示する方法を示しています。

```
' Visual Basic 6.0
Picture1.Picture = LoadPicture(App.Path & "\Logo.jpg")
```

VB

```
' Visual Basic 2005
PictureBox1.Image = System.Drawing.Bitmap.FromFile( _
  My.Application.Info.DirectoryPath & "\Logo.jpg")
```

アプリケーションの 2 番目のインスタンスが実行されないようにするコードの変更

次のコード例は、単一インスタンス アプリケーションの実装方法を示しています。

```
' Visual Basic 6.0
Private Sub Form_Load()
    If App.PrevInstance = True Then
        MsgBox("The application is already running!")
    End If
End Sub
```

VB

```
' Visual Basic 2005
' Assumes that the Make Single Instance Application checkbox in the
' Project Designer has been checked.

Private Sub MyApplication_StartupNextInstance(ByVal sender As Object, _
ByVal e As Microsoft.VisualBasic.ApplicationServices. _
StartupNextInstanceEventArgs) Handles Me.StartupNextInstance
    MsgBox("The application is already running!")
End Sub
```

App オブジェクト プロパティで対応するもの

次の表は、Visual Basic 6.0 の**App** オブジェクトのすべてのプロパティおよびメソッドと、Visual Basic 2005 でそれらに相当するものを示します。

Visual Basic 6.0	Visual Basic 2005 で対応するもの
Comments	My.Application.Info.Description プロパティ
CompanyName	My.Application.Info.CompanyName プロパティ
EXEName	My.Application.Info.AssemblyName プロパティ
FileDescription	My.Application.Info.Title プロパティ
HelpFile	新規に実装されました。**HelpProvider** コンポーネントを使用します。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HInstance	GetHINSTANCE
LegalCopyright	My.Application.Info.Copyright プロパティ
LegalTrademarks	My.Application.Info.Trademark プロパティ
LogEvent method	WriteEntry メソッド (My.Application.Log and My.Log)
LogMode **LogPath**	新規に実装されました。詳細については、「アプリケーションからの情報のログ記録」を参照してください。
Major	My.Application.Info.Version プロパティ **メモ：** バージョン番号の形式は、Visual Basic 2005 では異なります。詳細については、「バージョン番号 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Minor	My.Application.Info.Version プロパティ **メモ：** バージョン番号の形式は、Visual Basic 2005 では異なります。詳細については、「バージョン番号 (Visual Basic 6.0 ユーザー向け)」を参照してください。
NonModalAllowed	新規に実装されました。これは ActiveX の .dll ファイルに関連した読み取り専用プロパティです。共通言語ランタイムでは、この動作を自動的に管理します。

OleRequestPendingMsgText **OleRequestPendingMsgTitle** **OleRequestPendingTimeout** **OleServerBusyMsgText** **OleServerBusyMsgTitle** **OleServerBusyRaiseError** **OleServerBusyTimeout**	対応する項目はありません。これらのプロパティは、Visual Basic 2005 ではサポートされていない OLE オートメーションに関連するプロパティです。
Path	My.Application.Info.DirectoryPath プロパティ
PrevInstance	IsSingleInstance **メモ：** **IsSingleInstance** プロパティは、**Protected** プロパティです。これは、デザイン時にプロジェクト デザイナの [複数インスタンス] チェック ボックスをオンまたはオフにすることによってのみ設定できます。
ProductName	My.Application.Info.ProductName プロパティ
RetainedProject	新規に実装されました。Visual Basic 2005 では、プロジェクトをメモリに保持できません。
Revision	My.Application.Info.Version プロパティ **メモ：** バージョン番号の形式は、Visual Basic 2005 では異なります。詳細については、「バージョン番号 (Visual Basic 6.0 ユーザー向け)」を参照してください。
StartLogging method	新規に実装されました。詳細については、「アプリケーションからの情報のログ記録」を参照してください。
StartMode	対応する項目はありません。これは、ActiveX コンポーネントの作成に関連するプロパティです。ActiveX コンポーネントは、Visual Basic 2005 ではサポートされません。
TaskVisible	新規に実装されました。タスク マネージャに表示しないアプリケーションを作成するには、Windows サービスまたはコンソール アプリケーションのプロジェクトを作成します。
ThreadID	新規に実装されました。このプロパティは、実行中のスレッドの ID を返します。スレッド処理モデルは、Visual Basic 2005 では大きく異なります。
Title	My.Application.Info.Title プロパティ
UnattendedApp	新規に実装されました。Visual Basic 2005 の無人アプリケーションの場合は、コンソール アプリケーション プロジェクトを作成します。

アップグレード メモ

アプリケーションを Visual Basic 6.0 からアップグレードすると、**Major** プロパティと **Minor** プロパティだけがアップグレードされます。Visual Basic 2005 は、Revision プロパティと **Build** プロパティに新しい値を割り当てます。

参照

関連項目

My.Application オブジェクト

概念

ヘルプ サポート (Visual Basic 6.0 ユーザー向け)

バージョン番号 (Visual Basic 6.0 ユーザー向け)

CheckBox コントロール (Visual Basic 6.0 ユーザー向け)

ここでは、Visual Basic 6.0 の **CheckBox** コントロールと Visual Basic 2005 でそれらに相当するものを比較します。

Visual Basic 6.0 の **CheckBox** コントロールは、Visual Basic 2005 では Windows フォームの CheckBox コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Click イベント

Visual Basic 6.0 では、**CheckBox** の状態がプログラムで変更されると、**Click** イベントが生成されます。Visual Studio 2005 では、**Click** イベントの代わりに CheckStateChanged イベントを使用します。

他の違い

加えて、データ連結、フォント処理、ドラッグ アンド ドロップ、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォームの概要」および「Windows フォームと Windows フォーム コントロールの新機能」を参照してください。

CheckBox コントロールを扱うコードの変更

次のコード例は、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

CheckBox コントロールのクリックに応答するコードの変更

次のコード例は、ユーザーが **CheckBox** コントロールをオンまたはオフにしたときにコントロールのテキストを変更する方法を示しています。Visual Basic 6.0 の **Value** プロパティは、Visual Basic 2005 では Checked プロパティに置き換えられています。**Caption** プロパティは、Text プロパティで置き換えられています。Visual Basic 2005 のコード例では、Visual Basic 6.0 の **Click** イベントの動作を複製するために、**CheckStateChanged** イベントを使用しています。

```
' Visual Basic 6.0
Private Sub Check1_Click()
    If Check1.Value = 1 Then
        Check1.Caption = "Checked"
    Else
        Check1.Caption = "Unchecked"
    End If
End Sub
```

VB

```
' Visual Basic 2005
Private Sub CheckBox1_CheckStateChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles CheckBox1.CheckStateChanged
    If CheckBox1.Checked = True Then
        CheckBox1.Text = "Checked"
    Else
        CheckBox1.Text = "Unchecked"
    End If
End Sub
```

CheckBox の状態を確認するコードの変更

次のコード例は、実行時に **CheckBox** コントロールの状態をチェックする方法の例を示しています。

```
' Visual Basic 6.0
Select Case Check1.Value
    Case vbUnchecked
        Check1.Caption = "Unchecked"
    Case vbChecked
        Check1.Caption = "Checked"
    Case vbGrayed
        Check1.Caption = "Disabled"
End Select
```

VB

```
' Visual Basic 2005
Select Case CheckBox1.CheckState
    Case CheckState.Unchecked
        CheckBox1.Text = "Unchecked"
    Case CheckState.Checked
        CheckBox1.Text = "Checked"
    Case CheckState.Indeterminate
        CheckBox1.Text = "Disabled"
End Select
```

CheckBox コントロールのプロパティ、メソッド、およびイベントに対応するもの

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

必要に応じて、動作の違いを説明するトピックへのリンクが示されています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0 プロパティ	**Visual Basic 2005 で対応するもの**
Alignment **0** (左揃え) **1** (右揃え)	CheckAlign MiddleLeft 列挙値 MiddleRight 列挙値
Appearance **0** (フラット) **1** (3D)	FlatStyle 列挙値 Flat 列挙値 Standard 列挙値
BackColor	BackColor **メモ：** 色の定数の一覧については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。 **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Caption	**Text**
Container	Parent
DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	新規に実装されました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。

DisabledPicture **DownPicture**	新規に実装されました。詳細については、 「方法 : アップグレードしたアプリケーションで Visual Basic 6.0 の 3 ステート コントロールをエミュレートする」を参照してください。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ :** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、 「フォント処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ :** 色の定数の一覧については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。 **メモ :** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height , Size **メモ :** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ :** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
MaskColor	新規に実装されました。詳細については、「MaskColor (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。

MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
Picture	Image
RightToLeft **True**	RightToLeft Yes 列挙値 No 列挙値
Style **0** (標準) **1** (グラフィカル)	Appearance Normal 列挙値 Button 列挙値 **メモ :** Visual Basic 2005 では **Graphical** のスタイルの扱い方が異なります。詳細については、「方法 : アップグレードしたアプリケーションで Visual Basic 6.0 の 3 ステート コントロールをエミュレートする」を参照してください。
Tag	Tag
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ :** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
UseMaskColor	新規に実装されました。詳細については、「MaskColor (Visual Basic 6.0 ユーザー向け)」を参照してください。
Value **0** (**vbUnchecked**) **1** (**vbChecked**) **2** (**vbGrayed**)	CheckState Unchecked 列挙値 Checked 列挙値 Indeterminate 列挙値
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。

Width	Width **, Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0 メソッド	**Visual Basic 2005 で対応するもの**
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder **0** (**vbBringToFront**) **1** (**vbSendToBack**)	BringToFront, SendToBack

イベント

Visual Basic 6.0 イベント	**Visual Basic 2005 で対応するもの**
Click	**CheckStateChanged** **メモ：** Visual Basic 6.0 では、**CheckBox** の状態がプログラムで変更されると、**Click** イベントが生成されます。Visual Basic 2005 では、**Click** イベントが発生しないので、**CheckStateChanged** イベントを使用する必要があります。
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
LostFocus	Leave

OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Validate	Validating

アップグレード メモ

アップグレード ウィザードを使って Visual Basic 6.0 アプリケーションをアップグレードすると、**CheckBox** コントロールは Windows フォームの **CheckBox** コントロールにアップグレードされ、コード内のプロパティ、メソッド、およびイベントは、それぞれに相当するものにアップグレードされます。相当するものがなかったり、動作に潜在的な相違点がある部分のコードには、コメントとヘルプ トピックへのリンクが挿入されます。

参照

関連項目

CheckBox コントロールの概要 (Windows フォーム)

概念

データ アクセス (Visual Basic 6.0 ユーザー向け)
Style プロパティ (Visual Basic 6.0 ユーザー向け)
フォント処理 (Visual Basic 6.0 ユーザー向け)
座標系 (Visual Basic 6.0 ユーザー向け)
コントロール配列 (Visual Basic 6.0 ユーザー向け)
ヘルプ サポート (Visual Basic 6.0 ユーザー向け)
MousePointer (Visual Basic 6.0 ユーザー向け)
ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

Clipboard オブジェクト (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Clipboard** オブジェクトは、Visual Basic 2005 では、よく似ている **Clipboard** オブジェクトに置き換えられています。

概念の違い

Visual Basic 6.0 では、システム クリップボードとの間でテキスト、画像、およびデータの保存や取得を行うために **Clipboard** オブジェクトを使用します。

Visual Basic 2005 では、**My.Computer** 名前空間に **Clipboard** オブジェクトがあり、新しいメソッドが追加され、一部のメソッドの動作が多少変更されています。

クリップボードのデータ形式

Visual Basic 6.0 では、**Clipboard** との情報の受け渡しに使うデータ型が定数として定義されています。

Visual Basic 2005 では、このデータ型を定義するために DataFormats オブジェクトを使用します。新しいデータ型がいくつかサポートされています。Visual Basic 6.0 のクリップボード形式定数と、Visual Basic 2005 でそれらに相当するものについては、このトピックの後半にある一覧に示します。

Clipboard オブジェクトを扱うコードの変更

次のコード例は、**Clipboard** を使ってテキストを保存および取得する方法を示します。

```
' Visual Basic 6.0
Clipboard.Clear
Clipboard.SetText "Hello", vbCFText
If Clipboard.GetFormat(vbCFText) Then
   Text1.Text = Clipboard.GetText(vbCFText)
End If
```

VB

```
' Visual Basic 2005
My.Computer.Clipboard.Clear()
My.Computer.Clipboard.SetText("Hello")
If My.Computer.Clipboard.ContainsText Then
  TextBox1.Text = My.Computer.Clipboard.GetText
End If
```

Clipboard 定数の対応表

Visual Basic 6.0	Visual Basic 2005 での同等物
vbCFBitmap	Bitmap
vbCFDIB	Dib
vbCFEMetafile	EnhancedMetafile
vbCFFiles	FileDrop
vbCFLink	対応するキーワードなし詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
vbCFMetafile	MetafilePict
vbCFPalette	Palette

vbCFRTF	Rtf
vbCFText	Text

メモ：

DataFormats オブジェクトは、System.Windows.Forms 名前空間の一部です。**DataFormats** では、上に示したクリップボード形式以外にも、Visual Basic 6.0 でサポートされないいくつかの形式をサポートしています。

参照

処理手順

方法 : Visual Basic でクリップボードを消去する
方法 : Visual Basic でクリップボードから読み込む
方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する
方法 : Visual Basic でオーディオ ストリームをクリップボードに保存する
方法 : Visual Basic でクリップボードからイメージを取得する
方法 : Visual Basic でクリップボードに書き込む

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard オブジェクトのメンバ

ComboBox コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **ComboBox** コントロールは、Visual Basic 2005 では Windows フォームの ComboBox コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Change イベント

Visual Basic 6.0 では、**ComboBox** コントロールの **Change** イベントが生成されるのは、このコントロールのテキストが変更されたときです。**Change** イベントは、コントロールの一覧部分でアイテムが選択されたときには生成されません。また、アイテムのテキストをプログラムで変更した場合にも **Change** イベントは発生しません。

Visual Basic 2005 では、**Change** イベントはなくなりました。何らかの理由でテキストが変更されると、TextChanged イベントが生成されます。テキストが変更されるのは、次のような場合です。

- テキスト入力部分が変更された。
- 一覧でアイテムが選択された。
- 一覧のアイテムがプログラムで変更された。
- **AddItem** メソッドが呼び出された。

ItemData プロパティ

Visual Basic 6.0 では、デザイン時に [プロパティ] ウィンドウで **ComboBox** コントロールの **ItemData** プロパティを設定して、**ComboBox** のアイテムに **Integer** を関連付けることができます。

Visual Basic 2005 では、**ItemData** プロパティが存在しません。Microsoft.VisualBasic 互換性ライブラリには、**Integer** をアイテムに関連付けることができる **SetItemData** 関数が含まれています。アイテムを取得する場合は **GetItemData** 関数を使用できます。

Locked プロパティ

Visual Basic 6.0 では、**ComboBox** コントロールのテキスト ボックス部分が編集できるかどうかは、コントロールの **Locked** プロパティによって決定されます。

Visual Basic 2005 では、**Locked** プロパティは、デザイン時にコントロールの移動を禁止するために使用します。Visual Basic 6.0 の **Locked** プロパティに直接相当するものはありませんが、**ComboBox** コントロールの DropDownStyle プロパティを DropDownList に設定することによって、同じ結果が得られます。

メモ　Visual Basic 6.0 では、**Locked** プロパティを **True** に設定することでも、選択内容を変更できないように設定できます。これと同じような効果を得るには、`MouseDown` イベントの選択をキャンセルします。

NewIndex プロパティ

Visual Basic 6.0 では、**ComboBox** コントロールに追加された最も新しいアイテムのインデックスを取得するために、**NewIndex** プロパティを使用していました。

Visual Basic 2005 では、NewIndex プロパティが存在しません。**Item.Add** メソッドの戻り値を使うと、追加したアイテムのインデックスを取得できます。

TopIndex プロパティ

Visual Basic 6.0 では、**ComboBox** コントロールや **ListBox** コントロールでどのアイテムを一番上に表示するかを指定する値を **TopIndex** プロパティを使って取得または設定できます。このプロパティは一般に、アイテムを選択せずにリストをスクロールするために使用されます。

Visual Basic 2005 では、**ComboBox** コントロールで **TopIndex** プロパティはサポートされなくなりました。Style プロパティを 1 - SimpleCombo に設定した場合以外は、**TopIndex** プロパティを設定しても目に見える違いはないので、ほとんどの場合は問題ありません。Style プロパティを 1 - SimpleCombo に設定している場合は、ListBox コントロールと TextBox コントロールを使用することにより、この **ComboBox** の動作をエミュレートできます。**ListBox** コントロールは引き続き TopIndex プロパティをサポートしています。

Scroll イベント

Visual Basic 6.0 の **Scroll** イベントは、**TopIndex** プロパティと組み合わせて、リストのスクロール時にアクションを実行するために使用します。Visual Basic 2005 では、**Scroll** イベントはサポートされませんが、ほとんどの場合、SelectedIndexChanged イベントで適切に置き換えることができます。

その他の違い

また、データ連結、フォント処理、ドラッグ アンド ドロップ操作、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあ

ります。詳細については、「Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)」を参照してください。

ComboBox コントロールを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

ComboBox コントロールを読み取り専用にするコードの変更

次のコード例は、ユーザーが **ComboBox** コントロールに新しいアイテムを追加できないようにする方法を示します。

```
' Visual Basic 6.0
Combo1.Locked = True
```

VB

```
' Visual Basic 2005
ComboBox1.DropDownStyle = ComboBoxStyle.DropDownList
```

ComboBox コントロールに追加された最も新しいアイテムを選択するコードの変更

次のコード例は、**ComboBox** コントロールにプログラムで追加されたアイテムを選択状態にする方法を示します。

```
' Visual Basic 6.0
Combo1.AddItem "This is a new item"
Combo1.ListIndex = Combo1.NewIndex
```

VB

```
' Visual Basic 2005
Dim i As Integer
i = ComboBox1.Items.Add("This is a new item")
ComboBox1.SelectedIndex = i
```

ItemData を関連付ける Private Sub コードの変更

次のコード例は、**ComboBox** コントロールの一覧に含まれる各従業員に従業員番号を関連付け、その番号を実行時に取得する方法を示します。

```
' Visual Basic 6.0
Private Sub Form_Load
    Combo1.AddItem "Nancy Davolio"
    Combo1.ItemData(Combo1.NewIndex) = 12345
    Combo1.AddItem "Judy Phelps"
    Combo1.ItemData(Combo1.NewIndex) = 67890
End Sub
Private Sub Combo1_Click()
    Label1.Caption = "Empoyee #" & _
    CStr(Combo1.ItemData(Combo1.ListIndex))
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load

   ComboBox1.Items.Add("Nancy Davolio")
   Microsoft.VisualBasic.Compatibility.VB6. _
      SetItemData(ComboBox1, ComboBox1.Items.Count() - 1, 12345)

   ComboBox1.Items.Add("Judy Phelps")
   Microsoft.VisualBasic.Compatibility.VB6. _
      SetItemData(ComboBox1, ComboBox1.Items.Count() - 1, 67890)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged

   Label1.Text = "Employee #" & CStr( _
   Microsoft.VisualBasic.Compatibility.VB6. _
      GetItemData(ComboBox1, ComboBox1.SelectedIndex))
End Sub
```

ComboBox コントロールのプロパティ、メソッド、およびイベントで対応するもの

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表には、動作の違いを説明するトピックへのリンクも含まれています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Appearance	新規に実装されました。詳細については、「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Container	Parent
DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	新規に実装されました。詳細については、「Data コントロール (Visual Basic 6.0 ユーザー向け)」を参照してください。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ItemData	新規に実装されました。詳細については、「ItemData プロパティをアップグレードできない」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
List	Items
ListCount	Count
ListIndex	SelectedIndex
Locked	**DropDownStyle** = **DropDownList** **メモ：** Visual Basic 6.0 では、**Locked** プロパティを **True** に設定することでも、選択内容を変更できないように設定できます。Visual Basic 2005 で、これと同じような効果を得るには、MouseDown イベントで選択をキャンセルします。

MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
NewIndex	新規に実装されました。詳細については、「NewIndex プロパティをアップグレードできない」を参照してください。
OLEDragMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
RightToLeft	RightToLeft
SelLength	SelectionLength
SelStart	SelectionStart
SelText	SelectedText
Style	**DropDownStyle**
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
TopIndex	新規に実装されました。詳細については、「TopIndex プロパティおよび Scroll イベント (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	**Visual Basic 2005 での同等物**

AddItem	Add AddRange Insert
Clear	Clear
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け).
RemoveItem	**Items.Remove**
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder	BringToFront SendToBack 関数

イベント

Visual Basic 6.0	**Visual Basic 2005 で対応するもの**
Change	**TextChanged** **メモ：** **TextChanged** イベントの動作は少し変更されています。詳細については、「ComboBox コントロールの Change イベント (Visual Basic 6.0 ユーザー向け)」を参照してください。
Click	**SelectedIndexChanged**
DblClick	新規に実装されました。**SelectedIndexChanged** イベントまたは **TextChanged** イベントを使用します。
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
LostFocus	Leave

OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Scroll	新規に実装されました。詳細については、「TopIndex プロパティおよび Scroll イベント (Visual Basic 6.0 ユーザー向け)」を参照してください。
Validate	Validating

アップグレード メモ

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、**ComboBox** コントロールの **Change** イベントは Visual Basic 2005 の **ComboBox** コントロールの **TextChanged** イベントに対応付けられます。**TextChanged** イベントの動作は、**Change** イベントとは異なります。この相異によって、コードで意図した結果が得られない場合があります。

ItemData、**Locked**、**NewIndex**、または **TopIndex** プロパティを参照するコード、または **Scroll** イベント プロシージャ内のコードは、アップグレードされません。このようなコードには警告のコメントが挿入されるので、アプリケーションをコンパイルする前にコードを削除するか変更する必要があります。

参照

関連項目

ComboBox コントロールの概要 (Windows フォーム)

概念

色の動作 (Visual Basic 6.0 ユーザー向け)
フォント オブジェクト (Visual Basic 6.0 ユーザー向け)
MousePointer (Visual Basic 6.0 ユーザー向け)
ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)

CommandButton コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **CommandButton** コントロールは、Visual Basic 2005 では Windows フォームの Button コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Default プロパティおよび Cancel プロパティ

Visual Basic 6.0 では、**CommandButton** コントロールのブール型の **Default** プロパティによって Enter キーへの応答が決定され、**Cancel** プロパティによって Esc キーへの応答が決定されます。

Visual Basic 2005 では、**Button** コントロールの **Default** プロパティおよび **Cancel** プロパティはなくなりました。それらと同じ機能は、Form オブジェクトの AcceptButton プロパティおよび CancelButton プロパティで提供されます。このプロパティは、**Button** コントロールの名前を引数として受け取ります。

Value プロパティ

Visual Basic 6.0 では、**CommandButton** コントロールのブール型の **Value** プロパティは、コントロールが選択されているかどうかを示します。このプロパティを **True** に設定すると、ボタンの **Click** イベントが呼び出されます。

Visual Basic 2005 では、**Button** コントロールの **Value** プロパティはなくなりました。コントロールが選択されているかどうかは、GotFocus イベントを使って確認できます。Click イベントを呼び出すには、PerformClick メソッドを使用できます。

Style、Picture、DownPicture、および DisabledPicture プロパティ

Visual Basic 6.0 では、**CommandButton** コントロールの Style プロパティを 1 – Graphical に設定すると、コントロールの外観がイメージを表示するように変更されます。**Picture**、**DownPicture**、および **DisabledPicture** の各プロパティを使用して、状態の変化に応じて表示するイメージを割り当てることができます。たとえば、**CommandButton** コントロールがクリックされると **DownPicture** イメージが表示され、**CommandButton** コントロールがオフになると **DisabledPicture** イメージが表示されます。

Visual Basic 2005 では、**Style**、**Picture**、**DownPicture**、または **DisabledPicture** の各プロパティはサポートされません。**Style** プロパティは不要になりました。画像を Image プロパティに割り当てると、**Style** プロパティを **Graphical** に設定したのと同じ効果が得られます。**Picture** プロパティは、**Image** プロパティに置き換えられています。**DownPicture** プロパティと **DisabledPicture** プロパティと同じような機能は、ImageList コントロールに複数の画像を含めることで実現できます。

MaskColor プロパティ

Visual Basic 6.0 では、**CommandButton** コントロールの **MaskColor** プロパティを使用することにより、背景イメージを透かして表示することのできる透過色を定義します。このプロパティを使うには、Style プロパティが Graphical に設定され、**UseMaskColor** プロパティが **True** に設定され、ビットマップが **Picture** プロパティに割り当てられている必要があります。

Visual Basic 2005 には、**MaskColor** プロパティに直接相当するものはありません。ただし、Bitmap オブジェクトの MakeTransparent メソッドを使うと、コントロールを透明に設定できます。

その他の違い

加えて、データ連結、フォント処理、ドラッグ アンド ドロップ、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)」を参照してください。

CommandButton コントロールを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

既定のボタンとキャンセル ボタンを設定するコードの変更

次のコード例は、2 つのボタンをフォームの既定のボタンとキャンセル ボタンとして設定する方法を示しています。

```
' Visual Basic 6.0
' Set the first button to respond to the Enter key.
Command1.Default = True
' Set the second button to respond to the Esc key.
Command2.Cancel = True
```

VB

```
' Visual Basic 2005
' Set the first button to respond to the Enter key.
Me.AcceptButton = Button1
```

```
' Set the second button to respond to the Esc key.
Me.CancelButton = Button2
```

ボタンに透明の領域を追加するコードの変更

次のコード例は、画像のあるボタンに透明の領域を定義する方法を示しています。画像の白い部分が透明になります。

```
' Visual Basic 6.0
' Assumes a picture has been assigned to the Picture property
' and that the Style property has been set to Graphical.
Command1.UseMaskColor = True
Command1.MaskColor = vbWhite
```

VB

```
' Visual Basic 2005
' Assumes a picture has been assigned to the BackgroundImage property.
Dim ButtonBitmap As New System.Drawing.Bitmap(Button1.BackgroundImage)
ButtonBitmap.MakeTransparent(System.Drawing.Color.White)
Button1.BackgroundImage = ButtonBitmap
```

CommandButton コントロールのプロパティ、メソッド、およびイベントで対応するもの

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

これらの表には、動作の違いを説明するトピックへのリンクも含まれています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Cancel	**CancelButton** (**Form** オブジェクト)
Caption	Text
Container	Parent
Default	**AcceptButton** (**Form** オブジェクト)
DisabledPicture **DownPicture**	新規に実装されました。詳細については、「Style プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。

DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
MaskColor	新規に実装されました。詳細については、「MaskColor (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
Picture	**Image**
RightToLeft	RightToLeft
Style	新規に実装されました。詳細については、「Style プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。

ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
UseMaskColor	新規に実装されました。詳細については、「MaskColor (Visual Basic 6.0 ユーザー向け)」を参照してください。
Value	新規に実装されました。**PerformClick** メソッドは、**Value** を **True** に設定することに相当します。その他の **Value** プロパティの使い方には、対応するものはありません。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder	BringToFront 関数または SendToBack 関数

イベント

Visual Basic 6.0	**Visual Basic 2005 での同等物**
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter

LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

アップグレード メモ

アップグレード ウィザードを使って Visual Basic 6.0 アプリケーションをアップグレードすると、**CommandButton** コントロールは Windows フォームの **Button** コントロールにアップグレードされ、コード内のプロパティ、メソッド、およびイベントは、それぞれに相当するものにアップグレードされます。相当するものがなかったり、動作に潜在的な相違点がある部分のコードには、コメントとヘルプ トピックへのリンクが挿入されます。

アプリケーションを Visual Basic 2005 にアップグレードするときに、**Style** プロパティが 1 – **Graphical** に設定されている場合、アップグレードされたコントロールの **FlatStyle** プロパティは **Standard** に設定され、デザイン時に **Picture** プロパティに割り当てた画像は、アップグレードされたコントロールの `Image` プロパティに割り当てられます。

デザイン時または実行時に **DownPicture** プロパティまたは **DisabledPicture** プロパティを設定していた場合は、アップグレードしたアプリケーションを変更して **ImageList** コントロールを使う必要があります。詳細については、
「方法 : アップグレードしたアプリケーションで Visual Basic 6.0 の 3 ステート コントロールをエミュレートする」を参照してください。

アップグレード ウィザードは、**MaskColor** プロパティを使用するコードをアップグレードしません。アップグレードに関する警告がコードに挿入されます。アプリケーションを実行する前にコードを変更する必要があります。

参照

概念

MaskColor (Visual Basic 6.0 ユーザー向け)

CommonDialog コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **CommonDialog** コントロールは、Visual Basic 2005 では特別ないくつかのコンポーネントに置き換えられています。

概念の違い

Visual Basic 6.0 では、**CommonDialog** ActiveX コントロールを使って、さまざまなコモン ダイアログ ([開く]、[上書き保存]、[カラー]、[フォント]、[印刷]、および [ヘルプ]) をアプリケーションに表示します。

Visual Basic 2005 では、**CommonDialog** コントロールは、ダイアログ表示用の個別のコンポーネント (OpenFileDialog、SaveFileDialog、ColorDialog、FontDialog、および PrintDialog) に置き換えられています。

メモ :

Visual Basic 2005 には、ヘルプ ダイアログ表示に直接相当するものはありません。**CommonDialog** コントロールは、Windows ヘルプだけをサポートしました。Visual Basic 2005 は、HTML ヘルプだけをサポートします。Visual Basic 2005 では、HelpProvider コンポーネントを使ってアプリケーションのヘルプを表示します。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。

CommonDialog コントロールを扱うコードの変更

次のコード例では、**CommonDialog** コントロールの一般的な使い方における Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

[ファイルを開く] ダイアログ ボックスを表示するコードの変更

次のコード例は、[Program Files] ディレクトリに初期化された [ファイルを開く] ダイアログ ボックスを表示する方法を示しています。

```
' Visual Basic 6.0
' Uses a CommonDialog control.
CommonDialog1.InitDir = "C:\Program Files"
CommonDialog1.ShowOpen
```

VB

```
' Visual Basic 2005
' Uses a OpenFileDialog component.
OpenFileDialog1.InitialDirectory = "C:\Program Files"
OpenFileDialog1.ShowDialog()
```

[名前を付けて保存] ダイアログ ボックスを表示するコードの変更

次のコード例は、[名前を付けて保存] ダイアログ ボックスを表示し、ファイルをアプリケーションのフォルダに保存する方法を示しています。

```
' Visual Basic 6.0
' Uses a CommonDialog control.
CommonDialog1.InitDir = App.Path
CommonDialog1.ShowSave
```

VB

```
' Visual Basic 2005
' Uses a SaveFileDialog component.
SaveFileDialog1.InitialDirectory = My.Application.Info.DirectoryPath
SaveFileDialog1.ShowDialog()
```

[印刷] ダイアログ ボックスを表示するコードの変更

次のコード例は、[印刷] ダイアログ ボックスを表示し、アプリケーションのフォルダ内のファイルを印刷する方法を示しています。

```
' Visual Basic 6.0
' Uses a CommonDialog control.
```

```
CommonDialog1.FileName = App.Path & "MyFile.txt"
CommonDialog1.ShowPrinter
```

VB

```
' Visual Basic 2005
' Uses PrintDocument and PrintDialog components.
PrintDocument1.DocumentName = My.Application.Info.DirectoryPath _
& "MyFile.txt"
PrintDialog1.Document = PrintDocument1
PrintDialog1.ShowDialog()
```

ヘルプを表示するコードの変更

次のコード例は、アプリケーションのヘルプ ファイルを表示し、目次を開く方法を示しています。

```
' Visual Basic 6.0
' Uses a CommonDialog control.
CommonDialog1.HelpFile = "C:\Windows\Help\calc.hlp"
CommonDialog1.HelpCommand = cdlHelpContents
CommonDialog1.ShowHelp
```

VB

```
' Visual Basic 2005
' Uses the Help.ShowHelp method.
Help.ShowHelp(Me, "file://C:\Windows\Help\calc.chm", _
HelpNavigator.TableOfContents)
```

CommonDialog コントロールのプロパティとメソッドの対応関係

Visual Basic 6.0 のプロパティおよびメソッドと、それらに相当する Visual Basic 2005 のプロパティおよびメソッドを次の表に示します。同じ名前で同じ動作を持つプロパティおよびメソッドは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

必要に応じて、動作の違いを説明するトピックへのリンクが示されています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 で対応するもの
Action	新規に実装されました。Visual Basic 6.0 では、表示するダイアログは **Action** プロパティによって決定されます。Visual Basic 2005 では、ダイアログごとに異なるコンポーネントを使用します。
CancelError	Cancel
Copies	Copies
DialogTitle	Title (**OpenFileDialog** および **SaveFileDialog** コンポーネントのみ) ほかのコンポーネントのために新規に実装されました。Windows の標準タイトル (**Color**、**Font**、および **Print**) が表示され、これらをオーバーライドすることはできません。
FileName	FileNames
FileTitle	新規に実装されました。Visual Basic 6.0 の **FileTitle** プロパティは、**FileName** をパスなしで返します。**FileNames** プロパティを解析することにより、パスなしの名前を取得できます。

Flags	Visual Basic 6.0 の **Flags** プロパティには、各種のコモン ダイアログにさまざまな属性を設定するための定数が用意されています。ダイアログ コンポーネントでは、定数ではなくプロパティを使って属性を設定します。
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
FromPage	FromPage
hDC	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpCommand	HelpNavigator
HelpFile	HelpNamespace
HelpKey	ShowHelp メソッドの *parameter* パラメータ。
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
InitDir	InitialDirectory
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Max	MaxSize (**FontDialog** コンポーネント) MaximumPage (**PrintDialog** コンポーネント)
Min	MinSize (**FontDialog** コンポーネント) MinimumPage (**PrintDialog** コンポーネント)
MaxFileSize	新規に実装されました。この Visual Basic 6.0 のプロパティは、非常に長いファイル名にメモリを割り当てます。マネージ コードでは、この機能は不要です。
Orientation	Landscape
Parent	FindForm メソッド
PrinterDefault	新規に実装されました。この Visual Basic 6.0 のプロパティは、**hDC** プロパティと組み合わせて、グラフィック デバイス インターフェイス メソッドを使って印刷を行うために使用します。この機能は、サポートされなくなりました。

Top	**P:System.Windows.Forms.Control.Top** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ToPage	ToPage

メソッド

Visual Basic 6.0	Visual Basic 2005 で同等のもの
AboutBox	新規に実装されました。**AboutBox** プロパティは、**CommonDialog** コントロールの [バージョン情報] ボックスを表示しました。これは、Microsoft 向けにサード パーティによって作成されました。
ShowColor	ShowDialog (**ColorDialog** コンポーネント)
ShowFont	**ShowDialog** (**FontDialog** コンポーネント)
ShowHelp	ShowHelp
ShowOpen	**ShowDialog** (**OpenFileDialog** コンポーネント)
ShowPrinter	**ShowDialog** (**PrintDialog** コンポーネント)
ShowSave	**ShowDialog** (**SaveFileDialog** コンポーネント)

アップグレード メモ

アプリケーションを Visual Basic 6.0 から Visual Basic 2005 にアップグレードすると、**CommonDialog** コントロールは、対応するダイアログ コンポーネント (**OpenFileDialog**、**SaveFileDialog**、**ColorDialog**、**FontDialog**、または **PrintDialog**) コントロールにアップグレードされます。

Visual Basic 2005 には、**Help** ダイアログの表示に直接相当するものはありません。**CommonDialog** コントロールが **Help** ダイアログとして使用されている場合、そのコントロールはアップグレードされず、警告が発行されます。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。

1 つの **CommonDialog** コントロールが、複数の種類のダイアログを表示するために使用されている場合があります。この場合、コントロールは **OpenFileDialog** コンポーネントにアップグレードされ、警告が発行されます。それ以外のダイアログ コンポーネントを手動で追加する必要があります。

また、**CommonDialog** コントロールが変数として使用されている場合は、アップグレード後に、特定のダイアログ コンポーネントを参照するように変更する必要があります。

参照

関連項目

FontDialog コンポーネントの概要 (Windows フォーム)
HelpProvider コンポーネントの概要 (Windows フォーム)

その他の技術情報

ColorDialog コンポーネント (Windows フォーム)
OpenFileDialog コンポーネント (Windows フォーム)
SaveFileDialog コンポーネント (Windows フォーム)
PrintDialog コンポーネント (Windows フォーム)
PrintDocument コンポーネント (Windows フォーム)

Controls コレクション (Visual Basic 6.0 ユーザー向け)

The Visual Basic 6.0 の **Controls** コレクションは、Visual Basic 2005 では ControlCollection クラスに置き換えられています。

概念の違い

Visual Basic 6.0 の **Controls** コレクションは、フォームまたはコンテナ コントロール上のコントロールを表す要素を持つコレクションです。

Visual Basic 2005 では、**Controls** コレクションは **ControlCollection** クラスに置き換えられています。フォームには、`Me.Controls` という構文でアクセスできる既定の **ControlCollection** クラスがあります。

Add メソッド

Visual Basic 6.0 では、**Controls** コレクションの **Add** メソッドは遅延バインディングです。**Control** クラスを引数として渡すことにより、**Add** メソッドでコントロールが作成されます。

Visual Basic 2005 では、**ControlCollection** クラスの Add メソッドを使ってコレクションに追加するコントロールは、**New** キーワードを使って作成されている必要があります。

Remove メソッド

Visual Basic 6.0 の **Controls** コレクションの **Remove** メソッドは、**Add** メソッドで追加されたコントロールに対してしか使用できませんでした。Visual Basic 2005 の **ControlCollection** クラスにはこのような制限はありません。

Timer コントロールおよび Menu コントロール

Visual Basic 6.0 では、**Timer** コントロールと **Menu** コントロールは **Controls** コレクションのメンバです。Visual Basic 2005 では、これらのコントロールは Timer および MainMenu または ContextMenu コンポーネントで置き換えられています。コンポーネントは、**ControlCollection** クラスのメンバではありません。

コンテナ内のコントロール

Visual Basic 6.0 の **Controls** コレクションには、コンテナ コントロールの子であるコントロール (**Frame** コントロール上にあるコントロールなど) が含まれます。Visual Basic 2005 の **ControlCollection** クラスには、これらは含まれません。フォーム上のすべてのコントロールを反復処理するには、各コンテナ コントロールの **Controls** クラスを再帰的に反復処理する必要があります。

Controls コレクションを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

コントロールの追加や削除を行うコードの変更

次のコード例は、Visual Basic 6.0 の **Controls** コレクションと Visual Basic 2005 の **ControlCollection** クラスの相違点を示しています。

```
' Visual Basic 6.0
Private Sub Command1_Click()
    ' Declare a new Control variable.
    Dim c As Control
    ' Create and add the new control.
    Set c = Controls.Add("VB.TextBox", "Text1")
    ' Make the new control visible.
    c.Visible = True
    ' Set the initial text.
    c.Text = "Hello"
    ' Retrieve the text from the new TextBox.
    If Controls.Count > 1 Then
        MsgBox (Controls("Text1").Text)
    End If
    ' Remove the new control.
    Controls.Remove (Text1)
    ' The following line causes a compilation error.
    ' You cannot remove controls added at design time.
    Controls.Remove (Command1)
End Sub
```

VB

```
' Visual Basic 2005
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handl
es Button1.Click
    ' Create a new TextBox control.
    Dim TextBox1 As New System.Windows.Forms.TextBox
    TextBox1.Name = "TextBox1"
    ' Add the new control to the form's Controls collection.
    Me.Controls.Add(TextBox1)
    ' No need to set Visible property.
    ' Set the initial text.
    TextBox1.Text = "Hello"
    ' Retrieve the text from the new TextBox.
    If Me.Controls.Count > 1 Then
        MsgBox(Me.Controls("TextBox1").Text)
    End If
    ' Remove the new control.
    Me.Controls.Remove(TextBox1)
    ' Remove the control added at design time.
    Me.Controls.Remove(Button1)
End Sub
```

Controls コレクションを反復処理するコードの変更

次のコード例は、フォーム上のすべてのコントロールを反復処理してからすべての CheckBox コントロールをオフにする関数を示しています。この例では、**CheckBox** コントロールがフォームではなく GroupBox コントロールまたは Panel コントロールの上にあると仮定します。Visual Basic 2005 のコード例では、フォームの Controls コレクションに含まれるのはフォーム上に直接配置されたコントロールだけなので、関数は子のあるコントロールに対して自分自身を再帰的に呼び出します。

```
' Visual Basic 6.0
Private Sub ClearChecks()
    For Each Control in Me.Controls
        If TypeOf Control Is CheckBox Then
            Control.Value = vbUnchecked
        End If
    Next
End Sub
```

VB

```
' Visual Basic 2005
Private Sub ClearChecks(ByVal Container As Control)
    Dim ctl As Control
    Dim chk As CheckBox
    For Each ctl In Container.Controls
        If TypeOf ctl Is CheckBox Then
            chk = ctl
            chk.Checked = False
        End If
        ' Recursively call this function for any container controls.
        If ctl.HasChildren Then
            ClearChecks(ctl)
        End If
    Next
End Sub
```

アップグレード メモ

Visual Basic 6.0 と Visual Basic 2005 では **Controls** コレクションに相違点があるので、**Add** メソッドの呼び出しはアップグレードされません。新しい **Add** メソッドを使用して、アプリケーションの動作を再作成するコードを追加する必要があります。

参照

関連項目

Control.Controls Property
Form.ControlCollection Class

Data コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Data** コントロールは、DAO を使用するデータベースにコントロールをバインドする機構として使用します。Visual Basic 2005 には、**Data** コントロールに相当するものはありません。データ バインディング アーキテクチャが変更され、DAO はサポートされなくなりました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。

Data コントロールは、ボタンを使ってデータベース テーブル内の行を前後に移動できるインターフェイスも提供します。Visual Basic 2005 でこれに相当する BindingNavigator コントロールには、行の追加や削除に使用できるボタンもあります。

参照

関連項目

BindingNavigator コントロールの概要 (Windows フォーム)

概念

アップグレードを行うにあたっての注意事項

データ アクセス (Visual Basic 6.0 ユーザー向け)

DataGrid コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、Visual Basic 6.0 の **DataGrid** コントロールの代わりに、Windows フォームの DataGridView コントロールが用意されています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念上の相違点

データ連結

Visual Basic 2005 の **DataGridView** コントロールは、すべての動作がデータ ソース経由で行われるため、データ固有のメソッドやイベントを必要としません。このようにデータを表示形式から分離することで、ユーザー インターフェイスの入力の有無にかかわらず、データ ソースを変更できます。また、同じデータ ソースに連結された複数のコントロールは、常に同期しています。

移動

DataGrid コントロール内での表示や移動のためのプロパティ (**TabAction**、**EnterAction**、**AllowArrows**、**WrapCellPointer**、**Scrollable** など) は不要になりました。たとえば、グリッドは **Scrollable** プロパティが **True** に設定されているときと同様に機能し、表示できる以上のデータが存在する場合は、スクロール バーが自動的に表示されます。既定では、グリッド内の移動は Excel と同様の形式になります。Tab キーを押すと前方へ移動し、Shift キーを押しながら Tab キーを押すと後方へ移動できます。詳細については、「Windows フォーム DataGridView コントロールの既定のキーボード処理とマウス処理」を参照してください。

Caption プロパティ

Visual Basic 6.0 では、**Caption** プロパティを使用して、グリッドの上にタイトル バーを表示します。**Caption** プロパティを設定しない場合は、タイトル バーが表示されません。

Visual Basic 2005 の **DataGridView** コントロールは、タイトル バーをサポートしていません。ただし、Label コントロールを使用して同じ効果を実現できます。

データの書式設定

Visual Basic 6.0 では、**DataGrid** コントロール内のデータの書式設定は、**DataFormat** プロパティと **StdDataFormat** オブジェクトを使用して行います。書式設定は、列単位に適用されます。

Visual Basic 2005 の **DataGridView** コントロールでは、DataGridViewCellStyle オブジェクトの Format プロパティを使用して、書式設定を行います。書式設定は、セル単位、列単位、または行単位で適用できます。詳細については、「Windows フォーム DataGridView コントロールでのデータの書式設定」を参照してください。

hWndEditor プロパティ

Visual Basic 6.0 では、**hWndEditor** プロパティを使用して、**DataGrid** コントロールの編集ウィンドウに割り当てられたウィンドウ ハンドルを、Windows API の呼び出しに渡します。

Visual Basic 2005 の **DataGridView** コントロールには、編集モードのときに個別のウィンドウ ハンドルが割り当てられません。**DataGridView** コントロールの Handle プロパティを使用するか、埋め込みのエディット コントロールを使用します。

MarqueeStyle プロパティ

Visual Basic 6.0 では、**MarqueeStyle** プロパティを使用して、選択したセルまたは行の境界線スタイルを変更したり、前景色と背景色を反転したり、編集ウィンドウを起動したりすることにより、その外観を制御します。

Visual Basic 2005 には、**DataGridView** コントロールに直接相当するものはありません。しかし、SelectionMode、CellBorderStyle、および **Format** の各プロパティを組み合わせることによって同じ効果を実現できます。詳細については、「Windows フォーム DataGridView コントロールの選択モード」を参照してください。

SelLength プロパティ、SelStart プロパティ、SelText プロパティ

Visual Basic 6.0 の **DataGrid** コントロールでは、セルが編集モードになったときに、**SelLength**、**SelStart**、および **SelText** の各プロパティを使用して、キャレット (^) の最初の位置を設定したり、セル内でテキストの一部を強調表示したりできます。

Visual Basic 2005 の **DataGridView** コントロールでは、これらのプロパティがなくなりました。**DataGridView** コントロール内のセルは、TextBox コントロールを基準としています。EditingControlShowing イベント ハンドラにコードを追加することにより、基底のコントロールの SelectionLength、SelectionStart、および SelectedText の各プロパティにアクセスできます。

分割ビュー

Visual Basic 6.0 の **DataGrid** コントロールは、分割ビューをサポートするため、同じデータを並べて表示できます。**Split** オブジェクト、**Split** プロパティ、**Splits** プロパティ、および **TabAcrossSplits** プロパティによって、分割ビューを表示できるかどうかを制御しています。

Visual Basic 2005 の **DataGridView** コントロールでは、これらのプロパティがなくなりました。ただし、1 つ以上の SplitContainer コントロール

と複数の **DataGridView** コントロールを使用して、同じ効果を実現できます。**TabAcrossSplits** プロパティの機能を複製するには、**DataGridView** コントロールの StandardTab プロパティを使用します。

DataGrid コントロールに関するコードの変更点

次のコードでは、**DataGridView** コントロール内のセルが選択されたときに、セル内のテキストを強調表示する方法をそれぞれ示すことによって、Visual Basic 6.0 と Visual Basic 2005 の相違を示しています。

```
' Visual Basic 6.0
Private Sub DataGrid1_Click()
    DataGrid1.SelStart = 1
    DataGrid1.SelLength = DataGrid1.Text
    MsgBox("The selected text is " & DataGrid1.SelText)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub DataGridView1_EditingControlShowing( _
ByVal sender As Object, ByVal e As System.Windows.Forms. _
DataGridViewEditingControlShowingEventArgs) _
Handles DataGridView1.EditingControlShowing
    CType(e.Control, TextBox).SelectionStart = 0
    CType(e.Control, TextBox).SelectionLength = Len(CType(e.Control, _
      TextBox).Text)
    MsgBox("The selected text is " & CType(e.Control, _
      TextBox).SelectedText)
End Sub
```

DataGrid コントロールのプロパティ、メソッド、およびイベントの対応

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に対応付けられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。直接 Visual Basic 2005 に対応するメンバがない場合は、代わりの説明を示すトピックへのリンクがあります。

プロパティ

Visual Basic 6.0	**対応する Visual Basic 2005 のイベント**
AddNewMode	新規に実装されました。データの操作はデータ ソースで処理されます。詳細については、「Windows フォーム DataGridView コントロールでのデータの表示」を参照してください。
Align	Dock および Anchor
AllowAddNew	AllowNew (**BindingSource**)
AllowArrows	調整可能なプロパティではなくなりました。方向キーによる移動は常に可能です。詳細については、「Windows フォーム DataGridView コントロールの既定のキーボード処理とマウス処理」を参照してください。
AllowDelete	AllowRemove (**BindingSource**)
AllowRowSizing	AllowUserToResizeRows
AllowUpdate	AllowEdit (**BindingSource**)

Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
ApproxCount	新規に実装されました。データの操作はデータ ソースで処理されます。詳細については、 「Windows フォーム DataGridView コントロールでのデータの表示」を参照してください。
BackColor	BackgroundColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Bookmark	新規に実装されました。すべての項目に直接アクセスできます。
Caption	新規に実装されました。タイトル バーをシミュレートするには、**Label** コントロールを使用します。
Col	SelectedColumns
ColumnHeaders	ColumnHeadersVisible
Container	Parent。**Control** からの継承メンバです。
CurrentCellModified	IsCurrentCellDirty, IsCurrentCellInEditMode
CurrentCellVisible	CurrentCell **メモ：** 画面上に見えていないセルに **CurrentCell** プロパティが設定されている場合は、グリッドがスクロールしてそのセルを表示します (**CurrentCellVisible** を **True** に設定した場合と同じ)。
DataChanged	**IsCurrentCellDirty**, IsCurrentRowDirty
DataFormats	**DataGridViewCellStyle** オブジェクト。 **メモ：** 書式はセル単位、列単位、または行単位で設定できます。
DefColWidth	Width **DataGridViewColumn** オブジェクト。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
EditActive	**IsCurrentCellInEditMode**
FirstRow	FirstDisplayedScrollingRowIndex

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「Color 動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HeadFont	**DataGridViewCellStyle** オブジェクト。 **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
HeadLines	直接対応する項目はありません。**DataGridViewCellStyle** オブジェクトの WrapMode プロパティを ColumnHeadersHeight と組み合わせて使用します。
Height	Height。**Control** クラスからの継承メンバです。 **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
hWnd	**Handle**
hWndEditor	新規に実装されました。**Handle** を使用します。
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left。**Control** クラスからの継承メンバです。 **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
LeftCol	FirstDisplayedScrollingColumnIndex

MarqueeStyle	直接対応する項目はありません。**SelectionMode**、**CellBorderStyle**、および **Format** の各プロパティを使用します。
RecordSelectors	RowHeadersVisible
Row	SelectedRows
RowDividerStyle	GridColor、**CellBorderStyle**、RowHeadersBorderStyle、ColumnHeadersBorderStyle の各プロパティ。
RowHeight	Height
SelBookmarks	新規に実装されました。すべての項目に直接アクセスできます。
SelEndCol **SelStartCol**	SelectedCells, **SelectedColumns**
SelLength	直接対応する項目はありません。**SelectionLength** プロパティを **EditingControlShowing** イベント ハンドラで使用します。
SelStart	直接対応する項目はありません。**SelectionStart** プロパティを **EditingControlShowing** イベント ハンドラで使用します。
SelText	直接対応する項目はありません。**SelectedText** プロパティを **EditingControlShowing** イベント ハンドラで使用します。
Split **Splits** **TabAcrossSplits**	新規に実装されました。分割ビューは直接サポートされていません。**SplitContainer** コントロールを使用します。
TabAction	**StandardTab**
Tag	新規に実装されました。
Text	**CurrentCell.Value** **メモ：** **Value** プロパティは **Object** を返します。これを **String** に変換するには、**CStr** または **ToString** を使用します。
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

VisibleCols	**DisplayedColumnCount** メソッド。 **メモ：** Visual Basic 6.0 では、部分的に表示されている列は常にカウントに含まれます。**DisplayedColumnCount** メソッドは、ブール値の **includePartialColumns** パラメータを使用して、部分的に表示されている列を含めるかどうかを決定します。
VisibleRows	**DisplayedRowCount** メソッド。 **メモ：** Visual Basic 6.0 では、部分的に表示されている行は常にカウントに含まれます。**DisplayedRowCount** メソッドは、ブール値の **includePartialRows** パラメータを使用して、部分的に表示されている行を含めるかどうかを決定します。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width。**Control** クラスからの継承メンバです。 **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WrapCellPointer	調整可能なプロパティではなくなりました。既定の動作は **WrapCellPointer** = **True** です。

メソッド

Visual Basic 6.0	**対応する Visual Basic 2005 のイベント**
CaptureImage	新規に実装されました。**DataGridView** コントロールの内容を **PictureBox** コントロールに取り込む機能はサポートされていません。
ClearFields	新規に実装されました。再バインディング時には列の書式設定が自動で行われます。
ClearSelCols	ClearSelection
ColContaining	IndexOf (**DataGridViewColumnCollection**)
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GetBookmark	新規に実装されました。ブックマークはサポート対象から除外されました。
HoldFields	新規に実装されました。再バインディング時には列の書式設定が保存されます。
Move	SetBounds。**Control** クラスからの継承メンバです。 **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Rebind	ResetBindings。**Control** クラスからの継承メンバです。

RowBookmark	新規に実装されました。ブックマークはサポート対象から除外されました。
RowContaining	IndexOf (**DataGridViewColumnCollection**)
RowTop	GetContentBounds (**DataGridViewCell**)
Scroll	新規に実装されたメソッド。**CurrentCell** プロパティを使用します。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
SplitContaining	新規に実装されました。分割ビューは直接サポートされていません。**SplitContainer** コントロールを使用します。
ZOrder	**BringToFront()** 関数または **SendToBack()** 関数

イベント

Visual Basic 6.0	**対応する Visual Basic 2005 のイベント**
AfterColEdit	CellEndEdit
AfterColUpdate **AfterDelete**	新規に実装されました。データの操作はデータ ソースで処理されます。詳細については、 「Windows フォーム DataGridView コントロールでのデータの表示」を参照してください。
AfterUpdate	RowsAdded
BeforeColEdit	CellBeginEdit
BeforeColUpdate **BeforeDelete** **BeforeInsert** **BeforeUpdate**	新規に実装されました。データの操作はデータ ソースで処理されます。詳細については、 「Windows フォーム DataGridView コントロールでのデータの表示」を参照してください。
ButtonClick	Click (**Button** コントロール)
Change	TextChanged **メモ：** **TextChanged** イベントの動作は少し変更されています。詳細については、 「ComboBox コントロールの Change イベント (Visual Basic 6.0 ユーザー向け)」を参照してください。
Click	SelectedIndexChanged

ColEdit	**CellBeginEdit**
ColResize	ColumnWidthChanged
DblClick	CellMouseDoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Error	DataError
HeadClick	ColumnHeaderMouseClick
MouseDown	CellMouseDown
MouseMove	CellMouseMove
MouseUp	CellMouseUp
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
OnAddNew	**RowsAdded**
RowColChange	CurrentCellChanged
RowResize	RowHeightChanged
SelChange	SelectionChanged
SplitChange	新規に実装されました。分割ビューは直接サポートされていません。**SplitContainer** コントロールを使用します。
Validate	Validating

アップグレード メモ

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、**DataGrid** コントロールの **Change** イベントが、Visual Basic 2005 の **DataGridView** コントロールの **TextChanged** イベントにマップされます。**TextChanged** イベントの動作は、**Change** イベントの動作とは異なります。この相異によって、コードで意図した結果が得られない場合があります。

データ固有のメソッドに関連するコードや、データ固有のイベントに関連するコードはアップグレードされません。警告のコメントがコードに追加されます。そのコードは、アプリケーションをコンパイルする前に削除または変更する必要があります。

参照

関連項目

DataGridView コントロールの概要 (Windows フォーム)

各言語およびライブラリにおける、コントロールとプログラミング可能オブジェクトの比較

概念

DataGridView コントロール テクノロジの概要 (Windows フォーム)

Windows フォームと Windows フォーム コントロールの新機能

その他の技術情報

DataGridView コントロール (Windows フォーム)

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

Windows フォームでのデータ バインディング

DirListBox コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **DirListBox** コントロールは、Visual Basic 2005 では OpenFileDialog コンポーネントおよび SaveFileDialog コンポーネントに置き換えられています。

概念の違い

Visual Basic 6.0 の **DirListBox** コントロールは、通常、ファイルを開いたり保存したりするダイアログ ボックスで、ディレクトリやパスを表示するために使用されていました。

Visual Basic 2005 では、Windows フォーム **OpenFileDialog** コンポーネントと **SaveFileDialog** コンポーネントを使って、ファイルを扱う標準の Windows ダイアログ ボックスを作成できます。ほとんどの場合、**DirListBox** コントロールを使う必要はありません。

メモ :

OpenFileDialog コンポーネントと **SaveFileDialog** コンポーネントを使用して、一貫性のある使い慣れた操作方法をユーザーに提供することをお勧めします。独自のファイル ダイアログ ボックスを作成する場合、Visual Basic 2005 では、Microsoft Visual Basic 互換性ランタイム ライブラリに含まれている **DirListBox** コントロールを使用します。詳細については、
「方法 : Visual Basic 6.0 ファイル システムのコントロールをアプリケーションに追加する」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、既存の **DirListBox** コントロールは互換性ライブラリ (**Microsoft.VisualBasic.Compatibility**) に含まれる **VB6.DirListBox** コントロールにアップグレードされます。

参照

処理手順

方法 : Visual Basic 6.0 ファイル システムのコントロールをアプリケーションに追加する

関連項目

OpenFileDialog コンポーネントの概要 (Windows フォーム)
SaveFileDialog コンポーネントの概要 (Windows フォーム)

DriveListBox コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **DriveListBox** コントロールは、Visual Basic 2005 では OpenFileDialog コンポーネントおよび SaveFileDialog コンポーネントに置き換えられています。

概念の違い

Visual Basic 6.0 の **DriveListBox** コントロールは、通常、[ファイルを開く] ダイアログ ボックスまたは [名前を付けて保存] ダイアログ ボックスにドライブの一覧を表示するために使用されていました。

Visual Basic 2005 では、Windows フォーム **OpenFileDialog** コンポーネントと **SaveFileDialog** コンポーネントを使って、ファイルを扱う標準の Windows ダイアログ ボックスを作成できます。ほとんどの場合、**DriveListBox** コントロールを使う必要はありません。

メモ :

OpenFileDialog コンポーネントと **SaveFileDialog** コンポーネントを使うと、一貫性のある使い慣れた操作方法がユーザーに提供されます。独自のファイル ダイアログ ボックスを作成する必要がある場合、Visual Basic 2005 では、Microsoft Visual Basic 互換性ランタイム ライブラリに含まれている **DriveListBox** コントロールを使用します。詳細については、
「方法 : Visual Basic 6.0 ファイル システムのコントロールをアプリケーションに追加する」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、既存の **DriveListBox** コントロールは **Microsoft.VisualBasic.Compatibility** ライブラリに含まれる **VB6.DriveListBox** コントロールにアップグレードされます。

参照

処理手順

方法 : Visual Basic 6.0 ファイル システムのコントロールをアプリケーションに追加する

関連項目

OpenFileDialog コンポーネントの概要 (Windows フォーム)
SaveFileDialog コンポーネントの概要 (Windows フォーム)

FileListBox コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **FileListBox** コントロールは、Visual Basic 2005 では OpenFileDialog コンポーネントおよび SaveFileDialog コンポーネントに置き換えられています。

概念の違い

Visual Basic 6.0 の **FileListBox** コントロールは、通常、[ファイルを開く] ダイアログ ボックスまたは [名前を付けて保存] ダイアログ ボックスでファイルを選択するために使用されていました。

Visual Basic 2005 では、Windows フォーム **OpenFileDialog** コンポーネントと **SaveFileDialog** コンポーネントを使って、ファイルを扱う標準の Windows ダイアログ ボックスを作成できます。ほとんどの場合、**FileListBox** コントロールを使う必要はありません。

メモ :

OpenFileDialog コンポーネントと **SaveFileDialog** コンポーネントを使用して、一貫性のある使い慣れた操作方法をユーザーに提供することをお勧めします。独自のファイル ダイアログ ボックスを作成する場合、Visual Basic 2005 では、Microsoft Visual Basic 互換性ランタイム ライブラリに含まれている **FileListBox** コントロールを使用します。詳細については、
「方法 : Visual Basic 6.0 ファイル システムのコントロールをアプリケーションに追加する」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、既存の **FileListBox** コントロールは **VB6.FileListBox** コントロールにアップグレードされます。このコントロールは、Microsoft.VisualBasic.Compatibility ライブラリに含まれています。

参照

処理手順

方法 : Visual Basic 6.0 ファイル システムのコントロールをアプリケーションに追加する

関連項目

OpenFileDialog コンポーネントの概要 (Windows フォーム)
SaveFileDialog コンポーネントの概要 (Windows フォーム)

フォント オブジェクト (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では **Font** プロパティを設定する方法が 2 つありますが、Visual Basic 2005 では Font クラスだけを使用します。

概念の違い

Visual Basic 6.0 の **Font** プロパティは、プロパティを直接設定するのではなく、**stdFont** オブジェクトを使用して設定できました。Visual Basic 6.0 の **stdFont** オブジェクトは、Visual Basic 2005 では **Font** クラスに置き換えられています。直接対応するプロパティおよびメソッドはありません。詳細については、「フォント処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。

参照

概念

フォント処理 (Visual Basic 6.0 ユーザー向け)

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Form** オブジェクトは、Visual Basic 2005 では Form クラスに置き換えられました。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Activate イベントおよび Deactivate イベント

Visual Basic 6.0 では、**Activate** イベントと **Deactivate** イベントは、フォーム間の切り替えがあったときにだけ発生します。

Visual Basic 2005 では、Activated イベントと Deactivate イベントは、フォームと他のアプリケーション間の切り替えがあったときにも発生します。

QueryUnload イベント

Visual Basic 6.0 では、**QueryUnload** イベントは、*Cancel* および *UnloadMode* という 2 つの引数を受け取ります。*UnloadMode* 引数を調べることにより、フォームがアンロードされる理由を確認し、必要に応じてアンロードをキャンセルできます。

Visual Basic 2005 では、**QueryUnload** イベントは、FormClosing イベントに置き換えられています。*UnloadMode* は、CloseReason に置き換えられています。

Picture プロパティ

Visual Basic 6.0 では、**Picture** プロパティに割り当てた画像は、ビットマップがフォームより小さい場合にフォームの左上隅に表示されます。

Visual Basic 2005 では、**Picture** プロパティは BackgroundImage プロパティに置き換えられています。**BackgroundImage** プロパティに割り当てられた画像がフォームより小さい場合、既定で画像は並べて表示されます。

Moveable プロパティ

Visual Basic 6.0 では、フォームの **Moveable** プロパティを **False** に設定すると、ユーザーが実行時にフォームを移動できなくなります。Visual Basic 2005 の Windows フォームには、これに相当するプロパティはありません。

これと同じような動作を Visual Basic 2005 で実現するには、FormBorderStyle プロパティを **None** に設定し、ControlBox プロパティを **False** に設定します。ただし、このようなユーザー インターフェイスのデザインは一般に推奨されません。

ValidateControls メソッド

Visual Basic 6.0 では、フォームを閉じるときにフォーカスがあるコントロールに **Validate** イベントを発生させるために **ValidateControls** メソッドを使用します。**Validate** イベントが失敗すると、エラーが発生します。

Visual Basic 2005 では、**ValidateControls** メソッドは、Validate メソッドに置き換えられています。このメソッドは、**True** または **False** を返します。

MDI フォームのマウス イベント

Visual Basic 6.0 では、MDI フォームでマウス イベントがサポートされています。Visual Basic 2005 では、MDI フォーム上にマウス イベントを受け取るクライアント領域がないため、MDI フォームでは **Click**、**MouseDown**、**MouseMove**、**MouseUp** の各イベントはサポートされません。

フォーム オブジェクトを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

フォームを閉じる理由を確認するコードの変更

次のコード例は、フォームを閉じる理由と、理由に応じて処理を実行する方法を示します。Visual Basic 6.0 のコード例では、**QueryUnload** イベントの *UnloadMode* 引数を使用します。Visual Basic 2005 のコード例では、この引数は **FormClosing** イベント ハンドラの **CloseReason** パラメータに置き換えられています。

```
' Visual Basic 6.0
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    If UnloadMode = vbFormControlMenu Then
        Cancel = True
    End If
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_FormClosing(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles _
MyBase.FormClosing
    If e.CloseReason.UserClosing Then
        e.Cancel = True
    End If
End Sub
```

フォームを閉じるときにコントロールを検証するコードの変更

次のコード例では、フォームを閉じるときにフォーカスがあるコントロールを検証する方法を示しています。この例では、**TextBox** コントロールの **CausesValidation** プロパティが既定値の **True** に設定されていると仮定します。

```
' Visual Basic 6.0
Private Sub Text1_Validate(Cancel As Boolean)
    If Text1.Text = "" Then
        MsgBox ("Please enter a name")
        Cancel = True
End If

Private Sub Form_Unload(Cancel As Integer)
    OnError GoTo ERR_HANDLER
    Me.ValidateControls

ERR_HANDLER:
    ' If validation failed cancel the Unload event.
    If Err.Number = 380 Then
        Cancel = True
    End If
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TextBox1_Validating(ByVal sender As Object, ByVal e As _
System.ComponentModel.CancelEventArgs) Handles TextBox1.Validating
    If TextBox1.Text = "" Then
        MsgBox("Please enter a name")
        e.Cancel = True
    End If
End Sub
```

VB

```
Private Sub Form1_FormClosing(ByVal sender As System.Object, ByVal e _
As System.Windows.Forms.FormClosingEventArgs) Handles _
MyBase.FormClosing
    ' If validation failed cancel the Closing event.
    If Me.Validate = False Then
        e.Cancel = True
    End If
End Sub
```

フォーム オブジェクトのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、それらに対応する Visual Basic 2005 のプロパティを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

必要に応じて、動作の違いを説明するトピックへのリンクが示されています。Visual Basic 2005 に直接対応するものがない場合は、代わりの

項目を示すトピックへのリンクを示します。

Form のプロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
AutoRedraw	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
BorderStyle	**FormBorderStyle**
Caption	Text
ClipControls	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Controls	Controls **メモ：** Visual Basic 2005 では、**Controls** コレクションに相違点があります。詳細については、 「Controls コレクション (Visual Basic 6.0 ユーザー向け)」を参照してください。
Count	Count **メモ：** Visual Basic 2005 では、**Controls** コレクションに相違点があります。詳細については、 「Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)」を参照してください。
CurrentX **CurrentY**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DrawMode **DrawStyle** **DrawWidth**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
FillColor **FillStyle**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HasDC	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
HDC	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Image	新規に実装されました。Visual basic 6.0 の **Image** プロパティはビットマップへのハンドルを返しましたが、Visual Basic 2005 では画像にハンドルはありません。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
LinkMode **LinkTopic**	対応する項目はありません。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
MaxButton	MaximizeBox

MDIChild	MdiParent **メモ：** Visual Basic 2005 では、MDI の動作は異なります。詳細については、「MDI (Visual Basic 6.0 ユーザー向け)」を参照してください。
MinButton	MinimizeBox
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
Moveable	新規に実装されました。詳細については、「Moveable プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Name	Name
NegotiateMenus	新規に実装されました。詳細については、「Menu オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Palette **PaletteMode**	新規に実装されました。詳細については、「パレット (Visual Basic 6.0 ユーザー向け)」を参照してください。
Picture	**BackgroundImage** **メモ：** Visual Basic 6.0 では、**Picture** はフォームの左上隅に表示されましたが、Visual Basic 2005 では、**BackgroundImage** は並べて表示されます。
RightToLeft: **True** **False**	RightToLeft Yes 列挙値
ScaleHeight **ScaleLeft** **ScaleMode** **ScaleTop** **ScaleWidth**	新規に実装されました。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

StartUpPosition: **0 – Manual** **1 – CenterOwner** **2 – CenterScreen** **3 – Windows Default**	StartPosition Manual 列挙値 CenterParent 列挙値 CenterScreen 列挙値 WindowsDefaultLocation 列挙値
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisButton	HelpButton **メモ：** Visual Basic 2005 では、ヘルプの動作に相違点があります。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelp	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Form のメソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Circle	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Cls	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Line	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
PaintPicture	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Point	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。

PopupMenu	新規に実装されました。詳細については、「Menu オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
PrintForm	新規に実装されました。詳細については、「印刷の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Pset	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Scale **ScaleX** **ScaleY**	新規に実装されました。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Activate
Show	Show または、次のようにも指定できます。ShowDialog
TextHeight	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
TextWidth	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
ValidateControls	Validate **メモ：** **Validate** メソッドは、**True** または **False** を返します。検証が失敗した場合、**ValidateControls** はエラーを発生します。
WhatsThisMode	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	BringToFront 関数または SendToBack 関数 **BringToFront** **SendToBack**

Form のイベント

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Activate	**Activated** **メモ：** Visual Basic 6.0 では、**Activate** イベントは、アプリケーション内でフォーム間の切り替えがあったときにのみ発生します。Visual Basic 2005 では、**Activated** イベントは、他のアプリケーションとの間で切り替えがあったときにも発生します。
Click	Click **メモ：** **Click** イベントは MDI フォームでサポートされていません。
DblClick	DoubleClick

Deactivate	**Deactivate** **メモ：** Visual Basic 6.0 では、**Deactivate** イベントは、アプリケーション内でフォーム間の切り替えがあったときにのみ発生します。Visual Basic 2005 では、他のアプリケーションへの切り替えがあったときにも発生します。
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
Initialize	**New** メソッド **メモ：** **New** メソッドの動作は **Initialize** イベントの動作とは異なります。詳細については、「フォームのタスク (Visual Basic 6.0 ユーザー向け)」を参照してください。
LinkClose **LinkError** **LinkExecute** **LinkOpen**	対応する項目はありません。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseDown	MouseDown **メモ：** **MouseDown** イベントは MDI フォームでサポートされていません。
MouseMove	MouseMove **メモ：** **MouseMove** イベントは MDI フォームでサポートされていません。
MouseUp	MouseUp **メモ：** **MouseUp** イベントは MDI フォームでサポートされていません。

OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
QueryUnload	**FormClosing** **メモ：** Visual Basic 6.0 では、**QueryUnload** は、*Cancel* および *UnloadMode* という 2 つの引数を受け取ります。Visual Basic 2005 では、*Cancel* は、Cancel に置き換えられています。*UnloadMode* は、**CloseReason** に置き換えられています。
Terminate	Dispose メソッド **メモ：** **Dispose** メソッドの動作は **Terminate** イベントの動作とは異なります。詳細については、「フォームのイベント (Visual Basic 6.0 ユーザー向け)」を参照してください。
Unload	**FormClosing** **メモ：** **FormClosing** イベントの動作は **Unload** イベントの動作とは異なります。詳細については、「フォームのイベント (Visual Basic 6.0 ユーザー向け)」を参照してください。

参照

概念

フォームのタスク (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォーム

Forms コレクション (Visual Basic 6.0 ユーザー向け)

The Visual Basic 6.0 の **Forms** コレクションは、Visual Basic 2005 では My.Application.OpenForms プロパティ プロパティに置き換えられました。

概念の違い

Visual Basic 6.0 の **Forms** コレクションは、アプリケーション内に読み込まれた各フォームを表す要素で構成されるコレクションです。このコレクションには、アプリケーションの MDI フォーム、MDI 子フォーム、および MDI 以外のフォームが含まれます。**Forms** コレクションの唯一のプロパティ **Count** は、コレクション内の要素の数を表します。

Visual Basic 2005 の OpenForms プロパティは、アプリケーションで開かれているすべてのフォームを含む FormCollection オブジェクトを返します。これは **Forms** コレクションと動作は同じなので、同じ方法で使用できます。

Forms コレクションを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

開かれているフォームの数を取得するコードの変更

次のコード例は、アプリケーションで現在開かれているフォームの数を返す方法を示します。

```
' Visual Basic 6.0
MsgBox Forms.Count
```

VB

```
' Visual Basic 2005
MsgBox(CStr(My.Application.OpenForms.Count))
```

開かれているすべてのフォームのプロパティを設定するコードの変更

次のコード例は、アプリケーションで現在開かれている各フォームのタイトルを変更する方法を示します。

```
' Visual Basic 6.0
For Each Form in Forms
    Forms(i).Caption = "Hello"
Next
```

VB

```
' Visual Basic 2005
For Each f As Form In My.Application.OpenForms
    f.Text = "Hello"
Next
```

参照

処理手順

方法 : アプリケーションで開いているすべてのフォームにアクセスする

関連項目

My.Application.OpenForms プロパティ

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

Frame コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Frame** コントロールは、Visual Basic 2005 では GroupBox コントロールおよび Panel コントロールに置き換えられています。

概念の違い

Visual Basic 6.0 では、複数のコントロールをまとめるコンテナとして **Frame** コントロールが使用されます。Visual Basic 2005 では、**Frame** コントロールは、**GroupBox** コントロールまたは **Panel** コントロールに置き換えられています。

GroupBox コントロールは、BorderStyle プロパティが 1 – Fixed Single の **Frame** コントロールに相当します。このコントロールは輪郭が画面に表示され、オプションでキャプションも表示できます。

Panel コントロールは、BorderStyle プロパティが 0 – None の **Frame** コントロールに相当します。このコントロールには輪郭もキャプションも表示されません。

加えて、データ連結、フォント処理、ドラッグ アンド ドロップ、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)」を参照してください。

Frame コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

必要に応じて、動作の違いを説明するトピックへのリンクが示されています。Visual Basic 2005 に対応する項目がない場合は、代わりの項目を示すトピックへのリンクがあります。

Frame プロパティ

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Appearance	FlatStyle (**GroupBox** コントロールのみ)
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
BorderStyle	BorderStyle (**GroupBox** コントロールのみ)
Caption	Text (**GroupBox** コントロールのみ) **メモ：** **Panel** コントロールには、**Text** プロパティはありません。
ClipControls	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Container	Parent
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
RightToLeft	RightToLeft
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。

Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Frame メソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder	BringToFront 関数または SendToBack 関数

Frame のイベント

Visual Basic 6.0	Visual Basic 2005 での同等物
Click	Click (**Panel** のみ。**GroupBox** に相当するものはありません)
DblClick	DoubleClick (**Panel** のみ。**GroupBox** に相当するものはありません)

DragDrop **DragOver** **OLECompleteDrag** **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、BorderStyle プロパティが 0 - None に設定されている **Frame** コントロールは **Panel** コントロールにアップグレードされます。それ以外のすべての **Frame** コントロールは、**GroupBox** コントロールにアップグレードされます。

参照

関連項目

GroupBox コントロールの概要 (Windows フォーム)

概念

Panel コントロールの概要

HScrollBar コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **HScrollBar** コントロールは、Visual Basic 2005 では Windows フォームの HScrollBar コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Change イベント

Visual Basic 6.0 では、**HScrollBar** コントロールの **Value** プロパティが変更されると、**Change** イベントが生成されます。

Visual Basic 2005 では、**Change** イベントの代わりに ValueChanged イベントが生成されます。

Value プロパティ

Visual Basic 6.0 では、**HScrollBar** コントロールの **Scroll** イベントまたは **Change** イベントが発生すると、現在のスクロール値が **Value** プロパティに代入されます。

Visual Basic 2005 では、Scroll イベントまたは **ValueChanged** イベントが発生しても、その時点ではまだコントロールの Value プロパティの値は更新されません。Visual Basic 6.0 の動作をエミュレートする必要がある場合は、値を取得するヘルパ関数を作成できます。詳細については、「コードがイベントからプロシージャに変更されている」を参照してください。

LargeChange プロパティ

Visual Basic 6.0 では、スクロール ボックスとスクロール矢印の間にある領域がクリックされたときにスクロール バー コントロールの **Value** プロパティの値をどの程度変更するかは、**LargeChange** プロパティの値によって決定されます。

Visual Basic 2005 では LargeChange プロパティの既定値は 10 ですが、Visual Basic 6.0 では 1 です。

Max プロパティ

Visual Basic 6.0 では、**Max** プロパティによって、スクロール バー コントロールの最大 **Value** プロパティ設定が決定されます。

Visual Basic 2005 では、**Max** プロパティは Maximum プロパティに置き換えられています。既定値は 100 ですが、Visual Basic 6.0 では 32767 です。

Min プロパティ

Visual Basic 6.0 では、**Min** プロパティによって、スクロール バー コントロールの最小 **Value** プロパティ設定が決定されます。**Min** プロパティは、**Max** プロパティより大きな値に設定できます。

Visual Basic 2005 では、**Min** プロパティは Minimum プロパティに置き換えられました。このプロパティの値は、常に **Maximum** プロパティより小さい必要があります。

その他の違い

また、データ連結、フォント処理、ドラッグ アンド ドロップ機能、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。

HScrollBar コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

必要な場合には、動作の違いを説明するトピックへのリンクを示します。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

HScrollBar のプロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Container	Parent
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
LargeChange	LargeChange **メモ：** Visual Basic 6.0 での既定値は 1 でしたが、Visual Basic 2005 での既定値は 10 です。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Max	Maximum **メモ：** Visual Basic 6.0 での既定値は 32767 でしたが、Visual Basic 2005 での既定値は 100 です。
Min	Minimum **メモ：** Visual Basic 6.0 では、**Min** は **Max** より大きな値に設定できますが、Visual Basic 2005 では、**Minimum** プロパティは **Maximum** より大きな値に設定できません。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
RightToLeft: **True** **False**	RightToLeft Yes 列挙値 No 列挙値

Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Value	**Value** **メモ：** Visual Basic 6.0 では、**Value** が変更されると、**Change** イベントが生成されます。Visual Basic 2005 では、**Change** イベントの代わりに **ValueChanged** イベントが使用されます。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

HScrollBar のメソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	BringToFront 関数または SendToBack 関数 **BringToFront** **SendToBack**

HScrollBar のイベント

Visual Basic 6.0	Visual Basic 2005 での同等物
Change	**ValueChanged** **メモ：** **Change** イベントと **Scroll** イベントの動作は、Visual Basic 2005 では変更されています。詳細については、「コードがイベントからプロシージャに変更されている」を参照してください。

DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
LostFocus	Leave
Scroll	**Scroll** **メモ：** **Change** イベントと **Scroll** イベントの動作は、Visual Basic 2005 では変更されています。詳細については、「コードがイベントからプロシージャに変更されている」を参照してください。
Validate	Validating

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**HScrollBar** コントロールまたは **VScrollBar** コントロールの **Change** イベント ハンドラまたは **Scroll** イベント ハンドラのコードは、イベントの順序の違いに従ってプロシージャに変更されます。詳細については、「コードがイベントからプロシージャに変更されている」を参照してください。

参照

関連項目

HScrollBar コントロールと VScrollBar コントロールの概要 (Windows フォーム)

Image コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Image** コントロールに直接対応するものは、Visual Basic 2005 にありません。**Image** コントロールは、ウィンドウレス コントロールです。Visual Basic 2005 は、ウィンドウレス コントロールをサポートしません。

Stretch プロパティのアップグレード

Image コントロールのプロパティ、メソッド、およびイベントは、**Stretch** プロパティを除き、**PictureBox** コントロールのプロパティ、メソッド、およびイベントのサブセットです。詳細については、「PictureBox コントロール (Visual Basic 6.0 ユーザー向け)」を参照してください。

次の表は、**Stretch** プロパティと、Visual Basic 2005 の PictureBox コントロールでそれらに対応するものを示しています。

Stretch プロパティの同等物

Visual Basic 6.0	Visual Basic 2005 での同等物
Stretch = **False**	PictureBoxSizeMode = Normal
Stretch = **True**	**PictureBoxSizeMode** = StretchImage

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**Image** コントロールは **PictureBox** コントロールにアップグレードされます。

参照

関連項目

PictureBox コントロールの概要 (Windows フォーム)

概念

PictureBox コントロール (Visual Basic 6.0 ユーザー向け)

ImageList コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **ImageList** コントロールは、Visual Basic 2005 では Windows フォームの ImageList コンポーネントに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

BackColor

Visual Basic 6.0 では、**ImageList** コントロールに **BackColor** プロパティがありますが、コントロールは実行時に表示されないため、このプロパティは無効です。

Visual Basic 2005 では、新しい **ImageList** コントロールに **BackColor** プロパティはありません。

Left プロパティと Top プロパティ

Visual Basic 6.0 では、**ImageList** コントロールには、フォーム上でのコントロールの位置を設定する **Left** プロパティと **Top** プロパティがあります。

Visual Basic 2005 では、**ImageList** コンポーネントは、フォームではなくコンポーネント トレイにあるため、**Left** プロパティや **Top** プロパティは必要ありません。

UseMaskColor プロパティ

Visual Basic 6.0 では、**UseMaskColor** プロパティは、**MaskColor** プロパティに割り当てられている色をマスクとして使用するかどうかを判断します。

Visual Basic 2005 では、**ImageList** コンポーネントには **UseMaskColor** プロパティはなく、**MaskColor** プロパティは **TransparentColor** プロパティに置き換えられています。**TransparentColor** プロパティで定義される色をマスクとして使用しないためには、このプロパティを **Color.Transparent** に設定します。

Overlay メソッド

Visual Basic 6.0 では、**ImageList** コントロールの **Overlay** メソッドを使用すると、2 つの **ListImage** オブジェクトから 3 番目の複合イメージを作成できます。たとえば、あるイメージに対して、2 番目のイメージ部分は透き通って表示するマスクを定義できます。

Visual Basic 2005 では、**ImageList** コンポーネントで **Overlay** メソッドはサポートされなくなりました。同じ効果を実現するには、グラフィックス プログラムを使用して複合イメージを作成し、そのイメージを **ImageList.** に追加する必要があります。

ImageList コントロールを扱うコードの変更

次のコード例は、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

MaskColor の設定を扱うコードの変更

次のコードは、マスクとして使用する色を指定する方法と、そのオンとオフを切り替える方法を示しています。

```
' Visual Basic 6.0
' Specify the color to be used as a mask.
ImageList1.MaskColor = vbWhite
' Use the mask.
ImageList1.UseMaskColor = True
' Don't use the mask.
ImageList1.UseMaskColor = False
```

VB

```
' Visual Basic 2005
' Specify the color to be used as a mask and use the mask.
ImageList1.TransparentColor = Color.White
' Don't use the mask.
ImageList1.TransparentColor = Color.Transparent
```

ImageList コントロールのプロパティとメソッドの対応関係

次の表は、Visual Basic 6.0 のプロパティおよびメソッドと、対応する Visual Basic 2005 のプロパティおよびメソッドの一覧を示しています。同じ名前で同じ動作を持つプロパティおよびメソッドは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙

型は System.Windows.Forms 名前空間に割り当てられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
BackColor	新規に実装されました。**ImageList** はコンポーネントです。
hImageList	Handle
ImageHeight **ImageWidth**	ImageSize
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	新規に実装されました。**ImageList** はコンポーネントです。
ListImage	Image (**ImageCollection** クラス)
ListImages	ImageCollection
MaskColor	TransparentColor
Parent	FindForm メソッド
Top	新規に実装されました。**ImageList** はコンポーネントです。
UseMaskColor	新規に実装されました。**TransparentColor** を **Transparent** に設定します。

メソッド

名前	Visual Basic 2005 での同等物
Overlay	新規に実装されました。

アップグレード メモ

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、**ImageList** コントロールは Windows フォームの **ImageList** コンポーネントにアップグレードされます。同等のプロパティ、メソッド、およびイベントがない場合、または動作に相違がある場合は、アップグレードに関する注意や警告がコードに追加されます。

参照

関連項目

ImageList コンポーネントの概要 (Windows フォーム)

その他の技術情報

ImageList コンポーネント (Windows フォーム)

Label コントロール (Visual Basic 6.0 ユーザー向け)

ここでは、Visual Basic 6.0 の **Label** コントロールと Visual Basic 2005 でそれらに相当するものを比較します。

Visual Basic 6.0 の **Label** コントロールは、Visual Basic 2005 では Windows フォームの Label コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

BackStyle プロパティ

Visual Basic 6.0 では、**Label** コントロールの **BackStyle** プロパティを使用して、ラベルの背景を透明にするかどうかを指定します。**BackStyle** プロパティを 0 – Transparent に設定すると、ラベルの背景にあるイメージが透けて見えます。ラベルが他のコントロールの前面にある場合、そのコントロールも透けて見えます。

Visual Basic 2005 では、**BackStyle** プロパティはなくなり、透明化の動作が少し変わりました。Visual Basic 2005 では、透明化の動作をエミュレートするために、BackColor プロパティを Transparent に設定します。これにより、背景のイメージが透けて見えるようになります。ラベルが他のコントロールの前面にある場合、Z オーダーの高いコントロールであればそれも透けて見えます。

メモ :
Visual Basic 2005 では、**ZOrder** メソッドはなくなりました。デザイン時には [書式] メニューの [最前面へ移動] コマンドまたは [最背面へ移動] コマンドを使用し、実行時にはBringToFront 関数または SendToBack 関数を使用できます。

WordWrap プロパティ

Visual Basic 6.0 では、ラベルに 1 行で収まらないテキストを複数行に折り返して表示するかどうかを **WordWrap** プロパティで指定します。

Visual Basic 2005 では、**Label** コントロール内のテキストは自動的に行を折り返して表示されます。折り返しを禁じる唯一の方法は、**Label** コントロールの高さをテキスト 1 行の高さと同じにすることです。

その他の違い

また、データ連結、フォント処理、ドラッグ アンド ドロップ機能、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Label コントロールを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

ラベルを透明にするコードの変更

次のコード例は、ラベル コントロールの背景を透明にする方法を示します。Visual Basic 6.0 では、**Label** の背後に他のコントロールがある場合、それらは透けて見えます。Visual Basic 2005 のコード例では、**Label** の背後にあるコントロールは、必要に応じて **BringToFront** 関数と **SendToBack** 関数を使って、透けて見えるようにできます。

```
' Visual Basic 6.0
Private Sub MakeTransparent()
    Label1.BackStyle = vbTransparent
End Sub
```

VB

```
' Visual Basic 2005
Private Sub MakeTransparent()
  Label1.BackColor = System.Drawing.Color.Transparent
  ' Let controls behind the label show through.
  Label1.SendToBack()
  ' Make the portion of controls behind the label transparent
  Label1.BringToFront()
End Sub
```

Label コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示し

ています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表には、動作の違いを説明するトピックへの必要なリンクも含まれています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

Label のプロパティ

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Alignment: **0 - Left Justify** **1 - Right Justify**	TextAlign MiddleLeft 列挙値 MiddleRight 列挙値
Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
AutoSize	AutoSize
BackColor	**BackColor** **メモ**　定数の一覧については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。 **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
BackStyle	新規に実装されました。詳細については、前の「概念の違い」を参照してください。
Caption	Text
Container	Parent
DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	新規に実装されました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ**　Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、 「フォント処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。

ForeColor	ForeColor **メモ：** 定数の一覧については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。 **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
LinkItem **LinkMode** **LinkTimeOut** **LinkTopic**	対応する項目がありません。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
RightToLeft: **True** **False**	RightToLeft Yes 列挙値 No 列挙値
ToolTipText	ToolTip 構成要素 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。

Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WordWrap	新規に実装されました。 **メモ：** Visual Basic 2005 の **Label** コントロール内では、テキストは既定で次の行へ折り返され、変更できません。

Label のメソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
LinkExecute **LinkPoke** **LinkRequest** **LinkSend**	対応する項目がありません。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	**BringToFront** メソッドまたは **SendToBack** メソッド **BringToFront** **SendToBack**

Label のイベント

Visual Basic 6.0	Visual Basic 2005 での同等物
Change	TextChanged
DblClick	DoubleClick

DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
LinkClose **LinkError** **LinkNotify** **LinkOpen**	対応する項目がありません。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**Label** コントロールは Windows フォームの **Label** コントロールにアップグレードされ、プロパティ、メソッド、およびイベントは、それぞれに相当するものにアップグレードされます。動作に違いがある部分のコードには、アップグレード コメントが挿入されます。

参照

概念

Label コントロールの概要

Line コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Line** コントロールに相当するものは、Visual Basic 2005 にありません。ただし、グラフィックス メソッドを使って、同じ結果を得ることができます。

概念の違い

Visual Basic 6.0 では、**Line** コントロールを使って、デザイン時にフォームに線を簡単に描画できます。**Line** コントロールは "軽量コントロール"です。つまり、このコントロールは Windows ハンドル (**HWnd** とも呼びます) を持ちません。

Visual Basic 2005 では、**Line** コントロールに相当するものはなく、軽量コントロールもサポートされなくなりました。ただし、デザイン時と実行時の両方で線をフォームに描画する方法があります。

デザイン時に垂直線または水平線をフォーム上に描画するには、Label コントロールを追加し、Text プロパティを空の文字列に設定し、BorderStyle プロパティを None に設定し、Width プロパティまたは Height プロパティを 1 に設定します。

新しい Graphics オブジェクトを作成し、そのメソッドを呼び出すことにより、実行時にフォームの Paint イベント ハンドラで垂直線、水平線、または斜線を描画できます。

Visual Basic 6.0 では、**Line** コントロールを使って、**PictureBox** コントロールまたは **Frame** コントロールなどのコンテナ コントロールの前面に線を描画するには、**Line** コントロールをコンテナに追加します。

Visual Basic 2005 では、DrawLine メソッドをコンテナ コントロールの **Paint** イベント内で呼び出すことにより、同じ結果が得られます。

Line コントロールを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

水平線または垂直線の描画

次のコード例は、実行時に水平線や垂直線をフォームに描画する方法を示しています。Visual Basic 6.0 のコード例では、**Line** コントロールを使用し、2 つの **Line** コントロールがデザイン時に追加されていると仮定します。Visual Basic 2005 のコード例では、**Label** コントロールを使用する方法と、**Graphics** の各メソッドを使用する方法の 2 つを示します。

メモ :
Visual Basic 6.0 での既定の測定単位は twip でした。Visual Basic 2005 での測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Load()
    ' Draw a horizontal line 200 twips from the top of the form.
    Line1.X1 = 0
    Line1.X2 = Me.Width
    Line1.Y1 = 200
    Line1.Y2 = 200
    Line1.BorderColor = vbRed
    Line1.BorderWidth = 1
    ' Draw a vertical line 200 twips from the left of the form.
    Line1.Y1 = 0
    Line1.Y2 = Me.Height
    Line1.X1 = 200
    Line1.X2 = 200
    Line1.BorderColor = vbBlue
    Line1.BorderWidth = 1
```

VB

```
' Visual Basic 2005
' Using Label controls.
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
    Dim Line1 As New System.Windows.Forms.Label
    Dim Line2 As New System.Windows.Forms.Label
    ' Draw a horizontal line 14 pixels from the top of the form.
    Line1.Location = New System.Drawing.Point(0, 14)
    Line1.Size = New System.Drawing.Size(Me.Width, 1)
```

```
    Line1.BorderStyle = BorderStyle.None
    Line1.BackColor = System.Drawing.Color.Red
    Line1.Text = ""
    Controls.Add(Line1)
    ' Draw a vertical line 14 pixels from the left of the form.
    Line2.Location = New System.Drawing.Point(14, 0)
    Line2.Size = New System.Drawing.Size(1, Me.Height)
    Line2.BorderStyle = BorderStyle.None
    Line2.BackColor = System.Drawing.Color.Blue
    Line2.Text = ""
    Controls.Add(Line2)
End Sub
```

VB

```
' Visual Basic 2005
' Using Graphics methods
Private Sub Form1Paint(ByVal sender As Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    ' Draw a horizontal line 28 pixels from the top of the form.
    e.Graphics.DrawLine(Pens.Red, 0, 28, Me.Width, 28)
    ' Draw a vertical line 28 pixels from the left of the form.
    e.Graphics.DrawLine(Pens.Blue, 28, 0, 28, Me.Height)
End Sub
```

斜線の描画

次のコードは、実行時にフォーム上に斜線を描画する方法を示します。Visual Basic 6.0 のコード例では、**Line** コントロールを使用し、**Line** コントロールがデザイン時に追加されていると仮定します。Visual Basic 2005 のコード例では、**Graphics** の各メソッドを使用します。

メモ：
Visual Basic 6.0 での既定の測定単位は twip でした。Visual Basic 2005 での測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Load()
    ' Draw a diagonal line from the top left to the lower right.
    Line1.X1 = 0
    Line1.X2 = Me.ScaleWidth
    Line1.Y1 = 0
    Line1.Y2 = Me.ScaleHeight
    Line1.BorderColor = vbBlack
    Line1.BorderWidth = 1
End Sub
```

VB

```
' Visual Basic 2005
Private Sub FormPaint(ByVal sender As Object, _
ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    ' Draw a diagonal line from the top left to the lower right.
    e.Graphics.DrawLine(Pens.Black, 0, 0, Me.ClientSize.Width, _
    Me.ClientSize.Height)
End Sub
```

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、垂直または水平の **Line** コントロールは Windows フォームの **Label** コントロールに置き換えられ、Text プロパティは空の文字列に設定され、BorderStyle は **None** に設定され、BackColor、**Width**、および **Height** の各プロパティは元のコントロールと同じ値に設定されます。

垂直、水平のいずれでもない **Line** コントロールはアップグレードされません。**Line** コントロールは、.NET Framework に組み込まれた

Graphics 関数に置き換えることができます。

参照

概念

グラフィックス (Visual Basic 6.0 ユーザー向け)

ListBox コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **ListBox** コントロールは、Visual Basic 2005 では ListBox コントロールまたは CheckedListBox コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

オンになっている ListBox

Visual Basic 6.0 では、各テキスト アイテムの横にチェック ボックスを表示するかどうかは **ListBox** コントロールの **Style** プロパティによって決定されます。**ListBox** 内の複数のアイテムを選択するには、アイテムの **MultiSelect** プロパティが **False** に設定されているかどうかにかかわらず、アイテムの横のチェック ボックスをオンにします。アイテムがオンになっていることをプログラムで確認する方法はありません。この機能が必要な場合は、代わりに **ListView** コントロールを使用します。

Visual Basic 2005 では、新しい **CheckedListBox** コントロールによってチェック ボックスが各アイテムの横に表示されます。**ListBox** コントロール内にチェック ボックスを表示することはできなくなりました。**CheckedListBox** コントロールを使うと、各アイテムがオンになっているかどうかを CheckedItemCollection コレクションを通じてプログラムで確認できます。

Columns プロパティ

Visual Basic 6.0 では、**Columns** プロパティは、表示する列の番号を示す整数でした。

Visual Basic 2005 では、MultiColumn プロパティはブール値であり、ColumnWidth プロパティは幅をピクセル数で示す整数です。実行時に **ListBox** コントロールの幅が変更された場合は、各列の **ColumnWidth** プロパティの設定も必要な場合があります。

ItemCheck イベント

Visual Basic 6.0 の **ListBox** コントロールでは、**ItemCheck** イベントが発生するときに、チェックの状態が既に変わっていました。

Visual Basic 2005 では、**CheckedListBox** コントロールは、ItemCheck イベントが発生したときにはまだチェックの状態に変化はありません。イベントに渡される *ItemCheckEventArgs* 引数から保留中の値を取得できます。

ListBox コントロールを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

ListBox コントロールにアイテムを追加および削除するコードの変更

次のコード例は、**ListBox** アイテムを追加および削除する方法を示します。

```
' Visual Basic 6.0
' Add an item at the end of the list.
List1.AddItem "Tokyo"
' Insert an item at the top of the list.
List1.AddItem "Copenhagen", 0
' Remove the first item.
List1.RemoveItem 0
```

VB

```
'Visual Basic 2005
' Add an item at the end of the list.
ListBox1.Items.Add("Tokyo")
' Insert an item at the top of the list.
ListBox1.Items.Insert(0, "Copenhagen")
' Remove the first item.
ListBox1.Items.RemoveAt(0)
```

ListBox コントロールの特定のアイテムにアクセスするコードの変更

次のコード例は、**ListBox** アイテムの値を返す方法を示します。

```
' Visual Basic 6.0
Private Function GetItemText(i As Integer) As String
   ' Return the text of the item using the index:
   GetItemText = ListBox1.List(1)
```

```
End Function
```

VB

```
' Visual Basic 2005
Private Function GetItemText(ByVal i As Integer) As String
    ' Return the text of the item using the index:
   GetItemText = CStr(ListBox1.Items(i))
End Function
```

CheckedListBox コントロールでアイテムがオンになっているかどうかを確認するコードの変更

次のコード例は、**CheckedListBox** コントロール内にあるアイテムのチェックの状態を確認する方法を示します。

```
' Visual Basic 6.0
' The Visual Basic 6.0 ListBox control didn't support this scenario,
' a ListView control had to be used instead.
Dim s As String
Dim i As Integer
' Loop through all items
For i = 1 To ListView1.ListItems.Count
    ' If an item is checked, add it to the string
   If ListView1.ListItems(i).Checked = True Then
      s = s & "Checked Item" & CStr(i) & " = " & _
ListView1.ListItems(i) & vbCrLf
   End If
Next
' Determine if any items are checked.
If s <> "" Then
    MsgBox s
End If
```

VB

```
' Visual Basic 2005
' Determine if there are any items checked.
If CheckedListBox1.CheckedItems.Count <> 0 Then
  ' If so, loop through all checked items and print results.
  Dim x As Integer
  Dim s As String = ""
  For x = 0 To CheckedListBox1.CheckedItems.Count - 1
      s = s & "Checked Item " & CStr(x + 1) & " = " & _
      CStr(CheckedListBox1.CheckedItems(x)) & ControlChars.CrLf
  Next x
  MessageBox.Show(s)
End If
```

ListBox コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表には、動作の違いを説明するトピックへのリンクも含まれています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。

BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Columns	**MultiColumn** および **ColumnWidth**
Container	Parent
DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	新規に実装されました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。

IntegralHeight	IntegralHeight
ItemData	新規に実装されました。詳細については、「ItemData プロパティをアップグレードできない」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
List	Items
ListCount	Count (**List.Count**)
ListIndex	SelectedIndex
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
MultiSelect	SelectionMode
NewIndex	新規に実装されました。詳細については、「NewIndex プロパティをアップグレードできない」を参照してください。
OLEDragMode **OLEDropMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
SelCount	Count (**SelectedItem.Count**)
Selected	GetSelected、SetSelected (**ListBox** コントロール) GetItemChecked、SetItemChecked (**CheckedListBox** コントロール)
Style	新規に実装されたプロパティ。0 – Standard は **CheckedListBox** コントロールに対応付けられ、1 – Checkbox は **CheckedListBox** コントロールに対応付けられます。
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

名前	Visual Basic 2005 での同等物
AddItem	Add AddRange Insert
Clear	Clear
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
RemoveItem	Remove
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder	BringToFront メソッドまたは SendToBack メソッド

イベント

Visual Basic 6.0	Visual Basic 2005 での同等物
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
ItemCheck	**ItemCheck** (**CheckedListBox** のみ)
LostFocus	Leave

OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Scroll	新規に実装されました。詳細については、「TopIndex プロパティおよび Scroll イベント (Visual Basic 6.0 ユーザー向け)」を参照してください。
Validate	Validating

アップグレード メモ

アップグレード時には、Visual Basic 6.0 の **Listbox** の **Style** プロパティが 1 – Checkbox に設定されている場合は **CheckedListBox** コントロールにアップグレードされ、それ以外の場合は **ListBox** コントロールにアップグレードされます。

参照

関連項目

ListBox コントロールの概要 (Windows フォーム)
ListBox コントロールの概要 (Windows フォーム)
CheckedListBox コントロールの概要 (Windows フォーム)

ListView コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **ListView** コントロールは、Visual Basic 2005 では ListView コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

DropHighlight プロパティ

Visual Basic 6.0 では、**ListView** コントロール内の **ListItem** オブジェクトにカーソルを移動すると、**DropHighlight** プロパティによって、**ListItem** オブジェクトがシステムの強調色で強調表示されます。このプロパティは、ドラッグ アンド ドロップ操作で一般に使用されます。

Visual Basic 2005 では、**DropHighlight** プロパティがなくなりました。**MouseEnter** イベントおよび **MouseLeave** イベントで強調表示をオーナー描画 (カスタム描画とも呼ばれます) することによって、同じ効果を実現できます。詳細については、「組み込みのオーナー描画サポートを備えたコントロール」を参照してください。

FlatScrollBar プロパティ

Visual Basic 6.0 では、**FlatScrollBar** プロパティによって、**ListView** コントロール内のスクロール バーがフラットな (2 次元の) 外観を持つかどうかが決まります。

Visual Basic 2005 では、**FlatScrollBar** プロパティがなくなりました。**ListView** コントロール内のスクロール バーは、常に 3 次元の外観を持ちます。

HideColumnHeaders プロパティ

Visual Basic 6.0 では、**HideColumnHeaders** プロパティによって、**ListView** コントロール内の **ColumnHeader** オブジェクトが **Report** ビューで非表示になるかどうかが決まります。

Visual Basic 2005 では、**HideColumnHeaders** プロパティがなくなりました。列ヘッダーの表示は、HeaderStyle プロパティを設定することによって制御できます。

ItemClick イベント

Visual Basic 6.0 の **ListView** コントロールには、**Click** イベントと **ItemClick** イベントがあります。**ItemClick** イベントは、**ListItem** オブジェクトを引数として受け取ります。

Visual Basic 2005 では、**ItemClick** イベントがなくなりました。代わりに、**Click** イベントと **EventSender.FocusedItem** 引数を使用して、クリックされた **ListViewItem** を判断します。

並べ替え

Visual Basic 6.0 では、**ListView** コントロールの並べ替えは、**Sorted**、**SortKey**、および **SortOrder** の各プロパティを組み合わせて実現されます。**Sorted** は、並べ替えが有効かどうかを決定します。**SortOrder** は、並べ替えが昇順か降順かを決定します。**SortKey** は、テキストの代わりに並べ替えに使用するサブアイテムを指定します。

Visual Basic 2005 では、並べ替えは、Sorting プロパティを使用して実現されます。このプロパティを使用すると、並べ替えの有効化と並べ替え順序の設定を 1 回の手順で実行できます。**SortKey** プロパティは、ListViewItemSorter プロパティに置き換えられています。

View プロパティ

Visual Basic 6.0 では、**View** プロパティによって、**ListView** コントロール内のアイテムの表示方法が決まります。アイテムは、大きなアイコンとテキスト (**lvwIcon**)、小さなアイコンとテキスト (**lvwSmallIcon**)、一覧 (**lvwList**)、または一覧とサブアイテム (**lvwReport**) の形式で表示されます。

Visual Basic 2005 では、**View** プロパティは引き続き存在しますが、列挙体が異なります。**lvwIcon** は LargeIcon に、**lvwSmallIcon** は SmallIcon に、**lvwList** は List に、**lvwReport** は Details に、それぞれ置き換えられています。また、Tile という新しいビューが追加されています。このビューでは、各アイテムが最大サイズのアイコンで表示され、その右側にアイテムのラベルとサブアイテム情報が表示されます。

ListView コントロールを扱うコードの変更

次のコード例は、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

ListView コントロール内の選択アイテムの確認を扱うコードの変更

次のコード例では、選択された **ListView** アイテムを変数に割り当てます。

```
' Visual Basic 6.0
Dim theItem As ListItem
theItem = ListView1.SelectedItem
```

VB

```
' Visual Basic 2005
Dim theItem As ListViewItem
If ListView1.SelectedItems.Count > 0 Then
    theItem = ListView1.SelectedItems(0)
Else
    theItem = Nothing
End If
```

ListView コントロール内のすべての選択アイテムの確認を扱うコードの変更

次のコード例では、メッセージ ボックスを表示し、**ListView** コントロール内で選択された各アイテムのテキストを表示します。

```
' Visual Basic 6.0
For i = 1 To ListView1.ListItems.Count
    If ListView1.ListItems(i).Selected = True Then
        MsgBox(ListView1.SelectedItem
    End If
Next i
```

VB

```
' Visual Basic 2005
For Each selectedItem As ListViewItem In ListView1.SelectedItems
    MsgBox(selectedItem.Text)
Next
```

ListView コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Arrange	Alignment
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、 「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ColumnHeaderIcons	ImageIndex, ImageKey
ColumnsHeaders	Columns
Container	Parent

DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
DropHighlight	新規に実装されました。
FlatScrollBar	新規に実装されました。
Font Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Icons	LargeImageList
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ListItems	Items
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDragMode **OLEDropMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
Picture	BackgroundImage

PictureAlignment	BackgroundImageTiled
SelectedItem	SelectedItems
SmallIcons	SmallImageList
Sorted **SortOrder**	**Sorting**
SortKey	**ListViewItemSorter**
TextBackground	BackColor
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

名前	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
FindItem	FindItemWithText
GetFirstVisible	TopItem
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Refresh	RedrawItems

SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
StartLabelEdit	BeginEdit
ZOrder	BringToFront メソッドまたは SendToBack メソッド

イベント

Visual Basic 6.0	Visual Basic 2005 での同等物
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
ItemClick	直接対応する項目はありません。
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Validate	Validating

アップグレード メモ

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、**ListView** コントロールは Windows フォームの **ListView** コントロールにアップグレードされます。同等のプロパティ、メソッド、およびイベントがない場合、または動作に相違がある場合は、アップグレードに関する注意や警告がコードに追加されます。

参照

その他の技術情報

ListView コントロール (Windows フォーム)

MaskEdBox コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **MaskEdBox** コントロールは、Visual Basic 2005 では Windows フォームの **MaskedTextBox** コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Visual Basic 6.0 では、**AutoTab** プロパティによって、**MaskEdBox** コントロールの **Text** プロパティに有効なデータが入力されると同時に、タブ オーダー内の次のコントロールにフォーカスが設定されるかどうかが決定されます。

Visual Basic 2005 には、**AutoTab** プロパティに相当するものはありません。入力を監視および検証し、次のコントロールにフォーカスを自分で設定することによって、同じ効果を実現できます。

Visual Basic 6.0 では、**PasswordChar** プロパティは **String** 型ですが、Visual Basic 2005 では **Char** 型です。

また、データ バインディング、フォント処理、ドラッグ アンド ドロップ、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)」を参照してください。

MaskedEdBox コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 で対応するもの
AllowPrompt	AllowPromptAsInput
Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
AutoTab	新規に実装されました。
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ClipMode	SkipLiterals
ClipText	Text **メモ：** Visual Basic 6.0 の **ClipText** プロパティはリテラルを除外します。Visual Basic 2005 では、**SkipLiterals** が **true** に設定されていない限り、リテラルが含まれます。
Container	Parent

DataBinding **DataChanged** **DataField** **DataFormat**	新規に実装されました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Format	新規に実装されました。FormatNumber 関数 (Visual Basic) や FormatDateTime 関数 (Visual Basic) などの書式設定関数を使用します。
FormattedText	**Text** **メモ：** Visual Basic 6.0 の **ClipText** プロパティはリテラルを除外します。Visual Basic 2005 では、**SkipLiterals** が **true** に設定されていない限り、リテラルが含まれます。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDragMode **OLEDropMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
PromptInclude	TextMaskFormat
SelLength	SelectionLength
SelStart	SelectionStart
SelText	SelectedText
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	**Visual Basic 2005 で対応するもの**
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder	BringToFront 関数または SendToBack 関数

イベント

Visual Basic 6.0	Visual Basic 2005 で対応するもの
Change	TextChanged
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Validate	Validating
ValidationError	MaskInputRejected

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**MaskEdBox** コントロールは Windows フォームの **MaskEdBox** コントロールにアップグレードされます。プロパティ、メソッド、およびイベントは対応するメンバにアップグレードされます。動作が異なる場合は、アップグレードに関するコメントがコードに挿入されます。

参照

処理手順

チュートリアル : MaskedTextBox コントロールの使用
MaskedTextBox コントロールのサンプル
方法 : MaskedTextBox コントロールにデータをバインドする

その他の技術情報

MaskedTextBox コントロール (Windows フォーム)

MDIForm オブジェクト (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、**MDIForm** は MDI (マルチ ドキュメント インターフェイス) アプリケーション用のコンテナとして機能する特別な種類のフォームでした。Visual Basic 2005 では、IsMdiContainer プロパティが **true** に設定されたフォームは、MDI コンテナとして動作できます。

概念の違い

子フォームと複数の MDI フォーム

Visual Basic 6.0 では、子フォームに **MDIChild** プロパティを設定することにより、そのフォームが MDI フォームの子であることを示します。MDI アプリケーションは MDI フォームを 1 つしか持つことができません。

Visual Basic 2005 では、**MDIChild** プロパティの代わりに MdiParent プロパティを使用し、子フォームを含む MDI フォームを示します。MDI アプリケーションは複数の MDI コンテナを持つことができます。

BackColor プロパティおよび Picture プロパティ

Visual Basic 6.0 では、**MDIForm** の背景色は、**BackColor** プロパティを設定することで変更できます。背景画像は **Picture** プロパティを設定することで表示できます。

Visual Basic 2005 では、MDI フォームの背景色を変更したり、MDI フォームに画像を表示したりすることはできません。フォームの BackColor プロパティや BackgroundImage プロパティを設定することはできますが、**IsMdiContainer** プロパティを **true** に設定すると、これらの設定は無効になります。

AutoShowChildren プロパティ

Visual Basic 6.0 では、MDI フォームの **AutoShowChildren** プロパティを使用して、フォームの読み込み時に MDI 子フォームを自動的に表示します。この自動表示は標準フォームには適用されておらず、**Load** メソッドを呼び出してもフォームは表示されません。

Visual Basic 2005 では、MDI 子フォームを表示するために、Show メソッドを明示的に呼び出す必要があります。

MDI フォーム上のコントロールおよびグラフィックス

Visual Basic 6.0 では、ほとんどのコントロールは **MDIForm** に追加できません。**Menu**、**StatusBar**、または **ToolBar** コントロールなどのドッキングされたコントロールだけを追加できます。また、**Circle** や **Line** などのグラフィックス メソッドは、**MDIForm** に使用できません。

Visual Basic 2005 では、任意のコントロールを MDI フォームに追加できます。ただし、コントロールが予期しない動作を見せる場合があります。フォーム上の **IsMdiContainer** プロパティが **true** に設定されたコントロールは、MDI 子フォームの前面に "浮き" ます。Paint イベント内でグラフィックス メソッドを呼び出すことはできますが、実行時にグラフィックスは表示されません。

MDI フォームを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

MDI 子フォームを表示するコードの変更

次のコード例は、MDI 子フォームの 5 つのインスタンスを表示する方法を示しています。

```
' Visual Basic 6.0
Dim F(1 To 4) As New Form1
Private Sub MDIForm_Load()
   Dim i As Integer
   Load Form1
   For i = 1 To 4
      F(i).Caption = "Form" & i + 1
      F(i).Show
   Next I
End Sub
```

VB

```
' Visual Basic 2005
Private Sub MDIForm1_Load(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles MyBase.Load
  Dim i As Integer = 0
  For i = 0 To 4
      Dim F As New FormChild
      F.Text = "Form " & CStr(i + 1)
      F.Show()
```

```
    Next
End Sub
```

MDI 子ウィンドウを配置するコードの変更

次のコード例は、MDI アプリケーションのウィンドウの配置を変更するコードの書き方を示しています。ここでは、各オプションのメニュー項目が既に作成されていると仮定します。

```
' Visual Basic 6.0
Private Sub menuCascade_Click()
   Me.Arrange vbCascade
End Sub
Private Sub menuTileHorizontally_Click()
   Me.Arrange vbHorizontal
End Sub
Private Sub menuTileVertically_Click()
   Me.Arrange vbVertical
End Sub
```

VB

```
' Visual Basic 2005
Private Sub CascadeToolStripMenuItem_Click(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles CascadeToolStripMenuItem.Click

  Me.LayoutMdi(MdiLayout.Cascade)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TileVerticleToolStripMenuItem_Click(ByVal sender As Object, _
ByVal e As EventArgs) Handles TileVerticalToolStripMenuItem.Click

  Me.LayoutMdi(MdiLayout.TileVertical)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TileHorizontalToolStripMenuItem_Click(ByVal sender As Object, _
ByVal e As EventArgs) Handles TileHorizontalToolStripMenuItem.Click

  Me.LayoutMdi(MdiLayout.TileHorizontal)
End Sub
```

MDI フォーム プロパティとメソッドの対応関係

Visual Basic 6.0 のプロパティおよびメソッドに相当する Visual Basic 2005 のプロパティおよびメソッドを次の表に示します。ここでは、**MDIForm** オブジェクトに固有のプロパティとメソッドだけを表に示しています。他のプロパティ、メソッド、およびイベントについては、「Form オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。

MDIForm のプロパティおよびメソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
ActiveForm プロパティ	ActiveMdiChild
Arrange メソッド	LayoutMdi

AutoShowChildren プロパティ	新規に実装されました。詳細については、「MDI フォームの Show の動作が変更されている」を参照してください。
BackColor プロパティ	新規に実装されました。MDI フォームの背景は、常に既定 (**Control**) です。
Picture プロパティ	新規に実装されました。Visual Basic 2005 では、MDI フォームに画像を直接表示することはできません。

アップグレード メモ

MDI アプリケーションを Visual Basic 6.0 から Visual Basic 2005 にアップグレードすると、**MDIForm** オブジェクトは通常のフォームにアップグレードされ、**IsMdiContainer** プロパティは **true** に設定されます。

Visual Basic 2005 では、**AutoShowChildren** プロパティが存在しません。Visual Basic 6.0 での動作をエミュレートするために、元のアプリケーションで **AutoShowChildren** が **true** に設定されていた場合には、アップグレード時に、子フォームを表示させるためのコードが 1 行追加されます。

参照

概念

MDI (Visual Basic 6.0 ユーザー向け)
Form オブジェクト (Visual Basic 6.0 ユーザー向け)
フォームのタスク (Visual Basic 6.0 ユーザー向け)

その他の技術情報

マルチ ドキュメント インターフェイス (MDI) アプリケーション

Menu オブジェクト (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Menu** オブジェクトは、Visual Basic 2005 では MenuStrip コントロールまたは ContextMenuStrip コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Visual Basic 6.0 では、メニュー エディタを使うと **Menu** オブジェクトが作成されます。**Menu** オブジェクトは、作成時のフォームに結び付けられますが、実行時に変更したり、コンテキスト メニューとして表示したりできます。**Menu** オブジェクトを使って作成したメニューは、網かけ、アイコン、または埋め込みコントロールをネイティブにサポートしません。Windows 98 スタイルの "平面的な" メニューだけを作成できます。

Visual Basic 2005 では、**Menu** オブジェクトは、**MenuStrip** コンポーネントおよび **ContextMenuStrip** コンポーネントに置き換えられています。メニューは、デザイン時に埋め込み先編集を使って作成したり、コードで作成したりできます。**MenuStrip** コンポーネントと **ContextMenuStrip** コンポーネントを使うと、網かけ領域、アイコン、および埋め込みコントロール (ドロップダウン リストなど) を完全にサポートした最新の Office ツール バー スタイルのメニューを作成できます。詳細については、「MenuStrip コントロールの概要 (Windows フォーム)」を参照してください。

コンテキスト メニューの表示

Visual Basic 6.0 では、**PopupMenu** メソッドを呼び出して、トップレベルの **Menu** オブジェクトを渡すことにより、コンテキスト メニューを表示できます。たとえば、`mnuEdit` という名前の [編集] メニューに [切り取り]、[コピー]、および [貼り付け] サブメニューがある場合、`PopupMenu mnuEdit` を呼び出して、[切り取り]、[コピー]、および [貼り付け] コマンドがあるコンテキスト メニューを表示できます。

Visual Basic 2005 では、コンテキスト メニューの表示に別の **ContextMenuStrip** コンポーネントを使用します。**PopupMenu** メソッドに相当するものはありませんが、デザイン時に **ContextMenuStrip** を作成してイベント ハンドラを共有することにより、**MenuStrip** と **ContextMenuStrip** でコマンドを共有できます。

Menu オブジェクトを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

コンテキスト メニューを表示するコードの変更

次のコード例は、既存の [編集] メニューの [切り取り]、[コピー]、および [貼り付け] コマンドをコンテキスト メニューに表示する方法を示します。

```
' Visual Basic 6.0
Private Sub mnuCut_Click()
    MsgBox "You selected Cut"
End Sub
Private Sub mnuCopy_Click()
    MsgBox "You selected Copy"
End Sub

Private Sub mnuPaste_Click()
    MsgBox "You selected Paste"
End Sub

Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    If Button = vbRightButton Then
        PopupMenu mnuEdit
    End If
End Sub
```

VB

```
' Visual Basic 2005
' You must first add a ContextMenuStrip component to the form at design
' time and add Cut, Copy, and Paste menu items named
' CutContextMenuItem, CopyContextMenuItem, and PasteContextMenuItem.

Private Sub CutToolStripMenuItem_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles CutToolStripMenuItem.Click
    MsgBox("Cut")
End Sub

Private Sub CopyToolStripMenuItem_Click(ByVal sender As System.Object, _
```

```
  ByVal e As System.EventArgs) Handles CopyToolStripMenuItem.Click
      MsgBox("Copy")
  End Sub

  Private Sub PasteToolStripMenuItem_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles PasteToolStripMenuItem.Click
      MsgBox("Paste")
  End Sub
```

VB

```
  Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As _
  System.Windows.Forms.MouseEventArgs) Handles Me.MouseDown

    If e.Button = Windows.Forms.MouseButtons.Right Then
      Me.ContextMenuStrip = ContextMenuStrip1

      AddHandler CutContextMenuItem.Click, _
        AddressOf CutToolStripMenuItem_Click

      AddHandler CopyContextMenuItem.Click, _
        AddressOf CopyToolStripMenuItem_Click

      AddHandler PasteContextMenuItem.Click, _
        AddressOf PasteToolStripMenuItem_Click
    End If
  End Sub
```

メニュー プロパティとメソッドの対応関係

次の表は、Visual Basic 6.0 のプロパティおよびメソッドと、Visual Basic 2005 でそれらに相当するものを示しています。同じ名前で同じ動作を持つプロパティとメソッドは、表に含まれません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表には、動作の違いを説明するトピックへのリンクも含まれています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Caption	Text (**MenuStripItem**)
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Index	新規に実装されました。ToolStripItemCollection クラスの **Index** プロパティを参照してください。
NegotiatePosition	同等の項目はありません。このプロパティは、OLE リンクと埋め込みにだけ使用されていましたが、現在はサポートされていません。
Parent	FindForm メソッド
WindowList	MdiWindowListItem

メソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
PopupMenu	新規に実装されました。**ContextMenuStrip** コンポーネントを使用します。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードしたときに、**PopupMenu** メソッドの呼び出しはアップグレードされません。代わりとなるコンテキスト メニューを作成する必要があります。

参照

関連項目

MenuStrip コントロールの概要 (Windows フォーム)

概念

メニュー処理 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

OLE Container コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、挿入可能な OLE オブジェクトをフォームに追加するために **OLE Container** コントロールを使用していました。Visual Basic 2005 には、**OLE Container** コントロールはありません。

概念の違い

OLE Container コントロールの機能が必要な場合、状況によっては、Visual Basic 2005 WebBrowser コントロールを **OLE Container** に置き換えて使用できることもあります。

主に **OLE Container** コントロールは、フォーム内に Word 文書または Excel スプレッドシートを埋め込むために使用されました。**WebBrowser** コントロールは、Internet Explorer をカプセル化します。これには、これらの種類のドキュメントを表示する機能があります。また、Visual Studio Tools for Office を使って、Visual Basic 2005 から Office に直接通信することもできます。

参照

関連項目

WebBrowser コントロールの概要

概念

アップグレードを行うにあたっての注意事項

その他の技術情報

Visual Studio Tools for Office

OptionButton コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **OptionButton** コントロールは、Visual Basic 2005 では Windows フォームの RadioButton コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Visual Basic 6.0 では、**OptionButton** コントロールの **Click** イベントが生成されるのは、このコントロールの **Value** プロパティが **True** に変更されたときだけでした。

Visual Basic 2005 では、**RadioButton** コントロールの CheckedChanged イベントは、Checked プロパティの値が **True** または **False** に変更されるたびに生成されます。

また、データ連結、フォント処理、ドラッグ アンド ドロップ機能、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)」を参照してください。

OptionButton コントロールを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

OptionButton を選択するコードの変更

次のコード例は、実行時に 2 つの **OptionButton** コントロールまたは **RadioButton** コントロールのうち 1 つを選択する方法を示します。

```
' Visual Basic 6.0
If Option1.Value = True Then
    Option2.Value = True
Else
    Option1.Value = True
End If
```

VB

```
' Visual Basic 2005
If RadioButton1.Checked = True Then
    RadioButton2.Checked = True
Else
    RadioButton1.Checked = True
End If
```

OptionButton の選択状態を確認するコードの変更

次のコード例は、**OptionButton** コントロールの **Click** イベントを処理する方法と、**RadioButton** コントロールの **CheckedChanged** イベントを処理する方法を示しています。

```
' Visual Basic 6.0
' The Click event is only fired when the Value is True
Private Sub Option1_Click()
    MsgBox "Option1 was clicked"
End Sub

Private Sub Option2_Click()
    MsgBox "Option2 was clicked"
End Sub
```

VB

```
' Visual Basic 2005
' The CheckChanged event fires each time the RadioButton's Checked
' value changes to either True or False.
Private Sub RadioButton1_CheckedChanged(ByVal sender As Object, ByVal _
e As System.EventArgs) Handles RadioButton1.CheckedChanged, _
RadioButton2.CheckedChanged
    ' Only execute if the Checked value is True.
```

```
    If sender.Checked = True Then
        MsgBox(sender.Name & " was clicked")
    End If
End Sub
```

OptionButton コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表には、動作の違いを説明するトピックへのリンクも含まれています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Alignment	TextAlign
Appearance	FlatStyle
BackColor	BackColor **メモ：** 定数の一覧については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。 **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Caption	Text
Container	Parent
DisabledPicture **DownPicture**	新規に実装されました。詳細については、「Style プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。

ForeColor	ForeColor **メモ：** 定数の一覧については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。 **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
MaskColor	新規に実装されました。詳細については、「MaskColor (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
Picture	Image
RightToLeft: **True** **False**	RightToLeft Yes 列挙値 No 列挙値

Style	Appearance **メモ：** Visual Basic 2005 ではグラフィックスのスタイルの扱い方が異なります。詳細については、「Style プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
ToolTipText	ToolTip 構成要素 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
UseMaskColor	新規に実装されました。詳細については、「MaskColor (Visual Basic 6.0 ユーザー向け)」を参照してください。
Value	Checked
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder	BringToFront メソッドまたは SendToBack メソッド

イベント

Visual Basic 6.0	Visual Basic 2005 での同等物

Click	**CheckedChanged** **メモ：** Visual Basic 6.0 では、**Click** イベントが生成されるのは、状態が **True** に変更されたときだけでした。Visual Basic 2005 では、**True** と **False** のどちらに変更されたときでもこのイベントが生成されます。
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Validate	Validating

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**OptionButton** コントロールは Windows フォームの **RadioButton** コントロールにアップグレードされ、プロパティ、メソッド、およびイベントは、それぞれに相当するものにアップグレードされます。動作に違いがある部分のコードには、アップグレード コメントが挿入されます。

参照

関連項目

RadioButton コントロールの概要 (Windows フォーム)

PictureBox コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **PictureBox** コントロールは、Visual Basic 2005 では Windows フォームの PictureBox コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

コンテナとしての PictureBox

Visual Basic 6.0 の **PictureBox** コントロールは、コンテナ コントロールです。画像を表示することに加え、他のコントロールをグループ化して表示するためにも使用できます。

Visual Basic 2005 の **PictureBox** コントロールは、コンテナ コントロールではありません。画像の表示だけに使用します。コンテナ コントロールが必要な場合は、新しい Panel コントロールを **PictureBox** コントロールの代わりに使用できます。

AutoSize プロパティ

Visual Basic 6.0 では、**AutoSize** プロパティによって、**PictureBox** コントロールをデザイン時のサイズに維持するか (`AutoSize = False`)、画像の大きさに合わせて自動的にサイズ変更するか (`AutoSize = True`) が決定されます。

Visual Basic 2005 では、**AutoSize** プロパティは SizeMode プロパティで置き換えられています。Normal モードと AutoSize モードに加え、画像を中央寄せ、拡大、または縮小するオプションもあります。

グラフィックス プロパティおよびメソッド

Visual Basic 6.0 では、**PictureBox** コントロール上に線、形状、およびテキストを描画する場合は、グラフィックスの各種プロパティおよびメソッドを使用できます。

Visual Basic 2005 では、コントロールにはグラフィックスのプロパティやメソッドはなくなりました。**PictureBox** コントロール上に **T:System.Drawing.Graphics** オブジェクトを使って描画を行うことは、これまでどおりできます。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。

その他の違い

また、データ連結、フォント処理、ドラッグ アンド ドロップ機能、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)」を参照してください。

PictureBox コントロールを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

画像を表示するコードの変更

次のコードは、実行時に **PictureBox** コントロールに画像を表示する方法を示します。

```
' Visual Basic 6.0
Picture1.Picture = LoadPicture(App.Path & "\somepicture.jpg")
```

VB

```
' Visual Basic 2005
PictureBox1.Image = System.Drawing.Bitmap.FromFile( _
  My.Application.Info.DirectoryPath & "\somepicture.jpg")
```

PictureBox コントロールをクリアするコードの変更

次のコード例は、実行時に **PictureBox** コントロールをクリアして画像が表示されないようにする方法を示します。

```
' Visual Basic 6.0
Picture1.Picture = LoadPicture("")
```

VB

```
' Visual Basic 2005
If Not (PictureBox1.Image Is Nothing) Then
    PictureBox1.Image.Dispose()
```

```
    PictureBox1.Image = Nothing
End If
```

PictureBox コントロール上に描画するコードの変更

次のコードは、実行時に **PictureBox** コントロールの中央に直径 40 ピクセルの赤い円を描画する方法を示します。

```
' Visual Basic 6.0
Private Sub Picture1_Paint
    Dim x As Integer
    Dim y As Integer
    Picture1.ScaleMode = vbPixels
    x = Picture1.ScaleWidth / 2
    y = Picture1.ScaleHeight / 2
    Picture1.Circle (x, y), 20, vbRed
End Sub
```

VB

```
' Visual Basic 2005
Private Sub PictureBox1_Paint(ByVal sender As Object, ByVal e _
As System.Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint
    Dim radius As Integer = 20
    Dim diameter As Integer = radius * 2
    Dim x As Integer = (PictureBox1.Width / 2) - radius
    Dim y As Integer = (PictureBox1.Height / 2) - radius
    e.Graphics.DrawEllipse(Pens.Red, x, y, diameter, diameter)
End Sub
```

PictureBox コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表には、動作の違いを説明するトピックへのリンクも含まれています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Align	Dock
Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
AutoRedraw	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
AutoSize	**SizeMode**
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。

BorderStyle	BorderStyle **メモ：** Visual Basic 6.0 では、既定は **Fixed Single** ですが、Visual Basic 2005 では、None です。
ClipControls	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Container	Parent
CurrentX **CurrentY**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	新規に実装されました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
DrawMode **DrawStyle** **DrawWidth** **FillColor** **FillStyle**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。

HasDC **HDC**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Image	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。 **メモ：** Visual Basic 2005 では、**Image** プロパティはグラフィックス プロパティではありません。これは、Visual Basic 6.0 の **Picture** プロパティに相当します。
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
LinkItem **LinkMode** **LinkTimeOut** **LinkTopic**	対応する項目はありません。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDragMode **OLEDropMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm
Picture	Image

ScaleHeight **ScaleLeft** **ScaleMode** **ScaleTop** **ScaleWidth**	新規に実装されました。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Circle **Cls**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Line	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
LinkExecute **LinkPoke** **LinkRequest** **LinkSend**	対応する項目はありません。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

PaintPicture **Point** **Print** **PSet**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Scale **ScaleX** **ScaleY**	新規に実装されました。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
TextHeight **TextWidth**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder	BringToFront メソッドまたは SendToBack メソッド

イベント

Visual Basic 6.0	Visual Basic 2005 での同等物
Change	新規に実装されました。プロパティが変更されると、個別のイベント (BackgroundImageChanged、SizeChanged など) が発生します。
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
KeyDown **KeyPress** **KeyUp**	新規に実装されました。Visual Basic 2005 の **PictureBox** コントロールは、コンテナ コントロールではなく、キーストロークを受け取りません。
LinkClose **LinkError** **LinkNotify** **LinkOpen**	対応する項目はありません。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
LostFocus	Leave

OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Validate	Validating

アップグレード メモ

コントロールを含む **PictureBox** コントロールは、アップグレード時に **Panel** コントロールにアップグレードされます。Visual Basic 6.0 の **PictureBox** コントロールが **Panel** コントロールにアップグレードされると、**Picture** プロパティに割り当てられていた画像は、**Panel** コントロールの **BackgroundPicture** プロパティに対応付けられます。

参照

関連項目

PictureBox コントロールの概要 (Windows フォーム)

プリンタ オブジェクト (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、Visual Basic 6.0 の **Printer** オブジェクトの代わりに、PrintDocument コンポーネントが用意されています。これら 2 つの動作は大きく異なりますが、機能の大半が再現できます。

概念上の相違点

Visual Basic 6.0 での印刷は、**Printer** オブジェクトを作成し、グラフィックス メソッドを使用して仮想ページにテキストやグラフィックスを描画することにより実行します。プロパティやメソッドは、**DeviceName**、**PrintQuality**、または **Copies** などのプリンタ属性を定義するために使用します。**Orientation** や **PaperSize** などその他のプロパティでは、ページ自体の属性を定義します。**EndDoc** メソッドを使用すると、**Printers** コレクションの定義に従って、アプリケーションの既定のプリンタに出力が送信されます。

Visual Basic 2005 では、**Printer** オブジェクトがなくなりました。代わりに、**PrintDocument** コンポーネントを使用してグラフィックスやテキストを定義し、**PrinterSettings** オブジェクトを使用してプリンタ属性を定義し、**PageSettings** クラスを使用してページ属性を定義します。

印刷は特定のデバイスに制限されず、アプリケーションの既定のプリンタという概念は廃止されました。代わりに、**PrintDocument** コンポーネントの **PrintPage** メソッドを使用して、任意のデバイスで印刷します。既定のプリンタはシステム全体に適用されます。**PrintDialog**、**PrintPreviewDialog**、および **PageSetupDialog** の各コンポーネントを使用すると、ユーザーが実行時にプリンタや印刷オプションを選択できるようになります。

ColorMode プロパティ

Visual Basic 6.0 では、出力をカラー プリンタでモノクロ印刷するかどうかは **ColorMode** プロパティによって制御します。

Visual Basic 2005 では、この制御を詳細オプションとして公開するかどうかは、プリンタによって異なります。**PrinterSettings** クラスの **SupportsColor** プロパティを使用すると、色に関するプリンタの機能を調べることができます。

DriverName プロパティ

Visual Basic 6.0 では、**Printer** オブジェクトの **DriverName** プロパティを使用してプリンタ ドライバを指定します。初期のバージョンの Windows や MS-DOS では、出力をプリンタが解釈できる形式に変換するために、各ブランドやモデルごとにプリンタ ドライバが必要でした。Visual Basic 6.0 の時代には、プリンタ ドライバはほとんど不要になっていましたが、下位互換性維持のためにプロパティは残されました。

Visual Basic 2005 では、**DriverName** プロパティがなくなりました。プリンタ ドライバは Windows によって管理されるため、別のドライバを指定できなくなったためです。

hDC プロパティ

Visual Basic 6.0 では、**Printer** オブジェクトの **hDC** プロパティを使用して、デバイス コンテキスト (Windows ベースのアプリケーション、デバイス ドライバ、およびプリンタなど出力デバイス間のリンク) を識別するハンドルを指定します。

Visual Basic 2005 では、**hDC** プロパティがなくなりました。**PrintDocument** コンポーネントのインスタンスがデバイス コンテキストに相当します。

メモ：
デバイス コンテキストを識別するハンドルは不要になりましたが、グラフィックス オブジェクトには、高度なシナリオで使用する可能性のある GetHdc メソッドと ReleaseHdc メソッドがあります。

Page プロパティ

Visual Basic 6.0 では、**Page** プロパティは、アプリケーションの起動以降、または **Printer** オブジェクトに対して **EndDoc** ステートメントが最後に使用されたとき以降に印刷されたページのカウントを返します。このプロパティは、多くの場合、印刷時に各ページにページ番号を追加するために使用されています。

Visual Basic 2005 では、ページ番号は逐一追跡されませんが、**BeginPrint** イベントに変数を設定し、**PrintPage** イベントでこの変数をインクリメントすることにより、簡単にカウントを取ることができます。

Port プロパティ

Visual Basic 6.0 では、**Port** プロパティは、ドキュメントをプリンタに送信するときに使用されるポート名を返します。

Visual Basic 2005 では、**Port** プロパティがなくなり、**PrintDialog** コントロールと **PrintPreviewDialog** コントロールによって、ポート情報が自動的に管理されます。

RightToLeft プロパティ

Visual Basic 6.0 では、**RightToLeft** プロパティにより、**Printer** オブジェクトが双方向プラットフォーム (アラビア語の Windows 95 やヘブライ語の Windows 95 など) で出力を書式設定する方法が決定されます。

Visual Basic 2005 では、**RightToLeft** プロパティが不要になりました。Windows の最近のバージョンでは、印刷の方向がローカリゼーション設定によって制御されるためです。

TrackDefault プロパティ

Visual Basic 6.0 では、**TrackDefault** プロパティによって、**Printer** オブジェクトが常に同じプリンタを指すか、オペレーティング システムのコントロール パネルで既定のプリンタの設定が変更されたときに、指すプリンタを変更するかが決定されます。印刷ジョブの途中で **TrackDefault** プロパティの設定を変更すると、印刷が直ちに中断されます。

Visual Basic 2005 では、**TrackDefault** プロパティがなくなりました。**PrinterSettings** クラスの **IsDefaultPrinter** プロパティを使用して、プリンタを既定に設定するかどうかを決定できますが、既定のプリンタを変更しても、印刷が中断されることはなくなりました。

Zoom プロパティ

Visual Basic 6.0 では、**Zoom** プロパティによって、印刷時に出力を拡大または縮小する倍率が決定されます。たとえば、レター サイズのページを **Zoom** を 50 に設定して印刷すると、17 × 22 インチのページと同量のデータが印刷されます。これは、テキストおよびグラフィックスが、元の高さおよび幅の 2 分の 1 に縮小されて印刷されるためです。

Visual Basic 2005 では、**Zoom** プロパティがなくなりました。プリンタにズーム機能がある場合は、その設定が [印刷] ダイアログ ボックスに自動的に公開されます。また、グラフィックス メソッドを使用すると、出力を **PrintDocument** コンポーネントに割り当てる前に、その出力を拡大または縮小できます。

グラフィックス プロパティとグラフィックス メソッド

Visual Basic 6.0 では、さまざまなグラフィックス プロパティとグラフィックス メソッドを使用して、**Printer** オブジェクトに線や図形、テキストを描画できます。

Visual Basic 2005 では、ほとんどのオブジェクトにグラフィックス プロパティやグラフィックス メソッドがありません。線、図形、およびテキストの描画は、**Graphics** オブジェクトを作成および使用することによって行います。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。

Printer オブジェクトに関するコードの変更点

Visual Basic 6.0 と Visual Basic 2005 の印刷モデルには数多くの相違点があるため、コーディング技術を比較してもあまり意味はありません。Visual Basic 2005 での印刷の例については、「印刷のサンプル」を参照してください。

Printer オブジェクトのプロパティおよびメソッドの対応

次の表は、Visual Basic 6.0 のプロパティおよびメソッドと、対応する Visual Basic 2005 のプロパティおよびメソッドの一覧を示しています。同じ名前で同じ動作を持つプロパティおよびメソッドは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に対応付けられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。直接 Visual Basic 2005 に対応するメンバがない場合は、代わりの説明を示すトピックへのリンクがあります。

プロパティ

Visual Basic 6.0	対応する Visual Basic 2005 のイベント
ColorMode	新規に実装されました。PrinterSettings クラスの SupportsColor プロパティを使用すると、色に関するプリンタの機能を調べることができます。
Copies	**PrinterSettings** クラスの Copies プロパティ
CurrentX **CurrentY**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DeviceName	**PrinterSettings** クラスの PrinterName プロパティ
DrawMode **DrawStyle** **DrawWidth**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。

DriverName	新規に実装されました。プリンタ ドライバは Windows によって管理されるため、現在では必要ありません。
Duplex	**PrinterSettings** クラスの Duplex プロパティ
FillColor **FillStyle**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font **FontBold** **FontCount** **FontItalic** **FontName** **Fonts** **FontSize** **FontStrikeThru** **FontTransparent** **FontUnderline**	新規に実装されました。詳細については、「フォント処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	新規に実装されました。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
hDC	この項目は必要なくなりました。**PrintDocument** コンポーネントのインスタンスが、デバイス コンテキストに相当します。
Height	PageSettings クラスの PaperSize プロパティ
Orientation	**PageSettings** クラスの Landscape プロパティ
Page	新規に実装されました。ページ番号は逐一追跡されませんが、BeginPrint イベントに変数を設定し、PrintPage イベントでこの変数をインクリメントすることにより、同じことを行うことができます。
PaperBin	**PrinterSettings** クラスの PaperSources プロパティ
PaperSize	**PageSettings** クラスの **PaperSize** プロパティ
Port	この項目は必要なくなりました。ポート情報は PrintPreviewDialog コントロールにより自動的に設定されます。
PrintQuality	**PrinterSettings** クラスの PrinterResolutions プロパティ
RightToLeft	この項目は必要なくなりました。印刷方向は、Windows のローカリゼーション設定によって制御されます。

ScaleHeight **ScaleLeft** **ScaleMode** **ScaleTop** **ScaleWidth**	新規に実装されました。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
TrackDefault	直接対応する項目はありません。**PrinterSettings** クラスの IsDefaultPrinter プロパティを使用すると、プリンタを既定に設定するかどうかを決定できます。
TwipsPerPixelX **TwipsPerPixelY**	この項目は必要なくなりました。Visual Basic 2005 では常にピクセル単位で測定が行われます。
Width	**PageSettings** クラスの **PaperSize** プロパティ
Zoom	この項目は必要なくなりました。プリンタにズーム機能がある場合は、設定が自動的に [印刷] ダイアログ ボックスに公開されます。

メソッド

Visual Basic 6.0	対応する Visual Basic 2005 のイベント
Circle	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
EndDoc	Print
KillDoc	Cancel
Line	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
NewPage	HasMorePages
PaintPicture **PSet**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。
Scale **ScaleX** **ScaleY**	新規に実装されました。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
TextHeight **TextWidth**	新規に実装されました。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードしても、**Printer** オブジェクトのインスタンスはアップグレードされず、アップグレード エラーが発生します。**Printer** オブジェクトのコードを削除し、**PrintDocument** コンポーネントを使用して印刷を再度実装する必要があります。

参照

処理手順

印刷のサンプル

関連項目

PageSettings Class

PrinterSettings Class

概念

印刷の変更点 (Visual Basic 6.0 ユーザー向け)

グラフィックス (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォームにおける印刷のサポート

Printers コレクション (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 には、Visual Basic 6.0 の **Printers** コレクションに相当するものがありません。

概念上の相違点

Visual Basic 6.0 では、**Printers** コレクションを使用して、システムで使用できるプリンタ情報を返します。一般的には、**Printers** コレクションを反復処理して特定の属性を持つプリンタを検索し、そのプリンタをアプリケーションの既定のプリンタに設定します。たとえば、ドット マトリックス プリンタではなく、レーザー プリンタを選択したりします。

Visual Basic 2005 では、**Printers** コレクションに直接相当するものはなく、モデルが変更されています。つまり、アプリケーション用のプリンタを選択するのではなく、ユーザーがプリンタを選択できるようにする必要があります。

PrintDialog コンポーネントを使用すると、使用可能なプリンタの一覧をユーザーに表示できます。ユーザーが選択したプリンタのプロパティは、PrinterSettings クラスで取得できます。

Visual Basic 6.0 の **Printers** コレクションには、**Item**、**Count** という 2 つのプロパティがあります。**PrintDialog** コンポーネントには、これに相当するプロパティはありません。**PrinterSettings** クラスには、**Item** および **Count** を備えた **InstalledPrinters** コレクションがありますが、これは **String** コレクションであり、これを使用してプリンタの属性を照会することはできません。

Printers コレクションに関するコードの変更点

次の例は、Visual Basic 6.0 と Visual Basic 2005 のコーディング技術の相違点を示しています。

使用できるプリンタの一覧を返すコードの変更点

次の例は、使用できるプリンタの一覧を **ListBox** コントロールに表示する方法を示しています。

```
' Visual Basic 6.0
Dim X As Printer
For Each X In Printers
    List1.AddItem X.DeviceName
Next
```

VB

```
' Visual Basic 2005
Dim i As Integer
Dim pkInstalledPrinters As String
For i = 0 To System.Drawing.Printing.PrinterSettings. _
  InstalledPrinters.Count - 1

  pkInstalledPrinters = System.Drawing.Printing.PrinterSettings. _
    InstalledPrinters.Item(i)
  ListBox1.Items.Add(pkInstalledPrinters)
Next
```

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードしても、**Printers** コレクションのインスタンスはアップグレードされず、アップグレード エラーが発生します。**Printers** コレクションのコードを削除し、**PrintDialog** コンポーネントを使用するようコードを変更する必要があります。

参照

関連項目

PrintDialog コンポーネントの概要 (Windows フォーム)

PrinterSettings Class

概念

印刷の変更点 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォームにおける印刷のサポート

ProgressBar コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **ProgressBar** コントロールは、Visual Basic 2005 では Windows フォームの ProgressBar コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Negotiate プロパティ

Visual Basic 6.0 では、**ProgressBar** コントロールの **Negotiate** プロパティを使用すると、ツール バーを表示するときにコントロールを自動的に非表示にできます。

Visual Basic 2005 の **ProgressBar** コントロールには **Negotiate** プロパティはなく、実行時に **Visible** プロパティを **false** に設定することで同じ効果を実現できます。

Orientation プロパティ

Visual Basic 6.0 では、**ProgressBar** コントロールは、**Orientation** プロパティを設定することによって水平方向または垂直方法に表示できます。Visual Basic 2005 の **ProgressBar** コントロールは、水平方向にのみ表示できます。

ProgressBar コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

必要に応じて、動作の違いを説明するトピックへのリンクが示されています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Align	Dock
Appearance **BorderStyle**	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Container	Parent
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	**Handle**
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ**　Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、 「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Max	Maximum
Min	Minimum
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
Negotiate	新規に実装されました。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Orientation	新規に実装されました。
Parent	FindForm
Scrolling	Style
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ**　Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Value	Value **メモ：** Visual Basic 6.0 では、**Value** が変更されると、**Change** イベントが生成されます。Visual Basic 2005 では、**Change** イベントの代わりに ValueChanged イベントが使用されます。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, Size **メモ**　Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder: **0—vbBringToFront** **1—vbSendToBack**	BringToFront 関数または SendToBack 関数 **BringToFront** **SendToBack**

イベント

Visual Basic 6.0	Visual Basic 2005 での同等物
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

アップグレード メモ

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、**ProgressBar** コントロールは Windows フォームの **ProgressBar** コントロールにアップグレードされます。同等のプロパティ、メソッド、およびイベントがない場合、または動作に相違がある場合は、アップグレードに関する注意や警告がコードに追加されます。

参照

処理手順

方法 : Windows フォーム ProgressBar コントロールによって表示される値を設定する

その他の技術情報

ProgressBar コントロール (Windows フォーム)

RDO データ コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 には、Visual Basic 6.0 の RDO データ コントロールに相当するものがありません。

概念上の相違点

Visual Basic 6.0 の RDO データ コントロールは、リモート データ オブジェクト (RDO: Remote Data Object) を使用してコントロールをデータベースにバインドするための機構として使用されています。Visual Basic 2005 では、データ バインディングのアーキテクチャが変更されたため、RDO がサポートの対象から除外されました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードしても、RDO データ コントロールのインスタンスはアップグレードされず、アップグレード エラーが発生します。アップグレードの前に Visual Basic 6.0 アプリケーションを変更して ActiveX データ オブジェクト (ADO: ActiveX Data Objects) を使用するようにするか、アップグレード後のアプリケーションでデータ バインディングに関するコードを変更して新しいデータ アーキテクチャを使用するようにする必要があります。

参照

概念

アップグレードを行うにあたっての注意事項
データ アクセス (Visual Basic 6.0 ユーザー向け)

RichTextBox コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **RichTextBox** コントロールは、Visual Basic 2005 では Windows フォームの **RichTextBox** コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

AutoVerbMenu プロパティ

Visual Basic 6.0 では、**RichTextBox** コントロールの **AutoVerbMenu** プロパティを使用して、一連の標準コマンドを含んだコンテキスト メニューを表示できます。

Visual Basic 2005 では、**AutoVerbMenu** プロパティが存在しません。RichTextBox コントロールにコンテキスト メニューを用意するには、ContextMenuStrip コンポーネントを使用します。詳細については、「AutoVerbMenu プロパティはアップグレードされませんでした。」を参照してください。

DisableNoScroll プロパティ

Visual Basic 6.0 では、**DisableNoScroll** プロパティを使用して、**RichTextBox** コントロール内のテキスト行が少なくて縦方向にスクロールできない場合、または文字が少なくて横方向にスクロールできない場合に、スクロール バーを無効として表示するかどうかを決定します。

Visual Basic 2005 では、**DisableNoScroll** プロパティが存在しません。**ScrollBars** プロパティを使用するとスクロール バーを強制的に表示できますが、スクロール バーを無効として表示することはできません。

FileName プロパティと LoadFile メソッド

Visual Basic 6.0 では、**FileName** プロパティまたは **LoadFile** メソッドを使用して、.rtf ファイルまたはテキスト ファイルを **RichTextBox** コントロールに読み込むことができます。

Visual Basic 2005 では、**LoadFile** メソッドのみ使用できます。デザイン時にファイルを割り当てることはできません。

SelPrint メソッド

Visual Basic 6.0 では、**RichTextBox** コントロールの **SelPrint** メソッドを使用して、このコントロールの内容を印刷できます。このメソッドは、デバイスのデバイス コンテキストを指定する 1 つの引数 **hDC** を受け取ります。

Visual Basic 2005 では、**SelPrint** メソッドはなくなりました。また、デバイス コンテキストを印刷に使用することもなくなりました。**RichTextBox** コントロールには、内容を印刷するためのメソッドはありません。しかし、**EM_FORMATRANGE** メッセージを使用して **RichTextBox** クラスを拡張することはできます。これにより、**RichTextBox** の内容をプリンタなどの出力デバイスに送信できるようになります。詳細については、「SelPrint プロパティはアップグレードされませんでした。」を参照してください。

SelTabCount プロパティ

Visual Basic 6.0 では、**SelTabCount** プロパティを使用して、選択された段落、またはカーソル位置に続く段落におけるタブ位置の数を決定します。

Visual Basic 2005 では、**SelTabCount** プロパティはなくなり、**SelectionTabs** プロパティの **Length** パラメータ (**SelectionTabs.Length**) に置き換えられています。

Span メソッド

Visual Basic 6.0 では、**RichTextBox** コントロールの **Span** メソッドを使用して、指定した文字セットに基づいてテキスト範囲を選択できます。

Visual Basic 2005 では、**Span** メソッドはなくなりました。**Find** メソッドと **characterSet()** 引数を使用して指定された文字を検索し、**SelectedText** プロパティを使用して範囲を選択できます。

Upto メソッド

Visual Basic 6.0 では、**RichTextBox** コントロールの **Upto** メソッドを使用して、カーソル位置を、指定した文字セット メンバの先頭文字まで移動できます。ただし、この先頭文字は含みません。

Visual Basic 2005 では、**Upto** メソッドはなくなりました。**Find** メソッドと **characterSet()** 引数を使用して指定された文字を検索し、**SelectionStart** プロパティを使用してカーソル位置を移動できます。

他の違い

また、データ バインディング、フォント処理、ドラッグ アンド ドロップ機能、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)」を参照してください。

RichTextBox コントロールを扱うコードの変更

次のコード例は、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

RichTextBox コントロール内を検索するコードの変更点

次のコードは、**RichTextBox** コントロール内で、カーソル位置からセンテンスの終端までの範囲で検索を行い、結果を強調表示する方法を示しています。この例では、**Text** プロパティに複数のセンテンスが入力された **RichTextBox** コントロールが必要であり、**Button** コントロールが必要です。コードはボタンの **Click** イベントから呼び出されるため、**RichTextBox** の **HideSelection** プロパティは **False** に設定する必要があります。False に設定しないと、強調表示されません。

```
' Visual Basic 6.0
' Make sure that HideSelection is set to False.
Private Sub Command1_Click()
    RichTextBox1.Span ".?!", True, True
End Sub
```

VB

```
' Visual Basic 2005
' Make sure that HideSelection is set to False.
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click

    Dim endChars() As Char = New Char() {".", "!", "?"}
    Dim intEnd As Integer
    intEnd = RichTextBox1.Find(endChars, RichTextBox1.SelectionStart)
    RichTextBox1.SelectionLength = intEnd - RichTextBox1.SelectionStart
End Sub
```

RichTextBox コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
AutoVerbMenu	新規に実装されました。詳細については、「AutoVerbMenu プロパティはアップグレードされませんでした。」を参照してください。
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Container	Parent

DataBindings **DataChanged** **DataField** **DataFormat** **DataMember** **DataSource**	新規に実装されました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DisableNoScroll	新規に実装されました。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
FileName	LoadFile メソッド
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDragMode **OLEDropMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
SelAlignment	SelectionAlignment
SelBold	SelectionFont
SelBullet	SelectionBullet
SelCharOffset	SelectionCharOffset
SelColor	SelectionColor
SelFontName	**SelectionFont**
SelFontSize	**SelectionFont**
SelHangingIndent	SelectionHangingIndent
SelIndent	SelectionIndent
SelItalic	**SelectionFont**
SelLength	SelectionLength
SelProtected	SelectionProtected
SelRightIndent	SelectionRightIndent
SelRTF	SelectedRtf
SelStart	SelectionStart
SelStrikethru	**SelectionFont**

SelTabCount	**SelectionTabs.Length**
SelTabs	SelectionTabs
SelText	SelectedText
SelUnderline	**SelectionFont**
TextRTF	Rtf
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GetLineFromChar	GetLineFromCharIndex
LinkExecute **LinkPoke** **LinkRequest** **LinkSend**	対応する項目はありません。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
SelPrint	新規に実装されました。詳細については、「SelPrint プロパティはアップグレードされませんでした。」を参照してください。

SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Span	直接対応する項目はありません。**Find** メソッドと **characterSet** 引数を使用します。
Upto	直接対応する項目はありません。**Find** メソッドと **characterSet** 引数を使用します。
ZOrder	BringToFront 関数または SendToBack 関数

イベント

Visual Basic 6.0	Visual Basic 2005 での同等物
Change	TextChanged
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
SelChange	SelectionChanged
Validate	Validating

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**RichTextBox** コントロールは Windows フォームの **RichTextBox** コントロールにアップグレードされます。プロパティ、メソッド、およびイベントは対応するメンバにアップグレードされます。動作が異なる場合は、アップグレードに関するコメントがコードに挿入されます。

参照

関連項目

RichTextBox コントロールの概要 (Windows フォーム)

その他の技術情報

RichTextBox コントロール (Windows フォーム)

Screen オブジェクト (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 には、Visual Basic 6.0 の **Screen** オブジェクトに相当するものはありませんが、その機能の大部分は、.NET Framework を使用して再現できます。

概念上の相違点

Visual Basic 6.0 では、**Screen** オブジェクトにより、アプリケーションのアクティブなフォームおよびコントロールにアクセスしたり、アプリケーションが表示されている画面に関する情報を取得したり、カーソルの外観を制御したりできます。

Visual Basic 2005 には、直接 **Screen** オブジェクトに相当するものはありませんが、その機能の大部分は、.NET Framework を使用して再現できます。

メモ：

Visual Basic 2005 にも、**Screen** プロパティ (**My.Computer.Screen**) があります。**My.Computer.Screen** は、Visual Basic 6.0 の **Screen** オブジェクトと異なり、画面のデバイス名、作業領域、色深度など、画面に関する読み取り専用の情報を返すのみです。詳細については、「My.Computer.Screen プロパティ」を参照してください。

ActiveControl プロパティ

Visual Basic 6.0 では、**Screen** オブジェクトの **ActiveControl** プロパティを使用して、フォーカスが設定されているコントロールを特定します。**ActiveControl** プロパティは、`Screen.ActiveControl` のようにグローバルに使用できます。この場合、現在選択されているフォーム上のアクティブなコントロールが返されます。`Form2.ActiveControl` などとして特定のフォームを参照すると、**ActiveControl** は、参照先フォームがアクティブである場合にフォーカスを得るコントロールを指定します。

Visual Basic 2005 では、グローバルな **ActiveControl** プロパティはなく、フォームのインスタンスに固有の **ActiveControl** プロパティがあります。このプロパティは、Visual Basic 6.0 で特定のフォームを参照する場合と同様に機能します。現在選択されているフォームのアクティブなコントロールを特定するには、まず **OpenForms** コレクションに対して **ContainsFocus** プロパティのチェックを反復的に行い、アクティブなフォームを特定する必要があります。

ActiveForm プロパティ

Visual Basic 6.0 では、**Screen** オブジェクトの **ActiveForm** プロパティを使用して、現在フォーカスが設定されているフォームを特定します。MDI 親フォームにフォーカスが設定されている場合、**ActiveForm** は、最後にフォーカスが設定されていた MDI 子フォームを返します。

Visual Basic 2005 では、グローバルな **ActiveForm** プロパティがなくなりました。アクティブなフォームを特定するには、**OpenForms** コレクションを反復処理して、**ContainsFocus** プロパティが **True** に設定されているフォームを探す必要があります。

Visual Basic 2005 の MDI 親フォーム (**IsMDIContainer** が **True** に設定されているフォーム) には、**ActiveMDIChild** プロパティがあります。これにより、**OpenForms** コレクションを使用せずにアクティブな子フォームを返すことができます。

MousePointer プロパティ

Visual Basic 6.0 では、**Screen** オブジェクトの **MousePointer** プロパティを使用して、カーソルの外観を変更します。これを設定すると、アプリケーション内のすべてのフォームに適用されます。

Visual Basic 2005 では、グローバルな **MousePointer** プロパティがなくなりました。各フォームに **Cursor** プロパティがあり、これを使用すると、そのフォームでのみカーソルの外観を変更できます。

TwipsPerPixel プロパティ

Visual Basic 6.0 では、**Screen** オブジェクトの **TwipsPerPixelX** プロパティと **TwipsPerPixelY** プロパティを使用して、画面の計測値を論理 twip (Visual Basic 6.0 の標準の計測単位) からピクセルに変換します。

Visual Basic 2005 では、ピクセルが標準の計測単位であるため、変換の必要はなくなりました。

Screen オブジェクトに関するコードの変更点

次の例は、Visual Basic 6.0 と Visual Basic 2005 のコーディング技術の相違点を示しています。

アクティブなコントロールを特定するコードの変更点

次のコードは、現在選択されているフォームの現在選択されているコントロールから、テキストをクリップボードにコピーする方法を示しています。

```
' Visual Basic 6.0
If TypeOf Screen.ActiveControl Is TextBox Then
    Clipboard.SetText Screen.ActiveControl.Text
```

```
  End If
```

VB

```
' Visual Basic 2005
Dim i As Integer
For i = 0 To My.Application.OpenForms.Count - 1
  If My.Application.OpenForms.Item(i).ContainsFocus Then
    If TypeOf (My.Application.OpenForms.Item(i).ActiveControl) _
      Is TextBox Then
        My.Computer.Clipboard.SetText(My.Application.OpenForms. _
          Item(i).ActiveControl.Text)
    End If
  End If
Next
```

アクティブなフォームを特定するコードの変更点

次のコードは、現在選択されているフォームのキャプションを変更する方法を示しています。

```
' Visual Basic 6.0
Screen.ActiveForm.Caption = "This is the selected form"
```

VB

```
' Visual Basic 2005
Dim i As Integer
For i = 0 To My.Application.OpenForms.Count - 1
  If My.Application.OpenForms.Item(i).ContainsFocus Then
    My.Application.OpenForms.Item(i).Text = _
      "This is the selected form"
  End If
Next
```

MDI アプリケーションのアクティブなフォームを特定するコードの変更点

次のコードは、現在選択されている MDI 子フォームのキャプションを変更する方法を示しています。

```
' Visual Basic 6.0
Screen.ActiveForm.Caption = "This is the selected child form"
```

VB

```
' Visual Basic 2005
Me.ActiveMdiChild.Text = "This is the selected child form"
```

Screen オブジェクトのプロパティの対応

Visual Basic 6.0 のプロパティに相当する Visual Basic 2005 のプロパティを次の表に示します。必要に応じて、動作の違いを説明するトピックへのリンクが示されています。直接 Visual Basic 2005 に対応するメンバがない場合は、代わりの説明を示すトピックへのリンクがあります。

プロパティ

Visual Basic 6.0	Visual Basic 2005 で対応するもの
ActiveControl	**My.Application.OpenForms(0).ActiveControl**
ActiveForm	**My.Application.OpenForms(0).ContainsFocus** または ActiveMdiChild (MDI アプリケーション)

FontCount **Fonts**	新規に実装されました。フォントを列挙する動作は変更されています。詳細については、「フォント処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	**My.Computer.Screen.Bounds.Height**
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	**System.Windows.Forms.Cursor**
TwipsPerPixelX **TwipsPerPixelY**	新規に実装されました。Visual Basic 2005 では、座標がピクセルで表され、twip は単位として使用されません。
Width	**My.Computer.Screen.Bounds.Width**

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**Screen** オブジェクトのプロパティが対応する Visual Basic 2005 のプロパティにアップグレードされます。動作が異なる場合は、アップグレードに関するコメントがコードに挿入されます。

参照

関連項目

My.Application.OpenForms プロパティ
My.Computer.Screen プロパティ

概念

App オブジェクト (Visual Basic 6.0 ユーザー向け)
フォント処理 (Visual Basic 6.0 ユーザー向け)

Shape コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Shape** コントロールに相当するものは、Visual Basic 2005 にありません。ただし、Graphics メソッドを使って、同じ結果を得ることができます。

概念の違い

Visual Basic 6.0 では、**Shape** コントロールを使って、四角形、円、およびその他の形状をデザイン時にフォームに簡単に描画できます。**Shape** コントロールは、"軽量の" コントロールです。つまり、Windows ハンドル (**HWnd** とも呼ばれます) を持たないコントロールです。

Visual Basic 2005 では、**Shape** コントロールに相当するものはなく、軽量コントロールもサポートされなくなりました。ただし、デザイン時と実行時の両方で形状をフォームに描画する方法があります。

デザイン時に正方形または四角形をフォーム上に描画するには、Label コントロールを追加し、Text プロパティを空の文字列に設定し、BorderStyle プロパティを FixedSingle に設定し、BackColor、Width、および Height を適切な色とサイズに設定します。

Graphics クラスから新しいオブジェクトを作成し、そのメソッドを呼び出すことにより、実行時にフォームの Paint イベント ハンドラで四角形、楕円形、および複雑な形状を描画できます。

Visual Basic 6.0 では、**Shape** コントロールを使って、**PictureBox** コントロールまたは **Frame** コントロールなどのコンテナ コントロールの前面に形状を描画するには、**Shape** コントロールをコンテナに追加します。

Visual Basic 2005 では、**Graphics** メソッドをコンテナ コントロールの `Paint` イベント内で呼び出すことにより、同じ結果が得られます。

Shape コントロールを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

四角形を描画するコードの変更

次のコードは、実行時にフォーム上に塗りつぶされた四角形を描画する方法を示します。Visual Basic 6.0 のコード例では、**Shape** コントロールを使用し、**Line** コントロールがデザイン時に追加されていると仮定します。Visual Basic 2005 のコード例では、**Label** コントロールを使用する方法と、**Graphics** の各メソッドを使用する方法の 2 つを示します。

メモ :

Visual Basic 6.0 での既定の測定単位は twip でした。Visual Basic 2005 での測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Load()
    ' Show a solid red rectangle 200 twips from the top left.
    Shape1.Top = 200
    Shape1.Left = 200
    Shape1.FillColor = vbRed
    Shape1.FillColor= vbFSSolid
    Shape1.BorderColor = vbRed
End Sub
```

VB

```
' Visual Basic 2005
' Using a Label control
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
    Dim Shape1 As New System.Windows.Forms.Label
    ' Show a solid red rectangle 14 pixels from the top left.
    Shape1.Location = New System.Drawing.Point(14, 14)
    Shape1.Size = New System.Drawing.Size(200, 100)
    Shape1.BorderStyle = BorderStyle.None
    Shape1.BackColor = System.Drawing.Color.Red
    Shape1.Text = ""
    Controls.Add(Shape1)
End Sub
```

VB

```
' Visual Basic 2005
' Using Graphics methods
Private Sub Form2_Paint(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    ' Draw a solid blue rectangle below the red rectangle.
    e.Graphics.FillRectangle(Brushes.Blue, 14, 120, 200, 100)
End Sub
```

円を描画するコードの変更

次のコードは、実行時にフォーム上に円を描画する方法を示します。Visual Basic 6.0 のコード例では、**Shape** コントロールを使用し、**Shape** コントロールがデザイン時に追加されていると仮定します。Visual Basic 2005 のコード例では、**Graphics** の各メソッドを使用します。

メモ：

Visual Basic 6.0 での既定の測定単位は twip でした。Visual Basic 2005 での測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Load()
    Draw a 1000 twip diameter red circle
    Shape1.Top = 0
    Shape1.Left = 0
    Shape1.Height = 1000
    Shape1.Width = 1000
    Shape1.Shape = vbShapeCircle
    Shape1.BorderColor = vbRed
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form3_Paint(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    ' Draw a 70 pixel diameter red circle.
    e.Graphics.DrawEllipse(Pens.Red, 0, 0, 70, 70)
End Sub
```

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**Shape** コントロールは Windows フォームの **Label** コントロールにアップグレードされ、**BorderStyle** プロパティは **FixedSingle** に設定され、BackColor、**Width**、および **Height** の各プロパティは元のコントロールと同じサイズに設定されます。

メモ：

Label コントロールは、元の **Shape** コントロールとは異なる順序でフォームに追加されることがあります。形状が部分的に重なっている場合は、BringToFront 関数または SendToBack 関数を呼び出して、表示されるレイアウトを同じにする必要があります。

Shape プロパティが 2-Oval または 3-Circle に設定された **Shape** コントロールはアップグレードできず、プレースホルダとして **Label** コントロールに置き換えられます。.NET Framework に組み込まれたグラフィックス関数で **Shape** コントロールを置き換えることができます。詳細については、「グラフィックス (Visual Basic 6.0 ユーザー向け)」を参照してください。

参照

概念

グラフィックス (Visual Basic 6.0 ユーザー向け)

StatusBar コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **StatusBar** コントロールは、Visual Basic 2005 では Windows フォームの StatusStrip コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Visual Basic 6.0 の **StatusBar** コントロールは、テキストとイメージを表示できる **Panel** オブジェクトのコレクションで構成されています。

Visual Basic 2005 の **StatusStrip** コントロールは、**ToolStrip** コントロールによく似ています。**StatusStrip** コントロールは、パネル (現在では **ToolStripStatusLabel** コントロールと呼ばれています) に加えて、プログレス バー、ドロップダウン メニュー、および分割ボタンを含むことができます。Windows フォームの **StatusBar** コントロールもありますが、代わりに **StatusStrip** を使用することをお勧めします。

SimpleText プロパティ

Visual Basic 6.0 では、**Style** プロパティが **sbrSimple** に設定されている場合に表示されるテキストは、**StatusBar** コントロールの **SimpleText** プロパティによって定義されます。

Visual Basic 2005 の **StatusStrip** コントロールでは、Simple スタイルはサポートされていません。そのため、**SimpleText** プロパティがなくなりました。

Style プロパティ

Visual Basic 6.0 では、**Style** プロパティによって、**StatusBar** コントロールに複数のパネルが表示されるか (**sbrNormal**)、または大きな 1 つのパネルだけが表示されるか (**sbrSimple**) が決定されます。Simple スタイルに設定されると、傾斜面のスタイルは境界線なしの浮き出し表示になります。

Visual Basic 2005 には **Style** プロパティがありません。Simple スタイルをエミュレートするには、ToolStripStatusLabel オブジェクトを 1 つ **StatusStrip** コントロールに追加し、その BorderStyle プロパティを設定します。

StatusBar コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

必要に応じて、動作の違いを説明するトピックへのリンクが示されています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

<table>
<tr><th>Visual Basic 6.0</th><th>Visual Basic 2005 での同等物</th></tr>
<tr><td>Align</td><td>Dock</td></tr>
<tr><td>Container</td><td>Parent</td></tr>
<tr><td>DragIcon
DragMode</td><td>新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。</td></tr>
<tr><td>Font
FontBold
FontItalic
FontName
FontSize
FontStrikethrough
FontUnderline</td><td>Font
メモ：
Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。</td></tr>
</table>

Height	Height **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	**Handle**
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Panels	ToolStripItemCollection
Parent	FindForm
ShowTips	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
SimpleText	新規に実装されました。
Style	新規に実装されました。
ToolTipText	**ToolTip** コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Value	Value **メモ：** Visual Basic 6.0 では、**Value** プロパティが変更されると、**Change** イベントが生成されます。Visual Basic 2005 では、**Change** イベントの代わりに ValueChanged イベントが使用されます。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	BringToFront 関数または SendToBack 関数 **BringToFront** **SendToBack**

イベント

Visual Basic 6.0	Visual Basic 2005 での同等物
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
PanelClick	Click
PanelDblClick	DoubleClick

アップグレード メモ

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、**StatusBar** コントロールは Windows フォームの **StatusBar** コントロールにアップグレードされます。同等のプロパティ、メソッド、およびイベントがない場合、または動作に相違がある場合は、アップグレードに関する注意や警告がコードに追加されます。

参照

処理手順

StatusStrip コントロールのサンプル

関連項目

StatusStrip

TextBox コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、Visual Basic 6.0 の **TextBox** コントロールの代わりに、Windows フォームの **TextBox** コントロールが用意されています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念上の相違点

Visual Basic 6.0 では、**MaxLength** プロパティによって、**TextBox** コントロールに入力できる文字数が決定されます。テキストをプログラムで挿入しようとすると、そのテキストは **MaxLength** プロパティで指定された長さに切り詰められます。

Visual Basic 2005 では、**MaxLength** プロパティの動作は、プログラムで追加されるテキストには適用されません。Visual Basic 6.0 と同様の動作を実現するには、自分自身で文字列を切り詰める必要があります。

Visual Basic 6.0 では、**PasswordChar** プロパティは **String** 型ですが、Visual Basic 2005 では **Char** 型です。

また、データ バインディング、フォント処理、ドラッグ アンド ドロップ操作、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)」を参照してください。

TextBox コントロールに関するコードの変更点

次の例は、Visual Basic 6.0 と Visual Basic 2005 のコーディング技術の相違点を示しています。

TextBox コントロール内を検索するコードの変更点

次のコードは、**TextBox** コントロール内の文字列を検索し、それを強調表示する方法を示しています。

```
' Visual Basic 6.0
Private Sub Form_Load ()
    Text1.Text = "Two of the peak human experiences"
    Text1.Text = Text1.Text & " are good food and classical music."
End Sub
Private Sub Form_Click ()
    Dim Search, Where
    ' Get search string from user.
    Search = InputBox("Enter text to be found:")
    ' Find string in text.
    Where = InStr(Text1.Text, Search)
    If Where Then
        Text1.SetFocus
        Text1.SelStart = Where - 1
        Text1.SelLength = Len(Search)
    Else
        MsgBox "String not found."
    End If
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
    TextBox1.Text = "Two of the peak human experiences are "
    TextBox1.Text = TextBox1.Text & "good food and classical music."
End Sub
Private Sub Form1_Click(ByVal sender As Object, ByVal e As _
System.EventArgs) Handles Me.Click
    Dim Search As String
    Dim Where As String
    ' Get search string from user.
    Search = InputBox("Enter text to be found:")
    ' Find string in text.
    Where = InStr(TextBox1.Text, Search)
    If Where Then
        TextBox1.Focus()
        TextBox1.SelectionStart = Where - 1
        TextBox1.SelectionLength = Len(Search)
```

```
    Else
        MsgBox("String not found.")
    End If
End Sub
```

TextBox コントロールの文字数を制限するコードの変更点

次のコードは、**MaxLength** プロパティを使用して最大文字数を指定する方法を示しています。

```
' Visual Basic 6.0
Private Sub Form_Load()
    Text1.MaxLength = 5
End Sub
Private Sub SetText()
    ' Only the first five characters will be displayed.
    Text1.Text = "Hello World"
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load2(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
    TextBox1.MaxLength = 5
End Sub
Private Sub SetText()
    ' Truncate the string to equal MaxLength.
    TextBox1.Text = Strings.Left("Hello World", TextBox1.MaxLength)
End Sub
```

TextBox コントロールのプロパティ、メソッド、およびイベントの対応

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に対応付けられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。直接 Visual Basic 2005 に対応するメンバがない場合は、代わりの説明を示すトピックへのリンクがあります。

プロパティ

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Alignment	TextAlign
Appearance	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Container	Parent

DataChanged **DataField** **DataFormat** **DataMember** **DataSource**	新規に実装されました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
ForeColor	ForeColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の処理 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

LinkItem **LinkMode** **LinkTimeOut** **LinkTopic**	新規に実装されました。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Locked	ReadOnly
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDragMode **OLEDropMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
SelLength	SelectionLength
SelStart	SelectionStart
SelText	SelectedText
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

LinkExecute **LinkPoke** **LinkRequest** **LinkSend**	新規に実装されました。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder	BringToFront 関数または SendToBack 関数

イベント

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Change	TextChanged
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
LinkClose **LinkError** **LinkNotify** **LinkOpen**	新規に実装されました。詳細については、「ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)」を参照してください。
LostFocus	Leave
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Validate	Validating

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**TextBox** コントロールは Windows フォームの **TextBox** コントロールにアップグレードされます。プロパティ、メソッド、およびイベントは対応するメンバにアップグレードされます。動作が異なる場合は、アップグレードに関するコメントがコードに挿入されます。

参照

関連項目

TextBox コントロールの概要 (Windows フォーム)

Timer コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、Visual Basic 6.0 の **Timer** コントロールの代わりに、Timer コンポーネントが用意されています。プロパティやイベントの中には、名称が異なるものや、動作の異なるものもあります。

概念上の相違点

Visual Basic 6.0 の **Timer** コントロールは、デザイン時にフォーム上に配置される実際のコントロールですが、実行時には表示されません。

Visual Basic 2005 の **Timer** は、デザイン時にトレイに追加されるコンポーネントです。コンポーネントであるため、**Parent** プロパティがありません。

メモ：
Timer コンポーネントのほかに、.NET Framework には、Timer および Timer という 2 つのタイマ クラスがあります。

Interval プロパティ

Visual Basic 6.0 では、**Interval** プロパティを 0 に設定することによって **Timer** コントロールを無効にできます。

Visual Basic 2005 では、Interval プロパティの最小値が 1 になっています。このため、Interval を 0 に設定すると、実行時に例外がスローされます。Visual Basic 2005 の **Timer** コンポーネントでは、Enabled プロパティを使用して、無効にするか有効にするかを決定できます。

Timer コントロールに関するコードの変更点

次の例は、Visual Basic 6.0 と Visual Basic 2005 のコーディング技術の相違点を示しています。

Timer コントロールを起動および停止するコードの変更点

次のコードは、**Timer** コントロールを実行時に有効および無効にする方法を示しています。

```
' Visual Basic 6.0
Public Function TimerOn(Interval As Integer)
    If Interval > 0 Then
        ' Start the timer.
        Timer1.Interval = Interval
    Else
        ' Stop the timer.
        Timer1.Interval = 0
    End If
End Function
```

VB

```
' Visual Basic 2005
Public Sub TimerOn(ByRef Interval As Short)
    If Interval > 0 Then
        ' Start the timer.
        Timer1.Enabled = True
    Else
        ' Stop the timer
        Timer1.Enabled = False
    End If
End Sub
```

Timer コントロールのプロパティおよびイベントの対応

次の表は、Visual Basic 6.0 のプロパティおよびイベントと、対応する Visual Basic 2005 のプロパティおよびイベントを示しています。同じ名前で同じ動作を持つプロパティおよびイベントは、一覧に含まれていません。

この表では、動作の相違点について説明するトピックへのリンクを示します。直接 Visual Basic 2005 に対応するメンバがない場合は、代わりの説明を示すトピックへのリンクがあります。

プロパティおよびイベント

Visual Basic 6.0	Visual Basic 2005 での同等物
Index プロパティ	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Interval プロパティ	**Interval** **メモ：** **Interval** プロパティの動作は変更されています。詳細については、「Timer の Interval プロパティの動作が変更されている」を参照してください。
Parent プロパティ	新規に実装されました。**Timer** はコンポーネントであり、親を持つことはできません。
Timer イベント	**Tick**

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**Timer** コントロールは Windows フォームの **Timer** コンポーネントにアップグレードされます。**Interval** プロパティを 0 に設定するコードが検出されると、アップグレードに関する警告がコードに追加されます。

参照

関連項目

Timer コンポーネントの概要 (Windows フォーム)

ToolBar コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **ToolBar** コントロールは、Visual Basic 2005 では ToolStrip コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Visual Basic 6.0 の **ToolBar** コントロールは ActiveX コントロールであり、ボタンとコンボ ボックスから成る単純なツール バーを作成するときに使用します。

Visual Basic 2005 の **ToolStrip** コントロールを使用すると、カスタマイズされたツール バーおよびユーザー インターフェイス要素を作成することや、Microsoft Windows XP、Microsoft Office、または Microsoft Internet Explorer の外観と動作を持つツール バーおよびユーザー インターフェイスを作成することができます。これらのユーザー インターフェイス要素は、オーバーフローおよび実行時の項目の並べ替えをサポートします。**ToolStrip** コントロールは、埋め込み先編集の有効化、カスタム レイアウト、およびラフティング (水平スペースまたは垂直スペースを共有するツール バーの機能) を含む豊富なデザイン時機能を提供します。

メモ :
Visual Basic 2005 には、以前のバージョンの ToolBar コントロールも残されていますが、優れた機能を備えている **ToolStrip** コントロールの使用をお勧めします。

AllowCustomize プロパティ、Customize メソッド、Change イベント

Visual Basic 6.0 では、**ToolBar** コントロールの **AllowCustomize** プロパティを使用して、標準の [ユーザー定義のツール バー] ダイアログ ボックスを表示できます。このダイアログ ボックスでは、ツール バー ボタンの非表示、表示、または再配置を指定できます。ダイアログ ボックスをプログラムで起動するには、**Customize** メソッドを使用します。ユーザーが [ツールボックスのカスタマイズ] ダイアログ ボックスを閉じたとき、**Change** イベントが発生します。

Visual Basic 2005 の **ToolStrip** コントロールには、これに相当するプロパティやメソッドはありません。独自のダイアログ ボックスを作成し、**ShowDialog** メソッドでそのダイアログ ボックスを表示する必要があります。詳細については、「ToolStrip のカスタマイズ サンプル」を参照してください。

DisabledImageList プロパティ、HotImageList プロパティ、ImageList プロパティ

Visual Basic 6.0 では、**ToolBar** コントロールの **DisabledImageList**、**HotImageList**、および **ImageList** の各プロパティを使用して、コントロールと **ImageList** コントロールを関連付けます。標準 (**ImageList**)、無効 (**DisabledImageList**)、および選択 (**HotImageList**) という異なる状態に、個々のイメージを関連付けることができます。

Visual Basic 2005 の **ToolStrip** コントロールには、これに相当するプロパティはありません。**ToolStripButton** オブジェクトには **Image** プロパティがあります。**ToolStripButton** の **Enabled** プロパティを **False** に設定すると、無効なイメージが自動的に描画されます。Visual Basic 6.0 における "選択" 状態の動作をエミュレートするには、実行時に異なるイメージを割り当てるか、**ToolStripButton** の **BackColor** プロパティを変更します。

Style プロパティ

Visual Basic 6.0 では、**ToolBar** コントロールの **Style** プロパティによって、テキストに関連してイメージが表示される場所が決定されます。このプロパティを **tbrTransparent** に設定すると、ホット トラッキングも有効になります。

Visual Basic 2005 の **ToolStrip** コントロールには、これに相当するプロパティはありません。イメージとテキストの関係は **ToolStripItem** の **TextImageAlign** プロパティを設定することによって制御されます。Visual Basic 6.0 とは異なり、項目ごとに異なる配置を指定できます。ホット トラッキングは直接サポートされていませんが、**MouseEnter** イベントと **MouseLeave** イベント内の **Image** プロパティを変更することによって、その動作をエミュレートできます。

Wrappable プロパティ

Visual Basic 6.0 では、**ToolBar** の **Wrappable** プロパティによって、ウィンドウのサイズが変更されたときにツール バー ボタンが自動的に折り返すかどうかが決定されます。

Visual Basic 2005 の **ToolStrip** コントロールには、これに直接相当するプロパティはありません。代わりに、ドロップダウン オーバーフロー メニューの使用がサポートされています。フォームの現在のサイズを前提に **ToolStrip** に割り当てられた領域を超える領域を必要とする **ToolStripItem** 要素を追加すると、**ToolStrip** 上に **ToolStripOverflowButton** が自動的に表示されます。**ToolStripOverflowButton** が表示されると、オーバーフローが有効になった項目がオーバーフローのドロップダウン メニューに移動します。**CanOverflow** プロパティを **False** に設定すると、この動作は無効化されます。

また、**Overflow** プロパティを設定することによって、個々の **ToolStripItem** 要素のオーバーフロー動作を指定できます。詳細については、「方法 : Windows フォームの ToolStrip オーバーフローを管理する」を参照してください。

RestoreToolBar メソッド、SaveToolBar メソッド

Visual Basic 6.0 では、**RestoreToolBar** メソッドと **SaveToolBar** メソッドを使用して、**ToolBar** の構成を管理します。**SaveToolBar** によって、構成が初期 (.ini) ファイルに保存されます。**RestoreToolBar** によって、カスタマイズされた構成が元の状態に戻されます。

Visual Basic 2005 の **ToolStrip** コントロールには、これに相当するメソッドはありません。**ToolStrip** コントロールの構成を管理するには、設定デザイナを使用します。詳細については、「アプリケーションの設定の管理」を参照してください。

ToolBar コントロールのプロパティとメソッドの対応関係

次の表は、Visual Basic 6.0 のプロパティおよびメソッドと、対応する Visual Basic 2005 のプロパティおよびメソッドの一覧を示しています。同じ名前で同じ動作を持つプロパティおよびメソッドは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Align	Dock
AllowCustomize	新規に実装されました。独自のダイアログ ボックスを提供する必要があります。
Appearance **BorderStyle**	新規に実装されました。詳細については、 「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
ButtonHeight	Height, Size (**ToolStripButton**)
Buttons	ToolStripItemCollection
ButtonWidth	**Height**, **Size** (**ToolStripButton**)
Container	Parent
Controls	**ToolStripItemCollection**
DataBindings	新規に実装されました。詳細については、「データ アクセス (Visual Basic 6.0 ユーザー向け)」を参照してください。
DisabledImageList	新規に実装されました。**ToolStripButton** オブジェクトの **Image** プロパティを使用します。
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	**Height** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

HelpContextID **HelpFile**	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HotImageList	新規に実装されました。**ToolStripButton** オブジェクトの **Image** プロパティを使用します。
HWnd	**Handle**
ImageList	新規に実装されました。**ToolStripButton** オブジェクトの **Image** プロパティを使用します。
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ**　Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDropMode	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm
ShowTips	ShowItemToolTips
Style	新規に実装されました。**ToolStripItem** の **TextImageAlign** プロパティを使用します。
TextAlignment	TextAlign (**ToolStripItem**)
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Value	Value **メモ：** Visual Basic 6.0 では、**Value** が変更されると、**Change** イベントが生成されます。Visual Basic 2005 では、**Change** イベントの代わりに ValueChanged イベントが使用されます。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ**　Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Wrappable	CanOverflow, Overflow

メソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Customize	新規に実装されました。独自のダイアログ ボックスを提供する必要があります。
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
RestoreToolbar **SaveToolbar**	新規に実装されました。アプリケーション設定機能を使用します。
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder:	BringToFront 関数または SendToBack 関数。

イベント

Visual Basic 6.0	Visual Basic 2005 での同等物
Change	新規に実装されました。[ツールボックスのカスタマイズ] ダイアログ ボックスはなくなりました。
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

アップグレード メモ

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、**ToolBar** コントロールは Windows フォームの **ToolStrip** コントロールにアップグレードされます。同等のプロパティ、メソッド、およびイベントがない場合、または動作に相違がある場合は、アップグレードに関する注意や警告がコードに追加されます。

参照

関連項目

ToolStrip コントロールの概要 (Windows フォーム)

その他の技術情報

ToolStrip コントロール (Windows フォーム)

TreeView コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **TreeView** コントロールは、Visual Basic 2005 では TreeView コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

SingleSel プロパティ

Visual Basic 6.0 では、**TreeView** コントロールの **SingleSel** プロパティは、ノードが選択されたときに、ノードを展開して子ノードを表示するかどうかを決定します。このプロパティを **True** に設定してノードを選択すると、ノードは展開され、それまでに選択されていたノードは縮小します。

Visual Basic 2005 では、**SingleSel** プロパティが存在しません。既定では、ノードをクリックしてもノードは展開されません (既定の状態は、Visual Basic 6.0 で **SingleSel** プロパティを **False** に設定した状態に相当します)。AfterSelect イベントを使用すると、ノードが選択されているかどうかを判断できます。次に、Expand メソッドまたは Collapse メソッドを使用すると、展開の動作を制御できます。

Image プロパティと SelectedImage プロパティ

Visual Basic 6.0 では、**Image** プロパティを割り当てなくても、**SelectedImage** プロパティを **TreeView** コントロールに割り当てることができます。

Visual Basic 2005 では、**TreeView** コントロールの **TreeNode** オブジェクトは、**Image** プロパティがなければ、**SelectedImage** プロパティを持つことができません。Visual Basic 6.0 の動作をエミュレートする必要がある場合は、空のイメージを **Image** プロパティに割り当てます。

その他の違い

また、データ連結、フォント処理、ドラッグ アンド ドロップ操作、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)」を参照してください。

TreeView コントロールを扱うコードの変更

次のコード例は、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

TreeView コントロール内の選択ノードの展開を扱うコードの変更

次のコードは、選択された **TreeView** ノードを展開する方法を示しています。

```
' Visual Basic 6.0
TreeView1.SingleSel = True
```

VB

```
' Visual Basic 2005
Private Sub TreeView1_AfterSelect(ByVal sender As Object, ByVal e As _
System.Windows.Forms.TreeViewEventArgs) Handles TreeView1.AfterSelect

    TreeView1.SelectedNode.Expand()
End Sub
```

TreeView コントロールへのノードの追加を扱うコードの変更

次のコードは、新しいノードを現在選択されているノードの子として追加する方法を示しています。

```
' Visual Basic 6.0
Dim nodX As Node
Set nodX = TreeView1.Nodes.Add(Node, tvwChild, , "New Node")
```

VB

```
' Visual Basic 2005
Dim nodX As TreeNode = New TreeNode("New Node")
TreeView1.SelectedNode.Nodes.Add(nodX)
```

TreeView コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Appearance	新規に実装されました。詳細については、「Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)」を参照してください。
BackColor	BackColor **メモ：** Visual Basic 2005 では、色は別の方法で処理されます。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Container	Parent
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Font Font **FontBold** **FontItalic** **FontName** **FontSize** **FontStrikethrough** **FontUnderline**	Font **メモ：** Visual Basic 2005 では、フォントは別の方法で処理されます。詳細については、「フォント オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Indentation	Indent
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
LineStyle	ShowRootLines
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDragMode **OLEDropMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm メソッド
Scroll	Scrollable
SingleSel	新規に実装されました。NodeMouseClick イベントを使用します。
Style	新規に実装されたプロパティ。0 または Standard は CheckedListBox コントロールに相当し、1 または Checkbox は **CheckedListBox** コントロールに相当します。
ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

名前	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

GetVisibleCount	VisibleCount プロパティ
HitTest	GetNodeAt, GetNodeAt
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
StartLabelEdit	BeginEdit
ZOrder	BringToFront メソッドまたは SendToBack メソッド

イベント

Visual Basic 6.0	**Visual Basic 2005 での同等物**
Collapse	BeforeCollapse
DblClick	DoubleClick
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Expand	AfterExpand
GotFocus	Enter
LostFocus	Leave
NodeCheck	AfterCheck
NodeClick	**NodeMouseClick**
OLECompleteDrag **OLEDragDrop** **OLEDragOver** **OLEGiveFeedback** **OLESetData** **OLEStartDrag**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Validate	Validating

アップグレード メモ

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、**TreeView** コントロールは Windows フォームの **TreeView** コントロールにアップグレードされます。同等のプロパティ、メソッド、およびイベントがない場合、または動作に相違がある場合は、アップグレードに関する注意や警告がコードに追加されます。

参照

処理手順

方法 : TreeView コントロールまたは ListView コントロール (Windows フォーム) にカスタム情報を追加する

方法 : Windows フォーム TreeView コントロールのすべてのノードを反復処理する

その他の技術情報

TreeView コントロール (Windows フォーム)

VBControlExtender オブジェクト (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 には、Visual Basic 6.0 の **VBControlExtender** オブジェクトに相当するものがなく、ActiveX コントロールをホストするためにコントロール エクステンダを使用する必要はなくなりました。

概念上の相違点

Visual Basic 6.0 では、ActiveX コントロールからホスト フォームのプロパティにアクセスするときに、**VBControlExtender** オブジェクトが使用されます。**VBControlExtender** オブジェクトは、**Add** メソッドを使用してコントロールを **Controls** コレクションに動的に追加するときに、主に使用されます。**VBControlExtender** オブジェクトが特にこの目的で役立つのは、開発者がプロパティ、イベント、およびメソッドの汎用的なセットを使用できるようになるからです。

Visual Basic 2005 では、ActiveX コントロールを Windows フォーム上で直接ホストできます。このため、エクステンダ オブジェクトがなくても、ActiveX コントロールからホスト フォームのプロパティにアクセスできるようになりました。ActiveX コントロールをプロジェクトに追加すると、COM 相互運用ラッパーが作成され、他のコントロールと同様に ActiveX コントロールを使用できるようになります。

アップグレード メモ

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、**VBControlExtender** オブジェクトのインスタンスは無視されます。各 ActiveX コントロールに対して COM 相互運用ラッパーが作成され、プロパティ、メソッド、およびイベントが同等の機能に変換されます。同等の機能がない場合は、コードにアップグレード警告が追加されます。

参照

処理手順

方法 : Windows フォームに ActiveX コントロールを追加する

関連項目

System.Windows.Forms Namespace

概念

Windows フォームで ActiveX コントロールをホストする場合の考慮事項

VScrollBar コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **VScrollBar** コントロールは、Visual Basic 2005 では Windows フォームの VScrollBar コントロールに置き換えられました。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Change イベント

Visual Basic 6.0 では、**VScrollBar** コントロールの **Value** プロパティが変更されると、**Change** イベントが生成されます。

Visual Basic 2005 では、**Change** イベントの代わりに ValueChanged イベントが生成されます。

Value プロパティ

Visual Basic 6.0 では、**VScrollBar** コントロールの **Scroll** イベントまたは **Change** イベントが発生すると、現在のスクロール値が **Value** プロパティに代入されます。

Visual Basic 2005 では、Scroll イベントが発生しても、その時点ではまだ **Value** プロパティの値は更新されません。Visual Basic 6.0 の動作をエミュレートする必要がある場合は、値を取得するヘルパー関数を作成できます。詳細については、「コードがイベントからプロシージャに変更されている」を参照してください。

LargeChange プロパティ

Visual Basic 6.0 では、スクロール ボックスとスクロール矢印の間にある領域がクリックされたときにスクロール バー コントロールの **Value** プロパティの値をどの程度変更するかは、**LargeChange** プロパティの値によって決定されます。

Visual Basic 2005 では **LargeChange** プロパティの既定値は 10 ですが、Visual Basic 6.0 では 1 です。

Max プロパティ

Visual Basic 6.0 では、**Max** プロパティによって、スクロール バー コントロールの最大 **Value** プロパティ設定が決定されます。

Visual Basic 2005 では **Max** プロパティは Maximum プロパティに置き換えられています。既定値は 100 ですが、Visual Basic 6.0 では 32767 でした。

Min プロパティ

Visual Basic 6.0 では、**Min** プロパティによって、スクロール バー コントロールの最小 **Value** プロパティ設定が決定されます。**Min** プロパティは、**Max** プロパティより大きな値に設定できます。

Visual Basic 2005 では、**Min** プロパティは Minimum プロパティに置き換えられました。このプロパティの値は、常に **Maximum** プロパティより小さい必要があります。

その他の違い

加えて、データ連結、フォント処理、ドラッグ アンド ドロップ、ヘルプ サポートなど、すべてのコントロールに当てはまる概念上の相違が数多くあります。詳細については、「Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)」を参照してください。

VScrollBar コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。必要に応じて、プロパティまたはメソッドの下に定数が示されています。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

必要に応じて、動作の違いを説明するトピックへのリンクが示されています。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	Visual Basic 2005 での同等物
Container	Parent
DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。

Height	Height **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	**Handle**
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
LargeChange	LargeChange **メモ：** Visual Basic 6.0 での既定値は 1 でしたが、Visual Basic 2005 での既定値は 10 です。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Max	Maximum **メモ：** Visual Basic 6.0 での既定値は 32767 でしたが、Visual Basic 2005 での既定値は 100 です。
Min	Minimum **メモ：** Visual Basic 6.0 では、**Min** は **Max** より大きな値に設定できますが、Visual Basic 2005 ではできません。
MouseIcon	新規に実装されました。詳細については、「カスタム MousePointer を設定できない」を参照してください。
MousePointer	Cursor 定数の一覧については、「MousePointer (Visual Basic 6.0 ユーザー向け)」を参照してください。
Parent	FindForm
RightToLeft: **True** **False**	RightToLeft Yes 列挙値 No 列挙値

Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Value	Value **メモ：** Visual Basic 6.0 では、**Value** が変更されると、**Change** イベントが生成されます。Visual Basic 2005 では、**Change** イベントの代わりに **ValueChanged** イベントが使用されます。
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	Visual Basic 2005 での同等物
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
SetFocus	Focus
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder: **0 – vbBringToFront** **1 - vbSendToBack**	BringToFront 関数または SendToBack 関数 **BringToFront** **SendToBack**

イベント

Visual Basic 6.0	Visual Basic 2005 での同等物
Change	**ValueChanged** **メモ：** **Change** イベントと **Scroll** イベントの動作は、Visual Basic 2005 では変更されています。詳細については、「コードがイベントからプロシージャに変更されている」を参照してください。

DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
LostFocus	Leave
Scroll	**Scroll** **メモ：** **Change** イベントと **Scroll** イベントの動作は、Visual Basic 2005 では変更されています。詳細については、「コードがイベントからプロシージャに変更されている」を参照してください。
Validate	Validating

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、HScrollBar コントロールまたは **VScrollBar** コントロールの **Scroll** イベント ハンドラのコードは、イベントの順序の違いに従ってプロシージャに変更されます。詳細については、「コードがイベントからプロシージャに変更されている」を参照してください。

参照

関連項目

HScrollBar コントロールと VScrollBar コントロールの概要 (Windows フォーム)

WebBrowser コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **WebBrowser** コントロールは、Visual Basic 2005 では Windows フォームの **WebBrowser** コントロールに置き換えられています。プロパティ、メソッド、イベント、および定数の中には、名称が異なるものや、動作の異なるものもあります。

概念の違い

Microsoft Internet Controls または Shdocvw.dll とも呼ばれる Visual Basic 6.0 の **WebBrowser** コントロールは、アプリケーション内で Internet Explorer をホストするための ActiveX コントロールです。

Visual Basic 2005 の **WebBrowser** コントロールは、**WebBrowser** ActiveX コントロール用のマネージ ラッパーを提供します。このコントロールを使用すると、Windows フォーム クライアント アプリケーションで、Web ページを表示できます。**WebBrowser** コントロールを使用すると、アプリケーションで Internet Explorer の Web ブラウザ同様の機能を利用できます。または、Internet Explorer の既定の機能を無効にして、コントロールを単純な HTML ドキュメント ビューアとして使用することもできます。また、このコントロールを使用してフォームに DHTML ベースのユーザー インターフェイス要素を追加することによって、これらの機能が **WebBrowser** コントロールでホストされているという事実を隠すこともできます。この方法によって、Web コントロールと Windows フォーム コントロールとを 1 つのアプリケーションでシームレスに組み合わせることができます。

Internet Explorer オブジェクト

Visual Basic 6.0 の Microsoft Internet Controls は、**WebBrowser** コントロールと **Internet Explorer** オブジェクトという 2 つのコンポーネントで構成されており、OLE オートメーションを通じて Internet Explorer のインスタンスを制御できます。

Visual Basic 2005 には、**Internet Explorer** オブジェクトに相当する機能はありません。関連するプロパティ、メソッド、およびイベントも存在しません。OLE オートメーションはサポートされておらず、VBScript などのスクリプト言語を使用して Internet Explorer の外部インスタンスを制御します。

Document プロパティ

Visual Basic 6.0 では、**Document** プロパティは HTML ドキュメントの名前が含まれる文字列を返します。Visual Basic 2005 では、HTML ドキュメント自体を返します。

Refresh メソッドと Refresh2 メソッド

Visual Basic 6.0 の **Refresh2** メソッドでは、更新レベルを指定するパラメータが必要です。Visual Basic 2005 の **Refresh** メソッドでは、更新レベルを指定する省略可能なパラメータを使用できます。

WebBrowser コントロールのプロパティ、メソッド、およびイベントの同等物

次の表は、Visual Basic 6.0 のプロパティ、メソッド、およびイベントと、対応する Visual Basic 2005 のプロパティ、メソッド、およびイベントを示しています。同じ名前で同じ動作を持つプロパティ、メソッド、およびイベントは、一覧に含まれていません。特に記述されていない限り、すべての Visual Basic 2005 列挙型は System.Windows.Forms 名前空間に割り当てられます。

この表では、動作の相違点について説明するトピックへのリンクを示します。Visual Basic 2005 に直接対応するものがない場合は、代わりの項目を示すトピックへのリンクを示します。

プロパティ

Visual Basic 6.0	対応する Visual Basic 2005 のイベント
AddressBar	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
Application	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
Busy	IsBusy
Container	Parent
Document	Document **メモ：** Visual Basic 6.0 では、**Document** プロパティは HTML ドキュメントの名前が含まれる文字列を返します。Visual Basic 2005 では、HTML ドキュメント自体を返します。

DragIcon **DragMode**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
FullName	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
Height	Height, Size **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
HelpContextID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
HWnd	Handle
Index	新規に実装されました。詳細については、「コントロール配列 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Left	Left **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
LocationName **LocationURL**	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
MenuBar	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
Offline	IsOffline
Parent	FindForm メソッド
Path	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
RegisterAsBrowser	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
RegisterAsDropTarget	AllowWebBrowserDrop
Resizable	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
Silent	ScriptErrorsSuppressed
StatusBar **StatusText**	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
TheaterMode	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
ToolBar	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。

ToolTipText	ToolTip コンポーネント 詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Top	Top **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
TopLevelContainer	新規に実装されました。これは、**Internet Explorer** オブジェクトのプロパティでした。
Type	DocumentType
WhatsThisHelpID	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
Width	Width, **Size** **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。

メソッド

Visual Basic 6.0	**対応する Visual Basic 2005 のイベント**
ClientToWindow	PointToClient
Drag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
ExecWB	新規に実装されました。これは、**Internet Explorer** オブジェクトのメソッドでした。
GetProperty	新規に実装されました。これは、**Internet Explorer** オブジェクトのメソッドでした。
Move	SetBounds **メモ：** Visual Basic 2005 では、座標は別の方法で処理されます。詳細については、「座標系 (Visual Basic 6.0 ユーザー向け)」を参照してください。
Navigate2	新規に実装されました。これは、**Internet Explorer** オブジェクトのメソッドでした。
OLEDrag	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
PutProperty	新規に実装されました。これは、**Internet Explorer** オブジェクトのメソッドでした。
QueryStatusWB	新規に実装されました。これは、**Internet Explorer** オブジェクトのメソッドでした。

Quit	新規に実装されました。これは、**Internet Explorer** オブジェクトのメソッドでした。
Refresh **Refresh2**	Refresh **メモ：** Visual Basic 6.0 の **Refresh2** メソッドでは、更新レベルを指定するパラメータが必要でした。Visual Basic 2005 の **Refresh** メソッドでは、更新レベルを指定する省略可能なパラメータを使用できます。
SetFocus	Focus
ShowBrowserBar	新規に実装されました。これは、**Internet Explorer** オブジェクトのメソッドでした。
ShowWhatsThis	新規に実装されました。詳細については、「ヘルプ サポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
ZOrder	BringToFront 関数または SendToBack 関数

イベント

Visual Basic 6.0	**対応する Visual Basic 2005 のイベント**
BeforeNavigate **BeforeNavigate2**	Navigating
ClientToHostWindow	新規に実装されました。これは、**Internet Explorer** オブジェクトのイベントでした。
CommandStateChanged	CanGoBackChanged CanGoForwardChanged
DblClick	DoubleClick
DocumentComplete **DownloadBegin** **DownloadComplete**	DocumentCompleted
DragDrop **DragOver**	新規に実装されました。詳細については、「ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)」を参照してください。
GotFocus	Enter
LostFocus	Leave
NavigateComplete **NavigateComplete2**	Navigated
NavigateError	新規に実装されました。これは、Internet Explorer オブジェクトのイベントでした。
NewWindow **NewWindow2** **NewWindow3**	NewWindow

OnFullScreen **OnMenuBar** **OnQuit** **OnStatusBar** **OnTheaterMode** **OnToolBar** **OnVisible**	新規に実装されました。これは、**Internet Explorer** オブジェクトのイベントでした。
PrintTemplateInstantiation **PrintTemplateTearDown** **PrivacyImpactedStateChange**	新規に実装されました。これは、**Internet Explorer** オブジェクトのイベントでした。
ProgressChange	ProgressChanged
PropertyChange	新規に実装されました。これは、**Internet Explorer** オブジェクトのイベントでした。
SetSecureLockIcon	EncryptionLevelChanged
StatusTextChange	StatusTextChanged
TitleChange	DocumentTitleChanged
UpdatePageStatus	新規に実装されました。これは、**Internet Explorer** オブジェクトのイベントでした。
Validate	Validating
WindowClosing **WindowSetHeight** **WindowSetLeft** **WindowSetResizable** **WindowSetTop** **WindowSetWidth**	新規に実装されました。これは、**Internet Explorer** オブジェクトのイベントでした。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**WebBrowser** コントロールは Windows フォームの **WebBrowser** コントロールにアップグレードされます。プロパティ、メソッド、およびイベントは対応するメンバにアップグレードされます。動作が異なる場合は、アップグレードに関するコメントがコードに挿入されます。

参照

処理手順

方法 : .NET Compact Framework で WebBrowser コントロールを使用する

関連項目

WebBrowser コントロールの概要

その他の技術情報

WebBrowser コントロール (Windows フォーム)

コントロールのプロパティ、メソッド、およびイベントの変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 のコントロールおよびオブジェクトに含まれるプロパティ、メソッド、およびイベントの多くは、Visual Basic 2005 では別の動作のプロパティ、メソッド、およびイベントに置き換えられています。

以下のトピックでは、主要なプロパティ、メソッド、およびイベントに対する変更点について説明します。

このセクションの内容

Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)

各種コントロールの **Appearance** プロパティと **BorderStyle** プロパティに加えられた変更点について説明します。

ComboBox コントロールの Change イベント (Visual Basic 6.0 ユーザー向け)

ComboBox コントロールの **Change** イベントの動作に加えられた変更点について説明します。

座標系 (Visual Basic 6.0 ユーザー向け)

Height プロパティ、**Width** プロパティ、**Move** メソッド、および **ScaleMode** 設定に加えられた変更点について説明します。

フォームのイベント (Visual Basic 6.0 ユーザー向け)

各種のフォーム レベルのイベントに加えられた変更点について説明します。

MaxLength プロパティ (Visual Basic 6.0 ユーザー向け)

TextBox コントロールの **MaxLength** プロパティに加えられた変更点について説明します。

MousePointer (Visual Basic 6.0 ユーザー向け)

MousePointer のプロパティおよび定数に加えられた変更点について説明します。

Moveable プロパティ (Visual Basic 6.0 ユーザー向け)

フォームの **Moveable** プロパティに加えられた変更点について説明します。

Style プロパティ (Visual Basic 6.0 ユーザー向け)

各種コントロールの **Style**、**Picture**、**Disabled Picture**、および **DownPicture** の各プロパティに加えられた変更点について説明します。

TopIndex プロパティおよび Scroll イベント (Visual Basic 6.0 ユーザー向け)

ComboBox コントロールおよび **ListBox** コントロールの **TopIndex** プロパティおよび **Scroll** イベントに加えられた変更点について説明します。

関連するセクション

Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)

フォームとコントロールを操作する上での一般的な概念の相違について説明します。

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 のすべてのコントロールおよびオブジェクトを、Windows フォームで対応する機能と比較します。

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 のすべての定数と、相当する Visual Basic 2005 の定数を示します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 における Visual Basic 言語の相違について説明します。

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic 6.0 と Visual Basic 2005 の比較に関するすべてのトピックの入り口です。

Appearance プロパティおよび BorderStyle プロパティ (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、**Appearance** プロパティと **BorderStyle** プロパティは、コントロールの外観を制御するために使用します。

概念の違い

Visual Basic 6.0 の **Frame**、**Image**、**Label**、**PictureBox**、および **TextBox** の各コントロールでは、**Appearance** プロパティと **BorderStyle** プロパティの両方がサポートされます。2 つのプロパティを組み合わせて設定することにより、境界線のないコントロール、境界線が単線のコントロール、または境界線が 3 次元のコントロールを作成できます。

Visual Basic 2005 では、これらのコントロールに **Appearance** プロパティが存在しません。その機能は、外観を一度に設定する `BorderStyle` プロパティに組み込まれています。

Visual Basic 6.0 の **CommandButton**、**ListBox**、**ComboBox** の各コントロールでは、**Appearance** プロパティだけがサポートされます。

Visual Basic 2005 の Button コントロールには、**Appearance** プロパティに相当する FlatStyle プロパティがあります。ListBox コントロールは、BorderStyle プロパティを FixedSingle に設定することでフラットに見せることができます。**ComboBox** コントロールは、実際には Visual Basic 6.0 でフラットな外観をサポートしていなかったため、相当する機能はありません。

メモ :

Visual Basic 6.0 フォームには、複数の値をサポートする **BorderStyle** プロパティもあります。これは、Visual Basic 2005 では、FormBorderStyle プロパティに置き換えられています。

Appearance プロパティと BorderStyle プロパティを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

コントロールの外観を変更するコードの変更

次のコード例は、実行時にコントロールの外観を変更するためにプロパティを設定する方法を示します。

```
' Visual Basic 6.0
' Give a TextBox a flat look with no border.
Text1.Appearance = 0
Text1.BorderStyle = 0
' Give a TextBox a three-dimensional appearance.
Text1.Appearance = 1
Text1.BorderStyle = 1
' Give a CommandButton a flat look
Command1.Appearance = 0
' Give a ListBox a flat look
List1.Appearance = 0
```

VB

```
' Visual Basic 2005
' Give a TextBox a flat look with a single border.
TextBox1.BorderStyle = BorderStyle.None
' Give a TextBox a three-dimensional appearance.
TextBox2.BorderStyle = BorderStyle.Fixed3D
' Give a Button a flat look
Button1.FlatStyle = FlatStyle.Flat
' Give a ListBox a flat look
ListBox1.BorderStyle = BorderStyle.FixedSingle
```

Appearance および BorderStyle の対応表

Visual Basic 6.0 の **Appearance** 定数および **BorderStyle** 定数に相当する Visual Basic 2005 の列挙を次の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物

BorderStyle = 0 – None	None
Appearance = 0 – Flat **BorderStyle = 1 – Fixed Single**	**FixedSingle**
Appearance = 1 – 3D **BorderStyle = 1 – Fixed Single**	Fixed3D

アップグレード メモ

Visual Basic 6.0 アプリケーションをアップグレード ウィザードを使ってアップグレードすると、**Frame** コントロールは Visual Basic 2005 の GroupBox コントロールまたは Panel コントロールにアップグレードされます。**Appearance** と **BorderStyle** の対応付けは、**Panel** にアップグレードされる **Frame** コントロールにのみ適用されます。**GroupBox** コントロールには、**BorderStyle** プロパティはありません。詳細については、「Frame コントロール (Visual Basic 6.0 ユーザー向け)」を参照してください。

ComboBox、**CommandButton**、および **ListBox** の各コントロールのアップグレード時には、**Appearance** プロパティは対応付けられません。**Appearance** プロパティがコードで設定されている場合、アップグレードの警告がコードに追加されます。

参照

概念

ComboBox コントロール (Visual Basic 6.0 ユーザー向け)
CommandButton コントロール (Visual Basic 6.0 ユーザー向け)
Frame コントロール (Visual Basic 6.0 ユーザー向け)
Label コントロール (Visual Basic 6.0 ユーザー向け)
PictureBox コントロール (Visual Basic 6.0 ユーザー向け)
TextBox コントロール (Visual Basic 6.0 ユーザー向け)
User コントロール (Visual Basic 6.0 ユーザー向け)

ComboBox コントロールの Change イベント (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、コントロールのテキスト入力部分のテキストが変更されると、**ComboBox** コントロールの **Change** イベントが発生します。コントロールのリスト部分でアイテムが選択されたときには、このイベントは発生しません。また、アイテムのテキストをプログラムで変更した場合にも **Change** イベントは発生しません。

プロジェクトを Visual Basic 2005 に移行すると、**ComboBox** コントロールの **Change** イベントは Visual Basic 2005 の ComboBox コントロールの TextChanged イベントに対応付けられます。**TextChanged** イベントの動作は、**Change** イベントとは異なります。この相異によって、コードで意図した結果が得られない場合があります。

Visual Basic 2005 の **TextChanged** イベントは、テキストが変更されるたびに発生します。テキストが変更されるのは、次のようなときです。

- テキスト入力部分が変更された。
- 一覧でアイテムが選択された。
- 一覧のアイテムがプログラムで変更された。
- Add メソッドが呼び出された。

動作の違いを次の例に示します。

```
' Visual Basic 6.0
Private Sub Form_Load()
   ' Does not raise the Change event.
   Combo1.AddItem "A"
End Sub
Private Sub Form_Click()
   ' Does not raise the Change event.
   Combo1.List(0) = "B"
End If
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load()
   ' Raises the TextChanged event.
   ComboBox1.Items.Add("A")
End Sub
Private Sub Form1_Click(ByVal sender As System.Object, ByVal _
e As System.EventArgs)
   ' Uses the SetItemString method from the VB6 compatibility library;
   ' there is no equivalent method in Visual Basic 2005.
   ' Raises the TextChanged event.
   Microsoft.VisualBasic.Compatibility.VB6. _
      SetItemString(ComboBox1, ComboBox1.Items.Count, "B")
End Sub
```

次の操作

- **TextChanged** イベント プロシージャにブレークポイントを設定し、コードを実行してイベントが発生するかどうかを確認します。必要に応じてコードを変更します。

参照

概念

ComboBox コントロール (Visual Basic 6.0 ユーザー向け)

座標系 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、フォームおよびコントロールの座標は Visual Basic 6.0 とは異なる方法で表現されます。フォームのサイズ変更や移動の方法も異なります。

概念の違い

Visual Basic 6.0 では、フォームとコントロールの座標は twip で表現されます。Visual Basic 2005 では、座標はピクセルで表現されます。

Height プロパティおよび Width プロパティ

Visual Basic 6.0 では、**Height** プロパティと **Width** プロパティが、フォームまたはコントロールのサイズ変更に使用されます。Visual Basic 2005 では、`Size` プロパティを使って、高さと幅を一度に変更することもできます。

Move メソッド

Visual Basic 6.0 では、フォームまたはコントロールの位置を実行時に変更するために **Move** メソッドを使用します。Visual Basic 2005 では、**Move** は SetBounds メソッドに置き換えられ、座標はピクセルで表現されます。

ScaleMode プロパティ

Visual Basic 6.0 では、**ScaleMode** プロパティを使って別の座標系を定義することもできます。Visual Basic 2005 では、サポートされる座標系はピクセルだけです。**ScaleMode** プロパティおよびそれと関連するプロパティやメソッド (**Scale**、**ScaleHeight**、**ScaleLeft**、**ScaleTop**、**ScaleWidth**、**ScaleX**、**ScaleY**) は Visual Basic 2005 ではサポートされません。詳細については、「ScaleMode がサポートされていない」を参照してください。

座標を扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

フォームのサイズ変更

実行時にフォームのサイズを変更する方法の例を次に示します。

```
' Visual Basic 6.0
' Measurements are in twips.
Me.Width = 8000
Me.Height = 6000
```

VB

```
' Visual Basic 2005
' Measurements are in pixels.
Me.Size = New System.Drawing.Size(640, 480)
```

コントロールの移動

実行時にコントロールを移動する方法の例を次に示します。

```
' Visual Basic 6.0
' Move and retain original size.
CommandButton2.Move 2000, 1000
' Move and resize to 1200 by 800 twips.
CommandButton1.Move 0, 0, 1200, 800
```

VB

```
' Visual Basic 2005
' Move and retain original size.
Button2.SetBounds(20, 10, 0, 0, BoundsSpecified.X Or BoundsSpecified.Y)
' Move and resize to 120 by 80 pixels.
Button1.SetBounds(0, 0, 120, 80)
```

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、フォームおよびコントロールの Height プロパティと Width プロパティの値は自動的にピクセルに変換されます。

元のアプリケーションで **ScaleMode** プロパティを使用していた場合、アップグレード ツールは、**ScaleMode** プロパティのデザイン時設定が twip であると仮定します。それ以外の場合は、正しく変換されないため、修正が必要です。詳細については、「ScaleMode がサポートされていない」を参照してください。

参照

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

コントロールのプロパティ、メソッド、およびイベントの変更点 (Visual Basic 6.0 ユーザー向け)

フォームのイベント (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 では、フォーム イベントの動作に少し違いがあります。

概念の違い

Initialize イベント

Visual Basic 6.0 では、フォームを読み込む前にコードを実行する場合、**Initialize** イベントを使用します。

Visual Basic 2005 では、次の例に示すように、フォーム コンストラクタ (**Sub New()**) 内で **InitializeComponent()** の呼び出しの後に初期化コードを追加する必要があります。

```
' Visual Basic 6.0
Private Sub Form_Initialize()
    MsgBox("The form is loading")
End Sub
```

VB

```
' Visual Basic .NET
Public Sub New()
    ' This call is required by the Windows Form Designer.
    InitializeComponent()
    ' Add any initialization after the InitializeComponent() call.
    MsgBox("The form is loading")
End Sub
```

メモ :

一般に **Initialize** イベントは、フォームの読み込み中に情報を表示する "スプラッシュ" フォームを表示するために使用されます。Visual Basic 2005 アプリケーションには **SplashScreen** プロパティがあり、これを使ってアプリケーションの起動時に自動的にフォームを表示できます。詳細については、「方法 : アプリケーションのスプラッシュ スクリーンを指定する (Visual Basic)」を参照してください。

Terminate イベント

Visual Basic 6.0 では、フォームをアンロードした後にコードを実行する場合、**Terminate** イベントを使用します。

Visual Basic 2005 では、**Terminate** イベントはサポートされていません。**Dispose** メソッド内で、**MyBase.Dispose()** の呼び出しの前に終了コードを実行する必要があります。

メモ :

Dispose メソッドは、アプリケーションのメイン フォームに対しては自動的に呼び出されます。それ以外のフォームに対しては、明示的に呼び出す必要があります。

違いを示す例を次に示します。

```
' Visual Basic 6.0
Private Sub Form_Terminate()
    MsgBox "The form was terminated"
End Sub
```

VB

```
' Visual Basic .NET
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing AndAlso components IsNot Nothing Then
        MsgBox("The form was terminated")
```

```
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub
```

Unload イベント

Visual Basic 6.0 では、**Unload** イベントに Cancel 引数があります。Visual Basic 2005 では、このイベントは **Closed** イベントに置き換えられ、Cancel 引数はなくなりました。アンロードを途中でキャンセルする場合は、**Closing** イベントを使用します。

MDI フォームのマウス イベント

Visual Basic 6.0 では、MDI フォームでマウス イベントがサポートされています。Visual Basic 2005 では、MDI フォーム上にマウス イベントを受け取るクライアント領域がないため、MDI フォームでは **Click**、**MouseDown**、**MouseMove**、**MouseUp** の各イベントはサポートされません。

参照

関連項目

Form Constructor
Form.Closing Event

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

MaxLength プロパティ (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、**TextBox** コントロールの **MaxLength** プロパティを使用して、テキスト ボックスに入力または表示できる文字列の長さを絶対的に制御できます。プログラムでそれより長い文字列も入力できますが、その場合は、**MaxLength** プロパティに設定された長さに自動的に切り詰められます。

Visual Basic 2005 では、プログラムで入力された文字列の長さが **MaxLength** プロパティに優先します。

動作の違いを示すコード例を次に示します。

```
' Visual Basic 6.0
Text1.MaxLength = 5
Text1.Text = "Longer than five"
MsgBox Text1.Text       ' Displays "Longe".
```

VB

```
' Visual Basic 2005
Text1.MaxLength = 5
Text1.Text = "Longer than five"
MsgBox(Text1.Text)       ' Displays "Longer than five".
```

次の操作

- アプリケーションのデザインを見直して、テキスト ボックス内のテキストを **MaxLength** プロパティで設定された長さに常に制限してよいかどうかを確認します。
    - 長さを常に制限する場合は、**Text** プロパティを設定しているコードをすべて探し、それぞれに対して、次の例のような文字列を切り詰めるコードを追加します。

      VB

      ```
      ' Old code
      Text1.Text = SomeString

      ' Replacement code
      If Len(SomeString) > Text1.MaxLength Then
        SomeString = Microsoft.VisualBasic.Left(SomeString, Text1.MaxLength)
      End If

      Text1.Text = SomeString
      ```

    - それ以外の場合は、コードをそのままにしておきます。ユーザーは **MaxLength** の設定による制限を受けますが、プログラムでは **MaxLength** の設定値より長い文字列も代入できます。

参照

概念

TextBox コントロール (Visual Basic 6.0 ユーザー向け)

MousePointer (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、Visual Basic 6.0 の **MousePointer** プロパティは Cursor プロパティに置き換えられ、**MousePointer** 定数の名前も変更されています。

概念の違い

Visual Basic 6.0 では、フォームまたはコントロールの **MousePointer** プロパティを使って、マウスのカーソルがフォームまたはコントロールの上を通過するときに外観を変更します。

Visual Basic 2005 では、**Cursor** プロパティが **MousePointer** と同じ機能を持ちます。

カスタムの MousePointer

Visual Basic 6.0 では、アイコン (.ico) ファイルまたはカーソル (.cur) ファイルをカスタムの **MousePointer** として使用できます。

Visual Basic 2005 では、カーソル ファイルだけがサポートされています。実行時にカスタムのカーソルを設定するには、カーソル ファイルを **Cursor** プロパティに割り当てます。

vbIconPointer 定数

Visual Basic 6.0 には、カーソルをアイコンのシンボル (正方形に囲まれた小さい正方形) に変更するための **MousePointer** 定数 **vbIconPointer** があります。この定数は、レガシとして使用する目的でのみ用意されています。新しいオペレーティング システムでは、この定数は効力を持たず、既定のカーソルが表示されます。Visual Basic 2005 には、これに相当するものはありません。

MousePointer プロパティを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

カーソルの外観の変更

次のコード例は、実行時にマウスのカーソルがテキスト ボックス コントロールの上を通過したときに、カーソルの外観を既定の矢印カーソルから砂時計カーソルに変更する方法を示しています。Visual Basic 6.0 のコード例では、**MousePointer** プロパティを設定します。Visual Basic 2005 のコード例では、**Cursor** プロパティが **MousePointer** に相当します。Visual Basic 2005 のコード例では、新しい MouseEnter イベントも使用します。

```
' Visual Basic 6.0
Private Sub Text1_MouseMove(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
    Text1.MousePointer = vbHourGlass
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TextBox1_MouseEnter(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles TextBox1.MouseEnter
    TextBox1.Cursor = System.Windows.Forms.Cursors.WaitCursor
End Sub
```

カスタムのカーソルの表示

次のコード例では、カーソルがテキスト ボックス コントロールの上を通過したときに、手の形のカーソルを表示する方法を示しています。Visual Basic 6.0 のコード例では、**MouseIcon** プロパティと **MousePointer** プロパティを設定します。Visual Basic 2005 のコード例では、**Cursor** プロパティを Cursor オブジェクトの新しいインスタンスに設定します。

```
' Visual Basic 6.0
Private Sub Text1_MouseMove(Button As Integer, Shift As Integer, _
X As Single, Y As Single)
    Text1.MouseIcon = LoadPicture("C:\Windows\Cursors\hmove.cur")
    Text1.MousePointer = vbHourGlass
End Sub
```

VB

```
' Visual Basic 2005
Private Sub TextBox2_MouseEnter(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles TextBox2.MouseEnter
    TextBox2.Cursor = New System.Windows.Forms.Cursor _
    ("C:\mypath\mycursor.cur")
End Sub
```

MousePointer 定数の対応表

次の表は、Visual Basic 6.0 の定数と Visual Basic 2005 でそれらに対応するものを示しています。

Visual Basic 6.0	Visual Basic 2005 で対応するもの
0 – vbDefault	Default
1 – vbArrow	Arrow
2 – vbCrossHair	Cross
3 – vbIBeam	IBeam
4 – vbIconPointer	互換性のために残されています — **Default** で置き換えられています。
5 – vbSizePointer	SizeAll
6 – vbSizeNESW	SizeNESW
7 – vbSizeNS	SizeNS
8 – vbSizeNWSE	SizeNWSE
9 – vbSizeWE	SizeWE
10 – vbUpArrow	UpArrow
11 – vbHourGlass	WaitCursor
12 – vbNoDrop	No
13 – vbArrowHourGlass	AppStarting
14 – vbArrowQuestion	Help
15 – vbSizeAll	**SizeAll**
99 – vbCustom	対応要素なし — 詳細については、「カスタム MousePointer を設定できない」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**MousePointer** プロパティは **Cursor** プロパティに置き換えられます。**MousePointer** 定数を使用するコードは、Visual Basic 2005 の Cursors 列挙を使用するように変更されます。

Visual Basic 6.0 アプリケーションでカスタムの **MousePointer** を使用している場合は、デザイン時の設定またはカスタムの **MousePointer** コードはアップグレードされず、警告が発行されます。

また、Visual Basic 6.0 アプリケーションでデザイン時または実行時に **MousePointer** を **vbIconPointer** 定数に設定している場合は、アップグレード中に既定のカーソルに置き換えられます。

参照

関連項目

カスタム MousePointer を設定できない

その他の技術情報

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

コントロールのプロパティ、メソッド、およびイベントの変更点 (Visual Basic 6.0 ユーザー向け)

Moveable プロパティ (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 フォームの **Moveable** プロパティに相当するものは、Visual Basic 2005 にありません。

概念の違い

Visual Basic 6.0 では、フォームの **Moveable** プロパティを **False** に設定すると、ユーザーが実行時にフォームを移動できなくなります。Visual Basic 2005 の Windows フォームには、これに相当するプロパティはありません。

これは一般にユーザー インターフェイスのデザインとして望ましくないと考えられていますが、Visual Basic 2005 でも FormBorderStyle プロパティを None に設定して ControlBox プロパティを **False** に設定すると、同じ動作を実現できます。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**Moveable** プロパティへのデザイン時または実行時の参照はアップグレードされず、警告のコメントがコードに追加されます。

参照

関連項目

ControlBox

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

MDIForm オブジェクト (Visual Basic 6.0 ユーザー向け)

Style プロパティ (Visual Basic 6.0 ユーザー向け)

CommandButton、**CheckBox**、および **OptionButton** コントロールに画像を表示する手順は、Visual Basic 6.0 と Visual Basic 2005 で異なります。

概念の違い

Visual Basic 6.0 では、**CommandButton**、**CheckBox**、または **OptionButton** コントロールの **Style** プロパティを `1 - Graphical` に設定すると、コントロールに画像を表示できます。**Picture**、**DownPicture**、および **DisabledPicture** の各プロパティを使用して、状態の変化に応じて表示するイメージを割り当てることができます。たとえば、**CheckBox** コントロールがオンになると **DownPicture** イメージが表示され、**CheckBox** コントロールがオフになると **DisabledPicture** イメージが表示されます。

Visual Basic 2005 では、**Style**、**Picture**、**DownPicture**、および **DisabledPicture** の各プロパティはサポートされません。**Style** プロパティは不要になりました。画像を Image プロパティに割り当てると、**Style** プロパティを `Graphical` に設定したのと同じ効果が得られます。**Picture** プロパティは、**Image** プロパティに置き換えられています。**DownPicture** プロパティと **DisabledPicture** プロパティと同じような機能は、ImageList コントロールに複数の画像を含めることで実現できます。

アップグレード メモ

Style プロパティが `1 - Graphical` に設定されたアプリケーションをアップグレードすると、アップグレードされたコントロールの **Appearance** プロパティが **Button** に設定され、デザイン時に **Picture** プロパティに割り当てた画像は、アップグレードされたコントロールの **Image** プロパティに割り当てられます。

デザイン時または実行時に **DownPicture** プロパティまたは **DisabledPicture** プロパティを設定していた場合は、アップグレードしたアプリケーションを変更して **ImageList** コントロールを使う必要があります。詳細については、
「方法 : アップグレードしたアプリケーションで Visual Basic 6.0 の 3 ステート コントロールをエミュレートする」を参照してください。

参照

処理手順

方法 : アップグレードしたアプリケーションで Visual Basic 6.0 の 3 ステート コントロールをエミュレートする

概念

CheckBox コントロール (Visual Basic 6.0 ユーザー向け)
CommandButton コントロール (Visual Basic 6.0 ユーザー向け)
OptionButton コントロール (Visual Basic 6.0 ユーザー向け)
OptionButton コントロール (Visual Basic 6.0 ユーザー向け)

TopIndex プロパティおよび Scroll イベント (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、ComboBox コントロールの **TopIndex** プロパティと、ListBox コントロールまたは **ComboBox** コントロールの **Scroll** イベントはサポートされなくなりました。

概念の違い

TopIndex プロパティ

Visual Basic 6.0 では、**TopIndex** プロパティを使用して、**ComboBox** コントロールや **ListBox** コントロールでどのアイテムを一番上に表示するかを指定する値を返したり、設定したりできます。このプロパティは一般に、アイテムを選択せずにリストをスクロールするために使用されます。

Visual Basic 2005 では、**ComboBox** コントロールで **TopIndex** プロパティはサポートされなくなりました。**Style** プロパティを 1 - SimpleCombo に設定した場合以外は、**TopIndex** プロパティを設定しても目に見える違いはないので、ほとんどの場合は問題ありません。**Style** プロパティを 1 - SimpleCombo に設定している場合は、**ListBox** コントロールと TextBox コントロールを使用して **ComboBox** の **SimpleCombo** 動作をエミュレートできます。**ListBox** コントロールは引き続き TopIndex プロパティをサポートしています。

Scroll イベント

Visual Basic 6.0 の **Scroll** イベントは、**TopIndex** プロパティと組み合わせて、リストのスクロール時にアクションを実行するために使用します。Visual Basic 2005 では、**Scroll** イベントはサポートされませんが、ほとんどの場合、SelectedIndexChanged イベントで適切に置き換えることができます。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードしたときに、**ComboBox** コントロールの **TopIndex** プロパティを参照するコードや、**ComboBox** コントロールまたは **ListBox** コントロールの **Scroll** イベント プロシージャを参照するコードはアップグレードされません。アップグレード ウィザードは、警告のコメントをコードに挿入します。アプリケーションをコンパイルするために、コードを削除する必要があります。

参照

関連項目

SelectedIndexChanged

SelectedIndexChanged

概念

ComboBox コントロール (Visual Basic 6.0 ユーザー向け)

ListBox コントロール (Visual Basic 6.0 ユーザー向け)

Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 のフォームとコントロールを使用した経験があれば、Visual Basic 2005 の Windows フォームと似た点が多数あることがわかります。しかし、概念的に異なる点もいくつかあります。次のトピックでは、そうした相違点について説明します。

このセクションの内容

次のトピックでは、Visual Basic 6.0 と Visual Basic 2005 におけるフォームとコントロールの操作に関する概念上の相違について説明します。

色の処理 (Visual Basic 6.0 ユーザー向け)

コントロール配列 (Visual Basic 6.0 ユーザー向け)

ダイアログ ボックスの変更点 (Visual Basic 6.0 ユーザー向け)

ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)

ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)

フォント処理 (Visual Basic 6.0 ユーザー向け)

フォームの配置 (Visual Basic 6.0 ユーザー向け)

フォームのタスク (Visual Basic 6.0 ユーザー向け)

グラフィックス (Visual Basic 6.0 ユーザー向け)

ヘルプ サポート (Visual Basic 6.0 ユーザー向け)

方法 : Visual Basic 6.0 ファイル システムのコントロールをアプリケーションに追加する

メニュー処理 (Visual Basic 6.0 ユーザー向け)

MDI (Visual Basic 6.0 ユーザー向け)

印刷の変更点 (Visual Basic 6.0 ユーザー向け)

ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)

バージョン番号 (Visual Basic 6.0 ユーザー向け)

関連するセクション

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 フォームと Windows フォーム コントロールの相違点について説明します。

Windows フォームの概要

フォームに関する基本概念を紹介します。

Windows フォームの新機能 (Visual Basic 6.0 ユーザー向け)

Windows フォームの新機能について説明します。

Visual Studio の Windows アプリケーションの名前空間

Windows フォームに関連する名前空間を紹介します。

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic 6.0 と Visual Basic 2005 の違いをより詳細に説明するトピックへのリンクを示します。

色の処理 (Visual Basic 6.0 ユーザー向け)

ここでは、Visual Basic 6.0 と Visual Basic 2005 での色の処理を比較します。

Visual Basic 2005 で色を操作する方法は、Visual Basic 6.0 の場合と似ていますが、理解する必要がある概念の違いが多少あります。また、Visual Basic 6.0 で色の指定に使う定数は、Visual Basic 2005 では新しい色の列挙に置き換えられます。

概念の違い

Visual Basic 6.0 では、色は **Long** 型の値でしたが、Visual Basic 2005 では Color 型の値です。Visual Basic 6.0 では、標準色を表す 8 つの定数がありましたが、Visual Basic 2005 では、名前の付けられた色が 100 以上あります。

ヒント
標準色ではない Visual Basic 6.0 の色に相当する色を見つけるには、Visual Basic 6.0 の色を表す **Long** 型の値を ColorTranslator クラスに渡します。

色の定数

Visual Basic 6.0 では、色をユーザーのシステム設定に対応付けるために使うシステム カラーに定数が用意されていました。Visual Basic 2005 では、システム カラーは SystemColors 型です。

BackColor プロパティおよび ForeColor プロパティ

Visual Basic 6.0 では、コントロールの **BackColor** プロパティと **ForeColor** プロパティをデザイン時または実行時に明示的に設定する必要があったため、色設定については継承できませんでした。Visual Basic 2005 では、デザイン時または実行時に色が明示的に設定されていない場合は、親の色設定を継承します。詳細については、「色の動作 (Visual Basic 6.0 ユーザー向け)」を参照してください。

Palette プロパティおよび PaletteMode プロパティ

Visual Basic 6.0 では、フォームの **Palette** プロパティおよび **PaletteMode** プロパティを使用して、256 色のモニタでイメージを表示するときに使用する色のセットを制御していました。Visual Basic 2005 では、**Palette** プロパティまたは **PaletteMode** プロパティはサポートされません。詳細については、「パレット (Visual Basic 6.0 ユーザー向け)」を参照してください。

MaskColor プロパティ

Visual Basic 6.0 の **MaskColor** プロパティは、CheckBox、Button、および RadioButton の各コントロールから削除されました。グラフィックス メソッドを使って、**MaskColor** プロパティを列挙できます。詳細については、「MaskColor (Visual Basic 6.0 ユーザー向け)」を参照してください。

色を扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

色をシステム カラーに設定するコードの変更

次のコードは、フォームの BackColor を、ユーザーがアクティブなタイトル バーに選択した色に一致させます。

```
' Visual Basic 6.0

Me.BackColor = vbActiveTitleBar
```

VB

```
' Visual Basic 2005
Me.BackColor = System.Drawing.SystemColors.ActiveCaption
```

色を Visual Basic 6.0 値に設定するコードの変更

次の例は、TextBox コントロールの BackColor プロパティを Visual Basic 6.0 のカラー ピッカーの最も明るい緑色 (`&H00C0FFC0`) に相当する値に設定します。

```
' Visual Basic 6.0

textBox1.Text = &H00C0FFC0
```

VB

```
' Visual Basic 2005
TextBox1.BackColor = System.Drawing.ColorTranslator.FromOle(&HC0FFC0)
```

定数の対応表

次の表は、Visual Basic 6.0 の定数と Visual Basic 2005 でそれらに対応するものを示しています。

カラー定数の対応表

Visual Basic 6.0	Visual Basic 2005 での同等物
vbBlack	Black
vbRed	Red
vbGreen	Lime
vbYellow	Yellow
vbBlue	Blue
vbMagenta	Magenta
vbCyan	Cyan
vbWhite	White

システム カラー定数の対応表

Visual Basic 6.0	Visual Basic 2005 での同等物
vb3DDKShadow	ControlDarkDark
vb3DFace	Control
vb3DHighlight	ControlLightLight
vb3DLight	ControlLight
vb3DShadow	ControlDark
vbActiveBorder	ActiveBorder
vbActiveTitleBar	ActiveCaption
vbActiveTitleBarText	ActiveCaptionText
vbApplicationWorkspace	AppWorkspace
vbButtonFace	**Control**
vbButtonShadow	**ControlDark**
vbButtonText	ControlText
vbDesktop	Desktop
vbGrayText	GrayText

vbHighlight	Highlight
vbHighlightText	HighlightText
vbInactiveBorder	InactiveBorder
vbInactiveCaptionText	InactiveCaptionText
vbInactiveTitleBar	InactiveCaption
vbInactiveTitleBarText	**InactiveCaptionText**
vbInfoBackground	Info
vbInfoText	InfoText
vbMenuBar	Menu
vbMenuText	MenuText
vbScrollBars	ScrollBar
vbTitleBarText	**ActiveCaptionText**
vbWindowBackground	Window
vbWindowFrame	WindowFrame
vbWindowText	WindowText

アップグレード メモ

アップグレード ウィザードを使って Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、色は FromOle メソッドを使って変換されます。

アップグレード後に、**BackColor** プロパティまたは ForeColor プロパティを実行時に明示的に設定するコードがないかを検索します。この設定が親に対して行われている場合は、デザイン時に子コントロールの色を明示的に設定します。それ以外の場合は、子コントロールは親から色を継承します。

Palette、**PaletteMode**、または **MaskColor** プロパティを使用するコードは、Visual Basic 2005 では書き直す必要があります。

参照

概念

色の動作 (Visual Basic 6.0 ユーザー向け)
パレット (Visual Basic 6.0 ユーザー向け)
MaskColor (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

色の動作 (Visual Basic 6.0 ユーザー向け)

コントロールの **ForeColor** プロパティおよび **BackColor** プロパティの動作は、Visual Basic 6.0 と Visual Basic 2005 で異なります。

概念の違い

Visual Basic 6.0 では、コントロールの **BackColor** プロパティと **ForeColor** プロパティをデザイン時または実行時に明示的に設定する必要があったため、色設定については継承できませんでした。Visual Basic 2005 では、デザイン時または実行時に色が明示的に設定されていない場合は、親の色設定が継承されます。

色の動作を扱うコードの変更

色設定の違いを示す例として、`Command1` と `Command2` という 2 つの **CommandButton** コントロールを含むフォームを次に示します。

```
' Visual Basic 6.0
' Command1's BackColor is left at its default (gray).
' Command2's BackColor is explicitly set.
Command2.BackColor = vbBlack
' Explicitly set the BackColor of the form.
Form1.BackColor = vbRed
```

上のコードを実行すると、`Command1` の **BackColor** は既定の灰色のままですが、`Command2` の **BackColor** は黒になります。

次の例は、Visual Basic 2005 の列挙型の動作を示しています。Visual Basic 2005 では、**CommandButton** コントロールは、Button コントロールに置き換えられています。

VB

```
' Visual Basic 2005
' Command1's BackColor is left at its default (gray).
' Command2's BackColor is explicitly set.
Command2.BackColor = System.Drawing.Color.Black
' Explicitly set the BackColor of the form.
Me.BackColor = System.Drawing.Color.Red
```

上のコードを実行すると、`Command1` の BackColor は赤になり、`Command2` の **BackColor** は黒になります。`Command1` の **BackColor** は明示的に設定されていませんが、赤がフォームから継承されます。

メモ :

これは、フォームに限らず、すべての親に当てはまります。**Button** がフォーム内の Panel コントロールに含まれている場合は、パネルまたはフォームのいずれかの色を変更すると、**Button** の色が変更されます。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードするときに、色が継承されるかどうかはアップグレード ウィザードにはわかりません。アップグレード後に、実行時に **BackColor** プロパティまたは ForeColor プロパティを明示的に設定するコードを検索してください。それが親に対して設定されている場合は、デザイン時に子コントロールの色を明示的に設定します。

参照

関連項目

色の処理 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

MaskColor (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **MaskColor** プロパティに直接対応するものは Visual Basic 2005 にありませんが、グラフィックス メソッドを使用して同じ結果を得ることができます。

概念の違い

Visual Basic 6.0 では、**CheckBox** コントロール、**CommandButton** コントロール、または **OptionButton** コントロールの **MaskColor** プロパティを使用することにより、バックグラウンド イメージを透かして表示することのできる透過色を定義していました。このプロパティを使うには、**Style** プロパティが **Graphical** に設定され、**UseMaskColor** プロパティが **True** に設定され、ビットマップが **Picture** プロパティに割り当てられている必要があります。

Visual Basic 2005 には、**MaskColor** プロパティに直接相当するものはありません。ただし、Bitmap オブジェクトの MakeTransparent メソッドを使うと、コントロールを透明に設定できます。

MaskColor を扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

```
' Visual Basic 6.0

' Assumes a picture has been assigned to the Picture property

' and that the Style property has been set to Graphical.

Command1.UseMaskColor = True

Command1.MaskColor = vbWhite
```

VB

```
' Visual Basic 2005
' Assumes a picture has been assigned to the BackgroundImage property.
Dim g As New System.Drawing.Bitmap(Button1.BackgroundImage)
g.MakeTransparent(System.Drawing.Color.White)
Button1.BackgroundImage = g
```

アップグレード メモ

アプリケーションを Visual Basic 6.0 から Visual Basic 2005 へアップグレードするときに、アップグレード ウィザードは、**MaskColor** プロパティを使用しているコードをアップグレードしません。アップグレードに関する警告がコードに挿入されます。アプリケーションを実行する前にコードを変更する必要があります。

参照

関連項目

色の処理 (Visual Basic 6.0 ユーザー向け)

MakeTransparent

概念

CheckBox コントロール (Visual Basic 6.0 ユーザー向け)

CommandButton コントロール (Visual Basic 6.0 ユーザー向け)

OptionButton コントロール (Visual Basic 6.0 ユーザー向け)

パレット (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Palette** プロパティおよび **PaletteMode** プロパティに相当するものは、Visual Basic 2005 にありません。

概念の違い

Visual Basic 6.0 では、フォーム コントロールの **Palette** プロパティおよび **PaletteMode** プロパティを使用して、256 色のモニタでイメージを表示するときに使用する色を制御していました。これらのプロパティは、256 色を超える色をサポートするモニタには影響しません。

Visual Basic 2005 では、**Palette** プロパティまたは **PaletteMode** プロパティはサポートされていません。通常、最新のモニタ機器は数千色から数万色を表示するので、イメージのパレットを指定する必要はなくなりました。

メモ：
Visual Basic 2005 の Image オブジェクトには、イメージを描画するための Palette プロパティがありますが、Visual Basic 6.0 の **Palette** プロパティと機能は同じではありません。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、**Palette** プロパティまたは **PaletteMode** プロパティはアップグレードされません。アップグレード ウィザードは、警告をコードに挿入します。これらのプロパティを参照するコードは削除する必要があります。

参照

関連項目

色の処理 (Visual Basic 6.0 ユーザー向け)

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

コントロール配列 (Visual Basic 6.0 ユーザー向け)

コントロール配列は Visual Basic 2005 ではサポートされなくなりましたが、イベント モデルを使うことで、コントロール配列に近い機能を実現したり、さらにそれ以上の機能に拡張したりできます。

概念の違い

Visual Basic 6.0 では、コントロール配列は、フォーム上のコントロールを管理するために使用できました。コントロール配列には、イベント ハンドラの共有、コントロールのグループへの反復処理、および実行時のコントロールの追加という機能がありました。

Visual Basic 2005 では、コントロール配列はサポートされていません。イベント モデルが変更されたため、コントロール配列は不要になりました。.NET Framework には、コントロールを扱うためにコントロール配列と同じ機能が用意されています。

イベント ハンドラの共有

Visual Basic 6.0 では、複数のイベントを共有するコントロールのグループを指定する場合は、コントロール配列が使用されていました。これらのコントロールは、同じ型と名前を持つ必要がありました。

Visual Basic 2005 では、複数のコントロールからのイベントを 1 つのイベント ハンドラで処理できます。名前や型が異なるコントロールであってもかまいません。

たとえば、2 つの Button コントロール (`Button1` と `Button2`) と 1 つの CheckBox コントロール (`CheckBox1`) をフォームに追加し、3 つすべてのコントロールの Click イベントを処理するイベント ハンドラを作成できます。

```
Private Sub MixedControls.Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click, Button2.Click, CheckBox1.Click
```

コントロールの反復処理

Visual Basic 6.0 コントロール配列の別の機能として、**Index** プロパティを使ってコントロールのグループを反復処理する機能がありました。たとえば、コントロール配列内のすべての **TextBox** コントロールのテキストをクリアするには、**Index** プロパティをループ変数として使って、コントロール配列をループ処理できます。

Visual Basic 2005 のコントロールには **Index** プロパティはありませんが、Control クラスの ControlCollection を使うと、フォーム上またはコンテナ上のコントロールを反復処理できます。

Visual Basic 6.0 では、同じコントロール配列に含まれるコントロールが複数のコンテナに配置されている場合がありました。たとえば、別々の **Frame** 上に配置された 2 つの **TextBox** コントロールを同じコントロール配列に格納できます。

Visual Basic 2005 では、**Controls** コレクションは、同じコンテナに配置されたコントロールを常に返します。反復処理は、各コンテナ コントロールのコントロールに個別に行う必要があります。この処理は、再帰関数を使って実行できます。

実行時のコントロールの追加

Visual Basic 6.0 では、**Load** ステートメントを使って、実行時にコントロールをコントロール配列に追加できました。コントロールはコントロール配列と同じ型であることが必要で、デザイン時には少なくとも 1 つの要素を格納してコントロール配列を作成する必要があります。コントロールを追加した後で、**Visible** プロパティを **True** に設定する必要があります。

Visual Basic 2005 でコントロールを実行時に追加するには、**Dim** ステートメントで **New** キーワードを使用してから、コントロールを追加するコンテナの Add メソッドを使用します。

実行時のイベント ハンドラの追加

Visual Basic 6.0 では、実行時にコントロールをコントロール配列に追加すると、新しいコントロールのイベントはコントロール配列のイベントで自動的に処理されました。

Visual Basic 2005 では、実行時に追加するコントロールにはイベント ハンドラを定義する必要があります。これは、**AddHandler** ステートメントを使って行うことができます。

コントロール配列を扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

イベント ハンドラの共有

次のコード例は、3 つの TextBox コントロールの **Change** イベント ハンドラ (Visual Basic 2005 では TextChanged) を共有する方法を示します。Visual Basic 2005 では、イベント ハンドラの **Handles** 句を使って、イベントがどのコントロールを処理するのかを指定します。イベント ハンドラは、汎用の **Object** を返すので、**DirectCast** メソッドを使って処理対象のオブジェクト型 (この例では **TextBox**) にキャストする必要があり

ます。

```
' Visual Basic 6.0

Private Sub Text1_Change(Index As Integer)

Select Case Index

Case 0

MsgBox("The text in the first TextBox has changed")

Case 1

MsgBox("The text in the second TextBox has changed")

Case 2

MsgBox("The text in the third TextBox has changed")

End Select

End Sub
```

VB

```
' Visual Basic 2005
Private Sub TextBoxes_TextChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles TextBox1.TextChanged, _
TextBox2.TextChanged, TextBox3.TextChanged
    Select Case DirectCast(sender, TextBox).Name
        Case TextBox1.Name
            MsgBox("The text in the first TextBox has changed")
        Case TextBox2.Name
            MsgBox("The text in the second TextBox has changed")
        Case TextBox3.Name
            MsgBox("The text in the third TextBox has changed")
    End Select
End Sub
```

メモ：

イベント ハンドラの動作は、Visual Basic 2005 では多少違います。コントロールが (たとえば `Form_Load` イベント内で) 初期化されたときや、テキストが実行時に変更されたときに、**TextChanged** イベントが生成されます。Visual Basic 6.0 では、**Change** イベントが生成されるのはテキストが変更されたときだけでした。

コントロールの反復処理

次のコード例は、テキスト ボックス コントロールのグループを反復処理し、それらのテキストをクリアするための関数を示します。Visual Basic 6.0 のコード例では、コントロール配列の **Index** プロパティをループ変数として使用します。

Visual Basic 2005 では、**Control** オブジェクトを引数として渡します。このオブジェクトには **ControlCollection** コレクションがあり、このコントロール上に配置されているすべてのコントロールがこのコレクションに含まれています。**Typeof** 演算子を使って、各コントロールが **TextBox** 型かどうかを確認します。

メモ：

Form オブジェクトは、**Control** 型です。**Form** を引数として渡すこともできます。

入れ子になったコントロールは **ControlCollection** コレクションに格納されないので、HasChildren メソッドを使って、コントロールに他のコントロールが含まれているかどうかを確認し、含まれている場合は `ClearText` 関数を再帰的に呼び出します。

```
' Visual Basic 6.0

Private Sub ClearText()

For i = 0 To Text1().UBound

Text1(i).Text = ""

Next

End Sub
```

VB

```
' Visual Basic 2005
Private Sub ClearText(ByVal container As Control)
    Dim ctrl As Control
    For Each ctrl In container.Controls
        If TypeOf (ctrl) Is TextBox Then
            ctrl.Text = ""
        End If
        If ctrl.HasChildren Then
             ClearText(ctrl)
        End If
    Next
End Sub
```

実行時のコントロールの追加

次のコード例は、実行時にフォームにテキスト ボックス コントロールを追加する方法を示します。Visual Basic 6.0 では、コントロールはコントロール配列に追加されます。Visual Basic 2005 では、コントロールは **ControlCollection** コレクションに追加されます。Visual Basic 6.0 では、新しい **TextBox** のイベントは、自動的にコントロール配列で処理されました。Visual Basic 2005 では、**AddHandler** ステートメントを使ってイベント処理を設定する必要があります。

どちらのコード例も、テキスト ボックス コントロールがデザイン時にフォームに追加されると仮定し、Visual Basic 6.0 のコード例は、要素が 1 つあるコントロール配列が作成されたと仮定します。また、Visual Basic 2005 のコード例は、`TextChangedHandler` という名前のイベント ハンドラが最初の **TextBox** コントロールにあると仮定します。

```
' Visual Basic 6.0

Private Sub AddControl()

' Add a TextBox as the second element of a control array.

Load Text1(1)

' Set the location below the first TextBox.

Text1(1).Move Text1(0).Left, Text1(0).Top + 500

' Make the new TextBox visible

Text1(1).Visible = True
```

VB

```
' Visual Basic 2005
' Declare a new TextBox.
Dim TextBox2 As New TextBox
' Set the location below the first TextBox
TextBox2.Left = TextBox1.Left
TextBox2.Top = TextBox1.Top + 30
' Add the TextBox to the form's Controls collection.
Me.Controls.Add(TextBox2)
AddHandler TextBox2.TextChanged, AddressOf TextChangedHandler
```

アップグレード メモ

Visual Basic 6.0 で作成したアプリケーションを Visual Basic 2005 にアップグレードすると、コントロール配列は特別なコントロール固有のコントロール配列クラスにアップグレードされます。このようなクラスは、**Microsoft.VisualBasic.Compatibility.VB6** 名前空間に格納され、Visual Basic 6.0 のコントロール配列の動作をエミュレートするためにアップグレード ツールで使用されます。

これらのコントロール配列クラスを新しい Visual Basic 2005 開発に利用することは可能ですが、.NET Framework のイベント モデルと関数を使うことをお勧めします。

参照

概念

イベントおよびイベント処理 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

ダイアログ ボックスの変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 でダイアログ ボックスを表示する方法は、Visual Basic 6.0 の場合とは異なります。

概念の違い

Visual Basic 6.0 では、フォームをモーダル ダイアログ ボックスとして表示するには、**Show** メソッドを **vbModal** パラメータで呼び出します。

Visual Basic 2005 では、フォームをモーダルで表示するときに、ShowDialog メソッドを使用します。**Show** メソッドは非モーダルで表示するときに使用します。

さらに、Visual Basic 6.0 のコモン ダイアログ (**CommonDialog**) コントロールには、定義済みダイアログ ボックスがいくつか用意されています。Visual Basic 2005 では、このコントロールは ColorDialog、FontDialog、OpenFileDialog、PageSetupDialog、PrintDialog、PrintPreviewDialog、および SaveFileDialog の各コントロールで置き換えられています。

参照

概念

CommonDialog コントロール (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォームのダイアログ ボックス

ダイアログ ボックス コントロールおよびコンポーネント (Windows フォーム)

ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 でドラッグ アンド ドロップ編集を実装するモデルは、Visual Basic 6.0 のモデルと大きく異なります。

概念の違い

Visual Basic 6.0 では、ドラッグ アンド ドロップ編集は 2 種類の方法で実現されています。1 つはフォーム上のコントロール間でドラッグするための標準的なドラッグであり、もう 1 つはフォームとアプリケーションとの間でドラッグするための OLE ドラッグです。

Visual Basic 2005 では、ドラッグ アンド ドロップ編集に単一のモデルが使用されます。このモデルは OLE ドラッグに似ていますが、ドラッグ アンド ドロップのメソッドとイベントに対する名前とパラメータが異なっている上に、イベント モデルも異なります。

ドラッグ アンド ドロップを実行するコードの変更

テキストをドラッグ アンド ドロップするコードの変更

次の例は、**TextBox** コントロールから別の **TextBox** コントロールへテキストをドラッグするコードの相違点を示します。

```
' Visual Basic 6.0
Private Sub Text1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Text1.Text = "Hello World"
    ' Begin dragging by calling the OLEDrag method.
    Text1.OLEDrag
End Sub

Private Sub Text1_OLEStartDrag(Data As DataObject, AllowedEffects As Long)
    ' Only allow copying.
    AllowedEffects = vbDropEffectCopy
    Data.Clear
    ' Populate the Data object with the text to copy and the format.
    Data.SetData Text1.Text, vbCFText
End Sub

Private Sub Text2_OLEDragOver(Data As DataObject, Effect As Long, Button As Integer, Shift
As Integer, X As Single, Y As Single, State As Integer)
    ' Make sure that the format is text.
    If Data.GetFormat(vbCFText) Then
      ' If it is text, enable dropping for the second TextBox.
      Text2.OLEDropMode = vbOLEDropManual
    End If
End Sub

Private Sub Text2_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, Shift
As Integer, X As Single, Y As Single)
    ' Copy the text from the Data object to the second TextBox.
    Text2.Text = Data.GetData(vbCFText)
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles MyBase.Load

    ' Dropping must be enabled before the dragging occurs.
    TextBox2.AllowDrop = True
End Sub

Private Sub TextBox1_MouseDown(ByVal sender As Object, ByVal e _
As System.Windows.Forms.MouseEventArgs) Handles TextBox1.MouseDown

    TextBox1.Text = "Hello World"
    ' Begin dragging by calling the DoDragDrop method.
    ' OLEStartDrag is replaced by arguments on the method.
    TextBox1.DoDragDrop(TextBox1.Text, DragDropEffects.Copy)
End Sub
```

```
Private Sub TextBox2_DragEnter(ByVal sender As Object, ByVal e _
As System.Windows.Forms.DragEventArgs) Handles TextBox2.DragEnter

    ' Make sure that the format is text.
    If (e.Data.GetDataPresent(DataFormats.Text)) Then
      ' Allow drop.
      e.Effect = DragDropEffects.Copy
    Else
      ' Do not allow drop.
      e.Effect = DragDropEffects.None
    End If
End Sub

Private Sub TextBox2_DragDrop(ByVal sender As Object, ByVal e _
As System.Windows.Forms.DragEventArgs) Handles TextBox2.DragDrop

    ' Copy the text to the second TextBox.
    TextBox2.Text = e.Data.GetData(DataFormats.Text).ToString
End Sub
```

アップグレード メモ

ドラッグ アンド ドロップ コードは、Visual Basic 2005 に自動的にはアップグレードされません。したがって、新しいモデルを使用したコードに書き換える必要があります。アップグレードの過程で、すべてのドラッグ アンド ドロップ コードには警告が挿入されます。

参照

その他の技術情報

ドラッグ アンド ドロップ操作とクリップボードのサポート

ダイナミック データ エクスチェンジ (Visual Basic 6.0 ユーザー向け)

ダイナミック データ エクスチェンジは、Visual Basic 2005 でサポートされなくなりました

概念の違い

ダイナミック データ エクスチェンジ (DDE) は、Visual Basic 3.0 でアプリケーション間の情報交換のために使用された初期のテクノロジです。最近のバージョンの Visual Basic では DDE の代わりに OLE オートメーションが使用されましたが、DDE は下位互換性のために Visual Basic 6.0 でもサポートされていました。

Visual Basic .NET 2002 の導入に伴い、DDE はサポートされなくなりました。DDE のみをサポートするレガシ アプリケーションと情報を交換する必要がある Visual Basic アプリケーションは、Visual Basic 6.0 を使用してください。

次に示す DDE のプロパティ、メソッド、およびイベントは、使用できません。

カテゴリ	名前
プロパティ	**LinkItem** **LinkMode** **LinkTimeOut** **LinkTopic**
メソッド	**LinkExecute** **LinkPoke** **LinkRequest** **LinkSend**
イベント	**LinkClose** **LinkError** **LinkExecute** **LinkNotify** **LinkOpen**

また、**Clipboard** オブジェクトの **vbCFLink** 定数も使用できません。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードしても、DDE のプロパティやメソッドを参照するコードと DDE イベント プロシージャ内のコードはアップグレードされず、警告がコードに追加されます。

参照

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)
Label コントロール (Visual Basic 6.0 ユーザー向け)
PictureBox コントロール (Visual Basic 6.0 ユーザー向け)
TextBox コントロール (Visual Basic 6.0 ユーザー向け)
Clipboard オブジェクト (Visual Basic 6.0 ユーザー向け)

フォント処理 (Visual Basic 6.0 ユーザー向け)

ここでは、Visual Basic 6.0 と Visual Basic 2005 でのフォント処理テクニックを比較します。

概念の違い

Visual Basic 6.0 では、2 つの異なる方法でフォントを取り扱います。フォームおよびコントロールのフォント プロパティを使用する方法と、**stdFont** オブジェクトを使用する方法です。

Visual Basic 2005 では、**Font** オブジェクトは **System.Drawing.Font** の 1 つだけです。フォームまたはコントロールの **Font** プロパティは、**Font** オブジェクトを引数として受け取ります。

フォント プロパティの設定

Visual Basic 6.0 のフォント プロパティは、**stdFont** オブジェクトを割り当てるか、またはプロパティをコントロールに直接設定して、実行時に設定できます。どちらの方法を使っても結果は同じです。

Visual Basic 2005 では、コントロールの **Font** プロパティは、実行時には読み取り専用です。プロパティを直接設定することはできません。プロパティを設定するたびに、新しい **Font** オブジェクトのインスタンスを生成する必要があります。

フォントの継承

Visual Basic 6.0 では、コントロールまたはフォームごとにフォント プロパティを個別に設定する必要があります。**stdFont** オブジェクトを使うことで処理は簡単になりますが、それでもコードは必要です。

Visual Basic 2005 では、子オブジェクトに対して明示的にフォント プロパティが設定されている場合を除き、親のフォント プロパティが自動的に継承されます。たとえば、フォーム上に 2 つのラベル コントロールがある場合に、フォームのフォント プロパティを Arial に変更すると、ラベル コントロールのフォントも Arial に変更されます。その後で一方のラベルのフォントを Times Roman に変更すると、それ以降フォームのフォントを変更しても、そのラベルのフォントには影響しません。

フォントの互換性

Visual Basic 6.0 では下位互換性のためにラスタ フォントがサポートされていますが、Visual Basic 2005 では TrueType フォントと OpenType フォントだけがサポートされています。

フォントの列挙

Visual Basic 6.0 では、**Screen.FontCount** プロパティと共に **Screen.Fonts** コレクションを使用して、使用できる画面フォントを列挙できます。

Visual Basic 2005 には **Screen** オブジェクトがありません。システムで使用できるフォントを列挙する場合は、**System.Drawing.FontFamily** 名前空間を使用する必要があります。

メモ :
Visual Basic 6.0 では、すべての種類のフォントが列挙されます。Visual Basic 2005 では、TrueType フォントと OpenType フォントだけがサポートされています。その他の種類のフォントは列挙されません。また、Visual Basic 6.0 ではフォント ファミリ内での文字セットの各バージョン (たとえば、Arial、Arial Baltic、Arial Greek など) が列挙されますが、Visual Basic 2005 ではフォント ファミリだけが列挙されます。

フォントを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

フォント プロパティを設定するコードの変更

次のコード例は、フォント プロパティを実行時に設定する方法を示しています。Visual Basic 6.0 では、コントロール上でプロパティを直接設定できますが、Visual Basic 2005 では、プロパティを設定する必要があるたびに、新しい **Font** オブジェクトを作成し、それをコントロールに割り当てる必要があります。

```
' Visual Basic 6.0
' Set font properties directly on the control.
Label1.FontBold = True
' Create a stdFont object.
Dim f As New stdFont
' Set the stdFont object to the Arial font.
f.Name = "Arial"
' Assign the stdFont to the control's font property.
```

```
Set Label1.Font = f
' You can still change properties at run time.
Label1.FontBold = True
Label1.FontItalic = True
```

VB

```
' Visual Basic 2005
' Create a new Font object  Name and Size are required.
Dim f As New System.Drawing.Font("Arial", 10)
' Assign the font to the control
Label1.Font = f
' To set additional properties, you must create a new Font object.
Label1.Font = New System.Drawing.Font(Label1.Font, FontStyle.Bold Or FontStyle.Italic)
```

フォントを列挙するコードの変更

次のコード例は、コンピュータにインストールされているフォントの一覧を **ListBox** コントロールに設定する方法を示しています。

メモ：

Visual Basic 6.0 では、すべての種類のフォントが列挙されます。Visual Basic 2005 では、TrueType フォントと OpenType フォントだけがサポートされています。その他の種類のフォントは列挙されません。また、Visual Basic 6.0 ではフォント ファミリ内での文字セットの各バージョン (たとえば、Arial、Arial Baltic、Arial Greek など) が列挙されますが、Visual Basic 2005 ではフォント ファミリだけが列挙されます。

```
' Visual Basic 6.0
Dim i As Integer
For i = 0 To Screen.FontCount - 1
   List1.AddItem Screen.Fonts(i)
Next i
```

VB

```
' Visual Basic 2005
Dim ff As FontFamily
For Each ff In System.Drawing.FontFamily.Families
  listBox1.Items.Add(ff.Name)
Next
```

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、フォントを扱うコードは新しい **Font** オブジェクトを使うように更新されます。

Visual Basic 2005 でのフォントの継承によって、アプリケーションの表示に予期しない変化が生じる可能性があります。変換したアプリケーションをチェックして、フォントをフォーム レベルまたはコンテナ レベルで明示的に設定するコードがあるかどうかを確認し、親から継承されるフォントが子コントロールにとって適切ではない場合にフォントを変更する必要があるかどうかを確認してください。

アップグレードを行うとき、ラスタ フォントは既定の OpenType フォントである Microsoft Sans Serif に変換されます。太字や斜体などの書式は保持されません。詳細については、「OpenType フォントと TrueType フォントだけがサポートされている」を参照してください。

フォントを列挙するコードがアプリケーションにある場合、アップグレードしたアプリケーションではラスタ フォントは列挙されません。また、列挙されるのは、文字セットの各バージョンではなく、フォント ファミリです。

参照

関連項目

Font Class

FontFamily.Families Property

その他の技術情報

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

フォームの配置 (Visual Basic 6.0 ユーザー向け)

フォームを画面に配置する方法は、Visual Basic 6.0 と Visual Basic 2005 で異なります。

概念の違い

Left プロパティと Top プロパティ

Visual Basic 6.0 では、フォームの初期画面位置は、[フォーム レイアウト] ウィンドウを使用してデザイン時に設定するか、[プロパティ] ウィンドウの **Left** プロパティおよび **Top** プロパティを設定することにより指定できます。**Left** プロパティおよび **Top** プロパティはプログラムで設定することもできます。

Visual Basic 2005 では、初期画面位置は、**Location** プロパティを使用してデザイン時に設定するか、**Location** プロパティを **Point** オブジェクトに設定することによりプログラムで設定します。

StartUpPosition プロパティ

また、Visual Basic 6.0 では、Windows が自動的に位置 (画面の中央など) を設定できるようにするための **StartUpPosition** プロパティがサポートされています。既定では、**Manual** に設定され、**Left** プロパティおよび **Top** プロパティが優先されます。

Visual Basic 2005 では、**StartPosition** プロパティがこれと同じ機能を持ちます。既定では、**WindowsDefaultLocation** に設定され、**Location** プロパティは無視されます。

参照

処理手順

方法 : Windows フォームの画面上の位置を設定する

フォームのタスク (Visual Basic 6.0 ユーザー向け)

Windows フォームと Visual Basic 6.0 フォームには、さまざまな違いがあります。フォームを使用して実行できる作業およびその手順は、次の表のとおりです。既存の Visual Basic 6.0 フォームのプロジェクトを Windows フォーム対応の新規の .NET Framework モデルに変換する場合は、この一覧を参照してください。

作業	Visual Basic 6.0 フォーム	Windows フォーム
フォームのすべてのコントロールのタブ インデックスを設定します。	フォーム上の各コントロールを選択し、**TabIndex** プロパティをフォーム内の次のタブ オーダー番号に設定します。	[表示] メニューの [タブ オーダー] をクリックします。フォームの各コントロールの横に、コントロールの TabIndex プロパティの設定を示す数値が表示されます。目的のタブ オーダーに従って、各コントロールをクリックします。詳細については、「方法 : Windows フォーム上のタブ オーダーを設定する」を参照してください。
キーボードとマウスのイベント中に押す修飾子キー (**Alt**、**Shift**、**Ctrl**) を決定します。	マウス イベント ハンドラで指定されたパラメータを使用します。	フォームのクラスの ModifierKeys メソッドを Keys 列挙値内の修飾子キーの値と比較します。詳細については、「方法 : どの修飾子キーが押されたかを判断する」を参照してください。
基準となるコントロールの位置とサイズに合わせて、複数のコントロールを配置またはサイズ変更します。	配置またはサイズ変更するコントロールをそれぞれクリックします。最後に選択したコントロールが、その他のコントロールの位置とサイズを決定する基準となります。	配置またはサイズ変更するコントロールをそれぞれクリックします。最初に選択したコントロールが、その他のコントロールの位置とサイズを決定する基準となります。詳細については、「方法 : Windows フォーム上の複数のコントロールを配置する」を参照してください。
フォーム名の変更	[プロパティ] ウィンドウ内のフォームの **Name** プロパティで指定されている名前を変更します。	ソリューション エクスプローラでフォームのクラス ファイルをクリックし、[プロパティ] ウィンドウで **Filename** プロパティを変更します。
フォームの実行時の表示位置を変更します。	[フォーム レイアウト] ウィンドウを使用してフォームを位置設定するか、または [プロパティ] ウィンドウで **Startup** プロパティを設定します。	Windows フォーム デザイナ内でフォームを選択した状態で、[プロパティ] ウィンドウの StartPosition プロパティを設定します。
ダイアログ ボックスとしてフォームを表示します。	コード内で **Show** メソッドを使用し、**vbModal** 値を使用して、モーダルで表示するフォームを指定します。	コード内で、フォームの ShowDialog メソッドを使用します。詳細については、「方法 : Windows フォームのダイアログ ボックスを表示する」を参照してください。
フォーム上のコントロールをグループ化またはレイヤ化します。	Frame コントロール内でコントロールをレイヤ化します。コントロールをグループごと移動するには、グループ化するコントロールをすべて選択します。	GroupBox コントロール内にコントロールを配置します。グループ ボックスを移動すると、中のコントロールも同様に移動します。
フォーム上で、反復処理でコントロールを作成します。	コントロール配列を使用します。	Controls Collection を使用します。詳細については、「Visual Basic におけるコレクション」を参照してください。

参照

関連項目

Windows フォームの概要

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

コントロール配列 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォームの概念 (Visual Basic 6.0 ユーザー向け)

グラフィックス (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、**Form** コントロールまたは **PictureBox** コントロールに描画する場合は、グラフィックスの各種メソッドおよびプロパティを使用します。Visual Basic 6.0 のグラフィックスは、Windows のグラフィック デバイス インターフェイス (GDI) の API に基づいてます。

Visual Basic 2005 のグラフィックスは、GDI+ API にカプセル化された System.Drawing 名前空間によって提供されます。GDI+ は Visual Basic 6.0 のグラフィックス機能に基づいて拡張されていますが、メソッドに互換性はありません。

概念の違い

Visual Basic 6.0 では、グラフィックス メソッドは、**Form** オブジェクトおよび **PictureBox** コントロールだけに適用されます。

Visual Basic 2005 では、グラフィックス メソッドは、フォームに加えて、PictureBox、Panel、GroupBox など、Paint イベントをサポートする任意のコントロールに適用されます。また、グラフィックス メソッドは、ListView、TreeView、Button コントロールなど、OwnerDraw プロパティをサポートする任意のコントロールにも適用されます。

AutoRedraw プロパティ

Visual Basic 6.0 のグラフィックス メソッドは、任意のイベント プロシージャから呼び出すことができます。グラフィックス メソッドが、**Paint** 以外のイベントから呼び出された場合は、グラフィックスを永続化するために **AutoRedraw** プロパティが使用されます。

Visual Basic 2005 では、グラフィックス メソッドは、**Paint** イベント プロシージャからのみ呼び出されます。ただし、オーナー描画コントロールの場合、さまざまな **Draw** イベント プロシージャ (DrawItem、DrawSubItem など) からも呼び出されます。**Paint** イベントおよび **Draw** イベントで自動的にグラフィックスが永続化されるので、**AutoRedraw** プロパティは不要になり、サポートされなくなりました。

ClipControls プロパティ

Visual Basic 6.0 では、フォームまたはコントロールの描画を制御するために、**ClipControls** プロパティを使用します。このプロパティを **True** に設定すると、それまで見えていなかった領域のみが再描画され、理論上はパフォーマンスが向上します。

ClipControls プロパティに相当するものは、Visual Basic 2005 にはありません。GDI+ によるパフォーマンス向上と、最新のビデオ アダプタの使用で、このプロパティの必要はなくなりました。

DrawMode プロパティ

Visual Basic 6.0 では、パターンを別のパターンの上に描画するときに **DrawMode** プロパティによってグラフィックス オブジェクトの色が制御されます。このプロパティが効力を持つのは、モノクロ ディスプレイまたは解像度の低い (256 色以下) ディスプレイだけです。

DrawMode プロパティに相当するものは、Visual Basic 2005 にありません。現在のディスプレイでは不要になりました。

DrawStyle プロパティ

Visual Basic 6.0 では、**Line** メソッドを使用して描画する線の外観を制御するために、**DrawStyle** プロパティを使用します。**DrawWidth** プロパティが 1 より大きい値に設定されている場合、**DrawStyle** プロパティは無効になり、線は常に実線になります。

Visual Basic 2005 では、**DrawLine** メソッドの 1 つで使用される **System.Drawing.Pen** クラスの DashStyle プロパティを設定することで、線の外観を制御します。線の幅は、このプロパティに影響されません。

DrawWidth プロパティ

Visual Basic 6.0 では、**DrawWidth** プロパティで線のピクセル幅が決定されます。通常、**DrawWidth** プロパティは、グラフィックス メソッドを実行する前に設定します。

Visual Basic 2005 では、**System.Drawing.Pen** コントロールの **Pen.Width** プロパティで線の幅が決定されます。**Pen** の作成時に **Width** プロパティをパラメータとして設定するか、**Pen** の作成後に **Pen.Width** を設定できます。**Pen.Width** プロパティを指定しない場合、既定は 1 ピクセルです。

Image プロパティ

Visual Basic 6.0 では、フォームまたは **PictureBox** コントロールの **Image** プロパティは、ビットマップのハンドルを返します。このハンドルは、**Picture** プロパティに割り当てたり、Windows API の呼び出し時に値として渡したりできます。

Visual Basic 2005 では、ビットマップにハンドルはなくなりました。ビットマップそのものを Bitmap 型のオブジェクトとして渡します。**Bitmap** コントロールは、**PictureBox** コントロールの Image プロパティに割り当てることができますが、Windows API の呼び出し時に渡すことはできません。

Line メソッド

Visual Basic 6.0 では、**Line** メソッドは、左上と下部の座標およびオプションの引数 B を指定して、四角形を描画するために使用します。**FillColor** プロパティは純色で四角形を塗りつぶすのに使用し、**FillStyle** プロパティはクロスハッチ パターンで四角形を塗りつぶすのに使用します。

Visual Basic 2005 では、DrawRectangles メソッドは四角形の境界線を描画するのに使用し、FillRectangle メソッドは四角形を塗りつぶすのに使用します。**FillRectangle** は、Brush オブジェクトをパラメータとして受け取ります。**FillColor** プロパティは SolidBrush に置き換えられ、**FillStyle** プロパティは HatchBrush クラスのメンバに置き換えられました。

Point メソッド

Visual Basic 6.0 では、フォームまたは **PictureBox** コントロールの **Point** メソッドは、指定の場所にあるピクセルのカラー値を取得するのに使用します。**Point** メソッドは、画像のないフォームまたはコントロールでも使用できますが、ほとんどの場合は、**Picture** プロパティに割り当てられたビットマップのカラーを取得するために使用されます。

Visual Basic 2005 では、**Point** メソッドはなくなりました。ビットマップからカラー値を取得するには、**M:System.Drawing.Bitmap.GetPixel(System.Int32,System.Int32)** メソッドを使用します。フォームまたはコントロールに画像がない場合は、BackColor プロパティを使用できます。

Print メソッド

Visual Basic 6.0 では、**Print** メソッドは、フォームまたは **PictureBox** コントロールにテキストを表示するのに使用します。テキストの表示に使うフォントは、フォームまたはコントロールの **Font** プロパティで決定され、色は **ForeColor** プロパティで決定されます。**Print** メソッドには、テキストの配置を決めたり、テキストを縦方向に表示したりする機能はありません。

Visual Basic 2005 では、テキストの表示に **DrawString** メソッドを使用します。フォントは Font オブジェクトによって決定され、色は Brush オブジェクトによって決定されます。どちらのオブジェクトも、パラメータとして **DrawString** メソッドに渡します。**DrawString** メソッドには、*X* および *Y* というパラメータもあり、これらを使ってテキストの開始位置を指定できます。また、オプションの *Format* パラメータに StringFormat オブジェクトを渡して、テキストを縦方向に表示することもできます。

PSet メソッド

Visual Basic 6.0 では、**PSet** メソッドは、フォーム上または **PictureBox** コントロール上のピクセルの色を変更するのに使用します。**DrawWidth** プロパティが 1 より大きい値に設定されている場合は、**PSet** メソッドは塗りつぶされた円を描画します。省略された **ForeColor** を使用する場合は、省略可能なパラメータを使って色を指定します。

Visual Basic 2005 には、**PSet** メソッドに相当するものはありません。フォーム上または **PictureBox** コントロール上の 1 ピクセルの色を変更するには、**DrawEllipse** メソッドを使って高さと幅が 1 ピクセルの円を描画します。**DrawWidth** が 1 より大きい場合に **PSet** と同等の機能を実現するには、**FillEllipse** メソッドを使用します。

グラフィックスを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

単純な線の描画

次のコードは、実行時にフォーム上に線を描画する方法を示します。Visual Basic 6.0 のコード例では、**Line** メソッドを使用し、パラメータとして線の開始位置と終了位置の X 座標と Y 座標を渡し、オプションの引数として色を渡します。Visual Basic 2005 のコード例では、**DrawLine** メソッドを使用し、引数として Pens オブジェクトと、開始位置と終了位置の X 座標および Y 座標を渡します。

メモ：

Visual Basic 6.0 の既定の測定単位は twip ですが、Visual Basic 2005 の測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a solid black line 200 twips from the top of the form.
    Line (0, 200) - (ScaleWidth, 200), vbBlack
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint(ByVal sender As Object, ByVal e _
As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a solid black line 25 pixels from the top of the form.
    e.Graphics.DrawLine(Pens.Black, 0, 25, Me.Width, 25)
End Sub
```

点線の描画

次のコードは、実行時にフォーム上に点線を描画する方法を示します。Visual Basic 6.0 のコード例では、**DrawStyle** プロパティを使用して線の外観を指定します。Visual Basic 2005 のコード例では、Pen オブジェクトを使用し、**DashStyle** プロパティを設定して外観を指定します。

メモ：

Visual Basic 6.0 の既定の測定単位は twip ですが、Visual Basic 2005 の測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a dotted line 200 twips from the top of the form.
    Me.DrawStyle = vbDot
    Line (0, 200) - (ScaleWidth, 200), vbBlack
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint1(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a dotted black line 25 pixels from the top of the form.
    Dim LPen As New System.Drawing.Pen(System.Drawing.Color.Black)
    LPen.DashStyle = Drawing2D.DashStyle.Dot
    e.Graphics.DrawLine(LPen, 0, 25, Me.Width, 25)
End Sub
```

線の太さの制御

次のコードは、実行時にフォーム上に太さの異なる複数の線を描画する方法を示します。Visual Basic 6.0 のコード例では、**DrawWidth** プロパティを使用します。Visual Basic 2005 のコード例では、**Pens** オブジェクトの **Width** プロパティを使用します。

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a line with a thickness of 1 pixel.
    DrawWidth = 1
    Line (0, 200)-(ScaleWidth, 200), vbBlack
    ' Draw a line with a thickness of 5 pixels.
    DrawWidth = 5
    Line (0, 400)-(ScaleWidth, 400), vbBlack
    ' Draw a line with a thickness of 10 pixels.
    DrawWidth = 10
    Line (0, 600)-(ScaleWidth, 600), vbBlack
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint2(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a line with a thickness of 1 pixel.
    Dim TPen As New System.Drawing.Pen(System.Drawing.Color.Black, 1)
    e.Graphics.DrawLine(TPen, 0, 25, Me.Width, 25)
    ' Draw a line with a thickness of 5 pixels.
    TPen.Width = 5
    e.Graphics.DrawLine(TPen, 0, 50, Me.Width, 50)
    ' Draw a line with a thickness of 10 pixels.
    TPen.Width = 10
    e.Graphics.DrawLine(TPen, 0, 75, Me.Width, 75)
End Sub
```

円の描画

次のコードは、実行時にフォーム上に円を描画する方法を示します。Visual Basic 6.0 のコード例では、**Circle** メソッドを使用し、引数として円の中心点の X および Y 座標と半径を渡し、オプションで色を渡します。Visual Basic 2005 のコード例では、**DrawEllipse** メソッドを使用し、引数として **Pen** オブジェクト、外接する四角形の左上隅の X 座標および Y 座標、幅と高さを渡します。

メモ：
Visual Basic 6.0 の既定の測定単位は twip ですが、Visual Basic 2005 の測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a 1000 twip diameter red circle
    Circle (500, 500), 500, vbRed
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint3(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a 70 pixel diameter red circle.
    e.Graphics.DrawEllipse(Pens.Red, 0, 0, 70, 70)
End Sub
```

塗りつぶされた四角形の描画

次のコードでは、実行時に 2 つの四角形をフォーム上に描画します。1 つの四角形は純色で塗りつぶし、もう 1 つの四角形はクロスハッチ パターンで塗りつぶします。Visual Basic 6.0 のコード例では、**FillColor** プロパティと **FillStyle** プロパティおよび **Line** メソッドを使用します。*B* パラメータを指定して **Line** メソッドを呼び出して、四角形を描画します。

Visual Basic 2005 のコード例では、**Graphics.Rectangle** メソッドを使って輪郭を描画し、**Graphics.FillRectangle** メソッドに引数として **Brush** オブジェクトを渡します。この例では、**SolidBrush** コントロールと **HatchBrush** コントロールの両方を使用します。

メモ：
Visual Basic 6.0 の既定の測定単位は twip ですが、Visual Basic 2005 の測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Draw a solid red rectangle.
    FillColor = vbRed
    FillStyle = vbSolid
    Line (10, 10)- (1000, 500), vbRed, B
    ' Draw a rectangle filled with a crosshatch pattern.
    FillColor = vbBlack
    FillStyle = vbCross
    Line (10, 500)- (1000, 1000), vbBlack, B
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint4(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Draw a solid red rectangle.
    Dim SBrush As New System.Drawing.SolidBrush _
      (System.Drawing.Color.Red)
    e.Graphics.DrawRectangle(Pens.Red, 2, 2, 70, 40)
    e.Graphics.FillRectangle(SBrush, 2, 2, 70, 40)

    ' Draw a rectangle filled with a crosshatch pattern.
    Dim HBrush As New System.Drawing.Drawing2D.HatchBrush( _
      System.Drawing.Drawing2D.HatchStyle.Cross, _
```

```
      System.Drawing.Color.Black, System.Drawing.Color.Transparent)
    e.Graphics.DrawRectangle(Pens.Black, 2, 40, 70, 40)
    e.Graphics.FillRectangle(HBrush, 2, 40, 70, 40)
End Sub
```

フォーム上での画像の表示

次のコードは、グラフィックス メソッドを使って、実行時にフォーム上に画像を表示する方法を示します。Visual Basic 6.0 のコード例では、**PaintPicture** メソッドを使用します。Visual Basic 2005 のコード例では、**DrawImage** メソッドを使用します。

```
' Visual Basic 6.0
Private Sub Form_Paint()
    ' Create a stdPicture object.
    Dim Pict1 As New stdPicture
    Pict1 = LoadPicture("C:\Windows\Greenstone.bmp")
    PaintPicture Pict1, 0, 0
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint5(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    ' Create a Bitmap object.
    Dim Pict1 As New Bitmap("C:\Windows\Greenstone.bmp")
    e.Graphics.DrawImage(Pict1, 0, 0)
End Sub
```

フォーム上でのテキストの表示

次のコードは、グラフィックス メソッドを使って、実行時にフォーム上にテキスト文字列を表示する方法を示します。Visual Basic 6.0 のコード例では、**Print** メソッドを使用します。Visual Basic 2005 のコード例では、**DrawString** メソッドを使用します。

```
' Visual Basic 6.0
Private Sub Form_Paint()
    Me.Font.Size = 24
    Me.Font.Bold = True
    Me.ForeColor = vbRed
    Print "Hello World!"
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint6(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    Dim TextFont As New System.Drawing.Font("Arial", 24, FontStyle.Bold)
    Dim TextBrush As New System.Drawing.SolidBrush(System.Drawing.Color.Red)
    e.Graphics.DrawString("Hello World!", TextFont, TextBrush, 10, 10)
    TextFont.Dispose()
    TextBrush.Dispose()
End Sub
```

文字列の高さと幅の確認

次のコードは、グラフィックス メソッドを使って、実行時にフォーム上の文字列のサイズを確認し、その周囲に四角形を描画する方法を示します。Visual Basic 6.0 のコード例では、**TextHeight** メソッドと **TextWidth** メソッドを使用します。Visual Basic 2005 のコード例では、MeasureString メソッドを使用します。このメソッドは、SizeF 構造体を返します。

```
' Visual Basic 6.0
```

```
Private Sub Form_Paint()
    Me.Font.Size = 24
    Me.Font.Bold = True
    Me.ForeColor = vbRed
    Print "Hello World!"
    Line (0, 0)-(TextWidth("Hello World!"), _
TextHeight("Hello World!")), vbBlack, B
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint7(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    Dim TextFont As New System.Drawing.Font("Arial", 24, FontStyle.Bold)
    Dim TextBrush As New System.Drawing.SolidBrush(System.Drawing.Color.Red)
    e.Graphics.DrawString("Hello World!", TextFont, TextBrush, 10, 10)
    Dim TextSize As New System.Drawing.SizeF
    TextSize = e.Graphics.MeasureString("Hello World!", TextFont)
    e.Graphics.DrawRectangle(Pens.Black, 10, 10, TextSize.Width, TextSize.Height)
    TextFont.Dispose()
    TextBrush.Dispose()
End Sub
```

単一のピクセルの描画

次のコードは、グラフィックス メソッドを使って、実行時にフォーム上の 1 ピクセルの色を変更する方法を示します。Visual Basic 6.0 のコード例では、**PSet** メソッドを使用します。Visual Basic 2005 のコード例では、**DrawEllipse** メソッドを使用し、**Height** パラメータと **Width** パラメータを 1 に設定します。

メモ　Visual Basic 6.0 の既定の測定単位は twip ですが、Visual Basic 2005 の測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Paint()
    Me.DrawWidth = 1
    PSet (1000, 1000), vbRed
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint8(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    e.Graphics.DrawEllipse(Pens.Red, 70, 70, 1, 1)
End Sub
```

単一ピクセルの色の確認

次のコードは、グラフィックス メソッドを使って、実行時にフォーム上の画像の指定した位置にある 1 ピクセルの色を確認し、その色で四角形を塗りつぶして描画する方法を示します。Visual Basic 6.0 のコード例では、**Point** メソッドを使ってカラー値を取得します。Visual Basic 2005 のコード例では、GetPixel メソッドを使用します。

メモ：

Visual Basic 6.0 の既定の測定単位は twip ですが、Visual Basic 2005 の測定単位はピクセルです。

```
' Visual Basic 6.0
Private Sub Form_Paint()
    Dim PixelColor As Long
    Picture1.Picture = LoadPicture("C:\Windows\Greenstone.bmp")
    PixelColor = Picture1.Point(10, 10)
```

```
    FillColor = PixelColor
    Line (0, 0)-(100, 500), PixelColor, B
End Sub
```

VB

```
' Visual Basic 2005
Private Sub Form1_Paint9(ByVal sender As Object, ByVal e As _
System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint

    Dim Pict1 As New Bitmap("C:\Windows\Greenstone.bmp")
    Picture1.Image = Pict1
    Dim PixelColor As Color = Pict1.GetPixel(4, 4)
    Dim PixelBrush As New SolidBrush(PixelColor)
    e.Graphics.FillRectangle(PixelBrush, 0, 0, 100, 100)
End Sub
```

グラフィックス プロパティとメソッドの対応関係

次の表は、Visual Basic 6.0 のグラフィックス プロパティおよびメソッドと、Visual Basic 2005 でそれらに相当するものを示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
AutoRedraw プロパティ	新規に実装されました。グラフィックスを永続化するには、**Paint** イベントにグラフィックス メソッドを記述します。
Circle メソッド	**DrawEllipse** メソッド
ClipControls プロパティ	新規に実装されました。**ClipControls** プロパティは不要になりました。
Cls メソッド	**Clear** メソッド
CurrentX プロパティ	各種グラフィックス メソッドの *x* パラメータ。たとえば、DrawRectangle(pen, *x*, y, width, height)。
CurrentY プロパティ	各種グラフィックス メソッドの *y* パラメータ。たとえば、**DrawRectangle** (pen, x, *y*, width, height)。
DrawMode プロパティ	新規に実装されました。**DrawMode** プロパティは不要になりました。
DrawStyle プロパティ	**DashStyle** プロパティ
DrawWidth プロパティ	Width プロパティ
FillColor プロパティ	**SolidBrush** オブジェクト
FillStyle プロパティ	**HatchBrush** オブジェクト
HasDC プロパティ	新規に実装されました。GDI+ では、デバイス コンテキストは不要になりました。
HDC プロパティ	新規に実装されました。GDI+ では、デバイス コンテキストは不要になりました。
Image プロパティ	新規に実装されました。
Line メソッド	**DrawLine** メソッド
PaintPicture メソッド	**DrawImage** メソッド
Point メソッド	直接対応する項目はありません。ビットマップには、**Bitmap.GetPixel** を使用します。フォームまたはコントロールには、BackColor プロパティを使用します。

Print メソッド	**DrawString** メソッド
Pset メソッド	**DrawEllipse** メソッド、**FillEllipse** メソッド
TextHeight、**TextWidth** プロパティ	**MeasureString** メソッド

アップグレード メモ

アプリケーションを Visual Basic 6.0 から Visual Basic 2005 にアップグレードすると、グラフィックス メソッドはアップグレードされず、警告がコードに挿入されます。GDI と GDI+ では大きな違いがあるので、既存のグラフィックス コードは書き直す必要があります。

参照

処理手順

ユーザー コントロールのカスタム描画のサンプル

その他の技術情報

グラフィックスの概要 (Windows フォーム)
グラフィックス プログラミングについて
GDI+ マネージ コードについて
マネージ グラフィックス クラスの使用

ヘルプ サポート (Visual Basic 6.0 ユーザー向け)

ここでは、Visual Basic 6.0 と Visual Basic 2005 でのヘルプ実装のサポートを比較します。

ほとんどのアプリケーションは、ヘルプ ファイル、ポップアップ ヘルプ、またはツールヒントという形式でヘルプをユーザーに提供します。Visual Basic 6.0 と Visual Basic 2005 は、どちらもすべてのヘルプ機構をサポートしていますが、実装する方法はかなり異なります。

概念の違い

Visual Basic 6.0 では、HTML ヘルプ形式または従来の Windows ヘルプ形式のいずれかを使用してヘルプを提供できました。Visual Basic 2005 では、HTML ヘルプだけがサポートされています。

Visual Basic 6.0 では、[プロジェクト プロパティ] ダイアログ ボックスでヘルプ ファイルの名前を指定することによって、ヘルプ サポートがプロジェクト単位で実装されていました。各フォームとコントロールには、ヘルプ ファイルの特定のトピックへのリンクを提供できる **HelpContextID** プロパティがありました。

Visual Basic 2005 では、1 つ以上の HelpProvider コンポーネントをフォームに追加することにより、ヘルプ サポートをフォーム単位で実装します。各フォームとコントロールには **HelpKeyword** プロパティおよび **HelpNavigator** プロパティがあり、特定のトピックへのリンクに使用できます。詳細については、「方法 : Windows アプリケーションにヘルプを提供する」を参照してください。

ポップアップ ヘルプ

Visual Basic 6.0 では、フォームの **WhatsThisButton** プロパティおよび **WhatsThisHelp** プロパティを使用して、ポップアップ ヘルプが実装されました。**WhatsThisButton** プロパティが **True** で、**MaxButton** プロパティと **MinButton** プロパティの両方が **False** の場合、[説明の表示] ボタンが表示されました。ボタンを有効化するには、コード内で **WhatsThisMode** プロパティを設定する必要がありました。

Visual Basic 2005 では、フォームの HelpButton プロパティを使用して、ポップアップ ヘルプが実装されます。[ヘルプ] ボタンが表示されるのは、**HelpButton** プロパティが **True** に設定され、MaximizeBox プロパティと MinimizeBox プロパティの両方が **False** に設定されている場合だけです。このボタンは自動的に有効になります。詳細については、「方法 : ポップアップ ヘルプを表示する」を参照してください。

ツールヒント

Visual Basic 6.0 では、コントロールの **ToolTipText** プロパティを使用してツールヒントが実装されました。

Visual Basic 2005 では、ToolTip コンポーネントをフォームに追加することにより、ツーツヒントが実装されます。詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。

ShowHelp メソッド

Visual Basic 6.0 では、**CommonDialog** コントロールの **ShowHelp** メソッドを使用して Windows ヘルプを開くことにより、ヘルプを表示することもできました。Windows ヘルプは、Visual Basic 2005 ではサポートされていません。このヘルプを表示するための同等のコントロールはありません。

ヘルプ サポートを扱うコードの変更

次のコード例は、ヘルプ プロパティの一般的な使い方における Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

ヘルプ ファイルの指定

次のコード例は、ヘルプ ファイルをアプリケーションに指定する方法を示しています。ここでは、フォームに 2 つの **OptionButton** コントロールがあり、英語のヘルプ ファイルとフランス語のヘルプ ファイルを選択できると仮定します。

```
' Visual Basic 6.0

If Option1(0).Value = True Then

App.HelpFile = App.Path & "\EnglishHelp.chm"

Else

App.HelpFile = App.Path & "\FrenchHelp.chm"

End If
```

VB

```
' Visual Basic 2005
' Assumes a HelpProvider component has been added to the form.
If RadioButton1.Checked = True Then
    HelpProvider1.HelpNamespace = My.Application.Info.DirectoryPath & _
        "\EnglishHelp.chm"
```

```
    Else
        HelpProvider1.HelpNamespace = My.Application.Info.DirectoryPath & _
            "\FrenchHelp.chm"
    End If
```

ツールヒントの表示

次のコードは、ツールヒントを表示する方法の例を示します。

```
' Visual Basic 6.0

Private Sub Text1_Change()

Text1.ToolTipText = "The text has changed"

End Sub
```

VB

```
    ' Visual Basic 2005
    ' Assumes a ToolTip component has been added to the form.
    Private Sub TextBox1_TextChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TextBox1.TextChanged
        ToolTip1.SetToolTip(TextBox1, "The text has changed")
    End Sub
```

ヘルプ サポートに対応するもの

Visual Basic 6.0	Visual Basic 2005
App.HelpFile プロパティ	**HelpProvider** コンポーネント
HelpContextID プロパティ	SetHelpKeyword メソッド詳細については、「方法 : Windows アプリケーションにヘルプを提供する」を参照してください。
ToolTipText プロパティ	SetToolTip メソッド詳細については、「ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)」を参照してください。
WhatsThisButton プロパティ	**HelpButton** プロパティ
WhatsThisHelp プロパティ	**HelpButton** プロパティ
WhatsThisHelpID プロパティ	HelpString プロパティ
WhatsThisMode プロパティ	対応する項目がありません。既定で有効になります。

アップグレード メモ

アップグレード ウィザードを使用して Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、ヘルプ関連のプロパティまたはコードはアップグレードされません。アップグレード後に、ヘルプ サポートをアプリケーションに再実装する必要があります。HTML ヘルプを使用して作成されたヘルプ ファイルは再利用できますが、Windows ヘルプを使用して作成されたヘルプ ファイルは書き直す必要があります。

参照

処理手順

方法 : Windows アプリケーションにヘルプを提供する

方法 : ポップアップ ヘルプを表示する

概念

ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォームでのヘルプの統合

メニュー処理 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 でメニューを作成したり操作したりするテクニックは、Visual Basic 6.0 の場合と大きく異なります。

概念の違い

Visual Basic 6.0 では、メニュー エディタを使用してメニューを作成しました。メニューをプログラムによって変更または追加することはできましたが、作成することはできませんでした。

Visual Basic 2005 では、MenuStrip コントロールを使ってメニューを作成します。このコントロールは、グラフィカルなメニュー デザイナを備えています。プログラムによってメニューを作成することもできます。

コンテキスト メニュー

Visual Basic 6.0 では、フォームまたはコントロールの **PopupMenu** メソッドを呼び出すことによってコンテキスト メニューを作成しました。プログラムによってコンテキスト メニューを作成することはできず、既存のメニューがなければ **PopupMenu** メソッドは使用できませんでした。

Visual Basic 2005 では、ContextMenuStrip コントロールを使ってコンテキスト メニューを作成します。このコントロールは、グラフィカルなメニュー デザイナを備えています。また、**ContextMenuStrip** クラスの新しいインスタンスを作成することにより、プログラムでコンテキスト メニューを作成することもできます。詳細については、「ContextMenuStrip コントロールの概要」を参照してください。

メニューのマージ

Visual Basic 6.0 の場合、フォーム上のオブジェクトのメニューをフォームのメニューにマージするかどうかは、フォームの **NegotiateMenus** プロパティで設定していました。このプロパティは、デザイン時にだけ設定できました。

Visual Basic 2005 では、メニューのマージには、**MenuStrip** コントロールまたは **ContextMenuStrip** コントロールの `AllowMerge` プロパティを使用できます。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、既存のメニューは MainMenu コンポーネントにアップグレードされます。コンテキスト メニューとして使っていたメニューは、**ContextMenuStrip** コントロールを使うように手動で変更する必要があります。

参照

概念

Menu オブジェクト (Visual Basic 6.0 ユーザー向け)
Windows フォーム (Visual Basic 6.0 ユーザー向け)

その他の技術情報

MenuStrip コントロール (Windows フォーム)

MDI (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 では、どちらもマルチ ドキュメント インターフェイス (MDI: Multiple-Document Interface) アプリケーションを作成できますが、作成する方法と動作の一部に相違点があります。

概念の違い

Visual Basic 6.0 の場合、マルチ ドキュメント インターフェイス アプリケーションは、MDI フォームをプロジェクトに追加し、子フォームの **MDIChild** プロパティを設定することによって作成されます。

Visual Basic 2005 には、MDI フォームはありません。IsMdiContainer プロパティを **True** に設定することにより、任意のフォームを MDI 親フォームにすることができます。

Visual Basic 6.0 の MDI フォームに適用されるプロパティおよびメソッドの多くは、Visual Basic 2005 では変更されています。詳細については、「MDIForm オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。

MDI アプリケーションの動作も変更されています。Visual Basic 6.0 の場合、MDI の子以外のフォームを含む MDI アプリケーションは、MDI の親フォームを閉じても、それ自体のフォームを閉じるまで終了されません。Visual Basic 2005 では、アプリケーションに MDI 以外のフォームが含まれるかどうかに関係なく、スタートアップ フォームを閉じたときにアプリケーションが終了します。

参照

概念

MDIForm オブジェクト (Visual Basic 6.0 ユーザー向け)
Windows フォーム (Visual Basic 6.0 ユーザー向け)

その他の技術情報

マルチ ドキュメント インターフェイス (MDI) アプリケーション
Visual Basic 6.0 ユーザー向けのヘルプ

印刷の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 には、ドキュメント印刷の制御、印刷設定の変更とプリンタの選択、および印刷のプレビューを実現する各種のクラスが用意され、印刷についてのサポートが大幅に強化されています。

概念の違い

Printer オブジェクト

Visual Basic 6.0 では、印刷するには **Printer** オブジェクトを使用します。このオブジェクトは、**Print**、**Line**、**PaintPicture** など、印刷を制御するためのさまざまなグラフィックス メソッドをサポートしています。

Visual Basic 2005 では、**Printer** オブジェクトは PrintDocument コンポーネントに置き換えられています。対応するグラフィックス メソッドは、Graphics クラスの DrawString、DrawLine、および DrawImage メソッドです。

詳細については、「プリンタ オブジェクト (Visual Basic 6.0 ユーザー向け)」を参照してください。

Printers コレクション

Visual Basic 6.0 では、**Printers** コレクションは、**Printer** オブジェクトの出力を別のプリンタに送るときに使用します。**Printers** コレクションには利用できるプリンタの一覧が含まれており、これは Windows の [印刷] ダイアログ ボックスに表示される一覧に対応しています。

Visual Basic 2005 では、**Printers** コレクションが PrintDialog コントロールに置き換えられており、このコントロールが Windows の標準の [印刷] ダイアログ ボックスを表示します。

詳細については、「Printers コレクション (Visual Basic 6.0 ユーザー向け)」を参照してください。

PrintForm メソッド

Visual Basic 6.0 では、フォームのイメージをプリンタに送るときに、フォームの **PrintForm** メソッドを使用します。Visual Basic 2005 では、**PrintForm** メソッドはなくなりました。

PrintForm メソッドの結果は、画面の解像度およびプリンタの解像度に依存するところが大きく、印刷の方法としては推奨されていません。**PrintForm** メソッドの機能を再現する必要がある場合は、サードパーティ製のグラフィックス ツールに用意されているスクリーン キャプチャ機能を自動化してフォームのイメージをキャプチャおよび印刷できます。

印刷プレビュー

Visual Basic 6.0 では、印刷プレビューを実装するためには、サードパーティ製のコントロールを使用するしかありません。Visual Basic 2005 では、PrintPreviewDialog コントロールを使用して Windows 標準の [印刷プレビュー] ダイアログ ボックスを利用できます。詳細については、「PrintPreviewDialog コントロールの概要 (Windows フォーム)」を参照してください。

ページ設定

Visual Basic 6.0 では、ページ設定を行うためのユーザー インターフェイスは用意されていません。Visual Basic 2005 では、PageSetupDialog コントロールを使用して Windows 標準の [ページ設定] ダイアログ ボックスを利用できます。詳細については、「PageSetupDialog コンポーネントの概要 (Windows フォーム)」を参照してください。

ファイルからの印刷

Visual Basic 6.0 では、ファイルからテキストを印刷するには膨大なコードが必要です。Visual Basic 2005 では StreamReader クラスを使用することにより、テキスト ファイルの内容を **PrintDocument** コンポーネントに直接渡すことができます。詳細については、**PrintDocument** クラスの Print メソッドの解説を参照してください。

参照

概念

プリンタ オブジェクト (Visual Basic 6.0 ユーザー向け)
Printers コレクション (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Windows フォームにおける印刷のサポート

ツールヒントのサポート (Visual Basic 6.0 ユーザー向け)

ツールヒントを表示する方法は、Visual Basic 6.0 と Visual Basic 2005 で大きく異なります。

概念の違い

Visual Basic 6.0 では、実行時にコントロールの **ToolTipText** プロパティを使用してツールヒントを表示します。

Visual Basic 2005 では、単一の **ToolTip** コンポーネントを使って、フォーム上のすべてのコントロールについてツールヒントを制御できます。**ToolTip** コンポーネントは、ツールボックスからフォームに追加できます。コントロールの名前に基づいて各ツールヒントにテキストを設定するには、**SetToolTip** メソッドを使用します。

ツールヒントの非表示

Visual Basic 6.0 では、**ToolTipText** プロパティにテキストが含まれる場合にツールヒントが表示され、テキストが含まれない場合はツールヒントは表示されません。複数のツールヒントのテキストをクリアするには、**Controls** コレクションをループして、**ToolTipText** プロパティを空の文字列に設定する必要があります。

Visual Basic 2005 では、空の文字列を **SetToolTip** メソッドに渡すと、ツールヒントは表示されません。**ToolTip** コンポーネントに関連付けられたすべてのツールヒントのテキストをクリアするには、**Active** プロパティを **false** に設定します。

ツールヒントのカスタマイズ

Visual Basic 6.0 では、Windows API 呼び出しに頼らずにツールヒントをカスタマイズする方法はありません。

Visual Basic 2005 では、ツールヒントの外観や動作をカスタマイズできる新しいプロパティがあります。これらのプロパティを使用すると、色の変更、ツールヒント表示の遅延時間の設定、複数行のツールヒント作成などを行うことができます。

ツールヒントを扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

ツールヒントを表示するコードの変更

次のコード例は、Visual Basic 6.0 および Visual Basic 2005 で **Button** コントロールにツールヒントを設定する方法を示します。Visual Basic 2005 の例では、デザイン時にツールヒント コンポーネントがフォームに追加されていると仮定します。

```
' Visual Basic 6.0
Button1.ToolTipText = "Save changes"
```

VB

```
' Visual Basic 2005
ToolTip1.SetToolTip(Button1, "Save changes")
```

ツールヒントを非表示にするコードの変更

次のコード例は、Visual Basic 6.0 および Visual Basic 2005 で **Button** コントロールのツールヒントを非表示にする方法を示します。Visual Basic 2005 の例では、デザイン時にツールヒント コンポーネントがフォームに追加されていると仮定します。

メモ :

Visual Basic 2005 の **ToolTip** コンポーネントには、**Active** プロパティもあります。このプロパティを **false** に設定すると、その **ToolTip** コンポーネントに関連付けられているすべてのコントロールのツールヒントが非表示になります。

```
' Visual Basic 6.0
' Hide a single ToolTip.
Button1.ToolTipText = ""
' Hide all ToolTips.
For Each Control in Me.Controls
    Control.ToolTipText = ""
Next
```

VB

```
' Visual Basic 2005
' Hide a single ToolTip.
ToolTip1.SetToolTip(Button1, "")
' Hide all ToolTips.
ToolTip1.Active = False
```

アップグレード メモ

Visual Basic 6.0 を Visual Basic 2005 にアップグレードすると、デザイン時または実行時の **ToolTipText** プロパティへの参照はアップグレードされず、コードにはコメントが挿入されます。`ToolTip1` という名前の **ToolTip** コンポーネントがフォームに追加されるので、ツールヒントをフックするには、デザイン時に **ToolTip on ToolTip1** プロパティを設定するか、実行時に **SetToolTip** メソッドを呼び出します。

Visual Basic 6.0 アプリケーションでツールヒントをカスタマイズするために Windows API 呼び出しを使っている場合は、コードを変更して、**ToolTip** コンポーネントのカスタマイズ機能を使用する必要があります。

参照

関連項目

ToolTip コンポーネントの概要 (Windows フォーム)

その他の技術情報

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

バージョン番号 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 のバージョン番号は、Visual Basic 6.0 とは異なる方法で割り振られます。また、バージョン番号を設定および取得するメソッドも異なります。

概念の違い

Visual Basic 6.0 では、**App** オブジェクトの **Major**、**Minor**、**Revision** の各プロパティを設定することで、アプリケーションのバージョン番号を制御します。たとえば、**Major** を 1、**Minor** を 2、**Revision** を 3 に設定すると、バージョン番号は 1.2.0.3 となります。3 番目のバージョン番号 (0) は Visual Basic では表示されませんが、Visual Basic 6.0 でコンパイルしたファイルのプロパティを見ると確認できます。

Visual Basic 2005 では、バージョン番号のプロパティは、**AssemblyVersion** 属性に置き換えられています。この属性は、**Major**、**Minor**、**Build**、および **Revision** の 4 つの部分で構成されます。上の例の場合、得られるバージョン番号は 1.2.*x*.4 になります。*x* はビルド番号です。**Revision** 値が、バージョン番号の 4 番目になっていることに注意してください。

バージョン プロパティの設定

Visual Basic 6.0 のバージョン番号プロパティは、[プロジェクトのプロパティ] ダイアログ ボックスで設定しますが、Visual Basic 2005 では [アセンブリ情報] ダイアログ ボックスで設定します。このダイアログ ボックスを表示するには、プロジェクト デザイナの [アプリケーション] タブの [アセンブリ情報] ボタンをクリックします。

メモ：
Visual Basic 6.0 のバージョン番号プロパティは、プロジェクト (.vbp) ファイルに保存され、メモ帳などのテキスト エディタを使って直接編集できます。アセンブリ属性は、AssemblyInfo.vb ファイルに保存され、コード エディタで直接編集できます。

バージョン番号を扱うコードの変更

次のコード例では、Visual Basic 6.0 と Visual Basic 2005 でのコーディング テクニックの違いを示します。

アプリケーションのバージョン番号を表示するコードの変更

次のコード例は、アプリケーションのバージョン番号を取得し、ラベルに表示する方法を示しています。

```
' Visual Basic 6.0

Label1.Caption = "Version: " & App.Major & "." & App.Minor & "." _

& App.Revision
```

VB

```
'Visual Basic 2005
Label1.Text = My.Application.Info.Version.ToString()
```

バージョン番号プロパティの同等物

次の表は、Visual Basic 6.0 のバージョン番号プロパティと Visual Basic 2005 でそれらに対応するものを示しています。

Visual Basic 6.0	**Visual Basic 2005**
同等の項目はありません。	**My.Application.AppInfo.Version.Build**
Major	**My.Application.AppInfo.Version.Major**
Minor	**My.Application.AppInfo.Version.Minor**

Revision	**My.Application.AppInfo.Version.Revision** **メモ：** **Revision** は、Visual Basic 6.0 ではバージョン番号の 4 番目の部分ですが、Visual Basic 2005 では 3 番目です。

アップグレード メモ

アプリケーションを Visual Basic 6.0 からアップグレードすると、**Major** プロパティと **Minor** プロパティだけがアップグレードされます。Visual Basic 2005 は、**Revision** プロパティと **Build** プロパティに新しい値を割り当てます。

参照

概念

App オブジェクト (Visual Basic 6.0 ユーザー向け)

方法：Visual Basic 6.0 ファイル システムのコントロールをアプリケーションに追加する

ファイル システムにアクセスする機能を提供するには、OpenFileDialog コンポーネントおよび SaveFileDialog コンポーネントを使用することをお勧めします。ただし、独自のファイル ダイアログ ボックスを作成する場合、Visual Basic 2005 では、Microsoft Visual Basic 互換性ランタイム ライブラリに含まれている **DirListBox**、**DriveListBox**、および **FileListBox** の各コントロールを使用できます。

DirListBox、**DriveListBox**、または **FileListBox** を Windows アプリケーション プロジェクトに追加するには、以下の手順に従います。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

プロジェクトへのコントロールの追加

互換性ライブラリへの参照を追加するには

1. [プロジェクト] メニューの [参照の追加] をクリックします。
2. [参照の追加] ダイアログ ボックスの [.NET] タブをクリックします。[コンポーネント名] ボックスの一覧の [Microsoft.VisualBasic.Compatibility] をクリックし、[OK] をクリックします。

   プロジェクトに参照が追加されます。

コントロールをツールボックスに追加するには

1. [表示] メニューの [ツールボックス] をクリックします。
2. [すべての Windows フォーム] セクションを展開します。タイトル バーを右クリックし、[アイテムの選択] をクリックします。
3. [ツールボックス アイテムの選択] ダイアログ ボックスの [DirListBox]、[DriveListBox]、および [FileListBox] の各コンポーネントのチェック ボックスをオンにし、[OK] をクリックします。

   コントロールがツールボックスに追加され、他の Windows フォーム コントロールと同じように使用できるようになります。

メモ：
追加したコントロールは、明示的に削除するまでツールボックスに残ります。

参照

関連項目

OpenFileDialog コンポーネントの概要 (Windows フォーム)
SaveFileDialog コンポーネントの概要 (Windows フォーム)

概念

DirListBox コントロール (Visual Basic 6.0 ユーザー向け)
DriveListBox コントロール (Visual Basic 6.0 ユーザー向け)
FileListBox コントロール (Visual Basic 6.0 ユーザー向け)

デバッグ (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 でアプリケーションをデバッグするプロセスは、本質的には Visual Basic 6.0 と同じですが、小さな相違点がいくつかあります。また Visual Basic 2005 には、デバッグに関連する新機能がいくつか追加されています。詳細については、「Visual Basic アプリケーションのデバッグ」を参照してください。

概念の違い

ウォッチの設定

Visual Basic 6.0 では、ウォッチはデザイン時、または中断モードでの実行時に設定できます。Visual Basic 2005 では、ウォッチは中断モードでのみ設定できます。

コンパイル エラー

Visual Basic 6.0 では、コードにコンパイル エラーがあると、デバッグ時にエラー メッセージ ダイアログ ボックスが表示されます。

Visual Basic 2005 では、コンパイル エラーは、コード エディタ ウィンドウ内で青色の下線付きで強調表示されます。エラーを修正する前にアプリケーションをデバッグしようとすると、ビルド エラーの警告が表示され、[タスク一覧] ウィンドウにエラーが表示されます。

ランタイム例外

Visual Basic 6.0 では、デバッグ時に例外が発生すると、モーダル ダイアログ ボックスが表示され、エラー番号が表示されます。ダイアログ ボックスを閉じると、コード エディタに問題のコード行が表示されます。

Visual Basic 2005 では、例外が発生するとコード エディタに、モーダルではない [例外処理アシスタント] が表示されます。[例外処理アシスタント] には、例外の種類、トラブルシューティングのヒント、および対処方法が表示されます。詳細については、「例外処理アシスタント」を参照してください。

デバッグ用のショートカット キー

Visual Studio では、デバッグ用のいくつかのショートカット キーは、言語によってそれぞれ異なります。環境に合わせて設定をカスタマイズできます。

Visual Basic 2005 を初めて実行すると、開発設定を選択するよう要求されます。[Visual Basic 開発設定] を選択すると、ショートカット キーは Visual Basic 6.0 で使用されるショートカット キーに一致します。

メモ :
ショートカット キーが Visual Basic 6.0 設定に一致しない場合は、[ツール] メニューから使用できる設定のインポートとエクスポート ウィザードにある [Visual Basic 開発設定] を適用します。詳細については、「方法 : 選択した設定を変更する」を参照してください。

参照

概念

Visual Basic アプリケーションのデバッグ
統合開発環境 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Visual Basic 6.0 ユーザー向けのヘルプ

セットアップと配置 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、アプリケーションおよびコンポーネントを配置する方法が Visual Basic 6.0 の場合と大きく異なります。

概念の違い

Visual Basic 6.0 では、作成したアプリケーションを配布およびインストールするためのセットアップ プログラム (.exe ファイル) を作成するには、パッケージおよび配置ウィザードを使用します。

メモ：
Visual Studio 6.0 の後期リリースには、Windows インストーラ ファイル (.msi ファイル) を作成するときに使用する Visual Studio インストーラ アドインも含まれています。Visual Studio インストーラで作成されるプロジェクトには、Visual Basic 2005 との互換性はありません。

Visual Basic 2005 には、Windows ベースのアプリケーションを配置するための 2 つのストラテジが用意されています。1 つは ClickOnce テクノロジを使用してアプリケーションを発行する方法であり、もう 1 つは Windows インストーラ テクノロジによる従来のセットアップによってアプリケーションを配置する方法です。

ClickOnce による配置では、開発者がアプリケーションを一元的な場所 (通常は Web サーバーやファイル共有) に発行し、ユーザーはその場所からアプリケーションをインストールまたは実行します。ClickOnce アプリケーションは自己更新が可能です。開発者がアプリケーションに対する更新を発行すると、エンドユーザーはその更新を自動的にダウンロードできます。これによって、すべてのユーザーが同じバージョンを使用することを保証できます。詳細については、「ClickOnce の配置」を参照してください。

Windows インストーラによる配置では、開発者が [セットアップと配置] プロジェクトを使用してアプリケーションを Setup.exe ファイルにパッケージ化し、そのファイルをユーザーに配布します。ユーザーは、Setup.exe ファイルを使用してアプリケーションをインストールします。詳細については、「Windows インストーラ配置」を参照してください。

メモ：
Visual Basic Express Edition では、ClickOnce による配置のみがサポートされます。

必要条件と依存関係

Visual Basic 6.0 では、アプリケーションのファイル依存関係に関する情報は依存関係 (.dep) ファイルに格納されます。すべてのアプリケーションにとって Visual Basic ランタイム ファイルは必要条件であり、データ ライブラリなど、その他の必要条件が要求されるアプリケーションも数多くあります。依存関係と必要条件の完全なリストを決定するには、Vb6dep.ini ファイルを手動で編集することがしばしば必要です。

Visual Basic 2005 では、ClickOnce による配置と Windows インストーラによる配置のいずれの場合も、アプリケーションのすべての依存関係が自動的に判断され、パッケージ化されます。どちらの配置の場合も、.NET Framework ランタイム ライブラリなどの前提条件も自動的にインストールされます。その他の必要条件をブートストラップ (アプリケーションと共にインストール) することもできます。詳細については、「配置の必要条件 (Visual Studio)」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、配置情報は失われます。ClickOnce または Windows インストーラを使用して、新しい配置を実装する必要があります。

参照

概念

配置ストラテジの選択

その他の技術情報

アプリケーションとコンポーネントの配置

ローカリゼーションとグローバリゼーション (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Visual Basic 2005 は、国際対応のアプリケーションをそれぞれサポートします。ただし、アプリケーションのローカライズとグローバライズの概念とテクニックは異なります。

概念の違い

リソースのローカライズ

Visual Basic 6.0 では、アプリケーションの国際対応バージョンを作成するときに、ローカライズ可能な情報 (文字列など) を言語ごとに個別のリソース ファイル (.res) に格納します。ロケール固有のリソースは、実行時に **LoadResString**、**LoadResPicture**、および **LoadResData** の各関数を呼び出すことによって、リソース ファイルから読み込まれます。

Visual Basic 2005 では、デザイン時にフォームの **Language** プロパティを変更することによって、アプリケーションの国際対応バージョンを作成します。個別のリソース ファイル (.resx) は、選択された各ロケールに対して自動的に作成されます。リソースは、ユーザーのロケールに応じて自動的に読み込まれるため、コードから明示的に読み込む必要がなくなりました。詳細については、「Windows フォームのグローバル化」を参照してください。

リソースの編集

Visual Basic 6.0 のリソース ファイルは、リソース エディタ アドインまたは Visual C++ のリソース エディタを使用して編集できます。

Visual Basic 2005 では、リソース エディタは、プロジェクト デザイナの一部として IDE に組み込まれています。詳細については、「アプリケーション リソースの管理」を参照してください。

Unicode

Visual Basic 6.0 では、文字列を内部的には Unicode 文字として表現しますが、実際の表示には Windows コード ページが使用されます。ANSI コード ページと DBCS コード ページの間で文字列を変換するには、**StrConv** 関数と一緒に、文字列操作関数のバイナリ バージョンと Unicode バージョン (**ChrB** や **ChrW** など) を使用する必要があります。

Visual Basic 2005 では、フォームが Unicode に完全に対応しているため、コード ページ間の変換は不要です。詳細については、「エンコード方式および Windows フォームのグローバリゼーション」を参照してください。

日付と通貨の形式

Visual Basic 6.0 では、コードにおける日付と通貨の形式に細心の注意を払う必要があります。これは、テキストとして入力した値が、ローカライズされたアプリケーションで日付や通貨に変換されるときに、誤って解釈される可能性があるためです。

Visual Basic 2005 では、ユーザーのカルチャに応じて日付と通貨の形式が自動的に設定されます。この設定は **System.Globalization** 名前空間の関数を使用して、必要に応じてオーバーライドできます。詳細については、「グローバルな Windows フォームおよび Web フォームにおけるカルチャ固有のクラス」を参照してください。

参照

処理手順

方法 : Visual Basic で、ローカライズされたリソースを取得する

概念

.NET Framework ベースの国際対応アプリケーションの概要

配置とローカリゼーション

その他の技術情報

アプリケーションのグローバライズとローカライズ

Visual Studio のグローバリゼーションおよびローカリゼーションの名前空間

Windows API プログラミング (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 では、通常、Windows API を使用する必要はありません。また、Windows API を呼び出す方法は、Visual Basic 6.0 の場合と多少異なります。

概念の違い

Visual Basic 6.0 では、Visual Basic 自体で提供されない高度な機能を実現するために、Windows API をしばしば呼び出す必要がありました。

Visual Basic 2005 には .NET Framework が組み込まれています。Windows API の機能の大部分は .NET Framework にラップされているため、Windows API の呼び出しは必要ありません。

Windows API の宣言

Visual Basic 6.0 の場合、Visual Basic では利用できない機能にアクセスするために Windows API を呼び出すときに **Declare** ステートメントが使用されます。

Visual Basic 2005 でも **Declare** ステートメントを使用して API を呼び出すことはできますが、注意が必要ないくつかの相違があります。たとえば、Visual Basic 2005 の一部のデータ型では、直接対応する型が存在しない場合があります。したがって、これらのデータ型は、マーシャリングしてから API 呼び出しに渡す必要があります。詳細については、「チュートリアル : Windows API の呼び出し」を参照してください。

アップグレード メモ

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすると、Windows API への呼び出しも適切にアップグレードされます。.NET Framework における同等の項目に置き換えられるわけではありません。データ型の問題が発生する場合は、コードとアップグレード レポートにアップグレードの警告が追加されます。ほとんどの場合、これらの API 呼び出しは .NET Framework における同等の項目に置き換える必要があります。

参照

関連項目

Declare ステートメント

概念

データ型の変更点 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Visual Basic 6.0 ユーザー向けのヘルプ

レジストリ アクセス (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 におけるレジストリ アクセスの方法は、Visual Basic 6.0 の場合と異なります。

概念の違い

Visual Basic 6.0 では、**GetSetting** 関数または **SaveSetting** 関数を使用してレジストリにアクセスできますが、これらの関数でアクセスできるのは一部のレジストリ キーだけに限られています。その他のレジストリ キーにアクセスするには、Windows API を呼び出す必要があります。

Visual Basic 2005 では **My.Computer.Registry** オブジェクトが用意されており、これを使用してレジストリ キーにアクセスします。詳細については、「My を使用したレジストリからの読み取りとレジストリへの書き込み」を参照してください。

NET Framework では、**Registry** クラスを使用することによりすべてのレジストリにアクセスできます。

メモ：
Visual Basic 2005 でも、**GetSetting** 関数と **SaveSetting** 関数で一部のレジストリにアクセスできます。詳細については、「レジストリの概要」を参照してください。

レジストリ アクセスおよび配置

Visual Basic 6.0 では、Setup.lst ファイルを手動で編集することで、インストール時に登録するファイルをマーキングできます。

Visual Basic 2005 では、セットアップ プロジェクトと配置プロジェクトによって、インストール時のレジストリ操作を完全にサポートできます。詳細については、「配置とレジストリ」を参照してください。

参照

処理手順

チュートリアル : レジストリ キーの作成と値の変更
トラブルシューティング : レジストリの操作

関連項目

Registry Class
My.Computer.Registry オブジェクト

概念

一般的なレジストリ タスク
My を使用したレジストリからの読み取りとレジストリへの書き込み
Microsoft.Win32 名前空間を使用したレジストリの読み取りと書き込み
セキュリティとレジストリ

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 では、Visual Basic 言語の多数の定数が定義されていました。Visual Basic 2005 では、大部分の定数が .NET Framework の列挙体に置き換えられています。多くの場合、Visual Basic 6.0 定数をそのまま使用することもできます。

Visual Basic 6.0 の定数に相当する Visual Basic 2005 の定数を示すトピックへのリンクを次に示します。

メモ：
一覧に定数がない場合は、対応する定数がないことを示します。たとえば、Visual Basic 2005 では DDE (ダイナミック データ エクスチェンジ) がサポートされないため、一覧に DDE はありません。

このセクションの内容

Align 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Align** 定数と、相当する Visual Basic 2005 の定数を示します。

Alignment 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Alignment** 定数と、相当する Visual Basic 2005 の定数を示します。

Arrange 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Arrange** 定数と、相当する Visual Basic 2005 の定数を示します。

BorderStyle 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **BorderStyle** 定数と、相当する Visual Basic 2005 の定数を示します。

CallType 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **CallType** 定数と、相当する Visual Basic 2005 の定数を示します。

CheckBox 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **CheckBox** 定数と、相当する Visual Basic 2005 の定数を示します。

Color 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Color** 定数と、相当する Visual Basic 2005 の定数を示します。

CompareMethod 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **CompareMethod** 定数と、相当する Visual Basic 2005 の定数を示します。

Date Format 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **DateFormat** 定数と、相当する Visual Basic 2005 の定数を示します。

DayOfWeek 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **DayOfWeek** 定数と、相当する Visual Basic 2005 の定数を示します。

File Attribute 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **FileAttribute** 定数と、相当する Visual Basic 2005 の定数を示します。

IMEMode 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **IMEMode** 定数と、相当する Visual Basic 2005 の定数を示します。

Key Code 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Key Code** 定数と、相当する Visual Basic 2005 の定数を示します。

Mouse Pointer 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Mouse Pointer** 定数と、相当する Visual Basic 2005 の定数を示します。

MsgBox Result定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **MsgBox Result** 定数と、相当する Visual Basic 2005 の定数を示します。

MsgBox Style 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **MsgBox Style** 定数と、相当する Visual Basic 2005 の定数を示します。

MultiSelect 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **MultiSelect** 定数と、相当する Visual Basic 2005 の定数を示します。

OLEDropEffect 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **OLEDropEffect** 定数と、相当する Visual Basic 2005 の定数を示します。

ScrollBar 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **ScrollBar** 定数と、相当する Visual Basic 2005 の定数を示します。

StartUpPosition 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **StartUpPosition** 定数と、相当する Visual Basic 2005 の定数を示します。

StrConv 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **StrConv** 定数と、相当する Visual Basic 2005 の定数を示します。

Style 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Style** 定数と、相当する Visual Basic 2005 の定数を示します。

System Color 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **System Color** 定数と、相当する Visual Basic 2005 の定数を示します。

TriState 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **TriState** 定数と、相当する Visual Basic 2005 の定数を示します。

バリアント型の定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Variant Type** 定数と、相当する Visual Basic 2005 の定数を示します。

VarType 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **VarType** 定数と、相当する Visual Basic 2005 の定数を示します。

VBA 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の Visual Basic for Applications 定数と、相当する Visual Basic 2005 の定数を示します。

WeekOfYear 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **WeekOfYear** 定数と、相当する Visual Basic 2005 の定数を示します。

WindowState 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **WindowState** 定数と、相当する Visual Basic 2005 の定数を示します。

WindowStyle 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Window Style** 定数と、相当する Visual Basic 2005 の定数を示します。

関連するセクション

Visual Basic の定数と列挙体

Visual Basic 2005 の列挙体を示します。

Windows フォーム コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 と Windows フォームでの、コントロールおよびオブジェクトの違いを説明するトピックへのリンクを提供します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 言語に加えられた変更点について説明します。

Align 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Align** 定数および値と、Visual Basic 2005 でそれらに相当する定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
VbAlignNone (0)	None
VbAlignTop (1)	Top
VbAlignBottom (2)	Bottom
VbAlignLeft (3)	Left
VbAlignRight (4)	Right

参照

概念

PictureBox コントロール (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

Alignment 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Alignment** 定数および値と、Visual Basic 2005 でそれらに相当する定数および値を以下の表に示します。

CheckBox および OptionButton の Alignment 定数

Visual Basic 6.0	**Visual Basic 2005**
VbLeftJustify (0)	MiddleLeft
VbRightJustify (1)	MiddleRight

TextBox の Alignment 定数

Visual Basic 6.0	**Visual Basic 2005**
VbLeftJustify (0)	Left
VbRightJustify (1)	Right
VbCenter (2)	Center

Label の Alignment 定数

Visual Basic 6.0	**Visual Basic 2005**
VbLeftJustify (0)	TopLeft
VbRightJustify (1)	TopRight
VbCenter (2)	TopCenter

参照

概念

CheckBox コントロール (Visual Basic 6.0 ユーザー向け)
OptionButton コントロール (Visual Basic 6.0 ユーザー向け)
TextBox コントロール (Visual Basic 6.0 ユーザー向け)
Label コントロール (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)
Visual Basic の定数と列挙体

Arrange 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Arrange** 定数および値に相当する Visual Basic 2005 の定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
VbCascade (0)	Cascade
VbTileHorizontal (1)	TileHorizontal
VbTileVertical (2)	TileVertical
VbArrangeIcons (3)	ArrangeIcons

参照
概念
MDIForm オブジェクト (Visual Basic 6.0 ユーザー向け)
その他の技術情報
定数の対応関係 (Visual Basic 6.0 ユーザー向け)
Visual Basic の定数と列挙体

BorderStyle 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **BorderStyle** 定数および値に相当する Visual Basic 2005 の定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 で相当する要素
VbBSNone (0)	None
VbFixedSingle (1)	FixedSingle
VbSizable (2)	Sizable
VbFixedDialog (3)	FixedDialog
VbFixedToolWindow (4)	FixedToolWindow
VbSizableToolWindow (5)	SizableToolWindow

参照
概念
Form オブジェクト (Visual Basic 6.0 ユーザー向け)
その他の技術情報
定数の対応関係 (Visual Basic 6.0 ユーザー向け)
Visual Basic の定数と列挙体

CallType 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **CallType** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するために、Visual Basic 6.0 の **CallType** 定数を Visual Basic 2005 のコードでも使用できます。詳細については、「CallType 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbMethod	**CallType.Method**
vbGet	**CallType.Get**
vbSet	**CallType.Set**
vbLet	**CallType.Let**

参照

関連項目

CallByName 関数

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

CheckBox 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **CheckBox** 定数および値と、Visual Basic 2005 でそれらに相当する定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbUnchecked (0)	Unchecked
vbChecked (1)	Checked
vbGrayed (2)	Indeterminate

参照

概念

CheckBox コントロール (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

Color 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **Color** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbBlack	Black
vbRed	Red
vbGreen	Lime
vbYellow	Yellow
vbBlue	Blue
vbMagenta	Magenta
vbCyan	Cyan
vbWhite	White

参照

関連項目

色の処理 (Visual Basic 6.0 ユーザー向け)

System Color 定数 (Visual Basic 6.0 ユーザー向け)

概念

色の動作 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

定数と列挙型 (Visual Basic)

CompareMethod 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **Compare** 定数とそれらに対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するため、Visual Basic 6.0 の **Compare** 定数は Visual Basic 2005 でも使用できます。詳細については、「CompareMethod 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 で相当する要素
vbBinaryCompare	**CompareMethod.Binary**
vbTextCompare	**CompareMethod.Text**
vbDatabaseCompare	対応要素なし

参照

関連項目

CompareMethod 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

Date Format 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **Date Format** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するために、Visual Basic 6.0 の **Date Format** 定数を Visual Basic 2005 のコードでも使用できます。詳細については、「DateFormat 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbGeneralDate	**DateFormat.GeneralDate**
vbLongDate	**DateFormat.LongDate**
vbShortDate	**DateFormat.ShortDate**
vbLongTime	**DateFormat.LongTime**
vbShortTime	**DateFormat.ShortTime**

参照

関連項目

DateFormat 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

DayOfWeek 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **DayOfWeek** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するため、Visual Basic 6.0 の **DayOfWeek** 定数は Visual Basic 2005 のコードでも使用できます。詳細については、「FirstDayOfWeek Enumeration」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
VbUseSystemDayOfWeek	**FirstDayOfWeek.System**
vbSunday	**FirstDayOfWeek.Sunday**
vbMonday	**FirstDayOfWeek.Monday**
vbTuesday	**FirstDayOfWeek.Tuesday**
vbWednesday	**FirstDayOfWeek.Wednesday**
vbThursday	**FirstDayOfWeek.Thursday**
vbFriday	**FirstDayOfWeek.Friday**
vbSaturday	**FirstDayOfWeek.Saturday**

参照

関連項目

FirstDayOfWeek Enumeration

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

File Attribute 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **File Attribute** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するために、Visual Basic 6.0 の **File Attribute** 定数を Visual Basic 2005 のコードでも使用できます。詳細については、「FileAttribute 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbNormal	**FileAttribute.Normal**
vbReadOnly	**FileAttribute.ReadOnly**
vbHidden	**FileAttribute.Hidden**
vbSystem	**FileAttribute.System**
vbVolume	**FileAttribute.Volume**
vbDirectory	**FileAttribute.Directory**
vbArchive	**FileAttribute.Archive**
vbAlias	同等の項目はありません。

参照

関連項目

FileAttribute 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

IMEMode 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **IMEMode** 定数および値に相当する Visual Basic 2005 の定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbIMEAlphaDbl (7)	AlphaFull
vbIMEAlphaSng (8)	Alpha
vbIMEDisable (3)	Disable
vbIMEHiragana (4)	Hiragana
vbIMEKatakanaDbl (5)	Katakana
vbIMEKatakanaSng (6)	KatakanaHalf
vbIMEModeAlpha (8)	**Alpha**
vbIMEModeAlphaFull (7)	**AlphaFull**
vbIMEModeDisable (3)	**Disable**
vbIMEModeHangul (10)	Hangul
vbIMEModeHangulFull (9)	HangulFull
vbIMEModeHiragana (4)	**Hiragana**
vbIMEModeKatakana (5)	**Katakana**
vbIMEModeKatakanaHalf (6)	**KatakanaHalf**
vbIMEModeNoControl (0)	NoControl
vbIMEModeOff (2)	Off
vbIMEModeOn (1)	On
vbIMENoOp (0)	**NoControl**
vbIMEOff (2)	**Off**
vbIMEOn (1)	**On**

参照
概念
Form オブジェクト (Visual Basic 6.0 ユーザー向け)
その他の技術情報
定数の対応関係 (Visual Basic 6.0 ユーザー向け)
Visual Basic の定数と列挙体

Key Code 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Key Code** 定数および値と、Visual Basic 2005 でそれらに相当する定数および値を以下の表に示します。

Key Code 定数

Visual Basic 6.0	Visual Basic 2005 での同等物
vbKeyLButton (1)	LButton
vbKeyRButton (2)	RButton
vbKeyCancel (3)	Cancel
vbKeyMButton (4)	MButton
vbKeyBack (8)	Back
vbKeyTab (9)	Tab
vbKeyClear (12)	Clear
vbKeyReturn (13)	Return
vbKeyShift (16)	ShiftKey
vbKeyControl (17)	ControlKey
vbKeyMenu (18)	Menu
vbKeyPause (19)	Pause
vbKeyCapital (20)	Capital
vbKeyEscape (27)	Escape
vbKeySpace (32)	Space
vbKeyPageUp (33)	PageUp
vbKeyPageDown (34)	PageDown
vbKeyEnd (35)	End
vbKeyHome (36)	Home
vbKeyLeft (37)	Left
vbKeyUp (38)	Up
vbKeyRight (39)	Right
vbKeyDown (40)	Down

vbKeySelect (41)	Select
vbKeyPrint (42)	Print
vbKeyExecute (43)	Execute
vbKeySnapshot (44)	Snapshot
vbKeyInsert (45)	Insert
vbKeyDelete (46)	Delete
vbKeyHelp (47)	Help
vbKeyNumlock (144)	Numlock
vbKeyScrollLock (145)	Scroll

アルファベット Key Code 定数

Visual Basic 6.0	**Visual Basic 2005 での同等物**
VbKeyA (65)	A
VbKeyB (66)	B
VbKeyC (67)	C
VbKeyD (68)	D
VbKeyE (69)	E
VbKeyF (70)	F
VbKeyG (71)	G
VbKeyH (72)	H
VbKeyI (73)	I
VbKeyJ (74)	J
VbKeyK (75)	K
VbKeyL(76)	L
vbKeyM (77)	M
VbKeyN (78)	N
VbKeyO (79)	O
VbKeyP(80)	P
VbKeyQ (81)	Q

VbKeyR (82)	R
VbKeyS (83)	S
VbKeyT (84)	T
VbKeyU (85)	U
VbKeyV (86)	V
VbKeyW (87)	W
VbKeyX (88)	X
VbKeyY (89)	Y
VbKeyZ (90)	Z

数値キー定数

Visual Basic 6.0	Visual Basic 2005 での同等物
vbKey0 (48)	D0
vbKey1 (49)	D1
vbKey2 (50)	D2
vbKey3 (51)	D3
vbKey4 (52)	D4
vbKey5 (53)	D5
vbKey6 (54)	D6
vbKey7 (55)	D7
vbKey8 (56)	D8
vbKey9 (57)	D9

テンキーのキー定数

Visual Basic 6.0	Visual Basic 2005 での同等物
vbKeyNumpad0 (96)	NumPad0
vbKeyNumpad1 (97)	NumPad1
vbKeyNumpad2 (98)	NumPad2
vbKeyNumpad3 (99)	NumPad3
vbKeyNumpad4 (100)	NumPad4
vbKeyNumpad5 (101)	NumPad5

vbKeyNumpad6 (102)	NumPad6
vbKeyNumpad7 (103)	NumPad7
vbKeyNumpad8 (104)	NumPad8
vbKeyNumpad9 (105)	NumPad9
VbKeyMultiply (106)	Multiply
VbKeyAdd (107)	Add
VbKeySeparator (108)	Separator
VbKeySubtract (109)	Subtract
VbKeyDecimal (110)	Decimal
VbKeyDivide (111)	Divide

ファンクション キーの定数

Visual Basic 6.0	Visual Basic 2005 での同等物
vbKeyF1 (112)	F1
vbKeyF2 (113)	F2
vbKeyF3 (114)	F3
vbKeyF4 (115)	F4
vbKeyF5 (116)	F5
vbKeyF6 (117)	F6
vbKeyF7 (118)	F7
vbKeyF8 (119)	F8
vbKeyF9 (120)	F9
vbKeyF10 (121)	F10
vbKeyF11 (122)	F11
vbKeyF12 (123)	F12
vbKeyF13 (124)	F13
vbKeyF14 (125)	F14
vbKeyF15 (126)	F15
vbKeyF16 (127)	F16

参照

関連項目

Keys Enumeration

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

Mouse Pointer 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Mouse Pointer** 定数および値と、Visual Basic 2005 でそれらに相当する定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
VbDefault (0)	Default
VbArrow (1)	Arrow
VbCrosshair (2)	Cross
VbIbeam (3)	IBeam
VbIconPointer (4)	互換性のために残されています。**Default** に置き換えられています。
VbSizePointer (5)	SizeAll
VbSizeNESW (6)	SizeNESW
VbSizeNS (7)	SizeNS
VbSizeNWSE (8)	SizeNWSE
VbSizeWE (9)	SizeWE
VbUpArrow (10)	UpArrow
VbHourglass (11)	WaitCursor
VbNoDrop (12)	No
VbArrowHourglass (13)	AppStarting
VbArrowQuestion (14)	Help
VbSizeAll (15)	**SizeAll**
VbCustom (99) カスタム MousePointer を設定できない	対応するキーワードなし詳細については、「カスタム MousePointer を設定できない」を参照してください。

参照

概念

MousePointer (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

MsgBox Result定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **MsgBox Result** 定数および値と、それらに相当する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するために、Visual Basic 6.0 の **MsgBox Result** 定数を Visual Basic 2005 のコードでも使用できます。詳細については、「MsgBoxResult 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbOK	**MsgBoxResult.OK**
vbCancel	**MsgBoxResult.Cancel**
vbAbort	**MsgBoxResult.Abort**
vbRetry	**MsgBoxResult.Retry**
vbIgnore	**MsgBoxResult.Ignore**
vbYes	**MsgBoxResult.Yes**
vbNo	**MsgBoxResult.No**

参照

関連項目

MsgBoxResult 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

MsgBox Style 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **MsgBox Style** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するために、Visual Basic 6.0 の **MsgBox Style** 定数を Visual Basic 2005 のコードでも使用できます。詳細については、「MsgBoxStyle 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbOKOnly	**MsgBoxStyle.OKOnly**
vbOKCancel	**MsgBoxStyle.OKCancel**
vbAbortRetryIgnore	**MsgBoxStyle.AbortRetryIgnore**
vbYesNoCancel	**MsgBoxStyle.YesNoCancel**
vbYesNo	**MsgBoxStyle.YesNo**
vbRetryCancel	**MsgBoxStyle.RetryCancel**
vbCritical	**MsgBoxStyle.Critical**
vbQuestion	**MsgBoxStyle.Question**
vbExclamation	**MsgBoxStyle.Exclamation**
vbInformation	**MsgBoxStyle.Information**
vbDefaultButton1	**MsgBoxStyle.DefaultButton1**
vbDefaultButton2	**MsgBoxStyle.DefaultButton2**
vbDefaultButton3	**MsgBoxStyle.DefaultButton3**
vbDefaultButton4	同等の項目はありません。
vbApplicationModal	**MsgBoxStyle.ApplicationModal**
vbSystemModal	**MsgBoxStyle.SystemModal**
vbMsgBoxHelpButton	**MsgBoxStyle.MsgBoxHelp**
vbMsgBoxRight	**MsgBoxStyle.MsgBoxRight**
vbMsgBoxRtlReading	**MsgBoxStyle.MsgBoxRtlReading**
vbMsgBoxSetForeground	**MsgBoxStyle.MsgBoxSetForeground**

参照

関連項目

MsgBoxStyle 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)
Visual Basic の定数と列挙体

MultiSelect 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **MultiSelect** 定数および値に相当する Visual Basic 2005 の定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbMultiSelectNone (0)	One
vbMultiSelectSimple (1)	MultiSimple
vbvbMultiSelectExtended (2)	MultiExtended

参照

概念

ListBox コントロール (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

OLEDropEffect 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **OLEDropEffect** 定数および値に相当する Visual Basic 2005 の定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
VbDropEffectNone (0)	None
VbDropEffectCopy (1)	Copy
VbDropEffectMove (2)	Move
VbDropEffectScroll (&H80000000)	Scroll

参照

概念

ドラッグ アンド ドロップ (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

ScrollBar 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **ScrollBar** 定数および値と、Visual Basic 2005 でそれらに相当する定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbSBNone (0)	None
vbHorizontal (1)	Horizontal
vbVertical (2)	Vertical
vbBoth (3)	Both

参照

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

PictureBox コントロール (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

StartUpPosition 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **StartUpPosition** 定数および値と、Visual Basic 2005 でそれらに相当する定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbStartUpManual (0)	Manual
vbStartUpOwner (1)	CenterParent
vbStartUpScreen (2)	CenterScreen
vbStartUpWindowsDefault (3)	WindowsDefaultLocation

参照

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

StrConv 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **StrConv** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するために、Visual Basic 6.0 の **StrConv** 定数を Visual Basic 2005 のコードでも使用できます。詳細については、「VbStrConv 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbUpperCase	**VbStrConv.UpperCase**
vbLowerCase	**VbStrConv.LowerCase**
vbProperCase	**VbStrConv.ProperCase**
vbWide	**VbStrConv.Wide**
vbNarrow	**VbStrConv.Narrow**
vbKatakana	**VbStrConv.Katakana**
vbHiragana	**VbStrConv.Hiragana**
vbUnicode	相当する定数はありません。Convert メソッドを使用して、同じ機能を実行できます。
vbFromUnicode	同等の項目はありません。

参照

関連項目

VbStrConv 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

Style 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **Style** 定数および値と、Visual Basic 2005 でそれらに相当する定数および値を以下の表に示します。

ComboBox の Style 定数

Visual Basic 6.0	**Visual Basic 2005 での同等物**
vbComboDropdown (0)	DropDown
vbComboSimple (1)	Simple
vbComboDropdownList (2)	DropDownList

CheckBox および OptionButton の Style 定数

Visual Basic 6.0	**Visual Basic 2005 での同等物**
VbButtonStandard (0)	Normal
VbButtonGraphical (1)	Button

メモ :

Visual Basic 6.0 では、**Style** 定数は、**CommandButton** コントロールと **ListBox** コントロールにも定義されています。Visual Basic 2005 には、これに相当するものはありません。詳細については、
「方法 : アップグレードしたアプリケーションで Visual Basic 6.0 の 3 ステート コントロールをエミュレートする」または
「ListBox コントロール (Visual Basic 6.0 ユーザー向け)」を参照してください。

参照

概念

ComboBox コントロール (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

System Color 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **System Color** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

Visual Basic 6.0	Visual Basic 2005 での同等物
vb3DDKShadow	ControlDarkDark
vb3DFace	Control
vb3DHighlight	ControlLightLight
vb3DLight	ControlLight
vb3DShadow	ControlDark
vbActiveBorder	ActiveBorder
vbActiveTitleBar	ActiveCaption
vbActiveTitleBarText	ActiveCaptionText
vbApplicationWorkspace	AppWorkspace
vbButtonFace	**Control**
vbButtonShadow	**ControlDark**
vbButtonText	ControlText
vbDesktop	Desktop
vbGrayText	GrayText
vbHighlight	Highlight
vbHighlightText	HighlightText
vbInactiveBorder	InactiveBorder
vbInactiveCaptionText	InactiveCaptionText
vbInactiveTitleBar	InactiveCaption
vbInactiveTitleBarText	**InactiveCaptionText**
vbInfoBackground	Info
vbInfoText	InfoText
vbMenuBar	Menu
vbMenuText	MenuText

vbScrollBars	Scrollbar
vbTitleBarText	**ActiveCaptionText**
vbWindowBackground	Window
vbWindowFrame	WindowFrame
vbWindowText	WindowText

参照

関連項目

色の処理 (Visual Basic 6.0 ユーザー向け)

Color 定数 (Visual Basic 6.0 ユーザー向け)

概念

色の動作 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

TriState 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **TriState** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するため、Visual Basic 6.0 の **TriState** 定数は Visual Basic 2005 でも使用できます。詳細については、「TriState 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbTrue	**TriState.True**
vbFalse	**TriState.False**
vbUseDefault	**TriState.UseDefault**

参照

関連項目

TriState 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

バリアント型の定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **VarType** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbVEmpty	**VariantType.Empty**
vbVNull	**VariantType.Null**
vbVInteger	**VariantType.Short**
vbVLong	**VariantType.Integer**
vbVSingle	**VariantType.Single**
vbVDouble	**VariantType.Double**
vbVCurrency	**VariantType.Decimal**
vbVDate	**VariantType.Date**
vbVString	**VariantType.String**

参照

関連項目

VariantType 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

定数と列挙型 (Visual Basic)

VarType 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **VarType** 定数と値、およびそれらに対応する Visual Basic 2005 の VariantType 列挙型 の定数を示しています。

メモ：
下位互換性を維持するため、Visual Basic 6.0 の **VarType** 定数は Visual Basic 2005 でも使用できます。詳細については、「VariantType 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbEmpty	**VariantType.Empty**
vbNull	**VariantType.Null**
vbInteger	**VariantType.Short**
vbLong	**VariantType.Integer**
vbSingle	**VariantType.Single**
vbDouble	**VariantType.Double**
vbCurrency	**VariantType.Decimal**
vbDate	**VariantType.Date**
vbString	**VariantType.String**
vbObject	**VariantType.Object**
vbError	**VariantType.Error**
vbBoolean	**VariantType.Boolean**
vbVariant	**VariantType.Variant**
vbDataObject	**VariantType.DataObject**
vbDecimal	**VariantType.Decimal**
vbByte	**VariantType.Byte**
vbUserDefinedType	**VariantType.UserDefinedType**
vbArray	**VariantType.Array**

参照

関連項目

VariantType 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

VBA 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の Microsoft Visual Basic for Applications (VBA) 定数に相当する Visual Basic 2005 の定数を以下の表に示します。

メモ：
下位互換性を維持するため、Visual Basic 6.0 VBA 定数は Visual Basic 2005 でも使用できます。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbBack	**ControlChars.Back**
vbCr	**ControlChars.Cr**
vbCrLf	**ControlChars.CrLf**
vbFormFeed	**ControlChars.FormFeed**
vbLf	**ControlChars.Lf**
vbNewLine	**ControlChars.NewLine**
vbNullChar	**ControlChars.NullChar**
vbNullString	**Nothing** キーワード
vbObjectError	**vbObjectError**
vbTab	**ControlChars.Tab**

参照

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

WeekOfYear 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **WeekOfYear** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するため、Visual Basic 6.0 の **WeekOfYear** 定数は Visual Basic 2005 でも使用できます。詳細については、「FirstWeekOfYear 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbUseSystem	**FirstWeekOfYear.System**
VbFirstJan1	**FirstWeekOfYear.Jan1**
vbFirstFourDays	**FirstWeekOfYear.FirstFourDays**
vbFirstFullWeek	**FirstWeekOfYear.FirstFullWeek**

参照

関連項目

FirstWeekOfYear 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

WindowState 定数 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 の **WindowState** 定数および値と、Visual Basic 2005 でそれらに相当する定数および値を以下の表に示します。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbNormal (0)	Normal
vbMinimized (1)	Minimized
vbMaximized (2)	Maximized

参照

概念

Form オブジェクト (Visual Basic 6.0 ユーザー向け)

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

WindowStyle 定数 (Visual Basic 6.0 ユーザー向け)

次の表は、Visual Basic 6.0 の **WindowStyle** 定数と値、および対応する Visual Basic 2005 の定数を示しています。

メモ：
下位互換性を維持するため、Visual Basic 6.0 の **WindowStyle** 定数は Visual Basic 2005 でも使用できます。詳細については、「AppWinStyle 列挙型」を参照してください。

Visual Basic 6.0	Visual Basic 2005 での同等物
vbHide	**AppWinStyle.Hide**
vbNormalFocus	**AppWinStyle.NormalFocus**
vbMinimizedFocus	**AppWinStyle.MinimizedFocus**
vbMaximizedFocus	**AppWinStyle.MaximizedFocus**
vbNormalNoFocus	**AppWinStyle.NormalNoFocus**
vbMinimizedNoFocus	**AppWinStyle.MinimizedNoFocus**

参照

関連項目

AppWinStyle 列挙型

その他の技術情報

定数の対応関係 (Visual Basic 6.0 ユーザー向け)

Visual Basic の定数と列挙体

オブジェクト ライブラリと名前空間 (Visual Basic 6.0 ユーザー向け)

名前空間は、一見すると Visual Basic 2005 で新しく導入された概念のように思えますが、実際は Visual Basic 6.0 のオブジェクト ライブラリに似た概念です。

Visual Basic 6.0 では、各種ライブラリに、アプリケーションの作成に使用するオブジェクトが含まれています。たとえば Visual Basic ライブラリには、Windows ベースのアプリケーションの基礎となる、フォーム オブジェクトおよび組み込みコントロール オブジェクトが含まれています。オブジェクト ブラウザでこれらのオブジェクトを表示することにより、対象のライブラリに含まれているオブジェクトを参照できます。

Visual Basic 2005 では、.NET Framework クラス ライブラリを構成するアセンブリにオブジェクトが含まれています。各アセンブリが名前空間を表しています。たとえば、**System.Windows.Forms** 名前空間には、フォーム オブジェクトおよびコントロール オブジェクトが含まれています。Visual Basic 6.0 と同様、名前空間はオブジェクト ブラウザで調べることができます。

ヒント
Visual Basic 6.0 ユーザーにとって特に重要なのは **Microsoft.VisualBasic** 名前空間です。この名前空間には、Visual Basic 2005 で Visual Basic 6.0 構文を使用し続けることができる多くのオブジェクトと定数が含まれています。

参照

概念

Visual Studio の .NET Framework クラス ライブラリの概要

ランタイム ライブラリ (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 または Visual Basic 2005 で作成されたアプリケーションを実行するには、ユーザーのコンピュータにランタイム ライブラリがインストールされている必要があります。

概念の違い

Visual Basic 6.0 で作成されたアプリケーションを実行するためには、クライアント コンピュータに Visual Basic のランタイム ファイル (Msvbvm60.dll) がインストールされている必要があります。このファイルは、必要に応じてアプリケーションと一緒にインストールできるように、通常はセットアップ パッケージに含まれていました。

Visual Basic 2005 では、クライアント コンピュータでのコード実行を管理するためのランタイム環境として .NET Framework の共通言語ランタイムが必要です。共通言語ランタイム ファイルは、ClickOnce または Windows インストーラによる配置を使用して、対象のコンピュータにプレインストールできます。詳細については、「配置の必要条件 (Visual Studio)」を参照してください。

参照

概念

セットアップと配置 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

アプリケーションとコンポーネントの配置

.NET Framework の再頒布

Visual Basic アプリケーションのアップグレード

次の各セクションでは、Visual Basic 6.0 を使って作成したプログラムを Visual Basic 2005 にアップグレードする方法を説明します。

このセクションの内容

アップグレードの新機能

Visual Basic 6.0 プロジェクトのアップグレードに関連する新機能について説明します。

Visual Basic 6.0 からのアップグレード

アプリケーションを新しいバージョンの Visual Basic に変換する方法について説明します。

Visual Basic 6.0 のアップグレード リファレンス

プロジェクトを Visual Basic 2005 にアップグレードするツールの診断メッセージについて説明します。

関連するセクション

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic 6.0 のオブジェクト、概念、および技法と Visual Basic 2005 でそれらに相当するものとを比較する一連のトピックを示します。

ソリューション、プロジェクト、および項目の概要

統合開発環境 (IDE: integrated development environment) でのアプリケーション開発の基礎となる主要な概念について説明します。このトピックでは、アップグレードしたプロジェクトの管理およびプロジェクト関連タスクの実行に関する背景情報を提供します。

製品のサポートとユーザー補助

IDE のインストール、エディション、製品サポート、およびカスタマイズについて説明します。

アップグレードの新機能

Visual Basic 2005 には、Visual Basic 6.0 アプリケーションおよびコードを Visual Basic 2005 にアップグレードするための新機能が追加されています。

Visual Basic 2005 の新機能

機能	説明
MTS および COM+ のアップグレードのサポート	Microsoft Transaction Server (MTS) コンポーネントまたは COM+ コンポーネントを使用している Visual Basic 6.0 プロジェクトをアップグレードできるようになりました。
ActiveX コントロールのアップグレードのサポート	多くの Visual Basic 6.0 ActiveX コントロールを、対応する Visual Basic 2005 要素にアップグレードできるようになりました。サポートされているのは、WebBrowser、ToolBar、ImageList、TreeView、ListView、RichTextBox、ProgressBar、StatusBar、CommonDialog、MaskedTextBox などのコントロールです。
コードのアップグレードの強化	Visual Basic 2005 では、次のような多くの強化がコードのアップグレード機構に加えられています。 • フォーム コントロールのコレクションで文字列ベースのキーを使っているコードをアップグレードします。 • KeyPress イベント内でキー値を変更するコードをアップグレードします。 他に次のような強化が加えられています。 • Visual Basic 6.0 **App** オブジェクトを新しい **My.Application** オブジェクトにアップグレードします。 • **QueryUnload** イベントの *UnloadMode* パラメータを **FormClosing** イベントの *ClosingReason* 引数にアップグレードします。 • 新しい部分型を使う形式にフォームをアップグレードします。 • Visual Basic 2005 に用意された既定のインスタンスのサポートを使うように、Visual Basic 6.0 の既定のインスタンスをアップグレードします。 • **Forms** コレクションを Visual Basic 2005 の System.Windows.Forms コレクションにアップグレードします。 • Visual Basic 6.0 の **Clipboard** オブジェクトを新しい **My.Computer.Clipboard** オブジェクトにアップグレードします。 • **LoadResString**、**LoadResData**、および **LoadResPicture** の各メソッドを新しい **My.Resources** オブジェクトにアップグレードします。

参照

概念

Visual Basic 言語の新機能

Visual Basic 6.0 からのアップグレード

Visual Basic 2005 では、以前のバージョンの Visual Basic を強化するために、新しい開発環境、更新されたプログラミング言語、および開発をより簡単にする新しいフォーム パッケージが追加されました。Visual Basic 6.0 から Visual Basic 2005 へ移行する作業は予想するより簡単です。統合開発環境とヘルプ システムが強化されたので、Visual Basic 6.0 の概念をそれに相当する Visual Basic 2005 の概念に置き換えることで、より簡単にコードを正しくアップグレードできます。

Visual Basic 2005 について学べば、おそらく Visual Basic 6.0 プロジェクトをアップグレードすることに魅力を感じるでしょう。2 つのバージョンに相違点はありますが、用意されているツールがプロジェクトを正しくアップグレードするために役立ちます。

次のドキュメントは、既存のアプリケーションをアップグレードし、Visual Basic 2005 で加えられた変更を理解するために役立ちます。

このセクションの内容

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

既存のアプリケーションを Visual Basic 2005 にアップグレードする方法を説明します。

Visual Basic 6.0 と現在のバージョンの Visual Basic の両方を使用する

Visual Basic の 2 つのバージョンの互換性について説明します。

チュートリアル : Visual Basic 6.0 アプリケーションから現在のバージョンの Visual Basic へのアップグレード

Visual Basic 6.0 サンプル アプリケーションとそれに相当する Visual Basic 2005 のサンプル アプリケーションを比較します。

関連するセクション

Visual Basic 6.0 ユーザー向けの新機能

Visual Basic 2005 の新機能について説明します。

Visual Basic 6.0 ユーザー向けのヘルプ

複数のトピックに分けて、オブジェクト、概念、および技術に着目して Visual Basic 6.0 と Visual Basic 2005 を比較します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic に加えられた変更点について説明します。

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

Visual Basic 6.0 で作成したアプリケーションを Visual Basic 2005 にアップグレードすると、.NET Framework の利点を活かして開発を続けることができます。Visual Basic 6.0 のプロジェクト ファイル (.vbp) を初めて開くと、Visual Basic アップグレード ウィザードが表示されます。また、用意されているコマンド ライン ツールを使って、開発環境の外部でプロジェクトをアップグレードすることもできます。

アップグレード ツールは、プロジェクト内のコードを Visual Basic 2005 の構文に従うように変更し、フォームおよびコントロールを Visual Basic 2005 の対応するフォームおよびコントロールで置き換えます。Visual Basic 6.0 と Visual Basic 2005 の違いにより、プロジェクト内の一部の機能については直接変換できない場合もあります。そのような場合は、アップグレード レポートが表示されるので、その指示に従ってアプリケーションを修正してください。

一部の Visual Basic 6.0 アプリケーションでは、Visual Basic 2005 にアップグレードしても利点がない場合があります。このようなアプリケーションをアップグレードすることにした場合は、アップグレードが円滑に実行されるよう準備できます。

アップグレード プロセスの詳細については、以下のトピックを参照してください。

このセクションの内容

アップグレード レポートを開く方法について説明します。

方法 : コマンド ラインからプロジェクトをアップグレードする

コマンド ライン アップグレード ツールの使い方について説明します。

Visual Basic 6.0 からアップグレードされたアプリケーションのトラブルシューティング

アップグレード ツールでは検出されない既知の問題の一覧とその対処方法を示します。

関連するセクション

Visual Basic の新機能

Visual Basic 2005 の新機能について説明します。

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic 6.0 と Visual Basic 2005 の相違点の一覧を示します。

Visual Basic 6.0 と現在のバージョンの Visual Basic の両方を使用する

Visual Basic の 2 つのバージョンの互換性について説明します。

アップグレードが必要である理由

Visual Basic 2005 のデザイナには 2 つの選択肢がありました。それは、既存のコード ベースを改装して .NET Framework の上で実行する方法と、コードを最初から作り直して .NET Framework を最大限に活用する方法です。正しい選択は、コードを最初から作り直すことでした。それにより、次の利点を持つ製品が得られます。

- 顧客から最も求められている機能を提供できる (たとえば、継承やスレッド処理)。
- .NET Framework に完全および無制限にアクセスできる。
- Visual Basic を次世代の Web アプリケーションに前進させることができる。

たとえば、Windows フォーム (新しいフォーム パッケージ) に含まれている新機能の多くは、新しいコントロールまたは新しいプロパティとして既存のコードに追加することもできました。ただし、このようなことはできても、Windows フォームに固有の他の高度な機能 (セキュリティやビジュアルの継承など) は得られませんでした。

Visual Basic 2005 の大きな目標の 1 つは、Visual Basic コードが C# や C++ などの他の言語で作成されたコードと完全に相互運用できるようにすることです。また、別の目標は、Visual Basic 開発者が、Windows API を機能させるためにこれまで避けられなかったプログラミングの問題回避策に頼らずに、.NET Framework の力を簡単に利用できるようにすることです。Visual Basic では、共通言語ランタイムをターゲットとする Visual C++ などの言語と同じ変数型、配列、ユーザー定義型、クラス、およびインターフェイスを使用しています。しかし、固定長文字列など、いくつかの機能は削除する必要がありました。

Visual Basic は真のオブジェクト指向言語となり、**GoSub**/**Return** や **DefInt** などの直観的でなく整合性を乱す機能は削除されています。

その結果、新たな活力が注入された Visual Basic は、Windows ベースのアプリケーションを作成するための最も生産的なツールであり続けると同時に、次世代の Web サイトを作成するために最も適したツールとして位置付けられるようになっています。ただし、Visual Basic 6.0 のアプリケーションを Visual Basic およびフォーム機能に対応するようにアップグレードすることで、以前のバージョンとの互換性が失われるという点を考慮する必要があります。

参照

概念

Visual Basic 6.0 アプリケーションのアップグレードの準備
アップグレードを行うにあたっての注意事項

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

アップグレードを行うにあたっての注意事項

Visual Basic 2005 では、以前のバージョンの Visual Basic から大きな転換がありました。Visual Basic 2005 は .NET Framework の利点を活かすために一から設計されているため、多くの領域で以前のバージョンとの互換性が失われています。

メモ：

Visual Basic 2005 と Visual Basic 6.0 は、同じコンピュータにインストールして同時に実行できます。Visual Basic 2005 で作成されたアプリケーションと Visual Basic 6.0 で作成されたアプリケーションは、同じコンピュータ上にインストールして実行できます。Visual Basic 2005 で作成されたコンポーネントは、以前のバージョンの Visual Basic や他の言語で作成された COM コンポーネントと相互運用できます。

- ほとんどの場合は、Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードすることにより、.NET Framework の利点を活かして開発を続けることができます。ただし、アップグレードせずに Visual Basic 6.0 で開発を続けたほうがよい場合もあります。アプリケーションをアップグレードするかどうかを決定する要素としては、Visual Basic 2005 でサポートされていない機能や、非互換性やアーキテクチャ上の問題に起因する修正作業の量などがあります。

サポートされていない機能

Visual Basic 2005 ではサポートされていない Visual Basic 6.0 の機能を次に示します。詳細については、「Visual Basic 6.0 アプリケーションのアップグレードの準備」を参照してください。

- OLE コンテナ コントロール　Visual Basic 2005 には、これに相当するものはありません。このコントロールに依存するアプリケーションは、Visual Basic 6.0 のままにしておく必要があります。
- ダイナミック データ エクスチェンジ (DDE: Dynamic Data Exchange)　DDE メソッドはサポートされなくなりました。DDE を使用しているアプリケーションは、他の方法でアプリケーション間の通信を行うように修正するか、または Visual Basic 6.0 のままにしておく必要があります。
- DAO または RDO へのデータ連結　DAO データ ソースまたは RDO データ ソースへのデータ連結は、Visual Basic 2005 ではサポートされていません。**Data** コントロールおよび **RemoteData** コントロールに対応する機能はありません。DAO および RDO にはコードからのみアクセスできます。DAO または RDO へのデータ連結を使用しているアプリケーションは、ADO を使用するように変更するか、または Visual Basic 6.0 のままにしておく必要があります。
- Visual Basic 5.0 のコントロール　Visual Basic 6.0 には、Visual Basic 5.0 版の Windows コモン コントロールおよびデータ連結グリッド (DBGrid) コントロールが含まれていました。これらのコントロールは、Visual Basic 2005 とは互換性がありません。これらのコントロールを使用しているアプリケーションは、6.0 バージョンを使用するように更新するか、または Visual Basic 6.0 のままにしておく必要があります。
- DHTML アプリケーション　Visual Basic 2005 には、これに相当するものはありません。ただし、DHTML アプリケーションは Visual Basic 2005 テクノロジと相互運用できます。
- ActiveX ドキュメント　Visual Basic 2005 には、これに相当するものはありません。ただし、DHTML アプリケーションは Visual Basic 2005 テクノロジと相互運用できます。ActiveX ドキュメントは、ユーザー コントロールとして作成し直すか、または Visual Basic 6.0 のままにしておく必要があります。
- プロパティ ページ　Visual Basic 2005 には、これに相当するものはありません。プロパティ ページに強く依存するアプリケーションは、Visual Basic 6.0 のままにしておく必要があります。

必要な修正作業量の確認

Visual Basic 6.0 と Visual Basic 2005 には多くの違いがあるため、ほとんどのアプリケーションでは、直接変換できない機能やその他の非互換性に関しての修正作業が必要になります。必要な修正作業の量は、アプリケーションの種類、機能、使用される言語構成要素など、いくつかの要素によって決まります。

必要な修正作業の量を最も簡単に知る方法は、Visual Basic アップグレード ウィザードを実行して、アップグレード レポートに示された項目のリストを見ることです。アプリケーションをアップグレードするときには、新しいコピーが作成され、元のアプリケーションは変更されません。レポートを見た後で、アップグレードしないことに決めた場合は、新しいプロジェクトを削除し、Visual Basic 6.0 で開発を続けることができます。

アップグレードするかどうかの決定に影響する可能性があるその他の要素を次に示します。

- 1 階層のデータベース アプリケーション　DAO へのデータ連結がサポートされていないため、ローカル データベースに直接連結されたコントロールを使用する単純なアプリケーション (Microsoft Access で作成されたアプリケーションなど) では、相当な量の修正作業が必要になる場合があります。

- Visual Basic アドイン　Visual Basic 2005 は Visual Studio 統合開発環境を使用するため、機能拡張のためのオブジェクト モデルが Visual Basic 6.0 とは大きく異なります。アドインに対してはかなりの量の修正作業が必要になります。
- ゲーム　アーケード ゲームなど、Visual Basic 6.0 の特定のパフォーマンス特性に依存するアプリケーションは、Visual Basic 2005 のパフォーマンス特性が Visual Basic 6.0 とは異なるため、修正が必要になります。
- グラフィックス　フォームのグラフィックス メソッド、およびシェイプ コントロールやライン コントロールはサポートされていません。これらの機能を多く使用してフォーム上に描画しているアプリケーションでは、かなりの量の修正作業が必要になります。
- ドラッグ アンド ドロップ機能　ドラッグ アンド ドロップ機能のモデルは大きく異なっています。ドラッグ アンド ドロップ操作を行うコードを書き直す必要があります。
- バリアント型　アプリケーションをアップグレードすると、**Variant** データ型は **Object** 型に変換されます。アプリケーションでバリアント型が重要な役割を担っている場合は、この変換によってアプリケーションの動作がわずかに変化することがあります。たとえば、Visual Basic 6.0 の `IsObject(Variant)` 式は **false** に評価されますが、Visual Basic 2005 ではこの式が `IsObject(Object)` に変更され、評価結果は **true** になります。
- Windows API　言語が変更されているため、Windows API の呼び出しの多くは、修正するか、.NET Framework 関数で置き換える必要があります。

参照

概念

Visual Basic 6.0 アプリケーションのアップグレードの準備

アップグレードが必要である理由

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

Visual Basic 6.0 アプリケーションのアップグレードの準備

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードするプロセスの大部分は自動的に行われますが、多少の準備作業を行うことによって、アップグレードを最適化できます。

ここに示す推奨事項に従うことにより、既存のプロジェクトを Visual Basic 2005 にアップグレードした後に必要な変更作業を最小限に抑えることできます。また、変更作業を完全に回避できる場合もあります。ほとんどの場合、推奨事項では適切なプログラミング方法を紹介していますが、中には互換性のないオブジェクトやメソッドが紹介されている場合もあります。既存のプロジェクトを Visual Basic 2005 にアップグレードするときには、これらのオブジェクトおよびメソッドの使用を最小限にする必要があります。

一般的な推奨事項

当たり前のことですが、Visual Basic 6.0 でコンパイルして実行するプロジェクトでない場合は正常にアップグレードできません。アップグレードに使用するコンピュータ上に Visual Basic 6.0 をインストールしておくことをお勧めします。これにより、アプリケーションをテストできるだけでなく、必要なコントロール、コンポーネント、およびタイプ ライブラリがアップグレード時にすべて使用できるようになります。

Visual Basic 2005 では、Visual Basic 6.0 プロジェクトだけをアップグレードできます。Visual Basic Version 1 ～ 5 で記述されたプロジェクトの場合は、まずプロジェクトを Visual Basic 6.0 に読み込み (Microsoft ActiveX コントロールのアップグレードを選択)、コンパイルおよび保存してから Visual Basic 2005 にアップグレードします。

フォームおよびコントロールに関する注意事項

Visual Basic 2005 では、Windows フォームという新しいフォーム パッケージを使用します。Windows フォームは、Visual Basic 6.0 のフォーム パッケージとの互換性がありますが、次のような重要な違いもあります。

- Visual Basic 2005 は、OLE コンテナ コントロールをサポートしていません。Visual Basic 6.0 アプリケーションではこのコントロールの使用を避けてください。
- Visual Basic 2005 には、シェイプ コントロールはありません。正方形および長方形はラベルにアップグレードされますが、楕円および円はアップグレードできません。アプリケーションでこれらの形状を使用することは避けてください。
- Visual Basic 2005 には、ライン コントロールはありません。縦および横のラインはラベルにアップグレードされます。斜めのラインはアップグレードされないため、使用を避けてください。
- Visual Basic 2005 には、**Form** の **Circle**、**CLS**、**PSet**、**Line**、および **Point** メソッドに代わる新しいグラフィックス コマンドが用意されています。新しいオブジェクト モデルは Visual Basic 6.0 と大きく異なるため、これらのメソッドはアップグレードできません。
- **Timer** コントロールの **Interval** プロパティを 0 にしてもタイマは無効にならず、間隔が 1 にリセットされます。Visual Basic 6.0 プロジェクトでは、**Interval** を 0 に設定する代わりに、**Enabled** を **False** に設定してください。
- Visual Basic 2005 には MenuStrip と ContextMenuStrip という 2 種類のメニュー コントロールがありますが、Visual Basic 6.0 にはメニュー コントロールが 1 種類しかなく、このコントロールを **MainMenu** または **ContextMenu** コントロールとして開きます。すべてのメニュー コントロールは 1 つの **MenuStrip** コンポーネントにアップグレードされ、その中に各メニュー コントロールに対応する MenuItem が含まれます。アップグレード後にコンテキスト メニューを再度作成し、余分な **MenuStrip** コントロールを削除する必要があります。
- Visual Basic 2005 はダイナミック データ エクスチェンジ (DDE: Dynamic Data Exchange) をサポートしていません。
- Visual Basic 2005 では、**Form.PrintForm** メソッドはサポートされていません。
- Visual Basic 2005 はドラッグ アンド ドロップ機能をサポートしていますが、そのオブジェクト モデルは Visual Basic 6.0 の場合と異なります。したがって、Visual Basic 6.0 のドラッグ アンド ドロップに関するプロパティおよびメソッドはアップグレードできません。
- Visual Basic 2005 の **Clipboard** オブジェクト (**My.Computer.Clipboard**) は拡張されており、Visual Basic 6.0 の **Clipboard** オブジェクトよりも多くの機能とクリップボード形式をサポートしています。ただし、オブジェクト モデルの違いにより、クリップボード関連のステートメントは自動的にアップグレードできません。
- Visual Basic 2005 では、実行時にフォームおよびコントロールの **Name** プロパティがサポートされません。そのため、**Controls** コレクションを反復処理して特定の名前のコントロールを探すようなコードを記述しないでください。この機能を使用するには、.NET Framework の **System.Reflection** クラスを使用します。

データに関する推奨事項

Visual Basic 2005 には、ADO の強化版である ADO.NET が導入されています。これは分散アプリケーションのデータ処理用に最適化されているため、分散アプリケーションで使用すると ADO よりも優れた性能を発揮します。

Visual Basic 2005 のコード内でも、一部修正を加えれば、引き続き RDO と ADO を使用できます。ただし、Visual Basic 2005 では、コント

ロール、データ コントロール、または RDO ユーザー コネクションに対する DAO および RDO のデータ連結はサポートしていません。アプリケーションに DAO または RDO のデータ連結が含まれる場合は、アプリケーションを Visual Basic 6.0 のままにしておくか、またはプロジェクトを Visual Basic 2005 にアップグレードする前に DAO または RDO のデータ連結を ADO にアップグレードします。ADO のデータ連結は Windows フォームでサポートされています。Visual Basic 6.0 で DAO または RDO を ADO にアップグレードする方法については、Visual Basic 6.0 のヘルプを参照してください。

Web アーキテクチャに関する推奨事項

Visual Basic 2005 には、ASP の強化版である ASP.NET が導入されており、Visual Basic に似たイベント モデルを使用して Web ページを生成する Web フォームというテクノロジを用いたアーキテクチャが追加されています。アーキテクチャは、サーバーをベースとしています。

次の条件を満たす Web ベースのアプリケーションは簡単にアップグレードできます。

- Microsoft の多階層アーキテクチャ ガイドラインに従っている。
- Active Server Pages (ASP) を使用している。
- ビジネス ロジックに Visual Basic 6.0 または Visual C++ 6.0 の COM オブジェクトを使用している。

ASP は Visual Basic 2005 で完全にサポートされており、引き続き ASP、ASP.NET、および Web フォームを使用してアプリケーションを拡張できます。Visual Basic 6.0 および Visual C++ 6.0 のビジネス オブジェクトは、変更せずにそのまま使用することも、Visual Basic 2005 にアップグレードすることもできます。

Visual Basic 2005 では、WebClass はなくなりました。既存の Webclass アプリケーションは、Visual Basic 2005 の Web フォームおよび ASP アプリケーションと相互運用するか、または Web フォーム アプリケーションにアップグレードできます。

参照

概念

アップグレードを行うにあたっての注意事項
アップグレードが必要である理由

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード
言語に関するアップグレード上の推奨事項

言語に関するアップグレード上の推奨事項

Visual Basic 6.0 と Visual Basic 2005 では言語の相違点があるので、アプリケーションのアップグレード時にコードの変更が必要な場合があります。

Visual Basic 6.0 アプリケーションをアップグレードしたり、後でアップグレードする可能性のある Visual Basic 6.0 アプリケーションを開発したりするときには、以下の推奨事項を参照してください。

このセクションの内容

アップグレードに関する推奨事項 : 事前バインディングと明示的な型変換の使用

Visual Basic 6.0 アプリケーションの変数宣言およびデータ型の変換に関する推奨事項を示します。

アップグレードに関する推奨事項 : 日付データ型を使用した日付の格納

Visual Basic 6.0 アプリケーションでの日付の処理に関する推奨事項を示します。

アップグレードに関する推奨事項 : パラメータのない既定プロパティの解決

Visual Basic 6.0 アプリケーションでの既定のプロパティの参照に関する推奨事項を示します。

アップグレードに関する推奨事項 : NULL 値の反映の不使用

Visual Basic 6.0 アプリケーションでの null 値の処理に関する推奨事項を示します。

アップグレードに関する推奨事項 : 下限がゼロの配列の使用

Visual Basic 6.0 アプリケーションでの配列の作成に関する推奨事項を示します。

アップグレードに関する推奨事項 : 基になる値ではなく定数を使用

Visual Basic 6.0 アプリケーションでの定数の参照に関する推奨事項を示します。

アップグレードに関する推奨事項 : ユーザー定義型における配列および固定長文字列の不使用

Visual Basic 6.0 アプリケーションでのユーザー定義型に関する推奨事項を示します。

アップグレードに関する推奨事項 : レガシ キーワードの使用制限

Visual Basic 6.0 アプリケーションで互換性のために残されているキーワードの使用に関する推奨事項を示します。

アップグレードに関する推奨事項 : Win32 API に対するデータ型の調整

Visual Basic 6.0 アプリケーションでの Windows API の呼び出しに関する推奨事項を示します。

関連するセクション

Visual Basic 6.0 アプリケーションのアップグレードの準備

アップグレードのためにアプリケーションを準備する方法について説明します。

アップグレードを行うにあたっての注意事項

アップグレードに関する制限と注意点について説明します。

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

Visual Basic 6.0 アプリケーションのアップグレードについて説明します。

アップグレードに関する推奨事項：事前バインディングと明示的な型変換の使用

Visual Basic 6.0 と Visual Basic 2005 は、いずれも遅延バインディング オブジェクトをサポートしています。これは、変数を **Object** データ型として宣言し、実行時にクラスのインスタンスに割り当てる手法です。しかし、アップグレード プロセスでは、遅延バインディング オブジェクトが含まれていると、既定プロパティを解決するときや、基になるオブジェクト モデルが変更されてプロパティ、メソッド、イベントを変換する必要がある場合などに、問題が発生することがあります。たとえば、`Label1` というラベルを持つ `Form1` というフォームがある場合、次の Visual Basic 6.0 コードはラベルのキャプションを "SomeText" に設定します。

```
Dim o As Object
Set o = Me.Label1
o.Caption = "SomeText"
```

Visual Basic 2005 では、ラベル コントロールの **Caption** プロパティは Text プロパティで置き換えられています。コードをアップグレードすると、**Caption** プロパティのすべてのインスタンスが **Text** に変更されます。しかし、遅延バインディング オブジェクトには型がないため、Visual Basic はオブジェクトの型を検出できず、変換が必要なプロパティがあるかどうかも確認できません。そのような場合は、アップグレード後にコードを変更する必要があります。

事前バインディングされるオブジェクトを使用してコードを書き換えると、アップグレードが自動的にできるようになります。

```
Dim o As Label
Set o = Me.Label1
o.Text = "SomeText"
```

できる限り、変数を単に **Object** データ型として宣言するのではなく、適切なオブジェクト型の変数として宣言してください。

Visual Basic 6.0 のコードで **Object** 変数や **Variant** 変数を使用する場合は、変数を割り当てたり、変数に演算を実行したり、変数を関数に渡したりするときに、明示的な型変換を使用することをお勧めします。たとえば、次のコードでは '+' 演算子の意図が不明確です。

```
Dim Var1 As Variant
Dim Var2 As Variant
Dim Var3 As Variant
Var1 = "3"
Var2 = 4
'BAD: Should Var1 and Var2 be added as strings or integers?
Var3 = Var1 + Var2
```

この例は、Visual Basic 2005 ではランタイム エラーになることがあります。次に示すように、最後の行を明示的に変換するように書き換えると、コードが正常に動作するようになります。

```
'GOOD: explicit conversion
Var3 = CInt(Var1) + CInt(Var2)
```

Visual Basic 2005 は、パラメータの型に基づいた関数のオーバーロードをサポートしています。たとえば、**Environ** 関数には 2 つの形式があります。

```
Environ( Expression As Integer) As String
Environ( Expression As String ) As String
```

Visual Basic 2005 は、パラメータの型に基づいて、どちらの関数を呼び出すかを決定します。**Environ()** に整数を渡した場合は、整数を取る方の関数が呼び出されます。文字列を渡した場合は、文字列を取る方の関数が呼び出されます。オーバーロードされた関数に **Variant** または **Object** データ型を渡すと、コンパイル エラーやランタイム エラーが発生する可能性があります。次に示すような明示的な変換を使用すると、Visual Basic 2005 にアップグレードした後もコードが正しく動作します。

```
Dim a As String
Dim v As Variant
v = "Path"
'GOOD: explicit conversion
```

```
a = Environ(CStr(v))
```

遅延バインディング オブジェクトに対して明示的な型変換を使用するコードをお勧めします。これによって、コードの意図を解釈しやすくなり、プロジェクトを Visual Basic 2005 にアップグレードするのが簡単になります。

参照

その他の技術情報

言語に関するアップグレード上の推奨事項

アップグレードに関する推奨事項：日付データ型を使用した日付の格納

以前のバージョンの Visual Basic では、日付の格納および操作において **Double** データ型の使用がサポートされていました。Visual Basic 2005 では、日付を内部で倍精度小数点数として格納しないため、この操作は行うことができません。たとえば、次のコードは Visual Basic 6.0 では有効ですが、Visual Basic 2005 ではコンパイル エラーになります。

```
Dim dbl As Double
Dim dat As Date
dat = Now
'BAD: Date can't be assigned to a double
dbl = dat
'BAD: Double can't be used in date functions
dbl = DateAdd("d", 1, dbl)
'BAD: CDate can't convert a double to a date
dat = CDate(dbl)
```

.NET Framework では、ToOADate 関数および FromOADate 関数を使用して、倍精度小数点数と日付を相互に変換できます。ただし、プロジェクトを Visual Basic 2005 にアップグレードするときには、倍精度小数点数を使用して日付を格納しているとコードの意図がわかりにくくなります。Visual Basic 2005 でコードに不必要な修正が加えられるのを避けるために、日付を格納するときには常に **Date** データ型を使用してください。

参照

その他の技術情報

言語に関するアップグレード上の推奨事項

アップグレードに関する推奨事項：パラメータのない既定プロパティの解決

Visual Basic 6.0 では、多くのオブジェクトで既定のプロパティが公開されており、指定を省略してプログラミングを簡略化できます。たとえば、**TextBox** の既定プロパティは **Text** であるため、

```
MsgBox Form1.Text1.Text
```

の代わりに、次のように省略して記述できます。

```
MsgBox Form1.Text1
```

既定のプロパティは、コードのコンパイル時に解決されます。また、次の例のように、遅延バインディング オブジェクトで既定のプロパティを使用することもできます。

```
Dim obj As Object
Set obj = Form1.Text1
MsgBox obj
```

遅延バインディングの例では、既定のプロパティが実行時に解決され、**MsgBox** は **TextBox** の既定プロパティの値を **Text1** として表示します。

Visual Basic 2005 では、パラメータのない既定プロパティはサポートされていないため、このような簡略化はできません。プロジェクトをアップグレードすると、Visual Basic 2005 はパラメータのない既定プロパティを解決しますが、実行時に解決される遅延バインディングは自動的に解決できません。そのような場合は、アップグレード後にコードを変更する必要があります。さらに複雑なことに、多くのライブラリは既定のプロパティを実装するために **_Default**. **_Default** というプロパティを使用しています。このプロパティは、プロキシとして動作し、呼び出しを本当の既定のプロパティに渡します。したがって、プロジェクトをアップグレードすると、いくつかの既定プロパティが **_Default** に解決されます。コードは通常どおりに動作しますが、実際のプロパティを使用して明示的に記述されたコードに比べると、理解しにくくなります。以上の理由により、Visual Basic 6.0 のコードでパラメータのない既定プロパティを使用することは避けるようにします。次のようなコードは記述しないでください。

```
Dim obj As Object
Set obj = Me.Text1
'BAD: Relying on default property
MsgBox obj
'BAD: Relying on default property
MsgBox Me.Text1
```

代わりに、次のコードを使用します。

```
Dim obj As TextBox
Set obj = Me.Text1
'GOOD: Default property is resolved
MsgBox obj.Text
'GOOD: Default property is resolved
MsgBox Me.Text1.Text
```

Visual Basic 2005 では、パラメータのない既定プロパティはサポートされませんが、パラメータを持つ既定プロパティはサポートされています。この 2 種類の違いを理解するために、パラメータを持つ既定プロパティには常にインデックスが存在すると考えてください。たとえば、ADO **recordset** の既定プロパティは **Fields** コレクションです。コード例は、次のとおりです。

```
Dim rs As ADODB.Recordset
rs("CompanyName") = "SomeCompany"
rs!CompanyName = "SomeCompany"
```

これは、実際には次のコードを簡略化したものです。

```
Dim rs As ADODB.Recordset
rs.Fields("CompanyName").Value = "SomeCompany"
rs.Fields!CompanyName.Value = "SomeCompany"
```

この場合、**Fields** プロパティはパラメータを持つため Visual Basic 2005 では有効な使用例です。ただし、**Fields** プロパティの既定プロパティである **Value** はパラメータを持たないため、Visual Basic 2005 での正しい使用例は次のようになります。

```
Dim rs As ADODB.Recordset
rs("CompanyName").Value = "SomeCompany"
rs!CompanyName.Value = "SomeCompany"
```

この例に示すプロパティやその他の既定プロパティは、プロジェクトのアップグレード時に解決されます。したがって、Visual Basic 6.0 でそれを解決することは、良いプログラミング手法です。ただし、**Object** および **Variant** データ型に対して既定のプロパティを使用することは避けてください。これらのデータ型の既定プロパティは解決できず、プロジェクトをアップグレードした後でコードを修正する必要があります。

参照

その他の技術情報

言語に関するアップグレード上の推奨事項

アップグレードに関する推奨事項：NULL 値の反映の不使用

以前のバージョンの Visual Basic では、NULL 値の反映がサポートされています。NULL 値の反映は、式で NULL が使用されている場合に、式の結果自体が **Null** になるという前提を支えるものです。次の例では、どの場合にも `V` の結果が NULL になります。

```
Dim V
V = 1 + Null
V = Null + Right$("SomeText", 1)
V = Right("SomeText", 0)
```

NULL 値の反映は、Visual Basic 2005 ではサポートされていません。`1+Null` というステートメントは、Visual Basic 2005 では型の不一致を起こします。また、Visual Basic 6.0 の **Left** 関数には文字列を返す **Left$** と Null に設定できるバリアントを返す **Left** の 2 種類がありましたが、Visual Basic 2005 にあるのは 1 種類の **Left** だけで、これは常に文字列を返します。

Visual Basic 6.0 と Visual Basic 2005 の両方との互換性を持つためには、Null 値の反映を使用せずに、Null をテストするコードを常に記述する必要があります。また、Visual Basic 2005 では、以下の関数は Null を返しません。

Chr, **Command**, **CurDir**, **Date**, **Environ**, **Error**, **Hex**, **LCase**, **LTrim**, **Oct**, **Right**, **RTrim**, **Space**, **Str**, **Time**, **Trim**, **UCase**

NULL 値の反映は、データベースのフィールドに NULL が含まれるかどうかを調べる必要があるデータベース アプリケーションで、一般的に使用されています。そのような場合は、関数 **IsNull()** を使用して結果を確認し、適切なアクションを実行します。

参照

その他の技術情報

言語に関するアップグレード上の推奨事項

アップグレードに関する推奨事項：下限がゼロの配列の使用

Visual Basic 6.0 では、上限と下限に任意の整数を設定して配列を定義できます。また、**ReDim** を使用してバリアントを配列として再割り当てすることもできます。Visual Basic 2005 の配列では、他の言語との相互運用性を得るために、下限をゼロに設定する必要があります。また、**ReDim** は、事前に **Dim** キーワードで配列変数を宣言していないと使用できません。これによって配列の定義方法が制限されますが、Visual Basic 2005 と他の .NET Framework 言語との間で配列を渡すことができるようになります。この制限の例を次に示します。

```
'BAD: LBound must be 0
Dim a(1 To 10) As Integer
'BAD: Can't use ReDim without Dim
ReDim v(10)
'OK: Creates an array of 11 integers
Dim b(10) As Integer
'OK: Can ReDim previously Dimed variable
ReDim b(5) As Integer
```

上記の制限に伴い、**Option Base** ステートメントが言語から削除されています。

プロジェクトを Visual Basic 2005 にアップグレードすると、すべてのオプション ベース ステートメントがコードから削除されます。配列の下限がゼロである場合は変更されません。しかし、配列の下限がゼロでない場合は、次の例のように、配列の下限が削除され、コードに警告文が挿入されます。

```
Dim a(1 To 10) As Integer
```

このコードは次のように変更されます。

```
' UPGRADE_WARNING: Lower bound of an array was changed from 1 to 0.
Dim a(10) As Integer
```

多くの場合、アップグレードされたコードは、従来どおり機能します。ただし、下限を 1 に設定したロジックを使用するアプリケーションでは、若干の変更作業が必要になります。**Dim**、**ReDim**、および **LBound** の各ステートメントは、警告文で示されるため、変更点を簡単に確認できます。

このため、Visual Basic 6.0 のコードでは下限がゼロの配列を使用し、配列宣言に **ReDim** を使用することは避けてください。また、**Option Base** 1 の使用も避けてください。

参照

その他の技術情報

言語に関するアップグレード上の推奨事項

アップグレードに関する推奨事項：基になる値ではなく定数を使用

コードを記述するときには、定数を使用し、その基になる値は使用しないようにします。たとえば、実行時にフォームを最大化する場合は、次のようなコードを使用します。

```
' GOOD: Constant name
Me.WindowState = vbMaximized
```

次のようには記述しないでください。

```
'BAD: Underlying value
Me.WindowStyle = 2
'BAD: Variable
Me.WindowStyle = X
```

同様に、**True** と **False** を使用し、-1 と 0 は使用しないようにします。

Visual Basic 2005 では、これらの値が変更され、一部ではプロパティや定数の名前も変更されています。プロジェクトを Visual Basic 2005 にアップグレードすると、ほとんどの定数は自動的に変更されますが、定数名ではなくその基の値または変数を使用していると、多くの場合に自動的にアップグレードできません。定数名を使用すると、後で必要になる変更を最小限に減らすことができます。

参照

その他の技術情報

言語に関するアップグレード上の推奨事項

アップグレードに関する推奨事項：ユーザー定義型における配列および固定長文字列の不使用

Visual Basic 2005 の配列および構造体 (以前のユーザー定義型) に他の Visual Studio 言語との完全な互換性を持たせるために加えられた変更により、固定長文字列はサポートされなくなりました。固定長文字列の代用となる互換用のクラスがあるため、ほとんどの場合は問題はありません。

```
Dim FixedLengthString As String * 100
```

このコードは次のようにアップグレードされます。

```
Dim FixedLengthString As New VB6.FixedLengthString(100)
```

ただし、構造体で固定長文字列を使用すると問題が生じます。これは、構造体の作成時に固定長文字列クラスが自動的に生成されないことが原因です。同様に、構造体の作成時に固定サイズの配列は作成されません。

コードをアップグレードすると、固定長の文字列または配列を含むユーザー定義型は構造体に変換され、コードで構造体を参照する前に固定長の文字列または配列を初期化するように指示するコメントが挿入されます。このような変更を避けるには、固定長文字列の代わりに通常の文字列を使用し、固定サイズ配列の代わりに未初期化配列を使用するように、Visual Basic 6.0 のユーザー定義型を変更します。以下に例を示します。

```
Private Type uType
    anArray(5) As Integer
    aFixedString As String * 100
End Type
Sub SomeFunction()
    Dim aVariable As uType
End Sub
```

このコードは次のように変更できます。

```
Private Type uType
    anArray() As Integer
    aFixedString As String
End Type
Sub SomeFunction()
    Dim aVariable As uType
    ReDim aVariable.anArray(5) As Integer
    aVariable.aFixedString = String$(100, " ")
End Sub
```

参照

その他の技術情報

言語に関するアップグレード上の推奨事項

アップグレードに関する推奨事項：レガシ キーワードの使用制限

Visual Basic 2005 にアップグレードする可能性がある Visual Basic 6.0 プロジェクトでは、言語から削除された以下のキーワードを使わないようにしてください。

キーワード	説明
Def< *type* **>**	以前のバージョンの Visual Basic では、**DefBool**、**DefByte**、**DefInt**、**DefLng**、**DefCur**、**DefSng**、**DefDbl**、**DefDec**、**DefDate**、**DefStr**、**DefObj**、および **DefVar** は、変数の範囲を特定の型として定義するためにモジュールの宣言セクションで使用されます。たとえば、`DefInt A-C` は文字 A、B、または C で始まるすべての変数を整数として定義します。**Def**<*type*> ステートメントを使用する代わりに、変数を明示的に宣言する必要かあります。
計算後の **GoTo/GoSub**	計算後の **GoTo**/**GoSub** ステートメントは、次のような形をとります。 `On x GoTo 100, 200, 300` これらは、Visual Basic 2005 ではサポートされていません。代わりに **If** ステートメントまたは **Select Case** 構造を使用します。
GoSub/Return	**GoSub** ステートメントおよび **Return** ステートメントは Visual Basic 2005 ではサポートされていません。ほとんどの場合、これらのステートメントは、関数やプロシージャで置き換えることができます。
Option Base 0|1	**Option Base** ステートメントは、配列の既定の下限を 0 または 1 に指定するために使用されていました。Visual Basic 2005 がネイティブにサポートするのは下限が 0 の配列だけであるため、このステートメントは削除されました。下限がゼロでない配列は、互換性ライブラリを通じてサポートされています。
VarPtr, **ObjPtr**, **StrPtr**	**VarPtr**、**VarPtrArray**、**VarPtrStringArray**、**ObjPtr**、および **StrPtr** は、変数が格納されているメモリ アドレスを取得するために使用する非公開の関数です。これらの関数は、Visual Basic 2005 ではサポートされていません。
LSet	Visual Basic 6.0 では、あるユーザー定義型の変数を別のユーザー定義型の変数に代入するために **LSet** ステートメントを使用できます。この機能は、Visual Basic 2005 ではサポートされていません。

参照

その他の技術情報

言語に関するアップグレード上の推奨事項

アップグレードに関する推奨事項：Win32 API に対するデータ型の調整

多くの API は、Visual Basic 6.0 の場合とまったく同じように使用できますが、データ型を適宜調整する必要があります。Visual Basic 6.0 の長整数型 (**Long**) は、Visual Basic 2005 では整数型 (**Integer**) に置き換えられます。また、Visual Basic 6.0 の整数型 (**Integer**) は、Visual Basic 2005 では、短整数型 (**Short**) で置き換えられます。アップグレード時にはこの変換が自動的に行われ、単純な API ならば Visual Basic 6.0 の場合とまったく同じように動作します。以下に例を示します。

```
Private Declare Function GetVersion Lib "kernel32" () As Long
Function GetVer()
    Dim Ver As Long
    Ver = GetVersion()
    MsgBox ("System Version is " & Ver)
End Function
```

このコードは次のように変更されます。

```
Private Declare Function GetVersion Lib "kernel32" () As Integer
Function GetVer()
    Dim Ver As Integer
    Ver = GetVersion()
    MsgBox("System Version is " & Ver)
End Function
```

数値データ型のアップグレードに加えて、Visual Basic 6.0 の固定長文字列データ型は Visual Basic 2005 ではサポートされず、固定長文字列ラッパー クラスにアップグレードされます。多くの場合、Visual Basic 6.0 では通常の文字列を使用して同じ動作を実行できます。以下に例を示します。

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Long) As Long
Function GetUser()
    Dim Ret As Long
    Dim UserName As String
    Dim Buffer As String * 25
    Ret = GetUserName(Buffer, 25)
    UserName = Left$(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox (UserName)
End Function
```

このコードは、固定長文字列の代わりに長さ 25 に明示的に設定された通常の文字列を使用して記述することもできます。

```
Dim Buffer As String
Buffer = String$(25, " ")
```

これは、次のように Visual Basic 2005 にアップグレードされます。

```
Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
Function GetUser()
    Dim Ret As Integer
    Dim UserName As String
    Dim Buffer As String
    Buffer = New String(CChar(" "), 25)
    Ret = GetUserName(Buffer, 25)
    UserName = Left(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox(UserName)
End Function
```

場合によっては、Visual Basic 2005 の方が API に文字列を渡す処理に優れています。これは、**ANSI** キーワードと **UNICODE** キーワードを使用して、文字列を渡す方法を宣言することもできるためです。

次の 3 つの場合に、いくつかの変更が必要になります。1 つ目は、固定長文字列またはバイト配列を含むユーザー定義型を API に渡す場合です。Visual Basic 2005 では、ユーザー定義型内の固定長文字列とバイト配列のそれぞれに (System.Runtime.InteropServices 名前空間から) **MarshalAs** 属性を追加して、コードを変更することが必要な場合もあります。2 つ目は、**Declare** ステートメントで **As Any** 変数型を使用している場合です。Visual Basic 2005 は、@com.parameters ディレクティブのカスタム マーシャリングの使用をサポートしていません。**As Any** 型の変数は、文字列または **Null** の変数を渡すときによく使用されました。この Visual Basic 6.0 の機能は、長整数と文字列の 2 つの形式の API を宣言することで置き換えることができます。たとえば、**GetPrivateProfileString** API には、**As Any** 型の *lpKeyName* というパラメータがあります。

```
Private Declare Function GetPrivateProfileString Lib "kernel32" Alias _
"GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal _
lpKeyName As Any, ByVal lpDefault As String, ByVal lpReturnedString _
As String, ByVal nSize As Long, ByVal lpFileName As String) As Long
```

Declare を、それぞれ long を取る形式と string を取る形式の 2 つで置き換えることにより、この "**As Any**" を削除できます。

```
Private Declare Function GetPrivateProfileStringKey Lib "kernel32" Alias _
"GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal _
lpKeyName As String, ByVal lpDefault As String, ByVal lpReturnedString _
As String, ByVal nSize As Integer, ByVal lpFileName As String) As Integer

Private Declare Function GetPrivateProfileStringNullKey Lib "kernel32" _
Alias "GetPrivateProfileStringA" (ByVal lpApplicationName As String, _
ByVal lpKeyName As Integer, ByVal lpDefault As String, ByVal _
lpReturnedString As String, ByVal nSize As Long, ByVal lpFileName _
As String) As Integer
```

API に NULL 値を渡すには、**GetPrivateProfileStringNullKey** の方を使用します。これにより、関数を Visual Basic 2005 にアップグレードできます。

変更が必要になる最後のケースは、スレッド作成、Windows クラスの継承、メッセージ キューのフックなどを行う API を使用している場合です。このような関数の一部は、Visual Basic 2005 でランタイム エラーの原因となります。これらの API の多くは、Visual Basic 2005 または .NET Framework に同等のものがあります。それぞれ場合に応じて修正してください。

参照

その他の技術情報

言語に関するアップグレード上の推奨事項

Visual Basic 6.0 互換性ライブラリ

以前のバージョンの Visual Basic を使用した経験がある場合、Visual Basic 2005 では Visual Basic 言語にいくつかの変更が加えられていて、よく使用していた関数が見当たらないことに気が付くかもしれません。Visual Basic を .NET Framework および共通言語仕様 (CLS: Common Language Specification) に準拠させるためにこれらの変更が必要でした。

Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードするときに、構文またはアーキテクチャの違いにより、一部のコードを変換できない場合があります。このため Visual Basic 6.0 互換性ライブラリ (**Microsoft.VisualBasic.Compatibility**) の関数を使用して、既存のコードを大幅に変更することなく Visual Basic 2005 で動作させることができるようにしています。これらの関数は、共通言語仕様との互換性を維持したまま、Visual Basic 6.0 の動作を実現します。アップグレード ツールは、可能であれば、**Microsoft.VisualBasic** 名前空間または **System** 名前空間の関数にコードを直接移動します。

Compatibility 名前空間内の関数およびオブジェクトは、アップグレード ツールをサポートするために設計されましたが、Visual Basic 2005 で新しいアプリケーションを作成するときにも使用できます。ただし、ほとんどの場合、それらよりも高度な機能が .NET Framework に用意されています。

参照

処理手順

方法 : Visual Basic 6.0 ファイル システムのコントロールをアプリケーションに追加する

関連項目

VisualBasic.Compatibility 名前空間リファレンス

概念

Visual Basic 6.0 と現在のバージョンの Visual Basic の両方を使用する

その他の技術情報

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 6.0 からのアップグレード

Visual Basic アップグレード ウィザード

Visual Basic 2005 は、従来の Windows 開発から次世代の Web アプリケーションおよび N 階層アプリケーションの構築への基本的な移行を可能にします。このため、Visual Basic 2005 の利点を活用するには、コードをアップグレードする必要があります。

アップグレード プロセス

アップグレード プロセスは、Visual Basic 6.0 プロジェクトを Visual Basic 2005 で開いたときに自動的に行われます。Visual Basic アップグレード ウィザードが起動し、アップグレード プロセスを案内しながら、新しい Visual Basic 2005 プロジェクトを作成します。既存のプロジェクトはそのまま残されます。これは一方向のプロセスであり、新しい Visual Basic 2005 プロジェクトを Visual Basic 6.0 で開くことはできません。

プロジェクトをアップグレードすると、言語の変更に合わせて構文が変更され、Visual Basic 6.0 フォームは Windows フォームに変換されます。ほとんどの場合は、アップグレードされたコードにいくつかの変更を加える必要があります。これは、特定のオブジェクトや言語機能に対応するものが Visual Basic 2005 に存在しないか、または大きく異なっていて自動的にアップグレードできないためです。アップグレードした後で、Visual Basic 2005 のいくつかの新機能を利用するために、アプリケーションの変更が必要になる場合もあります。

変更時の助けになるように、Visual Basic 2005 はプロジェクトをアップグレードした後に、問題点を示すアップグレード レポートをプロジェクトに追加します。アップグレード後のコードにも、変更の必要があるステートメントを示すコメントが挿入されます。これらのコメントは新しい [タスク一覧] ウィンドウに 'TO DO' タスクとして表示されるため、必要な変更の内容がすぐにわかり、タスクをダブルクリックするだけで該当するコード ステートメントにジャンプできます。アップグレード レポート内の各タスクおよび項目は、コードの変更が必要な理由や必要な処置について詳細を説明したヘルプ トピックにリンクしています。

参照

概念

アップグレード プロセスの完了
アップグレードが必要である理由
Visual Basic 6.0 アプリケーションのアップグレードの準備

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

方法：Visual Basic アップグレード ウィザードを使ってプロジェクトをアップグレードする

Visual Basic アップグレード ウィザードを使用して Visual Basic 6.0 のプロジェクトをアップグレードすると、Visual Basic 2005 でプロジェクトの開発を続けることができます。アップグレード ウィザードは、元の Visual Basic 6.0 プロジェクトを変更するのではなく、元のプロジェクトに基づいて新しい Visual Basic 2005 プロジェクトを作成します。

メモ：

ウィザードを使用するとアップグレード プロセスが簡単になりますが、アップグレードされたアプリケーションで一部のコードの変更が必要になる場合があります。そのため、このプロセスを実行する前に、少し時間をかけて Visual Basic 2005 に習熟しておく必要があります。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードするには

1. Visual Basic 6.0 でプロジェクトを開きます。[ファイル] メニューの [<プロジェクト名> の作成] をクリックします。アプリケーションを実行し、エラーがないことを確認します。
2. Visual Studio 2005 を起動します。[ファイル] メニューの [開く] をポイントし、[プロジェクト/ソリューション] をクリックします。
3. [プロジェクトを開く] ダイアログ ボックスで Visual Basic 6.0 プロジェクトの場所を参照し、プロジェクト (.vbp) ファイルを選択します。

   これにより、Visual Basic アップグレード ウィザードが開きます。
4. ウィザードの 2 ページ目で、使用できるアップグレード オプションのいずれかを選択します。オプションは、アップグレードするプロジェクトの種類によって異なります。

   ヒント

   使用できるオプションの説明を表示するには、ウィザードの実行中に F1 キーを押してください。
5. ウィザードの 3 ページ目で、新しいプロジェクトを作成する場所を入力します。

   メモ：

   各 Visual Basic 2005 プロジェクトは、個別のフォルダに格納されている必要があります。指定したフォルダにファイルが含まれている場合は、別のフォルダを選択するように指示するメッセージが表示されます。
6. ウィザードの 4 ページ目で、[次へ] をクリックしてアップグレード プロセスを開始します。プロセスが完了すると、ソリューション エクスプローラに新しいプロジェクトが表示されます。
7. ソリューション エクスプローラで _UpgradeReport.htm ノードをダブルクリックしてアップグレード レポートを開きます。
8. アップグレード レポートの内容を確認します。エラーがある場合は、アプリケーションを実行する前に修正する必要があります。警告がある場合は、内容を調べて、アプリケーションの動作に影響がないことを確認します。

   ヒント

   エラーまたは警告の詳細については、記述をクリックしてヘルプ トピックを参照してください。
9. [デバッグ] メニューの [開始] をクリックします。アプリケーションを実行し、エラーがないこと、および Visual Basic 6.0 の場合と動作が同じであることを確認します。

   メモ：

アップグレード ウィザードは、可能性のある非互換性をすべて検出するわけではありません。アプリケーションを実行してから新たなエラーに遭遇する場合もあります。既知の問題の一覧については、
「Visual Basic 6.0 からアップグレードされたアプリケーションのトラブルシューティング」を参照してください。

参照

処理手順

方法 : アップグレード レポートを表示する

概念

Visual Basic 6.0 アプリケーションのアップグレードの準備

WebClass (Visual Basic 6.0 ユーザー向け)

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

[Visual Basic 6 コードのアップグレード] ダイアログ ボックス

Visual Basic 6.0 のコードを Visual Basic 2005 にアップグレードするには、[Visual Basic 6 コードのアップグレード] ダイアログ ボックスを使用して、[コード] ウィンドウにコードを入力するか、Visual Basic 6.0 のコード エディタからコードを貼り付けます。[アップグレード] をクリックすると、アップグレードされたコードが Visual Basic 2005 コード エディタのカーソル位置に挿入されます。

また、Visual Basic 6.0 のコードに必要な COM 参照を [参照設定] タブに追加することもできます。この COM 参照は、Visual Basic 2005 プロジェクトの参照設定に追加されます。

[Visual Basic 6 コードのアップグレード] ダイアログ ボックスを表示するには、[ツール] メニューの [Visual Basic 6 コードのアップグレード] をクリックします。

メモ :
[Visual Basic 6 コードのアップグレード] を使用できるのは、コード エディタを表示しているときだけです。

ユーザー インターフェイス要素の一覧

[コード]

Visual Basic 6.0 のコードは、[コード] ウィンドウに入力するか、コード エディタから貼り付けることができます。次に使用するときもコードは保持されているので、さらに変更もできます。

[参照設定]

Visual Basic 6.0 コードに必要な COM 参照を表示します。

[参照名]

COM コンポーネントの表示名を表示します。チェック ボックスをオンにした参照だけが Visual Basic 2005 プロジェクトに追加されます。

[バージョン]

COM コンポーネントのバージョン番号 (メジャーおよびマイナ) を表示します。

[パス]

COM コンポーネントのファイル パスとファイル名を表示します。

[参照の追加]

[参照の追加] ダイアログ ボックスを表示します。このダイアログ ボックスには、使用できる COM コンポーネントの一覧が表示されます。

[アップグレード]

[コード] ウィンドウに入力されたコードをアップグレードして、Visual Basic 2005 コード エディタに挿入します。また、[参照設定] ボックスでオンにした参照を Visual Basic 2005 プロジェクトに追加します。

[Visual Basic 6 コード のアップグレード] ダイアログ ボックスの使用に関するヒント

[Visual Basic 6 コードのアップグレード] ダイアログ ボックスでは、すべてのコードを完全には変換できません。このツールを最大限に利用するには、次のようにします。

- アップグレードを行うにあたっての注意事項 を参照してください。
- 「言語に関するアップグレード上の推奨事項」の推奨事項に従って、コードをアップグレードしやすい形に修正します。Visual Basic 6.0 のコードの記述方法がアップグレードに影響する場合があります。
- アップグレードを試す前に、Visual Basic 6.0 内でコードをテストします。Visual Basic 6.0 内でコードを実行できなければ、Visual Basic 2005 でも実行できません。
- Visual Basic 2005 内のイベント シグネチャは、Visual Basic 6.0 の対応するイベント シグネチャとは異なります。フォームまたはコントロールのイベント プロシージャのコードを入力する場合は、まず Visual Basic 2005 にプロシージャ宣言を追加してから、プロシージャの本体だけをアップグレードしてください。
- [Visual Basic 6 コードのアップグレード] ダイアログ ボックスは、コードを少しずつアップグレードするようにデザインされています。このダイアログ ボックスに入力したコードにより、アップグレードのコンテキストが決まります。

  たとえば、`L.Caption = "MyCaption"` というステートメントがあり、アップグレード ツールは `"L"` の型を判別して `Caption` プロパティを解

決できないので、このステートメントを変換できないとします。その場合は、このステートメントの前に `L` の宣言 (`Dim L As Label` など) を挿入すると、正常にアップグレードできるようになります。

- Visual Basic 6.0 のコードの構文が正しいことを確認します。構文が正しくないと、"`Untranslated statement in WrapperSub`" エラーが起きる可能性があります。
- 最高のパフォーマンスを得るには、フォーム モジュールではなく、モジュールのコードをアップグレードします。フォームをアップグレードする場合は、アップグレード ウィザードを使用してください。
- Unicode 文字が含まれるコードを貼り付ける場合は、Unicode 文字を削除するかどうかを確認するダイアログが表示されます。[OK] をクリックすると、コードが解析され、Unicode 文字が削除されます。

参照

処理手順

方法 : [Visual Basic 6 コードのアップグレード] ダイアログ ボックスを使って Visual Basic 6.0 のコードをアップグレードする

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

方法：[Visual Basic 6 コードのアップグレード] ダイアログ ボックスを使って Visual Basic 6.0 のコードをアップグレードする

[Visual Basic 6 コードのアップグレード] ダイアログ ボックスは、Visual Basic 6.0 のコードを同等の Visual Basic 2005 のコードに変換するツールです。アプリケーション全体をアップグレードする Visual Basic アップグレード ウィザードとは異なり、[Visual Basic 6 コードのアップグレード] ダイアログ ボックスは、デザイン時に使用して Visual Basic 2005 アプリケーションを作成しながらコードを少しずつ変換できます。

[Visual Basic 6 コードのアップグレード] ダイアログ ボックスには、次に示す 2 つの便利な使用法があります。

- 既存の Visual Basic 6.0 のコードをツールに貼り付けて利用できます。
- Visual Basic 6.0 の構文でコードを入力して、Visual Basic 2005 の同等のコードを調べることができます。

ツールを実行すると、Visual Basic 2005 のコードがコード エディタの現在のカーソル位置に挿入されます。正常に変換できない場合は、クリック可能なリンクにコメントが挿入されます。このリンクをクリックすると、問題とその回避方法について説明するヘルプ ページにジャンプします。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual Basic 6.0 のコードをアップグレードするには

1. コード エディタで、コードを挿入する位置にマウス ポインタを置きます。
2. [ツール] メニューの [Visual Basic 6 コード のアップグレード] をクリックします。

   [Visual Basic 6 コードのアップグレード] ダイアログ ボックスが表示されます。
3. [Visual Basic 6 コードのアップグレード] ダイアログ ボックスで、既存の Visual Basic 6.0 のコードを貼り付けるか、Visual Basic 6.0 の構文でコードを入力します。

メモ：
Unicode 文字が含まれるコードを貼り付ける場合は、Unicode 文字を削除するかどうかを確認するダイアログが表示されます。[OK] をクリックすると、コードが解析され、Unicode 文字が削除されます。
4. コードが COM コンポーネントを参照する場合は、[参照設定] タブをクリックして、[参照の追加] をクリックします。次に、[参照の追加] ダイアログ ボックスで参照を選択します。
5. [アップグレード] をクリックしてコードを変換し、コード エディタに挿入します。

   COM 参照はソリューション エクスプローラ内のプロジェクトの [参照] ノードに追加され、プロジェクト レベルの **Imports** ステートメントは **Microsoft.VisualBasic.Compatibility** 名前空間に追加されます。

参照

関連項目

[Visual Basic 6 コードのアップグレード] ダイアログ ボックス

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

方法：アップグレードしたアプリケーションで Visual Basic 6.0 の 3 ステート コントロールをエミュレートする

Visual Basic 6.0 では、**Picture**、**DownPicture**、および **DisabledPicture** の各プロパティは、**CheckBox**、**CommandButton**、、または **OptionButton** コントロールの状態に基づいて異なる画像を表示するために使用します。たとえば、**CheckBox** コントロールがオンのときには **DownPicture** 画像が表示され、オフの場合には **DisabledPicture** 画像が表示されます。

Visual Basic 2005 では、これと同じ効果を以下の例のように ImageList コントロールを使って実現できます。

メモ：

まず、Visual Basic 6.0 アプリケーションをチェックします。デザイン時または実行時に **DownPicture** プロパティおよび **DisabledPicture** プロパティを設定していない場合は、Visual Basic 2005 でも動作は同じです。

メモ：

使用している設定またはエディションによっては、ヘルプの記載と異なるダイアログ ボックスやメニュー コマンドが表示される場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ImageList コントロールの追加

DownPicture プロパティまたは **DisabledPicture** プロパティを設定している場合は、次の手順に従って、アップグレードしたアプリケーションを変更します。

3 ステート コントロールをエミュレートするには

1. **Picture**、**DownPicture**、**DisabledPicture** の各プロパティに割り当てられていたイメージのファイル名と場所を確認し、必要に応じて、ファイルを開発用コンピュータにコピーします。
2. ツールボックスから **ImageList** コントロールをフォームに追加します。
3. [プロパティ] ウィンドウで [Images] プロパティを選択します。
4. イメージ コレクション エディタで、**Picture**、**DownPicture**、および **DisabledPicture** に使用する 3 つの画像を追加します。
5. これらのプロパティが実行時に設定されている場合は、そのコードを削除します。プロパティがデザイン時に設定されている場合は、フォームの Load イベントに次のコードを追加します。

   VB

   ```
   ' Assign the first image (Picture) to the Image property.
   CheckBox1.Image = ImageList1.Images(0)
   ```

6. 実行時に **DownPicture** イメージを表示するには、CheckBox コントロールの CheckedChanged イベントに次のコードを追加します。

   VB

   ```
   If CheckBox1.Checked = True Then
     ' Assign the second image (DownPicture) to the Image property.
     CheckBox1.Image = ImageList1.Images(1)
   Else
     ' Assign the first image (Picture) to the Image property.
     CheckBox1.Image = ImageList1.Images(0)
   End If
   ```

7. 実行時に **DisabledPicture** イメージを表示するには、**CheckBox** コントロールの EnabledChanged イベントに次のコードを追加します。

VB

```
If CheckBox1.Enabled = False Then
  ' Assign the third image (DisabledPicture) to the Image property.
  CheckBox1.Image = ImageList1.Images(2)
ElseIf CheckBox1.Checked = True Then
  ' Assign the second image (DownPicture) to the Image property
  CheckBox1.Image = ImageList1.Images(1)
Else
  ' Assign the first image (Picture)to the Image property
  CheckBox1.Image = ImageList1.Images(0)
End If
```

これにより、アプリケーションは Visual Basic 6.0 のときとまったく同様に動作します。

参照

概念

Style プロパティ (Visual Basic 6.0 ユーザー向け)
CheckBox コントロール (Visual Basic 6.0 ユーザー向け)
CommandButton コントロール (Visual Basic 6.0 ユーザー向け)
OptionButton コントロール (Visual Basic 6.0 ユーザー向け)
OptionButton コントロール (Visual Basic 6.0 ユーザー向け)
User コントロール (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 への WebClass プロジェクトのアップグレード

Visual Basic 6.0 の WebClass プロジェクト (IIS アプリケーション プロジェクトとも呼ばれます) を ASP.NET Web アプリケーション プロジェクトにアップグレードするには、アップグレード ウィザードを使用します。WebClass プロジェクトのアップグレード手順は、他の種類のプロジェクトと基本的には同じですが、いくつかの注意事項があります。

WebClass プロジェクトのアップグレード

WebClass プロジェクトをアップグレードすると、既定では *projectname*.NET という新しい名前のプロジェクトが作成されます。*projectname* は、Visual Basic 6.0 プロジェクトの名前です。

Visual Basic 6.0 の WebClass プロジェクトを ASP.NET にアップグレードすると、プロジェクトの .asp ファイルが .aspx ファイルにアップグレードされます。HTML テンプレート ファイル内の .asp ファイルへの参照は、自動的には .aspx 参照に変更されません。テンプレート ファイルには WebClass プロジェクトの一部ではない他の .asp ファイルへの参照が含まれていることがあるため、参照はアップグレードされません。

また、アップグレード時に新しいプロジェクト ディレクトリにファイルがコピーされる場合は、HTML テンプレート ファイルだけがコピーされます。その他の .html ファイルやイメージ ファイルは、新しいディレクトリにコピーされません。

ASP.NET Web アプリケーション プロジェクトに追加した HTML ファイルは、既定でコンテンツ ファイルとして追加されます。WebClass プロジェクトをアップグレードすると、HTML ファイルは埋め込まれたリソースとして追加されます。アップグレード後に HTML ファイルをプロジェクトに追加する場合は、ファイルの [ビルド アクション] プロパティを [埋め込まれたリソース] に設定する必要があります。このように設定することで、HTML ファイルをアプリケーションに対して表示できるようになります。

ヒント
HTML ファイルの完全パスを指定すると、埋め込まれていない HTML ファイルを参照できます。

Visual Basic 6.0 WebClass プロジェクトを初めてアップグレードするまでに ASP.NET Web アプリケーションを作成したことがない場合は、プロジェクトの **StartUp** オブジェクトを設定する必要があります。そのためには、アップグレード後のプロジェクトを実行する前に、.aspx ファイルを右クリックしてショートカット メニューの [スタート ページに設定] をクリックします。

参照

概念

WebClass (Visual Basic 6.0 ユーザー向け)

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

アップグレード プロセスの完了

アップグレード ウィザードを実行した後で、アップグレードされたプロジェクトが Visual Basic 2005 で開かれます。アップグレードされたプロジェクトは、ソリューション エクスプローラに表示されます。アップグレードされたプロジェクトには、アップグレード済みのすべてのプロジェクト ファイルおよびアップグレード レポートのコピーが含まれています。

ほとんどの場合は、プロジェクトが Visual Basic 2005 で正しく動作するかどうかを確認する作業が必要になります。必要な作業は 3 つのカテゴリに分けられます。

- UPGRADE_ISSUE エラーは、アプリケーションがコンパイルできない原因となる項目を示しています。プロジェクトを実行する前に、エラーを修正する必要があります。エラーはアップグレード レポートに表示され、タスク一覧の項目としても表示されます。
- UPGRADE_TODO エラーは、プロジェクトのコンパイルには影響しないものの、ランタイム エラーの原因となる項目を示しています。プロジェクトを実行する前に、エラーを修正する必要があります。エラーはアップグレード レポートとタスク一覧の項目に表示されるほか、アップグレードされたコード内にもコメントとして示されます。
- UPGRADE_WARNING エラーは、プロジェクトのコンパイルには影響しないものの、ランタイム エラーの原因となる可能性のある項目を示しています。エラーの内容を調べ、必要に応じて修正します。エラーはアップグレード レポートとタスク一覧の項目に表示されるほか、アップグレードされたコード内にもコメントとして示されます。
- UPGRADE_NOTE 項目は、副作用を起こす可能性のある、コードへの大きな変更や動作上の相違点を示す項目です。内容を確認して、変更が必要かどうかを判断してください。この項目はコード内のコメントとして示されます。

アップグレード プロセスを完了するには、タスク一覧でエラーや警告がないかどうかを確認し、コード内に "UPGRADE_NOTE" で始まるコメントがないかどうかを確認する必要があります。

ヒント

アップグレード時にコードに挿入されたすべてのコメント行には、ハイパーリンクが含まれています。リンクをクリック (Ctrl + クリック) すると、関連する問題とその対処法について説明しているヘルプ ページにジャンプします。リンクはコメント行の終わりに表示されるため、スクロールしないと見えない場合もあります。

参照

概念

Visual Basic アップグレード ウィザード

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

Visual Basic のアップグレード レポート

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、ソリューション エクスプローラ内のプロジェクトにアップグレード レポートが追加されます。アップグレード レポートには、アップグレード プロセスに関する情報に加えて、アップグレード中に検出された、プロジェクトの実行前に対処する必要のある問題の一覧が表示されます。

レポートの詳細

アップグレード レポートは HTML 形式です。ソリューション エクスプローラ内で _UpgradeReport.htm ファイルをダブルクリックすると Visual Basic 2005 開発環境内でレポートを表示できます。また、コンテキスト メニューの [ブラウザで表示] をクリックして、外部のブラウザで表示することもできます。

レポートの最初の部分はアップグレードに関する全体的な情報であり、アップグレード時に使用された設定と、新しいプロジェクト ファイルの場所が含まれています。

レポートの残りの部分には、全体的な問題の一覧と、アップグレードされた各ファイルに関する問題の一覧が含まれています。各ファイルのセクションを展開すると、対処する必要のある問題の一覧が表示されます。各問題について、問題の重大度に関する情報、修正が必要なコードの場所、および問題の説明が記述されています。

参照

概念

アップグレード プロセスの完了

Visual Basic アップグレード ウィザード

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

方法：アップグレード レポートを表示する

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードすると、アップグレード レポートが作成され、新しいプロジェクト フォルダに格納されます。アップグレード レポートには、アップグレード プロセスに関する概要情報と、アップグレード中に検出されたエラーや問題に関する情報が含まれています。

メモ：
アップグレード レポートは、アップグレード ウィザードを使用してアップグレードしたプロジェクトに対してだけ生成されます。コマンド ラインからアップグレードしたプロジェクトに対しては、レポートは生成されません。

アップグレード レポートは、Visual Basic 2005 統合開発環境 (IDE: Integrated Development Environment) または任意の Web ブラウザで表示できます。

メモ：
使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual Studio 2005 でアップグレード レポートを表示するには

- ソリューション エクスプローラで [_UpgradeReport.htm] ノードをダブルクリックしてアップグレード レポートを開きます。

Web ブラウザでアップグレード レポートを表示するには

1. ソリューション エクスプローラで [_UpgradeReport.htm] ノードを選択します。
2. [ファイル] メニューの [ブラウザの選択] をクリックします。
3. [ブラウザの選択] ダイアログ ボックスで Web ブラウザを選択します。

参照

概念

Visual Basic 6.0 アプリケーションのアップグレードの準備

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

方法：コマンド ラインからプロジェクトをアップグレードする

Visual Basic アップグレード ウィザードを使用する以外に、コマンド ライン ツール Vbupgrade.exe を使用して Visual Basic 6.0 のプロジェクトを Visual Basic 2005 にアップグレードすることもできます。

メモ：

Vbupgrade.exe は、既定ではパスに含まれていません。コマンド ラインから Vbupgrade.exe を実行する前に、パスに追加するか、または Vbupgrade.exe がインストールされているディレクトリに切り替える必要があります。既定のインストール場所は、C:\Program Files\Microsoft Visual Studio 8\Vb\VBUpgrade です。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コマンド ラインからプロジェクトをアップグレードするには

1. Visual Basic 6.0 でプロジェクトを開きます。[ファイル] メニューの [<プロジェクト名> の作成] をクリックします。アプリケーションを実行し、エラーがないことを確認します。
2. コマンド プロンプトで、「**Vbupgrade <プロジェクト名>**」に加えて、コマンド ライン引数を入力します。<*projectname*> は、Visual Basic 6.0 アプリケーションの完全パスです。

   メモ：

   Vbupgrade コマンド ライン構文とオプションの詳細については、「Vbupgrade のコマンド ライン構文」を参照してください。

3. 作成されたプロジェクトを Visual Basic 2005 で開きます。コードを確認します。アップグレード時に、潜在的な問題について警告する注記がコードに挿入されています。エラーがある場合は、アプリケーションを実行する前に修正する必要があります。警告がある場合は、内容を調べて、アプリケーションの動作に影響がないことを確認します。

   ヒント

   エラーまたは警告の詳細については、記述をクリックしてヘルプ トピックを参照してください。

4. [デバッグ] メニューの [開始] をクリックします。アプリケーションを実行し、エラーがないこと、および Visual Basic 6.0 の場合と動作が同じであることを確認します。

   メモ：

   Vbupgrade ツールは、可能性のある非互換性をすべて検出するわけではありません。アプリケーションを実行してから新たなエラーに遭遇する場合もあります。既知の問題の一覧については、「Visual Basic 6.0 からアップグレードされたアプリケーションのトラブルシューティング」を参照してください。

参照

処理手順

方法 : アップグレード レポートを表示する

関連項目

Vbupgrade のコマンド ライン構文

概念

Visual Basic 6.0 アプリケーションのアップグレードの準備

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

Vbupgrade のコマンド ライン構文

Vbupgrade.exe コマンド ライン ツールを使って、Visual Basic 6.0 アプリケーションを Visual Basic 2005 にアップグレードできます。

```
vbupgrade[.exe] inputname [/out outputname] [/verbose [/nolog [/logfile logfilename]]]
vbupgrade[.exe] /?
```

引数
inputname

必ず指定します。

アップグレードする Visual Basic 6.0 プロジェクトのパスとファイル名を指定します。

/out *outputname*

Visual Basic 2005 プロジェクトを作成するフォルダのパスを指定します。既定のパスは \OutDir です。

/verbose

アップグレード中にすべての出力をコマンド プロンプト ウィンドウに表示します。

/nolog

アップグレード中にログ ファイルを作成しません。

/logfile *logfilename*

アップグレード中に作成されるログ ファイルのパスとファイル名を指定します。パスとファイル名を指定しない場合、ログ ファイルは Visual Basic 2005 プロジェクトと同じフォルダに作成されます。既定のファイル名は *ProjectFileName*.log です。*ProjectFileName* はプロジェクト ファイルの名前です。

/?

コマンド ライン オプションの一覧を表示します。

解説

スペースを含むパスまたはファイル名は、引用符で囲む必要があります。

参照
処理手順
方法 : コマンド ラインからプロジェクトをアップグレードする
関連項目
アップグレード ウィザード

Visual Basic 6.0 からアップグレードされたアプリケーションのトラブルシューティング

Visual Basic 2005 のアップグレード ツールは、アップグレードされたアプリケーションのあらゆる問題点を検出して報告するように注意深くデザインされていますが、一部のケースでは問題が正しく検出されない場合もあります。ここでは、アップグレード ツールでは検出されない既知の問題とその対処方法の一覧を示します。

Visual Studio をアップグレードした後でヘルプ リンクにアクセスできない

以前のバージョンの Visual Studio でアップグレードしたアプリケーションを使用している場合に、アップグレード ツールによって挿入されたヘルプ リンクをクリックすると、"ページが見つかりません。" エラーが表示されることがあります。このエラーは、リンクが以前のバージョンのヘルプ コレクションを参照していて、ヘルプ リンクの形式が変更されたために発生します。

この問題を解決するには、エラー文字列をコピーし、ヘルプの検索機能を使用して該当するトピックを検索します。エラー文字列の中に変数が含まれていることもあるので、その場合は一部の文字列だけを使用して検索してください。

ユーザー定義型における固定長文字列の動作の相違点

Visual Basic 6.0 では、ユーザー定義型の中の固定長文字列に割り当てられた文字列がその固定長を超える場合、文字列が自動的に切り詰められます。Visual Basic 2005 にアップグレードすると、文字列の切り詰め処理は行われません。このことにより、不正確な結果が生じる場合があります。

メモ :

アップグレード時に、ユーザー定義型の中の固定長文字列に **VBFixedString** 属性が追加されます。この属性により、Visual Basic 互換ライブラリのファイル関数が、それらを固定長文字列として扱うことができます。

この問題を修正するには、文字列を固定長文字列に割り当てるコードを検索し、文字列の長さをチェックするコードを追加し、必要に応じて文字列の切り詰め処理を行います。

VB

```
' Before
MyString = "1234567"
MyStruct.FixedString5 = MyString

' After
MyString = "1234567"
If Len(MyString) > 5 Then
  MyString = Microsoft.VisualBasic.Left(MyString, 5)
End If
MyStruct.FixedString5 = MyString
```

フォームを閉じると Dispose メソッドが呼び出される

Visual Basic 6.0 では、フォームをアンロードした後、**Show** メソッドを呼び出すことでそのフォームを再読み込みできます。Visual Basic 2005 では、フォームの **Close** メソッドを使用すると **Dispose** メソッドが呼び出されるので、自動的にガベージ コレクションが行われます。これにより、検出の困難な、動作上のわずかな違いが生じる場合があります。

- Visual Basic 2005 では、アンロードされたフォームの **Show** メソッドを呼び出した場合、そのフォームの新しいインスタンスが生成されます。そのため、基本クラスのプロパティ設定はすべて失われます。
- モーダル フォームの場合、**Dispose** は自動的に呼び出されません。場合によっては、リソースを解放するために **Dispose** を使用することもできます。

COM オブジェクトを遅延バインディングによって呼び出すと型の不一致エラーが発生する

Visual Basic 6.0 では、遅延バインディング COM オブジェクトが、遅延バインディングによる呼び出しのパラメータとして渡されると、そのオブジェクトは強制的に **Nothing** 型の **Variant** に変換されます。Visual Basic 2005 にアップグレードすると、**Object** 型として宣言された COM オブジェクトは、**Variants** と同じように処理されます (アップグレード時に常に **Object** 型に変換されます)。これらのオブジェクトはバリアント型の **Empty** にマーシャリングされます。このことが、Visual Basic 2005 で型の不一致が発生する原因となります。

この問題を修正するには、すべてのオブジェクトが事前バインディングされていることを確認する必要があります。

Err.Number によって返される値が異なる場合がある

Visual Basic 2005 によって返されるエラーは、Visual Basic 6.0 によって返されるエラーとは異なる場合があります。**Err.Number** の戻り値に依存するエラー処理コードの場合、アプリケーションの動作が変わってしまう可能性があります。

この例を次のコードに示します。

```
' Visual Basic 6.0
On Local Error GoTo Result
Dim x() As Boolean
Dim y As Variant

y = x(10)

Result:
If Err.Number = 9 Then
   ' Do something.
Else
   ' Do something else.
End If
```

アップグレード前は、**Err.Number** は常に 9 (インデックスが有効範囲にありません。) を返すため、**If** ステートメントの前半が実行されます。アップグレード後は、91 (オブジェクト変数または With ブロック変数が設定されていません。) が返されるため、**Else** 句が実行されます。これは、Visual Basic 6.0 では配列を宣言時に初期化するのに対し、Visual Basic 2005 では配列を参照する前に初期化するためです。

Err.Number の戻り値に依存するコードでは、結果を慎重にテストし、必要に応じてコードを変更する必要があります。

参照

概念

Visual Basic 6.0 と現在のバージョンの Visual Basic の両方を使用する

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

Visual Basic 6.0 と現在のバージョンの Visual Basic の両方を使用する

Visual Basic 2005 と Visual Basic 6.0、およびいずれかの言語で作成したアプリケーションは、すべて同じコンピュータにインストールして同時に実行できます。

Visual Basic 2005 で作成されたコンポーネントは、以前のバージョンの Visual Basic や他の言語で作成された COM コンポーネントと相互運用できます。たとえば、Visual Basic 6.0 で記述された ActiveX コントロールを Visual Basic 2005 の Windows フォームにドラッグしたり、Visual Basic 2005 のクラス ライブラリから Visual Basic 6.0 の COM オブジェクトを使用したり、Visual Basic 2005 のライブラリへの参照を Visual Basic 6.0 の実行可能ファイルに追加したりできます。

Visual Basic 2005 でコンパイルされたコンポーネントと Visual Basic 6.0 でコンパイルされたコンポーネントでは、実行時に微妙な違いがあります。Visual Basic 2005 のオブジェクトはガベージ コレクションによって解放されるため、オブジェクトが明示的に破棄されたときに、オブジェクトが実際にメモリから削除されるまでの間に時間差が生じることがあります。さらに、データ型やその他の言語上の変更に関する相違点もあります。これらの違いのため、Visual Basic 2005 アプリケーションと Visual Basic 6.0 アプリケーションの実行時の動作は、似てはいますがまったく同じではありません。

さらに、Visual Basic 2005 では、Visual Basic 2005 コンポーネントと Visual Basic 6.0 コンポーネントのバイナリ互換性が不要になっています。コンポーネントのバージョン管理と配置システムはより信頼性が高くなりました。ファイルはディレクトリをコピーするだけで配置でき、新しいバージョンのコンポーネントへのアップグレードは古いファイルを新しいファイルで置き換えるのと同じくらいに簡単です。ユーザーは、クラスおよびメソッドが以前のバージョンとの互換性を持つようにするだけで済みます。

参照

概念

アップグレードが必要である理由

Visual Basic 6.0 アプリケーションのアップグレードの準備

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

チュートリアル : Visual Basic 6.0 アプリケーションから現在のバージョンの Visual Basic へのアップグレード

このチュートリアルでは、Visual Basic のグラフィック サンプル アプリケーションを Visual Basic 2005 にアップグレードします。アップグレード プロセスの説明に加えて、グラフィック アーキテクチャにおける Visual Basic 6.0 と Visual Basic 2005 の違いについて重点的に説明します。この情報は、アプリケーションで生じるアーキテクチャ上の問題に対処する場合にも役に立ちます。

この例では、2 つのボタン、1 つのピクチャ ボックス、1 つのタイマ、および 1 つの非表示イメージ コントロールを実装する簡単なサンプル アプリケーションを使用します。頻繁に発生することはありませんが、アプリケーションをアップグレードする場合に直面する可能性のある困難な事例をいくつか挙げます。

メモ :
このチュートリアルを使用するには、開発コンピュータに Visual Basic 6.0 がインストールされている必要があります。

Visual Basic 6.0 アプリケーションを作成するには

1. Visual Basic 6.0 を開きます。[ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。
2. [新しいプロジェクト] ダイアログ ボックスの [標準 EXE] をクリックし、[OK] をクリックします。
3. **PictureBox** コントロールをフォームに追加し、下部に空白を残して、フォームの大部分を占めるようにコントロールのサイズを設定します。
4. **AutoRedraw** プロパティを **True**、**DrawStyle** プロパティを **0-Solid**、**FillStyle** プロパティを **0-Solid** に設定します。
5. 2 つの **CommandButton** コントロールを **PictureBox** コントロールに追加します。
6. 1 番目の **CommandButton** を選択し、**Name** プロパティを `ClearPictureBox`、**Caption** プロパティを `Clear` に設定します。
7. 2 番目の **CommandButton** を選択し、**Name** プロパティを `ShowImage`、**Caption** プロパティを `Show Image` に設定します。
8. **Image** コントロールをフォームに追加し、**Name** プロパティを `sourceImage`、**Visible** プロパティを **False** に設定します。
9. **Image** コントロールの **Picture** プロパティをビットマップ イメージに設定します。マイ ピクチャ フォルダ内のいずれかの画像を使用できます。
10. **Timer** コントロールをフォームに追加します。**Enabled** プロパティを **False**、**Interval** プロパティを 25 に設定します。
11. フォームをダブルクリックしてコード エディタを開き、次のコードを入力します。

```
Option Explicit
Private LeftPos As Double

Private Sub ClearPictureBox_Click()
    Picture1.Cls
End Sub

Private Sub ShowImage_Click()
    LeftPos = 1
    Me.Timer1.Enabled = True
End Sub

Private Sub Timer1_Timer()

    If LeftPos <= 0 Then
        Me.Timer1.Enabled = False
        Picture1.Print "Visual Basic ROCKS!"
    Else
        LeftPos = LeftPos - 0.01
        Picture1.Cls
```

```
            Picture1.PaintPicture sourceImage, LeftPos * _
    Picture1.Width, 0
        End If
    End Sub
```

12. F5 キーを押してアプリケーションを実行します。ボタンをクリックして動作を確認し、コードを 1 ステップずつ進みながらその動作を確認します。
13. [ファイル] メニューの [名前を付けてプロジェクトの保存] をクリックします。
14. [名前を付けてファイルを保存] ダイアログ ボックスで、フォームを `PicForm.frm` として保存し、プロジェクトを `Drawing.vbp` として保存します。

アップグレード ウィザードを実行するには

1. Visual Basic 2005 を開きます。[ファイル] メニューの [プロジェクトを開く] をクリックします。
2. [プロジェクトを開く] ダイアログ ボックスで、Drawing.vbp ファイルを探してそれを開きます。

   これにより、Visual Basic アップグレード ウィザードが起動します。最初のページに、ウィザードが実行する操作についての説明が表示されます。
3. [次へ] をクリックして、ウィザードの次のページに移動します。このページにはアップグレードのオプションが表示されますが、この場合は、選択できるオプションはありません。
4. [次へ] をクリックして、ウィザードの 3 ページ目に進みます。[次へ] をクリックして新規プロジェクトの既定の場所を受け入れるか、別の場所を入力します。既定の場所を受け入れた場合、新しいフォルダは Visual Basic 6.0 プロジェクト フォルダの直下に作成されます。

   新しくフォルダを作成するかどうかを確認するメッセージが表示された場合は [はい] をクリックします。
5. ウィザードの 4 ページ目で、[次へ] をクリックしてアップグレードを開始します。

   アップグレードが完了すると、ウィザードが終了し、ソリューション エクスプローラに新しいプロジェクトが表示されます。

アップグレード結果を表示するには

1. ソリューション エクスプローラで _UpgradeReport.htm を選択し、ダブルクリックしてアップグレード レポートを開きます。

   このレポートには、対処の必要なグローバルな問題は報告されません。PicForm.vb については、6 つのエラーが示され、警告は示されません。

   メモ :

   警告がある場合は、[タスク一覧] ウィンドウに UPGRADE_WARNING アイテムとして表示されます。警告は、アプリケーションの実行時の動作に多少の相違が生じる原因となるコードを示します。[タスク一覧] で UPGRADE_WARNING をダブルクリックすると、変更が必要なコードに直接移動できます。
2. PicForm.vb セクションを展開するには、New Filename 列のプラス記号をクリックします。これにより、フォームのアップグレードに関する問題の詳細な一覧が表示されます。

   問題は 3 か所 (**ClearPictureBox_Click** プロシージャ、**Timer1_Timer** プロシージャ、およびフォーム レイアウト) に対して適用されることに注意してください。各問題を説明するヘルプ トピックを表示するには、[説明] リンクをクリックします。この場合、レイアウトの問題に対する処理は必要ありません。ヘルプを確認すると、この問題は Visual Basic 6.0 PictureBox コントロールの 2 つのプロパティに相当する機能が、Visual Basic 2005 のプロパティには存在しないことが原因であることがわかります。

ClearPictureBox_Click エラーを修正するには

1. ソリューション エクスプローラで PicForm.vb を選択します。[表示] メニューの [コード] をクリックします。
2. コード エディタで、**ClearPictureBox_Click** プロシージャを選択します。

   ヒント

プロシージャには、UPGRADE_ISSUE コメントが追加されています。このコメントの最後までスクロールすると、関連するヘルプ トピックをクリックして表示するためのリンクが用意されています。レイアウトの問題を除くすべての問題は、アップグレード時にコードのコメントとして追加されます。このコメントには、アップグレード時に生じた問題について説明する UPGRADE_NOTE コメントが含まれます。これはアップグレード レポートには報告されません。

3. **ClearPictureBox_Click** プロシージャに次のコードを追加します。

   VB

```
Dim g As Graphics = Picture1.CreateGraphics()
g.Clear(Picture1.BackColor)
g.Dispose()
```

   元のプロシージャの唯一のエラーは、**Picture1.Cls()** メソッド呼び出しです。Visual Basic 6.0 とは異なり、Visual Basic 2005 の組み込みコントロールでは、メソッドによる描画サーフェイスへの直接アクセスは提供されません。すべてのグラフィックス操作は、**Graphics** 型の特別なオブジェクトによって処理されます。**CreateGraphics()** メソッドを呼び出すことにより、コントロールの描画サーフェイスにアクセスして、**Graphics** オブジェクトのインスタンスを取得できます。

4. 元の行の `Picture1.Cls()` を削除するか、またはコメント アウトします。

Timer1_Timer エラーを修正するには

1. コード エディタで、**Timer1_Tick** プロシージャを選択します。

   メモ：

   Visual Basic 6.0 **Timer** コントロールの **Timer** イベントは、Visual Basic 2005 **Timer** コンポーネントの **Tick** イベントにアップグレードされます。アップグレード レポートには従来のイベント名が示され、コード エディタでは新しいイベント名が使用されます。

2. 関数の先頭に次のコードを追加します。

   VB

```
Dim g As Graphics = Picture1.CreateGraphics()
```

   前のプロシージャと同様、このエラーも **Graphics** オブジェクトが不足していることに起因します。

3. 最初の UPGRADE_ISSUE である **Picture1.Print** メソッドを検索します。次のコードを追加します。

   VB

```
g.DrawString("VB .NET ROCKS!", Me.Font, New SolidBrush( _
  Color.Yellow), 0, 0)
```

   Graphics オブジェクトの **DrawString** メソッドが、Visual Basic 6.0 の **Print** メソッドに置き換わります。**Print** メソッドが 1 つの **Text** 引数を受け取るのに対して、**DrawString** も、**Font**、**Color** オブジェクトを指定する **Brush** オブジェクト、およびテキストを描画する開始座標を指定する引数を受け取ります。

4. 元の行の `Picture1.Print("Visual Basic ROCKS!")` を削除するか、またはコメント アウトします。

5. 次の UPGRADE_ISSUE である **Picture1.Cls()** を検索します。このコードを次のコードを使って置き換えます。

   VB

```
g.Clear(Picture1.BackColor)
```

6. 最後の UPGRADE_ISSUE である **Picture1.PaintPicture** を検索します。次のコードを追加します。

VB

```
g.DrawImage(sourceImage.Image, CSng(LeftPos * Picture1.Size.Width), _
  0)
```

ここでは、**Graphics** クラスの **DrawImage** メソッドが **PaintPicture** メソッドに置き換えられています。

7. 元の行の `Picture1.PaintPicture(sourceImage, LeftPos * VB6.PixelsToTwipsX(Picture1.Width), 0)` を削除するか、またはコメント アウトします。
8. プロシージャの最後に次のコードを追加します。

VB

```
g.Dispose()
```

Graphics オブジェクトに関連付けられたメモリ リソースを解放するには、**Dispose** メソッドが必要です。

アプリケーションをテストするには

1. [デバッグ] メニューの [開始] をクリックします。

メモ：

ソリューション ファイルを保存するかどうかを確認するメッセージが表示されたら、[保存] をクリックしてファイルを保存し、アプリケーションを実行します。

2. [ShowImage] をクリックします。

   イメージの描画時に、ちらつきが生じることがあります。これは、更新された各イメージが画面に描画される前に、領域全体がクリアされるためです。

アプリケーションのさらなる変更

これを解決するには、**Graphics** オブジェクトで領域全体を描画するのではなく、画面内の必要な箇所だけを描画する必要があります。画面上にイメージを描画するときに、**DrawImage()** への最後の呼び出しの後に残ったピクセルを背景色で隠すことが必要な場合があります。イメージは右から左に移動するため、隠す必要があるのは、ピクチャの右側の余分なピクセルだけです。

画面のちらつきの問題を修正するには

1. **Timer1_Tick** プロシージャで、`g.Clear(Picture1.BackColor)` メソッドを次のコードに置き換えます。

VB

```
Dim imageWidth As Integer = sourceImage.Image.Width
Dim imageHeight As Integer = sourceImage.Image.Height
Dim left As Double = imageWidth + (Picture1.Size.Width * LeftPos)

g.FillRectangle(New SolidBrush(Me.BackColor), _
  New Rectangle(left, 0, 6, imageHeight))
```

Clear メソッドの代わりに **FillRectangle** メソッドを使用して、描画中の現在のイメージの右側の残留ピクセルを背景色を使って隠します。これを行うには、最初に四角形の左端の座標を計算する必要があります。これは先頭の 3 行のコードによって実現されます。

2. [デバッグ] メニューの [開始] をクリックします。
3. [ShowImage] をクリックします。

   今度はイメージを移動してもちらつきは発生しません。

修正が必要な問題は、もう 1 つあります。**PictureBox** コントロールに表示されるメッセージのフォントが、元のバージョンのフォントよりも小さいこ

とです。フォントが小さいのは、Visual Basic 6.0 アプリケーションで、デザイン時に **PictureBox** コントロールの **Font** プロパティが Arial Bold 16 に設定されていたためです。Visual Basic 2005 の **PictureBox** コントロールには、**Font** プロパティはありません。代わりに、フォームの既定フォント (`Me.Font`) が使用されます。既定フォントは、新しい **Font** オブジェクトを宣言することで修正できます。

フォントを修正するには

1. **Timer1_Tick** プロシージャに次の宣言を追加します。

   VB

   ```
   Dim f As System.Drawing.Font = New System.Drawing.Font("Arial", _
     16, FontStyle.Bold)
   ```

2. `g.DrawString` の呼び出しで、フォント パラメータを `Me.Font` から `f` に変更します。

3. [デバッグ] メニューの [開始] をクリックします。

4. [ShowImage] をクリックします。

   これで、テキストが適切なフォントで表示されます。

これで、アプリケーションは元の Visual Basic 6.0 アプリケーションと等価な状態になりました。次に、このアプリケーションを強化するコードを追加します。結局のところ、Visual Basic 2005 を利用する必要がない場合は、アプリケーションをアップグレードしても意味はありません。

次の手順では、フォームを閉じるときにフェード アウトする機能を追加します。

メモ :

フェード効果は 256 色のディスプレイでは表示できません。この効果を表示するには、画面の色を High Color または True Color に設定する必要があります。**Opacity** プロパティは、Windows XP だけで使用できます。

アプリケーションを強化するには

1. コード エディタで、[クラス名] ボックスの一覧の [(Form1 Events)] をクリックします。

2. [メソッド名] ボックスの一覧で **FormClosing** イベントを選択します。

3. **Form1_FormClosing** プロシージャに次のコードを追加します。

   VB

   ```
   Dim i As Single
   For i = 1 To 0 Step -0.1
     Me.Opacity = i
     Application.DoEvents()
     System.Threading.Thread.Sleep(100)
   Next
   ```

4. [デバッグ] メニューの [開始] をクリックします。

5. フォームを閉じてフェード アウトの動作を確認します。

   Step のサイズを増やすか、または **Sleep** の遅延を減らすことによって、フェード アウトのスピードを制御できます。

参照

関連項目

Windows フォームの概要

概念

グラフィックス (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Visual Basic 6.0 からのアップグレード

Visual Basic 6.0 のアップグレード リファレンス

目次の次のセクション内のトピックは、アップグレード ウィザードおよび **VisualBasic.Compatibility** 名前空間に関係するリファレンス トピックです。

参照

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

アップグレード ウィザード

Visual Basic 6.0 プロジェクトを Visual Basic 2005 にアップグレードするには、アップグレード ウィザードを使用します。このウィザードは、新しいプロジェクトを作成し、元のプロジェクトから新しいプロジェクトに各ファイルをコピーし、必要に応じてファイルを変更し、ウィザードが行った処理およびアップグレードを完了するためにユーザーが行う必要のある作業の詳細を示したレポートを生成します。

アップグレード ウィザードを開くには、Visual Basic 2005 で任意の Visual Basic 6.0 プロジェクト ファイル (.vbp) を開きます。

ユーザー インターフェイス要素の一覧

アップグレード ウィザードでは、以下のオプションを利用できます。

[アップグレード後のプロジェクトの種類]

Visual Basic 6.0 のプロジェクトが ActiveX EXE プロジェクトまたは ActiveX ドキュメント EXE プロジェクトの場合は、.exe ファイルまたは .dll ファイルへアップグレードできます。それ以外のすべてのプロジェクト タイプについては、適切なオプションが既に選択されていて変更できません。

[新しいプロジェクトを作成する場所]

新しいプロジェクトの場所を指定します。既定では、新しいプロジェクトは、Visual Basic 6.0 プロジェクトのすぐ下のディレクトリに格納されます。新しいプロジェクトには、*ProjectName*.NET という名前が設定されます。*ProjectName* は、アップグレードされたプロジェクトの名前です。

参照

概念

Visual Basic アップグレード ウィザード

その他の技術情報

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

VisualBasic.Compatibility 名前空間リファレンス

VisualBasic.Compatibility 名前空間の関数およびオブジェクトは、Visual Basic 6.0 から Visual Basic 2005 へのアップグレード ツールで使用するために用意されています。これらの関数およびオブジェクトを使用することで、多くの場合、.NET Framework のその他の部分を使用して実現できる機能を達成できます。したがって、Visual Basic 6.0 のコード モデルが .NET の実装と著しく異なる場合を除いて、その他の部分を使用する必要はありません。

名前空間の関数およびオブジェクト

VisualBasic.Compatibility 名前空間の関数およびオブジェクトの一覧を次の表に示します。**Microsoft.Compatibility.Data** に関連するその他の関数およびオブジェクトは一覧にはありません。それらは、Visual Basic 6.0 の ADO データ バインディングの関数およびオブジェクトと等価です。

VisualBasic.Compatibility 関数

関数	説明
CopyArray	配列をバリアントに割り当てる Visual Basic 6.0 の機能を複製します。
CursorToIPicture	書式を変換するときに使用します。
Eqv	Visual Basic 6.0 の **Eqv** (等価) 演算子を複製します。
FontChangeBold	Visual Basic 6.0 のフォントを Visual Basic 2005 の **Font** オブジェクトに変換するときに使用します。
FontChangeGdiCharSet	Visual Basic 6.0 のフォントを Visual Basic 2005 の **Font** オブジェクトに変換するときに使用します。
FontChangeItalic	Visual Basic 6.0 のフォントを Visual Basic 2005 の **Font** オブジェクトに変換するときに使用します。
FontChangeName	Visual Basic 6.0 のフォントを Visual Basic 2005 の **Font** オブジェクトに変換するときに使用します。
FontChangeSize	Visual Basic 6.0 のフォントを Visual Basic 2005 の **Font** オブジェクトに変換するときに使用します。
FontChangeStrikeout	Visual Basic 6.0 のフォントを Visual Basic 2005 の **Font** オブジェクトに変換するときに使用します。
FontChangeUnderline	Visual Basic 6.0 のフォントを Visual Basic 2005 の **Font** オブジェクトに変換するときに使用します。
FontToIFont	書式を変換するときに使用します。
Format	Visual Basic 6.0 の **Format** 関数を複製します。
FromPixelsX	座標を変換するときに使用します。
FromPixelsY	座標を変換するときに使用します。
FromPixelsUserHeight	座標を変換するときに使用します。
FromPixelsUserWidth	座標を変換するときに使用します。
FromPixelsUserX	座標を変換するときに使用します。
FromPixelsUserY	座標を変換するときに使用します。
GetActiveControl	Visual Basic 6.0 の **Screen.ActiveControl** プロパティを複製します。
GetCancel	Visual Basic 6.0 の **CommandButton** から **Cancel** プロパティの機能を複製します。

GetDefault	Visual Basic 6.0 の **CommandButton** の **Default** プロパティの機能を複製します。
GetEXEName	Visual Basic 6.0 の **App.EXEName** プロパティを複製します。
GetHInstance	Visual Basic 6.0 の **App.HInstance** プロパティを複製します。
GetItemData	Visual Basic 6.0 の **ListBox** または **ComboBox** から **ItemData** プロパティの機能を複製します。
GetItemString	Visual Basic 6.0 の **ListBox** または **ComboBox** からリストの値を取得する機能を複製します。
GetPath	Visual Basic 6.0 の **App.Path** プロパティを複製します。
IconToIpicture	書式を変換するときに使用します。
IFontToFont	書式を変換するときに使用します。
ImageToIPicture	書式を変換するときに使用します。
ImageToIPictureDisp	書式を変換するときに使用します。
Imp	Visual Basic 6.0 の **Imp** (暗黙) 演算子を複製します。
IPictureDispToImage	書式を変換するときに使用します。
IPictureToImage	書式を変換するときに使用します。
LoadResData	Visual Basic 6.0 の **LoadResData** 関数を複製します。
LoadResPicture	Visual Basic 6.0 の **LoadResPicture** 関数を複製します。
LoadResString	Visual Basic 6.0 の **LoadResString** 関数を複製します。
PixelsToTwipsX	座標を変換するときに使用します。
PixelsToTwipsY	座標を変換するときに使用します。
SendKeys	Visual Basic 6.0 の **SendKeys** 関数を複製します。
SetCancel	Visual Basic 6.0 の **CommandButton** から **Cancel** プロパティの機能を複製します。
SetDefault	Visual Basic 6.0 の **CommandButton** の **Default** プロパティの機能を複製します。
SetItemData	Visual Basic 6.0 の **ListBox** または **ComboBox** から **ItemData** プロパティの機能を複製します。
SetItemString	Visual Basic 6.0 の **ListBox** または **ComboBox** からリストに値を設定する機能を複製します。
SetResourceBaseName	リソース ファイルに標準以外の名前付け規則が使用されているときに、**LoadRes** 関数で使用します。
ShowForm	Visual Basic 6.0 のフォームから **Show** メソッドの機能を複製します。
TabLayout	**Tab** キーワードまたは **Spc** キーワードを含む **Debug.Print** ステートメントを変換するときに使用します。

ToPixelsX	座標を変換するときに使用します。
ToPixelsY	座標を変換するときに使用します。
ToPixelsUserHeight	座標を変換するときに使用します。
ToPixelsUserWidth	座標を変換するときに使用します。
ToPixelsUserX	座標を変換するときに使用します。
ToPixelsUserY	座標を変換するときに使用します。
TwipsPerPixelX	Visual Basic の **Screen.TwipsPerPixelX** プロパティを複製します。
TwipsPerPixelY	Visual Basic の **Screen.TwipsPerPixelY** プロパティを複製します。
TwipsToPixelsX	座標を変換するときに使用します。
TwipsToPixelsY	座標を変換するときに使用します。
ValidateControls	Visual Basic 6.0 のフォームから **ValidateControls** メソッドの機能を複製します。
WhatsThisMode	Visual Basic 6.0 のフォームから **WhatsThisMode** メソッドの機能を複製します。
Zorder	コントロールの z オーダーを変換するために使用します。

VisualBasic.Compatibility オブジェクト

オブジェクト	説明
BaseControlArray	Visual Basic 6.0 のコントロール配列エミュレーションの親クラス。
BaseOcxArray	エミュレートされた ActiveX コントロール配列の親クラス。
ButtonArray	Visual Basic 6.0 の **CommandButton** コントロールのコントロール配列をエミュレートします。
CheckBoxArray	Visual Basic 6.0 の **CheckBox** コントロールのコントロール配列をエミュレートします。
CheckedListBoxArray	Visual Basic 6.0 の **ListBox** コントロールの **Style** プロパティが Checked に設定されたコントロール配列をエミュレートします。
ComboBoxArray	Visual Basic 6.0 の **ComboBox** コントロールのコントロール配列をエミュレートします。
DirListBox	Visual Basic 6.0 の **DirListBox** コントロールをエミュレートします。
DirListBoxArray	Visual Basic 6.0 の **DirListBox** コントロールのコントロール配列をエミュレートします。
DriveListBox	Visual Basic 6.0 の **DriveListBox** コントロールをエミュレートします。
DriveListBoxArray	Visual Basic 6.0 の **DriveListBox** コントロールのコントロール配列をエミュレートします。
FileListBox	Visual Basic 6.0 の **FileListBox** コントロールをエミュレートします。
FileListBoxArray	Visual Basic 6.0 の **FileListBox** コントロールのコントロール配列をエミュレートします。

FixedLengthString	Visual Basic 6.0 の固定長文字列をエミュレートします。
GroupBoxArray	Visual Basic 6.0 の **Frame** コントロールのコントロール配列をエミュレートします。
HScrollBarArray	Visual Basic 6.0 の **HScrollBar** コントロールのコントロール配列をエミュレートします。
LabelArray	Visual Basic 6.0 の **Label** コントロールのコントロール配列をエミュレートします。
ListBoxArray	Visual Basic 6.0 の **ListBox** コントロールのコントロール配列をエミュレートします。
ListBoxItem	**ListBox** コントロールの **ItemData** プロパティをエミュレートします。
MenuItemArray	Visual Basic 6.0 の **Menu** コントロールのコントロール配列をエミュレートします。
PanelArray	Visual Basic 6.0 の、子コントロールを含む **PictureBox** コントロールのコントロール配列をエミュレートします。
PictureBoxArray	Visual Basic 6.0 の **PictureBox** コントロールのコントロール配列をエミュレートします。
RadioButtonArray	Visual Basic 6.0 の **OptionButton** コントロールのコントロール配列をエミュレートします。
TabControlArray	Visual Basic 6.0 の **TabStrip** コントロールのコントロール配列をエミュレートします。
TextBoxArray	Visual Basic 6.0 の **TextBox** コントロールのコントロール配列をエミュレートします。
TimerArray	Visual Basic 6.0 の **Timer** コントロールのコントロール配列をエミュレートします。
VscrollBarArray	Visual Basic 6.0 の **VScrollBar** コントロールのコントロール配列をエミュレートします。

参照

概念

Visual Basic 6.0 互換性ライブラリ

その他の技術情報

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic でのアプリケーションの開発

このセクションでは、Visual Basic 言語の概念説明のドキュメントについて取り上げます。

このセクションの内容

Visual Basic Editor の使用

Visual Basic エディタの機能 (Intellisense コード スニペット、プロジェクトのプロパティ、Visual Basic 用の XML ドキュメントの作成など) についてのヘルプが含まれています。

セキュリティと Visual Basic 開発

セキュリティ上の懸念への対処が必要な、よくある 3 つの状況について説明します。

Visual Basic でのプログラミング

プログラミングのさまざまな側面について説明します。

My による開発

新しい機能 **My** について説明します。これは、アプリケーションとその実行時環境に関連する情報および既定のオブジェクト インスタンスへのアクセスを提供します。

Visual Basic アプリケーションにおけるデータ アクセス

Visual Basic でのデータ アクセスについてのヘルプが含まれています。

Visual Basic アプリケーションのデバッグ

Visual Studio に組み込まれているデバッグ機能に関するドキュメントへのリンクを示します。

Visual Basic での例外およびエラー処理

構造化および非構造化例外処理、警告の構成、スマート コンパイル自動修正、およびエラーの種類についてのヘルプが含まれています。

アプリケーションの配置

.NET Framework と Visual Studio の、配置の機能に関する概要を説明します。

Visual Basic でのコンポーネントの作成および使用

コンポーネントという用語の意味と、コンポーネントを作成する方法および目的について説明します。

Visual Basic への理解を深める

フォーム、ネットワーク処理、Web サービスなど、さまざまな側面に関する概要を説明します。

Visual Basic でのプロジェクトのカスタマイズと My の拡張

プロジェクト テンプレートをカスタマイズして、追加的な **My** オブジェクトを提供する方法について説明します。

関連するセクション

Visual Basic のプログラミング ガイド

Visual Basic でのプログラミングに不可欠な要素についてひととおり説明します。

Visual Basic リファレンス

Visual Basic 言語のリファレンス ドキュメントが含まれています。

Visual Basic Editor の使用

Visual Studio に用意されている基本的なユーザー インターフェイス機能の他に、Visual Basic には、生産性と操作性を向上させるための機能が追加されています。

このセクションの内容

Visual Basic の設定

開発環境における操作性を最適化する方法について説明します。

Visual Basic の IntelliSense コード スニペット

開発環境から利用できるコードのライブラリを紹介します。

方法 : コード属性を編集する

[プロパティ] ウィンドウを使用して、属性を型とメンバに適用して編集する方法について説明します。

[名前の変更] ダイアログ ボックス (Visual Basic)

コード内の識別子の名前を変更する際のしくみについて説明します。

Visual Basic 6.0 の既定のショートカット キー

ショートカット キーの一覧を示すさまざまなページへのリンクを提供します。

Visual Basic コードのイベント ハンドラのヘルプ

イベント ハンドラから F1 ヘルプを取得する方法について説明します。

方法 : Visual Basic でプロジェクトをコンパイルして実行する

開発環境内からのコンパイルおよび実行に使用するオプションについて説明します。

参照

エディット コンティニュ (Visual Basic)

中断モードでの実行中に、コードを変更することを可能にするデバッグ機能について説明します。

スマート コンパイル自動修正

エラーが発生したとき、この機能がどのようにして修正方法の候補を表示し、コードに適用する解決策の選択を可能にするかについて説明します。

Visual Basic での警告の構成

警告を無効にしたり、警告をエラー メッセージとして扱う方法について説明します。

Visual Basic の設定

Visual Basic の設定は、ユーザーの操作性を最適化し、Visual Basic 開発者の生産性を最大限に引き上げることを目的としています。これは、Visual Studio の設定 機能の一部です。このような機能を使用すると、Visual Studio 統合開発環境 (IDE: Integrated Development Environment) の設定をカスタマイズおよび保存できるようになります。

また、IDE の設定は、別のコンピュータに移植したり、再読み込みしたりできます。詳細については、「方法 : コンピュータ間で設定を共有する」を参照してください。

ウィンドウとビュー

機能	既定で表示される	備考
クラス ビュー	×	隠し型およびメンバを非表示にします。
コマンド ウィンドウ	×	
[ダイナミック ヘルプ] ウィンドウ	×	F1 キーを押しても表示されません。
[イミディエイト ウィンドウ]	○ (デバッグ開始時)	
オブジェクト ブラウザ	×	既定では、以下が表示されます。 • 名前空間 • パブリック メンバ • 継承されたメンバ
[出力] ウィンドウ	○ (ビルドの開始時のみ)	
[プロパティ] ウィンドウ	○	
ソリューション エクスプローラ	○	
サーバー エクスプローラ/データベース エクスプローラ	×	データ接続と使用可能なシステム サービスを表示します。
スタート ページ	○	IDE を初めて開始したときに表示されます。カスタマイズが可能です。
タスク一覧 (Visual Studio)	○ (コンパイル エラーまたは警告がある場合)	
ツールボックス	○	コントロールとコンポーネントをアルファベット順に表示します。

また、[Visual Basic 開発設定] を選択すると、次の要素で特定の動作が有効になります。

ダイアログ ボックス

機能	動作
[新しいプロジェクト] ダイアログ ボックス	コンピュータに他のプラットフォームがインストールされている場合、操作対象のプラットフォームを指定するコンボ ボックスを有効にします。
[オプション] ダイアログ ボックス (Visual Studio)	簡略化された [オプション] ページが有効になります。[すべての設定を表示] チェック ボックスをオンにすると、すべてのオプションの一覧が表示されます。

キーボード

機能	動作

Visual Basic 6.0 の既定のショートカット キー	標準の Visual Basic 6 キーボード割り当てをサポートします。

IDE のその他の要素

機能	動作
ツール ヒント	• マウス カーソルを上に置くと、すべてのメニュー項目およびツール バー ボタンが表示されます。
Visual Basic の IntelliSense コード スニペット	アプリケーションにそのまま挿入できるコード スニペットのライブラリを含みます。
Visual Basic 固有の IntelliSense	• Visual Basic 開発者用に生産性が一部強化された、すべての IntelliSense 機能を提供します。

参照

処理手順

方法 : [ダイナミック ヘルプ] をカスタマイズする

方法 : 選択した設定を変更する

方法 : チームの設定を指定する

関連項目

[設定のインポートとエクスポート] ([オプション] ダイアログ ボックス - [環境])

その他の技術情報

設定のインポートとエクスポート ウィザード

Visual Basic の IntelliSense コード スニペット

Visual Basic には、IntelliSense コード スニペットという、数百種類のコードから成るコード ライブラリが含まれており、作成するアプリケーションに挿入できるようになっています。各スニペットは、カスタム例外の作成、電子メール メッセージの送信、円の描画など、完全なプログラミング タスクを実行します。スニペットをソース コードに挿入するには、マウスを数回クリックするだけです。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。また、業務上のニーズに合った独自のスニペットを作成し、ライブラリに追加して、必要に応じて使用することもできます。詳細については、「IntelliSense コード スニペットの作成と使用」を参照してください。

Visual Basic ユーザー用の生産性向上ツール

これらのスニペットを使用すると、コード サンプルの検索に要する時間の短縮、未知の機能の使い方を身に付けるまでに要する時間の短縮、およびコードの再利用の促進が実現され、生産性が向上します。

このコード ライブラリは、次の機能をサポートします。

- いずれか 1 つのスニペットをコード エディタに挿入します。
- プロジェクトで再利用できる新しいタスクを作成します。
- 新しいタスクを作成し、ワークグループや同僚と共有します。
- タスクを編集します。
- サードパーティから追加的なタスクをダウンロードします。

これらのコード ブロックは、Visual Studio 全体で利用できます。

- コード エディタのショートカット メニューから追加できます。
- Windows エクスプローラからソース コード ファイルに XML コード ファイルをドラッグできます。

参照

処理手順

方法 : 既存のスニペットを変更する
方法 : スニペットを他の開発者と共有する
方法 : コードに追加したスニペットを削除する
方法 : ショートカット名をスニペットに割り当てる
スニペットのトラブルシューティング
方法 : コード スニペットを管理する
方法 : コード スニペットをオンラインで検索する

概念

IntelliSense コード スニペットの最適な使用方法
スニペットの使用に関するセキュリティ上の考慮事項

方法：コードにスニペットを挿入する (Visual Basic)

IntelliSense コード スニペットの挿入は、コード エディタのナビゲーション ツールを使用するか、または Windows エクスプローラからファイルをドラッグすることにより行うことができます。挿入するスニペットの名前がわかっている場合には、ショートカットを入力した後に Tab キーを押すことで挿入できます。目的に合ったスニペットを参照する必要がある場合には、スニペット ピッカーを使用できます。スニペット ピッカーでは、スニペットがカテゴリごとに一覧表示されており、その中から選択できます。

コード スニペットを挿入したら、アプリケーションに合わせて動作するように変更を加えます。たとえば、スニペットによっては、アプリケーションに新しいメソッドを挿入するものがあります。この新しいメソッドを既存のメソッドの中に挿入した場合、コードの余分な行を削除して、コンパイラ エラーを修復する必要があります。また、スニペット内で使用されている、ユーザー向けのメッセージやファイル名も、場合によっては変更が必要です。詳細については、「方法 : コードに挿入したスニペット コードをカスタマイズする」を参照してください。

スニペットをコード エディタで参照して挿入するには

1. コードを挿入する位置でコード エディタを右クリックします。
2. ショートカット メニューの [スニペットの挿入] をクリックします。IntelliSense コード スニペット ピッカーが表示されます。

   手順 1 および 2 の代わりに、コード エディタで、疑問符を入力した後に Tab キーを押しても、コード スニペット ピッカーが表示されます。
3. 目的のタスクに移動し、それをクリックします。コードにスニペット コードが挿入されます。

   コードにスニペットを追加したら、その一部にカスタマイズが必要な場合があります。たとえば、変数名をより適切な名前に置き換えるなどです。この処理にはエディタが役立ちます。詳細については、「方法 : コードに挿入したスニペット コードをカスタマイズする」を参照してください。

   ピッカーに表示されるスニペットは、特定のファイル位置のサブフォルダに、ファイルとして格納されています。ファイル位置のチェックおよび変更は、コード スニペット マネージャ を使用して行うことができます。各スニペットはそれぞれ 1 つのファイルに対応しており、ファイル名の拡張子は .snippet です。自作のスニペットや、他の入手元から手に入れたスニペットは、ファイル システムの任意の場所に格納できます。ただし、所定のフォルダのいずれかにないと、ショートカット メニューに表示されません。詳細については、スニペット マネージャを参照してください。

ショートカットがわかっているスニペットを挿入するには

- コード エディタで、スニペットのショートカットを入力した後に Tab キーを押します。スニペットが挿入されます。

  または
- コード エディタで、スニペットのショートカット名の先頭の数文字を入力した後に、疑問符および Tab キーを押します。ピッカーが表示され、入力した文字でショートカットが始まるスニペットの一覧が示されます。リスト内を移動すると、選択したショートカットのタイトルがツールヒントとして表示されます。

コード スニペットのショートカット名を見つける方法には次の 2 つがあります。

- コード スニペット マネージャ を使用して、すべてのスニペットのショートカットを参照します。
- コード スニペット ピッカーで、ツールヒントによる説明の中にショートカットが表示されます。

Windows エクスプローラを使用してコードにスニペットを挿入するには

1. Windows エクスプローラを開きます。
2. Windows エクスプローラで、挿入するファイルが格納されているフォルダに移動します。Visual Studio 2005 を既定の位置にインストールした場合は、スニペットは C:\Program Files\Microsoft Visual Studio 8\Vb に格納されています。Visual Basic 2005 Express Edition を既定の位置にインストールした場合は、スニペットは C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\VBExpress\Snippets\1033 に格納されています。
3. Windows エクスプローラから、コード内の挿入位置にファイルをドラッグします。

参照

処理手順

方法 : コード スニペットをオンラインで検索する

関連項目

IntelliSense コード スニペットの作成と使用

概念

方法：コードに挿入したスニペット コードをカスタマイズする

スニペットをコードに追加すると、挿入されたスニペット コードでは、1 つまたは複数の置換ポイントが強調表示されています。各置換ポイントを変更するかどうかは自由です。マウス ポインタを置換ポイントの上で止めると、コードに対して必要な変更を説明するツールヒントが表示されます。コード エディタでは、ソース コード ファイルを閉じるまでは、スニペットは周囲のコードとは別個のユニットとして認識されます。したがって、置換ポイントは、ソース ファイルを閉じるまでは強調表示されたままです。

挿入されたコードをアプリケーション用にカスタマイズするには

1. 次の置換ポイントへ移動するには、Tab キーを押します。前の置換ポイントへ移動するには、Shift + Tab キーを押します。
2. Ctrl + Space キーを押して、IntelliSense を起動します。
3. リストから項目を選択するか、または目的の置換する内容を入力します。
4. コードにコンパイラ エラーが残っている場合は、ソース コードに他の変更を加えることが必要な場合があります。たとえば、`TreeView1` という名前の TreeView コントロールでノードを検索するというタスクの場合は、`TreeView1` という名前の **TreeView** コントロールをフォームに追加することが必要な場合があります。

参照

処理手順

方法 : コードにスニペットを挿入する (Visual Basic)

関連項目

IntelliSense コード スニペットの作成と使用

概念

Visual Basic の IntelliSense コード スニペット
スニペットの使用に関するセキュリティ上の考慮事項
IntelliSense コード スニペットの最適な使用方法

スニペットの使用に関するセキュリティ上の考慮事項

Visual Studio によってインストールされる IntelliSense コード スニペットは、それ自体にはセキュリティ上の危険性はありません。使用場所と修正方法によっては、アプリケーションにセキュリティ リスクをもたらすことになり得ます。インターネットからダウンロードしたスニペットは、その他のダウンロード コンテンツと同様に扱う必要があります。

インターネットからダウンロードしたスニペット

インターネットからスニペット ファイルをダウンロードするときには、次の点に注意する必要があります。

- ファイル名の拡張子が .snippet でも、その中身がプレーンテキストの XML とは限りません。安全のために、ダウンロードは信頼できるサイトから行い、また最新のウイルス ソフトウェアを使用してください。
- スニペット ファイルに含まれるヘルプ URLにより、悪意のあるスクリプト ファイルが実行されたり、攻撃を行う Web サイトが表示されたりする可能性があります。ヘルプ URL は、スニペットを挿入するときのコード エディタには表示されませんが、XML エディタやその他のエディタ (メモ帳など) でスニペット ファイルを編集するときには参照できます。
- コード自体の中に、実行時にシステムに危害をもたらし得るコードが含まれている可能性があります。実行前にソースをよく読んでください。コードの一部が、折りたたまれた #Region ディレクティブ セクションに含まれている場合があります。該当するセクションを展開し、コードを読んでください。
- スニペットに参照が含まれている場合があります。これらの参照は、予告なしにプロジェクトに追加され、システム上の任意の場所から読み込まれます。これらの参照は、スニペットのダウンロード場所からコンピュータにダウンロードされたものである場合があります。そして、悪意のあるコードを実行するメソッドが参照内に含まれていて、スニペットがそうしたメソッドを呼び出すことがあります。こうした攻撃を防ぐためには、スニペットを挿入する前にスニペット ファイルをよく読むようにし、またダウンロードは信頼できるサイトから行うようにします。

すべてのスニペットのセキュリティ

スニペットがどの程度安全かは、ソース コード内での使用場所と、コード内に組み込んだ後の修正方法に応じて左右されます。次の一覧は、考慮が必要な部分をいくつか示したものです。

- ファイル アクセスとデータベース アクセス
- コード アクセス セキュリティ
- リソースの保護 (イベント ログ、レジストリなど)
- 秘密のデータの格納
- 入力の検査
- スクリプトへのデータの引き渡し

詳細については、「アプリケーションの保護」を参照してください。

参照

処理手順

方法 : コード スニペットをオンラインで検索する

関連項目

IntelliSense コード スニペットの作成と使用

概念

アプリケーションの保護

IntelliSense コード スニペットの最適な使用方法

IntelliSense コード スニペットの各コードは、タスクを実行するための基本的な方法のみを示しています。ほとんどのアプリケーションでは、アプリケーションに合うようにコードを変更する必要があります。このトピックでは、コードに対して加える必要のあるいくつかの一般的な変更点について説明します。

例外処理

通常、コードに **Try...Catch** ブロックが含まれている場合は、コードですべての例外をキャッチして再スローしています。これは、必ずしもプロジェクトに最適な方法でない可能性もあります。各例外に対する応答の方法はいくつかあります。考えられる処理を以下に示します。

- 必要な処理に対応するコードを、各 **Catch** ブロックに追加します。
- 何もしないよう、**Catch** ブロックに含まれているコードを削除します。こうすると、アプリケーションはそのエラーを無視します。この方法をとった場合、アプリケーションがエラーから適切に回復する可能性は低くなります。
- 状況の修復を試みるために、ユーザーに追加的な入力を求めます。
- 例外がスローされる前にアプリケーション データに加えられた変更を、**Try** ブロックの実行の間に元に戻します。
- 例外をスローして呼び出し元のメソッドに戻します。
- 当該のアプリケーション用に定義した例外をスローします。
- 当該の例外に対する **Catch** ステートメントを削除し、呼び出し元のメソッドに処理させます。この方法は、実行するタスクに関係ない例外の場合に特に適しています。
- **Catch** ブロックをコードに追加し、処理の対象とする特定の型の例外をキャッチします。
- **Finally** ブロックを追加して、**Try** ブロックおよび **Catch** ブロックの後で実行するコードを追加します。

文字列の置換

コードに文字列値が含まれている場合、通常は "c:\filename.txt" などの固有の文字列で指定されています。このようにハードコーディングされた文字列は、デモンストレーションのためには有用ですが、アプリケーションで使用する正しい文字列ではないのが普通です。アプリケーション内で文字列を使用している可能性があるのは、次のような部分です。

- 文字列変数。
- 文字列を返すメソッドまたはプロパティ (**InputBox** 関数など)。
- Windows フォーム コントロールの文字列 (TextBox、ComboBox など)。

ファイルの位置

コードの大半のファイル名は、My Documents または c:\ にあるものとして示されています。ファイルの位置を変更するときには、次のような点を考慮する必要があります。

- アクセス可能な場所を見つけます。ユーザーによっては、コンピュータの \Program Files フォルダにアクセスできない場合があるため、アプリケーション ファイルと一緒にファイルを格納してもうまくいかないことがあります。
- 安全な場所を見つけます。ファイルをルート フォルダ (c:\) に格納するのは安全ではありません。アプリケーション データについては、\Application Data フォルダの使用をお勧めします。個々のユーザー データについては、アプリケーションで \My Documents フォルダに各ユーザー用のファイルを作成する方法が使用できます。
- 有効なファイル名を使用します。OpenFileDialog コンポーネント (Windows フォーム) および SaveFileDialog コンポーネント (Windows フォーム) を使用すると、無効なファイル名の可能性を減らせます。ユーザーがファイルを選択してから、コードでそのファイルを操作するまでの間に、ファイルが削除される可能性がある点に注意してください。また、ユーザーがそのファイルへの書き込みのアクセス許可を持たない可能性もあります。

コントロールとコンポーネント

コードで参照されているコントロール名およびコンポーネント名では、既定のデザイナ名 (`Button1` や `TreeView1` など) が使用されているのが普通です。この場合、示されているコントロールの型は明確になりますが、アプリケーションで使用する名前とは異なることがあります。

コードの欠落

スニペットによっては、**Try...Catch** ブロックなど、空の言語構造のみが示されている場合があります。また、**IsValid** などの検証変数が既定で **True** に設定されていることもあります。提供されているコードはコンパイルおよび実行できますが、何も機能を持ちません。こうした例では、タスクに応じたコードを追加する必要があります。

セキュリティ

スニペットがどの程度安全かは、ソース コード内での使用場所と、コード内に組み込んだ後の修正方法に応じて左右されます。次の一覧は、考慮が必要な部分をいくつか示したものです。

- ファイル アクセスとデータベース アクセス
- コード アクセス セキュリティ
- リソースの保護 (イベント ログ、レジストリなど)
- 秘密のデータの格納
- 入力の検査
- スクリプトへのデータの引き渡し

詳細については、「スニペットの使用に関するセキュリティ上の考慮事項」および「アプリケーションの保護」を参照してください。

参照

処理手順

方法 : コードにスニペットを挿入する (Visual Basic)

関連項目

IntelliSense コード スニペットの作成と使用

概念

Visual Basic の IntelliSense コード スニペット
スニペットの使用に関するセキュリティ上の考慮事項
アプリケーションの保護

スニペットのトラブルシューティング

IntelliSense コード スニペットに伴う問題は、通常は 2 つの問題が原因で発生します。1 つはスニペット ファイルの破損、もう 1 つはスニペット ファイルの不適切な内容です。

一般的な問題

Windows エクスプローラから Visual Studio ソース ファイルにスニペットをドラッグできない

- スニペット ファイルの XML が破損している可能性があります。Visual Studio の [XML エディタ] を使用すると、XML の構造の問題を特定できます。
- スニペット ファイルがスニペットのスキーマにのっとっていない可能性があります。Visual Studio の [XML エディタ] を使用すると、XML の構造の問題を特定できます。

コードに、強調表示されていないコンパイラ エラーがある

- プロジェクト参照が不足している可能性があります。スニペットについてのドキュメントを調べてください。参照がコンピュータ上にない場合は、インストールする必要があります。通常は、スニペットを挿入すると、必要な参照がプロジェクトに追加されます。スニペットに参照情報がない場合は、スニペットの作成者に対して、エラーとして報告できます。
- 変数が未定義の可能性があります。通常は、スニペット内の未定義の変数は強調表示されます。そうなっていない場合は、スニペットの作成者に対して、エラーとして報告できます。

参照

関連項目

IntelliSense コード スニペットの作成と使用

方法：コード属性を編集する

Visual Basic のコード エディタでは、[プロパティ] ウィンドウを使用して、型およびメンバに対して属性を適用および編集できます。[プロパティ] ウィンドウに一覧表示されている属性を適用するには、属性を **True** に設定するか、または属性の **Apply** プロパティを **True** に設定します。適用した属性を編集するには、[プロパティ] ウィンドウに表示されるプロパティを編集します。詳細については、「Visual Basic における属性」を参照してください。

新しい属性を適用するには

1. [プロパティ] ウィンドウで目的の属性を探します。
2. 目的の属性が一覧表示されていない場合は、コード内で属性を手動で適用する必要があります。詳細については、「属性の適用」を参照してください。
3. 次のいずれかを実行します。

   - 属性がプロパティを持つ場合は、**Apply** プロパティを **True** に設定します。

   または

   - 属性がプロパティを持たない場合は、属性を **True** に設定します。

参照

関連項目

[プロパティ] ウィンドウ

概念

Visual Basic における属性
属性の一般的な使用方法
Visual Basic で使用される属性
属性の適用

[名前の変更] ダイアログ ボックス (Visual Basic)

Visual Basic の組み込み [名前の変更] ダイアログ ボックスを使用すると、フィールド、ローカル変数、メソッド、名前空間、プロパティ、型などのシンボルを表す、コードに含まれている識別子の名前を変更できます。[名前の変更] ダイアログ ボックスを開くには、要素の宣言を右クリックします。

[新しい名前]

コード要素の新しい名前を指定します。

[場所]

名前変更の操作を実行するときに検索する名前空間を指定します。

名前変更の操作

[名前の変更] ダイアログ ボックスで実行される名前変更の操作は、名前を変更する要素の型によって多少異なります。

要素の型	名前変更の操作
クラス	当該クラスのすべての宣言とすべての使用箇所が、新しい名前に変更されます。部分クラスについては、名前変更の操作はすべての部分に反映されます。
フィールド	当該フィールドの宣言と使用箇所が、新しい名前に変更されます。
ローカル変数	当該変数の宣言と使用箇所が、新しい名前に変更されます。
メソッド	当該メソッドの名前と、当該メソッドへのすべての参照が、新しい名前に変更されます。
名前空間	宣言、すべての **Imports** ステートメント、およびすべての完全修飾名において、当該名前空間の名前が新しい名前に変更されます。
プロパティ	当該プロパティの宣言と使用箇所が、新しい名前に変更されます。

リファクタリング

Visual Basic と Developer Express Inc. との提携により、開発者はより綿密なリファクタリングを行うことができます。このツールをダウンロードする方法の詳細については、MSDN Visual Basic Developer Center の「**Refactor!** (http://msdn.microsoft.com/vbasic/downloads/tools/refactor/)」を参照してください。「**Refactor!**」では、Reorder Parameters、Extract Method、Encapsulate Field、Create Overload などの操作を含む 15 以上のリファクタリング機能がサポートされています。

参照

処理手順

方法 : 識別子の名前を変更する

その他の技術情報

Visual Basic Editor の使用

方法：識別子の名前を変更する

Visual Studio 統合開発環境 (IDE) では、コードを分析し、スコープ情報を使用して、識別子の名前を簡単に変更できます。これを使用すると、検索と置換に比べて、名前の変更を適切に制御できます。

たとえば、`Test1` と `Test2` という 2 つのサブルーチンを定義しているクラスがあり、各サブルーチンで `Value` という名前の変数を定義しているとします。`Test1` の `Value` 変数の名前を `Counter` に変更すると、`Test1` のコードの中で `Value` を参照しているすべての部分が `Counter.` を参照するように更新されます。一方、`Test2` のコードは変更されません。

識別子の任意のインスタンスから名前を変更するには

1. 識別子を右クリックします。
2. ショートカット メニューの [名前の変更] をクリックし、[名前の変更] ダイアログ ボックスを開きます。
3. [新しい名前] ボックスの名前を、識別子の新しい名前に変更します。
4. [OK] をクリックして、その識別子を参照するすべての名前を変更します。

識別子の宣言から名前を変更するには

1. 識別子名を宣言している部分を選択します。
2. 識別子の名前を変更します。

   名前を変更すると、名前の右下にシンボルが表示されます。
3. シンボルの上にマウス ポインタを置き、スマート タグをアクティブにします。
4. スマート タグ グリフをクリックし、パネルを開きます。
5. [名前の変更] パネル項目を選択し、同じ識別子に対するその他の参照の名前を変更します。

参照

関連項目

[名前の変更] ダイアログ ボックス (Visual Basic)

その他の技術情報

Visual Basic Editor の使用

Visual Basic 6.0 の既定のショートカット キー

次に示すヘルプ ページでは、Visual Basic 6.0 のキーボード スキームで使用できる既定のショートカット キーを示します。既定の組み合わせを変更する方法の詳細については、「方法 : ショートカット キーの組み合わせを操作する」を参照してください。

このセクションの内容

グローバル ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

一般的なショートカット キーと IDE 内のさまざまな場所で使用できるショートカット キーの一覧を示します。

統合ヘルプのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

ヘルプ トピックを表示および参照しているときに使用できるショートカット キーの一覧を示します。

プロジェクト ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

プロジェクトへの新規項目の追加時、プロジェクトのビルド時、およびファイルやプロジェクトを開くときに使用できるショートカット キーの一覧を示します。

HTML デザイナのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

HTML デザイナの [デザイン] ビューおよび [HTML] ビューで操作しているときに使用できるショートカット キーの一覧を示します。

イメージ エディタのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

イメージ エディタで操作しているときに使用できるショートカット キーの一覧を示します。

デバッグのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

デバッガを使っているときに使用できるショートカット キーの一覧を示します。

検索と置換のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

[検索]、[置換]、[ファイル内の検索]、および [ファイル内の置換] の各ダイアログ ボックスで使用できるショートカット キーの一覧を示します。

Visual Database Tools のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

クエリ デザイナまたはデータベース デザイナを使っているときに使用できるショートカット キーの一覧を示します。

オブジェクト ブラウザのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

オブジェクト ブラウザで操作しているときに使用できるショートカット キーの一覧を示します。

コントロール操作のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

デザイン サーフェイス上のコントロールの移動やサイズ変更をするときに使用できるショートカット キーの一覧を示します。

テキスト移動のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

開いているドキュメントのコード内を移動するときに使用できるショートカット キーの一覧を示します。

テキスト選択のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

開いているドキュメントのテキストを選択するときに使用できるショートカット キーの一覧を示します。

テキスト操作のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

開いているドキュメントのテキストを移動、削除、または書式設定するときに使用できるショートカット キーの一覧を示します。

ウィンドウ管理のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

ツールやドキュメントのウィンドウの移動、クローズ、またはウィンドウ内を移動するときに使用できるショートカット キーの一覧を示します。

マクロ ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

マクロの作業をするときに使用できるショートカット キーの一覧を示します。

ツール ウィンドウのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

さまざまなツール ウィンドウを表示するショートカット キーの一覧を示します。

関連するセクション

方法 : ショートカット キーの組み合わせを操作する

現在のキーボード マップ スキームの確認方法、特定のコマンドのショートカット キーの確認方法、カスタム マップ スキームの作成方法、および新しいショートカット キーの作成方法について説明します。

ショートカット キー

Visual Studio 開発環境で使用できる定義済みのキーボード バインディング スキームの一覧を示します。

グローバル ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、統合開発環境 (IDE: Integrated Development Environment) 内のさまざまな場所で使用できます。

コマンド	ショートカット キー	説明
Edit.Copy	**Ctrl** + **C** Ctrl + Ins	現在選択されているアイテムの複製をシステムのクリップボードに配置します。
Edit.Cut	Ctrl + X Shift + Del	現在選択されているアイテムをシステムのクリップボードに移動します。
Edit.CycleClipboardRing	Ctrl + Shift + V Ctrl + Shift + Ins	ツールボックスの [クリップボード リング] タブのアイテムをファイル内のカーソル位置に貼り付け、貼り付けたアイテムを自動的に選択します。ショートカット キーを繰り返し押すと、クリップボードのアイテムを 1 つずつ確認できます。
Edit.Delete	Del	カーソルの右側にある文字を 1 文字削除します。
Edit.DeleteBackwards	BackSpace Shift + BackSpace	カーソルの左側にある文字を 1 文字削除します。
Edit.GotoNextLocation	F12	[タスク一覧] ウィンドウのタスクや、[検索結果] ウィンドウの検索の一致など、次のアイテムにカーソルを移動します。F12 キーを押すたびに、一覧の次のアイテムに移動します。
Edit.GotoPreviousLocation	**Shift** + **F12**	[タスク一覧] ウィンドウまたは [検索結果] ウィンドウでカーソルを前のアイテムに移動します。
Edit.ListMembers	**Ctrl** + **J**	コードの編集時に、ステートメント入力候補として現在のクラスのメンバー一覧を表示します。
Edit.ParameterInfo	**Ctrl** + **Shift** + **I**	現在の言語に基づいて、現在のパラメータに関するツール ヒントを表示します。
Edit.Paste	Ctrl + V Shift + Ins	カーソル位置に、クリップボードの内容を挿入します。
Edit.QuickInfo	**Ctrl** + **I**	カーソル位置の最も近くにある識別子の宣言全体をツール ヒントに表示します。
Edit.Redo	Ctrl + Shift + Z Shift + Alt + BackSpace	前回取り消した操作を再度実行します。
Edit.SelectionCancel	Esc	メニューまたはダイアログ ボックスを閉じ、実行中の操作をキャンセルします。または、現在のドキュメント ウィンドウにフォーカスを置きます。

Edit.Undo	Alt + BackSpace Ctrl + Z	最後に行った編集操作を元に戻します。
File.AddExistingItem	**Ctrl** + **D**	[既存項目の追加] ダイアログ ボックスを表示します。このダイアログ ボックスでは、現在のプロジェクトに既存のファイルを追加できます。
File.AddNewItem	**Ctrl** + **Shift** + **A**	[新しい項目の追加] ダイアログ ボックスを表示します。このダイアログ ボックスでは、現在のプロジェクトに新しいファイルを追加できます。
File.Exit	**Alt** + **Q**	統合開発環境を終了します。
File.Print	**Ctrl** + **P**	[印刷] ダイアログ ボックスを表示します。このダイアログ ボックスでは、プリンタの設定を選択できます。
File.SaveAll	**Ctrl** + **Shift** + **S**	現在のソリューションのドキュメントと外部ファイル プロジェクトのファイルをすべて保存します。
File.SaveSelectedItems	**Ctrl** + **S**	選択したアイテムを現在のプロジェクトに保存します。
Tools.GoToCommandLine	**Ctrl** + **/**	[標準] ツール バーの [検索] ボックスにキャレットを配置します。
View.NavigateBackward	**Ctrl** + **Shift** + **F2**	移動履歴にある前のドキュメントまたはウィンドウに戻ります。
View.NavigateForward	**Ctrl** + **Shift** + マイナス記号 (-)	移動履歴にある次のドキュメントまたはウィンドウに進みます。
View.WebNavigateBack	**Alt** + ←	履歴の表示で前のページを表示します。統合された Web ブラウザ以外では使用できません。
View.WebNavigateForward	**Alt** + →	履歴の表示で次のページを表示します。統合された Web ブラウザ以外では使用できません。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

プロジェクト ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

統合ヘルプのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

統合ヘルプのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、ヘルプ トピックを表示したり、ヘルプ トピックの間を移動したりするときに使用できます。

コマンド名	ショートカット キー	説明
Help.Contents	**Ctrl** + **Alt** + **F1**	MSDN に含まれるドキュメントの [目次] ウィンドウを表示します。
Help.DynamicHelp	**Ctrl** + **F1**	[ダイナミック ヘルプ] ウィンドウを表示します。[ダイナミック ヘルプ] ウィンドウには、製品の現在フォーカスがあるアイテムに応じたトピックが表示されます。
Help.F1Help	F1	現在選択されているユーザー インターフェイスに対応するヘルプ トピックを表示します。
Help.Index	**Ctrl** + **Alt** + **F2**	MSDN に含まれるドキュメントの [キーワード] ウィンドウを表示します。
Help.Nexttopic	**Alt** + ↓	目次内の次のトピックを表示します。[ヘルプ] (Web ブラウザ) のウィンドウでだけ使用できます。
Help.Previous topic	**Alt** + ↑	目次内の前のトピックを表示します。[ヘルプ] (Web ブラウザ) のウィンドウでだけ使用できます。
Help.Search	**Ctrl** + **Alt** + **F3**	[検索] ウィンドウを表示します。このウィンドウで、MSDN に含まれるドキュメントの単語または句を検索します。
Help.WindowHelp	**Shift** + **F1**	現在選択されているユーザー インターフェイスに対応するヘルプ トピックを表示します。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

ウィンドウ管理のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

プロジェクト ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、プロジェクトへの新規アイテムの追加、プロジェクトのビルド、およびファイルやプロジェクトをオープンするときに使用できます。

コマンド名	ショートカット キー	説明
Build.BuildSolution	**Ctrl** + **Shift** + **B**	現在のソリューション構成を使ってソリューション内のすべてのプロジェクトをビルドします。
Build.Compile	**Ctrl** + **F7**	選択したファイルのマシン語コード、リンカ ディレクティブ、セクション、外部参照、関数名またはデータ名を含むオブジェクト ファイルを作成します。
File.NewFile	**Ctrl** + **Shift** + **N**	[新しいファイル] ダイアログ ボックスを表示します。このダイアログ ボックスでは、新しいファイルを選択して現在のプロジェクトに追加できます。
File.NewProject	**Ctrl** + **N**	[新しいプロジェクト] ダイアログ ボックスを表示します。このダイアログ ボックスでは、プロジェクトを作成して現在のソリューションに追加できます。
File.OpenFile	**Ctrl** + **Shift** + **O**	[ファイルを開く] ダイアログ ボックスを表示します。このダイアログ ボックスでは、既存のファイルを選択して開くことができます。
File.OpenProject	**Ctrl** + **O**	[プロジェクトを開く] ダイアログ ボックスを表示します。このダイアログ ボックスでは、既存のプロジェクトをソリューションに追加できます。
File.AddExistingItem	**Ctrl** + **D**	[既存項目の追加] ダイアログ ボックスを表示します。このダイアログ ボックスでは、現在のプロジェクトに既存のファイルを追加できます。
File.AddNewItem	**Ctrl** + **Shift** + **A**	[新しい項目の追加] ダイアログ ボックスを表示します。このダイアログ ボックスでは、現在のプロジェクトに新しいファイルを追加できます。
Project.RunSelection	**Ctrl** + **Q**	現在の選択項目を実行します。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

グローバル ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

統合ヘルプのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

ウィンドウ管理のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

概念

ソリューション、プロジェクト、および項目の概要

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

HTML デザイナのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、HTML デザイナで編集作業を行っているときにだけ使用できます。デザイナの特定のビューでしか使用できないショートカット キーもあります。

コマンド	ショートカット キー	説明
Edit.CharTranspose	**Ctrl** + **T**	カーソルの両側にある文字を入れ替えます。たとえば、AC|BD の場合は、AB|CD になります。HTML デザイナの HTML ビューでだけ使用できます。
Edit.ClearAllBookmarks	**Ctrl** + **K**、**Ctrl** + **L**	現在のドキュメントで、名前のないブックマークをすべて削除します。HTML デザイナの HTML ビューでだけ使用できます。
Edit.CompleteWord	**Ctrl** + **Space**	現在の言語に基づいて、単語の入力候補を表示します。HTML デザイナの HTML ビューでだけ使用できます。
Edit.MakeLowercase	**Ctrl** + **U**	選択したテキストを小文字に変更します。HTML デザイナの HTML ビューでだけ使用できます。
Edit.MakeUppercase	**Ctrl** + **Shift** + **U**	選択したテキストを大文字に変更します。HTML デザイナの HTML ビューでだけ使用できます。
Edit.NextBookmark	**Ctrl** + **K**、**Ctrl** + **N**	ドキュメント内の次のブックマークに移動します。HTML デザイナの HTML ビューでだけ使用できます。
Edit.WordDeleteToEnd	**Ctrl** + **Del** Ctrl + Shift + Backspace	カーソル位置の右にある単語を削除します。HTML デザイナの HTML ビューでだけ使用できます。
Edit.WordDeleteToStart	**Ctrl** + **BackSpace**	カーソル位置の左にある単語を削除します。HTML デザイナの HTML ビューでだけ使用できます。
Format.Bold	**Ctrl** + **B**	選択したテキストを太字と標準の間で切り替えます。HTML デザイナのデザイン ビューでだけ使用できます。
Format.DecreaseIndent	**Ctrl** + **Shift** + **T**	選択した段落のインデントを 1 単位解除します。HTML デザイナのデザイン ビューでだけ使用できます。
Format.IncreaseIndent	**Ctrl** + **T**	選択した段落にインデントを 1 単位ずつ設定します。HTML デザイナのデザイン ビューでだけ使用できます。
Format.Italic	**Ctrl** + **I**	選択したテキストを斜体と標準の間で切り替えます。HTML デザイナのデザイン ビューでだけ使用できます。
Format.LockElement	**Ctrl** + **Shift** + **K**	絶対座標で配置された要素が誤って移動されないようにします。HTML デザイナのデザイン ビューでだけ使用できます。
Format.ShowGrid	**Ctrl** + **G**	グリッドの表示と非表示を切り替えます。HTML デザイナのデザイン ビューでだけ使用できます。

Format.SnaptoGrid	**Ctrl** + **Shift** + **G**	非表示のグリッドを使用して要素を整列します。グリッド間隔は、[オプション] ダイアログ ボックスの [HTML デザイナ] オプションの表示ペインで設定します。グリッドの変更は、次にドキュメントを開いたときに反映されます。HTML デザイナのデザイン ビューでだけ使用できます。
Format.Underline	**Ctrl** + **U**	選択したテキストを下線付きと標準の間で切り替えます。HTML デザイナのデザイン ビューでだけ使用できます。
Format.InsertBookmark	**Ctrl** + **Shift** + **L**	[ブックマーク] ダイアログ ボックスを表示します。このダイアログ ボックスでは、現在のドキュメント内にジャンプ先を作成できます。HTML デザイナのデザイン ビューと HTML ビューで使用できます。
Format.ConverttoHyperlink	**Ctrl** + **L**	テキストが選択されている場合は、[ハイパー リンク] ダイアログ ボックスを表示します。HTML デザイナのデザイン ビューと HTML ビューで使用できます。
Table.InsertRowAbove	**Ctrl** + **Alt** + ↑	テーブルの現在の行の上に行を 1 行追加します。HTML デザイナのデザイン ビューでだけ使用できます。
Table.InsertRowBelow	**Ctrl** + **Alt** + ↓	テーブルの現在の行の下に行を 1 行追加します。HTML デザイナのデザイン ビューでだけ使用できます。
Table.InsertColumnstotheLeft	**Ctrl** + **Alt** + ←	テーブルの現在の列の左に列を 1 列追加します。HTML デザイナのデザイン ビューでだけ使用できます。
Table.InsertColumnstotheRight	**Ctrl** + **Alt** + →	テーブルの現在の列の右に列を 1 列追加します。HTML デザイナのデザイン ビューでだけ使用できます。
View.Details	**Ctrl** + **Shift** + **Q**	コメント、スクリプト、絶対座標で配置された要素のアンカなど、ビジュアルな表示を持たない HTML 要素のシグナル アイコンを表示します。HTML デザイナのデザイン ビューでだけ使用できます。
View.NextView	**Ctrl** + **PageDown**	デザイン ビューと HTML ビューを切り替えます。HTML デザイナのデザイン ビューでだけ使用できます。
View.VisibleBorders	**Ctrl** + **Q**	BORDER 属性をサポートし、属性値がゼロに設定された HTML 要素を囲む、1 ピクセルの境界線を表示します。これらの要素には、テーブル、テーブルのセル、仕切りなどがあります。HTML デザイナのデザイン ビューでだけ使用できます。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

HTML デザイナ

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

イメージ エディタのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、イメージ エディタで使用できます。

コマンド	ショートカット キー	説明
Image.DrawOpaque	**Ctrl** + **J**	現在の選択項目を不透明または透明にします。イメージ エディタでだけ使用できます。
Image.FlipHorizontal	**Ctrl** + **H**	イメージを水平軸上で左右に反転させます。イメージ エディタでだけ使用できます。
Image.FlipVertical	**Shift** + **Alt** + **H**	イメージを垂直軸上で上下に反転させます。イメージ エディタでだけ使用できます。
Image.NewImageType	Ins	[アイコン イメージ タイプの新規作成] ダイアログ ボックスを表示します。このダイアログ ボックスでは、作成する新規イメージの種類を選択します。イメージ エディタでだけ使用できます。
Image.Rotate90Degrees	**Ctrl** + **Shift** + **H**	イメージを時計回りに 90 度回転させます。イメージ エディタでだけ使用できます。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

イメージ エディタ

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

デバッグのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、コードのデバッグ時に使用できます。

コマンド	ショートカット キー	説明
Debug.ApplyCodeChanges	**Alt** + **F10**	デバッガを停止させずにコードに対する変更を適用します。
Debug.Autos	**Ctrl** + **Alt** + **V**、**A**	[自動変数] ウィンドウを表示して、現在のプロシージャで現在実行している行のスコープに現在存在する変数の値を表示します。
Debug.Break All	**Ctrl** + **Break**	デバッグ セッションのすべてのプロセスの実行を一時的に停止します。実行モードでだけ使用できます。
Debug.Breakpoints	**Ctrl** + **Alt** + **B**	[ブレークポイント] ダイアログ ボックスを表示します。このダイアログ ボックスでは、ブレークポイントの追加と変更ができます。
Debug.CallStack	**Ctrl** + **L**	[呼び出し履歴] ウィンドウに、すべてのアクティブ プロシージャの一覧、または現在の実行スレッドのスタック フレームの一覧を表示します。実行モードでだけ使用できます。
Debug.Clear AllBreakpoints	**Ctrl** + **Shift** + **F9**	プロジェクトのブレークポイントをすべてクリアします。
Debug.Disassembly	**Ctrl** + **Alt** + **D**	[逆アセンブリ] ダイアログ ボックスを表示します。
Debug.Exceptions	**Ctrl** + **Alt** + **E**	[例外] ダイアログ ボックスを表示します。
Debug.Immediate	**Ctrl** + **G**	[イミディエイト] ウィンドウを表示します。このウィンドウでは、式を評価し、各コマンドを実行できます。
Debug.Locals	**Ctrl** + **Alt** + **V**、**L**	[ローカル] ウィンドウを表示して、現在のスタック フレームの各プロシージャの変数およびその値を表示します。
Debug.Memory1	Ctrl + Alt + M、1	[メモリ 1] ウィンドウを表示します。このウィンドウには、[ウォッチ] ウィンドウや [変数] ウィンドウでは正しく表示されない大きいバッファ、文字列、その他のデータが表示されます。
Debug.Memory2	Ctrl + Alt + M、2	[メモリ 2] ウィンドウを表示します。このウィンドウには、[ウォッチ] ウィンドウや [変数] ウィンドウでは正しく表示されない大きいバッファ、文字列、その他のデータが表示されます。
Debug.Memory3	Ctrl + Alt + M、3	[メモリ 3] ウィンドウを表示します。このウィンドウには、[ウォッチ] ウィンドウや [変数] ウィンドウでは正しく表示されない大きいバッファ、文字列、その他のデータが表示されます。

Debug.Memory4	Ctrl + Alt + M、4	[メモリ 4] ウィンドウを表示します。このウィンドウには、[ウォッチ] ウィンドウや [変数] ウィンドウでは正しく表示されない大きいバッファ、文字列、その他のデータが表示されます。
Debug.Modules	**Ctrl** + **Alt** + **U**	[モジュール] ウィンドウを表示します。このウィンドウでは、プログラムで使用されている .dll ファイルまたは .exe ファイルを表示できます。
Debug.NewBreakpoint	**Ctrl** + **B**	現在のコード行にブレークポイントを挿入します。または現在のコード行のブレークポイントを解除します。
Debug.QuickWatch	**Shift** + **F9**	[クイック ウォッチ] ダイアログ ボックスに、選択した式の現在値を表示します。中断モードだけで使用できます。このコマンドを使用して、ウォッチ式を定義していない変数、プロパティ、またはその他の式の現在の値をチェックします。
Debug.Registers	**Ctrl** + **Alt** + **G**	[レジスタ] ウィンドウを表示します。このウィンドウには、ネイティブ コード アプリケーションのデバッグ用の内容が表示されます。
Debug.Restart	**Shift** + **F5**	デバッグ セッションを終了し、リビルドして、最初からもう一度アプリケーションを実行します。中断モードと実行モードで使用できます。
Debug.RunningDocuments	**Ctrl** + **Alt** + **N**	[実行中のドキュメント] ウィンドウを表示します。このウィンドウには、デバッグ処理中のドキュメント セットが表示されます。実行モードで使用できます。
Debug.RunToCursor	**Ctrl** + **F8**	中断モードでは、現在のステートメントから選択したステートメントへコードの実行を再開します。[実行中の行] を示すマージン インジケータが [マージン インジケータ] バーに表示されます。デザイン モードでは、デバッガを起動し、カーソル位置までコードを実行します。
Debug.SetNextStatement	**Ctrl** + **F9**	選択したコード行から実行されるように設定します。
Debug.ShowNextStatemnt	Alt + NumLock *	次に実行されるステートメントが強調表示されます。
Debug.Start	F5	自動的にデバッガがアタッチされ、[<プロジェクト名> プロパティ] ダイアログ ボックスで指定したスタートアップ フォームからアプリケーションが実行されます。中断モードの場合は、続行に変更します。
Debug.StartWithoutDebugging	**Ctrl** + **F5**	デバッガを起動せずにコードを実行します。
Debug.StepInto	F8	関数呼び出しの実行後、一度に 1 ステートメントずつコードを実行します。
Debug.StepOut	**Ctrl** + **Shift** + **F8**	現在の実行ポイントがある関数の、残りの行を実行します。
Debug.StepOver	**Shift** + **F8**	コードの次の行を実行しますが、関数呼び出しによる実行は行いません。
Debug.This	**Ctrl** + **Alt** + **V**、**T**	[this] ウィンドウを表示します。このウィンドウでは、現在のメソッドに関連付けられているオブジェクトのデータ メンバを表示できます。

Debug.Threads	**Ctrl** + **Alt** + **H**	[スレッド] ウィンドウに、現在のプロセスのすべてのスレッドと、その情報を表示します。
Debug.ToggleBreakpoint	F9	現在の行のブレークポイントを設定または削除します。
Debug.ToggleDisassembly	**Ctrl** + **F11**	現在のソース ファイルの逆アセンブリ情報を表示します。中断モードだけで使用できます。
Tools.DebugProcesses	**Ctrl** + **Alt** + **P**	[プロセス] ダイアログ ボックスを表示します。このダイアログ ボックスを使用すると、単一のソリューション内で同時に複数のプログラムをデバッグできます。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

ツール ウィンドウのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

検索と置換のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下に、[検索]、[置換]、[ファイル内の検索]、および [ファイル内の置換] の各ダイアログ ボックスで使用できるショートカット キーの一覧を示します。

コマンド	ショートカット キー	説明
Edit.Find	**Ctrl** + **F**	[検索] ダイアログ ボックスを表示します。
Edit.FindNext	F3	前の検索テキストの、次の出現箇所を検索します。
Edit.FindNextSelected	**Ctrl** + **F3**	ドキュメントで現在選択しているテキストの、次の出現箇所を検索します。
Edit.FindPrevious	**Shift** + **F3**	検索テキストの、前の出現箇所を検索します。
Edit.FindPreviousSelected	**Ctrl** + **Shift** + **F3**	現在選択しているテキストまたはカレット位置にある単語の、前の出現箇所を検索します。
Edit.GoToFindCombo	**Ctrl** + **Shift** + **F**	[標準] ツール バーの [検索] にキャレットを配置します。
Edit.HiddenText	**Alt** + **F3**、**H**	[検索] ダイアログ ボックスの [非表示のテキストを検索] チェック ボックスをオンまたはオフにします。
Edit.IncrementalSearch	**Ctrl** + **Alt** + **I**	インクリメンタル検索を開始します。インクリメンタル検索の開始時に文字がまだ入力されていない場合は、前のパターンが再度呼び出されます。テキストが見つかった場合は、次の出現箇所を検索します。
Edit.MatchCase	**Alt** + **F3** + **C**	検索操作と 置換操作の [大文字と小文字を区別する] チェック ボックスをオンまたはオフにします。
Edit.RegularExpression	**Alt** + **F3** + **R**	検索操作と置換操作で特殊文字を使用するかどうかを指定する [正規表現] チェック ボックスをオンまたはオフにします。
Edit.Replace	**Ctrl** + **H**	[置換] ダイアログ ボックスを表示します。
Edit.ReplaceinFiles	**Ctrl** + **Shift** + **H**	[ファイル内の置換] ダイアログ ボックスを表示します。
Edit.ReverseIncrementalSearch	**Ctrl** + **Shift** + **Alt** + **I**	インクリメンタル検索の検索方向を変更して、ファイルの末尾から先頭に向けて検索します。
Edit.StopSearch	**Alt** + **F3** 、**S**	現在の [ファイル内の検索] 操作を中断します。
Edit.Up	**Alt** + **F3** 、**B**	検索操作と置換操作の [上へ検索] チェック ボックスをオンまたはオフにします。
Edit.WholeWord	**Alt** + **F3** 、**W**	検索操作と置換操作の [単語単位] チェック ボックスをオンまたはオフにします。

Edit.Wildcard	**Alt** + **F3**、**P**	検索操作と置換操作の [ワイルドカード] チェック ボックスをオンまたはオフにします。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

テキスト移動のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

テキスト選択のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

テキスト操作のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

Visual Database Tools のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、データベース デザイナまたはクエリ デザイナで使用できます。

コマンド	ショートカット キー	説明
Database.Run	**Ctrl** + **R**	現在選択されているデータベース オブジェクトを実行します。
Database.StepInto	**Ctrl** + **E**	現在アクティブなデータベース オブジェクトについてデバッグ モードに入ります。
Query.Run	**Ctrl** + **R**	クエリを実行します。Microsoft Visual Database Tools 以外では使用できません。
View.Diagram	**Ctrl** + **1**	クエリ デザイナの [ダイアグラム] ペインを表示します。クエリ デザイナとビュー デザイナでだけ使用できます。
View.Grid	**Ctrl** + **2**	クエリ デザイナの [グリッド] ペインを表示します。クエリ デザイナとビュー デザイナでだけ使用できます。
View.Results	**Ctrl** + **4**	クエリ デザイナの [結果] ペインを表示します。クエリ デザイナとビュー デザイナでだけ使用できます。
View.SQL	**Ctrl** + **3**	クエリ デザイナの [SQL] ペインを表示します。クエリ デザイナとビュー デザイナでだけ使用できます。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

グローバル ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

オブジェクト ブラウザのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、[オブジェクト ブラウザ] ウィンドウで使用できます。

コマンド	ショートカット キー	説明
Edit.FindSymbol	**Ctrl** + **Shift** + **Y** Ctrl + Shift + Alt + F12	[シンボルの検索] ダイアログ ボックスを表示します。
View.FindSymbolResults	**Ctrl** + **Alt** + **Y**	[シンボルの検索結果] ウィンドウを表示します。
View.ObjectBrowserBack	**Alt** + マイナス記号 (-)	オブジェクト ブラウザの選択履歴で前に選択したオブジェクトに戻ります。
View.ObjectBrowserForward	**Shift** + **Alt** + マイナス記号 (-)	オブジェクト ブラウザの選択履歴で次のオブジェクトに進みます。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

ツール ウィンドウのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

ウィンドウ管理のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

コントロール操作のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、デザイン サーフェイスのコントロールの移動、選択、およびサイズを変更するときに使用できます。

コマンド	ショートカット キー	説明
Edit.MoveControlDown	**Ctrl** + ↓	デザイン サーフェイスで、コントロールを下に「1」ずつ移動します。
Edit.MoveControlDownGrid	↓	デザイン サーフェイスで、コントロールを下に「8」ずつ移動します。
Edit.MoveControlLeft	**Ctrl** + ←	デザイン サーフェイス上のコントロールを左に 1 単位ずつ移動します。
Edit.MoveControlLeftGrid	←	デザイン サーフェイス上のコントロールを左に 8 単位ずつ移動します。
Edit.MoveControlRight	**Ctrl** + →	デザイン サーフェイスで、コントロールを右に「1」ずつ移動します。
Edit.MoveControlRightGrid	→	デザイン サーフェイス上のコントロールを右に 8 単位ずつ移動します。
Edit.MoveControlUp	**Ctrl** + ↑	デザイン サーフェイスで、コントロールを上に「1」ずつ移動します。
Edit.MoveControlUpGrid	↑	デザイン サーフェイスで、コントロールを上に「8」ずつ移動します。
Edit.SelectNextControl	Tab	デザイン サーフェイス上の次のコントロールを選択します。
Edit.SelectPreviousControl	**Shift** + **Tab**	デザイン サーフェイス上で前に選択されていたコントロールを選択します。
Edit.SizeControlDown	**Ctrl** + **Shift** + ↓	デザイン サーフェイスで、コントロールの高さを「1」ずつ増加します。
Edit.SizeControlDownGrid	**Shift** + ↓	デザイン サーフェイスで、コントロールの高さを「8」ずつ増加します。
Edit.SizeControlLeft	**Ctrl** + **Shift** + ←	デザイン サーフェイスで、コントロールの幅を「1」ずつ減少します。
Edit.SizeControlLeftGrid	**Shift** + ←	デザイン サーフェイスで、コントロールの幅を「8」ずつ減少します。
Edit.SizeControlRight	**Ctrl** + **Shift** + →	デザイン サーフェイスで、コントロールの幅を「1」ずつ増加します。
Edit.SizeControlRightGrid	**Shift** + →	デザイン サーフェイスで、コントロールの幅を「8」ずつ増加します。
Edit.SizeControlUp	**Ctrl** + **Shift** + ↑	デザイン サーフェイスで、コントロールの高さを「1」ずつ減少します。
Edit.SizeControlUpGrid	**Shift** + ↑	デザイン サーフェイスで、コントロールの高さを「8」ずつ減少します。
Format.BringToFront	**Ctrl** + **J**	選択したコントロールを Z オーダーの先頭に移動します。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

グローバル ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

テキスト移動のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、テキスト エディタで開いているドキュメント内を移動するときに使用できます。

コマンド	ショートカット キー	説明
Edit.CharLeft	←	カーソルを 1 文字左に移動します。
Edit.CharRight	→	カーソルを 1 文字右に移動します。
Edit.DocumentEnd	**Ctrl** + **End**	カーソルをドキュメントの最後の行に移動します。
Edit.DocumentStart	**Ctrl** + **Home**	カーソルをドキュメントの最初の行に移動します。
Edit.GotoBrace	Ctrl +]	カーソルをドキュメント内の次の中かっこ ({}) に移動します。
Edit.GoToDeclaration	**Ctrl** + **Alt** + **F12**	コードで選択したシンボルの定義を表示します。
Edit.GoToDefinition	**Shift** + **F2**	コードで選択したシンボルの宣言を表示します。
Edit.LineDown	↓	カーソルを 1 行下へ移動します。
Edit.LineEnd	End	カーソルを行末に移動します。
Edit.LineStart	Home	カーソルを行頭に移動します。
Edit.LineUp	↑	カーソルを 1 行上へ移動します。
Edit.PageDown	PageDown	ドキュメントを 1 ページ分下にスクロールします。
Edit.PageUp	PageUp	ドキュメントを 1 ページ分上にスクロールします。
Edit.PreviousBookmark	**Ctrl** + **K**、**Ctrl** + **P**	前のブックマークに移動します。
Edit.ScrollLineDown	**Ctrl** + ↓	テキストを 1 行下にスクロールします。
Edit.ScrollLineUp	**Ctrl** + ↑	テキストを 1 行上にスクロールします。
Edit.ToggleWordWrap	**Ctrl** + **R**、**Ctrl** + **R**	エディタのワード ラップを有効または無効にします。
Edit.ViewBottom	**Ctrl** + **PageDown**	現在のドキュメントの末尾に移動します。
Edit.ViewTop	**Ctrl** + **PageUp**	現在のウィンドウまたはドキュメントの先頭に移動します。
Edit.WordNext	**Ctrl** + →	カーソルを 1 単語右へ移動します。
Edit.WordPrevious	**Ctrl** + ←	カーソルを 1 単語左へ移動します。
View.ViewCode	F7	選択した項目をエディタのコード ビューに表示します。
View.ViewDesigner	**Shift** + **F7**	選択した項目をエディタのデザイン ビューに表示します。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

テキスト選択のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

テキスト操作のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

テキスト選択のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、テキスト エディタで開いているドキュメント内のテキストを選択するときに使用できます。

コマンド	ショートカット キー	説明
Edit.CharLeftExtend	**Shift** + ←	カーソルを 1 文字左に移動して選択範囲を拡張します。
Edit.CharLeftExtendColumn	**Shift** + **Alt** + ←	カーソルを 1 文字左に移動して列の選択範囲を拡張します。
Edit.CharRightExtend	**Shift** + →	カーソルを 1 文字右に移動して選択範囲を拡張します。
Edit.CharRightExtendColumn	**Shift** + **Alt** + →	カーソルを 1 文字右に移動して列の選択範囲を拡張します。
Edit.DocumentEndExtend	**Ctrl** + **Shift** + **End**	カーソル位置からドキュメントの最後の行までの文字列を選択します。
Edit.DocumentStartExtend	**Ctrl** + **Shift** + **Home**	カーソル位置からドキュメントの最初の行までの文字列を選択します。
Edit.GotoBraceExtend	**Ctrl** + **Shift** + 右角かっこ (**]**)	カーソルを次の中かっこ ({}) に移動して、選択範囲を拡張します。
Edit.LineDownExtend	**Shift** + ↓	カーソル位置から、テキスト選択範囲を 1 行下に拡張します。
Edit.LineDownExtendColumn	**Shift** + **Alt** + ↓	カーソルを 1 行下に移動して、列の選択範囲を拡張します。
Edit.LineEndExtend	**Shift** + **End**	カーソル位置から現在の行の行末までの文字列を選択します。
Edit.LineEndExtendColumn	**Shift** + **Alt** + **End**	カーソルを行末に移動して、列の選択範囲を拡張します。
Edit.LineStartExtend	**Shift** + **Home**	カーソル位置から行頭までの文字列を選択します。
Edit.LineUpExtend	**Shift** + ↑	カーソル位置から始めて、文字列を 1 行ずつ選択します。
Edit.LineUpExtendColumn	**Shift** + **Alt** + ↑	カーソルを 1 行上に移動して、列の選択範囲を拡張します。
Edit.PageDownExtend	**Shift** + **PageDown**	選択範囲を 1 ページ下に拡張します。
Edit.PageUpExtend	**Shift** + **PageUp**	選択範囲を 1 ページ上に拡張します。
Edit.SelectAll	**Ctrl** + **A**	現在のドキュメントの内容をすべて選択します。
Edit.SelectCurrentWord	**Ctrl** + **W**	カーソル位置にある単語、またはカーソルの右側にある単語を選択します。
Edit.SelectToLastGoBack	**Ctrl** + **=**	エディタの現在の位置から前の位置までの範囲を選択します。
Edit.ViewBottomExtend	**Ctrl** + **Shift** + **PageDown**	カーソルをビューの最終行に移動して選択範囲を拡張します。
Edit.ViewTopExtend	**Ctrl** + **Shift** + **PageUp**	現在のウィンドウまたはドキュメントの先頭まで選択範囲を拡張します。
Edit.WordNextExtend	**Ctrl** + **Shift** + →	選択範囲を右に 1 単語拡張します。

Edit.WordNextExtendColumn	**Ctrl** + **Shift** + **Alt** + →	カーソルを 1 単語右へ移動して、列の選択範囲を拡張します。
Edit.WordPreviousExtend	**Ctrl** + **Shift** + ←	選択範囲を左に 1 単語拡張します。
Edit.WordPreviousExtendColumn	**Ctrl** + **Shift** + **Alt** + ←	カーソルを 1 単語左へ移動して、列の選択範囲を拡張します。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

テキスト移動のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

テキスト操作のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

テキスト操作のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、テキスト エディタで開いているドキュメント内のテキストを削除、移動、または書式設定するときに使用できます。

コマンド	ショートカット キー	説明
Edit.BreakLine	Enter、Shift + Enter	改行を挿入します。
Edit.CollapsetoDefinition	**Ctrl** + **M**、**Ctrl** + **O**	コード内の領域 (プロシージャなど) を作成するための論理的な境界を自動的に決定し、非表示にします。
Edit.CommentSelection	**Ctrl** + **K**、**Ctrl** + **C**	プログラミング言語に適したコメント構文を使用して、現在のコード行をコメントとしてマークします。
Edit.DeleteHorizontalWhiteSpace	**Ctrl** + **K**、**Ctrl** + ****	選択範囲に含まれている空白を折りたたみます。範囲が選択されていない場合は、カーソルに隣接した空白を削除します。
Edit.FormatDocument	**Ctrl** + **K**、**Ctrl** + **D**	[オプション] ダイアログ ボックスの [テキスト エディタ] セクションの該当する言語の書式ペインで指定されているインデントとスペースの書式設定を適用します。
Edit.FormatSelection	**Ctrl** + **K**、**Ctrl** + **F**	選択したコード行を周囲のコード行に基づいて適切にインデントします。
Edit.HideSelection	**Ctrl** + **M**、**Ctrl** + **H**	選択したテキストを非表示にします。ファイル内の隠し文字列の位置は、シグナル アイコンによって表されます。
Edit.InsertTab	Tab	テキスト行にインデントを設定します。たとえば、空白を 5 つ挿入するように指定できます。.NET Framework Designer 以外では使用できません。
Edit.LineCut	**Ctrl** + **Y**	選択した行を切り取ってクリップボードに移動します。選択行がない場合は、現在の行をクリップボードに移動します。
Edit.LineDelete	**Ctrl** + **Shift** + **Y**	選択した行をすべて削除します。選択行がない場合は、現在の行を削除します。
Edit.LineOpenAbove	**Ctrl** + **Enter**	カーソル位置の上に空行を挿入します。
Edit.LineOpenBelow	**Ctrl** + **Shift** + **Enter**	カーソル位置の下に空行を挿入します。
Edit.LineTranspose	**Shift** + **Alt** + **T**	カーソルがある行を次の行の下に移動します。
Edit.OvertypeMode	Ins	文字を挿入する代わりにドキュメント内の文字を上書きします。テキスト エディタでだけ使用できます。
Edit.StopHidingCurrent	**Ctrl** + **M**、**Ctrl** + **U**	現在選択されている領域のアウトライン情報を削除します。
Edit.StopOutlining	**Ctrl** + **M**、**Ctrl** + **P**	ドキュメント全体からすべてのアウトライン情報を削除します。

Edit.SwapAnchor	**Ctrl** + **R**、**Ctrl** + **P**	現在の選択項目のアンカとエンドポイントを入れ替えます。
Edit.TabLeft	**Shift** + **Tab**	選択した行を 1 タブ ストップ分左に移動します。.NET Framework Designer 以外では使用できません。
Edit.ToggleAllOutlining	**Ctrl** + **M**、**Ctrl** + **L**	以前に隠し文字としてマークしたすべてのセクションの表示と非表示を切り替えます。
Edit.ToggleBookmark	**Ctrl** + **K**、**Ctrl** + **K**	現在の行のブックマークを設定または削除します。
Edit.ToggleOutliningExpansion	**Ctrl** + **M**、**Ctrl** + **M**	現在選択している隠し文字のセクションの表示と非表示を切り替えます。
Edit.ToggleTaskListShortcut	**Ctrl** + **K**、**Ctrl** + **H**	現在の行のショートカットを設定または削除します。
Edit.UncommentSelection	**Ctrl** + **K**、**Ctrl** + **U**	現在のコード行からコメント構文を削除します。
Edit.ViewWhiteSpace	**Ctrl** + **R**、**Ctrl** + **W**	空白とタブの記号の表示/非表示を切り替えます。
Edit.WordTranspose	**Ctrl** + **Shift** + **T**	カーソル位置の両側の単語を入れ替えます。
View.ShowReferences	**Alt** + **F12**	大文字と小文字を区別する完全一致のシンボル検索を実行し、結果を [シンボルの検索結果] ウィンドウに表示します。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

テキスト移動のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

テキスト選択のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

ウィンドウ管理のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、ツールやドキュメントのウィンドウの移動、クローズ、またはウィンドウ内を移動するときに使用できます。

コマンド	ショートカット キー	説明
View.FullScreen	**Shift** + **Alt** + **Enter**	全画面表示モードのオンとオフを切り替えます。
Window.ActivateDocumentWindow	Esc	メニューまたはダイアログ ボックスを閉じ、実行中の操作をキャンセルします。または、現在のドキュメント ウィンドウにフォーカスを置きます。
Window.CloseDocumentWindow	**Ctrl** + **F4**	現在の MDI 子ウィンドウを閉じます。
Window.CloseToolWindow	**Shift** + **Esc**	現在のツール ウィンドウを閉じます。
Window.MoveToDropDownBar	**Ctrl** + **F2**	コード ビューでエディタを表示しているときに、カーソルをドロップダウン バーに移動します。
Window.NextDocumentWindow	**Ctrl** + **F6** **Ctrl** + **Tab**	MDI 子ウィンドウを一度に 1 ウィンドウずつ順番に参照します。
Window.NextPane	**Ctrl** + **F12**	次のツール ウィンドウに移動します。
Window.NextSplitPane	F6	分割ペイン ビューのドキュメントの次のペインに移動します。
Window.NextTab	**Ctrl** + **Page Down**	ドキュメントまたはウィンドウ内の次のタブに移動します。
Window.PreviousDocumentWindow	**Ctrl** + **Shift** + **F6** **Ctrl** + **Shift** + **Tab**	エディタまたはデザイナで前のドキュメントに戻ります。
Window.PreviousPane	**Ctrl** + **Shift** + **F12**	前に選択していたウィンドウに移動します。
Window.PreviousSplitPane	**Shift** + **F6**	分割ペイン ビューのドキュメントの前のペインに移動します。
Window.PreviousTab	**Ctrl** + **Page Up**	ドキュメントまたはウィンドウ内の前のタブに移動します。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

ツール ウィンドウのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

グローバル ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

マクロ ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、マクロの作業をするときに使用できます。

コマンド	ショートカット キー	説明
Tools.MacrosIDE	**Alt** + **F11**	マクロ IDE、Microsoft Visual Studio Macros を起動します。
Tools.RecordTemporaryMacro	**Ctrl** + **Shift** + **R**	環境をマクロ記録モードに設定します。
Tools.RunTemporaryMacro	**Ctrl** + **Shift** + **P**	記録したマクロを再生します。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

関連項目

テキスト操作のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)
ウィンドウ管理のショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)
ツール ウィンドウのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)
プロジェクト ショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

その他の技術情報

Visual Studio マクロ
Visual Basic 6.0 の既定のショートカット キー

ツール ウィンドウのショートカット キー (Visual Basic 6.0 の既定のショートカット オプション)

以下のショートカット キーは、特定のツール ウィンドウを表示するときに使用できます。

コマンド	ショートカット キー	説明
Tools.CommandWindowMarkMode	**Ctrl** + **Shift** + **M**	ウィンドウ内のテキストを選択できるように [コマンド] ウィンドウのモードを変更します。
Tools.AddRemoveToolboxItems	**Ctrl** + **T**	[ツールボックスのカスタマイズ] ダイアログ ボックスを表示します。このダイアログ ボックスでは、ツールボックスのアイテムの追加や削除を行うことができます。
View.MacroExplorer	**Alt** + **F8**	[マクロ エクスプローラ] ウィンドウを表示します。このウィンドウには、現在のソリューションで使用できるすべてのマクロが一覧表示されます。
View.ClassView	**Ctrl** + **Shift** + **C**	[クラス ビュー] ウィンドウを表示します。
View.CommandWindow	**Ctrl** + **Alt** + **A**	[コマンド] ウィンドウを表示します。このウィンドウでは、IDE を操作するコマンドを入力できます。
View.DocumentOutline	**Ctrl** + **Alt** + **T**	[ドキュメント アウトライン] ウィンドウを表示して、現在のドキュメントのアウトラインをフラットまたは階層で表示します。
View.Favorites	**Ctrl** + **Alt** + **F**	Web ページへのショートカットの一覧を表示する [お気に入り] ウィンドウを表示します。
View.ObjectBrowser	F2	オブジェクト ブラウザを表示します。オブジェクト ブラウザでは、パッケージで利用できるクラス、プロパティ、メソッド、イベント、および定数と、プロジェクトのオブジェクト ライブラリやプロシージャを表示できます。
View.Output	**Ctrl** + **Alt** + **O**	[出力] ウィンドウに、実行時のステータス メッセージを表示します。
View.PropertiesWindow	F4	[プロパティ] ウィンドウを表示します。このウィンドウには、現在選択されているアイテムのデザイン時のプロパティおよびイベントの一覧が表示されます。
View.PropertyPages	**Shift** + **F4**	現在選択しているアイテムのプロパティ ページを表示します。
View.ResourceView	**Ctrl** + **Shift** + **E**	[リソース ビュー] ウィンドウを表示します。
View.ServerExplorer	**Ctrl** + **Alt** + **S**	サーバー エクスプローラを表示します。このウィンドウでは、データベース サーバー、イベント ログ、メッセージ キュー、Web サービス、およびその他さまざまなオペレーティング システム サービスを表示および操作できます。
View.SolutionExplorer	**Ctrl** + **R**	現在のソリューション内のプロジェクトとファイルの一覧を示すソリューション エクスプローラを表示します。
View.TaskList	**Ctrl** + **Alt** + **K**	[タスク一覧] ウィンドウを表示します。このウィンドウでは、タスク、コメント、ショートカット、警告、およびエラー メッセージをカスタマイズ、分類、および管理できます。

View.Toolbox	**Ctrl** + **Alt** + **X**	コードで挿入または使用できるコントロールやその他のアイテムを含む [ツールボックス] を表示します。

参照

処理手順

方法 : ショートカット キーの組み合わせを操作する

その他の技術情報

Visual Basic 6.0 の既定のショートカット キー

Visual Basic コードのイベント ハンドラのヘルプ

コード エディタで特定のイベント ハンドラに関するヘルプを表示するには、イベント プロシージャの最後にある **Handles** 句の上にカーソルを置き、F1 キーを押します。たとえば、次の `Form1_Load` ステートメントの場合、カーソルを置く正しい位置は `Handles` の後の太字の語の上です。

```
Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles MyBase.Load

End Sub
```

参照

概念

イベント ハンドラの概要 (Windows フォーム)

その他の技術情報

イベントの処理と発生

方法：Visual Basic でプロジェクトをコンパイルして実行する

Visual Studio 統合開発環境 (IDE) では、プロジェクトをコンパイルし、生成されたアプリケーションを実行するのが簡単です。

IDE の組み込みのデバッガを使用すると、プロジェクトを実行時にデバッグできます。デバッグの詳細については、「Visual Basic アプリケーションのデバッグ」および「エディット コンティニュ (Visual Basic)」を参照してください。

メモ：
使用している設定またはエディションによっては、ヘルプの記載と異なるダイアログ ボックスやメニュー コマンドが表示される場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

現在のプロジェクトをコンパイルして実行するには

- Visual Studio 統合開発環境 (IDE) で F5 キーを押します。

  プロジェクトがコンパイルされ、Visual Studio デバッガ内でアプリケーションが実行されます。

現在のプロジェクトをメニューからコンパイルして実行するには

- Visual Studio IDE の [デバッグ] メニューの [デバッグ開始] をクリックします。

  プロジェクトがコンパイルされ、Visual Studio デバッガ内でアプリケーションが実行されます。

デバッグせずに現在のプロジェクトをコンパイルして実行するには

- Visual Studio IDE で Ctrl + F5 キーを押します。

  プロジェクトがコンパイルされ、アプリケーションが実行されます。

コマンド ライン コンパイラを使用して Visual Basic コードをコンパイルおよび実行する方法については、「コマンド ラインからのビルド (Visual Basic)」を参照してください。

参照

処理手順

方法 : 実行を開始する

概念

Visual Basic アプリケーションのデバッグ

その他の技術情報

コマンド ラインからのビルド (Visual Basic)

エディット コンティニュ (Visual Basic)

セキュリティと Visual Basic 開発

Visual Basic 開発者は、.NET Framework の使用を始める当初から、セキュリティ上の主要な問題に対処しておく必要があります。この概要では、Windows アプリケーションと Web アプリケーションの両方を対象に、開発の実装、デバッグ、配置の各段階について説明します。

概要

Visual Studio では、実行中のアプリケーションのセキュリティを制御できるようになっています。.NET Framework の利用による制御が可能ですが、そのためには、セキュリティを視野に入れてプログラミングを考慮することが必要です。ユーザーにとって親しみやすくて使いやすいアプリケーションを構築するためには、セキュリティ上の問題に対処する必要があります。

セキュリティ上の懸念への対処が必要な状況としてよくあるのは次の 3 つです。

- アクセス許可 : アプリケーションを実行するユーザーは、アプリケーションに対する特権を拒否できます。このような状況が起きるのは、アプリケーションの実行元である場所が、何らかのシステム リソースへのアクセスを許可しないようにユーザーによって指定された場所である場合です。たとえばユーザーは、ネットワーク ドライブ上に格納されているアプリケーションについては、ファイルに対する特権を拒否するように共通言語ランタイムを構成できます。これについては、アプリケーションの作成時に認識し、拒否に対して適切に対応するようにコードを作成する必要があります。詳細については、「セキュリティ ポリシー」を参照してください。
- Web アプリケーション : Web サーバー上の Web アプリケーションにアクセスするユーザーについては、悪意のあるコードの実行やサーバー上のデータの破損を行うことができないようにするための防御が必要です。詳細については、「Web アプリケーションのセキュリティに関する基本的な対策 (Visual Studio)」を参照してください。
- Visual Studio の設定 : 悪意のあるコードからサーバーが攻撃を受けるリスクは、Visual Studio の設定しだいで変動します。詳細については、「ユーザーのアクセス許可と Visual Studio」を参照してください。

リソースのセキュリティ保護は、いくつかの技術を使用して、開発サイクル全体をとおした対処が必要です。アプリケーションの設計、実装、テスト、および配置を注意深く行えば、セキュリティ保護が非常に強固なアプリケーションを構築できます。アプリケーションのセキュリティ保護のためには、ASP.NET、オペレーティング システム、および Web ブラウザに備わっているセキュリティ技術を使用できます。

詳細については、「Visual Studio におけるセキュリティ」を参照してください。

参照

その他の技術情報

安全なコーディングのガイドライン
Windows フォームのセキュリティ
ASP.NET Web サイトのセキュリティ (Visual Studio)
デバイス プロジェクトにおけるセキュリティ
Visual Basic のセキュリティのサンプル

.NET Framework のセキュリティに関する基本事項

コード アクセス セキュリティは .NET Framework の一部で、コードの実行を制御することでリソースへのアクセスを制御します。このセキュリティ機能は、オペレーティング システムが提供するセキュリティとは分離された別個のセキュリティとして追加されます。

動作のしくみ

ユーザーがアプリケーションを実行すると、アプリケーションには .NET Framework 共通言語ランタイムによってゾーンが割り当てられます。5 つのゾーンがあります。

ゾーン	説明
マイ コンピュータ	アプリケーション コードはユーザーのコンピュータ上に存在します。
Local Intranet = ローカル イントラネット	アプリケーション コードはユーザーのイントラネットのファイル共有から実行されます。
インターネット	アプリケーション コードはインターネットから実行されます。
信頼済みサイト	これらは Internet Explorer で "信頼済み" として定義されたサイトのアプリケーションです。
信頼されないサイト	これらは Internet Explorer で "制限付き" として定義されたサイトのアプリケーションです。

最初の 3 つのゾーンの [マイ コンピュータ]、[ローカル イントラネット]、および [インターネット] は、コードが存在する場所に基づいて割り当てられます。この割り当ては、Internet Explorer で特定のサイトを [信頼済みサイト] または [信頼されないサイト] グループに割り当てることで変更できます。

これらの各ゾーンには、システム管理者によって特定のアクセス許可が設定されます。ゾーンのセキュリティ レベルは、完全信頼、中程度の信頼、低度の信頼、または信頼なしに設定できます。アプリケーションにアクセスできるリソースは信頼レベルによって定義されます。実行時にコードに付与されるアクセス許可は、ゾーンと他のセキュリティの証拠 (コードの発行者、厳密な名前、Web サイト、URL など) によって決まります。セキュリティの証拠の詳細については、「証拠」を参照してください。ユーザーのコンピュータのセキュリティ設定を開発者が制御することはできません。また、アプリケーションは実行時に検出した設定の範囲内で動作する必要があります。そのため、アプリケーションが特定のリソースにアクセスすることが拒否される場合があります。たとえば、ファイルにデータを書き込む必要がある場合でも、ユーザーのシステムで例外が発生すると実行時に書き込みアクセスが拒否されます。

このような状況に対応できるアプリケーションの開発が開発者には求められます。これは必ずしも、別の方法でデータを書き込むことができるようにするという意味ではなく、データを書き込むことができない場合があることを想定し、そのような場合への対処方法を考えておく必要があるということです。コードの信頼性を高めるために、より多くの例外処理 (「方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する」を参照) を使用したり、System.Security.Permissions 名前空間のいくつかのオブジェクトを使用したりすることもできます。これらの方法の概要については、この文書の後のセクション「部分信頼環境に対応した開発」を参照してください。

ゾーンのセキュリティ レベルを設定するには、.NET Framework のインストール時にインストールされる管理ツールを使用します。コンピュータ上でゾーンのセキュリティ レベルを設定する方法の詳細については、「管理ツール」を参照してください。

Full Trust

通常、開発者は完全に信頼されている環境で作業します。ソース コードは開発者のハード ドライブで管理され、アプリケーションは開発者の開発コンピュータでテストされます。完全に信頼されている環境では、開発者がコンパイルするコードはすべてローカル コンピュータ上で実行できます。ローカル コンピュータは既定で、完全に信頼されている環境として定義されるため、セキュリティ例外は発生しません。

部分信頼

部分信頼は、完全に信頼されてはいないゾーンを定義します。配置したアプリケーションは新しいゾーンに移動する場合があり、通常、このゾーンがアプリケーションに完全信頼を与えることはありません。部分信頼でコードを実行する最も一般的な 2 つの状況があります。

- インターネットからダウンロードしたコードを実行する。
- ネットワーク共有 (イントラネット) 上に存在するコードを実行する。

部分信頼ゾーンで拒否される可能性があるリソースのいくつかの例を次に示します。

- ファイル I/O 操作。たとえば、ファイルの読み取り、書き込み、作成、削除、印刷が該当します。
- システム コンポーネント。たとえば、レジストリ値、環境変数が該当します。
- サーバー コンポーネント。たとえば、ディレクトリ サービス、レジストリ、イベント ログ、パフォーマンス カウンタ、メッセージ キューが該当しま

す。

部分信頼ではどのようなことが許可されないでしょうか。それを判断するのは簡単なことではありません。.NET Framework では、クラスごとに、および各クラスのメソッドごとに、メソッドを実行するために必要な信頼レベルを定義するセキュリティ属性があり、これらのセキュリティ機能によって実行時に属性にアクセスできない場合があります。ゾーン レベルは、信頼レベルと属性の間の単純な対応付けではなく、特定のクラスとメソッドに対して与えられる特定のアクセス許可のコレクションです。アプリケーションで信頼レベルを単純に照会することはできません。また、使用できないリソースを予測することもできません。開発者は、アプリケーションが完全信頼で実行されているかどうかを判断できます。この判断を行う 1 つの方法について、次のセクション「部分信頼環境に対応した開発」で説明します。

部分信頼環境に対応した開発

このセクションでは、セキュリティへの配慮が記述するコードにどう影響するかについてごく簡単に説明します。部分信頼環境に対応した開発を行うための唯一の方法はありません。どのように対処するかは記述するアプリケーションによって異なります。また、信頼レベルはアプリケーションの実行中に変更される場合があり、既存の信頼レベルでテストしてから次に進むという単純な方法を採用することもできません。

セキュリティ例外コード

部分信頼ゾーンに対応した開発の最初の処理は、セキュリティ例外が発生することを認識したコードを記述することです。次のコードがあるとします。

```
Public Sub MakeABitmap()
    Dim b As Bitmap = New Bitmap(100, 100)
    ' Some code here to draw a nice picture in the bitmap
    b.Save("c:\PrettyPicture.bmp")
End Sub
```

プロジェクトおよびプロジェクト アセンブリが開発者のコンピュータのハード ディスクに格納されている場合、および開発者が自身のコンピュータの Administrators グループのメンバである場合、このメソッドは例外をスローせずに実行されます。このアプリケーションをイントラネットに配置すると、アプリケーションがビットマップ オブジェクトの保存を試みたときに SecurityException がスローされる可能性があります。このコードの近くに Try...Catch...Finally ステートメント (Visual Basic) ブロックがない場合、アプリケーションはこの例外によって終了します。これではユーザーは不満を感じると思います。例外処理コードを追加すると、アプリケーションでは次のように対処できます。

- 必要なすべてのタスクを完了できるとは限らないことをユーザーに警告できます。
- catch ブロックの後で実行されるコードが失敗しないように、既存のオブジェクトをクリーンアップできます。

ビットマップを保存するコードは次のように変更できます。追加したコードはユーザーに対して、セキュリティの拒否によってファイルが保存されていないことを警告します。また、このコードは、セキュリティ エラーを他のファイル I/O エラー (たとえば、ファイル名が誤っている) から分離しています。この方法を使用すると、セキュリティ ホールを防ぐことができます。ユーザー側では、アプリケーションを信頼するかアプリケーションを実行しないようにセキュリティを変更できます。

```
Public Sub MakeABitmap()
    Dim b As Bitmap = Nothing
    Try
        b = New Bitmap(100, 100)
        b.Save("c:\PrettyPicture.bmp")
    Catch ex As System.Security.SecurityException
        ' Let the user know the save won't work.
        MessageBox.Show("Permission to save the file was denied, " & _
           "and the bitmap was not saved.")
    Catch ex As System.Exception
        ' React to other exceptions here.
        MessageBox.Show("Unable to create and save the bitmap.")
    End Try
End Sub
```

System.Security.Permissions 名前空間からのクラス、属性、および列挙体では、アプリケーションのセキュリティ タスクをより細かく制御できます。他のアプリケーションから呼び出すことができるライブラリを記述する場合、そのライブラリで呼び出し元のコードのアクセス許可を確認することもできます。たとえば、単純に次のアセンブリレベル属性をコード ファイルに、つまり AssemblyInfo.vb または AssemblyInfo.cs ファイルの先頭に追加できます。詳細については、「アセンブリ属性の設定」を参照してください。

```
<Assembly: System.Security.Permissions.FileIOPermissionAttribute( _
    System.Security.Permissions.SecurityAction.RequestMinimum, _
    Write:="c:\PrettyPicture.bmp")>
```

ランタイムは、アセンブリの読み込み時にアクセス許可を確認します。ランタイムがアクセス許可の要求を拒否すると、アセンブリの読み込みは失敗し、セキュリティ例外がスローされます。この属性をスタンドアロン アプリケーションに追加すると、アプリケーションが実行されない可能性があります。この属性がクラス ライブラリに表示される場合、そのライブラリは実行時に読み込まれない可能性があります。クラス ライブラリを呼び出すコードに try/catch ブロックを追加する必要があります。

また、ランタイムからアクセス許可を明確に要求することもできます。この場合、次に示すように `Demand` メソッドを使用します。ランタイムは、この要求を許可または拒否する場合があります。セキュリティ例外が発生すると、要求は拒否されます。コードをこのように書き換えることで、ビットマップ ファイルを書き込むためのアクセス許可を明示的に要求できます。

```
Public Sub MakeABitmap()
    Dim b As Bitmap = Nothing
    Dim filename As String = "c:\PrettyPicture.bmp"
    Dim permission As New System.Security.Permissions.FileIOPermission( _
        System.Security.Permissions.FileIOPermissionAccess.Write, _
        filename)

    Try
        permission.Demand()
    Catch ex As System.Security.SecurityException
        ' Let the user know the save won't work.
        MessageBox.Show("Permission to save the file was denied, " & _
           "and the bitmap was not saved.")
    Catch ex As System.Exception
        ' React to other exceptions here.
        MessageBox.Show("Other error.")
    End Try

    Try
        b = New Bitmap(100, 100)
        b.Save(filename)
    Catch ex As System.Exception
        MessageBox.Show("Unable to create and save the bitmap.")
    End Try
End Sub
```

テスト

部分信頼ゾーンに対応した開発の 2 番目の処理は、複数の環境で、特にイントラネットとインターネットからテストすることです。このテストでは、セキュリティ例外を強制的にスローさせます。1 つのテストとして、ローカル コンピュータ上で、管理者権限を持たないユーザー アカウントを作成し、そのアカウントでアプリケーションを実行してみます。これは重要なテストです。

参照

概念

コード アクセス セキュリティの概要

その他の技術情報

コード アクセス セキュリティ

Visual Studio のセキュリティの名前空間

安全なコーディングのガイドライン

セキュリティ保護されたアプリケーションの作成に関するコーディング支援

セキュリティについて考える場合、悪意のあるコードやデータの破損から開発コンピュータとサーバーを保護する必要があります。開発環境には、開発サーバーをセキュリティで保護するのに役立ついくつかの機構が存在します。

Visual Studio のツール

Visual Studio をインストールすると、Users アカウント グループが追加されます。Users グループには、サーバー上で Web アプリケーションを作成および開発するために必要なファイル、共有、およびインターネット インフォメーション サービス (IIS: Internet Information Services) のアクセス許可が含まれています。また、Users グループでは、特定のコンピュータ上のプロセスをローカルまたはリモートのいずれでもデバッグできます。開発者は Users グループのメンバとして、ほとんどのリソースにアクセスできます。詳細については、「ユーザーのアクセス許可と Visual Studio」を参照してください。

デバッグ

デバッグは、配置先のコンピュータ上ではなく、テスト用コンピュータ上で実行することをお勧めします。配置先のサーバー上でデバッグする必要がある場合は、リモート デバッグ コンポーネントのみをインストールし、デバッグが完了したらそれをアンインストールします。デバッグ中はサーバーをオフラインにします。詳細については、「方法 : リモート デバッグをセットアップする」を参照してください。デバッグに関する一般的な情報については、「デバッガのセキュリティ」を参照してください。

配置

ほとんどのアプリケーションの場合、サーバーに .NET Framework がインストールされていれば十分です。配置先のコンピュータに Visual Studio または Visual Studio サーバー コンポーネントがインストールされている場合、そのコンピュータには Users グループが存在します。また、動的な検出を無効にすることもできます。

メモ :

配置先のサーバーには Visual Studio 2005 をインストールしないことを強くお勧めします。Visual Studio のセットアップにより、利用できるシステムにファイルとユーザーの両方が追加されます。Visual Studio 2005 をインストールしたシステムはセキュリティで保護できますが、配置先のサーバーで Visual Studio 2005 が必要ない場合はインストールしないことをお勧めします。

Visual Studio のプロジェクトのコピー機能では、開発で使用した構成ファイル (Web.config) とは別の構成ファイルをアプリケーションと共に配置するオプションを使用できます。開発用のファイルではデバッグが有効になっている場合があります。これを配置すると、例外がスローされたときにユーザー側で呼び出し履歴を調べることができます。したがって、デバッグを許可していない別の構成ファイルを配置することをお勧めします。

詳細については、「ClickOnce の配置とセキュリティ」および「方法 : 部分信頼アプリケーションをデバッグする」を参照してください。

参照

関連項目

このコンピュータ上で、Web 探索に対するプロキシ設定が正しく構成されていません。

概念

セキュリティと Visual Basic 開発
Web アプリケーションのセキュリティに関する基本的な対策 (Visual Studio)
Windows フォームのセキュリティの概要

Visual Basic でのプログラミング

このセクションでは、Visual Basic アプリケーションを作成するときに詳しく理解しておくと役立つプログラミング タスクについて説明します。

このセクションの内容

Visual Basic での実行中のアプリケーションへのアクセス

アプリケーションの実行時の動作を制御する方法、およびアプリケーションについての実行時の情報を取得する方法についてのドキュメントが含まれています。

アプリケーション フォームへのアクセス

My.Forms オブジェクトおよび **My.Application** オブジェクトを使用してアプリケーションのフォームにアクセスする方法についてのドキュメントが含まれています。

アプリケーション リソースへのアクセス

My.Resources オブジェクトを使用してアプリケーションのリソースにアクセスする方法についてのドキュメントが含まれています。

アプリケーション設定へのアクセス

My.Settings オブジェクトを使用してアプリケーションの設定にアクセスする方法についてのドキュメントが含まれています。

アプリケーションの Web サービスへのアクセス

My.WebServices オブジェクトを使用して、アプリケーションが参照している Web サービスにアクセスする方法についてのドキュメントが含まれています。

コンピュータ リソースへのアクセス

My.Computer オブジェクトを使用して、アプリケーションを実行中のコンピュータについての情報にアクセスする方法、およびコンピュータを制御する方法についてのドキュメントが含まれています。

ユーザー データへのアクセス

My.User オブジェクトを使用して実行できるタスクについてのドキュメントが含まれています。

アプリケーションからの情報のログ記録

My.Application.Log オブジェクトおよび **My.Log** オブジェクトを使用してアプリケーションから情報をログに記録する方法と、アプリケーションのログ機能を拡張する方法についてのドキュメントが含まれています。

ドライブ、ディレクトリ、およびファイルの処理

My.Computer.FileSystem オブジェクトを使用してファイル システムにアクセスする方法についてのドキュメントが含まれています。

アプリケーションの更新

My.Application.Deployment プロパティを使用して ClickOnce アプリケーションを更新する方法についてのドキュメントが含まれています。

参照

その他の技術情報

Visual Basic でのアプリケーションの開発

コンピュータ リソースへのアクセス

My.Computer オブジェクトは、**My** の中心となる 3 つのオブジェクトのうちの 1 つです。情報へのアクセスおよび共通で使用される機能へのアクセスを実現します。**My.Computer** には、アプリケーションを実行中のコンピュータにアクセスするためのメソッド、プロパティ、およびイベントが用意されています。含まれているオブジェクトは次のとおりです。

- My.Computer.Audio オブジェクト
- My.Computer.Clipboard オブジェクト
- My.Computer.Clock オブジェクト
- My.Computer.FileSystem オブジェクト
- My.Computer.Info オブジェクト
- My.Computer.Keyboard オブジェクト
- My.Computer.Mouse オブジェクト
- My.Computer.Network オブジェクト
- My.Computer.Ports オブジェクト
- My.Computer.Registry オブジェクト

このセクションの内容

サウンドの再生

バックグラウンドでのサウンドの再生など、**My.Computer.Audio** に関連するタスクを一覧表示します。

クリップボードのデータの格納と読み込み

クリップボードのデータの読み込みと書き込みなど、**My.Computer.Clipboard** に関連するタスクを一覧表示します。

コンピュータの時計へのアクセス

ローカルのシステム時刻の判断など、**My.Computer.Clock** に関連するタスクを一覧表示します。

コンピュータについての情報の取得

コンピュータの完全名や IP アドレスの判断など、**My.Computer.Info** に関連するタスクを一覧表示します。

キーボードへのアクセス

CapsLock キーがオンになっているかどうかの判断など、**My.Computer.Keyboard** に関連するタスクを一覧表示します。

マウスへのアクセス

マウスがあるかどうかの判断など、**My.Computer.Mouse** に関連するタスクを一覧表示します。

ネットワーク操作の実行

ファイルのアップロードまたはダウンロードなど、**My.Computer.Network** に関連するタスクを一覧表示します。

コンピュータのポートへのアクセス

利用可能なシリアル ポートの表示やシリアル ポートへの文字列の送信など、**My.Computer.Ports** に関連するタスクを一覧表示します。

レジストリからの読み取りとレジストリへの書き込み

レジストリ キーのデータの読み込みと書き込みなど、**My.Computer.Registry** に関連するタスクを一覧表示します。

サウンドの再生

My.Computer.Audio オブジェクトには、サウンドを再生するためのメソッドが用意されています。

このセクションの内容

方法 : Visual Basic でサウンドを再生し、完了まで待機する

サウンド ファイルまたはアプリケーション リソースのサウンドを再生し、その完了を待機する方法を説明します。

方法 : Visual Basic でサウンドを再生する

サウンド ファイルまたはアプリケーション リソースのサウンドをバックグラウンドで再生する方法を説明します。

方法 : Visual Basic でループ サウンドを再生する

サウンド ファイルまたはアプリケーション リソースのサウンドをループで継続して再生する方法を説明します。

方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する

バックグラウンドで再生しているサウンドを停止する方法を説明します。

方法 : Visual Basic でシステム サウンドを再生する

システム サウンドを再生する方法を説明します。

参照

My.Computer.Audio オブジェクト

My.Computer.Audio オブジェクトについて説明します。

My.Computer.Audio.Play メソッド

Play メソッドについて説明します。

My.Computer.Audio.PlaySystemSound メソッド

PlaySystemSound メソッドについて説明します。

My.Computer.Audio.Stop メソッド

Stop メソッドについて説明します。現在再生中のサウンドを停止するメソッドです。

方法：Visual Basic でサウンドを再生し、完了まで待機する

この例では、**My.Computer.Audio.Play** メソッドを使用して、サウンド ファイルおよびアプリケーション リソースのサウンドを再生し、完了まで待機します。

バックグラウンド再生を利用すると、サウンドの再生中にアプリケーションで他のコードを実行できます。**My.Computer.Audio.Play** メソッドでは、一度に 1 つのバックグラウンド サウンドのみを再生できます。新しいバックグラウンド サウンドを再生するときには、前のバックグラウンド サウンドの再生は停止されます。

My.Computer.Audio クラスには、オーディオ ファイルの再生に使用できるメソッドとプロパティが用意されています。

使用例

My.Computer.Audio.Play メソッドは、指定のサウンドを再生し、完了まで待機します。

ファイル名が、使用しているシステム上の .wav サウンド ファイルを参照していることを確認してください。

VB

```
Sub PlaySoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.WaitToComplete)
End Sub
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows Forms Applications] の [Sound] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

My.Computer.Audio.Play メソッドは、指定のサウンドを再生し、完了まで待機します。

アプリケーション リソースに `Waterfall` という名前の .wav サウンド ファイルが含まれていることを確認してください。

VB

```
Sub PlaySoundResource()
    My.Computer.Audio.Play(My.Resources.Waterfall, _
        AudioPlayMode.WaitToComplete)
End Sub
```

コードのコンパイル方法

これらのコード例は、Windows フォーム アプリケーション内またはコンソール アプリケーション内でのみ実行できます。詳細については、「My.Computer.Audio.Play メソッド」を参照してください。

堅牢性の高いプログラム

ファイル名は、使用しているシステム上の .wav サウンド ファイルを参照するものである必要があります。

サウンド ファイルの管理を容易にするためには、ファイルをアプリケーション リソースとして格納することをお勧めします。それらには、My.Resources オブジェクト を使用してアクセスできます。

参照

処理手順

方法 : Visual Basic でシステム サウンドを再生する
方法 : Visual Basic でサウンドを再生する

関連項目

My.Computer.Audio.Play メソッド
AudioPlayMode 列挙型
My.Computer.Audio オブジェクト

方法：Visual Basic でサウンドを再生する

この例では、バックグラウンドでサウンドを再生します。

バックグラウンド再生を利用すると、サウンドの再生中にアプリケーションで他のコードを実行できます。**My.Computer.Audio.Play** メソッドでは、一度に 1 つのバックグラウンド サウンドのみを再生できます。新しいバックグラウンド サウンドを再生するときには、前のバックグラウンド サウンドの再生は停止されます。サウンドを再生し、その完了を待機する方法については、
「方法 : Visual Basic でサウンドを再生し、完了まで待機する」を参照してください。

一般に、アプリケーションがループ サウンドを再生している場合には、どこかの時点でサウンドを停止する必要があります。詳細については、
「方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する」を参照してください。

使用例

My.Computer.Audio.Play メソッドは、**PlayMode.Background** が指定されたときに、バックグラウンドでサウンドを再生します。

ファイル名が、使用しているシステム上のサウンド ファイルを参照していることを確認してください。

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

My.Computer.Audio.Play メソッドは、**PlayMode.Background** が指定されたときに、指定のサウンドをバックグラウンドで再生します。

アプリケーション リソースに `Waterfall` という名前の .wav サウンド ファイルが含まれていることを確認してください。

VB

```
Sub PlayBackgroundSoundResource()
    My.Computer.Audio.Play(My.Resources.Waterfall, _
        AudioPlayMode.Background)
End Sub
```

コードのコンパイル方法

これらのコード例は、Windows フォーム アプリケーション内またはコンソール アプリケーション内でのみ実行できます。詳細については、
「My.Computer.Audio.Play メソッド」を参照してください。

堅牢性の高いプログラム

ファイル名は、使用しているシステム上の .wav サウンド ファイルを参照するものである必要があります。

サウンド ファイルの管理を容易にするためには、ファイルをアプリケーション リソースとして格納することをお勧めします。それらには、My.Resources オブジェクト を使用してアクセスできます。

参照

処理手順

方法 : Visual Basic でシステム サウンドを再生する
方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する
方法 : Visual Basic でループ サウンドを再生する

関連項目

My.Computer.Audio.Play メソッド
AudioPlayMode 列挙型

方法 : Visual Basic でループ サウンドを再生する

この例では、バックグラウンドでループ サウンドを再生します。

バックグラウンド再生を利用すると、サウンドの再生中にアプリケーションで他のコードを実行できます。これは、アプリケーションの実行をブロックしないようにサウンドを再生する場合に特に役立ちます。**My.Computer.Audio.Play** メソッドでは、一度に 1 つのバックグラウンド サウンドのみを再生できます。新しいバックグラウンド サウンドを再生するときには、前のバックグラウンド サウンドの再生は停止されます。

一般に、アプリケーションがループ サウンドを再生している場合には、どこかの時点でサウンドを停止する必要があります。詳細については、「方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する」を参照してください。

使用例

My.Computer.Audio.Play メソッドは、**PlayMode.BackgroundLoop** が指定されたときに、指定のサウンドをバックグラウンドで再生します。

ファイル名が、使用しているシステム上の .wav サウンド ファイルを参照していることを確認してください。

VB

```
Sub PlayLoopingBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.BackgroundLoop)
End Sub
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows Forms Applications] の [Sound] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

My.Computer.Audio.Play メソッドは、**PlayMode.BackgroundLoop** が指定されたときに、指定のサウンドをバックグラウンドで再生します。

アプリケーション リソースに `Waterfall` という名前の .wav サウンド ファイルが含まれていることを確認してください。

VB

```
Sub PlayLoopingBackgroundSoundResource()
    My.Computer.Audio.Play(My.Resources.Waterfall, _
          AudioPlayMode.BackgroundLoop)
End Sub
```

コードのコンパイル方法

これらのコード例は、Windows フォーム アプリケーション内またはコンソール アプリケーション内でのみ実行できます。詳細については、「My.Computer.Audio.Play メソッド」を参照してください。

堅牢性の高いプログラム

ファイル名は、使用しているシステム上の .wav サウンド ファイルを参照するものである必要があります。

サウンド ファイルの管理を容易にするためには、ファイルをアプリケーション リソースとして格納することをお勧めします。それらには、My.Resources オブジェクト を使用してアクセスできます。

参照

処理手順

方法 : Visual Basic でシステム サウンドを再生する
方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する
方法 : Visual Basic でサウンドを再生する

関連項目

My.Computer.Audio.Play メソッド
AudioPlayMode 列挙型

方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する

この例では、バックグラウンドで再生しているサウンドを停止します。

バックグラウンド再生を利用すると、サウンドの再生中にアプリケーションで他のコードを実行できます。詳細については、「方法 : Visual Basic でループ サウンドを再生する」および「方法 : Visual Basic でサウンドを再生する」を参照してください。

一般に、アプリケーションがループ サウンドを再生している場合には、どこかの時点でサウンドを停止する必要があります。

使用例

My.Computer.Audio.Stop メソッドを使用して、アプリケーションが現在再生しているバックグラウンド サウンドまたはループ サウンドを停止します。

VB

```
Sub StopBackgroundSound()
    My.Computer.Audio.Stop()
End Sub
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows フォーム アプリケーション] の [サウンド] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

このコード例は、Windows フォーム アプリケーション内またはコンソール アプリケーション内でのみ実行できます。詳細については、「My.Computer.Audio.Stop メソッド」を参照してください。

参照

処理手順

方法 : Visual Basic でサウンドを再生する

方法 : Visual Basic でファイルが存在するかどうかを確認する

関連項目

My.Computer.Audio.Stop メソッド

その他の技術情報

サウンドの再生

方法 : Visual Basic でシステム サウンドを再生する

この例では、**My.Computer.Audio.PlaySystemSound** メソッドを使用して、システム サウンドを再生します。

My.Computer.Audio.PlaySystemSound メソッドは、SystemSound クラスのいずれかの共有メンバをパラメータとして受け取ります。

使用例

My.Computer.Audio.PlaySystemSound メソッドを使用して、指定したシステム サウンドを再生します。

システム サウンド Asterisk は、一般にエラーを表します。詳細については、「**SystemSound**」を参照してください。

VB

```
Sub PlaySystemSound()
    My.Computer.Audio.PlaySystemSound( _
        System.Media.SystemSounds.Asterisk)
End Sub
```

コードのコンパイル方法

このコード例は、Windows フォーム アプリケーション内またはコンソール アプリケーション内でのみ実行できます。詳細については、「My.Computer.Audio.PlaySystemSound メソッド」を参照してください。

参照

処理手順

方法 : Visual Basic でサウンドを再生する
方法 : Visual Basic でループ サウンドを再生する
方法 : Visual Basic でファイルが存在するかどうかを確認する

関連項目

My.Computer.Audio.Stop メソッド
System.Media.SystemSound

クリップボードのデータの格納と読み込み

My.Computer.Clipboard オブジェクトには、クリップボードとのやり取りに使用できるメソッドとプロパティが用意されています。これにより、クリップボードへのデータの書き込み、クリップボードからのデータの取得、クリップボードに指定の形式のデータが格納されているかどうかのチェックが可能です。クリップボードは、アプリケーションがデータを転送できるようにするための関数とメッセージのセットです。クリップボードにはすべてのアプリケーションがアクセスできるので、アプリケーション間でデータを簡単に転送できます。

このセクションの内容

方法 : Visual Basic でクリップボードを消去する

クリップボードを消去する方法を説明します。

方法 : Visual Basic でクリップボードから読み込む

クリップボードからデータを読み込む方法を説明します。

方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する

クリップボードに格納されているデータの種類を判断する方法を説明します。

方法 : Visual Basic でクリップボードに書き込む

クリップボードにデータを書き込む方法を説明します。

方法 : Visual Basic でクリップボードからイメージを取得する

クリップボードからイメージを取得する方法を説明します。

方法 : Visual Basic でオーディオ ストリームをクリップボードに保存する

クリップボードにオーディオ データを保存する方法を説明します。

参照

My.Computer.Clipboard オブジェクト

My.Computer.Clipboard オブジェクトとそのメソッドおよびプロパティについて説明します。

関連するセクション

シリアル化

シリアル化について説明します。クラスをクリップボードに格納するには、シリアル化が可能なクラスであることが必要です。

方法：Visual Basic でクリップボードを消去する

My.Computer.Clipboard.Clear メソッド を使用すると、クリップボードが消去されます。クリップボードは他のプロセスと共有されているので、クリップボードを消去すると他のプロセスに影響が及ぶことがあります。

クリップボードを消去するには

- **Clear** メソッドを次のように使用します。

  VB

```
My.Computer.Clipboard.Clear()
```

参照

処理手順

方法 : Visual Basic でクリップボードから読み込む
方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する
方法 : Visual Basic でオーディオ ストリームをクリップボードに保存する
方法 : Visual Basic でクリップボードに書き込む

関連項目

My.Computer.Clipboard.Clear メソッド

方法：Visual Basic でクリップボードから読み込む

クリップボードは、テキストやイメージなどのデータの格納に使用できます。クリップボードはすべてのアクティブ プロセスが共有しているので、プロセス間でのデータの転送に使用できます。**My.Computer.Clipboard** オブジェクトを使用すると、クリップボードに簡単にアクセスでき、読み込みおよび書き込みを実行できます。クリップボードから読み込むデータの種類に応じて、**GetText**、**GetImage**、**GetData**、**GetAudioStream**、および **GetFileDropDownList** の各メソッドを使用できます。

クリップボードからデータを取得できない場合、ExternalException がスローされます。

クリップボードからテキストを読み込んで表示するには

- **My.Computer.Clipboard.GetText** メソッドを使用してテキストを読み込みます。次のコードでは、テキストを読み込み、メッセージ ボックスに表示します。この例が正しく動作するためには、クリップボードにテキストが格納されている必要があります。

  VB

  ```
  MsgBox(My.Computer.Clipboard.GetText())
  ```

  このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows Forms Applications] の [Clipboard] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

クリップボードからイメージを読み込むには

- **My.Computer.Clipboard.GetImage** メソッドを使用して、クリップボードからイメージを読み込みます。次のコードでは、クリップボードからイメージを読み込んで、 `Button1` の `Image` プロパティに割り当てます。この例が正しく動作するためには、クリップボードにイメージが格納されていて、`Button1` という名前のボタンがある必要があります。

  VB

  ```
  Button1.Image = My.Computer.Clipboard.GetImage()
  ```

  このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows Forms Applications] の [Clipboard] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

参照

処理手順

方法 : Visual Basic でクリップボードに書き込む
方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する

関連項目

My.Computer.Clipboard オブジェクトのメンバ
My.Computer.Clipboard オブジェクト

方法：クリップボードに格納されているファイルの種類を Visual Basic で判断する

クリップボードは、テキストやイメージなどのデータの格納に使用できます。クリップボードは複数のプロセスにまたがって共有されているため、プロセス間でのデータの転送に使用できます。**My.Computer.Clipboard** オブジェクトを使用すると、クリップボードに簡単にアクセスでき、読み込みおよび書き込みを実行できます。

クリップボードのデータは、テキスト、オーディオ ファイル、イメージなど、さまざまな形式をとります。クリップボードに格納されているファイルの種類を判断するには、**ContainsAudio**、**ContainsFileDropList**、**ContainsImage**、および **ContainsText** の各メソッドを使用できます。**ContainsData** メソッドを使用すると、カスタムの形式をチェックできます。

クリップボードにイメージが保持されているかどうかを判断するには

- **ContainsImage** 関数を使用して、クリップボードに格納されているデータがイメージかどうかを判断します。次のコードは、データがイメージかどうかを確認し、それに応じて報告します。

  VB

```
If My.Computer.Clipboard.ContainsImage() = True Then
    MsgBox("Clipboard contains an image.")
Else
    MsgBox("Clipboard does not contain an image.")
End If
```

参照

処理手順

方法 : Visual Basic でクリップボードから読み込む
方法 : Visual Basic でクリップボードに書き込む

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.ContainsAudio メソッド
My.Computer.Clipboard.ContainsFileDropList メソッド
My.Computer.Clipboard.ContainsImage メソッド
My.Computer.Clipboard.ContainsText メソッド
My.Computer.Clipboard.ContainsData メソッド

方法：Visual Basic でクリップボードに書き込む

クリップボードは、テキストやイメージなどのデータの格納に使用できます。クリップボードはすべてのプロセスが共有しているので、プロセス間でのデータの転送に使用できます。**My.Computer.Clipboard** オブジェクトを使用すると、クリップボードに簡単にアクセスでき、読み込みおよび書き込みを実行できます。**SetAudio**、**SetData**、**SetFileDropDownList**、**SetImage**、および **SetText** の各メソッドを使用すると、クリップボードにデータを配置できます。

セキュリティに関するメモ：
クリップボードには他のユーザーからもアクセス可能なので、クリップボードを使ってパスワードや極秘データなどの機密情報を格納しないでください。

クリップボードにテキストを書き込むには

- **My.Computer.Clipboard.SetText** メソッドを使用して、クリップボードにテキストを書き込みます。次のコードは、文字列 "This is a test string" をクリップボードに書き込みます。

  VB

```
My.Computer.Clipboard.SetText("This is a test string.")
```

テキストを特定の形式でクリップボードに書き込むには

- **My.Computer.Clipboard.SetText** メソッドを使用してクリップボードにテキストを書き込み、TextDataFormat の型を指定します。次のコードは、文字列 "This is a test string" を RTF テキストとしてクリップボードに書き込みます。

  VB

```
My.Computer.Clipboard.SetText("This is a test string.", _
System.Windows.Forms.TextDataFormat.Rtf)
```

クリップボードにデータを書き込むには

- **My.Computer.Clipboard.SetData** メソッドを使用して、クリップボードにデータを書き込みます。この例は、DataObject `dataChunk` をカスタム形式 `specialFormat` でクリップボードに書き込みます。

  VB

```
My.Computer.Clipboard.SetData("specialFormat", dataChunk)
```

参照

処理手順

方法 : Visual Basic でクリップボードから読み込む
方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.SetText メソッド
My.Computer.Clipboard.SetData メソッド
My.Computer.Clipboard.SetDataObject メソッド
TextDataFormat

方法：Visual Basic でクリップボードからイメージを取得する

My.Computer.Clipboard.GetImage メソッドを使用して、クリップボードからイメージを取得できます。

クリップボードに格納したアイテムは、アプリケーションの終了後も保持されます。

クリップボードからイメージを取得するには

- **My.Computer.Clipboard.GetImage** メソッドを使用して、クリップボードからイメージを取得します。この例では、クリップボードにイメージが格納されているかどうかを確認してから、そのイメージを取得し、 `PictureBox1` に割り当てます。

  VB

```
If My.Computer.Clipboard.ContainsImage() Then
  Dim grabpicture As System.Drawing.Image
  grabpicture = My.Computer.Clipboard.GetImage()
  picturebox1.Image = grabpicture
End If
```

参照

処理手順

方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard オブジェクトのメンバ
My.Computer.Clipboard.ContainsImage メソッド
My.Computer.Clipboard.GetImage メソッド

方法：Visual Basic でオーディオ ストリームをクリップボードに保存する

My.Computer.Clipboard.SetAudio メソッドを使用すると、オーディオ データをクリップボードに保存できます。

オーディオ データをクリップボードに保存するには

- **My.Computer.Clipboard.SetAudio** メソッドを使用して、オーディオ データをクリップボードに書き込みます。この例では、バイト配列 `musicReader` を作成し、ファイル `cool.wav` をその中に読み込んでから、クリップボードに書き込みます。

  VB

```
Dim musicReader As Byte()
musicReader = My.Computer.FileSystem.ReadAllBytes("cool.wav")
My.Computer.Clipboard.SetAudio(musicReader)
```

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.SetAudio メソッド
My.Computer.Clipboard.GetAudioStream メソッド
My.Computer.Clipboard.ContainsAudio メソッド

その他の技術情報

クリップボードのデータの格納と読み込み

コンピュータの時計へのアクセス

My.Computer.Clock オブジェクトには、現在の現地時刻および世界協定時刻 (グリニッジ標準時に等しい) にシステム時計からアクセスするためのプロパティが用意されています。

解説

この表は、**My.Computer.Clock** オブジェクトを使用する一般的なタスクと、それぞれを説明するトピックへのリンクを一覧表示したものです。

目的	参照項目
GMT 時刻を判断する	My.Computer.Clock.GmtTime プロパティ
ローカルのシステム時刻を判断する	My.Computer.Clock.LocalTime プロパティ
コンピュータの起動から現在までに発生したタイマ刻みの数を判断する	My.Computer.Clock.TickCount プロパティ

参照

関連項目

My.Computer.Clock オブジェクト
My.Computer.Clock オブジェクトのメンバ
System.DateTime

概念

固有カルチャの日付と時刻の形式指定

コンピュータについての情報の取得

My.Computer.Info オブジェクトには、コンピュータのメモリ、ロードされているアセンブリ、名前、およびオペレーティング システムについての情報を取得するためのプロパティが用意されています。

解説

この表は、**My.Computer.Info** オブジェクトを使用して一般的に実行されるタスクと、その方法を示したトピックへのリンクを一覧表示したものです。

目的	参照項目
アプリケーションがインストールされているコンピュータでどれだけの仮想アドレス空間が利用可能かを判断する	My.Computer.Info.TotalVirtualMemory プロパティ
アプリケーションを実行中のコンピュータのプラットフォームの種類を判断する	My.Computer.Info.OSPlatform プロパティ
アプリケーションを実行中のコンピュータのオペレーティング システムを判断する	My.Computer.Info.OSFullName プロパティ
アプリケーションを実行中のコンピュータにインストールされているサービス パックを判断する	My.Computer.Info.OSVersion プロパティ
アプリケーションを実行中のコンピュータにインストールされている **UICulture** を判断する	My.Computer.Info.InstalledUICulture プロパティ

参照

関連項目

My.Computer.Info オブジェクト
My.Computer.Info オブジェクトのメンバ

キーボードへのアクセス

My.Computer.Keyboard オブジェクトには、キーボードの現在の状態 (現在どのキーが押されているかなど) にアクセスするためのプロパティや、アクティブなウィンドウにキーストロークを送るためのメソッドが用意されています。

処理手順

この表は、**My.Computer.Keyboard** オブジェクトに関連するタスクと、それぞれの方法を説明するトピックへのリンクを示しています。

目的	参照項目
CapsLock キーがオンかどうかを判断する	方法 : Visual Basic で CapsLock キーがオンかどうかを確認する
Shift キーが押されているかどうかを判断する	My.Computer.Keyboard.ShiftKeyDown プロパティ
Alt キーが押されているかどうかを判断する	My.Computer.Keyboard.AltKeyDown プロパティ
Ctrl キーが押されているかどうかを判断する	My.Computer.Keyboard.CtrlKeyDown プロパティ
NumLock キーがオンかどうかを判断する	My.Computer.Keyboard.NumLock プロパティ
ScrollLock キーがオンかどうかを判断する	My.Computer.Keyboard.ScrollLock プロパティ
アプリケーションを開始し、キーストロークを送る	方法 : アプリケーションを起動してキーストロークを送る (Visual Basic)

参照

関連項目

My.Computer.Keyboard オブジェクト
System.Windows.Forms.Keys

方法：Visual Basic で CapsLock キーがオンかどうかを確認する

My.Computer.Keyboard オブジェクトを使用すると、NumLock キーや CapsLock キーがオンかどうかを含めた、キーボードの現在の状態を確認できます。

CapsLock キーがオンかどうかを確認するには

- CapsLock キーがオンかどうかを確認するには、**My.Computer.Keyboard.CapsLock** プロパティを使用します。次のコードは、CapsLock キーの状態についてのメッセージを表示します。

  VB

```
If My.Computer.Keyboard.CapsLock Then
    MsgBox("CAPS LOCK is on")
Else
    MsgBox("CAPS LOCK is off")
End If
```

参照

関連項目

My.Computer.Keyboard オブジェクト

概念

キーボードへのアクセス

方法：アプリケーションを起動してキーストロークを送る (Visual Basic)

この例では、**Shell** 関数を使用して電卓アプリケーションを起動し、**My.Computer.Keyboard.SendKeys** メソッドを使用してキーストロークを送って、2 つの数値を乗算します。

使用例

VB

```
Dim ProcID As Integer
' Start the Calculator application, and store the process id.
ProcID = Shell("CALC.EXE", AppWinStyle.NormalFocus)
' Activate the Calculator application.
AppActivate(ProcID)
' Send the keystrokes to the Calculator application.
My.Computer.Keyboard.SendKeys("22", True)
My.Computer.Keyboard.SendKeys("*", True)
My.Computer.Keyboard.SendKeys("44", True)
My.Computer.Keyboard.SendKeys("=", True)
' The result is 22 * 44 = 968.
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows Forms Applications] の [Forms] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

堅牢性の高いプログラム

要求されたプロセス ID のアプリケーションが見つからない場合には、ArgumentException 例外が発生します。

セキュリティ

Shell 関数の呼び出しは、完全に信頼されている必要があります (SecurityException クラス)。

参照

関連項目

My.Computer.Keyboard.SendKeys メソッド
Shell 関数
AppActivate 関数

マウスへのアクセス

My.Computer.Mouse オブジェクトには、コンピュータのマウスについての情報 (マウスがあるかどうか、マウス ボタンの数、マウスのホイールの詳細など) を取得するための方法が用意されています。

解説

この表は、**My.Computer.Mouse** オブジェクトに関連するタスクと、それぞれの方法を説明するトピックへのリンクを示しています。

目的	参照項目
マウスにスクロール ホイールが付いているかどうかを判断する。	My.Computer.Mouse.WheelExists プロパティ
マウスの左ボタンと右ボタンが交換されているかどうかを判断する。	My.Computer.Mouse.ButtonsSwapped プロパティ
マウス ホイールを 1 ノッチ回転したときのスクロール量を設定する。	My.Computer.Mouse.WheelScrollLines プロパティ

参照

関連項目

My.Computer.Mouse オブジェクト
My.Computer.Mouse オブジェクトのメンバ

ネットワーク操作の実行

次の表は、**My.Computer.Network** オブジェクトに関連するタスクの一覧です。

このセクションの内容

方法 : Visual Basic でファイルをアップロードする

My.Computer.Network を使用してファイルをアップロードし、リモートの場所に格納する方法を説明します。

方法 : Visual Basic でファイルをダウンロードする

My.Computer.Network を使用してリモートの場所からファイルをダウンロードする方法を説明します。

方法 : Visual Basic で接続ステータスをチェックする

コンピュータが有効なネットワーク接続を持つかどうかを判断する方法を示します。

方法 : Visual Basic でリモート コンピュータが利用可能かどうかを確認する

Ping メソッドを使用して、リモート コンピュータまたはホストが利用可能かどうかを判断する方法を示します。

参照

My.Computer.Network オブジェクト

ネットワークを扱うためのメソッド、プロパティ、およびイベントの一覧を示します。

My.Computer.Network.DownloadFile メソッド

DownloadFile メソッドについて説明します。

My.Computer.Network.Ping メソッド

Ping メソッドについて説明します。

My.Computer.Network.UploadFile メソッド

UploadFile メソッドについて説明します。

My.Computer.Network.IsAvailable プロパティ

IsAvailable プロパティについて説明します。

Custom

NetworkAvailable イベントについて説明します。

方法：Visual Basic でファイルをアップロードする

My.Computer.Network.UploadFile メソッドを使用すると、ファイルをアップロードして、リモートの場所に格納できます。*ShowUI* パラメータを **True** に設定した場合、アップロードの進行状況を示すダイアログ ボックスが表示され、ユーザーが操作をキャンセルできます。

ファイルをアップロードするには

- **UploadFile** メソッドを使用してファイルをアップロードします。その際、対象ファイルの場所、およびアップロード先のディレクトリの場所を表す文字列または URI (Uniform Resource Identifier) を指定します。この例では、`Order.txt` ファイルを `http://www.cohowinery.com/uploads.aspx` にアップロードします。

  VB

```
My.Computer.Network.UploadFile( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx")
```

操作の進行状況を表示しながらファイルをアップロードするには

- **UploadFile** メソッドを使用してファイルをアップロードします。その際、対象ファイルの場所、およびアップロード先のディレクトリの場所を表す文字列または URI を指定します。この例では、`Order.txt` ファイルを `http://www.cohowinery.com/uploads.aspx` にアップロードします。ユーザー名やパスワードは指定せず、進行状況を表示し、アップロードのタイムアウト間隔は 500 ミリ秒に設定しています。

  VB

```
My.Computer.Network.UploadFile( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx", "", "", True, 500)
```

ユーザー名とパスワードを指定してファイルをアップロードするには

- **UploadFile** メソッドを使用してファイルをアップロードします。その際、対象ファイルの場所、アップロード先のディレクトリの場所を表す文字列または URI、およびユーザー名とパスワードを指定します。この例では、ユーザー名に `anonymous` を、パスワードに空白を指定して、`Order.txt` ファイルを `http://www.cohowinery.com/uploads.aspx` にアップロードします。

  VB

```
My.Computer.Network.UploadFile( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx", "anonymous", "")
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外がスローされる可能性があります。

- ローカル ファイルのパスが無効な場合 (ArgumentException)。
- 認証が失敗した場合 (SecurityException)。
- 接続がタイムアウトした場合 (TimeoutException)。

参照

処理手順

方法 : Visual Basic でファイルをダウンロードする
方法 : Visual Basic でリモート コンピュータが利用可能かどうかを確認する
方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.Network オブジェクト
My.Computer.Network オブジェクトのメンバ
My.Computer.Network.UploadFile メソッド

方法：Visual Basic でファイルをダウンロードする

My.Computer.Network.DownloadFile メソッドを使用すると、リモート ファイルをダウンロードして、特定の場所に格納できます。*ShowUI* パラメータを **True** に設定した場合、ダウンロードの進行状況を示すダイアログ ボックスが表示され、ユーザーが操作をキャンセルできます。既定では、同じ名前を持つ既存のファイルは上書きされません。既存のファイルを上書きするには、*overwrite* パラメータを **True** に設定します。

次の条件を満たす場合は、例外が発生する可能性があります。

- ドライブ名が有効でない場合 (ArgumentException)。
- 必要な認証が与えられていない場合 (UnauthorizedAccessException または SecurityException)。
- 指定した *connectionTimeout* 内にサーバーが応答しない場合 (TimeoutException)。
- Web サイトが要求を拒否した場合 (WebException)。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

セキュリティに関するメモ：

ファイル名からファイルの内容を判断しないでください。たとえば、Form1.vb というファイルが Visual Basic のソース ファイルではない可能性もあります。アプリケーションでデータを使用する前に、入力をすべて検証してください。ファイルの内容が予想どおりでないことがあり、ファイルの内容を読み取るメソッドが失敗する可能性があります。

ファイルをダウンロードするには

- **DownloadFile** メソッドを使用してファイルをダウンロードします。その際、対象ファイルの場所を表す文字列または URI、およびファイルを格納する場所を指定します。この例では、`WineList.txt` ファイルを `http://www.cohowinery.com/downloads` からダウンロードし、`C:\Documents and Settings\All Users\Documents` に保存します。

  VB

```
My.Computer.Network.DownloadFile _
    ("http://www.cohowinery.com/downloads/WineList.txt", _
    "C:\Documents and Settings\All Users\Documents\WineList.txt")
```

タイムアウト間隔を指定してファイルをダウンロードするには

- **DownloadFile** メソッドを使用してファイルをダウンロードします。その際、対象ファイルの場所を表す文字列または URI、ファイルを格納する場所、およびミリ秒単位のタイムアウト間隔 (既定値は 1000) を指定します。この例では、タイムアウト間隔に 500 ミリ秒を指定して、`WineList.txt` ファイルを `http://www.cohowinery.com/downloads` からダウンロードし、`C:\Documents and Settings\All Users\Documents` に保存します。

  VB

```
My.Computer.Network.DownloadFile _
    ("http://www.cohowinery.com/downloads/WineList.txt", _
    "C:\Documents and Settings\All Users\Documents\WineList.txt", False, 500)
```

ユーザー名とパスワードを指定してファイルをダウンロードするには

- **DownLoadFile** メソッドを使用してファイルをダウンロードします。その際、対象ファイルの場所を表す文字列または URI、ファイルを格納する場所、ユーザー名、およびパスワードを指定します。この例では、ユーザー名に `anonymous` を、パスワードに空白を指定して、`WineList.txt` ファイルを `http://www.cohowinery.com/downloads` からダウンロードし、`C:\Documents and Settings\All`

`Users\Documents` に保存します。

VB

```
My.Computer.Network.DownloadFile _
    ("http://www.cohowinery.com/downloads/WineList.txt", _
    "C:\Documents and Settings\All Users\Documents\WineList.txt", "anonymous", "")
```

セキュリティに関するメモ：

DownLoadFile メソッドで使用される FTP プロトコルは、パスワードを含む情報をプレーンテキストで送信するため、重要な情報の送信には使用しないでください。

参照

処理手順

方法 : Visual Basic でファイルをアップロードする

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.Network オブジェクト

My.Computer.Network オブジェクトのメンバ

My.Computer.Network.DownloadFile メソッド

方法：Visual Basic で接続ステータスをチェックする

My.Computer.Network.IsAvailable プロパティを使用して、コンピュータが有効なネットワーク接続またはインターネット接続をしているかどうかを判断できます。

> **メモ：**
>
> 使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

コンピュータが有効な接続をしているかどうかをチェックするには

- **IsAvailable** プロパティが **True** と **False** のどちらであるかを確認します。次のコードは、プロパティのステータスをチェックし、それを報告します。

  VB

  ```
  If My.Computer.Network.IsAvailable = True Then
      MsgBox("Computer is connected.")
  Else
      MsgBox("Computer is not connected.")
  End If
  ```

  このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [接続とネットワーク] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

参照

処理手順

方法 : Visual Basic でリモート コンピュータが利用可能かどうかを確認する

関連項目

My.Computer.Network オブジェクト

My.Computer.Network.IsAvailable プロパティ

方法：Visual Basic でリモート コンピュータが利用可能かどうかを確認する

My.Computer.Network.Ping メソッドを使用すると、リモート コンピュータまたはホストを利用可能かどうかを判断できます。サーバーは、URL、コンピュータ名、または IP アドレスで指定できます。URL を指定するときには、`http://` は含めません。

Ping メソッドは、リモート コンピュータが利用できるかどうかを判断するうえで、フェール セーフなメソッドではありません。ping 先のコンピュータで当該のポートが閉じている場合や、ファイアウォールやルーターによって ping 要求がブロックされる場合があるからです。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

サーバーを ping するには

- **Ping** メソッドが **True** を返すかどうかを確認します。この例では、**Ping** メソッドが **True** を返すかどうかを確認することにより、サーバーを ping できるかどうかを報告します。`198.01.01.01` は、ping するサーバーの IP アドレス、URL、またはコンピュータ名に置き換えます。

VB

```
If My.Computer.Network.Ping("198.01.01.01") Then
  MsgBox("Server pinged successfully.")
Else
  MsgBox("Ping request timed out.")
End If
```

タイムアウトを指定してサーバーを ping するには

- タイムアウト間隔をミリ秒単位で指定して、**Ping** メソッドが **True** を返すかどうかを確認します。タイムアウト値を指定しなかった場合には、`500` が既定値として使用されます。この例では、**Ping** メソッドが **True** を返すかどうかを確認することにより、サーバーを ping できるかどうかを報告します。タイムアウト間隔は 1000 ミリ秒に設定しています。`www.cohowinery.com` は、ping するサーバーの IP アドレス、URL、またはコンピュータ名に置き換えます。

VB

```
If My.Computer.Network.Ping("www.cohowinery.com", 1000) Then
  MsgBox("Server pinged successfully.")
Else
  MsgBox("Ping request timed out.")
End If
```

参照

処理手順

方法 : Visual Basic で接続ステータスをチェックする

関連項目

My.Computer.Network オブジェクト
My.Computer.Network オブジェクトのメンバ
My.Computer.Network.Ping メソッド

コンピュータのポートへのアクセス

My.Computer.Ports オブジェクトには、コンピュータのシリアル ポートにアクセスするためのプロパティとメソッドが用意されています。

このセクションの内容

方法 : Visual Basic で利用可能なシリアル ポートを表示する

利用可能なシリアル ポートを示す方法を説明します。

方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする

コンピュータのシリアル ポートに接続されているモデムをダイヤルする方法を説明します。

方法 : Visual Basic でシリアル ポートに文字列を送信する

コンピュータのシリアル ポートに文字列を送信する方法を説明します。

方法 : Visual Basic でシリアル ポートから文字列を受信する

コンピュータのシリアル ポートから文字列を受信する方法を説明します。

参照

My.Computer.Ports オブジェクト

My.Computer.Ports オブジェクトとそのメンバについて説明します。

My.Computer.Ports.SerialPortNames プロパティ

SerialPortNames プロパティについて説明します。コンピュータのシリアル ポートの名前のコレクションを取得するプロパティです。

My.Computer.Ports.OpenSerialPort メソッド

OpenSerialPort メソッドについて説明します。System.IO.Ports.SerialPort オブジェクトの作成とオープンを行うメソッドです。

関連するセクション

SerialPort

.NET Framework の **SerialPort** クラスについて説明します。

方法：Visual Basic で利用可能なシリアル ポートを表示する

このトピックでは、**My.Computer.Ports** を使用して、コンピュータで利用可能なシリアル ポートを Visual Basic で表示する方法を説明します。

使用するポートをユーザーが選択できるようにするために、シリアル ポートの名前を ListBox コントロールに格納します。

使用例

この例では、**My.Computer.Ports.SerialPortNames** プロパティが返すすべての文字列に対してループします。これらの文字列は、コンピュータで利用可能なシリアル ポートの名前です。

通常は、利用可能なシリアル ポートの一覧から、アプリケーションで使用するシリアル ポートをユーザーが選択します。この例では、シリアル ポートの名前を **ListBox** コントロールに格納しています。詳細については、「ListBox コントロール (Windows フォーム)」を参照してください。

VB

```
Sub GetSerialPortNames()
    ' Show all available COM ports.
    For Each sp As String In My.Computer.Ports.SerialPortNames
        ListBox1.Items.Add(sp)
    Next
End Sub
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [接続とネットワーク] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System.Windows.Forms.dll に対するプロジェクト参照が必要です。
- System.Windows.Forms 名前空間のメンバに対するアクセスが必要です。コード内でメンバ名を完全修飾していない場合は、**Imports** ステートメントを追加します。詳細については、「Imports ステートメント」を参照してください。
- フォーム上に `ListBox1` という名前の **ListBox** コントロールが必要です。

堅牢性の高いプログラム

利用可能なシリアル ポートの名前は、必ずしも **ListBox** コントロールに表示しなくてもかまいません。ComboBox コントロールやその他のコントロールを使用することもできます。ユーザーからの応答が必要ないアプリケーションの場合は、TextBox コントロールを使用して情報を表示できます。

メモ：

My.Computer.Ports.SerialPortNames が返すポート名は、Windows 98 での実行時には正しくない場合があります。アプリケーション エラーを防ぐためには、ポート名を使用してポートを開くときに、**Try...Catch...Finally** ステートメントや **Using** ステートメントなどの例外処理を使用します。

参照

処理手順

方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする
方法 : Visual Basic でシリアル ポートに文字列を送信する
方法 : Visual Basic でシリアル ポートから文字列を受信する

関連項目

My.Computer.Ports オブジェクト

方法：Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする

このトピックでは、Visual Basic で **My.Computer.Ports** を使用してモデムをダイヤルする方法を説明します。

通常、モデムはコンピュータのいずれかのシリアル ポートに接続されています。アプリケーションがモデムとやり取りするためには、適切なシリアル ポートにコマンドを送信する必要があります。

モデムをダイヤルするには

1. モデムが接続されているシリアル ポートを確認します。この例では、モデムが COM1 に接続されていることを前提としています。
2. **My.Computer.Ports.OpenSerialPort** メソッドを使用して、ポートへの参照を取得します。詳細については、「My.Computer.Ports.OpenSerialPort メソッド」を参照してください。

   Using ブロックを使用すると、アプリケーションが例外を生成した場合でも、シリアル ポートを閉じることができます。シリアル ポートを操作するコードは、このブロックまたは **Try...Catch...Finally** ブロック内に記述する必要があります。

   VB
   ```
   Using com1 As IO.Ports.SerialPort = _
           My.Computer.Ports.OpenSerialPort("COM1", 9600)
   End Using
   ```
3. **DtrEnable** プロパティを設定し、コンピュータがモデムからの伝送を受け取る準備ができたことを通知します。

   VB
   ```
   com1.DtrEnable = True
   ```
4. Write メソッドを使用して、ダイヤル コマンドと電話番号をシリアル ポート経由でモデムに送信します。

   VB
   ```
   com1.Write("ATDT 555-0100" & vbCrLf)
   ```

使用例

VB
```
Sub DialModem()
    ' Dial a number via an attached modem on COM1.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1", 9600)
        com1.DtrEnable = True
        com1.Write("ATDT 555-0100" & vbCrLf)
        ' Insert code to transfer data to and from the modem.
    End Using
End Sub
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [接続とネットワーク] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例では、System 名前空間への参照が必要です。

堅牢性の高いプログラム

この例では、モデムが COM1 に接続されていることを前提としています。作成するコードでは、利用可能なシリアル ポートの一覧から、目的の

ポートをユーザーが選択できるようにすることをお勧めします。詳細については、「方法 : Visual Basic で利用可能なシリアル ポートを表示する」を参照してください。

この例では、アプリケーションが例外をスローした場合でもポートが閉じられるよう、**Using** ブロックを使用しています。詳細については、「Using ステートメント (Visual Basic)」を参照してください。

この例では、アプリケーションは、モデムをダイヤルした後でシリアル ポートを切断しています。実際には、モデムとの間でデータの転送が必要となります。詳細については、「方法 : Visual Basic でシリアル ポートから文字列を受信する」を参照してください。

参照

処理手順

方法 : Visual Basic でシリアル ポートに文字列を送信する
方法 : Visual Basic でシリアル ポートから文字列を受信する
方法 : Visual Basic で利用可能なシリアル ポートを表示する

関連項目

My.Computer.Ports オブジェクト
System.IO.Ports.SerialPort

方法 : Visual Basic でシリアル ポートに文字列を送信する

このトピックでは、**My.Computer.Ports** を使用して、Visual Basic でコンピュータのシリアル ポートに文字列を送信する方法を説明します。

使用例

この例では、COM1 シリアル ポートに文字列を送信します。コンピュータによっては、他のシリアル ポートを使用することが必要な場合があります。

My.Computer.Ports.OpenSerialPort メソッドを使用して、ポートへの参照を取得します。詳細については、「My.Computer.Ports.OpenSerialPort メソッド」を参照してください。

Using ブロックを使用すると、アプリケーションが例外を生成した場合でも、シリアル ポートを閉じることができます。シリアル ポートを操作するコードは、このブロックまたは **Try...Catch...Finally** ブロック内に記述する必要があります。

WriteLine メソッドはシリアル ポートにデータを送信します。

VB

```
Sub SendSerialData(ByVal data As String)
    ' Send strings to a serial port.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        com1.WriteLine(data)
    End Using
End Sub
```

コードのコンパイル方法

- この例では、コンピュータが COM1 を使用しているものと想定しています。

堅牢性の高いプログラム

この例では、コンピュータが COM1 を使用しているものと想定しています。実際に作成するコードでは、柔軟性を高めるために、利用可能なシリアル ポートの一覧から、目的のポートをユーザーが選択できるようにすることをお勧めします。詳細については、「方法 : Visual Basic で利用可能なシリアル ポートを表示する」を参照してください。

この例では、アプリケーションが例外をスローした場合でもポートが閉じられるよう、**Using** ブロックを使用しています。詳細については、「Using ステートメント (Visual Basic)」を参照してください。

参照

処理手順

方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする
方法 : Visual Basic で利用可能なシリアル ポートを表示する

関連項目

My.Computer.Ports オブジェクト
System.IO.Ports.SerialPort

方法：Visual Basic でシリアル ポートから文字列を受信する

このトピックでは、**My.Computer.Ports** を使用して、Visual Basic でコンピュータのシリアル ポートから文字列を受信する方法を説明します。

シリアル ポートから文字列を受信するには

1. 返す文字列を初期化します。

   VB

   ```
   Dim returnStr As String = ""
   ```

2. どのシリアル ポートから文字列を取得するかを決定します。この例では、COM1 であるものと想定しています。

3. **My.Computer.Ports.OpenSerialPort** メソッドを使用して、ポートへの参照を取得します。詳細については、「My.Computer.Ports.OpenSerialPort メソッド」を参照してください。

   Using ブロックを使用すると、アプリケーションが例外を生成した場合でも、シリアル ポートを閉じることができます。シリアル ポートを操作するコードは、このブロックまたは **Try...Catch...Finally** ブロック内に記述する必要があります。

   VB

   ```
   Using com1 As IO.Ports.SerialPort = _
           My.Computer.Ports.OpenSerialPort("COM1")
   End Using
   ```

4. **Do** ループを作成します。行がなくなるまでテキストの読み取りを繰り返すためのループです。

   VB

   ```
   Do
   Loop
   ```

5. ReadLine メソッドを使用して、シリアル ポートから次の行のテキストを読み取ります。

   VB

   ```
   Dim Incoming As String = com1.ReadLine()
   ```

6. **If** ステートメントを使用して、**ReadLine** メソッドが **Nothing** を返したかどうか (つまり、テキストがもうないかどうか) を判断します。**Nothing** を返した場合は、**Do** ループを終了します。

   VB

   ```
   If Incoming Is Nothing Then
       Exit Do
   End If
   ```

7. 文字列をきちんと読み取ることができた場合の処理を実行するための **Else** ブロックを **If** ステートメントに追加します。ブロックでは、シリアル ポートから取得した文字列を、返す文字列の末尾に追加します。

   VB

   ```
   Else
       returnStr &= Incoming & vbCrLf
   ```

8. 文字列を返します。

VB

```
Return returnStr
```

使用例

VB

```
Function ReceiveSerialData() As String
    ' Receive strings from a serial port.
    Dim returnStr As String = ""

    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        Do
            Dim Incoming As String = com1.ReadLine()
            If Incoming Is Nothing Then
                Exit Do
            Else
                returnStr &= Incoming & vbCrLf
            End If
        Loop
    End Using

    Return returnStr
End Function
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [接続とネットワーク] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例では、コンピュータが COM1 を使用しているものと想定しています。

堅牢性の高いプログラム

この例では、コンピュータが COM1 を使用しているものと想定しています。実際に作成するコードでは、柔軟性を高めるために、利用可能なシリアル ポートの一覧から、目的のポートをユーザーが選択できるようにすることをお勧めします。詳細については、「方法 : Visual Basic で利用可能なシリアル ポートを表示する」を参照してください。

この例では、アプリケーションが例外をスローした場合でもポートが閉じられるよう、**Using** ブロックを使用しています。詳細については、「Using ステートメント (Visual Basic)」を参照してください。

参照

処理手順

方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする
方法 : Visual Basic でシリアル ポートに文字列を送信する
方法 : Visual Basic で利用可能なシリアル ポートを表示する

関連項目

My.Computer.Ports オブジェクト
System.IO.Ports.SerialPort

レジストリからの読み取りとレジストリへの書き込み

レジストリに関連するタスクと概念説明のトピックを一覧表示します。

Visual Basic 2005 のプログラミングでは、レジストリへのアクセスには、Visual Basic 2005 に用意されている関数を使用する方法と、.NET Framework のレジストリ クラスを使用する方法のいずれかを選択できます。レジストリは、オペレーティング システムによる情報と、コンピュータにホストされるアプリケーションによる情報をホストします。レジストリの作業には、システム リソースや保護情報などへの不適切なアクセスを許可する場合があるため、セキュリティが損なわれる場合があります。

このセクションの内容

一般的なレジストリ タスク

レジストリ キーの値の取得または設定など、レジストリに関連するタスクを一覧表示します。

My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Computer.Registry オブジェクトを使用してレジストリにアクセスする方法を説明します。

Microsoft.Win32 名前空間を使用したレジストリの読み取りと書き込み

Visual Basic 2005 関数 (**DeleteSetting**、**GetSetting**、**GetAllSettings**、および **SaveSetting**) を使用してレジストリにアクセスする方法を説明します。

チュートリアル : レジストリ キーの作成と値の変更

レジストリ キーおよびそのサブキーの作成、設定値の変更、およびキーの削除を行う方法を説明します。

セキュリティとレジストリ

レジストリに関連するセキュリティ問題について説明します。

トラブルシューティング : レジストリの操作

レジストリのアクセス時に発生する一般的な問題とその対処方法を一覧表示します。

関連するセクション

My.Computer.Registry オブジェクト

My.Computer.Registry オブジェクトのメンバを一覧表示し、説明します。

Registry クラス

Registry クラスの概要を説明します。個々のキーとメンバへのリンクが用意されています。

一般的なレジストリ タスク

My.Computer.Registry オブジェクトは、レジストリにアクセスするためのプロパティおよびメソッドを提供します。

処理手順

レジストリを含む場合の例の一覧を、次の表に示します。

操作	参照項目
レジストリ キーを作成し、その値を設定する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
特定のレジストリ キーに値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
レジストリ キーから値を読み込む	方法 : Visual Basic で、レジストリ キーから値を読み取る
レジストリ キーの値を設定する	方法 : Visual Basic でレジストリ キーの値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する

参照

処理手順

チュートリアル : レジストリ キーの作成と値の変更

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

概念

My を使用したレジストリからの読み取りとレジストリへの書き込み

セキュリティとレジストリ

方法：Visual Basic でレジストリ キーを作成し、値を設定する

My.Computer.Registry オブジェクトの **CreateSubKey** メソッドを使用して、レジストリ キーを作成できます。

手順

レジストリ キーを作成するには

- **CreateSubKey** メソッドを使用して、キーを配置するハイブ、およびキーの名前を指定します。パラメータ `Subkey` では、大文字と小文字は区別されません。レジストリ キー `MyTestKey` を HKEY_CURRENT_USER の下に作成する例を次に示します。

  VB

```
My.Computer.Registry.CurrentUser.CreateSubKey("MyTestKey")
```

レジストリ キーを作成し、値を設定するには

1. **CreateSubkey** メソッドを使用して、キーを配置するハイブ、およびキーの名前を指定します。レジストリ キー `MyTestKey` を HKEY_CURRENT_USER の下に作成する例を次に示します。

   VB

```
My.Computer.Registry.CurrentUser.CreateSubKey("MyTestKey")
```

2. **SetValue** メソッドを使用して、値を設定します。この例では、文字列値を次のように設定します。"MyTestKeyValue" を "This is a test value" に。

   VB

```
My.Computer.Registry.SetValue("HKEY_CURRENT_USER\MyTestKey", _
"MyTestKeyValue", "This is a test value.")
```

使用例

レジストリ キー `MyTestKey` を HKEY_CURRENT_USER の下に作成し、文字列値 `MyTestKeyValue` を `This is a test value` に設定する例を次に示します。

VB

```
My.Computer.Registry.CurrentUser.CreateSubKey("MyTestKey")
' Change MyTestKeyValue to This is a test value.
My.Computer.Registry.SetValue("HKEY_CURRENT_USER\MyTestKey", _
"MyTestKeyValue", "This is a test value.")
```

堅牢性の高いプログラム

レジストリの構造を調べて、キーの適切な場所を見つけることができます。たとえば、現在のユーザーの HKEY_CURRENT_USER\Software key を開き、会社名を使用してキーを作成するとします。その後で、会社名のキーにレジストリ値を追加します。

Web アプリケーションからレジストリを読み取る場合、現在のユーザーは、Web アプリケーションで実装されている認証と偽装によって異なります。

ローカル コンピュータ (LocalMachine) よりもユーザー フォルダ (CurrentUser) にデータを書き込む方が安全です。

レジストリの値を作成するときは、その値が既存の値である場合の処理を決めておく必要があります。悪意のあるユーザーによって作成された別のプロセスが既に値を作成し、アクセス権を持っている可能性があります。レジストリ値にデータを設定すると、そのデータを他のプロセスから利用できるようになります。これを回避するために、GetValue メソッドを使用します。このメソッドは、キーがまだ存在しない場合、**Nothing** を返します。

レジストリ キーがアクセス制御リスト (ACL: Access Control List) によって保護されている場合でも、パスワードなどの秘密のデータをプレーンテキストでレジストリに格納するのは危険です。

次の条件を満たす場合は、例外が発生する可能性があります。

- キーの名前が **Nothing** (ArgumentNullException) である場合
- レジストリ キーを作成するためのアクセス許可がユーザーにない場合 (SecurityException)
- キー名が 255 文字の制限を超えている場合 (ArgumentException)
- キーが閉じている場合 (IOException)
- レジストリ キーが読み取り専用の場合 (UnauthorizedAccessException)

セキュリティ

このプロセスを実行するには、アセンブリに対して RegistryPermission クラスで権限レベルが許可されている必要があります。完全には信頼できないコンテキストでプロセスを実行している場合は、権限不足のため例外がスローされることがあります。同様に、設定の作成または書き込みを行うために、ユーザーは正しい ACL を持っている必要があります。たとえば、コード アクセス セキュリティ権限のあるローカル アプリケーションに、オペレーティング システム権限があるとは限りません。詳細については、「コード アクセス セキュリティの基礎」を参照してください。

参照

処理手順

トラブルシューティング : レジストリの操作
チュートリアル : レジストリ キーの作成と値の変更

関連項目

My.Computer.Registry オブジェクト
My.Computer.Registry.CurrentUser プロパティ
CreateSubKey

概念

一般的なレジストリ タスク
コード アクセス セキュリティの基礎

方法：Visual Basic で、レジストリ キーに値が存在するかどうかを確認する

My.Computer.Registry オブジェクトの **GetValue** メソッドは、指定した値が特定のレジストリ キーに存在するかどうかを確認するために使用できます。

Web アプリケーションからレジストリを読み取る場合、現在のユーザーは、Web アプリケーションで実装されている認証と偽装によって異なります。

レジストリ キーに値が存在するかどうかを確認するには

- **GetValue** メソッドを使用して、値を取得します。値をチェックし、値が存在しない場合はメッセージを返すコードの例を次に示します。

  VB

```
If My.Computer.Registry.GetValue("HKEY_LOCAL_MACHINE\TestApp", _
"TestValue", Nothing) Is Nothing Then
  MsgBox("Value does not exist.")
End If
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- キーの名前が、**Nothing** (ArgumentNullException) である場合
- レジストリ キーを作成するためのアクセス許可がユーザーにない場合 (SecurityException)
- キー名が 255 文字の制限を超えている場合 (ArgumentException)
- キーが閉じている場合 (IOException)
- レジストリ キーが読み取り専用の場合 (UnauthorizedAccessException)

セキュリティ

このプロセスを実行するには、アセンブリに対して RegistryPermission クラスで権限レベルが許可されている必要があります。完全には信頼できないコンテキストでプロセスを実行している場合は、権限不足のため例外がスローされることがあります。同様に、設定の作成または書き込みを行うために、ユーザーは正しいアクセス制御リストを持っている必要があります。たとえば、コード アクセス セキュリティ権限のあるローカル アプリケーションに、オペレーティング システム権限があるとは限りません。詳細については、「コード アクセス セキュリティの基礎」を参照してください。

参照

処理手順

トラブルシューティング : レジストリの操作

チュートリアル : レジストリ キーの作成と値の変更

関連項目

My.Computer.Registry オブジェクト

My.Computer.Registry.CurrentUser プロパティ

概念

コード アクセス セキュリティの基礎

一般的なレジストリ タスク

方法：Visual Basic で、レジストリ キーから値を読み取る

My.Computer.Registry オブジェクトの **GetValue** メソッドを使用すると、Windows レジストリ内の値を読み取ることができます。目的のキー (この場合は "Software\MyApp") が存在しないと、例外がスローされます。*ValueName* (この場合は "Name") が存在しないと、**Nothing** が返されます。

レジストリ キーの値を読み取るには

- **GetValue** メソッドをパスと名前を指定して使用し、レジストリ キーから値を読み取ります。次の例では、`HKEY_CURRENT_USER\Software\MyApp` から値 `Name` を読み取り、メッセージ ボックスにこの値を表示します。

  VB

```
Dim readValue As String
readValue = My.Computer.Registry.GetValue _
("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothing)
MsgBox("The value is " & readValue)
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows Operating System] の [Registry] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

堅牢性の高いプログラム

レジストリには、データを格納するために使用される最上位キー (ルート キー) が保持されています。たとえば、HKEY_LOCAL_MACHINE ルート キーは、すべてのユーザーが使用するマシン レベルの設定を格納するために使用されます。一方、HKEY_CURRENT_USER は、個々のユーザーに固有のデータを格納するために使用されます。

次の条件を満たす場合は、例外が発生する可能性があります。

- キーの名前が **Nothing** である場合 (ArgumentNullException)
- ユーザーにレジストリ キーを読み取る権限が与えられていない場合 (SecurityException)
- キー名が 255 文字の制限を超えている場合 (ArgumentException)

セキュリティ

このプロセスを実行するには、アセンブリに対して RegistryPermission クラスで特権レベルが許可されている必要があります。完全には信頼できないコンテキストでプロセスを実行している場合は、権限不足のため例外がスローされることがあります。同様に、ユーザーは、設定の作成または書き込みを行うための正しい ACL を持っている必要があります。たとえば、コード アクセス セキュリティ権限のあるローカル アプリケーションに、オペレーティング システム権限があるとは限りません。詳細については、「コード アクセス セキュリティの基礎」を参照してください。

参照

処理手順

チュートリアル : レジストリ キーの作成と値の変更

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

RegistryHive

概念

一般的なレジストリ タスク

方法 : Visual Basic でレジストリ キーの値を設定する

Windows レジストリの値を書き込むために、**My.Computer.Registry** オブジェクトの **SetValue** メソッドを使用できます。レジストリには、データを格納するために使用されるキーが、最上位またはルートに保持されます。たとえば、HKEY_LOCAL_MACHINE ルート キーは、すべてのユーザーに使用されるマシン レベルの設定を格納するために使用されます。HKEY_CURRENT_USER は、個々のユーザー固有のデータを格納するために使用されます。

完全なキーのパスが含まれる値が存在しない場合、作成されます。

手順

レジストリ キーに値を書き込むには

- **SetValue** メソッドを使用して、キーおよび値を指定します。この例では、`Name` の値を、HKEY_CURRENT_USER\Software\TestApp キーの "著者名" に設定します。

  VB

```
My.Computer.Registry.SetValue _
("HKEY_CURRENT_USER\Software\TestApp", "Name", "Author's Name")
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows オペレーティング システム > Registry] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

堅牢性の高いプログラム

ユーザー設定データが `Microsoft.Win32.Registry.CurrentUser` ハイブに書き込まれます。

レジストリ キーがアクセス制御リスト (ACL: Access Control List) によって保護されていても、パスワードなどの秘密のデータをプレーン テキストでレジストリに格納するのは危険です。

次の条件を満たす場合は、例外が発生する可能性があります。

- キーの名前が **Nothing** (ArgumentNullException) である場合。
- キー名が 255 文字の制限を超えている場合 (ArgumentException)
- このハイブは無効 (**ArgumentException**) です。
- キーが閉じている場合 (IOException)
- パスが無効 (**IOException**) です。
- レジストリ キーが読み取り専用の場合 (UnauthorizedAccessException)

セキュリティ

このプロセスを実行するには、アセンブリに対して RegistryPermission クラスで権限レベルが許可されている必要があります。完全には信頼できないコンテキストでプロセスを実行している場合は、権限不足のため例外がスローされることがあります。同様に、設定の作成または書き込みを行うために、ユーザーは正しい ACL を持っている必要があります。たとえば、コード アクセス セキュリティ権限のあるローカル アプリケーションに、オペレーティング システム権限があるとは限りません。詳細については、「コード アクセス セキュリティの基礎」を参照してください。

参照

処理手順

方法 : Visual Basic で、レジストリ キーから値を読み取る
チュートリアル : レジストリ キーの作成と値の変更
トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクトのメンバ
My.Computer.Registry.SetValue メソッド

概念

一般的なレジストリ タスク

方法：Visual Basic で、レジストリ キーを削除する

DeleteSubKey メソッドおよび DeleteSubKey メソッドを使用して、レジストリ キーを削除できます。

手順

レジストリ キーを削除するには

- `DeleteSubKey` メソッドを使用して、レジストリ キーを削除します。この例では、CurrentUser ハイブの Software/TestApp キーを削除します。これは、コード内で適切な文字列に変更できます。または、ユーザーが指定した情報を使用できます。

  VB

```
My.Computer.Registry.CurrentUser.DeleteSubKey(text)
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows Operating System] の [Registry] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

堅牢性の高いプログラム

キーと値の組み合わせが存在しない場合、`DeleteSubKey` メソッドは空の文字列を返します。

次の条件を満たす場合は、例外が発生する可能性があります。

- キーの名前が、**Nothing** (ArgumentNullException) である場合
- レジストリ キーを削除するためのアクセス許可がユーザーにない場合 (SecurityException)
- キー名が 255 文字の制限を超えている場合 (ArgumentException)
- レジストリ キーが読み取り専用の場合 (UnauthorizedAccessException)

セキュリティ

必要なアクセス許可が実行時に与えられない (RegistryPermission) 場合、またはユーザーが設定の作成や書き込みを行うための適切な (ACL によって決定された) アクセス権を持っていない場合、レジストリ呼び出しは失敗します。たとえば、コード アクセス セキュリティ権限のあるローカル アプリケーションに、オペレーティング システム権限があるとは限りません。

参照

処理手順

トラブルシューティング : レジストリの操作
チュートリアル : レジストリ キーの作成と値の変更

関連項目

DeleteSubKey
DeleteSubKey
RegistryKey

概念

セキュリティとレジストリ
一般的なレジストリ タスク

My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Computer.Registry オブジェクトには、レジストリ キーを操作するためのメソッドおよびプロパティが用意されています。

Windows のレジストリは、オペレーティング システムによる情報と、コンピュータにホストされるアプリケーションによる情報をホストします。このような情報を保存するには **My.Settings** の使用をお勧めしますが、従来のアプリケーションでもレジストリを使用できます。

My.Computer.Registry メンバの一覧については、「My.Computer.Registry オブジェクトのメンバ」を参照してください。

処理手順

My.Computer.Registry オブジェクトに関連するレジストリ キーのタスクの例を次の表に示します。

操作	参照項目
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する
値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
値を読み取る	方法 : Visual Basic で、レジストリ キーから値を読み取る

セキュリティとレジストリ

レジストリの作業には、システム リソースや保護情報などへの不適切なアクセスを許可する場合があるため、セキュリティが損なわれる場合があります。これらのプロパティを使用するには、レジストリ変数へのアクセスを制御する RegistryPermissionAccess 列挙型からの、読み取りおよび書き込み権限が必要です。完全に信頼された状態で実行中のコード (既定のセキュリティ ポリシーでは、ユーザーのローカル ハード ディスクにインストールされているすべてのコード) は、レジストリにアクセスするために必要なアクセス権限を持っています。詳細については、「T:System.Security.Permissions.RegistryPermission」を参照してください。

レジストリ変数は、RegistryPermission を持たないコードがアクセスできるメモリ位置に格納することはできません。同様に、権限を付与する場合には、ジョブを完了させるために必要な最小限の権限を付与する必要があります。

参照

処理手順

トラブルシューティング : レジストリの操作

チュートリアル : レジストリ キーの作成と値の変更

関連項目

My.Settings オブジェクト

概念

セキュリティとレジストリ

Microsoft.Win32 名前空間を使用したレジストリの読み取りと書き込み

My.Computer.Registry を使用すると、レジストリに対するプログラミングの基本的なニーズを満たすことができますが、それに加え、.NET Framework の Microsoft.Win32 名前空間にある Registry クラスおよび RegistryKey クラスも使用できます。

レジストリ クラスのキー

Registry クラスには、サブキーとその値にアクセスするために使用できる、基本レジストリ キーが用意されています。基本キーそのものは読み取り専用です。**Registry** クラスによって公開されている 7 つのキーについて次の表で説明します。

キー	説明
ClassesRoot	ドキュメントの型と、型に関連するプロパティを定義します。
CurrentConfig	各ユーザーに共通のハードウェア構成情報を格納します。
CurrentUser	環境変数など、現在のユーザー固有の設定に関する情報を格納します。
DynData	仮想デバイス ドライバによって使用されるレジストリ データなど、動的なレジストリ データを格納します。
LocalMachine	ローカル コンピュータの構成データを示す 5 つのサブキー (Hardware、SAM、Security、Software、および System) を格納します。
PerformanceData	ソフトウェア コンポーネントのパフォーマンス情報を格納します。
Users	既定のユーザー設定に関する情報を格納します。

セキュリティに関するメモ：

ローカル コンピュータ (LocalMachine) よりも現在のユーザー (CurrentUser) にデータを書き込む方が安全です。作成するキーが、悪意を持っている可能性がある別のプロセスによって既に作成されていた場合、通称 "スクワッティング (無断占拠)" と呼ばれる状態が発生します。このような事態を防ぐには、キーが存在しない場合に **Nothing** を返す GetValue などのメソッドを使用します。

レジストリから値を読み取る

HKEY_CURRENT_USER から文字列を読み取る方法を次の例に示します。

VB

```
Dim regVersion As Microsoft.Win32.RegistryKey
Dim keyValue As String
keyValue = "Software\\Microsoft\\TestApp\\1.0"
regVersion = Microsoft.Win32.Registry.CurrentUser.OpenSubKey(keyValue, False)
Dim intVersion As Integer = 0
If (Not regVersion Is Nothing) Then
    intVersion = regVersion.GetValue("Version", 0)
    regVersion.Close()
End If
```

HKEY_CURRENT_USER の文字列を読み取り、1 つ増加し、HKEY_CURRENT_USER に書き戻す例を次に示します。

VB

```
Dim regVersion As Microsoft.Win32.RegistryKey
regVersion = Microsoft.Win32.Registry.CurrentUser.OpenSubKey( _
             "SOFTWARE\\Microsoft\\TestApp\\1.0", True)
If regVersion Is Nothing Then
    ' Key doesn't exist; create it.
    regVersion = Microsoft.Win32.Registry.CurrentUser.CreateSubKey( _
                 "SOFTWARE\\Microsoft\\TestApp\\1.0")
End If
```

```
Dim intVersion As Integer = 0
If (Not regVersion Is Nothing) Then
    intVersion = regVersion.GetValue("Version", 0)
    intVersion = intVersion + 1
    regVersion.SetValue("Version", intVersion)
    regVersion.Close()
End If
```

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

SystemException Class

ApplicationException Class

My.Computer.Registry オブジェクト

概念

Visual Basic の構造化例外処理の概要

一般的なレジストリ タスク

セキュリティとレジストリ

My を使用したレジストリからの読み取りとレジストリへの書き込み

チュートリアル：レジストリ キーの作成と値の変更

このチュートリアルでは、ユーザーがキーを作成および削除できるように、コンピュータ上のレジストリ キーに移動するアプリケーションを作成する方法を示します。また、値の読み取り、取得、設定および削除を行う方法も示します。

メイン フォームを作成するには

1. [ファイル] メニューの [新しいプロジェクト] を選択し、[Windows アプリケーション] をクリックします。
2. `Value` という名前の TextBox をフォームに追加します。右下隅の [プロパティ] ウィンドウの [(オブジェクト名)] フィールドに、**値**を入力します。
3. `History` という名前の ListBox をフォームに追加します。右下隅の [プロパティ] ウィンドウの [(オブジェクト名)] フィールドに、「**History**」と入力します。
4. 追加の変数を作成し、クラスの宣言の直後に追加します。

   VB

   ```
   Dim tempKey As Microsoft.Win32.RegistryKey
   ```

ComboBox のレジストリ キーを参照するには

1. レジストリ ハイブを表示して選択できる、`selectHive` という名前のComboBox をフォームに追加します。次のコードをフォームの読み込みイベントに追加して、生成します。

   VB

   ```
   selectHive.Items.Add("ClassesRoot")
   selectHive.Items.Add("CurentConfig")
   selectHive.Items.Add("CurrentUser")
   selectHive.Items.Add("DynData")
   selectHive.Items.Add("LocalMachine")
   selectHive.Items.Add("PerformanceData")
   selectHive.Items.Add("Users")
   ```

2. クラスの宣言の後に次のコードをアタッチします。

   VB

   ```
   Dim registryObject As Microsoft.Win32.RegistryKey = Nothing
   ```

3. `selectHive` SelectedIndexChanged イベントに次のコードを追加します。

   VB

   ```
   Select Case selectHive.Text
     Case "ClassesRoot"
       registryObject = My.Computer.Registry.ClassesRoot
     Case "CurrentConfig"
       registryObject = My.Computer.Registry.CurrentConfig
     Case "CurrentUser"
       registryObject = My.Computer.Registry.CurrentUser
     Case "DynData"
       registryObject = My.Computer.Registry.DynData
     Case "LocalMachine"
       registryObject = My.Computer.Registry.LocalMachine
     Case "PerformanceData"
   ```

```
      registryObject = My.Computer.Registry.PerformanceData
    Case "Users"
      registryObject = My.Computer.Registry.Users
  End Select
```

レジストリ キーの値を読み取るには

1. "値の読み取り" のテキストを指定した、`ReadValueButton` という名前の Button をフォームに追加します。
2. "サブキーの入力" のテキストを指定した、`showSubKey` という名前の **TextBox** をフォームに追加します。
3. `ReadValueButton` Click イベントに次のコードを追加します。

   VB

```
tempKey = registryObject
If tempKey Is Nothing Then
  MsgBox("Please select a registry hive.")
  Return
End If
Value.Text = CStr(tempKey.GetValue(ShowSubKey.Text))
History.Items.Add("Read Value " & selectHive.Text & _
                  "\" & ShowSubKey.Text)
```

4. 既存のサブキーの名前を `showSubKey` テキスト ボックスに入力し、アプリケーションをテストします。`ReadValueButton` をクリックすると、[値] テキスト ボックスに値が表示されます。

レジストリキーに値を設定するには

1. "値の設定" のテキストを指定した、`SetValueButton` という名前のボタンをフォームに追加します。
2. `Click` イベントに次のコードを追加します。

   VB

```
tempKey = registryObject
If tempKey Is Nothing Then
  MsgBox("Please select a registry hive.")
  Return
End If
If Value.Text Is Nothing Then
  MsgBox("Please enter a value.")
  Return
End If
tempKey.SetValue(showSubKey.Text, Value.Text)
tempKey.Close()
History.Items.Add("Set Value " & selectHive.Text & _
                  "\" & showSubKey.Text)
```

3. [値] テキスト ボックスにサブキーの新しい値を入力し、`ReadValueButton` という名前のボタンによって値が変更されたことを確認して、アプリケーションをテストします。

レジストリ キーを作成するには

1. "キーの作成" のテキストを指定した、`CreateButton` という名前のボタンをフォームに追加します。
2. **Click** イベントに次のコードを追加します。

   VB

```
registryObject.CreateSubKey(showSubKey.Text)
History.Items.Add("Create Key " & selectHive.Text & _
                  "\" & showSubKey.Text)
```

3. 新しいキー名を [showSubKey] テキスト ボックスに入力し、キーが作成されたことをレジストリ エディタを使用して確認して、アプリケーションをテストします。

レジストリ キーを削除するには

1. "キーの削除" のテキストを指定した、`DeleteButton` という名前のボタンをフォームに追加します。
2. **Click** イベントに次のコードを追加します。

   VB

```
tempKey = registryObject
If tempKey Is Nothing Then
  MsgBox("Please select a registry hive.")
  Return
End If
If showSubKey.Text Is Nothing Then
  MsgBox("Please enter a subkey.")
  Return
End If
registryObject.DeleteSubKey(showSubKey.Text)
History.Items.Add("Delete Key " & selectHive.Text & _
                  "\" & showSubKey.Text)
```

3. サブキーを削除し、キーが削除されたことをレジストリ エディタを使用して確認して、コードをテストします。

参照

処理手順

トラブルシューティング : レジストリの操作

概念

一般的なレジストリ タスク

セキュリティとレジストリ

My を使用したレジストリからの読み取りとレジストリへの書き込み

Microsoft.Win32 名前空間を使用したレジストリの読み取りと書き込み

セキュリティとレジストリ

このページでは、レジストリにデータを格納する際のセキュリティに関連する事項について説明します。

アクセス許可

レジストリ キーが ACL (Access Control List) によって保護されているとはいえ、パスワードなどの秘密のデータを書式なしテキストでレジストリに格納するのは危険です。

レジストリの作業には、システム リソースや保護情報などへの不適切なアクセスを許可する場合があるため、セキュリティが損なわれる場合があります。これらのプロパティを使用するには、レジストリ変数へのアクセスを制御する、RegistryPermissionAccess 列挙型の、読み取りアクセス許可および書き込みアクセス許可を持っている必要があります。完全に信頼された状態で実行中のコード (既定のセキュリティ ポリシーでは、ユーザーのローカル ハードディスクにインストールされているすべてのコード) は、レジストリにアクセスするために必要なアクセス許可を持っています。詳細については、「T:System.Security.Permissions.RegistryPermission」を参照してください。

レジストリ変数は、RegistryPermission を持たないコードがアクセスできるメモリ位置に格納することはできません。同様に、アクセス許可が付与されるとき、作業を行うために必要な最小限の権限が付与されます。

レジストリ アクセス許可のアクセス値は、**RegistryPermissionAccess** 列挙型で定義します。各メンバを次の表に示します。

値	レジストリ変数へのアクセス
AllAccess	作成、読み取り、書き込み
Create	作成
NoAccess	アクセスなし
Read	読み取り
Write	書き込み

レジストリ キーの値のチェック

レジストリの値を作成するときは、その値が既存の値である場合の処理を決めておく必要があります。悪意のあるユーザーによって作成された別のプロセスが既に値を作成し、アクセス権を持っている可能性があります。レジストリ値にデータを設定すると、そのデータを他のプロセスから利用できるようになります。これを回避するために、**GetValue** メソッドを使用します。このメソッドは、キーがまだ存在しない場合、**Nothing** を返します。

セキュリティに関するメモ：
Web アプリケーションからレジストリを読み取る場合、現在のユーザーの ID は、Web アプリケーションで実装されている認証と偽装によって異なります。

参照

処理手順

チュートリアル : レジストリ キーの作成と値の変更

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

My.Computer.Registry オブジェクトのメンバ

概念

一般的なレジストリ タスク

My を使用したレジストリからの読み取りとレジストリへの書き込み

トラブルシューティング：レジストリの操作

レジストリの操作では、固有の例外に遭遇する場合があります。このトピックでは、特に一般的な例外と、それぞれの対処方法を示します。

例外

次の表は、レジストリの操作時に発生する可能性のある例外と、その対処方法のヒントを示しています。

例外	対処方法
ArgumentException	操作を実行している対象のキーが存在することと、指定したパスが 255 文字を超えていない正しいパスであることを確認します。
SecurityException	RegistryPermissionAccess から読み込みと書き込みのアクセス許可があることを確認します。
UnauthorizedAccessException	キーへの書き込みを試みている場合は、そのキーが書き込み用に開かれていることを確認します。

参照

処理手順

チュートリアル : レジストリ キーの作成と値の変更

関連項目

My.Computer.Registry オブジェクト

概念

一般的なレジストリ タスク

My を使用したレジストリからの読み取りとレジストリへの書き込み

Microsoft.Win32 名前空間を使用したレジストリの読み取りと書き込み

セキュリティとレジストリ

他のプロセスへのアクセス

このページは、Process クラスを使用して他のプロセスにアクセスする方法を説明するトピックの一覧です。

このセクションの内容

方法 : アプリケーションを中断する (Visual Basic)

実行中のアプリケーションを中断する方法です。

方法 : プロセスのリストの作成 (Visual Basic)

実行中のアプリケーション プロセスのリストを取得する方法です。

関連するセクション

方法 : アプリケーションを起動してキーストロークを送る (Visual Basic)

アプリケーションを開始する方法です。

Shell 関数

実行可能プログラムを実行し、そのプログラムが実行中である場合、プログラムのプロセス ID を表す整数を返します。

方法：アプリケーションを中断する (Visual Basic)

次に示すのは、実行中のアプリケーションを中断する例です。

使用例

VB

```
Dim pList() As System.Diagnostics.Process = _
    System.Diagnostics.Process.GetProcessesByName("notepad")
For Each proc As System.Diagnostics.Process In pList
    Dim resp As MsgBoxResult
    resp = MsgBox("Terminate " & proc.ProcessName & "?", _
        MsgBoxStyle.YesNo, "Terminate?")
    If resp = MsgBoxResult.Yes Then
        proc.Kill()
    End If
Next
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows Operating System] の [Processes] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例には、次の項目が必要です。

- System.Diagnostics 名前空間を指定する **Imports** ステートメント。詳細については、「Imports ステートメント」を参照してください。
- 実行中のメモ帳 (Notepad.exe) アプリケーション インスタンス

セキュリティ

この例では、プロセスが突然終了するため、アプリケーションで使用していたデータが失われる可能性があります。

Kill メソッドは完全に信頼されている必要があります (SecurityException クラス)。

参照

関連項目

Process

その他の技術情報

他のプロセスへのアクセス

方法：プロセスのリストの作成 (Visual Basic)

次に示すのは、実行中のアプリケーション プロセスのリストを作成する例です。この情報は、**AppActivate** 関数を呼び出すときに使用できます。

使用例

VB

```
Dim pList() As System.Diagnostics.Process = _
    System.Diagnostics.Process.GetProcesses
For Each proc As System.Diagnostics.Process In pList
    MsgBox(proc.ProcessName)
Next
```

コードのコンパイル方法

この例には、次の項目が必要です。

- System.Diagnostics 名前空間を指定する **Imports** ステートメント。詳細については、「Imports ステートメント」を参照してください。

参照

関連項目

AppActivate 関数

Process

その他の技術情報

Visual Basic Windows フォームのサンプル

アプリケーションの更新

My.Application.Deployment プロパティを使用すると、アプリケーションに対して利用可能な更新を確認できます。アプリケーションが ClickOnce アプリケーションとして配置されている場合に、この例では、適切なユーザー インターフェイス テキストが表示されます。

このセクションの内容

この表は、**My.Application.Deployment** プロパティに関連するタスクと、それぞれの方法を説明するトピックへのリンクを示しています。

目的	参照項目
アプリケーションの更新を確認する	方法 : ClickOnce アプリケーションの更新の有無をチェックする
アプリケーションの更新をダウンロードする	方法 : ClickOnce アプリケーションの更新をダウンロードする

参照

関連項目

My.Application.Deployment プロパティ
My.Application オブジェクト

方法：ClickOnce アプリケーションの更新の有無をチェックする

この例では、**My.Application.Deployment** オブジェクトを使用して、アプリケーションに対して利用可能な更新を確認します。アプリケーションが ClickOnce アプリケーションとして配置されている場合は、適切なユーザー インターフェイス (UI) テキストを表示します。

ClickOnce アプリケーションとその配置方法の詳細については、「ClickOnce の配置」および「ClickOnce アプリケーションの発行」を参照してください。

使用例

この例では、My.Application.IsNetworkDeployed プロパティを使用して、ネットワークから ClickOnce を使用してアプリケーションが配置されていることを確認します。次に、**My.Application.Deployment** オブジェクトの CheckForUpdate メソッドを使用して、アプリケーションの ClickOnce 更新が利用可能かどうかをチェックします。

VB

```
Sub CheckUpdateAvailability()
    If My.Application.IsNetworkDeployed() Then
        If My.Application.Deployment.CheckForUpdate() Then
            MsgBox("Update is available for download")
        Else
            MsgBox("No updates are available for download")
        End If
    Else
        MsgBox("Application is not ClickOnce deployed")
    End If
End Sub
```

My.Application.Deployment オブジェクトで更新できるのは、ClickOnce を使用して配置されたアプリケーションのみです。ClickOnce アプリケーションの配置の詳細については、「方法 : ClickOnce アプリケーションを発行する」を参照してください。

参照

処理手順

方法 : ClickOnce アプリケーションの更新をダウンロードする

関連項目

My.Application.Deployment プロパティ

方法：ClickOnce アプリケーションの更新をダウンロードする

この例では、**My.Application.Deployment** オブジェクトを使用して、アプリケーションの最新バージョンをダウンロードおよびインストールします。この例でアプリケーションが更新されるのは、アプリケーションが最新でなくなっていて、かつ ClickOnce アプリケーションとして配置されている場合のみです。

ClickOnce アプリケーションとその配置方法の詳細については、「ClickOnce の配置」および「ClickOnce アプリケーションの発行」を参照してください。

使用例

この例では、My.Application.IsNetworkDeployed プロパティを使用して、アプリケーションが ClickOnce により配置されていることを確認した後で、更新をダウンロードおよびインストールします。Update メソッドは、アプリケーションが最新の状態であるときはアプリケーションを更新しません。更新を使用するには、アプリケーションを再起動する必要があります。

VB

```
Sub UpdateApplication()
    If My.Application.IsNetworkDeployed Then
        My.Application.Deployment.Update()
    End If
End Sub
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [アプリケーション - コンパイル、リソース、および設定] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

My.Application.Deployment オブジェクトで更新できるのは、ClickOnce を使用して配置されたアプリケーションのみです。ClickOnce アプリケーションの配置の詳細については、「方法 : ClickOnce アプリケーションを発行する」を参照してください。

参照

処理手順

方法 : ClickOnce アプリケーションの更新の有無をチェックする

関連項目

My.Application.Deployment プロパティ

アプリケーションからの情報のログ記録

このセクションのトピックでは、**My.Application.Log** オブジェクトまたは **My.Log** オブジェクトを使用してアプリケーションから情報をログに記録する方法と、アプリケーションのログ機能を拡張する方法について取り上げます。

Log オブジェクトには、アプリケーションのログ リスナに情報を書き込むためのメソッドが用意されています。また、**Log** オブジェクトの高度な **TraceSource** プロパティでは、詳細な構成情報にアクセスできます。**Log** オブジェクトは、アプリケーションの構成ファイルにより構成されます。

My.Log オブジェクトは ASP.NET アプリケーションでのみ利用できます。クライアント アプリケーションでは、**My.Application.Log** を使用します。詳細については、「My.Application.Log オブジェクト」および「My.Log オブジェクト」を参照してください。

処理手順

目的	参照項目
イベント情報をアプリケーションのログに書き込む。	方法 : ログ メッセージを書き込む
例外情報をアプリケーションのログに書き込む。	方法 : Visual Basic で例外をログに記録する
アプリケーションが起動および終了するときに、アプリケーションのログにトレース情報を書き込む。	方法 : アプリケーションの起動時または終了時にメッセージをログに記録する
テキスト ファイルへ情報を書き込むように **My.Application.Log** を構成する。	方法 : イベント情報をテキスト ファイルに書き込む
イベント ログへ情報を書き込むように **My.Application.Log** を構成する。	方法 : アプリケーション イベント ログに書き込む
My.Application.Log が情報を書き込む先を変更する。	チュートリアル : My.Application.Log による情報の書き込み先の変更
My.Application.Log が情報を書き込む先を判断する。	チュートリアル : My.Application.Log による情報の書き込み先の確認
My.Application.Log 用のカスタム ログ リスナを作成する。	チュートリアル : カスタム ログ リスナの作成
My.Application.Log ログの出力をフィルタ処理する。	チュートリアル : My.Application.Log の出力のフィルタ処理

参照

処理手順

トラブルシューティング : ログ リスナ

関連項目

My.Application.Log オブジェクト

My.Log オブジェクト

概念

Visual Basic でのアプリケーション ログの使用

Visual Basic での .NET Framework のログ記録とトレース

方法 : ログ メッセージを書き込む

My.Application.Log オブジェクトおよび **My.Log** オブジェクトを使用すると、アプリケーションに関する情報をログに記録できます。この例では、**My.Application.Log.WriteEntry** メソッドを使用してトレース情報をログに記録する方法を示します。

例外情報をログに記録するには、**My.Application.Log.WriteException** メソッドを使用します。
「方法 : Visual Basic で例外をログに記録する」を参照してください。

使用例

この例では、**My.Application.Log.WriteEntry** メソッドを使用してトレース情報を書き込みます。

VB

```
Public Sub TracingTest(ByVal fileName As String)
    My.Application.Log.WriteEntry( _
        "Entering TracingTest with argument " & _
        fileName & ".")
    ' Code to trace goes here.
    My.Application.Log.WriteEntry( _
        "Exiting TracingTest with argument " & _
        fileName & ".")
End Sub
```

セキュリティ

ログに書き込むデータに、ユーザーのパスワードなどの機密情報が含まれないように注意してください。詳細については、
「Visual Basic でのアプリケーション ログの使用」を参照してください。

参照

処理手順

方法 : Visual Basic で例外をログに記録する
チュートリアル : My.Application.Log による情報の書き込み先の確認
チュートリアル : My.Application.Log による情報の書き込み先の変更
チュートリアル : My.Application.Log の出力のフィルタ処理

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
WriteEntry メソッド (My.Application.Log and My.Log)
WriteException メソッド (My.Application.Log および My.Log)

概念

Visual Basic でのアプリケーション ログの使用

方法：Visual Basic で例外をログに記録する

My.Application.Log オブジェクトおよび **My.Log** オブジェクトを使用すると、アプリケーション内で発生した例外に関する情報をログに記録できます。以下の例では、**My.Application.Log.WriteException** メソッドを使用して、明示的にキャッチした例外および未処理の例外をログに記録する方法を示します

トレース情報をログに記録するには、**My.Application.Log.WriteEntry** メソッドを使用します。詳細については、「WriteEntry メソッド (My.Application.Log and My.Log)」を参照してください。

処理した例外をログに記録するには

1. 例外情報を生成するメソッドを作成します。

   VB

   ```
   Public Sub ExceptionLogTest(ByVal fileName As String)
   End Sub
   ```

2. **Try...Catch** ブロックを使用して例外をキャッチします。

   VB

   ```
   Try
   Catch ex As Exception
   End Try
   ```

3. 例外が発生する可能性のあるコードを **Try** ブロックに記述します。

   `Dim` 行および `MsgBox` 行をコメントから外すと、NullReferenceException 例外が発生します。

   VB

   ```
   ' Code that might generate an exception goes here.
   ' For example:
   '    Dim x As Object
   '    MsgBox(x.ToString)
   ```

4. **Catch** ブロックで、**My.Application.Log.WriteException** メソッドを使用して例外情報を書き込みます。

   VB

   ```
   My.Application.Log.WriteException(ex, _
       TraceEventType.Error, _
       "Exception in ExceptionLogTest " & _
       "with argument " & fileName & ".")
   ```

未処理の例外をログに記録するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [アプリケーション イベントの表示] をクリックしてコード エディタを開きます。

   ApplicationEvents.vb ファイルが開かれます。

4. コード エディタで ApplicationEvents.vb ファイルを開きます。[全般] メニューの [MyApplication イベント] をクリックします。

5. [宣言] メニューの [UnhandledException] をクリックします。

   アプリケーションで、メイン アプリケーションの実行前に UnhandledException イベントが発生します。

6. **UnhandledException** イベント ハンドラに **My.Application.Log.WriteException** メソッドを追加します。

   VB

```
My.Application.Log.WriteException(e.Exception, _
    TraceEventType.Critical, _
    "Application shut down at " & _
    My.Computer.Clock.GmtTime.ToString)
```

使用例

次の例は、処理した例外をログに記録するコードの全体です。

VB

```
Public Sub ExceptionLogTest(ByVal fileName As String)
    Try
        ' Code that might generate an exception goes here.
        ' For example:
        '    Dim x As Object
        '    MsgBox(x.ToString)
    Catch ex As Exception
        My.Application.Log.WriteException(ex, _
            TraceEventType.Error, _
            "Exception in ExceptionLogTest " & _
            "with argument " & fileName & ".")
    End Try
End Sub
```

次の例は、未処理の例外をログに記録するコードの全体です。プロジェクト デザイナを使用すると、コード エディタでアプリケーション イベントにアクセスできます。詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

VB

```
Private Sub MyApplication_UnhandledException( _
    ByVal sender As Object, _
    ByVal e As ApplicationServices.UnhandledExceptionEventArgs _
) Handles Me.UnhandledException
    My.Application.Log.WriteException(e.Exception, _
        TraceEventType.Critical, _
        "Application shut down at " & _
        My.Computer.Clock.GmtTime.ToString)
End Sub
```

参照

処理手順

方法 : ログ メッセージを書き込む
チュートリアル : My.Application.Log による情報の書き込み先の確認
チュートリアル : My.Application.Log による情報の書き込み先の変更

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
WriteEntry メソッド (My.Application.Log and My.Log)
WriteException メソッド (My.Application.Log および My.Log)

概念

Visual Basic でのアプリケーション ログの使用

方法：アプリケーションの起動時または終了時にメッセージをログに記録する

My.Application.Log オブジェクトおよび **My.Log** オブジェクトを使用すると、アプリケーション内で発生したイベントに関する情報をログに記録できます。この例では、**My.Application.Log.WriteEntry** メソッドを **Startup** イベントおよび **Shutdown** イベントと組み合わせて使用して、トレース情報を書き込む方法を示します。

アプリケーションのイベント ハンドラ コードにアクセスするには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [アプリケーション イベントの表示] をクリックしてコード エディタを開きます。

   ApplicationEvents.vb ファイルが開かれます。

アプリケーションの起動時にメッセージをログに記録するには

1. コード エディタで ApplicationEvents.vb ファイルを開きます。[全般] メニューの [MyApplication イベント] をクリックします。
2. [宣言] メニューの [スタートアップ] をクリックします。

   アプリケーションでは、メイン アプリケーションの実行前に Startup イベントが発生します。
3. **Startup** イベント ハンドラに **My.Application.Log.WriteEntry** メソッドを追加します。

   VB

```
My.Application.Log.WriteEntry("Application started at " & _
    My.Computer.Clock.GmtTime.ToString)
```

アプリケーションの終了時にメッセージをログに記録するには

1. コード エディタで ApplicationEvents.vb ファイルを開きます。[全般] メニューの [MyApplication イベント] をクリックします。
2. [宣言] メニューの [シャットダウン] をクリックします。

   アプリケーションでは、メイン アプリケーションが実行された後、終了される前の段階で、Shutdown イベントが発生します。
3. **Shutdown** イベント ハンドラに **My.Application.Log.WriteEntry** メソッドを追加します。

   VB

```
My.Application.Log.WriteEntry("Application shut down at " & _
    My.Computer.Clock.GmtTime.ToString)
```

使用例

プロジェクト デザイナを使用して、コード エディタでアプリケーション イベントにアクセスできます。詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

VB

```
Private Sub MyApplication_Startup( _
    ByVal sender As Object, _
    ByVal e As ApplicationServices.StartupEventArgs _
) Handles Me.Startup
    My.Application.Log.WriteEntry("Application started at " & _
        My.Computer.Clock.GmtTime.ToString)
End Sub

Private Sub MyApplication_Shutdown( _
```

```
    ByVal sender As Object, _
    ByVal e As System.EventArgs _
) Handles Me.Shutdown
    My.Application.Log.WriteEntry("Application shut down at " & _
        My.Computer.Clock.GmtTime.ToString)
End Sub
```

参照

処理手順

方法 : アプリケーション イベントを処理する (Visual Basic)

関連項目

My.Log オブジェクト

My.Application.Log オブジェクト

WriteEntry メソッド (My.Application.Log and My.Log)

WriteException メソッド (My.Application.Log および My.Log)

概念

Visual Basic でのアプリケーション ログの使用

方法：アプリケーション イベント ログに書き込む

My.Application.Log オブジェクトおよび **My.Log** オブジェクトを使用すると、アプリケーション内で発生したイベントに関する情報を書き込むことができます。この例では、**My.Application.Log** がアプリケーション イベント ログにトレース情報を書き込むようにイベント ログ リスナを構成する方法を示します。

セキュリティ ログに書き込むことはできません。システム ログに書き込むためには、LocalSystem または Administrator アカウントのメンバであることが必要です。

イベント ログを参照するには、サーバー エクスプローラまたは Windows イベント ビューアを使用できます。詳細については、「方法 : サーバー エクスプローラでイベント ログを使用する」または「方法 : イベント ビューアを起動する」を参照してください。

メモ：

Windows 95、Windows 98、および Windows ME (Millennium Edition) では、イベント ログはサポートされていません。

イベント ログ リスナを追加および構成するには

1. ソリューション エクスプローラで app.config を右クリックし、[開く] をクリックします。

   または

   app.config ファイルがない場合は、次の操作を行います。

   a. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

   b. [新しい項目の追加] ダイアログ ボックスで、[アプリケーション構成ファイル] をクリックします。

   c. [追加] をクリックします。

2. アプリケーション構成ファイルで `<listeners>` セクションを見つけます。

   `<listeners>` セクションは、最上位の `<configuration>` セクションに入れ子になった `<system.diagnostics>` セクションにさらに入れ子になっている、名前属性が "DefaultSource" の `<source>` セクションにあります。

3. その `<listeners>` セクションに次の要素を追加します。

   ```
   <add name="EventLog"/>
   ```

4. 最上位の `<configuration>` セクション内の `<system.diagnostics>` セクションで、`<sharedListeners>` セクションを見つけます。

5. その `<sharedListeners>` セクションに次の要素を追加します。

   ```
   <add name="EventLog"
       type="System.Diagnostics.EventLogTraceListener, System, Version=2.0.0.0, Culture=n
   eutral, PublicKeyToken=b77a5c561934e089"
        initializeData="APPLICATION_NAME"/>
   ```

   `APPLICATION_NAME` をアプリケーションの名前に置き換えます。

   メモ：

   通常、アプリケーションがイベント ログに書き込むのはエラーのみです。ログ出力のフィルタ処理の詳細については、「チュートリアル : My.Application.Log の出力のフィルタ処理」を参照してください。

イベント情報をイベント ログに書き込むには

- **My.Application.Log.WriteEntry** メソッドまたは **My.Application.Log.WriteException** メソッドを使用して、イベント ログに情報を書き込みます。詳細については、「方法 : ログ メッセージを書き込む」および「方法 : Visual Basic で例外をログに記録する」を参照してください。

アセンブリに対してイベント ログ リスナを設定すると、そのアセンブリで **My.Applcation.Log** が書き込んだすべてのメッセージを受け取ります。

参照

処理手順

方法 : Visual Basic で例外をログに記録する

チュートリアル : My.Application.Log による情報の書き込み先の確認

関連項目

My.Application.Log オブジェクト

My.Log オブジェクト

WriteEntry メソッド (My.Application.Log and My.Log)

WriteException メソッド (My.Application.Log および My.Log)

概念

Visual Basic でのアプリケーション ログの使用

方法：イベント情報をテキスト ファイルに書き込む

My.Application.Log オブジェクトおよび **My.Log** オブジェクトを使用すると、アプリケーション内で発生したイベントに関する情報をログに記録できます。この例では、**My.Application.Log.WriteEntry** メソッドを使用してトレース情報のログをログ ファイルに記録する方法を示します。

ファイル ログ リスナを構成するには

1. ソリューション エクスプローラで app.config を右クリックし、[開く] をクリックします。

   または

   app.config ファイルがない場合は、次の操作を行います。

   a. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

   b. [新しい項目の追加] ダイアログ ボックスで、[アプリケーション構成ファイル] をクリックします。

   c. [追加] をクリックします。

2. アプリケーション構成ファイルで `<listeners>` セクションを見つけます。

   <listeners> セクションは、最上位の <configuration> セクションに入れ子になった <system.diagnostics> セクションにさらに入れ子になっている、名前属性が "DefaultSource" の <source> セクションにあります。

3. その `<listeners>` セクションに次の要素を追加します。

```
<add name="FileLogListener" />
```

4. 最上位の `<configuration>` セクションに入れ子になっている `<system.diagnostics>` セクションで、`<sharedListeners>` セクションを見つけます。

5. その `<sharedListeners>` セクションに次の要素を追加します。

```
<add name="FileLogListener"
    type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
          Microsoft.VisualBasic, Version=8.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a"
    initializeData="FileLogListenerWriter"
    location="Custom"
    customlocation="c:\temp\" />
```

   `customlocation` 属性の値をログ ディレクトリに変更します。

メモ：

リスナのプロパティの値を設定するには、プロパティと同じ名前の属性を使用します。ただし、名前のすべての文字を小文字にします。たとえば、`location` 属性と `customlocation` 属性は、Location プロパティと CustomLocation プロパティの値を設定します。

イベント情報をファイル ログに書き込むには

- **My.Application.Log.WriteEntry** メソッドまたは **My.Application.Log.WriteException** メソッドを使用して、ファイル ログに情報を書き込みます。詳細については、「方法 : ログ メッセージを書き込む」および「方法 : Visual Basic で例外をログに記録する」を参照してください。

  アセンブリに対してファイル ログ リスナを設定すると、そのアセンブリで **My.Application.Log** が書き込んだすべてのメッセージを受け取ります。

参照

処理手順

方法 : Visual Basic で例外をログに記録する

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
WriteEntry メソッド (My.Application.Log and My.Log)
WriteException メソッド (My.Application.Log および My.Log)

概念

Visual Basic でのアプリケーション ログの使用

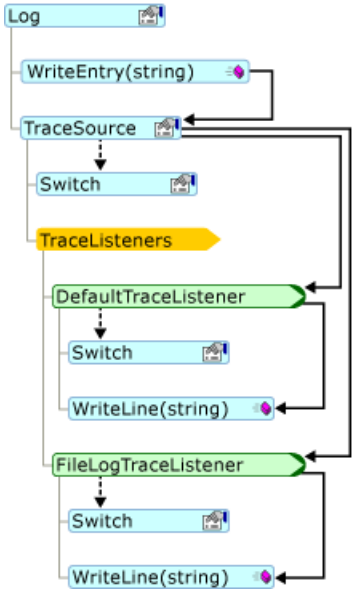
Visual Basic でのアプリケーション ログの使用

My.Applicaton.Log オブジェクトおよび **My.Log** オブジェクトを使用すると、ログおよびトレースの情報をログに簡単に書き込むことができます。

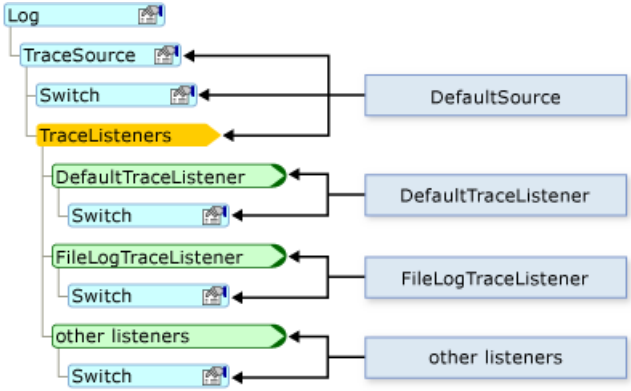
メッセージがログに記録される手順

最初に、ログの TraceSource プロパティの Switch プロパティで、メッセージの重大度レベルがチェックされます。既定では、重大度レベルが "情報" 以上のメッセージだけが、ログの **TraceListener** コレクションで指定されたトレース リスナに渡されます。次に各リスナは、メッセージの重大度レベルを、リスナの **Switch** プロパティと比較します。メッセージの重大度レベルが十分に高い場合には、リスナはメッセージを書き込みます。

次の図は、**WriteEntry** メソッドに書き込まれたメッセージが、ログのトレース リスナの **WriteLine** メソッドにどのように渡されるかを示しています。



アプリケーションの構成ファイルを変更すると、ログおよびトレース リスナの動作を変更できます。次の図は、ログと構成ファイルの各部分の対応を示します。



メッセージが記録される場所

アセンブリに構成ファイルがない場合、**My.Application.Log** オブジェクトと **My.Log** オブジェクトは DefaultTraceListener クラスを介してアプリケーションのデバッグ出力への書き込みを行います。また、**My.Application.Log** オブジェクトは FileLogTraceListener クラスを介してアセンブリのログ ファイルへの書き込みを行い、**My.Log** オブジェクトは、WebPageTraceListener クラスを介して ASP.NET Web ページの出力への書き込みを行います。

デバッグ出力は、アプリケーションをデバッグ モードで実行中のときに、Visual Studio の [出力] ウィンドウで参照できます。[出力] ウィンドウを開くには、[デバッグ] メニュー項目をクリックし、[ウィンドウ] をポイントして、[出力] をクリックします。[出力] ウィンドウで、[出力元の表示] ボックスの [デバッグ] を選択します。

既定では、**My.Application.Log** がログ ファイルを書き込む先は、ユーザーのアプリケーション データ用のパスです。このパスは、DefaultFileLogWriter プロパティ (My.Application.Log および My.Log) の FullLogFileName プロパティから取得できます。このパスの形式は次のとおりです。

*BasePath**CompanyName**ProductName**ProductVersion*

BasePath の一般的な値は次のとおりです。

C:\Documents and Settings*username*\Application Data

CompanyName、*ProductName*、および *ProductVersion* の値は、アプリケーションのアセンブリ情報から取得されます。ログ ファイル名の形式は *AssemblyName*.log です。*AssemblyName* は、アセンブリのファイル名から拡張子を取り除いたものです。複数のログ ファイルが必要な場合 (たとえば、アプリケーションがログに書き込もうとしているときで、元のログが利用できない場合) には、ログ ファイル名の形式は *AssemblyName-iteration*.log となります。*iteration* は正の **Integer** です。

コンピュータおよびアプリケーションの構成ファイルを追加または変更すると、既定の動作をオーバーライドできます。詳細については、「チュートリアル : My.Application.Log による情報の書き込み先の変更」を参照してください。

ログ設定を構成する

Log オブジェクトの既定の実装では、アプリケーション構成ファイル app.config がなくても動作するようになっています。既定の動作を変更するには、新しい設定を記述した構成ファイルを追加する必要があります。詳細については、「チュートリアル : My.Application.Log の出力のフィルタ処理」を参照してください。

app.config ファイルでは、ログの構成セクションは、メインの `<configuration>` ノードの `<system.diagnostics>` ノードにあります。ログ情報は、以下のノードに定義されています。

- **Log** オブジェクトのリスナは、DefaultSource という名前の `<sources>` ノードで定義されています。
- **Log** オブジェクトの重大度レベル フィルタは、DefaultSwitch という名前の `<switches>` ノードで定義されています。
- ログ リスナは `<sharedListeners>` ノードで定義されています。

次のコードに、`<sources>`、`<switches>`、および `<sharedListeners>` の各ノードの例を示します。

```
<configuration>
  <system.diagnostics>
    <sources>
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <add name="FileLog"/>
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="Information" />
    </switches>
    <sharedListeners>
      <add name="FileLog"
        type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
          Microsoft.VisualBasic, Version=8.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL"
        initializeData="FileLogWriter"
      />
    </sharedListeners>
  </system.diagnostics>
</configuration>
```

配置後のログ設定の変更

アプリケーションを開発すると、その構成の設定は、上記の例のように、app.config ファイルに格納されます。アプリケーションを配置した後でも、構成ファイルを編集することで、ログを構成できます。Windows ベースのアプリケーションでは、このファイルの名前は *applicationName*.exe.config で、実行可能ファイルと同じフォルダに存在する必要があります。Web アプリケーションの場合は、プロジェクトに関連付けられた Web.config ファイルになります。

アプリケーションは、クラスのインスタンスを作成するコードを初めて実行するときに、そのオブジェクトについての情報を構成ファイルでチェックします。**Log** オブジェクトの場合は、**Log** オブジェクトに初めてアクセスするときにそれを行います。各オブジェクトについて、システムが構成ファイルをチェックするのは、アプリケーションがそのオブジェクトを最初に作成するときの 1 回だけです。したがって、変更を有効にするためには、アプリケーションの再起動が必要な場合があります。

配置されたアプリケーションでは、アプリケーションの起動前にスイッチ オブジェクトを再設定することで、トレース コードを有効にできます。この作業では、通常、スイッチ オブジェクトのオンとオフを切り替えるか、またはトレース レベルを変更し、その後アプリケーションを再起動します。

セキュリティの考慮事項

ログにデータを書き込むときには、次の点を考慮する必要があります。

- **ユーザー情報を漏らさないようにします。** アプリケーションでは、認められた情報のみをログに書き込むようにします。たとえば、アプリケーション ログにユーザー名を書き込んでも問題ないとしても、ユーザーのパスワードを書き込むのは問題です。
- **ログは安全な場所に配置します。** 機密性の高い情報を含む可能性のあるログは、安全な場所に格納する必要があります。
- **不正な情報は避けます。** 一般に、アプリケーションでは、ユーザーが入力したデータは、使用する前にすべて検証する必要があります。これは、アプリケーション ログにデータを書き込むときも同じです。
- **サービスが滞らないようにします。** アプリケーションがログに書き込む情報が多すぎると、ログが満杯になったり、重要な情報を見つけにくくなったりする可能性があります。

参照

関連項目

My.Application.Log オブジェクト

My.Log オブジェクト

概念

アプリケーションからの情報のログ記録

チュートリアル：My.Application.Log による情報の書き込み先の確認

My.Application.Log オブジェクトは、複数のログ リスナに情報を書き込むことができます。ログ リスナは、コンピュータの構成ファイルで設定し、アプリケーションの構成ファイルでオーバーライドできます。このトピックでは、既定の設定についてと、アプリケーションの設定を確認する方法について説明します。

既定の出力場所の詳細については、「Visual Basic でのアプリケーション ログの使用」を参照してください。

My.Application.Log のリスナを確認するには

1. アセンブリの構成ファイルを見つけます。アセンブリを開発中の段階では、ソリューション エクスプローラから Visual Studio の app.config にアクセスできます。または、構成ファイルは、アセンブリの名前に ".config" を付け加えたファイル名で、アセンブリと同じディレクトリに配置されています。

メモ：
アセンブリによっては、構成ファイルがない場合もあります。

   構成ファイルは XML ファイルです。

2. `<sources>` セクション内にある、`name` 属性が "DefaultSource" の `<source>` セクションで、`<listeners>` セクションを見つけます。`<sources>` セクションは、最上位の `<configuration>` セクション内の `<system.diagnostics>` セクションにあります。

   これらのセクションがない場合には、**My.Application.Log** のログ リスナは、コンピュータの構成ファイルで設定されている可能性があります。以下の手順は、コンピュータの構成ファイルの定義を確認する方法の説明です。

   a. コンピュータの machine.config ファイルを見つけます。通常は、*SystemRoot*\Microsoft.NET\Framework*frameworkVersion*\CONFIG ディレクトリにあります。*SystemRoot* はオペレーティング システム ディレクトリ、*frameworkVersion* は .NET Framework のバージョンです。

      machine.config の設定は、アプリケーションの構成ファイルでオーバーライドできます。

      以下に示すオプションの要素が存在しない場合には、自分で作成できます。

   b. 最上位の `<configuration>` セクション内にある `<system.diagnostics>` セクション内の `<sources>` セクションで、`name` 属性が "DefaultSource" の `<source>` セクション内の `<listeners>` セクションを見つけます。

      これらのセクションがない場合には、**My.Application.Log** は既定のログ リスナのみを持ちます。

3. `<listeners>` セクションで `<add>` 要素を見つけます。

   これらの要素は、名前付きのログ リスナを **My.Application.Log** のソースに追加します。

4. 最上位の `<configuration>` セクション内にある `<system.diagnostics>` セクション内の `<sharedListeners>` セクションで、ログ リスナの名前の `<add>` 要素を見つけます。

5. 多くの型の共有リスナでは、リスナがデータを書き込む先は、リスナの初期化データで指定されています。

   - Microsoft.VisualBasic.Logging.FileLogTraceListener リスナは、導入部で説明したように、ファイル ログに書き込みます。
   - System.Diagnostics.EventLogTraceListener リスナは、`initializeData` パラメータで指定された、コンピュータのイベント ログに情報を書き込みます。イベント ログを参照するには、サーバー エクスプローラまたは Windows イベント ビューアを使用できます。詳細については、「方法 : サーバー エクスプローラでイベント ログを使用する」または「方法 : イベント ビューアを起動する」を参照してください。
   - System.Diagnostics.DelimitedListTraceListener リスナおよび System.Diagnostics.XmlWriterTraceListener リスナは、`initializeData` パラメータで指定されたファイルに書き込みます。
   - System.Diagnostics.ConsoleTraceListener リスナは、コマンド ライン コンソールに書き込みます。
   - 他の型のログ リスナが情報を書き込む先については、その型のドキュメントを参照してください。

参照

処理手順

方法 : Visual Basic で例外をログに記録する

方法 : ログ メッセージを書き込む
チュートリアル : My.Application.Log による情報の書き込み先の変更
方法 : サーバー エクスプローラでイベント ログを使用する
方法 : イベント ビューアを起動する
トラブルシューティング : ログ リスナ

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
DefaultTraceListener
EventLogTraceListener
DelimitedListTraceListener
XmlWriterTraceListener
ConsoleTraceListener
System.Diagnostics

概念

Visual Basic でのアプリケーション ログの使用

チュートリアル：My.Application.Log による情報の書き込み先の変更

My.Application.Log オブジェクトおよび **My.Log** オブジェクトを使用すると、アプリケーション内で発生したイベントに関する情報をログに記録できます。このチュートリアルでは、既定の設定をオーバーライドして、**Log** オブジェクトによる書き込み先を他のログ リスナに変更する方法を示します。

前提条件

Log オブジェクトは、複数のログ リスナに情報を書き込むことができます。ログ リスナの構成を変更する前に、現在の構成を確認する必要があります。詳細については、「チュートリアル : My.Application.Log による情報の書き込み先の確認」を参照してください。

必要に応じて、「方法 : イベント情報をテキスト ファイルに書き込む」および「方法 : アプリケーション イベント ログに書き込む」も参照してください。

リスナを追加するには

1. ソリューション エクスプローラで app.config を右クリックし、[開く] をクリックします。

   または

   app.config ファイルがない場合は、次の操作を行います。

   a. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

   b. [新しい項目の追加] ダイアログ ボックスで、[アプリケーション構成ファイル] をクリックします。

   c. [追加] をクリックします。

2. `<sources>` セクション内にある、`name` 属性が "DefaultSource" の `<source>` セクションで、`<listeners>` セクションを見つけます。`<sources>` セクションは、最上位の `<configuration>` セクション内の `<system.diagnostics>` セクションにあります。

3. その `<listeners>` セクションに次の要素を追加します。

```
<!-- Uncomment to connect the application file log. -->
<!-- <add name="FileLog" /> -->
<!-- Uncomment to connect the event log. -->
<!-- <add name="EventLog" /> -->
<!-- Uncomment to connect the event log. -->
<!-- <add name="Delimited" /> -->
<!-- Uncomment to connect the XML log. -->
<!-- <add name="XmlWriter" /> -->
<!-- Uncomment to connect the console log. -->
<!-- <add name="Console" /> -->
```

4. **Log** のメッセージを受け取らせるログ リスナをコメントから外します。

5. 最上位の `<configuration>` セクション内の `<system.diagnostics>` セクションで、`<sharedListeners>` セクションを見つけます。

6. その `<sharedListeners>` セクションに次の要素を追加します。

```
        <add name="FileLog"
             type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
                   Microsoft.VisualBasic, Version=8.0.0.0,
                   Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
             initializeData="FileLogWriter" />
        <add name="EventLog"
             type="System.Diagnostics.EventLogTraceListener,
                   System, Version=2.0.0.0,
                   Culture=neutral, PublicKeyToken=b77a5c561934e089"
             initializeData="sample application"/>
        <add name="Delimited"
             type="System.Diagnostics.DelimitedListTraceListener,
```

```
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="c:\temp\sampleDelimitedFile.txt"
            delimiter=";;;"
            traceOutputOptions="DateTime" />
        <add name="XmlWriter"
            type="System.Diagnostics.XmlWriterTraceListener,
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="c:\temp\sampleLogFile.xml" />
        <add name="Console"
            type="System.Diagnostics.ConsoleTraceListener,
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="true" />
```

7. app.config ファイルの内容は次の XML のようになります。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- This section configures My.Application.Log -->
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <add name="FileLog"/>
          <!-- Uncomment to connect the application file log. -->
          <!-- <add name="FileLog" /> -->
          <!-- Uncomment to connect the event log. -->
          <!-- <add name="EventLog" /> -->
          <!-- Uncomment to connect the event log. -->
          <!-- <add name="Delimited" /> -->
          <!-- Uncomment to connect the XML log. -->
          <!-- <add name="XmlWriter" /> -->
          <!-- Uncomment to connect the console log. -->
          <!-- <add name="Console" /> -->
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="Information" />
    </switches>
    <sharedListeners>
      <add name="FileLog"
          type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
                Microsoft.VisualBasic, Version=8.0.0.0,
                Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
          initializeData="FileLogWriter" />
      <add name="EventLog"
          type="System.Diagnostics.EventLogTraceListener,
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
          initializeData="sample application"/>
      <add name="Delimited"
          type="System.Diagnostics.DelimitedListTraceListener,
                System, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=b77a5c561934e089"
          initializeData="c:\temp\sampleDelimitedFile.txt"
          delimiter=";;;"
```

```
            traceOutputOptions="DateTime" />
        <add name="XmlWriter"
            type="System.Diagnostics.XmlWriterTraceListener,
                  System, Version=2.0.0.0,
                  Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="c:\temp\sampleLogFile.xml" />
        <add name="Console"
            type="System.Diagnostics.ConsoleTraceListener,
                  System, Version=2.0.0.0,
                  Culture=neutral, PublicKeyToken=b77a5c561934e089"
            initializeData="true" />
      </sharedListeners>
    </system.diagnostics>
</configuration>
```

リスナを再構成するには

1. `<sharedListeners>` セクションで、リスナの `<add>` 要素を見つけます。
2. `type` 属性はリスナの型の名前を表します。この型は TraceListener クラスを継承する必要があります。正しい型が確実に使用されるよう、型名には厳密な名前を使用します。詳細については、後述の「厳密な名前を指定された型を参照するには」を参照してください。

   使用できる型のいくつかを以下に示します。

   - Microsoft.VisualBasic.Logging.FileLogTraceListener リスナ。ファイル ログに書き込みます。
   - System.Diagnostics.EventLogTraceListener リスナ。`initializeData` パラメータで指定された、コンピュータのイベント ログに情報を書き込みます。
   - System.Diagnostics.DelimitedListTraceListener リスナおよび System.Diagnostics.XmlWriterTraceListener リスナ。`initializeData` パラメータで指定されたファイルに書き込みます。
   - System.Diagnostics.ConsoleTraceListener リスナ。コマンド ライン コンソールに書き込みます。

   他の型のログ リスナが情報を書き込む先については、その型のドキュメントを参照してください。
3. アプリケーションは、ログ リスナ オブジェクトを作成するときに、コンストラクタのパラメータとして `initializeData` 属性を渡します。`initializeData` 属性の意味はトレース リスナによって異なります。
4. アプリケーションは、ログ リスナを作成した後で、リスナのプロパティを設定します。これらのプロパティは、`<add>` 要素の他の属性で定義されています。特定のリスナのプロパティの詳細については、そのリスナの型のドキュメントを参照してください。

厳密な名前を指定された型を参照するには

1. ログ リスナとして正しい型を確実に使用するために、完全修飾型名と厳密な名前のアセンブリ名を使用する必要があります。厳密な名前を指定された型の構文は次のとおりです。

   <type name>, *<assembly name>*, *<version number>*, *<culture>*, *<strong name>*
2. 次のコード例は、完全修飾された型の厳密な名前を確認する方法を示します。この例では "System.Diagnostics.FileLogTraceListener" です。

   VB

```
Public Sub DisplayStrongName()
    Dim t As Type = GetType(Logging.FileLogTraceListener)
    MsgBox(t.FullName & ", " & t.Assembly.FullName)
End Sub
```

   次に示すのが出力です。これを使用して、前述の「リスナを追加するには」の手順で示した方法で、厳密な名前を指定された型を一意に参照できます。

```
Microsoft.VisualBasic.Logging.FileLogTraceListener, Microsoft.VisualBasic, Version=8.0.0.0,
```

```
    Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
```

参照

処理手順

方法 : イベント情報をテキスト ファイルに書き込む

方法 : アプリケーション イベント ログに書き込む

関連項目

My.Application.Log オブジェクト

My.Log オブジェクト

TraceListener

Microsoft.VisualBasic.Logging.FileLogTraceListener

System.Diagnostics.EventLogTraceListener

チュートリアル：My.Application.Log の出力のフィルタ処理

このチュートリアルでは、**My.Application.Log** オブジェクトの既定のログ フィルタ処理を変更する方法を説明します。これにより、**Log** オブジェクトからリスナに渡される情報やリスナによって書き込まれる情報を制御できます。構成情報はアプリケーションの構成ファイルに格納されているため、ログの動作は、アプリケーションをビルドした後でも変更できます。

概要

My.Application.Log が書き込む各メッセージには、重大度レベルが関連付けられています。フィルタ処理では、この重大度レベルを使用してログ出力が制御されるしくみになっています。このサンプル アプリケーションでは、**My.Application.Log** のメソッドを使用して、重大度レベルが異なるいくつかのログ メッセージを書き込みます。

サンプル アプリケーションをビルドするには

1. 新しい Visual Basic Windows アプリケーション プロジェクトを開きます。
2. Button1 という名前のボタンを Form1 に追加します。
3. Button1 の Click イベント ハンドラに次のコードを追加します。

   VB

   ```
   ' Activity tracing information
   My.Application.Log.WriteEntry("Entering Button1_Click", TraceEventType.Start)

   ' Tracing information
   My.Application.Log.WriteEntry("In Button1_Click", TraceEventType.Information)

   ' Create an exception to log.
   Dim ex As New ApplicationException
   ' Exception information
   My.Application.Log.WriteException(ex)

   ' Activity tracing information
   My.Application.Log.WriteEntry("Leaving Button1_Click", TraceEventType.Stop)
   ```

4. アプリケーションをデバッガで実行します。
5. Button1 をクリックします。

   アプリケーションのデバッグ出力およびログ ファイルに次の情報が書き込まれます。

   ```
   DefaultSource Information: 0 : In Button1_Click

   DefaultSource Error: 2 : Error in the application.
   ```

6. アプリケーションを閉じます。

アプリケーションのデバッグ出力ウィンドウを表示する方法については、「[出力] ウィンドウ」を参照してください。アプリケーションのログ ファイルの場所については、「チュートリアル : My.Application.Log による情報の書き込み先の確認」を参照してください。

メモ：
既定では、アプリケーションは、アプリケーションの終了時にログ ファイル出力をフラッシュします。

上の例では、2 番目の WriteEntry メソッド (My.Application.Log and My.Log) の呼び出しと WriteException メソッド (My.Application.Log および My.Log) の呼び出しではログ出力が生成され、最初と最後の **WriteEntry** メソッドの呼び出しでは生成されません。その理由は、**WriteEntry** と **WriteException** の重大度レベルは "Information" と "Error" であり、いずれも **My.Application.Log** オブジェクトの既定のログ フィルタ処理を通過するからです。一方、重大度レベルが "Start" と "Stop" のイベントは、ログ出力は生成されません。

すべての My.Application.Log リスナのフィルタ処理

My.Application.Log オブジェクトは、**WriteEntry** メソッドおよび **WriteException** メソッドからのどのメッセージをログ リスナに渡すかを、`DefaultSwitch` という名前の SourceSwitch を使用して制御します。`DefaultSwitch` を構成するには、アプリケーションの構成ファイルで、その値をいずれかの SourceLevels 列挙値に設定します。既定では、この値は "Information" です。

この表は、所定の `DefaultSwitch` 設定に応じて、Log がリスナにメッセージを書き込むのに必要な重大度レベルを示します。

DefaultSwitch 値	**メッセージの出力に必要な重大度レベル**
Critical	Critical
Error	**Critical** または Error
Warning	**Critical**、**Error**、または Warning
Information	**Critical**、**Error**、**Warning**、または Information
Verbose	**Critical**、**Error**、**Warning**、**Information**、または Verbose
ActivityTracing	Start、Stop、Suspend、Resume、または Transfer
All	すべてのメッセージが通過する
Off	すべてのメッセージがブロックされる

メモ：

WriteEntry メソッドと **WriteException** メソッドには、重大度レベルを指定しないオーバーロードがそれぞれあります。暗黙の重大度レベルは、**WriteEntry** のオーバーロードでは "Information"、**WriteException** のオーバーロードでは "Error" です。

この表を、前の例で示したログ出力に照らしてみると、既定の `DefaultSwitch` 設定である "Information" では **WriteEntry** メソッドの 2 番目の呼び出しと **WriteException** メソッドの呼び出しだけでログ出力が生成されるということの説明が付きます。

処理トレース イベントのみをログに記録するには

1. ソリューション エクスプローラで app.config を右クリックし、[開く] をクリックします。

   または

   app.config ファイルがない場合は、次の操作を行います。

   a. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

   b. [新しい項目の追加] ダイアログ ボックスで、[アプリケーション構成ファイル] をクリックします。

   c. [追加] をクリックします。

2. 最上位の `<configuration>` セクション内の `<system.diagnostics>` セクションで、`<switches>` セクションを見つけます。

3. スイッチのコレクションに `DefaultSwitch` を追加している要素を見つけます。次のような要素です。

   ```
   <add name="DefaultSwitch" value="Information" />
   ```

4. `value` 属性の値を "ActivityTracing" に変更します。

5. app.config ファイルの内容は次の XML のようになります。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- This section configures My.Application.Log -->
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <add name="FileLog"/>
```

```
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="ActivityTracing" />
    </switches>
    <sharedListeners>
      <add name="FileLog"
           type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
                 Microsoft.VisualBasic, Version=8.0.0.0,
                 Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a,
                 processorArchitecture=MSIL"
           initializeData="FileLogWriter"/>
    </sharedListeners>
  </system.diagnostics>
</configuration>
```

6. アプリケーションをデバッガで実行します。
7. Button1 をクリックします。

   アプリケーションのデバッグ出力およびログ ファイルに次の情報が書き込まれます。

   ```
   DefaultSource Start: 4 : Entering Button1_Click

   DefaultSource Stop: 5 : Leaving Button1_Click
   ```

8. アプリケーションを閉じます。
9. `value` 属性の値を "Information" に戻します。

メモ：

`DefaultSwitch` スイッチの設定で制御されるのは **My.Application.Log** のみです。.NET Framework の System.Diagnostics.Trace クラスと System.Diagnostics.Debug クラスの動作は変更されません。

My.Application.Log リスナの個別のフィルタ処理

前の例では、**My.Application.Log** のすべての出力のフィルタ処理を変更する方法を示しました。この例では、個別のログ リスナをフィルタ処理する方法を説明します。既定では、アプリケーションには、アプリケーションのデバッグ出力に書き込むリスナとログ ファイルに書き込むリスナの 2 つがあります。

構成ファイルでは、それぞれのログ リスナにフィルタを設定でき、それによりログ リスナの動作を制御します。このフィルタは、**My.Application.Log** のスイッチと同様のものです。ログ リスナは、ログの `DefaultSwitch` とログ リスナのフィルタの両方で認められている重大度レベルのメッセージのみを出力します。

この例では、新しいデバッグ リスナのフィルタ処理を構成し、**Log** オブジェクトに追加する方法を説明します。この新しいデバッグ リスナからデバッグ メッセージが来たということが明確になるように、既定のデバッグ リスナは **Log** オブジェクトから削除する必要があります。

処理トレース イベントのみをログに記録するには

1. ソリューション エクスプローラで app.config を右クリックし、[開く] をクリックします。

   または

   app.config ファイルがない場合は、次の操作を行います。

   a. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

   b. [新しい項目の追加] ダイアログ ボックスで、[アプリケーション構成ファイル] をクリックします。

   c. [追加] をクリックします。

2. ソリューション エクスプローラで、app.config を右クリックします。[開く] をクリックします。
3. `<sources>` セクション内にある、`name` 属性が "DefaultSource" の `<source>` セクションで、`<listeners>` セクションを見つけます。`<sources>` セクションは、最上位の `<configuration>` セクション内の `<system.diagnostics>` セクションにあります。

4. <listeners> セクションに次の要素を追加します。

```
<!-- Remove the default debug listener. -->
<remove name="Default"/>
<!-- Add a filterable debug listener. -->
<add name="NewDefault"/>
```

5. 最上位の <configuration> セクション内の <system.diagnostics> セクションで、<sharedListeners> セクションを見つけます。

6. その <sharedListeners> セクションに次の要素を追加します。

```
<add name="NewDefault"
     type="System.Diagnostics.DefaultTraceListener,
           System, Version=2.0.0.0, Culture=neutral,
           PublicKeyToken=b77a5c561934e089,
           processorArchitecture=MSIL">
    <filter type="System.Diagnostics.EventTypeFilter"
            initializeData="Error" />
</add>
```

EventTypeFilter フィルタの initializeData 属性は、いずれかの **SourceLevels** 列挙値をとります。

7. app.config ファイルの内容は次の XML のようになります。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- This section configures My.Application.Log -->
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <add name="FileLog"/>
          <!-- Remove the default debug listener. -->
          <remove name="Default"/>
          <!-- Add a filterable debug listener. -->
          <add name="NewDefault"/>
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="Information" />
    </switches>
    <sharedListeners>
      <add name="FileLog"
           type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
                 Microsoft.VisualBasic, Version=8.0.0.0,
                 Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a,
                 processorArchitecture=MSIL"
           initializeData="FileLogWriter"/>
      <add name="NewDefault"
           type="System.Diagnostics.DefaultTraceListener,
                 System, Version=2.0.0.0, Culture=neutral,
                 PublicKeyToken=b77a5c561934e089,
                 processorArchitecture=MSIL">
        <filter type="System.Diagnostics.EventTypeFilter"
                initializeData="Error" />
      </add>
    </sharedListeners>
```

```
    </system.diagnostics>
  </configuration>
```

8. アプリケーションをデバッガで実行します。
9. Button1 をクリックします。

   アプリケーションのログ ファイルに次の情報が書き込まれます。

   ```
   Default Information: 0 : In Button1_Click

   Default Error: 2 : Error in the application.
   ```

   フィルタ処理での絞り込みが多くなった分、アプリケーションがデバッグ出力に書き込む情報は少なくなります。

   ```
   Default Error 2 Error
   ```

10. アプリケーションを閉じます。

配置の後でログ設定を変更する方法の詳細については、「Visual Basic でのアプリケーション ログの使用」を参照してください。

参照

処理手順

チュートリアル : My.Application.Log による情報の書き込み先の確認
チュートリアル : My.Application.Log による情報の書き込み先の変更
チュートリアル : カスタム ログ リスナの作成
方法 : ログ メッセージを書き込む

概念

トレース スイッチ
アプリケーションからの情報のログ記録

チュートリアル：My.Application.Log の出力をオフにする

このチュートリアルでは、**My.Application.Log** オブジェクトの既定のログ フィルタ処理をオフにする方法を説明します。構成情報はアプリケーションの構成ファイルに格納されているため、ログの動作は、アプリケーションをビルドした後でも変更できます。

概要

My.Application.Log オブジェクトは、自らが取得した各メッセージをログ リスナに渡します。このサンプル アプリケーションでは、**My.Application.Log.WriteEntry** メソッドを使用して、リスナにメッセージを書き込みます。

サンプル アプリケーションをビルドするには

1. 新しい Visual Basic Windows アプリケーション プロジェクトを開きます。
2. Button1 という名前のボタンを Form1 に追加します。
3. Button1 の Click イベント ハンドラに次のコードを追加します。

   VB

   ```
   My.Application.Log.WriteEntry("Log entry")
   ```

4. アプリケーションをデバッガで実行します。
5. Button1 をクリックします。

   アプリケーションのデバッグ出力およびログ ファイルに次の情報が書き込まれます。

   ```
   DefaultSource Information: 0 : Log entry
   ```

6. アプリケーションを閉じます。

アプリケーションのデバッグ出力ウィンドウを表示する方法については、「[出力] ウィンドウ」を参照してください。アプリケーションのログ ファイルの場所については、「チュートリアル : My.Application.Log による情報の書き込み先の確認」を参照してください。

My.Application.Log からのリスナの削除

既定では、アプリケーションには、アプリケーションのデバッグ出力に書き込むリスナとログ ファイルに書き込むリスナの 2 つがあります。この例では、それらのリスナを削除する方法を説明します。

Log オブジェクトからログ リスナを削除するには

1. ソリューション エクスプローラで app.config を右クリックし、[開く] をクリックします。

   または

   app.config ファイルがない場合は、次の操作を行います。

   a. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

   b. [新しい項目の追加] ボックスで、[アプリケーション構成ファイル] をクリックします。

   c. [追加] をクリックします。

2. `name` 属性が `"DefaultSource"`. の `<source>` セクションで、`<listeners>` セクションを見つけます。

   これらのログ構成セクションは、構成ファイルのメインの `<configuration>` ノードの `<system.diagnostics>` ノードにあります。DefaultSource の XML は `<sources>` ノードにあります。

3. "FileLog" という `name` 属性を持つ `<add>` 要素を削除します。次のような要素です。

   ```
   <add name="FileLog"/>
   ```

4. `<listeners>` セクションに次の要素を追加します。

   ```
   <!-- Remove the default debug listener. -->
   ```

```
<remove name="Default"/>
```

5. app.config ファイルの内容は次の XML のようになります。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <!-- This section configures My.Application.Log -->
      <source name="DefaultSource" switchName="DefaultSwitch">
        <listeners>
          <!-- Remove the default debug listener. -->
          <remove name="Default"/>
        </listeners>
      </source>
    </sources>
    <switches>
      <add name="DefaultSwitch" value="Information" />
    </switches>
  </system.diagnostics>
</configuration>
```

6. アプリケーションをデバッガで実行します。

7. Button1 をクリックします。

   アプリケーションのログ ファイルやデバッグ出力に情報は書き込まれません。

配置の後でログ設定を変更する方法の詳細については、「Visual Basic でのアプリケーション ログの使用」を参照してください。

参照

処理手順

チュートリアル : My.Application.Log による情報の書き込み先の確認
チュートリアル : My.Application.Log による情報の書き込み先の変更
チュートリアル : My.Application.Log の出力のフィルタ処理
チュートリアル : カスタム ログ リスナの作成
方法 : ログ メッセージを書き込む

概念

アプリケーションからの情報のログ記録

チュートリアル：カスタム ログ リスナの作成

このチュートリアルでは、カスタム ログ リスナを作成する方法と、**My.Application.Log** オブジェクトの出力を待機するように構成する方法を説明します。

概要

ログ リスナは TraceListener クラスを継承する必要があります。

リスナを作成するには

- アプリケーションで、`SimpleListener` という名前のクラスを、**TraceListener** を継承して作成します。

  VB

```
Public Class SimpleListener
    Inherits System.Diagnostics.TraceListener

    <Security.Permissions.HostProtection(Synchronization:=True)> _
    Public Overloads Overrides Sub Write(ByVal message As String)
        MsgBox("Write: " & message)
    End Sub

    <Security.Permissions.HostProtection(Synchronization:=True)> _
    Public Overloads Overrides Sub WriteLine(ByVal message As String)
        MsgBox("WriteLine: " & message)
    End Sub
End Class
```

  基本クラスのために必要な Write メソッドと WriteLine メソッドでは、**MsgBox** を呼び出してその入力を表示します。

  HostProtectionAttribute 属性を **Write** メソッドと **WriteLine** メソッドに適用し、基本クラスのメソッドと属性を一致させます。**HostProtectionAttribute** 属性により、コードを実行するホストは、そのコードがホスト保護の同期を示していることを確認できます。

> **メモ：**
>
> **HostProtectionAttribute** 属性が有効なのは、共通言語ランタイムをホストし、ホスト保護を実装するアンマネージ アプリケーションにおいてのみです (SQL Server など)。

このログ リスナを **My.Application.Log** に確実に使用させるためには、ログ リスナが属するアセンブリに厳密な名前を指定する必要があります。

次に示すのは、厳密な名前を指定したログ リスナ アセンブリを作成する手順の概要です。詳細については、「厳密な名前付きアセンブリの作成と使用」を参照してください。

ログ リスナ アセンブリに厳密な名前を指定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [署名] タブをクリックします。
3. [アセンブリの署名] ボックスを選択します。
4. [厳密な名前のキー ファイルを選択してください] ボックスの一覧の [<新規作成>] を選択します。

   [厳密な名前キーの作成] ダイアログ ボックスが表示されます。
5. [キー ファイル] ボックスでキー ファイルの名前を指定します。
6. [パスワードの入力] ボックスと [パスワードの確認] ボックスにパスワードを入力します。

7. [OK] をクリックします。
8. アプリケーションをビルドし直します。

リスナの追加

アセンブリに厳密な名前を付けたので、**My.Application.Log** がログ リスナを使用するように、リスナの厳密な名前を確認する必要があります。

厳密な名前を指定された型の形式は次のとおりです。

<型名>, <アセンブリ名>, <バージョン番号>, <カルチャ>, <厳密な名前>

リスナの厳密な名前を確認するには

- 次のコードは、厳密な名前を指定された `SimpleListener` の型名を確認する方法を示しています。

  VB

  ```
  Public Sub DisplaySimpleListenerStrongName()
      Dim t As Type = GetType(SimpleListener)
      MsgBox(t.FullName & ", " & t.Assembly.FullName)
  End Sub
  ```

  型の厳密な名前はプロジェクトによって異なります。

厳密な名前を確認できたら、このリスナを **My.Application.Log** のログ リスナのコレクションに追加できます。

リスナを My.Application.Log に追加するには

1. ソリューション エクスプローラで app.config を右クリックし、[開く] をクリックします。

   または

   app.config ファイルがある場合は、次の操作を行います。

   a. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

   b. [新しい項目の追加] ダイアログ ボックスで、[アプリケーション構成ファイル] をクリックします。

   c. [追加] をクリックします。

2. `<sources>` セクション内にある、`name` 属性が "DefaultSource" の `<source>` セクションで、`<listeners>` セクションを見つけます。`<sources>` セクションは、最上位の `<configuration>` セクション内の `<system.diagnostics>` セクションにあります。

3. `<listeners>` セクションに次の要素を追加します。

   ```
   <add name="SimpleLog" />
   ```

4. 最上位の `<configuration>` セクション内の `<system.diagnostics>` セクションで、`<sharedListeners>` セクションを見つけます。

5. その `<sharedListeners>` セクションに次の要素を追加します。

   ```
   <add name="SimpleLog" type="SimpleLogStrongName" />
   ```

   `SimpleLogStrongName` の値をリスナの厳密な名前に変更します。

参照

処理手順

方法 : Visual Basic で例外をログに記録する

方法 : ログ メッセージを書き込む

チュートリアル : My.Application.Log による情報の書き込み先の変更

関連項目

My.Application.Log オブジェクト

My.Log オブジェクト

概念

Visual Basic でのアプリケーション ログの使用

トラブルシューティング：ログ リスナ

My.Application.Log オブジェクトおよび **My.Log** オブジェクトを使用すると、アプリケーション内で発生したイベントに関する情報をログに記録できます。

それらのメッセージを受け取るログ リスナを確認する方法については、「チュートリアル : My.Application.Log による情報の書き込み先の確認」を参照してください。

Log オブジェクトでは、ログ フィルタ処理を使用して、ログに記録する情報の量を制限できます。フィルタの構成が適切でない場合には、ログに誤った情報が記録されることがあります。フィルタ処理の詳細については、「チュートリアル : My.Application.Log の出力のフィルタ処理」を参照してください。

一方、ログの構成が適切でない場合には、現在のログの構成について、もっと詳しい情報が必要なことがあります。この情報は、ログの高度なプロパティである **TraceSource** プロパティで取得できます。

Log オブジェクトに対応するログ リスナをコードで確認するには

1. コード ファイルの先頭で System.Diagnostics 名前空間をインポートします。詳細については、「Imports ステートメント」を参照してください。

   VB

   ```
   Imports System.Diagnostics
   ```

2. ログの各リスナに対応する情報で構成される文字列を返す関数を作成します。

   VB

   ```
   Function GetListeners(ByVal listeners As TraceListenerCollection) As String
       Dim ret As String = ""
       For Each listener As TraceListener In listeners
           ret &= listener.Name
           Dim listenerType As Type = listener.GetType
           If listenerType Is GetType(DefaultTraceListener) Then
               Dim tmp As DefaultTraceListener = _
                   DirectCast(listener, DefaultTraceListener)
               ret &= ": Writes to the debug output."
           ElseIf listenerType Is GetType(Logging.FileLogTraceListener) Then
               Dim tmp As Logging.FileLogTraceListener = _
                   DirectCast(listener, Logging.FileLogTraceListener)
               ret &= ": Log filename: " & tmp.FullLogFileName
           ElseIf listenerType Is GetType(EventLogTraceListener) Then
               Dim tmp As EventLogTraceListener = _
                   DirectCast(listener, EventLogTraceListener)
               ret &= ": Event log name: " & tmp.EventLog.Log
           ElseIf listenerType Is GetType(XmlWriterTraceListener) Then
               Dim tmp As Diagnostics.XmlWriterTraceListener = _
                   DirectCast(listener, XmlWriterTraceListener)
               ret &= ": XML log"
           ElseIf listenerType Is GetType(ConsoleTraceListener) Then
               Dim tmp As ConsoleTraceListener = _
                   DirectCast(listener, ConsoleTraceListener)
               ret &= ": Console log"
           ElseIf listenerType Is GetType(DelimitedListTraceListener) Then
               Dim tmp As DelimitedListTraceListener = _
                   DirectCast(listener, DelimitedListTraceListener)
               ret &= ": Delimited log"
           Else
               ret &= ": Unhandeled log type: " & _
   ```

```
                listenerType.ToString
            End If
            ret &= vbCrLf
        Next

        Return ret
    End Function
```

3. ログのトレース リスナのコレクションを `GetListeners` 関数に渡し、戻り値を表示します。

   VB

```
Dim ListenerCollection As TraceListenerCollection
ListenerCollection = My.Application.Log.TraceSource.Listeners
Dim ListenersText As String = GetListeners(ListenerCollection)
MsgBox(ListenersText)
```

   詳細については、「TraceSource プロパティ (My.Application.Log and My.Log)」を参照してください。

参照

処理手順

チュートリアル : My.Application.Log による情報の書き込み先の確認

関連項目

My.Application.Log オブジェクト

My.Log オブジェクト

概念

Visual Basic でのアプリケーション ログの使用

Visual Basic での実行中のアプリケーションへのアクセス

このセクションには、Visual Basic アプリケーション モデルを使用してアプリケーションの実行時の動作を制御する方法、およびアプリケーションに関する実行時の情報を取得する方法についてのトピックが含まれています。

My.Application オブジェクトには、現在のアプリケーションに関するプロパティ、メソッド、およびイベントが用意されています。これを使用すると、アプリケーションについての情報を取得できます。たとえば、現在のカルチャ、アプリケーションが配置された方法、環境変数などです。**My.Application** オブジェクトが公開する Visual Basic アプリケーション モデルを使用すると、アプリケーションが起動および終了するときにコードを実行できます。また、**My.Application.Deployment** プロパティでは、現在のアプリケーションの ClickOnce 配置オブジェクトも同様に提供されます。

処理手順

目的	参照項目
コードの実行時に他のイベントが発生した場合にアプリケーションでそのイベントを処理できるようにする。	My.Application.DoEvents メソッド
アプリケーションのコマンド ライン引数を判断する。	方法 : コマンド ライン引数にアクセスする (Visual Basic)
アプリケーションの起動時に文字列 "/batch" が引数に指定されたどうかをチェックする。	方法 : Windows フォーム アプリケーションのバッチ モードを有効にする
アプリケーションでキャッチしていない例外を処理する	My.Application.UnhandledException イベント
Visual Basic アプリケーション モデルに用意されているイベントを使用して、アプリケーションが起動または終了するときにコードを実行する。	方法 : アプリケーションの起動時または終了時にコードを実行する
単一インスタンス アプリケーションが二重起動されたときのコマンド ライン引数をチェックする。	My.Application.StartupNextInstance イベント
ClickOnce アプリケーションの最新バージョンをダウンロードおよびインストールする。	方法 : ClickOnce アプリケーションの更新をダウンロードする

参照

処理手順

方法 : アセンブリ情報を指定する

関連項目

My.Application オブジェクト
My.Application.Log オブジェクト
My.Application.Info オブジェクト

概念

Visual Basic アプリケーション モデルの概要
アプリケーションからの情報のログ記録

方法：アプリケーションの起動時または終了時にコードを実行する

Visual Basic アプリケーション モデルに用意されているイベントを使用して、アプリケーションが起動または終了するときにコードを実行できます。アプリケーションのイベント ハンドラのコードには、プロジェクト デザイナでアクセスできます。

メモ：

Visual Basic アプリケーション モデルは、Windows フォーム アプリケーションでのみ使用できます。詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

アプリケーションのイベント ハンドラ コードにアクセスするには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [アプリケーション イベントの表示] をクリックしてコード エディタを開きます。

   ApplicationEvents.vb ファイルが開かれます。

アプリケーションの起動時にコードを実行するには

1. コード エディタで ApplicationEvents.vb ファイルを開きます。[全般] メニューの [MyApplication イベント] をクリックします。
2. [宣言] メニューの [スタートアップ] をクリックします。

   アプリケーションでは、メイン アプリケーションの実行前に Startup イベントが発生します。
3. 単一インスタンス アプリケーションの場合には、二重起動の処理をすることができます。それには、[全般] メニューの [MyApplication イベント] をクリックします。
4. [宣言] メニューの [StartupNextInstance] をクリックします。

   単一インスタンス アプリケーションの場合、同じアプリケーションのインスタンスが二重起動されたときに、最初のインスタンスで StartupNextInstance イベントが発生します。二重起動されたインスタンス自身は、イベントは発生せずに閉じます。

アプリケーションの終了時にコードを実行するには

1. コード エディタで ApplicationEvents.vb ファイルを開きます。[全般] メニューの [MyApplication イベント] をクリックします。
2. [宣言] メニューの [シャットダウン] をクリックします。

   アプリケーションでは、メイン アプリケーションが実行された後、終了される前の段階で、Shutdown イベントが発生します。
3. [全般] メニューの [MyApplication イベント] をクリックします。
4. [宣言] メニューの [UnhandledException] をクリックします。

   アプリケーションで未処理の例外が発生した場合、UnhandledException イベントが発生します。**UnhandledException** イベントの後では **Shutdown** イベントは発生しないので、**Shutdown** ハンドラで呼び出すのと同じ終了処理のコードを **UnhandledException** ハンドラで呼び出すことができます。

参照

処理手順

方法 : アプリケーション イベントを処理する (Visual Basic)

概念

Visual Basic アプリケーション モデルの概要

方法：Windows フォーム アプリケーションのバッチ モードを有効にする

この例では、**My.Application.Startup** イベントを使用して、アプリケーションの起動時に文字列 `/batch` が引数に指定されたかどうかをチェックします。

Windows フォーム アプリケーションのバッチモードを有効にするには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブの [アプリケーション イベントの表示] をクリックして、コード エディタを開きます。
3. My.Application.Startup イベントを処理するメソッドを作成します。詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

   VB

   ```
   Private Sub MyApplication_Startup( _
       ByVal sender As Object, _
       ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs _
   ) Handles Me.Startup
   ```

   VB

   ```
   End Sub
   ```

4. アプリケーションの各コマンド ライン引数に対して反復処理を行い、いずれかの引数が `/batch` の場合に、`e` オブジェクトの **Cancel** プロパティを **True** に設定します。

   Cancel プロパティが **True** に設定されている場合、スタートアップ フォームは開始されません。

   VB

   ```
   For Each s As String In My.Application.CommandLineArgs
       If s.ToLower = "/batch" Then
           ' Stop the start form from loading.
           e.Cancel = True
       End If
   Next
   ```

5. `e` オブジェクトの **Cancel** プロパティが **True** に設定されている場合、ウィンドウなしの操作用のメイン ルーチンを呼び出します。

   VB

   ```
   If e.Cancel Then
       ' Call the main routine for windowless operation.
       Dim c As New BatchApplication
       c.Main()
   End If
   ```

使用例

VB

```
Private Sub MyApplication_Startup( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs _
) Handles Me.Startup
    For Each s As String In My.Application.CommandLineArgs
```

```
            If s.ToLower = "/batch" Then
                ' Stop the start form from loading.
                e.Cancel = True
            End If
        Next
        If e.Cancel Then
            ' Call the main routine for windowless operation.
            Dim c As New BatchApplication
            c.Main()
        End If
    End Sub
    Class BatchApplication
        Sub Main()
            ' Insert code to run without a graphical user interface.
        End Sub
    End Class
```

参照

処理手順

方法 : コマンド ライン引数にアクセスする (Visual Basic)

関連項目

My.Application オブジェクト

My.Application.CommandLineArgs プロパティ

概念

Visual Basic アプリケーション モデルの概要

方法 : コマンド ライン引数にアクセスする (Visual Basic)

次のコード例は、アプリケーションの各コマンド ライン引数をループします。

使用例

VB

```
For Each argument As String In My.Application.CommandLineArgs
    ' Add code here to use the argument.
Next
```

参照

処理手順

方法 : Windows フォーム アプリケーションのバッチ モードを有効にする

関連項目

My.Application.CommandLineArgs プロパティ

Visual Basic プログラムの構造

ユーザー データへのアクセス

このセクションのトピックでは、**My.User** オブジェクトと、これを使用して実行できるタスクについて取り上げます。

My.User オブジェクトを使用すると、IPrincipal インターフェイスを実装するオブジェクトを取得でき、ログオン ユーザーについての情報にアクセスできます。

処理手順

目的	参照項目
ユーザーのログイン名を取得する。	方法 : ユーザーのログイン名を確認する
アプリケーションが Windows 認証を使用している場合に、ユーザーのドメイン名を取得する。	方法 : ユーザーのドメインを確認する
ユーザーのロールを判断する。	方法 : ユーザーがグループに属しているかどうかを確認する
カスタムの認証を実装する。	チュートリアル : カスタムの認証および承認の実装

参照

関連項目

My.User オブジェクト

その他の技術情報

Visual Basic での .NET Framework の認証と承認

方法：ユーザーのログイン名を確認する

My.User オブジェクトを使用すると、現在のユーザーに関する情報を取得できます。この例では、**My.User.Name** プロパティを使用してユーザーのログイン名を取得する方法を示します。

アプリケーションは、既定では Windows 認証を使用するため、**My.User** は、アプリケーションを起動したユーザーについての Windows 情報を返します。

使用例

この例では、アプリケーションが Windows 認証またはカスタム認証を使用しているかをチェックし、その情報を使用して **My.User.Name** プロパティを解析します。

VB

```
Function GetUserName() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' The application is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
        Dim parts() As String = Split(My.User.Name, "\")
        Dim username As String = parts(1)
        Return username
    Else
        ' The application is using custom authentication.
        Return My.User.Name
    End If
End Function
```

参照

処理手順

方法 : ユーザーのドメインを確認する

チュートリアル : カスタムの認証および承認の実装

関連項目

My.User.Name プロパティ

概念

ユーザー データへのアクセス

方法：ユーザーがグループに属しているかどうかを確認する

My.User オブジェクトを使用すると、現在のユーザーに関する情報を取得できます。この例では、**My.User.IsInRole** メソッドを使用して、ユーザーが特定のグループのメンバかどうかを確認する方法を示します。

使用例

この例では、リソースにアクセスする前に、**My.User.IsInRole** メソッドを使用して、ユーザーが管理者かどうかを確認します。

VB

```
If My.User.IsInRole( _
        ApplicationServices.BuiltInRole.Administrator) Then
    ' Insert code to access a resource here.
End If
```

参照

処理手順

方法 : ユーザーのドメインを確認する

チュートリアル : カスタムの認証および承認の実装

関連項目

My.User.IsInRole メソッド

概念

ユーザー データへのアクセス

方法：ユーザーのドメインを確認する

My.User オブジェクトを使用すると、現在のユーザーに関する情報を取得できます。この例では、アプリケーションが Windows 認証を使用している場合に、**My.User.Name** プロパティを使用してユーザーのドメイン名を取得する方法を示します。

アプリケーションは、既定では Windows 認証を使用するため、**My.User** は、アプリケーションを起動したユーザーについての Windows 情報を返します。

使用例

この例では、アプリケーションが Windows 認証を使用しているかどうかをチェックしてから、**My.User.Name** プロパティを解析して、ドメイン名を確認します。

この例は、アプリケーションがカスタム認証を使用している場合は空の文字列を返します。カスタム認証の実装からはドメイン情報は必ずしも提供されないためです。

VB

```
Function GetUserDomain() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' My.User is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
        Dim parts() As String = Split(My.User.Name, "\")
        Dim domain As String = parts(0)
        Return domain
    Else
        ' My.User is using custom authentication.
        Return ""
    End If
End Function
```

参照

処理手順

方法 : ユーザーのログイン名を確認する

チュートリアル : カスタムの認証および承認の実装

関連項目

My.User.Name プロパティ

概念

ユーザー データへのアクセス

アプリケーション フォームへのアクセス

My.Forms オブジェクトを使用すると、アプリケーションのプロジェクトで宣言されている各 Windows フォームのインスタンスに簡単にアクセスできます。また、**My.Application** オブジェクトのプロパティを使用して、アプリケーションのスプラッシュ スクリーンやメイン フォームにアクセスしたり、アプリケーションで開いているフォームのリストを取得したりできます。

処理手順

次の表は、アプリケーションのフォームへのアクセス方法を示す例の一覧です。

目的	参照項目
アプリケーション内で、あるフォームから別のフォームにアクセスする。	方法 : アプリケーション内のフォーム間でやり取りする
アプリケーションで開いているすべてのフォームのタイトルを表示する。	方法 : アプリケーションで開いているすべてのフォームにアクセスする
アプリケーションの起動時にステータス情報でスプラッシュ スクリーンを更新する。	My.Application.SplashScreen プロパティ

参照

関連項目

My.Forms オブジェクト
My.Application.OpenForms プロパティ
My.Application.SplashScreen プロパティ

方法：アプリケーション内のフォーム間でやり取りする

この例では、**My.Forms** オブジェクトを使用して、あるフォームから別のフォームにアクセスする方法を示します。

My.Forms オブジェクトでは、アプリケーションのプロジェクトで宣言されている Windows アプリケーションの各フォームのインスタンスにアクセスできます。これにより、あるフォームのコードで別のフォームとやり取りできます。

別のフォームと通信できるフォームにコードを作成するには

1. 新しい Windows アプリケーションを作成します。

   メイン フォームの既定の名前は Form1 です。

   詳細については、「方法 : Windows アプリケーション プロジェクトを作成する」を参照してください。

2. 別のフォームを追加します。名前は Form2 とします。

   詳細については、「方法 : プロジェクトに Windows フォームを追加する」を参照してください。

3. アプリケーション デザイナで、`Button1` という名前のボタンを Form1 に追加します。

4. Button1 をダブルクリックします。

5. `Button1` の Click イベントのイベント ハンドラに次のコードを追加します。

   VB

   ```
   My.Forms.Form2.Text = Now.ToString
   My.Forms.Form2.Show()
   ```

6. アプリケーションを実行します。

7. Button1 をクリックします。

   Form2 が開き、そのタイトルに現在時刻が表示されます。

8. 数秒待機し、Button1 を再度クリックします。

   Form2 は開いたままで、そのタイトルの現在時刻が更新されます。

参照

処理手順

方法 : アプリケーションで開いているすべてのフォームにアクセスする

関連項目

My.Forms オブジェクト

概念

アプリケーション フォームへのアクセス

方法：アプリケーションで開いているすべてのフォームにアクセスする

この例では、**My.Application.OpenForms** プロパティを使用して、アプリケーションで開いているすべてのフォームのタイトルを表示します。

My.Application.OpenForms プロパティは、どのスレッドが開いたフォームかにかかわらず、現在開いているすべてのフォームを返します。したがって、各フォームにアクセスする前に、その InvokeRequired プロパティをチェックする必要があります。これを行わない場合は、InvalidOperationException 例外がスローされる可能性があります。

この例では、各フォームのタイトルをスレッド セーフな方法で取得する関数を宣言します。まず、フォームの **InvokeRequired** プロパティをチェックし、必要がある場合は BeginInvoke メソッドを使用して、関数をそのフォームのスレッドで実行します。次に、関数はフォームのタイトルを返します。

使用例

この例では、アプリケーションで開いている各フォームをループし、それぞれのタイトルを ListBox コントロールに表示します。コードを単純にして、現在のスレッドから直接アクセスできるフォームのみを表示する方法については、「My.Application.OpenForms プロパティ」を参照してください。

VB

```
Private Sub GetOpenFormTitles()
    Dim formTitles As New Collection

    Try
        For Each f As Form In My.Application.OpenForms
            ' Use a thread-safe method to get all form titles.
            formTitles.Add(GetFormTitle(f))
        Next
    Catch ex As Exception
        formTitles.Add("Error: " & ex.Message)
    End Try

    Form1.ListBox1.DataSource = formTitles
End Sub

Private Delegate Function GetFormTitleDelegate(ByVal f As Form) As String
Private Function GetFormTitle(ByVal f As Form) As String
    ' Check if the form can be accessed from the current thread.
    If Not f.InvokeRequired Then
        ' Access the form directly.
        Return f.Text
    Else
        ' Marshal to the thread that owns the form.
        Dim del As GetFormTitleDelegate = AddressOf GetFormTitle
        Dim param As Object() = {f}
        Dim result As System.IAsyncResult = f.BeginInvoke(del, param)
        ' Give the form's thread a chance process function.
        System.Threading.Thread.Sleep(10)
        ' Check the result.
        If result.IsCompleted Then
            ' Get the function's return value.
            Return "Different thread: " & f.EndInvoke(result).ToString
        Else
            Return "Unresponsive thread"
        End If
    End If
End Function
```

参照

処理手順

方法 : Windows フォーム コントロールのスレッド セーフな呼び出しを行う

関連項目

My.Application.OpenForms プロパティ

アプリケーションの Web サービスへのアクセス

My.WebServices オブジェクトは、現在のプロジェクトから参照される各 Web サービスのインスタンスを表します。各インスタンスは、要求に応じてインスタンス化されます。これらの Web サービスには、**My.WebServices** オブジェクトのプロパティからアクセスできます。プロパティの名前は、そのプロパティでアクセスする Web サービスの名前と同じです。SoapHttpClientProtocol を継承するクラスはすべて Web サービスです。プロジェクトに Web サービスを追加する方法については、「方法 : マネージ コードを使用して XML Web サービスにアクセスする」を参照してください。

処理手順

次の表は、アプリケーションが参照する Web サービスにアクセスする方法を示しています。

目的	参照項目
Web サービスを呼び出す。	My.WebServices オブジェクト
Web サービスを非同期で呼び出し、完了したときにイベントを処理する。	方法 : Web サービスを非同期で呼び出す

参照

関連項目

My.WebServices オブジェクト

方法：Web サービスを非同期で呼び出す

この例では、XML Web サービスの非同期ハンドラ イベントにハンドラを割り当て、非同期メソッド呼び出しの結果を取得します。この例では、http://www.xmethods.net の DemoTemperatureService Web サービスを使用しています。

Visual Studio 統合開発環境 (IDE) では、プロジェクトで Web サービスを参照すると、**My.WebServices** オブジェクトにその Web サービスが追加され、その Web サービスにアクセスするためのクライアント プロキシ クラスが生成されます。

このプロキシ クラスを使用すると、Web サービス メソッドを同期で呼び出すことができます。つまりアプリケーションは、関数が終了するまで待機します。さらに、このプロキシは、メソッドを非同期で呼び出すために使用できる、追加的なメンバを作成します。Web サービスの各関数 (名前が *NameOfWebServiceFunction* だとする) に対して、プロキシは、*NameOfWebServiceFunction***Async** サブルーチン、*NameOfWebServiceFunction***Completed** イベント、および *NameOfWebServiceFunction***CompletedEventArgs** クラスを作成します。この例では、非同期のメンバを使用して、DemoTemperatureService Web サービスの `getTemp` 関数にアクセスする方法を説明します。

メモ：

ASP.NET は **My.WebServices** オブジェクトをサポートしないため、このコードは Web アプリケーションでは動作しません。

Web サービスを非同期で呼び出すには

1. http://www.xmethods.net の DemoTemperatureService Web サービスを参照します。アドレスは次のとおりです。

   ```
   http://www.xmethods.net/sd/2001/DemoTemperatureService.wsdl
   ```

   詳細については、「方法 : マネージ コードを使用して XML Web サービスにアクセスする」を参照してください。

2. `getTempCompleted` イベントのイベント ハンドラを追加します。

   ```
   Private Sub getTempCompletedHandler(ByVal sender As Object, _
       ByVal e As net.xmethods.www.getTempCompletedEventArgs)

       MsgBox("Temperature: " & e.Result)
   End Sub
   ```

   メモ：

   Handles ステートメントを使用して **My.WebServices** オブジェクトのイベントにイベント ハンドラを関連付けることはできません。

3. イベント ハンドラが `getTempCompleted` イベントに追加されているかどうかを追跡するためのフィールドを追加します。

   ```
   Private handlerAttached As Boolean = False
   ```

4. `getTempCompleted` イベントにイベント ハンドラを追加するためのメソッドを必要に応じて追加し、`getTempAsynch` メソッドを呼び出すためのメソッドを追加します。

   ```
   Sub CallGetTempAsync(ByVal zipCode As Integer)
       If Not handlerAttached Then
           AddHandler _
               My.WebServices.TemperatureService.getTempCompleted, _
               AddressOf Me.TS_getTempCompleted
           handlerAttached = True
       End If
       My.WebServices.TemperatureService.getTempAsync(zipCode)
   End Sub
   ```

`getTemp` Web メソッドを非同期で呼び出すには、`CallGetTempAsync` メソッドを呼び出します。Web メソッドが完了すると、その戻り値が `getTempCompletedHandler` イベント ハンドラに渡されます。

参照

処理手順

方法 : マネージ コードを使用して XML Web サービスにアクセスする

関連項目

My.WebServices オブジェクト

概念

アプリケーションの Web サービスへのアクセス

アプリケーション リソースへのアクセス

このセクションには、**My.Resources** オブジェクトに関連するヘルプ ページと、**My.Resources** オブジェクトを使用して実行できるタスクが示されています。

My.Resources オブジェクトのプロパティを使用すると、アプリケーションのリソースに読み取り専用でアクセスできます。

処理手順

次の表は、アプリケーションのリソースへのアクセス方法を示す例の一覧です。

目的	参照項目
オーディオ リソースを取得する。	方法 : Visual Basic でオーディオ リソースを取得する
アイコン リソースを取得する。	方法 : Visual Basic でアイコン リソースを取得する
イメージ リソースを取得する。	方法 : Visual Basic でイメージ リソースを取得する
文字列リソースを取得する。	方法 : Visual Basic で文字列リソースを取得する
ローカライズされたリソースを取得する。	方法 : Visual Basic で、ローカライズされたリソースを取得する

参照

関連項目

My.Resources オブジェクト

その他の技術情報

アプリケーション リソースの管理

方法：Visual Basic でオーディオ リソースを取得する

My.Resources オブジェクトでリソースのプロパティにアクセスすると、アプリケーションのオーディオ リソースを取得できます。

My.Resources オブジェクトは、それぞれのリソースを読み取り専用プロパティとして公開します。プロパティの名前はリソース名と同じです。プロパティの型はリソースの分類によって決定されます。詳細については、「My.Resources オブジェクト」を参照してください。

使用例

この例では、**My.Computer.Audio.Play** メソッドを使用して、アプリケーションのリソース ファイルに格納されている `Form1Greeting` という名前のオーディオ リソースのサウンドを再生します

VB

```
Sub PlayFormGreeting()
    My.Computer.Audio.Play(My.Resources.Form1Greeting, _
        AudioPlayMode.Background)
End Sub
```

この例が動作するためには、アプリケーションのリソース ファイルに `Form1Greeting` という名前のオーディオ リソースが含まれている必要があります。詳細については、「方法 : リソースを追加または削除する」を参照してください。

My.Computer.Audio.Play メソッドは、Windows フォーム アプリケーションでのみ使用できます。詳細については、「My.Computer.Audio.Play メソッド」を参照してください。

参照

処理手順

方法 : Visual Basic で文字列リソースを取得する
方法 : Visual Basic でイメージ リソースを取得する
方法 : Visual Basic でアイコン リソースを取得する
方法 : Visual Basic で、ローカライズされたリソースを取得する

関連項目

My.Resources オブジェクト
My.Computer.Audio.Play メソッド

その他の技術情報

アプリケーション リソースの管理

方法：Visual Basic でアイコン リソースを取得する

My.Resources オブジェクトでリソースのプロパティにアクセスすると、アプリケーションのアイコン リソースを取得できます。

My.Resources オブジェクトは、それぞれのリソースを読み取り専用プロパティとして公開します。プロパティの名前はリソース名と同じです。プロパティの型はリソースの分類によって決定されます。詳細については、「My.Resources オブジェクト」を参照してください。

使用例

この例では、フォームのアイコンを、アプリケーションのリソース ファイルに格納されている `Form1Icon` という名前のアイコンに設定します。

VB

```
Sub SetFormIcon()
    Me.Icon = My.Resources.Form1Icon
End Sub
```

この例が動作するためには、アプリケーションのリソース ファイルに `Form1Icon` という名前のアイコンが含まれている必要があります。詳細については、「方法 : リソースを追加または削除する」を参照してください。

参照

処理手順

方法 : Visual Basic で文字列リソースを取得する
方法 : Visual Basic でイメージ リソースを取得する
方法 : Visual Basic でオーディオ リソースを取得する
方法 : Visual Basic で、ローカライズされたリソースを取得する

関連項目

My.Resources オブジェクト

その他の技術情報

アプリケーション リソースの管理

方法：Visual Basic でイメージ リソースを取得する

My.Resources オブジェクトでリソースのプロパティにアクセスすると、アプリケーションのイメージ リソースを取得できます。

My.Resources オブジェクトは、それぞれのリソースを読み取り専用プロパティとして公開します。プロパティの名前はリソース名と同じです。プロパティの型はリソースの分類によって決定されます。詳細については、「My.Resources オブジェクト」を参照してください。

使用例

この例では、フォームのバックグラウンド イメージを、アプリケーションのリソース ファイルに含まれている `Form1Background` という名前のイメージ リソースに設定します。

VB

```
Sub SetFormBackgroundImage()
    Me.BackgroundImage = My.Resources.Form1Background
End Sub
```

この例が動作するためには、アプリケーションのリソース ファイルに `Form1Background` という名前のイメージ リソースが含まれている必要があります。詳細については、「方法 : リソースを追加または削除する」を参照してください。

参照

処理手順

方法 : Visual Basic で文字列リソースを取得する
方法 : Visual Basic でアイコン リソースを取得する
方法 : Visual Basic でオーディオ リソースを取得する
方法 : Visual Basic で、ローカライズされたリソースを取得する

関連項目

My.Resources オブジェクト

その他の技術情報

アプリケーション リソースの管理

方法：Visual Basic で文字列リソースを取得する

My.Resources オブジェクトでリソースのプロパティにアクセスすると、アプリケーションの文字列リソースを取得できます。

My.Resources オブジェクトは、それぞれのリソースを読み取り専用プロパティとして公開します。プロパティの名前はリソース名と同じです。プロパティの型はリソースの分類によって決定されます。詳細については、「My.Resources オブジェクト」を参照してください。

使用例

この例では、フォームのタイトルを、アプリケーションのリソース ファイルの `Form1Title` という名前の文字列リソースに設定します。

VB

```
Sub SetFormTitle()
    Me.Text = My.Resources.Form1Title
End Sub
```

この例が動作するためには、アプリケーションのリソース ファイルに `Form1Title` という名前の文字列が含まれている必要があります。詳細については、「方法 : リソースを追加または削除する」を参照してください。

参照

処理手順

方法 : Visual Basic でイメージ リソースを取得する
方法 : Visual Basic でアイコン リソースを取得する
方法 : Visual Basic でオーディオ リソースを取得する
方法 : Visual Basic で、ローカライズされたリソースを取得する

関連項目

My.Resources オブジェクト

その他の技術情報

アプリケーション リソースの管理

方法：Visual Basic で、ローカライズされたリソースを取得する

My.Resources オブジェクトを使用すると、アプリケーションを実行中のコンピュータのカルチャ設定に基づいて、ローカライズされたアプリケーション リソースを取得できます (用意されている場合)。コンピュータのカルチャ設定は、My.Application.UICulture プロパティを設定することによりオーバーライドできます。

ランタイムはカルチャの署名、つまり名前によって、ローカライズされたリソースを識別します。**My.Resources** オブジェクトで表示されるプロパティは、プロジェクトの既定のリソース ファイルである Resources.resx により決定されます。ローカライズされたリソースを用意するためには、以下の操作が必要です。

- リソース ファイルをコピーし、Resources.*CultureSignature*.resx に名前を変更します。
- 文字列、およびリソース ファイルが参照するファイルをローカライズします。
- ローカライズされたリソース ファイルをプロジェクトに追加します。

My.Resources オブジェクトは、それぞれのリソースを読み取り専用プロパティとして公開します。プロパティの名前はリソース名と同じです。プロパティの型はリソースの分類によって決定されます。詳細については、「My.Resources オブジェクト」および「アプリケーションのリソース」を参照してください。

各カルチャは一意の名前を持ちます。この名前は、言語と関連付けられた 2 文字の小文字のカルチャ名と、必要に応じて、国または地域と関連付けられた 2 文字の大文字のサブカルチャ名の組み合わせで構成されます。サブカルチャ名がある場合は、カルチャ名の後ろにダッシュ (-) で区切って続けます。たとえば、日本の日本語を表す ja-JP、米国の英語を表す en-US、ドイツのドイツ語を表す de-DE (オーストリアのドイツ語を表す de-AT などのフォールバック カルチャとは異なります) などがあります。カルチャ名の詳細については、「CultureInfo」を参照してください。

使用例

この例では、`Message` という名前の、フランスのカルチャのバージョンのアプリケーションの文字列リソースを取得します。

My.Resources オブジェクトが使用するカルチャを変更するために、この例では My.Application.ChangeUICulture メソッドを使用しています。

VB

```
Sub ShowLocalizedMessage()
    Dim culture As String = My.Application.UICulture.Name
    My.Application.ChangeUICulture("fr-FR")
    MsgBox(My.Resources.Message)
    My.Application.ChangeUICulture(culture)
End Sub
```

この例が動作するためには、アプリケーションのリソース ファイルに `Message` という名前の文字列があり、かつアプリケーションは、そのリソース ファイルのフランスのカルチャのバージョンである Resources.fr-FR.resx を持っている必要があります。詳細については、「方法 : リソースを追加または削除する」を参照してください。

アプリケーションに、このリソース ファイルのフランスのカルチャのバージョンがない場合は、**My.Resource** オブジェクトは、既定のカルチャのリソース ファイルからリソースを取得します。

参照

処理手順

方法 : Visual Basic で文字列リソースを取得する

関連項目

My.Resources オブジェクト

その他の技術情報

アプリケーション リソースの管理

アプリケーション設定へのアクセス

このセクションのトピックでは、**My.Settings** オブジェクトと、それを使用して実現できるタスクについて説明します。

My.Settings

My.Settings オブジェクトのプロパティを使用すると、アプリケーションの設定にアクセスできます。設定を追加または削除するには、プロジェクト デザイナを使用します。詳細については、「方法 : アプリケーション設定を追加または削除する」を参照してください。

My.Settings オブジェクトのメソッドを使用すると、現在のユーザー設定を保存したり、最後に保存したユーザー設定の値を復元したりできます。

処理手順

次の表は、アプリケーションの設定へのアクセス方法を示す例の一覧です。

目的	参照項目
ユーザー設定の値を更新する。	方法 : Visual Basic でユーザー設定を変更する
アプリケーション設定およびユーザー設定をプロパティ グリッドに表示する。	方法 : Visual Basic でユーザー設定のためのプロパティ グリッドを作成する
更新されたユーザー設定の値を保存する。	方法 : Visual Basic でユーザー設定を永続化する
ユーザー設定の値を確認する。	方法 : Visual Basic でアプリケーション設定を読み取る

参照

関連項目

My.Settings オブジェクト

その他の技術情報

アプリケーションの設定の管理

方法：Visual Basic でユーザー設定のためのプロパティ グリッドを作成する

PropertyGrid コントロールに対して、**My.Settings** オブジェクトのユーザー設定プロパティを設定すると、ユーザー設定のためのプロパティ グリッドを作成できます。

> **メモ：**
>
> この例が動作するためには、アプリケーションでユーザー設定が構成されている必要があります。ユーザー設定の追加については、「方法：アプリケーション設定を追加または削除する」を参照してください。

My.Settings オブジェクトでは、各設定がプロパティとして公開されています。プロパティの名前は設定の名前と同じで、プロパティの型は設定の型と同じです。プロパティが読み取り専用かどうかは、設定のスコープで決定されます。つまり、アプリケーション スコープの設定のプロパティは読み取り専用であるのに対し、ユーザー スコープの設定のプロパティは読み書き可能です。詳細については、「My.Settings オブジェクト」を参照してください。

> **メモ：**
>
> アプリケーションスコープの設定値を実行時に変更または保存することはできません。アプリケーションスコープの設定を変更するには、アプリケーションを作成するときにプロジェクト デザイナを使用して変更するか、またはアプリケーションの構成ファイルを編集するかのいずれかの方法のみが使用できます。詳細については、「アプリケーションの設定の管理」を参照してください。

この例では、**PropertyGrid** コントロールを使用して、**My.Settings** オブジェクトのユーザー設定プロパティにアクセスします。既定では、**PropertyGrid** には **My.Settings** オブジェクトのすべてのプロパティが表示されます。一方、ユーザー設定プロパティは UserScopedSettingAttribute 属性を持ちます。この例では、**PropertyGrid** の BrowsableAttributes プロパティを **UserScopedSettingAttribute** に設定して、ユーザー設定プロパティのみを表示します。

ユーザー設定のプロパティ グリッドを追加するには

1. アプリケーションのデザイン サーフェイス (ここでは `Form1` とする) に対して、ツールボックスから PropertyGrid コントロールを追加します。

   プロパティグリッド コントロールの既定の名前は `PropertyGrid1` です。

2. `Form1` のデザイン サーフェイスをダブルクリックし、フォーム読み込みのイベント ハンドラのコードを開きます。

3. **My.Settings** オブジェクトを、プロパティ グリッド用に選択されたオブジェクトとして設定します。

   VB

   ```
   PropertyGrid1.SelectedObject = My.Settings
   ```

4. ユーザー設定のみを表示するようにプロパティ グリッドを構成します。

   VB

   ```
   ' Attribute for the user-scope settings.
   Dim userAttr As New System.Configuration.UserScopedSettingAttribute
   Dim attrs As New System.ComponentModel.AttributeCollection(userAttr)
   PropertyGrid1.BrowsableAttributes = attrs
   ```

   > **メモ：**
   >
   > アプリケーションスコープの設定のみを表示するには、**UserScopedSettingAttribute** ではなく ApplicationScopedSettingAttribute 属性を使用します。

堅牢性の高いプログラム

アプリケーションがユーザー設定を保存するのは、アプリケーションが終了するときです。設定をすぐに保存するには、**My.Settings.Save** メソッドを呼び出します。詳細については、「方法：Visual Basic でユーザー設定を永続化する」を参照してください。

参照

処理手順

方法 : Visual Basic でアプリケーション設定を読み取る
方法 : Visual Basic でユーザー設定を変更する
方法 : Visual Basic でユーザー設定を永続化する
方法 : アプリケーション設定を追加または削除する

関連項目

My.Settings オブジェクト

その他の技術情報

アプリケーションの設定の管理

方法：Visual Basic でユーザー設定を永続化する

My.Settings.Save メソッドを使用して、ユーザー設定の変更内容を永続化できます。

通常、アプリケーションでは、ユーザー設定の変更内容を、アプリケーションの終了時に永続化するようにデザインされています。さまざまな要因によっては、設定の保存に数秒かかる場合もあるからです。

詳細については、「My.Settings オブジェクト」を参照してください。

メモ：

ユーザー スコープの設定値は、実行時に変更および保存できますが、アプリケーション スコープの設定は読み取り専用であり、プログラムによって変更できません。アプリケーション スコープの設定は、アプリケーションを作成するときにプロジェクト デザイナを使用するか、またはアプリケーションの構成ファイルを編集して変更できます。詳細については、「アプリケーションの設定の管理」を参照してください。

使用例

この例では、`LastChanged` ユーザー設定の値を変更し、**My.Settings.Save** メソッドを呼び出して、その変更内容を保存します。

VB

```
Sub ChangeAndPersistSettings()
    My.Settings.LastChanged = Today
    My.Settings.Save()
End Sub
```

この例が動作するためには、アプリケーションに **Date** 型の `LastChanged` ユーザー設定が必要です。詳細については、「方法 : アプリケーション設定を追加または削除する」を参照してください。

参照

処理手順

方法 : Visual Basic でアプリケーション設定を読み取る
方法 : Visual Basic でユーザー設定を変更する
方法 : Visual Basic でユーザー設定のためのプロパティ グリッドを作成する
方法 : アプリケーション設定を追加または削除する

関連項目

My.Settings オブジェクト

その他の技術情報

アプリケーションの設定の管理

方法 : Visual Basic でユーザー設定を変更する

My.Settings オブジェクトで設定のプロパティに新しい値を割り当てると、ユーザー設定を変更できます。

My.Settings オブジェクトでは、各設定がプロパティとして公開されています。プロパティの名前は設定の名前と同じで、プロパティの型は設定の型と同じです。プロパティが読み取り専用かどうかは、設定のスコープで決定されます。つまり、アプリケーション スコープの設定のプロパティは読み取り専用であるのに対し、ユーザー スコープの設定のプロパティは読み書き可能です。詳細については、「My.Settings オブジェクト」を参照してください。

メモ :

ユーザー スコープ設定の値は実行時に変更および保存できますが、アプリケーション スコープ設定は読み取り専用であり、プログラムから変更できません。アプリケーション スコープの設定は、アプリケーションを作成するときにプロジェクト デザイナを使用するか、またはアプリケーションの構成ファイルを編集して変更できます。詳細については、「アプリケーションの設定の管理」を参照してください。

使用例

この例では、`Nickname` ユーザー設定の値を変更します。

VB

```
Sub ChangeNickname(ByVal newNickname As String)
    My.Settings.Nickname = newNickname
End Sub
```

この例が動作するためには、アプリケーションに **String** 型の `Nickname` ユーザー設定が必要です。詳細については、「方法 : アプリケーション設定を追加または削除する」を参照してください。

アプリケーションがユーザー設定を保存するのは、アプリケーションが終了するときです。設定をすぐに保存するには、**My.Settings.Save** メソッドを呼び出します。詳細については、「方法 : Visual Basic でユーザー設定を永続化する」を参照してください。

参照

処理手順

方法 : Visual Basic でアプリケーション設定を読み取る
方法 : Visual Basic でユーザー設定を永続化する
方法 : Visual Basic でユーザー設定のためのプロパティ グリッドを作成する
方法 : アプリケーション設定を追加または削除する

関連項目

My.Settings オブジェクト

その他の技術情報

アプリケーションの設定の管理

方法：Visual Basic でアプリケーション設定を読み取る

My.Settings オブジェクトでユーザー設定のプロパティにアクセスすると、ユーザー設定を読み取ることができます。

My.Settings オブジェクトでは、各設定がプロパティとして公開されています。プロパティの名前は設定の名前と同じで、プロパティの型は設定の型と同じです。プロパティが読み取り専用かどうかは、設定のスコープでわかります。つまり、アプリケーション スコープの設定のプロパティは読み取り専用であるのに対し、ユーザー スコープの設定のプロパティは読み書き可能です。詳細については、「My.Settings オブジェクト」を参照してください。

使用例

この例では、`Nickname` 設定の値を表示します。

VB

```
Sub ShowNickname()
    MsgBox("Nickname is " & My.Settings.Nickname)
End Sub
```

この例が動作するためには、アプリケーションに **String** 型の `Nickname` 設定が必要です。詳細については、「方法 : アプリケーション設定を追加または削除する」を参照してください。

参照

処理手順

方法 : Visual Basic でユーザー設定を変更する
方法 : Visual Basic でユーザー設定を永続化する
方法 : Visual Basic でユーザー設定のためのプロパティ グリッドを作成する
方法 : アプリケーション設定を追加または削除する

関連項目

My.Settings オブジェクト

その他の技術情報

アプリケーションの設定の管理

ドライブ、ディレクトリ、およびファイルの処理

Visual Basic では、**My.Computer.FileSystem** オブジェクトを使用して、ドライブ、フォルダ、およびファイルを処理できます。このオブジェクトは、**FileOpen** 関数や **Write** 関数などの従来のメソッドに比べて、パフォーマンスが優れており、使用も簡単です (ただし、それら従来のメソッドも引き続き使用可能です)。以下のセクションでは、これらの方法について詳しく説明します。

このセクションの内容

Visual Basic におけるファイル アクセス

My.Computer.FileSystem オブジェクトを使ってファイル、ドライブ、およびフォルダを処理する方法を説明します。

チュートリアル : .NET Framework のメソッドによるファイル操作

.NET Framework を使用してファイルやフォルダを操作する方法を説明します。

チュートリアル : Visual Basic によるファイルとディレクトリの操作

My.Computer.FileSystem オブジェクトを使用してファイルやフォルダを操作する方法を説明します。

関連するセクション

プログラム構造とコード規則

プログラムの物理構造と外観のガイドラインを示します。

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem オブジェクトとそのメンバのリファレンス ドキュメントです。

Visual Basic での .NET Framework のファイル I/O

.NET Framework でのファイル I/O の概要を示します。

Visual Basic におけるファイル アクセス

My.Computer.FileSystem オブジェクトは、ファイルとフォルダを扱うためのツールです。そのプロパティ、メソッド、およびイベントを使用すると、ファイルとフォルダを作成、コピー、移動、調査、および削除できます。**My.Computer.FileSystem** は、下位互換性のために Visual Basic に用意されているレガシ関数 (**FileOpen**、**FileClose**、**Input**、**InputString**、**LineInput** など) よりもパフォーマンスが優れています。

このセクションの内容

Visual Basic でのファイルの読み取り

My.Computer.FileSystem オブジェクトを使用したファイルからの読み込みについてのトピックを一覧表示します。

Visual Basic でのファイルへの書き込み

My.Computer.FileSystem オブジェクトを使用したファイルへの書き込みについてのトピックを一覧表示します。

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

My.Computer.FileSystem オブジェクトを使用したファイルとフォルダの作成、コピー、削除、および移動についてのトピックを一覧表示します。

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.FileSystem オブジェクトを使用したファイル、フォルダ、およびドライブのプロパティの確認についてのトピックを一覧表示します。

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldReader を使用してログなどのテキスト ファイルを解析する方法について説明します。

ファイル エンコーディング

ファイル エンコーディングとその使用について説明します。

チュートリアル : Visual Basic によるファイルとディレクトリの操作

ファイルとフォルダについての情報を報告するユーティリティの作成方法の例を示します。

トラブルシューティング : テキスト ファイルの読み取りと書き込み

テキスト ファイルの読み込みと書き込みで発生する一般的な問題を一覧表示し、それぞれの解決策を示します。

Visual Basic でのファイルの読み取り

このセクションでは、ファイルからの読み取りに関連するタスクの一覧を示します。

このセクションの内容

方法 : Visual Basic でテキスト ファイルを読み取る

テキスト ファイルからの読み取り方法をデモンストレーションします。

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る

区切り記号が使用されたテキスト ファイルからの読み取り方法をデモンストレーションします。

方法 : Visual Basic で固定幅のテキスト ファイルを読み取る

固定幅のテキスト ファイルからの読み取り方法をデモンストレーションします。

方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

複数の書式を持つテキスト ファイルからの読み取り方法をデモンストレーションします。

方法 : Visual Basic でバイナリ ファイルを読み取る

バイナリ ファイルからの読み取り方法をデモンストレーションします。

方法 : My Documents の既存のテキスト ファイルを読み取る (Visual Basic)

My Documents ディレクトリのテキスト ファイルを読み取る方法をデモンストレーションします。

方法 : StreamReader を使用してファイルからテキストを読み取る (Visual Basic)

StreamReader を使用してファイルを読み取る方法をデモンストレーションします。

参照

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem オブジェクトとそのメンバについて説明します。

My.Computer.FileSystem.ReadAllText メソッド

ReadAllText メソッドについて説明します。

My.Computer.FileSystem.ReadAllBytes メソッド

ReadAllBytes メソッドについて説明します。

My.Computer.FileSystem.OpenTextFieldParser メソッド

OpenTextFieldParser メソッドについて説明します。

My.Computer.FileSystem.OpenTextFileReader メソッド

OpenTextFileReader メソッドについて説明します。

関連するセクション

方法 : Visual Basic でクリップボードから読み込む

クリップボードからデータを読み込む方法を説明します。

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser オブジェクトを使用してテキスト ファイルを読み取る方法の概要を示します。

チュートリアル : Visual Basic によるファイルとディレクトリの操作

ファイルおよびディレクトリに関して **My** 機能を使用する方法をデモンストレーションします。

チュートリアル : .NET Framework のメソッドによるファイル操作

ファイルおよびディレクトリに関して .NET Framework のメソッドを使用する方法をデモンストレーションします。

方法：Visual Basic でテキスト ファイルを読み取る

My.Computer.FileSystem オブジェクトの **ReadAllText** メソッドを使用すると、テキスト ファイルを読み取ることができます。ファイルの内容が ASCII や UTF-8 などのエンコーディングを使用している場合、ファイル エンコーディングを指定できます。

読み取るファイルで拡張文字が使用されている場合、ファイル エンコーディングを指定する必要があります。

テキスト ファイルを読み取るには

- **My.Computer.FileSystem** オブジェクトの **ReadAllText** メソッドを使用して、ファイルのパスを指定し、テキスト ファイルの内容を文字列に読み取ります。次の例は、test.txt の内容を文字列に読み取り、メッセージ ボックスに表示します。

  VB

```
Dim fileReader As String
fileReader = My.Computer.FileSystem.ReadAllText("C:\test.txt")
MsgBox(fileReader)
```

エンコードされているテキスト ファイルを読み取るには

- **My.Computer.FileSytem** オブジェクトの **ReadAllText** メソッドを使用して、ファイルのパスとファイル エンコードの種類を指定し、テキスト ファイルの内容を文字列に読み取ります。次の例は、UTF32 ファイルである test.txt の内容を文字列に読み取り、メッセージ ボックスに表示します。

  VB

```
Dim fileReader As String
fileReader = My.Computer.FileSystem.ReadAllText("C:\test.txt", _
   System.Text.Encoding.UTF32)
MsgBox(fileReader)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である場合。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない場合 (ArgumentNullException)。
- ファイルが存在しない場合 (FileNotFoundException)
- 他のプロセスがファイルを使用しているか、または I/O エラーが発生した場合 (IOException)。
- パスがシステムで定義されている最大長を超えている場合 (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である場合 (NotSupportedException)。
- 文字列をバッファに書き込むための十分なメモリがない場合 (OutOfMemoryException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない場合 (SecurityException)。

ファイル名からファイルの内容を判断しないでください。たとえば、Form1.vb というファイルは Visual Basic のソース ファイルではない可能性もあります。

アプリケーションでデータを使用する前に、入力をすべて検証してください。ファイルの内容が予想どおりでないことがあり、ファイルの内容を読み取るメソッドが失敗する可能性があります。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る

方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る
トラブルシューティング : テキスト ファイルの読み取りと書き込み
チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.ReadAllText メソッド

概念

ファイル エンコーディング

その他の技術情報

Visual Basic でのファイルの読み取り

方法：Visual Basic でコンマ区切りのテキスト ファイルを読み取る

TextFieldParser オブジェクトを使用すると、ログなどの構造化されたテキスト ファイルを簡単かつ効率的に解析できます。区切り記号が使用されたファイルと固定幅のテキスト フィールドを持つファイルのどちらであるかは、**TextFieldType** プロパティで定義します。

コンマ区切りのテキスト ファイルを解析するには

1. 新しい **TextFieldParser** を作成します。次のコードは、`MyReader` という名前の **TextFieldParser** を作成し、`test.txt` ファイルを開きます。

   VB

   ```
   Using MyReader As New _
   Microsoft.VisualBasic.FileIO.TextFieldParser _
   ("C:\TestFolder\test.txt")
   ```

2. **TextField** の型と区切り記号を定義します。次のコードは、**TextFieldType** プロパティを `Delimited` と定義し、区切り記号を "," と定義します。

   VB

   ```
   MyReader.TextFieldType = FileIO.FieldType.Delimited
   MyReader.SetDelimiters(",")
   ```

3. ファイル内の各フィールドをループします。破損している行がある場合は、エラーを報告し、解析を続けます。次のコードは、ファイルをループし、各フィールドを順番に表示して、書式に誤りのあるフィールドを報告します。

   VB

   ```
   Dim currentRow As String()
      While Not MyReader.EndOfData
         Try
            currentRow = MyReader.ReadFields()
            Dim currentField As String
            For Each currentField In currentRow
               MsgBox(currentField)
            Next
            Catch ex As _
            Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
            "is not valid and will be skipped.")
        End Try
   ```

4. **While** ブロックと **Using** ブロックを **End While** と **End Using** で閉じます。

   VB

   ```
      End While
   End Using
   ```

使用例

この例では、`test.txt` ファイルを読み取ります。

VB

```
Using MyReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\testfile.txt")
   MyReader.TextFieldType = FileIO.FieldType.Delimited
   MyReader.SetDelimiters(",")
   Dim currentRow As String()
   While Not MyReader.EndOfData
      Try
         currentRow = MyReader.ReadFields()
         Dim currentField As String
         For Each currentField In currentRow
            MsgBox(currentField)
         Next
      Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
      MsgBox("Line " & ex.Message & _
      "is not valid and will be skipped.")
      End Try
   End While
End Using
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- 指定の書式を使用して行を解析できない場合 (MalformedLineException)。例外メッセージは、例外が発生した行を表し、TextFieldParser.ErrorLine プロパティ には、行に含まれているテキストが代入される。
- 指定のファイルが存在しない場合 (FileNotFoundException)。
- 部分信頼の状況で、ファイルにアクセスするための十分なアクセス許可がユーザーにない場合 (SecurityException).
- パスが長すぎる場合 (PathTooLongException)
- ユーザーがファイルにアクセスするのに必要なアクセス許可がない場合 (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る
チュートリアル : Visual Basic によるファイルとディレクトリの操作
トラブルシューティング : テキスト ファイルの読み取りと書き込み
例外のトラブルシューティング : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

関連項目

TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

方法：Visual Basic で固定幅のテキスト ファイルを読み取る

TextFieldParser オブジェクトを使用すると、ログなどの構造化されたテキスト ファイルを簡単かつ効率的に解析できます。

区切り記号が使用されたファイルと固定幅のテキスト フィールドを持つファイルのどちらであるかは、**TextFieldType** プロパティで定義します。固定幅のファイル内で可変幅のフィールドを指定するには、フィールド幅を -1 として定義します。

固定幅のテキスト ファイルを解析するには

1. 新しい **TextFieldParser** を作成します。次のコードは、`Reader` という名前の **TextFieldParser** を作成し、`test.log` ファイルを開きます。

   VB

   ```
   Using Reader As New _
   Microsoft.VisualBasic.FileIO.TextFieldParser _
   ("C:\TestFolder\test.log")
   ```

2. **TextFieldType** プロパティを **FixedWidth** として定義し、幅と形式を定義します。次のコードは、テキストの列を定義します。最初は幅が 5 文字、その次は 10、その次は 11、その次は可変幅です。

   VB

   ```
   Reader.TextFieldType = _
   Microsoft.VisualBasic.FileIO.FieldType.FixedWidth
   Reader.SetFieldWidths(5, 10, 11, -1)
   ```

3. ファイル内の各フィールドをループします。破損している行がある場合は、エラーを報告し、解析を続けます。

   VB

   ```
   Dim currentRow As String()
      While Not Reader.EndOfData
         Try
            currentRow = Reader.ReadFields()
            Dim currentField As String
            For Each currentField In currentRow
               MsgBox(currentField)
            Next
         Catch ex As _
         Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
            "is not valid and will be skipped.")
    End Try
   ```

4. **While** ブロックと **Using** ブロックを **End While** と **End Using** で閉じます。

   VB

   ```
      End While
   End Using
   ```

使用例

この例では、`test.log` ファイルを読み取ります。

VB

```
Using Reader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\TestFolder\test.log")
   Reader.TextFieldType = _
   Microsoft.VisualBasic.FileIO.FieldType.FixedWidth
   Reader.SetFieldWidths(5, 10, 11, -1)
   Dim currentRow As String()
   While Not Reader.EndOfData
      Try
         currentRow = Reader.ReadFields()
         Dim currentField As String
         For Each currentField In currentRow
            MsgBox(currentField)
         Next
      Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
         MsgBox("Line " & ex.Message & _
         "is not valid and will be skipped.")
      End Try
   End While
End Using
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- 指定の書式を使用して行を解析できない場合 (MalformedLineException)。例外メッセージは、例外が発生した行を表し、TextFieldParser.ErrorLine プロパティ には、行に含まれているテキストが代入される。
- 指定のファイルが存在しない場合 (FileNotFoundException)。
- 部分信頼の状況で、ファイルにアクセスするための十分なアクセス許可がユーザーにない場合 (SecurityException).
- パスが長すぎる場合 (PathTooLongException)
- ユーザーがファイルにアクセスするのに必要なアクセス許可がない場合 (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る
チュートリアル : Visual Basic によるファイルとディレクトリの操作
トラブルシューティング : テキスト ファイルの読み取りと書き込み
例外のトラブルシューティング : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

関連項目

TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

方法：Visual Basic で複数の書式を持つテキスト ファイルを読み取る

TextFieldParser オブジェクトを使用すると、ログなどの構造化されたテキスト ファイルを簡単かつ効率的に解析できます。**PeekChars** メソッドを使用して、ファイルを解析するときに各行の書式を判断することにより、複数の書式を持つファイルを処理できます。

複数の書式を持つテキスト ファイルを解析するには

1. 目的の書式と、エラーが報告されるときに使用する書式を定義します

   VB

```
Dim StdFormat As Integer()= {5,10,11,-1}
Dim ErrorFormat As Integer() = {5,5,-1}
```

2. 幅と書式を指定して、新しい **TextFieldParser** オブジェクトを作成します。

   VB

```
Using MyReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\testfile.txt")
MyReader.TextFieldType = FileIO.FieldType.FixedWidth
```

3. 各行をループし、読み取り前に書式を調べます。

   VB

```
Dim CurrentRow As String()
While Not MyReader.EndOfData
   Try
      Dim RowType As String = MyReader.PeekChars(3)
      If String.Compare(RowType, "Err") = 0 Then
         ' If this line describes an error, the format of
         ' the row will be different.
         MyReader.SetFieldWidths(ErrorFormat)
         CurrentRow = MyReader.ReadFields
         MyReader.SetFieldWidths(StdFormat)
                    Else
         'Otherwise parse the fields normally
         CurrentRow = MyReader.ReadFields
         For Each newString As String In CurrentRow
            My.Computer.FileSystem.WriteAllText _
            ("newFile.txt", newString, True)
          Next
   End If
```

4. エラーをコンソールに書き込みます。

   VB

```
      Catch ex As _
      Microsoft.VisualBasic.FileIO.MalformedLineException
         MsgBox("Line " & ex.Message & " is invalid.")
      End Try
   End While
End Using
```

使用例
この例では、`testfile.txt` ファイルを読み取ります。

VB

```
Dim StdFormat As Integer() = {5, 10, 11, -1}
Dim ErrorFormat As Integer() = {5, 5, -1}
Using MyReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\testfile.txt")
   MyReader.TextFieldType = FileIO.FieldType.FixedWidth
   MyReader.FieldWidths = StdFormat
   Dim CurrentRow As String()
      While Not MyReader.EndOfData
         Try
            Dim RowType As String = MyReader.PeekChars(3)
            If String.Compare(RowType, "Err") = 0 Then
               ' If this line describes an error, the format of the row will be different.
               MyReader.SetFieldWidths(ErrorFormat)
               CurrentRow = MyReader.ReadFields
               MyReader.SetFieldWidths(StdFormat)
            Else
               ' Otherwise parse the fields normally
               CurrentRow = MyReader.ReadFields
               For Each newString As String In CurrentRow
                  My.Computer.FileSystem.WriteAllText("newFile.txt", newString, True)
               Next
            End If
         Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
         End Try
      End While
End Using
```

堅牢性の高いプログラム
次の条件を満たす場合は、例外が発生する可能性があります。

- 指定の書式を使用して行を解析できない場合 (MalformedLineException)。例外メッセージは、例外が発生した行を表し、TextFieldParser.ErrorLine プロパティ には、行に含まれているテキストが代入される。
- 指定のファイルが存在しない場合 (FileNotFoundException)。
- 部分信頼の状況で、ファイルにアクセスするための十分なアクセス許可がユーザーにない場合。(SecurityException).
- パスが長すぎる場合 (PathTooLongException)
- ユーザーがファイルにアクセスするのに必要なアクセス許可がない場合 (UnauthorizedAccessException)。

参照
処理手順
方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
関連項目
TextFieldParser オブジェクト
TextFieldParser.PeekChars メソッド
My.Computer.FileSystem.WriteAllText メソッド
TextFieldParser.EndOfData プロパティ
TextFieldParser.TextFieldType プロパティ
MalformedLineException
概念
TextFieldParser オブジェクトによるテキスト ファイルの解析

方法：Visual Basic でバイナリ ファイルを読み取る

My.Computer.FileSystem オブジェクトには、バイナリ ファイルを読み取るための **ReadAllBytes** メソッドが用意されています。

バイナリ ファイルを読み取るには

- **ReadAllBytes** メソッドを使用します。このメソッドは、ファイルの内容をバイト配列として返します。この例では、`C:/Documents and Settings/selfportrait.jpg` ファイルを読み取ります。

  VB

```
My.Computer.FileSystem.ReadAllBytes _
("C:/Documents and Settings/selfportrait.jpg")
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外がスローされる可能性があります。

- パスが無効である場合。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである、のいずれかの理由が考えられる (ArgumentException)
- パスが **Nothing** であるため、有効でない場合 (ArgumentNullException)
- ファイルが存在しない場合 (FileNotFoundException)
- 他のプロセスがファイルを使用しているか、または I/O エラーが発生した場合 (IOException)
- パスがシステムで定義されている最大長を超えている場合 (PathTooLongException)
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である場合 (NotSupportedException)
- 文字列をバッファに書き込むための十分なメモリがない場合 (OutOfMemoryException)
- ユーザーがパスを参照するのに必要なアクセス許可がない場合 (SecurityException)

ファイル名からファイルの内容を判断しないでください。たとえば、Form1.vb というファイルが Visual Basic のソース ファイルではない可能性もあります。

アプリケーションでデータを使用する前に、入力をすべて検証してください。ファイルの内容が予想どおりでないことがあり、ファイルの内容を読み取るメソッドが失敗する可能性があります。

参照

処理手順

方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem.ReadAllBytes メソッド

My.Computer.FileSystem.WriteAllBytes メソッド

その他の技術情報

Visual Basic でのファイルの読み取り

クリップボードのデータの格納と読み込み

方法：My Documents の既存のテキスト ファイルを読み取る (Visual Basic)

次のコード例は、My Documents フォルダのテキスト ファイルの内容を単一のファイルに読み取ります。

使用例

VB

```
Dim filePaths As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
Dim allText As String
Try
   filePaths = My.Computer.FileSystem.GetFiles _
   (My.Computer.FileSystem.SpecialDirectories.MyDocuments)
   For Each file As String In filePaths
      allText = My.Computer.FileSystem.ReadAllText(file)
      My.Computer.FileSystem.WriteAllText("bigfile.txt", allText, True)
   Next
Catch fileException As Exception
   Throw fileException
End Try
```

コードのコンパイル方法

"bigfile.txt" の部分を、書き込みを行うファイルの名前に置き換えてください。

堅牢性の高いプログラム

読み取る対象のファイルはテキスト ファイルであることが必要です。

OpenFileDialog コンポーネント (Windows フォーム) および SaveFileDialog コンポーネント (Windows フォーム) を使用すると、アクセス許可関連のランタイム エラーの可能性を減らすことができます。

ファイル名からファイルの内容を判断しないでください。たとえば、Form1.vb というファイルは Visual Basic のソース ファイルではない可能性もあります。

アプリケーションでデータを使用する前に、入力をすべて検証してください。ファイルの内容が予想どおりでないことがあり、ファイルの内容を読み取るメソッドが失敗する可能性があります。

セキュリティ

ファイルを読み取るには、アセンブリに対して FileIOPermission クラスで特権レベルが許可されている必要があります。部分的に信頼されているコンテキストでプロセスを実行している場合は、特権不足のため例外がスローされることがあります。詳細については、「コード アクセス セキュリティの基礎」を参照してください。また、ユーザーは、ファイルに対するアクセスも必要です。詳細については、「アクセス制御リスト (ACL)」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト
My.Computer.FileSystem.ReadAllText メソッド
My.Computer.FileSystem.WriteAllText メソッド
OpenFileDialog
SaveFileDialog

方法：StreamReader を使用してファイルからテキストを読み取る (Visual Basic)

My.Computer.FileSystem オブジェクトには、TextReader および TextWriter を開くためのメソッドがあります。**OpenTextFileWriter** メソッドと **OpenTextFileReader** メソッドです。これらは高度なメソッドで、[すべての候補] タブを選択しないと IntelliSense で表示されません。

テキスト リーダーを使用してファイルから行を読み取るには

- **OpenTextFileReader** メソッドにファイルを指定して **TextReader** を開きます。この例では、`testfile.txt` という名前のファイルを開き、1 行を読み取って、その行をメッセージ ボックスに表示します。

  VB

```
Dim fileReader As System.IO.StreamReader
fileReader = _
My.Computer.FileSystem.OpenTextFileReader("C:\\testfile.txt")
Dim stringReader As String
stringReader = fileReader.ReadLine()
MsgBox("The first line of the file is " & stringReader)
```

堅牢性の高いプログラム

読み取るファイルはテキスト ファイルである必要があります。

ファイル名からファイルの内容を判断しないでください。たとえば、Form1.vb というファイルが Visual Basic のソース ファイルではない可能性もあります。

アプリケーションでデータを使用する前に、入力をすべて検証してください。ファイルの内容が予想どおりでないことがあり、ファイルの内容を読み取るメソッドが失敗する可能性があります。

セキュリティ

ファイルを読み取るには、アセンブリに対して FileIOPermission クラスで特権レベルが許可されている必要があります。部分的に信頼されているコンテキストでプロセスを実行している場合は、特権不足のため例外がスローされることがあります。詳細については、「コード アクセス セキュリティの基礎」を参照してください。ユーザーも、ファイルに対するアクセス権を必要とします。詳細については、「アクセス制御リスト (ACL)」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.OpenTextFileWriter メソッド
My.Computer.FileSystem.OpenTextFileReader メソッド
OpenFileDialog

その他の技術情報

SaveFileDialog コンポーネント (Windows フォーム)
Visual Basic でのファイルの読み取り

Visual Basic でのファイルへの書き込み

このセクションでは、ファイルへの書き込みに関連するタスクの一覧を示します。

このセクションの内容

方法 : Visual Basic でテキストをファイルに書き込む

テキスト ファイルへの書き込み方法をデモンストレーションします。

方法 : Visual Basic でテキスト ファイルに追記する

テキスト ファイルにテキストを追加する方法をデモンストレーションします。

方法 : Visual Basic でバイナリ ファイルに書き込む

バイナリ ファイルへの書き込み方法をデモンストレーションします。

方法 : Visual Basic で My Documents ディレクトリのファイルにテキストを書き込む

My Documents ディレクトリに新しいテキスト ファイルを作成して書き込む方法をデモンストレーションします。

方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

System.IO.StreamWriter オブジェクトでのファイルへの書き込み方法をデモンストレーションします。

参照

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem オブジェクトとそのメソッドおよびプロパティについて説明します。

My.Computer.FileSystem.OpenTextFileWriter メソッド

OpenTextFileWriter メソッドについて説明します。

My.Computer.FileSystem.WriteAllBytes メソッド

WriteAllBytes メソッドについて説明します。

My.Computer.FileSystem.WriteAllText メソッド

WriteAllText メソッドについて説明します。

関連するセクション

Visual Basic でのファイルの読み取り

ファイルからの読み込みに関連するタスクの一覧を示します。

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

ファイルとディレクトリの作成、削除、移動、および名前の変更に関連するタスクの一覧を示します。

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

ファイル、ディレクトリ、およびドライブのプロパティの確認に関連するタスクの一覧を示します。

方法 : Visual Basic でクリップボードに書き込む

クリップボードにデータを書き込む方法を説明します。

ファイル エンコーディング

ファイル エンコーディングの概要を説明します。

方法：Visual Basic でテキストをファイルに書き込む

My.Computer.FileSystem.WriteAllText メソッドを使用すると、テキストをファイルに書き込むことができます。指定したファイルが存在しない場合は、新たに作成されます。

手順

テキストをファイルに書き込むには

- **WriteAllText** メソッドを使用して、ファイルおよび書き込むテキストを指定し、テキストをファイルに書き込みます。この例では、`"This is new text."` という行を `test.txt` という名前のファイルに書き込みます。ファイルに既存のテキストがある場合は、この行を追加します。

  VB

```
My.Computer.FileSystem.WriteAllText("C:\TestFolder1\test.txt", _
"This is new text to be added.",True)
```

複数の文字列をファイルに書き込むには

- 文字列コレクションに対し反復処理を行います。**WriteAllText** メソッドを使用して、テキストをファイルに書き込みます。その際、対象のファイルおよび追加する文字列を指定し、*append* を **True** に設定します。

  この例では、`Documents and Settings` ディレクトリにあるファイルの名前を `FileList.txt` に書き込みます。その際、それぞれの間に復帰を挿入して、読みやすくなるようにします。

  VB

```
For Each foundFile As String In _
My.Computer.FileSystem.GetFiles("C:\Documents and Settings")
foundFile = foundFile & vbCrLf
My.Computer.FileSystem.WriteAllText _
("C:\Documents and Settings\FileList.txt", foundFile, True)
Next
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *File* は存在しないパスを指している (FileNotFoundException または DirectoryNotFoundException)。
- 他のプロセスがファイルを使用しているか、または I/O エラーが発生した (IOException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- ディスクの空き領域がなく、**WriteAllText** の呼び出しが失敗した (**IOException**)。

部分的に信頼されているコンテキストでプロセスを実行している場合は、特権不足のため例外がスローされることがあります。詳細については、「コード アクセス セキュリティの基礎」を参照してください。

参照

処理手順

方法 : Visual Basic でテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.WriteAllText メソッド

方法：Visual Basic でテキスト ファイルに追記する

My.Computer.FileSystem.WriteAllText メソッドを使用して、*append* パラメータに **True** を指定すると、テキスト ファイルに追記できます。

テキスト ファイルに追記するには

- **WriteAllText** メソッドを使用して、対象のファイルと追記する文字列を指定し、*append* パラメータを **True** に設定します。

  次の例では、文字列 `"This is a test string."` を `Testfile.txt` という名前のファイルに書き込みます。

  VB

```
Dim inputString As String = "This is a test string."
My.Computer.FileSystem.WriteAllText _
("C://testfile.txt", inputString, True)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *File* は存在しないパスを指している (FileNotFoundException または DirectoryNotFoundException)。
- 他のプロセスがファイルを使用しているか、または I/O エラーが発生した (IOException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。

参照

関連項目

My.Computer.FileSystem.WriteAllText メソッド
My.Computer.FileSystem オブジェクト

その他の技術情報

Visual Basic でのファイルへの書き込み

方法：Visual Basic でバイナリ ファイルに書き込む

My.Computer.FileSystem.WriteAllBytes メソッドはバイナリ ファイルにデータを書き込みます。*append* パラメータが **True** の場合は、ファイルにデータを追加します。その他の場合は、ファイルのデータを上書きします。

指定したパス (ファイル名 以外) が有効でない場合は、DirectoryNotFoundException 例外がスローされます。パスが有効でファイルが存在しない場合は、ファイルが作成されます。

バイナリ ファイルに書き込むには

- **WriteAllBytes** メソッドを使用し、ファイルのパスと名前、および書き込むバイト数を指定します。この例では、データ配列 `CustomerData` を、`CollectedData.dat` という名前のファイルに追加します。

  VB

```
My.Computer.FileSystem.WriteAllBytes _
("C:\MyDocuments\CustomerData", CustomerData, True)
```

堅牢性の高いプログラム

例外を引き起こす可能性のある状態を次に示します。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、のいずれかの理由が考えられる。(ArgumentException).
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *File* は存在しないパスを指している (FileNotFoundException または **DirectoryNotFoundException**)。
- 他のプロセスがファイルを使用しているか、または I/O エラーが発生した (IOException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。

参照

処理手順

方法 : Visual Basic でテキストをファイルに書き込む

関連項目

My.Computer.FileSystem.WriteAllBytes メソッド

方法：Visual Basic で My Documents ディレクトリのファイルにテキストを書き込む

My.Computer.FileSystem.SpecialDirectories オブジェクトを使用すると、MyDocuments ディレクトリなどの特別なディレクトリにアクセスできます。

手順

My Documents ディレクトリに新しいテキスト ファイルを書き込むには

1. **My.Computer.FileSystem.SpecialDirectories.MyDocuments** プロパティを使用してパスを指定します。

   VB
   ```
   Dim filePath As String
   filePath = System.IO.Path.Combine( _
   My.Computer.FileSystem.SpecialDirectories.MyDocuments, "test.txt")
   ```

2. **WriteAllText** メソッドを使用して、指定のファイルにテキストを書き込みます。

   VB
   ```
   My.Computer.FileSystem.WriteAllText(filePath, "some text", True)
   ```

使用例

VB
```
Try
    Dim filePath As String
    filePath = System.IO.Path.Combine( _
               My.Computer.FileSystem.SpecialDirectories.MyDocuments, "test.txt")
            My.Computer.FileSystem.WriteAllText(filePath, "some text", False)
Catch fileException As Exception
    Throw fileException
End Try
```

コードのコンパイル方法

`test.txt` の部分を、書き込みを行うファイルの名前に置き換えてください。

堅牢性の高いプログラム

このコードでは、ファイルにテキストを書き込むときに生じる可能性のあるすべての例外を再スローしています。OpenFileDialog コンポーネントおよび SaveFileDialog コンポーネントなどの Windows フォーム コントロールを使用すると、例外の可能性を減らすことができます。有効なファイル名のみをユーザーが選択できるようになっているからです。ただし、これらのコントロールを使用しても、完璧とは限りません。ユーザーがファイルを選択してから、コードが実行されるまでの間に、ファイル システムが変更される可能性があります。したがって、ファイルを操作するときには、例外処理はほぼ必須です。

セキュリティ

部分的に信頼されているコンテキストでプロセスを実行している場合は、特権不足のため例外がスローされることがあります。詳細については、「コード アクセス セキュリティの基礎」を参照してください。

この例では新しいファイルを作成します。アプリケーションがファイルを作成する必要がある場合は、フォルダの作成のアクセス許可が必要になります。アクセス許可は、アクセス制御リストを使用して設定します。ファイルが既に存在する場合、アプリケーションに必要なのは書き込みのアクセス許可だけです。可能な場合は、フォルダに対する作成の権限を付与するのではなく、配置時にファイルを作成し、1 つのファイルに対する読み取りの権限だけを付与する方が安全です。また、ルート フォルダや Program Files フォルダにデータを書き込むよりも、ユーザー フォルダに書き込む方が安全です。詳細については、「アクセス制御リスト (ACL)」を参照してください。

参照

関連項目

My.Computer オブジェクト
My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.WriteAllText メソッド
My.Computer.FileSystem.SpecialDirectories オブジェクト
System.IO.Path.Combine(System.String,System.String)

方法：Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

この例では、**My.Computer.FileSystem.OpenTextFileWriter** メソッドを使用して StreamWriter オブジェクトを開き、**StreamWriter** クラスの WriteLine メソッドを使用して、テキスト ファイルに文字列を書き込みます。

使用例

VB

```
Dim file As System.IO.StreamWriter
file = My.Computer.FileSystem.OpenTextFileWriter("c:\test.txt", True)
file.WriteLine("Here is the first string.")
file.Close()
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- ファイルが存在するものの、読み取り専用の場合 (IOException)
- ディスクの空き領域がない場合 (**IOException**)
- パスが長すぎる場合 (PathTooLongException)

セキュリティ

次のコード例では、ファイルが存在しない場合は新規にファイルを作成します。アプリケーションでファイルを作成する必要がある場合、そのアプリケーションにはフォルダに対する **Create** アクセスが必要です。ファイルが既に存在する場合、アプリケーションに必要なのは、より低い権限である **Write** アクセスだけです。フォルダに対して **Create** アクセスを許可するのではなく、可能な限りアプリケーションの配置時にファイルを作成しておき、1 つのファイルに対してのみ **Read** アクセスを許可する方が安全です。

参照

処理手順

方法 : Visual Basic でテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem.OpenTextFileWriter メソッド

StreamWriter

その他の技術情報

Visual Basic でのファイルへの書き込み

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

このセクションでは、Visual Basic でのファイルおよびディレクトリの作成、削除、移動、および名前の変更に関連するタスクの一覧を示します。

このセクションの内容

方法 : Visual Basic で特定のパターンを持つファイルをディレクトリにコピーする

名前が特定のパターンに合致するファイル (たとえば .txt ファイルのみ) をディレクトリにコピーする方法を説明します。

方法 : Visual Basic でファイルのコピーを同じディレクトリに作成する

ファイルのコピーを同じディレクトリに作成する方法を説明します。

方法 : Visual Basic でファイルのコピーを別のディレクトリに作成する

ファイルを別のディレクトリにコピーする方法を説明します。

方法 : Visual Basic でファイルを作成する

ファイルを作成する方法を説明します。

方法 : Visual Basic でファイルを削除する

ファイルを削除する方法を説明します。

方法 : Visual Basic でディレクトリ内のすべてのファイルを削除する

指定したディレクトリのすべてのファイルを削除する方法を説明します。

方法 : Visual Basic で特定のパターンに一致するファイルを検索する

ディレクトリ内のファイルのうちで、名前が特定のパターンに合致するものだけを一覧表示する方法を説明します。

方法 : Visual Basic でファイルを移動する

ファイルを別のディレクトリに移動する方法を説明します。

方法 : Visual Basic でファイルのコレクションを移動する

複数のファイルを別のディレクトリに移動する方法を説明します。

方法 : Visual Basic でファイルの名前を変更する

ファイルの名前を変更する方法を説明します。

方法 : Visual Basic でディレクトリの名前を変更する

ディレクトリの名前を変更する方法を説明します。

方法 : Visual Basic でディレクトリを別のディレクトリにコピーする

ディレクトリを別の場所にコピーする方法を説明します。

方法 : Visual Basic でディレクトリを作成する

ディレクトリを作成する方法を説明します。

方法 : Visual Basic でディレクトリを削除する

ディレクトリを削除する方法を説明します。

方法 : Visual Basic で特定のパターンに一致するサブディレクトリを検索する

名前が特定のパターンに合致するディレクトリを一覧表示する方法を説明します。

方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する

ディレクトリ内のファイルを一覧表示する方法を説明します。

方法 : ディレクトリに含まれているファイルの数を確認する

ディレクトリ内にあるファイルの数を確認する方法を説明します。

方法 : Visual Basic でディレクトリを移動する

ディレクトリを移動する方法を説明します。

方法 : Visual Basic でディレクトリの内容を移動する

ディレクトリの内容を移動する方法を説明します。

方法 : Visual Basic で My Documents ディレクトリの内容を取得する

特別なディレクトリを読み取る方法を説明します。

方法 : Visual Basic でファイル パスを解析する

My メソッドを使用してファイル パスを結合する方法を説明します。

参照

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem オブジェクトとそのメンバについて説明します。

My.Computer.FileSystem.CombinePath メソッド

CombinePath メソッドについて説明します。

My.Computer.FileSystem.CopyDirectory メソッド

CopyDirectory メソッドについて説明します。

My.Computer.FileSystem.CopyFile メソッド

CopyFile メソッドについて説明します。

My.Computer.FileSystem.CreateDirectory メソッド

CreateDirectory メソッドについて説明します。

My.Computer.FileSystem.DeleteDirectory メソッド

DeleteDirectory メソッドについて説明します。

My.Computer.FileSystem.DeleteFile メソッド

DeleteFile メソッドについて説明します。

My.Computer.FileSystem.GetParentPath メソッド

GetParentPath メソッドについて説明します。

My.Computer.FileSystem.MoveDirectory メソッド

MoveDirectory メソッドについて説明します。

My.Computer.FileSystem.MoveFile メソッド

MoveFile メソッドについて説明します。

My.Computer.FileSystem.RenameDirectory メソッド

RenameDirectory メソッドについて説明します。

My.Computer.FileSystem.RenameFile メソッド

RenameFile メソッドについて説明します。

My.Computer.FileSystem.SpecialDirectories オブジェクト

SpecialDirectories オブジェクトについて説明します。

関連するセクション

Visual Basic でのファイルの読み取り

ファイルからの読み込みに関連するタスクの一覧を示します。

Visual Basic でのファイルへの書き込み

ファイルへの書き込みに関連するタスクの一覧を示します。

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

ファイル、ディレクトリ、およびドライブのプロパティの取得および設定に関連するタスクの一覧を示します。

方法：Visual Basic で特定のパターンを持つファイルをディレクトリにコピーする

My.Computer.FileSystem.GetFiles メソッドは、ファイルのパス名を表す文字列の読み取り専用のコレクションを返します。*wildCards* パラメータを使用して、特定のパターンを指定できます。

一致するファイルが見つからなかった場合は、空のコレクションが戻ります。

My.Computer.FileSystem.CopyFile メソッドを使用して、ファイルをディレクトリにコピーできます。

特定のパターンを持つファイルをディレクトリにコピーするには

1. **GetFiles** メソッドを使用してファイルのリストを取得します。この例では、指定のディレクトリにある、すべての .rtf ファイルを取得します。

   VB

   ```
   For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
       My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
       FileIO.SearchOption.SearchTopLevelOnly, "*.rtf")
   ```

2. **CopyFile** メソッドを使用してファイルをコピーします。この例では、`testdirectory` という名前のディレクトリにファイルをコピーします。

   VB

   ```
   My.Computer.FileSystem.CopyFile(foundFile, "C:\testdirectory\" & foundFile)
   ```

3. **For** ステートネントを **Next** ステートメントで閉じます。

   VB

   ```
   Next
   ```

使用例

次の例は、上記のスニペットを完全な形で示したものです。指定したディレクトリのすべての .rtf ファイルを `testdirectory` という名前のディレクトリにコピーします。

VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchTopLevelOnly, "*.rtf")

    My.Computer.FileSystem.CopyFile(foundFile, "C:\testdirectory\" & foundFile)
Next
```

セキュリティ

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- ディレクトリが存在しない (DirectoryNotFoundException)。
- ディレクトリが既存のファイルを指している (IOException)。

- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic で特定のパターンに一致するサブディレクトリを検索する
トラブルシューティング : テキスト ファイルの読み取りと書き込み
方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する

関連項目

My.Computer.FileSystem.CopyFile メソッド
My.Computer.FileSystem.GetFiles メソッド

方法：Visual Basic でファイルのコピーを同じディレクトリに作成する

My.Computer.FileSystem.CopyFile メソッドを使用すると、ファイルをコピーできます。そのパラメータを使用して、既存のファイルを上書きしたり、ファイルの名前を変更したり、操作の進行状況を表示したり、ユーザーが操作をキャンセルできるようにしたりできます。

ファイルのコピーを同じフォルダに作成するには

- **CopyFile** メソッドを使用します。その際、対象のファイルと場所を指定します。次の例では、`test.txt` のコピーを `test2.txt` という名前で作成します。

  VB

```
My.Computer.FileSystem.CopyFile("C:\TestFolder\test.txt", _
"C:\TestFolder\test2.txt", Microsoft.VisualBasic.FileIO.UIOption.OnlyErrorDialogs, Fil
eIO.UICancelOption.DoNothing)
```

ファイルのコピーを同じフォルダに作成し、既存のファイルを上書きするには

- **CopyFile** メソッドを使用します。その際、対象のファイルと場所を指定し、*overwrite* を **True** に設定します。次の例では、`test.txt` のコピーを `test2.txt` という名前で作成し、同じ名前のファイルが既に存在する場合は上書きします。

  VB

```
My.Computer.FileSystem.CopyFile("C:\TestFolder\test.txt", _
"C:\TestFolder\test2.txt", True)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外がスローされる可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- システムが絶対パスを取得できなかった (**ArgumentException**)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- ソース ファイルが有効でないか、または存在しない (FileNotFoundException)。
- パスを組み合わせると既存のディレクトリと同じになる (IOException)。
- 移動先にファイルが既に存在し、*overwrite* が **False** に設定されている (**IOException**)。
- ユーザーがファイルにアクセスするのに必要なアクセス許可がない (**IOException**)。
- 移動先フォルダの同名のファイルが使用中である (**IOException**).
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- *ShowUI* が **True** に、*onUserCancel* が **ThrowException** に、それぞれ設定されている状況で、ユーザーが操作をキャンセルした (OperationCanceledException)。
- *ShowUI* が **True** に、*onUserCancel* が **ThrowException** に、それぞれ設定されている状況で、未指定の I/O エラーが発生した (**OperationCanceledException**)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。

参照

処理手順

方法 : Visual Basic で特定のパターンを持つファイルをディレクトリにコピーする
方法 : Visual Basic でファイルのコピーを別のディレクトリに作成する
方法 : Visual Basic でディレクトリを別のディレクトリにコピーする
方法 : Visual Basic でファイルの名前を変更する

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.CopyFile メソッド
UICancelOption 列挙型

方法：Visual Basic でファイルのコピーを別のディレクトリに作成する

My.Computer.FileSystem.CopyFile メソッドを使用すると、ファイルをコピーできます。そのパラメータを使用して、既存のファイルを上書きしたり、ファイルの名前を変更したり、操作の進行状況を表示したり、ユーザーが操作をキャンセルできるようにしたりできます。

テキスト ファイルを別のフォルダにコピーするには

- **CopyFile** メソッドを使用して、ファイルをコピーします。その際、移動するファイルと移動先のディレクトリを指定します。*overwrite* パラメータを使用して、既存のファイルを上書きするかどうかを指定できます。次のコード例は、**CopyFile** の使い方を示しています。

  VB

```
' Copy the file to a new location without overwriting existing file.
My.Computer.FileSystem.CopyFile( _
    "C:\UserFiles\TestFiles\testFile.txt", _
    "C:\UserFiles\TestFiles2\testFile.txt")

' Copy the file to a new folder, overwriting existing file.
My.Computer.FileSystem.CopyFile( _
    "C:\UserFiles\TestFiles\testFile.txt", _
    "C:\UserFiles\TestFiles2\testFile.txt", _
    FileIO.UIOption.AllDialogs, _
    FileIO.UICancelOption.DoNothing)

' Copy the file to a new folder and rename it.
My.Computer.FileSystem.CopyFile( _
    "C:\UserFiles\TestFiles\testFile.txt", _
    "C:\UserFiles\TestFiles2\NewFile.txt", _
    FileIO.UIOption.AllDialogs, _
    FileIO.UICancelOption.DoNothing)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外がスローされる可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- システムが絶対パスを取得できなかった (**ArgumentException**)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- ソース ファイルが有効でないか、または存在しない (FileNotFoundException)。
- パスを組み合わせると既存のディレクトリと同じになる (IOException)。
- 移動先にファイルが既に存在し、*overwrite* が **False** に設定されている (**IOException**)。
- ユーザーがファイルにアクセスするのに必要なアクセス許可がない (**IOException**)。
- 移動先フォルダの同名のファイルが使用中である (**IOException**).
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- *ShowUI* が **True** に、*onUserCancel* が **ThrowException** に、それぞれ設定されている状況で、ユーザーが操作をキャンセルした (OperationCanceledException)。
- *ShowUI* が **True** に、*onUserCancel* が **ThrowException** に、それぞれ設定されている状況で、未指定の I/O エラーが発生した (**OperationCanceledException**)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。

- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。

参照

処理手順

方法 : Visual Basic で特定のパターンを持つファイルをディレクトリにコピーする
方法 : Visual Basic でファイルのコピーを同じディレクトリに作成する
方法 : Visual Basic でディレクトリを別のディレクトリにコピーする
方法 : Visual Basic でファイルの名前を変更する

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.CopyFile メソッド
UICancelOption 列挙型

方法：Visual Basic でファイルを作成する

この例では、File クラスの Create メソッドを使用して、指定したパスに空のテキスト ファイルを作成します。

使用例

VB

```
Dim file As System.IO.FileStream
file = System.IO.File.Create("c:\test.txt")
```

コードのコンパイル方法

`file` 変数を使用して、ファイルに書き込みます。

堅牢性の高いプログラム

ファイルが既に存在する場合は、新しいファイルに置き換えられます。

次の条件を満たす場合は、例外が発生する可能性があります。

- パス名の形式に誤りがある場合。たとえば、不正な文字が含まれているか、空白のみである (ArgumentException)。
- パスが読み取り専用である場合 (IOException)。
- パス名が **Nothing** である場合 (ArgumentNullException)。
- パス名が長すぎる場合 (PathTooLongException)。
- パスが無効である場合 (DirectoryNotFoundException)。
- パスにコロン (":") だけが指定されている場合 (NotSupportedException)。

セキュリティ

部分信頼の環境では、SecurityException がスローされることがあります。

Create メソッドの呼び出しでは、FileIOPermission が必要です。

ユーザーにファイル作成のアクセス許可がない場合には、UnauthorizedAccessException がスローされます。

参照

関連項目

System.IO
Create

概念

部分信頼コードからのライブラリの使用
コード アクセス セキュリティの基礎

方法：Visual Basic でファイルを削除する

My.Computer.FileSystem オブジェクトの **DeleteFile** メソッドを使用すると、ファイルを削除できます。削除したファイルをごみ箱に送るかどうか、ファイルを削除することをユーザーに確認するかどうか、ユーザーが操作をキャンセルした場合の処理方法などがオプションとして用意されています。

テキスト ファイルを削除するには

- **DeleteFile** メソッドを使用してファイルを削除します。次のコードは、`test.txt` という名前のファイルを削除する方法の例です。

  VB

```
My.Computer.FileSystem.DeleteFile("C:\test.txt")
```

ユーザーに確認したうえでテキスト ファイルを削除するには

- **DeleteFile** メソッドを使用してファイルを削除します。その際、*showUI* を **AllDialogs** に設定します。次のコードは、`test.txt` という名前のファイルを、ユーザーに確認したうえで削除する方法の例です。

  VB

```
My.Computer.FileSystem.DeleteFile("C:\test.txt", _
        FileIO.UIOption.AllDialogs, FileIO.RecycleOption.DeletePermanently, FileIO.UIC
ancelOption.DoNothing)
```

テキスト ファイルを削除してごみ箱に送るには

- **DeleteFile** メソッドを使用してファイルを削除します。その際、*recycle* パラメータに **SendToRecycleBin** を指定します。次のコードは、`test.txt` という名前のファイルを削除してごみ箱に送る方法の例です。

  VB

```
My.Computer.FileSystem.DeleteFile("C:\test.txt", _
FileIO.UIOption.AllDialogs, FileIO.RecycleOption.SendToRecycleBin)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効な場合。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない場合 (ArgumentNullException)。
- パスがシステムで定義されている最大長を超えている場合 (PathTooLongException)。
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効な場合 (NotSupportedException)。
- ファイルが使用中の場合 (IOException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない場合 (SecurityException)。
- ファイルが存在しない場合 (FileNotFoundException)
- ファイルの削除に必要なアクセス許可がユーザーにないか、またはファイルが読み取り専用の場合 (UnauthorizedAccessException)。
- 部分信頼の状況でユーザーに十分なアクセス許可がない場合 (**SecurityException**)。
- ユーザーが操作をキャンセルし、*onUserCancel* が Microsoft.VisualBasic.FileIO.UICancelOption.ThrowException に設定されてい

る場合 (OperationCanceledException)。

参照

処理手順

方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する
方法 : Visual Basic でディレクトリを削除する

関連項目

UICancelOption 列挙型
My.Computer.FileSystem オブジェクト
UIOption 列挙型
RecycleOption 列挙型

方法：Visual Basic でディレクトリ内のすべてのファイルを削除する

My.Computer.FileSystem オブジェクトの **DeleteFile** メソッドを使用すると、ファイルを削除できます。削除したファイルをごみ箱に送るかどうか、ファイルを削除することをユーザーに確認するかどうか、ユーザーが操作をキャンセルした場合の処理方法などがオプションとして用意されています。

フォルダ内のすべてのファイルを削除するには

1. **My.Computer.FileSystem.GetFiles** メソッドを使用して、ディレクトリ内のファイルを表す文字列のコレクションを取得します。
2. **For...Each** ループと **DeleteFile** メソッドを使用して、各ファイルを順番に削除します。

   次の例は、My Documents フォルダのすべてのファイルを削除します。

   VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchAllSubDirectories, "*.*")

    My.Computer.FileSystem.DeleteFile(foundFile, _
        FileIO.UIOption.AllDialogs, _
        FileIO.RecycleOption.DeletePermanently)
Next
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効な場合。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)
- パスが **Nothing** であるため、有効でない場合 (ArgumentNullException)
- パスがシステムで定義されている最大長を超えている場合 (PathTooLongException)
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効な場合 (NotSupportedException)
- ファイルが使用中の場合 (IOException)
- ユーザーがパスを参照するのに必要なアクセス許可がない場合 (SecurityException)
- ファイルが存在しない場合 (FileNotFoundException)
- ファイルの削除に必要なアクセス許可がユーザーにないか、またはファイルが読み取り専用の場合 (UnauthorizedAccessException)
- 部分信頼の状況でユーザーに十分なアクセス許可がない場合 (**SecurityException**)
- ユーザーが操作をキャンセルし、*onUserCancel* が Microsoft.VisualBasic.FileIO.UICancelOption.ThrowException に設定されている場合 (OperationCanceledException)

参照

処理手順

方法 : Visual Basic でファイルを削除する
方法 : Visual Basic でディレクトリを削除する
方法 : Visual Basic でファイルの名前を変更する
方法 : Visual Basic でファイルの絶対パスを確認する

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.DeleteFile メソッド
RecycleOption 列挙型
UICancelOption 列挙型

方法：Visual Basic で特定のパターンに一致するファイルを検索する

My.Computer.FileSystem.GetFiles メソッド は、ファイルのパス名を表す文字列の読み取り専用のコレクションを返します。*wildCards* パラメータを使用して、特定のパターンを指定できます。サブディレクトリを検索対象に含めるには、*searchType* パラメータを **SearchOption.SearchAllSubDirectories** に設定します。

指定したパターンに一致するファイルが見つからなかった場合は、空のコレクションが戻ります。

特定のパターンに一致するファイルを検索するには

- **GetFiles** メソッドを使用して、検索するディレクトリの名前とパス、およびパターンを指定します。次の例では、拡張子が `.dll` のすべてのファイルをディレクトリ内から取得し、`ListBox1` に追加します。

  VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchAllSubDirectories, "*.dll")

    Listbox1.Items.Add(foundFile)
Next
```

セキュリティ

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *directory* が存在しない (DirectoryNotFoundException)。
- *directory* は既存のファイルである (IOException)。
- パスがシステムで定義されている最大長を越えている (PathTooLongException)。
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic で特定のパターンに一致するサブディレクトリを検索する
トラブルシューティング : テキスト ファイルの読み取りと書き込み
方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する

関連項目

My.Computer.FileSystem.GetFiles メソッド

方法：Visual Basic でファイルを移動する

My.Computer.FileSystem.MoveFile メソッドを使用すると、ファイルを別のフォルダに移動できます。移動先が存在しない場合は作成されます。

ファイルを移動するには

- **MoveFile** メソッドを使用し、ファイル名および移動元と移動先のファイルの場所を指定してファイルを移動します。この例では、`test.txt` という名前のファイルを `TestDir1` から `TestDir2` に移動します。移動先ファイル名は、移動元ファイル名と同じであっても指定する点に注意してください。

  VB

```
My.Computer.FileSystem.MoveFile("C:\TestDir1\test.txt", _
    "C:\TestDir2\test.txt")
```

ファイルを移動するときに名前を変更するには

- **MoveFile** メソッドを使用してファイルを移動します。その際、移動元ファイルの名前と場所、移動先の場所、および移動先での新しい名前を指定します。この例では、`test.txt` という名前のファイルを `TestDir1` から `TestDir2` に移動し、名前を `nexttest.txt` に変更します。

  VB

```
My.Computer.FileSystem.MoveFile("C:\TestDir1\test.txt", _
    "C:\TestDir2\nexttest.txt", _
    FileIO.UIOption.AllDialogs, _
    FileIO.UICancelOption.ThrowException)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *destinationFileName* が **Nothing** または空の文字列である (**ArgumentNullException**)。
- ソース ファイルが有効でないか、または存在しない (FileNotFoundException)。
- パスを組み合わせると既存のディレクトリと同じになる、移動先のファイルが既に存在し *overwrite* が **False** に設定されている、移動先ディレクトリの同名ファイルが使用中である、またはユーザーがファイルにアクセスするのに必要なアクセス許可がない (IOException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- *showUI* が **True** に設定され、*onUserCancel* が **ThrowException** に設定されている状態で、ユーザーが操作をキャンセルしたか、または未指定の I/O エラーが発生した (OperationCanceledException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でファイルの名前を変更する
方法 : Visual Basic でファイルのコレクションを移動する

方法 : Visual Basic でファイルのコピーを別のディレクトリに作成する
方法 : Visual Basic でファイル パスを解析する
方法 : Visual Basic でディレクトリを移動する

関連項目

My.Computer.FileSystem.MoveFile メソッド

方法：Visual Basic でファイルのコレクションを移動する

My.Computer.FileSystem.MoveFile メソッドを使用すると、ディレクトリ間でファイルを移動できます。

移動先が存在しない場合は作成されます。

あるディレクトリから別のディレクトリにファイルのコレクションを移動するには

- ファイルのコレクションを定義し、**MoveFile** メソッドを呼び出します。この例では、`MyDocuments` ディレクトリのすべてのファイルを `StorageDir` フォルダに移動します。

  VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchAllSubDirectories, "*.*")

    Dim foundFileInfo As New System.IO.FileInfo(foundFile)
    My.Computer.FileSystem.MoveFile(foundFile, "C:\StorageDir\" & foundFileInfo.Name)
Next
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- ソース ファイルが有効でないか、または存在しない (FileNotFoundException)。
- パスを組み合わせると既存のディレクトリと同じになる、移動先のファイルが既に存在し *overwrite* が **False** に設定されている、移動先ディレクトリの同名ファイルが使用中である、またはユーザーがファイルにアクセスするのに必要なアクセス許可がない (IOException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- *showUI* が **True** に設定され、*onUserCancelOption* が **ThrowException** に設定されている状態で、ユーザーが操作をキャンセルしたか、または未指定の I/O エラーが発生した (OperationCanceledException)。
- パスがシステムで定義されている最大長を越えている (PathTooLongException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でファイルの名前を変更する
方法 : Visual Basic でファイルを移動する
方法 : Visual Basic でファイルのコピーを別のディレクトリに作成する
方法 : Visual Basic でファイル パスを解析する
方法 : Visual Basic でディレクトリを移動する
方法 : Visual Basic でディレクトリの内容を移動する

関連項目

My.Computer.FileSystem.MoveFile メソッド
My.Computer.FileSystem.GetFiles メソッド

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

方法：Visual Basic でファイルの名前を変更する

My.Computer.FileSystem オブジェクトの **RenameFile** メソッドを使用すると、ファイルの現在の位置、現在のファイル名、および新しいファイル名を指定して、ファイルの名前を変更できます。このメソッドでは、ファイルを移動することはできません。ファイルを移動して名前を変更するには、**MoveFile** メソッドを使用します。

ファイルの名前を変更するには

- **My.Computer.FileSystem.RenameFile** メソッドを使用してファイルの名前を変更します。この例では、`Test.txt` というファイル名を `SecondTest.txt` に変更します。

  VB

```
' Change "c:\test.txt" to the path and filename for the file that
' you want to rename.
My.Computer.FileSystem.RenameFile("C:\Test.txt", "SecondTest.txt")
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、このスニペットは [ファイル システム - ドライブ、フォルダ、およびファイルの処理] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- *newName* にパス情報が含まれている (**ArgumentException**)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *newName* が **Nothing** または空の文字列である (**ArgumentNullException**)。
- ソース ファイルが有効でないか、または存在しない (FileNotFoundException)。
- *newName* で指定したのと同じ名前のファイルまたはディレクトリが既に存在する (IOException)。
- パスがシステムで定義されている最大長を越えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でファイルを移動する
方法 : Visual Basic でファイルのコレクションを移動する
方法 : Visual Basic でファイルのコピーを同じディレクトリに作成する
方法 : Visual Basic でファイルのコピーを別のディレクトリに作成する

関連項目

My.Computer.FileSystem.RenameFile メソッド

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

方法：Visual Basic でディレクトリの名前を変更する

My.Computer.FileSystem オブジェクトの **RenameDirectory** メソッドを使用すると、ディレクトリの名前を変更できます。その際、ディレクトリの現在の場所と名前、およびディレクトリの新しい名前を指定します。このメソッドでは、ディレクトリを移動することはできません。ディレクトリを移動して名前を変更するには、**MoveDirectory** メソッドを使用します。

ディレクトリの名前を変更するには

- **My.Computer.FileSystem.RenameDirectory** メソッドを使用してディレクトリの名前を変更します。次のコードは、`Test` ディレクトリの名前を `SecondTest` に変更します。

  VB

```
My.Computer.FileSystem.RenameDirectory("C:MyDocuments\Test", _
"SecondTest")
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、この例は [ファイル システム - ドライブ、フォルダ、およびファイルの処理] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- *newName* にパス情報が含まれている (**ArgumentException**)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *newName* が **Nothing** または空の文字列である (**ArgumentNullException**)。
- 対象のディレクトリが有効でないか、または存在しない (DirectoryNotFoundException)。
- *newName* で指定したのと同じ名前のファイルまたはフォルダが既に存在する (IOException)。
- フォルダがルート フォルダである (**IOException**)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でディレクトリを別のディレクトリにコピーする
方法 : Visual Basic でファイル パスを解析する
方法 : Visual Basic でディレクトリを移動する
方法 : Visual Basic でディレクトリの内容を移動する

関連項目

My.Computer.FileSystem.RenameDirectory メソッド

方法：Visual Basic でディレクトリを別のディレクトリにコピーする

My.Computer.FileSystem.CopyDirectory メソッド メソッドを使用すると、ディレクトリを別のディレクトリにコピーできます。このメソッドでは、ディレクトリ自体とその内容がコピーされます。コピー先のディレクトリが存在しない場合は作成されます。コピー先の場所に同じ名前のディレクトリが存在し、*overwrite* が **False** に設定されている場合は、2 つのディレクトリの内容がマージされます。操作の中で、ディレクトリの新しい名前を指定できます。

ディレクトリ内のファイルをコピーするときに、特定のファイルが原因で例外がスローされることがあります。たとえば、*overwrite* が **False** に設定されていて、マージを実行しているときに、既に存在するファイルなどが原因です。こうしてスローされた例外は、単一の例外に統合され、その `Data` プロパティにエントリが保持されています。それらのエントリでは、ファイルまたはディレクトリ パスがキーとなっており、それに対応する値には、該当する例外メッセージが格納されています。

ディレクトリを別のディレクトリにコピーするには

- **CopyDirectory** メソッドを使用し、コピー元とコピー先のディレクトリ名を指定します。次の例では、`TestDirectory1` という名前のディレクトリを `TestDirectory2` にコピーします。その際、既存のファイルは上書きします。

  VB

  ```
  My.Computer.FileSystem.CopyDirectory("C:\TestDirectory1", "C:\TestDirectory2", True)
  ```

  このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、コード例は [ファイル システム - ドライブ、フォルダ、およびファイルの処理] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- ディレクトリに指定された新しい名前にコロン (:) またはスラッシュ (\ または /) が含まれている (ArgumentException)。
- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (**ArgumentException**)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *destinationDirectoryName* が **Nothing** または空の文字列である (**ArgumentNullException**)。
- コピー元のディレクトリが存在しない (DirectoryNotFoundException)。
- コピー元のディレクトリがルート ディレクトリである (IOException)。
- パスを組み合わせると既存のファイルと同じになる (**IOException**)。
- コピー元とコピー先のパスが同じである (**IOException**)。
- *ShowUI* が **UIOption.AllDialogs** に設定されており、ユーザーが操作をキャンセルしたか、またはディレクトリ内のいくつかのファイルをコピーできなかった (OperationCanceledException)。
- 操作が巡回している (InvalidOperationException)。
- パスにコロン (:) が含まれている (NotSupportedException)。
- パスがシステムで定義されている最大長を越えている (PathTooLongException)。
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効である (**NotSupportedException**)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- コピー先のファイルが存在するが、アクセスできない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic で特定のパターンに一致するサブディレクトリを検索する

方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する
方法 : Visual Basic でディレクトリを移動する
方法 : Visual Basic でディレクトリの内容を移動する

関連項目

My.Computer.FileSystem.CopyDirectory メソッド

方法：Visual Basic でディレクトリを作成する

My.Computer.FileSystem オブジェクトの **CreateDirectory** メソッドを使用すると、ディレクトリを作成できます。

ディレクトリが既に存在している場合、例外はスローされません。

ディレクトリを作成するには

- **CreateDirectory** メソッドに、ディレクトリを作成する場所の完全パスを指定します。この例では、`NewDirectory` ディレクトリを `C:\Documents and Settings\All Users\Documents` に作成します。

  VB

```
My.Computer.FileSystem.CreateDirectory _
("C:\Documents and Settings\All Users\Documents\NewDirectory")
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- ディレクトリ名が不適切である。たとえば、不正な文字が含まれているか、空白のみである (ArgumentException)。
- 作成するディレクトリの親ディレクトリが読み取り専用である (IOException)。
- ディレクトリ名が **Nothing** である (ArgumentNullException)。
- ディレクトリ名が長すぎる (PathTooLongException)。
- ディレクトリ名がコロン (":") である (NotSupportedException)。
- ディレクトリを作成するためのアクセス許可がユーザーにない (UnauthorizedAccessException)。
- 部分信頼の状況でユーザーにアクセス許可がない (SecurityException)。

参照

処理手順

方法 : Visual Basic でディレクトリが存在するかどうかを確認する

関連項目

My.Computer.FileSystem.CreateDirectory メソッド

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

方法：Visual Basic でディレクトリを削除する

My.Computer.FileSystem オブジェクトの **DeleteDirectory** メソッドを使用すると、ディレクトリを削除できます。指定できるオプションには、ディレクトリの内容を削除するかどうか、削除したディレクトリをごみ箱に送るかどうか、および削除の進行状況を表示するかどうかがあります。

空の場合のみディレクトリを削除するには

- **DeleteDirectory** メソッドを使用してディレクトリを削除します。その際、*onDirectoryNotEmpty* を **False** に設定します。この例では、`OldDirectory` という名前のディレクトリを、空の場合のみ削除します。

  VB

```
My.Computer.FileSystem.DeleteDirectory("C:\OldDirectory", _
FileIO.DeleteDirectoryOption.ThrowIfDirectoryNonEmpty)
```

ディレクトリを削除してごみ箱に送るには

- **DeleteDirectory** メソッドを使用してディレクトリを削除します。その際、*recycle* を **RecycleOption.SendToRecycleBin** に設定します。この例では、操作の進行状況を表示しながら、`OldDirectory` という名前のディレクトリおよびそのすべての内容を削除してごみ箱に送ります。

  VB

```
My.Computer.FileSystem.DeleteDirectory("C:\OldDirectory", FileIO.UIOption.AllDialogs,
FileIO.RecycleOption.SendToRecycleBin)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが長さが 0 の文字列であるか、形式に誤りがあるか、空白だけが含まれているか、または無効な文字 (ワイルドカード文字を含む) が含まれている (ArgumentException)。
- パスがデバイス パスである (\\.\ で開始されている) (**ArgumentException**)。
- パスが **Nothing** である (ArgumentNullException)。
- ディレクトリが存在しないか、またはファイルである (DirectoryNotFoundException)。
- ディレクトリまたはサブディレクトリを削除するためのアクセス許可がユーザーにない (IOException)。
- ディレクトリまたはサブディレクトリ内のファイルが使用中である (**IOException**)。
- ファイル名またはディレクトリ名にコロン (:) が含まれている (NotSupportedException)。
- *OnUserCancel* が **ThrowException** に設定されており、ユーザーが操作をキャンセルした (OperationCanceledException)。
- *OnUserCancel* が **ThrowException** に設定されており、かつディレクトリを削除できなかった (**OperationCanceledException**)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- *showUI* が **AllDialogs** に設定されており、かつユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でファイルを削除する

関連項目

UICancelOption 列挙型
My.Computer.FileSystem.DeleteDirectory メソッド
RecycleOption 列挙型
UIOption 列挙型

DeleteDirectoryOption 列挙型

方法：Visual Basic で特定のパターンに一致するサブディレクトリを検索する

My.Computer.FileSystem.GetDirectoryInfo メソッドは、ディレクトリ内のサブディレクトリのパス名を表す文字列の読み取り専用のコレクションを返します。*wildCards* パラメータを使用して、特定のパターンを指定できます。サブディレクトリの内容を検索対象に含めるには、*searchType* パラメータを **SearchOption.SearchAllSubDirectories** に設定します。

指定したパターンに一致するディレクトリが見つからなかった場合は、空のコレクションが戻ります。

特定のパターンに一致するサブディレクトリを検索するには

- **GetDirectories** メソッドを使用して、検索するディレクトリの名前およびパスを指定します。次の例では、ディレクトリ構造内のディレクトリのうちで、名前に "Logs" という単語を含むものをすべて取得し、`ListBox1` に追加します。

  VB

```
For Each foundDirectory As String In _
  My.Computer.FileSystem.GetDirectories( _
  My.Computer.FileSystem.SpecialDirectories.MyDocuments, True, _
  "*Logs*")

  ListBox1.Items.Add(foundDirectory)
Next
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- 指定したワイルドカード文字の中に、**Nothing**、空の文字列、または空白のみが含まれている (**ArgumentNullException**)。
- *directory* が存在しない (DirectoryNotFoundException)。
- *directory* は既存のファイルである (IOException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic で特定のパターンに一致するファイルを検索する

関連項目

My.Computer.FileSystem.GetDirectoryInfo メソッド

方法：Visual Basic でディレクトリにあるファイルのコレクションを取得する

My.Computer.FileSystem.GetFiles メソッドは、ディレクトリ内に格納されている各ファイルの名前を表す文字列の読み取り専用のコレクションを返します。*wildCards* パラメータを使用して、特定のパターンを指定できます。サブディレクトリを検索対象に含めるには、*searchType* パラメータを **SearchOption.SearchAllSubDirectories** に設定します。

指定したパターンに一致するファイルが見つからなかった場合は、空のコレクションが戻ります。

ディレクトリにあるファイルのリストを取得するには

- **GetFiles** メソッドを使用して、検索するディレクトリの名前およびパスを指定します。次の例では、ディレクトリ内のすべてのファイルを取得し、 `ListBox1` に追加します。

  VB

  ```
  For Each foundFile As String In My.Computer.FileSystem.GetFiles _
  (My.Computer.FileSystem.SpecialDirectories.MyDocuments)
      listBox1.Items.Add(foundFile)
  Next
  ```

  このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、コード例は [ファイル システム - ドライブ、フォルダ、およびファイルの処理] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *directory* が存在しない (DirectoryNotFoundException)。
- *directory* は既存のファイルである (IOException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic で特定のパターンに一致するファイルを検索する
方法 : Visual Basic で特定のパターンに一致するサブディレクトリを検索する

関連項目

My.Computer.FileSystem.GetFiles メソッド

方法：ディレクトリに含まれているファイルの数を確認する

My.Computer.FileSystem.GetFiles メソッドを使用すると、指定したディレクトリに含まれている各ファイルの名前を表す文字列を、読み取り専用のコレクションとして取得できます。次に、Count プロパティを使用して、ファイルの数を確認できます。

ディレクトリに含まれているファイルの数を確認するには

1. **GetFiles** メソッドを使用して、指定したディレクトリに含まれているファイルのコレクションを取得します。この例では、`TestDir` という名前のディレクトリに含まれているファイルを取得します。

   VB

   ```
   Dim counter As _
   System.Collections.ObjectModel.ReadOnlyCollection(Of String)
   counter = My.Computer.FileSystem.GetFiles("C:\TestDir")
   ```

2. **Count** プロパティを使用して、コレクションに含まれているファイルの数を確認します。この例では、メッセージ ボックスに結果を表示します。

   VB

   ```
   MsgBox("number of files is " & CStr(counter.Count))
   ```

使用例

この例は、上のスニペットを完全な形式で表したものです。`TestDir` に含まれているファイル数を数え、メッセージ ボックスで報告します。

VB

```
Dim counter As _
System.Collections.ObjectModel.ReadOnlyCollection(Of String)
counter = My.Computer.FileSystem.GetFiles("C:\TestDir")
MsgBox("number of files is " & CStr(counter.Count))
```

コードのコンパイル方法

必要な条件は次のとおりです。

- System.Collections 名前空間のメンバに対するアクセスが必要です。コード内でメンバ名を完全修飾していない場合は、**Imports** ステートメントを追加します。詳細については、「Imports ステートメント」を参照してください。
- 指定の場所に `TestDir` という名前のディレクトリが必要です。調べる対象のディレクトリのパスに置き換えます。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- ディレクトリが存在しないか、またはファイルである (DirectoryNotFoundException)。
- *directory* は既存のファイルである (IOException)。
- パスがシステムで定義されている最大長を越えている (PathTooLongException)。
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。

- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する

方法 : Visual Basic で特定のパターンに一致するファイルを検索する

関連項目

My.Computer.FileSystem.GetFiles メソッド

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

方法：Visual Basic でディレクトリを移動する

My.Computer.FileSystem.MoveDirectory メソッドを使用してディレクトリを移動できます。

存在しないディレクトリの中に移動しようとした場合は、移動先のディレクトリが作成されます。

overwrite が **True** で、移動先のディレクトリが既に存在する場合は、移動先のディレクトリ内の既存のファイルに新しいファイルが追加されます。移動元と移動先の両方のディレクトリに同名のファイルが含まれている場合は、移動先のファイルは上書きされます。

ディレクトリを移動するには

- **MoveDirectory** メソッドを使用して、ディレクトリを移動します。その際、移動元と移動先のディレクトリを指定します。次の例は、`Dir1` を `Dir2` の中に移動します。

  VB

```
My.Computer.FileSystem.MoveDirectory("C:\Dir1", "C:\Dir2")
```

ディレクトリを移動し、既存のディレクトリを上書きするには

- **MoveDirectory** メソッドを使用して、ディレクトリを移動します。その際、移動元と移動先のディレクトリを指定します。次の例は、`Dir1` を `Dir2` の中に移動します。その際、移動先のディレクトリが既に存在する場合は、既存のファイルに移動元のファイルが追加されます。

  VB

```
My.Computer.FileSystem.MoveDirectory("C:\Dir1", "C:\Dir2", True)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** である (ArgumentNullException)。
- 移動元が無効である (DirectoryNotFoundException)。
- 移動元がルート ディレクトリである (IOException)。
- パスを組み合わせると既存のファイルと同じになる (**IOException**)。
- 移動元と移動先のパスが同じである (**IOException**)。
- ファイルが既に存在し、*overwrite* が **False** に設定されている (**IOException**)。
- ファイルのサブディレクトリをコピーできない (**IOException**)。
- 操作が巡回している (InvalidOperationException)。
- パスの中のファイル名またはディレクトリ名にコロン (:) が含まれている (NotSupportedException)。
- *onUserCancel* が **UICancelOption.ThrowException** に設定されており、ユーザーが操作をキャンセルした (OperationCanceledException)。
- *onUserCancel* が **UICancelOption.ThrowException** に設定されており、操作を完了できなかった (**OperationCanceledException**)。
- パスがシステムで定義されている最大長を越えている (PathTooLongException)。
- *onUserCancel* が **UICancelOption.ThrowException** に設定されており、必要なアクセス許可がユーザーにない (SecurityException)。

- ファイルの更新に必要なアクセス許可がユーザーにない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でディレクトリの内容を移動する
方法 : Visual Basic でディレクトリを別のディレクトリにコピーする
方法 : Visual Basic でディレクトリの名前を変更する
方法 : Visual Basic でファイル パスを解析する

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

方法：Visual Basic でディレクトリの内容を移動する

My.Computer.FileSystem.GetFiles メソッドを使用して、フォルダ内のファイルのリストを取得でき、My.Computer.FileSystem.MoveFile メソッドを使用して、ディレクトリ間でファイルを移動できます。

MoveFile を呼び出したときに移動先が存在しない場合は作成されます。

ディレクトリの内容を移動するには

- **GetFiles** メソッドを使用して、ディレクトリ内のファイルのリストを取得します。次に、**MoveFile** メソッドを呼び出して、移動するファイルと移動先のディレクトリを指定します。この例では、My Documents ディレクトリのすべてのファイルを、`StorageDir` という名前のディレクトリに移動します。

  VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles( _
    My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
    FileIO.SearchOption.SearchAllSubDirectories, "*.*")

    My.Computer.FileSystem.MoveFile(foundFile, "C:\StorageDir")
Next
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (`\\.\` で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- *directory* が存在しない (DirectoryNotFoundException)。
- *directory* は既存のファイルである (IOException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。
- ユーザーに必要なアクセス許可がない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でファイルの名前を変更する
方法 : Visual Basic でファイルを移動する
方法 : Visual Basic でファイルのコピーを別のディレクトリに作成する
方法 : Visual Basic でファイル パスを解析する
方法 : Visual Basic でディレクトリを移動する
トラブルシューティング : テキスト ファイルの読み取りと書き込み
方法 : Visual Basic でファイルのコレクションを移動する

関連項目

My.Computer.FileSystem.MoveFile メソッド
My.Computer.FileSystem.GetFiles メソッド

方法：Visual Basic で My Documents ディレクトリの内容を取得する

My.Computer.FileSystem.SpecialDirectories オブジェクトを使用すると、Desktop や My Documents など、多くの All Users ディレクトリを読み取ることができます。

My Documents フォルダを読み取るには

- **ReadAllText** メソッドを使用して、指定したディレクトリの各ファイルからテキストを読み取ります。次のコードでは、ディレクトリとファイルを指定し、**ReadAllText** を使用して、`patients` という名前の文字列に読み取ります。

  VB

```
Dim path As String
Dim patients As String
path = My.Computer.FileSystem.SpecialDirectories.MyDocuments & "\" & "Patients.txt"
patients = My.Computer.FileSystem.ReadAllText(path)
```

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト
My.Computer.FileSystem.SpecialDirectories オブジェクトのメンバ
My.Computer.FileSystem.ReadAllText メソッド

方法：Visual Basic でファイル パスを解析する

My.Computer.FileSystem オブジェクトには、ファイル パスを解析するときに役立つメソッドがいくつか用意されています。

- My.Computer.FileSystem.CombinePath メソッドは、2 つのパスを受け取り、適切な書式で結合されたパスを返します。
- My.Computer.FileSystem.GetParentPath メソッドは、指定されたパスの親の絶対パスを返します。
- My.Computer.FileSystem.GetFileInfo メソッドは FileInfo オブジェクトを返します。このオブジェクトを照会すると、ファイルのプロパティ (名前やパスなど) を確認できます

ファイル名の拡張子に基づいてファイルの内容を判断しないでください。たとえば、Form1.vb というファイルが Visual Basic のソース ファイルではない可能性もあります。

ファイルの名前とパスを確認するには

- **FileInfo** オブジェクトの DirectoryName プロパティと Name プロパティを使用して、ファイルの名前とパスを確認します。この例は、名前とパスを確認し、それらを表示します。

  VB

```
Dim testFile As System.IO.FileInfo
testFile = My.Computer.FileSystem.GetFileInfo("C:\TestFolder1\test1.txt")
Dim folderPath As String = testFile.DirectoryName
MsgBox(folderPath)
Dim fileName As String = testFile.Name
MsgBox(fileName)
```

ファイルの名前とディレクトリを結合して完全パスを作成するには

- **CombinePath** メソッドを使用し、ディレクトリと名前を指定します。この例では、前の例で作成した文字列 `folderPath` と `fileName` を受け取って、両者を結合し、結果を表示します。

  VB

```
Dim fullPath As String
fullPath = My.Computer.FileSystem.CombinePath(folderPath, fileName)
MsgBox(fullPath)
```

参照

処理手順

方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する
方法 : Visual Basic でファイルの絶対パスを確認する
方法 : Visual Basic でファイルについての情報を取得する

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.CombinePath メソッド
My.Computer.FileSystem.GetFileInfo メソッド
FileInfo

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

このセクションには、ファイル、ディレクトリ、およびドライブのプロパティを取得および設定する方法についてのトピックが含まれています。

このセクションの内容

方法 : Visual Basic でファイルについての情報を取得する

ファイルについての情報 (最後のアクセス時刻など) を取得する方法を説明します。

方法 : Visual Basic でファイルの絶対パスを確認する

ファイルの絶対パスを確認する方法を説明します。

方法 : Visual Basic でファイルの拡張子を確認する

ファイルの拡張子を確認する方法を説明します。

方法 : Visual Basic でファイルの最終アクセス時刻を確認する

ファイルに最後にアクセスした時刻を確認する方法を説明します。

方法 : Visual Basic でファイルの最終更新時刻を確認する

ファイルを最後に更新した時刻を確認する方法を説明します。

方法 : Visual Basic でファイルのサイズを確認する

ファイルのサイズをバイト単位で確認する方法を説明します。

方法 : Visual Basic でファイルの作成時刻を確認する

ファイルの作成時刻を確認する方法を説明します。

方法 : Visual Basic でファイルが隠しファイルかどうかを調べる

ファイルが隠しファイルかどうかを確認する方法を説明します。

方法 : Visual Basic でファイルが読み取り専用かどうかを確認する

ファイルが読み取り専用かどうかを確認する方法を説明します。

方法 : Visual Basic でファイルの属性を確認する

ファイルの属性を確認する方法を説明します。

方法 : Visual Basic でディレクトリの作成時刻を確認する

ディレクトリの作成時刻を確認する方法を説明します。

方法 : Visual Basic でディレクトリが読み取り専用かどうかを確認する

ディレクトリが読み取り専用かどうかを確認する方法を説明します。

方法 : Visual Basic でディレクトリの属性を確認する

ディレクトリの属性を確認する方法を説明します。

方法 : Visual Basic でファイルが存在するかどうかを確認する

ファイルが存在するかどうかを確認する方法を説明します。

方法 : Visual Basic でディレクトリが存在するかどうかを確認する

ディレクトリが存在するかどうかを確認する方法を説明します。

方法 : Visual Basic でドライブのボリューム ラベルを確認する

ドライブのボリューム ラベルを確認する方法を説明します。

方法 : Visual Basic でドライブのフォーマットを確認する

ドライブのファイル システムの種類を確認する方法を説明します。

方法 : Visual Basic でドライブの種類を確認する

ドライブの種類を確認する方法を説明します。

方法 : Visual Basic でドライブの合計領域を確認する

ドライブの合計領域を確認する方法を説明します。

方法 : Visual Basic でドライブの物理空き容量を確認する

ドライブの物理空き容量を確認する方法を説明します。

方法 : Visual Basic でドライブのルート ディレクトリを確認する

ドライブのルート ディレクトリを確認する方法を説明します。

方法 : Visual Basic で Windows のシステム ディレクトリを確認する

Windows のシステム ディレクトリの位置を確認する方法を説明します。

参照

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem オブジェクトとそのメソッドおよびプロパティについて説明します。

My.Computer.FileSystem.DirectoryExists メソッド

DirectoryExists メソッドについて説明します。

My.Computer.FileSystem.FileExists メソッド

FileExists メソッドについて説明します。

My.Computer.FileSystem.GetDirectoryInfo メソッド

GetDirectoryInfo メソッドについて説明します。

My.Computer.FileSystem.GetFileInfo メソッド

GetFileInfo メソッドについて説明します。

My.Computer.FileSystem.GetParentPath メソッド

GetParentPath メソッドについて説明します。

関連するセクション

FileInfo

FileInfo オブジェクトについて説明します。

DirectoryInfo

DirectoryInfo オブジェクトについて説明します。

FileAttributes

FileAttributes 列挙体について説明します。

Visual Basic でのファイルへの書き込み

ファイルへの書き込みに関連するタスクの一覧を示します。

Visual Basic でのファイルの読み取り

ファイルからの読み込みに関連するタスクの一覧を示します。

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

ディレクトリとファイルの作成、削除、および移動に関連するタスクの一覧を示します。

方法：Visual Basic でファイルについての情報を取得する

My.Computer.FileSystem.GetFileInfo メソッドを使用すると、ファイルのプロパティについての情報を簡単に確認できます。**FileInfo** オブジェクトのプロパティには、属性、作成日時、ディレクトリ、ディレクトリ名、存在するかどうか、拡張子、完全名、最終アクセス日時、最終書き込み日時、長さ、および名前が含まれています。

ファイルが存在しない場合でも例外はスローされません。オブジェクトのプロパティに初めてアクセスしたときにスローされます。

> **メモ：**
>
> 使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

手順

ファイルに関する情報を取得するには

1. **GetFileInfo** メソッドを使用して **FileInfo** オブジェクトを取得します。このオブジェクトを調べることにより、プロパティを確認できます。ファイル `MyLogFile.log` に対応する **FileInfo** オブジェクトを取得する例を次に示します。

   VB

   ```
   Dim information As System.IO.FileInfo
   information = My.Computer.FileSystem.GetFileInfo("C:\MyLogFile.log")
   ```

2. **FileInfo** オブジェクトを調べ、必要な情報を抽出します。次のコードは、ファイルの完全名、最終アクセス日時、および長さを報告します。

   VB

   ```
   MsgBox("The file's full name is " & information.FullName & ".")
   MsgBox("Last access time is " & information.LastAccessTime & ".")
   MsgBox("The length is " & information.Length & ".")
   ```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パス名の形式に誤りがある場合。たとえば、無効な文字が含まれている場合や、空白のみの場合 (ArgumentException)
- ファイルが存在しないか、または **Nothing** である場合 (ArgumentNullException)
- パス文字列の途中にコロンが含まれている場合 (NotSupportedException)
- パスが長すぎる場合 (PathTooLongException)
- ユーザーに必要なアクセス許可がない場合 (SecurityException)
- 当該ファイルに対して、ユーザーがアクセス制御リスト (ACL) のアクセス許可を持っていない場合 (UnauthorizedAccessException)

参照

処理手順

チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

FileInfo

その他の技術情報

Visual Basic におけるファイル アクセス

方法：Visual Basic でファイルの絶対パスを確認する

My.Computer.FileSystem.GetFileInfo メソッドが返す FileInfo オブジェクトを使用すると、ファイルについての情報を確認できます。ファイルの場所は FullName プロパティで取得できます。

ファイルが存在しない場合、**GetFileInfo** では例外はスローされず、**System.IO.FileInfo** オブジェクトのプロパティに初めてアクセスしたときに例外がスローされます。

手順

ファイルの絶対パスを確認するには

- **GetFileInfo** メソッドを使用して、調べる対象のファイルの **FileInfo** オブジェクトを取得します。その **FullName** プロパティに絶対パスが含まれています。次の例は、`Test.txt` の絶対パスを確認し、メッセージ ボックスに表示します。

  VB

```
Dim getInfo As System.IO.FileInfo
getInfo = My.Computer.FileSystem.GetFileInfo("C:\TestFolder1\test.txt")
MsgBox(getInfo.FullName)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パス名の形式に誤りがある場合。たとえば、無効な文字が含まれているか、空白のみである (ArgumentException)。
- ファイルが存在しないか、または **Nothing** である場合 (ArgumentNullException)。
- パス文字列の途中にコロンが含まれている場合 (NotSupportedException)。
- パスが長すぎる場合 (PathTooLongException)
- ユーザーに必要なアクセス許可がない場合 (SecurityException)。
- 当該ファイルに対して、ユーザーがアクセス制御リスト (ACL) のアクセス許可を持っていない (UnauthorizedAccessException)。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.FileSystem.GetFileInfo メソッド

FileInfo

方法：Visual Basic でファイルの拡張子を確認する

My.Computer.FileSystem.GetFiles メソッドは、ファイルの名前を読み取り専用の文字列のコレクションとして返します。これを、Split 関数 (Visual Basic) で解析します。

ファイルの拡張子を確認するには

- ファイルのパスまたは名前を取得し、**Split** 関数に区切り記号を指定して、拡張子を確認します。この例では、**GetFiles** メソッドを使用して、`testDirectory` ディレクトリに格納されているファイル名のコレクションを取得し、各ファイルの拡張子を報告します。

  VB

```
For Each foundFile As String In _
My.Computer.FileSystem.GetFiles("C:\TestDir")
    Dim check As String = _
    System.IO.Path.GetExtension(foundFile)
    MsgBox("The file extension is " & check)
Next
```

参照

処理手順

方法 : 文字列の配列内で文字列を検索する (Visual Basic)

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.FileSystem.GetFiles メソッド

Path

方法：Visual Basic でファイルの最終アクセス時刻を確認する

My.Computer.FileSystem.GetFileInfo メソッドを使用すると、指定されたファイルに関する情報を格納している FileInfo オブジェクトを取得できます。

LastAccessTime プロパティまたは **LastAccessTimeUtc** プロパティを使用すると、最終アクセス時刻を取得または設定できます。**LastAccessTimeUtc** では、時刻は世界協定時刻 (UTC) で示されます。詳細については、「日付と時刻の書式指定文字列」を参照してください。

> **メモ：**
>
> 使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ファイルの最終アクセス時刻を確認するには

- **GetFileInfo** メソッドを使用して、ファイルの **FileInfo** オブジェクトを取得し、このオブジェクトに照会して情報を取得します。この例では、`Testfile.txt` の **FileInfo** オブジェクトを取得し、**LastAccessTime** プロパティを使用して、最終アクセス時刻を表示します。

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
MsgBox("File was last accessed on " & infoReader.LastAccessTime)
```

参照

処理手順

方法 : Visual Basic でファイルの属性を確認する
方法 : Visual Basic でファイルの作成時刻を確認する

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.GetFileInfo メソッド

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動
Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

方法：Visual Basic でファイルの最終更新時刻を確認する

My.Computer.FileSystem.GetFileInfo メソッドを使用すると、指定したファイルに関する情報を格納している FileInfo オブジェクトを取得できます。

LastWriteTime プロパティまたは **LastWriteTimeUtc** プロパティを使用すると、ファイルが最後に更新された時刻を取得または設定できます。**LastWriteTimeUtc** では、時刻は世界協定時刻 (UTC) で示されます。詳細については、「日付と時刻の書式指定文字列」を参照してください。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ファイルの最終更新時刻を確認するには

- **GetFileInfo** メソッドを使用して、ファイルの **FileInfo** オブジェクトを取得し、このオブジェクトを照会して情報を取得します。この例では、`Testfile.txt` の **FileInfo** オブジェクトを取得し、**LastWriteTime** プロパティを使用して、最終アクセス時刻を表示します。

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
MsgBox("File was last modified on " & infoReader.LastWriteTime)
```

参照

処理手順

方法 : Visual Basic でファイルの属性を確認する
方法 : Visual Basic でファイルの作成時刻を確認する

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.GetFileInfo メソッド

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

方法：Visual Basic でファイルのサイズを確認する

My.Computer.FileSystem.GetFileInfo メソッドを使用すると、指定したファイルに関する情報を格納している FileInfo オブジェクトを取得できます。

Length プロパティを使用すると、ファイルのサイズをバイト単位で確認できます。

メモ：
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ファイルのサイズを確認するには

- **GetFileInfo** メソッドを使用して、ファイルの **FileInfo** オブジェクトを取得し、このオブジェクトを照会して情報を取得します。この例では、`Testfile.txt` の **FileInfo** オブジェクトを取得し、**Length** プロパティを使用して、ファイルのサイズをバイト単位で表示します。

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
MsgBox("File is " & infoReader.Length & " bytes.")
```

参照

処理手順

方法 : Visual Basic でファイルの属性を確認する

関連項目

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem.GetFileInfo メソッド

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

方法：Visual Basic でファイルの作成時刻を確認する

My.Computer.FileSystem.GetFileInfo メソッドを使用すると、指定したファイルに関する情報を格納している FileInfo オブジェクトを取得できます。

CreationTime プロパティまたは **CreationTimeUtc** プロパティを使用すると、最終アクセス時刻を取得または設定できます。**CreationTimeUtc** では、時刻は世界協定時刻 (UTC) で示されます。詳細については、「日付と時刻の書式指定文字列」を参照してください。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ファイルの作成時刻を確認するには

- **GetFileInfo** メソッドを使用して、ファイルの **FileInfo** オブジェクトを取得し、このオブジェクトを照会して情報を取得します。この例では、`Testfile.txt` の **FileInfo** オブジェクトを取得し、**CreationTime** プロパティを使用して、作成時刻を表示します。

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
MsgBox("File was created on " & infoReader.CreationTime)
```

参照

処理手順

方法 : Visual Basic でファイルの属性を確認する

方法 : Visual Basic でファイルの最終アクセス時刻を確認する

関連項目

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem.GetFileInfo メソッド

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

方法：Visual Basic でファイルが隠しファイルかどうかを調べる

My.Computer.FileSystem.GetFileInfo メソッドを使用すると、指定されたファイルに関する情報を格納している FileInfo オブジェクトを取得でき、その中に FileAttributes 列挙体が含まれています。

ファイルが隠しファイルかどうかを確認するには

1. 調べるファイルに対応する **FileInfo** オブジェクトを取得します。この例では、`Testfile.txt` ファイルに対応する **FileInfo** を取得します。

   VB

   ```
   Dim infoReader As System.IO.FileInfo
   infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
   ```

2. **FileInfo** オブジェクトから **FileAttributes** オブジェクトを取得します。この例では、**FileInfo** オブジェクトから **FileAttributes** を取得します。

   VB

   ```
   Dim attributeReader As System.IO.FileAttributes
   attributeReader = infoReader.Attributes
   ```

3. **FileAttributes** を照会して、ファイルが隠しファイルかどうかを確認します。この例では、ファイルが隠しファイルかどうかを確認し、それに応じた結果を表示します。

   VB

   ```
   If (attributeReader And System.IO.FileAttributes.Hidden) > 0 Then
       MsgBox("File is hidden!")
   Else
       MsgBox("File is not hidden!")
   End If
   ```

参照

処理手順

方法 : Visual Basic でファイルの属性を確認する

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem オブジェクトのメンバ
My.Computer.FileSystem.GetFileInfo メソッド

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

方法：Visual Basic でファイルが読み取り専用かどうかを確認する

My.Computer.FileSystem.GetFileInfo メソッドを使用すると、指定したファイルに関する情報を格納している FileInfo オブジェクトを取得できます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ファイルが読み取り専用かどうかを確認するには

- **GetFileInfo** メソッドを使用して、ファイルの **FileInfo** オブジェクトを取得し、このオブジェクトを照会して情報を取得します。この例では、`Testfile.txt` に対応する **FileInfo** オブジェクトを取得し、**IsReadOnly** プロパティが **True** に設定されているかどうかに応じて、適切なメッセージを表示します。

  VB

```
Dim infoReader As System.IO.FileInfo
infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
If infoReader.IsReadOnly = True Then
    MsgBox("File is readonly!")
End If
```

参照

処理手順

方法 : Visual Basic でディレクトリが読み取り専用かどうかを確認する

関連項目

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem.GetFileInfo メソッド

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

方法：Visual Basic でファイルの属性を確認する

My.Computer.FileSystem.GetFileInfo メソッドを使用すると、指定されたファイルに関する情報を格納している FileInfo オブジェクトを取得でき、その中に FileAttributes 列挙体が含まれています。

次の表は、**FileAttributes** のメンバを示します。

メンバ	説明
Archive	ファイルのアーカイブ ステータスです。アプリケーションは、ファイルにバックアップまたは削除のマークを付けるために、この属性を使います。
Compressed	ファイルは圧縮ファイルです。
Device	このメンバは現時点では使用されていません。
Directory	ファイルはディレクトリです。
Encrypted	ファイルのすべてのデータは暗号化されています。
Hidden	ファイルは隠しファイルであるため、通常のディレクトリー覧には表示されません。
Normal	ファイルには、他の属性は設定されていません。
NotContentIndexed	ファイルは、オペレーティング システムによるコンテンツ インデックス サービスでインデックスされません。
Offline	ファイルはオフラインです。ファイルのデータはすぐには利用できません。
ReadOnly	ファイルは読み取り専用です。
ReparsePoint	ファイルにリパース ポイントが含まれています。リパース ポイントとは、ユーザー定義のデータのブロックです。
SparseFile	ファイルはスパース ファイルです。スパース ファイルは、通常は大きなファイルで、含まれているデータの大半はゼロです。
System	ファイルはシステム ファイルです。このファイルは、オペレーティング システムの一部であるか、またはオペレーティング システムが排他的に使用します。
Temporary	ファイルは一時ファイルです。ファイル システムは、データを大容量ストレージにフラッシュするのではなく、すべてのデータをメモリに保持するよう試みて、すばやくアクセスできるようにします。一時ファイルが不要になったら、アプリケーションが直ちに削除する必要があります。

ファイルが暗号化されているかどうかを確認するには

1. 調べるファイルに対応する **FileInfo** オブジェクトを取得します。この例では、`Testfile.txt` ファイルに対応する **FileInfo** を取得します。

   VB

   ```
   Dim infoReader As System.IO.FileInfo
   infoReader = My.Computer.FileSystem.GetFileInfo("C:\testfile.txt")
   ```

2. **FileInfo** オブジェクトから **FileAttributes** オブジェクトを取得します。この例では、**FileInfo** オブジェクトから **FileAttributes** を取得します。

   VB

```
Dim attributeReader As System.IO.FileAttributes
attributeReader = infoReader.Attributes
```

3. **FileAttributes** を照会します。この例では、ファイルが暗号化されているかどうかを確認し、それに応じた結果を表示します。

   VB

```
If (attributeReader And System.IO.FileAttributes.Encrypted) > 0 Then
    MsgBox("File is encrypted!")
Else
    MsgBox("File is not encrypted!")
End If
```

参照

処理手順

方法 : Visual Basic でファイルが隠しファイルかどうかを調べる

関連項目

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem.GetFileInfo メソッド

FileAttributes

FileInfo

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

方法：Visual Basic でディレクトリの作成時刻を確認する

My.Computer.FileSystem.GetDirectoryInfo メソッドは DirectoryInfo オブジェクトを返します。このオブジェクトを照会して、ディレクトリについての情報を取得できます。

ディレクトリが存在しない場合でも、**DirectoryInfo** オブジェクトのプロパティに初めてアクセスするまでは、例外はスローされません。

メモ：
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ディレクトリの作成時刻を確認するには

- **GetDirectoryInfo** メソッドを使用して、目的のディレクトリに対応する **DirectoryInfo** オブジェクトを取得し、CreationTime プロパティを照会します。この例では、 `C:\Documents and Settings` の **CreationTime** を表示します。

  VB

```
Dim getInfo As System.IO.DirectoryInfo
getInfo = My.Computer.FileSystem.GetDirectoryInfo _
("C:\Documents and Settings")
MsgBox("The directory was created at " & getInfo.CreationTime)
```

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。

参照

処理手順

方法 : Visual Basic でディレクトリが存在するかどうかを確認する
トラブルシューティング : テキスト ファイルの読み取りと書き込み

関連項目

My.Computer.FileSystem.GetDirectoryInfo メソッド
DirectoryInfo
CreationTime

方法：Visual Basic でディレクトリが読み取り専用かどうかを確認する

My.Computer.FileSystem.GetDirectoryInfo メソッド メソッドは DirectoryInfo オブジェクトを返します。このオブジェクトには Attributes プロパティがあり、これを照会すると、ディレクトリについての情報を確認できます。ディレクトリが読み取り専用かどうかもその中に含まれています。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ディレクトリが読み取り専用かどうかを確認するには

1. **GetDirectoryInfo** メソッドを使用して、指定のディレクトリに対応する **DirectoryInfo** オブジェクトを取得します。この例では、`TestDirectory` ディレクトリに対応する **DirectoryInfo** を取得します。

   VB

   ```
   Dim reader As System.IO.DirectoryInfo
   reader = My.Computer.FileSystem.GetDirectoryInfo("C:\testDirectory")
   ```

2. オブジェクトの **Attributes** プロパティを照会して、読み取り専用かどうかを確認します。

   VB

   ```
   If (reader.Attributes And System.IO.FileAttributes.ReadOnly) > 0 Then
       MsgBox("Directory is readonly!")
   End If
   ```

使用例

次の例は、上記のスニペットを完全な形式で表したものです。`testDirectory` ディレクトリが読み取り専用かどうかを確認し、その結果をメッセージ ボックスで報告します。

VB

```
Dim reader As System.IO.DirectoryInfo
reader = My.Computer.FileSystem.GetDirectoryInfo("C:\testDirectory")
If (reader.Attributes And System.IO.FileAttributes.ReadOnly) > 0 Then
    MsgBox("File is readonly!")
End If
```

コードのコンパイル方法

ディレクトリが存在しない場合でも、**DirectoryInfo** オブジェクトのプロパティに初めてアクセスするまでは、例外はスローされません。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられる (ArgumentException)。
- パスが **Nothing** であるため、有効でない (ArgumentNullException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効である (NotSupportedException)。
- ユーザーがパスを参照するのに必要なアクセス許可がない (SecurityException)。

参照

処理手順

方法 : Visual Basic でディレクトリの属性を確認する

関連項目

My.Computer.FileSystem.GetDirectoryInfo メソッド

方法：Visual Basic でディレクトリの属性を確認する

My.Computer.FileSystem.GetDirectoryInfo メソッド メソッドは DirectoryInfo オブジェクトを返します。その Attributes プロパティを調べると、ディレクトリについての情報を確認できます。

次の表は、**Attributes** プロパティで使用する FileAttributes 列挙体のメンバの一覧です。

メンバ	数値	説明
ReadOnly	1	ファイルは読み取り専用です。
Hidden	2	ファイルは隠しファイルであるため、通常のディレクトリ一覧には表示されません。
System	4	ファイルはシステム ファイルです。このファイルは、オペレーティング システムの一部であるか、またはオペレーティング システムが排他的に使用します。
Directory	16	ファイルはディレクトリです。
Archive	32	ファイルのアーカイブ ステータスです。アプリケーションは、ファイルにバックアップまたは削除のマークを付けるために、この属性を使います。
Device	64	使用しません。
Normal	128	ファイルは通常のファイルで、他の属性は設定されていません。この属性は、単独で使用された場合だけ有効です。
Temporary	256	ファイルは一時ファイルです。ファイル システムは、すべてのデータをメモリに保持するよう試みて、すばやくアクセスできるようにします。一時ファイルは、不要になったときには削除する必要があります。
SparseFile	512	ファイルはスパース ファイルです。スパース ファイルは、通常は大きなファイルで、含まれているデータの大半はゼロです。
ReparsePoint	1024	ファイルにリパース ポイントが含まれています。リパース ポイントとは、ファイルまたはディレクトリと関連付けられている、ユーザー定義のデータのブロックです。
Compressed	2048	ファイルは圧縮ファイルです。
Offline	4096	ファイルはオフラインで、そのデータはすぐには利用できません。
NotContentIndexed	8192	ファイルは、オペレーティング システムによるコンテンツ インデックス サービスでインデックスされません。
Encrypted	16384	ファイルまたはディレクトリは暗号化されています。ファイルの場合は、ファイル内のすべてのデータが暗号化されていることを示します。ディレクトリの場合は、新たに作成されるファイルとディレクトリが既定で暗号化されることを示します。

ディレクトリが隠しディレクトリかどうかを判断するには

- **GetDirectoryInfo** メソッドを使用して、**DirectoryInfo** オブジェクトを取得します。この例では、`TestDir` ディレクトリの **DirectoryInfo** を取得し、その **DirectoryInfo** オブジェクトから **FileAttributes** オブジェクトを取得して、これをチェックすることで、隠しディレクトリかどうかを判断します他の属性も同様の方法で調べることができます。

  VB

```
Dim checkFile As System.IO.DirectoryInfo
checkFile = My.Computer.FileSystem.GetDirectoryInfo("C:\TestDir")
Dim attributeReader As System.IO.FileAttributes
attributeReader = checkFile.Attributes

If (attributeReader And System.IO.FileAttributes.Hidden) > 0 Then
    MsgBox("Directory is hidden")
End If
```

参照

処理手順

方法 : Visual Basic でファイルの属性を確認する

関連項目

My.Computer.FileSystem.GetDirectoryInfo メソッド

DirectoryInfo

FileAttributes

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

方法：Visual Basic でファイルが存在するかどうかを確認する

My.Computer.FileSystem.FileExists メソッドを使用すると、指定したファイルが存在するかどうかを確認できます。

指定したファイルの読み取りに必要なアクセス許可がアプリケーションに与えられていない場合、パスが存在していても、**FileExists** メソッドは **False** を返します。例外はスローされません。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

手順

ファイルが存在するかどうかを確認するには

- **My.Computer.FileSystem.FileExists** メソッドにパスを指定し、そのファイルが存在するかどうかを確認します。次の例では、`Check.txt` ファイルが存在するかどうかを確認し、その情報をメッセージ ボックスで提供します。

  VB

```
If My.Computer.FileSystem.FileExists("c://Check.txt") Then
    MsgBox("File found.")
Else
    MsgBox("File not found.")
End If
```

参照

処理手順

方法 : Visual Basic でディレクトリが存在するかどうかを確認する
トラブルシューティング : テキスト ファイルの読み取りと書き込み
チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

My.Computer.FileSystem.FileExists メソッド

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

方法：Visual Basic でディレクトリが存在するかどうかを確認する

My.Computer.FileSystem.DirectoryExists メソッドを使用して、指定したディレクトリが存在するかどうかを確認できます。

DirectoryExists の呼び出しでは、FileIOPermission が必要です。

このメソッドは例外をスローしません。

> **メモ：**
>
> 使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

手順

ディレクトリが存在するかどうかを確認するには

- **My.Computer.FileSystem.DirectoryExists** メソッドにパスを指定し、そのディレクトリが存在するかどうかを確認します。次の例では、`C:\backup\logs` ディレクトリが存在するかどうかを確認し、そのプロパティをチェックします。

  VB

```
Dim logDirectoryProperties As System.IO.DirectoryInfo
If My.Computer.FileSystem.DirectoryExists("C:\backup\logs") Then
    logDirectoryProperties = My.Computer.FileSystem.GetDirectoryInfo("C:\backup\logs")
End If
```

参照

処理手順

方法 : Visual Basic でファイルが存在するかどうかを確認する
トラブルシューティング : テキスト ファイルの読み取りと書き込み
チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

My.Computer.FileSystem.DirectoryExists メソッド

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

方法：Visual Basic でドライブのボリューム ラベルを確認する

My.Computer.FileSystem.GetDriveInfo メソッドは、コンピュータ上のドライブに関する情報の問い合わせに使用できる DriveInfo オブジェクトを提供します。VolumeLabel プロパティを使用して、ドライブのラベルを確認できます。必要なアクセス許可を持たずにこのプロパティにアクセスしようとすると、SecurityException 例外がスローされます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ドライブのボリューム ラベルを確認するには

- **VolumeLabel** プロパティを使用してドライブのラベルを確認します。次のコードはその例です。

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.VolumeLabel)
```

参照

処理手順

方法 : Visual Basic でドライブの合計領域を確認する
方法 : Visual Basic でドライブのフォーマットを確認する
方法 : Visual Basic でドライブの種類を確認する

関連項目

My.Computer.FileSystem.GetDriveInfo メソッド

方法：Visual Basic でドライブのフォーマットを確認する

My.Computer.FileSystem.GetDriveInfo メソッドは、コンピュータ上のドライブに関する情報の問い合わせに使用できる DriveInfo オブジェクトを提供します。DriveFormat プロパティを使用して、ドライブのフォーマットを確認できます。必要なアクセス許可を持たずにこのプロパティにアクセスしようとすると、SecurityException 例外がスローされます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ドライブのフォーマットを確認するには

- **DriveFormat** プロパティを使用してドライブのフォーマットを確認します。次のコードはその例です。

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.DriveFormat)
```

参照

処理手順

方法 : Visual Basic でドライブの合計領域を確認する
方法 : Visual Basic でドライブのボリューム ラベルを確認する
方法 : Visual Basic でドライブの種類を確認する

関連項目

My.Computer.FileSystem.GetDriveInfo メソッド

方法：Visual Basic でドライブの種類を確認する

My.Computer.FileSystem.GetDriveInfo メソッドは、コンピュータ上のドライブに関する情報の問い合わせに使用できる DriveInfo オブジェクトを提供します。**DriveType** プロパティを使用して、ドライブの種類を確認できます。必要なアクセス許可を持たずにこのプロパティにアクセスしようとすると、SecurityException 例外がスローされます。

DriveType 列挙体のメンバを次の表に示します。

メンバ	値	説明
Unknown	0	ドライブの種類を確認できません。
CDRom	1	ドライブは CD-ROM です。読み取り専用の CD-ROM ドライブと読み書き可能な CD-ROM ドライブは区別されません。
DVDRom	2	ドライブは DVD-ROM です。
Fixed	3	ドライブは固定の (リムーバブルでない) メディアを持ちます。この値には、すべてのハード ドライブが含まれます。リムーバブルなハード ドライブかどうかは関係ありません。
RamDisk	4	ドライブは、ローカル コンピュータ上のランダム アクセス メモリ (RAM) のブロックで、ディスク ドライブのように機能します。
Remote	5	ネットワーク ドライブです。これには、ネットワーク上の任意の場所で共有されているドライブが含まれます。
Removable	6	ドライブはリムーバブル メディアを持ちます。これには、すべてのフロッピー ディスク ドライブが含まれます。

ドライブの種類を確認するには

- **DriveType** プロパティを使用して種類を確認します。次のコードはその例です。

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.DriveType.ToString)
```

参照

処理手順

方法 : Visual Basic でドライブの合計領域を確認する
方法 : Visual Basic でドライブのボリューム ラベルを確認する

関連項目

My.Computer.FileSystem.GetDriveInfo メソッド
DriveInfo

方法：Visual Basic でドライブの合計領域を確認する

My.Computer.FileSystem.GetDriveInfo メソッドは、コンピュータ上のドライブに関する情報の問い合わせに使用できる DriveInfo オブジェクトを提供します。**TotalSize** プロパティを使用して、ドライブの領域の容量を確認できます。必要なアクセス許可を持たずにこのプロパティにアクセスしようとすると、SecurityException 例外がスローされます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ドライブの合計ディスク容量を確認するには

- **TotalSize** プロパティを使用してドライブの領域の合計容量を確認します。次のコードはその例です。

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.TotalSize)
```

参照

処理手順

方法 : Visual Basic でドライブの物理空き容量を確認する
方法 : Visual Basic でドライブのボリューム ラベルを確認する
方法 : Visual Basic でドライブの種類を確認する

関連項目

My.Computer.FileSystem.GetDriveInfo メソッド
DriveInfo

方法：Visual Basic でドライブの物理空き容量を確認する

My.Computer.FileSystem.GetDriveInfo メソッドは、コンピュータ上のドライブに関する情報の問い合わせに使用できる DriveInfo オブジェクトを提供します。**TotalFreeSpace** プロパティを使用して、ドライブの空き領域の容量を確認できます。必要なアクセス許可を持たずにこのプロパティにアクセスしようとすると、SecurityException 例外がスローされます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ドライブの空き領域の容量を確認するには

- **TotalFreeSpace** プロパティを使用して空き領域の容量を確認します。次のコードはその例です。

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox("Total free space: " & CStr(cdrive.TotalFreeSpace))
```

  このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、コード例は [ファイル システム - ドライブ、フォルダ、およびファイルの処理] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

参照

処理手順

方法 : Visual Basic でドライブの合計領域を確認する
方法 : Visual Basic でドライブのフォーマットを確認する
方法 : Visual Basic でドライブの種類を確認する

関連項目

My.Computer.FileSystem.GetDriveInfo メソッド
DriveInfo

方法：Visual Basic でドライブのルート ディレクトリを確認する

My.Computer.FileSystem.GetDriveInfo メソッドは、コンピュータ上のドライブに関する情報の問い合わせに使用できる DriveInfo オブジェクトを提供します。**RootDirectory** プロパティを使用すると、ルート ディレクトリを確認できます。必要なアクセス許可を持たずにこのプロパティにアクセスしようとすると、SecurityException 例外がスローされます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

ドライブのルート ディレクトリを確認するには

- **RootDirectory** プロパティを使用してルート ディレクトリを確認します。次のコードはその例です。

  VB

```
Dim cdrive As System.IO.DriveInfo
cdrive = My.Computer.FileSystem.GetDriveInfo("C:\")
MsgBox(cdrive.RootDirectory)
```

参照

処理手順

方法 : Visual Basic でドライブの合計領域を確認する
方法 : Visual Basic でドライブの種類を確認する

関連項目

My.Computer.FileSystem.GetDriveInfo メソッド
DriveInfo

方法：Visual Basic で Windows のシステム ディレクトリを確認する

この例は、Windows のシステム ディレクトリへのパスを示す文字列を設定します。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置が、ヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

使用例

VB

```
Dim systemDirectory As String
systemDirectory = System.Environment.SystemDirectory
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Windows Operating System] の [System Information] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

このプロパティは "c:\WINNT\System32" などの値を返します。

セキュリティ

このコードには、FileIOPermission クラスが必要です。

参照

処理手順

方法 : Visual Basic でディレクトリの属性を確認する

関連項目

Environment.GetFolderPath Method

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser オブジェクトを使用すると、ログ ファイルやレガシ データベース情報など、区切り文字や幅に応じて複数列に区切られたテキストとして構造化されている巨大なファイルを解析および処理できます。テキスト ファイルを **TextFieldParser** で解析するのは、テキスト ファイルを反復処理するのと同様です。解析メソッドでテキストのフィールドを抽出するのは、区切り文字の付いた文字列を文字列操作メソッドでトークン化するのと同様です。

さまざまな種類のテキスト ファイルの解析

テキスト ファイルは、コンマやタブ空白などの文字で可変幅のフィールドに区切られている場合があります。次の例のように、**TextFieldType** および区切り記号を定義します。この例では、**SetDelimiters** メソッドを使用して、タブ区切りのテキスト ファイルを定義しています。

VB

```
testReader.SetDelimiters(vbTab)
```

また、テキスト ファイルによっては、固定幅のフィールドを持つ場合もあります。その場合には、次の例のように、**TextFieldType** を **FixedWidth** と定義し、各フィールドの幅を定義する必要があります。この例では、**SetFieldWidths** メソッドを使用して、テキストの列を定義しています。最初は幅が 5 文字、その次は 10、その次は 11、その次は可変幅です。

VB

```
testReader.SetFieldWidths(5, 10, 11, -1)
```

書式を定義したら、ファイルをループし、**ReadFields** メソッドを使用して、各行を順番に処理します。

フィールドが指定の書式に一致しない場合には、MalformedLineException 例外がスローされます。この例外がスローされたときには、**ErrorLine** プロパティと **ErrorLineNumber** プロパティに、例外の原因になったテキストと、そのテキストの行番号が保持されています。

複数の書式を持つファイルの解析

TextFieldParser オブジェクトの **PeekChars** メソッドを使用すると、各フィールドを読み取り前にチェックできます。これにより、フィールドに対して複数の書式を定義して、適切に対応できます。詳細については、「方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る」を参照してください。

参照

処理手順

例外のトラブルシューティング : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

関連項目

My.Computer.FileSystem.OpenTextFieldParser メソッド
TextFieldParser オブジェクト
TextFieldParser.PeekChars メソッド
TextFieldParser.ReadFields メソッド
TextFieldParser.CommentTokens プロパティ
TextFieldParser.Delimiters プロパティ
TextFieldParser.ErrorLine プロパティ
TextFieldParser.ErrorLineNumber プロパティ
TextFieldParser.FieldWidths プロパティ
TextFieldParser.HasFieldsEnclosedInQuotes プロパティ
TextFieldParser.LineNumber プロパティ
TextFieldParser.TextFieldType プロパティ
TextFieldParser.TrimWhiteSpace プロパティ
TextFieldParser.SetDelimiters メソッド
TextFieldParser.SetFieldWidths メソッド

ファイル エンコーディング

ファイル エンコーディングは、文字エンコーディングとも呼ばれ、テキスト処理での文字の表現方法を指定するものです。どの言語の文字を処理できるか (またはできないか) という点から見て、あるエンコーディングが別のエンコーディングより好ましい場合もありますが、通常は Unicode が好まれます。

ファイルに対する読み取りと書き込みでは、ファイル エンコーディングがきちんと一致しないと、例外が発生したり、不正な結果となる場合があります。

エンコーディングの種類

ファイルを扱うときには、Unicode は好ましいエンコーディングです。Unicode は、世界共通の文字エンコーディング規則で、技術的な記号や出版で使用される特殊文字などを含む、現在のコンピューティングで使用されるすべての文字を、16 ビットのコード値で表します。

これまでの文字エンコーディング規則は、Windows ANSI 文字セットなどの従来からある文字セットで構成されており、8 ビットのコード値や 8 ビット値の組み合わせを使用して、特定の言語や地域で使用する文字を表していました。

Encoding クラス

Encoding クラスは文字エンコーディングを表します。次の表は、利用可能なエンコーディングの型の一覧とそれぞれの説明です。

名前	説明
ASCIIEncoding	Unicode 文字の ASCII 文字エンコーディングを表します。
UnicodeEncoding	Unicode 文字の UTF-16 エンコーディングを表します。
UTF32Encoding	Unicode 文字の UTF-32 エンコーディングを表します。
UTF7Encoding	Unicode 文字の UTF-7 エンコーディングを表します。
UTF8Encoding	Unicode 文字の UTF-8 エンコーディングを表します。

参照

その他の技術情報

Visual Basic でのファイルの読み取り

Visual Basic でのファイルへの書き込み

トラブルシューティング : テキスト ファイルの読み取りと書き込み

このトピックでは、テキスト ファイルを扱うときに生じる一般的な問題について説明し、それぞれの対処方法を示します。

一般的な問題

テキスト ファイルを扱うときに特によくある問題点には、セキュリティ例外、ファイル エンコーディング、および無効なパスがあります。

セキュリティ例外

セキュリティ エラーが発生すると SecurityException がスローされます。これは、ユーザーに必要なアクセス許可がないことが原因で生じる場合が多く、アクセス許可を追加するか、またはファイルを分離ストレージで扱うことにより解決できることがあります。詳細については、「例外のトラブルシューティング : System.Security.SecurityException」および「分離ストレージ」を参照してください。

ファイル エンコーディング

ファイル エンコーディングは、文字エンコーディングとも呼ばれ、テキスト処理での文字の表現方法を指定するものです。テキスト ファイルに予期しない文字が含まれていると、エンコーディングが不正となる場合があります。大半のファイルでは、どの言語の文字を処理できるか (またはできないか) という点から見て、あるエンコーディングが別のエンコーディングより好ましいという場合があります。ただし通常は Unicode が好まれます。詳細については、「ファイル エンコーディング」および「Encoding」を参照してください。

不正なパス

ファイル パス (特に相対パス) を解析するときには、間違ったデータを指定しがちです。多くの問題は、正しいパスを指定しているかどうかを確認することで修正できます。詳細については、「方法 : Visual Basic でファイル パスを解析する」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

その他の技術情報

Visual Basic でのファイルの読み取り

Visual Basic でのファイルへの書き込み

チュートリアル：.NET Framework のメソッドによるファイル操作

このチュートリアルでは、StreamReader クラスによるファイルのオープンと読み取り、ファイルがアクセス中かどうかのチェック、**StreamReader** クラスのインスタンスによって読み取ったファイルからの文字列の検索、StreamWriter クラスによるファイルへの書き込みの方法を説明します。

メモ：
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

アプリケーションの作成

Visual Studio を起動し、ユーザーが指定のファイルに書き込みを行うためのフォームを作成して、プロジェクトを開始します。

プロジェクトを作成するには

1. [ファイル] メニューの [新しいプロジェクト] をクリックします。
2. 新しいプロジェクト ペインの [Windows アプリケーション] をクリックします。
3. [プロジェクト名] ボックスに「**MyDiary**」と入力し、[OK] をクリックします。

   Visual Studio によってプロジェクトがソリューション エクスプローラに追加され、Windows フォーム デザイナが表示されます。
4. 次の表に示すコントロールをフォームに追加し、対応する値をプロパティに設定します。

オブジェクト	プロパティ	値
Button	Name Text	**Submit** **Submit Entry**
Button	Name Text	**Clear** **Clear Entry**
TextBox	Name Text Multiline	**Entry** **Please enter something.** **False**

ファイルへの書き込み

ファイルへの書き込み機能をアプリケーションに追加するには、**StreamWriter** クラスを使用します。**StreamWriter** は、特定のエンコーディングによる文字出力用としてデザインされています。一方、Stream クラスは、バイトの入出力用としてデザインされています。標準のテキスト ファイルにデータ行を書き込むには、**StreamWriter** を使用します。**StreamWriter** クラスの詳細については、「StreamWriter クラス」を参照してください。

書き込み機能を追加するには

1. [表示] メニューの [コード] をクリックして、コード エディタを開きます。
2. アプリケーションが System.IO 名前空間を参照できるように、コードの最初、つまり `Public Class Form1` で始まるフォームのクラス宣言の前に、次のステートメントを追加します。

   VB

```
Imports System
Imports System.IO
```

   ファイルに書き込む前に、**StreamWriter** クラスのインスタンスを作成する必要があります。

3. [表示] メニューの [デザイナ] をクリックして、Windows フォーム デザイナに戻ります。`Submit` ボタンをダブルクリックしてボタンの Click イベント ハンドラを作成し、次のコードを追加します。

VB

```
Dim fw As StreamWriter


```

メモ：

コード エディタに戻り、コードを追加するイベント ハンドラにカーソルを配置する動作は、Visual Studio 統合開発環境 (IDE: Integrated Development Environment) によって行われます。

1. ファイルに書き込むには、**StreamWriter** クラスの Write メソッドを使用します。`Dim fw As StreamWriter` の直後に次のコードを追加します。ファイルが見つからない場合に例外がスローされることはありません。まだ存在していない場合は、ファイルが作成されます。

VB

```
Dim ReadString As String
Try
    'Pass the file path and name to the StreamWriter constructor.
    'Indicate that Append is True, so file will not be overwritten.
    fw = New StreamWriter("C:\MyDiary.txt", True)
    ReadString = Entry.Text
    fw.WriteLine(ReadString)
Finally
    'Close the file.
    fw.Close()
End Try


```

2. ユーザーが何も入力しないで [Submit Entry] をクリックすることがないように、`Dim ReadString As String` の直後に次のコードを追加します。

VB

```
If (Entry.Text = "" Or Entry.Text = "Please enter something.") Then
    Entry.Text = "Please enter something."
    Return
End If


```

3. これは日記なので、入力のたびに日付が入るようにします。`Today` 変数を現在の日付に設定するために、`fw = New StreamWriter("C:\MyDiary.txt", True)` の後に次のコードを挿入します。

VB

```
Dim Today As DateTime
Today = Now
fw.Write(CStr(Today))
fw.Write(ControlChars.CrLf)


```

4. 最後に、**TextBox** をクリアするコードを追加します。`Clear` ボタンの **Click** イベントに次のコードを追加します。

VB

```
Entry.Text = ""


```

日記の表示機能の追加

このセクションでは、`DisplayEntry` **TextBox** に最新のエントリを表示する機能を追加します。また、さまざまなエントリを表示する ComboBox を追加し、ユーザーがそこからエントリを選択して `DisplayEntry` **TextBox** に表示できるようにします。**StreamReader** クラスのインスタンスでは、`MyDiary.txt` から読み込みを行います。**StreamReader** は、**StreamWriter** クラスと同じく、テキスト ファイルに対して使用するためのものです。

ここでは、次の表に示すコントロールをフォームに追加し、対応する値をプロパティに設定します。

コントロール	プロパティ	値
TextBox	Name Visible Size Multiline	**DisplayEntry** **False** **120,60** **True**
Button	Name Text	**Display** **Display**
Button	Name Text	**GetEntries** **Get Entries**
ComboBox	Name Text Enabled	**PickEntries** **Select an Entry** **False**

コンボ ボックスの内容を設定するには

1. `PickEntries` **ComboBox** を使用して、ユーザーが各エントリを送信 (Submit) した日付を表示します。ユーザーは、そこから特定の日付のエントリを選択できます。`GetEntries` ボタンの **Click** イベント ハンドラを作成し、次のコードを追加します。

   VB

   ```
   Dim fr As StreamReader
   Dim FileString As String
   FileString = ""
   Try
       fr = New System.IO.StreamReader("C:\MyDiary.txt")
       PickEntries.Items.Clear()
       PickEntries.Enabled = True
       Do
           FileString = fr.ReadLine
           If IsDate(FileString) Then
               PickEntries.Items.Add(FileString)
           End If
       Loop Until (FileString Is Nothing)
   Finally
       If fr IsNot Nothing Then
           fr.Close()
       End If
   End Try
   PickEntries.Enabled = True
   ```

2. コードをテストするには、F5 キーを押してアプリケーションをコンパイルし、[Get Entries] をクリックします。**ComboBox** のドロップダウン矢印をクリックして、エントリの日付を表示します。

個別の入力内容を選択して表示するには

1. `Display` ボタンの **Click** イベント ハンドラを作成し、次のコードを追加します。

   VB

```
Dim fr As StreamReader
Dim ReadString As String
'Make sure ReadString begins empty.
ReadString = ""
Dim FileString As String
fr = New StreamReader("C:\MyDiary.txt")
'If no entry has been selected, show the whole file.
If PickEntries.Enabled = False Or PickEntries.SelectedText Is Nothing Then
    Do
        'Read a line from the file into FileString.
        FileString = fr.ReadLine
        'add it to ReadString
        ReadString = ReadString & ControlChars.CrLf & FileString
    Loop Until (FileString = Nothing)
Else
    'An entry has been selected, find the line that matches.
    Do

        FileString = fr.ReadLine
    Loop Until FileString = CStr(PickEntries.SelectedItem)
    FileString = CStr(PickEntries.SelectedItem) & ControlChars.CrLf
    ReadString = FileString & fr.ReadLine

    'Read from the file until EOF or another Date is found.
    Do Until ((fr.Peek < 0) Or (IsDate(fr.ReadLine)))
        ReadString = ReadString & fr.ReadLine
    Loop
End If
fr.Close()
DisplayEntry.Visible = True
DisplayEntry.Text = ReadString
```

2. コードをテストするには、F5 キーを押してアプリケーションをコンパイルし、エントリを送信します。[Get Entries] をクリックし、**ComboBox** からエントリを選択して、[Display] をクリックします。選択したエントリの内容が `DisplayEntry` **TextBox** に表示されます。

ユーザーがエントリを削除または変更できるようにする

`DeleteEntry` ボタンと `EditEntry` ボタンを使用してユーザーがエントリを削除または変更できる機能を追加できますエントリが表示されるまでは、どちらのボタンも無効にします。

次の表に示すコントロールをフォームに追加し、対応する値をプロパティに設定します。

コントロール	プロパティ	値
Button	Name Text Enabled	**DeleteEntry** **Delete Entry** **False**
Button	Name Text Enabled	**EditEntry** **Edit Entry** **False**

Button	Name Text Enabled	**SubmitEdit** **Submit Edit** **False**

エントリを削除または変更できるようにするには

1. `Display` ボタンの **Click** イベントにある `DisplayEntry.Text = ReadString` の後に次のコードを追加します。

   VB

```
DeleteEntry.enabled = True
```

2. `DeleteEntry` ボタンの **Click** イベント ハンドラを作成し、次のコードを追加します。

   VB

```
Dim fr As StreamReader
Dim ReadString As String
Dim WriteString As String
Dim ConfirmDelete As MsgBoxResult
fr = New StreamReader("C:\MyDiary.txt")
ReadString = fr.ReadLine
' Read through the textfile
Do Until (fr.Peek < 0)
    ReadString = ReadString & vbCrLf & fr.ReadLine
Loop
WriteString = Replace(ReadString, DisplayEntry.Text, "")
fr.Close()
' Check to make sure the user wishes to delete the entry
ConfirmDelete = MsgBox("Do you really wish to delete this entry?", _
  MsgBoxStyle.OKCancel)
If ConfirmDelete = MsgBoxResult.OK Then
    File.Delete("C:\MyDiary.txt")
    Dim fw As StreamWriter = File.CreateText("C:\MyDiary.txt")
    fw.WriteLine(WriteString)
    fw.Close()
    ' Reset controls on the form
    DisplayEntry.Text = ""
    PickEntries.Text = ""
    PickEntries.Items.Clear()
    PickEntries.Enabled = False
    DeleteEntry.Enabled = False
End If
```

3. ユーザーが入力内容を表示すると、`EditEntry` ボタンが有効になるようにします。`Display` ボタンの **Click** イベントにある `DisplayEntry.Text = ReadString` の後に次のコードを追加します。

   VB

```
EditEntry.Enabled = True
```

4. `EditEntry` ボタンの **Click** イベント ハンドラを作成し、次のコードを追加します。

   VB

```
Entry.Text = DisplayEntry.Text
SubmitEdit.Enabled = True
```

5. `SubmitEdit` ボタンの **Click** イベント ハンドラを作成し、次のコードを追加します。

VB

```
Dim fr As StreamReader
Dim ReadString As String
Dim WriteString As String
If Entry.Text = "" Then
    MsgBox("Use Delete to Delete an Entry")
    Return
End If
fr = New StreamReader("C:\MyDiary.txt")
ReadString = fr.ReadLine
Do Until (fr.Peek < 0)
    ReadString = ReadString & vbCrLf & fr.ReadLine
Loop
WriteString = Replace(ReadString, DisplayEntry.Text, Entry.Text)
fr.Close()
File.Delete("C:\MyDiary.txt")
Dim fw As StreamWriter = File.CreateText("C:\MyDiary.txt")
fw.WriteLine(WriteString)
fw.Close()
DisplayEntry.Text = Entry.Text
Entry.Text = ""
EditEntry.Enabled = False
SubmitEdit.Enabled = False
```

コードをテストするには、F5 キーを押してアプリケーションをコンパイルします。[Get Entries] をクリックし、エントリを選択して、[Display] をクリックします。`DisplayEntry` **TextBox** にエントリが表示されます。[Edit Entry] をクリックします。`Entry` **TextBox** にエントリが表示されます。`Entry` **TextBox** の入力内容を編集し、[Submit Edit] をクリックします。`MyDiary.txt` ファイルを開いて修正内容を確認します。次に、エントリを選択し、[Delete Entry] をクリックします。確認を求める MessageBox が表示されたら、[OK] をクリックします。アプリケーションを終了し、`MyDiary.txt` を開いて削除を確認します。

参照

関連項目

StreamReader Class

StreamWriter Class

その他の技術情報

Visual Basic 言語のチュートリアル

ウォークスルー：Visual Basic .NET でファイルおよびフォルダを操作する

Cat Francis
Visual Studio Team
Microsoft Corporation

January 2002
日本語版最終更新日 2003 年 5 月 6 日

要約：このウォークスルーでは、Microsoft® Visual Basic® .NET におけるファイル I/O の基礎について説明します。機能を具体的に説明するために、ディレクトリ内のテキスト ファイルを参照する小規模なアプリケーションを作成し、ファイルの属性、前回のアクセス時刻、最初の 80 文字などの情報を表示します。また、ログ ファイルに情報を書き込むこともできます。

目次

はじめに

開発者で・れば、どこかの段階で必ずファイルとディレクトリに対する操作を行わなければなりません。アプリケーションの・ J発過程においては、以下のような基本タスクが考えられます。

- ログ ファイルへの結果の書き込み。ログ ファイルが・るかどうかの確認と (ない場合は作成します)、収集した情報の追加を含みます。
- ファイルの最新アクセス日時やサイズなど、ファイルに関する情報の収集。
- ディレクトリ内のファイル一覧の生成と表示。1 つのディレクトリから別のディレクトリへの移動が必要な場合も・ります。

これらのタスクに携わる開発者は、使用可能なメソッドとクラスについて理解している必要が・ります。このウォークスルー・ナは、Microsoft Visual Basic .NET のファイル I/O 機能を使用して FileExplorer というアプリケーションを作成します。このアプリケーションでは、ディレクトリの参照、ディレクトリ内のファイルに関する情 報の確認と表示、およびログ ファイルへの情報の書き込みを行うことができます。アプリケーションが完成したら、プログラムによるファイルの選択、そのフ ァイルに関する情報の抽出、およびログ ファイルへの情報の書き込みが可能になります。

アプリケーションを作成する

プロジェクトを開始するには、ユーザーによるディレクトリの選択、ディレクトリ内のファイルの選択、およびそのファイル・ ノついて抽出する情報の選択を行うことができるフォームを作成します。

プロジェクトを作成するには

1. [ファイル] メニューの [新規作成] をポイントし、[プロジェクト] をクリックします。

   [新しいプロジェクト] ダイアログ ボックスが表示されます。

2. [プロジェクトの種類] ペインで [Visual Basic プロジェクト] をクリックします。次に、[テンプレート] ペインで [Windows アプリケーション] をクリックします。

3. [名前] ボックスに「FileExplorer」と入力して、プロジェクト名を設定します。

Visual Studio® によってプロジェクトがソリューション エクスプローラに追加され、Windows® フォーム デザイナが開きます。

4. 次の表のコントロールをフォームに追加し、プロパティに対応する値を設定します。

オブジェクト	プロパティ	値
TextBox	Name	txtDirectory
	Text	ディレクトリ
Button	Name	btnSubmit
	Text	送信(&S)
Button	Name	btnExamine
	Text	確認(&E)
ListBox	Name	lstFilePick
CheckBox	Name	chkAttributes
	Text	属性
	Checked	True
CheckBox	Name	chkFileLength
	Text	ファイルの長さ
	Checked	True
CheckBox	Name	chkLastAccess
	Text	前回のアクセス時刻
	Checked	True
CheckBox	Name	chkFirstLine
	Text	1 行目を表示する
	Checked	True
CheckBox	Name	chkSave
	Text	結果を保存する
	Checked	False

現在のディレクトリを表示する

どの開発プロセスにも開始地点が・ります。txtDirectory テキスト ボックスでは、CurDir 関数を使用して現在のパスを示す文字列を返し、表示します。

現在のディレクトリを返すには

1. txtDirectory テキスト ボックスに現在の場所を表示するには、次のコードを `Form1_Load` に追加します。

```
txtDirectory.Text = CurDir()
```

2. プログラムを実行して、CurDir 関数が正しいパスを返すかどうかを確認します。

   txtDirectory テキスト ボックスに現在のディレクトリが表示されます。

ディレクトリを変更する

別のディレクトリに・るファイルを選択する場合は、ChDir 関数を使用してディレクトリを切り替えます。別のディレクトリに変更するには、ユーザーは txtDirectory テキスト ボックスに新しいパスを入力します。

ディレクトリを変更するには

- 次のコードを `btnSubmit_Click` に追加します。

```
Dim NewPath As String
' NewPath には、ユーザーが入力したパスが格納されます。
NewPath = txtDirectory.Text
' 場所を NewPath に変更します。
ChDir(NewPath)
```

有効なパスの入力を確認する

ChDir 関数には Path パラメータが必要です。ドライブを指定することもできますが、指定しなくてもかまいません。パスでドライブが指定されていな い場合、ChDir は現在のドライブを使用します。混乱を回避するために、Try...Catch ステートメントを組み込んで、パスが指定されないことによって発生する例外をキャッチします。

有効なパスを確保するには

1. イベント `btnSubmit_Click` のコード行 `Dim NewPath As String` の後の新しい行に、`Dim ErrorMessage As String` を追加します。

2. コード行 `ChDir(NewPath)` の前に、次のように Try ステートメントの行を追加します。

```
      Try
```

3. コード行 `ChDir(NewPath)` の後に、次のコードを追加します。

```
      ' パスが空白でないかどうかをチェックします。
Catch ex As Exception When NewPath =  ""
          ErrorMessage = "パスを入力してください。"
      ' 無効なパスによって発生したエラーをキャッチします。
      Catch
          ErrorMessage = "有効なパスを入力してください。別の
          ドライブにアクセスする場合、ドライブ名を指定する必要が
          あります。"
      Finally
          ' エラー メッセージがある場合のみ表示します。
          If ErrorMessage <> "" Then
             MsgBox(ErrorMessage)
          End If
       End Try
```

リスト ボックスにディレクトリの内容を表示する

現在のディレクトリの内容を表示するには、指定されたパターンと一致するファイル、ディレクトリ、またはフォルダの名前・ロヲす文字列を返す Dir 関数を使用できます。Dir 関数は最初に一致したアイテムのみを返すので、各アイテムをチェックするには Do Until...Loop を組み込みます。また、[送信] ボタンに次のコードを追加すると、.txt ファイルのみを一覧表示することができます。

> **注 :** このプロセスでは、引数を指定しないで Dir 関数を呼び出した場合、前の一致したパターンがそのまま残されるという事実を利用しています。

ディレクトリの内容を表示するには

1. イベント `btnSubmit_Click` の最初に `Dim ContentItem As String` を挿入します。

2. コード行 `End Try` の後に、次のコードを挿入します。

```
LstFilePick.Items.Clear
' .txt ファイルのみを表示します。
ContentItem = Dir("*.txt")
If ContentItem = "" Then
   ErrorMessage = "テキスト ファイルが見つかりません。"
   MsgBox(ErrorMessage)
End If
' ディレクトリの内容を反復します。
Do Until ContentItem = ""
   ' 一覧に追加します。
   lstFilePick.Items.Add(ContentItem)
   ' 次のテキスト ファイルに移動します。
   ContentItem = Dir()
   Loop
```

収集された情報が lstFilePick リスト ボックスに表示され、特定のファイルを選択して参照できるようになります。

アプリケーションをテストするには、まず .txt ファイルがないディレクトリでアプリケーションを実行して、次に複数の .txt ファイルが・るディレクトリで実行します。.txt ファイルがないディレクトリでアプリケーションを実行した場合、その内容を示すエラー メッセージが表示されます。.txt ファイルが複数・るディレクトリで実行した場合は、txtDirectory テキスト ボックスで指定したディレクトリ内の .txt ファイルがすべて表示されます。

ユーザーによる参照ファイルの選択を可能にする

ユーザーは、lstFilePick リスト ボックスに表示されるディレクトリ内のすべてのファイルの中から特定のファイルを選択し、内容を確認することができます。

- ファイルの選択を確認するには、イベント `btnExamine_Click` の最初に次のコードを追加します。

```
Dim thisFile As Object
thisFile = lstFilePick.SelectedItem
   ' ファイルが選択されているかどうかをチェックします。
   If thisFile Is Nothing Then
       MsgBox("ファイルが選択されていません。")
       Exit Sub
   End If
```

収集する情報をユーザーが決定できるようにする

lstFilePick リスト ボックスにファイルが表示されたら、コードを追加して、収集する情報をユーザーが指定できるようにします。たとえば、ファイ ルの最終アクセス日時のみを参照することができます。また、同時にファイルのサイズを参照することもできます。ユーザーは、チェック ボックス (chkAttributes、chkLastAccess、chkFileLength、chkFirstLine) をオンまたはオフにして、結果をカスタマイズできます。

特定の情報を表示するには

1. イベント `btnExamine_Click` の最初で、以下の変数を宣言します。

```
Dim Attributes As FileAttribute
Dim AttributeResult As String
Dim LastAccess As Date
Dim Length As Long
```

```
Dim FirstLine As String
Dim FinalString As String
Dim ErrorMessage As String
```

GetAttr 関数は、ファイル、ディレクトリ、またはフォルダの属性を示す FileAttribute 列挙のメンバを返します。返される値は、列挙値の合計です。設定されている属性を確認するには、And 演算子を使用して、GetAttr によって返された値と必要なファイル属性の値とのビット比較を実行します。このアプリケーションではテキスト ファイルのみが調べられるので、ディレクトリやシステムなど、GetAttr によって返される可能性が・る一部の値は無視してもかまいません。

2. ユーザーがファイルの属性を確認できるようにするには、イベント `btnExamine_Click` の `End If` の後に次のコードを追加します。

```
' 属性をチェックします。
If chkAttributes.Checked = True Then
    Attributes = GetAttr(thisFile)
    If Attributes = 2 Then
        AttributeResult = "ReadOnly"
    ElseIf Attributes = 32 Then
        AttributeResult = "Archive"
    ElseIf Attributes = 34 Then
        AttributeResult = "ReadOnly and Archive"
    Else
        AttributeResult = "Normal"
    End If
        ' 結果のメッセージ ボックスに追加します。
        stringAttributes = "ファイル属性 : "
        FinalString = stringAttributes + AttributeResult + "." + _
        vbCr
End If
```

3. イベント `btnExamine_Click` に次のコードを追加します。

```
' 前回のアクセス時刻をチェックします。
If chkLastAccess.checked = True Then
     LastAccess = FileDateTime(thisFile)
End If
```

FileDateTime 関数は、ファイルの最終アクセス日時を確認します。返される日付の値は、ファイルの作成日時または最終変更日時を示します。

4. イベント `btnExamine_Click` に次のコードを追加します。

```
' 長さをチェックします。
If chkFileLength.checked = True Then
     Length = FileLen(thisFile)
End If
```

FileLen 関数はファイルの長さを確認し、ファイルの長さ (バイト単位) を示す長整数型 (long) の値を返します。

5. ユーザーがファイルの 1 行目を取得できるようにするには、イベント `btnExamine_Click` に次のコードを追加します。

```
' 1 行目をチェックします。
```

```
        If chkFirstLine.Checked = True Then
             Try
                  FileOpen(1, thisFile, OpenMode.Input)
                  FirstLine = (InputString(1, 80))
             Catch ex As Exception
                  ErrorMessage = "確認しようとしているファイルにエラーが
                  発生しています。テキスト ファイルが空でないことを
                  確認してください。"
                  MsgBox (ErrorMessage)
                  Exit Sub
             End Try
             FileClose(1)
        End If
```

結果を表示する

このアプリケーションの機能を完了させるため、収集した情報をメッセージ ボックスに表示します。

結果を表示するには

1. イベント `btnExamine_Click` の冒頭で、以下の変数を宣言し、開始します。

```
        Dim stringAttributes As String
        stringAttributes = "ファイルの属性 : "
        Dim stringLastAccess As String
        stringLastAccess = "前回のアクセス時刻 : "
        Dim stringLength As String
        stringLength = "ファイルの長さ (バイト単位) : "
        Dim strFirstLine As String
        strFirstLine = "最初の 80 文字 : "
```

2. 次のコードを、`chkAttributes` チェック ボックスがオンになっているかどうかを確認する `If` ステートメントの最後の `End If` の前に追加します。

```
        ' 結果のメッセージ ボックスに追加します。
        FinalString = stringAttributes + AttributeResult + "." + vbCr
```

3. 次のコードを、`chkLastAccess` チェック ボックスがオンになっているかどうかを確認する `If` ステートメントの最後の `End If` の前に追加します。

```
        ' メッセージ ボックスに追加します。
        FinalString = FinalString + stringLastAccess + LastAccess + "." _
        + vbCr
```

4. 次のコードを、`chkFileLength` チェック ボックスがオンになっているかどうかを確認する `If` ステートメントの最後の `End If` の前に追加します。

```
        ' メッセージ ボックスに追加します。
        FinalString = FinalString + stringLength + CStr(Length) + "." _
        + vbCr
```

5. 次のコードを、`chkFirstLine` チェック ボックスがオンになっているかどうかを確認する `If` ステートメントの最後の `End If` の前に追加します。

```
        ' メッセージ ボックスに追加します。
```

```
        FinalString = FinalString + strFirstLine + Firstline + vbCr
```

6. 関数の最後の `End Sub` の前に、次のコードを追加します。

```
        If FinalString = "" Then
             MsgBox("チェック ボックスがオンになっていません。")
        Else
             MsgBox(FinalString)
        End If
```

結果を保存する

ユーザーは、ファイルの確認結果を保存することもできます。そのためには、ログ ファイルが・るかどうかを確認し (必要に応じて作成し)、ログ ファイルに結果を書き込むコードを追加します。

ログ ファイルを作成するには

イベント `btnExamine_Click` の最後に、次のコードを追加します。

```
        ' 結果を保存する必要があるかどうかをチェックします。
        If chkSave.Checked = True Then
             ' ファイルがない場合、FileOpen を使用してファイルを作成します。
             FileOpen(1, "log.txt", OpenMode.Append)
        Writeline (1, finalString)
        FileClose(1)
        End If
```

アプリケーションをテストするには

1. 選択したディレクトリに test.txt という名前のテキスト ファイルを作成し、1 行目に「これは、最初のファイルの 1 行目です。FileExplorer アプリケーションは、テキスト ファイルのみを参照します。」と入力します。同じディレクトリに test2.txt という名前の 2 つ目のテキスト ファイルを作成し、1 行目に「これは、2 つ目のファイルの 1 行目です。FileExplorer アプリケーションは、テキスト ファイルのみを参照します。」と入力します。

2. アプリケーションを起動します。

3. 無効なパスを入力し、[送信] をクリックします。

   "有効なパスを入力してください。別のドライブにアクセスする場合、ドライブを指定する必要が・ります。" というメッセージが表示されます。

4. test.txt が保存されているディレクトリへのパスを入力し、[送信] をクリックします。

   lstFilePick リスト ボックスにテキスト ファイルが表示されます。

5. lstFilePick リスト ボックスで [test.txt] を選択します。チェック ボックスがすべてオンになっていることを確認し、[参照] をクリックします。

   結果フォームに、ファイルの属性、前回のアクセス時刻、長さ、および最初の 80 文字が表示されます。

6. lstFilePick リスト ボックスで [test2.txt] を選択し、チェック ボックスをすべてオフにしてから [参照] をクリックします。

   "チェック ボックスがオンになっていません。" というエラー メッセージが表示されます。

7. [属性] および [結果を保存する] をオンにし、[参照] をクリックします。

   結果フォームに、ファイルの属性のみが表示されます。

8. FileExplorer を終了します。

[結果を保存する] チェック ボックスをオンにしたので、テキスト ファイルと同じディレクトリに log.txt という名前のログ ファイルが生成されています。

ログをチェックするには

- 現在のディレクトリで log.txt を開き、FileExplorer によって正しい情報が記録されていることを確認します。

結論

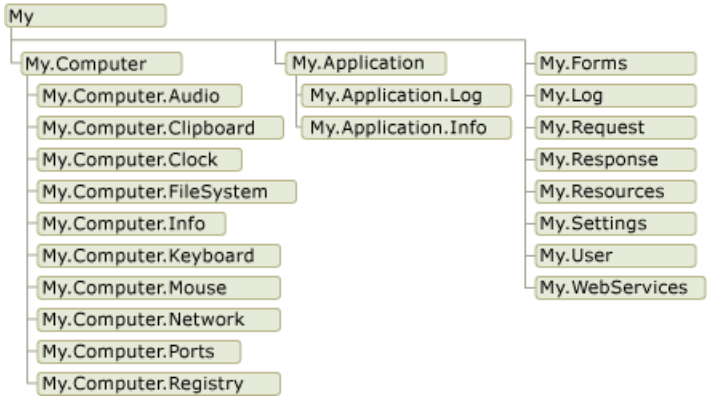
このウォークスルーでは、Visual Basic .NET のファイル I/O の基礎について説明しました。ただし、これは最初のレベルに過ぎません。ファイル、ディレクトリ、およびドライブを操作する コードを記述する場合、Visual Basic 固有のランタイム関数から離れて、Microsoft Visual Studio .NET の・らゆる言語に共通するクラスを提供する System.IO 名前空間を探索することができます。たとえば、ディレクトリやファイルが変更されたときにイベントを発生させる FileSystemWatcher クラスが・ります。詳細については、「入出力操作」を参照してください。

My による開発

Visual Basic には、RAD (Rapid Application Development) のための強力な新機能が用意されており、生産性と使いやすさを向上できます。そうした機能の 1 つが **My** です。アプリケーションおよびその実行時環境に関連する情報や、既定のオブジェクトのインスタンスへのアクセスを実現します。この情報は、IntelliSense を介して検出できる形式で構成されており、用途に応じて論理的に記述されます。

My のトップレベルのメンバはオブジェクトとして公開されています。各オブジェクトは、名前空間や **Shared** メンバを持つクラスに対して同様に動作し、関連するメンバを公開します。

この表は、**My** のトップレベルのオブジェクトと、それぞれの間の関係を示します。



このセクションの内容

My.Application、My.Computer、および My.User でのタスクの実行

My の中心となる、**My.Application**、**My.Computer**、および **My.User** の 3 つのオブジェクトについて説明します。情報および機能へのアクセスを提供するオブジェクトです。

My.Forms と My.WebServices が提供する既定のオブジェクト インスタンス

My.Forms オブジェクトおよび **My.WebServices** オブジェクトについて説明します。アプリケーションで使用するフォーム、データ ソース、および XML Web サービスへのアクセスを提供するオブジェクトです。

My.Resources と My.Settings による Rapid Application Development

My.Resources オブジェクトおよび **My.Settings** オブジェクトについて説明します。アプリケーションのリソースおよび設定へのアクセスを提供するオブジェクトです。

Visual Basic アプリケーション モデルの概要

Visual Basic アプリケーションの起動/終了モデルについて説明します。

プロジェクトの種類に応じた My の機能

プロジェクトの各種類でどの **My** 機能を利用できるかについて詳細を示します。

参照

関連項目

My.Application オブジェクト
My.Computer オブジェクト
My.User オブジェクト
My.Forms オブジェクト
My.WebServices オブジェクト

概念

プロジェクトの種類に応じた My の機能

My.Application、My.Computer、および My.User でのタスクの実行

情報や一般的に使用される機能へのアクセスを提供する 3 つの中心的な **My** オブジェクトとして、My.Application オブジェクト、My.Computer オブジェクト、および My.User オブジェクト があります。これらのオブジェクトを使用すると、現在のアプリケーションに関する情報、アプリケーションがインストールされているコンピュータに関する情報、またはアプリケーションの現在のユーザーに関する情報にそれぞれアクセスできます。

My.Application、My.Computer、My.User

次の例は、**My** を使用して情報を取得する方法を具体的に示しています。

VB

```
' Displays a message box that shows the full command line for the
' application.
Dim args As String = ""
For Each arg As String In My.Application.CommandLineArgs
    args &= arg & " "
Next
MsgBox(args)
```

VB

```
' Gets a list of subfolders in a folder
My.Computer.FileSystem.GetDirectories _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments, True, "*Logs*")
```

情報を取得できるだけでなく、これらの 3 つのオブジェクトを介して公開されるメンバは、そのオブジェクトに関連するメソッドを実行することもできます。たとえば、**My.Computer** を利用すると、ファイルの操作やレジストリの更新を行うさまざまなメソッドにアクセスできます。

My には、ファイル、ディレクトリ、およびドライブを操作するさまざまなメソッドとプロパティが含まれており、これを使用するとファイル I/O が大幅に簡単かつ高速になります。TextFieldParser オブジェクト を使用すると、区切り記号で区切られたフィールドや固定幅のフィールドを持つ大きな構造化ファイルからデータを読み取ることができます。この例では、`reader` という **TextFieldParser** を開き、それを使用して `C:\TestFolder1\test1.txt` から読み込みを行います。

VB

```
Dim reader As Microsoft.VisualBasic.FileIO.TextFieldParser
reader = My.Computer.FileSystem.OpenTextFieldParser _
("C:\TestFolder1\test1.txt")
reader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
reader.Delimiters = New String() {","}
Dim currentRow As String()
While Not reader.EndOfData
  Try
      currentRow = reader.ReadFields()
      Dim currentField As String
        For Each currentField In currentRow
            MsgBox(currentField)
        Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
          MsgBox("Line " & ex.Message & _
          "is not valid and will be skipped.")
    End Try
End While
```

My.Application を使用すると、アプリケーションのカルチャを変更できます。次の例は、このメソッドを呼び出す方法を具体的に示しています。

VB

```
' Changes the current culture for the application to Jamaican English.
My.Application.ChangeCulture("en-JM")
```

参照

関連項目

My.Application オブジェクト

My.Computer オブジェクト

My.User オブジェクト

概念

プロジェクトの種類に応じた My の機能

My.Forms と My.WebServices が提供する既定のオブジェクト インスタンス

My.Forms および My.WebServices オブジェクトは、アプリケーションで使用されるフォーム、データ ソース、および XML Web サービスへのアクセスを提供します。このアクセスは、これらの各オブジェクトがそれぞれの既定インスタンスのコレクションを提供することで実現します。

既定インスタンス

既定インスタンスとは、ランタイムが提供するクラスのインスタンスで、**Dim** ステートメントおよび **New** ステートメントを使用して宣言およびインスタンス化する必要はありません。次の例は、`Form1` という名前の Form クラスのインスタンスを宣言およびインスタンス化した方法と、**My.Forms** を通じてこの **Form** クラスの既定インスタンスを取得できるようにする方法を具体的に示しています。

VB

```
' The old method of declaration and instantiation
Dim myForm As New Form1
myForm.show()
```

VB

```
' With My.Forms, you can directly call methods on the default
' instance()
My.Forms.Form1.Show()
```

My.Forms オブジェクトは、プロジェクト内に存在するすべての **Form** クラスの既定インスタンスのコレクションを返します。同様に、**My.WebServices** は、アプリケーション内で参照を作成したすべての Web サービスのプロキシ クラスの既定インスタンスを提供します。

参照

関連項目

My.Forms オブジェクト

My.WebServices オブジェクト

概念

プロジェクトの種類に応じた My の機能

My.Resources と My.Settings による Rapid Application Development

My.Resources オブジェクトを使用すると、アプリケーションのリソースへのアクセスが可能になり、アプリケーションに必要なリソースを動的に取得できるようになります。

リソースの取得

オーディオ ファイル、アイコン、イメージ、文字列などの多数のリソースを **My.Resources** オブジェクトを介して取得できます。たとえば、アプリケーションのカルチャ固有のリソース ファイルにアクセスできます。次の例は、アプリケーションのリソース ファイルに格納された `Form1Icon` という名前のアイコンを、フォームのアイコンに設定します。

VB

```
Sub SetFormIcon()
    Me.Icon = My.Resources.Form1Icon
End Sub
```

My.Resources オブジェクトはグローバルなリソースだけを公開します。フォームに関連付けられたリソース ファイルへのアクセスはできません。フォームのリソースには、フォームからアクセスする必要があります。詳細については、「チュートリアル : Windows フォームのローカリゼーション」を参照してください。

同様に、**My.Settings** オブジェクトはアプリケーションの設定値へのアクセスを提供し、これによって、プロパティの設定値およびアプリケーションに関するその他の情報を動的に格納および取得できます。詳細については、「My.Resources オブジェクト」および「My.Settings オブジェクト」を参照してください。

参照

関連項目

My.Resources オブジェクト
My.Settings オブジェクト

概念

アプリケーション リソースへのアクセス
アプリケーション設定へのアクセス

Visual Basic アプリケーション モデルの概要

Visual Basic には、Windows フォーム アプリケーションの動作を制御するための確たるモデルが用意されています。それが Visual Basic アプリケーション モデルです。このモデルには、アプリケーションの起動とシャットダウンを処理するためのイベントや、未処理の例外をキャッチするためのイベントがあります。また、単一インスタンス アプリケーションの開発もサポートされています。このアプリケーション モデルには拡張性があるため、開発者は、きめ細かな制御が必要な場合には、オーバーライド可能なメソッドをカスタマイズできます。

アプリケーション モデルの用途

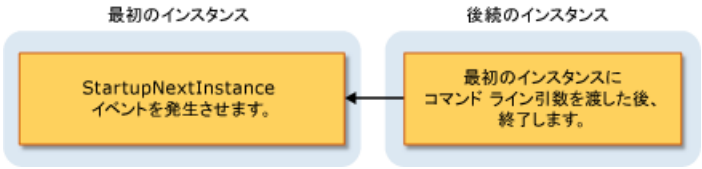
一般に、アプリケーションでは、起動時と終了時にタスクを実行する必要があります。たとえば、アプリケーションの起動時には、スプラッシュ スクリーンの表示、データベース接続の確立、保存しておいた状態の読み込みなどを実行できます。アプリケーションの終了時には、データベース接続のクローズや現在の状態の保存などを実行できます。また、アプリケーションは、未処理の例外の発生時など、予測できない形で終了するときに、独自のコードを実行できます。

Visual Basic アプリケーション モデルでは、単一インスタンス アプリケーションを簡単に作成できます。単一インスタンス アプリケーションが通常のアプリケーションと違うのは、当該アプリケーションのインスタンスは一度に 1 つのみ実行できるということです。単一インスタンス アプリケーションの別のインスタンスを二重起動しようとすると、元のインスタンスに対して、その旨が **StartupNextInstance** イベントにより通知されます。この通知には、二重起動されたインスタンスのコマンド ライン引数が含まれています。そして、二重起動されたインスタンスは、初期化が実行される前の段階で閉じられます。

単一インスタンス アプリケーションの起動時には、それが当該アプリケーションの最初のインスタンスなのか、それとも二重起動されたインスタンスなのかがチェックされます。

- 最初のインスタンスの場合には、通常どおり起動されます。
- 最初のインスタンスの実行中に、同じアプリケーションを二重起動しようとした場合には、実行される動作は異なります。二重起動の場合は、最初のインスタンスに対してコマンド ライン引数が通知され、すぐに終了されます。最初のインスタンスは、**StartupNextInstance** イベントを処理し、二重起動されたインスタンスのコマンド ライン引数を判断したうえで、実行を継続します。

  この図は、二重起動されたインスタンスから最初のインスタンスに対する通知方法を示しています。



StartupNextInstance イベントを処理すると、単一インスタンス アプリケーションの動作を制御できます。たとえば、Microsoft Outlook は、通常は単一インスタンス アプリケーションとして動作します。Outlook が実行中のときに、Outlook を再度起動しようとすると、元のインスタンスにフォーカスが移り、新しいインスタンスは開かれません。

アプリケーション モデルのイベント

以下に示すのは、アプリケーション モデルに含まれているイベントです。

- **アプリケーションの起動**。アプリケーションの起動時には Startup イベントが発生します。このイベントを処理すると、メイン フォームの読み込み前にアプリケーションを初期化するコードを追加できます。**Startup** イベントには、起動処理のその段階でアプリケーションの実行を必要に応じてキャンセルするための方法も用意されています。

  アプリケーションの構成により、スタートアップ コードの実行時にスプラッシュ スクリーンを表示させることができます。アプリケーション モデルの既定では、`/nosplash` または `-nosplash` のいずれかのコマンド ライン引数が指定されている場合、スプラッシュ スクリーンは表示されません。

- **単一インスタンス アプリケーション**。単一インスタンス アプリケーションが二重起動されたときには、StartupNextInstance イベントが発生します。このイベントは、二重起動されたインスタンスのコマンド ライン引数を渡します。

- **未処理の例外**。アプリケーションで未処理の例外が発生した場合、UnhandledException イベントが発生します。このイベントのハンドラでは、例外を調べ、実行を続けるかどうかを決定できます。

  UnhandledException イベントは、状況によっては発生しない場合もあります。詳細については、「My.Application.UnhandledException イベント」を参照してください。

- **ネットワーク接続の変更**。コンピュータのネットワークの可用性が変更された場合、アプリケーションでは NetworkAvailabilityChanged イベントが発生します。

NetworkAvailabilityChanged イベントは、状況によっては発生しない場合もあります。詳細については、「My.Application.NetworkAvailabilityChanged イベント」を参照してください。

- **アプリケーションの終了**。アプリケーションには、終了しようとしている旨を通知するための Shutdown イベントが用意されています。このイベント ハンドラでは、アプリケーションで実行する必要のある操作 (たとえばクローズや保存など) が完了したかどうかを確認できます。アプリケーションの構成により、メイン フォームが閉じたときに終了するか、それとも、すべてのフォームが閉じたときにのみ終了するかを設定できます。

可用性

既定では、Visual Basic アプリケーション モデルは Windows フォーム プロジェクトで利用可能です。異なるスタートアップ オブジェクトを利用するようにアプリケーションを構成した場合や、カスタムの `Sub Main` でアプリケーション コードを開始する場合は、そのオブジェクトまたはクラスで WindowsFormsApplicationBase クラスの実装を用意しないと、アプリケーション モデルを利用できない場合があります。スタートアップ オブジェクトの変更については、「方法 : アプリケーションのスタートアップ オブジェクトを変更する」を参照してください。

参照

関連項目

My.Application オブジェクト
My.Application.Startup イベント
My.Application.StartupNextInstance イベント
My.Application.UnhandledException イベント
My.Application.Shutdown イベント
My.Application.NetworkAvailabilityChanged イベント
WindowsFormsApplicationBase

概念

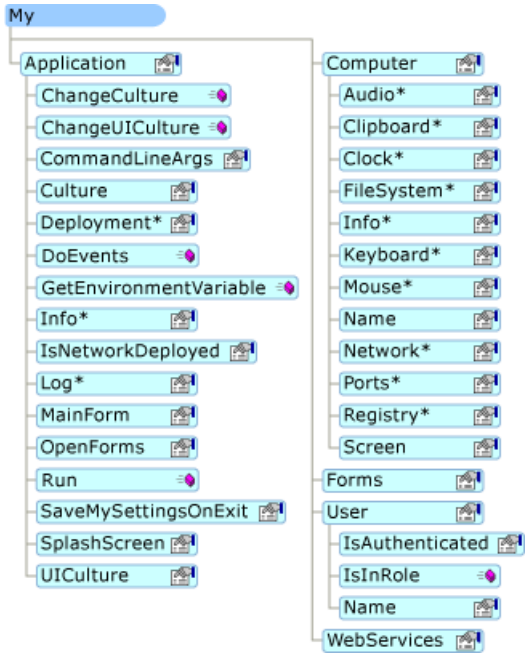
Visual Basic アプリケーション モデルの拡張

プロジェクトの種類に応じた My の機能

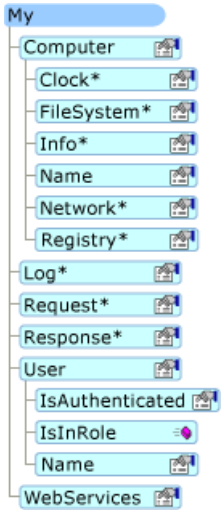
My では、対象のプロジェクトの種類で必要なオブジェクトのみが公開されます。たとえば、**My.Forms** オブジェクトは、Windows フォーム アプリケーションでは利用できますが、コンソール アプリケーションでは利用できません。このトピックでは、プロジェクトの各種類でどの **My** オブジェクトを利用できるかについて説明します。

Windows アプリケーションおよび Web サイトでの My

My では、現在のプロジェクトの種類で利用できるオブジェクトのみが公開され、対象外のオブジェクトは無効になります。たとえば、次のイメージは、Windows フォーム プロジェクトでの **My** のオブジェクト モデルを示します。



Web サイト プロジェクトでは、**My** は、Web 開発者に関係するオブジェクト (たとえば **My.Request** オブジェクトや **My.Response** オブジェクト) は公開する一方、関係しないオブジェクト (たとえば **My.Forms** オブジェクト) は無効にします。次のイメージは、Web サイト プロジェクトでの **My** のオブジェクト モデルを示します。



プロジェクトの詳細

次の表は、8 つのプロジェクトの種類 (Windows アプリケーション、クラス ライブラリ、コンソール アプリケーション、Windows コントロール ライブラリ、Web コントロール ライブラリ、Windows サービス、空のプロジェクト、および Web サイト) において、どの **My** オブジェクトが既定で有効になっているかを示します。

My.Application オブジェクトには 3 つのバージョンがあり、**My.Computer** オブジェクトと **My.User** オブジェクトにはそれぞれ 2 つのバージョンがあります。各バージョンの詳細については、表の後の脚注で説明します。

My オブジェクト	Windows アプリケーション	クラス ライブラリ	コンソール アプリケーション	Windows コントロール ライブラリ	Web コントロール ライブラリ	Windows サービス	空	Web サイト

My.Application	○ [1]	○ [2]	○ [3]	○ [2]	×	○ [3]	×	×
My.Computer	○ [4]	○ [4]	○ [4]	○ [4]	○ [5]	○ [4]	×	○ [5]
My.Forms	○	×	×	○	×	×	×	×
My.Log	×	×	×	×	×	×	×	○
My.Request	×	×	×	×	×	×	×	○
My.Resources	○	○	○	○	○	○	×	×
My.Response	×	×	×	×	×	×	×	○
My.Settings	○	○	○	○	○	○	×	×
My.User	○ [6]	○ [6]	○ [6]	○ [6]	○ [7]	○ [6]	×	○ [7]
My.WebServices	○	○	○	○	○	○	×	×

[1] **My.Application** の Windows フォーム バージョンです。コンソール バージョン (注 3 を参照) から派生されています。アプリケーションのウィンドウとのやり取りのサポートが追加され、また Visual Basic アプリケーション モデルが備わっています。

[2] **My.Application** のライブラリ バージョンです。アプリケーションが必要とする基本機能が備わっており、アプリケーション ログへの書き込みや、アプリケーション情報へのアクセスのためのメンバが用意されています。

[3] **My.Application** のコンソール バージョンです。ライブラリ バージョン (注 2 を参照) から派生され、アプリケーションのコマンド ライン引数や ClickOnce 配置情報にアクセスするためのメンバが加わっています。

[4] **My.Computer** の Windows バージョンです。サーバー バージョン (注 5 を参照) から派生され、クライアント マシンで有用なオブジェクト (キーボード、画面、マウスなど) にアクセスできます。

[5] **My.Computer** のサーバー バージョンです。名前や時計へのアクセスなど、コンピュータについての基本情報が備わっています。

[6] **My.User** の Windows バージョンです。このオブジェクトは、スレッドの現在の ID と関連付けられています。

[7] **My.User** の Web バージョンです。このオブジェクトは、アプリケーションの現在の HTTP 要求のユーザー ID と関連付けられています。

参照

関連項目

/define (Visual Basic)
My.Application オブジェクト
My.Computer オブジェクト
My.Forms オブジェクト
My.Log オブジェクト
My.Request オブジェクト
My.Response オブジェクト
My.User オブジェクト
My.WebServices オブジェクト

概念

My で利用可能なオブジェクトのカスタマイズ
条件付きコンパイルの概要

Visual Basic アプリケーションにおけるデータ アクセス

Visual Basic には、データにアクセスするアプリケーションを開発する際に役立ついくつかの新機能が用意されています。Windows アプリケーションのデータ バインド フォームは、[データ ソース] ウィンドウからフォームに項目をドラッグすることにより作成できます。データをコントロールにバインドするには、[データ ソース] ウィンドウから既存のコントロールに項目をドラッグします。

関連するセクション

クライアント データ アプリケーションの作成

アプリケーションにデータ アクセス機能を取り込む方法について説明したページへのリンクを示します。

データに関するチュートリアル

データ アクセス関連の特定のシナリオに関するページへのリンクを示します。

データ アクセスを使用した作業の開始

Visual Studio を使用して、データを操作するアプリケーションを作成する方法に関するページへのリンクを示します。

Visual Studio でのデータ アプリケーションの作成

理解する必要がある Visual Studio でのデータ操作の基本概念を説明します。

Visual Studio でのデータへの接続

Visual Studio で、デザイン時ツールを使用してアプリケーションをデータに接続する方法と、ADO.NET 接続オブジェクトを使用してアプリケーションをデータに接続する方法に関するページへのリンクを示します。

アプリケーションでデータを受け取る準備

データセットの概要、新しいデータセットの作成方法、およびデータセットを構成する個々のオブジェクトの作成と編集の方法について説明しているページへのリンクを示します。

アプリケーションへのデータのフェッチ

データセットにデータを読み込む方法と、SQL ステートメントおよびストアド プロシージャを実行する方法を説明しているページへのリンクを示します。

Windows アプリケーションのフォームでのデータの表示

データ バインド コントロールを使用して Windows フォームにデータを表示する方法について説明しているページへのリンクを示します。

アプリケーションでのデータ編集

データセットのデータ テーブルにあるデータを操作する方法について説明しているページへのリンクを示します。

データの検証

列と行の変更時にデータセットに対する検証を追加する方法について説明しているページへのリンクを示します。

データの保存

更新済みのデータをアプリケーションからデータベースに送信する方法について説明しているページへのリンクを示します。

データの新機能

クライアント アプリケーションとデータ層アプリケーションの新しいデータ機能に関する情報とリンクを示します。

Visual Basic の新機能

Visual Basic の新機能を説明します。

ADO.NET の新機能

ADO.NET の新機能を紹介します。

Microsoft Visual Database Tools の新機能

Visual Database Tools の新機能と改善された機能について説明します。

ASP.NET データ アクセスの新機能

ASP.NET アプリケーションのデータにアクセスするための新機能について説明します。

ADO.NET

.NET Framework プログラマにデータ アクセス サービスを公開する ADO.NET クラスについて説明します。

ASP.NET データ アクセス (Visual Studio)

ASP.NET Web ページでのデータ操作方法に関する情報へのリンクを示します。

マネージ デバイス プロジェクトのデータ

Visual Studio でデバイス用のデータを管理する方法について説明します。

Office ソリューションにおけるデータ

Office ソリューションでデータを操作する方法を説明するページへのリンクを示します。スキーマ指向プログラミング、データ キャッシュ、およびサーバー側データ アクセスに関する説明が含まれます。

Visual Basic アプリケーションのデバッグ

このページでは、Visual Studio に内蔵されているデバッグ機能を説明するドキュメントへのリンクを示します。

これらの中でも特に重要なのはデバッガです。デバッガでは、プログラムの実行時の動作を検証して、セマンティック エラーの位置を確認できます。

デバッガを使用すると、変数値を出力するための追加の呼び出しを挿入せずに、プログラムの変数値をチェックできます。同様に、目的の位置で実行を中断するためのブレークポイントをコードに挿入することもできます。

実行の制御

次の表は、実行制御に関連するデバッグ タスクと、対応するヘルプ ページへのリンクの一覧です。

目的	参照項目
実行を中断します。	方法 : 実行を中断する
実行を開始します。	方法 : 実行を開始する
デザイン時にデバッグします。	チュートリアル : デザイン時のデバッグ
デバッガを起動します。	方法 : デバッガを自動的に起動する
自分で作成したコードにのみステップ インし、他のコード (システム コールなど) は無視します。	方法 : マイ コードのみにステップ インする
コードにステップ インします。	方法 : コードにステップ インする
デバッグを停止します。	方法 : デバッグの停止と実行の停止
Just-In-Time デバッグを有効にします。Visual Studio の外部で実行中のプログラムで致命的なエラーが発生したときに、Visual Studio デバッガを起動する機能です。	Just-In-Time デバッグ

例外処理

次の表は、例外処理に関連するデバッグ タスクと、対応するヘルプ ページへのリンクの一覧です。

目的	参照項目
未処理の例外の発生時に中断します。	方法 : ユーザーに処理されない例外で中断する
例外がスローされたときに中断します。	方法 : 例外がスローされたときに中断する
初回例外で中断します。	方法 : 例外がスローされたときに中断する
例外処理アシスタントを使用します。	方法 : 例外処理アシスタント を使用してランタイム エラーを修正する
新しい例外を追加します。	方法 : 新しい例外を追加する
例外がスローされた後で実行を継続します。	例外後の実行の継続

エディット コンティニュ

次の表は、エディット コンティニュに関連するデバッグ タスクと、対応するヘルプ ページへのリンクの一覧です。

目的	参照項目
エディット コンティニュのオフとオンを切り替えます。	方法 : エディット コンティニュを有効および無効にする

エディット コンティニュによるコード変更の適用を停止します。	方法 : コード変更を中断する
中断モードで編集を適用します。	方法 : エディット コンティニュの中断モード時に編集を適用する

デバッグ データの調査

次の表は、デバッグ データの表示に関連するデバッグ タスクと、対応するヘルプ ページへのリンクの一覧です。

目的	参照項目
[レジスタ] ウィンドウを使用して、レジスタの内容を表示します。	方法 : [レジスタ] ウィンドウを使用する
[呼び出し履歴] ウィンドウを使用して、現在呼び出し履歴にある関数呼び出しやプロシージャ呼び出しを表示します。	方法 : [呼び出し履歴] ウィンドウを使用する
[逆アセンブリ] ウィンドウを使用して、コンパイラによって作成された命令に対応するアセンブリ コードを表示します。	方法 : [逆アセンブル] ウィンドウを使用する
[モジュール] ウィンドウを使用して、プログラムが使用しているモジュールの一覧とその説明を表示します。	方法 : [モジュール] ウィンドウを使用する
[スクリプト エクスプローラ] ウィンドウを使用して、現在プログラムに読み込まれているスクリプト ファイルの一覧を示します。	方法 : [スクリプト] エクスプローラ ウィンドウを使用する
[プロセス] ウィンドウを使用して、Visual Studio にアタッチしている、またはそこから起動したすべてのプロセスを表示します。	方法 : [プロセス] ウィンドウを使用する
[スレッド] ウィンドウを使用して、プログラムのスレッドをチェックおよび制御します。	方法 : [スレッド] ウィンドウを使用する

ブレークポイントの設定

次の表は、ブレークポイントに関連するデバッグ タスクと、対応するヘルプ ページへのリンクの一覧です。

目的	参照項目
ブレークポイントを設定します。	方法 : 単純なブレークポイントを設定する
ブレークポイントを削除します。	方法 : ブレークポイントを削除する
ブレークポイントを有効または無効にします。	方法 : ブレークポイントを有効または無効にする
ブレークポイントの位置を変更します。	方法 : ブレークポイントの位置を編集する
フィルタを使用して、特定のコンピュータ、プロセス、またはスレッドに対するブレークポイントの動作を制限します。	方法 : ブレークポイント フィルタを指定する
ブレークポイントを実行するかどうかを、条件に応じて制御します。	方法 : ブレークポイント条件を指定する
[ブレークポイント] ウィンドウを使用して、ブレークポイント関連のタスクを実行します。	方法 : [ブレークポイント] ウィンドウを使用する

参照

処理手順

チュートリアル : Windows フォームのデバッグ

方法 : エディット コンティニュの中断モード時に編集を適用する

概念

SQL のデバッグ

実行制御

その他の技術情報

マネージ コードのデバッグ
ネイティブ コードのデバッグ
Web アプリケーションのデバッグ
デバッグ用ユーザー インターフェイス リファレンス
デバッグの設定と準備
デバッガのロードマップ
デバッグの準備 : C#、J#、および Visual Basic のプロジェクト

Visual Basic での例外およびエラー処理

Visual Basic は構造化例外 (エラー) 処理をサポートしています。これを使用すると、実行中のエラーをプログラムで検出でき、場合によってはその回復も可能です。Visual Basic では、他の言語 (C++ など) で既にサポートされている **Try...Catch...Finally** 構文の拡張バージョンを使用します。構造化例外処理では、最新の制御構造 (**Select Case** や **While** に似た制御構造) と、例外、コードの保護ブロック、およびフィルタが組み合わされています。

構造化例外処理を使うと、信頼性が高く包括的なエラー ハンドラを備えたプログラムの作成および管理を簡単に行うことができます。Visual Basic のエラー処理には、構造化例外処理を使用することをお勧めします。**On Error** を使った非構造化例外処理では、アプリケーションのパフォーマンスの低下を招いたり、コードのデバッグや管理が難しくなったりする可能性があります。

このセクションの内容

例外処理の概要

プログラムで例外を処理する方法についてまとめています。

構造化例外処理と非構造化例外処理に適した状況

両方の種類の例外処理を説明し、それぞれの使用が適切な状況について示します。

エラーの種類

構文エラー、ランタイム エラー、および論理エラーの概要を説明します。

スマート コンパイル自動修正

スマート コンパイル自動修正機能とその使用方法について説明します。

方法 : 自動修正でコンパイラ エラーを修正する

スマート コンパイル自動修正機能を使用して、コード エディタでコンパイラ エラーを修正する方法を示します。

Visual Basic での警告の構成

Visual Basic でコンパイラの警告をオンおよびオフにする方法の詳細を説明します。

Visual Basic の構造化例外処理の概要

Visual Basic の構造化例外処理の説明と例を示します。

非構造化例外処理の概要

Visual Basic の非構造化例外処理の説明と例を示します。

関連するセクション

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic の言語要素の変更点の概要を示します。

デバッガのロードマップ

Visual Studio デバッガの使用方法の基本について説明します。デバッグの基本知識、実行の制御、実行中のプログラムへのアタッチ、Just-In-Time デバッグ、デバッガの自動起動、ダンプ、ブレークポイント、プログラムの検査、例外処理、エディット コンティニュ、およびデバッガにおける式の使用についての説明が含まれます。

Just-In-Time デバッグ

Just-In-Time デバッグについて説明します。Visual Studio の外部で実行中のプログラムで致命的なエラーが発生したときに、Visual Studio デバッガを自動的に起動する機能です。

マネージ コードのデバッグ

マネージ アプリケーションにおけるデバッグの一般的な問題と手法について説明します。

例外処理アシスタント

ランタイム エラーのトラブルシューティングを手助けする [例外処理アシスタント] 機能について説明します。

エディット コンティニュ

[エディット コンティニュ] について説明します。プログラムが中断モードのときにソース コードに変更を加えることができ、時間を節約できる機能

です。

COM および ActiveX のデバッグ

COM アプリケーションと ActiveX コントロールのデバッグに関するヒントを紹介します。

例外処理の概要

Visual Basic では、構造化および非構造化の両方の例外 (エラー) 処理がサポートされています。アプリケーション内に例外処理のコードを配置することによって、アプリケーションの実行時に発生するほとんどのエラーを処理し、アプリケーションをそのまま実行させることができます。構造化および非構造化のエラー処理を使用して、アプリケーションに影響を与えないように、起こりうるエラーに対する計画を立てることができます。

例外を生成する可能性がある演算子を使用するメソッドや、例外を生成する可能性があるプロシージャを呼び出したりそれにアクセスしたりするメソッドでは、例外処理を使用することをお勧めします。

例外処理が組み込まれていないメソッドで例外が発生した場合、その例外は呼び出し元のメソッド (前のメソッド) に戻されます。前のメソッドにも例外ハンドラがない場合は、さらにそのメソッドの呼び出し元に戻されます。ハンドラを探して、"呼び出し履歴" (アプリケーション内で呼び出された一連のプロシージャを順次さかのぼっていくこと) を続けます。最終的に例外ハンドラが見つからなかった場合は、エラー メッセージが表示され、アプリケーションが終了します。

メモ :
メソッドには、構造化例外処理と非構造化例外処理のどちらを組み込むこともできますが、両方を 1 つのメソッドに組み込むことはできません。

構造化例外処理

構造化例外処理では、コードのブロックがカプセル化され、各ブロックに 1 つ以上のハンドラが関連付けられます。各ハンドラは、それぞれが処理する例外の種類に対してなんらかのフィルタ条件を指定します。保護ブロックのコードによって例外が発生すると、対応する一連のハンドラが順番に検索されて、一致するフィルタ条件を持つ最初のハンドラが実行されます。1 つのメソッドに複数の構造化例外処理ブロックを使用でき、ブロックが互いに入れ子になっていてもかまいません。

構造化例外処理のためには、**Try...Catch...Finally** ステートメントが使用されます。詳細については、「Visual Basic の構造化例外処理の概要」を参照してください。

非構造化例外処理

非構造化例外処理のためには、**On Error** ステートメントが使用されます。非構造化例外処理の **On Error** ステートメントは、コード ブロックの先頭に配置します。このステートメントは、そのブロック全体にわたるスコープを持ち、そのブロック内で発生したすべてのエラーを処理します。プログラムが他の **On Error** ステートメントに行き当たると、そのステートメントが有効になり、最初のステートメントは無効になります。詳細については、「非構造化例外処理の概要」を参照してください。

参照

処理手順

例外処理のトラブルシューティング

チュートリアル : 構造化例外処理

関連項目

On Error ステートメント (Visual Basic)

概念

エラーの種類

Visual Basic の構造化例外処理の概要

非構造化例外処理の概要

その他の技術情報

例外処理のタスク

構造化例外処理と非構造化例外処理に適した状況

構造化例外処理は、例外、分離されたコード ブロック、およびフィルタを含む制御構造を使用して、例外処理機構を作成します。構造化例外処理を使用すると、エラーの種類を区別し、状況に応じた処理を行うことができます。非構造化例外処理では、コードの先頭に配置された **On Error** ステートメントがすべての例外を処理します。

解説

構造化例外処理は、非構造化例外処理と比べて、汎用性、信頼性、柔軟性の面でかなり優れています。できれば、構造化例外処理を使用してください。ただし、次の状況では、非構造化例外処理を使用することもできます。

- 以前のバージョンの Visual Basic で作成したアプリケーションをアップグレードする場合
- アプリケーションの暫定版を開発しており、プログラムが正常に終了しなくてもよい場合
- 例外のはっきりした原因が前もってわかっている場合
- 期限が迫っており、すばやく処理する必要があり、スピードの柔軟性を犠牲にする場合
- コードが短く、例外を生成するコード部分だけをテストすればよい場合
- 構造化例外処理でサポートされていない **Resume Next** ステートメントを使用する場合

同じ関数の中で構造化例外処理と非構造化例外処理を組み合わせることはできません。**On Error** ステートメントを使用する場合、同じ関数内で **Try...Catch** ステートメントは使用できません。

コードでどちらの例外処理を使用するかに関係なく、コードの作成に着手する前に、コードの前提条件を検討する必要があります。たとえば、ユーザーに電話番号の入力を求めるアプリケーションの場合、次の前提条件が考えられます。

- ユーザーは文字ではなく数字を入力する。
- 番号は一定の形式になっている。
- ユーザーは null 文字列を入力しない。
- ユーザーの電話番号は 1 つである。

ユーザーの入力がこれらの前提条件のいずれかまたはすべてに違反する可能性があります。信頼性の高いコードにするには、アプリケーションがこうした違反から正常に回復できるための十分な例外処理が必要です。

メソッドがどのような状況でも決して例外をスローしないと保証できない限り、ユーザーに通知を行う例外処理を検討してください。例外処理では、十分な情報を出力するようにしてください。例外処理のメッセージでは、エラーの発生を示すだけでなく、エラーの原因と場所も示す必要があります。"エラーが発生しました" だけしか情報がないメッセージでは、ユーザーをいらいらさせるだけです。

参照

処理手順

例外処理のトラブルシューティング

概念

エラーの種類

Visual Basic の構造化例外処理の概要

非構造化例外処理の概要

エラーの種類

Visual Basic のエラー (例外とも呼ばれます) は、構文エラー、ランタイム エラー、および論理エラーの 3 つに分類できます。

構文エラー

構文エラーは、コードを記述している間に検出されます。Visual Basic では、コード エディタ ウィンドウでコードを入力しているときにチェックされ、単語のスペルミスした場合や、言語の要素を不正に使用した場合など間違いがあったときに警告が出ます。構文エラーは最も一般的なエラーです。エラーが発生したときに、その場で簡単に修正できます。

メモ :
Option Explicit ステートメントは、構文エラーを防ぐ方法の 1 つです。このステートメントを指定した場合、事前に宣言してからでないとアプリケーション内で変数を使用できません。このため、宣言した変数をコード内で使用するときに入力ミスなどのエラーがあるとすぐに検出され、エラーを修正できます。

ランタイム エラー

ランタイム エラーは、コードをコンパイルして実行してからでないと発生しないエラーです。構文エラーがなく、正しいように見えたのに実行できないコードなどがこれに該当します。たとえば、ファイルを開くコード行を正しく記述したとします。このとき、そのファイルが破損していると、アプリケーションは **Open** 関数を実行できないため停止します。ほとんどのランタイム エラーは、問題のあるコードを書き直して再コンパイルおよび再実行することによって修正できます。

論理エラー

論理エラーは、アプリケーションが実際に使用されるようになってから発生するエラーです。これは、通常、ユーザーによる操作の結果として発生する、好ましくないまたは予期しない結果です。たとえば、キーが間違って入力されるなどの外的要因によって、予測されているパラメータの範囲内でアプリケーションが機能しなくなったり、アプリケーション全体が停止したりする場合があります。論理エラーは、原因が必ずしも明らかではないため、一般に最も修正が難しいエラーです。

参照

概念

例外処理の概要
Visual Basic の構造化例外処理の概要
非構造化例外処理の概要

その他の技術情報

例外処理のタスク
デバッガのロードマップ

スマート コンパイル自動修正

スマート コンパイル自動修正機能では、エラーが発生したときに修正方法の候補が表示され、コードに適用する解決策を選択できます。

エラーが発生したときに、波線の右下にシンボルが表示されている場合には、その波線の上にマウス ポインタを置くか、またはタスク一覧でエラー メッセージをダブルクリックすると、シンボルがスマート タグ パネルに変わります。スマート タグ パネルをクリックするか、またはその上にマウス ポインタを置くと、[エラー修正のオプション] ヘルパー ウィンドウが開き、その中には、エラーの説明や、エラーの修復方法の候補が表示されています。ヘルパー ウィンドウでは、必要に応じて、修復のプレビューを別個のウィンドウに表示できます。

適用した変更は、元に戻す操作 (Ctrl + Z) を行うと、元に戻すことができます。

[スマート コンパイル自動修正] ヘルパー ウィンドウを消すには、エディタの他の部分をクリックします。

次の表は、[スマート コンパイル自動修正] ウィンドウに表示されている要素の一覧とその説明です。

要素	説明
[エラー メッセージ]	エラーに関連するメッセージを表示します。
[推奨される修正]	エラーの修復方法の候補を表示します。いずれかをクリックすると、コードに適用されます。
[プレビュー]	修正後のコードのプレビューを表示します。エラーを修復してもコードは変わらない場合には、プレビューは表示されません。
[すべてのプレビューを展開]	このボックスをチェックすると、すべてのプレビューが展開されます。

参照

処理手順

方法 : 自動修正でコンパイラ エラーを修正する

概念

エラーの種類

Visual Basic での警告の構成

方法：自動修正でコンパイラ エラーを修正する

スマート コンパイル自動修正機能では、エラーが発生したときに修正方法の候補が表示され、コードに適用する解決策を選択できます。

スマート コンパイル自動修正でエラーを修正するには

1. スマート コンパイル自動修正で修正できるエラーが発生した場合、波線の右下にグリフが表示されます。波線の上にマウス ポインタを置くと、シンボルがスマート タグ パネルに変わります。
2. スマート タグ パネルをクリックするか、またはその上にマウス ポインタを置きます。[エラー修正のオプション] ヘルパー ウィンドウが開き、エラーの説明や、エラーの修復方法の候補が表示されます。適用する修復方法をクリックします。
3. 適用した変更は、[元に戻す] 操作 (Ctrl + Z) を行うと、元に戻すことができます。[エラー修正のオプション] ヘルパー ウィンドウを消すには、エディタの他の部分をクリックします。

参照

処理手順

方法 : Visual Basic コンパイル エラーに関する情報を取得する

関連項目

スマート コンパイル自動修正

Visual Basic での警告の構成

Visual Basic コンパイラには、ランタイム エラーが発生する可能性のあるコードについての一連の警告が用意されています。その情報を使用して、より高速でバグの少ない、すっきりとした優れたコードを作成できます。たとえばユーザーが、未代入のオブジェクト変数のメンバを呼び出そうとしたり、戻り値を設定せずに関数から制御を戻そうとしたり、例外をキャッチするロジックにエラーがある **Try** ブロックを実行しようとしたりすると、コンパイラは警告を生成します。

発生が予想されるエラーの心配をすることなく、目的のタスクにユーザーが専念できるように、コンパイラがユーザーの代わりに追加的なロジックを提供することもあります。以前のバージョンの Visual Basic では、**Option Strict** を使用して、Visual Basic のコンパイラが提供する追加的なロジックを制限していました。警告の構成を使用すると、個々の警告のレベルで、このロジックをよりきめ細かく制御できます。

場合によっては、プロジェクトをカスタマイズして、アプリケーションに無関係な一部の警告をオフにしたり、その他の警告をエラーとして扱うことができます。このページでは、個々の警告をオンおよびオフにする方法を説明します。

警告をオフおよびオンにする

警告を構成する方法は 2 つあります。プロジェクト デザイナを使用する方法と、**/warnaserror** および **/nowarn** の各コンパイラ オプションを使用する方法です。

[プロジェクト デザイナ] ページの [コンパイル] タブを使用すると、警告をオンおよびオフにできます。すべての警告を無効にするには、[すべての警告を表示しない] チェック ボックスをオンにします。すべての警告をエラーとして扱うには、[すべての警告をエラーとして扱う] をオンにします。表示されている表を使用すると、一部の警告をエラーまたは警告として個別に切り替えることができます。

Option Strict を Off に設定した場合、Option Strict に関連する警告をそれぞれ個別に扱うことはできません。Option Strict を On に設定した場合、関連する警告は、そのステータスにかかわらず、エラーとして扱われます。コマンド ライン コンパイラで **/optionstrict:custom** と指定することによって Option Strict を Custom に設定した場合、Option Strict の警告のオンとオフは個別に切り替えることができます。

また、コンパイラの **/warnaserror** コマンド ライン オプションを使用して、警告をエラーとして扱うかどうかを指定することもできます。このオプションにコンマ区切りのリストを指定し、+ または - を指定すると、警告をエラーと警告のどちらとして扱うかを指定できます。使用可能なオプションの詳細を次の表に示します。

コマンド ライン オプション	機能
`/warnaserror+`	すべての警告をエラーとして扱います。
`/warnsaserror-`	警告をエラーとして扱いません。これは、既定の設定です。
`/warnaserror+:<warning list>`	特定の警告をエラーとして扱います。その対象は、コンマ区切りのリストでエラー ID 番号で指定します。
`/warnaserror-:<warning list>`	特定の警告をエラーとして扱いません。その対象は、コンマ区切りのリストでエラー ID 番号で指定します。
`/nowarn`	警告を報告しません。
`/nowarn:<warning list>`	特定の警告を報告しません。その対象は、コンマ区切りのリストでエラー ID 番号で指定します。

警告リストには、エラーとして扱う警告のエラー ID 番号を指定します。これをコマンド ライン オプションで使用して、特定の警告をオンまたはオフにできます。警告リストに無効な番号が含まれている場合、エラーが報告されます。

例

この表は、コマンド ライン引数の例と、各引数の動作の説明です。

引数	説明
`vbc /warnaserror`	すべての警告をエラーとして扱うよう指定しています。
`vbc /warnaserror:42024`	警告 42024 をエラーとして扱うよう指定しています。
`vbc /warnaserror:42024,42025`	警告 42024 および 42025 をエラーとして扱うよう指定しています。

`vbc /nowarn`	警告を一切報告しないよう指定しています。
`vbc /nowarn:42024`	警告 42024 を報告しないよう指定しています。
`vbc /nowarn:42024,42025`	警告 42024 および 42025 を報告しないよう指定しています。

警告の種類

次に示すのは、エラーとして扱うのに適している警告の一覧です。

暗黙の型変換の警告

暗黙の型変換のインスタンスに対して生成されます。**&** 演算子を使用しているときの、組み込みの数値型から文字列への暗黙の型変換は含まれません。新しいプロジェクトの既定値はオフです。

ID: 42016

遅延バインディングによるメソッド呼び出しとオーバーロードの解決の警告

遅延バインディングのインスタンスに対して生成されます。新しいプロジェクトの既定値はオフです。

ID: 42017

型オブジェクトのオペランドの警告

Object 型のオペランドがあり、それによって **Option Strict On** でエラーが発生する場合に生成されます。新しいプロジェクトの既定値はオンです。

ID: 42018 および 42019

宣言に 'As' 句が必要との警告

変数、関数、またはプロパティの宣言に **As** 句がなく、それによって **Option Strict On** でエラーが発生する場合に生成されます。型が割り当てられていない変数は **Object** 型と見なされます。新しいプロジェクトの既定値はオンです。

ID: 42020 (変数宣言)、42021 (関数宣言)、および 42022 (プロパティ宣言)

Null 参照の例外の可能性の警告

値を代入する前に変数を使用している場合に生成されます。新しいプロジェクトの既定値はオンです。

ID: 42104、42030

未使用のローカル変数の警告

宣言したローカル変数を 1 度も参照していない場合に生成されます。既定値はオンです。

ID: 42024

インスタンス変数から共有メンバへのアクセスの警告

インスタンスから共有メンバへアクセスすることで副作用が生じる可能性がある場合、または、インスタンス変数から共有メンバへのアクセスが式の右側にないか、またはそれをパラメータとして渡している場合に生成されます。新しいプロジェクトの既定値はオンです。

ID: 42025

演算子またはプロパティへの再帰的アクセスの警告

演算子またはプロパティを定義するルーチンの本体で、その演算子またはプロパティ自身を使用している場合に生成されます。新しいプロジェクトの既定値はオンです。

ID: 42004 (演算子)、42026 (プロパティ)

戻り値のない関数または演算子の警告

関数または演算子で戻り値を指定していない場合に生成されます。これには、関数と同じ名前を持つ暗黙のローカル変数への **Set** の省略も含まれます。新しいプロジェクトの既定値はオンです。

ID: 42105 (関数)、42016 (演算子)

モジュールでのオーバーロード修飾子の使用の警告

Overloads を **Module** で使用している場合に生成されます。新しいプロジェクトの既定値はオンです。

ID: 42028

Catch ブロックの重複またはオーバーラップの警告

他に定義されている **Catch** ブロックとの関係で、到達することのない **Catch** ブロックがある場合に生成されます。新しいプロジェクトの既定値はオンです。

ID: 42029、42031

参照

処理手順

方法 : コンパイラの警告を有効または無効にする

関連項目

例外処理アシスタント ダイアログ ボックス

/nowarn

/warnaserror (Visual Basic)

Compiler Warnings That Are Off by Default

概念

エラーの種類

その他の技術情報

例外処理のタスク

Visual Basic での構造化例外処理

構造化例外処理では、例外、分離されたコード ブロック、およびフィルタを含む制御構造を使用して、例外処理機構を作成します。構造化例外処理を使用したコードでは、異なる種類のエラーを区別し、それぞれに応じた処理を実行できます。

このセクションの内容

例外処理のタスク

例外をキャッチする方法や、**Catch** ブロックで例外をフィルタ処理する方法など、一般的な例外処理のタスクを一覧表示します。

Visual Basic の構造化例外処理の概要

構造化例外処理のメリットと構造について、概要を示します。

Visual Basic の例外クラス

Exception クラスについて説明します。

チュートリアル : 構造化例外処理

簡単なアプリケーションの作成と例外処理コードの挿入についてのチュートリアルを示します。

例外処理のトラブルシューティング

構造化例外処理の使用時に一般的に発生するエラーを一覧表示し、その修正方法を説明します。

参照

Throw ステートメント (Visual Basic)

Throw ステートメントについて説明し、これを使用して例外をスローする方法を示します。

Try...Catch...Finally ステートメント (Visual Basic)

Try...Catch...Finally ステートメントについて説明し、これを使用してコード ブロックをテストする方法を示します。

関連するセクション

構造化例外処理と非構造化例外処理に適した状況

構造化例外処理と非構造化例外処理の長所について説明します。

非構造化例外処理の概要

非構造化例外処理について説明し、その使用方法を示します。

例外処理のタスク

Visual Basic では、構造化および非構造化の両方の例外 (エラー) 処理がサポートされています。アプリケーション内に独自の例外処理コードを記述することによって、ユーザーが遭遇する可能性のあるエラーの大半を処理できます。構造化例外処理および非構造化例外処理を使用して、発生する可能性のあるエラーの影響がアプリケーションに及ぶのを防ぎます。

このセクションの内容

方法 : Visual Basic で例外をキャッチする

Try...Catch ブロックを使用して例外をキャッチする方法を説明します。

方法 : Visual Basic で例外をスローする

Throw ステートメントを使用して例外をスローする方法を説明します。

方法 : Visual Basic で I/O Try...Catch ブロックを実装する

Try...Catch ブロックを使用して入力処理または出力処理をテストする方法を説明します。

方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する

Try...Catch ブロックを使用してコードのセクションをテストする方法を説明します。

方法 : Visual Basic で Try...Finally ブロックを使用してリソースをクリーンアップする

Try...Catch...Finally ブロックを使用して、コードが使用したリソースをクリーンアップする方法を説明します。

方法 : Visual Basic で Catch ブロックを使用してエラーをフィルタ処理する

Try...Catch ブロック内で特定の型のエラーをフィルタ処理する方法を説明します。

方法 : Visual Basic で例外のメッセージを表示する

エラー メッセージに関連付けられている文字列を取得する方法を説明します。

方法 : Visual Basic で新しい例外クラスを作成する

新しい例外クラスを作成する方法を説明します。

方法 : 例外の内部例外をチェックする

例外の **InnerException** プロパティを参照することによって元の例外を判断する方法を説明します。

参照

Throw ステートメント (Visual Basic)

Throw ステートメントとその使用方法について説明します。

Try...Catch...Finally ステートメント (Visual Basic)

Try、**Catch**、および **Finally** の各ステートメントとその使用方法について説明します。

方法：Visual Basic で例外をキャッチする

この例は、**Try** ブロックと **Catch** ブロックを使用して、例外をキャッチする方法を示します。

使用例

この例は、**Try...Catch** ブロックを使用して、OverflowException をキャッチする方法を示します。

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

VB

```
Dim Top As Double = 5
Dim Bottom As Double = 0
Dim Result As Integer
Try
    Result = CType(Top / Bottom, Integer)
Catch Exc As System.OverflowException
    MsgBox("Attempt to divide by zero resulted in overflow")
End Try
```

この例で必要な要素は次のとおりです。

- System 名前空間への参照

Exception、IOException、および **IOException** から派生したすべての例外を処理する、**Try...Catch** ブロックを実装するコード例を次に示します。

VB

```
Try
    ' Add code for your I/O task here.
Catch dirNotFound As System.IO.DirectoryNotFoundException
    Throw dirNotFound
Catch fileNotFound As System.IO.FileNotFoundException
    Throw fileNotFound
Catch pathTooLong As System.IO.PathTooLongException
    Throw pathTooLong
Catch ioEx As System.IO.IOException
    Throw ioEx
Catch security As System.Security.SecurityException
    Throw security
Catch ex As Exception
    Throw ex
Finally
    ' Dispose of any resources you used or opened in the Try block.
End Try
```

実行するコードを **Try** ブロックに追加します。

堅牢性の高いプログラム

Try...Catch ステートメントでのデータの操作をラッピングするための開始点として、このコードのブロックを使用します。この **Try...Catch** ブロックは、すべての例外をキャッチして再スローするように設計されています。これは、必ずしもプロジェクトに最適な方法でない可能性もあります。例外処理のオプションの詳細については、「IntelliSense コード スニペットの最適な使用方法」を参照してください。

ユーザーが有効なファイル名のみを選択できるように制限されている、OpenFileDialog コンポーネント (Windows フォーム) コンポーネント コントロールおよび SaveFileDialog コンポーネント (Windows フォーム) コンポーネント コントロールなどの Windows フォーム コントロールを使用して、例外の可能性を減らすことができます。FileInfo.Exists プロパティによって、ファイルを開く前にファイルが存在するかどうかをチェックできます。しかし、これらのコントロールやクラスの使用は安全な方法ではありません。ユーザーがファイルを選択してから、コードが実行されるまでの間に、ファイル システムが変更される可能性があります。したがって、ファイルの操作中は、ほとんど常時例外処理が必要になります。

セキュリティ

多くのファイルに関するタスクでは、アセンブリに対して FileIOPermission クラスで特権レベルが許可されている必要があります。部分的に信頼されているコンテキストでプロセスを実行している場合は、権限不足のため例外がスローされることがあります。詳細については、「コード アクセス セキュリティの基礎」を参照してください。また、ユーザーがファイルに対してアクセスしなければならない場合は、権限が必要となります。詳細については、「アクセス制御リスト (ACL)」を参照してください。

ファイル名の拡張子に基づいてファイルの内容を判断しないでください。たとえば、Form1.vb というファイルは Visual Basic のソース ファイルではない可能性もあります。

参照

処理手順

方法 : Visual Basic で例外をスローする
方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する
方法 : Visual Basic で Try...Finally ブロックを使用してリソースをクリーンアップする
方法 : Visual Basic で Catch ブロックを使用してエラーをフィルタ処理する
方法 : 例外の内部例外をチェックする
例外処理のトラブルシューティング

概念

構造化例外処理と非構造化例外処理に適した状況

方法：Visual Basic で例外をスローする

ApplicationException 例外をスローする例を示します。

使用例

VB

```
Throw New ApplicationException
' Code to react to possible causes of the exception.
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

必要な条件は次のとおりです。

- System 名前空間への参照

堅牢性の高いプログラム

例外が処理されなかった場合、アプリケーションの実行が中断する可能性があります。発生する可能性のある例外に備え、処理するには、**Try...Catch** ブロックを使用します。

参照

処理手順

方法 : Visual Basic で例外をキャッチする
方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する
方法 : Visual Basic で Try...Finally ブロックを使用してリソースをクリーンアップする
方法 : Visual Basic で Catch ブロックを使用してエラーをフィルタ処理する
方法 : 例外の内部例外をチェックする
例外処理のトラブルシューティング

関連項目

Exception Class
Throw ステートメント (Visual Basic)

概念

構造化例外処理と非構造化例外処理に適した状況

方法：Visual Basic で I/O Try...Catch ブロックを実装する

Exception、IOException、および **IOException** から派生するすべての例外を処理する **Try...Catch** ブロックを実装するコードの例は、次のとおりです。

使用例

この例では、最も固有な例外から一般的な例外の順に例外を配置します。各々の例外は順にテストされます。

このコード例は、IntelliSense のコード スニペットとしても使用できます。コード スニペット ピッカーでは、コード例は [ファイル システム - ドライブ、フォルダ、およびファイルの処理] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

VB

```
Try
    ' Add code for your I/O task here.
Catch dirNotFound As System.IO.DirectoryNotFoundException
    ' Code to handle DirectoryNotFoundException.
Catch fileNotFound As System.IO.FileNotFoundException
    ' Code to handle FileNotFoundException.
Catch pathTooLong As System.IO.PathTooLongException
    ' Code to handle PathTooLongException.
Catch ioEx As System.IO.IOException
    ' Code to handle IOException.
Catch security As System.Security.SecurityException
    ' Code to handle SecurityException.
Catch ex As Exception
    ' Rethrow exception if anything else has occurred.
    Throw ex
Finally
    ' Dispose of any resources you used or opened in the Try block.
End Try
```

コードのコンパイル方法

Try ブロックに対して実行するコードを追加します。

堅牢性の高いプログラム

このコード ブロックは、**Try...Catch** ステートメントでデータ演算をラッピングするための開始点として使用します。この **Try...Catch** ブロックは、すべての例外をキャッチし、再スローするように設計されています。これは、必ずしもプロジェクトに最適な方法でない場合もあります。例外処理オプションの詳細については、「IntelliSense コード スニペットの最適な使用方法」を参照してください。

有効なファイル名のユーザーの選択肢を制限する OpenFileDialog コンポーネント (Windows フォーム) コンポーネントおよび SaveFileDialog コンポーネント (Windows フォーム) コンポーネントのコントロールなどの Windows フォーム コントロールを使用して、例外発生の可能性を減らすことができます。FileInfo.Exists プロパティは、ファイルを開く前にファイルが存在するかどうかをチェックできます。ただし、これらのコントロールやクラスの使用は、安全な方法ではありません。ユーザーがファイルを選択してからコードが実行されるまでの間に、ファイル システムが変更される可能性があります。したがって、ファイルの操作中はほとんど常時例外処理が必要となります。

セキュリティ

多数のファイル タスクで、アセンブリを実行する際には FileIOPermission クラスで許可された特権レベルが必要となります。部分的に信頼されているコンテキストでプロセスを実行している場合は、権限不足のため例外がスローされることがあります。詳細については、「コード アクセス セキュリティの基礎」を参照してください。また、ユーザーがファイルに対してアクセスしなければならない場合は、権限が必要となります。詳細については、「アクセス制御リスト (ACL)」を参照してください。

ファイル名の拡張子に基づいてファイルの内容を判断しないでください。たとえば、Form1.vb というファイルは Visual Basic のソース ファイルではない可能性もあります。

参照

処理手順

方法 : Visual Basic で Catch ブロックを使用してエラーをフィルタ処理する

方法 : 例外の内部例外をチェックする

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

IntelliSense コード スニペットの最適な使用方法

Visual Basic の構造化例外処理の概要

構造化例外処理と非構造化例外処理に適した状況

その他の技術情報

例外処理のタスク

方法：Visual Basic で Try...Catch ブロックを使用してコードを検査する

Try ブロック内で **Catch** ステートメントを使用すると、特定の例外や複数の例外をキャッチし処理できます。実行時に **Try** セクション内のコードで例外が発生すると、Visual Basic コンパイラは、その例外に一致する条件が指定された **Catch** ステートメントが見つかるまでブロック内を探します。一致する **Catch** ステートメントが見つからなかった場合は、エラーが生成されます。

特定の例外をキャッチするには

1. 次の例のように、コード ブロックを **Try** と **End Try** で囲み、このコード ブロックを **Try** ブロックを使用して検査します。このコード例では、`MyLog` ファイルを同じディレクトリにコピーし、名前を `BackupLog` に変更しています。

   VB

   ```
   Try
       My.Computer.FileSystem.CopyFile("MyLog", "BackupLog")
   Catch ex As System.IO.IOException
       MsgBox("An error occurred")
   End Try
   ```

2. **Try** ブロック内に、特定のエラーをキャッチするための複数の **Catch** ステートメントを指定します。このとき、特殊なエラーほど先にキャッチし、より一般的なエラーへと順にキャッチしていくようにします。この例では、最初に **Catch** ステートメントで IOException 例外をキャッチし、それから一般的な例外を検査しています。

   VB

   ```
   Catch ex As System.IO.FileNotFoundException
       MsgBox("No such file in this directory.")
   Catch ex As System.Exception
       MsgBox("An unspecified error occurred.")
   ```

参照

処理手順

方法 : Visual Basic で Try...Finally ブロックを使用してリソースをクリーンアップする
方法 : Visual Basic で Catch ブロックを使用してエラーをフィルタ処理する
方法 : 例外の内部例外をチェックする
例外処理のトラブルシューティング

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

構造化例外処理と非構造化例外処理に適した状況

その他の技術情報

例外処理のタスク

方法：Visual Basic で Try...Finally ブロックを使用してリソースをクリーンアップする

Try ブロック内で **Finally** ステートメントを使用すると、割り当てられているリソースを確実に解放できます。**Finally** ブロック内のコードは、例外処理コードの後で、呼び出し元のプロシージャに制御を返す前に実行されます。**Finally** ブロック内のコードは、たとえコードで例外がスローされた場合でも、また、**Catch** ブロック内に明示的な **Exit Function** ステートメント (または **Exit Sub** ステートメント) がある場合でも実行されます。

特定の例外をキャッチする必要がない場合には、**Using** ステートメントを使用すると **Try...Finally** ブロックと同じように動作し、どのようにブロック内の処理が終了する場合でも確実にリソースを解放できます。これは、未処理の例外の場合にも該当します。詳細については、「Using ステートメント (Visual Basic)」を参照してください。

Finally ステートメントを使用してリソースをクリーンアップするには

- 例外の種類にかかわらず実行するコードを、**Finally** ブロック内に配置します。次のコードでは、StreamReader を作成し、これを使用してファイルを読み込みます。

  VB

```
Dim reader As New System.IO.StreamReader("C:\testfile")
Try
    reader.ReadToEnd()
Catch ex As System.IO.IOException
    MsgBox("Could not read file")
Finally
    'This command is executed whether or not the file can be read
    reader.Close()
End Try
```

参照

処理手順

方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する
方法 : 例外の内部例外をチェックする
方法 : システム リソースを破棄する

関連項目

Using ステートメント (Visual Basic)

概念

例外処理 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

例外処理のタスク

方法：Visual Basic で Catch ブロックを使用してエラーをフィルタ処理する

Catch ステートメントでは、いくつかの方法でエラーをフィルタできます。1 つは、例外の種類によってフィルタする方法です。この場合、特殊な例外から一般的な例外へと順にキャッチしていくことが重要です。**Catch** ステートメントは、指定されている順序で実行されるためです。

特定のエラー番号などの条件式によってフィルタする場合には、**When** 句も使用できます。両方の方法を組み合わせて使用することもできます。

メモ：
使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

例外の種類によってフィルタするには

- チェックする例外のそれぞれの種類について、最も特殊な例外から一般的な例外へと順に **Catch** ステートメントを挿入します。

  VB

```
Try
    Throw New Exception
Catch ex As System.IO.IOException
    ' Code reacting to IOException
Catch ex As System.NullReferenceException
    ' Code reacting to NullReferenceException
Catch ex As Exception
    ' Code reacting to any exception
End Try
```

条件式によってフィルタするには

- 条件式によってフィルタする場合は、**Catch When** ステートメントを使用します。条件式が **True** に評価されると、その **Catch** ブロックに続くコードが実行されます。

  VB

```
Try
    ' Code goes here.
    ' Check for type mismatch error.
Catch ex As Exception When Err.Number = 5
    ' Code reacting to exception.
End Try
```

参照

処理手順

方法 : 例外の内部例外をチェックする

例外処理のトラブルシューティング

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

例外処理 (Visual Basic 6.0 ユーザー向け)

構造化例外処理と非構造化例外処理に適した状況

その他の技術情報

例外処理のタスク

方法：Visual Basic で例外のメッセージを表示する

例外のオブジェクトには、コードの場所、型、ヘルプファイルの URL、および例外の理由の識別に役立つ、いくつかのプロパティが含まれています。これらの 1 つには、現在の例外を説明する **Message** プロパティがあります。

例外に関連付けられた文字列の表示

- **Message** プロパティを使用して、現在の例外に関する情報を表示します。この例では、WebException をキャッチし、関連付けられたメッセージを表示します。

  VB

```
Try
    Throw New System.Net.WebException
Catch ex As System.Net.WebException
    MsgBox("An exception occurred:" & vbCrLf & ex.Message)
End Try
```

参照

処理手順

方法 : Visual Basic で例外をキャッチする
方法 : Visual Basic で例外をスローする
方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する
方法 : 例外の内部例外をチェックする
例外処理のトラブルシューティング

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

その他の技術情報

Visual Basic での構造化例外処理

方法：Visual Basic で新しい例外クラスを作成する

Exception クラスを継承することによって、アプリケーションの例外クラスを独自に作成できます。例外のクラス名は、末尾を `Exception` にすることをお勧めします。たとえば、`OutOfMoneyException` や `TooMuchRainException` のようなクラス名を使用します。

例外クラスを実装するための基本的なコード例を次に示します。

使用例

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

VB

```
Public Class YourProblemException
    Inherits Exception

    Public Sub New()
        ' Add other code for custom properties here.
    End Sub

    Public Sub New(ByVal message As String)
        MyBase.New(message)
        ' Add other code for custom properties here.
    End Sub

    Public Sub New(ByVal message As String, ByVal inner As Exception)
        MyBase.New(message, inner)
        ' Add other code for custom properties here.
    End Sub

    Public Sub New( _
        ByVal info As System.Runtime.Serialization.SerializationInfo, _
        ByVal context As System.Runtime.Serialization.StreamingContext)
        MyBase.New(info, context)
        ' Insert code here for custom properties here.
    End Sub
End Class
```

コードのコンパイル方法

- `YourProblemException` の部分を、作成する例外クラスの名前に置換してください。通常、例外クラスの名前は末尾を "Exception" とします。発生したエラーに関する追加情報を受け渡すには、プロパティを追加します。

セキュリティ

例外処理では、アプリケーションやアプリケーションのデータに関する情報を公開しないようにしてください。このような情報はアプリケーションに対する攻撃に使用される可能性があります。

参照

処理手順

例外処理のトラブルシューティング

関連項目

ApplicationException Class

概念

Visual Basic の構造化例外処理の概要

例外処理 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

例外処理のタスク

方法：例外の内部例外をチェックする

前の例外の直接的な結果として例外がスローされた場合、InnerException プロパティによって元のエラーが説明されます。この情報により、エラーをより効率的に処理できます。元のエラーがない場合、**InnerException** の値は、null 参照または Visual Basic の **Nothing** になります。このプロパティは読み取り専用です。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

内部例外をチェックするには

- 例外の **InnerException** プロパティをチェックして、元のエラーの原因を判断します。

  VB

```
Try
    My.Computer.FileSystem.CopyFile("file1", "file2")
Catch ex As System.IO.IOException
    MsgBox(ex.InnerException)
End Try
```

参照

処理手順

方法 : Visual Basic で例外のメッセージを表示する

例外処理のトラブルシューティング

関連項目

InnerException

その他の技術情報

例外処理のタスク

Visual Basic の構造化例外処理の概要

Visual Basic は、構造化例外処理をサポートしています。構造化例外処理を使うと、信頼性が高く包括的なエラー ハンドラによって、プログラムの作成および管理を行うことができます。構造化例外処理とは、実行時にエラーの検出および処理を行うためのコードであり、制御構造 (**Select Case** や **While** に似た制御構造) と、例外、コードの保護ブロック、およびフィルタが組み合わされています。

Try...Catch...Finally ステートメントを使用すると、エラーが発生する可能性のあるコードのブロックを保護できます。例外ハンドラを入れ子にすることもできます。この場合、それぞれのブロックで宣言した変数は、ローカル スコープを持ちます。

Try...Catch...Finally ステートメント

Try...Catch...Finally ステートメントの構造体を、次のコードに示します。

```
Try
' Starts a structured exception handler.
' Place executable statements that may generate
' an exception in this block.
Catch [optional filters]
' This code runs if the statements listed in
' the Try block fail and the filter on the Catch statement is true.
[Additional Catch blocks]
Finally
' This code always runs immediately before
' the Try statement exits.
End Try
' Ends a structured exception handler.
```

Try...Catch...Finally 例外ハンドラの **Try** ブロックには、例外が監視されるコードのセクションが含まれています。このセクションの実行中にエラーが発生した場合、そのエラーに一致する条件が見つかるまで、**Try...Catch...Finally** 内のそれぞれの **Catch** ステートメントが Visual Basic によって調べられます。一致が見つかった場合、その **Catch** ブロックの最初の行に制御が移ります。一致する **Catch** ステートメントが見つからなかった場合は、例外が発生したブロックの外側にある **Try...Catch...Finally** ブロックの **Catch** ステートメントが検索されます。このプロセスは、現在のプロシージャで一致する **Catch** ブロックが見つかるまで、スタック全体にわたって続けられます。一致が見つからなかった場合は、エラーが生成されます。

Finally セクション内のコードは、**Catch** ブロック内のコードが実行されたかどうかに関係なく、常に最後 (エラー処理ブロックがスコープを失う直前) に実行されます。**Finally** セクションには、クリーンアップ コード (ファイルを閉じたりオブジェクトを解放したりするコードなど) を配置します。例外をキャッチする必要はないけれども、リソースをクリーンアップする必要がある場合、**Finally** セクションではなく、**Using** ステートメントを使用します。詳細については、「Using ステートメント (Visual Basic)」を参照してください。

Catch ブロックにおけるエラーのフィルタ処理

Catch ブロックで特定のエラーをフィルタ処理するには、3 とおりの方法があります。まず、次のように、例外のクラス (次の例では `ClassLoadException`) に基づいてエラーをフィルタ処理する方法があります。

```
Try
' "Try" block.
Catch e as ClassLoadException
' "Catch" block.
Finally
' "Finally" block.
End Try
```

`ClassLoadException` エラーが発生すると、指定した **Catch** ブロック内のコードが実行されます。

エラーをフィルタ処理する 2 番目の方法では、**Catch** セクションで任意の条件式をフィルタ処理できます。この型の **Catch** フィルタは、たとえば次のコードのように、特定のエラー番号をテストする場合などによく使用されます。

```
Try
   ' "Try" block.
Catch When ErrNum = 5 'Type mismatch.
   ' "Catch" block.
Finally
   ' "Finally" block.
```

```
End Try
```

一致するエラー ハンドラが Visual Basic で見つかると、そのハンドラ内のコードが実行され、**Finally** ブロックに制御が渡されます。

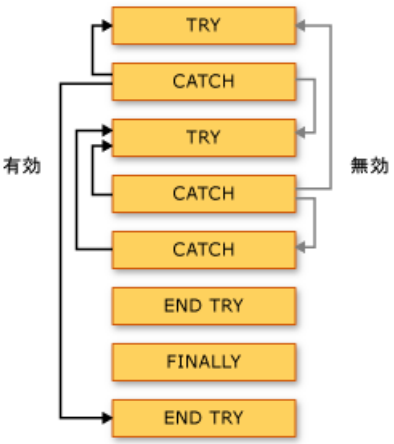
メモ：

例外を処理するために **Catch** ブロックを見つけるときには、一致が見つかるまで各ブロックのハンドラが評価されます。これらのハンドラは関数の呼び出しである場合もあるため、予期しない副作用が起こることがあります。たとえば、そのような呼び出しによってパブリック変数が変更され、後でその変数が、最終的に例外を処理する別の **Catch** ブロックのコードで使用される場合があります。

3 番目の方法として、1 番目と 2 番目の方法を組み合わせ、両方を使用して例外処理を行うこともできます。**Catch** ステートメントの特定性を最大から最小に移行する必要があります。**Catch** ブロック自体は、Exception から派生したすべての例外をキャッチします。したがって、常に **Finally** の前の、最後のブロックである必要があります。

Try...Catch ブロックの分岐

Catch ブロックから、最初の **Try** ステートメントまたは **End Try** ステートメントに分岐することは可能です。しかし、囲まれた **Try...Catch** ブロック内への分岐はできません。これについて、次に示します。



構造化例外処理の例

Try...Catch...Finally ステートメントを使った単純なエラー ハンドラの例を次に示します。

```
Function GetStringsFromFile(ByVal FileName As String) As Collection
Dim Strings As New Collection
Dim Stream As System.IO.StreamReader = System.IO.File.OpenText(FileName)   'Open the file.

Try
While True
' Loop terminates with an EndOfStreamException
' error when end of stream is reached.
Strings.Add(Stream.ReadLine())
End While
Catch eos As System.IO.EndOfStreamException
' No action is necessary; end of stream has been reached.
Catch IOExcep As System.IO.IOException
' Some kind of error occurred. Report error and clear collection.
MsgBox(IOExcep.Message)
Strings = Nothing
Finally
Stream.Close()
' Close the file.
End Try

Return Strings
End Function
```

Finally ブロックは、前の各 **Catch** ブロックで行われた動作に関係なく、常に実行されます。構造化例外処理で **Resume** または **Resume Next** を使用することはできません。

メモ：

前の例では、**IOException** クラスや EndOfStreamException クラス以外の例外は、処理されずに呼び出し元に戻されます。

参照

処理手順

例外処理のトラブルシューティング

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

例外処理の概要

エラーの種類

非構造化例外処理の概要

その他の技術情報

例外処理のタスク

Visual Basic の例外クラス

Visual Basic では、構造化例外処理を使いやすくするために、標準のコードと例外処理コードを分けることができます。例外処理コードは、Exception クラスのインスタンスにアクセスして、発生した例外に関する情報を取得します。

解説

例外がスローされると、グローバルな **Err** オブジェクトが設定され、**Exception** クラスの新しいインスタンスが作成されます。

Exception クラスのプロパティは、例外が発生したコードの位置、例外の種類、およびその原因を特定するのに役立ちます。たとえば、StackTrace プロパティでは、例外が発生するまでに呼び出されたメソッドの一覧を確認できるため、コードのどこでエラーが発生したのかを突き止めるのに便利です。Message プロパティは、エラーを説明するテキスト メッセージを返します。このメッセージをわかりやすいメッセージに変更することもできます。エラー メッセージ テキストの文字列を指定していない場合、既定値が使用されます。HelpLink により、関連付けられたヘルプ ファイルへのリンクが取得または設定されます。Source により、エラーの原因となるオブジェクトの名前、または例外の発生元であるアセンブリの名前が含まれる文字列が、取得または設定されます。

メモ：
特定のエラー メッセージをテストするコードを記述しないでください。生成されるメッセージは、クラスのバージョン間で変わる可能性があります。その代わりに、例外の種類をテストするか、またはグローバルな **Err** オブジェクトの例外番号を使用してください。

例外クラスのプロパティ

Exception クラスのプロパティについて、次の表で説明します。

プロパティ	説明
HelpLink	この例外に関連付けられたヘルプ ファイルへのリンクを、取得または設定します。
HResult	特定の例外に割り当てられた、コード化された数値である、`HRESULT` を取得または設定します。
InnerException	現在の例外の原因となる **Exception** インスタンスを取得します。
Message	現在の例外を説明するメッセージを取得します。
Source	エラーの原因となるアプリケーションまたはオブジェクトの名前を、取得または設定します。
StackTrace	現在の例外がスローされたときの、コール スタックのフレームの文字列形式を取得します。
TargetSite	現在の例外がスローされたメソッドを取得します。

参照

処理手順

例外処理のトラブルシューティング

概念

例外処理の概要

エラーの種類

Visual Basic の構造化例外処理の概要

非構造化例外処理の概要

その他の技術情報

例外処理のタスク

チュートリアル：構造化例外処理

非構造化例外処理を行う場合は、従来と同様に **On Error** ステートメントを使用してコードの例外を処理できます。一方、Visual Basic 2005 では構造化例外処理もサポートされています。構造化例外処理を使用すると、包括的な例外処理を備えたプログラムの作成や、管理を行うことができます。構造化例外処理では、コード ブロックが特定の状況をテストし、適切な処理を行います。

このチュートリアルでは、構造化例外処理をプログラムに追加する方法を示します。特に、**Try...Catch...Finally** ステートメントを使用して例外を処理する方法、および **Catch** ブロック内でエラーをフィルタ処理する方法について示します。

メモ：
構造化例外処理と非構造化例外処理を 1 つのプロシージャの中で組み合わせて使用することはできません。

メモ：
使用している設定またはエディションによっては、ヘルプの記載と異なるダイアログ ボックスやメニュー コマンドが表示される場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

アプリケーションの作成

次のアプリケーションは、テディ ベアを販売する会社の顧客注文フォームです。ユーザー インターフェイスは、次のコントロールから構成されています。

- 顧客の名前用の TextBox が 1 つ。
- テディベアの色とサイズを選択するための ComboBox コントロールが 2 つ。
- 注文用の Button が 1 つ。
- 各コントロールの目的をユーザーに伝えるラベルが 3 つ。
- ユーザーが必要な情報を入力して [Order] ボタンをクリックすると、このアプリケーションは注文の概要を表示します。

アプリケーションを作成するには

1. [ファイル] メニューの [新しいプロジェクト] をクリックします。[新しいプロジェクト] ダイアログ ボックスが表示されます。
2. [プロジェクトの種類] ウィンドウの、[Visual Basic]、[Windows] を選択し (まだこれらが選択されていない場合)、[テンプレート] ウィンドウで [Windows アプリケーション] をクリックします。
3. [プロパティ] ウィンドウで、[プロジェクト名] に「**TeddyBearProject**」と入力し、[OK] をクリックします。プロジェクトがソリューション エクスプローラに追加され、Windows フォーム デザイナが表示されます。
4. コントロールをフォームに追加し、指定されたとおりにプロパティを設定します。

コントロール	プロパティ	プロパティ値
Label	名前 テキスト	customerLabel Bear Order Form
TextBox	名前 テキスト	customerName Customer Name
Label	名前 テキスト	bearColorLabel Available Colors
ComboBox	名前 項目 テキスト	bearColor Black, Brown, Spotted Bear Color

Label	名前 テキスト	bearSizeLabel Available Sizes
ComboBox	名前 項目 テキスト	bearSize Small, Normal, Large Size
Button	名前 テキスト	order Order

機能の追加

コントロールとそのプロパティを追加および設定したところで、これらのコントロールを機能させるためのコードを記述する必要があります。次のコードは、[Order] ボタンの Click イベントを処理し、顧客にメッセージを表示します。

フォームおよびコントロールに機能を追加するには

- [Order] ボタンの **Click** イベントに次のコードを追加します。

  VB

```
Dim bearOrder As String
bearOrder = _
  String.Format("You have ordered a {0} {1} bear.", _
  bearSize.SelectedItem, bearColor.SelectedItem)
MsgBox(bearOrder)
```

この時点で、構造化例外処理コードをアプリケーションに追加できます。

Try...Catch ブロックの追加

顧客がぬいぐるみの色を必ず指定するように、**Try...Catch** ステートメントを追加します。次のことに留意してください。

- 識別子のない **Catch** 句はすべての例外をキャッチします。
- **When** 句のある **Catch** 句は、式が **True** に評価されたときだけ例外をキャッチします。式の型は **Boolean** に暗黙に変換できる必要があります。

単純な Try...Catch ブロックを追加するには

- [Order] ボタンの **Click** イベントにある、サイズと色の値をテストするセクション、つまり `String.Format ("You have ordered a {0} {1} bear.", BearSize.SelectedItem, BearColor.SelectedItem)` の後に、次のコードを追加します。このコードは、色の値が無効な場合に例外をスローします。**Try** ステートメントを追加すると、エディタによって自動的に **End Try** がステートメントの最後に追加されます。

  VB

```
Try
    If ((bearColor.SelectedIndex < 0) Or _
          (bearColor.SelectedIndex > 2)) Then
        Throw New System.Exception()
    End If
    ' The Catch statement handles errors caused by a lack of bear color.
Catch ex As Exception When bearColor.SelectedIndex < 0
    bearOrder = String.Format("You must select a bear color!")
Finally
    Beep() ' Beep at the end.
End Try
```

別の Catch 句を追加するには

1. 新しい項目 "Purple" を `bearColor` コンボ ボックス コントロールに追加します。
2. `bearOrder = String.Format("You must select a bear color!")` コード行の後ろに、次のコードを追加します。

   VB

```
Catch ex As Exception When bearColor.SelectedIndex = 3
    bearOrder = String.Format("There are no bears of that color.")
```

必要に応じて任意の数の **Catch** 句をコードに追加できます。

テスト

アプリケーションをテストして、正常に動作することを確認します。

アプリケーションをビルドして実行するには

1. [ビルド] メニューの [TeddyBearProject のビルド] をクリックします。
2. F5 キーを押してアプリケーションを実行します。メイン フォームが表示されます。

アプリケーションをテストするには

1. [Customer name] ボックスに名前を入力し、[Available Colors] ボックスで色を、[Available Sizes] ボックスでサイズを選択します。
2. [Order] ボタンをクリックします。前の手順で [Black]、[Brown]、または [Spotted] のいずれかの色を指定した場合は、注文したぬいぐるみのサイズと色を示すメッセージが表示されます。

   色を指定しなかった場合は、色の指定を求めるメッセージが表示されます。
3. メッセージをキャンセルするには、[OK] をクリックします。
4. [Available Colors] ボックスで [Purple] を選択し、[Order] ボタンをクリックします。紫色のテディ ベアは選択できないことを示すメッセージが表示されます。

参照

関連項目

On Error ステートメント (Visual Basic)
Try...Catch...Finally ステートメント (Visual Basic)
Throw ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

その他の技術情報

例外処理のタスク

例外処理のトラブルシューティング

このトピックでは、例外を扱うときによく生じる問題とその対処方法を説明します。

Visual Basic は、構造化例外処理をサポートしています。構造化例外処理を使うと、信頼性が高く包括的なエラー ハンドラによって、プログラムの作成および管理を行うことができます。構造化例外処理とは、実行時にエラーの検出および処理を行うためのコードであり、制御構造 (**Select Case** や **While** に似た制御構造) と、例外、コードの保護ブロック、およびフィルタが組み合わされています。

内部例外

前の例外の直接的な結果として例外がスローされた場合、InnerException プロパティによって元のエラーが説明されます。この情報により、エラーをより効率的に処理できます。元のエラーがない場合、**InnerException** の値は、null 参照または Visual Basic の **Nothing** になります。このプロパティは読み取り専用です。詳細については、「方法 : 例外の内部例外をチェックする」を参照してください。

Try...Catch ステートメント

コードの **Catch** ブロックの順序が間違っていると、例外を正しくキャッチできない場合があります。**Catch** ステートメントは、具体的なものから汎用的なものへという順序で記述する必要があります。**Catch** ブロック単独では、Exception から派生したすべての例外をキャッチします。したがって、常に **Finally** の直前のブロックとして配置する必要があります。

参照

処理手順

チュートリアル : 構造化例外処理

概念

構造化例外処理と非構造化例外処理に適した状況

その他の技術情報

例外処理のタスク

Visual Basic での非構造化例外処理

このセクションのトピックでは、非構造化例外処理、**Err** オブジェクト、およびエラー情報の取得方法について説明します。

このセクションの内容

非構造化例外処理の概要

非構造化例外処理と **Err** オブジェクトの使用方法について説明します。

非構造化例外処理での Err オブジェクト

Err オブジェクトとそのプロパティについて説明します。

方法 : エラー オブジェクトから情報を取得する

Err オブジェクトのプロパティから情報を取得する方法について説明します。

Visual Basic でトラップできるエラー

トラップできるエラーを一覧表示します。

参照

Err オブジェクト (Visual Basic)

Err オブジェクトについて説明し、そのメンバを一覧表示します。

Description プロパティ (Err オブジェクト)

Err オブジェクトの **Description** プロパティとその使用方法について説明します。

HelpContext プロパティ (Err オブジェクト)

Err オブジェクトの **HelpContext** プロパティとその使用方法について説明します。

HelpFile プロパティ (Err オブジェクト)

Err オブジェクトの **Helpfile** プロパティとその使用方法について説明します。

LastDllError プロパティ (Err オブジェクト)

Err オブジェクトの **LastDLLError** プロパティとその使用方法について説明します。

Number プロパティ (Err オブジェクト)

Err オブジェクトの **Number** プロパティとその使用方法について説明します。

Source プロパティ (Err オブジェクト)

Err オブジェクトの **Source** プロパティとその使用方法について説明します。

On Error ステートメント (Visual Basic)

On Error ステートメントとその使用方法について説明します。

Resume ステートメント

Resume ステートメントとその使用方法について説明します。

Exit ステートメント (Visual Basic)

Exit ステートメントとその使用方法について説明します。

関連するセクション

構造化例外処理と非構造化例外処理に適した状況

構造化例外処理と非構造化例外処理の長所および短所について説明します。

エラーの種類

コード内で出現するエラーの種類を一覧表示します。

非構造化例外処理の概要

非構造化例外処理では、コードのブロックの先頭に配置する **On Error** ステートメントによって、そのブロック内で発生するすべてのエラーが処理されます。**On Error** ステートメントの実行後にプロシージャ内で例外が発生すると、**On Error** ステートメントの引数 *line* で指定した行にプログラムが分岐します。引数 *line* には、例外ハンドラの場所を示す行番号または行ラベルを指定します。

元のプロシージャから他のプロシージャが呼び出され、呼び出されたプロシージャで例外が発生することがあります。このような場合、呼び出されたプロシージャによって例外が処理されないと、例外が呼び出し元のプロシージャに戻されて、引数 *line* で指定した行に実行が分岐します。

メモ：

On Error を使った非構造化のエラー処理では、アプリケーションのパフォーマンスの低下を招いたり、コードのデバッグや管理が難しくなったりする可能性があります。構造化例外処理を使用することをお勧めします。詳細については、「Visual Basic の構造化例外処理の概要」を参照してください。

On Error GoTo Line

On Error GoTo Line ステートメントでは、必須の *line* 引数で指定した行でエラー処理コードが開始されると想定されています。ランタイム エラーが発生すると、引数で指定した行ラベルまたは行番号に制御が分岐して、エラー ハンドラがアクティブになります。**On Error GoTo Line** ステートメントと同じプロシージャ内の行を指定する必要があります。それ以外の行を指定すると、Visual Basic でコンパイル エラーが発生します。行ラベルを使ったエラー ハンドラの使用例を次に示します。

```
Sub TestSub
   On Error GoTo ErrorHandler
      ' Code that may or may not contain errors.
   Exit Sub

   ErrorHandler:
      ' Code that handles errors.
      Resume
End Sub
```

この例には、`ErrorHandler` というエラー ハンドラが含まれています。`TestSub` サブルーチン内のコードがエラーを生成すると、`ErrorHandler` ラベルに続くコードがすぐに Visual Basic で実行されます。エラー処理ブロックの最後に、**Resume** ステートメントによって、最初にエラーが発生したコードの行に制御が戻されます。その後、エラーが発生しなかった場合と同じように、サブルーチンの残りのコードが実行されます。

メモ：

エラー処理ブロックの直前に **Exit Sub** ステートメントを配置する必要があります。そうしないと、Visual Basic によってサブルーチンの最後にエラー処理コードが実行され、予想外の結果になる可能性があります。

On Error Resume Next

On Error Resume Next ステートメントを使用すると、ランタイム エラーが発生した場合に、エラーが発生したステートメントの次のステートメントに制御が渡されます。この時点では引き続き実行され、**On Error Resume Next** ステートメントによって、プロシージャ内の他の場所に制御を移す代わりに、エラーが発生する可能性がある場所にエラー処理ルーチンを配置できます。

メモ：

プロシージャによって他のプロシージャが呼び出された場合、呼び出されたプロシージャの実行中に、**On Error Resume Next** ステートメントが非アクティブになります。このため、呼び出されるプロシージャごとに、必要に応じて **On Error Resume Next** ステートメントを配置する必要があります。これは、**Resume Next** の動作が、**On Error Resume Next** ステートメントを含むプロシージャにしか適用されないためです。呼び出されたプロシージャで処理できないエラーが発生すると、例外が呼び出し元のプロシージャに戻され、その呼び出しの次のステートメントから実行が再開されます。この場合、エラーは処理されません。

Resume は、**On Error** ステートメントの外で、単独で使用することもできます。**Resume** を単独で使用すると、Visual Basic によって、エラーを引き起こしたステートメントに制御が返されます。**Resume** は、通常、エラー ハンドラがエラーを修正した後に使用されます。

Visual Basic には、エラーが発生した行の次の行に制御を移す **Resume Next** ステートメントも用意されています。**Resume Next** は、エラーによってアプリケーションが停止しない場合に使用できます。また、エラーによってサブルーチンの結果が変わらない場合にも使用できます。

Resume ステートメントでの他のバリエーションには、**On Error GoTo Line** に類似した **Resume Line** があります。**Resume Line** によって、*line* 引数で指定した行に制御が渡されます。**Resume Line** を使用できるのは、エラー ハンドラの中だけです。

メモ：
コードをデバッグするときは、**On Error Resume Next** ステートメントを無効にする必要があります。

On Error GoTo 0

On Error GoTo 0 ステートメントにより、現在のプロシージャのエラー ハンドラが無効化されます。**On Error GoTo 0** ステートメントがなくても、例外ハンドラを含むプロシージャが終了すると、エラー ハンドラは無効になります。

メモ：
On Error GoTo 0 ステートメントは、プロシージャに行番号 0 の行が含まれている場合でも、エラー処理コードの開始行として行 0 を指定するわけではありません。

On Error GoTo -1

On Error GoTo -1 ステートメントにより、現在のプロシージャの例外ハンドラが無効化されます。**On Error GoTo -1** ステートメントがなくても、プロシージャが終了すると例外は自動的に無効になります。

メモ：
On Error GoTo -1 ステートメントは、プロシージャに行番号 -1 の行が含まれている場合でも、エラー処理コードの開始行として行 -1 を指定するわけではありません。

非構造化例外処理の例

`DivideByZero` という例外ハンドラが特定のエラー (この場合は 0 による除算) を処理する例を次に示します。これ以外のエラーが発生すると、Visual Basic でランタイム エラーが発生してアプリケーションが停止します。

```
Sub ErrorTest ()
' Declare variables.
   Dim x As Integer, y As Integer, z As Integer
   ' The exception handler is named "DivideByZero".
   On Error GoTo DivideByZero
   ' The main part of the code, which might cause an error.
   x = 2
   y = 0
   z = x \ y

   ' This line disables the exception handler.
   On Error GoTo 0
      Console.WriteLine(x & "/" & y & " = " & z)

   ' Exit the subroutine before the error-handling code.
   ' Failure to do so can create unexpected results.
   Exit Sub

   ' This is the exception handler, which deals with the error.
   DivideByZero:
   ' Include a friendly message to let the user know what is happening.
   Console.WriteLine("You have attempted to divide by zero!")

   ' Provide a solution to the error.
   y = 2

   ' The Resume statement returns to the point at which
   ' the error first occurred, so the application
   ' can continue to run.
   Resume

End Sub
```

参照

処理手順

方法 : エラー オブジェクトから情報を取得する

関連項目

End ステートメント
Err オブジェクト (Visual Basic)
Exit ステートメント (Visual Basic)
On Error ステートメント (Visual Basic)
Resume ステートメント

概念

非構造化例外処理での Err オブジェクト
例外処理の概要
エラーの種類
構造化例外処理と非構造化例外処理に適した状況

非構造化例外処理での Err オブジェクト

エラー処理コードを記述するには、どのエラーが発生したのかを知ることが必要です。非構造化例外処理では、Err オブジェクト (Visual Basic) を使用して検知できます。

Err オブジェクトには、発生したエラーによって値が決まるプロパティがあります。**Number** プロパティには、エラーの原因が含まれています。**Description** プロパティには、エラーの詳細を説明するテキスト メッセージが含まれています。**Helpfile** と **HelpContext** を使用すると、ユーザーが [ヘルプ] ボタンまたは F1 キーを押したときに、関連するヘルプ ファイルを表示させることができます。**LastDLLError** は、最後に呼び出された DLL と、呼び出しが正常に実行されたかどうかを示します。**Source** は、エラーを生成したオブジェクトまたはプログラムを表す文字列式を示します。

エラー ハンドラは、他のエラーが発生する前に、関連するプロパティ値をテストまたは保存する必要があります。コードで一度に処理できるエラーは 1 つだけなので、エラー処理が完了してからでないと他のエラーの処理を始めることはできません。

メモ：
Err オブジェクトは、**On Error GoTo** ステートメントによってキャッチされた例外についてのみ使用できます。

アプリケーション固有のエラー

Visual Basic によって生成されるエラーの他に、アプリケーション固有のエラーを処理することもできます。

他のオブジェクトにアクセスするオブジェクトを作成する場合は、アクセス先のオブジェクトが戻す未処理のエラーを処理するためのコードも記述する必要があります。必要な場合は、**Err.Number** のエラー コードをアプリケーション固有のエラーのいずれかに割り当てて、オブジェクトの呼び出し元に渡すこともできます。

参照

関連項目

On Error ステートメント (Visual Basic)
Description プロパティ (Err オブジェクト)
Number プロパティ (Err オブジェクト)
HelpFile プロパティ (Err オブジェクト)
HelpContext プロパティ (Err オブジェクト)
LastDllError プロパティ (Err オブジェクト)
Source プロパティ (Err オブジェクト)

概念

例外処理の概要
エラーの種類
Visual Basic の構造化例外処理の概要
非構造化例外処理の概要

方法：エラー オブジェクトから情報を取得する

ランタイム エラーが発生すると、そのエラーを一意に識別するための情報と、エラー処理に使用できる情報が、**Err** オブジェクトに格納されます。

Err オブジェクトの各プロパティは、エラー処理ルーチン内の **On Error Resume Next** ステートメントおよび **Exit Sub** または **Exit Function** ステートメントの後で、0 または長さ 0 の文字列 ("") にリセットされます。**Clear** メソッドを使用すると、**Err** を明示的にリセットできます。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

エラー オブジェクトから情報を取得するには

1. フィルタをかけることで特定のエラーのみ処理できます。この例では、エラーをチェックし、FileNotFound エラーである場合には対応する処理を実行します。

   VB

   ```
   If Err.Number = 53 Then
     MsgBox("File Not Found")
   End If
   ```

2. また、**Description**、**Erl**、**HelpContext**、**Helpfile**、**LastDLLError**、**Number**、**Source** などエラー オブジェクトの特定のプロパティを確認することもできます。この例では、メッセージ ボックスに説明を表示します。

   VB

   ```
   MsgBox(Err.Description)
   ```

参照

関連項目

Err オブジェクト (Visual Basic)
Err オブジェクトのメンバ
Description プロパティ (Err オブジェクト)
Erl プロパティ (Err オブジェクト)
HelpContext プロパティ (Err オブジェクト)
HelpFile プロパティ (Err オブジェクト)
LastDllError プロパティ (Err オブジェクト)
Number プロパティ (Err オブジェクト)
Source プロパティ (Err オブジェクト)

概念

構造化例外処理と非構造化例外処理に適した状況

アプリケーションの配置

.NET Framework には、ClickOnce 配置など、さまざまなアプリケーションを簡単に配置できるようにするためのいくつかの基本機能が用意されています。

.NET Framework の配置機能

.NET Framework の以下の基本機能を使用すると、配置が簡単になります。

- ゼロインパクト アプリケーション
- 制御コードの共有
- side-by-side でのバージョン管理
- 実行時更新
- 部分信頼のコード

詳細については、「.NET Framework の配置機能」を参照してください。

配置の基本

配置とは、完成したアプリケーションやコンポーネントをほかのコンピュータにインストールできるように配布するためのプロセスです。Visual Studio での配置は、Microsoft Windows インストーラ テクノロジと ClickOnce テクノロジのいずれかに基づいて行うことができます。詳細については、「配置ストラテジの選択」を参照してください。

アプリケーションのパッケージ化は、単一のアセンブリ、複数のアセンブリのコレクション、キャビネット (CAB) ファイル、Microsoft Windows インストーラ 2.0 パッケージ、または他の形式のインストーラ パッケージのいずれかの形で行うことができます。詳細については、「.NET Framework アプリケーションの配置シナリオ」を参照してください。

アセンブリは、相互に連携して 1 つの論理的な機能単位を形成するように構築された型やリソースの集合です。アセンブリは、配置、バージョン制御、再利用、アクティベーションのスコープ、およびセキュリティのアクセス許可の基本単位を形成します。また、型の実装を認識するために必要な情報を共通言語ランタイム (CLR) に提供します。

次の表は、一般的な配置タスクの一覧です。

目的	参照項目
アセンブリを作成する。	アセンブリの作成
アセンブリに厳密な名前で署名する。	方法 : 厳密な名前でアセンブリに署名する
アセンブリに署名するが、秘密キーの割り当ては後で行う。	方法 : アセンブリに遅延署名する (Visual Studio)
厳密な名前の作成で使用する公開キーと秘密キーのペアを作成する。	方法 : 公開キーと秘密キーのキー ペアを作成する
ログ ファイルのバインディング情報を表示する。	アセンブリ バインディング ログ ビューア (Fuslogvw.exe)
Windows フォーム コントロールを配置する。	方法 : シンプルな Windows フォーム コントロールを開発する
COM からアクセスできるようにアプリケーションを配置する。	COM アクセスに対するアプリケーションの配置
XML Web サービスを配置する。	XML Web サービスの配置
.NET Framework アプリケーションを配置する。	.NET Framework アプリケーションの配置

バージョン管理

バージョン管理とは、アセンブリの固有のバージョンおよび依存アセンブリのバージョンをアセンブリのマニフェストに記録することで、厳密な名前を持つアセンブリに対してのみ行われます。厳密な名前が付けられたアセンブリは、アセンブリの ID と、公開キーとデジタル署名で構成される ID を持ちます。

既定では、CLR は、アプリケーションがビルドされたアセンブリとまったく同一のバージョンのものを見つけてバインドしようとします。ただしこの動作

は、構成ファイルの設定によりオーバーライドできます。

詳細については、「共通言語ランタイムのアセンブリ」、「アセンブリのバージョン管理」、および「厳密な名前付きアセンブリ」を参照してください。

ClickOnce の配置

ClickOnce 配置では、Windows アプリケーションを Web サーバーまたはネットワーク ファイル共有に発行して、簡単にインストールできるようになります。作成した Windows アプリケーションをネットワーク サーバーに発行する場合は、Visual Studio は、ClickOnce に必要な XML マニフェスト ファイルを自動的に生成し、指定のサーバーにアプリケーションを発行します。

ClickOnce アプリケーションは、単体で使用でき、ユーザー単位でインストールできます。つまり、管理権限は必要ありません。ClickOnce テクノロジを使用して配置されるアプリケーションは、セキュリティ ゾーンに基づいた限定されたアクセス許可セットを持ちます。詳細については、「ClickOnce の配置とセキュリティ」を参照してください。

ClickOnce アプリケーションは自動で更新させることができ、更新を配置するタイミングと方法を制御できます。

次の表は、ClickOnce 配置に関連するタスクの一覧です。

目的	参照項目
アプリケーションで使用する ClickOnce 配置ストラテジ (Web やネットワーク共有からのインストール、CD からのインストール、または Web やネットワーク共有からの起動) を選択する。	ClickOnce 配置ストラテジの選択
アプリケーションで使用する ClickOnce 更新ストラテジを選択する。	ClickOnce の更新方法の選択
ClickOnce アプリケーションを配置する。	方法 : ClickOnce アプリケーションを発行する
データの読み取りおよび書き込みで、ClickOnce データ ディレクトリ、分離ストレージ、またはその他のローカル ファイルを使用するかどうかを選択する。	ClickOnce アプリケーションにおけるローカル データおよびリモート データへのアクセス
アプリケーションを正常に配置するための必要条件を確認する。	配置の必要条件 (Visual Studio)
配置を更新するための別の場所を指定する。	方法 : 配置の更新用に別の場所を指定する
クライアント コンピュータで ClickOnce アプリケーションのアクセス許可のレベルを昇格させる。	信頼されたアプリケーションの配置の概要

ClickOnce でアプリケーションを配置するときに、HTTP の圧縮の問題、マニフェストの解析の問題、サーバーの構成の問題、およびバージョン管理の問題によって困難が生じることがあります。詳細については、「ClickOnce の配置のトラブルシューティング」を参照してください。

Windows インストーラ配置

Microsoft Windows インストーラ テクノロジでは、インストールのプロセスを完全に制御できるインストーラを作成します。

次の表は、Windows インストーラ配置に関連するタスクの一覧です。

目的	参照項目
CD-ROM などのメディアでファイルまたはアプリケーションを配布する。	配布可能なメディアによる配置
新しいセットアップ プロジェクトを作成するか、または既存の配置プロジェクトをソリューションに追加する。	方法 : セットアップ プロジェクトを作成または登録する
新しい .cab ファイルを Web ダウンロード用に作成するか、または既存の .cab ファイルをソリューションに追加する。	方法 : Cab プロジェクトを作成または登録する

コンポーネントのマージ モジュールを作成する。	方法 : 配置プロジェクトにマージ モジュールを登録する
Windows ベースのアプリケーションを他のコンピュータに配置する。	チュートリアル : Windows ベースのアプリケーションの配置
Web サイトを Web サーバーに配置する。	ASP.NET Web サイトの配置 (Visual Studio)
カスタム動作を作成して、インストール後にユーザーを Web サイトに送る。	チュートリアル : カスタム動作の作成
カスタム動作を使用してインストール中に動的なプロパティにデータを渡す。	チュートリアル : カスタム動作を使用した、インストール時のメッセージの表示
カスタム動作を使用して、インストール中にアセンブリをプリコンパイルする。	チュートリアル : カスタム動作を使用した、インストール時のアセンブリのプリコンパイル
カスタム動作を使用して、インストール中にデータベースを作成する。	チュートリアル : カスタム動作を使用して、インストール時にデータベースを作成する
別の XML Web サービスを対象とするためにリダイレクトできる Web アプリケーションを作成する。	チュートリアル : インストール時にアプリケーションを別の XML Web サービスにリダイレクトする
ファイルおよびアプリケーションを Web サーバーに配置する。	Web Setup プロジェクトの配置

参照

概念

配置の代替手段

その他の技術情報

Windows インストーラでの配置に関するチュートリアル

Visual Basic でのコンポーネントの作成および使用

コンポーネントとは、System.ComponentModel.IComponent インターフェイスを実装するクラス、または **IComponent** を実装するクラスから直接または間接的に派生されたクラスです。.NET Framework のコンポーネントは、再利用可能なオブジェクトで、他のオブジェクトとやり取りでき、外部リソースの制御やデザイン時サポートが備わっています。

コンポーネントの重要な特徴の 1 つが、コンポーネントはデザイン可能だということです。つまり、コンポーネントであるクラスは Visual Studio 統合開発環境で使用できます。コンポーネントは、ツールボックスへの追加、フォームへのドラッグ アンド ドロップ、およびデザイン サーフェイスでの操作が可能です。コンポーネントのデザイン時サポートは .NET Framework に組み込まれているため、コンポーネント開発者は追加の作業を行わずに基本のデザイン時機能を利用できます。

デザイン可能という点では、コントロール もコンポーネントに似ています。しかし、コントロールはユーザー インターフェイスを持つのに対し、コンポーネントは持ちません。コントロールは、Control または Control のいずれかの基本コントロール クラスを派生する必要があります。

コンポーネントの作成が必要な状況

デザイン サーフェイス (Windows フォーム デザイナ、Web フォーム デザイナなど) で使用するクラスがユーザー インターフェイスを持たない場合は、そのクラスはコンポーネントにする必要があり、**IComponent** を実装するか、または **IComponent** を直接または間接的に実装するクラスから派生させます。

Component クラスおよび MarshalByValueComponent クラスは **IComponent** インターフェイスの基本実装です。両クラスの主要な違いは、**Component** クラスは参照渡しでマーシャリングされるのに対し、**IComponent** は値渡しでマーシャリングされるということです。実装のためのガイドラインを次に示します。

- コンポーネントを参照渡しでマーシャリングする必要がある場合には、**Component** から派生します。
- コンポーネントを値渡しでマーシャリングする必要がある場合には、**MarshalByValueComponent** から派生します。
- 単一継承のためにいずれかの基本実装からコンポーネントを派生できない場合には、**IComponent** を実装します。

デザイン時サポートの詳細については、「コンポーネントのデザイン時属性」および「デザイン時サポートの拡張」を参照してください。

コンポーネントのクラス

System.ComponentModel 名前空間には、コンポーネントやコントロールの実行時動作およびデザイン時動作を実装するためのクラスが用意されています。この名前空間には、属性コンバータと型コンバータの実装、データ ソースとの連結、およびコンポーネントのライセンス処理のための基本クラスとインターフェイスが含まれています。

中心となるコンポーネント クラスを以下に示します。

- **Component**。**IComponent** インターフェイスの基本実装です。このクラスによりアプリケーション間でオブジェクトを共有できるようになります。
- **MarshalByValueComponent**。**IComponent** インターフェイスの基本実装です。
- Container。IContainer インターフェイスの基本実装です。このクラスは、いくつかのコンポーネントをカプセル化します (コンポーネントがない場合もあります)。

コンポーネントのライセンス処理に使用する主なクラスを以下に示します。

- License。すべてのライセンスの抽象基本クラスです。ライセンスは、コンポーネントの特定のインスタンスに対して付与されます。
- LicenseManager。コンポーネントにライセンスを追加し、LicenseProvider を管理するためのプロパティおよびメソッドが用意されています。
- **LicenseProvider**。ライセンス プロバイダを実装するための抽象基本クラスです。
- LicenseProviderAttribute。クラスに対して使用する **LicenseProvider** クラスを指定します。

コンポーネントの説明や永続化に一般的に使用するクラスを以下に示します。

- TypeDescriptor。属性、プロパティ、イベントなど、コンポーネントの特性についての情報を提供します。
- EventDescriptor。イベントについての情報を提供します。
- PropertyDescriptor。プロパティについての情報を提供します。

関連項目

クラス、コンポーネント、コントロール

コンポーネントとコントロールの定義を示し、それらとクラスの違いについて説明します。

コンポーネントの作成

コンポーネントの使用を始めるためのロードマップです。

コンポーネント作成のチュートリアル

コンポーネント プログラミングの詳細な手順について説明するトピックにリンクします。

コンポーネントのクラス

クラスがコンポーネントになるために必要な要素、コンポーネントの機能を公開する方法、コンポーネントへのアクセスを制御する方法、およびコンポーネントのインスタンスをどのように作成するかを制御する方法について説明します。

コントロールとコンポーネントの作成時のトラブルシューティング

一般的な問題に対処する方法について説明します。

参照

処理手順

方法 : Windows フォームでデザイン時サポートにアクセスする
方法 : デザイン モードでコントロールの外観と動作を拡張する
方法 : デザイン モードでコントロールのカスタム初期化を行う

Visual Basic への理解を深める

My にはさまざまな機能がありますが、場合によっては、**My** 以外にも .NET Framework が持つ機能をもっと深く掘り下げようとすることもあります。以下のページには、フォーム、ネットワーク操作、Web サービスなど、.NET Framework の各分野についての概要や、関連ドキュメントへのリンクが示されています。

このセクションの内容

Visual Basic での .NET Framework の認証と承認

.NET Framework を使用してユーザー認証や承認を行う方法を説明します。

Visual Basic における .NET Framework のコード アクセス許可とセキュリティ

.NET Framework でのコードのアクセス許可とセキュリティのしくみについて説明します。

Visual Basic での .NET Framework のコレクション

.NET Framework に用意されている、データの格納や取得の専用クラスについて説明します。

Visual Basic での .NET Framework のファイル I/O

.NET Framework を使用してファイル I/O を行う方法を説明します。

Visual Basic での .NET Framework のグラフィックス

.NET Framework を使用してグラフィックスを操作する方法を説明します。

Visual Basic での .NET Framework のログ記録とトレース

.NET Framework を使用してログやトレースを作成および操作する方法を説明します。

Visual Basic でのプロジェクトのカスタマイズと My の拡張

アプリケーション モデルに対する独自の拡張機能を指定し、プロジェクト テンプレートをカスタマイズして、追加的な **My** オブジェクトを提供する方法を説明します。

Visual Basic による .NET Framework でのネットワーク操作

.NET Framework を使用してネットワーク操作を実行する方法を説明します。

Visual Basic による .NET Framework でのポート操作

.NET Framework を使用してポート操作を実行する方法を説明します。

Visual Basic での .NET Framework のシリアル化

.NET Framework を使用してシリアル化を行う方法を説明します。

Visual Basic によるスマート デバイス向けの開発

スマート デバイス向けのプログラミングの基本について、概要を説明します。

Visual Basic での高度なマルチスレッド処理

スレッド処理モデルの概要を説明します。

Visual Basic での Windows フォーム アプリケーションの概念

Windows アプリケーションの概要を説明します。

Visual Studio アプリケーションへの印刷可能なレポートの追加

使用できるレポートの書き込みオプションについて説明します。

Visual Basic での .NET Framework の認証と承認

.NET Framework には、ユーザーを承認および認証するための、拡張性のあるフレームワークが用意されています。このセクションのトピックでは、これらの考え方の概要と、一般的に使用するクラスの一覧を示します。

このセクションの内容

.NET Framework のユーザー操作の基礎

.NET Framework でアプリケーションの承認を扱う方法と、ロール ベース セキュリティの使用方法を説明します。

チュートリアル : カスタムの認証および承認の実装

カスタムの認証および承認を実装する方法、およびアプリケーション スレッドの既定の ID をオーバーライドする方法を説明します。

.NET Framework でユーザー操作に使用するクラス

.NET Framework で使用する、ユーザー認証システムの基盤となるクラスの一覧を示します。

参照

関連項目

My.User オブジェクト

System.Security.Principal

概念

ユーザー データへのアクセス

その他の技術情報

Visual Basic への理解を深める

.NET Framework のユーザー操作の基礎

.NET Framework では、System.Security.Principal 名前空間にロール ベース セキュリティが実装されています。アプリケーションでは、これを使用して、ユーザーの承認や認証を実行できます。このトピックでは、.NET Framework でアプリケーションの承認を扱う方法や、ユーザーを表す IIdentity オブジェクトおよび IPrincipal オブジェクトを作成する方法について解説します。

IIdentity は、認証されたユーザーをカプセル化します。**IPrincipal** は、ユーザーの ID と、そのユーザーが持つロールの組み合わせです。**System.Security.Principal** 名前空間で定義済みの ID クラスおよびプリンシパル クラスを使用することもできるし、これらのインターフェイスを実装するクラスを作成してカスタム認証を追加することもできます。

これらのインターフェイスを使用するには、インターフェイス名を完全修飾するか、または、関係するソース コード ファイルの先頭に **Imports** ステートメントを記述して、適切な名前空間をインポートする必要があります。詳細については、「Imports ステートメント」を参照してください。

.NET Framework アプリケーションでのロール ベース セキュリティの使用

各アプリケーション スレッドには、コードを実行中のユーザーのセキュリティ コンテキストを表すプリンシパル オブジェクトが関連付けられます (CurrentPrincipal プロパティでアクセス可能)。プリンシパル オブジェクトは、ユーザーのユーザー セキュリティ コンテキスト (つまり ID) オブジェクトをカプセル化します。2 つのオブジェクトがあるおかげで、認証 (ID オブジェクトによる) と承認 (プリンシパルによる) を分離できます。

ID オブジェクトは **IIdentity** インターフェイスを実装する必要があります。ID オブジェクトは、特定のユーザーを表し、**IIdentity** インターフェイスに必要な Name、IsAuthenticated、および AuthenticationType の各プロパティを公開します。通常、ID オブジェクトには、ユーザー認証を実行するプライベート メンバが追加されています。

プリンシパル オブジェクトは **IPrincipal** インターフェイスを実装する必要があります。プリンシパル オブジェクトは、**IPrincipal** インターフェイスで必要なメンバを公開することにより、ユーザーのセキュリティ コンテキストをカプセル化します。そのメンバの 1 つは、承認を実行する IsInRole メソッドです。もう 1 つは、ユーザーの ID オブジェクトへのアクセスを提供する Identity プロパティです。

ID の操作

.NET Framework には、**IIdentity** インターフェイスを実装した、次の 4 つのクラスが用意されています。

- WindowsIdentity
- GenericIdentity
- PassportIdentity
- FormsIdentity

各クラスでは、それぞれ異なる種類のユーザー ID を扱うことができます。Windows 認証を使用するアプリケーションの現在の **WindowsIdentity** オブジェクトにアクセスするには、**WindowsIdentity** クラスの GetCurrent 静的メソッドを使用します。また、My.User.InitializeWithWindowsUser メソッドを呼び出すと、現在のスレッドのプリンシパルを設定できます。

また、独自のカスタム クラスに **IIdentity** インターフェイスを実装することにより、カスタムの ID クラスを作成できます。カスタム ID を作成する方法の詳細については、「チュートリアル : カスタムの認証および承認の実装」を参照してください。

プリンシパルの操作

.NET Framework には、ユーザー ロールと ID をリンクさせるための **IPrincipal** インターフェイスが用意されています。アプリケーションで承認を実行する場合は、**IPrincipal** を実装したオブジェクトを使用する必要があります。たとえば、**WindowsIdentity** クラスや **GenericIdentity** クラスには、**IPrincipal** の実装が組み込まれています。または、**IPrincipal** を使用して、独自のカスタム プリンシパル クラスを作成することもできます。

IPrincipal オブジェクトをスレッドの **CurrentPrincipal** プロパティまたは My.User.CurrentPrincipal プロパティに割り当てると、現在のスレッドをこのオブジェクトにリンクできます。そして、ユーザーが特定のロールのメンバかどうか調べることにより、承認チェックを実行できます。それには、プリンシパルの **IsInRole** メソッドを使用します。

ASP.NET アプリケーションの場合は、**IPrincipal** オブジェクトの処理が、他の .NET Framework アプリケーションの場合と異なります。ASP.NET では、セッションの外観を、状態のない HTTP プロトコル上で作成します。このセッションの一部として、ユーザーの要求を実行するすべてのコードについて、ユーザーを表す **IPrincipal** オブジェクトを、HttpContext オブジェクトの User プロパティから利用できます。共通言語ランタイムは、Global.asax ファイルの OnAuthenticate イベントの後で、**CurrentPrincipal** を **User** の値で自動的に更新します。ASP.NET アプリケーションでは、**User** プロパティを使用して承認チェックを実行することがよくあります。

メモ :

User を手動で変更すると、同じ HTTP コンテキスト内で実行されているすべてのスレッドについて、**CurrentPrincipal** プロパティが自動的に更新されます。一方、**CurrentPrincipal** を変更しても、**User** プロパティには影響しません。影響が及ぶのは、当該スレッドのリクエストの残りの部分に対してのみです。

ASP.NET アプリケーションでは、My.User.CurrentPrincipal プロパティにより **User** プロパティが更新されます。

独自の **IPrincipal** 型の作成の詳細については、「チュートリアル : カスタムの認証および承認の実装」を参照してください。

IIdentity オブジェクトおよび IPrincipal オブジェクトを扱うためのアクセス許可の付与

IIdentity オブジェクトを扱うためのアクセス許可を与えるときには注意が必要です。これらのオブジェクトは、機密性の高いユーザー関連情報を利用できるようにするものだからです。アプリケーションの現在の **IPrincipal** オブジェクトが変更されないように保護する必要があります。アプリケーションの承認機能は、現在のプリンシパルに基づいて決まるからです。

.NET Framework では、この保護を実現するために、これらの操作にはコード アクセス セキュリティのアクセス許可が必要とされています。これらのオブジェクトを操作する必要のあるアプリケーションに対し、コード アクセス セキュリティ ポリシー ツール (Caspol.exe) を使用して System.Security.Permissions.SecurityPermissionAttribute.ControlPrincipal アクセス許可を与えます。

既定では、このアクセス許可は、ローカルにインストールされたアプリケーションすべてに対して与えられています。これらのアプリケーションは FullTrust アクセス許可セットの下で実行されるからです。

以下のメソッドを実行するには、**ControlPrincipal** アクセス許可が必要です。

- System.AppDomain.SetPrincipalPolicy(System.Security.Principal.PrincipalPolicy)
- **System.Security.Principal.WindowsIdentity.GetCurrent**
- System.Security.Principal.WindowsIdentity.Impersonate
- **System.Threading.Thread.CurrentPrincipal**

参照

処理手順

チュートリアル : カスタムの認証および承認の実装

関連項目

My.User オブジェクト

その他の技術情報

Visual Basic での .NET Framework の認証と承認

チュートリアル：カスタムの認証および承認の実装

このチュートリアルでは、IIdentity および IPrincipal から派生するクラスを使用してカスタムの認証および承認を実装する方法を説明します。また、このチュートリアルでは、**IPrincipal** から派生するクラスのインスタンスを **My.User.CurrentPrincipal** に設定することによって、アプリケーション スレッドの既定の ID である Windows の ID をオーバーライドする方法も説明します。新しいユーザー情報は **My.User** オブジェクトを通じてすぐに利用できます。このオブジェクトは、スレッドの現在のユーザー ID についての情報を返します。

ビジネス アプリケーションでは、ユーザーによって提示される資格情報に基づいて、データやリソースへのアクセスを与えることがよくあります。このようなアプリケーションは、通常、ユーザーのロールを調べ、そのロールに基づいたリソースへのアクセスを許可します。共通言語ランタイムには、Windows アカウントまたはカスタム ID に基づくロール ベース承認のサポートが用意されています。詳細については、「ロール ベース セキュリティ」を参照してください。

概要

まずは、メイン フォームとログイン フォームを持つプロジェクトを設定し、カスタムの認証を使用するように構成します。

サンプル アプリケーションを作成するには

1. 新しい Visual Basic Windows アプリケーション プロジェクトを作成します。詳細については、「方法 : Windows アプリケーション プロジェクトを作成する」を参照してください。

   メイン フォームの既定の名前は `Form1` です。

2. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

3. [ログイン フォーム] テンプレートを選択し、[追加] をクリックします。

   ログイン フォームの既定の名前は `LoginForm1` です。

4. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

5. [クラス] テンプレートを選択し、名前を `SampleIIdentity` に変更して、[追加] をクリックします。

6. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

7. [クラス] テンプレートを選択し、名前を `SampleIPrincipal` に変更して、[追加] をクリックします。

8. [プロジェクト] メニューの [<ApplicationName> のプロパティ] をクリックします。

9. プロジェクト デザイナで、[アプリケーション] タブをクリックします。

10. [認証モード] ボックスを [アプリケーション定義] に変更します。

メイン フォームを構成するには

1. フォーム デザイナで `Form1` に切り替えます。

2. ツールボックスから [Button] を `Form1` に追加します。

   ボタンの既定の名前は `Button1` です。

3. ボタンのテキストを **Authenticate** に変更します。

4. ツールボックスから [Label] を `Form1` に追加します。

   ラベルの既定の名前は `Label1` です。

5. ラベルのテキストを空の文字列に変更します。

6. ツールボックスから [Label] を Form1 に追加します。

   ラベルの既定の名前は `Label2` です。

7. ラベルのテキストを空の文字列に変更します。

8. Button1 をダブルクリックし、Click イベントのイベント ハンドラを作成して、コード エディタを開きます。

9. `Button1_Click` メソッドに次のコードを追加します。

VB

```
My.Forms.LoginForm1.ShowDialog()
' Check if the user was authenticated.
If My.User.IsAuthenticated Then
    Me.Label1.Text = "Authenticated " & My.User.Name
Else
    Me.Label1.Text = "User not authenticated"
End If

If My.User.IsInRole(ApplicationServices.BuiltInRole.Administrator) Then
    Me.Label2.Text = "User is an Administrator"
Else
    Me.Label2.Text = "User is not an Administrator"
End If
```

このアプリケーションは実行できますが、認証コードがないため、ユーザーを認証しません。認証コードの追加については、次のセクションで説明します。

ID の作成

.NET Framework では、認証および承認の基盤として **IIdentity** インターフェイスおよび **IPrincipal** インターフェイスを使用します。アプリケーションでは、これらのインターフェイスを実装することにより、カスタムのユーザー認証を使用できます。以下の手順はその例です。

IIdentity を実装するクラスを作成するには

1. ソリューション エクスプローラで SampleIIdentity.vb ファイルを選択します。

   このクラスはユーザーの ID をカプセル化します。

2. `Public Class SampleIIdentity` の次の行に、次のコードを追加して、**IIdentity** を継承します。

   VB

   ```
   Implements System.Security.Principal.IIdentity
   ```

   このコードを追加して Enter キーを押すと、実装が必要なスタブ プロパティがコード エディタによって作成されます。

3. ユーザー名、およびユーザーが認証されているかどうかを示す値を格納するためのプライベート フィールドを追加します。

   VB

   ```
   Private nameValue As String
   Private authenticatedValue As Boolean
   Private roleValue As ApplicationServices.BuiltInRole
   ```

4. AuthenticationType プロパティに次のコードを入力します。

   AuthenticationType プロパティでは、現在の認証機構を示す文字列を返す必要があります。

   この例では、明示的に指定された認証を使用するため、この文字列は "Custom Authentication" です。ユーザー認証データが SQL Server データベースに格納される場合なら、この値は "SqlDatabase" となります。

   VB

   ```
   Return "Custom Authentication"
   ```

5. IsAuthenticated プロパティに次のコードを入力します。

   VB

```
Return authenticatedValue
```

IsAuthenticated プロパティでは、ユーザーが認証されているかどうかを示す値を返す必要があります。

6. Name プロパティでは、この ID と関連付けられたユーザーの名前を返す必要があります。

   Name プロパティに次のコードを入力します。

   VB

```
Return nameValue
```

7. ユーザーのロールを返すプロパティを作成します。

   VB

```
Public ReadOnly Property Role() As ApplicationServices.BuiltInRole
    Get
        Return roleValue
    End Get
End Property
```

8. **Sub New** メソッドを作成します。この中では、ユーザーを認証してから、名前とパスワードに基づいてユーザーの名前とロールを設定することにより、クラスを初期化します。

   このメソッドでは、ユーザー名とパスワードの組み合わせが有効かどうかを判断するために、`IsValidNameAndPassword` という名前のメソッドを呼び出します。

   VB

```
Public Sub New(ByVal name As String, ByVal password As String)
    ' The name is not case sensitive, but the password is.
    If IsValidNameAndPassword(name, password) Then
        nameValue = name
        authenticatedValue = True
        roleValue = ApplicationServices.BuiltInRole.Administrator
    Else
        nameValue = ""
        authenticatedValue = False
        roleValue = ApplicationServices.BuiltInRole.Guest
    End If
End Sub
```

9. ユーザー名とパスワードの組み合わせが有効かどうかを判断する、`IsValidNameAndPassword` という名前のメソッドを作成します。

セキュリティに関するメモ：
認証アルゴリズムでは、パスワードを安全に扱う必要があります。たとえば、パスワードをクラス フィールドとして格納しないようにします。

   ユーザー パスワードは、システム内に格納しないようにします。その情報がリークしたら、セキュリティは無効になるからです。格納するとしたら、各ユーザーのパスワードのハッシュを格納します (ハッシュ関数を使用すると、データがかく乱され、出力データから入力データを導き出すことができないようになります)。パスワードのハッシュからパスワード自体を直接的に確認することはできません。

   しかし、悪意のあるユーザーは、可能性のあるすべてのパスワードのハッシュの辞書を、時間をかけて生成したうえで、目的のハッシュに対応するパスワードを探すという手を使う可能性があります。この種類の攻撃に対する保護のためには、ハッシュを求める前のパスワードに salt を追加して、salt 処理されたハッシュを生成する必要があります。salt とは、各パスワードに対して一意の、追加的なデータです。これにより、ハッシュ辞書を事前計算するのは不可能となります。

悪意のあるユーザーからパスワードを保護するには、パスワードの salt 処理されたハッシュのみを格納する必要があり、できれば、セキュリティで保護されたコンピュータに格納します。悪意のあるユーザーが、salt 処理されたハッシュからパスワードを復元するのは、非常に困難です。この例では、`GetHashedPassword` メソッドおよび `GetSalt` メソッドを使用して、ユーザーのハッシュ化パスワードと salt を読み込みます。

VB

```
Private Function IsValidNameAndPassword( _
    ByVal username As String, _
    ByVal password As String) _
    As Boolean

    ' Look up the stored hashed password and salt for the username.
    Dim storedHashedPW As String = GetHashedPassword(username)
    Dim salt As String = GetSalt(username)

    'Create the salted hash.
    Dim rawSalted As String = salt & Trim(password)
    Dim saltedPwBytes() As Byte = _
        System.Text.Encoding.Unicode.GetBytes(rawSalted)
    Dim sha1 As New _
        System.Security.Cryptography.SHA1CryptoServiceProvider
    Dim hashedPwBytes() As Byte = sha1.ComputeHash(saltedPwBytes)
    Dim hashedPw As String = Convert.ToBase64String(hashedPwBytes)

    ' Compare the hashed password with the stored password.
    Return hashedPw = storedHashedPW
End Function
```

10. `GetHashedPassword` および `GetSalt` という名前の関数を作成します。指定されたユーザーに対応する、ハッシュ化パスワードと salt を返す関数です。

セキュリティに関するメモ：

ハッシュ化パスワードと salt をクライアント アプリケーションにハードコーディングすることは、次の 2 つの理由により避ける必要があります。1 つは、悪意のあるユーザーからコードにアクセスされて、ハッシュ衝突を見つけ出される可能性があるからです。もう 1 つは、ユーザーのパスワードの変更や取り消しができないからです。アプリケーションでは、所定のユーザーに対応する、ハッシュ化パスワードと salt を、管理者によって管理されている、セキュリティで保護された場所から取得する必要があります。

この例では、単純化のために、ハッシュ化パスワードと salt をハードコーディングしていますが、実際に使用するコードでは、もっと安全な方法を使用する必要があります。たとえば、SQL Server データベースにユーザー情報を格納して、ストアド プロシージャでアクセスするという方法があります。詳細については、「方法 : データベース内のデータに接続する」を参照してください。

メモ：

ハードコーディングしている、このハッシュ化パスワードに対応するパスワードについては、「アプリケーションのテスト」のセクションを参照してください。

VB

```
Private Function GetHashedPassword(ByVal username As String) As String
    ' Code that gets the user's hashed password goes here.
    ' This example uses a hard-coded hashed passcode.
    ' In general, the hashed passcode should be stored
    ' outside of the application.
    If Trim(username).ToLower = "testuser" Then
        Return "ZFFzgfsGjgtmExzWBRmZI5S4w6o="
    Else
        Return ""
```

```
        End If
    End Function

    Private Function GetSalt(ByVal username As String) As String
        ' Code that gets the user's salt goes here.
        ' This example uses a hard-coded salt.
        ' In general, the salt should be stored
        ' outside of the application.
        If Trim(username).ToLower = "testuser" Then
            Return "Should be a different random value for each user"
        Else
            Return ""
        End If
    End Function
```

この時点で、SampleIIdentity.vb ファイルの内容は次のコードのようになります。

VB

```
Public Class SampleIIdentity
    Implements System.Security.Principal.IIdentity

    Private nameValue As String
    Private authenticatedValue As Boolean
    Private roleValue As ApplicationServices.BuiltInRole

    Public ReadOnly Property AuthenticationType() As String Implements System.Security.Prin
cipal.IIdentity.AuthenticationType
        Get
            Return "Custom Authentication"
        End Get
    End Property

    Public ReadOnly Property IsAuthenticated() As Boolean Implements System.Security.Princi
pal.IIdentity.IsAuthenticated
        Get
            Return authenticatedValue
        End Get
    End Property

    Public ReadOnly Property Name() As String Implements System.Security.Principal.IIdentit
y.Name
        Get
            Return nameValue
        End Get
    End Property

    Public ReadOnly Property Role() As ApplicationServices.BuiltInRole
        Get
            Return roleValue
        End Get
    End Property

    Public Sub New(ByVal name As String, ByVal password As String)
        ' The name is not case sensitive, but the password is.
        If IsValidNameAndPassword(name, password) Then
            nameValue = name
            authenticatedValue = True
            roleValue = ApplicationServices.BuiltInRole.Administrator
        Else
            nameValue = ""
            authenticatedValue = False
            roleValue = ApplicationServices.BuiltInRole.Guest
        End If
    End Sub
```

```
    Private Function IsValidNameAndPassword( _
        ByVal username As String, _
        ByVal password As String) _
        As Boolean

        ' Look up the stored hashed password and salt for the username.
        Dim storedHashedPW As String = GetHashedPassword(username)
        Dim salt As String = GetSalt(username)

        'Create the salted hash.
        Dim rawSalted As String = salt & Trim(password)
        Dim saltedPwBytes() As Byte = _
            System.Text.Encoding.Unicode.GetBytes(rawSalted)
        Dim sha1 As New _
            System.Security.Cryptography.SHA1CryptoServiceProvider
        Dim hashedPwBytes() As Byte = sha1.ComputeHash(saltedPwBytes)
        Dim hashedPw As String = Convert.ToBase64String(hashedPwBytes)

        ' Compare the hashed password with the stored password.
        Return hashedPw = storedHashedPW
    End Function

    Private Function GetHashedPassword(ByVal username As String) As String
        ' Code that gets the user's hashed password goes here.
        ' This example uses a hard-coded hashed passcode.
        ' In general, the hashed passcode should be stored
        ' outside of the application.
        If Trim(username).ToLower = "testuser" Then
            Return "ZFFzgfsGjgtmExzWBRmZI5S4w6o="
        Else
            Return ""
        End If
    End Function

    Private Function GetSalt(ByVal username As String) As String
        ' Code that gets the user's salt goes here.
        ' This example uses a hard-coded salt.
        ' In general, the salt should be stored
        ' outside of the application.
        If Trim(username).ToLower = "testuser" Then
            Return "Should be a different random value for each user"
        Else
            Return ""
        End If
    End Function

End Class
```

プリンシパルの作成

次に、**IPrincipal** を派生するクラスを実装し、`SampleIIdentity` クラスのインスタンスを返させる必要があります。

IPrincipal を実装するクラスを作成するには

1. ソリューション エクスプローラで SampleIPrincipal.vb ファイルを選択します。

   このクラスはユーザーの ID をカプセル化します。**My.User** オブジェクトを使用すると、このプリンシパルを現在のスレッドに割り当てて、ユーザーの ID にアクセスできます。

2. `Public Class SampleIPrincipal` の次の行に、次のコードを追加して、**IPrincipal** を継承します。

   VB

```
    Implements System.Security.Principal.IPrincipal
```

このコードを追加して Enter キーを押すと、実装が必要なスタブ プロパティおよびメソッドがコード エディタによって作成されます。

3. このプリンシパルに関連付けられた ID を格納するためのプライベート フィールドを追加します。

   VB

```
Private identityValue As SampleIIdentity
```

4. Identity プロパティに次のコードを入力します。

   VB

```
Return identityValue
```

   Identity プロパティでは、現在のプリンシパルのユーザー ID を返す必要があります。

5. IsInRole メソッドに次のコードを入力します。

   IsInRole メソッドでは、指定したロールに現在のプリンシパルが属するかどうかを判断します。

   VB

```
Return role = identityValue.Role.ToString
```

6. 所定のユーザー名およびパスワードに対応する `SampleIIdentity` の新しいインスタンスでクラスを初期化する **Sub New** メソッドを作成します。

   VB

```
Public Sub New(ByVal name As String, ByVal password As String)
    identityValue = New SampleIIdentity(name, password)
End Sub
```

   このコードでは、`SampleIPrincipal` クラスに対するユーザー ID を設定します。

この時点で、SampleIPrincipal.vb ファイルの内容は次のコードのようになります。

VB

```
Public Class SampleIPrincipal
    Implements System.Security.Principal.IPrincipal

    Private identityValue As SampleIIdentity

    Public ReadOnly Property Identity() As System.Security.Principal.IIdentity Implements S
ystem.Security.Principal.IPrincipal.Identity
        Get
            Return identityValue
        End Get
    End Property

    Public Function IsInRole(ByVal role As String) As Boolean Implements System.Security.Pr
incipal.IPrincipal.IsInRole
        Return role = identityValue.Role.ToString
    End Function

    Public Sub New(ByVal name As String, ByVal password As String)
        identityValue = New SampleIIdentity(name, password)
    End Sub

End Class
```

ログイン フォームの接続

アプリケーションでは、ログイン フォームを使用してユーザー名とパスワードを取得できます。この情報を使用して、`SampleIPrincipal` クラスのインスタンスを初期化でき、また **My.User** オブジェクトを使用して、現在のスレッドの ID をそのインスタンスに対して設定できます。

ログイン フォームを構成するには

1. デザイナで `LoginForm1` を選択します。
2. [OK] ボタンをダブルクリックして、**Click** イベントをコード エディタで開きます。
3. `OK_Click` メソッドのコードを次のコードで置き換えます。

   VB

```
Dim samplePrincipal As New SampleIPrincipal( _
    Me.UsernameTextBox.Text, Me.PasswordTextBox.Text)
Me.PasswordTextBox.Text = ""
If (Not samplePrincipal.Identity.IsAuthenticated) Then
    ' The user is still not validated.
    MsgBox("The username and password pair is incorrect")
Else
    ' Update the current principal.
    My.User.CurrentPrincipal = samplePrincipal
    Me.Close()
End If
```

アプリケーションのテスト

アプリケーションに認証コードを追加したので、アプリケーションを実行して、ユーザーの認証を試してみることができます。

アプリケーションをテストするには

1. アプリケーションを起動します。
2. [Authenticate] をクリックします。

   ログイン フォームが開きます。
3. [ユーザー名] ボックスに「**TestUser**」、[パスワード] ボックスに「**BadPassword**」と入力し、[OK] をクリックします。

   ユーザー名とパスワードのペアが正しくないことを通知するメッセージ ボックスが表示されます。
4. [OK] をクリックしてメッセージ ボックスを閉じます。
5. [キャンセル] をクリックしてログイン フォームを閉じます。

   メイン フォームのラベルには、[User not authenticated] および [User is not an Administrator] と表示されます。
6. [Authenticate] をクリックします。

   ログイン フォームが開きます。
7. [ユーザー名] ボックスに「**TestUser**」、[パスワード] ボックスに「**Password**」と入力し、[OK] をクリックします。パスワードの大文字と小文字は正しく入力してください。

   メイン フォームのラベルには、[Authenticated TestUser] および [User is an Administrator] と表示されます。

参照

処理手順

方法 : データベース内のデータに接続する

関連項目

My.User オブジェクト

IIdentity

IPrincipal

概念

ユーザー データへのアクセス

その他の技術情報

Visual Basic での .NET Framework の認証と承認

Visual Basic での .NET Framework の認証と承認

.NET Framework でユーザー操作に使用するクラス

IIdentity インターフェイスおよび IPrincipal インターフェイスは、.NET Framework のユーザー認証システムの基盤となるインターフェイスです。.NET Framework に用意されている、これらのインターフェイスの実装を使用することもできるし、そうしたクラスを自分で実装することもできます。

ユーザー操作に使用する基本的なクラス

ユーザー操作をサポートするクラスの大半は System.Security.Principal 名前空間に属しています。より詳細なリストについては、.NET Framework のドキュメントを参照してください。

以下のクラスは、ID オブジェクトの基本機能を定義する **IIdentity** インターフェイスを実装しています。

クラス	説明
GenericIdentity	汎用ユーザーを表します。
WindowsIdentity	Windows ユーザーを表します。
IdentityReference	ID を表し、NTAccount クラスおよび SecurityIdentifier クラスの基本クラスです。
NTAccount	ユーザーまたはグループ アカウントを表します。
SecurityIdentifier	セキュリティ識別子 (SID) を表し、SID のマーシャリングおよび比較演算を実現します。

以下のクラスは、プリンシパル オブジェクトの基本機能を定義する **IPrincipal** インターフェイスを実装しています。

クラス	説明
GenericPrincipal	汎用プリンシパルを表します。
WindowsPrincipal	Windows ユーザーの Windows グループ メンバシップをコードでチェックできます。

この表は、.NET Framework のユーザー操作で使用する、特に重要な列挙値の一覧です。

列挙値	説明
PrincipalPolicy	アプリケーション ドメインに対してプリンシパルおよび ID オブジェクトをどのように作成するかを定めます。
WellKnownSidType	一般的に使用されるセキュリティ識別子 (SID) を定義します。
WindowsAccountType	使用する Windows アカウントの種類を定めます。
WindowsBuiltInRole	IsInRole で使用する一般的なロールを定めます。

参照

処理手順

チュートリアル : カスタムの認証および承認の実装

関連項目

My.User オブジェクト

その他の技術情報

Visual Basic での .NET Framework の認証と承認

Visual Basic における .NET Framework のコード アクセス許可とセキュリティ

.NET Framework では、コード アクセス セキュリティによって、コードからアクセスできる対象が保護されたリソースと保護された操作に制限されます。共通言語ランタイムに対応するすべてのアプリケーションは、そのランタイムのセキュリティ システムと対話する必要があります。アプリケーションは実行時にランタイムによって自動的に評価され、一連のアクセス許可を与えられます。アプリケーションが受け取るアクセス許可に応じて、そのアプリケーションが正常に実行されるか、またはセキュリティ例外が生成されます。

コードが受け取るアクセス許可は、最終的にはそれぞれのコンピュータのローカル セキュリティ設定によって制御されます。これらの設定はコンピュータごとに異なる可能性があるため、作成したコードが実行に必要なアクセス許可を必ず受け取るかどうかはわかりません。詳細については、「コード アクセス セキュリティの基礎」を参照してください。

コード アクセス許可

コード アクセス許可オブジェクトは、承認されていないユーザーがリソースと操作を使用することを防止するために使用されます。アクセス許可オブジェクトは、マネージ コードにセキュリティ制限を適用するための共通言語ランタイムの機構の基本的な部分です。

コード アクセス許可により、ユーザーは保護されたリソース (ファイルなど) にアクセスしたり、保護されたアクション (マネージ コードへのアクセスなど) を実行したりできます。コード アクセス許可はすべてコードから要求でき、アクセス許可が付与されるかどうかはランタイムが決定します。各コード アクセス許可は CodeAccessPermission クラスから派生します。したがって、アクセス許可は一般的なメソッドとして、**Assert**、**Demand**、**Deny**、**PermitOnly**、**IsSubSetOf**、**Intersect**、**Union** の各メソッドを持ちます。

.NET Framework に用意されているアクセス許可

次の表は、.NET Framework に用意されているコード アクセス許可を示しています。

アクセス許可クラス	実行できる操作
AspNetHostingPermission	ASP.NET によってホストされた環境のリソースへのアクセス。
DirectoryServicesPermission	**System.DirectoryServices** クラスへのアクセス。
DnsPermission	ネットワーク上のドメイン ネーム システム (DNS: Domain Name System) サーバーへのアクセス。
EnvironmentPermission	環境変数の読み取りと書き込み。
EventLogPermission	イベント ログ サービスからの読み取りと書き込み。
FileDialogPermission	[ファイルを開く] ダイアログ ボックスからのファイルの読み取りと書き込み。
FileIOPermission	ファイルまたはディレクトリからの読み取りと書き込み。
IsolatedStorageFilePermission	分離ストレージ内のファイルまたはディレクトリからの読み取りと書き込み。
MessageQueuePermission	管理されたメッセージ キュー (MSMQ とも呼ばれます) インターフェイスを通じてのメッセージ キューへのアクセス。
OdbcPermission	ODBC (Open Database Connectivity) データ ソースへのアクセス。
OleDbPermission	OLE DB を使用したデータベースへのアクセス。
OraclePermission	Oracle データベースへのアクセス。
PerformanceCounterPermission	パフォーマンス カウンタへのアクセス。
PrintingPermission	プリンタへのアクセス。

ReflectionPermission	実行時における型情報の確認。
RegistryPermission	レジストリ キーとレジストリ値の読み取り、書き込み、作成、および削除。
SecurityPermission	アクセス許可の実行とアサート、アンマネージ コードの呼び出し、検査のスキップ、およびその他のセキュリティ関連の権限。
ServiceControllerPermission	実行中または停止中のサービスへのアクセス。
SocketPermission	トランスポート アドレスでの接続の確立と承認。
SqlClientPermission	SQL データベースへのアクセス。
UIPermission	ユーザー インターフェイスの機能へのアクセス。
WebPermission	Web アドレスでの接続の確立と承認。

独自のアクセス許可の作成

.NET Framework は、.NET Framework が公開するリソースを中心に、特定のリソースと操作をまとめて保護できるようにするための一連のコード アクセス許可クラスを提供します。ほとんどの場合は、組み込みのコード アクセス許可で十分です。しかし、場合によっては、独自のコード アクセス許可を定義する必要があります。詳細については、「独自のコード アクセス許可の作成」を参照してください。

ID アクセス許可

ID アクセス許可は、アセンブリを識別する特性を表します。共通言語ランタイムは、アセンブリに関して取得した情報に基づいて、アセンブリにその読み込み時に ID アクセス許可を与えます。詳細については、「ID アクセス許可」を参照してください。

ロール ベース アクセス許可

ビジネス アプリケーションでは、ユーザーによって提示される資格情報に基づいて、データやリソースへのアクセスを与えることがよくあります。このようなアプリケーションは、通常、ユーザーのロールを調べ、そのロールに基づいたリソースへのアクセスを許可します。共通言語ランタイムには、Windows アカウントまたはカスタム ID に基づくロール ベース承認のサポートが用意されています。詳細については、「ロール ベース セキュリティ」を参照してください。

セキュリティ タスク

次の表は、アクセス許可とセキュリティの関連タスクの一覧です。

実行内容	参照項目
アンマネージ コードにアクセスするためのアクセス許可の要求	方法 : アンマネージ コードへのアクセス許可を要求する
オプションのアクセス許可の要求	方法 : RequestOptional フラグを使用してオプションのアクセス許可を要求する
名前付きアクセス許可セットに対するアクセス許可の要求	方法 : 名前付きアクセス許可セットに対するアクセス許可を要求する
XML でエンコードされたアクセス許可の要求	XML でエンコードされたアクセス許可の要求
アクセス許可の拒否	方法 : RequestRefuse フラグを使用することにより、アクセス許可を拒否する
強制セキュリティ チェックの実行	方法 : 強制セキュリティ チェックを実行する
宣言セキュリティ チェックの実行	宣言セキュリティ チェックの実行
セキュリティ チェックのオーバーライド	セキュリティ チェックのオーバーライド

部分的な信頼のコードとのライブラリの共有	部分的な信頼のコードとのライブラリの共有
AllowPartiallyTrustedCallersAttribute アセンブリ内の型に対する完全信頼の要求	AllowPartiallyTrustedCallersAttribute アセンブリ内の型に対する完全な信頼の要求
WindowsIdentity および WindowsPrincipal オブジェクトの作成	方法 : WindowsPrincipal プロジェクトを作成する
GenericPrincipal および GenericIdentity オブジェクトの作成	方法 : GenericPrincipal オブジェクトと GenericIdentity オブジェクトを作成する

.NET Framework のセキュリティ ポリシー モデル

.NET Framework のセキュリティ ポリシー モデルは 5 つの要素で構成されます。各要素を次に示します。

- エンタープライズ、マシン、ユーザー、アプリケーション ドメインの各セキュリティ ポリシー レベル
- エンタープライズ、マシン、およびユーザー ポリシー レベル内に階層構造として存在するコード グループ
- 各コード グループに関連付けられている名前付きアクセス許可セット
- コードの識別に関する情報を提供する証拠
- コードに関する証拠を共通言語ランタイムに提供するアプリケーション ドメイン ホスト

セキュリティ ポリシー レベル

.NET Framework には、アセンブリまたはアプリケーション ドメインに与えるアクセス許可を計算するために、4 つのセキュリティ ポリシー レベルがあります。各ポリシー レベルには、それぞれ独自のコード グループ階層構造とアクセス許可セットがあります。ランタイムは、付与するアクセス許可セットを、ポリシー内のすべての参加レベルで付与するアクセス許可の合計として計算します。

次のようなレベルがあります。

- エンタープライズ ポリシー。エンタープライズ構成ファイルが配布されているエンタープライズ設定に含まれるすべてのマネージ コードに適用されます。
- コンピュータ ポリシー。コンピュータ上のすべてのマネージ コードに適用されます。
- ユーザー ポリシー。共通言語ランタイムの起動時に現在のオペレーティング システムのユーザーに関連付けられている全プロセスに含まれるコードに適用されます。
- アプリケーション ドメイン ポリシー。ホストのアプリケーション ドメインに含まれるマネージ コードに適用されます。

詳細については、「セキュリティ ポリシー レベル」を参照してください。

コード グループ

コード グループは、メンバシップについて特定の条件を持つコードの論理的な集まりです。このメンバシップ条件を満たす任意のコードが、そのコード グループのメンバになります。コード グループは、ポリシーの適用時に評価されるアクセス許可セットに関連付けられています。詳細については、「コード グループ」を参照してください。

名前付きアクセス許可セット

名前付きアクセス許可セットは、管理者がコード グループに関連付けることができる一連のアクセス許可です。名前付きアクセス許可セットは、少なくとも 1 つのアクセス許可と、そのアクセス許可セットの名前および説明とで構成されます。特定のアクセス許可セットに対して、複数のコード グループを関連付けることができます。

次の表は、共通言語ランタイムに用意されている名前付きアクセス許可セットを示しています。

名前	説明
Nothing	アクセス許可なし (コードは実行できません)。
Execution	実行するためのアクセス許可。保護されているリソースの使用は許可されません。
Internet	不明な発生元からのコンテンツに適した既定のポリシー アクセス許可セット。

Local Intranet	エンタープライズ内の既定のポリシー アクセス許可セット。
Everything	検証を省略するためのアクセス許可を除く、すべての標準 (組み込み) アクセス許可セット。
FullTrust	すべてのリソースへの完全なアクセス許可。

詳細については、「名前付きアクセス許可セット」を参照してください。

証拠

証拠は、共通言語ランタイムがセキュリティ ポリシーに基づいてアクセス許可を決定するときに使用する情報です。証拠はランタイムに対して、コードには特定の特性があり、アプリケーションのディレクトリ、発行元、サイト、URL などを含めることができることを示します。詳細については、「証拠」を参照してください。

アプリケーション ドメイン ホスト

各 .NET Framework アプリケーションは、アプリケーション ドメインを作成し、そこにアセンブリを読み込むホストの制御下で、そのアプリケーション ドメイン内で実行されます。アプリケーション ドメインには次のホストを含めることができます。

- ブラウザ ホスト。Web サイトのコンテキスト内でコードを実行します。
- カスタム デザイン ホスト。ドメインを作成し、そのドメインにアセンブリ (動的アセンブリなど) を読み込みます。
- サーバー ホスト。サーバーに送信された要求を処理するコードを実行します。
- シェル ホスト。シェルからアプリケーション (.exe ファイル) を起動します。

次の表は、アプリケーション ドメイン関連のタスクの一覧です。

実行内容	参照項目
アプリケーション ドメインの作成	方法 : アプリケーション ドメインを作成する
アプリケーション ドメイン レベルのセキュリティ ポリシーの設定	アプリケーション ドメイン レベルのセキュリティ ポリシーの設定
アプリケーション ドメインの構成	方法 : アプリケーション ドメインを構成する
特定のアプリケーション ドメインでの関数の呼び出し	特定のアプリケーション ドメイン内での関数の呼び出し
アプリケーション ドメインからのセットアップ情報の取得	アプリケーション ドメインからのセットアップ情報の取得
アプリケーション ドメインのアンロード	方法 : アプリケーション ドメインをアンロードする

詳細については、「AppDomain」、「アプリケーション ドメイン ホスト」、および「アプリケーション ドメインを使用したプログラミング」を参照してください。

参照

処理手順

コード アクセス セキュリティ例外のトラブルシューティング

概念

ユーザー、ドメイン、およびアセンブリ別の分離

その他の技術情報

.NET Framework 構成ツール (Mscorcfg.msc) を使用したセキュリティ ポリシーの設定

コード アクセス セキュリティ ポリシー ツール (Caspol.exe) によるセキュリティ ポリシーの構成

Visual Basic での .NET Framework のコレクション

.NET Framework には、データの格納と取得のための専用のクラスが用意されています。これらのクラスでは、スタック、キュー、リスト、およびハッシュ テーブルがサポートされています。大半のコレクション クラスは同じインターフェイスを実装しています。それらのインターフェイスを実装すると、必要に応じて独自のコレクション クラスを作成することもできます。

どの種類のコレクションが自分のニーズに最も合っているかを判断する必要があります。

Visual Basic と .NET Framework のコレクション クラスの違い

.NET Framework のコレクション クラスは、System.Collections、System.Collections.Generic、System.Collections.Specialized、および System.Collections.ObjectModel の各名前空間で定義されています。Visual Basic と .NET Framework のコレクション クラスの主な違いは以下のとおりです。

- **先頭のインデックス番号**。.NET Framework のコレクションはインデックス番号が 0 から始まるのに対し、Visual Basic のコレクションはインデックス番号が 1 から始まります。つまり、Visual Basic のコレクションでは、要素のインデックス値は 1 から Count プロパティ (Collection オブジェクト) の値までなのに対し、.NET Framework のコレクションでは、要素のインデックス値は 0 からコレクションの Count プロパティより 1 小さい値までです。

- **要素の型**。Visual Basic のコレクションは Object 型の要素をサポートします。このコレクションはタイプ セーフではありません。任意のデータ型の要素を追加できます。通常は、これによりパフォーマンスが低下します。**Object** データ型から実際の型に要素を変換する必要があるからです。

  .NET Framework のコレクションの中にも、**Object** 型の要素を持つものもありますが、それ以外の大半は、厳密に型指定された要素を持ちます。つまり、固有の型の要素をサポートします。このため、タイプ セーフなコレクションとなり、通常は最適なパフォーマンスを実現できます。

- **キー付き要素**。Visual Basic のコレクションでは、要素を追加するときにキーを指定できます。キーは一意の文字列値であり、後でその文字列を使用して当該要素にアクセスできます。.NET Framework のコレクションでは、キーの扱いはさまざまです。キーをサポートするものとしないものがあります。

System.Collections のクラス

System.Collections 名前空間のクラスでは、要素は、固有の型のオブジェクトとしてではなく **Object** 型のオブジェクトとして格納されます。

次の表は、よく使用する主なクラスの一覧です。

クラス	説明
ArrayList	必要に応じてサイズが動的に拡大される配列を使用して IList インターフェイスを実装します。
BitArray	**Boolean** 値として表されるビット値のコンパクトな配列を管理します。**True** はビットがオン (1) であることを示し、**False** はビットがオフ (0) であることを示します。
Hashtable	キーのハッシュ コードに基づいて編成された、キーと値のペアのコレクションを表します。
Queue	先入れ先出し (FIFO) のオブジェクトのコレクションを表します。
Stack	後入れ先出し (LIFO) のオブジェクトの簡単な非ジェネリック コレクションを表します。

System.Collections.Generic と System.Collections.ObjectModel のクラス

System.Collections.Generic 名前空間および **System.Collections.ObjectModel** 名前空間には、厳密に型指定されたコレクションを作成し、要素を作成するときにデータ型を指定できる、ジェネリック型が用意されています

次の表は、よく使用する主なクラスの一覧です。

クラス	説明
Collection	ジェネリック コレクションの基本クラスです。
Dictionary	キーに基づいて編成された、キーと値のペアのコレクションを表します。

KeyedCollection	値の中にキーが埋め込まれているコレクションの抽象基本クラスを提供します。
LinkedList	ダブルリンク リストを表します。
LinkedListNode	**LinkedList** のノードを表します。このクラスは継承できません。
List	必要に応じてサイズが動的に拡大される配列を使用して IList インターフェイスを実装します。
Queue	先入れ先出しのオブジェクトのコレクションを表します。
SortedDictionary	キーに基づいて並べ替えられた、キーと値のペアのコレクションを表します。
SortedList	関連付けられた IComparer 実装に基づいて、キーにより並べ替えられた、キーと値のペアのコレクションを表します。
Stack	任意の同じ型のインスタンスの、後入れ先出し (LIFO) の可変サイズのコレクションを表します。
ReadOnlyCollection	読み取り専用のジェネリック コレクションの基本クラスです。

System.Collections.Specialized のクラス

System.Collections.Specialized 名前空間には、文字列専用のコレクションやリンク リスト、ハイブリッド ディクショナリなど、厳密に型指定された専用のコレクション クラスが用意されています。

次の表は、よく使用する主なクラスの一覧です。

クラス	説明
CollectionsUtil	文字列の大文字小文字の違いを無視するコレクションを作成します。
HybridDictionary	コレクションが小さいうちは ListDictionary を使用し、コレクションが大きくなってきたら **Hashtable** に切り替える IDictionary を実装します。
ListDictionary	シングルリンク リストを使用して **IDictionary** を実装します。おおむね 10 個以下の項目を格納するコレクションでの使用をお勧めします。
NameObjectCollectionBase	キーまたはインデックスのいずれかでアクセスできる、関連付けられた文字列キーとオブジェクト値のコレクションの抽象基本クラスです。
NameValueCollection	キーまたはインデックスのいずれかでアクセスできる、関連付けられた文字列キーと文字列値のコレクションを表します。
OrderedDictionary	キーまたはインデックスに基づいて並べ替えられた、キーと値のペアのコレクションを表します。
StringCollection	文字列のコレクションを表します。
StringDictionary	キーと、オブジェクトではなく文字列として厳密に型指定された値とのハッシュ テーブルを実装します。

参照

関連項目

Collection オブジェクト (Visual Basic)

概念

Visual Basic におけるコレクション

その他の技術情報

Visual Basic への理解を深める

Visual Basic での .NET Framework のファイル I/O

分離ストレージなど、複雑なファイル I/O 操作を実行するときに、Visual Basic ユーザーは、.NET Framework およびその中に用意されているクラスを調査することが必要な場合があります。このセクションでは、.NET Framework でのファイル I/O および **FileSystem** の考え方の概要を説明し、一般的に使用するクラスの一覧を示します。

このセクションの内容

.NET Framework のファイル I/O とファイル システムの基礎

.NET Framework におけるファイル I/O の考え方の概要を示します。ストリーム、分離ストレージ、ファイル イベント、ファイル属性、ファイル アクセスなどです。

.NET Framework のファイル I/O とファイル システムで使用するクラス

.NET Framework のファイル I/O と **FileSystem** で使用するクラスの一覧を示します。たとえば、基本的なファイル I/O クラス、ストリームを作成するクラス、ストリーム ライタおよびストリーム リーダーを作成するクラスなどです。

参照

処理手順

方法 : StreamReader を使用してファイルからテキストを読み取る (Visual Basic)
方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む
方法 : Visual Basic でテキスト ファイルを読み取る

関連項目

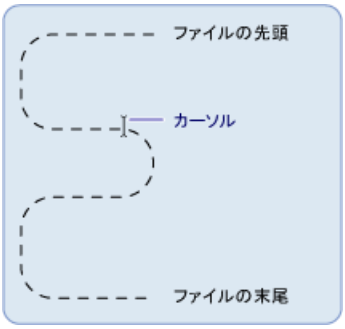
System.IO Namespace

.NET Framework のファイル I/O とファイル システムの基礎

System.IO 名前空間には、.NET Framework でファイルとディレクトリを操作する機能を実現する File クラスと Directory クラスが用意されています。これらのオブジェクトのメソッドは静的メンバまたは共有メンバであるため、最初にクラスのインスタンスを作成しなくてもメンバを直接使用できます。これらのクラスに関連するクラスとして、FileInfo クラスと DirectoryInfo クラスがあります。**My** 機能を使用しているユーザーにはおなじみのクラスです。これらのクラスを使用するには、名前を完全修飾するか、または、関係するコードの先頭に **Imports** ステートメントを記述して、適切な名前空間をインポートする必要があります。詳細については、「Imports ステートメント」を参照してください。

ストリームの定義

.NET Framework では、ファイルに対する読み取りと書き込みをサポートするストリームを使用できます。ストリームとは、1 次元の連続したデータの集まりと考えることができます。ストリームには先頭と末尾があり、カーソルでストリーム内での現在の位置を示します。



ストリームの操作

ストリームに格納されているデータは、メモリ、ファイル、または TCP/IP ソケットから取得したものです。ストリームに対しては、次の基本操作を実行できます。

- 読み取り。ストリームを読み取ったり、文字列やバイト配列などのデータ構造にストリームからデータを転送したりできます。
- 書き込み。ストリームに書き込んだり、データ ソースからストリームにデータを転送したりできます。
- シーク。ストリーム内の現在の位置をクエリおよび変更できます。

詳細については、「ストリームの構成」を参照してください。

ストリームの種類

.NET Framework では、ストリームは Stream クラスで表されます。その他のすべてのストリームのための抽象クラスです。**Stream** クラスのインスタンスを直接作成することはできません。これを実装するいずれかのクラスを使用する必要があります。

ストリームにはさまざまな種類がありますが、ファイルの入出力 (I/O) を処理するという目的のために最も重要なのは、ファイルに対する読み取りと書き込みを実現する FileStream クラスと、分離ストレージに対するファイルやディレクトリの作成を実現する IsolatedStorageFileStream クラスです。この他、ファイル I/O を処理するときに使用できるストリームには、以下のものがあります。

- BufferedStream
- CryptoStream
- MemoryStream
- NetworkStream

次の表は、ストリームで一般的に実行するタスクの一覧です。

目的	参照項目
データ ファイルに対する読み取りと書き込み	方法 : 新しく作成されたデータ ファイルに対して読み書きする
XML ファイルの読み取り	共通の XmlReader タスク
ファイルにテキストを書き込むためのストリーム ライタの作成	ライタの作成
ファイルのテキストの読み取り	方法 : ファイルからテキストを読み取る

ファイルへのテキストの書き込み	方法 : ファイルにテキストを書き込む
文字列からの文字の読み取り	方法 : 文字列から文字を読み取る
文字列への文字の書き込み	方法 : 文字列に文字を書き込む
データの暗号化	データの暗号化
データの復号化	データの復号化

ファイル アクセスと属性

ファイルの作成、オープン、および共有の方法は、FileAccess、FileMode、および FileShare の各列挙体で制御できます。これらの列挙体には、**FileStream** クラスのコンストラクタで使用するフラグが含まれています。たとえば、**FileStream** を開くかまたは新規作成するときに、ファイルを追加書き込み用に開くかどうか、指定のファイルが存在しない場合にファイルを新規作成するかどうか、ファイルを上書きするかどうか、などを **FileMode** 列挙体で指定できます。

FileAttributes 列挙体を使用すると、ファイル固有の情報を収集できます。**FileAttributes** 列挙体は、格納されているファイルの属性を返します。これらの属性によって、圧縮ファイル、暗号化されたファイル、隠しファイル、読み取り専用ファイル、アーカイブ、ディレクトリ、システム ファイル、一時ファイルであるかどうかがわかります。

次の表は、ファイル アクセスとファイル属性に関連するタスクの一覧です。

目的	参照項目
ログ ファイルのオープンとテキストの追加	方法 : ログ ファイルを開いて情報を追加する
ファイルの属性の判断	**FileAttributes**

ファイルのアクセス許可

ファイルおよびディレクトリに対するアクセスの制御は、FileIOPermission クラスで行うことができます。これは、Web フォームの開発者には特に重要な場合があります。既定では、Web フォームは、ASPNET という名前の特別なローカル ユーザー アカウントのコンテキストで実行されます。ASPNET は、ASP.NET および .NET Framework のインストール時に作成されます。ASPNET ユーザー アカウントはアクセス許可が制限されているため、アプリケーションがリソースへのアクセスを要求したときに、ユーザーが処理を実行できない場合があります (たとえば、Web アプリケーションからファイルへの書き込みなど)。詳細については、「アクセス許可」、「アクセス許可の要求」、および「FileIOPermission クラス」を参照してください。

次の表は、ファイルのアクセス許可に関連するタスクの一覧です。

目的	参照項目
最低限のアクセス許可の要求	方法 : アンマネージ コードへのアクセス許可を要求する
オプションのアクセス許可の要求	方法 : RequestOptional フラグを使用してオプションのアクセス許可を要求する
アクセス許可の拒否	方法 : RequestRefuse フラグを使用することにより、アクセス許可を拒否する
組み込みのアクセス許可の要求	方法 : 名前付きアクセス許可セットに対するアクセス許可を要求する
XML でエンコードされたアクセス許可の要求	XML でエンコードされたアクセス許可の要求

分離ファイル ストレージ

分離ストレージとは、ファイルを扱うときに、必要なアクセス許可をユーザーまたはコードが持っていないために生じる問題を解決するためのものです。分離ストレージでは、各ユーザーにデータ コンパートメントが割り当てられます。このデータ コンパートメントには、1 つまたは複数のストアを保持できます。ストアは、ユーザーおよびアセンブリごとに互いに分離できます。ストアにアクセスできるのは、それを作成したユーザーおよびアセンブリのみです。ストアは、完全な仮想ファイル システムとして機能します。つまり、ストア内にディレクトリやファイルを作成および操作できます。

次の表は、分離ファイル ストレージに一般に関連するタスクの一覧です。

目的	参照項目
分離ストアの作成	方法 : 分離ストレージでストアを取得する
分離ストアの列挙	方法 : 分離ストレージでストアを列挙する
分離ストアの削除	方法 : 分離ストレージでストアを削除する
分離ストレージのファイルまたはディレクトリの作成	方法 : 分離ストレージでファイルおよびディレクトリを作成する
分離ストレージのファイルの検索	方法 : 分離ストレージ内でファイルおよびディレクトリを検索する
分離ストレージのファイルに対する読み取りと書き込み	方法 : 分離ストレージ内でファイルの読み取りと書き込みを行う
分離ストレージのファイルまたはディレクトリの削除	方法 : 分離ストレージでファイルおよびディレクトリを削除する

ファイルのイベント

FileSystemWatcher コンポーネントを使用すると、自システム上のファイルとディレクトリ、またはネットワークでアクセスできる任意のコンピュータ上のファイルとディレクトリの変更を監視できます。たとえば、ファイルが変更されたときに、その旨をユーザーに警告することが必要な場合があります。変更が行われると、1 つまたは複数のイベントが発生し、バッファに格納され、**FileSystemWatcher** コンポーネントに渡されて処理されます。詳細については、「ファイル システム イベントへの応答」を参照してください。

目的	参照項目
ファイル システム イベントのハンドラの作成	方法 : ファイル システム イベントに対するハンドラを作成する
FileSystemWatcher コンポーネント インスタンスの構成	方法 : FileSystemWatcher コンポーネントのインスタンスを設定する

参照

概念

ストリームの構成
基本のファイル I/O
非同期ファイル I/O
.NET Framework のファイル I/O とファイル システムで使用するクラス

その他の技術情報

分離ストレージ

.NET Framework のファイル I/O とファイル システムで使用するクラス

以下の表は、.NET Framework のファイル I/O で一般的に使用するクラスの一覧です。ファイル I/O クラス、ストリームの作成に使用するクラス、ストリームの読み取りと書き込みに使用するクラスに分類されています。

.NET Framework 2.0 ドキュメントで詳細な一覧を参照するには、「.NET Framework クラス ライブラリの概要」を参照してください。

ファイル、ドライブ、およびディレクトリ用の基本 I/O クラス

次の表は、ファイル I/O に使用する主要なクラスの一覧とその説明です。

クラス	説明
System.IO.Directory	ディレクトリやサブディレクトリを作成、移動、および反復処理するための静的メソッドを提供します。
System.IO.DirectoryInfo	ディレクトリやサブディレクトリを作成、移動、および反復処理するためのインスタンス メソッドを提供します。
System.IO.DriveInfo	ドライブを作成、移動、および反復処理するためのインスタンス メソッドを提供します。
System.IO.File	ファイルを作成、コピー、削除、移動、およびオープンするための静的メソッドを提供し、また **FileStream** の作成を支援します。
System.IO.FileAccess	読み取り専用、書き込み専用、読み取り/書き込みの各ファイル アクセスの定数を定義します。
System.IO.FileAttributes	Archive、Hidden、ReadOnly など、ファイルおよびディレクトリの属性を提供します。
System.IO.FileInfo	ファイルを作成、コピー、削除、移動、およびオープンするための静的メソッドを提供し、また **FileStream** の作成を支援します。
System.IO.FileMode	ファイルを開く方法を制御します。このパラメータは、**FileStream** および **IsolatedStorageFileStream** の数多くのコンストラクタで、および **File** と **FileInfo** の **Open**メソッドで指定します。
System.IO.FileShare	他のファイル ストリームが同一のファイルに対して行うことができるアクセスの種類を制御するための定数を定義します。
System.IO.Path	ディレクトリ文字列を処理するためのメソッドとプロパティを提供します。
System.Security.Permissions.FileIOPermission	Read、Write、Append、および PathDiscovery の各アクセス許可を定義し、ファイルおよびフォルダへのアクセスを制御します。

ストリームの作成に使用するクラス

次の表は、ストリームの作成に使用する主要なクラスの一覧とその説明です。

クラス	説明
System.IO.BufferedStream	他のストリームの読み取りおよび書き込み操作に対しバッファリング層を追加します。
System.IO.FileStream	Seek メソッドにより、ファイルへのランダム アクセスをサポートします。**FileStream** は、既定ではファイルを同期で開きますが、非同期の操作もサポートしています。
System.IO.MemoryStream	バッキング ストアがファイルではなくメモリであるストリームを作成します。

System.Net.Sockets.NetworkStream	ネットワーク アクセスの基になるデータ ストリームを提供します。
System.Security.Cryptography.CryptoStream	データ ストリームを暗号化方式にリンクするストリームを定義します。

ストリームの読み取りと書き込みに使用するクラス

次の表は、ストリームによるファイルの読み取りと書き込みに使用する固有のクラスの一覧です。

クラス	説明
System.IO.BinaryReader	エンコードされた文字列とプリミティブ データ型を **FileStream** から読み取ります。
System.IO.BinaryWriter	エンコードされた文字列とプリミティブ データ型を **FileStream** に書き込みます。
System.IO.StreamReader	**FileStream** から文字を読み取ります。その際、CurrentEncoding を使用して、文字とバイトの間の変換を行います。**StreamReader** には、当該のストリームの正しい **CurrentEncoding** の確認を試みるコンストラクタがあります。確認は、バイト順マークなどの **CurrentEncoding** 固有のプリアンブルがあるかどうかに基づいて行われます。
System.IO.StreamWriter	**FileStream** に文字を書き込みます。その際、Encoding を使用して、文字をバイトに変換します。
System.IO.StringReader	**String** から文字を読み取ります。出力は、任意のエンコーディングのストリームまたは **String** のいずれかです。
System.IO.StringWriter	**String** に文字を書き込みます。出力は、任意のエンコーディングのストリームまたは **String** のいずれかです。

参照

概念

ストリームの構成
基本のファイル I/O
非同期ファイル I/O
.NET Framework のファイル I/O とファイル システムの基礎

その他の技術情報

分離ストレージ

Visual Basic での .NET Framework のグラフィックス

.NET Framework には、グラフィックスを操作するための GDI+ アプリケーション プログラミング インターフェイス (API) があります。GDI+ は、Windows の グラフィカル デバイス インターフェイス (GDI) の高度な実装です。GDI+ を使用すると、グラフィックの作成やテキストの描画を行ったり、グラフィカル イメージをオブジェクトとして操作できます。

GDI+ は使いやすさと性能を両立するようにデザインされています。GDI+ を使用することにより、Windows フォームやコントロール上でグラフィカル イメージをレンダリングできます。Web フォーム上で GDI+ を直接使用することはできませんが、Image Web サーバー コントロールによってグラフィカル イメージを表示できます。

GDI+ について

Windows フォーム コントロールを作成するときには、GDI+ を使用してイメージへアクセスし、更新できます。また、GDI+ を使用して、アプリケーションのユーザー インターフェイスとは無関係に、独自のイメージを作成することもできます。

.NET Framework でイメージ上に描画するには、イメージに関連付けられた Graphics オブジェクトを使用する必要があります。

場合によっては、イメージの **Graphics** オブジェクトを直接取得できることがあります。たとえば、Windows フォーム コントロールを作成するときには、OnPaint メソッドをオーバーライドして、コントロールのイメージに対応する **Graphics** オブジェクトにアクセスできます。

その他の場合、たとえば独自のイメージを作成するときには、グラフィックス オブジェクトを作成する必要があります。FromImage 共有メソッドは、イメージを引数に受け取り、そのイメージに関連付けられた **Graphics** オブジェクトを返します。

Graphics クラスには、描画操作およびイメージ操作のメソッドが数多くあります。よく使用するメソッドの一部を以下に示します。

- 線を描画するためのメソッド : DrawArc、DrawBezier、DrawEllipse、DrawImage、DrawLine、DrawPolygon、DrawRectangle、および DrawString。
- 図形を塗りつぶすためのメソッド : FillClosedCurve、FillEllipse、FillPath、FillPolygon、および FillRectangle。
- 描画サーフェイスをクリアするためのメソッド : Clear。
- 新しい **Graphics** オブジェクトをイメージから作成するためのメソッド : **FromImage**。

上記のメソッドの中には、System.Drawing 名前空間で定義されている構造体またはクラスを引数として受け取るものがあります。次の表は、特によく使用する GDI+ のクラスおよび構造体の一覧です。

クラス/構造体	説明
System.Drawing.Bitmap	GDI+ ビットマップをカプセル化します。GDI+ ビットマップは、グラフィックス イメージのピクセル データとその属性で構成されます。**Bitmap** は、ピクセル データで定義されるイメージの操作に使用するオブジェクトです。
System.Drawing.Brushes	すべての標準色のブラシを定義します。
System.Drawing.Color	ARGB カラーを表します。
System.Drawing.Font	フォント フェイス、ポイント数、スタイル属性など、テキストの特定の形式を定義します。
System.Drawing.Pen	直線および曲線の描画に使用するオブジェクトを定義します。
System.Drawing.Pens	すべての標準色のペンを定義します。
System.Drawing.Point	2 次元面の 1 点を定義する、順序付けされた整数の X 座標と Y 座標のペアを表します。
System.Drawing.Rectangle	四角形の位置とサイズを表す 4 つの整数を格納します。もっと高度な領域関数が必要な場合は、Region オブジェクトを使用します。
System.Drawing.SolidBrush	単一色のブラシを定義します。ブラシは、四角形、楕円、扇形、多角形、パスなどのグラフィックス図形の塗りつぶしに使用します。

System.Drawing.TextureBrush	**TextureBrush** クラスの各プロパティは、イメージを使用して図形の内部を塗りつぶす Brush オブジェクトです。

リソースの管理

描画クラスの多くは、アンマネージ システム リソースをカプセル化しているため、IDisposable を実装しています。これらのいずれかのクラスの新しいインスタンスを作成する場合は、オブジェクトの使用が済んだときに、クラスの Dispose メソッドを呼び出す必要があります。

または、**Using** ステートメントでオブジェクトを作成することもできます。この場合、オブジェクトの **Dispose** メソッドが暗黙的に呼び出されます。詳細については、「オブジェクトの有効期間 : オブジェクトの作成と破棄」および「Using ステートメント (Visual Basic)」を参照してください。

関連項目

グラフィックス (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 におけるグラフィック レンダリング モデルの変更点について説明します。

Windows フォームにおけるグラフィックスと描画

Windows フォーム アプリケーションでグラフィックスを使用するためのロードマップです。

グラフィックスの概要 (Windows フォーム)

グラフィックス関連のマネージ クラスの概要を示します。

GDI+ マネージ コードについて

GDI+ のマネージ クラスに関する情報を示します。

マネージ グラフィックス クラスの使用

GDI+ のマネージ クラスを使用してさまざまなタスクを実行する方法を説明します。

コントロールのカスタム描画およびレンダリング

コントロールの描画のためのコードを提供する方法について説明します。

Image Web サーバー コントロールの概要

Web フォーム ページにイメージを表示し、コードでイメージを管理するためのコントロールについて説明します。

イメージ エディタ

イメージ エディタを使用してアプリケーション用のイメージ ファイルを作成する方法について説明するトピックへのリンクを示します。

参照

関連項目

Using ステートメント (Visual Basic)
System.Drawing

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄

Visual Basic での .NET Framework のログ記録とトレース

My.Application.Log オブジェクトおよび **My.Log** オブジェクトは、.NET Framework のログ記録クラスを基に構築されています。新しいリスナの追加かログ出力のフィルタ処理のいずれかによって、**Log** オブジェクトをカスタマイズする必要があるときには、.NET Framework を使用します。このセクションでは、.NET Framework でのトレースとデバッグの考え方の概要を説明し、一般的に使用するクラスの一覧を示します。

このセクションの内容

.NET Framework のログの基礎

.NET Framework でのログ記録、トレース、およびインストルメントの概念について、概要を説明します。

.NET Framework でログの記録に使用するクラス

.NET Framework でのログ記録、トレース、およびインストルメントで使用するクラスの一覧を示します。

参照

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
System.Diagnostics

概念

アプリケーションからの情報のログ記録

その他の技術情報

アプリケーションのトレースとインストルメント

.NET Framework のログの基礎

Visual Basic アプリケーションから情報をログに記録する方法は、Visual Basic の **My.Application.Log** オブジェクトおよび **My.Log** オブジェクトを使用する方法以外にも数多くあります。つまり、.NET Framework クラスに用意されているさまざまなクラスを使用できます。情報をログに記録すると、アプリケーションのパフォーマンスを監視および計測したり、エラーを診断したりできます。

トレース、デバッグ、およびインストルメント

.NET Framework のドキュメントでは、ログに関連する用語として、トレース、デバッグ、およびインストルメントという用語が使われています。

- トレースとは、実行中のアプリケーションの実行状態を監視する方法です。通常は Trace クラスを使用して実装されます。アプリケーションの開発時に、アプリケーションに対してトレース ステートメントを追加できます。アプリケーションの開発中でも開発したアプリケーションの配置後でも、そのインストルメンテーションを使用できます。
- デバッグはトレースと似ていますが、一般には、出力する情報はこちらの方が多く、またリリース ビルドではオフにします。通常は **Trace** クラスを使用して実装されます。
- インストルメントとは、製品のパフォーマンスのレベルを監視または計測するコードをアプリケーションに追加することです。ログ、トレース、およびデバッグは、それぞれインストルメンテーションの形態の 1 つです。コード内の重要な場所にトレース ステートメントを配置することにより、アプリケーションをインストルメントできます。これは、分散アプリケーションでは特に有用です。

  トレース ステートメントを使用してアプリケーションをインストルメントすると、不適切な動作についての情報を表示できるだけでなく、アプリケーションの実行状態を監視するための情報も表示できます。

Trace クラスおよび Debug クラスを使用すると、Visual Basic の **Log** オブジェクトと同様に、エラーおよびアプリケーションの実行についての情報を後で分析するために、ログ、テキスト ファイル、またはその他のデバイスに記録できます。また、出力をより細かく制御できるメソッドも用意されています。その他のトレース クラスについては、「.NET Framework でログの記録に使用するクラス」を参照してください。

ログとトレースの基礎

アプリケーションのインストルメントにより、以下が実現されます。

- **コードのトレース。** 実行時に、アプリケーションの実行状態について示すメッセージを受け取ります。詳細については、「実装とトレースの概要」を参照してください。
- **デバッグ出力の受け取り。** 開発中のアプリケーションのプログラミング エラーを追跡して修正します。詳細については、「Visual Studio でのデバッグ」を参照してください。
- **パフォーマンス カウンタへのアクセス。** アプリケーションのパフォーマンスを追跡します。詳細については、「パフォーマンスしきい値の監視の概要」を参照してください。
- **イベント ログへの書き込み。** アプリケーションを実行中の主なイベントを追跡します。詳細については、「アプリケーション、サーバー、およびセキュリティ イベントのログの記録」を参照してください。

Trace クラスおよび **Debug** クラスは、アプリケーションの開発中または配置後に、アプリケーションのパフォーマンスを追跡および検査する手段を提供します。たとえば、**Trace** クラスを使用して、配置済みのアプリケーションで発生した特定の種類のアクション (新しいデータベース接続の作成など) を追跡できます。これにより、アプリケーションの効率を監視できます。

参照

関連項目

System.Diagnostics

概念

Visual Basic での .NET Framework のログ記録とトレース
実装とトレースの概要
.NET Framework でログの記録に使用するクラス

その他の技術情報

アプリケーションのトレースとインストルメント

.NET Framework でログの記録に使用するクラス

このトピックでは、.NET Framework のログの記録およびトレースで一般的に使用されるクラスの一覧表を示します。表は、ログ出力クラス、ログ リスナ クラス、およびログ フィルタ処理クラスに分かれています。.NET Framework 2.0 ドキュメントで詳細な一覧を参照するには、「.NET Framework クラス ライブラリの概要」を参照してください。

ログ出力の基本的なクラス

次の表は、ログ出力に使用する主要なクラスの一覧とその説明です。

クラス	説明
Debug	コードのデバッグに使用する一連のメソッドとプロパティを提供します。このクラスは継承できません。
Trace	コードの実行をトレースするための一連のメソッドとプロパティを提供します。このクラスは継承できません。
TraceSource	コードの実行をトレースするための一連のメソッドとプロパティを提供します。

ログ リスナ クラス

次の表は、主要なログ リスナ クラスの一覧とその説明です。

クラス	説明
TraceListener	トレース出力およびデバッグ出力を監視するリスナの抽象基本クラスです。
ConsoleTraceListener	トレース出力またはデバッグ出力を、標準出力と標準エラー出力ストリームのいずれかに転送します。
DefaultTraceListener	トレースの既定の出力メソッドと動作を提供します。
DelimitedListTraceListener	トレース出力またはデバッグ出力を、TextWriter (StreamWriter など) または Stream (FileStream など) のいずれかに転送します。
EventLogTraceListener	トレース出力またはデバッグ出力を EventLog に転送する簡単なリスナを提供します。
FileLogTraceListener	ログ出力をファイルに転送する簡単なリスナを提供します。
TextWriterTraceListener	トレース出力またはデバッグ出力を、**TextWriter** または **Stream** (Out や **FileStream** など) のいずれかに転送します。
XmlWriterTraceListener	トレース出力またはデバッグ出力を、**TextWriter** または **Stream** (**FileStream** など) のいずれかに転送します。

ログのフィルタ処理クラスおよびスイッチ クラス

次の表は、主要なログ フィルタ処理クラスおよびスイッチ クラスの一覧とその説明です。

クラス	説明
Switch	新しいデバッグ スイッチおよびトレース スイッチを作成するための抽象基本クラスです。
BooleanSwitch	デバッグ出力およびトレース出力を制御する単純なオン/オフ スイッチを提供します。
SourceSwitch	コードの再コンパイルなしでトレース出力およびデバッグ出力を制御するための複数レベルのスイッチを提供します。
TraceSwitch	コードの再コンパイルなしでトレース出力およびデバッグ出力を制御するための複数レベルのスイッチを提供します。

EventTypeFilter	トレース出力を制御する、重大度レベルに基づくスイッチを提供します。
SourceFilter	リスナがトレース元に基づいてメッセージをトレースする必要があるかどうかを指定します。
TraceFilter	トレース フィルタの実装のためのメソッドを定義します。

参照

関連項目

System.Diagnostics

概念

Visual Basic での .NET Framework のログ記録とトレース

.NET Framework のログの基礎

その他の技術情報

アプリケーションのトレースとインストルメント

Visual Basic による .NET Framework でのネットワーク操作

Visual Basic ユーザーは、複雑なネットワーク操作を実行するときに、.NET Framework およびその中に用意されているクラスを調査することが必要な場合があります。このセクションでは、.NET Framework でのネットワークの考え方の概要を説明し、一般的に使用するクラスの一覧を示します。

このセクションの内容

.NET Framework のネットワーク操作の基礎

.NET Framework でのネットワークの考え方の概要を示します。WebRequest クラスおよび WebResponse クラス、セキュリティ、ネットワークのトレースなどです。

.NET Framework でネットワーク操作に使用するクラス

.NET Framework でのネットワーク操作で使用するクラスの一覧を示します。

参照

WebRequest

Uniform Resource Identifier (URI) に対する要求を実行します。

WebResponse

URI からの応答を提供します。

WebClient

URI で識別されるリソースとの間でデータを送受信するための共通のメソッドが用意されています。

.NET Framework のネットワーク操作の基礎

Microsoft .NET Framework には、アプリケーションにすばやく簡単に統合できる、複数層の拡張可能なインターネット サービスのマネージ実装が用意されています。これらのクラスを使用するには、名前を完全修飾するか、または、関係するコードの先頭に 1 つまたは複数の **Imports** ステートメントを記述して、適切な名前空間をインポートする必要があります。詳細については、「Imports ステートメント」を参照してください。

インターネット リソースへのアクセス

要求/応答モデルでインターネット リソースにアクセスするには、3 つの情報が必要です。これらは、それぞれ固有の .NET Framework クラスとして用意されています。

- Uri クラスには、インターネット リソースの URI (Uniform Resource Identifier) が格納されています。URI は、少なくとも 3 つの部分で構成され、場合によっては 4 つになります。具体的には、通信プロトコルを識別するスキーム識別子、サーバー識別子、パス識別子、および省略可能なクエリ文字列の 4 つです。
- WebRequest クラスには、リソースに対する要求が格納されています。
- WebResponse クラスは、受け取る応答用のコンテナです。

データのアップロードとダウンロード

インターネット リソースに対して単純な要求を行う必要があるアプリケーションについては、データをアップロードまたはダウンロードするためのメソッドが WebClient クラスに用意されています。**WebClient** は **WebRequest** クラスを使用してインターネット リソースへのアクセスを実現するので、登録されているプラグ可能なプロトコルはどれでも使用できます。

System.Net.Sockets 名前空間には、要求/応答モデルを使用しないアプリケーションやネットワークに対して待機する必要があるアプリケーション用として、TcpClient、TcpListener、および UdpClient の各クラスが用意されています。これらのクラスは、さまざまなトランスポート プロトコルを使用して接続のための詳細を処理し、ネットワーク接続をストリームとしてアプリケーションに公開します。

詳細については、「データの要求」を参照してください。

次の表は、インターネット リソースに対するデータ要求に関する一般的なタスクの一覧です。

目的	参照項目
インターネット要求を作成する。	インターネット要求の作成
ネットワークにデータを送信する。	ネットワークでのストリームの使用
インターネット リソースに対して非同期の要求を行う。	非同期要求の作成
HTTP 固有のプロパティにアクセスする。	HTTP
特定の要求を接続プールに関連付ける。	接続のグループ化
プロキシ インスタンスを構成する。	プロキシによるインターネット アクセス
インターネット リソースに対して非同期の要求を行う。	非同期要求の作成
TCP を使用してデータを要求する。	TCP サービスの使用
UDP を使用してデータを要求する。	UDP サービスの使用

プラグ可能なプロトコル

WebRequest 抽象クラスと **WebResponse** 抽象クラスは、プラグ可能なプロトコルの基本クラスです。**WebRequest** クラスと **WebResponse** クラスからプロトコル固有のクラスを派生することで、使用するプロトコルを指定せずに、インターネット リソースからのデータを要求し、応答を読み取ることができます。

詳細については、「プラグ可能なプロトコルのプログラミング」を参照してください。

次の表は、プラグ可能なプロトコルに関する一般的なタスクの一覧です。

目的	参照項目
WebRequest を派生する。	WebRequest からの派生
WebResponse を派生する。	WebResponse からの派生
プロトコル固有のプロパティにアクセスする。	方法 : WebRequest を型キャストしてプロトコル固有のプロパティにアクセスする

ネットワークのトレース

ネットワークのトレースでは、メソッド呼び出しについての情報、およびマネージ アプリケーションによって生成されるネットワーク トラフィックについての情報にアクセスできます。この機能は、開発中のアプリケーションのデバッグや、配置済みのアプリケーションの分析に役立ちます。ネットワークのトレースの出力は、開発時および稼動環境でのさまざまな使用方法をサポートするようにカスタマイズできます。

詳細については、「ネットワークのトレース」を参照してください。

次の表は、ネットワークのトレースに関する一般的なタスクの一覧です。

目的	参照項目
ネットワークのトレースを有効にする。	ネットワーク トレースの有効化
トレース情報を読み取る。	ネットワーク トレースの解釈
ネットワークのトレースを構成する。	方法 : ネットワーク トレースを構成する

キャッシュ管理

キャッシュとは、アプリケーションによって要求されたリソースの一時的なストレージです。アプリケーションが、同じリソースを 2 回以上要求した場合、そのリソースをキャッシュから返すことができます。これにより、同じリソースをサーバーに再度要求するというオーバーヘッドを避けることができます。

要求されたリソースのキャッシュ コピーで要求を満たせるかどうかの判断は、キャッシュ ポリシーによって定義した規則に基づいて行われます。

詳細については、「ネットワーク アプリケーションのキャッシュ管理」を参照してください。

次の表は、キャッシュ管理に関する一般的なタスクの一覧です。

目的	参照項目
位置ベースのキャッシュ ポリシーを設定する。	方法 : 場所ベースのキャッシュ ポリシーをアプリケーションに対して設定する
既定の時間ベースのキャッシュ ポリシーを設定する。	方法 : 既定の時間ベースのキャッシュ ポリシーをアプリケーションに対して設定する
時間ベースのキャッシュ ポリシーをカスタマイズする。	方法 : 時間ベースのキャッシュ ポリシーをカスタマイズする
要求に対してキャッシュ ポリシーを設定する。	方法 : 要求に対するキャッシュ ポリシーを設定する

セキュリティ

System.Net クラスには、インターネット アプリケーションで共通で使用される認証機構のサポートや .NET Framework のコードアクセス許可のサポートが組み込まれており、インターネット アプリケーション用にセキュリティ保護された環境を提供します。

詳細については、「System.Net クラスのベスト プラクティス」を参照してください。

次の表は、セキュリティに関する一般的なタスクの一覧です。

目的	参照項目
HTTP サーバーに対して SSL (Secure Sockets Layer) 接続を使用する。	SSL (Secure Sockets Layer) の使用
HTTP 認証メソッドを使用して、HTTP サーバーに対し、認証された接続を確立する。	インターネット認証

インターネット接続を使用するアプリケーションにコード アクセス セキュリティを設定する。	Web アクセス許可とソケット アクセス許可
基本認証およびダイジェスト認証を使用する。	基本認証とダイジェスト認証
NTLM 認証および Kerberos 認証を使用する。	NTLM 認証と Kerberos 認証
System.Net を使用して XML Web サービスをセキュリティ保護する。	ASP.NET を使用して作成した XML Web サービスのセキュリティ

参照

関連項目

System.Net
System.Net.Sockets

概念

System.Net クラスのベスト プラクティス
インターネット アプリケーションの構成

その他の技術情報

ネットワーク プログラミング
Visual Studio のネットワークの名前空間

.NET Framework でネットワーク操作に使用するクラス

System.Net、System.Net.Sockets、および Uri には、アプリケーションにすばやく簡単に統合できる、複数層の拡張可能なインターネット サービスのマネージ実装が用意されています。

ネットワーク操作で使用する基本的なクラス

この表は、.NET Framework のネットワーク操作で使用する、特に重要なクラスの一覧です。より詳細なリストについては、.NET Framework のドキュメントを参照してください。

クラス	説明
System.Net	ネットワークで使用されている多くのプロトコル用の単純なプログラミング インターフェイスが用意されています。
System.Net.Authorization	インターネット サーバー用の認証メッセージが含まれています。
Cookie	Cookie の管理に使用する一連のメソッドとプロパティが用意されています。
WebClient	URI (Uniform Resource Identifier) で識別されるリソースとの間でデータを送受信するための共通のメソッドが用意されています。
WebRequest	URI に対して要求を行います。
WebResponse	URI からの応答を提供します。
System.Net.Sockets	ネットワークへのアクセスを厳密に制御する必要のある開発者のための、Windows ソケット (Winsock) インターフェイスのマネージ実装が用意されています。
System.Net.Sockets.TcpClient	TCP ネットワーク サービス用のクライアント接続を提供します。
System.Net.Sockets.TcpListener	TCP ネットワーク クライアントからの接続を待機します。
System.Net.Sockets.UdpClient	ユーザー データグラム プロトコル (UDP) のネットワーク サービスを提供します。
System.Uri	URI のオブジェクト表現を提供します。
UriBuilder	URI を作成および修飾するための、カスタムのコンストラクタおよび修飾子が用意されています。

参照

概念

.NET Framework のネットワーク操作の基礎

Visual Basic による .NET Framework でのポート操作

.NET Framework の System.IO.Ports 名前空間のクラスを使用して、コンピュータのシリアル ポートにアクセスできます。最も重要なクラスは SerialPort です。このクラスは、同期 I/O やイベント ドリブン I/O の実行、ピンやブレーク状態へのアクセス、およびシリアル ドライバのプロパティへのアクセスのためのフレームワークを提供します。これは Stream オブジェクトでラップでき、BaseStream プロパティを通じてアクセスできます。**SerialPort** を **Stream** オブジェクトでラップすると、ストリームを使用するクラスでシリアル ポートにアクセスできます。この名前空間には、シリアル ポートの制御を簡素化する列挙体が含まれています。

SerialPort オブジェクトを作成するには、My.Computer.Ports.OpenSerialPort メソッドを使用する方法が最も簡単です。

メモ：
.NET Framework のクラスでは、他の種類のポート (パラレル ポートや USB ポートなど) には直接アクセスできません。コンピュータのネットワークにアクセスする方法については、「.NET Framework でネットワーク操作に使用するクラス」を参照してください。

列挙体

この表は、シリアル ポートへのアクセスで使用する主要な列挙体の一覧とその説明です。

列挙体	説明
Handshake	**SerialPort** オブジェクトでシリアル ポート通信の確立に使用する制御プロトコルが定められています。
Parity	**SerialPort** オブジェクトのパリティ ビットが定められています。
SerialData	**SerialPort** オブジェクトのシリアル ポートで受信した文字の種類が定められています。
SerialError	**SerialPort** オブジェクトで発生するエラーが定められています。
SerialPinChange	**SerialPort** オブジェクトで発生した変更の種類が定められています。
StopBits	**SerialPort** オブジェクトで使用されているストップ ビットの数が定められています。

処理手順

この表は、シリアル ポートへのアクセスの主要なタスクの一覧とその説明です。

目的	参照項目
利用可能なシリアル ポートを表示する。	方法 : Visual Basic で利用可能なシリアル ポートを表示する
コンピュータのシリアル ポートに接続されているモデムをダイヤルする。	方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする
コンピュータのシリアル ポートに文字列を送信する。	方法 : Visual Basic でシリアル ポートに文字列を送信する
コンピュータのシリアル ポートから文字列を受信する。	方法 : Visual Basic でシリアル ポートから文字列を受信する

参照

関連項目

My.Computer.Ports オブジェクト

その他の技術情報

コンピュータのポートへのアクセス

Visual Basic への理解を深める

Visual Basic での .NET Framework のシリアル化

シリアル化とは、オブジェクトをバイトのストリームに変換して、メモリ、データベース、またはファイルに保持できるようにするプロセスのことです。.NET Framework に用意されている機能を使用して、こうしたオブジェクトを保存および再構築できます。

このセクションの内容

.NET Framework でのシリアル化の基礎

.NET Framework のクラスを使用したシリアル化の概要を示します。

.NET Framework でシリアル化に使用するクラス

シリアル化で使用する .NET Framework のクラスの一覧を示します。

関連するセクション

XML シリアル化の例

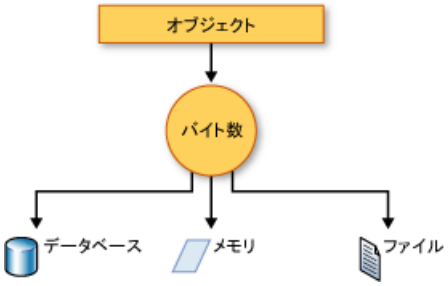
データセットなど、さまざまなオブジェクトの XML シリアル化の例を示します。

.NET Framework でのシリアル化の基礎

シリアル化とは、オブジェクトをバイトのストリームに変換して、メモリ、データベース、またはファイルに保持できるようにするプロセスのことです。その主な目的は、必要になったときに再構築できるように、オブジェクトの状態を保存することです。その逆のプロセスは逆シリアル化と呼ばれています。

シリアル化のしくみ

次の図は、シリアル化のプロセス全体を示します。



オブジェクトはストリームにシリアル化されます。ストリームの中には、データのみならず、オブジェクトの型についての情報も保持されます。たとえば、バージョン、カルチャ、アセンブリ名などです。そのストリームから、データベース、ファイル、またはメモリに格納できます。

シリアル化の用途

開発者は、シリアル化を使用して、オブジェクトの状態を保存し、必要に応じて再構築できます。それにより、オブジェクトのストレージやデータ交換が実現できます。シリアル化を通じて開発者が実行できる処理には、Web サービスによりリモート アプリケーションへオブジェクトを送信する処理、あるドメインから別のドメインへオブジェクトを渡す処理、XML 文字列としてファイアウォールを越えてオブジェクトを渡す処理、セキュリティまたはユーザー固有の情報をアプリケーション間で維持する処理などがあります。

オブジェクトをシリアル化できるようにする

オブジェクトをシリアル化するために必要なのは、シリアル化する対象のオブジェクト、シリアル化したオブジェクトを格納するストリーム、および Formatter です。オブジェクトのシリアル化と逆シリアル化に必要なクラスは、System.Runtime.Serialization に備わっています。

型のインスタンスがシリアル化できるということを示すためには、その型に SerializableAttribute 属性を適用します。シリアル化しようとした型が **SerializableAttribute** 属性を持たない場合には、SerializationException 例外がスローされます。

クラス内の特定のフィールドをシリアル化できないようにするには、NonSerializedAttribute 属性を適用します。シリアル化できる型の中に、ポインタやハンドルなど特定の環境に固有のデータ構造を格納するフィールドがあり、そのフィールドを別の環境で意味ある形に再構築できない場合は、そのフィールドはシリアル化できないようにすることをお勧めします。

シリアル化するクラスの中に、**SerializableAttribute** が指定されている他のクラスのオブジェクトへの参照が格納されている場合、それらのオブジェクトもシリアル化されます。

バイナリ シリアル化と XML シリアル化

シリアル化では、バイナリ シリアル化と XML シリアル化のいずれかを使用できます。バイナリ シリアル化では、読み取り専用のものも含めたすべてのメンバがシリアル化され、パフォーマンスは向上します。XML シリアル化では、コードの可読性が高まるのに加え、オブジェクトの共有や相互運用性を目的とした使用での柔軟性も高まります。

バイナリ シリアル化

バイナリ シリアル化とは、バイナリ エンコーディングを使用して、ストレージやソケットベースのネットワーク ストリームなどの用途のために、コンパクトにシリアル化する処理です。ファイアウォールを越えてデータを渡すのには適していませんが、データを格納するときのパフォーマンスは上です。

XML シリアル化

XML シリアル化とは、オブジェクトのパブリック フィールドやパブリック プロパティ、またはメソッドのパラメータや戻り値を、特定の XML スキーマ定義言語 (XSD: XML Schema Definition Language) ドキュメントに準拠する XML ストリームにシリアル化する処理です。XML シリアル化では、パブリックなプロパティおよびフィールドを持つ、厳密に型指定されたクラスが、XML に変換されます。System.Xml.Serialization には、XML のシリアル化および逆シリアル化に必要なクラスが備わっています。

クラスおよびクラス メンバに属性を適用すると、XmlSerializer がクラスのインスタンスをどのようにシリアル化または逆シリアル化するかを制御できます。詳細については、「属性を使用した XML シリアル化の制御」および「XML シリアル化を制御する属性」を参照してください。

次の表は、XML シリアル化に関連するタスクの一覧です。

目的	参照項目
オブジェクトをシリアル化する。	方法 : オブジェクトをシリアル化する
オブジェクトを逆シリアル化する。	方法 : オブジェクトを逆シリアル化する
クラスおよび XML スキーマ ドキュメントを生成する。	方法 : XML スキーマ定義ツールを使用してクラスおよび XML スキーマ ドキュメントを生成する
XML 要素および XML 属性名を修飾する。	方法 : XML 要素および XML 属性名を修飾する
XML ストリームの代替要素名を指定する。	方法 : XML ストリームの代替要素名を指定する
派生クラスのシリアル化を制御する。	方法 : 派生クラスのシリアル化を制御する

SOAP シリアル化

また、XML シリアル化を使用すると、オブジェクトを SOAP 仕様に準拠する XML ストリームにシリアル化することもできます。SOAP は、XML を使用してプロシージャ呼び出しを転送するために特別にデザインされた、XML に基づくプロトコルです。通常の XML シリアル化と同じく、属性を使用すると、XML Web サービスによって生成されるリテラル スタイルの SOAP メッセージを制御することもできます。詳細については、「XML Web サービスを使用した XML シリアル化」および「エンコード済み SOAP シリアル化を制御する属性」を参照してください。

次の表は、SOAP エンコード済み XML シリアル化に関連するタスクの一覧です。

目的	参照項目
オブジェクトを SOAP エンコード済み XML ストリームとしてシリアル化する。	方法 : オブジェクトを SOAP エンコード済み XML ストリームとしてシリアル化する
SOAP エンコード済み XML シリアル化をオーバーライドする。	方法 : エンコード済みの SOAP XML シリアル化をオーバーライドする

基本的なシリアル化とカスタムのシリアル化

シリアル化の実行方法には、基本的な方法とカスタムの方法の 2 種類があります。基本的なシリアル化では、.NET Framework を使用して、オブジェクトを自動でシリアル化します。

基本的なシリアル化

基本的なシリアル化で必要な唯一の要件は、対象のオブジェクトに **SerializableAttribute** 属性が適用されていることです。**NonSerializedAttribute** を使用すると、特定のフィールドをシリアル化の対象から除外できます。

基本的なシリアル化を使用するときには、オブジェクトのバージョン管理により問題が生じる可能性があり、その場合はカスタムのシリアル化の方が好ましいことがあります。基本的なシリアル化は、シリアル化を実行するのに最も簡単な方法ですが、そのプロセスをあまり制御できません。

シリアル化のカスタマイズ

カスタムのシリアル化では、シリアル化の対象のオブジェクトやシリアル化の方法を明確に指定できます。対象のクラスは、**SerializableAttribute** が適用され、ISerializable インターフェイスを実装している必要があります。

オブジェクトの逆シリアル化もカスタムの方法で行うためには、カスタム コンストラクタを使用する必要があります。

デザイナ シリアル化

デザイナ シリアル化とは、特別な形式のシリアル化で、通常は開発ツールで使用されるような種類のオブジェクト永続化が行われます。デザイナ シリアル化は、オブジェクト グラフをソース ファイルに変換するプロセスです。後でそのソース ファイルを使用して、オブジェクト グラフを復元できます。ソース ファイルには、コードやマークアップ、または SQL テーブル情報が含まれることもあります。詳細については、「デザイナのシリアル化の概要」を参照してください。

参照

処理手順

方法 : シリアル化されたデータをチャンクする

概念

XML シリアル化の例

.NET Framework でシリアル化に使用するクラス

.NET Framework でシリアル化に使用するクラス

このトピックでは、.NET Framework でのシリアル化で一般的に使用するクラスの一覧を示します。.NET Framework 2.0 ドキュメントで詳細な一覧を参照するには、「System.Runtime.Serialization」を参照してください。

シリアル化で使用する基本のクラス

次の表は、シリアル化で使用する主要なクラスおよび名前空間の一覧と説明です。

クラスまたは名前空間	説明
System.Runtime.Serialization	オブジェクトのシリアル化と逆シリアル化に必要なクラスが含まれています。
System.Xml.Serialization	XML のシリアル化と逆シリアル化に必要なクラスが含まれています。
SerializableAttribute	オブジェクトをシリアル化できるようにするために、オブジェクトに適用する必要があります。
NonSerializedAttribute	オブジェクトの特定のフィールドをシリアル化の対象から除外できます。

参照

概念

.NET Framework でのシリアル化の基礎

Visual Basic によるスマート デバイス向けの開発

Visual Studio 2005 では、Visual Basic によるスマート デバイス アプリケーションの開発がサポートされています。Pocket PC や Smartphone など、Windows CE .NET ベースのプラットフォーム向けにアプリケーションを開発するときに必要なツールとフレームワークが用意されています。

デバイス向けのアプリケーションの開発

デバイス向けのアプリケーションとしては、大きく分けて次の 2 種類を開発できます。

- Web サーバー上で実行され、ブラウザを備えたモバイル デバイスの範囲に別の形式で表示されるモバイル Web アプリケーション。詳細については、「ASP.NET モバイル Web ページの作成」を参照してください。
- デバイス自体で動作する、Windows CE ベースのリッチ クライアント アプリケーション。"スマート デバイス向けアプリケーション" という言葉が指すのは、通常はこちらの方法です。
- スマート デバイス向けの開発では、デスクトップ アプリケーションを開発するときと同じ Visual Studio 環境を使用しますが、いくつかの違いが生じます。たとえば次のような違いです。
- リモート デバイスに接続してデバッグするための追加的なツールを使用します。
- プロジェクト作成時にプロジェクトの種類とテンプレートを選択することに加え、アプリケーションの実行およびデバッグに使用するデバイスを選択する必要があります。選択できるデバイスは、開発用コンピュータに接続された物理デバイス、ネットワーク接続されたデバイス、開発用コンピュータ上で実行されるデバイス エミュレータのいずれかです。
- デバイス向けに開発するときには、クラスおよびそのメンバに違いがあります。クラスおよびそのメンバを利用できるかどうかを確認するには、ドキュメントを参照するか、IntelliSense を使用するか、またはプロジェクトがアクティブなときに Visual Studio オブジェクト ブラウザを使用します。詳細については、「.NET Compact Framework」を参照してください。

デバイス アプリケーションの作成

Visual Studio 2005 の強化された [新しいプロジェクト] ダイアログ ボックスは、Visual Studio .NET 2003 のスマート デバイス アプリケーション ウィザードに代わるものです。Visual Studio 2005 では、プロジェクトの種類やプロジェクト テンプレートに関するすべての項目を [新しいプロジェクト] ダイアログ ボックスで選択できます。詳細については、
「方法 : Visual Basic または Visual C# を使用してデバイス アプリケーションを作成する」を参照してください。

スマート デバイスに関連するタスクの一覧については、「スマート デバイス アプリケーション (Visual Basic での操作方法)」を参照してください。

データとデバイス

.NET Compact Framework では、デバイスに対して ADO.NET の充実した実装を提供し、DataSet クラスと Dataview クラスをサポートします。このサポートには、DataRelation クラスと Constraint クラス、および **DataSet** を定義および操作するその他のクラスが含まれます。.NET Compact Framework には、SQL Server .NET データ プロバイダも含まれます。詳細については、「データ アクセスと XML サポート」および「SQL Server Compact Edition と .NET Compact Framework」を参照してください。

ネットワーク

.NET Compact Framework では、HTTP、DNS、Web 要求と応答など、ネットワーク ソケットレベルの API とさらに上位レベルの抽象化をサポートしています。接続は、IrDA (Infrared Data Association) や、ソケット API を通じた TCP/IP トランスポートで提供されます。詳細については、「ネットワーキングと接続性」を参照してください。

Pocket PC 向けの開発

次の表は、Pocket PC に固有のタスクの一覧です。

目的	参照項目
DocumentList コントロールを使用してアプリケーションのファイル管理タスクを処理する。	方法 : DocumentList コントロールを使用する
アプリケーションでフォーム要素を使用する。	Pocket PC のフォーム スタイル
アプリケーションの HardwareButton コンポーネントを使用して物理的なハードウェア ボタンからアプリケーションをアクティブにする。	方法 : HardwareButton コンポーネントを使用する
アプリケーションで InputPanel コンポーネントを使用する。	方法 : InputPanel コンポーネントを使用する

アプリケーションでユーザー入力に使用する Pocket PC の入力メソッドを選択する。	方法 : Pocket PC の入力方法を設定する
アプリケーションで通知を送信および応答する。	方法 : 通知を送信する
移動キーを検出する。	方法: 移動キーを検出する

Smartphone 向けの開発

.NET Compact Framework は、Windows Mobile 2003 以降の Smartphone にインストールできます。

次の表は、Smartphone 開発に固有のタスクの一覧です。

目的	参照項目
アプリケーションで使用する Smartphone の入力メソッドを設定する。	方法 : Smartphone の入力モードを設定する
Back キーをオーバーライドする。	方法 : Smartphone の Back キーをオーバーライドする
Smartphone メニューを使用する。	方法 : Smartphone のメニューを使用する

参照

概念

.NET Compact Framework の使用方法のトピック

その他の技術情報

Pocket PC の開発と .NET Compact Framework
Smartphone の開発と .NET Compact Framework
組み込みの Windows CE 開発
.NET Compact Framework の製品情報

Visual Basic での高度なマルチスレッド処理

マルチスレッド アプリケーションでは、複数の異なるタスクを同時に実行できます。このセクションのトピックでは、各タスクが競合なしでスムーズに連係して動作するようにタスクを管理する方法を説明します。

このセクションの内容

スレッドの同期

マルチスレッド アプリケーションの同期をとる方法について説明します。

スレッド プール

タスクをキューに追加し、新しいスレッドが作成されたときにタスクを起動する方法について説明します。

スレッド タイマ

各スレッドのプロシージャを一定の間隔で実行する方法について説明します。

高度な同期化技法

マルチスレッド アプリケーションで複数のスレッドの同期をとるために待機ハンドルと監視オブジェクトを使用する方法について説明します。

関連項目

Visual Basic におけるマルチスレッド

Visual Basic アプリケーションでのマルチスレッド処理の概要を示します。

コンポーネントのマルチスレッド

コンポーネント プログラミングでマルチスレッドを使用する方法についてのトピックへのリンクが用意されています。

チュートリアル : Visual Basic による簡単なマルチスレッド コンポーネントの作成

マルチスレッド コンポーネントの作成方法を示します。

System.Threading

マルチスレッド プログラミングを実現する .NET Framework のクラスとインターフェイスを示します。

スレッド タイマ

System.Threading.Timer クラスは、タスクを別々のスレッドで定期的に実行するのに便利です。たとえば、スレッド タイマを使用すると、データベースのステータスと整合性をチェックしたり、重要なファイルをバックアップしたりできます。

スレッド タイマの例

2 秒ごとにタスクを起動し、フラグを使用して Dispose メソッドでタイマを停止する例を次に示します。この例は、ステータスを出力ウィンドウに出力します。そのため、コードをテストする前に、Ctrl + Alt + O キーを押して、このウィンドウを表示する必要があります。

VB

```
Class StateObjClass
    ' Used to hold parameters for calls to TimerTask
    Public SomeValue As Integer
    Public TimerReference As System.Threading.Timer
    Public TimerCanceled As Boolean
End Class

Sub RunTimer()
    Dim StateObj As New StateObjClass
    StateObj.TimerCanceled = False
    StateObj.SomeValue = 1
    Dim TimerDelegate As New Threading.TimerCallback(AddressOf TimerTask)
    ' Create a timer that calls a procedure every 2 seconds.
    ' Note: There is no Start method; the timer starts running as soon as
    ' the instance is created.
    Dim TimerItem As New System.Threading.Timer(TimerDelegate, StateObj, _
                                                2000, 2000)
    StateObj.TimerReference = TimerItem  ' Save a reference for Dispose.

    While StateObj.SomeValue < 10 ' Run for ten loops.
        System.Threading.Thread.Sleep(1000)  ' Wait one second.
    End While

    StateObj.TimerCanceled = True  ' Request Dispose of the timer object.
End Sub

Sub TimerTask(ByVal StateObj As Object)
    Dim State As StateObjClass = CType(StateObj, StateObjClass)
    ' Use the interlocked class to increment the counter variable.
    System.Threading.Interlocked.Increment(State.SomeValue)
    System.Diagnostics.Debug.WriteLine("Launched new thread  " & Now)
    If State.TimerCanceled Then    ' Dispose Requested.
        State.TimerReference.Dispose()
        System.Diagnostics.Debug.WriteLine("Done  " & Now)
    End If
End Sub
```

スレッド タイマは、コンソール アプリケーションを開発するときなど、System.Windows.Forms.Timer オブジェクトを使用できないときに特に有効です。

参照

関連項目

SyncLock ステートメント

System.Threading

概念

Visual Basic での高度なマルチスレッド処理

マルチスレッド アプリケーション

スレッド プール

スレッド プールは、タスクをキューに追加し、スレッドが作成されると自動的にタスクを起動するマルチスレッドの形式です。スレッド プールでは、実行するプロシージャのデリゲートを指定して System.Threading.ThreadPool.QueueUserWorkItem(System.Threading.WaitCallback) メソッドを呼び出すと、Visual Basic がスレッドを作成し、プロシージャを実行します。

スレッド プールの例

スレッド プールを使用していくつかのタスクを起動する方法を次の例に示します。

VB

```
Sub DoWork()
    ' Queue a task
    System.Threading.ThreadPool.QueueUserWorkItem( _
        New System.Threading.WaitCallback(AddressOf SomeLongTask))
    ' Queue another task
    System.Threading.ThreadPool.QueueUserWorkItem( _
        New System.Threading.WaitCallback(AddressOf AnotherLongTask))
End Sub
Sub SomeLongTask(ByVal state As Object)
    ' Insert code to perform a long task.
End Sub
Sub AnotherLongTask(ByVal state As Object)
    ' Insert code to perform another long task.
End Sub
```

スレッド プールは、各スレッドのプロパティを個別に設定せずに複数のタスクを起動するときに役立ちます。各スレッドは、既定のスタック サイズとプロパティで起動します。既定では、1 つのシステム プロセッサにつき、スレッド プールのスレッドを 25 個まで実行できます。この制限を超えてスレッドをキューに追加することもできますが、このスレッドは他のスレッドが終了するまで起動しません。

スレッド プールの利点の 1 つに、引数を状態オブジェクトでタスク プロシージャに渡すことができます。呼び出し先のプロシージャが複数の引数を必要とする場合、構造体またはクラスのインスタンスを **Object** データ型にキャストできます。

スレッド プールのパラメータと戻り値

スレッド プールのスレッドから値を返すのは簡単ではありません。スレッド プールのキューに追加できるプロシージャは **Sub** プロシージャだけのため、関数呼び出しから値を返すという基本的な方法は使用できません。パラメータと戻り値を用意する方法の 1 つとして、パラメータ、戻り値、メソッドをラッパー クラスにラップするやり方があります。その方法については、「マルチスレッド プロシージャのパラメータと戻り値」を参照してください。

この方法よりも簡単にパラメータと戻り値を用意するには、**QueueUserWorkItem** メソッドの省略可能な **ByVal** 状態オブジェクト変数を使用します。この変数を使用してクラスのインスタンスへの参照を渡すと、インスタンスのメンバをスレッド プールのスレッドで変更し、戻り値として使用できます。

値によって渡される変数の参照先オブジェクトを変更できるかどうかが明確でない場合もあります。これを変更できるのは、オブジェクト参照のみが値渡しされるからです。オブジェクト参照によって参照されるオブジェクトのメンバに変更を加えると、実際のクラス インスタンスに変更が適用されます。

状態オブジェクトの中の値を返すために構造体を使用することはできません。構造体は値型であるため、非同期プロセスで変更しても、元の構造体のメンバを変更することはできません。構造体は、戻り値を必要としないときに、パラメータを指定するために使用してください。

参照

関連項目

SyncLock ステートメント
QueueUserWorkItem
System.Threading

概念

Visual Basic での高度なマルチスレッド処理
マルチスレッド アプリケーション
スレッドの同期
フォームとコントロールでのマルチスレッド

スレッドの同期

スレッド処理されたアプリケーションを作成するときに、スレッドとプログラムの他の部分との同期をとることが必要な場合があります。同期化機能を使うと、マルチスレッド プログラミングの非構造的な性質と、同期的な処理の構造的な順次性を折衷できます。

使用できる同期化技法は次のとおりです。

- 特定の順序でタスクを実行する必要のあるときには、コードの実行順序を常に明示的に制御する。

  または

- 2 つのスレッドで同時に同じリソースを共有するときに発生する可能性のある問題を回避する。

たとえば、同期を使用すると、別のスレッドで実行中のデータ取得プロシージャが終了するまで、表示プロシージャを待たせることができます。

同期化技法

同期には、ポーリングと同期オブジェクトの使用の 2 とおりの方法があります。同期オブジェクトの詳細については、「高度な同期化技法」を参照してください。

ポーリング

ポーリングは、ループの中から非同期呼び出しのステータスを繰り返しチェックします。ポーリングは各種のスレッド プロパティのステータスを繰り返しチェックするため、リソースの浪費が多く、スレッドの管理には最も非効率な方法です。

たとえば、ポーリング時に IsAlive プロパティを使用すると、スレッドが終了したかどうかを確認できます。動作しているスレッドが必ずしも実行中であるとは限らないため、このプロパティの使用には注意が必要です。

スレッドの ThreadState プロパティを使用すると、スレッドのステータスに関する詳細な情報を取得できます。スレッドは複数の状態を持つことがあるため、**ThreadState** に格納されている値は System.Threading.ThreadState 列挙型の値の組み合わせの場合があります。したがって、ポーリングのときには、関連するスレッドの状態をすべて入念にチェックする必要があります。たとえば、状態が Running でない場合、スレッドは終了している可能性があります。その一方で、スレッドが中断またはスリープしている可能性もあります。

スレッドの終了の待機

System.Threading.Thread.Join メソッドは、別のタスクを開始する前にスレッドが完了したかどうかを判断するために役立ちます。**Join** メソッドは、スレッドが終了するまで指定された時間待機します。タイムアウトする前にスレッドが終了した場合、**Join** メソッドは **True** を返します。それ以外の場合は **False** を返します。

スレッドの状態を判断できるように、ポーリングはスレッドの実行順序を制御できる代わりに、マルチスレッドの利点をかなり犠牲にします。ポーリングは非常に非効率であるため、一般にはお勧めできません。**Join** メソッドを使用してスレッドを制御する方法の方が効率的です。**Join** を使用すると、スレッドが終了するか、または呼び出しがタイムアウトになるか (タイムアウトが指定されている場合) のいずれかまで、呼び出し元のプロシージャは待機します。"join" という名前は、新しいスレッドの作成が実行パスのフォークにあたるという意味を持ちます。**Join** を使用すると、個別の実行パスを単一のスレッドに再度マージできます。

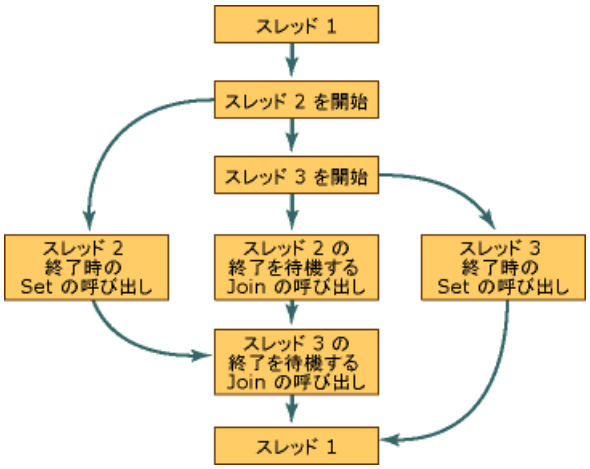


図 1 スレッド処理

Join が同期呼び出し、つまりブロックする呼び出しであるという点は明確です。**Join** メソッドまたは待機ハンドルの待機メソッドを呼び出すと、呼び出し元のプロシージャは停止し、スレッドからの終了通知を待ちます。

VB

```
Sub JoinThreads()
```

```
    Dim Thread1 As New System.Threading.Thread(AddressOf SomeTask)
    Thread1.Start()
    Thread1.Join()      ' Wait for the thread to finish.
    MsgBox("Thread is done")
End Sub
Sub SomeTask()
    ' Insert code to perform a task here.
End Sub
```

この方法はスレッドを簡単に制御でき、少数のスレッドを管理するのには便利ですが、多くのプロジェクトに使用するのは困難です。スレッドの同期をとるために使用できるいくつかの手法については、「高度な同期化技法」を参照してください。

参照

概念

Visual Basic での高度なマルチスレッド処理

マルチスレッド プロシージャのパラメータと戻り値

その他の技術情報

Visual Basic におけるマルチスレッド

高度な同期化技法

マルチスレッド アプリケーションは、複数のスレッドの同期をとるために待機ハンドルと監視オブジェクトを使用することがよくあります。以下のセクションでは、.NET Framework のクラスを使用してスレッドの同期をとる方法について説明します。ここで取り上げるのは、AutoResetEvent、Interlocked、ManualResetEvent、Monitor、Mutex、ReaderWriterLock、Timer、および WaitHandle の各クラスです。

待機ハンドル

待機ハンドルは、あるスレッドのステータスを別のスレッドに通知するオブジェクトです。スレッドは、待機ハンドルを使用して、リソースの排他アクセスが必要であることを他のスレッドに通知できます。通知を受けたスレッドは、待機ハンドルが使用されなくなるまで、このリソースの使用を待つ必要があります。待機ハンドルの状態には、"シグナル状態" と "非シグナル状態" の 2 種類があります。どのスレッドにも所有されていない待機ハンドルは、"シグナル状態" です。スレッドによって所有されている待機ハンドルは、"非シグナル状態" です。

スレッドは、WaitOne、WaitAny、WaitAll などのいずれかの待機メソッドを呼び出して、待機ハンドルの所有権を要求します。待機メソッドは、個別のスレッドに対する Join メソッドと同様に、呼び出しをブロックするメソッドです。

- 待機ハンドルを所有しているスレッドがない場合、待機メソッドを呼び出すとすぐに **True** が戻ります。待機ハンドルの状態は非シグナル状態に変わり、待機ハンドルを所有するスレッドは実行を継続します。
- スレッドが待機ハンドルの待機メソッドを呼び出すときに待機ハンドルが別のスレッドに所有されている場合、呼び出し元スレッドは一定時間待つか (タイムアウトを指定する場合)、待機ハンドルを所有しているスレッドが待機ハンドルを解放するまで待ちます (タイムアウトを指定しない場合)。タイムアウトを指定し、タイムアウトが経過する前に待機ハンドルが解放された場合、**True** が戻ります。それ以外の場合は、**False** が戻り、呼び出し元スレッドは実行を継続します。

待機ハンドルを所有するスレッドは、終了時または待機ハンドルが不要になったときに Set メソッドを呼び出します。他のスレッドは、Reset メソッドを呼び出すか、または **WaitOne**、**WaitAny**、または **WaitAll** を呼び出し、スレッドが **Set** を呼び出すのをきちんと待機することによって、待機ハンドルの状態を非シグナル状態にリセットできます。**AutoResetEvent** ハンドルは、単一の待機スレッドが解放された後で、システムによって自動的に非シグナル状態にリセットされます。待機中のスレッドがない場合、イベント オブジェクトはシグナル状態のままです。

Visual Basic で一般的に使用される待機ハンドルには、ミューテックス オブジェクト、**ManualResetEvent**、および **AutoResetEvent** の 3 種類があります。**ManualResetEvent** と **AutoResetEvent** は、同期イベントと呼ばれます。

ミューテックス オブジェクト

ミューテックス オブジェクトは、同時に 1 つのスレッドでしか所有できない同期オブジェクトです。"ミューテックス" (mutex) という名前は、ミューテックス オブジェクトの所有が相互に排他的 (mutually exclusive) であることに由来します。スレッドは、リソースを排他的にアクセスする必要があるときに、ミューテックス オブジェクトの所有を要求します。ミューテックス オブジェクトを同時に所有できるスレッドは 1 つだけであるため、他のスレッドはミューテックス オブジェクトの所有を待ってからリソースを使用する必要があります。

WaitOne メソッドを呼び出すと、呼び出し元スレッドはミューテックス オブジェクトの所有を待ちます。ミューテックス オブジェクトを所有しているスレッドが通常どおりに終了すると、ミューテックス オブジェクトはシグナル状態になり、待機中の次のスレッドが所有します。

同期イベント

同期イベントは、なんらかの処理が行われたこと、またはリソースが使用できることを他のスレッドに通知します。"イベント" という言葉が使われてはいますが、同期イベントは、Visual Basic の他のイベントとは異なり、その実体は待機ハンドルです。他の待機ハンドルと同様に、同期イベントの状態には、"シグナル状態" と "非シグナル状態" の 2 種類があります。

同期イベントの待機メソッドのいずれかを呼び出すスレッドは、別のイベントが **Set** メソッドを呼び出してイベントを通知するまで待つ必要があります。同期イベントには、**ManualResetEvent** と **AutoResetEvent** の 2 つのクラスがあります。

スレッドは、**Set** メソッドを使用して、**ManualResetEvent** インスタンスをシグナル状態に設定します。また、スレッドは、**Reset** メソッドを使用して、または待機中の **WaitOne** 呼び出しに制御が戻ったときに、**ManualResetEvent** インスタンスを非シグナル状態に設定します。

AutoResetEvent クラスのインスタンスも **Set** によってシグナル状態に設定できますが、イベントがシグナル状態になったことを示す通知を待機中のスレッドが受け取った時点で、すぐに自動的に非シグナル状態に戻ります。

監視オブジェクトと SyncLock

監視オブジェクトは、コードのブロックが他のスレッドで実行中のコードによって中断されずに実行されるように使用します。つまり、他のスレッドのコードは、同期されたコード ブロックのコードが終了するまで実行できません。

たとえば、データの読み出しと結果の表示を非同期で繰り返し行うプログラムがあるとします。プリエンプティブ マルチタスクを使用するオペレーティング システムでは、他のスレッドの実行に時間を割り当てるために、実行中のスレッドがオペレーティング システムによって中断されることがありま

す。同期を行わない場合、データの表示中にデータを表すオブジェクトが別のスレッドによって更新されると、データの表示が部分的に更新されることがあります。監視オブジェクトにより、コードのセクションの実行が中断されないことが保証されます。Visual Basic では、**SyncLock** ステートメントと **End SyncLock** ステートメントを使用して、監視オブジェクトを簡単に利用できます。Visual C# では、**Lock** キーワードを同様に使用します。

メモ：
オブジェクトへのアクセスは、アクセスするコードが同じオブジェクト インスタンスの **SyncLock** ブロック内にある場合にだけロックアウトされます。

SyncLock ステートメントの詳細については、「SyncLock ステートメント」を参照してください。

Interlocked クラス

Interlocked クラスのメソッドを使用すると、複数のスレッドが同じ値を同時に更新または比較しようとするときに発生する問題を回避できます。このクラスのメソッドによって、どのスレッドの値も安全にインクリメント、デクリメント、交換、比較することができます。別々のスレッドで実行中のプロシージャによって共有されている変数をインクリメントするために Increment メソッドを使用する方法を次の例に示します。

VB

```
Sub ThreadA(ByRef IntA As Integer)
    System.Threading.Interlocked.Increment(IntA)
End Sub

Sub ThreadB(ByRef IntA As Integer)
    System.Threading.Interlocked.Increment(IntA)
End Sub
```

ReaderWriter ロック

場合によっては、データを書き込むときだけリソースをロックし、データが更新されないときには複数のクライアントにデータの同時読み取りを許可する場合があります。**ReaderWriterLock** クラスを使用すると、スレッドがリソースを変更する間はリソースに排他アクセスを適用し、リソースを読み取るときには排他的でないアクセスを許可できます。ReaderWriter ロックは、データの更新が必要ないときでも、他のスレッドを待たせる方法として、排他ロックに代わる有効な手段です。

デッドロック

スレッドの同期は、マルチスレッド アプリケーションにとって非常に大切ですが、*deadlock* を生じさせてしまう危険性が常にあります。つまり、複数のスレッドが互いに待機しあって、アプリケーションが中断してしまう状態です。デッドロックをたとえて言えば、四方向一時停止の交差点で停止した車どうしが、相手の車を先に行かせるよう互いに譲り合い、どちらも動けなくなっている状態と同じです。デッドロックを防ぐことは重要です。その鍵となるのは、綿密なプランです。コーディングを始める前にマルチスレッド アプリケーションを図式化すると、デッドロック状態を予想できることがよくあります。

参照

関連項目

SyncLock ステートメント
System.Threading

概念

Visual Basic での高度なマルチスレッド処理
スレッドの同期
スレッド状態
マルチスレッド アプリケーション

その他の技術情報

コンポーネントのマルチスレッド

Visual Basic での Windows フォーム アプリケーションの概念

Windows フォーム アプリケーションは、ユーザーのコンピュータで実行され、情報を表示し、ユーザーに対して入力を要求し、ネットワーク経由でリモート コンピュータと通信するクライアント アプリケーションです。ある程度の規模のアプリケーションを開発するときには、.NET Framework およびその中に用意されているクラスを調べることが、場合によっては必要です。このセクションでは、.NET Framework クラスの概要と、Windows フォーム アプリケーションに関連する概念を示します。

このセクションの内容

Windows フォーム アプリケーションの基礎

.NET Framework のクラスを使用した Windows フォーム アプリケーションの作成の概要を示します。

.NET Framework のフォームで使用するクラス

Windows フォーム アプリケーションで使用する .NET Framework のクラスの一覧を示します。

関連項目

Windows ベース アプリケーションの概要

Visual Studio 2005 での Windows フォーム アプリケーションの作成の詳細について説明します。

Windows フォームの概要

スマートクライアント アプリケーションのメリットと、Windows フォーム プログラミングの主な機能について説明します。

参照

関連項目

Windows フォームの概要

概念

Visual Basic アプリケーション モデルの概要

その他の技術情報

Visual Basic への理解を深める

Windows フォーム アプリケーションの基礎

Visual Basic の重要な要素に、ユーザーのコンピュータ上でローカルに動作する Windows フォーム アプリケーションを作成する機能があります。Visual Studio 2005 では、Windows フォームを使用するアプリケーションおよびユーザー インターフェイスを作成できます。Windows フォーム アプリケーションは、System.Windows.Forms 名前空間のクラスを基に作成します。

Windows フォーム アプリケーションのデザイン

Visual Studio では、Windows フォーム アプリケーションと Windows サービス アプリケーションを作成できます。詳細については、次のトピックを参照してください。

- 「Windows フォームについて」。Windows フォームを作成およびプログラミングする方法についての情報です。
- 「Windows フォームのチュートリアルおよび方法のトピック」。Windows フォームをベースとした、一般的に作成される Windows フォーム アプリケーションについて、詳細な開発手順を提供するトピックの一覧を示します。
- 「Windows フォーム コントロール」。Windows フォーム コントロールの使用について詳細に説明したトピックを集めたものです。
- 「Windows サービス アプリケーション」。Windows サービスの作成方法を説明するトピックの一覧を示します。

充実した対話形式のユーザー インターフェイスの構築

Windows フォームは、.NET Framework のスマート クライアント コンポーネントであり、ファイル システムへの読み書きなど、共通のアプリケーション タスクを実現するマネージ ライブラリの集まりです。Visual Studio などの開発環境を使用して、情報の表示、ユーザーに対する入力要求、ネットワーク経由でのリモート コンピュータとの通信を実行する Windows フォーム アプリケーションを作成できます。

Windows フォームでいうフォームとは、ユーザーに対して情報を表示する外観部分の土台のことです。Windows フォーム アプリケーションの作成では、フォーム上にコントロールを配置し、ユーザー アクション (マウスのクリックやキーの押下など) に対する応答を開発するという方法が一般的です。コントロールとは、データの表示やデータ入力の受け付けを行う、個別のユーザー インターフェイス (UI) 要素です。

イベント

フォームまたはその上のコントロールに対してユーザーが何らかの操作を行うと、イベントが生成されます。アプリケーションでは、こうしたイベントに応答するためのコードを使用して、イベントの発生時に処理を行います。詳細については、「Windows フォーム内でのイベント ハンドラの作成」を参照してください。

コントロール

Windows フォームにはさまざまなコントロールが用意されており、フォーム上に配置できます。たとえば、テキスト ボックス、ボタン、ドロップダウン ボックス、オプション ボタンを表示するコントロールがあるのに加え、Web ページを表示するコントロールもあります。フォーム上で使用できるすべてのコントロールの一覧については、「Windows フォームで使用するコントロール」を参照してください。Windows フォームでは、既存のコントロールでニーズに対応できない場合には、UserControl クラスを使用して独自のカスタム コントロールを作成することもできます。

Windows フォームには、Microsoft Office のようなハイエンド アプリケーションの機能をエミュレートする、豊富な UI コントロールが用意されています。ToolStrip コントロールおよび MenuStrip コントロールを使用すると、テキストおよびイメージを含むツール バーやメニューの作成、サブメニューの表示、および他のコントロール (テキスト ボックスやコンボ ボックスなど) のホストが可能です。

Visual Studio のドラッグ アンド ドロップ フォーム デザイナを使用すると、Windows フォーム アプリケーションを簡単に作成できます。コントロールをポインタで選択し、フォーム上の目的の位置に配置するだけです。このデザイナには、グリッド線や "スナップ線" などのツールが備わっており、コントロールの配置で苦労する必要がありません。また、Visual Studio を使用する場合でも、コマンド ラインでコンパイルする場合でも、FlowLayoutPanel、TableLayoutPanel、および SplitContainer の各コントロールを使用して、最小限の時間と労力で高度なフォーム レイアウトを作成できます。

カスタムの UI 要素

独自のカスタム UI 要素を作成する必要がある場合には、線、円、およびその他の形状をフォーム上に直接描画するために必要なすべてのクラスが、System.Drawing 名前空間に含まれています。

これらの機能の使用手順の詳細については、以下のヘルプ トピックを参照してください。

目的	参照項目
Visual Studio で Windows フォーム アプリケーションを新規作成する。	チュートリアル : 簡単な Windows フォームの作成
フォーム上でコントロールを使用する。	方法 : Windows フォームにコントロールを追加する

フォームおよびそのコントロールからのイベントを処理する。	方法 : デザイナを使用してイベント ハンドラを作成する
ToolStrip コントロールを使用する。	方法 : デザイナを使用して標準アイテムで基本的な Windows フォーム ToolStrip を作成する
System.Drawing でグラフィックスを作成する。	グラフィックス プログラミングについて
カスタム コントロールを作成する。	方法 : UserControl クラスを継承する

データの表示と操作

多くのアプリケーションでは、データベース、XML ファイル、XML Web サービス、およびその他のデータ ソースのデータを表示する必要があります。Windows フォームには、DataGridView コントロールという柔軟なコントロールが用意されています。このコントロールでは、行と列から成る従来型の表形式のデータを表示でき、データの各項目を、それぞれ別個のセルに配置できます。**DataGridView** にはさまざまな機能があります。個々のセルの表示形式のカスタマイズ、任意の行や列のロック、セル内での複合コントロールの表示などです。

Windows フォームのスマート クライアントでは、ネットワーク経由でのデータ ソースへの接続も簡単です。Visual Studio 2005 および .NET Framework 2.0 の Windows フォームの新機能である BindingSource コンポーネントは、データ ソースへの接続を表し、コントロールへのデータのバインド、前のレコードや次のレコードへの移動、レコードの編集、元のソースへの変更の保存などを実行するためのメソッドを公開します。BindingNavigator コントロールは、**BindingSource** コンポーネントの簡単なインターフェイスとなるもので、ユーザーがレコード間を移動できます。

データ バインド コントロール

[データ ソース] ウィンドウを使用して、データ バインド コントロールを簡単に作成できます。このウィンドウには、データベース、Web サービス、プロジェクト内のオブジェクトなどのデータ ソースが表示されます。このウィンドウからプロジェクト内のフォームに項目をドラッグすると、データ バインド コントロールを作成できます。また、[データ ソース] ウィンドウから既存のコントロールにオブジェクトをドラッグするという方法でも、既存のコントロールをデータにバインドできます。

設定

Windows フォームでは、設定もデータ バインディングの一種として管理できます。大半のスマート クライアント アプリケーションでは、実行時の状態 (前回のフォームのサイズなど) についての情報や、ユーザー設定のデータ (保存するファイルの既定の場所など) を保持しておく必要があります。アプリケーション設定機能では、両方の種類の設定をクライアント コンピュータに格納するための簡単な方法が用意されており、これらの要件に対応できます。これらの設定は、Visual Studio またはコード エディタを使用していったん定義すると、XML として保持され、実行時には自動でメモリに読み込まれます。

これらの機能の使用手順の詳細については、以下のヘルプ トピックを参照してください。

目的	参照項目
BindingSource コンポーネントを使用する。	方法 : デザイナを使用して Windows フォーム コントロールを BindingSource コンポーネントにバインドする
ADO.NET データ ソースを操作する。	方法 : Windows フォーム BindingSource コンポーネントで ADO.NET データを並べ替える/フィルタ処理する
[データ ソース] ウィンドウを使用する。	チュートリアル : Windows アプリケーションのフォームでのデータの表示
アプリケーション設定を使用する。	方法 : デザイナを使用してアプリケーション設定を作成する

クライアント コンピュータへのアプリケーションの配置

アプリケーションを作成したら、ユーザーに送る必要があります。各自のクライアント コンピュータにインストールして実行してもらうためです。ClickOnce テクノロジを使用すると、Visual Studio からの数回のクリック操作だけでアプリケーションを配置でき、Web 上でのアプリケーションの置き場所を示す URL をユーザーに伝えることができます。ClickOnce は、アプリケーション内のすべての要素と依存関係を管理し、アプリケーションがクライアント コンピュータに適切にインストールされるようにします。

ClickOnce アプリケーションは、ユーザーがネットワークに接続されている場合のみ実行できるように構成したり、オンラインとオフラインの両方で実

行できるように構成したりできます。オフラインでの操作もサポートするようにアプリケーションに指定した場合には、ClickOnce は、ユーザーの [スタート] メニューに当該アプリケーションへのリンクを追加します。ユーザーが URL を使用しなくても開けるようにするためです。

アプリケーションを更新するときには、新しい配置マニフェストと、アプリケーションの新しいコピーを Web サーバーに対して発行します。ClickOnce は、更新が利用可能であることを検出し、ユーザーのインストール内容をアップグレードします。古いアセンブリを更新するためのカスタム プログラミングは不要です。

ClickOnce の詳細については、「ClickOnce の配置の概要」を参照してください。これらの機能の使用手順の詳細については、以下のヘルプ トピックを参照してください。

目的	参照項目
ClickOnce でアプリケーションを配置する。	方法 : ClickOnce アプリケーションを発行する チュートリアル : ClickOnce アプリケーションを手動で配置する
ClickOnce 配置を更新する。	方法 : ClickOnce アプリケーションの更新を管理する
アプリケーションの更新を確認する	方法 : ClickOnce アプリケーションの更新の有無をチェックする
ClickOnce でセキュリティを管理する。	方法 : ClickOnce のセキュリティ設定を有効にする

他のコントロールと機能

Windows フォームには、共通のタスクをすばやく簡単に実装するための機能が、他にもたくさんあります。ダイアログ ボックスの作成、印刷、ヘルプおよびドキュメントの追加、複数言語へのアプリケーションのローカライズのサポートなどです。加えて、Windows フォームは、.NET Framework の堅牢なセキュリティ システムを基盤としており、顧客にリリースするアプリケーションの安全性を高めることができます。

これらの機能の使用手順の詳細については、以下のヘルプ トピックを参照してください。

目的	参照項目
フォームの内容を印刷する。	方法 : Windows フォームでグラフィックスを印刷する 方法 : Windows フォームで複数ページのテキスト ファイルを印刷する
Windows フォーム アプリケーションをグローバル化する。	チュートリアル : Windows フォームのローカリゼーション
Windows フォームのセキュリティについて理解を深める。	Windows フォームのセキュリティの概要

参照

関連項目

Windows フォームの概要
My.Forms オブジェクト
My.Application オブジェクト

概念

Windows ベース アプリケーションの概要
.NET Framework のフォームで使用するクラス

.NET Framework のフォームで使用するクラス

以下の各表は、.NET Framework の Windows フォーム アプリケーションで一般的に使用するクラスの一覧です。基本クラス、コントロール クラス、コンポーネント クラス、およびダイアログ ボックス クラスに分類されています。

.NET Framework ドキュメントで詳細な一覧を参照するには、System.Windows.Forms 名前空間に関するトピックを参照してください。

基本クラス

次の表は、**System.Windows.Forms** 名前空間の主要な Windows フォーム クラスの一覧とその説明です。

クラス	説明
Control	コントロール (ビジュアルな表示を持つコンポーネント) の基本クラスを定義します。**Form** 上に表示されるすべてのコントロールに共通する基本機能を提供します。
Form	通常のウィンドウ、モードレス ウィンドウ、ダイアログ ボックス、マルチ ドキュメント インターフェイス (MDI) クライアント、または MDI 親ウィンドウを表します。フォームはアプリケーションのユーザー インターフェイスを構成します。
UserControl	他のコントロールの作成、または他の複数のコントロールを組み合わせたカスタム コントロールの作成に使用できる空のコントロールを提供します。

コントロール クラス

次の表は、**System.Windows.Forms** 名前空間の主なコントロール クラスの一覧とその説明です。これらのコントロールを使用して、多彩なユーザー インターフェイスを作成できます。

クラス	説明
TextBox	データ入力コントロールです。Windows のテキスト ボックス コントロールを表します。
ComboBox	データ入力コントロールです。Windows のコンボ ボックス コントロールを表します。
Label	データ表示コントロールです。Windows の標準のラベルを表します。
ListView	データ表示コントロールです。Windows のリスト ビュー コントロールを表します。4 種類のビューのいずれかを使用して、項目のコレクションを表示します。
Button	コマンド ボタン コントロールです。Windows のボタン コントロールを表します。
ToolStrip	コマンド ボタン コントロールです。Windows のツール バー オブジェクト用のコンテナを提供します。
PropertyGrid	デザイナ コントロールです。オブジェクトのプロパティを参照するためのユーザー インターフェイスを提供します。

コンポーネントのクラス

System.Windows.Forms 名前空間には、コントロール以外のクラスも用意されています。**Control** クラスからは派生されていないものの、Windows ベースのアプリケーションにビジュアル機能を提供するクラスです。次の表は、利用可能なコンポーネント クラスの一部です。

クラス	説明
ToolTip	データ表示コンポーネントです。ユーザーがポインタをコントロール上に移動したときに、コントロールの用途についての簡単な説明を表示する、小さな四角形のポップアップ ウィンドウを表します。
ErrorProvider	データ表示コンポーネントです。フォーム上のコントロールに関連するエラーがあることを示すためのユーザー インターフェイスを提供します。
ToolStripDropDownMenu	メニュー コンポーネントです。ContextMenuStrip コントロールの基本機能を提供します。

ContextMenuStrip	メニュー コンポーネントです。ショートカット メニューを表します。
Help	ヘルプ コンポーネントです。HTML ヘルプ 1.0 エンジンをカプセル化します。
HelpProvider	ヘルプ コンポーネントです。コントロールのポップアップ ヘルプまたはオンライン ヘルプを提供します。

ダイアログ ボックス クラス

Windows には、いくつかのコモン ダイアログ ボックスが用意されています。これらを使用すると、ファイルのオープンや保存、フォントやテキストの色の操作、印刷などのタスクを実行するときに、アプリケーション間で統一のとれたユーザー インターフェイスを実現できます。次の表は、ストリームによるファイルの読み取りと書き込みで一般的に使用する主なクラスの一覧です。

クラス	説明
OpenFileDialog	開くファイルや保存するファイルの名前をユーザーが参照および入力できるようにするためのダイアログ ボックスを表示する機能を提供します。
SaveFileDialog	開くファイルや保存するファイルの名前をユーザーが参照および入力できるようにするためのダイアログ ボックスを表示する機能を提供します。
FontDialog	アプリケーションで使用するフォントの各種要素を変更するためのダイアログ ボックスを表示します。
PageSetupDialog	文書印刷の各種項目をユーザーが制御できるようにするためのダイアログ ボックスを表示します。
PrintDialog	文書印刷の各種項目をユーザーが制御できるようにするためのダイアログ ボックスを表示します。
MessageBox	ユーザーに対してデータを表示したり、ユーザーからデータを取得したりできるメッセージ ボックスを表示します。

参照

関連項目

Windows フォームの概要
System.Windows.Forms
System.Drawing.Printing

概念

Visual Basic での Windows フォーム アプリケーションの概念
Windows フォーム アプリケーションの基礎
Windows ベース アプリケーションの概要

Visual Studio アプリケーションへの印刷可能なレポートの追加

Visual Studio では、Visual Basic アプリケーションにリッチ データ レポート機能を追加するのに役立つさまざまなレポート ソリューションが用意されています。レポート ビューア コントロール、Crystal Reports、または SQL Server Reporting Services を使用して、レポートを作成したり追加したりできます。

メモ：
SQL Server Reporting Services は、Visual Studio 2005 ではなく SQL Server 2005 の一部です。Reporting Services は、SQL Server 2005 をインストールしていなければ、システムにインストールされていません。

このページでは、レポート ビューア コントロールと Reporting Services について説明します。Crystal Reports の詳細については、「Crystal Reports の概要」を参照してください。

Visual Basic アプリケーションの Microsoft レポート テクノロジの概要

アプリケーションで Microsoft レポート テクノロジを使用するには、以下のいずれかの方法を選択します。

- レポート ビューア コントロールの 1 つ以上のインスタンスを Visual Basic Windows アプリケーションに追加します。
- レポート サーバー Web サービスを呼び出すことにより、プログラムによって SQL Server Reporting Services を統合します。
- レポート ビューア コントロールと Microsoft SQL Server 2005 Reporting Services を併用します。レポート ビューア コントロールはレポート ビューアとして、レポート サーバーはレポート プロセッサとして使用します。レポート サーバーと レポート ビューア コントロールを併用する場合は、SQL Server 2005 バージョンの Reporting Services を使用する必要があります。

レポート ビューア コントロールの使用

レポート機能を Visual Basic Windows アプリケーションに組み込む一番簡単な方法は、アプリケーションのフォームにレポート ビューア コントロールを追加することです。このコントロールは、レポート処理機能を直接アプリケーションに追加し、統合レポート デザイナを提供して、ADO.NET データ オブジェクトのデータを使用してレポートを作成できるようにします。多機能 API により、プログラムによってコントロールおよびレポートにアクセスして、ランタイム機能を構成できます。

レポート ビューアには、単一の無償配布データ コントロール内にレポート処理機能とレポート表示機能が組み込まれています。次のようなレポート機能が必要な場合は、レポート ビューア コントロールを選択します。

- クライアント アプリケーションでのレポート処理。処理されたレポートは、コントロールが提供するビュー領域に表示されます。
- ADO.NET データ テーブルへのデータ バインディング。コントロールに渡される DataTable インスタンスを使用するレポートを作成できます。ビジネス オブジェクトに直接バインドすることもできます。
- アプリケーションに組み込むことができる再配布可能なコントロール。
- ページの移動、印刷、検索、エクスポート形式などのランタイム機能。レポート ビューアのツール バーでこれらの操作がサポートされています。

レポート ビューア コントロールを使用する場合は、レポート ビューア コントロールを、Visual Studio のツールボックスの [データ] セクションから Visual Basic Windows アプリケーションのフォームにドラッグできます。レポート ビューアの詳細については、「ReportViewer Controls (Visual Studio)」および「Samples and Walkthroughs」を参照してください。

Visual Studio でのレポート ビューア コントロールに対応するレポートの作成

レポート ビューアで実行するレポートを作成するには、プロジェクトに レポート テンプレートを追加します。Visual Studio により、クライアント レポート定義ファイル (.rdlc) が作成され、プロジェクトにファイルが追加され、Visual Studio ワークスペース上に統合レポート デザイナが開きます。

Visual Studio レポート デザイナは [データ ソース] ウィンドウと統合されています。[データ ソース] ウィンドウからレポートにフィールドをドラッグすると、レポート デザイナはデータ ソースについてのメタデータをレポート定義ファイルにコピーします。このメタデータは、レポート ビューア コントロールがデータ バインディング コードを自動的に生成するために使用します。

Visual Studio レポート デザイナにプレビュー機能はありません。レポートをプレビューするには、アプリケーションを実行し、そのアプリケーションに埋め込まれているレポートをプレビューします。

アプリケーションに基本的なレポート機能を追加するには

1. レポート ビューア コントロールを、ツールボックスの [データ] タブからフォーム上にドラッグします。

2. [プロジェクト] メニューの [新しい項目の追加] をクリックします。[新しい項目の追加] ダイアログ ボックスの [レポート] アイコンを選択し、[追加] をクリックします。

   開発環境にレポート デザイナが開き、レポート (.rdlc) ファイルがプロジェクトに追加されます。

3. レポート アイテムをツールボックスからレポート レイアウトにドラッグし、必要に応じて配置替えします。

4. フィールドを [データ ソース] ウィンドウからレポート レイアウト上のレポート アイテムにドラッグします。

レポート ビューアとレポート デザイナを使用する際の次の手順

目的	参照項目
レポート ビューア コントロールを追加および構成する	Adding and Configuring the ReportViewer Controls
ローカル レポートを作成およびプレビューする	Creating Client Report Definition (.rdlc) Files
レポート ビューア コントロールを使用したレポートにデータ ソースを追加する	Creating Data Sources for a ReportViewer Report
レポートのレイアウトを定義する	Defining a Report Layout
レポート ビューア コントロールを使用したレポートに Visual Basic 式を使用する	Using Expressions in a ReportViewer Report
レポートにカスタム コードを挿入する	Adding Custom Code to a ReportViewer Report
レポートとレポート ビューア コントロールをアプリケーションの一部として配布する	Deploying Reports and ReportViewer Controls
レポートを印刷する	Printing Reports from ReportViewer

さまざまなシナリオでのレポートの作成方法とコントロールの構成方法の手順の詳細については、「Samples and Walkthroughs」を参照してください。

Visual Basic アプリケーションでの Reporting Services の使用

Reporting Services は SQL Server に組み込まれているサーバー ベースのレポート テクノロジです。Reporting Services には、レポート ビューア コントロールにはない機能が含まれています。次のような機能が必要な場合は、Reporting Services を選択します。

- 複雑なレポートや長期のレポート、および大容量レポート アクティビティに対して優れたパフォーマンスを提供する、スケール アウトの配置とサーバー側のレポート処理。
- カスタム レポート制御とリッチ レンダリング出力形式をサポートする、統合されたデータ処理とレポート処理。
- レポートの実行時間を正確に指定できるレポート スケジュール処理。
- 電子メールを使用した、またはファイル共有場所への、サブスクライバ ベースのレポート配布。
- ビジネス ユーザーが適宜レポートを作成できるアドホック レポート。
- カスタマイズされたレポート出力を動的な受信者リストにルーティングするデータドリブン サブスクリプション。
- データ処理、レポート配信、カスタム認証、およびレポート レンダリング用のカスタム拡張機能。

レポート サーバーは Web サービスとして実装されます。レポートおよび他のメタデータにアクセスするには、アプリケーション コードに Web サービスへの呼び出しが含まれている必要があります。Web サービスにより、プログラムによってレポート サーバー インスタンスにアクセスできます。

Reporting Services は Web ベースのレポート テクノロジなので、既定のビューアでは、レポートは HTML 形式で表示されます。既定のレポート表示形式として HTML 形式を使用しない場合は、アプリケーションに対してカスタム レポート ビューアを記述する必要があります。

Reporting Services の詳細については、SQL Server 2005 Books Online の「SQL Server Reporting Services」と「Reporting Services のチュートリアル」を参照してください。

Visual Studio での Reporting Services に対応するレポートの作成

レポート サーバーを実行するレポートを作成するには、SQL Server 2005 に組み込まれている Business Intelligence Development Studio を介して Visual Studio 上にレポート定義 (.rdl) ファイルを作成します。

メモ：

SQL Server Reporting Services と Business Intelligence Development Studio を使用するには、SQL Server 2005 がインストールされている必要があります。

Business Intelligence Development Studio は、SQL Server コンポーネント固有のプロジェクト テンプレートを追加します。レポートを作成するには、レポート サーバー プロジェクト テンプレートまたは レポート サーバー プロジェクト ウィザード テンプレートを選択します。SQL Server、Oracle、Analysis Services、XML、SQL Server Integration Services などさまざまなデータ ソース タイプに対するデータ ソース接続およびクエリを指定できます。[データ]、[レイアウト]、および [プレビュー] の各タブにより、データの定義、レポート レイアウトの作成、およびレポートのプレビューを同じワークスペースから実行できます。

コントロールまたはレポート サーバー用に作成するレポート定義は、どのテクノロジでも再利用できます。詳細については、「Converting RDL and RDLC Files」を参照してください。

レポート サーバー上で実行するレポートを作成するには

1. [ファイル] メニューの [新しいプロジェクト] をクリックします。

   [新しいプロジェクト] ダイアログ ボックスが表示されます。

2. [プロジェクトの種類] ペインで、[ビジネス インテリジェンス プロジェクト] をクリックします。

3. [テンプレート] ペインで、[レポート サーバー プロジェクト] または [レポート サーバー プロジェクト ウィザード] を選択します。

レポート ビューア コントロールと SQL Server Reporting Services の併用

レポート ビューア コントロールと SQL Server 2005 Reporting Services を同じアプリケーションで併用できます。

- レポート ビューア コントロールは、アプリケーション上でレポートを表示するためのビューアを提供します。
- Reporting Services は、レポートを提供し、リモート サーバー上のすべての処理を実行します。

リモートの Reporting Services レポート サーバー上で格納および処理されるレポートを表示するようにレポート ビューア コントロールを構成できます。この種類の構成は、リモート処理モードと呼ばれます。リモート処理モードでは、コントロールはリモート レポート サーバー上に格納されるレポートを要求します。レポート サーバーが、レポート処理、データ処理、およびレポート レンダリングをすべて実行します。処理が終了し、レンダリングされたレポートがコントロールに返され、ビュー領域に表示されます。

レポート サーバー上で実行するレポートは、他のエクスポート形式をサポートし、異なるレポート パラメータ化を実装します。また、レポート サーバーによってサポートされるデータ ソースの種類を使用し、レポート サーバー上のロール ベースの承認モデルを介してアクセスされます。

リモート処理モデルを使用するには、レポート ビューア コントロールを構成するときにサーバー レポートの URL とパスを指定します。コントロールの構成およびサーバー レポートとクライアント レポートとの比較の詳細については、「Configuring ReportViewer for Remote Processing」を参照してください。

参照

関連項目

Microsoft.Reporting.WebForms
Microsoft.Reporting.WinForms

その他の技術情報

ReportViewer Web Server and Windows Forms Controls
Creating Client Report Definition (.rdlc) Files
Reporting Services プログラミングの概要
レポートの作成、管理、および配信の概要
レポート レイアウトでのデータの操作
Reporting Services のデータに対する作業

Visual Basic でのプロジェクトのカスタマイズと My の拡張

プロジェクト テンプレートをカスタマイズして、追加的な **My** オブジェクトを提供できます。こうすると、自分の作成したオブジェクトを他の開発者が見つけたり使用したりしやすくなります。

通常、この種のカスタマイズが特に役立つのはアドインです。

このセクションの内容

Visual Basic アプリケーション モデルの拡張

WindowsFormsApplicationBase クラスのメンバをオーバーライドすることによって、アプリケーション モデルに対して独自の拡張機能を指定する方法を説明します。

My で利用可能なオブジェクトのカスタマイズ

プロジェクトの _MYTYPE 条件付きコンパイル定数を設定することによって、有効にする **My** オブジェクトを制御する方法を説明します。

関連するセクション

My による開発

プロジェクトの種類に応じて、どの **My** オブジェクトが既定で利用可能になっているかを説明します。

Visual Basic アプリケーション モデルの概要

Windows フォーム アプリケーションの動作を制御するための Visual Basic のモデルを説明します。

プロジェクトの種類に応じた My の機能

プロジェクトの種類に応じて、どの **My** オブジェクトが既定で利用可能になっているかを説明します。

条件付きコンパイルの概要

条件付きコンパイルを使用して、コードの特定のセクションだけを選択してコンパイルする方法を説明します。

My.Application オブジェクト

現在のアプリケーションに関するプロパティ、メソッド、およびイベントを提供する **My** オブジェクトについて説明します。

参照

その他の技術情報

Visual Basic への理解を深める

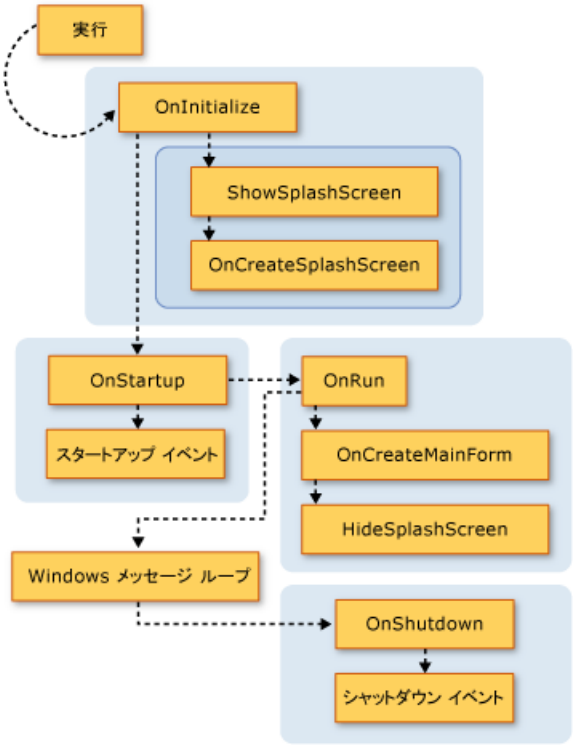
Visual Basic アプリケーション モデルの拡張

WindowsFormsApplicationBase クラスの **Overridable** メンバをオーバーライドすることによって、アプリケーション モデルに対して機能を追加できます。このテクニックを使用すると、アプリケーション モデルの動作をカスタマイズしたり、アプリケーションの起動時および終了時に独自のメソッドの呼び出しを追加したりできます。

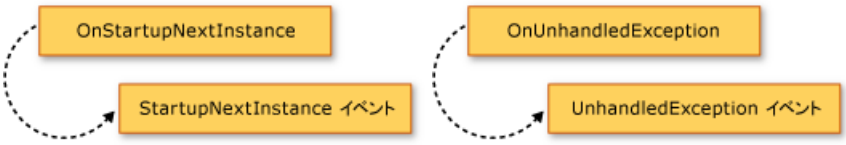
アプリケーション モデルの視覚的な概要

このセクションでは、Visual Basic アプリケーション モデルにおける関数呼び出しのシーケンスを視覚的に示します。各関数の目的については、この後のセクションで詳しく説明します。

次の図は、一般的な Visual Basic Windows フォーム アプリケーションにおけるアプリケーション モデルの呼び出しシーケンスを示しています。シーケンスは `Sub Main` プロシージャによる Run メソッドの呼び出しで始まります。



Visual Basic アプリケーション モデルでは、StartupNextInstance イベントと UnhandledException イベントも使用できます。次の図は、これらのイベントが発生するしくみを示しています。



基本メソッドのオーバーライド

Application のメソッドが実行される順序は、My.Application.Run メソッドにより定義されます。既定では、Windows フォーム アプリケーションの `Sub Main` プロシージャから **Run** メソッドが呼び出されます。

アプリケーションが通常のアプリケーション (複数インスタンス アプリケーション) の場合、または単一インスタンス アプリケーションの最初のインスタンスの場合には、**Run** メソッドは一連の **Overridable** メソッドを次の順序で実行します。

1. OnInitialize。既定では、このメソッドは、visual スタイル、テキスト表示スタイル、およびメイン アプリケーション スレッドの現在のプリンシパル (Windows 認証を使用するアプリケーションの場合) を設定し、コマンド ライン引数に `/nosplash` と `-nosplash` のいずれも指定されていない場合には **ShowSplashScreen** を呼び出します。

   この関数が **False** を返した場合、アプリケーションの起動処理はキャンセルされます。これは、状況に応じてアプリケーションの実行が不要な場合に役立ちます。

   OnInitialize メソッドは以下のメソッドを呼び出します。

   a. ShowSplashScreen。アプリケーションにスプラッシュ スクリーンが定義されているかどうかを判断し、定義されている場合はそのスプ

ラッシュ スクリーンを別スレッドで表示します。

ShowSplashScreen メソッドには、MinimumSplashScreenDisplayTime プロパティで指定されたミリ秒以上スプラッシュ スクリーンを表示するコードが含まれています。この機能を使用するためには、プロジェクト デザイナを使用してアプリケーションにスプラッシュ スクリーンを追加するか (この場合は **My.Application.MinimumSplashScreenDisplayTime** プロパティは 2 秒に設定されます)、または **OnInitialize** メソッドか OnCreateSplashScreen メソッドをオーバーライドするメソッドで **My.Application.MinimumSplashScreenDisplayTime** プロパティを設定する必要があります。詳細については、「My.Application.MinimumSplashScreenDisplayTime プロパティ」を参照してください。

b. **OnCreateSplashScreen**。スプラッシュ スクリーンを初期化するコードをデザイナが追加できます。

既定では、このメソッドは何も実行しません。Visual Basic プロジェクト デザイナでアプリケーションのスプラッシュ スクリーンを選択した場合、デザイナは、SplashScreen プロパティをスプラッシュ スクリーン フォームの新しいインスタンスに設定するメソッドで **OnCreateSplashScreen** メソッドをオーバーライドします。

2. OnStartup。**Startup** イベントの発生を機能拡張するための部分です。この関数が **False** を返した場合、アプリケーションの起動処理は中断されます。

既定では、このメソッドは Startup イベントを発生させます。イベント ハンドラが、イベント引数の Cancel プロパティを `True` に設定した場合、このメソッドは `False` を返し、アプリケーションの起動処理はキャンセルされます。

3. OnRun。初期化処理の完了後、メイン アプリケーションを実行する準備ができたときのための開始点です。

既定では、このメソッドは、Windows フォームのメッセージ ループに入る前に、**OnCreateMainForm** メソッド (アプリケーションのメイン フォームを作成するため) および **HideSplashScreen** メソッド (スプラッシュ スクリーンを閉じるため) を呼び出します。

a. OnCreateMainForm。メイン フォームを初期化するコードをデザイナが追加するための手段を提供します。

既定では、このメソッドは何も実行しません。しかし、Visual Basic プロジェクト デザイナでアプリケーションのメイン フォームを選択した場合、デザイナは、MainForm プロパティをメイン フォームの新しいインスタンスに設定するメソッドで **OnCreateMainForm** メソッドをオーバーライドします。

b. HideSplashScreen。スプラッシュ スクリーンが定義されているアプリケーションで、スプラッシュ スクリーンが開かれている場合には、このメソッドがそれを閉じます。

既定では、このメソッドはスプラッシュ スクリーンを閉じます。

4. OnStartupNextInstance。単一インスタンス アプリケーションのインスタンスが二重起動されたときの動作をカスタマイズする手段を提供します。

既定では、このメソッドは StartupNextInstance イベントを発生させます。

5. OnShutdown。**Shutdown** イベントの発生を機能拡張するための部分です。このメソッドは、メイン アプリケーションで未処理の例外が発生した場合には実行されません。

既定では、このメソッドは Shutdown イベントを発生させます。

6. OnUnhandledException。上記のいずれかのメソッドで未処理の例外が発生したときに実行されます。

既定では、デバッガがアタッチされておらず、かつアプリケーションが **UnhandledException** イベントを処理している場合に、このメソッドは UnhandledException イベントを発生させます。

単一インスタンス アプリケーションの場合で、同じアプリケーションが既に実行中のときには、二重起動された方のインスタンスは、同じアプリケーションの元のインスタンスの **OnStartupNextInstance** メソッドを呼び出し、自らは終了します。

WindowsFormsApplicationBase コンストラクタは UseCompatibleTextRendering プロパティを呼び出して、アプリケーションのフォームにどのテキスト描画エンジンを使用するかを判断します。既定では、**UseCompatibleTextRendering** プロパティは **False** を返します。これは Visual Basic 2005 の既定である GDI テキスト描画エンジンが使用されることを示します。**UseCompatibleTextRendering** プロパティは、**True** を返すようにオーバーライドできます。この値は、Visual Basic .NET 2002 および Visual Basic .NET 2003 の既定である GDI+ のテキスト描画エンジンが使用されることを示します。

アプリケーションの構成

Visual Basic アプリケーション モデルの一環として、**WindowsFormsApplicationBase** クラスには、アプリケーションを構成するプロテクト プロパティが用意されています。これらのプロパティは、実装するクラスのコンストラクタで設定する必要があります。

既定の Windows フォーム プロジェクトでは、プロジェクト デザイナは、デザイナの設定でこれらのプロパティを設定するコードを作成します。これらのプロパティが使用されるのはアプリケーションの起動時のみです。アプリケーションが起動された後で値を設定しても効果はありません。

プロパティ	意味	プロジェクト デザイナでの設定
IsSingleInstance	アプリケーションが単一インスタンス アプリケーションまたは複数インスタンス アプリケーションとして動作するかどうかを表します。	方法 : アプリケーションのインスタンス化の動作を指定する
EnableVisualStyles	アプリケーションが XP visual スタイルを使用するかどうかを表します。	方法 : Visual スタイルを有効にする
SaveMySettingsOnExit	アプリケーションの終了時に、ユーザー設定の変更をアプリケーションが自動的に保存するかどうかを表します。	プロジェクト デザイナのアプリケーション ペインの [シャットダウン時に My.Settings を保存する] チェック ボックスをオンにします。
ShutdownStyle	メイン フォームが閉じたときや、すべてのフォームが閉じたときなど、どのようなときにアプリケーションを終了させるかを表します。	方法 : アプリケーションのシャットダウン動作を指定する

参照

関連項目

My.Application オブジェクト
My.Application.Startup イベント
My.Application.StartupNextInstance イベント
My.Application.UnhandledException イベント
My.Application.Shutdown イベント
My.Application.NetworkAvailabilityChanged イベント
WindowsFormsApplicationBase

概念

Visual Basic アプリケーション モデルの概要

My で利用可能なオブジェクトのカスタマイズ

このトピックでは、プロジェクトの **_MYTYPE** 条件付きコンパイル定数を設定することによって、有効にする **My** オブジェクトを制御する方法を説明します。Visual Studio 統合開発環境 (IDE) では、プロジェクトの **_MYTYPE** 条件付きコンパイル定数とプロジェクトの種類が対応付けられています。

定義済みの _MYTYPE 値

_MYTYPE 条件付きコンパイル定数を設定するには、**/define** コンパイラ オプションを使用する必要があります。**_MYTYPE** 定数に独自の値を指定するときには、円記号と引用符のシーケンス (\") で文字列値を囲む必要があります。たとえば、次のように指定します。

```
/define:_MYTYPE=\"WindowsForms\"
```

次の表は、いくつかのプロジェクトの種類に対して設定されている **_MYTYPE** 条件付きコンパイル定数です。

プロジェクトの種類	_MYTYPE 値
クラス ライブラリ	"Windows"
コンソール アプリケーション	"Console"
Web	"Web"
Web コントロール ライブラリ	"WebControl"
Windows アプリケーション	"WindowsForms"
カスタムの **Sub Main** で開始される Windows アプリケーション	"WindowsFormsWithCustomSubMain"
Windows コントロール ライブラリ	"Windows"
Windows サービス	"Console"
空	"Empty"

メモ：

条件付きコンパイル文字列の比較では、**Option Compare** ステートメントの設定にかかわらず、大文字と小文字は常に区別されます。

値に応じて決まる _MY 系コンパイル定数

一方、**_MYTYPE** 条件付きコンパイル定数の値に応じて、他のいくつかの **_MY** 系コンパイル定数の値が、次のように決まります。

_MYTYPE	_MYAPPLICATIONTYPE	_MYCOMPUTERTYPE	_MYFORMS	_MYUSERTYPE	_MYWEBSERVICES
"Console"	"Console"	"Windows"	未定義	"Windows"	TRUE
"Custom"	未定義	未定義	未定義	未定義	未定義
"Empty"	未定義	未定義	未定義	未定義	未定義
"Web"	未定義	"Web"	FALSE	"Web"	FALSE
"WebControl"	未定義	"Web"	FALSE	"Web"	TRUE

"Windows" または ""	"Windows"	"Windows"	未定義	"Windows"	TRUE
"WindowsForms"	"WindowsForms"	"Windows"	TRUE	"Windows"	TRUE
"WindowsFormsWithCustomSubMain"	"Console"	"Windows"	TRUE	"Windows"	TRUE

既定では、未定義の条件付きコンパイル定数は **FALSE** に解決されます。プロジェクトをコンパイルするときに、既定の動作をオーバーライドするように未定義の定数の値を指定できます。

メモ：

_MYTYPE を "Custom" に設定すると、プロジェクトに **My** 名前空間が含まれますが、オブジェクトは含まれません。一方、**_MYTYPE** を "Empty" に設定すると、**My** 名前空間とそのオブジェクトのいずれも含まれません。

次の表は、定義済みの **_MY** 系コンパイル定数の値の効果の説明です。

定数	説明
_MYAPPLICATIONTYPE	定数が "Console"、"Windows"、または "WindowsForms" の場合、**My.Application** が有効になります。 • "Console" バージョンは ConsoleApplicationBase から派生され、メンバは "Windows" バージョンより少なくなります。 • "Windows" バージョンは ApplicationBase から派生され、メンバは "WindowsForms" バージョンより少なくなります。 • **My.Application** の "WindowsForms" バージョンは WindowsFormsApplicationBase から派生されます。**TARGET** 定数が "winexe" と定義されている場合、クラスには **Sub Main** メソッドが含まれます。
_MYCOMPUTERTYPE	定数が "Web" または "Windows" の場合、**My.Computer** が有効になります。 • "Web" バージョンは ServerComputer から派生され、メンバは "Windows" バージョンより少なくなります。 • **My.Computer** の "Windows" バージョンは Computer から派生されます。
_MYFORMS	定数が **TRUE** の場合、**My.Forms** が有効になります。
_MYUSERTYPE	定数が "Web" または "Windows" の場合、**My.User** が有効になります。 • **My.User** の "Web" バージョンは、現在の HTTP 要求のユーザー ID と関連付けられています。 • **My.User** の "Windows" バージョンは、スレッドの現在のプリンシパルと関連付けられています。
_MYWEBSERVICES	定数が **TRUE** の場合、**My.WebServices** が有効になります。
_MYTYPE	定数が "Web" の場合、**My.Log**、**My.Request**、および **My.Response** が有効になります。

参照

関連項目

/define (Visual Basic)
My.Application オブジェクト
My.Computer オブジェクト
My.Forms オブジェクト
My.Log オブジェクト
My.Request オブジェクト
My.Response オブジェクト
My.User オブジェクト
My.WebServices オブジェクト

概念

プロジェクトの種類に応じた My の機能

条件付きコンパイルの概要

Visual Basic のプログラミング ガイド

他の最新のプログラミング言語と同様、Visual Basic は、多くの一般的なプログラミング構成要素と言語要素をサポートします。このガイドでは、Visual Basic によるプログラミングの基本的な要素について案内します。

このセクションの内容

プログラム構造とコード規則

Visual Basic の基本的な構造およびコード規則 (名前付け規則、コード内のコメント、Visual Basic の制限など) についてのドキュメントが含まれています。

Visual Basic 言語の機能

Visual Basic の主要なコンポーネントを紹介および説明します。

Visual Basic におけるオブジェクト指向プログラミング

オブジェクト指向プログラミングについて、クラスやインターフェイス、オブジェクトの作成、イベントやデリゲート、継承、およびオブジェクトのグループなどを説明します。

COM 相互運用

Visual Basic でのコンポーネント オブジェクト モデル (COM: Component Object Model) オブジェクトの作成および使用に関連する相互運用性の問題について説明します。

Visual Basic におけるマルチスレッド

複数のタスクを同時に実行するコードを記述してアプリケーションのパフォーマンスと応答性を向上させる方法について説明します。

関連するセクション

Visual Basic リファレンス

Visual Basic プログラミングのさまざまな側面に関するリファレンス情報を提供します。

Visual Basic コンパイラ

Visual Basic のコマンド ライン コンパイラ、コンパイラ オプション、およびキーワード アップグレード ツールの使用について説明します。

プログラム構造とコード規則

ここでは、一般的な Visual Basic プログラムの構造を紹介します。また、単純な Visual Basic プログラム "Hello World" を示し、Visual Basic のコード規則について説明します。コード規則は、プログラムのロジックではなくプログラムの物理的な構造と外観に焦点を合わせた提案です。コード規則に従うと、コードの読み取り、理解、保守が簡単になります。コード規則には、以下の内容が含まれます。

- コードのラベル付けとコメント付けに関する標準化された書式
- コードのスペーシング、書式指定、およびインデント設定に関するガイドライン
- オブジェクト、変数、およびプロシージャの名前付け規則

以下のトピックでは、Visual Basic プログラムの一連のプログラミング ガイドラインを適切な使用例と共に示します。

このセクションの内容

Visual Basic プログラムの構造

Visual Basic プログラムを構成する要素の概要を示します。

Visual Basic バージョンの Hello World!

代表的なプログラム "Hello World" を Visual Basic で作成する手順を示します。

Visual Basic の Main プロシージャ

アプリケーションの開始点となり、アプリケーションの総合的な制御を行うプロシージャについて説明します。

Visual Basic の名前付け規則

プロシージャ、定数、変数、引数、およびオブジェクトの名前付けに関する一般的なガイドラインを示します。

Visual Basic のコーディング規則

このドキュメントのサンプルを開発するときに使用したガイドラインについて説明します。

条件付きコンパイル (Visual Basic)

特定のコード ブロックを選択的にコンパイルし、その間は他のコード ブロックを無視する方法についてのトピックへのリンクがあります。

方法 : コード内でステートメントを分割および連結する

長いステートメントを複数の行に分割する方法と、複数の短いステートメントを 1 行に結合する方法を示します。

方法 : ステートメントにラベル付けする

On Error Goto などのステートメントで使用するために、コード行に識別用のマーキングをする方法について説明します。

コード内の特殊文字

英数字以外の文字を使用する方法と場所について説明します。

コード内のコメント

説明的なコメントをコードに追加する方法について説明します。

Visual Basic の制限事項

Visual Basic の既知のコーディングの制限の解除について説明します。

関連するセクション

表記規則とコード規則

Visual Basic の標準的なコーディング規則について説明します。

テキスト、コード、およびマークアップの編集

Visual Studio のコード エディタの概要を説明します。

Visual Basic プログラムの構造

Visual Basic プログラムは、標準ビルド ブロックから構築されます。ソリューションは、1 つ以上のプロジェクトで構成されます。プロジェクトは、1 つ以上のアセンブリで構成されます。各アセンブリは、1 つ以上のソース ファイルからコンパイルされます。ソース ファイルは、クラス、構造体、モジュール、およびインターフェイスの定義と実装を提供します。すべてのコードがここに含まれます。

ファイル レベルのプログラミング要素

プロジェクトまたはファイルを開始してコード エディタを開くと、一部のコードが既に適切な順序で配置されて表示されます。作成するコードはすべて以下の順序に従う必要があります。

1. **Option** ステートメント
2. **Imports** ステートメント
3. **Namespace** ステートメントおよび名前空間レベルの要素

ステートメントをこれとは別の順序で入力すると、コンパイル エラーが発生することがあります。

プログラムには、条件付きコンパイル ステートメントを指定することもできます。このようなステートメントは、ソース ファイル内で上記の順序で記述するステートメントの中に混在できます。

Option ステートメント

Option ステートメントは後続のコードの基本的な規則を確立し、構文エラーや論理エラーの発生を防ぎます。Option Explicit ステートメント (Visual Basic) は、すべての変数が正しく宣言され、スペルに誤りがないことを保証するので、デバッグ時間が短縮されます。Option Strict ステートメント ステートメントは、異なる型の変数を使用するときに起こり得るデータの損失や論理エラーを減らすために役立ちます。Option Compare ステートメント には、**Binary** 値または **Text** 値に基づいて文字列を比較する方法を指定します。

Imports ステートメント

プロジェクトの外部で定義された名前をインポートするために、Imports ステートメント を指定できます。**Imports** ステートメントを使うと、インポートした名前空間に定義されているクラスおよびその他の型を修飾なしで参照できます。**Imports** ステートメントは必要に応じていくつでも使用できます。詳細については、「参照と Imports ステートメント」を参照してください。

Namespace ステートメント

名前空間は、プログラミング要素を整理および分類してグループ分けやアクセスをしやすくするために役立ちます。Namespace ステートメント を使って、以下に示すステートメントを特定の名前空間に分類できます。詳細については、「Visual Basic における名前空間」を参照してください。

条件付きコンパイル ステートメント

条件付きコンパイル ステートメントは、ソース ファイル内のほとんどの場所で使用できます。特定の条件に基づいて、コードの一部をコンパイル時に含めることや除外することができます。条件付きコードはデバッグ モードだけで実行されるため、条件付きコンパイル ステートメントはアプリケーションのデバッグにも使用できます。詳細については、「条件付きコンパイルの概要」を参照してください。

名前空間レベルのプログラミング要素

クラス、構造体、およびモジュールのすべてのコードはソース ファイルに含まれます。これらは名前空間レベルの要素であり、名前空間またはソース ファイルのレベルで指定できます。これらの要素には、他のすべてのプログラミング要素の宣言が格納されます。要素のシグネチャを定義するだけで実装を提供しないインターフェイスも、モジュール レベルで指定します。モジュール レベルの各要素の詳細については、次のトピックを参照してください。

- Class ステートメント (Visual Basic)
- Structure ステートメント
- Module ステートメント
- Interface ステートメント (Visual Basic)

名前空間レベルのデータ要素は、列挙およびデリゲートです。

モジュール レベルのプログラミング要素

プロシージャ、演算子、プロパティ、およびイベントだけが、実行可能コード (実行時にアクションを実行するステートメント) を含むことができるプログラミング要素です。これらは、プログラムのモジュール レベルの要素です。プロシージャ レベルの各要素の詳細については、次のトピックを参照してください。

- Function ステートメント (Visual Basic)
- Sub ステートメント (Visual Basic)
- Declare ステートメント
- Operator ステートメント
- Property ステートメント
- Event ステートメント

モジュール レベルのデータ要素は、変数、定数、列挙、およびデリゲートです。

プロシージャ レベルのプログラミング要素

ほとんどのプロシージャ レベル要素の内容は、実行可能なステートメントであり、プログラム コードの一部を構成します。すべての実行可能コードは、プロシージャ (**Function**、**Sub**、**Operator**、**Get**、**Set**、**AddHandler**、**RemoveHandler**、**RaiseEvent**) に含める必要があります。詳細については、「実行可能なステートメント」を参照してください。

プロシージャ レベルのデータ要素は、ローカル変数とローカル定数だけです。

Main プロシージャ

`Main` プロシージャは、アプリケーションが読み込まれた後で最初に実行されるコードです。`Main` は、アプリケーションの開始地点であり、アプリケーション全体を制御する場所です。`Main` には次の 4 種類があります。

- `Sub Main()`
- `Sub Main(ByVal cmdArgs() As String)`
- `Function Main() As Integer`
- `Function Main(ByVal cmdArgs() As String) As Integer`

最も一般的に使用されるプロシージャは、`Sub Main()` です。詳細については、「Visual Basic の Main プロシージャ」を参照してください。

参照
概念
Visual Basic バージョンの Hello World!
Visual Basic の Main プロシージャ
Visual Basic の名前付け規則
Visual Basic の制限事項

Visual Basic バージョンの Hello World!

次のコンソール プログラムは、Visual Basic バージョンの "Hello World!" プログラムです。このプログラムは "Hello, World!" という文字列を表示します。

VB

```
' A "Hello, World!" program in Visual Basic.
Module Hello
  Sub Main()
      MsgBox("Hello, World!") ' Display message on computer screen.
  End Sub
End Module
```

このプログラムでは次の点が重要です。

- コメント
- `Main` プロシージャ
- 入出力
- コンパイルと実行

コメント

この例の 1 行目はコメントです。

VB

```
' A "Hello, World!" program in Visual Basic.
```

単一引用符 (') は、その行の残りがコメントであることを意味し、コンパイラで無視されます。1 行全体をコメントにすることも、次に示すように、他のステートメントの最後にコメントを追加することもできます。

VB

```
MsgBox("Hello, World!") ' Display message on computer screen.
```

Main プロシージャ

すべての Visual Basic アプリケーションには、`Main` という名前のプロシージャが含まれている必要があります。このプロシージャは、アプリケーションの開始点となり、アプリケーションの総合的な制御を行います。このプロシージャは、モジュールが読み込まれるときに呼び出されます。

`Main` には次の 4 種類があります。

- `Sub Main()`
- `Sub Main(ByVal cmdArgs() As String)`
- `Function Main() As Integer`
- `Function Main(ByVal cmdArgs() As String) As Integer`

最も一般的に使用されるプロシージャは、`Sub Main()` です。Windows フォーム アプリケーションを作成する場合以外は、アプリケーションのエントリ ポイントとなる `Main` プロシージャを作成する必要があります。詳細については、「Visual Basic の Main プロシージャ」を参照してください。

入出力

この例では、標準の Visual Basic ランタイム ライブラリを使用します。このライブラリは Microsoft.VisualBasic 名前空間を通じて使用できます。統合開発環境 (IDE: Integrated Development Environment) でプログラムをコンパイルする場合、**Microsoft.VisualBasic** のすべてのプロシージャとプロパティをインポートなしで使用できます。コマンド ラインを使ってコンパイルする場合は、Imports ステートメントをソース コードで使用するか、/imports (Visual Basic) コマンド ライン コンパイラ オプションを使用して、**Microsoft.VisualBasic** メンバをプログラムで使用

できるようにする必要があります。

`Main` プロシージャは MsgBox 関数 (Visual Basic) を呼び出して、文字列 Hello, World! を含むメッセージ ボックスを表示します。

VB

```
MsgBox("Hello, World!") ' Display message on computer screen.
```

コンパイルと実行

"Hello World!" プログラムをコンパイルするには、Visual Studio 統合開発環境またはコマンド ラインを使用します。

コマンド ラインからプログラムをコンパイルおよび実行するには

1. 任意のテキスト エディタを使用してソース ファイルを作成し、`Hello.vb` などのファイル名で保存します。
2. コンパイラを起動するには、次のコマンドを入力します。

   ```
   vbc Hello.vb
   ```

   ソース ファイル内に **Microsoft.VisualBasic** 名前空間のための **Imports** ステートメントがない場合は、**/imports** コマンド ライン コンパイラ オプションを `vbc` コマンドに追加できます。

   ```
   vbc Hello.vb /imports:Microsoft.VisualBasic
   ```

3. プログラムにコンパイル エラーがない場合は、コンパイラによって `Hello.exe` ファイルが作成されます。
4. プログラムを実行するには、次のコマンドを入力します。

   ```
   Hello
   ```

`vbc` コマンドに **/main** コマンド ライン コンパイラ オプションを追加して、`Main` を提供する名前空間とモジュールを指定することもできます。

IDE からプログラムをコンパイルおよび実行するには

1. Visual Basic コンソール アプリケーション プロジェクトを作成します。
2. コードをプロジェクト内にコピーします。
3. [ビルド] メニューの適切なコマンドをクリックするか、または F5 キーを押して、プログラムをビルドおよび実行します ([デバッグ] メニューの [開始] に対応します)。

Visual Basic のコンパイラおよびコンパイラ オプションの詳細については、「コマンド ラインからのビルド (Visual Basic)」を参照してください。

参照

関連項目

Visual Basic プログラムの構造
Imports ステートメント
/imports (Visual Basic)
MsgBox 関数 (Visual Basic)
/main
Microsoft.VisualBasic

概念

Visual Basic の Main プロシージャ

その他の技術情報

コマンド ラインからのビルド (Visual Basic)

Visual Basic の Main プロシージャ

すべての Visual Basic アプリケーションには、`Main` という名前のプロシージャが含まれている必要があります。このプロシージャは、アプリケーションの開始点となり、アプリケーションの総合的な制御を行います。.NET Framework は、アプリケーションをロードして、アプリケーションに制御を渡せる状態になると、`Main` プロシージャを呼び出します。Windows フォーム アプリケーションを作成する場合以外は、アプリケーションのエントリ ポイントとなる `Main` プロシージャを作成する必要があります。

`Main` には、最初に実行するコードを記述します。`Main` では、プログラムの起動時に最初に読み込まれるフォームを決定したり、同一アプリケーションが既にシステム上で稼動しているかどうかを調べたり、アプリケーションで使用する一連の変数を作成したり、アプリケーションで必要なデータベースを開いたりできます。

Main プロシージャの要件

単独で実行されるファイル (通常は拡張子 .exe を持つ) には、`Main` プロシージャを含める必要があります。ライブラリ (拡張子 .dll を持つものなど) は単独では実行されないので、`Main` プロシージャは必要ありません。各種のプロジェクトについての要件を次に示します。

- コンソール アプリケーションは単独で実行されるので、少なくとも 1 つの `Main` プロシージャを記述する必要があります。.
- Windows フォーム アプリケーションは単独で実行されます。しかし、このようなアプリケーションでは Visual Basic コンパイラが自動的に `Main` プロシージャを生成するので、独自に記述する必要はありません。
- クラス ライブラリには `Main` プロシージャは必要ありません。これには Windows コントロール ライブラリと Web コントロール ライブラリが含まれます。Web アプリケーションはクラス ライブラリとして配置されます。

Main プロシージャの宣言

`Main` プロシージャを宣言するには 4 種類の方法があります。引数を受け取るものと受け取らないもの、値を返すものと返さないものです。

メモ：
クラス内で `Main` プロシージャを宣言する場合は、**Shared** キーワードを使用する必要があります。モジュールで `Main` プロシージャを宣言する場合、**Shared** キーワードは不要です。

- 最も単純な方法は、引数を受け取らず、値を返さない **Sub** プロシージャとして宣言することです。

```
Module mainModule
    Sub Main()
        MsgBox("The Main procedure is starting the application.")
        ' Insert call to appropriate starting place in your code.
        MsgBox("The application is terminating.")
    End Sub
End Module
```

- また、`Main` プロシージャは **Integer** 値を返すことができます。オペレーティング システムは、この値をプログラムの終了コードとして使用します。他のプログラムは、Windows ERRORLEVEL 値を確認することで、このコードをテストできます。終了コードを返すには、`Main` を **Sub** プロシージャではなく **Function** プロシージャとして宣言する必要があります。

```
Module mainModule
    Function Main() As Integer
        MsgBox("The Main procedure is starting the application.")
        Dim returnValue As Integer = 0
        ' Insert call to appropriate starting place in your code.
        ' On return, assign appropriate value to returnValue.
        ' 0 usually means successful completion.
        MsgBox("The application is terminating with error level " _
            & CStr(returnValue) & ".")
        Return returnValue
    End Function
End Module
```

- `Main` は、引数として **String** 配列を受け取ることができます。配列内の各文字列には、プログラムの実行に使用するコマンド ライン引数の 1 つが含まれます。これらの値に応じて、異なる動作を実行できます。

```
Module mainModule
    Function Main(ByVal cmdArgs() As String) As Integer
        MsgBox("The Main procedure is starting the application.")
        Dim returnValue As Integer = 0
        ' See if there are any arguments.
        If cmdArgs.Length > 0 Then
            For argNum As Integer = 0 To UBound(cmdArgs, 1)
                ' Insert code to examine cmdArgs(argNum) and take
                ' appropriate action based on its value.
            Next argNum
        End If
        ' Insert call to appropriate starting place in your code.
        ' On return, assign appropriate value to returnValue.
        ' 0 usually means successful completion.
        MsgBox("The application is terminating with error level " _
            & CStr(returnValue) & ".")
        Return returnValue
    End Function
End Module
```

- コマンド ライン引数を受け取り、終了コードを返さない場合は、`Main` を次のように宣言します。

```
Module mainModule
    Sub Main(ByVal cmdArgs() As String)
        MsgBox("The Main procedure is starting the application.")
        Dim returnValue As Integer = 0
        ' See if there are any arguments.
        If cmdArgs.Length > 0 Then
            For argNum As Integer = 0 To UBound(cmdArgs, 1)
                ' Insert code to examine cmdArgs(argNum) and take
                ' appropriate action based on its value.
            Next argNum
        End If
        ' Insert call to appropriate starting place in your code.
        MsgBox("The application is terminating."
    End Sub
End Module
```

参照

関連項目

Visual Basic プログラムの構造
/main
Shared (Visual Basic)
MsgBox 関数 (Visual Basic)
Sub ステートメント (Visual Basic)
Function ステートメント (Visual Basic)
整数型 (Integer) (Visual Basic)
文字列型 (String) (Visual Basic)
UBound 関数 (Visual Basic)
Length

概念

Visual Basic バージョンの Hello World!

Visual Basic の名前付け規則

Visual Basic アプリケーションの要素に名前を付ける場合は、最初の文字を英字、漢字、ひらがな、カタカナ、アンダースコア (_) のいずれかにする必要があります。ただし、アンダースコアで始まる名前は 共通言語仕様 (CLS) に準拠しないので注意してください。

名前付けには以下の推奨事項が適用されます。

- `FindLastRecord` や `RedrawMyForm` のように、名前の各単語は大文字で開始します。
- `InitNameArray` や `CloseDialog` のように、関数名とメソッド名は動詞で開始します。
- `EmployeeName` や `CarAccessory` のように、クラス、構造体、モジュール、およびプロパティの名前は名詞で開始します。
- インターフェイス名は "I" で開始し、その後には `IComponent` のように名詞または名詞句を続けるか、または `IPersistable` のようにインターフェイスの動作を説明する形容詞を続けます。アンダースコアは使用しないでください。また、省略形を使用すると混乱を招く可能性があるため、使用しないようにしてください。
- イベント ハンドラ名はイベントの種類を説明する名詞で開始し、その後にサフィックス "`EventHandler`" を付け、"`MouseEventHandler`" のように記述します。
- イベント引数クラスの名前には、サフィックス "`EventArgs`" を含めます。
- イベントに前後の概念がある場合は、"`ControlAdd`" や "`ControlAdded`" のように現在形または過去形を使用します。
- 長い用語や頻繁に使用する用語は、省略形を使って名前を適切な長さにします。たとえば、"Hypertext Markup Language" の代わりに "HTML" を使用します。通常、32 文字より長い変数名は、解像度の低いディスプレイでは読みにくくなります。省略形は、必ずアプリケーション全体で統一してください。プロジェクト内で "HTML" と "Hypertext Markup Language" の両方を不規則に使用すると混乱が生じます。
- 外部スコープにある名前と同じ名前を内部スコープで使用することは避けてください。不正な変数にアクセスされるとエラーが発生します。同じ名前の変数とキーワードの間で競合が発生した場合は、適切なタイプ ライブラリを先頭に付けてキーワードを識別する必要があります。たとえば `Date` という変数がある場合は、System.DateTime.Date の呼び出しによってのみ、組み込み関数 **Date** を使用できます。

参照

概念

コード内の要素名としてのキーワード
Visual Basic における Me、My、MyBase、MyClass
宣言された要素の名前

その他の技術情報

プログラム構造とコード規則
Visual Basic リファレンス

コード内の要素名としてのキーワード

変数、クラス、メンバなどのプログラム要素には、予約キーワードと同じ名前を付けることができます。たとえば、Loop という名前の変数を作成できます。ただし、予約された **Loop** キーワードと同じ名前を持つ別の Loop を指すには、次の例に示すように、先頭に完全な修飾文字列を付けるか、角かっこ (**[ ]**) で囲む必要があります。

VB

```
' The following statement precedes Loop with a full qualification string.
sampleForm.Loop.Visible = True
' The following statement encloses Loop in square brackets.
 [Loop].Visible = True
```

このどちらも行わない場合、次の例に示すように、Visual Basic では組み込みの **Loop** キーワードを使用していると想定し、エラーを生成します。

```
' The following statement causes a compiler error.
Loop.Visible = True
```

フォームやコントロールを参照するとき、予約キーワードと同じ名前の変数を宣言するとき、または予約キーワードと同じ名前のプロシージャを定義するときには角かっこを使用できます。修飾名を使用しなかったり、名前を角かっこで囲むことを忘れると、コードにエラーが組み込まれたり、コードが読みにくくなったりします。このため、プログラム要素の名前と同じ予約キーワードは使用しないことをお勧めします。ただし、既存のフォーム名またはコントロール名と競合する新規キーワードが将来の Visual Basic のバージョンで定義された場合は、新しいバージョンで動作するようにコードを更新するときにこの技法を使用できます。

メモ :

参照される他のアセンブリから提供される要素名をプログラムに含めることもできます。これらの要素名が予約キーワードと競合する場合は、名前を角かっこで囲むことによって、Visual Basic は定義された要素として解釈するようになります。

参照

概念

Visual Basic の名前付け規則

その他の技術情報

プログラム構造とコード規則

キーワード (Visual Basic)

Visual Basic における Me、My、MyBase、MyClass

Visual Basic の **Me**、**My**、**MyBase**、**MyClass** という概念はよく似ているので、慣れない人は混乱するおそれがあります。ここでは、これらの概念の違いを明確にするために、それぞれの特徴について説明します。

Me

Me キーワードは、現在実行しているコードから、そのコードを含んでいるクラスまたは構造体の特定のインスタンスを参照するための方法です。**Me** は、現在のインスタンスを参照するオブジェクト変数または構造体変数と同じように機能します。特に、別のクラス、構造体、またはモジュールのプロシージャに、クラスや構造体の現在のインスタンスに関する情報を渡す場合などには、**Me** が役立ちます。

My

My は、.NET Framework の数多くのクラスに簡単かつ直観的にアクセスするための手段であり、Visual Basic ユーザーがコンピュータ、アプリケーション、設定、リソースなどとやり取りできるようにします。

MyBase

MyBase キーワードは、クラスの現在のインスタンスの基本クラスを参照するオブジェクト変数と同じように機能します。**MyBase** は一般に、派生クラス内でオーバーライドまたはシャドウされている基本クラスのメンバにアクセスする目的で使用されます。**MyBase.New** は、派生クラスのコンストラクタから基本クラスのコンストラクタを明示的に呼び出すときに使用されます。

MyClass

MyClass キーワードは、クラスの現在のインスタンスをもともとの実装どおりの状態で参照するオブジェクト変数のような動作をします。**MyClass** は **Me** に似ていますが、これに対するメソッド呼び出しはすべて、そのメソッドが **NotOverridable** であるかのように扱われます。

参照

関連項目

MyBase
MyClass
Me

概念

My による開発
継承の基本

Visual Basic のコーディング規則

以下のガイドラインは、サンプルおよびドキュメントの開発で Microsoft が使用しているものです。Visual Basic 言語仕様では、コーディング規則は定義されていません。

- コーディング規則があると、コードの見た目が統一されるため、コードを読むときにレイアウトではなく内容に専念できます。
- 規則がある方が、コードをすばやく理解できます。以前の経験に基づいて予測がつくようになるからです。
- 規則がある方が、コードのコピー、変更、およびメンテナンスが簡単になります。
- 規則を見ると、Visual Basic の "ベスト プラクティス" がわかります。

説明

名前付け規則

- 名前付けのガイドラインについては、「クラス ライブラリ開発のデザイン ガイドライン」を参照してください。
- Visual Studio デザイナ ツールで作成されるオブジェクトの名前は、ガイドラインに則るように変更する必要はありません。
- Imports ステートメントを追加するのではなく、名前空間修飾を使用します。名前空間が既定でプロジェクトにインポートされた場合、コードを完全修飾する必要はありません。コピーおよび貼り付けされたときに、IntelliSense により修飾なしで実行されるからです。長い行のコードを改行して読みやすくするときには、修飾名は "." の後で改行できます。以下に例を示します。

```
Dim collection As System.Diagnostics. _
       InstanceDataCollectionCollection
```

- "My" または "my" を変数名の一部として使用しないようにします。**My** オブジェクトとの混同を招くからです。

レイアウト規則

コードの構造を強調する書式が使用され、コードが読みやすくなっているのが、優れたレイアウトです。

- 再フォーマット機能を使用して、既定の設定 (スマート インデント、4 文字インデント、タブを空白として保存) でコードを書式設定します。詳細については、「[VB 固有] ([オプション] ダイアログ ボックス - [テキスト エディタ] - [Basic])」を参照してください。
- 1 つの行には 1 つのステートメントのみを記述します。":" は使用しないようにします。
- 1 つの行には 1 つの宣言のみを記述します。
- 継続行はタブ 1 つ分インデントします。
- メソッド定義とプロパティ定義の間に少なくとも 1 行の空白行を追加します。

コメント規則

- コード行の末尾にコメントを記述しないようにします。コメントは別個の行に記述します。
- 英語でコメントを記述する場合、コメント テキストの始まりは英大文字を使用します。
- コメント テキストの終わりは句点で終了します。
- コメント デリミタ (') とコメント テキストの間に空白を 1 つ挿入します。

```
' Here is a comment.
```

- アスタリスク (*) を整形したブロックでコメントを囲まないようにします。

プログラムの構造

- **Main** メソッドを使用するときには、新しいコンソール アプリケーションの既定の構造を使用し、コマンド ライン引数には **My** を使用します。

```
Sub Main()
    For Each argument As String In My.Application.CommandLineArgs
```

```
        ' Add code here to use the string variable.
    Next
End Sub
```

言語ガイドライン
文字列型 (String)

- 文字列の連結には & を使用します。

```
MsgBox("hello" & vbCrLf & "goodbye")
```

- ループ内での文字列の追加には StringBuilder オブジェクトを使用します。

```
Dim longString As New System.Text.StringBuilder
For count As Integer = 1 To 1000
    longString.Append(count)
Next
```

Unsigned データ型

- メモリに余裕がない場合を除いては、unsigned 型ではなく **Integer** を使用します。

配列

- 宣言行で配列を初期化するときには、次のような短い構文を使用します。

```
Dim letters() As String = {"a", "b", "c"}
```

次のような記述は避けます。

```
Dim letters() As String = New String() {"a", "b", "c"}
```

- 配列指定子は、次のように、型ではなく変数に指定します。

```
Dim letters() As String = {"a", "b", "c"}
```

次のような記述は避けます。

```
Dim letters As String() = {"a", "b", "c"}
```

- 基本データ型の配列の宣言と初期化では、次のような { } 構文を使用します。

```
Dim letters() As String = {"a", "b", "c"}
```

次のような記述は避けます。

```
Dim letters(2) As String
letters(0) = "a"
letters(1) = "b"
letters(2) = "c"
```

With キーワードの使用

1 つのオブジェクトに対する呼び出しが続く場合には、**With** キーワードの使用を検討します。

ループ変数は For または For Each ステートメント内で宣言

- 例：

- 例 :

```
For count As Integer = 0 To 2
    MsgBox(names(count))
Next
```

- 例 :

```
For Each name As String In names
    MsgBox(name)
Next
```

Try...Catch の使用

- **Dispose** メソッドを実装するオブジェクトでは、常に **Try...Catch** を使用します。
- On Error Goto は使用しないようにします。

Using ステートメントの使用

Try...Catch ステートメントを使用する場合で、**Finally** ブロックのコードが **Dispose** の呼び出しのみのときには、**Using** を代わりに使用します。

IsNot キーワード

Not ... Is Nothing よりは **IsNot** キーワードを使用します。

フォームの既定のインスタンス

`My.Forms.Form1.ShowDialog` ではなく `Form1.ShowDialog` を使用します。

New キーワード

- 短い形式のインスタンス化を使用します。

```
Dim employees As New Collection()
```

次のような記述は避けます。

```
Dim employees As Collection = New Collection()
```

- パラメータなしのコンストラクタを使用してから、**With** を使用してプロパティを設定します。

```
Dim orderLog As New EventLog()
With orderLog
    .Log = "Application"
    .Source = "Application Name"
    .MachineName = "Machine Name"
End With
```

イベント処理

- **AddHandler** ではなく **Handles** を使用します。

```
Private Sub MenuItem1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MenuItem1.Click
```

- **AddressOf** を使用し、デリゲートの明示的なインスタンス化は避けます。

```
Dim closeItem As New MenuItem("Close", AddressOf MenuItem1_Click)
Me.MainMenu1.MenuItems.Add(closeItem)
```

- イベントを定義するときには、短い構文を使用し、デリゲートの定義はコンパイラに任せます。

```
Public Event WhatHappened(ByVal source As Object, _
    ByVal e As WhatHappenedEventArgs)
```

共有メンバの使用

Shared メンバの呼び出しにはクラス名を使用し、インスタンス変数からは行わないようにします。

MsgBox 関数

`MessageBox.Show` または `Console.WriteLine` の代わりに **MsgBox** を使用します。

My 名前空間の使用

.NET Framework クラス ライブラリや Visual Basic ランタイム ライブラリよりも **My** の機能を使用します。

Visual Basic ランタイム ライブラリのメンバの使用

.NET Framework クラス ライブラリよりも Visual Basic ランタイム ライブラリを使用します。

サンプルのガイドライン

全般

- クラス ライブラリ開発者向けのデザイン ガイドライン (http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/cpgenref/html/cpconnetframeworkdesignguidelines.asp) のデザイン ガイドラインに従います。
- **MsgBox** 呼び出しではプロンプトとタイトルを指定します。
- 必要に応じてリソース ファイルを使用します。
- 各ファイルまたはプロジェクト設定で **Option Strict On** を指定します。
- コンパイルではすべての警告をオンにします。
- **Class**、**Structure**、または **Interface** は、1 つのファイル内で 1 つのみ定義します。
- ファイルの保存には既定のエンコーディングを使用します。

ローカリゼーション

- 可能な場合は `AutoSize` プロパティを使用します。
- コントロールを非表示にしたり重ねたりしないようにします。
- 複数のコントロールを整列して 1 つの文を作成するのは避けます。
- ある文字列からいくつかの文字を取り去ることによって別の文字列を組み立てるのは避けます。
- カルチャに依存しないグラフィックスを使用します。
- フォントは Tahoma または MS Sans Serif のみを使用します。

ユーザー補助

- カラー ピッカー ダイアログ ボックスの [システム] タブにある色を使用します。
- すべてのメニュー、ラベル、ボタンなどにおいて、アクセラレータを使用します。
- コントロールのプロパティを、次の表に示すように設定します。

プロパティ	設定値
AccessibleDescription	コントロールの説明。
AccessibleName	コントロールの名前。
AccessibleRole	既定値。または、コントロールに別の役割を割り当てる場合は、このプロパティを変更します。

TabIndex	論理的な順序に設定します。
Text	クリック可能なコントロールすべてに対し、キーボード アクセス キー (ショートカット) を設定します。
Font size	既定値、または 10 ポイント以上に設定します。
Forecolor	既定値
Backcolor	既定値
BackgroundImage	既定値

セキュリティ

「安全なコーディングのガイドライン」のガイドラインに従います。

参照

その他の技術情報

クラス ライブラリ開発のデザイン ガイドライン
安全なコーディングのガイドライン

条件付きコンパイル (Visual Basic)

条件付きコンパイルでは、プログラムの特定のコード ブロックだけを選択してコンパイルできます。プログラムの他の部分は無視されます。

このセクションの内容

条件付きコンパイルの概要

条件付きコンパイルを行う目的について説明し、**#Const** ディレクティブと **#If...Then...#Else** ディレクティブについて解説します。

方法 : 条件付きコンパイル定数を宣言する

[プロパティ ページ] ダイアログ ボックス、コマンド ライン、またはコード内で条件付きコンパイル定数を宣言する方法について説明します。

方法 : コードのセクションを折りたたんで非表示にする

Visual Studio のアウトライン機能を使って展開したり折りたたんだりする、コード ブロックの指定方法について説明します。

関連するセクション

プログラム構造とコード規則

読みやすく管理しやすいコードを作成するためのヒントを紹介します。

MSIL へのコンパイル

コンパイラがソース コードを MSIL (Microsoft Intermediate Language) に翻訳する方法について説明します。

MSIL からネイティブ コードへのコンパイル

MSIL をネイティブ (CPU 固有の) コードに変換する .NET Framework Just-In-Time (JIT) コンパイラの概要を示します。

条件付きコンパイルの概要

条件付きコンパイルを使用すると、コードの特定のセクションだけを選択してコンパイルできます。たとえば、同じプログラミング タスクに対する異なるアプローチについて速度を比較するデバッグ ステートメントを記述する場合や、アプリケーションを混合言語にローカライズする場合などがあります。条件付きコンパイル ステートメントは、実行時ではなくコンパイル時に実行されます。

条件付きコンパイル定数をコード内で宣言するには **#Const** ディレクティブを使用し、条件に基づいてコンパイルするコードのブロックを指定するには **#If...Then...#Else** ディレクティブを使用します。たとえば、同じソース コードからフランス語バージョンおよびドイツ語バージョンのアプリケーションを作成するには、定義済みの定数 `FrenchVersion` および `GermanVersion` を使用して、プラットフォーム固有のコードを **#If...Then** ステートメントの中に記述します。この方法を次の例で示します。

VB

```
#If FrenchVersion Then
    ' <code specific to the French language version>.
#ElseIf GermanVersion Then
    ' <code specific to the German language version>.
#Else
        ' <code specific to other versions>.
#End If
```

コンパイル時に定数 `FrenchVersion` の値を **True** に設定すると、フランス語バージョンの部分のコードがコンパイルされます。定数 `GermanVersion` の値を **True** に設定すると、ドイツ語バージョンのコードがコンパイルされます。どちらの値も **True** に設定しない場合は、最後の **Else** ブロックのコードが実行されます。

メモ :

コードが現在の分岐の一部ではない場合、コードの編集時、および条件付きコンパイル ディレクティブの使用時に、オートコンプリートは機能しません。

参照

処理手順

方法 : 条件付きコンパイル定数を宣言する

方法 : コードのセクションを折りたたんで非表示にする

関連項目

#Const ディレクティブ

#If...Then...#Else ディレクティブ

その他の技術情報

コマンド ラインからのビルド (Visual Basic)

方法：条件付きコンパイル定数を宣言する

条件付きコンパイル定数は、次の 3 つのいずれかで設定できます。

- プロジェクト デザイナ
- コマンド ライン (コマンド ライン コンパイラを使用する場合)
- コード内

条件付きコンパイル定数は特殊なスコープを持つため、標準のコードからアクセスすることはできません。条件付きコンパイル定数のスコープは、定数の設定方法によって異なります。次の表は、上記の 3 つの方法を使って宣言した定数のスコープを示しています。

定数の設定方法	定数のスコープ
プロジェクト デザイナ	プロジェクト内のすべてのファイルに対してパブリック
コマンド ライン	コマンド ライン コンパイラに渡されたすべてのファイルに対してパブリック
コード内の **#Const** ステートメント	宣言されたファイル内でプライベート

プロジェクト デザイナで定数を設定するには

- 実行可能ファイルを作成する前に、「方法 : プロジェクト プロパティおよび構成設定を変更する」で説明されている手順に従って、プロジェクト デザイナで定数を設定します。

コマンド ラインで定数を設定するには

- 次のように、**/d** スイッチを使って条件付きコンパイル定数を入力します。

  ```
  vbc MyProj.vb /d:conFrenchVersion=-1:conANSI=0
  ```

  /d スイッチと最初の定数の間にスペースは必要ありません。詳細については、「/define (Visual Basic)」を参照してください。

  コマンド ラインで設定する定数の値はプロジェクト デザイナで行った設定より優先されますが、プロジェクト デザイナで行った設定は消去されません。プロジェクト デザイナで行った引数の設定は、以降のコンパイルでも有効です。

  コード自体に定数を記述する場合は、定数の宣言を行ったモジュール全体が定数のスコープになるため、配置場所に関する厳密な規則はありません。

コード内で定数を設定するには

- 定数を使用するモジュールの宣言ブロックに定数を配置します。このように配置すると、コードが整理されてわかりやすくなります。

参照

処理手順

方法 : コードのセクションを折りたたんで非表示にする

関連項目

条件付きコンパイル定数
/define (Visual Basic)
#If...Then...#Else ディレクティブ
#Const ディレクティブ

概念

条件付きコンパイルの概要

その他の技術情報

コマンド ラインからのビルド (Visual Basic)

方法：コードのセクションを折りたたんで非表示にする

#Region ディレクティブを使用すると、Visual Basic ファイルのコードをセクションごとに折りたたんで非表示にできます。**#Region** ディレクティブでは、Visual Studio コード エディタを使用するときに展開や折りたたみが可能なコードのブロックを指定できます。コードを選択して非表示にすると、管理しやすく読みやすいファイルになります。詳細については、「方法 : コードをアウトライン表示する/非表示にする」を参照してください。

#Region ディレクティブは、**#If...#End If** などのコード ブロック セマンティクスをサポートします。これらは、あるブロックで始まって他のブロックで終わることはできません。始まりと終わりが同じブロック内にあることが必要です。**#Region** ディレクティブは関数内ではサポートされていません。

コードのセクションを折りたたんで非表示にするには

- 次の例のように、コードのセクションを **#Region** ステートメントと **#End Region** ステートメントの間に配置します。

  VB

```
#Region "This is the code to be collapsed"
    Private components As System.ComponentModel.Container
    Dim WithEvents Form1 As System.Windows.Forms.Form

    Private Sub InitializeComponent()
        components = New System.ComponentModel.Container
        Me.Text = "Form1"
    End Sub
#End Region
```

  1 つのコード ファイルで複数の **#Region** ブロックを使用できます。したがって、折りたたむことができるようにするプロシージャやクラスのブロックを各ユーザーが定義できます。また、ある **#Region** ブロックを他の **#Region** ブロックの中にネストすることもできます。

メモ：
非表示にしたコードもコンパイルの対象となり、**#If...#End If** ステートメントにも影響しません。

参照

処理手順

方法 : 条件付きコンパイル定数を宣言する

方法 : コードをアウトライン表示する/非表示にする

関連項目

#Region ディレクティブ

#If...Then...#Else ディレクティブ

概念

条件付きコンパイルの概要

方法：コード内でステートメントを分割および連結する

コード エディタでコードを記述するときに、ステートメントが長くなりすぎて横スクロールが必要になることがあります。ステートメントが長くてもコードの実行には影響しませんが、コードがディスプレイに表示されたときに読みにくくなります。このような場合は、単一の長いステートメントを複数行に分割することを検討する必要があります。

また、複数のステートメントを 1 行に収めることが必要な場合もあります。たとえば、短い複数のステートメントがあり、スペースを節約する必要がある場合などです。この機能は、モジュール内で変数またはコマンドを編成する場合にも便利です。

単一のステートメントを複数行に分割するには

- 行を分割する位置で、行連結文字つまりアンダースコア (_) を使用します。アンダースコアの直前には空白を指定し、直後には行終端記号 (復帰) を指定する必要があります。

  次の例では、最終行以外の末尾に行連結文字を指定して、1 つのステートメントを 4 行に分割しています。

  VB

```
cmd.CommandText = _
    "SELECT * FROM Titles JOIN Publishers " _
    & "ON Publishers.PubId = Titles.PubID " _
    & "WHERE Publishers.State = 'CA'"
```

  行連結シーケンスを使用すると、オンラインの場合にも出力した場合にもコードが読みやすくなります。

メモ：
行連結文字は、行の末尾に指定する必要があります。同じ行で行連結シーケンスの後ろに文字を記述しないでください。

  引数名の途中など、行連結文字を使用できる場所に関していくつかの制限事項があります。引数リストを行連結文字で分割することはできますが、個々の引数名は分割せずに残す必要があります。

メモ：
行連結文字を使用してコメントを複数行に続けて記述しないでください。コンパイラは、コメントの開始点以降、文字に特別な意味があるかどうかをチェックしません。複数行にコメントを記述する場合は、各行にコメント記号 (') を繰り返し記述してください。

各ステートメントを別個の行に配置する方法をお勧めしますが、Visual Basic では複数のステートメントを同じ行に配置することもできます。

複数のステートメントを同じ行に配置するには

- 次の例に示すように、ステートメントをコロン (:) で区切ります。

  VB

```
text1.Text = "Hello" : text1.BackColor = System.Drawing.Color.Red
```

参照

その他の技術情報

プログラム構造とコード規則

Visual Basic におけるステートメント

方法：ステートメントにラベル付けする

ステートメント ブロックは、コロン区切りのコード行から構成されます。識別文字列または識別番号を先頭に付加したコード行は、"ラベル付けされた" と表現されます。ステートメント ラベルは、コード行のマーキングに使用します。このマーキングによって、**On Error Goto** などのステートメントで使用する行を識別します。

ラベルとして使用できるのは、Visual Basic 2005 の有効な識別子 (プログラミング要素を識別するものなど) または整数リテラルです。ラベルはソース コード行の先頭に記述し、ラベルの直後にはコロンを記述する必要があります。同じ行にステートメントが続くかどうかに関係なく、コロンは必ず記述してください。

コンパイラは、行の先頭が、定義済み識別子と一致するかどうかでラベルを識別します。一致しない場合、コンパイラはそれをラベルと判断します。

ラベルにはそれぞれの宣言空間があり、それぞれの識別子で干渉することはありません。ラベルのスコープは、メソッドの本体です。あいまいな場合は、ラベル宣言が優先されます。

メモ：
ラベルは、メソッド内の実行可能なステートメントに対してだけ使用できます。

コード行にラベル付けするには

- 識別子とそれに続くコロンをソース コード行の先頭に置きます。

  たとえば、次に示すコード行には、それぞれ `Jump` と `120` のラベルが付けられています。

  VB

```
Jump:   FileOpen(1, "testFile", OpenMode.Input)
        ' ...
120:    FileClose(1)
```

参照

概念

ステートメントの概要

その他の技術情報

Visual Basic におけるステートメント

プログラム構造とコード規則

コード内の特殊文字

コードに、標準の英数字ではない特殊文字の使用が必要な場合があります。Visual Basic の文字セットに含まれる区切り記号と特殊文字には、プログラム テキストの整理から、コンパイラやコンパイル済みプログラムが実行するタスクの定義まで、さまざまな用途があります。実行するオペレーションを指定するのには使用されません。

かっこ

Sub や **Function** などのプロシージャを定義するときはかっこを使用します。すべてのプロシージャの引数リストは、かっこで囲む必要があります。また、かっこは変数や引数を論理グループに含める場合にも使用します。具体的には、複合式の中で演算子の既定の優先順位をオーバーライドする場合にかっこが必要になります。次に例を示します。

VB

```
Dim a, b, c, d, e As Double
a = 3.2
b = 7.6
c = 2
d = b + c / a
e = (b + c) / a
```

このコードを実行すると、`d` の値は 8.225、`e` の値は 3 になります。`d` の計算では、**/** の優先順位は既定で **+** よりも高いため、`d = b + (c / a)` を実行しても結果は同じになります。`e` の計算に使用されているかっこは、既定の優先順位をオーバーライドします。

区切り記号

区切り記号は、その名前が示すとおり、コードのセクションを区切ります。Visual Basic では、区切り記号はコロン (**:**) です。複数のステートメントを複数行ではなく単一行に配置して、スペースを節約し、コードを読みやすくする場合に区切り記号を使用します。コロン (**:**) で区切られた 3 つのステートメントの例を次に示します。

VB

```
a = 3.2 : b = 7.6 : c = 2
```

連結

& 演算子は、連結 つまり文字列を結びつけるために使用します。数値を加算する **+** 演算子と混同しないでください。**+** 演算子を使用して連結すると、数値を連結する場合に誤った結果が生成されることがあります。次にコード例を示します。

VB

```
var1 = "10.01"
var2 = 11
resultA = var1 + var2
resultB = var1 & var2
```

このコードを実行すると、`resultA` の値は 21.01、`resultB` の値は "10.0111" になります。

メンバ アクセス演算子

型のメンバにアクセスするには、型名とメンバ名の間に、ドット (**.**) 演算子または感嘆符 (**!**) 演算子を用います。

ドット (.) 演算子

. 演算子は、クラス、構造体、インターフェイス、列挙値にメンバ アクセス演算子として使用します。フィールド、プロパティ、イベント、メソッドなどのメンバに使用できます。使い方は次の例のようになります。

VB

```
Dim nextForm As New System.Windows.Forms.Form
' Access Text member (property) of Form class (on nextForm object).
nextForm.Text = "This is the next form"
' Access Close member (method) on nextForm.
```

```
  nextForm.Close()
```

感嘆符 (!) 演算子

クラスまたはインターフェイスに限っては、感嘆符 (!) 演算子をディクショナリ アクセス演算子として使用します。クラスまたはインターフェイスには、単一の文字列型の引数を受け入れる既定のプロパティが必要です。**!** 演算子のすぐ後に識別子を続けると、既定のプロパティへの文字列引数になります。次にコード例を示します。

VB

```
Public Class hasDefault
  Default Public ReadOnly Property index(ByVal s As String) As Integer
    Get
      Return 32768 + AscW(s)
    End Get
  End Property
End Class
Public Class testHasDefault
  Public Sub compareAccess()
    Dim hD As hasDefault = New hasDefault()
    MsgBox("Traditional access returns " & hD.index("X") & vbCrLf & _
      "Default property access returns " & hD("X") & vbCrLf & _
      "Dictionary access returns " & hD!X)
  End Sub
End Class
```

MsgBox の 3 つの出力行は、いずれも値 `32856` を表示します。最初の行では `index` プロパティに通常の方法でアクセスし、2 行目では `index` が `hasDefault` クラスの既定のプロパティであることが利用されています。また、3 行目ではディクショナリ アクセスを使ってクラスにアクセスしています。

! 演算子の 2 つ目のオペランドは、文字列を二重引用符 (" ") で囲まずに定義する必要があることに注意してください。つまり、文字列リテラルや文字列変数を使うことはできません。**MsgBox** 呼び出しの最後の行を次のように書き換えると、文字列リテラルが `"X"` のように二重引用符で囲まれているためエラーが発生します。

```
"Dictionary access returns " & hD!"X")
```

メモ：

既定コレクションの参照は、必ず明示的に指定してください。特に、遅延バインディング変数では **!** 演算子を使用できません。

感嘆符 (!) は、**Single** の型宣言文字としても使用されます。

参照

概念

型文字

その他の技術情報

プログラム構造とコード規則

コード内のコメント

コード例にはコメント記号 (') がしばしば見られます。この記号は、後続のテキスト (コメント) を無視するように Visual Basic コンパイラに指示します。コメントは、コードを読むユーザーに役立つように追加される簡単な説明です。

プロシージャの先頭に、そのプロシージャの機能の特性 (何を実行するか) について説明する簡単なコメントを常に配置するのは、推奨されるプログラミング方法です。コードを作成した本人にとっても、コードを調べる他人にとっても、この説明は役に立ちます。実装の詳細 (プロシージャの実行手順) は、機能の特性を説明するコメントとは別に記述する必要があります。実装の詳細を記述に入れる場合は、関数を更新するときにその説明も更新してください。

同じ行のステートメントの後にコメントを入れたり、1 行全体をコメントにしたりできます。両方の例を次のコードに示します。

VB

```
' This is a comment beginning at the left edge of the screen.
text1.Text = "Hi!"    ' This is an inline comment.
```

コメントを複数行に記述する必要がある場合は、以下に例を示すとおりに、各行にコメント記号を記述します。

VB

```
' This comment is too long to fit on a single line, so we break
' it into two lines. Some comments might need three or more lines.
```

コメントのガイドライン

次の表は、どの種類のコメントをコードのセクションの前に配置できるかに関する一般的なガイドラインを示しています。これらは推奨であり、Visual Basic ではコメントの追加に規則はありません。コードの作成者自身およびコードを読む他のユーザーに最適な内容を記述してください。

コメント タイプ	コメントの説明
目的	プロシージャが行う内容 (手順ではありません) を説明します。
外部からの影響	各外部変数、コントロール、開いているファイル、またはプロシージャからアクセスされるその他の要素の一覧を示します。
影響	影響を受ける外部変数、コントロール、またはファイル、およびその効果 (明白でない場合のみ) の一覧を示します。
受け取る値	引数の目的を指定します。
戻り値	プロシージャから返される値について説明します。

次のことに留意してください。

- 重要な変数を宣言する場合は、宣言した変数の用途を説明するためのインライン コメントを必ず前に配置します。
- コメントには複雑な実装の詳細だけを記述して済むように、変数、コントロール、およびプロシージャにはわかりやすい名前を付けます。
- 同じ行の行連結シーケンスの後にコメントを付けることはできません。

コード ブロックのコメント記号を追加または削除するには、1 行以上のコードを選択し、[編集] ツール バーの [選択範囲のコメント] ボタン () または [選択範囲のコメント解除] ボタン () をクリックします。

メモ :
テキストの前に **REM** キーワードを付けて、コードにコメントを追加することもできます。ただし、' 記号および [選択範囲のコメント] と [選択範囲のコメントの解除] の各ボタンの方が使いやすく、必要なスペースとメモリが少なくて済みます。

参照

関連項目

REM ステートメント (Visual Basic)
その他の技術情報
プログラム構造とコード規則

方法 : コードにコメントを追加する (Visual Basic)

次に示すのは、ソース コードの行全体、行の一部、および複数行に挿入された各コメントの例です。

使用例

VB

```
' This entire line is a comment.
Dim DailyTotal As Decimal = 0   ' Sales total for today.
' This comment is so long that it requires more than one line, so
' the comment character (') must be repeated on the second line.
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System 名前空間への参照

> **メモ :**
> 行連結シーケンス (_) を使用してコメントを複数行に続けることはできません。コメントの各行の先頭にコメント文字 (') を繰り返し記述する必要があります。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

概念

コード内のコメント

XML の使用によるコードのドキュメントの作成 (Visual Basic)

Visual Basic では、XML を使用して、コードのドキュメントを作成できます。

XML ドキュメント コメント

Visual Basic では、プロジェクトの XML ドキュメントを簡単な方法で自動的に作成できます。型およびメンバの XML スケルトンを自動的に生成したうえで、概要、各パラメータの説明ドキュメント、およびその他のコメントを指定できます。適切なセットアップにより、XML ドキュメントは、プロジェクトと同じ名前に .xml という拡張子を付けた XML ファイルとして自動的に作成されます。詳細については、「/doc」を参照してください。

この XML ファイルを利用したり、何らかの形で XML として操作することが可能です。このファイルは、プロジェクトの出力ファイル (.exe ファイルまたは .dll ファイル) と同じディレクトリに配置されます。

XML ドキュメントの先頭は `'''` です。コメントの処理には、次の制限があります。

- ドキュメントは、適切な XML であることが必要です。XML が整形式でない場合、警告が生成され、エラーが発生したことを示すコメントがドキュメント ファイルに記述されます。
- 開発者は、独自のタグ セットを自由に作成できます。タグのセットには、推奨のものがあります (このトピックの「関連項目」を参照してください)。推奨されるタグには、次のような特殊な意味を持つタグがあります。
    - <param> タグは、パラメータの記述に使用します。このタグを使用した場合、コンパイラはパラメータが存在するかどうか、およびすべてのパラメータがドキュメントに記述されているかどうかを検査します。検査に失敗した場合、コンパイラは警告を発行します。
    - **cref** 属性をタグに追加すると、コード要素を参照できます。コンパイラは、このコード要素が存在することを検査します。検査に失敗した場合、コンパイラは警告を発行します。また、コンパイラは、**cref** 属性で指定されている型を検索するときに、**Imports** ステートメントを考慮に入れます。
    - <summary> タグは、Visual Studio で型またはメンバについての追加的な情報を表示するために、IntelliSense によって使用されます。

関連項目

ドキュメント コメントを記述した XML ファイルの作成の詳細については、以下のトピックを参照してください。

- /doc
- ドキュメント コメントとして推奨される XML タグ (Visual Basic)
- XML ファイルの処理 (Visual Basic)
- 方法 : Visual Basic で XML ドキュメントを作成する
- XML コメントのサンプル
- Visual Studio の XML ツール

参照

その他の技術情報

Visual Basic でのアプリケーションの開発
Visual Basic のプログラミング ガイド

XML ファイルの処理 (Visual Basic)

コンパイラは、ドキュメントを生成するためにタグ付けされたコードの構成体ごとに、ID 文字列を生成します。(コードにタグを付ける方法については、「ドキュメント コメントとして推奨される XML タグ (Visual Basic)」を参照してください)。ID 文字列によって、構成体は一意に識別されます。XML ファイルを処理するプログラムは、ID 文字列を使用して、ドキュメントの適用対象となる .NET Framework メタデータ/リフレクション項目を識別できます。

XML ファイルは、コードの階層的表現ではなく、要素ごとに生成された ID のあるフラットなリストです。

コンパイラは、次の規則に基づいて ID 文字列を生成します。

- 生成される文字列に空白は含まれません。
- ID 文字列の最初の部分には、単一の文字とコロンで、識別されるメンバの種類を示します。使用されるメンバ型は次のとおりです。

文字	説明
N	名前空間。 名前空間にはドキュメント コメントを追加できませんが、サポートされている場合は、ドキュメント コメントへの CREF 参照を作成できます。
T	型 : **Class**、**Module**、**Interface**、**Structure**、**Enum**、**Delegate**
F	フィールド : **Dim**
P	プロパティ : **Property** (既定のプロパティを含む)
M	メソッド : **Sub**、**Function**、**Declare**、**Operator**
E	イベント : **Event**
!	エラー文字列 エラーに続く文字列で、エラーの内容を示します。Visual Basic コンパイラは、解決できないリンクについてのエラー情報を生成します。

- **String** の 2 番目の部分は、項目の完全修飾名です。名前は、名前空間のルートから始まります。項目の名前、項目が含まれている型、および名前空間は、ピリオド (.) で区切られます。項目の名前自体にピリオドが含まれている場合は、シャープ記号 (#) に置き換えられます。項目の名前にはシャープ記号がないことが前提です。たとえば、**String** コンストラクタの完全修飾名は `System.String.#ctor` となります。
- プロパティおよびメソッドについては、メソッドに引数がある場合は、引数のリストをかっこで囲み、メソッドに続けて指定します。引数がない場合は、かっこはありません。引数の区切り文字には、コンマを使用します。各引数のエンコーディングは、.NET Framework のシグネチャでの引数のエンコーディング方法にそのまま従います。

使用例

次のコードは、クラスおよびそのメンバの ID 文字列がどのように生成されるかを示します。

VB

```
Namespace SampleNamesapce

    ''' <summary>Signature is
    ''' "T:SampleNamesapce.SampleClass"
    ''' </summary>
    Public Class SampleClass

        ''' <summary>Signature is
        ''' "M:SampleNamesapce.SampleClass.#ctor"
        ''' </summary>
```

```vb
Public Sub New()
End Sub

''' <summary>Signature is
''' "M:SampleNamesapce.SampleClass.#ctor(System.Int32)"
''' </summary>
Public Sub New(ByVal i As Integer)
End Sub

''' <summary>Signature is
''' "F:SampleNamesapce.SampleClass.SampleField"
''' </summary>
Public SampleField As String

''' <summary>Signature is
''' "F:SampleNamesapce.SampleClass.SampleConstant"
''' </summary>
Public Const SampleConstant As Integer = 42

''' <summary>Signature is
''' "M:SampleNamesapce.SampleClass.SampleFunction"
''' </summary>
Public Function SampleFunction() As Integer
End Function

''' <summary>Signature is
''' "M:SampleNamesapce.SampleClass.
''' SampleFunction(System.Int16[],System.Int32[0:,0:])"
''' </summary>
Public Function SampleFunction( _
    ByVal array1D() As Short, _
    ByVal array2D(,) As Integer) _
    As Integer
End Function

''' <summary>Signature is
''' "M:SampleNamesapce.SampleClass.
''' op_Addition(SampleNamesapce.SampleClass,
'''             SampleNamesapce.SampleClass)"
''' </summary>
Public Shared Operator +( _
    ByVal operand1 As SampleClass, _
    ByVal operand2 As SampleClass) _
    As SampleClass

    Return Nothing
End Operator

''' <summary>Signature is
''' "P:SampleNamesapce.SampleClass.SampleProperty"
''' </summary>
Public Property SampleProperty() As Integer
    Get
    End Get
    Set(ByVal value As Integer)
    End Set
End Property

''' <summary>Signature is
''' "P:SampleNamesapce.SampleClass.Item(System.String)"
''' </summary>
Default Public ReadOnly Property Item( _
    ByVal s As String) As Integer

    Get
    End Get
End Property
```

```
        ''' <summary>Signature is
        ''' "T:SampleNamesapce.SampleClass.NestedClass"
        ''' </summary>
        Public Class NestedClass
        End Class

        ''' <summary>Signature is
        ''' "E:SampleNamesapce.SampleClass.SampleEvent(System.Int32)"
        ''' </summary>
        Public Event SampleEvent As SampleDelegate

        ''' <summary>Signature is
        ''' "T:SampleNamesapce.SampleClass.SampleDelegate"
        ''' </summary>
        Public Delegate Sub SampleDelegate(ByVal i As Integer)
    End Class
End Namespace
```

参照

処理手順

方法 : Visual Basic で XML ドキュメントを作成する

関連項目

/doc

方法：Visual Basic で XML ドキュメントを作成する

この例は、XML ドキュメントのコメントをコードに追加する方法を示しています。

型またはメンバに対して XML ドキュメントを作成するには

1. コード エディタで、ドキュメントを作成する対象の型またはメンバの上の行にカーソルを置きます。
2. ''' (3 つの単一引用符) を入力します。

   型またはメンバに対する XML スケルトンがコード エディタに追加されます。
3. 適切なタグの間に説明情報を追加します。

メモ：
XML ドキュメント ブロック内に行を追加した場合、各行の先頭は ''' である必要があります。

4. XML ドキュメントの新しいコメントを指定した型またはメンバを使用するコードを追加します。

   IntelliSense により、当該の型またはメンバに対する <summary> タグのテキストが表示されます。
5. コードをコンパイルし、ドキュメントのコメントを含む XML ファイルを生成します。詳細については、「/doc」を参照してください。

参照

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

/doc

概念

XML の使用によるコードのドキュメントの作成 (Visual Basic)

Visual Basic の制限事項

以前のバージョンの Visual Basic では、コードに対して、変数名の長さ、モジュールで許可される変数の数、モジュール サイズなどの制限が課されていました。Visual Basic 2005 ではこれらの制限が緩和され、コードの記述と配置がより自由になりました。

物理的な制限は、コンパイル時の環境よりも実行時のメモリに左右される部分が大きくなりました。慎重なプログラミング習慣に従い、大きなアプリケーションを複数のクラスとモジュールに分割していれば、Visual Basic の内部的な制限にぶつかることはほとんどありません。

極端なケースで遭遇し得るいくつかの制限を次に示します。

- **名前の長さ。** すべての宣言されるプログラミング要素には、名前の文字数に上限があります。要素名が修飾されている場合は、修飾された文字列全体に対して制限が適用されます。「宣言された要素の名前」を参照してください。
- **行の長さ。** ソース コードの物理行の最大文字数は 65535 文字です。行連結文字を使用すると、それよりも長い論理ソース コード行を記述できます。「方法 : コード内でステートメントを分割および連結する」を参照してください。
- **配列の次元。** 配列で宣言できる次元の数には上限があります。これにより、配列要素を指定するときに使用できるインデックスの数が制限されます。「Visual Basic における配列の次元」を参照してください。
- **文字列の長さ。** 1 つの文字列に格納できる Unicode 文字数には上限があります。「文字列型 (String) (Visual Basic)」を参照してください。
- **環境文字列の長さ。** コマンド ライン引数として使用される環境文字列の最大文字数は 32768 文字です。これは全プラットフォームにおける制限です。「方法 : コマンド ライン引数にアクセスする (Visual Basic)」を参照してください。

参照

概念

Visual Basic の名前付け規則

その他の技術情報

プログラム構造とコード規則

Visual Basic 言語の機能

次に示す各トピックでは、オブジェクト指向プログラミング言語である Visual Basic に不可欠なコンポーネントを紹介し、説明します。フォームおよびコントロールを使用してアプリケーションのユーザー インターフェイスを作成した後で、アプリケーションの動作を定義するコードを記述する必要があります。他の最新のプログラミング言語と同様、Visual Basic は、多くの一般的なプログラミング構成要素と言語要素をサポートします。

他の言語でプログラムを作成した経験がある場合、このセクションで説明されている内容の多くが、既知の内容である可能性もあります。Visual Basic 言語の構成要素のほとんどは他の言語と似ていますが、イベント ドリブンの性質を持つため、いくつかの微妙な違いがあります。

プログラミングの経験がない場合、ここで扱う内容は、コードを記述するときの基礎になります。基本的な事項を理解すると、Visual Basic を使用して強力なアプリケーションを作成できます。

このセクションの内容

Visual Basic と .NET Framework

.NET Framework における Visual Basic の役割について説明し、アセンブリと属性について解説します。

Visual Basic における宣言された要素

宣言可能なプログラミング要素と、各要素の名前、特徴、およびコンパイラがプログラミング要素への参照を解決する方法について説明します。

Visual Basic におけるデータ型

プログラミング要素が保持できるデータの種類とそのデータの保存方法について説明します。

Visual Basic における変数

変数を宣言して値を格納する方法、およびこれらを使用してオブジェクトを参照する方法について説明します。

Visual Basic における配列

複数の関連する値を格納する配列を宣言して使用することによって、コードをよりコンパクトで強力にする方法について説明します。

Visual Basic におけるオブジェクト

オブジェクトとクラスの概要、使用方法、および相互関係について説明します。また、オブジェクトとクラスによって公開されるプロパティ、メソッド、およびイベントについても説明します。

Visual Basic における文字列

一連の文字を持つ変数の宣言方法および操作方法について説明します。

Visual Basic の定数と列挙体

不変の値を繰り返し使用するために格納することについて、関連する定数値のセットなどを説明します。

Visual Basic の演算子および式

値を保持する要素を操作するコード要素、それらを効果的に使用する方法、およびそれらを組み合わせて新しい値を取得する方法について説明します。

Visual Basic におけるステートメント

Visual Basic の命令の基本単位である、宣言ステートメントと実行ステートメントの概要について説明します。

Visual Basic におけるプロシージャ

Sub、**Function**、**Property**、および **Operator** の各プロシージャについて説明します。また、再帰プロシージャやオーバーロードされたプロシージャなどの高度なトピックについても説明します。

Visual Basic における制御フロー

プログラムの実行フローを制御する方法について説明します。

関連するセクション

Visual Basic におけるオブジェクト指向プログラミング

オブジェクト指向プログラミングについて、クラスやインターフェイス、オブジェクトの作成、イベントやデリゲート、継承、およびオブジェクトのグループなどを説明します。

Visual Basic リファレンス

Visual Basic プログラミングのさまざまな側面に関するリファレンス情報を提供します。

Visual Basic と .NET Framework

Visual Basic は .NET Framework を基礎としてデザインされており、これがセキュリティの向上、メモリ管理、バージョン管理、および配置を可能にしています。.NET Framework では、.NET Framework プログラミング言語を使って作成したオブジェクト間の相互運用性もサポートされています。このため、他の .NET Framework 言語で簡単に使用できるオブジェクトを Visual Basic で作成したり、他の .NET Framework 言語のオブジェクトを Visual Basic で作成したオブジェクトとまったく同じように使用したりできます。

このセクションの内容

アセンブリ

アセンブリを定義し、使い方を説明します。

参照と Imports ステートメント

他のアセンブリのオブジェクトを参照する方法を説明します。

Visual Basic における名前空間

アセンブリ内のオブジェクトが名前空間でどのように編成されているのかを説明します。

Visual Basic における属性

Visual Basic アプリケーション内で定義されているエンティティに関する追加情報が、.NET Framework 属性によって提供されるしくみについて説明します。

関連するセクション

方法 : Visual Studio のリファレンスを追加または削除する

外部コンポーネントに対するコードを記述できるように参照を追加する方法、および不要な参照を削除する方法を説明します。

.NET Framework 概念の概要

.NET Framework のアーキテクチャとコンポーネントについて説明します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

変更された言語要素の一覧を示して説明します。

アセンブリ

アセンブリは、.NET ベースのアプリケーションの配置、バージョン管理、再利用、アクティベーション スコープ、およびセキュリティ権限の基本単位を形成します。アセンブリは実行可能 (.exe) ファイルまたはダイナミック リンク ライブラリ (.dll) ファイルであり、.NET Framework のビルド ブロックとして機能します。共通言語ランタイムは、実装されている型を認識するために必要な情報をアセンブリから受け取ります。アセンブリは、機能の論理単位を形成し、ビルドされることによって連係して機能する、型およびリソースの集まりと見なすことができます。

Visual Basic では、以前のバージョンの Visual Basic におけるタイプ ライブラリとほとんど同じように、アセンブリの内容を使用したりアセンブリへの参照を追加したりします。ただし、アセンブリには、タイプ ライブラリに含まれている情報だけでなく、アプリケーションやコンポーネントを使用するときに必要なあらゆる情報が含まれています。この点で、以前のバージョンの Windows の .exe ファイルや .dll ファイルとは異なります。

アセンブリ マニフェスト

すべてのアセンブリに、アセンブリ マニフェストがあります。アセンブリ マニフェストは目次のようなものであり、次の情報が含まれています。

- アセンブリの ID (名前とバージョン)。
- アセンブリを構成している他のすべてのファイルの情報を含むファイル テーブル。たとえば、.exe ファイルや .dll ファイルが依存している作成済みのアセンブリ、ビットマップ ファイル、Readme ファイルなどがあります。
- アセンブリ参照リスト。アプリケーションで必要な .dll やその他のファイルなど、すべての外部依存関係のリストです。他のユーザーによって作成されている場合もあります。アセンブリ参照には、グローバル オブジェクトへの参照とプライベート オブジェクトへの参照の両方が含まれます。グローバル オブジェクトはグローバル アセンブリ キャッシュに常駐しています。グローバル アセンブリ キャッシュとは、System32 ディレクトリのように、他のアプリケーションも使用できる領域です。グローバル アセンブリ キャッシュ内のアセンブリには、たとえば Microsoft.VisualBasic 名前空間などがあります。プライベート オブジェクトは、アプリケーションがインストールされるディレクトリと同じレベルか、またはそれより下のディレクトリにあることが必要です。

アセンブリにはコンテンツ、バージョン管理、および依存関係に関する情報が含まれているため、Visual Basic で作成したアプリケーションは、レジストリ値に依存せずに適切に機能します。アセンブリによって DLL の競合が減少するため、アプリケーションの信頼性が向上し、配置も簡単になります。多くの場合、.NET ベースのアプリケーションは、ファイルを目的のコンピュータにコピーするだけでインストールできます。

参照

アセンブリを使用するには、アセンブリへの参照を追加する必要があります。詳しくは「方法 : Visual Studio のリファレンスを追加または削除する」を参照してください。次に、**Imports** ステートメントを使って、使用するアイテムの名前空間を選択します。この方法については「参照と Imports ステートメント」を参照してください。アセンブリが参照され、インポートされたら、その名前空間のアクセス可能なすべてのクラス、プロパティ、メソッド、およびその他のメンバが (そのコードがまるでソース ファイルの一部であるかのように) アプリケーションから利用できます。1 つのアセンブリに複数の名前空間を含めることができます。また、各名前空間には、他の名前空間を含めて、それぞれ異なる項目グループを含めることができます。

アセンブリの作成については、「方法: アセンブリを作成および使用する」を参照してください。

参照

処理手順

方法: アセンブリを作成および使用する
方法 : Visual Studio のリファレンスを追加または削除する
方法 : アセンブリの内容を表示する

関連項目

Imports ステートメント
Microsoft.VisualBasic

概念

Visual Basic における名前空間
参照と Imports ステートメント

その他の技術情報

共通言語ランタイムのアセンブリ
言語の変更点 (Visual Basic 6.0 ユーザー向け)

方法: アセンブリを作成および使用する

アセンブリは、Visual Studio ベースのアプリケーションを構成する 1 つ以上の .exe ファイルまたは .dll ファイルです。アセンブリは、Visual Basic ソース ファイルのコンパイル時に自動的に作成されます。

メモ :

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

アセンブリを作成するには

- [ビルド] メニューの [ビルド] をクリックするか、またはコマンド ラインでコマンド ライン コンパイラを使って、アプリケーションをコンパイルします。コマンド ラインからアセンブリをビルドする方法の詳細については、「コマンド ラインからのビルド (Visual Basic)」を参照してください。

別のアセンブリの参照を追加するには

- [プロジェクト] メニューの [参照の追加] をクリックし、使用するアセンブリを選択します。詳細については、「方法 : Visual Studio のリファレンスを追加または削除する」を参照してください。

他のアセンブリのオブジェクトを使用するには

- オブジェクトの完全限定名を指定するか、または **Imports** ステートメントを使ってオブジェクトに定義したエイリアスを使用します。完全修飾名の詳細については、「Visual Basic における名前空間」を参照してください。参照の追加と **Imports** ステートメントの使用の詳細については、「参照と Imports ステートメント」を参照してください。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する
方法 : アセンブリの内容を表示する

関連項目

Imports ステートメント

概念

アセンブリ
アセンブリの作成
Visual Basic における名前空間
参照と Imports ステートメント

その他の技術情報

共通言語ランタイムのアセンブリ
言語の変更点 (Visual Basic 6.0 ユーザー向け)
コマンド ラインからのビルド (Visual Basic)

参照と Imports ステートメント

プロジェクトで外部オブジェクトを使用できるようにするには、[プロジェクト] メニューの [参照の追加] をクリックします。Visual Basic の参照では、アセンブリを指すことができます。アセンブリは、タイプ ライブラリに似ていますが、タイプ ライブラリより多くの情報を含んでいます。

Imports ステートメント

アセンブリには名前空間があります。アセンブリへの参照を追加するときに、モジュール内におけるそのアセンブリの名前空間の参照可能範囲を制御する **Imports** ステートメントをモジュールに追加することもできます。**Imports** ステートメントを使用すると、スコープのコンテキストを利用できるようになるため、名前空間の一部だけを使って参照を一意にできます。

Imports ステートメントの構文は次のとおりです。

Imports [*Aliasname* =] *Namespace*

Aliasname には、インポートした名前空間を参照するためにコード内で使用する短い名前を指定します。*Namespace* には、プロジェクト参照、プロジェクト内の定義、または前の **Imports** ステートメントを通じて使用できる名前空間を指定します。

1 つのモジュールで使用できる **Imports** ステートメントの数に制限はありません。**Imports** ステートメントは、**Option**ステートメントとその他のコードとの間に記述する必要があります (**Option** ステートメントがある場合)。

メモ :

プロジェクト参照を **Imports** ステートメントや **Declare** ステートメントと混同しないようにしてください。プロジェクト参照は、アセンブリ内のオブジェクトなど、外部オブジェクトを Visual Basic プロジェクトで使用できるようにします。**Imports** ステートメントは、プロジェクト参照へのアクセスを簡単にするためのものであり、**Imports** ステートメントを使って外部オブジェクトにアクセスすることはできません。**Declare** ステートメントは、ダイナミック リンク ライブラリ (DLL) の外部プロシージャへの参照を宣言するときに使用します。

Imports ステートメントでのエイリアスの使用

Imports ステートメントを使用すると、参照の完全限定名を明示的に入力する必要がなくなるため、クラスのメソッドへのアクセスが簡単になります。エイリアスを使用すると、名前空間の一部に覚えやすい名前を割り当てることができます。たとえば、1 つの文字列を複数の行に表示するキャリッジ リターンとライン フィードは、Microsoft.VisualBasic 名前空間にある ControlChars モジュール (Visual Basic) モジュールの一部です。この定数をエイリアスを使わずにプログラムで使用するには、次のコードを入力する必要があります。

VB

```
MsgBox("Some text" & Microsoft.VisualBasic.ControlChars.CrLf _
    & "Some more text")
```

Imports ステートメントは、モジュールの **Option** ステートメントの次の行に記述する必要があります。Microsoft.VisualBasic.ControlChars モジュールをインポートして、エイリアスを割り当てる例は、次のコード片のとおりです。

VB

```
Imports CtrlChrs = Microsoft.VisualBasic.ControlChars
```

これ以降、この名前空間を参照する場合は、次のように入力するだけで済みます。

VB

```
MsgBox("Some text" & CtrlChrs.CrLf & "Some more text")
```

Imports ステートメントでエイリアス名を指定しなかった場合は、インポートした名前空間で定義されている要素を修飾子を付けずにモジュール内で使用できます。エイリアス名を指定した場合は、その名前空間に含まれている名前の修飾子としてエイリアス名を使用する必要があります。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する

方法: アセンブリを作成および使用する

関連項目

Imports ステートメント
ControlChars モジュール (Visual Basic)
Microsoft.VisualBasic

概念

Visual Basic における名前空間
アセンブリ

その他の技術情報

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic における名前空間

アセンブリ内で定義されているオブジェクトは、名前空間によって編成されています。アセンブリには複数の名前空間を含めることができます。さらに、名前空間の中に他の名前空間を含めることもできます。名前空間を使用するとあいまいさがなくなるため、多数のオブジェクトを使用する場合 (クラス ライブラリを使用する場合など) に参照が簡単になります。

たとえば、.NET Framework では System.Windows.Forms 名前空間の中に ListBox クラスが定義されています。このクラスの完全限定名を使って変数を宣言するコードは、たとえば次のようになります。

VB

```
Dim LBox As System.Windows.Forms.ListBox
```

名前の衝突を避けるには

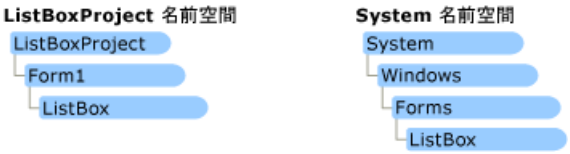
.NET Framework の名前空間は、他のライブラリで似た名前が使用されている場合にクラス ライブラリの開発者が遭遇する、名前空間の汚染と呼ばれる問題に対処しています。このような既存コンポーネントとの競合は、名前の衝突とも呼ばれます。

たとえば、`ListBox` という名前の新しいクラスを作成した場合、このクラスをプロジェクト内で使用する場合は、修飾子を付ける必要はありません。しかし、同じプロジェクトで .NET Framework の **ListBox** クラスを使用する場合は、参照を一意にするために完全修飾参照を使用する必要があります。参照が一意でないと、Visual Basic は、名前があいまいであることを指摘するエラーを生成します。この 2 つのオブジェクトを宣言するコードとしては、次のような例があります。

VB

```
' Define a new object based on your ListBox class.
Dim LBC As New ListBox
' Define a new Windows.Forms ListBox control.
Dim MyLB As New System.Windows.Forms.ListBox
```

次の図は、いずれも `ListBox` という名前のオブジェクトを持つ、2 つの名前空間の階層を表しています。



Visual Basic で作成するすべての実行可能ファイルには、既定でプロジェクトと同名の名前空間が含まれます。たとえば、`ListBoxProject` という名前のプロジェクト内でオブジェクトを定義した場合、実行可能ファイル ListBoxProject.exe には `ListBoxProject` という名前空間が含まれます。

複数のアセンブリで同じ名前空間を使用することができます。Visual Basic はこれらを 1 つの名前セットとして扱います。たとえば、`Assemb1` というアセンブリの `SomeNameSpace` という名前空間のクラスを定義した後に、`Assemb2` というアセンブリの同じ名前空間のクラスを定義できます。

完全修飾名

完全修飾名とは、オブジェクトが定義されている名前空間の名前で始まるオブジェクト参照です。他のプロジェクトで定義されているオブジェクトを使用するには、[プロジェクト] メニューの [参照の追加] をクリックしてそのクラスへの参照を作成し、コード内でそのオブジェクトの完全修飾名を使用します。次のコードは、完全限定名を使って他のプロジェクトの名前空間のオブジェクトを使用する方法を示しています。

VB

```
Dim LBC As New ListBoxProject.Form1.ListBox
```

完全修飾名を使用すると、どのオブジェクトを使用するかをコンパイラが正しく認識できるため、名前の衝突を防止できます。ただし、名前自体が長くなるため、使いにくくなります。この問題を回避するには、**Imports** ステートメントを使ってエイリアスを定義します。エイリアスとは、完全修飾名の代わりに使用できる短い名前です。次のコード例では、2 つの完全修飾名に対してエイリアスを作成し、作成したエイリアスを使って 2 つのオブジェクトを定義しています。

VB

```
Imports LBControl = System.Windows.Forms.ListBox
```

```
Imports MyListBox = ListBoxProject.Form1.ListBox
```

VB

```
Dim LBC As LBControl
Dim MyLB As MyListBox
```

エイリアスを指定せずに **Imports** ステートメントを使用すると、インポートした名前空間のすべての名前を修飾子を付けずに使用できます。ただし、それらの名前がプロジェクト内で一意であることが必要です。同じ名前の複数の項目を持つ名前空間をインポートする **Imports** ステートメントがある場合は、それらの名前を使用するときに完全限定名を使用する必要があります。たとえば、プロジェクトに次の 2 つの **Imports** ステートメントがあるとします。

VB

```
' This namespace contains a class called Class1.
Imports MyProj1
' This namespace also contains a class called Class1.
Imports MyProj2
```

このときに完全修飾名を使わずに `Class1` を使おうとすると、Visual Basic から、`Class1` という名前があいまいであることを指摘するエラーが生成されます。

Namespace Level ステートメント

名前空間の中では、モジュール、インターフェイス、クラス、デリゲート、列挙型、構造体、他の名前空間などの項目を定義できます。プロパティ、プロシージャ、変数、イベントなどの項目を名前空間のレベルで定義することはできません。これらの項目は、モジュール、構造体、クラスなどのコンテナ内で宣言する必要があります。

メモ：

入れ子になった階層構造の名前空間を定義すると、その階層構造内のコードが 同じ名前の別の名前空間に属するクラスにアクセスするのを防止できます。たとえば、`SpecialSpace` という名前空間内に `System` という名前空間を定義した場合、.NET Framework の **System** 名前空間のメンバには、**Global** キーワードで完全修飾しない限りアクセスできません。詳細については、「Global」を参照してください。

参照

処理手順

方法: アセンブリを作成および使用する

関連項目

Global
Imports ステートメント
ListBox
System.Windows.Forms

概念

アセンブリ
参照と Imports ステートメント

その他の技術情報

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic における属性

属性は、Visual Basic アプリケーションで定義されたエンティティに関して追加情報を指定できる、キーワードのようなタグです。属性は、アセンブリのメタデータと共に保存され、型、フィールド、メソッド、プロパティなどのプログラミング要素に注釈を付けます。Visual Basic コンパイラや ASP.NET コンパイラなどの他のアプリケーションでは、属性の情報を参照して、オブジェクトの使用方法を判断できます。

次に示す各セクションでは、属性とその定義、適用、および取得方法に関する情報について説明します。

このセクションの内容

Visual Basic における属性

属性、および属性の機能について説明します。

Visual Basic で使用される属性

Visual Basic で非常によく使用する属性をいくつか紹介します。

属性の一般的な使用方法

属性の一般的な使用方法について説明します。

属性の適用

属性をプログラム要素に適用する方法について説明します。

Visual Basic におけるグローバル属性

モジュールやアセンブリに適用する属性について説明します。

Visual Basic におけるカスタム属性

独自の属性を定義する方法について説明します。

関連するセクション

Visual Basic におけるオブジェクト指向プログラミング

オブジェクト指向プログラミング、およびその使用方法について説明します。

メタデータと自己言及的なコンポーネント

属性を含む、Visual Studio で使用されるメタデータの種類について説明します。

Visual Basic における属性

属性は、型、フィールド、メソッド、プロパティなどのプログラミング要素に関する追加情報を表す、説明的なタグです。Visual Basic コンパイラなどの他のアプリケーションでは、属性の補足情報を参照して、これらの項目の使用方法を判断できます。

属性とメタデータ

属性は、Visual Basic アセンブリのメタデータと共に保存されます。メタデータは、ランタイムで管理される要素を説明する情報です。デバッグやガベージ コレクションに必要な情報、セキュリティ属性、マーシャリング データ、拡張クラスとメンバの定義、バージョン連結、ランタイムに必要なその他の情報などが含まれます。

属性を使用して、**Public** や **Private** などのキーワードでアクセス レベルに関する情報を指定するのと同じ方法で、メタデータを指定します。ただし、キーワードの場合とは異なり、ほとんどの属性は言語に固有ではありません。属性を使用すると、コンパイラを変更することなく、Visual Basic 言語の機能を拡張できます。

.NET Framework および Visual Basic 言語は、多数の有益な属性を定義できます。また、アプリケーションに対して有効な独自のカスタム属性を定義することもできます。カスタム属性は、System.Attribute クラスに基づいており、AttributeUsageAttribute 属性を使用して、属性の使用方法に関する追加情報を提供しています。

属性の機能

属性についてはいくつかの重要な点があります。

- アセンブリ全体、モジュール全体、クラスやプロパティなどの小さいプログラム要素に、1 つ以上の属性を適用できます。
- 属性は、メソッドやプロパティの場合と同じ方法で引数を受け取ることができます。
- 属性からメタデータを取得するプロセスは、リフレクションと呼ばれます。リフレクションでは、オブジェクトがメンバに関するメタデータを取得およびチェックできるツールを使用します。詳細については、「リフレクションのサンプル」を参照してください。

参照

処理手順

方法 : 独自の属性を定義する

関連項目

AttributeUsageAttribute
System.Attribute

概念

属性の一般的な使用方法
属性の適用
Visual Basic におけるグローバル属性
Visual Basic で使用される属性
属性とデザイン時サポート

その他の技術情報

Visual Basic におけるカスタム属性

Visual Basic で使用される属性

Visual Basic で使用される一般的な属性を次の表に示します。特定の属性に関する情報については、ヘルプ システムのキーワード ボックスまたは検索ボックスに属性名を入力してください。

属性	目的
ComClassAttribute クラス	クラスを COM オブジェクトとして公開する必要があることをコンパイラに指示します。Visual Basic 固有です。
HideModuleNameAttribute クラス	モジュールに必要な修飾子のみを指定してモジュール メンバにアクセスできます。
VBFixedStringAttribute クラス	ファイルの入出力関数で使用する、構造体内の固定長文字列のサイズを指定します。Visual Basic 固有です。
VBFixedArrayAttribute クラス	ファイルの入出力関数で使用する、構造体内の固定配列のサイズを指定します。Visual Basic 固有です。
WebMethodAttribute	SOAP プロトコルを使用してメソッドを呼び出すことができるようにします。XML Web サービスで使用します。
SerializableAttribute	クラスをシリアル化できることを示します。
MarshalAsAttribute	Visual Basic のマネージ コードと、Windows API などのアンマネージ コードの間で、パラメータがどのようにマーシャリングされるかを決定します。共通言語ランタイムによって使用されます。
AttributeUsageAttribute	属性の使用方法を指定します。
DllImportAttribute	属性指定されたメソッドをアンマネージ DLL からのエクスポートとして実装することを示します。

Visual Basic 固有の属性

COMClassAttribute、**VBFixedStringAttribute**、および **VBFixedArray** の 3 つの属性が Visual Basic に固有です。

COMClassAttribute

COMClassAttribute を使うと、Visual Basic から COM コンポーネントを作成する処理を簡略化できます。COM オブジェクトは .NET Framework アセンブリとは大きく異なります。また、**COMClassAttribute** を使用せずに Visual Basic から COM オブジェクトを生成するには、多くの手順を実行する必要があります。**COMClassAttribute** が適用されているクラスでは、これらの手順の多くがコンパイラによって自動的に実行されます。

HideModuleNameAttribute

モジュールに必要な修飾子のみを使用してモジュール メンバにアクセスできるようにするには、**HideModuleNameAttribute** を使用します。

VBFixedStringAttribute

Visual Basic で強制的に固定長の文字列を作成するようにするには、**VBFixedStringAttribute** を使用します。文字列は既定では可変長です。この属性は、文字列をファイルに格納する場合に役立ちます。次のコードで例を示します。

VB

```
Structure Worker
    ' The runtime uses VBFixedString to determine
    ' if the field should be written out as a fixed size.
    <VBFixedString(10)> Public LastName As String
    <VBFixedString(7)> Public Title As String
    <VBFixedString(2)> Public Rank As String
End Structure
```

VBFixedArrayAttribute

固定サイズの配列を宣言するには、**VBFixedArrayAttribute** を使用します。Visual Basic の文字列と同様に、既定では配列は可変長です。この属性は、データをファイルにシリアル化する場合や書き込む場合に役立ちます。

参照

関連項目

ComClassAttribute クラス
VBFixedArrayAttribute クラス
VBFixedStringAttribute クラス
System.Runtime.InteropServices

概念

Visual Basic におけるグローバル属性
属性の適用
属性に格納されている情報の取得
カスタム属性の記述

その他の技術情報

属性を使用したメタデータの拡張

属性の一般的な使用方法

共通言語ランタイムおよびクラス ライブラリを使用するときは、ほとんどの場合、ある時点で属性を使用する必要があります。コードでの一般的な属性の使用方法を次に示します。

- XML Web サービスの **WebMethod** 属性をメソッドに指定すると、メソッドを SOAP プロトコルで呼び出すことを示します。詳細については、「WebMethodAttribute」を参照してください。
- ネイティブ コードと相互運用するときに、メソッドのパラメータをマーシャリングする方法を示します。詳細については、「MarshalAsAttribute」を参照してください。
- クラス、メソッド、およびインターフェイスの COM プロパティを示します。
- コンポーネントを COM として指定し、Visual Basic コンパイラが COM コンポーネントを作成するために必要な追加コードを生成するようにします。詳細については、「ComClassAttribute」を参照してください。
- DllImportAttribute クラスを使用してアンマネージ コードを呼び出します。
- アセンブリのタイトル、バージョン、説明、または商標を示します。
- 永続性をシリアル化するクラスのメンバを示します。
- XML シリアル化のクラス メンバと XML ノードの割り当て方法を示します。
- メソッドのセキュリティ要件を示します。
- セキュリティを適用するために使用する特性を指定します。
- コードをデバッグしやすいように、Just-In-Time (JIT) コンパイラによる最適化を制御します。

属性は他にも数多くの方法で使用できます。カスタム属性を作成することもできます。

参照

処理手順

方法 : 独自の属性を定義する

概念

属性とデザイン時サポート
Visual Basic における属性
属性の適用
Visual Basic におけるグローバル属性
Visual Basic で使用される属性

その他の技術情報

Visual Basic におけるカスタム属性

属性の適用

属性は、プロパティ、メソッド、イベント、クラス、アセンブリなどのプログラム要素に、属性ブロックを追加して適用します。属性ブロックは、山かっこ (< >) で囲んだコンマ区切りの属性の宣言リストで構成されます。属性の宣言は、**Module**、**Assembly** などの省略可能な属性修飾子、属性名、(通常オーバーロードされた) 必須の位置指定パラメータのリスト、および省略可能な名前付き引数のリストで構成されます。修飾子付きの属性は、ソース ファイルの先頭にある属性セクションに記述する必要があります。たとえば次のコードでは、アセンブリのタイトルを定義するアセンブリ属性と、モジュールが共通言語仕様 (CLS: Common Language Specification) 準拠であることを示すモジュール属性を設定しています。

VB

```
Imports System.Reflection
<Assembly: AssemblyTitleAttribute("Production assembly 4"), _
Module: CLSCompliant(True)>
```

アセンブリ属性は、AssemblyInfo.vb ファイルを通じても適用できます。このファイルは、Visual Studio ユーザー インターフェイスによって自動的にプロジェクトに追加されます。このファイルには、既定値または Empty 値を持つアセンブリ レベルの属性が含まれています。

プロパティなどのプログラム要素に適用する場合は、そのプログラム要素の前に属性を置きます。たとえば、次のコードは属性をクラス定義に適用しています。

VB

```
<CustomAttr(Update:=True)> Class Class1
```

規則では、属性名の終わりにはすべて "Attribute" が付きます。これにより、属性と .NET Framework の他の項目とを区別できます。ただし、属性を使用するときには、必ずしもこのサフィックスを使用する必要はありません。たとえば、`CustomAttrAttribute` という属性名の場合、`<CustomAttr(Update:=True)>` と指定しても `<CustomAttrAttribute(Update:=True)>` と指定しても同じです。

属性の引数

属性で省略可能な引数、必須の引数、位置指定引数、および名前付き引数を使用する方法は、オブジェクトでこれらの引数を使用する方法と同じです。位置指定引数は、属性のコンストラクタで宣言されている順に指定する引数です。たとえば、次のコードは、これらの 2 つの値を持つ属性の **Sub New** コンストラクタを呼び出します。

VB

```
<CustomAttr(True, False)> Class Class1
```

属性クラスの **Sub New** に渡される引数は、多くの場合、フィールドやプロパティの値を初期化するために使用されます。

名前付き引数を使用すると、プロパティやフィールドの値を直接設定できます。名前付き引数を指定するには、引数の名前の後に ":=" および指定される値を付加します。位置指定引数とは異なり、名前付き引数は任意の順序で指定できます。たとえば、次のコードは `Update` フィールドに **True** を設定し、`Keep` フィールドに **False** を設定します。

VB

```
<CustomAttr(Update:=True, Keep:=False)> Class Class1
```

メモ :

属性の引数と、標準メソッドの呼び出しで使用される引数には、重要な相違点が 1 つあります。属性クラスの **Sub New** コンストラクタで使用される引数には、位置指定引数を使用する必要があります。名前付き引数は、属性クラスのフィールドとプロパティの値を設定する場合にだけ使用できます。

必須の引数は、常に指定する必要がある引数です。省略可能な引数は、位置指定引数を使用するときに、コンマをプレースホルダに使用して省略できる引数や、名前付き引数を使用するときに単に省略できる引数です。

属性の引数は定数式であることが必要です。

属性の例

次のプロシージャは、属性の宣言の例です。

MarshalAs 属性を使用して、パラメータをマーシャリングする方法を制御するには

1. System.Runtime.InteropServices 名前空間の **Imports** ステートメントをソース コードの先頭に追加します。

   VB

   ```
   Imports System.Runtime.InteropServices
   ```

2. パラメータに MarshalAsAttribute 属性のプリフィックスを付けて、必要なデータ型を指定します。たとえば、次のコードでは、2 つのパラメータを Windows API 関数用の文字列への long ポインタ データ型 (**LPStr**) としてマーシャリングします。

   VB

   ```
   Declare Auto Sub CopyFile Lib "Kernel32.Lib" ( _
      <MarshalAs(UnmanagedType.LPStr)> ByVal existingfile As String, _
      <MarshalAs(UnmanagedType.LPStr)> ByVal newfile As String, _
      ByVal failifexists As Boolean _
   )
   ```

共通言語ランタイムは、**MarshalAsAttribute** 属性を使用して、Visual Basic のマネージ コードと、Windows API 呼び出しのアンマネージ コード間で、パラメータがどのようにマーシャリングされるかを決定します。

リモート Web クライアントにメソッドを公開するには

1. [ファイル] メニューの [プロジェクト] をポイントし、[ASP.NET Web サービス] テンプレートをクリックします。System.Web 名前空間に **Imports** ステートメントを追加します。

   VB

   ```
   Imports System.Web.Services
   ```

2. メソッドを定義し、WebMethodAttribute 属性を使用して、メソッドをリモートの Web クライアントから呼び出すことができるようにします。

   VB

   ```
   <WebMethod()> Public Function HelloWorld() As String
       HelloWorld = "Hello World..."
   End Function
   ```

XML Web サービスのメソッドを **Public** に指定するだけでは、Web クライアントに公開したことにはなりません。リモート Web クライアントから呼び出すことができるようにするには、メソッドに **WebMethodAttribute** 属性を明示的に適用する必要があります。

参照

処理手順

方法 : 独自の属性を定義する

関連項目

Imports ステートメント
WebMethodAttribute
MarshalAsAttribute
System.Web

概念

Visual Basic における属性
属性の一般的な使用方法
位置と名前による引数渡し

その他の技術情報

Visual Basic におけるカスタム属性

Visual Basic におけるグローバル属性

ほとんどの属性は、クラスやメソッドなどの特定の言語要素に結びつけられます。ただし、いくつかの属性はグローバルで、アセンブリ全体またはモジュール全体に適用できます。

Visual Studio 統合開発環境 (IDE: Integrated Development Environment) 内の属性の多くは、[アセンブリ情報] ダイアログ ボックスをとおして設定できます。詳細については、「アプリケーション プロパティの管理」および「アセンブリおよびマニフェストへの署名の管理」を参照してください。

アセンブリ属性

アセンブリ レベルの属性は次の構文を使用して指定します。

```
<Assembly: Attribute1, Assembly: Attribute2..., Assembly: AttributeN>
```

モジュール レベルの属性も同様の構文を使用して指定します。

```
<Module: Attribute1, Module: Attribute2..., Module: AttributeN>
```

グローバル属性は、ソース コードの **Option Explicit** ステートメントや **Imports** ステートメントなどのトップ レベル ディレクティブの後に記述します。ただし、型や名前空間の宣言よりも前に記述します。グローバル属性はプロジェクト内の複数のソース ファイルに記述できますが、通常は、Visual Basic プロジェクトと共に自動的に作成される AssemblyInfo.vb ファイルに記述します。

アセンブリ属性は、アセンブリについての情報を提供する値です。アセンブリ属性は、以下のカテゴリに分けられます。

- アセンブリ ID 属性
- 情報属性
- アセンブリ マニフェスト属性
- 厳密な名前の属性

アセンブリ ID 属性

名前、バージョン、およびカルチャの 3 つの属性 (適用可能な場合は厳密な名前で) で、アセンブリの ID が決定されます。これらの属性は、アセンブリの完全な名前を形成し、コードでアセンブリを参照する場合に必要です。アセンブリのバージョンとカルチャは、属性を使用して設定できます。ただし、名前の値は、コンパイラ、Visual Studio IDE ([アセンブリ情報] ダイアログ ボックス 内で)、またはアセンブリ リンカ (Al.exe) によって、アセンブリが作成されるときにアセンブリ マニフェストを含むファイルに基づいて設定されます。AssemblyFlagsAttribute 属性は、アセンブリの複数のコピーが共存できるかどうかを指定します。

次の表は、ID 属性を示しています。

属性	目的
AssemblyName	アセンブリの ID を完全に記述します。
AssemblyVersionAttribute	アセンブリのバージョンを指定します。
AssemblyCultureAttribute	アセンブリがサポートするカルチャを指定します。
AssemblyFlagsAttribute	同一のコンピュータ上、同じプロセス内、または同じアプリケーション ドメイン内で、アセンブリが side-by-side 実行をサポートするかどうかを指定します。

次のコードは、アセンブリにバージョン属性とカルチャ属性を適用します。

VB

```
'Set version number for assembly.
<Assembly: Reflection.AssemblyVersionAttribute("4.3.2.1")>
'Set culture as German.
<Assembly: Reflection.AssemblyCultureAttribute("de")>
```

情報属性

情報属性は、アセンブリに追加の会社情報または製品情報を指定する場合に使用できます。次の表は、System.Reflection 名前空間で定義されている情報属性を示しています。

属性	目的
AssemblyProductAttribute	アセンブリ マニフェストの製品名を指定するカスタム属性を定義します。
AssemblyTrademarkAttribute	アセンブリ マニフェストの商標を指定するカスタム属性を定義します。
AssemblyInformationalVersionAttribute	アセンブリ マニフェストの補足バージョンを指定するカスタム属性を定義します。
AssemblyCompanyAttribute	アセンブリ マニフェストの会社名を指定するカスタム属性を定義します。
AssemblyCopyrightAttribute	アセンブリ マニフェストの著作権を指定するカスタム属性を定義します。
AssemblyFileVersionAttribute	Win32 ファイル バージョン リソースに特定のバージョン番号を使用するように、コンパイラに指示します。
CLSCompliantAttribute	アセンブリが共通言語仕様 (CLS: Common Language Specification) に準拠しているかどうかを示します。

アセンブリ マニフェスト属性

アセンブリ マニフェスト属性を使用すると、タイトル、説明、既定のエイリアス、構成など、アセンブリ マニフェストの情報を提供します。次の表は、**System.Reflection** 名前空間で定義されているアセンブリ マニフェスト属性を示しています。

属性	目的
AssemblyTitleAttribute	アセンブリ マニフェストのアセンブリ タイトルを指定するカスタム属性を定義します。
AssemblyDescriptionAttribute	アセンブリ マニフェストのアセンブリ説明を指定するカスタム属性を定義します。
AssemblyConfigurationAttribute	アセンブリ マニフェストのアセンブリ構成 (リリース、デバッグなど) を指定するカスタム属性を定義します。
AssemblyDefaultAliasAttribute	アセンブリ マニフェストのわかりやすい既定のエイリアスを定義します。

厳密な名前の属性

厳密な名前は、アセンブリの ID と整合性を保護する一意の識別子です。アセンブリの署名は、Visual Studio IDE 内から [署名] ページ (プロジェクト デザイナ)をとおして行うことができます。詳細については、「アセンブリおよびマニフェストへの署名の管理」を参照してください。

また、厳密な名前の属性を使用して、アセンブリに厳密な名前を設定することもできます。次の表は、**System.Reflection** 名前空間で定義されている厳密な名前の属性を示しています。

属性	目的
AssemblyDelaySignAttribute	厳密な名前の署名用に実行ファイル内に予約領域を確保するかどうかを指定するブール型のクラスですが、実際の署名は後の段階で行われます。詳細については、「アセンブリへの遅延署名」を参照してください。
AssemblyKeyFileAttribute	キーを含むファイルを示します。KeyFile の場所は、プロジェクトの出力ディレクトリ (%Project Directory%\obj\<configuration>) からの相対パスで指定する必要があります。たとえば、KeyFile がプロジェクト ディレクトリに置かれている場合、AssemblyKeyFile 属性は次のように指定します。 [assembly: AssemblyKeyFile("..\\..\\mykey.snk")]
AssemblyKeyNameAttribute	コンピュータ上の暗号化サービス プロバイダ (CSP: Crypto Service Provider) にインストールされているキーを参照します。ファイルが署名されるように、キーを指定する必要があります。

KeyFile の値と KeyName の値を両方指定した場合は、次の処理が行われます。

- KeyName が CSP で見つかった場合は、そのキーが使用されます。
- KeyName が存在せず、KeyFile が存在する場合、KeyFile のキーが CSP にインストールされ、使用されます。
- 詳細については、「アセンブリのセキュリティに関する考慮事項」を参照してください。

アセンブリの署名

アセンブリに署名するには、厳密な名前を使用するか、署名を使用するという、相互に補完的な 2 種類の方法があります。署名は Visual Studio IDE の[署名] ページ (プロジェクト デザイナ)またはファイル署名ツール (Signcode.exe) を使用して行うことができます。厳密な名前を使用してアセンブリに署名すると、アセンブリ マニフェストを格納しているファイルに公開キー暗号化が追加されます。厳密な名前による署名では、名前の一意性の検証を支援し、名前の悪用を防止し、参照が解決されたときに呼び出し元に ID を提供できます。詳細については、「アセンブリおよびマニフェストへの署名の管理」および「方法 : 厳密な名前でアセンブリに署名する」を参照してください。

厳密な名前を使用してアセンブリに署名するには

1. 厳密名ツール (Sn.exe) を使用して、キー ペアを含むキー ファイルを作成します。
2. Visual Basic プロジェクトで自動的に作成された AssemblyInfo.vb ファイルに、厳密な名前の属性を追加します。このファイルを編集するには、ソリューション エクスプローラでファイル名をダブルクリックします。

   次の例では、遅延署名を使用して、`myKey.snk` という公開キー ファイルを持つ、厳密な名前が設定されたアセンブリを作成します。

   VB

```
<Assembly: Reflection.AssemblyKeyFile("myKey.snk")>
<Assembly: Reflection.AssemblyDelaySign(True)>
```

メモ :

コマンド ラインから **/target:module** オプションを使用してコンパイルする場合など、アセンブリを作成しない場合には、アセンブリ レベルの属性は無視されます。

参照

処理手順

方法 : 独自の属性を定義する
方法 : 厳密な名前でアセンブリに署名する

関連項目

厳密名ツール (Sn.exe)
ファイル署名ツール (Signcode.exe)
[署名] ページ (プロジェクト デザイナ)
[アセンブリ情報] ダイアログ ボックス

概念

属性の適用
アセンブリのセキュリティに関する考慮事項

その他の技術情報

Visual Basic におけるカスタム属性
アプリケーション プロパティの管理
アセンブリおよびマニフェストへの署名の管理

Visual Basic におけるカスタム属性

カスタム属性は、プログラム要素に関する追加情報を提供するユーザー定義の属性です。たとえば、プロシージャを実行するために呼び出し元で必要なアクセス許可を指定する、カスタム セキュリティ属性を定義できます。

カスタム属性は、System.Attribute クラスに基づく属性クラスで定義します。属性クラス自身は、AttributeUsageAttribute と呼ばれる属性を使用して、属性の使用方法に関する情報を提供します。`Inherited = True` を指定すると、属性が派生クラスにも反映されることを示します。AllowMultiple プロパティを **True** に設定すると、属性の 1 つ以上のインスタンスを 1 つのプログラム要素に適用できます。AttributeTargets 列挙型を使用すると、属性を適用できるプログラム要素の種類を定義できます。

次のコードの **AttributeUsageAttribute** 属性は、属性が任意の型の項目に適用され、継承が有効で、属性は一度だけ適用されることを示します。

VB

```
<AttributeUsage(AttributeTargets.All, Inherited:=True, AllowMultiple:=False)> _
Class TestAttribute1
    Inherits Attribute
End Class
```

Or 演算子を使用すると、**AttributeTargets** 列挙型の複数の項目を組み合わせることができます。次に例を示します。

VB

```
<AttributeUsage(AttributeTargets.Class Or AttributeTargets.Method)> _
Class TestAttribute2
    Inherits Attribute
End Class
```

このセクションの内容

方法 : 独自の属性を定義する

属性クラスを使用して独自の属性を作成する方法を説明します。

方法 : カスタム属性を取得する

GetCustomAttribute または GetCustomAttributes を使用してカスタム属性を取得する方法を示します。

カスタム属性の使用例

クラスにだけ適用可能なカスタム属性を定義するコード例を示し、新しい属性を使用する方法を説明します。

関連するセクション

Visual Basic と .NET Framework

.NET Framework における Visual Basic の役割を記述します。

Visual Basic におけるオブジェクト指向プログラミング

オブジェクト指向プログラミング、およびその使用方法について説明します。

メタデータと自己言及的なコンポーネント

属性を含む、Visual Studio で使用されるメタデータの種類について説明します。

方法：独自の属性を定義する

属性クラスを使用してカスタム属性を作成し、.NET Framework 属性に追加して使用することによって、プログラム要素に関する追加情報を提供できます。

カスタム属性を定義するには

1. クラスを宣言し、そのクラスに AttributeUsageAttribute 属性を適用します。次のコードに示すように、クラスの名前は新しい属性の名前になります。

   VB

   ```
   <AttributeUsage(AttributeTargets.All)> Class TestAttribute
   ```

2. そのクラスが System.Attribute から継承することを宣言します。

   VB

   ```
   Inherits System.Attribute
   ```

3. プロパティ値を格納する **Private** フィールドを定義します。

   VB

   ```
   Private m_SomeValue As String
   ```

4. 必要に応じて、属性のコンストラクタを作成します。

   VB

   ```
   Public Sub New(ByVal Value As String)
       m_SomeValue = Value
   End Sub
   ```

5. 属性のメソッド、フィールド、およびプロパティを定義します。

   VB

   ```
   Public Sub Attr(ByVal AttrValue As String)
       'Add method code here.
   End Sub
   Public Property SomeValue() As String  ' A named parameter.
       Get
           Return m_SomeValue
       End Get
       Set(ByVal Value As String)
           m_SomeValue = Value
       End Set
   End Property
   ```

6. **End Class** でクラスを終了します。

   VB

   ```
   End Class
   ```

参照

関連項目

AttributeUsageAttribute

概念

属性の適用

オブジェクトの有効期間 : オブジェクトの作成と破棄

方法：カスタム属性を取得する

Attribute クラスの GetCustomAttribute メソッドまたは GetCustomAttributes メソッドを使用して、カスタム属性を取得できます。

クラスからカスタム属性の単一のインスタンスを取得するには

1. ソース コードの先頭に **Imports** ステートメントを追加して、System 名前空間から **Attribute** クラスをインポートします。

   VB

   ```
   Imports System.Attribute
   ```

2. 属性を取得するプロシージャを作成します。

   VB

   ```
   Sub RetrieveAttribute()

   End Sub
   ```

3. プロシージャ内で、**Attribute** 型の変数と、取得する属性と同じ型の変数を宣言します。

   VB

   ```
   Dim Attr As Attribute
   Dim CustAttr As CustomAttribute
   ```

4. **GetType** 演算子を使用して、クラスおよび属性の型を **GetCustomAttribute** メソッドの呼び出しに渡し、**Attribute** で宣言された変数に戻り値を代入します。

   VB

   ```
   Attr = GetCustomAttribute(Me.GetType, _
                             GetType(CustomAttribute), False)
   ```

5. **CType** 関数を使用して、属性のデータ型を汎用の属性から取得した型の属性に変換します。カスタム属性型として宣言した変数に結果を代入します。

   VB

   ```
   CustAttr = CType(Attr, CustomAttribute)
   ```

6. 属性が取得されたかどうかを確認します。属性が取得された場合は、属性のフィールド、プロパティ、およびメソッドを使用します。

   VB

   ```
   If CustAttr Is Nothing Then
       MsgBox("The attribute was not found.")
   Else
       'Get the label and value from the custom attribute.
       MsgBox("The attribute label is: " & CustAttr.Label)
       MsgBox("The attribute value is: " & CustAttr.Value)
   End If
   ```

上の例では、カスタム属性を `ThisClass` クラスに適用するために、`RetrieveAttribute` プロシージャによって、**System.Attribute** クラスの **GetCustomAttribute** メソッドが呼び出されます。**GetCustomAttribute** は共有メソッドであるため、**System.Attribute** のインスタンスを最初に作成する必要はありません。**CType** 関数は、返された属性を **System.Attribute** 型からカスタム属性型の `CustomAttribute` に変換します。

参照

処理手順

方法 : 独自の属性を定義する

関連項目

GetType 演算子
CType 関数
IsNothing 関数
GetAttr 関数
GetCustomAttribute
GetCustomAttributes
AttributeUsageAttribute

概念

属性の適用
属性に格納されている情報の取得

カスタム属性の使用例

次の例では、クラスにだけ適用可能なカスタム属性を定義しています。

例

VB

```
<AttributeUsage(AttributeTargets.Class)> Public Class CustomAttribute
    Inherits System.Attribute

    'Declare two private fields to store the property values.
    Private m_LlabelValue As String
    Private m_VValueValue As Integer

    'The Sub New constructor is the only way to set the properties.
    Public Sub New(ByVal _Label As String, ByVal _Value As Integer)
        m_LlabelValue = _Label
        m_VValueValue = _Value
    End Sub

    Public ReadOnly Property Label() As String
        Get
            Return m_LlabelValue
        End Get
    End Property

    Public ReadOnly Property Value() As Integer
        Get
            Return m_VValueValue
        End Get
    End Property
End Class
```

属性クラスのコンストラクタだけが、この属性で定義されているプロパティを設定できます。次のコードは属性を使用する方法を示します。

VB

```
' Apply the custom attribute to this class.
<Custom("Some metadata", 66)> Class ThisClass
    ' Add class members here.
End Class
```

参照

処理手順

方法 : 独自の属性を定義する

方法 : カスタム属性を取得する

関連項目

AttributeUsageAttribute

概念

属性の適用

属性に格納されている情報の取得

方法：複数の属性を適用する

この例では、同一の要素に複数の属性を適用する方法を示します。

各属性をコンマで区切って、同じ属性ブロック内で複数の属性を指定できます。

使用例

この例では、EditorBrowsableAttribute 属性および ObsoleteAttribute 属性を同一の関数に適用する方法を示します。

VB

```
<System.ComponentModel.EditorBrowsable( _
    System.ComponentModel.EditorBrowsableState.Never), _
Obsolete("This method should not be used.")> _
Public Shadows Function Update(ByVal x As Integer) As Integer
    ' The function code goes here.
End Function
```

参照

概念

Visual Basic における属性

属性の適用

その他の技術情報

Visual Basic における属性

Visual Basic における宣言された要素

宣言された要素とは、宣言ステートメントで定義されたプログラミング要素です。変数、定数、列挙値、クラス、構造体、モジュール、インターフェイス、プロシージャ、プロシージャのパラメータ、関数の戻り値、外部のプロシージャ参照、演算子、プロパティ、イベント、およびデリゲートは、すべて宣言された要素です。

宣言ステートメントの例を以下に示します。

- Dim ステートメント (Visual Basic)
- Const ステートメント (Visual Basic)
- Enum ステートメント (Visual Basic)
- Class ステートメント (Visual Basic)
- Structure ステートメント
- Module ステートメント
- Interface ステートメント (Visual Basic)
- Function ステートメント (Visual Basic)
- Sub ステートメント (Visual Basic)
- Declare ステートメント
- Operator ステートメント
- Property ステートメント
- Event ステートメント
- Delegate ステートメント

このセクションの内容

宣言された要素の名前

要素の名前の付け方、および大文字と小文字の区別について説明します。

宣言された要素の特性

宣言された要素の特性 (スコープなど) について説明します。

宣言された要素の参照

コンパイラが参照を宣言に対応させる方法、および名前に修飾子を付ける方法について説明します。

関連するセクション

Visual Basic におけるオブジェクト指向プログラミング

オブジェクト指向プログラミングの基礎について説明します。

プログラム構造とコード規則

コードの読み取り、理解、および保守を簡単にするためのガイドラインを示します。

Visual Basic の宣言ステートメント

プロシージャ、変数、配列、および定数に名前を付けて定義するステートメントについて説明します。

宣言コンテキストと既定のアクセス レベル

宣言された要素の種類を示し、それぞれについてその宣言ステートメント、どのコンテキストで宣言できるか、および既定のアクセスレベルは何かを説明します。

宣言された要素の名前

すべての宣言された要素には名前があります。この名前は識別子とも呼ばれ、宣言された要素をコードで参照するときに使用します。

ルール

Visual Basic での要素名は、次のルールに従う必要があります。

- アルファベットまたはアンダースコア (_) で始まる。
- アルファベット、10 進数、アンダースコア以外の文字を含まない。
- アンダースコアで始まる場合は、アルファベットまたは 10 進数を少なくとも 1 文字含む必要がある。
- 長さが 1023 文字を超えない。

最大 1023 文字の制限は、`outerNamespace.middleNamespace.innerNamespace.thisClass.thisElement` などの完全修飾名の文字列全体にも適用されます。

有効な要素名の例を次に示します。

```
aB123__45

_567
```

無効な要素名の例を次に示します。1 番目はアンダースコアだけを含み、2 番目は 10 進数で始まり、3 番目には無効な文字 ($) が含まれています。

```
' Three INVALID element names

_

12ABC

xyz$wv
```

注意

アンダースコア (_) で始まる要素名は 共通言語仕様 (CLS) になりません。したがって、CLS 準拠のコードはそのような名前を定義したコンポーネントを使用できません。ただし、先頭以外であれば、要素名のどの位置にアンダースコアを使用しても CLS 準拠になります。

名前の長さのガイドライン

実際の問題として、名前は、要素の特性を明確に区別しながら可能な限り短いものである必要があります。これによってコードが読みやすくなり、行の長さとソース ファイルのサイズを小さくできます。

一方で、開発者の名前は、要素が何を表すのか、さらにコードで要素を使用する方法を適切に記述できる長さである必要があります。これは、コードの読みやすさの点で重要です。だれかが理解しようとした場合、または開発者自身がこれを作成した後しばらくたってからこれを見たときに、適切な要素名が付いていると時間を節約できます。

エスケープされた名前

通常は、Visual Basic によって予約されているキーワード (**Case** や **Friend** など) と一致する名前を要素名として使用することはできません。ただし、名前を角かっこ (**[ ]**) で囲んで、エスケープされた名前として定義できます。角かっこによってあいまいさがなくなるため、エスケープされた名前では、Visual Basic のキーワードと一致する名前を使用できます。定義した名前を後でコード内で使用するときも、同じように角かっこで囲みます。

一般には、エスケープされた名前を使用するのは次の場合だけです。

- コードが、名前として使用されている用語がキーワードとして予約されていない、Visual Basic の前のバージョンから移行されたものである。
- 指定されたキーワードが予約されていない別の言語で作成されたコードを使用している。

それ以外の場合で名前がキーワードと競合した場合は、要素名の変更を検討する必要があります。統合開発環境 (IDE: Integrated Development Environment) には、これを行うための簡単な方法が用意されています。詳細については、
「方法 : 識別子の名前を変更する」を参照してください。

名前の大文字と小文字の区別

Visual Basic の要素名は、大文字と小文字を区別します。したがって、大文字と小文字だけで区別される 2 つの名前は、コンパイラによって同じ名前であると解釈されます。たとえば、`ABC` と `abc` は、宣言された同じ要素を参照していると見なされます。

しかし、共通言語ランタイム (CLR: Common Language Runtime) のバインディングでは大文字と小文字が区別されます。このため、アセンブリまたは DLL を作成し、他のアセンブリで使用できるようにする場合は、大文字と小文字が区別されるようになります。たとえば、`ABC` という名前の要素を持つクラスを定義し、他のアセンブリから共通言語ランタイムを通じてこのクラスを使用する場合は、この要素を `ABC` として参照する必要があります。後でクラスを再コンパイルして要素の名前を `abc` に変更すると、このクラスを使用していた他のアセンブリがこの要素にアクセスできなくなります。したがって、アセンブリを更新してリリースするときには、パブリックな要素の名前の大文字と小文字を変更しないようにする必要があります。

名前およびロケール

名前の比較はロケールには依存しません。1 つのロケール内で 2 つの名前が一致した場合、これはすべてのロケールで一致することになります。

参照

概念

宣言された要素の特性

Visual Basic の宣言ステートメント

その他の技術情報

Visual Basic における宣言された要素

宣言された要素の参照

宣言された要素の特性

宣言された要素の特性は、コードとその要素とのやり取りに影響します。宣言された各要素には、次の特性の 1 つ以上が関連付けられています。

- データ型 - 要素が保持できる値およびその格納方法です。詳細については、「データ型の概要 (Visual Basic)」を参照してください。
- 有効期間 - 実行時間のうち、その要素を使用できる期間です。詳細については、「方法 : 変数の有効期間を延長する」を参照してください。
- スコープ - 名前に修飾子を付けずに要素を参照できる全コードの範囲です。詳細については、「方法 : 変数のスコープを制御する」を参照してください。
- アクセス レベル - コードがこの要素を利用するためのアクセス許可です。詳細については、「方法 : 変数の可用性を制御する」を参照してください。

要素の特性

宣言された要素、およびそれぞれの要素に適用される特性を次の表に示します。

要素	データ型	有効期間	スコープ [1]	アクセス レベル
変数	可	可	可	可
定数	可	不可	可	可
列挙型	可	不可	可	可
構造体	不可	不可	可	可
プロパティ	可	可	可	可
メソッド	不可	可	可	可
プロシージャ (**Sub** または **Function**)	不可	可	可	可
プロシージャ パラメータ	可	可	可	不可
関数の戻り値	可	可	可	不可
演算子	可	不可	可	可
インターフェイス	不可	不可	可	可
クラス	不可	不可	可	可
イベント	不可	不可	可	可
デリゲート	不可	不可	可	可

[1] スコープは参照範囲と呼ばれる場合もあります。

参照

概念

宣言された要素の名前
Visual Basic における有効期間
Visual Basic におけるスコープ
Visual Basic でのアクセス レベル

Visual Basic におけるデータ型
Visual Basic での変数宣言

その他の技術情報

Visual Basic における宣言された要素
宣言された要素の参照

Visual Basic における有効期間

宣言された要素の有効期間とは、その要素を使用できる期間です。変数は、有効期間がある唯一の要素です。このために、コンパイラでは、プロシージャ パラメータおよび関数の戻り値が、変数の特別な例として扱われます。変数の有効期間は、変数が値を保持できる期間を表します。保持されている値は変わっても、有効期間中は常になんらかの値が保持されています。

さまざまな有効期間

(プロシージャ外部の、モジュール レベルで宣言された) メンバ変数は、通常、宣言された要素と同じ有効期間を持っています。クラスまたは構造体で宣言された非共有変数は、宣言されたクラスまたは構造体の各インスタンスの別個のコピーとして存在します。そのような各変数は、インスタンスと同じ有効期間を持っています。ただし、共有 (**Shared**) 変数の有効期間は 1 つだけで、アプリケーションが終了するまで存続します。

(プロシージャの内部で宣言された) ローカル変数は、宣言されたプロシージャが実行されている間だけ存在します。そのプロシージャのパラメータ、および関数の戻り値についても同様です。ただし、プロシージャが他のプロシージャを呼び出した場合は、呼び出されたプロシージャの実行中もローカル変数の値は保持されます。

有効期間の開始

ローカル変数の有効期間は、宣言されたプロシージャの制御が開始されるときに始まります。すべてのローカル変数は、プロシージャの実行が開始されるとすぐに、データ型の既定値に初期化されます。初期値を指定する Dim ステートメント (Visual Basic) がプロシージャで検出されると、コードによって他の値が既に割り当てられていたとしても、これらの値にこれらの変数が設定されます。

構造体変数の各メンバは、独立した変数として初期化されます。同様に、配列変数の各要素も個別に初期化されます。

プロシージャ内部のブロック内で宣言された変数 (**For** ループなど) は、そのプロシージャへのエントリ時点で初期化されます。これらの初期化は、コードによってそのブロックが実行されるかどうかにかかわらず、有効になります。

有効期間の終了

プロシージャが終了すると、ローカル変数の値は保持されず、Visual Basic によってメモリが確保されます。次にプロシージャを呼び出すときに、すべてのローカル変数が再度作成され、再初期化されます。

クラスまたは構造体のインスタンスが終了すると、非共有変数はメモリおよび値を失います。クラスまたは構造体のそれぞれの新しいインスタンスによって、非共有変数が作成および再初期化されます。しかし、**Shared** 変数はアプリケーションが実行を停止するまで保持されます。

有効期間の延長

Static (Visual Basic) キーワードを持つローカル変数を宣言した場合、有効期間はプロシージャの実行時間よりも長くなります。**Static** 変数が存在する長さが、どのようにプロシージャ宣言によって決定されるのかを、次の表に示します。

プロシージャの場所および共有	静的変数の有効期間の開始	静的変数の有効期間の終了
モジュール内 (既定で共有)	プロシージャが最初に呼び出されたとき	アプリケーションの実行が停止されたとき
クラスまたは構造体の **Shared** (プロシージャはインスタンス メンバではありません)	特定のインスタンス、またはクラス名や構造体名 自体のいずれかで、プロシージャが最初に呼び出されたとき	アプリケーションの実行が停止されたとき
Shared ではなく、クラスまたは構造体のインスタンス (プロシージャはインスタンス メンバではありません)	特定のインスタンスで、プロシージャが最初に呼び出されたとき	ガベージ コレクション (GC) に対して、インスタンスが解放されたとき

参照

処理手順

方法 : 変数の有効期間を延長する

データ型のトラブルシューティング

関連項目

Shared (Visual Basic)

Nothing (Visual Basic)

概念

宣言された要素の名前

Visual Basic におけるスコープ

Visual Basic でのアクセス レベル

Visual Basic における変数
Visual Basic での変数宣言
その他の技術情報
宣言された要素の参照

方法：変数の有効期間を延長する

通常、変数が宣言されているプログラミング要素が存在しなくなると、変数も存在しなくなります。ただし、Static (Visual Basic) キーワードを使用して宣言すると、その変数のコンテナ要素より変数の有効期間を延長できます。

詳細については、「Visual Basic における有効期間」を参照してください。

コンテナ要素より変数の有効期間を延長するには

- 変数がローカル変数 (プロシージャ内で宣言) である場合は、これを宣言している Dim ステートメント (Visual Basic) に **Static** キーワードを含めます。静的変数は、変数を宣言しているプロシージャを含むクラスまたはモジュールが存続している限り有効です。

  ローカル変数が Shared (Visual Basic) プロシージャ内部にある場合、そのプロシージャと変数は、アプリケーションが実行されている間有効です。この場合には **Static** を使用しないでください。

- 変数がメンバ変数 (プロシージャ外部のクラスまたは構造体で宣言) である場合は、その変数を宣言している **Dim** ステートメントに **Shared** キーワードを含めます。共有変数はどのクラスまたは構造体のインスタンスにも関連付けられておらず、アプリケーションが実行されている間存在します。メンバ変数に対して **Static** は使用できません。

  メンバ変数がモジュール内で宣言されている場合、アプリケーションが実行されている間、常に存在します。この場合には **Shared** を使用しないでください。

使用例

Static (Visual Basic) キーワードを使用して変数を宣言する例を、次に示します (Dim ステートメント (Visual Basic) で、**Static** などの修飾子を使用している場合は、**Dim** キーワードは不要です)。

```
Function runningTotal(ByVal num As Integer) As Integer
    Static applesSold As Integer
    applesSold = applesSold + num
    Return applesSold
End Function
```

前の例では、変数 `applesSold` は、プロシージャ `runningTotal` が呼び出し元のコードに戻された後でも、引き続き存在します。次に `runningTotal` が呼び出されたとき、`applesSold` は前に計算した値を保持しています。

Static を使用しないで `applesSold` が宣言された場合、は `runningTotal` が呼び出されても事前に蓄積された値は保持されません。次に `runningTotal` が呼び出されたときに、`applesSold` は再作成され、0 に初期化されます。`runningTotal` は、呼び出されたときと同じ値を返すだけです。

コードのコンパイル方法

静的ローカル変数の値は、宣言の一部として初期化できます。配列を **Static** として宣言する場合は、ランク (次元数)、各次元の長さ、および各要素の値を初期化できます。

同じ名前の静的変数

複数のプロシージャで、同じ名前の静的変数を宣言できます。これを実行すると、Visual Basic のコンパイラは、このような各変数は別々の要素であると見なします。いずれか 1 つの変数を初期化しても、その他の変数の値には影響しません。これは、一連のオーバーロードを持つプロシージャを定義し、それぞれのオーバーロードで同じ名前の静的変数を定義した場合も同様です。

静的変数のコンテナ要素

静的ローカル変数は、クラス内 (つまりクラスのプロシージャの中) で宣言できます。ただし、構造体のメンバまたはその構造体内部のプロシージャのローカル変数のいずれとしてでも、ローカルの静的変数を構造体内部で宣言することはできません。

セキュリティ

前の例では、`applesSold` をモジュール レベルで宣言して、同じ有効期間にできます。ただし、この方法で変数のスコープを変更すると、プロシージャが変数に排他的にアクセスできなくなります。この場合、他のプロシージャが `applesSold` にアクセスして値を変更する可能性があるため、現在の合計を信頼できなくなり、コードの管理が難しくなります。

参照

概念

Visual Basic における有効期間
宣言された要素の特性

Visual Basic におけるスコープ

宣言された要素のスコープとは、名前に修飾子を付けたり Imports ステートメントを使用して有効にすることなく、その要素を参照できるコードの範囲です。要素は、次のいずれかのレベルのスコープを持つことができます。

レベル	説明
ブロック スコープ	要素が宣言されたコード ブロック内でのみ使用可能
プロシージャ スコープ	要素が宣言されたプロシージャ内のすべてのコードで使用可能
モジュール スコープ	要素が宣言されたモジュール内、クラス内、または構造体内のすべてのコードで使用可能
名前空間スコープ	要素が宣言された名前空間内のすべてのコードで使用可能

上に列挙したスコープのレベルは、下にいくほどスコープが広くなります。つまり、ブロック スコープが最も狭いスコープ、名前空間スコープが最も広いスコープです。ここでいう最も狭いスコープとは、修飾子を付けずにその要素を参照できるコード範囲が最も小さいという意味です。詳細については、このページの「スコープのレベル」を参照してください。

スコープの指定と変数の定義

要素のスコープは、要素を宣言するときに指定します。スコープは以下の要因によって決まります。

- 要素を宣言する領域 (ブロック、プロシージャ、モジュール、クラス、または構造体)
- 要素の宣言を含む名前空間
- 要素に宣言するアクセス レベル

名前が同じでスコープが異なる変数を定義する場合は、予想外の結果を招く可能性があるので注意してください。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」を参照してください。

スコープのレベル

プログラミング要素は、宣言した領域全体にわたって使用できます。同じ領域内のコードで要素を参照する場合は、名前に修飾子を付ける必要はありません。

ブロック スコープ

ブロックとは、次のように、開始宣言ステートメントと終了宣言ステートメントで囲まれた一連のステートメントです。

- **Do** および **Loop**
- **For** [**Each**] および **Next**
- **If** および **End If**
- **Select** および **End Select**
- **SyncLock** および **End SyncLock**
- **Try** および **End Try**
- **While** および **End While**
- **With** および **End With**

ブロック内で変数を宣言した場合、その変数はそのブロック内でのみ使用できます。たとえば、次の例で整数型の変数 `cube` のスコープは **If** と **End If** の間のブロックであるため、実行ステップがこのブロックの外に出てしまうと `cube` を参照できなくなります。

```
If n < 1291 Then
    Dim cube As Integer
    cube = n ^ 3
End If
```

メモ：

スコープがブロック内に制限されている変数でも、有効期間はプロシージャ全体の有効期間と同じです。プロシージャ内で同じブロックが複数回実行される場合、各ブロック変数には前に実行されたときの値が保持されています。そのような場合に予想外の結果が生じるのを避けるために、ブロックの先頭ではブロック変数を初期化することをお勧めします。

プロシージャ スコープ

プロシージャ内で宣言した要素は、そのプロシージャの外では使用できません。要素を使用できるのは、その要素の宣言を含むプロシージャだけです。このレベルの変数は、ローカル変数とも呼ばれます。ローカル要素を宣言するには、Dim ステートメント (Visual Basic) を使用します。Static (Visual Basic) キーワードは指定することも省くこともできます。

プロシージャとブロック スコープの間には密接な関係があります。プロシージャ内のどのブロックにも含まれていない場所で変数を宣言した場合、その変数は、プロシージャ全体から成るブロックのブロック スコープを持つと見なすことができます。

メモ：

Static 変数を含め、すべてのローカル要素は、宣言されたプロシージャ内にプライベートです。プロシージャ内で Public (Visual Basic) キーワードを使って要素を宣言することはできません。

モジュール スコープ

便宜上、モジュール、クラス、および構造体に対して、モジュール レベルという 1 つの用語を使用します。要素をモジュール レベルで宣言するには、モジュール、クラス、または構造体の中で、プロシージャやブロックの外に宣言ステートメントを配置します。

モジュール レベルで要素を宣言するときは、選択するアクセス レベルによってスコープが決まります。また、モジュール、クラス、または構造体を含む名前空間もスコープに影響します。

Private (Visual Basic) のアクセス レベルを宣言した要素は、モジュール内のすべてのプロシージャから参照できますが、他のモジュール内のコードからは参照できません。アクセス レベル キーワードを使用しないと、モジュール レベルの **Dim** ステートメントは既定で **Private** になります。ただし、**Dim** ステートメントで **Private** キーワードを使用すると、スコープとアクセス レベルがより明確になります。

次の例の場合、モジュール内で定義されているすべてのプロシージャから、文字列変数 `strMsg` を参照できます。2 番目のプロシージャが呼び出されると、文字列変数 `strMsg` の内容がダイアログ ボックスに表示されます。

```
' Put the following declaration at module level (not in any procedure).
Private strMsg As String
' Put the following Sub procedure in the same module.
Sub initializePrivateVariable()
    strMsg = "This variable cannot be used outside this module."
End Sub
' Put the following Sub procedure in the same module.
Sub usePrivateVariable()
    MsgBox(strMsg)
End Sub
```

名前空間スコープ

Friend (Visual Basic) キーワードまたは Public (Visual Basic) キーワードを使ってモジュール レベルで要素を宣言すると、宣言した名前空間内のすべてのプロシージャで使用できるようになります。上の例を次のように変更した場合、文字列変数 `strMsg` は、この変数を宣言した名前空間内のすべてのコードから参照できます。

```
' Include this declaration at module level (not inside any procedure).
Public strMsg As String
```

名前空間スコープには、入れ子にした名前空間も含まれます。名前空間内で使用できる要素は、その名前空間内に入れ子にした名前空間の中からも使用できます。

プロジェクト内に Namespace ステートメントが 1 つもない場合は、プロジェクト全体が同じ名前空間に属します。この場合、名前空間スコープはプロジェクト スコープであると考えることができます。モジュール、クラス、または構造体での **Public** 要素は、このプロジェクトを参照するすべてのプロジェクトで使用できます。

スコープの選択

変数を宣言する際は、次の点を念頭に置いてスコープを選択する必要があります。

ローカル変数の利点

ローカル変数は、次の理由により、各種の一時的な計算を行う場合に最適です。

- **名前の競合の回避。** ローカル変数には名前の重複が発生しません。たとえば、`intTemp` という名前の変数を持つプロシージャを複数作成できます。それぞれの `intTemp` をローカル変数として宣言する限り、各プロシージャは自分の `intTemp` 以外は認識しません。いずれかのプロシージャがローカルの `intTemp` の値を変更しても、他のプロシージャの `intTemp` に影響が及ぶことはありません。

- **メモリの使用量。** ローカル変数は、そのプロシージャの実行中にしかメモリを消費しません。制御がプロシージャから呼び出し側のコードに戻ると、メモリは解放されます。逆に、Shared (Visual Basic) 変数および Static (Visual Basic) 変数は、アプリケーションが実行を終えるまでメモリ リソースを消費するため、これらの変数は必要な場合にのみ使用してください。インスタンス変数は、インスタンスが存在している間メモリを消費します。このため、ローカル変数ほど効率的ではありませんが、**Shared** および **Static** 変数よりも効率的である可能性があります。

スコープをできるだけ狭くする

変数や定数を宣言する場合に、通常、できる限りスコープを狭くすることをお勧めします (最も狭いのはブロック スコープです)。スコープを狭くすると、メモリを節約できます。また、間違った変数を参照する可能性も低くなります。同様に、プロシージャの呼び出し間で値を保持する必要がある場合だけ、変数を Static (Visual Basic) として宣言してください。

参照

処理手順

方法 : 変数のスコープを制御する

概念

宣言された要素の特性
Visual Basic における有効期間
Visual Basic でのアクセス レベル
同じ名前を持つ複数の変数がある場合に参照を解決する
Visual Basic での変数宣言

方法：変数のスコープを制御する

通常、変数が宣言された領域内全体がその変数のスコープとなります。つまり、その範囲から変数を参照できます。場合によっては、変数のアクセス レベルがスコープに影響を及ぼします。

詳細については、「Visual Basic におけるスコープ」を参照してください。

ブロックまたはプロシージャ レベルのスコープ

変数をブロック内でのみ参照できるようにするには

- 変数の Dim ステートメント (Visual Basic) を、そのブロックの開始ステートメントと終了ステートメントの間に置きます。たとえば、**For** ループの **For** ステートメントと **Next** ステートメントの間です。

  これにより、ブロック内でのみ変数が参照できます。

変数をプロシージャ内でのみ参照できるようにするには

- 変数の **Dim** ステートメントを、プロシージャの内側の、しかも何のブロック内 (**With**...**End With** ブロックなど) にも属さない場所に記述します。

  これにより、プロシージャに含まれている各ブロックの内部も含めたプロシージャの内部からのみ変数を参照できます。

モジュールまたは名前空間レベルのスコープ

便宜上、モジュール、クラス、および構造体に対して、モジュール レベル という 1 つの用語を使用します。モジュール レベルの変数のアクセス レベルによってスコープが決まります。また、モジュール、クラス、または構造体を含んでいる名前空間もスコープに影響します。

モジュール、クラス、または構造体全体にわたって変数を参照できるようにするには

1. 変数の **Dim** ステートメントを、モジュール、クラス、または構造体の内側の、しかも何のプロシージャ内にも属さない場所に記述します。
2. **Dim** ステートメントには Private (Visual Basic) キーワードを含めます。
3. これにより、モジュール、クラス、または構造体のどこからでも変数を参照できます。外側からは参照できません。

名前空間全体にわたって変数を参照できるようにするには

1. 変数の **Dim** ステートメントを、モジュール、クラス、または構造体の内側の、しかも何のプロシージャ内にも属さない場所に記述します。
2. **Dim** ステートメントには Friend (Visual Basic) キーワードまたは Public (Visual Basic) キーワードを含めます。
3. これにより、モジュール、クラス、または構造体を含む名前空間内のどこからでも変数を参照できます。

使用例

次の例では、モジュール レベルで変数を宣言し、モジュール内のコードからのみ参照できるように制限しています。

```
Module demonstrateScope
    Private strMsg As String
    Sub initializePrivateVariable()
        strMsg = "This variable cannot be used outside this module."
    End Sub
    Sub usePrivateVariable()
        MsgBox(strMsg)
    End Sub
End Module
```

上の例では、`demonstrateScope` モジュールで定義されているすべてのプロシージャから、**String** 型の変数 `strMsg` を参照できます。`usePrivateVariable` プロシージャを呼び出すと、文字列変数 `strMsg` の内容がダイアログ ボックスに表示されます。

上の例を次のように変更すると、宣言した名前空間内のすべてのコードで文字列変数 `strMsg` を参照できるようになります。

```
    Public strMsg As String
```

堅牢性の高いプログラム

変数のスコープが狭いほど、同じ名前の別の変数を誤って参照してしまう可能性は少なくなります。また、参照先との一致に関する問題も最小化できます。

セキュリティ

変数のスコープが狭いほど、悪意のあるコードによって変数が不正に使用される可能性が少なくなります。

参照

関連項目

Dim ステートメント (Visual Basic)

概念

Visual Basic におけるスコープ
Visual Basic における有効期間
Visual Basic でのアクセス レベル
Visual Basic における変数
Visual Basic での変数宣言

型の上位変換

モジュール内でプログラミング要素を宣言すると、Visual Basic はその範囲をモジュールを含む名前空間にまで上位変換します。これを、型の上位変換と呼びます。

モジュールのスケルトン定義およびそのモジュールの 2 つのメンバを、次の例に示します。

```
Namespace projNamespace
    Module projModule
        Public Enum basicEnum As Integer
            one = 1
            two = 2
        End Enum
        Public Class innerClass
            Public Sub numberSub(ByVal firstArg As Integer)
            End Sub
        End Class
    End Module
End Namespace
```

`projModule` 内部で、モジュール レベルで宣言されたプログラミング要素は `projNamespace` に上位変換されます。前の例では、`basicEnum` および `innerClass` は用意変換されますが、`numberSub` はモジュール レベルで宣言されていないため上位変換されません。

型の上位変換による効果

型の上位変換による効果は、修飾文字列にモジュール名を含める必要がなくなるということです。次の例では、前の例のプロシージャに対して 2 つの呼び出しを行っています。

```
Sub usePromotion()
    projNamespace.projModule.innerClass.numberSub(projNamespace.projModule.basicEnum.one)
    projNamespace.innerClass.numberSub(projNamespace.basicEnum.two)
End Sub
```

前の例では、最初の呼び出しで完全な修飾文字列を使用しています。しかし、型の上位変換によりこれは必要ありません。2 番目の呼び出しでもモジュールのメンバにアクセスしていますが、修飾文字列に `projModule` を含めていません。

型の上位変換の無効化

名前空間に既にモジュール メンバと同じ名前のメンバがある場合、そのモジュール メンバに対する型の上位変換は無効になります。列挙型のスケルトン定義および同じ名前空間内のモジュールを、次の例に示します。

```
Namespace thisNamespace
    Public Enum abc
        first = 1
        second
    End Enum
    Module thisModule
        Public Class abc
            Public Sub abcSub()
            End Sub
        End Class
        Public Class xyz
            Public Sub xyzSub()
            End Sub
        End Class
    End Module
End Namespace
```

前の例では、名前空間レベルで同じ名前の列挙型が既に存在するため、Visual Basic ではクラス `abc` を `thisNameSpace` に上位変換できません。`abcSub` にアクセスするには、完全な修飾文字列 `thisNamespace.thisModule.abc.abcSub` を指定する必要があります。しかし、クラス `xyz` は上位変換されたままなので、短い修飾文字 `thisNamespace.xyz.xyzSub` を使用して `xyzSub` にアクセスできます。

部分型に対する型の上位変換の無効化

モジュール内のクラスまたは構造体で Partial (Visual Basic) キーワードが使用されている場合、名前空間に同じ名前のメンバがあるかどうかに関係なく、そのクラスまたは構造体に対する型の上位変換は自動的に無効化されます。モジュール内のその他の要素は、そのまま型の上位変換の対象となります。

結果。 部分定義の型の上位変換の無効化は、予期しない結果、さらにコンパイル エラーを生じさせる可能性があります。クラスのスケルトン部分定義、そのうちの 1 つはモジュール内にある場合を、次の例に示します。

```
Namespace sampleNamespace
    Public Partial Class sampleClass
        Public Sub sub1()
        End Sub
    End Class
    Module sampleModule
        Public Partial Class sampleClass
            Public Sub sub2()
            End Sub
        End Class
    End Module
End Namespace
```

前の例では、開発者は `sampleClass` の 2 つの部分定義がコンパイラによってマージされることを想定しています。しかし、コンパイラでは `sampleModule` 内の部分定義の上位変換は考慮されません。その結果、両方とも名前は `sampleClass` ですがパスの修飾が異なる、2 つの独立し、区別されたクラスをコンパイルしようとします。

コンパイラは、完全修飾されたパスがまったく同じ場合にのみ、部分定義をマージします。

推奨

プログラミングでは、次の方法に従うことをお勧めします。

- **一意の名前。** プログラミング要素の名前付けについて完全に制御する場合、どのような場合でも一意の名前を使用することをお勧めします。名前がまったく同じであると余分に修飾する必要があり、コードが読み取りにくくなります。また名前が同じであると軽度のエラーが発生し、予期しない結果が生じる可能性もあります。
- **完全修飾。** 同一の名前空間内でモジュールと他の要素を使用して作業する場合、最も安全な方法はすべてのプログラミング要素に対して常に、完全修飾を使用することです。あるモジュール メンバの上位変換が行われず、そのメンバを完全に修飾しない場合、誤って別のプログラミング要素にアクセスしてしまう可能性があります。

参照

処理手順

方法 : 変数のスコープを制御する

関連項目

Module ステートメント
Namespace ステートメント
Partial (Visual Basic)

概念

Visual Basic におけるスコープ
同じ名前を持つ複数の変数がある場合に参照を解決する

Visual Basic でのアクセス レベル

宣言された要素のアクセス レベルとは、その要素にアクセスするために必要な権限の程度、つまり、その要素に対する読み取りと書き込みの許可がどのようなコードに与えられるかを示します。アクセス レベルは、要素自体を宣言する方法だけでなく、要素のコンテナのアクセス レベルによっても左右されます。コンテナ要素にアクセスできないコードは、そのコンテナ要素に含まれる要素にも一切アクセスできません。含まれる要素が **Public** として宣言されていても同じです。たとえば、**Private** 構造体の中の **Public** 変数は、その構造体を含むクラス内からはアクセスできますが、そのクラスの外からはアクセスできません。

Public

宣言ステートメントで Public (Visual Basic) キーワードを使用して宣言した要素は、同じプロジェクト内からも、そのプロジェクトを参照している他のプロジェクトからも、そのプロジェクトからビルドされたアセンブリからもアクセスできます。**Public** 宣言の例を次に示します。

```
Public Class classForEverybody
```

Public は、モジュール レベル、インターフェイス レベル、または名前空間レベルでのみ使用できます。つまり、パブリック要素は、ソース ファイルまたは名前空間のレベルか、インターフェイス内、モジュール内、クラス内、または構造体内では宣言できますが、プロシージャ内では宣言できません。

Protected

宣言ステートメントで Protected (Visual Basic) キーワードを使用して宣言した要素は、同じクラス、またはそのクラスの派生クラスからしかアクセスできません。**Protected** 宣言の例を次に示します。

```
Protected Class classForMyHeirs
```

Protected は、クラス レベルでのみ、しかも、クラスのメンバを宣言する場合にのみ使用できます。つまり、プロテクト要素は、クラス内では宣言できますが、ソース ファイルや名前空間のレベルと、インターフェイス内、モジュール内、構造体内、およびプロシージャ内では宣言できません。

Friend

宣言ステートメントで Friend (Visual Basic) キーワードを使用して宣言した要素は、同じアセンブリからはアクセスできますが、そのアセンブリの外部からはアクセスできません。**Friend** 宣言の例を次に示します。

```
Friend stringForThisProject As String
```

Friend は、モジュール レベル、インターフェイス レベル、または名前空間レベルでのみ使用できます。つまり、フレンド要素は、ソース ファイルまたは名前空間のレベルか、インターフェイス内、モジュール内、クラス内、または構造体内では宣言できますが、プロシージャ内では宣言できません。

Protected Friend

宣言ステートメントで **Protected** キーワードと **Friend** キーワードの両方を使用して宣言した要素は、派生クラス内と同じアセンブリ内のいずれか、または両方からアクセスできます。次のコードは、**Protected Friend** 宣言の例を示します。

```
Protected Friend stringForProjectAndHeirs As String
```

Protected Friend は、クラス レベルでのみ、しかも、クラスのメンバを宣言する場合にのみ使用できます。つまり、プロテクト フレンド要素は、クラス内では宣言できますが、ソース ファイルや名前空間のレベルと、インターフェイス内、モジュール内、構造体内、およびプロシージャ内では宣言できません。

Private

宣言ステートメントで Private (Visual Basic) キーワードを使用して宣言した要素は、同じモジュール、クラス、または構造体からしかアクセスできません。**Private** 宣言の例を次に示します。

```
Private numberForMeOnly As Integer
```

Private は、モジュール レベルでのみ使用できます。つまり、プライベート要素は、モジュール内、クラス内、または構造体内では宣言できます

が、ソース ファイルや名前空間のレベルと、インターフェイス内およびプロシージャ内では宣言できません。

モジュール レベルでは、アクセス レベル キーワードのない **Dim** ステートメントは、**Private** 宣言と等価です。ただし、**Private** キーワードを使った方がコードがわかりやすくなります。

アクセス修飾子

アクセス レベルを指定するキーワードは、アクセス修飾子と呼ばれます。アクセス修飾子の比較を次の表に示します。

アクセス修飾子	付与されるアクセス レベル	このアクセス レベルで宣言できる要素	この修飾子を使用できる宣言コンテキスト
Public	無制限 : パブリック要素を参照できるすべてのコードがアクセスできます。	インターフェイス モジュール クラス 構造体 構造体メンバ プロシージャ プロパティ メンバ変数 定数 列挙型 イベント 外部宣言 デリゲート	ソース ファイル 名前空間 インターフェイス モジュール クラス 構造体
Protected	派生 : プロテクト要素を宣言するクラス内のコード、またはそのクラスの派生したクラスが、この要素にアクセスできます。	インターフェイス クラス 構造体 プロシージャ プロパティ メンバ変数 定数 列挙型 イベント 外部宣言 デリゲート	クラス

Friend	アセンブリ: フレンド要素が宣言されているアセンブリ内のコードが、この要素にアクセスできます。	インターフェイス モジュール クラス 構造体 構造体メンバ プロシージャ プロパティ メンバ変数 定数 列挙型 イベント 外部宣言 デリゲート	ソース ファイル 名前空間 インターフェイス モジュール クラス 構造体
Protected Friend	**Protected** と **Friend** の結合 : プロテクト フレンド要素と同じクラスまたは同じアセンブリ内のコード、または要素のクラスから派生した各クラス内から、要素にアクセスできます。	インターフェイス クラス 構造体 プロシージャ プロパティ メンバ変数 定数 列挙型 イベント 外部宣言 デリゲート	クラス
Private	宣言コンテキスト : 含まれている型の中のコードを含め、プライベート要素が宣言されている型内のコードが、要素にアクセスできます。	インターフェイス クラス 構造体 構造体メンバ プロシージャ プロパティ メンバ変数 定数 列挙型 イベント 外部宣言 デリゲート	モジュール クラス 構造体

参照

処理手順

方法 : 変数の可用性を制御する

関連項目

Dim ステートメント (Visual Basic)

Static (Visual Basic)

概念

宣言された要素の名前

宣言された要素の特性

Visual Basic における有効期間

Visual Basic におけるスコープ

Visual Basic における変数

Visual Basic での変数宣言

その他の技術情報

宣言された要素の参照

方法：変数の可用性を制御する

変数のアクセス レベルを指定して、可用性を制御します。アクセス レベルによって、どのコードが、変数の読み取りまたは書き込みのアクセス許可を持っているかが決まります。

- (モジュール レベルおよびプロシージャの外部で定義される) メンバ変数は、既定値であるパブリック アクセスに設定されます。つまり、メンバ変数を参照できるすべてのコードが、メンバ変数にアクセスできます。アクセス修飾子を指定すると、これを変更できます。
- (プロシージャの内部で定義される) ローカル変数は、名目上パブリック アクセスを持っていますが、プロシージャ内のコードのみがローカル変数にアクセスできます。ローカル変数のアクセス レベルは変更できませんが、ローカル変数が含まれるプロシージャのアクセス レベルは変更できます。

詳細については、「Visual Basic でのアクセス レベル」を参照してください。

プライベート アクセスおよびパブリック アクセス

モジュール、クラス、または構造体内からだけ変数にアクセスできるようにするには

1. 変数の Dim ステートメント (Visual Basic) を、プロシージャの外部ではなく、モジュール、クラス、または構造体の内部に配置します。
2. **Dim** ステートメントには Private (Visual Basic) キーワードを含めます。

   モジュール、クラス、または構造体内から変数の読み取りまたは書き込みを行うことができますが、外部からはできません。

変数を参照できるすべてのコードが、変数にアクセスできるようにするは

1. メンバ変数に対して、変数の **Dim** ステートメントを、プロシージャの外部ではなく、モジュール、クラス、または構造体の内部に配置します。
2. **Dim** ステートメントには Public (Visual Basic) キーワードを含めます。

   アセンブリと相互運用しているどのコードからでも、変数の読み取りまたは書き込みを行うことができます。

または

1. ローカル変数に対して、変数の **Dim** ステートメントをプロシージャ内部に配置します。
2. **Dim** ステートメントには **Public** キーワードを含めないようにしてください。

   プロシージャ内から変数の読み取りまたは書き込みを行うことができますが、外部からはできません。

Protected アクセスおよび Friend アクセス

変数のアクセス レベルを、クラスおよび派生クラス、またはアセンブリに制限できます。また、これらの制限の共用体を指定して、派生クラスまたは同じアセンブリ内の他の場所にあるコードからのアクセスを可能にすることもできます。同じ宣言内の **Protected** キーワードと **Friend** キーワードを組み合わせて、この共用体を指定します。

クラスおよび派生クラスからだけ変数にアクセスできるようにするには

1. 変数の **Dim** ステートメントを、プロシージャの外部ではなく、クラスの内部に配置します。
2. **Dim** ステートメントには Protected (Visual Basic) キーワードを含めます。

   クラス内および派生したクラスのどの場所からでも、変数の読み取りまたは書き込みを行うことができますが、派生チェーンのクラスの外部からはできません。

同じアセンブリ内からだけ変数にアクセスできるようにするには

1. 変数の **Dim** ステートメントを、プロシージャの外部ではなく、モジュール、クラス、または構造体の内部に配置します。
2. **Dim** ステートメントには Friend (Visual Basic) キーワードを含めます。

   モジュール、クラス、または構造体内のすべての場所、および同じアセンブリ内のどのコードからでも、変数の読み取りまたは書き込みを行うことができますが、アセンブリの外部からはできません。

使用例

Public、**Protected**、**Friend**、**Protected Friend**、および **Private** アクセス レベルを使用して、変数を宣言する例を示します。**Dim** ステートメントによってアクセス レベルが指定される場合、**Dim** キーワードを含める必要はありません。

```
Public Class classForEverybody
Protected Class classForMyHeirs
Friend stringForThisProject As String
Protected Friend stringForProjectAndHeirs As String
Private numberForMeOnly As Integer
```

セキュリティ

変数のアクセス レベルが制限されるほど、悪意のあるコードによって不正使用される機会が少なくなります。

参照

関連項目

Dim ステートメント (Visual Basic)
Public (Visual Basic)
Protected (Visual Basic)
Friend (Visual Basic)
Private (Visual Basic)

概念

Visual Basic でのアクセス レベル

宣言された要素の参照

宣言された要素をコードから参照するときには、Visual Basic のコンパイラが、参照内の名前をその名前の適切な宣言に対応させます。複数の要素が同じ名前で宣言されている場合は、名前に修飾子を付けることによって、参照する要素を指定できます。

このセクションの内容

同じ名前を持つ複数の変数がある場合に参照を解決する

コンパイラがどのように参照を宣言に対応させるのかを説明します。

方法 : 宣言された要素名を修飾する

参照を対応させやすくするには要素名をどのように修飾するかを説明します。

方法 : 同じ名前を持つ 2 つの要素を区別する

Visual Basic コンパイラに参照と要素の対応付けをさせるときに、要素名をどのように修飾するかを説明します。

Visual Basic におけるシャドウ

同じ名前を持つ要素の扱いについて説明します。

関連するセクション

宣言された要素の名前

要素の名前の付け方、および大文字と小文字の区別について説明します。

宣言された要素の特性

宣言された要素の特性 (スコープなど) について説明します。

同じ名前を持つ複数の変数がある場合に参照を解決する

コンパイラでは、名前の参照を名前の宣言に対応させようとするとき、最も狭いスコープの中で一致する宣言を検索します。これは、この参照が行われているコードを起点として、起点を含んでいる最も内側のコンテナ要素から外側のレベルへと順に検索範囲を拡大していくことを意味します。

この検索処理をオーバーライドし、より広いスコープで宣言されている名前を指定する場合には、より広いスコープを持つコンテナ要素によって変数名を修飾する必要があります。場合によっては、コンテナ要素をさらに修飾する必要があります。名前の修飾の詳細については、「方法 : 宣言された要素名を修飾する」を参照してください。

同じ名前を持つ複数のプログラミング要素にアプリケーションからアクセスする場合にも、名前参照の修飾が必要になることがあります。例については、このヘルプ ページの「同じ名前のクラス」および「方法 : 同じ名前を持つ 2 つの要素を区別する」を参照してください。

最も狭いスコープ

次の例は、同じ名前を持つ 2 つの変数への参照を示します。

```
' Assume these two modules are both in the same assembly.
Module container
    Public totalCount As Integer = 1
    Public Sub showCount()
        Dim totalCount As Integer = 6000
        ' The following statement displays the local totalCount (6000).
        MsgBox("Unqualified totalCount is " & CStr(totalCount))
        ' The following statement displays the module's totalCount (1).
        MsgBox("container.totalCount is " & CStr(container.totalCount))
    End Sub
End Module
Module callingModule
    Public Sub displayCount()
        container.showCount()
        ' The following statement displays the containing module's totalCount (1).
        MsgBox("container.totalCount is " & CStr(container.totalCount))
    End Sub
End Module
```

上の例では、同じ `totalCount` という名前を持ち、`container` モジュール内でのスコープ レベルが異なる 2 つの変数を宣言しています。`showCount` プロシージャで、修飾しないで `totalCount` を表示すると、Visual Basic コンパイラは、最も狭いスコープの宣言、つまり `showCount` 内でローカル宣言されている変数に参照を解決します。コンテナであるモジュール `container` で `totalCount` を修飾すると、コンパイラは参照を広いスコープの宣言に解決します。

別のコンテナ要素のメンバ

他のクラスや構造体の非共有メンバを使用するときには、最初に、クラスや構造体のインスタンスを指す変数または式でメンバ名を修飾する必要があります。次の例の `demoClass` は、`class1` というクラスのインスタンスです。

```
Dim demoClass As class1 = New class1()
demoClass.someSub[(argumentlist)]
```

共有 (Shared (Visual Basic)) ではないメンバを修飾するときは、クラス名 自体を使用することはできません。最初に、オブジェクト変数 (この場合は `demoClass`) 内にインスタンスを作成する必要があります。次に、その変数名でこのインスタンスを参照します。

クラスまたは構造体に **Shared** メンバがある場合、そのメンバは、クラスまたは構造体の名前で修飾することも、インスタンスを指す変数または式で修飾することもできます。

モジュールには個別のインスタンスがありません。モジュールのすべてのメンバは、既定で **Shared** です。したがって、モジュール メンバはモジュール名で修飾します。

参照の修飾例

モジュール メンバ プロシージャに対する参照を修飾する例を次に示します。

```
' Assume these three modules are all in the same assembly.
Module module1
```

```
    Public Sub perform()
        MsgBox("module1.perform() now returning")
    End Sub
End Module
Module module2
    Public Sub perform()
        MsgBox("module2.perform() now returning")
    End Sub
    Public Sub doSomething()
        ' The following statement calls perform in module2, the active module.
        perform()
        ' The following statement calls perform in module1.
        module1.perform()
    End Sub
End Module
Module module3
    Public Sub callPerform()
        ' The following statement calls perform in module1.
        module1.perform()
        ' The following statement makes an unresolvable name reference
        ' and therefore generates a COMPILER ERROR.
        perform() ' INVALID statement
    End Sub
End Module
```

上の例では、同じ `perform` という名前を持ち、プロジェクト内の別々のモジュールにある 2 つの **Sub** プロシージャを宣言しています。どちらのプロシージャについても、宣言されたモジュール内で指定するときには修飾子を必要としませんが、それ以外の場所から参照するときには修飾子が必要です。`module3` 内にある最後の参照では、`perform` が修飾されていないため、コンパイラはこの参照を解決できません。

プロジェクトへの参照

別のプロジェクトで定義されている Public (Visual Basic) 要素を使用するには、最初にプロジェクトのアセンブリまたはタイプ ライブラリへの参照を設定する必要があります。参照を設定するには、[プロジェクト] メニューの [参照の追加] をクリックするか、コマンド ラインで /reference (Visual Basic) コンパイラ オプションを使用します。

たとえば、.NET Framework の XML オブジェクト モデルを使用できます。System.Xml 名前空間への参照を設定すると、XmlDocument など、この名前空間にあるすべてのクラスを宣言し、使用できます。**XmlDocument** の使用方法を次の例に示します。

```
' Assume this project has a reference to System.Xml
' The following statement creates xDoc as an XML document object.
Dim xDoc As System.Xml.XmlDocument
```

コンテナ要素のインポート

Imports ステートメントを使うと、使用するモジュールやクラスを含む名前空間をインポートできます。インポートした名前空間で定義されている要素は、名前を完全修飾せずに参照できます。次の例では、上の例を書き直して **System.Xml** 名前空間をインポートしています。

```
' Assume this project has a reference to System.Xml
' The following statement must precede all your declarations.
Imports System.Xml
' The following statement creates xDoc as an XML document object.
Dim xDoc As XmlDocument
```

また、**Imports** ステートメントでは、インポートする各名前空間のインポート エイリアスを定義することもできます。インポート エイリアスを使用すると、ソース コードが短くてわかりやすいものになります。次の例では、上の例を書き直して、**System.Xml** 名前空間のエイリアスとして `xD` を使用しています。

```
' Assume this project has a reference to System.Xml
' The following statement must precede all your declarations.
Imports xD = System.Xml
' The following statement creates xDoc as an XML document object.
Dim xDoc As xD.XmlDocument
```

Imports ステートメントを指定しても、他のプロジェクトの要素をアプリケーションで使用できるようになるわけではありません。つまり、参照を設定

する代わりになるわけではありません。名前空間をインポートした場合は、その名前空間で定義されている名前に修飾子を付ける必要がなくなるだけです。

Imports ステートメントを使って、モジュール、クラス、構造体、および列挙値をインポートすることもできます。インポートした要素のメンバは、修飾子を付けずに使用できます。ただし、クラスや構造体の非共有メンバについては、常に、クラスまたは構造体のインスタンスになる変数または式を使って修飾する必要があります。

同じ名前のクラス

オブジェクトの新しいインスタンスを作成するときには、所属する名前空間やタイプ ライブラリでクラスを修飾することが必要となる場合もあります。たとえば、System.Windows.Forms 名前空間と System.Web.UI.WebControls 名前空間の両方に `Label` クラスが含まれています (System.Windows.Forms.Label と System.Web.UI.WebControls.Label)。アプリケーションで両方のクラスを使用する場合、またはアプリケーション独自の `Label` クラスを定義する場合は、それぞれの `Label` オブジェクトを区別する必要があります。変数の宣言に名前空間を含めるか、エイリアスをインポートします。次の例では、インポート エイリアスを使用しています。

```
' The following statement must precede all your declarations.
Imports win = System.Windows.Forms, web = System.Web.UI.WebControls
' The following statement references the Windows.Forms.Label class.
Dim winLabel As New win.Label()
```

名前付けのガイドライン

複数のプログラミング要素を同じ名前で定義すると、コンパイラがその名前への参照を解決しようとするときに、名前のあいまいさの問題が発生する可能性があります。スコープ内に複数の定義がある場合、またはスコープ内に定義がない場合、参照は解決できません。例については、このヘルプ ページの「参照の修飾例」を参照してください。

名前のあいまいさを回避するには、すべての要素に一意の名前を付けます。そうすることで、名前空間、モジュール、またはクラスによって名前を修飾することなく、任意の要素を参照できます。また、誤って異なる要素を参照する可能性も少なくなります。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する
方法 : 宣言された要素名を修飾する
方法 : 同じ名前を持つ 2 つの要素を区別する

関連項目

Imports ステートメント
New (Visual Basic)
Public (Visual Basic)

概念

Visual Basic における変数

その他の技術情報

宣言された要素の参照

方法：宣言された要素名を修飾する

宣言された要素をコードから参照するときには、Visual Basic のコンパイラが、参照内の名前をその名前の適切な宣言に対応させる必要があります。要素がコードの外で定義されている場合、あるいは同じ名前で複数の要素が宣言されている場合、コンパイラであいまいさを解決できるように、あるいは要素を特定できるようにするため、名前を修飾する必要があります。

名前を修飾するということは、ソース ステートメント内で、ターゲットの要素が定義された場所を識別する情報よりも前に置くことを意味します。この情報は、修飾文字列と呼ばれます。1 つまたは複数の名前空間とモジュール、クラス、または構造体を含めることができます。

修飾文字列では、明確にモジュール、クラス、またはターゲット要素を含む構造体を指定する必要があります。コンテナは、その代わりに他のコンテナ要素、通常は名前空間に配置されます。修飾文字列にはいくつかのコンテナ要素を含める必要があります。

名前を修飾して宣言された要素にアクセスするには

1. 要素が定義された場所を特定します。これには、名前空間、あるいは名前空間の階層までもが含まれる場合があります。最下位レベルの名前空間では、要素はモジュール、クラス、または構造体に含まれている必要があります。

```
' Assume the following hierarchy exists outside your code.
Namespace outerSpace
    Namespace innerSpace
        Module holdsTotals
            Public Structure totals
                Public thisTotal As Integer
                Public Shared grandTotal As Long
            End Structure
        End Module
    End Namespace
End Namespace
```

2. ターゲット要素の場所に基づいて修飾パスを決定します。最高位レベルの名前空間から始まり、最下位レベルの名前空間まで進み、ターゲット要素を含むモジュール、クラス、または構造体で終わります。パス内の各要素は、その後に続く要素を含む必要があります。

   `outerSpace → innerSpace → holdsTotals → totals`

3. ターゲット要素の修飾文字列を準備します。パス内の各要素の後ろにはピリオド (.) を付ける必要があります。アプリケーションは、修飾文字列内のすべての要素に対するアクセス権限を持っている必要があります。

```
outerSpace.innerSpace.holdsTotals.totals.
```

4. 通常の方法で、ターゲット要素を参照する式または代入ステートメントを作成します。

```
grandTotal = 9000
```

5. ターゲットの要素名は修飾文字列に先行します。要素を含むモジュール、クラス、または構造体の後ろのピリオド (.) の直後に、名前を指定する必要があります。

```
' Assume the following module is part of your code.
Module accessGrandTotal
    Public Sub setGrandTotal()
        outerSpace.innerSpace.holdsTotals.totals.grandTotal = 9000
    End Sub
End Module
```

6. コンパイラは修飾文字列を使用して、ターゲット要素の参照と対応する、明白で、あいまいでない宣言を特定します。

参照

処理手順

方法 : 同じ名前を持つ 2 つの要素を区別する

方法 : 自分で宣言した変数と同じ名前の変数を隠す
方法 : 継承された変数を隠す
方法 : 派生クラスによって非表示になっている変数にアクセスする

概念

宣言された要素の特性
同じ名前を持つ複数の変数がある場合に参照を解決する
Visual Basic の宣言ステートメント

その他の技術情報

宣言された要素の参照

方法：同じ名前を持つ 2 つの要素を区別する

アプリケーションがアクセスできる範囲に同じ名前の要素が複数存在する場合は、名前を修飾することにより、Visual Basic コンパイラは、意図した特定の要素に参照を対応させることができます。詳細については、「方法 : 宣言された要素名を修飾する」を参照してください。

使用例

次の例では、同じ名前の 2 つの変数が、同じ名前の異なるコンテナ要素に含まれています。変数は **Shared** で宣言してありますが、これは単にコード例を短くするためです。

```
Namespace space1
    Public Class innerClass
        ' String showMe is declared Shared to facilitate reference.
        Public Shared showMe As String = "Shared string 1"
    End Class
End Namespace
Namespace space2
    Public Class innerClass
        ' String showMe is declared Shared to facilitate reference.
        Public Shared showMe As String = "Shared string 2"
    End Class
End Namespace
Public Module callShowMe
    Public Sub showStrings()
        MsgBox("From space1: " & space1.innerClass.showMe _
            & vbCrLf & "From space2: " & space2.innerClass.showMe)
    End Sub
End Module
```

上の例では、同じ `showMe` という名前で 2 つの変数を宣言しています。いずれの変数も、それぞれ `innerClass` というクラスの中にあります。2 つの変数を含むコンテナ要素の名前が同じであるため、呼び出し元のコードでは、変数 `showMe` をコンテナ `innerClass` で修飾するだけでは不十分です。`innerClass` を、さらにそのコンテナである `space1` または `space2` で修飾する必要があります。それぞれの修飾文字列に、一意な宣言への、あいまいさのないパスが指定されているため、Visual Basic コンパイラではそれぞれの参照を解決できます。

堅牢性の高いプログラム

同じ名前で宣言してある変数が少なければ少ないほど、誤って他の変数を参照してしまう可能性は小さくなります。また、参照先との一致に関する問題も最小化できます。

セキュリティ

同じ名前で宣言してある変数が少なければ少ないほど、悪意のあるコードによって変数が不正に使用される可能性も少なくなります。

参照

処理手順

方法 : 宣言された要素名を修飾する
方法 : 自分で宣言した変数と同じ名前の変数を隠す
方法 : 継承された変数を隠す
方法 : 派生クラスによって非表示になっている変数にアクセスする

概念

宣言された要素の特性
同じ名前を持つ複数の変数がある場合に参照を解決する
Visual Basic の宣言ステートメント

その他の技術情報

宣言された要素の参照

Visual Basic におけるシャドウ

2 つのプログラミング要素が同じ名前を持つときに、一方の要素によってもう一方の要素が隠されることがあります。これをシャドウと呼びます。そのような場合、シャドウで隠された要素は参照できません。要素の名前をコードで使用すると、Visual Basic コンパイラはそれをシャドウする要素 (隠されていない方の要素) に解決します。

目的

シャドウの主な目的は、派生クラスのメンバ定義を保護することにあります。基本クラスは、既に定義されている要素と同じ名前の要素を作成するように変更される可能性もあります。このような場合、**Shadows** 修飾子を指定してあると、派生クラスを通じた参照は新しい基本クラスの要素に解決されず、派生クラスで定義したメンバに解決されます。

シャドウの種類

要素が他の要素をシャドウするのは、次の 2 つの場合です。シャドウする方の要素が、シャドウされる要素を含む領域のサブ領域の内部で宣言されていると、スコープによってシャドウが発生します。または、派生クラスが基本クラスのメンバを再定義すると、継承によってシャドウが発生します。

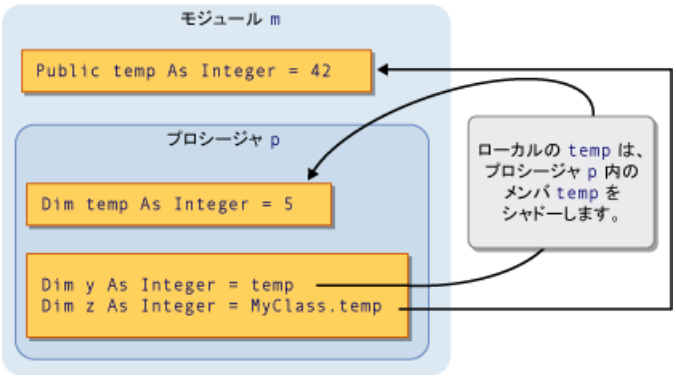
スコープによるシャドウ

同じモジュール、クラス、または構造体の中に、名前が同じでスコープが異なるプログラミング要素を含めることができます。このように宣言した 2 つの要素によって共有されている名前をコードが参照すると、スコープが狭い方の要素がもう一方の要素をシャドウします (最も狭いのはブロックスコープです)。

たとえば、モジュールで `temp` という名前の **Public** 変数を定義し、そのモジュール内のプロシージャで同じ `temp` という名前のローカル変数を宣言したとします。このプロシージャ内で `temp` を参照するとローカル変数へのアクセスとなり、プロシージャの外で `temp` を参照すると **Public** 変数へのアクセスとなります。この場合、プロシージャの変数 `temp` がモジュールの変数 `temp` をシャドウしています。

次の図は、同じ `temp` という名前を持つ 2 つの変数を示しています。ローカル変数 `temp` に、この変数が宣言されているプロシージャ `p` 内でアクセスした場合、メンバ変数 `temp` がシャドウされます。ただし、**MyClass** キーワードを使用すると、シャドウをバイパスしてメンバ変数にアクセスできます。

スコープによるシャドウ



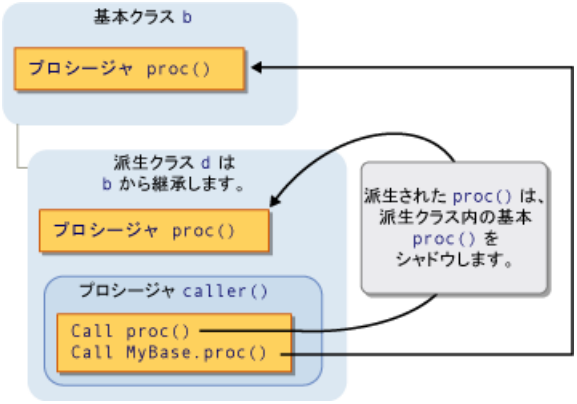
スコープによるシャドウの例については、「方法 : 自分で宣言した変数と同じ名前の変数を隠す」を参照してください。

継承によるシャドウ

基本クラスから継承したプログラミング要素を派生クラスで再定義すると、再定義した要素が元の要素をシャドウします。任意の型の宣言された要素 (またはオーバーロードされた要素の集合) を他の任意の型でシャドウできます。たとえば、**Integer** 型の変数が **Function** プロシージャをシャドウできます。プロシージャを他のプロシージャでシャドウすると、異なるパラメータ リストおよび異なる戻り値の型を使用できます。

次の図は、基本クラス `b` と、`b` を継承した派生クラス `d` を示しています。基本クラスに `proc` という名前のプロシージャが定義してあり、派生クラスでは同じ名前の別のプロシージャでこれをシャドウしています。最初の **Call** ステートメントは、シャドウしている派生クラスの `proc` にアクセスします。ただし、**MyBase** キーワードを使用すると、シャドウをバイパスして、シャドウされた基本クラスのプロシージャにアクセスできます。

継承によるシャドウ



継承によるシャドウの例については、「方法 : 自分で宣言した変数と同じ名前の変数を隠す」および「方法 : 継承された変数を隠す」を参照してください。

シャドウとアクセス レベル

派生クラスを使用しても、シャドウする要素にコードから必ずアクセスできるとは限りません。たとえば、シャドウする要素が **Private** で宣言されている場合があります。このような場合、シャドウは行われません。コンパイラは参照を、シャドウが行われない場合と同じ要素に解決します。この要素は、シャドウするクラスから最小の継承ステップを戻ってアクセスできる要素です。シャドウされる要素がプロシージャの場合、同じ名前、同じパラメータ リスト、および同じ戻り値の型を持つ最も近いアクセス可能なバージョンに参照が解決されます。

次の例は 3 つのクラスの継承階層を示します。各クラスが `display` という **Sub** プロシージャを定義し、各派生クラスが、それぞれの基本クラスの `display` プロシージャをシャドウしています。

```
Public Class firstClass
    Public Sub display()
        MsgBox("This is firstClass")
    End Sub
End Class
Public Class secondClass
    Inherits firstClass
    Private Shadows Sub display()
        MsgBox("This is secondClass")
    End Sub
End Class
Public Class thirdClass
    Inherits secondClass
    Public Shadows Sub display()
        MsgBox("This is thirdClass")
    End Sub
End Class
Module callDisplay
    Dim first As New firstClass
    Dim second As New secondClass
    Dim third As New thirdClass
    Public Sub callDisplayProcedures()
        ' The following statement displays "This is firstClass".
        first.display()
        ' The following statement displays "This is firstClass".
        second.display()
        ' The following statement displays "This is thirdClass".
        third.display()
    End Sub
End Module
```

上の例では、派生クラス `secondClass` は、**Private** なプロシージャによって `display` をシャドウします。`callDisplay` モジュールが `secondClass` の `display` を呼び出す場合、この呼び出し元のコードは `secondClass` の外側にあるため、プライベートな `display` プロシージャにアクセスできません。シャドウは行われず、コンパイラは、参照を基本クラスの `display` プロシージャに解決します。

一方、さらなる派生クラス `thirdClass` では `display` が **Public** として宣言されているため、`callDisplay` 内のコードからこれにアクセスできます。

シャドウとオーバーライド

シャドウをオーバーライドと混同しないでください。どちらも、派生クラスが基本クラスから継承するときに使用され、宣言された要素を他の要素で

再定義します。しかし、シャドウとオーバーライドの間には大きな違いがあります。比較については、「シャドウとオーバーライドの違い」を参照してください。

シャドウとオーバーロード

基本クラスの同じ要素を、派生クラスの複数の要素でシャドウする場合、シャドウする要素はその要素のオーバーロードされたバージョンになります。詳細については、「プロシージャのオーバーロード」を参照してください。

シャドウされた要素へのアクセス

派生クラスから要素にアクセスする場合は、通常、要素名を Me キーワードで修飾することによって、派生クラスの現在のインスタンスを通じてアクセスします。派生クラスが基本クラスの要素をシャドウしている場合、基本クラスの要素にアクセスするには、要素名を MyBase キーワードで修飾します。

シャドウされた要素にアクセスする方法の例については、「方法 : 派生クラスによって非表示になっている変数にアクセスする」を参照してください。

オブジェクト変数の宣言

派生クラスがシャドウする要素にアクセスするか、シャドウされる要素にアクセスするかは、オブジェクト変数を作成する方法によっても変わります。次の例では、派生クラスから 2 つのオブジェクトを作成していますが、1 つのオブジェクトは基本クラスとして、もう 1 つは派生クラスとして宣言しています。

```
Public Class baseCls
    ' The following statement declares the element that is to be shadowed.
    Public z As Integer = 100
End Class
Public Class dervCls
    Inherits baseCls
    ' The following statement declares the shadowing element.
    Public Shadows z As String = "*"
End Class
Public Class useClasses
    ' The following statement creates the object declared as the base class.
    Dim basObj As baseCls = New dervCls()
    ' Note that dervCls widens to its base class baseCls.
    ' The following statement creates the object declared as the derived class.
    Dim derObj As dervCls = New dervCls()
    Public Sub showZ()
    ' The following statement outputs 100 (the shadowed element).
        MsgBox("Accessed through base class: " & basObj.z)
    ' The following statement outputs "*" (the shadowing element).
        MsgBox("Accessed through derived class: " & derObj.z)
    End Sub
End Class
```

上の例では、変数 `basObj` は基本クラスとして宣言されています。この変数に `dervCls` オブジェクトを代入することは、拡大変換となるため有効です。ただし、基本クラスは派生クラス内のシャドウする変数 `z` にアクセスできないため、コンパイラは `basObj.z` を元の基本クラス値に解決します。

参照

関連項目

Shadows
Overrides
Me
MyBase

概念

Visual Basic におけるスコープ
拡大変換と縮小変換

その他の技術情報

宣言された要素の参照

シャドウとオーバーライドの違い

基本クラスを継承するクラスを定義する際に、基本クラスの要素のいくつかを派生クラスで再定義する場合があります。シャドウとオーバーライドは、どちらもこの目的で使用できます。

比較

シャドウとオーバーライドは混同しやすい機能です。どちらも、派生クラスが基本クラスから継承するときに使用され、宣言された要素を他の要素で再定義します。しかし、シャドウとオーバーライドの間には大きな違いがあります。

シャドウとオーバーライドの比較を次の表に示します。

比較のポイント	シャドウ	オーバーライド
目的	派生クラスで既に定義されているメンバを追加するような基本クラスの変更を防ぐ	呼び出しシーケンスが同じで実装が異なるプロシージャまたはプロパティを定義することにより、ポリモーフィズムを実現する [1]
再定義される要素	任意の宣言された要素型	プロシージャ (**Function**、**Sub**、**Operator**) またはプロパティのみ
再定義する要素	任意の宣言された要素型	呼び出しシーケンスが同じプロシージャまたはプロパティのみ [1]
再定義する要素のアクセス レベル	任意のアクセス レベル	オーバーライドされた要素のアクセス レベルは変更できない
再定義する要素の読み取りと書き込みの許可	任意の組み合わせ	オーバーライドされたプロパティの読み取りと書き込みの許可は変更できない
再定義の制御	基本クラスの要素にシャドウを強制または禁止することはできない	基本クラスの要素に **MustOverride**、**NotOverridable** または **Overridable** を指定できる
キーワードの使用法	派生クラスでは **Shadows** を推奨。**Shadows** と **Overrides** のどちらも指定されない場合は、**Shadows** と見なされる [2]	基本クラスでは **Overridable** または **MustOverride** が必要。派生クラスでは **Overrides** が必要
派生クラスから派生するクラスでの、再定義する要素の継承	シャドウする要素は以降の派生クラスでも継承され、シャドウされる要素は引き続き隠される [3]	オーバーライドする要素は以降の派生クラスでも継承され、オーバーライドされる要素は引き続きオーバーライドされる

[1] 呼び出しシーケンスは、要素型 (**Function**、**Sub**、**Operator**、または **Property**)、名前、パラメータ リスト、および戻り値の型で構成されます。プロシージャをプロパティでオーバーライドしたり、プロパティをプロシージャでオーバーライドしたりはできません。ある種類のプロシージャ (**Function**、**Sub**、または **Operator**) を他の種類のプロシージャでオーバーライドすることはできません。

[2] **Shadows** と **Overrides** のどちらも指定していない場合、どちらの再定義を使用するのかを確認するよう促すための警告メッセージがコンパイラから出力されます。警告を無視すると、シャドウが使用されます。

[3] 派生クラスからさらに派生するクラスで、シャドウする側の要素にアクセスできない場合、シャドウは継承されません。たとえば、シャドウする側の要素を **Private** として宣言した場合、派生クラスから派生するクラスは、その要素の代わりに元の要素を継承します。

ガイドライン

次のような場合は、通常、オーバーライドを使用します。

- ポリモーフィックな派生クラスを定義している場合。
- 安全のために、コンパイラで同一の要素型と呼び出しシーケンスを適用させる場合。

次のような場合は、通常、シャドウを使用します。

- 基本クラスが変更される可能性があり、使用している名前と同じ名前で要素を定義する可能性がある場合。
- 要素型や呼び出しシーケンスを自由に変更できるようにする場合。

参照

処理手順

方法 : 同じ名前を持つ 2 つの要素を区別する
方法 : 自分で宣言した変数と同じ名前の変数を隠す
方法 : 継承された変数を隠す
方法 : 派生クラスによって非表示になっている変数にアクセスする

関連項目

Shadows
Overrides

概念

同じ名前を持つ複数の変数がある場合に参照を解決する
Visual Basic におけるシャドウ

その他の技術情報

宣言された要素の参照

方法：自分で宣言した変数と同じ名前の変数を隠す

シャドウすることで、変数を隠すことができます。シャドウとは、同じ名前の変数を使用して、変数を再定義することです。以下のいずれかの方法で、隠す変数をシャドウできます。

- **スコープによるシャドウ。** 隠す変数が含まれる領域のサブ領域内で再宣言することにより、スコープによるシャドウを行うことができます。
- **継承によるシャドウ。** 隠す変数がクラス レベルで定義されている場合、派生クラスで Shadows キーワードを使用して再宣言することにより、継承によるシャドウを行うことができます。

変数を隠す 2 つの方法

スコープによるシャドウを行って、変数を隠すには

1. 隠す変数を定義する領域を決め、変数を使用して再定義するサブ領域を決めます。

変数の領域	再定義が許可されたサブ領域
モジュール	モジュール内のクラス
クラス	クラス内のサブクラス クラス内のプロシージャ

   たとえば、**If**...**End If** 構造または **For** ループ内など、プロシージャ内のブロックにあるプロシージャ変数は再定義できません。

2. サブ領域が既に存在していない場合は、作成します。
3. サブ領域内で、変数をシャドウする Dim ステートメント (Visual Basic) 宣言を記述します。

   サブ領域内のコードによって変数名が参照されているとき、変数のシャドウへの参照をコンパイラが解決します。

   スコープによるシャドウ、およびシャドウを使用しない参照の例を次に示します。

```
Module shadowByScope
    ' The following statement declares num as a module-level variable.
    Public num As Integer
    Sub show()
        ' The following statement declares num as a local variable.
        Dim num As Integer
        ' The following statement sets the value of the local variable.
        num = 2
        ' The following statement displays the module-level variable.
        MsgBox(CStr(shadowByScope.num))
    End Sub
    Sub useModuleLevelNum()
        ' The following statement sets the value of the module-level variable.
        num = 1
        show()
    End Sub
End Module
```

   上の例では、モジュール レベルおよびプロシージャ レベル (プロシージャ `show` 内) の両方で、`num` 変数が宣言されています。ローカル変数 `num` が、`show` 内のモジュール レベルの変数 `num` をシャドウします。したがって、ローカル変数が 2 に設定されます。しかし、`useModuleLevelNum` プロシージャ内の `num` をシャドウするローカル変数はありません。したがって、`useModuleLevelNum` によって、モジュール レベルの変数の値が 1 に設定されます。

   `show` 内部の **MsgBox** 呼び出しは、モジュール名を使用して `num` を修飾することにより、シャドウ機構を使用しないようにします。したがって、ローカル変数ではなく、モジュール レベルの変数が表示されます。

継承によるシャドウを行って、変数を隠すには

1. 隠す変数が、クラスおよびクラス レベル (プロシージャの外部) で宣言されているようにします。そうしないと、継承によるシャドウを行うことができません。
2. 変数のクラスから派生したクラスが既に存在しない場合、定義します。
3. 派生クラスの内部で、変数を宣言する **Dim** ステートメントを記述します。宣言には Shadows キーワードを含めます。

   派生クラスのコードによって変数名が参照されているとき、変数への参照をコンパイラが解決します。

   継承によるシャドウの例を次に示します。これにより 2 つの参照が作成されます。1 つはシャドウ変数にアクセスする参照、もう 1 つはシャドウを使用しない参照です。

```
Public Class shadowBaseClass
    Public shadowString As String = "This is the base class string."
End Class
Public Class shadowDerivedClass
    Inherits shadowBaseClass
    Public Shadows shadowString As String = "This is the derived class string."
    Public Sub showStrings()
        Dim s As String = "Unqualified shadowString: " & shadowString _
            & vbCrLf & "MyBase.shadowString: " & MyBase.shadowString
        MsgBox(s)
    End Sub
End Class
```

   上の例では、基本クラスで `shadowString` 変数を宣言し、派生クラスでシャドウします。名前 `shadowString` が修飾されていない場合、派生クラスの `showStrings` プロシージャには、文字列のシャドウ バージョンが表示されます。次に、`shadowString` が MyBase キーワードで修飾されている場合には、シャドウされたバージョンが表示されます。

堅牢性の高いプログラム

シャドウでは、同じ名前の複数のバージョンの変数を取り入れます。コード ステートメントによって変数名が参照されているとき、コンパイラによって参照が解決されるバージョンは、コード ステートメントの位置および修飾する文字列の存在などの要因に依存しています。これにより、意図しないバージョンのシャドウされた変数を参照するリスクが増加する可能性があります。シャドウされた変数へのすべての参照を完全修飾することによって、このリスクを低減させることができます。

参照

処理手順

方法 : 同じ名前を持つ 2 つの要素を区別する
方法 : 継承された変数を隠す
方法 : 派生クラスによって非表示になっている変数にアクセスする

関連項目

Overrides
MyBase

概念

同じ名前を持つ複数の変数がある場合に参照を解決する
Visual Basic におけるシャドウ
シャドウとオーバーライドの違い

その他の技術情報

宣言された要素の参照

方法：継承された変数を隠す

派生クラスは、基本クラスのすべての定義を継承します。基本クラスの要素として同じ名前を使用して変数を定義する場合、派生クラスに変数を定義する際に、基本クラスの要素を非表示にしたり、シャドウにできます。これを行う場合、シャドウ機構を明示的にバイパスしない限り、派生クラス内のコードはその変数にアクセスします。

継承された変数を非表示にするもう 1 つの理由は、基本クラスのリビジョン変更から保護することです。継承している要素を変更するように、基本クラスを変えることもできます。このような変更があった場合、**Shadows** 修飾子は、派生クラスからの参照を、基本クラスの要素に解決するのではなく、継承された変数に解決します。

継承された変数を隠すには

1. 隠す変数がクラス レベル (プロシージャの外) で宣言されていることを確認します。そうでない場合、これを隠す必要がありません。
2. 派生クラス内で、変数を宣言する Dim ステートメント (Visual Basic) を記述します。継承された変数の名前と同じ名前を使用します。
3. 宣言に Shadows キーワードを含めます。

   派生クラス内のコードから変数名を参照するとき、コンパイラは参照を継承した変数に解決します。

   継承された変数のシャドウの例を次に示します。

```
Public Class shadowBaseClass
    Public shadowString As String = "This is the base class string."
End Class
Public Class shadowDerivedClass
    Inherits shadowBaseClass
    Public Shadows shadowString As String = "This is the derived class string."
    Public Sub showStrings()
        Dim s As String = "Unqualified shadowString: " & shadowString _
            & vbCrLf & "MyBase.shadowString: " & MyBase.shadowString
        MsgBox(s)
    End Sub
End Class
```

   前の例では、基本クラスで `shadowString` 変数を宣言し、派生クラスでこれをシャドウしています。派生クラスのプロシージャ `showStrings` では、名前 `shadowString` が修飾されていない場合、文字列のシャドウ バージョンが表示されます。`shadowString` が MyBase キーワードによって修飾されている場合は、次にシャドウされたバージョンが表示されます。

堅牢性の高いプログラム

シャドウでは、同じ名前で複数のバージョンの変数を取り入れます。コード ステートメントで変数名を参照する際、コンパイラが参照を解決するバージョンは、コード ステートメントの場所や修飾文字列の有無などによって異なります。このことは、意図しないバージョンのシャドウ変数を参照してしまう危険性が増大することになります。シャドウ変数に対するすべての参照を完全に修飾することによって、この危険性を軽減することができます。

参照

処理手順

方法 : 同じ名前を持つ 2 つの要素を区別する
方法 : 自分で宣言した変数と同じ名前の変数を隠す
方法 : 派生クラスによって非表示になっている変数にアクセスする

関連項目

Overrides

概念

同じ名前を持つ複数の変数がある場合に参照を解決する
Visual Basic におけるシャドウ
シャドウとオーバーライドの違い

その他の技術情報

宣言された要素の参照

方法：派生クラスによって非表示になっている変数にアクセスする

派生クラスのコードが変数にアクセスする場合、通常はコンパイラによって、アクセス可能な最も近いバージョンに参照が解決されます。つまり、アクセスするクラスから、最小の派生ステップを戻ってアクセスできるバージョンに参照が解決されます。変数が派生クラスで定義されている場合、通常、コードはその定義にアクセスします。

派生クラスの変数によって基本クラスの変数がシャドウされる場合、基本クラスのバージョンは非表示になります。しかし、MyBase キーワードで修飾することにより、基本クラスの変数にアクセスできます。

派生クラスによって非表示になっている基本クラスの変数にアクセスするには

- 式または代入ステートメントでは、変数名の前に **MyBase** キーワードおよびピリオド (.) を記述します。

  コンパイラによって、変数の基本クラスのバージョンへの参照が解決されます。

  継承によるシャドウの例を次に示します。これにより 2 つの参照が作成されます。1 つはシャドウ変数にアクセスする参照、もう 1 つはシャドウを使用しない参照です。

```
Public Class shadowBaseClass
    Public shadowString As String = "This is the base class string."
End Class
Public Class shadowDerivedClass
    Inherits shadowBaseClass
    Public Shadows shadowString As String = "This is the derived class string."
    Public Sub showStrings()
        Dim s As String = "Unqualified shadowString: " & shadowString _
            & vbCrLf & "MyBase.shadowString: " & MyBase.shadowString
        MsgBox(s)
    End Sub
End Class
```

  上の例では、基本クラスで `shadowString` 変数を宣言し、派生クラスでシャドウします。`shadowString` の名前が修飾されていない場合、派生クラスの `showStrings` プロシージャに、シャドウする文字列が表示されます。次に、`shadowString` が MyBase キーワードで修飾されたときに、シャドウされた文字列が表示されます。

堅牢性の高いプログラム

シャドウされた変数へのすべての参照を完全に修飾することで、シャドウされた変数の意図しないバージョンを参照するリスクを軽減できます。シャドウにより、同じ名前の複数の変数が追加されます。コード ステートメントによって変数名が参照されているとき、コンパイラによって参照が解決されるバージョンは、コード ステートメントの位置および修飾する文字列の存在などの要因より決まります。これにより、誤った変数を参照するリスクが増加する可能性があります。

参照

処理手順

方法 : 同じ名前を持つ 2 つの要素を区別する
方法 : 自分で宣言した変数と同じ名前の変数を隠す
方法 : 継承された変数を隠す

関連項目

Shadows
Overrides

概念

同じ名前を持つ複数の変数がある場合に参照を解決する
Visual Basic におけるシャドウ
シャドウとオーバーライドの違い

その他の技術情報

宣言された要素の参照

Visual Basic におけるデータ型

プログラミング要素のデータ型は、保持できるデータの種類やデータの格納方法を示します。データ型は、コンピュータのメモリに格納できるすべての値に適用され、式の評価にも関与します。変数、リテラル、定数、列挙値、プロパティ、プロシージャのパラメータ、プロシージャの引数、およびプロシージャの戻り値にはすべてデータ型があります。

データ型の宣言

型宣言を省略したプログラミング以外では、すべてのプログラミング要素のデータ型を宣言する必要があります。詳細については、「Visual Basic での型宣言を省略したプログラミング」を参照してください。

プログラミング要素は宣言ステートメントで定義し、**As** 句を使用してデータ型を指定します。さまざまな要素の宣言に使用するステートメントを次の表に示します。

プログラミング要素	データ型の宣言

変数	Dim ステートメント (Visual Basic) 内で宣言
	`Dim amount As Double`
	`Static yourName As String`
	`Public billsPaid As Decimal = 0`

リテラル	リテラルの型宣言文字を使用する場合は、「型文字」の「リテラルの型文字」を参照してください。 `Dim searchChar As Char = "."C`
定数	Const ステートメント (Visual Basic) 内で宣言 `Const modulus As Single = 4.17825F`
列挙値	Enum ステートメント (Visual Basic) 内で宣言 `Public Enum colors`
プロパティ	Property ステートメント内で宣言 `Property region() As String`
プロシージャ パラメータ	Sub ステートメント (Visual Basic)、Function ステートメント (Visual Basic)、または Operator ステートメント内で宣言 `Sub addSale(ByVal amount As Double)`
プロシージャの引数	呼び出し元のコードでは、宣言済みのプログラミング要素、または宣言済みの要素を含む式を各引数に指定 `subString = Left( inputString , 5 )`
プロシージャの戻り値	Function ステートメント (Visual Basic) または Operator ステートメント内で宣言 `Function convert(ByVal b As Byte) As String`

参照

処理手順

データ型のトラブルシューティング

関連項目

データ型の概要 (Visual Basic)

概念

型文字
複合データ型
Visual Basic におけるジェネリック型
Visual Basic での型宣言を省略したプログラミング

その他の技術情報

基本データ型
データ型の実装

Visual Basic における型変換
構造体 : 独自のデータ型

型文字

宣言ステートメントでデータ型を指定するだけでなく、型文字を使用してプログラミング要素のデータ型を強制的に指定することもできます。型文字は要素の直後に指定する必要があります。間にはどのような文字も指定しないでください。

型文字は要素名には含まれません。型文字を使用して定義した要素を参照するときには、型文字を省略できます。

識別子の型文字

Visual Basic には識別子の型文字が用意されています。それらの型文字は、宣言で変数または定数のデータ型を指定するために使用できます。次の表は、使用できる識別子の型文字とその使用例を示しています。

識別子の型文字	データ型	例
%	**Integer**	`Dim L%`
&	**Long**	`Dim M&`
@	**Decimal**	`Const W@ = 37.5`
!	**Single**	`Dim Q!`
#	**Double**	`Dim X#`
$	**String**	`Dim V$ = "Secret"`

Boolean、**Byte**、**Char**、**Date**、**Object**、**SByte**、**Short**、**UInteger**、**ULong**、および **UShort** の各データ型や、配列、構造体などの複合データ型に対応する識別子の型文字はありません。

場合によっては、**$** 文字を Visual Basic の関数に追加することもできます。たとえば、**Left** ではなく **Left$** を使用すると、**String** 型の戻り値を取得できます。

いずれの場合でも、識別子の型文字は識別子名の直後に指定する必要があります。

リテラルの型文字

リテラルは、何らかのデータ型の特定の値のテキスト表現です。

既定のリテラル型

通常、コード内のリテラルの形式によってその値のデータ型が決まります。それらの既定の型を次に示します。

リテラルのテキスト形式	既定のデータ型	例
小数部のない数値	**Integer**	`2147483647`
小数部がなく、**Integer** 型で表すには大きすぎる数値	**Long**	`2147483648`
小数部のある数値	**Double**	`1.2`
二重引用符で囲まれたリテラル	**String**	`"A"`
シャープ記号で囲まれたリテラル	**Date**	`#5/17/1993 9:32 AM#`

強制的なリテラル型

Visual Basic には、リテラルの型文字が用意されています。それらの型文字は、リテラルに対して、そのリテラル形式が示す以外のデータ型を指定するために使用できます。このためには、リテラルの型文字をリテラルの最後に付けます。次の表は、使用できるリテラルの型文字とその使用例を示しています。

リテラルの型文字	データ型	例

S	**Short**	`I = 347S`
I	**Integer**	`J = 347I`
L	**Long**	`K = 347L`
D	**Decimal**	`X = 347D`
F	**Single**	`Y = 347F`
R	**Double**	`Z = 347R`
US	**UShort**	`L = 347US`
UI	**UInteger**	`M = 347UI`
UL	**ULong**	`N = 347UL`
C	**Char**	`Q = "."C`

Boolean、**Byte**、**Date**、**Object**、**SByte**、および **String** の各データ型や、配列、構造体などの複合データ型に対応するリテラルの型文字はありません。

また、変数、定数、および式に使用する場合と同様に、リテラルにも識別子の型文字 (**%**、**&**、**@**、**!**、**#**、**$**) を使用できます。ただし、リテラルの型文字 (**S**、**I**、**L**、**D**、**F**、**R**、**C**) はリテラルにしか使用できません。

いずれの場合でも、リテラルの型文字はリテラル値の直後に指定する必要があります。

16 進数および 8 進数リテラル

通常、コンパイラは、整数リテラルを 10 進数 (基数 10) として解釈します。整数リテラルにプレフィックス **&H** を付けると、強制的に 16 進数 (基数 16) として解釈します。また、プレフィックス **&O** を付けると、8 進数 (基数 8) として解釈します。プレフィクスの後には、該当する記数法に従った文字で値を記述する必要があります。たとえば、次のように表されます。

基数	プレフィックス	有効な文字	例
16 進数 (基数 16)	**&H**	0-9 および A-F	`&HFFFF`
10 進数 (基数 8)	**&O**	0-7	`&O77`

プレフィックスを付けたリテラルの後には、リテラルの型文字を続けることができます。次に例を示します。

```
Dim counter As Short = &H8000S
Dim flags As UShort = &H8000US
```

上の例では、`counter` には 10 進数の -32768 が設定され、`flags` には 10 進数の +32768.が設定されます。

参照

処理手順

データ型のトラブルシューティング

関連項目

データ型の概要 (Visual Basic)

概念

Visual Basic におけるデータ型
Visual Basic での型宣言を省略したプログラミング
Visual Basic での変数宣言

その他の技術情報

基本データ型
データ型の実装

Visual Basic における型変換

基本データ型

Visual Basic には、多くのプログラミング要素として利用できる、一連の定義済みのデータ型が用意されています。このセクションでは、これらの型とその使用方法について説明します。

このセクションの内容

数値データ型

整数型および非整数型の数値型について説明します。

文字データ型

Char および **String** について説明します。

その他のデータ型

ブール型 (**Boolean**)、日付型 (**Date**)、オブジェクト型 (**Object**) の各データ型について説明します。

関連するセクション

Visual Basic におけるデータ型

Visual Basic データ型を示し、使用方法について説明します。

データ型の概要 (Visual Basic)

Visual Basic で用意されている基本データ型の概要を示します。

数値データ型

Visual Basic には、さまざまな表現で数値を処理するための数値データ型が用意されています。整数型は整数 (正、負、および 0) だけを表し、非整数型は整数部分と小数部分がある数値を表します。

Visual Basic データ型の side-by-side の比較を示す表については、「データ型の概要 (Visual Basic)」を参照してください。

整数型

整数型は、小数部分のない数だけを表すデータ型です。

符号付き整数型は、SByte 型 (Visual Basic) (8 ビット)、Short 型 (Visual Basic) (16 ビット)、整数型 (Integer) (Visual Basic) (32 ビット)、および Long データ型 (Visual Basic) (64 ビット) です。小数値ではなく、整数が常に変数に格納されている場合、次の型のいずれかとして宣言します。

符号なし整数型は、バイト型 (Byte) (Visual Basic) (8 ビット)、UShort 型 (Visual Basic) (16 ビット)、UInteger データ型 (32 ビット)、および ULong データ型 (Visual Basic) (64 ビット) です。変数にバイナリ データまたは状態が不明のデータが含まれる場合は、これらの型のいずれかとして変数を宣言します。

パフォーマンス

算術演算の処理は、他のデータ型よりも整数型の方が高速です。算術演算の処理は、Visual Basic では、**Integer** 型および **UInteger** 型で最速です。

大きい整数

Integer データ型が保持できる整数よりも、より大きい整数を保持する必要がある場合、代わりに、**Long** データ型を使用できます。**Long** 変数は、-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 までの数を保持できます。**Long** を使用した演算は、**Integer** を使用した場合よりも少し遅くなります。

さらに大きい値が必要な場合は、10 進型 (Decimal) (Visual Basic) を使用できます。小数部分を使用しない場合、**Decimal** 変数には、-79,228,162,514,264,337,593,543,950,335 から 79,228,162,514,264,337,593,543,950,335 までの数を保持できます。しかし、**Decimal** の数値を使用した演算は、他の数値データ型を使用した場合よりも大幅に遅くなります。

小さい整数

Integer データ型のすべての範囲が必要ない場合、-32,768 から 32,767 までの整数を保持できる、**Short** データ型を使用できます。最も小さい整数の範囲に対しては、**SByte** データ型に、-128 から 127 までの整数が保持されます。小さい整数が保持される変数が非常に多くある場合、**Short** 変数および **SByte** 変数を、共通言語ランタイムに格納することもできます。この方法はより効率的であり、メモリの使用量を節約できます。しかし、**Short** および **SByte** を使用した演算は、**Integer** を使用した演算よりも多少遅くなります。

符号なし整数

変数が負の数値を保持する必要がないことが分かっている場合、符号なしの型 **Byte**、**UShort**、**UInteger**、および **ULong** を使用できます。これらのデータ型のそれぞれで、対応する符号付き型の 2 倍の正の整数を保持できます (**SByte**、**Short**、**Integer**、および **Long**)。パフォーマンスの面では、それぞれの符号なしの型は、対応する符号付き型と同じくらい効率的です。特に、**UInteger** と **Integer** は、共に非常に効率的にすべての基本数値データ型を処理します。

非整数型の数値型

非整数型のデータ型は、整数部分と小数部分の両方を含む数値を表す型です。

非整数型の数値データ型は、**Decimal** (128 ビットの固定小数点)、単精度浮動小数点型 (Single) (Visual Basic) (32 ビットの浮動小数点)、および 倍精度浮動小数点数型 (Double) (Visual Basic) (64 ビットの浮動小数点) です。これらの型はすべて符号付きです。変数に小数が含まれる可能性がある場合には、この 3 つのいずれかの型で宣言します。

Decimal は、浮動小数点のデータ型ではありません。**Decimal** の数値には、バイナリ整数値、および値のどの部分が小数部分であるのかを指定する、整数スケーリング要因があります。

浮動小数点数 (**Single** および **Double**) の数値は **Decimal** の数値よりも大きい範囲ですが、丸め誤差が発生する可能性があります。浮動小数点型がサポートする有効桁数は **Decimal** よりも少ないですが、より大きい値を表すことができます。

非整数型の数値は、*mmmEeee* として表すことができます。ここでは、*mmm* は 仮数 (有効桁数) であり、*eee* は指数 (10 の累乗) です。非整数型の最も大きい正の値は、**Decimal** では 7.9228162514264337593543950335E+28、**Single** では 3.4028235E+38、および **Double** では 1.79769313486231570E+308 です。

パフォーマンス

現在のプラットフォーム上のプロセッサでは、浮動小数点の演算が倍精度で実行されるため、**Double** は、最も効率的な小数データ型です。

しかし、**Double** を使用した演算は、**Integer** などの整数型を使用した演算ほどは速くありません。

小さい絶対値

考えられる最も小さい絶対値 (0 に最も近い) を持つ数に対しては、負の値には -4.94065645841246544E-324 を、および正の値には 4.94065645841246544E-324 の数を、**Double** 変数に保持できます。

小さい小数値

Double データ型のすべての範囲が必要ない場合、-3.4028235E+38 から 3.4028235E+38 までの浮動小数点数を保持できる、**Single** データ型を使用できます。**Single** 変数の最も小さいマグニチュードは、負の値に対しては -1.401298E-45、正の値に対しては 1.401298E-45 です。小さい浮動小数点数が保持される変数が非常に多くある場合、**Single** 変数を、共通言語ランタイムに格納することもできます。この方法はより効率的であり、メモリの使用量を節約できます。

参照

処理手順

データ型のトラブルシューティング
方法 : 変数に整数を保持する
方法 : 変数で小数桁を保持する
方法 : 変数に可能な最大値を保持する
方法 : unsigned 型を使用して正の整数のストレージを最適化する
方法 : 符号なしの型を使用する Windows の機能を呼び出す
方法 : 変数で最大有効桁数を保持する
方法 : 変数で通貨値を保持する

概念

文字データ型
その他のデータ型

その他の技術情報

基本データ型

方法：変数に整数を保持する

整数には、正の数、ゼロ、負の数がすべて含まれます。変数を **Integer** 型で宣言すると、整数を保持できます。整数型 (Integer) (Visual Basic) には、-2,147,483,648 ～ 2,147,483,647 の整数を保持できます。

整数を保持するためには、Long データ型 (Visual Basic)、Short 型 (Visual Basic)、および SByte 型 (Visual Basic) も使用できます。詳細については、「数値データ型」を参照してください。

変数に整数を保持するには

1. 変数を Dim ステートメント (Visual Basic) で宣言します。
2. 変数名の後に **As** 句を指定します。
3. **As** キーワードの後に **Integer** キーワードを指定します。

```
Dim students As Integer
```

参照

関連項目

データ型の概要 (Visual Basic)

概念

Visual Basic におけるデータ型

型文字

その他の技術情報

基本データ型

方法：変数に可能な最大値を保持する

変数を **Decimal** のデータ型として宣言すると、有効桁数を持つ、考えられる最大値が保持されます。次に大きな整数の収容力を持つのは、**ULong** のデータ型です。整数データ型の精度が必要ない場合は、より大きな値を許容する浮動小数点型を使用できます。

"最大" という言葉の 2 つの意味

最大の正確な値。 単位桁までの完全な精度を持つ大きな整数を保持する必要がある場合は、10 進型 (Decimal) (Visual Basic) を使用できます。**Decimal** 型は、-79,228,162,514,264,337,593,543,950,335 ～ 79,228,162,514,264,337,593,543,950,335 (*7.9...E+28*) の範囲の整数を保持できます。

最大マグニチュード。 小数型 **Single** および **Double** では、マグニチュードの大きな値を保持できますが、精度が伴わない場合があります。単精度浮動小数点型 (Single) (Visual Basic) では 8 桁の有効桁数がサポートされ、
倍精度浮動小数点数型 (Double) (Visual Basic) では 18 桁の有効桁数がサポートされます。

変数で最大の整数値を保持するには

1. 変数を Dim ステートメント (Visual Basic) で宣言します。
2. 変数名の後に **As** 句を入力し、**Decimal** キーワードを指定します。

```
Dim atomsInTheUniverse As Decimal
```

効率的なデータ型

Decimal 型は、基本数値データ型の中でパフォーマンスが最も低い型です。整数の値がそれほど大きなものでなく、常に正かゼロである場合は、**ULong** 型の使用を検討します。

ULong データ型 (Visual Basic) の変数は、0 ～ 18,446,744,073,709,551,615 (*1.8...E+19*) の整数を保持できます。**ULong** 数値を使用した操作は、**Decimal** を使用した場合に比べてはるかに速いですが、**UInteger** を使用したときほど効率的ではありません。

効率的なパフォーマンスを維持しながら負でない大きな整数を変数で保持するには

1. **Dim** ステートメントで変数を宣言します。
2. 変数名の後に **As** 句を入力し、**ULong** キーワードを指定します。

```
Dim atomsInTheEarth As ULong
```

参照

関連項目

データ型の概要 (Visual Basic)
整数型 (Integer) (Visual Basic)
Long データ型 (Visual Basic)
10 進型 (Decimal) (Visual Basic)
UInteger データ型
ULong データ型 (Visual Basic)

概念

型文字

その他の技術情報

基本データ型

方法：unsigned 型を使用して正の整数のストレージを最適化する

正の値 (または 0) だけを格納し、値が 4,294,967,295 を超えない変数の場合、**Long** ではなく **UInteger** で宣言できます。

UInteger の利点は、32 ビット プラットフォームにおいて、32 ビットの整数型である **Integer** 型と **UInteger** 型は最も効率的なデータ型であり、アプリケーションのパフォーマンスが最適化されることです。

正の値が 2,147,483,647 を超えない場合には、**Integer** 型の変数を使用できます。

正の値だけを格納する整数を宣言するには

- 変数を **As UInteger** で宣言します。次に例を示します。

```
Public Function memoryRequired(ByVal m As UInteger) As UInteger
    Static r As UInteger = 0
    Try
        r += m
    Catch eo As System.OverflowException
        r = 0
    Catch ex As System.Exception
        MsgBox("Incrementing required memory causes """ & ex.Message & """")
    End Try
    Return r
End Function
```

次のコードを使用すると `memoryRequired` 関数の機能を確認できます。

```
Public Sub consumeMemoryRequired()
    Dim m1 As UInteger = UInteger.MaxValue - 100
    Dim m2 As UInteger = 100
    MsgBox("Max = " & CStr(UInteger.MaxValue) & vbCrLf & _
        CStr(m1) & " -> " & CStr(memoryRequired(m1)) & vbCrLf & _
        "+ " & CStr(m2) & " -> " & CStr(memoryRequired(m2)) _
        & vbCrLf & "+ 1 -> " & CStr(memoryRequired(1)))
End Sub
```

注意

UInteger データ型は、共通言語仕様 (CLS: Common Language Specification) の一部ではありません。したがって、CLS 準拠のコードでは、このデータ型を使用しているコンポーネントを使用できません。

参照

処理手順

方法 : 符号なしの型を使用する Windows の機能を呼び出す

関連項目

データ型の概要 (Visual Basic)

整数型 (Integer) (Visual Basic)

UInteger データ型

方法：符号なしの型を使用する Windows の機能を呼び出す

符号なし整数型のメンバがあるクラス、モジュール、または構造体を使用している場合、Visual Basic を使用してこれらのメンバにアクセスできます。

符号なしの型を使用する Windows の機能を呼び出すには

1. Declare ステートメントを使用して、機能を保持しているライブラリ、ライブラリ内での名前、呼び出し元のシーケンス、呼び出し時に文字列を変換する方法を Visual Basic に伝えます。
2. **Declare** ステートメントで、符号なしの型を持つ各パラメータに応じて **UInteger**、**ULong**、**UShort**、または **Byte** を使用します。
3. 呼び出している Windows の機能のドキュメントを参照し、使用されている定数の名前および値を調べます。これらの多くは、WinUser.h ファイルで定義されています。
4. コードで、必要な定数を宣言します。多くの Windows 定数は 32 ビットの符号なしの値であり、これらの **As UInteger** を宣言する必要があります。
5. 通常の方法で機能を呼び出します。符号なしの整数の引数を使用する、Windows の機能である **MessageBox** を呼び出す例を次に示します。

```
Public Class windowsMessage
    Private Declare Auto Function mb Lib "user32.dll" Alias "MessageBox" _
        (ByVal hWnd As Integer, _
        ByVal lpText As String, _
        ByVal lpCaption As String, _
        ByVal uType As UInteger) As Integer
    Private Const MB_OK As UInteger = 0
    Private Const MB_ICONEXCLAMATION As UInteger = &H30
    Private Const IDOK As UInteger = 1
    Private Const IDCLOSE As UInteger = 8
    Private Const c As UInteger = MB_OK Or MB_ICONEXCLAMATION
    Public Function messageThroughWindows() As String
        Dim r As Integer = mb(0, "Click OK if you see this!", _
            "Windows API call", c)
        Dim s As String = "Windows API MessageBox returned " _
            & CStr(r)& vbCrLf & "(IDOK = " & CStr(IDOK) _
            & ", IDCLOSE = " & CStr(IDCLOSE) & ")"
        Return s
    End Function
End Class
```

`messageThroughWindows` の機能をテストするためには、次のコードを使用してください。

```
Public Sub consumeWindowsMessage()
    Dim w As New windowsMessage
    w.messageThroughWindows()
End Sub
```

注意

UInteger、**ULong**、**UShort**、および **SByte** データ型は、共通言語仕様 (CLS) 一部ではありません。したがって、CLS 準拠のコードでは、これらを使用するコンポーネントを使用できません。

セキュリティに関するメモ：

Windows アプリケーション プログラミング インターフェイス (API: Application Programming Interface) などのアンマネージ コードへの呼び出しを実行すると、潜在的なセキュリティ リスクにコードが公開されます。

セキュリティに関するメモ：
Windows API の呼び出しにはアンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

参照

処理手順

方法 : unsigned 型を使用して正の整数のストレージを最適化する
チュートリアル : Windows API の呼び出し

関連項目

データ型の概要 (Visual Basic)
整数型 (Integer) (Visual Basic)
UInteger データ型
Declare ステートメント

方法：変数で小数桁を保持する

変数を **Double** 型として宣言すると、小数桁のある数値を保持できます。倍精度浮動小数点数型 (Double) (Visual Basic) で保持できる浮動小数点の範囲は -1.79769313486231570E+308 ～ 1.79769313486231570E+308 です。

単精度浮動小数点型 (Single) (Visual Basic) または 10 進型 (Decimal) (Visual Basic) を使用しても小数を保持できます。詳細については、「数値データ型」を参照してください。

変数で小数桁を保持するには

1. Dim ステートメント (Visual Basic) を指定して変数を宣言します。
2. 変数名の後に **As** 句を指定します。
3. **As** キーワードの後に **Double** キーワードを指定します。

参照

関連項目

データ型の概要 (Visual Basic)

概念

型文字

数値データ型

その他の技術情報

基本データ型

方法：変数で最大有効桁数を保持する

変数を **Decimal** データ型として宣言すると、小数桁のある数値を保持できます。

10 進型 (Decimal) (Visual Basic) の変数は、小数点以下 28 桁を含む、有効桁数 29 の数字を保持できます。

Performance.Decimal は、数値データ型の中では最も非効率な型です。**Decimal** データ型の完全な精度が必要でない場合は、**Double** データ型を使用でき、有効桁数 18 を保持できます。**Double** を指定した場合の操作は、**Decimal** を指定した場合より早くなり、**Double** 変数で使用するメモリ空間はより少なくてすみます。**Double** の詳細については、「方法 : 変数で小数桁を保持する」を参照してください。

データ型を選択する前に、精度の重要性とパフォーマンスとを比較検討する必要があります。

Larger Magnitude.変数で **Decimal** データ型よりも大きな数字の値を保持する必要がある場合は、より大きなデータ範囲をサポートする **Double** または **Single** データ型を使用できます。**Decimal** 変数では最高 7.92281625142643375935439503 35E+28、**Single** 変数では最高 3.4028235E+38、および **Double** 変数では最高 1.79769313486231570E+308 の値を保持できます。

変数で最大有効桁数を保持するには

1. Dim ステートメント (Visual Basic) を指定して変数を宣言します。
2. 変数名の後に **As** 句を指定します。
3. **As** キーワードの後に **Decimal** キーワードを指定します。

参照

関連項目

データ型の概要 (Visual Basic)
10 進型 (Decimal) (Visual Basic)
倍精度浮動小数点数型 (Double) (Visual Basic)

概念

型文字

その他の技術情報

基本データ型

方法：変数で通貨値を保持する

変数を **Decimal** のデータ型として宣言すると、通貨値が変数に保持されます。

10 進型 (Decimal) (Visual Basic) の変数は、小数点以下 28 桁を含む、有効桁数 29 の数字を保持できます。通貨値に関しては、必要な小数点以下の桁数は通常 2 桁または 3 桁です。しかし、トランザクション実行中の中間結果としては、たとえば利率で掛け算する場合など、正確性を保つためより多くの小数点以下の桁数が必要な場合があります。

通貨値で **Decimal** 変数を使用する利点は、値の正確さにあります。**Double** データ型は高速で少ないメモリしか使用しませんが、四捨五入による誤差が生じる可能性があります。**Decimal** データ型は、必要に応じて 28 桁までの正確性を完全に維持します。

末尾のゼロ文字。 Visual Basic には、**Double** など浮動小数点データ型における末尾のゼロ文字による内部表現がありません。**Decimal** 変数では計算によって取得された末尾のゼロが保持されますが、**Decimal** リテラルでは末尾のゼロは保存されません。使用例を含む詳細については、「10 進型 (Decimal) (Visual Basic)」を参照してください。

Visual Basic では、表示または印刷の際に末尾のゼロを表示しない場合があります。たとえば、値 4.2000 は、4.2 と表示される場合があります。該当するデータ型に対して System.Decimal.ToString または System.Double.ToString メソッドを呼び出すと、より綿密に出力形式を制御できます。

変数で通貨値を保持するには

1. Dim ステートメント (Visual Basic) を指定して変数を宣言します。
2. 変数名の後に **As** 句を指定します。
3. **As** キーワードの後に **Decimal** キーワードを指定します。

参照

関連項目

データ型の概要 (Visual Basic)
単精度浮動小数点型 (Single) (Visual Basic)
10 進型 (Decimal) (Visual Basic)
倍精度浮動小数点数型 (Double) (Visual Basic)

概念

Visual Basic におけるデータ型
型文字

その他の技術情報

基本データ型

文字データ型

Visual Basic には、出力する文字や表示する文字を処理する文字データ型が用意されています。char 型 (**Char**) は 1 文字、文字列型 (**String**) は不特定数の文字を格納します。どちらの型も Unicode 文字を処理します。

「データ型の概要 (Visual Basic)」には、Visual Basic のデータ型を並べて比較した表があります。

char 型 (Char)

char データ型 (**Char**) は、2 バイト (16 ビット) の Unicode 文字の 1 文字です。常に 1 文字しか格納しない変数の場合は、**Char** として宣言します。

詳細については、「文字型 (Char) (Visual Basic)」を参照してください。

文字列型 (String)

文字列型 (**String**) は、0 個以上の 2 バイト (16 ビット) の Unicode 文字列です。変数に格納する文字数が不定の場合は、変数を **String** 型で宣言します。

詳細については、「文字列型 (String) (Visual Basic)」を参照してください。

参照

処理手順

データ型のトラブルシューティング
方法 : 変数内に文字を保持する

概念

複合データ型
Visual Basic におけるジェネリック型
Visual Basic での型宣言を省略したプログラミング
型文字

その他の技術情報

基本データ型
データ型の実装
Visual Basic における型変換

方法：変数内に文字を保持する

変数を **Char** または **String** のデータ型として宣言すると、変数で個別の文字が保持されます。

文字型 (Char) (Visual Basic) の変数は、Unicode 1 文字を保持できます。文字列型 (String) (Visual Basic) の変数は、0 個以上の Unicode 文字列を保持できます。

Unicode 文字。Char または **String** 変数で指定可能な値は、Unicode 文字セット内のコード ポイント、または文字コードです。Unicode 文字には、基本 ASCII 文字セット、その他各種のアルファベット文字、アクセント、通貨記号、分数、発音記号、数学および科学技術記号が含まれます。

1 つの変数内で複数の文字を保持する必要がまったくない場合は、**Char** データ型を使用します。**String** データ型はより多くのメモリを必要とするため、パフォーマンスが低下します。

メモ：
Unicode 文字セットでは、コード ポイント D800 ～ DFFF (10 進数表現で 55296 ～ 55551) をサロゲート ペア用に予約します。サロゲート ペアでは、単一のコード ポイントを表すのに 2 つの 16 ビット値が必要です。**Char** 変数では、サロゲート ペアを保持することはできません。また、**String** では、このようなペアを保持するためには 2 桁を使用します。

変数で単一の文字を保持するには

1. Dim ステートメント (Visual Basic) を使用して変数を宣言します。
2. 変数名の後に **As** 句を指定します。
3. **As** キーワードの後に **Char** キーワードを指定します。

変数内に連続する文字を保持するには

1. **Dim** ステートメントを使用して変数を宣言します。
2. 変数名の後に **As** 句を指定します。
3. **As** キーワードの後に **String** キーワードを指定します。

参照

関連項目

データ型の概要 (Visual Basic)
単精度浮動小数点型 (Single) (Visual Basic)
文字型 (Char) (Visual Basic)
文字列型 (String) (Visual Basic)

概念

Visual Basic におけるデータ型
型文字

その他の技術情報

基本データ型

その他のデータ型

Visual Basic には、数値や文字を対象としないデータ型が用意されています。これらのデータ型では、Yes/No 型や日付/時刻型、およびオブジェクト アドレスなどの特殊なデータが処理されます。

「データ型の概要 (Visual Basic)」には、Visual Basic のデータ型を並べて比較した表があります。

ブール型 (Boolean)

ブール型 (Boolean) (Visual Basic) は、**True** と **False** のいずれかに解釈される符号なしの値です。データのサイズは、実装されているプラットフォームによって異なります。変数に真/偽、はい/いいえ、オン/オフなど 2 つの状態の値しか含まれない場合は、ブール型 (**Boolean**) として変数を宣言します。

日付型 (Date)

日付型 (Date) (Visual Basic) は、日付と時刻の両方の情報を保持する 64 ビットの値です。増分は、グレゴリオ暦の西暦 1 年 1 月 1 日午前 12 時からの経過時間の 100 ナノ秒を表します。変数に日付値、時刻値、またはその両方が含まれる場合は、日付型 (**Date**) として変数を宣言します。

オブジェクト型 (Object)

オブジェクト型 (Object) は、そのアプリケーション内または他のアプリケーション内にあるオブジェクト インスタンスを指す 32 ビットのアドレスです。**Object** 変数は、アプリケーションで認識できる各オブジェクトや各データ型のデータに関連付けることができます。コンパイル時には判断できないクラスのインスタンスへのポインタを変数に格納する場合や、変数でさまざまなデータ型のデータを指せるようにする場合は、オブジェクト型 (**Object**) として変数を宣言します。

参照

処理手順

データ型のトラブルシューティング
方法 : True および False の値を変数に保持する
方法 : 日付および時刻の値を変数に保持する
方法 : 型が不明なデータを変数に保持する

概念

型文字
数値データ型
文字データ型

その他の技術情報

基本データ型

方法：True および False の値を変数に保持する

Boolean データ型として宣言した場合、変数によって true および false の値が保持されます。

ブール型 (Boolean) (Visual Basic) の変数には、true/false、on/off、および yes/no などの 2 状態論理値を保持できます。

数値を保持する必要がある場合、**Boolean** ではなく、**Integer** または **Double** などの数値のデータ型を使用します。**Boolean** 値および同等の数値を、数として使用しないでください。詳細については、「データ型のトラブルシューティング」を参照してください。

true および false の値を変数に保持するには

1. 変数を Dim ステートメント (Visual Basic) で宣言します。
2. 変数名の後に **As** 句を指定します。
3. **As** キーワードの後に **Boolean** キーワードを指定します。

参照

処理手順

データ型のトラブルシューティング

関連項目

データ型の概要 (Visual Basic)
単精度浮動小数点型 (Single) (Visual Basic)
ブール型 (Boolean) (Visual Basic)
整数型 (Integer) (Visual Basic)
倍精度浮動小数点数型 (Double) (Visual Basic)

概念

Visual Basic におけるデータ型
型文字

その他の技術情報

基本データ型

方法：日付および時刻の値を変数に保持する

日付および時刻の値を **Date** のデータ型として宣言すると、それらの値が変数に保持されます。

日付型 (Date) (Visual Basic) の変数には、0001 年 1 月 1 日から 9999 年 12 月 31 日までの範囲の日付値、および 12:00:00 AM (深夜) から 11:59:59 PM までの範囲の時刻値を保持できます。

Date のデータ型には、日付値および時刻値の両方が常に保持されます。**Date** 変数に時刻値が割り当てられない場合、既定値である日の先頭の深夜に設定されます。**Date** 変数に日付値が割り当てられない場合、既定値である 0001 年 1 月 1 日に設定されます。

変数に日付を保持するには

1. 変数を Dim ステートメント (Visual Basic) で宣言します。
2. 変数名の後に **As** 句を指定します。
3. **As** キーワードの後に **Date** キーワードを指定します。

参照

関連項目

データ型の概要 (Visual Basic)
単精度浮動小数点型 (Single) (Visual Basic)
日付型 (Date) (Visual Basic)

概念

Visual Basic におけるデータ型
型文字

その他の技術情報

基本データ型

方法：型が不明なデータを変数に保持する

変数を **Object** データ型で宣言すると、その変数はデータへのポインタを保持します。

オブジェクト型 (Object) の変数は、任意のデータ型の値へのポインタを保持できます。これには、**Integer**、**Boolean**、および構造体のインスタンスなどの値型と、**String**、Form、配列インスタンスなどのクラスから作成されたオブジェクトのインスタンスである参照型の両方が含まれます。

パフォーマンス。Objectデータ型の利点は、任意のデータ型のデータをポイントできることです。欠点は、実行に時間を必要とする追加の演算が行われるためアプリケーションの動作が遅くなることです。値型に対して **Object** 変数を使用する場合は、ボックス化およびボックス化解除が必要です。参照型に対して使用する場合は、遅延バインディングが必要です。

変数に状況に応じてさまざまなデータ型を保持する必要がある場合や、変数に保持する (複数の) データ型がコンパイル時にわからない場合には、**Object** データ型を使用する必要があります。

型が不明のデータを変数に保持するには

1. 変数を Dim ステートメント (Visual Basic) で宣言します。
2. 変数名の後に **As** 句を指定します。
3. **As** キーワードの後に **Object** キーワードを指定します。

参照

関連項目

データ型の概要 (Visual Basic)
単精度浮動小数点型 (Single) (Visual Basic)
オブジェクト型 (Object)

概念

Visual Basic におけるデータ型
型文字
値型と参照型

その他の技術情報

基本データ型

複合データ型

Visual Basic に用意されている基本データ型以外にも、異なる型のアイテムを組み合わせて、構造体、配列、クラスなどの複合データ型を作成できます。基本型および他の複合型から複合データ型を作成できます。たとえば、構造体要素の配列や配列メンバを持つ構造体を定義できます。

データ型

複合型は、その構成要素のデータ型とは異なります。たとえば、整数型 (**Integer**) の要素の配列は整数型 (**Integer**) ではありません。

配列データ型は通常、要素の型、かっこ、および必要に応じてコンマを使用して表されます。たとえば、**String** 型の要素を持つ 1 次元配列は、**String()** として表され、**Boolean** 型の要素を持つ 2 次元配列は、**Boolean(,)** として表されます。

構造体型

すべての構造体を包括する 1 つのデータ型はありません。逆に、すべての構造体定義はそれぞれ異なるデータ型を表します。たとえば、まったく同じ要素が同じ順番で定義されている 2 つの構造体があっても、それらは同じデータ型と見なされません。ただし、同じ構造体の複数のインスタンスを作成した場合、Visual Basic はそれらを同じデータ型と見なします。

配列型

すべての配列を包括する 1 つのデータ型はありません。配列の特定のインスタンスのデータ型は、次の事項によって決まります。

- 配列であること。
- 配列のランク (次元数)。
- 配列の要素型。

特に、次元の長さはインスタンスのデータ型に含まれません。次に例を示します。

```
Dim arrayA( ) As Byte = New Byte(12) {}
Dim arrayB( ) As Byte = New Byte(100) {}
Dim arrayC( ) As Short = New Short(100) {}
Dim arrayD( , ) As Short
Dim arrayE( , ) As Short = New Short(4, 10) {}
```

この例では、配列変数 `arrayA` と `arrayB` は異なる長さに初期化されますが、データ型は同じ **Byte()** であると見なされます。変数 `arrayB` と `arrayC` は、要素型が異なるため同じデータ型ではありません。変数 `arrayC` と `arrayD` は、ランクが異なるため同じデータ型ではありません。変数 `arrayD` と変数 `arrayE` は、`arrayD` がまだ初期化されていませんが、ランクも要素型も同じなので、同じ **Short(,)** 型です。

配列の詳細については、「Visual Basic における配列」を参照してください。

クラス型

すべてのクラスを包括する 1 つのデータ型はありません。あるクラスが別のクラスを継承することはできますが、それらは別のデータ型です。同じクラスの複数のインスタンスは、同じデータ型です。あるクラス インスタンス変数を別のクラス インスタンス変数に代入すると、それらの変数はデータ型が同じであるだけでなく、メモリ内の同じクラス インスタンスを指すことになります。

クラスの詳細については、「クラスについて」を参照してください。

参照

処理手順

データ型のトラブルシューティング
方法 : 変数内で複数の値を保持する

概念

Visual Basic におけるデータ型
Visual Basic におけるジェネリック型
Visual Basic での型宣言を省略したプログラミング
.NET Framework 型のデータ型

その他の技術情報

基本データ型
データ型の実装
Visual Basic における型変換
構造体 : 独自のデータ型

方法：変数内で複数の値を保持する

変数を複合データ型として宣言すると、変数は複数の値を保持します。

複合データ型 には、構造体、配列、およびクラスが含まれます。複合データ型の変数は、基本データ型とその他の複合型を組み合わせて保持できます。構造体とクラスは、データだけでなくコードも保持できます。

変数内で複数の値を保持するには

1. 変数で使用する複合データ型を決定します。
2. 複合データ型がまだ定義されていない場合は、それを定義して変数で使用できるようにします。
   - Structure ステートメント を使用して構造体を宣言します。
   - Dim ステートメント (Visual Basic) を使用して配列を定義します。
   - Class ステートメント (Visual Basic) を使用してクラスを定義します。
3. **Dim** ステートメントを使用して変数を宣言します。
4. 変数名の後に **As** 句を指定します。
5. **As** キーワードの後ろに、適切な複合データ型の名前を指定します。

参照

関連項目

データ型の概要 (Visual Basic)

概念

型文字
複合データ型
.NET Framework 型のデータ型

その他の技術情報

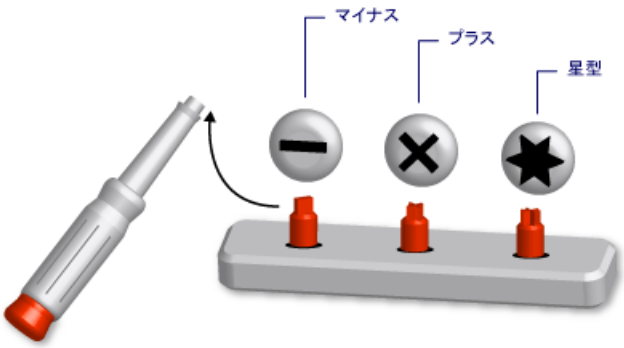
構造体 : 独自のデータ型
Visual Basic における配列
クラスについて

Visual Basic におけるジェネリック型

ジェネリック型はさまざまなデータ型に対して同じ機能を実行するために必要な処理を行う、1 つのプログラミング要素です。ジェネリック クラスまたはジェネリック プロシージャを定義すると、同じ機能を実行させる各データ型に対して、その機能を別々に定義する必要がありません。

これは、ヘッドの部分が交換可能な、ねじ回しのセットにたとえることができます。回すねじを調べて、そのねじに合った正しいヘッド (マイナス、プラス、星型) を選択します。ねじ回しのハンドルに正しいヘッドを挿入したら、ねじ回しを使ってまったく同じ作業 (ねじを回すこと) を行います。

汎用的な道具であるねじ回しのセット



ジェネリック型を定義するには、1 つ以上のデータ型でジェネリック型をパラメータ化します。これにより、ジェネリック型を使用するコードで、データ型をコードの要件に合わせて変更できるようになります。コードでは、1 つのジェネリックな要素から複数のプログラミング要素を宣言し、それぞれを異なるデータ型に使用できます。ただし、使用するデータ型が異なっていても、宣言した要素はどれも同じロジックを実行します。

たとえば、**String** などの特定のデータ型を操作するキュー クラスを作成し、使用する必要があるとします。次の例に示すように、このようなクラスは、System.Collections.Generic.Queue から宣言できます。

VB

```
Public stringQ As New System.Collections.Generic.Queue(Of String)
```

このときに、`stringQ` を使って、**String** 値だけを扱うように指定できます。`stringQ` は、**Object** 値を汎用的に扱うのではなく **String** だけを扱うことを意味するので、遅延バインディングまたは型変換は行いません。その結果、実行時間が短縮され、ランタイム エラーが減少します。

ジェネリック型の使い方の詳細については、「方法 : ジェネリック クラスを使用する」を参照してください。

ジェネリック クラスの例

次のコード例は、ジェネリック クラスのスケルトン定義を示しています。

VB

```
Public Class classHolder(Of t)
    Public Sub processNewItem(ByVal newItem As t)
        Dim tempItem As t
        ' Insert code that processes an item of data type t.
    End Sub
End Class
```

このスケルトンでは、`t` は型パラメータです。これはプレースホルダなので、このクラスを宣言するときには適切なデータ型で置き換えてください。コードの他の部分では、`t` をさまざまなデータ型で置き換えることにより、`classHolder` のさまざまなバージョンを宣言できます。このようにして宣言した 2 つのクラスを次の例に示します。

VB

```
Public integerClass As New classHolder(Of Integer)
Friend stringClass As New classHolder(Of String)
```

このステートメントでは、構成されるクラスを宣言し、その中で型パラメータを特定の型に置き換えています。この置き換えは、構成されるクラスのコード全体に反映されます。次の例は、`integerClass` の `processNewItem` プロシージャがどのようなコードになるかを示しています。

VB

```
Public Sub processNewItem(ByVal newItem As Integer)
    Dim tempItem As Integer
    ' Inserted code now processes an Integer item.
End Sub
```

この例より完全なコード例については、「方法 : 複数のデータ型に同一の機能を提供できるクラスを定義する」を参照してください。

ジェネリックに適したプログラミング要素

ジェネリックのクラス、構造体、インターフェイス、プロシージャ、およびデリゲートを宣言し、使用できます。.NET Framework では、よく使われるジェネリックな要素を表すジェネリックのクラス、構造体、およびインターフェイスが定義されています。System.Collections.Generic 名前空間には、ディクショナリ、リスト、キュー、およびスタックが用意されています。独自のジェネリックな要素を定義する前に、それに相当する要素が既に **System.Collections.Generic** に用意されていないかを確認してください。

プロシージャは型ではありませんが、ジェネリック プロシージャを宣言し、使用できます。Visual Basic におけるジェネリック プロシージャ を参照してください。

ジェネリック型の利点

ジェネリック型は、それぞれが特定のデータ型を操作する複数のプログラミング要素を宣言するための基礎となります。ジェネリック型の代わりになりうるものを以下に示します。

1. **Object** データ型を操作する単一の型。
2. 型の型固有バージョンのセット。それぞれの型は、個別にコーディングされ、**String**、**Integer**、または `customer` などのユーザー定義型などの特定のデータ型を操作します。

ジェネリック型には、これらの代替手段にはない次の利点があります。

- **タイプ セーフ。** ジェネリック型では、コンパイル時に型がチェックされます。一方、**Object** に基づく型はすべてのデータ型を受け入れるので、入力したデータ型が受け入れられる型かどうかをコードでチェックする必要があります。ジェネリック型を使うと、型の不一致は実行時ではなくコンパイル時に検出されます。
- **パフォーマンス。** ジェネリック型は、それぞれが特定の 1 つのデータ型に特化されるので、データをボックス化したり、ボックス化を解除したりする必要がありません。**Object** に基づいて操作を実行する場合、入力したデータ型をボックス化して **Object** に変換したり、出力時にデータのボックス化を解除したりする必要があります。ボックス化とボックス化解除は、パフォーマンスを低下させます。

  また、**Object** に基づく型は遅延バインディングです。つまり、この型のメンバにアクセスするには、実行時に余分なコードが必要になります。これも、パフォーマンスを低下させます。
- **コードの統合。** ジェネリック型のコードは、一度だけ定義します。1 つの型の一連の型固有バージョンでは、同じコードが各バージョンに複製されます。バージョンによって異なるのは、扱うデータ型だけです。ジェネリック型では、すべての型固有バージョンが元のジェネリック型から生成されます。
- **コードの再利用。** 特定のデータ型に依存しないコードは、ジェネリックである場合に、さまざまなデータ型で再利用できます。予期しなかったデータ型に再利用できることもあります。
- **IDE サポート。** ジェネリック型から宣言した構成型を使用すると、コードの開発中に統合開発環境 (IDE: Integrated Development Environment) から提供されるサポートが増えます。たとえば、IntelliSense™ は、コンストラクタまたはメソッドの引数に使用できる型固有のオプションを提示できます。
- **ジェネリックなアルゴリズム。** 型に依存しない抽象アルゴリズムは、ジェネリック型にすることをお勧めします。たとえば、IComparable インターフェイスを使って項目を並べ替えるジェネリック プロシージャは、**IComparable** を実装する任意のデータ型に使用できます。

制約

ジェネリック型定義のコードはできる限り型に依存しない必要がありますが、なんらかのデータ型の機能がジェネリック型に必要な場合もあります。たとえば、並べ替えや照合順序のために 2 つの項目を比較する必要がある場合、それらのデータ型は **IComparable** インターフェイスを実装する必要があります。この要件を強制するには、制約を型パラメータに追加します。

制約の例

次のコード例は、**IComparable** の実装を型引数に強制する制約があるクラスのスケルトン定義を示しています。

VB

```
Public Class itemManager(Of t As IComparable)
    ' Insert code that defines class members.
End Class
```

後続のコードで、**IComparable** を実装しない型を渡して `itemManager` からクラスを作成しようとすると、コンパイラがエラーを生成します。

制約の種類

次の要件を任意に組み合わせて制約を指定できます。

- 型引数は、1 つまたは複数のインターフェイスを実装する。
- 型引数は、クラスの型そのものであるか、最大 1 つのクラスを継承する。
- 型引数はパラメータなしのコンストラクタを公開し、そのコンストラクタからオブジェクトを作成するコードで使用できるようにする。
- 型引数は、常に参照型である。または、常に値型である。

複数の要件を指定する場合は、コンマで区切られた制約リストを中かっこ ({ }) で囲みます。アクセス可能なコンストラクタの作成を必須とするには、New (Visual Basic) キーワードをリストに追加します。参照型であることを必須とするには、Class (Visual Basic) キーワードを追加し、値型であることを必須とするには、Structure (Visual Basic) キーワードを追加します。

制約の詳細については、「型リスト」を参照してください。

複数の制約の例

次のコード例は、型パラメータに制約リストがあるジェネリック クラスのスケルトン定義を示しています。このクラスのインスタンスを作成するコードでは、型引数が **IComparable** インターフェイスと IDisposable インターフェイスの両方を実装し、参照型であり、アクセス可能なパラメータなしのコンストラクタを公開する必要があります。

VB

```
Public Class thisClass(Of t As {IComparable, IDisposable, Class, New})
    ' Insert code that defines class members.
End Class
```

重要な用語

ジェネリック型に関連して、以下の新しい用語が使用されます。

- ジェネリック型。宣言時に少なくとも 1 つのデータ型を指定するクラス、構造体、インターフェイス、プロシージャ、またはデリゲートの定義です。
- 型パラメータ。ジェネリック型定義で、データ型を宣言するときにデータ型の代わりに指定するプレースホルダです。
- 型引数。構成型をジェネリック型から宣言するときに、型パラメータを置き換える特定のデータ型です。
- 制約。型パラメータに指定できる型引数を制限する条件です。型引数が特定のインターフェイスを実装すること、特定のクラスであるか特定のクラスを継承すること、アクセス可能なパラメータなしのコンストラクタを持つこと、または参照型または値型であることを強制できます。このような制約は組み合わせて指定できますが、指定できるのは 1 つのクラスだけです。
- 構成型。型パラメータに型引数を指定することにより、ジェネリック型から宣言されるクラス、構造体、インターフェイス、プロシージャ、またはデリゲートです。

参照

処理手順

データ型のトラブルシューティング

関連項目

データ型の概要 (Visual Basic)
Of
As
オブジェクト型 (Object)

概念

Visual Basic におけるデータ型
型文字
値型と参照型

その他の技術情報

方法：複数のデータ型に同一の機能を提供できるクラスを定義する

複数のデータ型に同一の機能を提供するオブジェクトを作成するために使用できるクラスを定義できます。これを行うには、1 つ以上の型パラメータを定義内で指定します。このようなクラスは、さまざまなデータ型を使用するオブジェクトのテンプレートとして使用できます。この方法で定義したクラスは、ジェネリック クラスと呼ばれます。

ジェネリック クラスを定義すると、クラスを一度だけ定義すれば、コードの中でそれを使って、さまざまなデータ型を使用する多くのオブジェクトを作成できるようになります。それにより、クラスを **Object** 型で定義した場合よりパフォーマンスが向上します。

クラスに加え、ジェネリックの構造体、インターフェイス、プロシージャ、およびデリゲートを定義および使用できます。

型パラメータを使ってクラスを定義するには

1. 通常の方法でクラスを定義します。
2. クラス名の直後に **(Of** *typeparameter***)** を追加し、型パラメータを指定します。
3. 複数の型パラメータがある場合は、コンマで区切られたリストを作成し、かっこで囲みます。**Of** キーワードは繰り返さないでください。
4. 型パラメータに対して単純な代入以外の操作を実行する場合は、その型パラメータの後に **As** 句を付けて 1 つ以上の制約を追加します。制約は、その型パラメータに渡される型が以下のような要件を満たすことを保証します。

   - コードで実行する > などの操作をサポートする
   - コードでアクセスするメソッドなどのメンバをサポートする
   - パラメータなしのコンストラクタを公開する

   制約を指定しない場合、コードで使用できる操作とメンバは、オブジェクト型 (Object) でサポートされるものだけです。詳細については、「型リスト」を参照してください。
5. 渡された型を使って宣言する各クラス メンバを識別し、それを **As** *typeparameter* として宣言します。これは、内部ストレージ、プロシージャ パラメータ、および戻り値にも当てはまります。
6. コードでは、`itemType` に渡される可能性があるデータ型でサポートされる操作とメソッドだけを使用します。

   次のコード例は、かなり単純なリストを管理するクラスを定義しています。このクラスは、リストを内部配列 `items` に格納します。このクラスを使用するコードでは、このリストの要素のデータ型を宣言できます。パラメータ化されたコンストラクタを使うと、コードで `items` の上限を設定できます。既定のコンストラクタでは、この上限は 9 (合計で 10 アイテムの場合) に設定されます。

   VB

```
Public Class simpleList(Of itemType)
  Private items() As itemType
  Private top As Integer
  Private nextp As Integer
  Public Sub New()
    Me.New(9)
  End Sub
  Public Sub New(ByVal t As Integer)
    MyBase.New()
    items = New itemType(t) {}
    top = t
    nextp = 0
  End Sub
  Public Sub add(ByVal i As itemType)
    insert(i, nextp)
  End Sub
  Public Sub insert(ByVal i As itemType, ByVal p As Integer)
    If p > nextp OrElse p < 0 Then
      Throw New System.ArgumentOutOfRangeException("p", _
        " less than 0 or beyond next available list position")
    ElseIf nextp > top Then
```

```
        Throw New System.ArgumentException("No room to insert at ", _
          "p")
      ElseIf p < nextp Then
        For j As Integer = nextp To p + 1 Step -1
          items(j) = items(j - 1)
        Next j
      End If
      items(p) = i
      nextp += 1
    End Sub
    Public Sub remove(ByVal p As Integer)
      If p >= nextp OrElse p < 0 Then
          Throw New System.ArgumentOutOfRangeException("p", _
              " less than 0 or beyond last list item")
      ElseIf nextp = 0 Then
          Throw New System.ArgumentException("List empty; cannot remove ", _
              "p")
      ElseIf p < nextp - 1 Then
          For j As Integer = p To nextp - 2
              items(j) = items(j + 1)
          Next j
      End If
      nextp -= 1
    End Sub
    Public ReadOnly Property listLength() As Integer
      Get
        Return nextp
      End Get
    End Property
    Public ReadOnly Property listItem(ByVal p As Integer) As itemType
      Get
        If p >= nextp OrElse p < 0 Then
          Throw New System.ArgumentOutOfRangeException("p", _
            " less than 0 or beyond last list item")
          End If
        Return items(p)
      End Get
    End Property
  End Class
```

`simpleList` からは、**Integer** 値のリストを格納するクラス、**String** 値のリストを格納するクラス、および **Date** 値のリストを格納するクラスを宣言できます。リスト メンバのデータ型が異なるだけで、これらのクラスから作成されるオブジェクトの動作はまったく同じです。

コードから `itemType` に渡される型引数として、**Boolean** または **Double**、構造体、列挙、任意の型のクラス (アプリケーションで独自に定義したクラスを含む) などの組み込みの型を使用できます。

`simpleList` クラスをテストするためには、次のコードを使用してください。

VB

```
  Public Sub useSimpleList()
    Dim iList As New simpleList(Of Integer)(2)
    Dim sList As New simpleList(Of String)(3)
    Dim dList As New simpleList(Of Date)(2)
    iList.add(10)
    iList.add(20)
    iList.add(30)
    sList.add("First")
    sList.add("extra")
```

```
    sList.add("Second")
    sList.add("Third")
    sList.remove(1)
    dList.add(#1/1/2003#)
    dList.add(#3/3/2003#)
    dList.insert(#2/2/2003#, 1)
    Dim s As String = _
      "Simple list of 3 Integer items (reported length " _
      & CStr(iList.listLength) & "):" _
      & vbCrLf & CStr(iList.listItem(0)) _
      & vbCrLf & CStr(iList.listItem(1)) _
      & vbCrLf & CStr(iList.listItem(2)) _
      & vbCrLf _
      & "Simple list of 4 - 1 String items (reported length " _
      & CStr(sList.listLength) & "):" _
      & vbCrLf & CStr(sList.listItem(0)) _
      & vbCrLf & CStr(sList.listItem(1)) _
      & vbCrLf & CStr(sList.listItem(2)) _
      & vbCrLf _
      & "Simple list of 2 + 1 Date items (reported length " _
      & CStr(dList.listLength) & "):" _
      & vbCrLf & CStr(dList.listItem(0)) _
      & vbCrLf & CStr(dList.listItem(1)) _
      & vbCrLf & CStr(dList.listItem(2))
    MsgBox(s)
  End Sub
```

参照

処理手順

方法 : ジェネリック クラスを使用する

関連項目

Of

型リスト

オブジェクト型 (Object)

概念

Visual Basic におけるデータ型

Visual Basic におけるジェネリック型

共通言語仕様

方法：ジェネリック クラスを使用する

型パラメータを受け取るクラスをジェネリック クラスと呼びます。ジェネリック クラスを使用する場合、各型パラメータに型引数を指定することによって、ジェネリック クラスから構成されるクラスを生成できます。そして、構成されるクラスの型の変数を宣言し、構成されるクラスのインスタンスを作成して、その変数を割り当てることができます。

クラスの他に、ジェネリックな構造体、インターフェイス、プロシージャ、およびデリゲートも定義して使用できます。

次のプロシージャは .NET Framework で定義されたジェネリック クラスを受け取り、そこからインスタンスを作成します。

型パラメータを受け取るクラスを使用するには

1. ソース ファイルの先頭に Imports ステートメント をインクルードして、System.Collections.Generic 名前空間をインポートします。これにより、System.Collections.Queue などのキュー クラスと区別するために完全に修飾しなくても、System.Collections.Generic.Queue クラスを参照できます。

2. 通常の方法でオブジェクトを作成します。ただし、クラス名のすぐ後に **(Of** *type***)** を指定する必要があります。

   次の例では、同じ (**System.Collections.Generic.Queue**) クラスを使用して、異なるデータ型の項目を格納する 2 つのキュー オブジェクトを作成します。各キューの末尾に項目が追加された後、各キューの頭から項目が削除され、表示されます。

   VB

```
Public Sub usequeue()
  Dim queueDouble As New System.Collections.Generic.Queue(Of Double)
  Dim queueString As New System.Collections.Generic.Queue(Of String)
  queueDouble.Enqueue(1.1)
  queueDouble.Enqueue(2.2)
  queueDouble.Enqueue(3.3)
  queueDouble.Enqueue(4.4)
  queueString.Enqueue("First string of three")
  queueString.Enqueue("Second string of three")
  queueString.Enqueue("Third string of three")
  Dim s As String = "Queue of Double items (reported length " _
    & CStr(queueDouble.Count) & "):"
  For i As Integer = 1 To queueDouble.Count
    s &= vbCrLf & CStr(queueDouble.Dequeue())
  Next i
  s &= vbCrLf & "Queue of String items (reported length " _
    & CStr(queueString.Count) & "):"
  For i As Integer = 1 To queueString.Count
    s &= vbCrLf & queueString.Dequeue()
  Next i
  MsgBox(s)
End Sub
```

参照

処理手順

方法 : 複数のデータ型に同一の機能を提供できるクラスを定義する

関連項目

Of

Imports ステートメント

概念

Visual Basic におけるデータ型

Visual Basic におけるジェネリック型

共通言語仕様

Visual Basic におけるジェネリック プロシージャ

ジェネリック プロシージャは、少なくとも 1 つの型パラメータで定義されるプロシージャであり、ジェネリック メソッドとも呼ばれます。これを使用すると、呼び出し元のコードは、プロシージャを呼び出すたびにその要件に合わせてデータ型を調整できます。

プロシージャは、ジェネリック クラスまたはジェネリックな構造体内で定義されていることだけでジェネリックになるわけではありません。ジェネリックであるためには、プロシージャは、必要に応じて使用される通常のパラメータ以外に、少なくとも 1 つの型パラメータを使用する必要があります。ジェネリック クラスまたはジェネリックな構造体が、非ジェネリック プロシージャを含むことも、非ジェネリックなクラス、構造体、またはモジュールが、ジェネリック プロシージャを含むこともできます。

ジェネリック プロシージャは、その型パラメータを、その通常のパラメータ リスト、その戻り値の型があればその型、およびそのプロシージャ コードで使用できます。

型の推定

型引数を一切指定しないでジェネリック プロシージャを呼び出すことができます。この方法で呼び出す場合、コンパイラは、プロシージャの型引数に渡す適切なデータ型を決定しようとします。これは、型の推定と呼ばれます。コンパイラが **String** 型を型パラメータ `t` に渡すと推定する呼び出しを次のコードに示します。

VB

```
Public Sub testSub(Of t)(ByVal arg As t)
End Sub
Public Sub callTestSub()
    testSub("Use this string")
End Sub
```

コンパイラが呼び出しのコンテキストから型引数を推定できない場合、エラーが報告されます。このようなエラーの原因の 1 つは、配列ランクの不一致です。たとえば、通常のパラメータを型パラメータの配列として定義するとします。異なるランク (次元数) の配列を指定するジェネリック プロシージャを呼び出すと、不一致により型の推定が失敗します。1 次元配列を受け取るプロシージャに 2 次元配列が渡される呼び出しを次のコードに示します。

```
Public Sub demoSub(Of t)(ByVal arg() As t)

End Sub

Public Sub callDemoSub()

Dim twoDimensions(,) As Integer

demoSub(twoDimensions)

End Sub
```

型の推定は、すべての型引数を省略することによってのみ呼び出すことができます。型引数を 1 つでも指定する場合は、すべての型引数を指定する必要があります。

型の推定がサポートされるのは、ジェネリック プロシージャの場合だけです。型の推定を、ジェネリック クラス、構造体、インターフェイス、またはデリゲートで呼び出すことはできません。

例

次の例では、配列内の特定の要素を検索するためのジェネリック **Function** プロシージャを定義しています。これは、1 つの型パラメータを定義し、それを使用して、パラメータ リストに 2 つのパラメータを構築します。

VB

```
Public Function findElement(Of T As IComparable) _
    (ByVal searchArray As T(), ByVal searchValue As T) As Integer
    If searchArray.GetLength(0) > 0 Then
        For i As Integer = 0 To searchArray.GetUpperBound(0)
            If searchArray(i).CompareTo(searchValue) = 0 Then Return i
        Next i
    End If
    Return -1
End Function
```

前の例では、searchValue と searchArray の各要素を比較する機能が必要です。この機能が確実に動作するように、型パラメータ T は IComparable インターフェイスを実装するように制限されています。コードでは、= 演算子の代わりに CompareTo メソッドを使用します。これは、T によって提供される型引数が = 演算子をサポートしているという保証がないからです。

findElement プロシージャをテストするためには、次のコードを使用してください。

VB

```
Public Sub tryFindElement()
    Dim stringArray() As String = {"abc", "def", "xyz"}
    Dim stringSearch As String = "abc"
    Dim integerArray() As Integer = {7, 8, 9}
    Dim integerSearch As Integer = 8
    Dim dateArray() As Date = {#4/17/1969#, #9/20/1998#, #5/31/2004#}
    Dim dateSearch As Date = Microsoft.VisualBasic.DateAndTime.Today
    MsgBox(CStr(findElement(Of String)(stringArray, stringSearch)))
    MsgBox(CStr(findElement(Of Integer)(integerArray, integerSearch)))
    MsgBox(CStr(findElement(Of Date)(dateArray, dateSearch)))
End Sub
```

前の **MsgBox** 呼び出しでは、それぞれ "0"、"1"、および "-1" が表示されます。

参照

処理手順

方法 : 複数のデータ型に同一の機能を提供できるクラスを定義する

方法 : ジェネリック クラスを使用する

関連項目

型リスト

パラメータの一覧

概念

Visual Basic におけるジェネリック型

Visual Basic におけるプロシージャ

プロシージャのパラメータと引数

定義された値を持たない可能性のある値型

特定の状況においては、定義された値を持たない値型を扱うことがあります。たとえば、データベース内のフィールドには、意味のある値が代入されている場合と、値がまったく代入されていない場合とがあり、これを区別する必要があります。値型を拡張して、通常の値または null 値を受け入れるようにすることができます。このような拡張は、null 許容型と呼ばれます。

各 null 許容型は、ジェネリックな Nullable 構造体から作成されます。データベースを使って、仕事に関連する活動を記録すると仮定します。次のコード例は、null 許容 **Boolean** データ型を作成し、この型の変数を宣言する方法を示しています。

```
Dim ridesBusToWork As Nullable(Of Boolean)
```

変数 `ridesBusToWork` には、**True** または **False** の値を格納できるほか、まったく値が格納されない状態も許されます。初期の既定値は、まったく値のない状態です。今回の例では、これはその人物についてまだ情報が取得されていないことを意味します。これとは対照的に、**False** は、その人物に関する情報が取得され、仕事のバスに乗務していないことを示します。

null 許容型で変数やプロパティを宣言したり、null 許容型の要素で配列を宣言したりすることができます。null 許容型をパラメータとしてプロシージャを宣言することや、null 許容型を戻り値として **Function** プロシージャから返すこともできます。

null 許容型を配列、**String**、クラスなどの参照型で作成することはできません。基になる型は、値型である必要があります。詳細については、「値型と参照型」を参照してください。

null 許容型変数の使用

null 許容型の最も重要なメンバは、HasValue プロパティと Value プロパティです。null 許容型の変数に定義済みの値が格納されているかどうかは、**HasValue** を使って確認できます。**HasValue** が **True** の場合、値を **Value** から読み取ることができます。**HasValue** と **Value** は両方とも **ReadOnly** のプロパティであることに注意してください。

既定値

null 許容型の変数を宣言すると、その変数の **HasValue** プロパティは、既定で **False** 値に設定されます。つまり、基になる値型の既定値が変数に格納されるのではなく、定義済みの値が格納されていないのが既定の状態です。次のコード例では、**Integer** 型の既定値は 0 ですが、変数 `numberOfChildren` に定義済みの初期値は格納されません。

```
Dim numberOfChildren As Nullable(Of Integer)
```

null 値は、未定義または未知の値を示すには便利です。`numberOfChildren` を **Integer** として宣言した場合、情報が現在使用できないことを示すために使用できる値はありません。

値の格納

null 許容型の変数またはプロパティに値を格納する方法は、通常と変わりません。次のコード例は、前掲の例で宣言した変数 `numberOfChildren` に値を代入しています。

```
numberOfChildren = 2
```

null 許容型の変数またはプロパティに定義済みの値が格納されている場合でも、値が代入されていない初期状態に戻すことができます。これを行うには、変数またはプロパティを **Nothing** に設定します。以下に例を示します。

```
numberOfChildren = Nothing
```

メモ：

Nothing は null 許容型の変数に代入できますが、変数の値が **Nothing** であるかどうかをテストすることはできません。**HasValue** プロパティが **False** かどうかをテストする必要があります。

値の取得

null 許容型の変数から値を取得するには、その変数の **HasValue** プロパティを使って、変数に値が含まれているかどうかを最初にテストする必要があります。**HasValue** が **False** の場合に変数の値を読み取ろうとすると、Visual Basic から InvalidOperationException 例外がスローされます。次のコード例は、前掲の例で使った変数 `numberOfChildren` から値を読み取るための推奨される方法を示しています。

```
If numberOfChildren.HasValue Then
    MsgBox("There are " & CStr(numberOfChildren) & " children.")
Else
    MsgBox("It is not known how many children there are.")
End If
```

データのある null 許容型の使用

データベースは、null 許容型が使用される最も重要な場所です。現在すべてのデータベース オブジェクトで null 許容型がサポートされているわけではありませんが、デザイナ生成テーブル アダプタではサポートされています。詳細については、「TableAdapter の概要」の「TableAdapter での Null 許容型のサポート」を参照してください。

参照

処理手順

データ型のトラブルシューティング

関連項目

InvalidOperationException

概念

Visual Basic におけるデータ型

値型と参照型

TableAdapter の概要

その他の技術情報

データ型の実装

データ型の実装

Visual Basic のデータ型は、その型の変数にデータそのものが格納されるのか、それともデータへのポインタが格納されるのかによって分類されます。データそのものが格納される場合は値型、メモリのどこか別の場所にあるデータへのポインタを格納する場合は参照型になります。この分類は、データ型の実装方法に影響します。

このセクションの内容

値型と参照型

値型と参照型の違いとデータ型との対応について説明します。

.NET Framework 型のデータ型

Visual Basic のデータ型が、その基盤である .NET Framework でどのように実装されるかについて説明します。

関連するセクション

Visual Basic におけるデータ型

Visual Basic データ型を示し、使用方法について説明します。

データ型の概要 (Visual Basic)

Visual Basic で用意されている基本データ型を一覧で紹介します。

値型と参照型

データ型が自身のメモリ内にデータを保持する場合、そのデータ型は値型です。参照型には、データを保持する別のメモリ位置へのポインタが格納されます。

値型

値型には、次のようなものがあります。

- すべての数値データ型
- **Boolean**、**Char**、および **Date**
- すべての構造体 (メンバが参照型の場合でも)
- 列挙型 (基になる型が常に **SByte**、**Short**、**Integer**、**Long**、**Byte**、**UShort**、**UInteger**、または **ULong** であるため)

参照型

参照型には、次のようなものがあります。

- **String**
- すべての配列 (要素が値型の場合でも)
- クラス型 (Form など)
- デリゲート

型ではない要素

次のプログラミング要素は、宣言された要素のデータ型として指定できないため、型として修飾しません。

- 名前空間
- モジュール
- イベント
- プロパティおよびプロシージャ
- 変数、定数およびフィールド

オブジェクトのデータ型の操作

オブジェクト型 (Object) の変数には参照型と値型のどちらでも代入できます。**オブジェクト型 (Object)** の変数は常にデータそのものではなくデータへのポインタを保持します。ただし、**オブジェクト型 (Object)** の変数に値型を代入すると、独自のデータを保持しているように動作します。詳細については、「オブジェクト型 (Object)」を参照してください。

Object 変数を、Microsoft.VisualBasic 名前空間の Information クラス上にある IsReference メソッドに渡すことにより、参照型または値型のどちらとして動作しているかどうかを調べることができます。**Object** 変数の内容が参照型を表している場合、**Microsoft.VisualBasic.Information.IsReference(System.Object)** は **True** を返します。

参照

関連項目

オブジェクト型 (Object)

概念

.NET Framework 型のデータ型

データ型の有効な使用方法

その他の技術情報

Visual Basic における型変換

構造体 : 独自のデータ型

.NET Framework 型のデータ型

Visual Basic のすべての基本データ型は、System 名前空間の構造体またはクラスによってサポートされています。コンパイラでは、基になる構造体またはクラスに対して各データ型のキーワードをエイリアスとして使用します。たとえば、予約語 **Byte** を使用する変数の宣言は、完全修飾構造体名 System.Byte を使用する宣言と同じです。

.NET Framework で利用できる、追加のクラスおよび構造体

共通言語ランタイム (CLR: Common Language Runtime) では、Visual Basic に用意されていない構造体とクラスもサポートされます。たとえば、System.Guid 構造体には、グローバル一意識別子 (GUID: globally unique identifier) が用意されています。また、System.TimeZone クラスでは、タイム ゾーンがサポートされています。これらの型を使用して変数および定数を宣言したり、.NET Framework によってこれらの型に実装されるメソッドにアクセスできます。ただし、Visual Basic に用意されていない型に関連する演算や型変換はサポートされません。

値型と参照型

.NET Framework では、構造体は値型であり、クラスは参照型です。このため、**Char** および **Integer** などの値型は、.NET Framework 構造体によって実装されます。また、**Object** および **String** などの参照型は、.NET Framework クラスによってサポートされます。メンバが値型であっても、すべての配列は参照型です。また、参照型メンバを持っていたとしても、すべての構造体は値型です。

すべての参照型は基になる .NET Framework のクラスを表すため、初期化する場合には New (Visual Basic) キーワードを使用する必要があります。次のステートメントで配列を初期化します。

```
Dim totals() As Single = New Single(8) {}
```

値型を初期化するときにも **New** キーワードを使用できます。これは、その型にパラメータをとるコンストラクタがある場合に特に有効です。これに関する例は、指定した部分から新しい **Decimal** 値を作成する、Decimal コンストラクタです。

メンバを持つデータ型

これらは、.NET Framework 構造体およびクラスによってサポートされているため、Visual Basic データ型にはメンバがあります。これらのメンバには、コンストラクタ、メソッド、プロパティ、およびフィールドがあります。変数のメンバ (コンストラクタを除く) にアクセスするときは、オブジェクトのメソッドやプロパティにアクセスするときと同じ方法を使用できます。

次の例では、現在の月に残っている日数を決定するために、Year プロパティ、Month プロパティ、および Day プロパティ、および System.DateTime 構造体の DaysInMonth メソッドが使用されています。

```
Dim current As Date = Now
Dim daysRemaining As Integer
daysRemaining = Date.DaysInMonth(current.Year, current.Month) - current.Day
```

データ型のメンバへの参照は、型の名前 (`Date`) またはその型に宣言された変数の名前 (`current`) で修飾する必要があります。

データ型メンバの例

次のコード プロトタイプでは、データ型で有効なメソッド、プロパティ、およびフィールドの一部を示します。

```
<Char>.IsDigit() ' Returns True if character is a numeric digit.

<Char>.IsLower() ' Returns True if character is a lowercase letter.

<Date>.IsLeapYear() ' Returns True if current year is a leap year.

<Date>.ToUniversalTime() ' Returns local date/time converted to UTC.

<Double>.IsInfinity() ' Returns True if contents represent infinity.

<Double>.IsNaN() ' Returns True if contents are not a number (0/0).

<Long>.MaxValue ' Constant representing largest positive Int64 value.

<Object>.GetType() ' Returns Type object representing type of <Object>.

<Object>.GetType().GetTypeCode() ' Returns type code of <Object>.

<String>.Chars(<index>) ' Character at position <index> of <String>.

<String>.Length ' Number of characters currently in <String>.
```

Byte および **Char** を含むすべての数値型は、MaxValue および MinValue の各パブリック フィールドを公開します。これらのフィールドは、数値型を扱うときに便利です。

保証されていない、データ型メンバの同等物

.NET Framework には、Visual Basic の関数やキーワードと同じような、データ型に対する多数のメソッドも用意されています。しかし、Visual Basic では、変換や他の操作を実行するために .NET Framework メソッドが常に使用されるとは限りません。また、結果が常に同じであるとは限りません。

たとえば、ToSingle メソッドでは、**CSng** キーワードが **Decimal** 式に対して行うアクションと、同じ型のアクションが実行されます。しかし、**CSng** が、**System.**<*data type*>**.ToSingle** を使用するとは限らないため、微妙な条件の下では、結果が同じになる保証はありません。

一般的には、Visual Basic プログラミング要素のほうが使いやすく、コードがより読みやすくなるため、こちらを使用する必要があります。.NET Framework メソッドによって用意されている、いくつかの追加の機能が必要な場合もあります。この例については、「データ型のトラブルシューティング」の「Mod 演算子が正確な結果を返さない」を参照してください。

参照

関連項目

Boolean
Byte
Char
DateTime
Decimal
Double
Guid
TimeZone

概念

値型と参照型
構造体とクラス

Visual Basic における型変換

値のデータ型を変更するプロセスは変換と呼ばれます。変換の種類は、関連する型のデータ容量により、拡張と縮小のいずれかになります。また、ソース コードの構文に応じて、暗黙または明示的に行われます。

このセクションの内容

拡大変換と縮小変換

変換先の型がデータを保持できるかどうかによって分類される変換について説明します。

暗黙の型変換と明示的な型変換

Visual Basic で自動的に実行できるかどうかによって分類される変換について説明します。

変換時の値の変化

データ型の変換によって値やデータ表現がどのように変わるかについて説明します。

文字列とその他の型との変換

文字列と、数値、ブール型 (**Boolean**)、日時の値との変換について説明します。

方法 : Visual Basic でオブジェクトを別の型に変換する

Object 型の変数を他のデータ型に変換する方法について説明します。

配列の変換

配列を異なるデータ型の配列に変換するプロセスの詳細を説明します。

関連するセクション

Visual Basic におけるデータ型

Visual Basic データ型を示し、使用方法について説明します。

データ型の概要 (Visual Basic)

Visual Basic で用意されている基本データ型を一覧で紹介します。

データ型のトラブルシューティング

データ型を扱う際に多く見られる問題についていくつか説明します。

拡大変換と縮小変換

型変換で考慮しなければならない重要な点は、変換結果が変換先データ型の範囲内に入るかどうかということです。拡大変換では、変換元データのすべての値を含むことができるデータ型に値が変更されます。縮小変換では、値の一部を保持できない可能性があるデータ型に変更されます。

拡大変換

次の表は、標準の拡大変換を示しています。

データ型	拡大変換後のデータ型 [1]
SByte	**SByte**, **Short**, **Integer**, **Long**, **Decimal**, **Single**, **Double**
バイト	**Byte**, **Short**, **UShort**, **Integer**, **UInteger**, **Long**, **ULong**, **Decimal**, **Single**, **Double**
短整数型 (Short)	**Short**, **Integer**, **Long**, **Decimal**, **Single**, **Double**
ushort	**UShort**, **Integer**, **UInteger**, **Long**, **ULong**, **Decimal**, **Single**, **Double**
整数	**Integer**, **Long**, **Decimal**, **Single**, **Double**[2]
UInteger	**UInteger**, **Long**, **ULong**, **Decimal**, **Single**, **Double** [2]
長整数型 (Long)	**Long**, **Decimal**, **Single**, **Double** [2]
ulong	**ULong**, **Decimal**, **Single**, **Double** [2]
10 進数	**Decimal**, **Single**, **Double** [2]
単精度浮動小数点数型	**Single**, **Double**
倍精度浮動小数点数型	**Double**
すべての列挙型 (Enum)	基になる整数型および拡大変換の基になる任意の型
文字型 (Char)	**Char**, **String**
Char 配列	**Char** 配列、**String**
任意の型	オブジェクト
すべての派生型	[3] を派生するすべての基本型
任意の型	実装するすべてのインターフェイス
なし	すべてのデータ型またはオブジェクト型

[1] 定義上、すべてのデータ型は自動的に拡大されます。

[2] **Integer**、**UInteger**、**Long**、**ULong**、または **Decimal** から **Single** または **Double** への変換は、精度が失われる結果となる可能性がありますが、桁数が失われることはありません。この意味で考えると、情報の損失は起こりません。

[3] 派生型からその基本型への変換は拡大変換になります。派生型には基本型のすべてのメンバが含まれるため、基本型のインスタンスとして扱うことができるためです。逆に、基本型では派生型で定義されたメンバが含まれていません。

拡大変換は、実行時には常に正常に行われ、データ消失が発生することはありません。Option Strict ステートメントで型チェック スイッチを **On** または **Off** に設定して、拡大変換を常に暗黙的に実行できます。

縮小変換

標準の縮小変換は次のとおりです。

- 前の表で示した拡大変換の逆方向 (それ自体に拡大変換する場合の型を除く)
- ブール型 (Boolean) と任意の数値型の間の相互変換
- 任意の数値型から任意の列挙型への変換 (**Enum**)
- 文字列型と、任意の数値型、**Boolean**、または日付型の間の相互変換
- データ型またはオブジェクト型の、その派生型への変換

縮小変換は、実行時に常に正しく実行されるとは限りません。したがって失敗またはデータを消失する可能性があります。変換先のデータ型で変換対象の値を受け入れることができない場合に、エラーが発生します。たとえば、数値変換ではオーバーフローが発生することがあります。コンパイラは、Option Strict ステートメントで型チェック スイッチが **Off** に設定されていない限り、暗黙的な縮小変換の実行を許可しません。

縮小変換を使用するタイミング

変換元の値を変換先のデータ型に変換でき、エラーまたはデータの消失が発生しないことがわかっている場合に、縮小変換を使用します。たとえば、**文字列型 (String)** に "True" または "False" のいずれかが含まれることが明らかな場合は、**CBool** キーワードを使用してブール型 (**Boolean)** に変換できます。

変換中の例外

拡大変換は常に正しく実行されるため、例外をスローすることはありません。縮小変換が失敗すると、通常は次の例外をスローします。

- InvalidCastException — 2 つの型の間で変換が定義されていない場合
- OverflowException — (整数型の場合のみ) 変換された値が対象の型にしては大きすぎる場合

クラスまたは構造体で CType 関数を定義して、そのクラスまたは構造体の間の変換演算子として機能させると、その **CType** は適切と見なされる例外をスローできます。さらに、その **CType** は Visual Basic 関数または .NET Framework メソッドを呼び出して、さまざまな例外を順番にスローできます。

参照

処理手順

方法 : Visual Basic でオブジェクトを別の型に変換する

関連項目

データ型の概要 (Visual Basic)
データ型変換関数

概念

Visual Basic におけるデータ型
暗黙の型変換と明示的な型変換
変換時の値の変化
文字列とその他の型との変換
配列の変換
Visual Basic での型宣言を省略したプログラミング

その他の技術情報

Visual Basic における型変換

暗黙の型変換と明示的な型変換

暗黙の型変換では、ソース コードに特殊な構文は不要です。次の例では、Visual Basic は、k の値を暗黙的に単精度の浮動小数点値に変換してから q に代入しています。

```
Dim k As Integer
Dim q As Double
' Integer widens to Double, so you can do this with Option Strict On.
k = 432
q = K
```

明示的な型変換では、型変換キーワードを使用します。Visual Basic には、このようなキーワードが複数用意されており、かっこ内に指定した式を目的のデータ型に変換できます。これらのキーワードは関数のように動作しますが、コンパイラによってコード インラインが生成されるため、実行速度は関数呼び出しよりも多少速くなります。

次に示すのは上の例の続きですが、**CInt** キーワードによって q の値を整数に変換してから、k に代入しています。

```
' q had been assigned the value 432 from k.
q = Math.Sqrt(q)
k = CInt(q)
' k now has the value 21 (rounded square root of 432).
```

型変換のキーワード

次の表は、使用できる変換キーワードを示しています。

型変換キーワード	変換後のデータ型	変換できる式のデータ型
CBool	ブール型 (Boolean) (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**String**、**Object**
CByte	バイト型 (Byte) (Visual Basic)	すべての数値型 (**SByte** および列挙型を含む)、**Boolean**、**String**、**Object**
CChar	文字型 (Char) (Visual Basic)	**String**, **Object**
CDate	日付型 (Date) (Visual Basic)	**String**, **Object**
CDbl	倍精度浮動小数点数型 (Double) (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**String**、**Object**
CDec	10 進型 (Decimal) (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**String**、**Object**
CInt	整数型 (Integer) (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**String**、**Object**
CLng	Long データ型 (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**String**、**Object**
CObj	オブジェクト型 (Object)	任意の型
CSByte	SByte 型 (Visual Basic)	すべての数値型 (**Byte** および列挙型を含む)、**Boolean**、**String**、**Object**

CShort	Short 型 (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**String**、**Object**
CSng	単精度浮動小数点型 (Single) (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**String**、**Object**
CStr	文字列型 (String) (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**Char**、**Char** 配列、**Date**、**Object**
CType	コンマ (,) の後に指定する型	基本データ型 (基本型の配列を含む) に変換する場合は、対応する変換キーワードで使用できるのと同じ型。 複合データ型に変換する場合は、それが実装するインターフェイス、およびそれが継承するクラス。 **CType** をオーバーロードしたクラスまたは構造体に変換する場合は、そのクラスまたは構造体
CUInt	UInteger データ型	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**String**、**Object**
CULng	ULong データ型 (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**String**、**Object**
CUShort	UShort 型 (Visual Basic)	すべての数値型 (**Byte**、**SByte**、および列挙型を含む)、**Boolean**、**String**、**Object**

CType 関数

CType 関数 は 2 つの引数について実行されます。最初の引数は変換対象の式であり、2 番目の引数は変換先のデータ型またはオブジェクト クラスです。最初の引数は、型ではなく、式である必要があることに注意してください。

CType はインライン関数です。つまり、変換を行うコードはコンパイルによって生成されます。多くの場合、このコードに関数呼び出しは含まれていません。これにより、パフォーマンスが向上します。

CType とその他の型変換キーワードとの比較については、「DirectCast」および「TryCast」を参照してください。

基本型

次の例は **CType** の使い方を示しています。

```
k = CType(q, Integer)
' The following statement coerces w to the specific object class Label.
f = CType(w, Label)
```

複合型

CType を使用して、値を複合データ型および基本型に変換できます。また、次の例のように、オブジェクト クラスをそのインターフェイスの型に強制的に変換することもできます。

```
' Assume class cZone implements interface iZone.
Dim h As Object
' The first argument to CType must be an expression, not a type.
Dim cZ As cZone
' The following statement coerces a cZone object to its interface iZone.
h = CType(cZ, iZone)
```

配列型

CType は、次の例のように配列データ型を変換することもできます。

```
Dim v() As classV
Dim obArray() As Object
```

```
' Assume some object array has been assigned to obArray.
' Check for run-time type compatibility.
If TypeOf obArray Is classV()
    ' obArray can be converted to classV.
    v = CType(obArray, classV())
End If
```

使用例を含む詳細については、「配列の変換」を参照してください。

CType を定義する型

独自に定義したクラスや構造体に対して **CType** を定義できます。これによって、独自のクラスや構造体の型と別の型の間で値を相互に変換できます。使用例を含む詳細については、「方法 : 変換演算子を定義する」を参照してください。

メモ :

変換キーワードと共に使用する値は、変換先のデータ型において有効な値である必要があります。有効でない場合はエラーが発生します。たとえば、長整数型 **(Long)** を整数型 **(Integer)** に変換する場合は、その長整数型 **(Long)** の値が整数型 **(Integer)** の有効範囲内に含まれている必要があります。

注意

あるクラス型から別のクラス型に変換する **CType** を指定する場合、変換先の型が変換元の型から派生した型でなければ、実行時に失敗します。そのようなエラーが発生すると、InvalidCastException 例外がスローされます。

ただし、型の一方が独自に定義した構造体またはクラスであり、その構造体またはクラスに **CType** が定義してある場合、この **CType** の要件を満たしていれば変換は正常に実行されます。方法 : 変換演算子を定義する を参照してください。

明示的な型変換は、ある式から指定のデータ型またはオブジェクト クラスへのキャストとも呼ばれます。

参照

処理手順

方法 : Visual Basic でオブジェクトを別の型に変換する
データ型のトラブルシューティング

関連項目

データ型の概要 (Visual Basic)
データ型変換関数

概念

変換時の値の変化
文字列とその他の型との変換
Visual Basic での型宣言を省略したプログラミング

その他の技術情報

Visual Basic における型変換
構造体 : 独自のデータ型

変換時の値の変化

値型からの変換では、変換元の値のコピーが変換先に格納されます。ただし、このコピーは変換元の値の正確なイメージではありません。変換先のデータ型では値の格納方法が異なり、実行される変換の種類によっては、表現される値も変化する場合があります。

拡大変換と縮小変換での変化

縮小変換では、変換先にコピーされた値は変化し、場合によっては変換元に含まれていた情報が失われます。たとえば、小数値は整数型に変換されると丸められ、数値型は **Boolean** に変換されると **True** または **False** のいずれかになります。

拡大変換では、元の値は保持されますが、その表現は変化する場合があります。表現が変化するのは、整数型から 10 進型 (**Decimal**)、または Char 型 (**Char**) から文字列型 (**String**) に変換する場合です。

最初の変換元の値が変換の結果変化することはありません。

参照型変換での変化

参照型からの変換では、値へのポインタだけがコピーされます。値自体のコピーや変更は行われません。変化する可能性があるのは、ポインタを保持する変数のデータ型だけです。次の例では、データ型が派生クラスから基本クラスに変換されますが、両方の変数が指すことになるオブジェクトは変化しません。

```
' Assume class cSquare inherits from class cShape.
Dim shape As cShape
Dim square As cSquare = New cSquare
' The following statement performs a widening
' conversion from a derived class to its base class.
shape = square
```

参照

処理手順

方法 : Visual Basic でオブジェクトを別の型に変換する

関連項目

データ型の概要 (Visual Basic)

データ型変換関数

概念

拡大変換と縮小変換

暗黙の型変換と明示的な型変換

文字列とその他の型との変換

配列の変換

Visual Basic での型宣言を省略したプログラミング

その他の技術情報

Visual Basic における型変換

構造体 : 独自のデータ型

文字列とその他の型との変換

数値、ブール型 (**Boolean**)、または日付/時刻値は、**文字列型 (String)** に変換できます。文字列の内容が変換先のデータ型での有効な値として解釈される場合には、逆方向 (文字列値から数値、ブール型 (**Boolean**)、日付型 (**Date**) にも変換できます。変換できない場合は、ランタイム エラーが発生します。

これらすべての代入による変換は、いずれの方向の場合でも縮小変換です。型変換キーワード (**CBool**、**CByte**、**CDate**、**CDbl**、**CDec**、**CInt**、**CLng**、**CSByte**、**CShort**、**CSng**、**CStr**、**CUInt**、**CULng**、**CUShort**、および **CType**) を使用する必要があります。Format 関数 および Val 関数 を使用すると、文字列と数値の変換を詳細に制御できます。

定義されたクラスまたは構造体がある場合、**String** と、クラスまたは構造体の型との間での型変換演算子を定義できます。詳細については、「方法 : 変換演算子を定義する」を参照してください。

数値から文字列への変換

Format 関数を使用すると、数値を書式指定された文字列に変換できます。この文字列には、適切な数字だけではなく、通貨記号 (**$** など)、桁区切り記号つまり位取り記号 (**,** など)、小数点 (**.** など) などの書式指定のための記号も含まれます。**Format** では、Windows の [コントロール パネル] で指定された [地域のオプション] の設定に応じて、適切な記号が自動的に使用されます。

次の例のように、連結演算子 (**&**) を使用して数値を文字列に暗黙に変換できます。

```
' The following statement converts count to a String value.
Str = "The total count is " & count
```

文字列から数値への変換

Val 関数を使用して、明示的に、文字列内の数字を数値に変換できます。数字、スペース、タブ、改行、またはピリオド以外の文字を検出するまで、**Val** によって文字列が読み取られます。"&O" や "&H" というシーケンスによって基数法が変わり、スキャンが終了します。**Val** 関数では、読み取りを停止するまで、すべての該当する文字を数値に変換します。たとえば、次のステートメントでは値 `141.825` が返されます。

```
Val(" 14 1.825 miles")
```

Visual Basic で文字列を数値に変換する場合は、Windows の [コントロール パネル] の [地域のオプション] の設定を使用して、桁区切り記号、小数点の記号、および通貨記号を解釈します。つまり、ある設定では変換が正常に行われても、別の設定では失敗する場合があります。たとえば、`"$14.20"` は [英語 (U.S.)] ロケールでは受け入れることができますが、[フランス語] ロケールではできません。

参照

処理手順

方法 : Visual Basic でオブジェクトを別の型に変換する

関連項目

データ型の概要 (Visual Basic)
データ型変換関数

概念

拡大変換と縮小変換
暗黙の型変換と明示的な型変換
変換時の値の変化
配列の変換
.NET Framework ベースの国際対応アプリケーションの概要

その他の技術情報

Visual Basic における型変換

方法：Visual Basic でオブジェクトを別の型に変換する

Object 変数を他のデータ型に変換するには、CType 関数 などの変換キーワードを使用します。

使用例

次に示すのは、**Object** 変数を **Integer** 型から **String** 型へ変換する例です。

```
Public Sub objectConversion(ByVal anObject As Object)
    Dim anInteger As Integer
    Dim aString As String
    anInteger = CType(anObject, Integer)
    aString = CType(anObject, String)
End Sub
```

Object 変数の内容が特定のデータ型であることがわかっている場合は、変数をそのデータ型に変換することをお勧めします。**Object** 変数を使い続けると、ボックス化およびボックス化解除 (値型の場合)、または遅延バインディング (参照型の場合) が行われます。これらの操作はいずれも余分な実行時間を必要とするため、パフォーマンスが低下します。

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System 名前空間への参照

参照

関連項目

データ型の概要 (Visual Basic)
データ型変換関数
Object

概念

拡大変換と縮小変換
暗黙の型変換と明示的な型変換
変換時の値の変化
文字列とその他の型との変換
配列の変換
Visual Basic での型宣言を省略したプログラミング

その他の技術情報

Visual Basic における型変換
構造体 : 独自のデータ型

配列の変換

以下の条件が満たされる場合は、配列型を異なる配列型に変換できます。

- **ランクが等しいこと。** 2 つの配列のランクが同じである、つまり、次元数が同じである必要があります。ただし、それぞれの次元の長さは同じでなくてもかまいません。
- **要素のデータ型。** 両方の配列の要素のデータ型が、参照型である必要があります。整数型 (**Integer**) の配列を長整数型 (**Long**) またはオブジェクト型 (**Object**) の配列には変換できません。これは、少なくとも 1 つの値型が関連するためです。詳細については、「値型と参照型」を参照してください。
- **変換性。** 2 つの配列の要素型の間で、変換 (拡張または縮小) できる必要があります。この要件を満たさない例としては、**String** 配列と、System.Attribute から派生したクラスの配列との間での変換があります。これらの 2 つの型には共通する点はありません。また、両者の間では変換は行われません。

ある配列型から別の配列型への変換は、それぞれの要素の変換が拡張か縮小かによって、拡大変換または縮小変換になります。詳細については、「拡大変換と縮小変換」を参照してください。

オブジェクト型 (Object) の配列への変換

オブジェクト型 (**Object**) の配列を初期化せずに宣言すると、初期化されない限り、要素型はオブジェクト型 (**Object**) のままです。要素型を特定のクラスの配列に設定すると、そのクラスの型になります。ただし、基になる型はオブジェクト型 (**Object**) のままで、その後要素型を関係のないクラスの別な配列に設定できます。すべてのクラスがオブジェクト型 (**Object**) から派生するため、配列の要素型を任意のクラスから任意の他のクラスへ変更できます。

次の例では、`student` 型と **String** 型の間で変換は行われていませんが、共に **Object** から派生しているため、すべての代入が有効になります。

```
' Assume student has already been defined as a class.
Dim testArray() As Object
' testArray is still an Object array at this point.
Dim names() As String = New String(3) {"Name0", "Name1", "Name2", "Name3"}
testArray = New student(3) {}
' testArray is now of type student().
testArray = names
' testArray is now a String array.
```

基になる配列の型

最初に特定のクラスを使用して配列を宣言していた場合、配列の基となる要素型はそのクラスになります。後で要素型を他のクラスの配列に設定する場合は、2 つのクラス間で変換が必要になります。

次の例では、`students` が `student` 配列になります。**String** と `student` との変換はできないため、最後のステートメントはエラーになります。

```
Dim students() As student
Dim names() As String = New String(3) {"Name0", "Name1", "Name2", "Name3"}
students = New Student(3) {}
' The following statement fails at compile time.
students = names
```

参照

処理手順

方法 : Visual Basic でオブジェクトを別の型に変換する

関連項目

データ型の概要 (Visual Basic)
データ型変換関数

概念

Visual Basic におけるデータ型
暗黙の型変換と明示的な型変換
変換時の値の変化
文字列とその他の型との変換
Visual Basic での型宣言を省略したプログラミング

その他の技術情報

構造体：独自のデータ型

構造体は、以前のバージョンの Visual Basic でサポートされるユーザー定義型 (UDT: User-Defined Type) を一般化したものです。構造体は、フィールドの他、プロパティ、メソッド、およびイベントを公開します。構造体は 1 つ以上のインターフェイスを実装でき、フィールドごとにアクセス レベルを宣言できます。

異なる型のデータ項目を組み合わせて構造体を作成できます。構造体は、1 つ以上の要素を互いに、または構造体そのものと関連付けます。構造体を宣言すると複合データ型になり、その型の変数を宣言できます。

関連する複数の情報を 1 つの変数に保持する場合には構造体が有効です。たとえば、従業員名、内線番号、給与をまとめておく場合などです。この情報用に複数の変数を使用することもできますが、構造体を定義して個別の従業員用の変数として使用することもできます。従業員数が多く、その結果、変数のインスタンスを多数保持するような場合は、構造体は明らかに有効です。

このセクションの内容

方法 : 構造体を宣言する

構造体とその要素の宣言方法について説明します。

構造体の変数

変数への構造体の代入と要素へのアクセスについて説明します。

構造体とその他のプログラミング要素

構造体の配列、オブジェクト、プロシージャ、および構造体そのものとのやり取りについてまとめています。

構造体とクラス

構造体とクラスの類似点と相違点について説明します。

関連するセクション

Visual Basic におけるデータ型

Visual Basic データ型を示し、使用方法について説明します。

データ型の概要 (Visual Basic)

Visual Basic で用意されている基本データ型を一覧で紹介します。

方法：構造体を宣言する

構造体宣言は、Structure ステートメント で開始し、**End Structure** ステートメントで終了します。この 2 つのステートメントの間に、少なくとも 1 つの要素を宣言する必要があります。要素にはすべてのデータ型を使用できますが、少なくとも 1 つは、非共有変数または非共有で非カスタムのイベントのいずれかである必要があります。

構造体の宣言で構造体の要素を初期化することはできません。構造体型の変数を宣言するときは、変数をとおして要素にアクセスすることにより、要素に値を代入します。

構造体とクラスとの違いに関する詳細については、「構造体とクラス」を参照してください。

デモの目的で、従業員の名前、内線番号、および給与を追跡する状況を考えてみます。構造体により、単一の変数でこれを行うことができます。

構造体を宣言するには

1. 構造体の開始ステートメントと終了ステートメントを作成します。

   Public (Visual Basic) キーワード、Protected (Visual Basic) キーワード、Friend (Visual Basic) キーワード、または Private (Visual Basic) キーワードを使用して、構造体のアクセス レベルを指定できます。または既定の **Public** に設定できます。

   ```
   Private Structure employee
   End Structure
   ```

2. 構造体の本体に要素を追加します。

   構造体は、少なくとも 1 つの要素を持つ必要があります。すべての要素を宣言し、それぞれのアクセス レベルを指定する必要があります。キーワードを指定せずに Dim ステートメント (Visual Basic) を使用する場合、アクセシビリティは既定で **Public** になります。

   ```
   Private Structure employee
       Public givenName As String
       Public familyName As String
       Public phoneExtension As Long
       Private salary As Decimal
       Public Sub giveRaise(raise As Double)
           salary *= raise
       End Sub
       Public Event salaryReviewTime()
   End Structure
   ```

   上の例の `salary` フィールドは、**Private** です。つまり、含まれているクラスからでも、構造体の外部ではアクセスできません。ただし、`giveRaise` プロシージャは **Public** であるため、構造体以外から呼び出すことができます。同様に、構造体以外から `salaryReviewTime` イベントを発生させることができます。

   変数、**Sub** プロシージャ、およびイベントに加え、定数、**Function** プロシージャ、およびプロパティも構造体で定義できます。既定のプロパティとしてプロパティを 1 つ指定できます。ただし、少なくとも 1 つの引数を指定する必要があります。イベントは Shared (Visual Basic) **Sub** プロシージャを使用して処理できます。詳細については、「方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)」を参照してください。

参照

処理手順

データ型のトラブルシューティング

関連項目

ユーザー定義型

概念

Visual Basic におけるデータ型
複合データ型
構造体の変数
構造体とその他のプログラミング要素
構造体とクラス

その他の技術情報

基本データ型
データ型の実装
構造体 : 独自のデータ型

構造体の変数

構造体を作成したら、その型でプロシージャ レベルの変数およびモジュール レベルの変数を宣言できます。たとえば、コンピュータ システムに関する情報を記録する構造体を作成するとします。次にコード例を示します。

```
Public Structure systemInfo
    Public cPU As String
    Public memory As Long
    Public purchaseDate As Date
End Structure
```

これで、この型の変数を宣言できるようになります。次に例を示します。

```
Dim mySystem, yourSystem As systemInfo
```

メモ :

クラス内およびモジュール内では、Dim ステートメント (Visual Basic) を使用して宣言した構造体は既定でパブリック アクセスになります。構造体をプライベートにする場合は、Private (Visual Basic) キーワードを使用して宣言します。

構造体の値へのアクセス

構造体変数の要素で値を代入したり取得したりするには、オブジェクトのプロパティの設定や取得に使用する構文を指定します。メンバ アクセス演算子 (.) は、構造体変数名と要素名の間に配置します。`systemInfo` 型として既に宣言されている変数の要素にアクセスする例を次に示します。

```
mySystem.cPU = "486"
Dim tooOld As Boolean
If yourSystem.purchaseDate < #1/1/1992# Then tooOld = True
```

構造体変数の代入

両方の変数の構造体が同じ型の場合は、ある変数を別の変数に代入できます。これによって、一方の構造体のすべての要素が、もう一方の構造体の対応する要素にコピーされます。次に例を示します。

```
yourSystem = mySystem
```

構造体の要素が文字列型 (**String**) やオブジェクト型 (**Object**) などの参照型である場合は、データへのポインタがコピーされます。上の例で、`systemInfo` にオブジェクト変数が含まれていたとすると、`mySystem` から `yourSystem` にはポインタがコピーされることになります。そのため、一方の構造体をとおしてオブジェクトのデータに変更を加えると、もう一方の構造体をとおしてアクセスする内容にも変更が反映されます。

参照

処理手順

データ型のトラブルシューティング

方法 : 構造体を宣言する

関連項目

Structure ステートメント

概念

Visual Basic におけるデータ型

複合データ型

構造体とその他のプログラミング要素

構造体とクラス

その他の技術情報

基本データ型

データ型の実装

構造体 : 独自のデータ型

構造体とその他のプログラミング要素

構造体は、配列、オブジェクト、プロシージャ、および他の構造体と組み合わせて使用できます。このやり取りには、各要素が個別に使用する構文が使用されます。

メモ：

構造体の宣言で構造体の要素を初期化することはできません。構造体型として宣言された変数の要素に対してだけ、値を割り当てることができます。

構造体と配列

構造体には、1 つまたは複数の要素として配列を含めることができます。次に例を示します。

```
Public Structure systemInfo
    Public cPU As String
    Public memory As Long
    Public diskDrives() As String
    Public purchaseDate As Date
End Structure
```

オブジェクトのプロパティにアクセスするのと同じ方法で構造体内の配列の値にアクセスします。次に例を示します。

```
Dim mySystem As systemInfo
ReDim mySystem.diskDrives(3)
mySystem.diskDrives(0) = "1.44 MB"
```

構造体の配列を宣言することもできます。次に例を示します。

```
Dim allSystems(100) As systemInfo
```

このデータ アーキテクチャのコンポーネントにアクセスする際にも同じルールに従います。次に例を示します。

```
ReDim allSystems(5).diskDrives(3)
allSystems(5).CPU = "386SX"
allSystems(5).diskDrives(2) = "100M SCSI"
```

構造体とオブジェクト

構造体には、1 つまたは複数の要素としてオブジェクトを含めることができます。次に例を示します。

```
Protected Structure userInput
    Public userName As String
    Public inputForm As System.Windows.Forms.Form
    Public userFileNumber As Integer
End Structure
```

このような宣言では、オブジェクト型 (**Object**) ではなく特定のオブジェクト クラスを使用してください。

構造体とプロシージャ

プロシージャの引数として構造体を渡すことができます。次に例を示します。

```
Public currentCPUName As String = "700MHz Pentium compatible"
Public currentMemorySize As Long = 256
Public Sub fillSystem(ByRef someSystem As systemInfo)
    someSystem.cPU = currentCPUName
    someSystem.memory = currentMemorySize
    someSystem.purchaseDate = Now
```

```
End Sub
```

上の例では、参照によって構造体を渡していますが、この方法ではプロシージャで要素を変更でき、呼び出し側のコードで変更内容が有効になります。構造体が変更されないようにするには、構造体を値で渡します。

Function プロシージャから構造体を返すこともできます。次に例を示します。

```
Dim allSystems(100) As systemInfo
Function findByDate(ByVal searchDate As Date) As systemInfo
    Dim i As Integer
    For i = 1 To 100
        If allSystems(i).purchaseDate = searchDate Then Return allSystems(i)
    Next i
   ' Process error: system with desired purchase date not found.
End Function
```

構造体内の構造体

構造体には他の構造体を含めることもできます。次に例を示します。

```
Public Structure driveInfo
    Public type As String
    Public size As Long
End Structure
Public Structure systemInfo
    Public cPU As String
    Public memory As Long
    Public diskDrives() As driveInfo
    Public purchaseDate As Date
End Structure
Dim allSystems(100) As systemInfo
ReDim allSystems(1).diskDrives(3)
allSystems(1).diskDrives(0).type = "Floppy"
```

この方法を使用すると、あるモジュールで定義された構造体を別のモジュールで定義された構造体にカプセル化できます。

構造体は、任意の深さで入れ子にできます。

参照

処理手順

データ型のトラブルシューティング
方法 : 構造体を宣言する

関連項目

Structure ステートメント

概念

Visual Basic におけるデータ型
複合データ型
構造体の変数
構造体とクラス

その他の技術情報

基本データ型
データ型の実装
構造体 : 独自のデータ型

構造体とクラス

Visual Basic では、構造体とクラスの構文が統一され、両方のエンティティでほとんど同じ機能がサポートされるようになりました。ただし、構造体とクラスの間には重要な違いもあります。

クラスには、参照型としての利点があります。たとえば、参照型を渡す方が、すべてのデータが格納されている構造体変数を渡すよりも効率的です。一方、構造体の場合は、グローバル ヒープにメモリを割り当てる必要がありません。

構造体からは継承ができないため、構造体は拡張する必要のないオブジェクトにだけ使用することをお勧めします。構造体は、作成するオブジェクトのインスタンスのサイズが小さい場合に使用します。その場合はクラスと構造体の特性の違いを考慮してください。

類似点

構造体とクラスの類似点は次のとおりです。

- どちらも container 型であり、他の型をメンバとして含みます。
- どちらにもメンバが含まれます。メンバとしては、コンストラクタ、メソッド、プロパティ、フィールド、定数、列挙値、イベント、およびイベント ハンドラを含むことができます。しかし、これらのメンバと、宣言された構造体の要素とを混同しないでください。
- どちらのメンバにも、個別のアクセス レベルを設定できます。たとえば、一方のメンバを **Public** と宣言し、他方のメンバを **Private** と宣言することができます。
- どちらもインターフェイスを実装できます。
- どちらもパラメータ付きまたはパラメータなしの共有コンストラクタを含むことができます。
- どちらも既定のプロパティを公開できます。ただし、少なくとも 1 つのパラメータを指定する必要があります。
- どちらもイベントを宣言し発生させて、デリゲートを宣言できます。

異なる点

構造体とクラスの違いは次のとおりです。

- 構造体は値型、クラスは参照型です。クラス型がデータへの参照を格納するのとは異なり、構造体型の値は構造体のデータそのものを格納します。
- 構造体はスタック割り当てを使用し、クラスはヒープ割り当てを使用します。
- 構造体のすべての要素は既定で **Public** です。クラスでは、変数および定数は既定で **Private**、他のメンバは既定で **Public** です。これにより、クラスのメンバは Visual Basic 6.0 の既定値システムとの互換性を持ちます。
- 構造体には少なくとも 1 つの共有されない変数、または共有されない非カスタムのイベント要素を含む必要がありますが、クラスは完全に空にすることができます。
- 構造体の要素は、**Protected** として宣言できませんが、クラス メンバは宣言できます。
- 構造体のプロシージャは、Shared (Visual Basic) **Sub** プロシージャである場合にだけ、AddHandler ステートメントを使用してのみイベントを処理できますが、クラスのプロシージャはどれでも、Handles キーワードまたは **AddHandler** ステートメントを使用してイベントを処理できます。詳細については、「イベントとイベント ハンドラ」を参照してください。
- 構造体の変数宣言では、初期化子または配列の初期サイズを指定できませんが、クラスの変数宣言では指定できます。
- 構造体は暗黙に System.ValueType クラスから継承し、それ以外の型からは継承できませんが、クラスは **System.ValueType** 以外のすべてのクラスから継承できます。
- 構造体は継承できませんが、クラスは継承できます。
- 構造体は終了されないため、共通言語ランタイム (CLR: Common Language Runtime) は構造体に対して Finalize メソッドを呼び出しませんが、クラスはガベージ コレクタによって終了され、ガベージ コレクタはアクティブな参照が残っていないことを検出すると、クラスに対して **Finalize** を呼び出します。
- 構造体はコンストラクタを必要としませんが、クラスには必要です。
- 構造体にはパラメータを取る場合にだけ非共有コンストラクタを含むことができますが、クラスにはパラメータの有無には関係なく非共有コンストラクタを含むことができます。

すべての構造体には、パラメータがない暗黙のパブリック コンストラクタが含まれます。このコンストラクタによって、構造体のすべてのデータ要素が既定値に初期化されます。この動作は再定義できません。

インスタンスと変数

構造体は値型であるため、各構造体変数は個別の構造体インスタンスに恒久的に関連付けされます。一方、クラスは参照型であり、オブジェクト変数はさまざまなクラス インスタンスを異なる時点で参照できます。この違いは、構造体とクラスの使用方法に次のように影響します。

- **初期化** 構造体変数には、構造体のパラメータなしコンストラクタを使って、要素の初期化が暗黙的に含まれます。したがって、`Dim s As struct1` は `Dim s As struct1 = New struct1()` に相当します。

- **変数の割り当て** 構造体変数を別の構造体変数に代入するか、または構造体インスタンスをプロシージャ引数に渡すと、すべての変数要素の現在の値が新しい構造体にコピーされます。オブジェクト変数を別のオブジェクト変数に代入するか、またはオブジェクト変数をプロシージャに渡すと、参照ポインタだけがコピーされます。

- **Nothing の割り当て** 構造体変数には値 Nothing (Visual Basic) を代入できますが、インスタンスは引き続き変数に関連付けられています。変数要素は代入によって再初期化されますが、以降もその変数のメソッドを呼び出したり、そのデータ要素にアクセスしたりできます。

  一方、オブジェクト変数を **Nothing** に設定した場合は、クラス インスタンスとの関連付けが解除され、別のインスタンスを代入するまでその変数をとおしてメンバにアクセスすることはできません。

- **複数のインスタンス** オブジェクト変数には異なる時点で異なるクラス インスタンスを代入でき、複数のオブジェクト変数が同時に同じクラス インスタンスを参照できます。クラスのメンバの値に加えた変更は、同じインスタンスを指す別の変数をとおしてアクセスされたときにそのメンバに反映されます。

  一方、構造体の要素は、インスタンスごとに独立しています。メンバの値に加えられた変更は、他の構造体変数には反映されません。同じ **Structure** 宣言の他のインスタンスであっても反映されません。

- **等値式** 2 つの構造体の等価テストは、要素ごとに実行する必要があります。Equals メソッドを使用して、2 つのオブジェクト変数を比較できます。**Equals** は、2 つの変数が同じインスタンスを指しているかどうかを示します。

参照

処理手順

データ型のトラブルシューティング

概念

Visual Basic におけるデータ型
複合データ型
構造体とその他のプログラミング要素
値型と参照型

その他の技術情報

データ型の実装
構造体 : 独自のデータ型
クラスについて

Visual Basic での型宣言を省略したプログラミング

Visual Basic コンパイラには、ソース コードに影響を及ぼす次の 2 つの規則を適用する機能があります。

- すべてのローカル変数は、他のステートメントで使用する前に宣言ステートメントで指定する必要がある。この規則は既定で適用されます。
- すべての型の縮小変換は、型変換キーワードを使用して指定する必要がある。この規則は既定では適用されません。

規則を変更するコンパイラ オプション

この規則の 1 つまたは両方は、対応するコンパイラ オプションを再設定することによって変更できます。型変換の規則を緩めると、コンパイラは寛容な型指定規則に基づいて動作します。この場合は、変換キーワードを使用せずに縮小変換を実行できます。詳細については、「Visual Basic における型チェック」を参照してください。

変数宣言の規則を緩めると、型宣言を省略したプログラミングを使用できます。この場合は、宣言していない変数を参照できます。詳細については、「暗黙の宣言と明示的宣言」を参照してください。

通常、型宣言を省略したプログラミングはお勧めしません。データ型に関連した、発見しにくい実行エラーの原因となる場合があるためです。ただし、変数を宣言していない古いアプリケーションを移植する場合など、状況によっては役立ちます。

参照

処理手順

データ型のトラブルシューティング

関連項目

Option Explicit ステートメント (Visual Basic)
Option Strict ステートメント

概念

Visual Basic におけるデータ型
型文字
複合データ型
汎用データ型としてのオブジェクト型 (Object)
データ型の有効な使用方法

その他の技術情報

基本データ型
Visual Basic における型変換

暗黙の宣言と明示的宣言

既定では、Visual Basic コンパイラは明示的宣言を行います。この場合は、すべての変数を使用前に宣言する必要があります。この要件を取り除くと、暗黙の宣言を許可できます。

Visual Basic には、明示的宣言を制御するスイッチが用意されています。既定では、このスイッチは **On** に設定され、コンパイラは明示的な宣言を要求します。このスイッチを **Off** にすると、変数を宣言しなくても使用できます。

明示的宣言スイッチ。 明示的宣言スイッチの **On** と **Off** の設定は、次のいずれかの方法で行います。

- 統合開発環境 (IDE: Integrated Development Environment) で、適切なプロジェクト プロパティを設定します。[プロジェクト] メニューの [<プロジェクト名> プロパティ] をクリックし、[コンパイル] タブをクリックします。[Option Explicit]、[Option Strict]、および [Option Compare] の既定値を設定できます。
- /optionexplicit コマンド ライン コンパイラ オプションを指定します。
- コードの先頭に Option Explicit ステートメント (Visual Basic) を含めます。

Option Explicit ステートメントを使用すると、そのステートメントが指定されたソース コード ファイルについては、プロジェクト プロパティやコンパイラ オプションの設定よりも優先されます。

パフォーマンス上の利点。 **Option Explicit** を **On** にすると、実行時ではなくコンパイル時に型の推定を行うよう強制できる利点があります。これにより、パフォーマンスが向上します。

暗黙の宣言

Option Explicit を **Off** にすると、コード内で変数を単に使用することによって変数を暗黙的に宣言できます。暗黙的に宣言したすべての変数に対して、コンパイラは オブジェクト型 (Object) を割り当てます。ただし、すべての変数を明示的に特定のデータ型として宣言した方がアプリケーションは効率的に動作します。明示的な宣言によって、名前の競合によるエラーやスペル ミスの発生は減少します。また、整数型 (**Integer**) を短整数型 (**Short**) に代入するなどのランタイム エラーをコンパイラで検出できます。

発生する可能性のあるエラー
意図しない新規の変数

ローカル変数を宣言しないプロシージャを作成できます。次に例を示します。

```
Function safeSqrt(num)
' Make sure num is positive for square root.
    tempVal = Math.Abs(num)
    Return Math.Sqrt(tempVal)
End Function
```

Visual Basic では、`tempVal` がローカル変数として自動的に作成され、明示的に宣言した場合と同様に使用できます。これは便利ですが、変数名のスペルを間違えると、コード内で軽度のエラーが発生する場合があります。上の例で、次のようなプロシージャを記述したとします。

```
Function safeSqrt(num)
' Make sure num is positive for square root.
    tempVal = Math.Abs(num)
    Return Math.Sqrt(temVal)
End Function
```

一見すると、この 2 つのコードは同じです。しかし、`Sqrt` の引数の変数 `tempVal` にスペル ミスがあるため、コンパイラで `temVal` という別のローカル変数が作成されます。この変数には値が代入されないので、この関数は常に 0 を返します。

既存の要素への意図しない参照

Visual Basic で新しい名前が検出された場合、それが新規変数の暗黙的な宣言なのか、既存の変数名のスペルが間違っているのかを判断できません。その結果、この名前の変数が新規作成されます。また、同じ名前で変数または別のプログラミング要素が既に定義されている場合、コードでは定義済みの要素を無意識に使用することになります。

明示的宣言を使用すると、名前のスペル ミスによる問題は回避できます。

明示的宣言

上の例で、`safeSqrt` 関数を含むモジュールについて明示的宣言が有効になっていたとすると、Visual Basic では `tempVal` と `temVal` を宣言

されていない変数と認識し、両方についてエラーを生成します。そのため、tempVal を明示的に宣言するよう修正する必要があります。次に例を示します。

```
Function safeSqrt(ByVal num As Double) As Double
' Make sure num is positive for square root.
    Dim tempVal As Double = Math.Abs(num)
    Return Math.Sqrt(temVal)
End Function
```

この変更後のコードでは、Visual Basic ではスペル ミスがある temVal についてエラーメッセージが表示されるため、問題をすぐに特定できます。明示的宣言を使用すると、このようなエラーを見つけるのに役立つため、すべてのコードで使用することをお勧めします。

メモ :

Option Explicit ステートメントはファイル単位で機能します。このステートメントは、変数の明示的宣言の実行を制御するすべてのソース コード ファイルの先頭に指定する必要があります。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

関連項目

データ型の概要 (Visual Basic)
データ型変換関数

概念

Visual Basic での型宣言を省略したプログラミング
Visual Basic における型チェック
汎用データ型としてのオブジェクト型 (Object)
データ型の有効な使用方法
Visual Basic における変数
Visual Basic の名前付け規則

Visual Basic における型チェック

Visual Basic コンパイラは、データ型を変換する場合に厳密な型指定規則または寛容な型指定規則に基づいて動作します。厳密な型指定規則が有効な場合、拡大変換だけが暗黙に許可され、縮小変換は明示的に実行する必要があります。寛容な型指定規則では、拡大変換と縮小変換をすべて暗黙に試行できます。型変換規則は、オブジェクト型も含めて、すべてのデータ型の変換に適用されます。

型チェック オプションの設定方法

Visual Basic には、型チェックを制御するスイッチが用意されています。既定では、このスイッチは **Off** に設定され、コンパイラでは暗黙の縮小変換が許可されます。このスイッチを **On** にすると、コンパイラは厳密な型指定規則を適用します。

型チェック スイッチ

型チェック スイッチの **On** と **Off** の設定は、次のいずれかの方法で行います。

- 統合開発環境 (IDE: Integrated Development Environment) の [プロパティ] ウィンドウで適切なプロジェクト プロパティを設定する。
- コマンド ラインのコンパイラ オプション /optionstrict を指定する。
- コードの先頭に **Option Strict** ステートメントを含める。

Option Strict ステートメントを使用すると、そのステートメントが指定されたソース コード ファイルについては、プロジェクト プロパティやコンパイラ オプションの設定よりも優先されます。詳細については、「Option Strict ステートメント」を参照してください。

パフォーマンス上の利点

Option Strict を **On** にすると、事前バインディングを強制できる利点があります。これにより、パフォーマンスが向上します。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

関連項目

データ型の概要 (Visual Basic)
データ型変換関数

概念

Visual Basic におけるデータ型
Visual Basic での型宣言を省略したプログラミング
暗黙の宣言と明示的宣言
汎用データ型としてのオブジェクト型 (Object)
データ型の有効な使用方法
拡大変換と縮小変換
暗黙の型変換と明示的な型変換

汎用データ型としてのオブジェクト型 (Object)

オブジェクト型 (Object) は、.NET Framework および Visual Basic におけるルート型です。つまり、他のすべてのデータ型およびオブジェクト型は、直接的または最終的にこの型から派生しています。また、他のすべての基本型および複合型のデータ型は、オブジェクト型 (**Object**) に変換できます。

柔軟な型指定

Object は、汎用データ型として使用できます。これは、柔軟な型指定と呼ばれます。この柔軟性の例を次に示します。

```
Dim v As Object
v = "17"
' v contains the 2-character String value "17".
v = v - 15
' v now contains the Integer value 2.
v = "H" & v
' v now contains the 2-character String value "H2".
```

Object 変数に対する演算は、変数に含まれるデータの種類を特に考慮しなくても実行できますが、次の点に注意する必要があります。

- **Object** に対して算術演算や算術関数を実行する場合は、その変数に数値データが保持されている必要があります。そうでないと、ランタイム エラーが発生します。
- 文字列を連結する場合は、**+** 演算子ではなく **&** 演算子を使用します。**+** 演算子がどのような場合にオペランドの加算として機能し、どのような場合にオペランドどうしの連結として機能するかは、非常に複雑な規則によって決定されています。また、**+** 演算子では、場合によっては型チェックや変換を実行する必要があるためパフォーマンスが低下します。

オブジェクト変数のデータ型を変更する

Object 変数はすべての型のデータを受け入れることができますが、変数自体を異なるデータ型に変更することはできません。ただし、オブジェクト型 (**Object**) の変数からオブジェクト型 (**Object**) の変数の配列へは再次元設定できます。ReDim ステートメント (Visual Basic) の有効な使用例と無効な使用例を次に示します。

```
' The following statement declares a single Object.
Dim someObj As Object
' The following reallocation is valid only for Object.
ReDim someObj(8)
' The following statement attempts an INVALID change of data type.
ReDim someObj(8) As Double
```

最初の **ReDim** ステートメントでは、`someObj` を **Object** 型の配列に変更します。これが有効なのは、オブジェクト型 (**Object**) の場合だけです。2 番目の **ReDim** ステートメントでは、異なるデータ型が使用されているために無効です。この例で意図されている結果を得るには、もう 1 つ別の配列を使用します。次に例を示します。

```
' First allocate a separate array.
Dim someArray(8) As Double
' Then assign the new array to the Object variable.
someObj = someArray
```

構造体の代わりとしてのオブジェクト型 (Object)

オブジェクト型 (**Object**) はさまざまな型のデータを格納できるため、構造体を使用する多くの状況ではオブジェクト型 (**Object**) の配列を使用できます。オブジェクト型 (**Object**) の配列は、各要素に格納するデータ型をいつでも変更したり、配列を動的に作成して必要に応じてサイズを変更したりできるため、構造体よりも多少柔軟性が高くなります。ただし、**Object** 配列は、同等の構造体に比べて多くのメモリが必要となり、パフォーマンスも低下します。

参照

関連項目

データ型の概要 (Visual Basic)
データ型変換関数

概念

Visual Basic での型宣言を省略したプログラミング
暗黙の宣言と明示的宣言
Visual Basic における型チェック
データ型の有効な使用方法
オブジェクト変数の宣言
オブジェクト変数の代入
オブジェクト変数の値

その他の技術情報

Visual Basic における型変換

データ型の有効な使用方法

宣言されていない変数、およびデータ型を指定しないで宣言された変数には、**Object** データ型が割り当てられます。したがって、プログラムの記述が簡単になりますが、実行速度は遅くなることがあります。

厳密な型指定

すべての変数についてデータ型を指定することは、厳密な型指定と呼ばれます。厳密な型指定を使用する利点は次のとおりです。

- 変数に対する IntelliSense® サポートが有効になります。これにより、コードの入力時に、プロパティや他のメンバを表示できます。
- コンパイラの型チェック機能を利用できます。これにより、実行時にオーバーフローなどのエラーで失敗するステートメントが検出されます。また、オブジェクトでサポートされていないメソッドの呼び出しも検出されます。
- コードの実行速度が速くなります。

最も効率的なデータ型

小数を含まない変数では、整数型が非整数型よりも有効です。Visual Basic では、**Integer** および **UInteger** が最も効率的な数値型です。

小数値の場合、現在のプラットフォームのプロセッサでは浮動小数点演算が倍精度で実行されるため、**Double** が最も効率的なデータ型です。ただし、**Double** による演算は、**Integer** などの整数型ほど高速ではありません。

データ型の指定

特定の型の変数を宣言するには、Dim ステートメント (Visual Basic) を使用します。次の例のように、Public (Visual Basic) キーワード、Protected (Visual Basic) キーワード、Friend (Visual Basic) キーワード、または Private (Visual Basic) キーワードを使用して同時にアクセス レベルも指定できます。

```
Private x As Double
Protected s As String
```

文字の変換

AscW 関数と **ChrW** 関数は Unicode で動作します。Unicode との変換が必要な **Asc** や **Chr** よりも、これらを優先的に使用することをお勧めします。

参照

関連項目

Asc 関数、AscW 関数
Chr 関数、ChrW 関数

概念

Visual Basic におけるデータ型
Visual Basic での型宣言を省略したプログラミング
暗黙の宣言と明示的宣言
Visual Basic における型チェック
汎用データ型としてのオブジェクト型 (Object)
数値データ型
Visual Basic での変数宣言

その他の技術情報

IntelliSense の使用方法

データ型のトラブルシューティング

ここでは、組み込みデータ型の演算で起きる一般的な問題についていくつか説明します。

浮動小数点式での比較が等しくならない

浮動小数点数 (単精度浮動小数点型 (Single) (Visual Basic) および 倍精度浮動小数点数型 (Double) (Visual Basic)) を扱うときは、数値が 2 進数の小数として格納されることに注意してください。つまり、2 進数 (*k* / (2 ^ *n*) ではない値は正確に格納できません (*k* と *n* は整数)。たとえば、0.5 (= 1/2) や 0.3125 (= 5/16) は正確な値を格納できますが、0.2 (= 1/5) や 0.3 (= 3/10) は近似値のみを格納できます。

このような不正確性があるため、浮動点演算の結果が常に正確であると見なすことはできません。特に、理論的には等しい 2 つの値が、わずかに異なる数値として表現される場合があります。

浮動小数点の値を比較するには

1. System 名前空間にある Math クラスの Abs メソッドを使用して、2 つの数値の絶対的な差を計算します。
2. 許容できる最大の差を決めます。2 つの数値の差がこれより小さい場合、2 つが等価であると見なしても実用上の問題がないような値にしてください。
3. 差の絶対値と許容可能な差を比較します。

次のコード例は、2 つの **Double** 値を比較する誤った方法と正しい方法を示しています。

VB

```
Dim oneThird As Double = 1.0 / 3.0
Dim pointThrees As Double = 0.333333333333333

' The following comparison does not indicate equality.
Dim exactlyEqual As Boolean = (oneThird = pointThrees)

' The following comparison indicates equality.
Dim closeEnough As Double = 0.000000000000001
Dim absoluteDifference As Double = Math.Abs(oneThird - pointThrees)
Dim practicallyEqual As Boolean = (absoluteDifference < closeEnough)

MsgBox("1.0 / 3.0 is represented as " & oneThird.ToString("G17") _
  & vbCrLf & "0.333333333333333 is represented as " _
  & pointThrees.ToString("G17") _
  & vbCrLf & "Exact comparison generates " & CStr(exactlyEqual) _
  & vbCrLf & "Acceptable difference comparison generates " _
  & CStr(practicallyEqual))
```

この例では、Double 構造体の ToString メソッドを使用しているので、**CStr** キーワードを使用する場合より高い精度が得られます。既定では 15 桁ですが、"G17" 形式を使うと 17 桁に拡張されます。

Mod 演算子が正確な結果を返さない

浮動小数点数が正確に格納されないことがあるため、オペランドの少なくとも 1 つが浮動小数点数である場合に、Mod 演算子 (Visual Basic) が予期しない結果を返す可能性があります。

10 進型 (Decimal) (Visual Basic) は、浮動小数点表現を使用しません。**Single** および **Double** に正確に格納されない多くの値が、**Decimal** には正確に格納されます (たとえば 0.2 や 0.3)。**Decimal** に格納すると浮動小数点に格納した場合より演算は遅くなりますが、それによって得られる精度はパフォーマンスを犠牲にする価値があります。

浮動小数点数の整数剰余を求めるには

1. 変数を **Decimal** で宣言します。
2. リテラルの型文字 **D** を使って、強制的にリテラルを **Decimal** にします。これは、値が **Long** データ型で扱えない大きさである場合に備えるためです。

浮動小数点数のオペランドによって誤差が生じる例を次に示します。

VB

```
Dim two As Double = 2.0
Dim zeroPointTwo As Double = 0.2
Dim quotient As Double = two / zeroPointTwo
Dim doubleRemainder As Double = two Mod zeroPointTwo

MsgBox("2.0 is represented as " & two.ToString("G17") _
  & vbCrLf & "0.2 is represented as " & zeroPointTwo.ToString("G17") _
  & vbCrLf & "2.0 / 0.2 generates " & quotient.ToString("G17") _
  & vbCrLf & "2.0 Mod 0.2 generates " _
  & doubleRemainder.ToString("G17"))

Dim decimalRemainder As Decimal = 2D Mod 0.2D
MsgBox("2.0D Mod 0.2D generates " & CStr(decimalRemainder))
```

この例では、**Double** 構造体の **ToString** メソッドを使用しているので、**CStr** キーワードを使用する場合より高い精度が得られます。既定では 15 桁ですが、"G17" 形式を使うと 17 桁に拡張されます。

`zeroPointTwo` は **Double** 型なので、0.2 を格納すると、小数部分が永久に繰り返される 0.20000000000000001 という値になります。この値で 2.0 を除算すると、答えは 9.9999999999999995 となり、剰余が 0.19999999999999991 となります。

`decimalRemainder` の式では、リテラルの型文字 `D` が指定されているため、2 つのオペランドは 10 進型 (**Decimal**) になり、0.2 が正確に表現されます。したがって、**Mod** 演算子は、予期されるとおりの剰余 0.0 を生成します。

`decimalRemainder` を **Decimal** で宣言するだけでは十分ではないことに注意してください。強制的にリテラルを **Decimal** にすることも必要です。そうしなければ、リテラルは既定で **Double** となり、`decimalRemainder` は `doubleRemainder` と同じ不正確な値を受け取ります。

Boolean 型が数値型に正確に変換されない

ブール型 (Boolean) (Visual Basic) 値は数字としては格納されず、格納された値は数字と等しくなりません。以前のバージョンとの互換性維持のため、Visual Basic には、**Boolean** 型と数値型を相互に変換するための変換用のキーワード (CType 関数、**CBool**、**CInt** など) が用意されています。ただし、他の言語では、この変換が別の方法で実行されることもあります。たとえば、.NET Framework のメソッドがそうです。

コードを作成する際、**True** および **False** に相当する数値を利用しないようにしてください。ブール型 (**Boolean**) の変数に対しては、できるだけ本来の論理値だけを使用してください。ブール型 (**Boolean**) と数値を混在させる必要がある場合は、選択した変換メソッドを十分理解してから行うようにします。

Visual Basic における変換

変換用の **CType** キーワードまたは **CBool** キーワードを使って数値型を **Boolean** 型に変換すると、0 は **False** になり、それ以外のすべての値は **True** になります。変換用のキーワードを使って **Boolean** 値を数値型に変換すると、**False** は 0 になり、**True** は -1 になります。

フレームワークにおける変換

System 名前空間にある Convert クラスの ToInt32 メソッドを使うと、**True** は +1 に変換されます。

Boolean 値を数値データ型に変換する必要がある場合は、使用する変換メソッドに注意してください。

文字リテラルがコンパイル エラーを発生させる

型宣言文字がない場合、Visual Basic ではデータ型がリテラルであると見なされます。文字リテラル (二重引用符 (" ") で囲まれた文字) の既定の型は、**String** です。

String データ型は、文字型 (Char) (Visual Basic) に拡張されません。つまり、リテラルを **Char** 型の変数に代入する場合には、縮小変換を行うか、強制的にリテラルを **Char** 型にする必要があります。

Char リテラルを作成して変数または定数に代入するには

1. 変数または定数を **Char** で宣言します。
2. 文字値は二重引用符 (" ") で囲みます。
3. 二重引用符の後にリテラルの型文字 **C** を付けて、強制的にリテラルを **Char** 型にします。これが必要なのは型チェック スイッチ (Option Strict ステートメント) が **On** の場合ですが、常にこのようにすることを推奨します。

次のコード例は、リテラルを **Char** 型の変数に代入する誤った方法と正しい方法を示しています。

VB

```
Option Strict On
```

VB

```
Dim charVar As Char
' The following statement attempts to convert a String literal to Char.
' Because Option Strict is On, it generates a compiler error.
charVar = "Z"
' The following statement succeeds because it specifies a Char literal.
charVar = "Z"c
' The following statement succeeds because it converts String to Char.
charVar = CChar("Z")
```

縮小変換は実行時にエラーになる可能性があるため、常にリスクを伴います。たとえば、**String** から **Char** への変換は、**String** 値が 1 文字より多い場合にエラーになります。したがって、**C** 型文字を使う方が推奨されるプログラミング方法です。

文字列変換が実行時にエラーになる

文字列型 (String) (Visual Basic) が拡張変換にかかわることはめったにありません。**String** が拡張されるのは同じ文字列型か **Object** だけであり、**Char** および **Char()** (**Char** 配列) だけが **String** に拡張されます。その理由は、**String** の変数および定数には、他のデータ型には格納できない値が格納できるからです。

型チェック スイッチ (Option Strict ステートメント) が **On** の場合、コンパイラは、すべての暗黙の縮小変換を (**String** がかかわる縮小変換を含めて) 認めません。その場合でも、**CStr** や CType 関数 などの変換キーワードをコードで使用できます。これらがあると、.NET Framework は変換を試みます。

縮小変換の保護

縮小変換の欠点は、実行時にエラーになる可能性があることです。たとえば、**String** 変数に "True" または "False" 以外の値が格納されている場合、**Boolean** には変換できません。区切り記号が含まれている場合は、数値型への変換はエラーになります。変換先のデータ型で許容される値が常に **String** 変数に格納されるという確信がない限り、変換は行わないでください。

String から別のデータ型に変換する必要がある場合、最も安全な手順は、変換を試みる部分を Try...Catch...Finally ステートメント (Visual Basic) で囲むことです。このようにすると、ランタイム エラーを処理することができます。

文字配列

単一の **Char** と **Char** 要素の配列は、どちらも **String** に拡張されます。しかし、**String** は **Char()** に拡張されません。**String** 値を **Char** 配列に変換するには、System.String クラスの ToCharArray メソッドを使用します。

無意味な値

一般に、**String** 値は他のデータ型にとって無意味な値です。変換は不自然であり、危険です。**String** 変数は、できるだけ本来の文字の羅列だけに使うようにしてください。他の型でこれに相当する値に依存するコードは書かないようにしてください。

参照

関連項目

データ型の概要 (Visual Basic)
データ型変換関数

概念

Visual Basic におけるデータ型
型文字
Visual Basic での型宣言を省略したプログラミング
.NET Framework 型のデータ型
データ型の有効な使用方法

その他の技術情報

データ型の実装
Visual Basic における型変換

Visual Basic における変数

Visual Basic で計算を実行するときに、しばしば値を格納する必要が生じます。たとえば、複数の値を計算、比較し、比較の結果に応じて異なる処理を実行する場合があります。値を比較するためは、値を保持する必要があります。

使用法

Visual Basic では、ほとんどのプログラミング言語と同様に、変数を使って値を格納します。変数は、変数に格納されている値を参照するときに使用する "名前" と、変数に格納できるデータの種類を決定する "データ型" を持っています。密接な関係を持つ一連のデータ項目にインデックスを付けて格納する必要がある場合は、変数を配列として使用することもできます。

値の代入

計算を実行して結果を変数に代入するには、次の例のように、代入ステートメントを使用します。

VB

```
' The following statement assigns the value 10 to the variable.
applesSold = 10
' The following statement increments the variable.
applesSold = applesSold + 1
' The variable now holds the value 11.
```

メモ :

この例で使用されている等号 (=) は、等値演算子ではなく代入演算子です。値は変数 `applesSold` に代入されます。

詳細については、「方法 : 変数に値を格納する、および変数から値を取得する」を参照してください。

変数とプロパティ

変数と同様、プロパティもユーザーがアクセス可能な値を表します。ただし、プロパティは変数よりも複雑です。プロパティはコード ブロックを使用して、プロパティの値を設定および取得する方法を制御します。詳細については、「Visual Basic のプロパティと変数の違い」を参照してください。

参照

処理手順

Visual Basic における変数のトラブルシューティング

方法 : 変数に値を格納する、および変数から値を取得する

概念

Visual Basic での変数宣言

Visual Basic におけるオブジェクト変数

Visual Basic のプロパティと変数の違い

Visual Basic での変数宣言

変数を宣言して、変数の名前と特性を指定します。変数の宣言ステートメントは、Dim ステートメント (Visual Basic) です。このステートメントの場所と内容により、変数の特性が決まります。

変数の名前付け規則および考慮事項については、「宣言された要素の名前」を参照してください。

宣言のレベル

ローカル変数およびメンバ変数

プロシージャ内で宣言された変数をローカル変数と呼びます。メンバ変数は、Visual Basic 型のメンバです。これは、クラス、構造体、またはモジュールの内部のプロシージャ内ではなく、クラス、構造体、またはモジュール内のモジュール レベルで宣言されます。

共有変数およびインスタンス変数

クラスや構造体では、メンバ変数の変数のカテゴリは、その変数が共有されているかどうかによって決まります。共有キーワードを使って宣言した変数は共有変数となり、クラスや構造体のすべてのインスタンスが 1 つのコピーを共有します。

それ以外の変数はインスタンス変数となり、クラスや構造体の各インスタンスごとにコピーが作成されます。インスタンス変数のコピーは、作成されたインスタンス以外では使用できません。これは、他のインスタンスのコピーから独立しています。

データ型の宣言

宣言ステートメントで As 句を使用すると、宣言する変数のデータ型またはオブジェクトの型を定義できます。変数に指定できる型は次のとおりです。

- **Boolean**、**Long**、**Decimal** などの基本データ型
- 配列や構造体などの複合データ型
- 作成したアプリケーションまたはその他のアプリケーションのいずれかで定義された、オブジェクト型またはクラス
- Label または TextBox などの .NET Framework クラス
- IComparable または IDisposable などのインターフェイス型

1 つのステートメントで複数の変数を宣言できます。この場合、データ型を繰り返す必要はありません。次のステートメントでは、変数 `i`、`j`、および `k` は **Integer** 型として、`l` と `m` は **Long** 型として、`x` と `y` は **Single** 型として、それぞれ宣言されています。

```
Dim i, j, k As Integer
' All three variables in the preceding statement are declared as Integer.
Dim l, m As Long, x, y As Single
' In the preceding statement, l and m are Long, x and y are Single.
```

データ型の詳細については、「Visual Basic におけるデータ型」を参照してください。オブジェクトの詳細については、「Visual Basic におけるオブジェクト指向プログラミング」および「コンポーネントによるプログラミング」を参照してください。

特性の宣言

変数の有効期間とは、その変数を使用できる期間です。一般的には、(プロシージャまたはクラスなどの) 変数が宣言された要素が存在し続ける限り、変数は存在します。変数の有効期間を延長することが可能な場合もあります。詳細については、「Visual Basic における有効期間」を参照してください。

変数のスコープとは、名前に修飾子を付けずにその変数を参照できるコードの範囲です。変数のスコープは、変数を宣言した位置で決まります。変数を定義した領域にあるコードでは、名前に修飾子を付けずにその変数を使用できます。詳細については、「Visual Basic におけるスコープ」を参照してください。

変数のアクセス レベルは、変数へのアクセス許可を持ったコードのエクステントです。これは、**Dim** ステートメントで使用する、(Public (Visual Basic) または Private (Visual Basic) などの) アクセス修飾子で決まります。詳細については、「Visual Basic でのアクセス レベル」を参照してください。

参照

処理手順

方法 : 新しい変数を作成する
方法 : 値の変わらない変数を作成する
方法 : 変数に値を格納する、および変数から値を取得する

関連項目
データ型の概要 (Visual Basic)
Protected (Visual Basic)
Friend (Visual Basic)
Static (Visual Basic)
概念
定義する変数の型の決定
宣言された要素の特性

方法：新しい変数を作成する

変数を作成する場合は、Dim ステートメント (Visual Basic) を使用します。

新しい変数を作成するには

1. **Dim** ステートメントで変数を宣言します。

```
Dim newCustomer
```

2. Private (Visual Basic)、Static (Visual Basic)、Shadows、WithEvents など変数の特性を指定します。詳細については、「宣言された要素の特性」を参照してください。

```
Public Static newCustomer
```

   宣言で他のキーワードを使用する場合は、**Dim** キーワードは必要ありません。

3. 変数名の規則に従います。この規則は、Visual Basic の規則および規約に従う必要があります。詳細については、「宣言された要素の名前」を参照してください。

```
Public Static newCustomer
```

4. 名前の後に As 句を付けて、変数のデータ型を指定します。

```
Public Static newCustomer As Customer
```

   データ型を指定しなかった場合は、既定で **Object** 型になります。

5. **As** 句の後に等号 (=) を指定し、等号の後に変数の初期値を指定します。

   Visual Basic では、**Dim** ステートメントが実行するたびに、指定した値が変数に代入されます。初期値を指定しなかった場合、**Dim** ステートメントを含むコードが最初に実行されるとき、変数のデータ型に応じた既定の初期値が Visual Basic によって代入されます。

   変数が参照型の場合は、**As** 句に New (Visual Basic) キーワードを含めることによって、クラスのインスタンスを作成できます。**New** を指定しない場合、変数の初期値は Nothing (Visual Basic) となります。

```
Public Static newCustomer As New Customer
```

参照

概念

Visual Basic における変数
Visual Basic での変数宣言
宣言された要素の名前
宣言された要素の特性
値型と参照型
Visual Basic の宣言ステートメント

定義する変数の型の決定

変数を定義する際には、次の特性を決定する必要があります。

- データ型 - 変数に保持するデータの種類。
- 有効期間 - 変数が存続する期間。
- スコープ - 名前を修飾しないで変数を参照できるコード範囲。
- アクセス レベル - 変数に対する読み取りおよび書き込みの権限をどのコードに与えるか。

データ型

変数を宣言する Dim ステートメント (Visual Basic) には、適切なデータ型 (**Integer**、**String** など) を指定する As 句を含めます。変数のデータ型を選択する際には、次に示すページが役立ちます。

- 方法 : 変数に整数を保持する
- 方法 : 変数に可能な最大値を保持する
- 方法 : unsigned 型を使用して正の整数のストレージを最適化する
- 方法 : 変数で小数桁を保持する
- 方法 : 変数で最大有効桁数を保持する
- 方法 : 変数で通貨値を保持する
- 方法 : 変数内に文字を保持する
- 方法 : True および False の値を変数に保持する
- 方法 : 日付および時刻の値を変数に保持する
- 方法 : 型が不明なデータを変数に保持する
- 方法 : 構造体を宣言する
- オブジェクトの作成と使用

詳細については、「データ型の概要 (Visual Basic)」を参照してください。

有効期間

変数の有効期間については、変数を宣言するモジュール、クラス、またはプロシージャが存在しなくなる時点で変数も失われてよいかどうかを決定することが重要です。

コンテナ要素の有効期間よりも長く変数を存続させる必要がない場合、特に必要な対処はありません。コンテナ 要素の有効期間よりも長く変数を存続させる必要がある場合は、**Dim** ステートメントに **Static** キーワード、または **Shared** キーワードを含めます。
「方法 : 変数の有効期間を延長する」の手順に従ってください。

詳細については、「Visual Basic における有効期間」を参照してください。

スコープ

変数のスコープは通常、宣言空間、つまり、変数の宣言が含まれているコンテナ要素と同じです。変数のスコープの広さを決定することは常に必要です。

Dim ステートメントは、ブロック、プロシージャ、またはモジュール レベルなど、必ず適切なレベルに記述してください。
「方法 : 変数のスコープを制御する」の手順に従ってください。

詳細については、「Visual Basic におけるスコープ」を参照してください。

アクセス レベル

すべての変数には、宣言されている場所 (つまり、変数がどのような種類のコンテナ要素内に含まれているか) によって決定される既定のアクセス レベルがあります。

既定以外のアクセス レベルを指定する必要がある場合は、アクセス修飾子 (**Protected**、**Private** など) を、変数の **Dim** ステートメントに含

めます。これが可能なのはメンバ変数 (つまり、プロシージャの外側で宣言された変数) に対してだけです。「方法 : 変数の可用性を制御する」の手順に従ってください。

詳細については、「Visual Basic でのアクセス レベル」を参照してください。

参照

処理手順

方法 : 変数の有効期間を延長する
方法 : 変数のスコープを制御する

関連項目

As (Visual Basic)

概念

Visual Basic での変数宣言
宣言された要素の特性
Visual Basic におけるスコープ
Visual Basic でのアクセス レベル

方法：値の変わらない変数を作成する

値の変わらない変数という概念は、一見矛盾しているように見えます。しかし、場合によっては定数を使用できないこともあり、そのようなときに固定値を持つ変数を使用すると便利です。このような場合に、ReadOnly (Visual Basic) キーワードを使用してメンバ変数を定義できます。

次のような場合は、Const ステートメント (Visual Basic) を使用して定数を宣言し、値を割り当てることはできません。

- **Const** ステートメントが、使用するデータ型を受け付けない。
- コンパイル時には値がわからない。
- コンパイル時に定数値を計算できない。

値の変わらない変数を作成するには

1. モジュール レベルで、Dim ステートメント (Visual Basic) を指定してメンバ変数を宣言し、ReadOnly (Visual Basic) キーワードを含めます。

   ```
   Dim ReadOnly timeStarted
   ```

   メンバ変数に対してのみ **ReadOnly** を指定します。これは、プロシージャ外部で、モジュール レベルの変数を定義することが必要であることを意味します。

2. コンパイル時に単一のステートメントで値を計算できる場合は、**Dim** ステートメントで初期化の句を指定します。As 句の後ろに等号 (=) を付け、式を続けます。コンパイラでこの式を定数値に評価できることを確認します。

   ```
   Dim ReadOnly timeStarted As Date = Now
   ```

   ReadOnly 変数に値を割り当てることができるのは 1 回だけです。この値を割り当てた後は、いずれのコードもこの値を変更することはできません。

   コンパイル時に値がわからない、またはコンパイル時に単一ステートメントではこれを計算できない場合は、実行時にコンストラクタ内で値を割り当てることができます。実行時に値を割り当てるには、クラス レベルまたは構造体レベルで **ReadOnly** 変数を宣言する必要があります。そのクラスまたは構造体のコンストラクタ内では、変数の固定値を計算し、コンストラクタから返される前にその値を変数に割り当てます。

参照

関連項目

WriteOnly
Const ステートメント (Visual Basic)

方法：変数に値を格納する、および変数から値を取得する

変数にデータを格納するには、代入ステートメントの左側に変数名を置きます。

変数にデータを代入する

変数に値を格納するには

- 代入ステートメントの左側に変数名を指定します。

  次の例は、変数 `alpha` に値を設定します。

  ```
  alpha = (beta * 6.27) / (gamma + 2.1)
  ```

  代入ステートメントの右側で生成された値が、変数に格納されます。

変数からデータを取得する

変数の値を取得するには、式に変数名を含めます。

変数から値を取得するには

- 式の中で変数名を使用します。定数またはリテラルが使用できる場所ならどこでも変数を使用できます。ただし、定数の値を定義するための式の中を除きます。

  または

- 代入ステートメントの等号 (=) 記号の後ろに変数名を使用します。

  次の例では、変数 `startValue` の値を読み込み、次に変数 `counter` の値を式の中で使用します。

  ```
  counter = startValue
  cellValue = (counter + 5) ^ 2
  ```

  変数の値は、定数を指定した場合と同じように式の中で使用されます。式の結果は、代入ステートメントの左側にある変数またはプロパティに格納されます。

参照

概念

Visual Basic における変数
Visual Basic での変数宣言
Visual Basic におけるオブジェクト変数

Visual Basic におけるオブジェクト変数

変数では、直接に値を格納する他に、オブジェクトを参照することもできます。変数に値を代入する場合と同様に、変数にオブジェクトへの参照を代入すると、次のような利点が得られます。

- 変数名は、多くの場合、オブジェクト自体にアクセスする場合に必要なメソッドやプロパティへの完全パスに比べて短く覚えやすいものになります。
- メソッドやプロパティを通じてオブジェクト自体に何度もアクセスするより、オブジェクトを参照する変数を使う方が効率的です。
- 変数は、コードの実行中に、他のオブジェクトを参照するように変更できます。

コードを短くする

オブジェクト変数を使用すると、コードを短くできます。次の例では、メソッドおよびプロパティの完全パスを使用して Control オブジェクトにアクセスします。

```
' Assume Me is a valid Form, or replace Me with a valid Form.
Me.ActiveForm.ActiveControl.Text = "Test"
Me.ActiveForm.ActiveControl.Location = New Point(100, 100)
Me.ActiveForm.ActiveControl.Show()
```

この場合、コントロールのオブジェクト変数を使用すると、コードが短くなり、実行時間を短縮できます。オブジェクト変数を宣言する場合、変数に代入する特定のクラスを指定する必要があります (この場合は `Control`)。変数にオブジェクトを代入した後は、変数を参照先のオブジェクトと同じように扱うことができます。オブジェクトのプロパティを設定または取得したり、オブジェクトのメソッドを使用したりできます。次の例では、オブジェクト変数を使用して、上の例のコードを単純化しています。

```
Dim ctrlActv As System.Windows.Forms.Control = Me.ActiveForm.ActiveControl
ctrlActv.Text = "Test"
ctrlActv.Location = New Point(100, 100)
ctrlActv.Show()
```

参照

処理手順

方法 : 長い修飾パスをもつオブジェクトへのアクセス時間を短縮する

概念

Visual Basic での変数宣言
オブジェクト変数の宣言
オブジェクト変数の代入
オブジェクト変数の値

方法：長い修飾パスをもつオブジェクトへのアクセス時間を短縮する

複数のメソッドおよびプロパティの修飾パスを必要とするオブジェクトに頻繁にアクセスする場合、修飾パスを繰り返さないことによりコーディングの時間を短縮できます。

修飾パスの繰り返しを回避するには、2 つの方法があります。変数にオブジェクトを割り当てるか、またはオブジェクトを **With**...**End With** ブロックで使用します。

過度に修飾されたオブジェクトを変数に割り当ててそれに対するアクセス時間を短縮するには

1. 頻繁にアクセスするオブジェクトの型の変数を宣言します。宣言の初期化の部分で修飾パスを指定します。

```
Dim ctrlActv As Control = someForm.ActiveForm.ActiveControl
```

2. 変数を使用してオブジェクトのメンバにアクセスします。

```
ctrlActv.Text = "Test"
ctrlActv.Location = New Point(100, 100)
ctrlActv.Show()
```

With...End With ブロックを使用して、過度に修飾されたオブジェクトへのアクセス時間を短縮するには

1. **With** ステートメントに修飾パスを挿入します。

```
With someForm.ActiveForm.ActiveControl
```

2. **End With** ステートメントの前の部分で、**With** ブロックの内でオブジェクトのメンバにアクセスします。

```
    .Text = "Test"
    .Location = New Point(100, 100)
    .Show()
End With
```

参照

関連項目

With...End With ステートメント (Visual Basic)

概念

Visual Basic におけるオブジェクト変数

オブジェクト変数の宣言

オブジェクト変数を宣言する場合は、通常の宣言ステートメントを使用します。データ型については、**Object** (つまり、オブジェクト型 (Object)) か、オブジェクトの作成元となる、より具体的なクラスのいずれかを指定します。

変数を **Object** として宣言することは、System.Object として宣言するのと同じことです。

特定のオブジェクト クラスを使って宣言した変数は、そのクラスおよびそのクラスが継承している各クラスによって公開されている、すべてのメソッドとプロパティにアクセスできます。**Object** を指定して変数を宣言した場合、**Option Strict Off** を指定して遅延バインディングを有効にしていない限り、**Object** クラスのメンバにのみアクセスできます。

宣言の構文

オブジェクト変数を宣言するには、次の構文を使用します。

```
Dim variablename As [New] { objectclass | Object }
```

また、宣言には Public (Visual Basic)、Protected (Visual Basic)、Friend (Visual Basic)、**Protected Friend**、Private (Visual Basic)、Shared (Visual Basic)、または Static (Visual Basic) も指定できます。たとえば、次の例のような宣言も有効です。

```
Private objA As Object
Static objB As System.Windows.Forms.Label
Dim objC As System.OperatingSystem
```

遅延バインディングと事前バインディング

場合によっては、コードを実行するまで具体的なクラスがわからないことがあります。このような場合は、オブジェクト型 (**Object**) を使ってオブジェクト変数を宣言する必要があります。これによって任意の型のオブジェクトに対応する汎用の参照が作成され、実行時に具体的なクラスが割り当てられます。これは、遅延バインディングと呼ばれます。遅延バインディングを行う場合、実行に必要な時間が長くなります。また、遅延バインディングでは、直近に代入したクラスのメソッドおよびプロパティにしかコードからアクセスできません。そのため、別のクラスのメンバにアクセスしようとするコードがあると、実行時エラーが発生する可能性があります。

コンパイル時に具体的なクラスがわかっている場合は、そのクラスを指定してオブジェクト変数を宣言します。これは、事前バインディングと呼ばれます。事前バインディングでは、パフォーマンスが向上し、コードから特定のクラスのすべてのメソッドおよびプロパティに確実にアクセスできます。上の例の宣言では、変数 `objA` がクラス System.Windows.Forms.Label のオブジェクトだけを使用する場合は、宣言で `As System.Windows.Forms.Label` を指定する必要があります。

事前バインディングの利点

オブジェクト変数を特定のクラスとして宣言することには、次のようにいくつかの利点があります。

- 自動的な型チェック
- 特定のクラスのすべてのメンバに対するアクセスの保証
- コード エディタにおける Microsoft IntelliSense のサポート
- コードの読みやすさの向上
- コードのエラーの減少
- 実行時ではなくコンパイル時のエラー検出
- コードの実行の高速化

オブジェクト変数のメンバへのアクセス

Option Strict が **On** になっている場合、オブジェクト変数がアクセスできるのは、その変数の宣言時に指定したクラスのメソッドとプロパティだけです。次に例を示します。

```
' Option statements must precede all other source file lines.
Option Strict On
' Imports statement must precede all declarations in the source file.
Imports System.Windows.Forms
```

```
Public Sub accessMembers()
    Dim p As Object
    Dim q As System.Windows.Forms.Label
    p = New System.Windows.Forms.Label
    q = New System.Windows.Forms.Label
    Dim j, k As Integer
    ' The following statement generates a compiler ERROR.
    j = p.Left
    ' The following statement retrieves the left edge of the label in pixels.
    k = q.Left
End Sub
```

この例の場合、`p` で使用できるのは **Object** クラス自身のメンバだけです。`Left` プロパティは含まれません。一方、`q` は、**Label** 型として宣言されているので、System.Windows.Forms 名前空間の **Label** クラスのすべてのメソッドとプロパティを使用できます。

オブジェクト変数の柔軟性

継承階層のオブジェクトを使用する場合に、どのクラスを使ってオブジェクト変数を宣言するかを選択できます。クラスを選択するときに、オブジェクトを代入する場合の柔軟性とクラスのメンバへのアクセスとのバランスを考慮する必要があります。たとえば、System.Windows.Forms.Form クラスに至る継承階層は次のようになります。

Object

Component

Control

ScrollableControl

ContainerControl

Form

たとえば、**Form** クラスを継承する `specialForm` というフォーム クラスをアプリケーションで定義するとします。次の例のように宣言すると、`specialForm` だけを参照するオブジェクト変数を宣言できます。

```
Public Class specialForm
    Inherits System.Windows.Forms.Form
    ' Insert code defining methods and properties of specialForm.
End Class
Dim nextForm As New specialForm
```

上の例の宣言では、変数 `nextForm` を `specialForm` クラスのオブジェクトに制限していますが、`specialForm` のすべてのメソッドとプロパティに加え、`specialForm` が継承しているすべてのクラスのすべてのメンバも `nextForm` から使用できます。

次の例に示すように、オブジェクト変数を **Form** 型として宣言すると、汎用性が高くなります。

```
Dim anyForm As System.Windows.Forms.Form
```

上の例の宣言では、アプリケーション内の任意のフォームを `anyForm` に代入できます。ただし、`anyForm` は **Form** クラスのすべてのメンバにアクセスできますが、`specialForm` など特定のフォーム用に定義された追加のメソッドやプロパティは使用できません。

基本クラスのメンバはすべて派生クラスで使用できますが、派生クラスで追加されたメンバを基本クラスで使用することはできません。

参照

処理手順

方法 : Visual Basic でオブジェクト変数を宣言し、オブジェクト変数にオブジェクトを代入する

方法 : オブジェクトのメンバにアクセスする

関連項目

New (Visual Basic)

Option Strict ステートメント

概念

Visual Basic におけるオブジェクト変数

オブジェクト変数の代入

オブジェクト変数の値

Visual Basic での型宣言を省略したプログラミング

方法：オブジェクトのメンバにアクセスする

オブジェクトを参照するオブジェクト変数がある場合、メソッド、プロパティ、フィールド、およびイベントなどのオブジェクトのメンバで作業を行うことができます。たとえば、新しい Form オブジェクトを作成した後、Text プロパティを設定したり、または Focus メソッドを呼び出すことができます。

メンバへのアクセス

オブジェクトのメンバを参照する変数を使用して、オブジェクトのメンバにアクセスします。

オブジェクトのメンバにアクセスするには

- オブジェクト変数名とメンバ名との間で、メンバ アクセス演算子 (.) を使用します。

```
currentText = newForm.Text
```

メンバが Shared (Visual Basic) である場合、アクセスするための変数は必要ありません。詳細については、「方法 : オブジェクトの共有メンバおよび非共有メンバにアクセスする」を参照してください。

既知の型のオブジェクトのメンバへのアクセス

コンパイル時にオブジェクトの型がわかっている場合、それを参照している変数に対して、事前バインディングを使用できます。

コンパイル時に型がわかっているオブジェクトのメンバにアクセスするには

1. オブジェクト変数を、変数に割り当てようとしているオブジェクトの型として宣言します。

```
Dim extraForm As System.Windows.Forms.Form
```

Option Strict On を使用すると、**Form** オブジェクト (または **Form** から派生した型のオブジェクト) だけを、`extraForm` に割り当てることができます。**Form** への **CType** の拡大変換を使用して、クラスまたは構造体が定義されている場合、そのクラスまたは構造体を `extraForm` に割り当てることもできます。

2. オブジェクト変数名とメンバ名との間で、メンバ アクセス演算子 (.) を使用します。

```
extraForm.Show()
```

Option Strict がどのような設定になっていても、**Form** クラス固有のすべてのメソッドおよびプロパティにアクセスできます。

不明な型のオブジェクトのメンバへのアクセス

コンパイル時にオブジェクトの型がわからない場合、それを参照しているすべての変数に対して、遅延バインディングを使用する必要があります。

コンパイル時に型がわかっていないオブジェクトのメンバにアクセスするには

1. オブジェクト変数を オブジェクト型 (Object) として宣言します。(変数を **Object** として宣言すると、System.Object として宣言することと同等になります。)

```
Dim someControl As Object
```

Option Strict On を使用すると、**Object** クラスで定義されたメンバだけにアクセスできます。

2. オブジェクト変数名とメンバ名との間で、メンバ アクセス演算子 (.) を使用します。

```
someControl.GetType()
```

オブジェクト変数に割り当てたオブジェクトのメンバにアクセスできるようにするには、**Option Strict Off** を設定する必要があります。これを行うと、変数に割り当てたオブジェクトによって、指定されたメンバが公開されるかどうかをコンパイラは保証できません。アクセスしようとしたメンバをオブジェクトが公開しない場合、MemberAccessException 例外が発生します。

参照

関連項目

オブジェクト型 (Object)
Option Strict ステートメント
Object
Form
MemberAccessException

概念

Visual Basic におけるオブジェクト変数
オブジェクト変数の宣言

オブジェクト変数の代入

オブジェクト変数にオブジェクトを代入するには、通常の代入ステートメントを使用します。次に示す例のように、オブジェクト式または Nothing キーワードを代入できます。

```
Dim thisObject As Object
' The following statement assigns an object reference.
thisObject = Form1
' The following statement discontinues association with any object.
thisObject = Nothing
```

Nothing は、変数に現在代入されているオブジェクトがないことを表します。

初期化

コードが実行を開始すると、オブジェクト変数は **Nothing** に初期化されます。宣言に初期化が含まれている場合は、この宣言ステートメントが実行されるたびに、指定した値に再初期化されます。

New キーワードを使用すると、宣言に初期化を含めることができます。次に示す宣言ステートメントは、オブジェクト変数 `testUri` と `ver` を宣言し、特定のオブジェクトを代入します。それぞれ適切なクラスのオーバーロードされたコンストラクタのいずれか 1 つを使用してオブジェクトを初期化します。

```
Dim testUri As New System.Uri("http://www.microsoft.com")
Dim ver As New System.Version(6, 1, 0)
```

関連付けの解除

オブジェクト変数を **Nothing** に設定すると、変数と特定のオブジェクトとの関連付けは失われます。このため、変数を変更するときに誤ってオブジェクトを変更してしまうことがなくなります。また、次の例のように、オブジェクト変数が有効なオブジェクトを指しているかどうかをテストすることもできます。

```
If otherObject IsNot Nothing Then
    ' otherObject refers to a valid object, so your code can use it.
End If
```

変数が他のアプリケーションのオブジェクトを参照している場合は、アプリケーションが終了されているのか、単にオブジェクトが無効になっているのかをこのテストで調べることはできません。

値が **Nothing** のオブジェクト変数は、null 参照とも呼ばれます。

現在のインスタンス

オブジェクトの現在のインスタンスとは、その内部で現在コードが実行されているインスタンスです。すべてのコードはプロシージャ内で実行されるため、現在のインスタンスは、呼び出されたプロシージャでのインスタンスになります。

Me キーワードは、現在のインスタンスを参照するオブジェクト変数として機能します。プロシージャが Shared (Visual Basic) でない場合は、**Me** キーワードを使って現在のインスタンスへのポインタを取得できます。共有プロシージャをクラスの特定のインスタンスに関連付けることはできません。

Me キーワードは、他のモジュールのプロシージャに現在のインスタンスを渡す場合に特に便利です。たとえば、多数の XML ドキュメントがあり、標準テキストをそれらすべてに追加するとします。これを実行するプロシージャを定義する例を次に示します。

```
Sub addStandardText(XmlDoc As System.Xml.XmlDocument)
    XmlDoc.CreateTextNode("This text goes into every XML document.")
End Sub
```

次に、この定義したプロシージャをすべての XML ドキュメント オブジェクトで呼び出し、引数として現在のインスタンスを渡します。次にコード例を示します。

```
addStandardText(Me)
```

参照

処理手順

方法 : Visual Basic でオブジェクト変数を宣言し、オブジェクト変数にオブジェクトを代入する
方法 : オブジェクト変数がインスタンスを参照しないようにする

概念

Visual Basic におけるオブジェクト変数
オブジェクト変数の宣言
オブジェクト変数の値

方法：Visual Basic でオブジェクト変数を宣言し、オブジェクト変数にオブジェクトを代入する

オブジェクト型 (Object) の変数は、Dim ステートメント (Visual Basic) で **As Object** を指定して宣言します。このような変数にオブジェクトを代入するには、代入ステートメントまたは初期化処理句で等号 (=) の右側にオブジェクトを配置します。

使用例

次の例では、**Object** 変数を宣言し、現在のインスタンスを代入します。

```
      Dim thisObject As Object
thisObject = Me
```

宣言の一部として変数を初期化することによって、宣言と代入を結合できます。次の例は、上の例と同じ意味です。

```
Dim thisObject As Object = Me
```

コードのコンパイル方法

必要な条件は次のとおりです。

- System 名前空間への参照
- **Dim** ステートメントを記述するクラス、構造、またはモジュール
- 代入ステートメントを記述するプロシージャ

参照

関連項目

オブジェクト型 (Object)
Me
Dim ステートメント (Visual Basic)

概念

Visual Basic での変数宣言
Visual Basic におけるオブジェクト変数
オブジェクト変数の宣言

方法：オブジェクト変数がインスタンスを参照しないようにする

Nothing (Visual Basic) に設定して、オブジェクト インスタンスから、オブジェクト変数の関連付けを解除できます。

オブジェクト インスタンスからオブジェクト変数の関連付けを解除するには

- 代入ステートメントで、変数を **Nothing** に設定します。

```
' Assume account is a defined class
Dim currentAccount As account
currentAccount = Nothing
```

堅牢性の高いプログラム

Nothing に設定されたオブジェクト変数のメンバに、コードがアクセスしようとしている場合、NullReferenceException が発生します。オブジェクト変数を頻繁に **Nothing** に設定する場合、または変数を初期化しないことが可能である場合、**Try...Catch...Finally** ブロックにメンバ アクセスを入れることをお勧めします。

セキュリティ

重要情報が含まれているオブジェクトのオブジェクト変数を使用する場合、これらのオブジェクトをアクティブに処理していないときに、変数を **Nothing** に設定できます。これにより、悪意のあるコードがデータにアクセスできる機会を減らすことができます。

参照

関連項目

Nothing (Visual Basic)
Try...Catch...Finally ステートメント (Visual Basic)
例外のトラブルシューティング : System.NullReferenceException
NullReferenceException

概念

Visual Basic におけるオブジェクト変数
オブジェクト変数の代入

オブジェクト変数の値

オブジェクト型 (Object) の変数は、任意の型のデータを参照できます。**Object** 変数に格納した値はメモリ内の他の場所に保持され、変数自体はデータへのポインタを保持します。

オブジェクト分類関数

Visual Basic には、**Object** 変数の参照先に関する情報を返す次の表のような関数が用意されています。

関数	オブジェクト変数の参照先 (一致する場合に True を返す)
IsArray 関数 (Visual Basic)	1 つの値ではなく、値の配列
IsDate 関数 (Visual Basic)	日付型 (Date) (Visual Basic) の値、または日付と時刻の値として解釈できる文字列
IsDBNull 関数	データが不足しているか存在しないことを表す DBNull 型のオブジェクト
IsError 関数	Exception から派生した例外オブジェクト
IsNothing 関数	変数に現在代入されているオブジェクトがないことを表す Nothing (Visual Basic)
IsNumeric 関数 (Visual Basic)	数値、または数値として解釈できる文字列
IsReference 関数	参照型 (文字列、配列、デリゲート、クラス型など)

これらの関数を使用すると、演算やプロシージャに無効な値が送られるのを防ぐことができます。

TypeOf 演算子

また、TypeOf 演算子 (Visual Basic) を使って、オブジェクト変数が現在特定のデータ型を参照しているかどうかを調べることもできます。**TypeOf**...**Is** 式の評価は、オペランドの実行時の型が、指定した型から派生している場合または指定した型を実装している場合に **True** になります。

次の例では、値型を参照しているオブジェクト変数と参照型を参照しているオブジェクト変数に対して **TypeOf** を使用しています。

```
' The following statement puts a value type (Integer) in an Object variable.
Dim num As Object = 10
' The following statement puts a reference type (Form) in an Object variable.
Dim frm As Object = New Form()
If TypeOf num Is Long Then Debug.WriteLine("num is Long")
If TypeOf num Is Integer Then Debug.WriteLine("num is Integer")
If TypeOf num Is Short Then Debug.WriteLine("num is Short")
If TypeOf num Is Object Then Debug.WriteLine("num is Object")
If TypeOf frm Is Form Then Debug.WriteLine("frm is Form")
If TypeOf frm Is Label Then Debug.WriteLine("frm is Label")
If TypeOf frm Is Object Then Debug.WriteLine("frm is Object")
```

上記の例では、以下の行がデバッグ ウィンドウに書き込まれます。

```
num is Integer
num is Object
frm is Form
frm is Object
```

オブジェクト変数 `num` が参照しているのは **Integer** 型のデータであり、`frm` が参照しているのは Form クラスのオブジェクトです。

オブジェクト配列

Object 変数の配列を宣言して使用することもできます。こうすると、さまざまなデータ型やオブジェクト クラスを扱う必要がある場合に便利です。配列のすべての要素には、同じデータ型が宣言されている必要があります。このデータ型を **Object** として宣言すると、その他のデータ型と共に、オブジェクトやクラス インスタンスも配列に格納できるようになります。

参照

処理手順

方法 : オブジェクトの現在のインスタンスを参照する
方法 : オブジェクト変数で参照している型を確認する
方法 : 2 つのオブジェクトが関連しているかどうかを決める
方法 : 2 つのオブジェクトが同一であるかどうか判別する

関連項目

Is (Visual Basic)

概念

Visual Basic におけるオブジェクト変数
オブジェクト変数の宣言
オブジェクト変数の代入
Visual Basic におけるデータ型

方法：オブジェクトの現在のインスタンスを参照する

オブジェクトの現在のインスタンスとは、現在コードが実行されているインスタンスです。

現在のインスタンスを参照する場合には、Me キーワードを使用します。

現在のインスタンスを参照するには

- 通常はオブジェクト変数の名前を使用する箇所で、**Me** キーワードを使用します。

  ```
  Me.ForeColor = System.Drawing.Color.Crimson
  Me.Close()
  ```

  Me はオブジェクト変数のように動作しますが、このキーワードは宣言できず、何も代入できません。**Me** は、常に現在のインスタンスを参照します。

参照

関連項目

Me

概念

Visual Basic におけるオブジェクト変数

オブジェクト変数の代入

方法：オブジェクト変数で参照している型を確認する

オブジェクト変数には、別の場所に格納されているデータへのポインタが含まれています。データの型は、実行時に変わる可能性があります。任意のタイミングで、GetTypeCode メソッドを使用して、現在の実行時型を確認することや、TypeOf 演算子 (Visual Basic) を使用して、現在の実行時型と指定の型に互換性があるかどうかを確認できます。

オブジェクト変数で現在参照している正確な型を確認するには

1. オブジェクト変数で、GetType メソッドを呼び出して、System.Type オブジェクトを取得します。

```
Dim myObject As Object
myObject.GetType()
```

2. **System.Type** クラスで、共有メソッド **GetTypeCode** を呼び出してそのオブジェクト型の TypeCode 列挙値を取得します。

```
Dim myObject As Object
Dim datTyp As Integer = Type.GetTypeCode(myObject.GetType())
MsgBox("myObject currently has type code " & CStr(datTyp))
```

Double など、**TypeCode** の列挙値と突き合わせると、取得した列挙型のメンバがどの列挙値なのかを確認できます。

オブジェクト変数の型と指定の型に互換性があるかどうかを確認するには

- Is 演算子 (Visual Basic) と組み合わせて **TypeOf** 演算子を使用し、**TypeOf**...**Is** 式を用いてオブジェクトをテストします。

```
If TypeOf objA Is System.Windows.Forms.Control Then
    MsgBox("objA is compatible with the Control class")
End If
```

TypeOf...**Is** 式は、オブジェクトの実行時型と指定の型に互換性がある場合は、**True** を返します。

互換性の基準は、指定の型がクラス、構造体、またはインターフェイスのいずれであるかによって異なります。一般に、オブジェクトの型と指定の型が同じ型である場合、オブジェクトの型が指定の型から派生した型である場合、またはオブジェクトの型が指定した型を実装した型である場合、オブジェクトの型と指定の型には互換性があります。詳細については、「TypeOf 演算子 (Visual Basic)」を参照してください。

コードのコンパイル方法

変数または式を型として指定することはできません。指定する型は、クラス、構造体、インターフェイスなど、定義済みの型の名前である必要があります。これには、**Integer** および **String** などの組み込み型の型も含まれます。

参照

関連項目

オブジェクト型 (Object)
GetType
System.Type
GetTypeCode
TypeCode

概念

Visual Basic におけるオブジェクト変数
オブジェクト変数の値

方法：2 つのオブジェクトが関連しているかどうかを決める

2 つのオブジェクトの各作成元クラス間の関係 (もしあれば) を確認するために、それらのオブジェクトを比較することができます。指定されたクラスが現在のクラスから継承されている場合、または指定されたクラスが現在の型をサポートしている場合、System.Type クラスの IsInstanceOfType メソッドは **True** を返します。

あるオブジェクトが、他のオブジェクトのクラスまたはインターフェイスを継承しているかどうかを決めるには

1. 基本型と思われるオブジェクトで、GetType メソッドを呼び出します。
2. **GetType** によって返された **System.Type** オブジェクトで、**IsInstanceOfType** メソッドを呼び出します。
3. **IsInstanceOfType** の引数リスト内で、派生型であると思われるオブジェクトを指定します。

   引数型が **System.Type** オブジェクト型を継承している場合、**IsInstanceOfType** は **True** を返します。

使用例

次に示すのは、一方のオブジェクトの表すクラスが、もう一方のオブジェクトのクラスから派生したものかどうかを確認する例です。

```
Public Class baseClass
End Class
Public Class derivedClass : Inherits baseClass
End Class
Public Class testTheseClasses
    Public Sub seeIfRelated()
        Dim baseObj As Object = New baseClass()
        Dim derivedObj As Object = New derivedClass()
        Dim related As Boolean
        related = baseObj.GetType().IsInstanceOfType(derivedObj)
        MsgBox(CStr(related))
    End Sub
End Class
```

IsInstanceOfType への呼び出し内にある 2 つのオブジェクト変数が、期待どおり設定されない場合があることに注意してください。この処理では、想定される基本型を使用して **System.Type** クラスが生成され、想定される派生型が引数として **IsInstanceOfType** メソッドに渡されます。

参照

処理手順

方法 : 2 つのオブジェクトが同一であるかどうか判別する

関連項目

オブジェクト型 (Object)
GetType
System.Type
IsInstanceOfType

概念

Visual Basic におけるオブジェクト変数
オブジェクト変数の値

方法 : 2 つのオブジェクトが同一であるかどうか判別する

Visual Basic では、ポインタが同一である場合、つまり双方の変数がメモリ内で同一のクラス インスタンスを指している場合は 2 つのオブジェクトは同一であると見なします。たとえば、現在のインスタンス (Me) が `Form2` などの特定のインスタンスと同一であるかどうかを判別する際に、この比較を用いることができます。

Visual Basic には、ポインタを比較する 2 つの演算子が用意されています。Is 演算子 (Visual Basic) では、オブジェクトが同じである場合 **True** が返されます。IsNot 演算子 では、オブジェクトが異なる場合 **True** が返されます。

2 つのオブジェクトが同じであるかどうかの確認

2 つのオブジェクトが同じであるかどうかを確認するには

1. **Boolean** 式を設定して 2 つのオブジェクトをテストします。
2. テストを実行する式では、2 つのオブジェクトにオペランドとして **Is** 演算子を使用します。

   Is は、両方のオブジェクトが同じクラス インスタンスを指している場合 **True** を返します。

2 つのオブジェクトが同一でないかかどうかの確認

2 つのオブジェクトが同一でないかどうかを確認する場合があります。しかし、`If Not obj1 Is obj2` のように **Not** と **Is** を組み合わせて使用するのは大変面倒なことです。このような場合に、**IsNot** 演算子を使用できます。

2 つのオブジェクトが同一でないかどうかを確認するには

1. **Boolean** 式を設定して 2 つのオブジェクトをテストします。
2. テストを実行する式では、2 つのオブジェクトにオペランドとして **IsNot** 演算子を使用します。

   IsNot は、両方のオブジェクトが同じクラス インスタンスを指していない場合に **True** を返します。

使用例

1 組の **Object** 変数が同じクラス インスタンスを指しているかどうかをテストする例を、次に示します。

```
Dim objA, objB, objC as Object
objA = Me
objB = New System.Windows.Forms.Form()
objC = Me
MsgBox("objA different from objB? " & CStr(objA IsNot objB))
MsgBox("objA identical to objC? " & CStr(objA Is objC))
```

前の例では、次の出力が表示されます。

```
objA different from objB? True

objA identical to objC? True
```

参照

処理手順

方法 : 2 つのオブジェクトが関連しているかどうかを決める

関連項目

オブジェクト型 (Object)
Is 演算子 (Visual Basic)
IsNot 演算子

概念

Visual Basic におけるオブジェクト変数
オブジェクト変数の値

Visual Basic における変数のトラブルシューティング

ここでは、Visual Basic で変数を使用する際に起きる一般的な問題についていくつか説明します。

オブジェクトのメンバにアクセスできない

オブジェクトのプロパティまたはメソッドにアクセスしようとする場合、発生するエラーには 2 つの種類があります。

- オブジェクト変数を特定のデータ型で宣言した後で、その型で定義していないメンバを参照するためにその変数を使うと、コンパイラからエラー メッセージが生成されることがあります。
- オブジェクト変数に代入したオブジェクトが、コードからアクセスしようとしたメンバを公開していない場合、ランタイム MemberAccessException が発生します。オブジェクト型 (Object) の変数の場合、メンバが **Public** でない場合にもこの例外が発生します。原因は、遅延バインディングでアクセスできるのは **Public** メンバだけだからです。

Option Strict ステートメント が **On** になっている場合、オブジェクト変数がアクセスできるのは、その変数の宣言時に指定したクラスのメソッドとプロパティだけです。次に例を示します。

VB

```
Option Strict On
```

VB

```
Dim p As Object = New System.Windows.Forms.Label
Dim q As System.Windows.Forms.Label = New System.Windows.Forms.Label
Dim j, k As Integer
' The following statement generates a compiler error.
j = p.Left
' The following statement retrieves the left edge of the label
' in pixels.
k = q.Left
```

この例の場合、`p` で使用できるのは Object クラス自身のメンバだけです。`Left` プロパティは含まれません。一方、`q` は、Label 型として宣言されているので、System.Windows.Forms 名前空間の **Label** クラスのすべてのメソッドとプロパティを使用できます。

修正方法

特定のクラスのオブジェクトにあるすべてのメンバにアクセスできるようにするには、できる限りオブジェクト変数をそのクラスの型で宣言します。オブジェクト型がコンパイル時にわからないなどの理由で、そうすることができない場合は、**Option Strict** を **Off** に設定し、変数をオブジェクト型 (Object) として宣言します。このようにすると、任意の型のオブジェクトを変数に代入できるようになります。変数に代入しているオブジェクトが受け入れ可能な型であることを確認する必要があります。この確認には、TypeOf 演算子 (Visual Basic) を使用できます。

他のコンポーネントが変数にアクセスできない

Visual Basic では、名前の大文字と小文字は区別されません。2 つの名前の違いが英字の大文字と小文字の違いだけの場合、コンパイラでは 2 つを同じ名前と解釈します。たとえば、`ABC` と `abc` は、宣言された同じ要素を参照していると見なされます。

しかし、共通言語ランタイム (CLR: Common Language Runtime) のバインディングでは大文字と小文字が区別されます。このため、アセンブリまたは DLL を作成し、他のアセンブリで使用できるようにする場合は、大文字と小文字が区別されるようになります。たとえば、`ABC` という名前の要素を持つクラスを定義し、他のアセンブリから共通言語ランタイムを通じてこのクラスを使用する場合は、この要素を `ABC` として参照する必要があります。後でクラスを再コンパイルして要素の名前を `abc` に変更すると、このクラスを使用していた他のアセンブリがこの要素にアクセスできなくなります。したがって、アセンブリを更新してリリースするときには、パブリックな要素の名前の大文字と小文字を変更しないようにする必要があります。

詳細については、「共通言語ランタイム」を参照してください。

修正方法

他のコンポーネントからアクセスできる変数にするには、大文字と小文字が区別されるものとして変数の名前を扱います。作成したクラスまたはモジュールをテストするときには、他のアセンブリが適切な変数にバインドされることを確認します。コンポーネントの公開後は、大文字と小文字の変更を含め、既存の変数名は一切変更しないようにします。

誤った変数が使用される

同じ名前の変数が複数あると、Visual Basic のコンパイラは、その名前への参照を解決しようとします。変数のスコープが異なる場合、コンパイラは、参照を最も狭いスコープの宣言に解決します。変数のスコープが同じ場合、解決は失敗し、コンパイラはエラーを発行します。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」を参照してください。

修正方法

同じ名前で定義が異なる変数を使わないようにします。他のアセンブリまたはプロジェクトを使用している場合、そのような外部コンポーネントで定義された名前を使うことはできるだけ避けてください。同じ名前の変数が複数ある場合、変数への参照に修飾子を付けるようにします。詳細については、「方法 : 同じ名前を持つ 2 つの要素を区別する」を参照してください。

参照

処理手順

方法 : オブジェクトのメンバにアクセスする
方法 : オブジェクト変数で参照している型を確認する

概念

Visual Basic における変数
Visual Basic での変数宣言
Visual Basic におけるオブジェクト変数
オブジェクト変数の宣言
オブジェクト変数の値
同じ名前を持つ複数の変数がある場合に参照を解決する
宣言された要素の名前

Visual Basic における配列

配列を使用すると、複数の値を同じ名前で参照し、それぞれの値をインデックスまたは添字と呼ばれる数値を使用して区別できます。配列を使用すると、コードが短く単純になり、任意の数の要素を効率的に処理するループを作成できます。

このセクションの内容

Visual Basic の配列の概要

関連する用語の定義と配列のサンプルを示します。

Visual Basic における配列の次元

配列のランクと次元について説明します。

Visual Basic における多次元配列

複数の次元で構成される配列について説明します。

Visual Basic におけるジャグ配列

配列の配列 (ジャグ配列とも呼ばれる) について説明します。

Visual Basic における配列データ型

配列のデータ型を決定する要因について説明します。

Visual Basic における配列のサイズ

配列と各次元のサイズに関する考慮事項について説明します。

配列および配列要素の操作

配列の並べ替え、配列値の取得と設定などについて説明します。

配列の代わりとしてのコレクション

コレクションへの項目の格納と、配列への項目の格納の違いについて説明します。

配列のトラブルシューティング

配列を使用しているときに発生する一般的な問題について説明します。

関連するセクション

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic 2005 の新機能の概要を示します。

Visual Basic におけるオブジェクト指向プログラミング

オブジェクト指向プログラミングの基礎について説明します。

Visual Basic の配列の概要

配列は、学校の各学年の生徒の数など、互いに論理的に関連する一連の値です。

配列を使うと、これらの関連する値を同じ名前で参照できます。配列の各要素を区別するには、インデックスまたは添字と呼ばれる番号を使います。個々の値は、配列の要素と呼ばれます。これは、インデックス 0 から最も大きいインデックス値まで連続しています。

例

次の例では、学校の各学年の生徒の数を保持するために、配列変数を宣言しています。

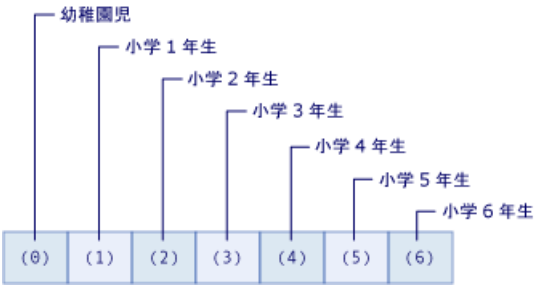
```
Dim students(6) As Integer
```

上の例の `students` 配列には、7 つの要素が含まれています。要素のインデックスの範囲は 0 から 6 までです。7 つの異なる変数を宣言するよりも、この配列の方がより簡単です。

`students` 配列を次の図に示します。配列の各要素は、次のとおりです。

- 要素のインデックスは、学年を表します (インデックス 0 は幼稚園を表します)。
- 要素に含まれている値は、その学年の生徒の数を表します。

"生徒" 配列の要素



次の例は、`students` 配列の先頭の要素、2 番目の要素、および最後の要素を参照する方法を示します。

```
Dim kindergarten As Integer = students(0)
Dim firstGrade As Integer = students(1)
Dim sixthGrade As Integer = students(6)
MsgBox("Students in kindergarten = " & CStr(kindergarten))
MsgBox("Students in first grade = " & CStr(firstGrade))
MsgBox("Students in sixth grade = " & CStr(sixthGrade))
```

インデックスを持たない配列変数名だけを使用して、配列全体を参照できます。

配列の型および他の型

データ型

すべての配列にデータ型がありますが、要素のデータ型とは同じではありません。たとえば、前の例の `students` 配列は **Integer()** 型であり、その要素のそれぞれは **Integer** 型です。**Integer()** の表記は、**Integer** 要素の配列を意味します。詳細については、「Visual Basic における配列データ型」を参照してください。

すべての配列は、System.Array クラスから継承しています。また、**Array** 型として変数を宣言できますが、**Array** 型の配列は作成できません。また、ReDim ステートメント (Visual Basic) は、**Array** 型として宣言された変数上では使用できません。これらの理由やタイプ セーフにより、すべての配列を、前の例の **Integer** などの特定の型として宣言することをお勧めします。

配列の次元

前の例の `students` 配列では、1 つのインデックスが使用されており、これを 1 次元と言います。複数のインデックスまたは添字が使用されている配列は、多次元と呼ばれます。

他の種類の配列とは、要素として他の配列を保持している配列です。これは、配列の配列またはジャグ配列と呼ばれます。ジャグ配列およびその要素は、1 次元または多次元のいずれかになることができます。

対照的な型

配列とは対照的に、変数には、スカラ変数と呼ばれる単一の値が含まれています。

配列は、コレクションとは異なります。詳細については、「配列の代わりとしてのコレクション」を参照してください。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
配列のトラブルシューティング

概念

Visual Basic における配列の次元
Visual Basic における多次元配列
Visual Basic におけるジャグ配列
Visual Basic における配列データ型

その他の技術情報

Visual Basic における配列

方法：配列変数を宣言する

配列変数は、他の変数と同じように **Dim** ステートメントを使用して宣言します。変数名に続けて 1 組以上のかっこを記述することにより、その変数がスカラ (1 つの値を含む変数) ではなく配列を保持することを示します。

配列変数の宣言

1 次元の配列変数を宣言するには

- 宣言で、変数名の後に 1 組のかっこを追加します。倍精度浮動小数点数型 (Double) (Visual Basic) の要素を持つ 1 次元配列を保持するために、変数を宣言するコード例を次に示します。

```
Dim cargoWeights() As Double
```

上の例では、配列変数を宣言しますが、配列は割り当てられません。しかし、1 次元配列を作成し、初期化し、`cargoWeights` に割り当てる必要があります。

多次元の配列変数を宣言するには

- 宣言の中で変数名の後に 1 組のかっこを追加し、かっこの中にコンマを記入して次元を区切ります。Short 型 (Visual Basic) の要素を持つ 4 次元配列を保持するために、変数を宣言するコード例を次に示します。

```
Dim atmospherePressures(,,,) As Short
```

上の例では、配列変数を宣言しますが、配列は割り当てられません。しかし、4 次元配列を作成し、初期化し、`atmospherePressures` に割り当てる必要があります。

ジャグ配列変数を宣言するには

- 宣言の中で、入れ子になった配列 1 レベルにつき 1 組のかっこを変数名の後に追加します。配列の配列の配列 (配列、その各要素が配列、その各要素が配列) を保持するために、バイト型 (Byte) (Visual Basic) の要素を持つ最も内側の配列を使用して、変数を宣言するコード例を次に示します。

```
Dim inquiriesByYearMonthDay()()() As Byte
```

上の例では、配列変数を宣言しますが、配列は割り当てられません。しかし、配列の配列の配列を作成し、初期化し、`inquiriesByYearMonthDay` に割り当てる必要があります。

参照

処理手順

方法 : 配列を作成する
方法 : 多次元配列を作成する
方法 : 配列の配列を作成する
方法 : 配列変数を初期化する
配列のトラブルシューティング

概念

Visual Basic の配列の概要
Visual Basic における多次元配列
Visual Basic におけるジャグ配列

その他の技術情報

Visual Basic における配列

方法：配列を作成する

配列はオブジェクトなので、New (Visual Basic) 句を使用して作成し、配列変数にこれを代入できます。この操作は、配列宣言の一部として行うことも、後続の代入ステートメントで行うこともできます。

配列宣言ステートメントで配列を作成するには

- 宣言で、変数名とそのかっこの後に **New** 句を追加します。次の例では 日付型 (Date) (Visual Basic) の要素を持つ配列を保持するための変数を宣言し、配列を作成し、変数に配列を代入します。

```
Dim validDates() As Date = New Date() {}
```

このステートメントを実行すると、`validDates` 変数の配列の長さが 0 になります。

メモ：

New 句では、型名に続けてかっこを指定し、かっこに続けて中かっこ **{}** を指定する必要があります。かっこは、配列コンストラクタへの呼び出しを表しているわけではありません。かっこは、オブジェクト型が配列型であることを示しています。中かっこには初期値を指定します。値を指定しない場合でも、コンパイラは中かっこを要求します。したがって、**New** 句には、かっこの中が空でも、かっこと中かっこの両方を含める必要があります。

別の代入ステートメントで配列を作成するには

- **New** 句を指定して、後続の代入ステートメントを使用します。次の例では 整数型 (Integer) (Visual Basic) の要素を持つ配列を保持するための変数を宣言し、配列を作成し、別のステートメントで変数に配列を代入します。

```
Dim scores() As Integer
scores = New Integer() {}
```

これらのステートメントを実行すると、`scores` 変数の配列の長さが 0 になります。

または

- 配列を作成するだけでなく長さも初期化する場合は ReDim ステートメント (Visual Basic) を使用します。

```
ReDim scores(4)
```

このステートメントを実行すると、変数 `scores` の配列は長さが 5 になり、各要素に既定値が設定されます。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 多次元配列を作成する
方法 : 配列の配列を作成する
方法 : 複数の要素型が混在する配列を作成する
方法 : 要素を持たない配列を作成する
方法 : 配列変数を初期化する
配列のトラブルシューティング

概念

Visual Basic の配列の概要

その他の技術情報

Visual Basic における配列

方法：複数の要素型が混在する配列を作成する

配列で宣言するのは 1 つのデータ型だけです。したがって、すべての要素はそのデータ型である必要があります。すべての要素は綿密に関連し合っており、類似した型の値を持つため、通常はこの制約は望ましいものです。しかし、場合によっては要素が綿密に関連していなかったり、類似した型の値を持っていない場合があります。そのような場合は、オブジェクト型 (Object)の配列要素を宣言できます。個々の要素は、数字、文字、文字列、オブジェクト、および他の配列など、異なる種類のデータをポイントできるようになります。

異なるデータ型を持つ要素が混在する配列を作成するには

- 配列を**Object**として宣言します。**Object**要素の配列を保持する変数を宣言して、配列を作成し、それを変数に割り当てる例を次に示します。

```
Dim mixedTypes As Object() = New Object() {}
```

Object データ型を使用する場合、特定のデータ型を指定した場合に比べてパフォーマンスの効率は劣ることを念頭に置いてください。これは実行時に、ボックス化とボックス化解除と呼ばれる操作によって、通常のデータ型と**Object**の間でデータ変換を行う必要があるためです。これを頻繁に実行すると、このような追加の処理によってパフォーマンスに影響を及ぼす場合があります。

配列内の異なるデータ型の要素にアクセスするには

- 通常の方法で要素を読み取りまたは書き込みます。任意のデータ型の要素を**Object**配列に保存したり、配列から取得したりできます。

  異なるデータ型の情報を**Object**配列に挿入する実際の例を次に示します。配列内の従業員情報を変数 `employeeData` に保存します。

```
Dim employeeData(3) As Object
employeeData(0) = "Alex Hankin"
employeeData(1) = "4242 Maple Street"
employeeData(2) = 48
employeeData(3) = #8/23/1956#
```

  Object配列から異なるデータ型の情報を取得するには、次の例で示すように、要素を適切なデータ型に変換します。

```
Dim age As Integer = CInt(employeeData(2))
Dim birthDate as Date = CDate(employeeData(3))
```

要素が互いに類似していない、または関連していない場合、別の方法として、要素を**Object**配列ではなく、コレクションに入れます。詳細については、「配列の代わりとしてのコレクション」を参照してください。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 多次元配列を作成する
方法 : 配列の配列を作成する
方法 : 要素を持たない配列を作成する
方法 : 配列変数を初期化する
配列のトラブルシューティング

概念

Visual Basic の配列の概要
配列の代わりとしてのコレクション

その他の技術情報

Visual Basic における配列

方法：要素を持たない配列を作成する

要素を持たない配列は、長さ 0 の配列と呼ばれます。長さ 0 の配列を保持する変数には、**Nothing** 値が保持されているのではありません。

次のような場合に、長さ 0 の配列を作成する必要があります。

- コードで、Length や Rank などの Array クラスのメンバにアクセスする必要がある場合、または NullReferenceException 例外を発生させることなく、UBound 関数 (Visual Basic) などの Visual Basic 関数を呼び出す必要がある場合。
- 特別なケースですが、**Nothing** をチェックする必要性をなくすことによって利用側のコードを簡素化する場合。
- コードで、長さ 0 の配列を 1 つ以上のプロシージャに渡す必要があるアプリケーション プログラミング インターフェイス (API: Application Programming Interface) とやり取りする場合、または API の 1 つ以上のプロシージャから長さ 0 の配列が返される場合。

要素を持たない配列を作成するには

- 配列の次元のいずれかを -1 に宣言します。次の例では、文字列型 (String) (Visual Basic) の要素を持つ配列を保持するよう変数を宣言していますが、初期の状態では空に設定しています。

```
Dim twoDimensionalStrings(-1, 3) As String
```

このステートメントを実行すると、`twoDimensionalStrings` 変数内の配列は、長さ 0 の 2 次元配列になります。この配列は空ですが存在はしています。したがって、この配列を指す変数は **Nothing** と等価ではありません。また、空でない配列を作成して `twoDimensionalStrings` に代入することもできます。

一方、次の例では、初期状態ではいずれの配列もポイントしない配列変数を宣言します。

```
Dim twoDimStrings( , ) As String
```

上の例の `twoDimensionalStrings` とは異なり、変数 `twoDimStrings` の値は **Nothing** です。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 多次元配列を作成する
方法 : 配列の配列を作成する
方法 : 複数の要素型が混在する配列を作成する
方法 : 配列変数を初期化する
配列のトラブルシューティング

概念

Visual Basic の配列の概要

その他の技術情報

Visual Basic における配列

方法：配列変数を初期化する

配列は、New (Visual Basic) 句の一部として、作成と同時に初期化できます。また、後続の代入ステートメントで、初期化することもできます。

初期化できる配列の要素は、次のとおりです。

- 配列の次元の長さが指定される、インデックスの上限
- いくつかの配列の要素、またはすべての配列の要素の値

これらのうちのいずれかを、他方を初期化せずに初期化できます。しかし、要素の値を指定して、上限を指定しない場合、指定した値の数によって上限が決まります。

作成時に、New 句で配列を初期化するには

- **New** 句では、かっこの内側でインデックスの上限を指定し、中かっこ (**{}**) の内側で要素の値を指定します。次の例では、変数を宣言、作成、および初期化して、文字型 (Char) (Visual Basic) の要素と共に配列を保持し、上限および値を指定します。

  ```
  Dim testChars As Char() = New Char(2) {"%"c, "&"c, "@"c}
  ```

  このステートメントの実行の後で、変数 `testChars` の配列が長さ 3 になり、インデックス 0 からインデックス 2 の要素によって、初期化された値が保持されます。上限および値の両方を指定した場合、インデックス 0 から上限までの、すべての要素の値を含める必要があります。

  Char データ型への文字リテラルを強制する、リテラルの型文字 **c** に留意してください。型文字がない場合、二重引用符 (" ") に囲まれたリテラルは、既定値の **String** に設定されます。

  New 句で要素の値を指定した場合は、インデックスの上限を指定する必要はありません。次の例では、変数を宣言、作成、および初期化し、ブール型 (Boolean) (Visual Basic) の要素と共に配列を保持し、要素の値だけを指定します。

  ```
  Dim answers As Boolean() = New Boolean() {True, True, False, True}
  ```

  このステートメントの実行の後で、変数 `answers` の配列の長さが 4 になり、インデックス 0 からインデックス 3 の要素によって、初期化された値が保持されます。

要素を初期化せずに、インデックスの上限を初期化できます。この方法で配列を作成した場合、後続の代入ステートメントを使用して、それぞれの要素の値を初期化する必要があります。

後続の代入ステートメントの配列を初期化するには

1. 配列変数の宣言で、インデックスの上限を指定します。
2. 1 つ以上の代入ステートメントを使用します。各代入ステートメントによって、配列の要素のいずれかに値が割り当てられます。次の例では、変数を宣言および作成し、文字列型 (String) (Visual Basic) の要素と共に配列を保持し、後続のステートメントで 2 つの要素の値を指定します。

   ```
   Dim comments(30) As String
   comments(0) = "This is the first comment."
   comments(5) = "This is the sixth comment."
   ```

   これらのステートメントの実行の後で、変数 `comments` の配列の長さが 31 になり、インデックス 0 および 5 の要素によって、初期化された値が保持され、他の 29 の要素によって既定値が保持されます。この方法で配列を初期化した場合、いくつかの要素を初期化して、その他をスキップできます。

または

- ReDim ステートメント (Visual Basic) を使用して、配列の長さを初期化します。

  ```
  ```

```
ReDim comments(5)
```

このステートメントの実行の後で、変数 `comments` の配列が長さ 6 になり、すべての要素によって既定値が保持されます。

メモ：

1 か所だけで、インデックスの上限を初期化できます。配列の変数名の後のかっこ内の上限を指定した場合、**New** 句を使用できません。**New** 句でかっこ内の上限を指定した場合、変数名の後のかっこを空にしておく必要があります。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 多次元配列を初期化する
方法 : ジャグ配列を初期化する
配列のトラブルシューティング

概念

Visual Basic の配列の概要

その他の技術情報

Visual Basic における配列

Visual Basic における配列の次元

次元とは、配列の要素の仕様を変更できる方向のことです。その月の売上合計を日付ごとに保持している配列は、1 次元となります (月の日付)。その月の日付ごとの売上合計を部門別に保持している配列は、2 次元となります (部門の番号と月の日付)。配列で保持している次元の数は、ランクと呼ばれます。

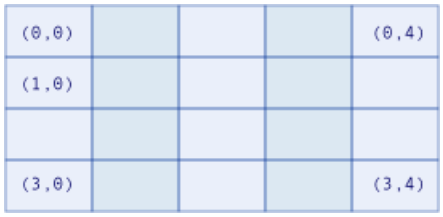
次元の使用

配列の要素を指定するには、その配列の各次元に対してインデックスまたは添字を提供します。要素は、インデックス 0 からその次元で最も高位のインデックスまで、各次元に従って連続しています。

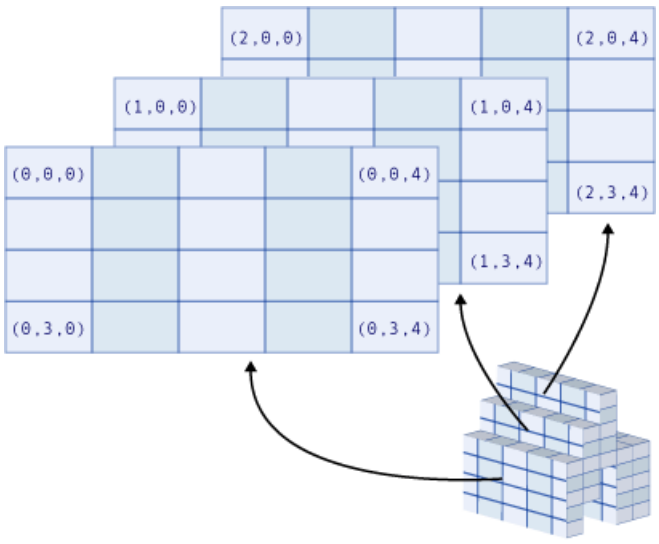
ランクが異なる配列の概念的構造を次の図に示します。図の各要素は、これにアクセスするインデックスの値を示しています。たとえば、2 次元配列の 2 行目の最初の要素にアクセスするには、インデックス `(1, 0)` を指定します。

1 次元配列

2 次元配列



3 次元配列



1 次元

各年齢の人数など、1 次元だけの配列は多数あります。要素を指定するのに唯一必要なことは、その要素でカウントする年齢だけです。したがって、このような配列では 1 つのインデックスだけを使用します。次の例では、年齢が 0 ～ 120 の年齢別カウントの 1 次元配列を保持する変数を宣言しています。

```
Dim ageCounts(120) As UInteger
```

2 次元

キャンパス内の各建物のフロアごとのオフィスの数など、2 次元の配列を持つものがあります。要素を指定するには、建物の番号とフロアの両方が必要で、さらに各要素は建物とフロアを組み合わせたカウントを保持する必要があります。したがって、このような配列では 2 つのインデックスを使用します。次の例では、建物の番号が 0 ～ 40、そしてフロアが 0 ～ 5 であるオフィスの数の 2 次元配列を保持する変数を宣言しています。

```
Dim officeCounts(40, 5) As Byte
```

2 次元配列は、多次元配列とも呼ばれます。

3 次元

3 次元空間の値など、3 次元での配列もいくつかあります。このような配列では 3 つのインデックスを使用し、その場合には物理的な空間を *x*、*y*、および *z* 軸で表します。次の例では、3 次元ボリュームのさまざまなポイントでの気温を表すため 3 次元配列を保持する変数を宣言しています。

```
Dim airTemperatures(99, 99, 24) As Single
```

3 次元より多くの次元

配列では最高 32 次元まで保持することが可能ですが、3 次元より多くなることは稀です。

メモ :

配列に必要なメモリ領域は多次元になるほど増大するので、多次元配列を宣言するときには注意してください。

異なる次元の使用

当月の売上額を日付ごとに追跡する場合を考えます。次の例で示されているように、月の日付ごとに 1 つずつ、31 の要素に対して 1 次元の配列を宣言する必要があります。

```
Dim salesAmounts(30) As Double
```

その月の日付についてだけでなく、年間をとおして毎月同じ情報を追跡する場合を考えます。次の例で示すように、12 行 (月) と 31 列 (日付) の 2 次元配列を宣言できます。

```
Dim salesAmounts(11, 30) As Double
```

配列に複数年の情報を保持させる場合を考えます。5 年間分の売上額を追跡する場合は、次の例で示すように、5 層、12 行、および 31 列の 3 次元配列を宣言できます。

```
Dim salesAmounts(4, 11, 30) As Double
```

各インデックスの値は 0 から最大値まで多岐にわたるため、`salesAmounts` の各次元は、その次元で必要な長さより小さく宣言されることに注意してください。また、配列のサイズは新しい次元ごとに増大していくことにも注意してください。前の例では 3 つのサイズは、それぞれ 31、372、および 1,860 要素です。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
配列のトラブルシューティング

概念

Visual Basic の配列の概要
Visual Basic における多次元配列
Visual Basic におけるジャグ配列
Visual Basic における配列データ型

その他の技術情報

Visual Basic における配列

方法：配列の下限に 0 を指定する

配列を宣言する場合、**To** キーワードでゼロ文字 (0) を指定して、各次元の下限を指定できます。これによって必要な下限が変更されるわけではありませんが、コードが読み取りが容易になります。

配列の下限を明示的にゼロに指定するには

1. 通常の方法で配列を宣言します。
2. かっこ内で、各次元の上限の前に `0 To` を追加します。

```
Public Sub declarelowerbounds()
    Dim monthtotal(0 To 11) As Double
    Dim cell(0 To 39, 0 To 19) As Integer
    MsgBox("Total number of elements:" _
        & vbCrLf & "monthtotal (0 To 11) length " & CStr(monthtotal.Length) _
        & vbCrLf & "cell (0 To 39, 0 To 19) length " & CStr(cell.Length))
End Sub
```

下限は常に 0 である必要がありますが、明示的にこれを宣言することによってコードが読み取りやすくなります。また、上限と下限の両方を指定させることによって、下限が 0 であることに注意を促すことにもなります。

代替配列の作成。 Dim ステートメント (Visual Basic) 句または New (Visual Basic) 句を使用せずに、配列を作成できます。たとえば、CreateInstance メソッドを呼び出すか、または別のコンポーネントでこの方法により作成された配列のコードを渡すことができます。このような配列では 0 以外の下限を持つことができます。GetLowerBound メソッドまたは LBound 関数 (Visual Basic) を使用して、いつでも次元の下限をテストできます。

参照

処理手順

配列のトラブルシューティング

概念

Visual Basic における配列の次元

その他の技術情報

Visual Basic における配列

方法：配列の次元数を確認する

配列の **Rank** プロパティは、ランク (次元の数) を返します。

配列のランクを確認するには

- 配列の Rank プロパティを読み取ります。配列名の後にかっこを指定しないでください。

```
Dim thisThreeDimArray(,,) As Byte = New Byte(4, 4, 4) {}
MsgBox("Rank of thisThreeDimArray is " & CStr(thisThreeDimArray.Rank))
```

MsgBox 呼び出しによって、"Rank of thisThreeDimArray is 3"と表示されます。

配列のランクは、データ型の一部です。したがって、たとえ **ReDim** ステートメントを使用しても、ランクを変更することはできません。

参照

処理手順

方法 : 配列の下限に 0 を指定する
方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
配列のトラブルシューティング

概念

Visual Basic における配列の次元

その他の技術情報

Visual Basic における配列

Visual Basic における多次元配列

配列には、1 次元の配列と多次元の配列があります。配列が複数ある場合、多次元配列と呼ばれます。複数の次元があるということは、要素として他の配列を持つジャグ配列とは同等ではありません。

次元およびサイズ

配列の次元の数またはランクは、各要素を識別するために使用するインデックスの数と一致します。最大で 32 次元まで指定できますが、3 次元より大きい次元を指定することはほとんどありません。2 次元の配列変数および 3 次元の配列変数を宣言するコード例を次に示します。

```
Dim populations(200, 3) As Long
Dim matrix(5, 15, 10) As Single
```

要素の総数は、すべての次元の長さの積です。上の例では、`populations` には総数 804 の要素 (201 x 4)、および `matrix` には 1056 の要素 (6 x 16 x 11) があります。各インデックスの範囲は、0 から次元で指定された長さまでです。

2 次元配列は、四角形配列とも呼ばれます。

メモ :

配列に必要なメモリ領域は多次元になるほど増大するので、多次元配列を宣言するときには注意してください。

配列クラス メンバ

すべての配列は、System 名前空間の Array クラスから継承されます。したがって、すべての配列で、**Array** のメソッドやプロパティにアクセスできます。**Array** の次のメンバは、便利です。

- Rank プロパティは、配列のランク (次元の数) を返します。
- GetLength メソッドは、指定した次元に従って長さを返します。
- GetUpperBound メソッドは、指定した次元に対する最も高いインデックス値を返します。各次元に対する最も低いインデックス値は、常に 0 です。
- Length プロパティは、配列の要素の総数を返します。
- System.Array.Sort メソッドにより、1 次元配列の要素がソートされます。

GetLength および **GetUpperBound** は、指定する次元に対して、0 ベースの引数を受け取ります。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 多次元配列を作成する
方法 : 多次元配列を初期化する
方法 : ジャグ配列を初期化する
方法 : 多次元配列を初期化する
配列のトラブルシューティング

概念

Visual Basic の配列の概要
Visual Basic における配列の次元
Visual Basic におけるジャグ配列
Visual Basic における配列のサイズ

その他の技術情報

Visual Basic における配列

方法：多次元配列を作成する

複数のインデックスを使用する配列は、多次元配列と呼ばれます。1 次元配列と同様に、多次元配列は New (Visual Basic) 句を使用して作成し、配列変数に代入します。この処理は、配列宣言の一部に含めることも、後続の代入ステートメントで行うこともできます。

多次元配列を作成するには

1. 変数名の後ろのかっこ内に適切な数のコンマを記述します。コンマの数は、次元の数よりも 1 つだけ少なくします。
2. **New** 句のかっこ内にも同じ数のコンマを記述します。要素の値を何も指定しない場合、中かっこ (**{}**) の中にコンマは必要ありません。

   倍精度浮動小数点数型 (Double) (Visual Basic) の要素を持つ 2 次元配列を保持する変数を宣言して、配列を作成し、変数にこれを代入する例を次に示します。

   ```
   Dim weights(,) As Double = New Double(,) {}
   ```

   このステートメントを実行すると、`weights` 変数の配列の長さが 0 になります。

   メモ：

   配列に必要なメモリ領域は多次元になるほど増大するので、多次元配列を宣言するときには注意してください。

多次元配列を効率的に使用するには

- 入れ子にした **For** ループで囲みます。

  次の例では、`matrix` のすべての要素を配列内の要素の場所に基づいて 0 ～ 99 までの値で初期化します。

  ```
  Dim matrix(9, 9) As Double
  Dim maxDim0 As Integer = UBound(matrix, 1)
  Dim maxDim1 As Integer = UBound(matrix, 2)
  For i As Integer = 0 To maxDim0
      For j As Integer = 0 To maxDim1
          matrix(i, j) = (i * 10) + j
      Next j
  Next i
  ```

  多次元配列は、ジャグ配列と同じではありません。詳細については、「方法 : 配列の配列を作成する」を参照してください。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 複数の要素型が混在する配列を作成する
方法 : 要素を持たない配列を作成する
方法 : 多次元配列を初期化する
配列のトラブルシューティング

関連項目

For...Next ステートメント (Visual Basic)

概念

Visual Basic における多次元配列

その他の技術情報

Visual Basic における配列

方法：多次元配列を初期化する

多次元配列変数は、1 次元の配列と同じ方法で初期化できます。ただし、各次元に応じた記述が必要です。

多次元の配列変数を宣言するには

- 配列変数の宣言で、かっこ内にコンマで区切って各インデックスの上限を指定します。Short 型 (Visual Basic) の要素を持つ 2 次元配列を保持するための変数を、上限だけを指定して宣言および作成する例を次に示します。

```
Dim sizes(1, 1) As Short
```

このステートメントを実行すると、`sizes` 変数の配列の要素数が合計 4 つになります。それらの要素はインデックス (0, 0)、(0, 1)、(1, 0)、(1, 1) にあり、それぞれ既定値が設定されます。この方法で配列を作成した場合、後続の代入ステートメントを使用して、それぞれの要素に値を代入する必要があります。

または

- 宣言の後に等号 (=) と New (Visual Basic) 句を置きます。**New** 句では、要素のデータ型を繰り返し、かっこの内側でインデックスの上限を指定し、空の中かっこ (**{}**) を指定します。**Short** データ型の要素を持つ 3 次元配列を保持するための変数を、上限だけを指定して宣言および作成する例を次に示します。

```
Dim replyCounts(,,) As Short = New Short(2, 1, 2) {}
```

このステートメントを実行すると、`replyCounts` 変数の配列の要素数が 18 になり、各要素に既定値が設定されます。この方法で配列を作成した場合、後続の代入ステートメントを使用して、それぞれの要素に値を代入する必要があります。

メモ：

インデックスの上限は、1 か所でだけ初期化できます。配列変数名の後のかっこ内で上限を指定する場合、**New** 句は使用できません。**New** 句のかっこ内で上限を指定する場合、変数名の後のかっこは空にしておく必要があります。

または

- **New** 句では、かっこ内で各インデックスの上限を指定し、中かっこ (**{}**) 内で要素の値を指定します。**Short** データ型の要素を持つ 2 次元配列を保持するための変数を、上限および値を指定して宣言、作成および初期化する例を次に示します。**New** 句に 2 つのレベルの中かっこがあることに注意してください。

```
Dim startingScores(,) As Short = New Short(1, 1) {{10, 10}, {10, 10}}
```

このステートメントを実行すると、変数 `startingScores` の配列には初期化された 4 つの要素が格納されます。上限と値の両方を指定する場合は、すべての次元について、インデックス 0 ～上限までの値を指定する必要があります。

または

- **New** 句では、かっこ内を適切な次元数を示すコンマのみにし、中かっこ (**{}**) の中に要素の値を指定します。単精度浮動小数点型 (Single) (Visual Basic) の要素を持つ 2 次元配列を保持するための変数を、要素の値だけを指定して宣言、作成および初期化する例を次に示します。**New** 句に 2 つのレベルの中かっこがあることに注意してください。

```
Dim diagonal(,) As Single = New Single(,) {{1, 0}, {0, 1}}
```

このステートメントを実行すると、変数 `diagonal` の配列には初期化された 4 つの要素が格納されます。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 多次元配列を作成する
方法 : 配列変数を初期化する
方法 : ジャグ配列を初期化する

配列のトラブルシューティング

概念

Visual Basic における多次元配列

その他の技術情報

Visual Basic における配列

Visual Basic におけるジャグ配列

各要素の配列自体が配列となっているものは、配列の配列、またはジャグ配列と呼ばれます。要素として配列を持つことは、1 つの配列に対して複数のインデックスを持つ多次元配列とは異なることに注意してください。

ジャグの意味

アプリケーションのデータ構造は、2 次元の配列であっても四角形の 2 次元配列ではない場合があります。たとえば、月の配列があり、その各要素が日の配列である場合があります。月によって日数が異なるため、月要素は四角形の 2 次元配列を形成しません。このような場合に、多次元配列の代わりにジャグ配列を使用できます。

例

倍精度浮動小数点数型 (Double) (Visual Basic) の要素が配列となっている配列を保持する配列変数を宣言する例を、次に示します。配列 `sales` の各要素自体が、月を表す配列となっています。各月の配列は、その月の各日付ごとの値を保持しています。

```
Dim sales()() As Double = New Double(11)() {}
Dim month As Integer
Dim days As Integer
For month = 0 To 11
    days = DateTime.DaysInMonth(Year(Now), month + 1)
    sales(month) = New Double(days - 1) {}
Next month
```

`sales` の宣言の **New** 句は、配列変数を 12 要素の配列に設定します。配列の各要素の型は **Double()** で、**Double** の要素を配列として持っています。**For** ループによって、今年 (**Year(Now)**) の各月の日数を決定し、対応する `sales` の要素を適切な長さの (**Double**) の配列に設定します。

この例の場合、ジャグ配列は 2 次元配列と比べて、7 つ少ない要素 (閏年の場合は 6 つ少ない要素) を保存します。極端な場合には、メモリを大幅に節約できます。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列の配列を作成する
方法 : ジャグ配列を初期化する
配列のトラブルシューティング

概念

Visual Basic の配列の概要
Visual Basic における配列の次元
Visual Basic における多次元配列
Visual Basic における配列データ型
CLS 準拠コードの記述

その他の技術情報

Visual Basic における配列

方法：配列の配列を作成する

要素の配列を持つ配列は、配列の配列、またはジャグ配列と呼ばれます。1 次元配列と同様に、New (Visual Basic) 句を使用して作成し、配列変数に割り当てます。配列宣言の一部として、または後続の代入ステートメント内でこれを行うことができます。

ジャグ配列の作成

1. 入れ子になった配列 1 レベルにつき 1 組のかっこを変数名の後に追加します。
2. 同じ数のかっこを、**New** 句に追加します。要素の値を何も指定していない場合、複数の中かっこ (**{}**) は必要ありません。次の例では、バイト型 (Byte) (Visual Basic) の要素を持つ配列をそのまま配列で保持する変数を宣言して、配列を作成し、変数に割り当てます。

   ```
   Dim ratings As Byte()() = New Byte()() {}
   ```

   このステートメントの実行すると、`ratings` 変数の配列の長さが 0 になります。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 多次元配列を作成する
方法 : 複数の要素型が混在する配列を作成する
方法 : 要素を持たない配列を作成する
方法 : ジャグ配列を初期化する
配列のトラブルシューティング

概念

Visual Basic におけるジャグ配列
CLS 準拠コードの記述

その他の技術情報

Visual Basic における配列

方法：ジャグ配列を初期化する

ジャグ配列変数を初期化するときには、トップレベルの配列に対してだけ次元の長さを指定できます。これにはいくつかの方法があります。

ジャグ配列変数を初期化するには

- 配列変数の宣言で、最初のかっこのペア内に、コンマで区切ってトップレベル インデックスの上限を指定します。**Byte** 型の要素のジャグ配列を保持するための変数を、トップレベルの上限だけを指定して宣言および作成する例を次に示します。

```
Dim rainfall(11)() As Byte
```

このステートメントを実行すると、`rainfall` 変数内の配列の要素数が 12 個になります。各要素には、**Byte** 型の要素を持つ空の配列がそれぞれ格納されます。

または

- **New** 句で、最初のかっこのペア内でトップレベル インデックスの上限を指定し、空の中かっこ (**{}**) を付けます。**Short** 型の要素のジャグ配列を保持するための変数を、トップレベルの上限だけを指定して宣言および作成する例を次に示します。

```
Dim snowfall()() As Short = New Short(11)() {}
```

このステートメントを実行すると、`snowfall` 変数内の配列の要素数が 12 個になります。各要素には、**Short** 型の要素を持つ空の配列がそれぞれ格納されます。

メモ：

トップレベルのインデックスの上限は、1 か所でだけ初期化できます。配列変数名の後のかっこ内で上限を指定する場合、**New** 句は使用できません。**New** 句のかっこ内で上限を指定する場合、変数名の後のかっこは空にしておく必要があります。

または

- **New** 句で、かっこ内でトップレベル インデックスの上限を指定し、中かっこ (**{}**) の中に要素の値を指定します。**Char** 型の要素のジャグ配列を保持するための変数を、トップレベルの上限および値を指定して宣言、作成、および初期化する例を次に示します。**New** 句を入れ子にすることで、下のレベルの配列を初期化しています。

```
Dim decodeValues()() As Char = New Char(1)() {New Char() {"a"c, "b"c}, New Char() {"p"
c, "q"c}}
```

このステートメントを実行すると、`decodeValues` 変数の配列の要素数が 2 個になります。各要素には長さが 1 の **Char** 配列がそれぞれ格納されます。また、それらの配列のインデックス 0 の要素にはそれぞれ初期値が設定されます。トップレベルで上限と値の両方を指定する場合は、インデックス 0 ～上限までの、すべてのトップレベル要素の値を指定する必要があります。

または

- **New** 句で、トップレベルのかっこを省略し、中かっこ (**{}**) の中に要素の値を指定します。**Byte** 型の要素のジャグ配列を保持するための変数を、要素の値だけを指定して宣言、作成、および初期化する例を次に示します。**New** 句に 2 つのレベルの中かっこがあることに注意してください。

```
Dim firstValues()() As Byte = {New Byte() {2, 1}, New Byte() {3, 0}}
```

このステートメントを実行すると、`firstValues` 変数の配列の長さが 2 になり、`firstValues(0)` 要素および `firstValues(1)` 要素ができます。各要素は 2 つの要素を持つ **Byte** 配列として初期化されます。その 1 番目の配列に含まれる要素の値は 2 と 1 であり、2 番目の配列に含まれる要素の値は 3 と 0 です。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列の配列を作成する

方法 : 配列変数を初期化する
方法 : 多次元配列を初期化する
配列のトラブルシューティング

関連項目

New (Visual Basic)

概念

Visual Basic におけるジャグ配列
CLS 準拠コードの記述

その他の技術情報

Visual Basic における配列

Visual Basic における配列データ型

すべての配列を包括する 1 つのデータ型はありません。配列のデータ型は次の要因によって決定されます。

- 配列であること。
- ランク (次元数)。
- 配列の要素のデータ型。

したがって、ランクと要素のデータ型が一致しない限り、2 つの配列変数が同じデータ型を持つと見なされることはありません。

各次元における長さは、配列のデータ型に影響を及ぼさないことに注意してください。

配列の例

次の例では、さまざまなデータ型を持つ 4 つの配列変数を宣言します。

```
Dim firstArray(12, 8) As UInteger
Dim secondArray(12, 8, 3) As UInteger
Dim thirdArray(12, 8) As String
Dim fourthArray(5, 20) As UInteger
```

上のステートメントを実行した後、これらの配列変数の間にはデータ型について次のような関係が成り立ちます。

- 変数 `firstArray` と変数 `secondArray` とは、ランクが異なるため同じデータ型ではありません。
- 変数 `firstArray` と変数 `thirdArray` とは、要素のデータ型が異なるため同じデータ型ではありません。
- 変数 `firstArray` と変数 `fourthArray` とは、同じデータ型です。したがって、一方の変数をもう一方の変数に代入できます。

ジャグ配列のデータ型

配列の配列、つまり、他の配列を要素として含む配列は、要素の配列の長さが必ずしも等しくないため、ジャグ配列とも呼ばれます。次の例では、異なるデータ型を持つ 2 つのジャグ配列変数を宣言します。

```
Dim twoDimOfOneDim(,)() As Integer
Dim oneDimOfTwoDim()(,) As Integer
```

`twoDimOfOneDim` の配列は 2 次元で、その要素のデータ型は **Integer()**、つまり 1 次元の **Integer** 配列です。`oneDimOfTwoDim` の配列は 1 次元で、その要素のデータ型は **Integer(,)**、つまり 2 次元の **Integer** 配列です。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列のデータ型を決定する
配列のトラブルシューティング

概念

Visual Basic の配列の概要
Visual Basic における配列の次元
Visual Basic における多次元配列
Visual Basic におけるジャグ配列

その他の技術情報

Visual Basic における配列

方法：配列のデータ型を決定する

配列のデータ型は、その要素とは同じではありません。いくつかの方法で、配列またはその要素のいずれかのデータ型を知ることができます。

- 変数の System.Object.GetType メソッドを呼び出して、実行時型の変数の Type オブジェクトを取得できます。**Type** オブジェクトでは、プロパティおよびメソッドに詳細情報が保持されます。
- 変数を TypeName 関数 (Visual Basic) に渡して、実行時型の名前が含まれる **String** を取得できます。
- 変数を VarType 関数 (Visual Basic) に渡して、変数の型の分類を表す **VariantType** 値を取得できます。

配列のデータ型を決定するには

- 配列名の **TypeName** を呼び出します。配列自体の型を要求しているため、配列名の後にかっこを指定しないでください。

```
Dim thisTwoDimArray(,) As Integer = New Integer(9, 9) {}
MsgBox("Type of thisTwoDimArray is " & TypeName(thisTwoDimArray))
```

MsgBox を呼び出すと、"thisTwoDimArray の型は Integer(,) です" が表示され、要素の型および次元の数の両方が表示されます。次元の現在の長さは配列のデータ型の一部ではないため、表示されません。

配列の要素のデータ型を決定するには

- 既存の要素を選択し、その要素の **TypeName** を呼び出します。

```
Dim thisTwoDimArray(,) As Integer = New Integer(9, 9) {}
MsgBox("Type of thisTwoDimArray(0, 0) is " & TypeName(thisTwoDimArray(0, 0)))
```

MsgBox を呼び出すと、"thisTwoDimArray(0, 0) の型は Integer です" が表示されます。

要素のデータ型は、配列のデータ型の一部です。このため、代入ステートメントまたは **ReDim** ステートメントを使用しても、データ型を変更できません。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
配列のトラブルシューティング

関連項目

TypeName 関数 (Visual Basic)
VarType 関数 (Visual Basic)
VariantType 列挙型

概念

Visual Basic における配列データ型

その他の技術情報

Visual Basic における配列

Visual Basic における配列のサイズ

配列のサイズは、そのすべての次元の長さの積です。これは、現在配列に含まれている要素の総数を表します。

次の例では 3 次元配列を宣言しています。

```
Dim prices(3, 4, 5) As Long
```

変数 `prices` の配列の全体のサイズは、*(3 + 1) x (4 + 1) x (5 + 1) = 120* です。

配列サイズの考慮事項

配列のサイズを扱う際に考慮する必要があるいくつかの点があります。

次元の長さ

各次元のインデックスは 0 ベースであり、範囲は 0 から上限です。したがって、次元の長さは、その次元に対する宣言された上限よりも 1 だけ大きくなります。

長さの制限

配列のすべての次元の長さは、*(2 ^ 31) - 1* である **Integer** データ型の最大値に制限されます。しかし、配列のサイズの総数も、システムで利用できるメモリによって制限されます。利用できる RAM の容量を超える配列を初期化する場合、共通言語ランタイムは OutOfMemoryException 例外をスローします。

サイズおよび要素のサイズ

配列のサイズは、その要素のデータ型には依存しません。サイズは常に、使用するストレージのバイト数ではなく、要素の合計数を表します。

メモリの使用量

配列がメモリに格納される方法に関して、推測で行うのは安全ではありません。ストレージは、データ幅の異なるプラットフォームによって変わります。したがって、配列が同じであれば、32 ビットのシステムよりも、64 ビットのシステムのほうがより多くのメモリを使用します。配列を初期化するとき、システム構成に応じて、要素をできるだけ近くに集めるように、またはハードウェア本来の境界条件にすべてをそろえるように、共通言語ランタイム (CLR: Common Language Runtime) によってストレージが割り当てられます。また、配列は制御情報のためのストレージのオーバーヘッドを必要とします。このオーバーヘッドは、次元が追加されるごとに増加します。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列のサイズを決定する
方法 : 配列のサイズを変更する
方法 : 配列の 1 次元の長さを決定する
配列のトラブルシューティング

その他の技術情報

Visual Basic における配列

方法：配列のサイズを決定する

配列の全体のサイズは、そのすべての次元の長さの積です。配列の Length プロパティは、この全体のサイズを返します。これは、配列が占有するストレージのバイト数ではなく、配列に現在含まれている要素の合計数を表します。

配列のサイズの総数を決定するには

- 配列の **Length** プロパティを読み取ります。配列名の後にかっこを指定しないでください。

```
Dim thisDoubleArray(,) As Char = New Char(4, 9) {}
MsgBox("Total length of thisDoubleArray is " & CStr(thisDoubleArray.Length))
```

  MsgBox を呼び出すと、"thisDoubleArray の合計長さは 50 です。" が表示されます。

配列の GetLength メソッドから、各次元の長さを知ることができます。

個々の次元の長さを変更できますが、全体のサイズが変わります。しかし、ランクを変更することはできません (次元の数)。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列のサイズを変更する
方法 : 配列の 1 次元の長さを決定する
配列のトラブルシューティング

関連項目

Length

概念

Visual Basic における配列のサイズ

その他の技術情報

Visual Basic における配列

方法：配列のサイズを変更する

配列変数のサイズを変更するには、新しい配列オブジェクトを代入します。標準の代入ステートメントおよび ReDim ステートメント (Visual Basic) のいずれかを使用できます。いずれの場合でも元の配列は新しい配列によって完全に置換され、配列変数は新しい配列を指します。

配列のサイズを変更することは、効率的なメモリ管理に役立ちます。たとえば、最初は小さなサイズの配列を作成し、より多くの要素が必要になった場合にサイズを大きくできます。または、最初は大きなサイズの配列を作成し、全部を使用する必要がなくなったらサイズを小さくしていくこともできます。この技法によって、必要なときにだけ追加のメモリを使用できます。

標準の代入ステートメントを使用して配列変数のサイズを変更するには

1. 新しい次元の長さを指定し、新しい配列オブジェクトを作成します。
2. 新しい配列オブジェクトを配列変数に代入します。

```
Dim thisArrayVariable() As Integer = New Integer(99) {}
thisArrayVariable = New Integer(49) {}
```

ReDim ステートメントを使用して配列変数のサイズを変更するには

- **ReDim** ステートメントで配列変数の新しい次元の長さを指定します。

```
Dim thisArrayVariable() As Integer = New Integer(99) {}
ReDim thisArrayVariable(9)
```

ReDim を使って配列のサイズを変更すると、通常、配列内にある要素の既存の値は失われます。しかし、**ReDim** ステートメントに **Preserve** キーワードを含めると、既存の値を保持できます。

既存の要素の値を維持しながら配列変数のサイズを変更するには

1. **ReDim** ステートメントで配列変数の新しい次元の長さを指定します。
2. **ReDim** ステートメントに Preserve キーワードを追加します。次の例では、新しい配列を作成し、その要素を既存の `arrayToIncrease` 配列の対応する要素で初期化します。次に、この初期化した配列を `arrayToIncrease` 配列変数に代入します。

```
Dim arrayToIncrease(9, 49)
ReDim Preserve arrayToIncrease(9, 199)
```

多次元配列で **Preserve** を使った場合、変更できるのは最後の次元の長さだけです。他の次元を変更しようとすると、ArrayTypeMismatchException 例外が発生します。

Preserve キーワードを使用して大きな配列のサイズを変更する場合、Visual Basic によって既存のすべての要素が新しい配列にコピーされることに注意してください。これによってパフォーマンスが低下する可能性があります。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列のサイズを決定する
方法 : 配列の 1 次元の長さを決定する
方法 : 配列を別の配列に代入する
方法 : 配列を他の配列に変更する
配列のトラブルシューティング

概念

Visual Basic における配列のサイズ

その他の技術情報

Visual Basic における配列

方法：配列の 1 次元の長さを決定する

配列の GetLength メソッドは、指定した次元に基づいた長さを返します。

配列の 1 次元の長さを決定するには

- 配列名の **GetLength** を呼び出します。**GetLength** の引数として使用する長さの次元を指定します。次元の引数は、0 ベースです。

```
Dim sampleTripleArray(,,) As Short = New Short(2, 3, 4) {}
MsgBox("Dimension lengths of sampleTripleArray are " & CStr(sampleTripleArray.GetLengt
h(0)) _
    & ", " & CStr(sampleTripleArray.GetLength(1)) & ", " & CStr(sampleTripleArray.GetL
ength(2)))
```

  MsgBox を呼び出すと、"sampleTripleArray の各辺の長さは、3、4、5 です。"が表示されます。

各次元に対する最も小さいインデックス値は、常に 0 であり、GetUpperBound メソッドは、最も大きい次元のインデックス値を返します。各次元では、**GetLength** は、**GetUpperBound** によって返された値よりも 1 大きい値を返します。**GetLength** と同様に、**GetUpperBound** に指定した次元は、0 ベースです。

Length プロパティから、配列の長さの総数を知ることができます。

個々の次元の長さを変更して、全体のサイズを変更できます。しかし、ランクを変更することはできません (次元の数)。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列のサイズを決定する
方法 : 配列のサイズを変更する
配列のトラブルシューティング

概念

Visual Basic における配列のサイズ

その他の技術情報

Visual Basic における配列

配列および配列要素の操作

ここでは、配列の並べ替え、配列値の設定と取得、およびその他の配列と配列要素の操作について説明します。

このセクションの内容

方法 : 配列に値を格納する

配列の要素への値の格納方法について説明します。

方法 : 配列から値を取得する

配列の要素から値を取得する方法を説明します。

方法 : Visual Basic で、配列内の要素を検索する

配列の特定の要素を見つける方法を説明します。

方法 : Visual Basic で配列内の内容を反転させる

配列の要素を逆順に並べ替える方法について説明します。

方法 : Visual Basic で配列を並べ替える

配列の要素をアルファベット順に並べ替える方法について説明します。

方法 : 配列を別の配列に代入する

配列を別の配列変数に代入するときの手順と規則を説明します。

方法 : 配列を他の配列に変更する

実行可能な変更と変更の手順を説明します。

方法 : プロシージャまたはプロパティに配列を渡す

配列をプロシージャまたはプロパティに引数として渡す方法を説明します。

方法 : プロシージャまたはプロパティから配列を返す

プロシージャまたはプロパティを呼び出すコードに配列を返す方法を説明します。

参照

Dim ステートメント (Visual Basic)

1 つ以上の変数にストレージ領域を宣言し、割り当てる方法を説明します。

System.Array

配列の作成、操作、検索、および並べ替えに使用するメソッドを提供することにより、共通言語ランタイムのすべての配列の基本クラスとなることができるクラスについて説明します。

関連するセクション

配列のトラブルシューティング

配列を操作するときによく起こる問題について説明します。

Visual Basic における配列

配列を宣言して使用することによってコードをよりコンパクトで強力にする方法について説明します。

方法：配列に値を格納する

配列名および適切なインデックスを使用して個別の要素を指定し、配列の値のいずれか 1 つを格納できます。

配列要素に値を格納するには

1. 等号 (=) の左側に、配列名を指定し、その後にかっこを続けます。
2. かっこの中には、格納する要素に対応する各インデックス用の式を含めます。各配列の次元ごとに 1 つのインデックスが必要です。配列に値を格納するステートメントの例を示します。

```
Dim numbers() As Integer

Dim matrix(,) As Double

numbers(i + 1) = 0

matrix(3, j * 2) = j
```

各配列の次元ごとに、GetUpperBound メソッドではインデックスが保持できる一番高い値が返されます。一番低いインデックス値は常に 0 です。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列から値を取得する
方法 : Visual Basic で、配列内の要素を検索する
方法 : Visual Basic で配列内の内容を反転させる
方法 : Visual Basic で配列を並べ替える
配列のトラブルシューティング

概念

Visual Basic における配列の次元

その他の技術情報

Visual Basic における配列

方法：配列から値を取得する

配列名および適切なインデックスを使用して各要素を指定し、配列の値を取得することができます。

配列要素から値を取得するには

1. 式の内部で、配列名を指定し、その後にかっこを続けます。
2. かっこの中には、取得する要素に対応する各インデックス用の式を含めます。各配列の次元ごとに 1 つのインデックスが必要です。配列から値を取得するステートメントの例を示します。

```
Dim sortedValues(), rawValues(), estimates(,,) As Double
lowestValue = sortedValues(0)
wTotal += (rawValues(v) ^ 2)
firstGuess = estimates(i, j, k)
```

各配列の次元ごとに、GetUpperBound メソッドではインデックスで保持できる一番高い値を返します。一番低いインデックス値は常に 0 です。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列に値を格納する
方法 : Visual Basic で、配列内の要素を検索する
方法 : Visual Basic で配列内の内容を反転させる
方法 : Visual Basic で配列を並べ替える
配列のトラブルシューティング

概念

Visual Basic における配列の次元

その他の技術情報

Visual Basic における配列

方法：Visual Basic で、配列内の要素を検索する

次に示すのは、`zooAnimals` という名前の **String** オブジェクトの配列を宣言し、値を代入し、その後 "turtle" という要素を見つけ出し、その要素の場所を表示する例です。

使用例

このコードの例は、IntelliSense コード スニペットとしても利用できます。.コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

```
Public Sub findAnimal()
    Dim zooAnimals(2) As String
    zooAnimals(0) = "lion"
    zooAnimals(1) = "turtle"
    zooAnimals(2) = "ostrich"
    Dim turtleIndex As Integer
    turtleIndex = (Array.IndexOf(zooAnimals,"turtle"))
    MsgBox("The turtle is element " & turtleIndex)
End Sub
```

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- Mscorlib.dll および System 名前空間にアクセスします。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- 空の配列である場合 (ArgumentNullException クラス)。
- 多次元の配列である場合 (RankException クラス)。
- 配列の 1 つ以上の要素が IComparable インターフェイスを実装していない場合 (InvalidOperationException クラス)。

参照

処理手順

方法 : 配列に値を格納する
方法 : 配列から値を取得する
方法 : Visual Basic で配列内の内容を反転させる
方法 : Visual Basic で配列を並べ替える
配列のトラブルシューティング

関連項目

System.Array.IndexOf

概念

Visual Basic の配列の概要

その他の技術情報

Visual Basic における配列

方法：Visual Basic で配列内の内容を反転させる

次に示すのは、`zooAnimals` という名前の **String** オブジェクトの配列を宣言し、値を代入してから、内容を反転させる例です。

使用例

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

```
Public Sub reverseAnimals()
    Dim zooAnimals(2) As String
    zooAnimals(0) = "lion"
    zooAnimals(1) = "turtle"
    zooAnimals(2) = "ostrich"
    Array.Reverse(zooAnimals)
End Sub
```

コードのコンパイル方法

必要な条件は次のとおりです。

- Mscorlib.dll と System 名前空間へのアクセス

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- 空の配列である場合 (ArgumentNullException クラス)。
- 多次元の配列である場合 (RankException クラス)。
- 配列内に、IComparable インターフェイスを実装していない要素が含まれている場合 (InvalidOperationException クラス)。

参照

処理手順

方法 : 配列に値を格納する
方法 : 配列から値を取得する
方法 : Visual Basic で、配列内の要素を検索する
方法 : Visual Basic で配列を並べ替える
配列のトラブルシューティング

関連項目

System.Array.Reverse

概念

Visual Basic の配列の概要

その他の技術情報

Visual Basic における配列

方法：Visual Basic で配列を並べ替える

次に示すのは、zooAnimals という名前の **String** オブジェクトの配列を宣言し、値を代入してから、アルファベット順に並べ替える例です。

使用例

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

```
Private Sub sortAnimals()
    Dim zooAnimals(2) As String
    zooAnimals(0) = "lion"
    zooAnimals(1) = "turtle"
    zooAnimals(2) = "ostrich"
    Array.Sort(zooAnimals)
End Sub
```

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- Mscorlib.dll と System 名前空間へのアクセス

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- 空の配列である場合 (ArgumentNullException クラス)。
- 多次元の配列である場合 (RankException クラス)。
- 配列内に、IComparable インターフェイスを実装していない要素が含まれている場合 (InvalidOperationException クラス)。

参照

処理手順

方法 : 配列に値を格納する
方法 : 配列から値を取得する
方法 : Visual Basic で、配列内の要素を検索する
方法 : Visual Basic で配列内の内容を反転させる
配列のトラブルシューティング

関連項目

System.Array.Sort

概念

Visual Basic の配列の概要

その他の技術情報

Visual Basic における配列

方法：配列を別の配列に代入する

配列はオブジェクトであるため、他のオブジェクト型と同じように代入ステートメントで使用できます。配列変数には、配列の要素、ランク、および長さの情報を構成するデータへのポインタが保持されています。代入ではこのポインタのみがコピーされます。

配列を別の配列に代入するには

1. 2 つの配列が同じランク (次元数) であり、要素のデータ型に互換性があることを確認します。
2. 標準の代入ステートメントを使用して、元の配列を代入先の配列に代入します。いずれの配列名の後にも、かっこを指定しないでください。

```
Dim formArray() As System.Windows.Forms.Form
Dim controlArray() As System.Windows.Forms.Control
controlArray = formArray
```

配列を別の配列に代入する場合、次の規則が適用されます。

- **ランクが等しいこと。** 代入先の配列と代入元の配列のランク (次元数) が一致している必要があります。

  両方の配列のランクが一致している場合、要素数は一致していなくてもかまいません。次元の要素の数は、代入するときに変更できます。

- **要素の型が合っていること。** 両方の配列の要素が参照型であるか、両方の配列の要素が値型である必要があります。詳細については、「値型と参照型」を参照してください。

  - 両方の配列の要素が値型である場合、要素のデータ型は完全に一致している必要があります。これには 1 つだけ例外があり、**Enum** 要素の配列を、その **Enum** の基本型の配列に代入することはできます。
  - 両方の配列の要素が参照型である場合、代入元の要素型が代入先の要素型から派生している必要があります。これに該当する場合、2 つの配列は、それらの配列の要素と同じ継承関係にあることになります。これは、配列の共変性と呼ばれます。

データ型に互換性がない場合や、ランクが一致していない場合など、上記の規則に対する違反がある場合は、コンパイラによってエラーが報告されます。コードにエラー処理を追加すると、代入を実行する前に配列の互換性を確認できます。また、例外がスローされるのを回避する必要がある場合は TryCast キーワードを使用することもできます。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列のサイズを変更する
方法 : 配列を他の配列に変更する
方法 : プロシージャまたはプロパティに配列を渡す
方法 : プロシージャまたはプロパティから配列を返す
配列のトラブルシューティング

関連項目

Enum ステートメント (Visual Basic)

概念

配列の変換

その他の技術情報

Visual Basic における配列

方法：配列を他の配列に変更する

配列オブジェクトと配列変数を区別することは重要です。配列変数は、配列オブジェクトへのポインタを保持し、配列要素、ランク、および長さの情報を保持します。

- 配列オブジェクトを一度作成すると、ランク (次元の数)、次元の長さ、その要素のデータ型は変更できません。変更できるのはその要素の内容だけです。
- 配列変数を宣言すると、ランクまたはその要素のデータ型は変更できません。ただし、変数が有効である間は、連続した異なる配列オブジェクトを変数に割り当てることができます。これらの配列オブジェクトは、異なる次元の長さを持つことができます。

異なる配列オブジェクトをポイントするよう配列変数を変更するには

- 標準の代入ステートメントを使用して、代入する配列を代入先の配列に割り当てます。

```
Dim array1(4), array2(19) As String
array2 = array1
```

次元の長さが異なる配列オブジェクトをポイントするように配列変数を変更できますが、データ型が異なる配列オブジェクトをポイントするように配列変数を変更することはできません。つまり、ランクと要素のデータ型は、本来配列変数のデータ型の一部であるため、そのまま同じにしておく必要があります。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列を別の配列に代入する
方法 : プロシージャまたはプロパティに配列を渡す
方法 : プロシージャまたはプロパティから配列を返す
方法 : 配列のサイズを変更する
配列のトラブルシューティング

その他の技術情報

Visual Basic における配列

方法：プロシージャまたはプロパティに配列を渡す

他の変数を渡す場合と同じ方法で、配列を渡します。プロシージャを呼び出したり、プロパティにアクセスするとき、適切な引数で配列変数名を指定します。

配列をプロシージャに渡すには

1. プロシージャのパラメータのいずれかで、同じランク (次元の数) および要素のデータ型を持つ配列が指定されていることを確認してください。
2. 引数リスト内の対応する場所で、配列変数を指定します。配列名の後にかっこを指定しないでください。

```
Public Function findLargest(ByVal numbers() As Double) As Double
    ' Insert code to calculate and return largest number.
End Function
Dim testNumbers() As Double = New Double() {5.0, 3.7, 1.2, 7.6}
Dim largestNumber As Double = findLargest(testNumbers)
```

配列をプロパティに渡すには

1. プロパティのパラメータのいずれかで、同じランク (次元の数) および要素のデータ型を持つ配列が指定されていることを確認してください。
2. 引数リスト内の対応する場所で、配列変数を指定します。配列名の後にかっこを指定しないでください。

```
Public Property bestMatch(ByVal formattedStrings() As String) As Double
    ' Insert Get and Set procedures for number best matching strings.
End Property
Dim testStrings() As String = New String() {}
Dim formattedNumber As Double = bestMatch(testStrings)
```

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列を別の配列に代入する
方法 : 配列を他の配列に変更する
方法 : プロシージャまたはプロパティから配列を返す
配列のトラブルシューティング

その他の技術情報

Visual Basic における配列

方法：プロシージャまたはプロパティから配列を返す

他のデータ型を返すときと同じ方法で、配列を返します。プロシージャまたはプロパティの戻り値の型として、配列型を指定します。

関数プロシージャから配列を返すには

1. Function ステートメント (Visual Basic) で、戻り値の型として、配列型 (ランクおよび要素のデータ型) を指定します。
2. プロシージャ内で、同じランクおよび要素のデータ型を持つローカルの配列変数を宣言します。
3. このローカルの配列変数を Return ステートメント (Visual Basic) に含めます。配列名の後にかっこを指定しないでください。

```
Public Function splitNumber(ByVal number As Double) As Char()
    Dim characters() As Char
    ' Insert code to split number into characters.
    Return characters
End Function
Dim piCharacters() As Char = splitNumber(3.14159265)
```

プロパティから配列を返すには

1. Property ステートメントで、プロパティの型として、配列型 (ランクおよび要素のデータ型) を指定します。
2. プロパティの Get プロシージャ内、または **Get** プロシージャで利用できる場所で、同じランクおよび要素データ型を持つローカルの配列変数を宣言します。
3. このローカルの配列変数を **Return** ステートメントに含めます。配列名の後にかっこを指定しないでください。

```
Private nameList() As String
Public Property stationNames As String()
    Get
        Return nameList
    End Get
    Set(ByVal Value As String())
        ' Insert code to store nameList values.
    End Set
End Property
Dim listOfNames() As String = stationNames
```

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
方法 : 配列を別の配列に代入する
方法 : 配列を他の配列に変更する
方法 : プロシージャまたはプロパティに配列を渡す
配列のトラブルシューティング

その他の技術情報

Visual Basic における配列

配列の代わりとしてのコレクション

コレクションは、通常は オブジェクト型 (Object) の操作に使われますが、任意のデータ型の操作にも使用できます。配列よりもコレクションにアイテムを格納した方が効率的な場合もあります。

配列のサイズを変更する必要がある場合、ReDim ステートメント (Visual Basic) を使用する必要があります。これを行うと、Visual Basic により新しい配列が作成され、前の配列が解放されて廃棄されます。これには、実行時間がかかります。したがって、操作を行う項目の数が頻繁に変更される場合、または必要な項目の最大数を予測できない場合、コレクションを使用するとパフォーマンスが向上します。

コレクションは、新しいオブジェクトを作成したり、既存の要素をコピーする必要がないため、**ReDim** を使用する必要がある配列よりも、短い実行時間でサイズ変更の処理ができます。サイズが変更されない場合、またはほとんど変更されない場合、配列の方がより効率的です。通常どおり、パフォーマンスは個々のアプリケーションに大きく依存します。配列とコレクションの両方を試してみる価値はあります。

専用コレクション

また、.NET Framework には、さまざまなクラス、インターフェイス、および構造体が、一般的および特殊なコレクション用に用意されています。System.Collections および System.Collections.Specialized 名前空間には、ディクショナリ、一覧、キュー、およびスタックを含む、定義および実装が含まれています。System.Collections.Generic 名前空間には、1 つ以上の型の引数を指定できる、これらのジェネリック バージョンが数多く用意されています。

コレクションが特定のデータ型だけの要素を保持する場合、ジェネリック コレクションはタイプ セーフを強制するという点で有利です。ジェネリックの詳細については、「Visual Basic におけるジェネリック型」を参照してください。

例

.NET Framework のジェネリック クラス System.Collections.Generic.List を使用して、`customer` 構造体の一覧コレクションを作成するコード例を次に示します。

```
' Define the structure for a customer.
Public Structure customer
    Public name As String
    ' Insert code for other members of customer structure.
End Structure
' Create a module-level collection that can hold 200 elements.
Public custFile As New List(Of customer)(200)
' Add a specified customer to the collection.
Private Sub addNewCustomer(ByVal newCust As customer)
    ' Insert code to perform validity check on newCust.
    custFile.Add(newCust)
End Sub
' Display the list of customers in the Debug window.
Private Sub printCustomers()
    For Each cust As customer In custFile
        Debug.WriteLine(cust)
    Next cust
End Sub
```

`custFile` コレクションの宣言により、`customer` 型だけの要素を含むことができることを指定します。また、200 要素分の初期容量も用意されています。`addNewCustomer` プロシージャにより、新しい要素の有効性がチェックされ、コレクションに追加されます。`printCustomers` プロシージャでは、コレクションを走査し、その要素を表示するために、**For Each** ループが使用されます。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 配列変数を初期化する
配列のトラブルシューティング

関連項目

ReDim ステートメント (Visual Basic)

概念

Visual Basic におけるコレクション
Visual Basic におけるジェネリック型

その他の技術情報

Visual Basic における配列

配列のトラブルシューティング

ここでは、配列を使用する際に起きる一般的な問題についていくつか説明します。

配列の宣言と初期化に関するコンパイル エラー

配列の宣言、作成、初期化に関する規則をよく理解していないと、コンパイル エラーが発生します。代表的なエラーの原因を次に示します。

- 配列変数の宣言の中で次元のサイズを指定した後に New (Visual Basic) 句を記述している。この種の無効な宣言の例を次に示します。

```
Dim INVALIDsingleDimByteArray(2) As Byte = New Byte()

Dim INVALIDtwoDimShortArray(1, 1) As Short = New Short(,)

Dim INVALIDjaggedByteArray(1)() As Byte = New Byte()()
```

- ジャグ配列のトップ レベル配列よりも大きい次元サイズを指定している。この種の無効な宣言の例を次に示します。

```
Dim INVALIDjaggedByteArray(1)(1) As Byte
```

- 要素値を指定するときに **New** キーワードを省略している。この種の無効な宣言の例を次に示します。

```
Dim INVALIDoneDimShortArray() As Short = Short() {0, 1, 2, 3}
```

- 中かっこ (**{}**) を使用せずに **New** 句を指定している。この種の無効な宣言の例を次に示します。

```
Dim INVALIDsingleDimByteArray() As Byte = New Byte()

Dim INVALIDsingleDimByteArray() As Byte = New Byte(2)

Dim INVALIDtwoDimShortArray(,) As Short = New Short(,)

Dim INVALIDtwoDimShortArray(,) As Short = New Short(1, 1)
```

配列の範囲外へのアクセス

配列の初期化プロセスでは、配列の各次元に上限と下限を割り当てます。配列の要素にアクセスするときには、それぞれの次元について有効なインデックスまたは添字を指定する必要があります。指定したインデックスが下限より小さいか、上限より大きい場合は、IndexOutOfRangeException 例外が発生します。コンパイラはこのようなエラーを検出できないので、実行時にエラーが発生します。

範囲の確認

別のコンポーネントからプロシージャ引数などの形でコードに配列が渡される場合は、その配列のサイズや各次元のサイズはわかりません。配列内の要素にアクセスするときには、事前に配列の各次元の上限を調べる必要があります。配列が Visual Basic の **New** 句以外の方法で作成されている場合は、下限が 0 でない可能性もあるので、下限も調べておいた方が安全です。

次元の指定

多次元配列の範囲を調べるときには、次元の指定方法に注意してください。GetLowerBound メソッドおよび GetUpperBound メソッドの *dimension* パラメータは 0 から始まりますが、Visual Basic の LBound 関数 (Visual Basic) および UBound 関数 (Visual Basic) の *Rank* パラメータは 1 から始まります。

参照

処理手順

方法 : 配列変数を宣言する
方法 : 配列を作成する
方法 : 多次元配列を作成する
方法 : 配列の配列を作成する
方法 : 複数の要素型が混在する配列を作成する
方法 : 要素を持たない配列を作成する
方法 : 配列変数を初期化する
方法 : 多次元配列を初期化する
方法 : ジャグ配列を初期化する

その他の技術情報

Visual Basic における配列

Visual Basic におけるオブジェクト

Visual Basic でアプリケーションを作成する場合は、常にオブジェクトが作業対象になります。コントロール、フォーム、データ アクセス オブジェクトなど、Visual Basic に用意されたオブジェクトを使用できます。また、Visual Basic 内の、他のアプリエーションのオブジェクトも使用できます。さらに、独自のオブジェクトを作成し、作成したオブジェクトにプロパティやメソッドを追加することも可能です。オブジェクトは、プログラムの既成部品として機能します。オブジェクトを使うと、一度記述したコードを繰り返し再利用できます。

以下のトピックでは、オブジェクトについて詳しく説明します。

このセクションの内容

クラスとオブジェクト

オブジェクトとクラスの概要を説明します。

方法 : オブジェクトを作成する

オブジェクトを作成して、そのメンバを使用する方法について説明します。

方法 : 作業用コンポーネントを再利用する

作成とデバッグが終了している機能を使用する方法について説明します。

方法 : 既存クラスのメンバを使用するクラスを定義する

基本クラスのプロパティとメソッドを、派生クラスで使用する方法について説明します。

オブジェクト間の関係

オブジェクト階層、コンテナ、およびコレクションについて説明します。

オブジェクト メンバ

オブジェクトによって公開されるフィールド、プロパティ、メソッド、およびイベントについてまとめています。

方法 : オブジェクトの共有メンバおよび非共有メンバにアクセスする

作成したオブジェクトのメンバの読み取り、書き込み、呼び出し方法について説明します。

関連するセクション

Visual Basic のオブジェクトの概要

Visual Basic におけるオブジェクトの使い方の概要を説明します。

オブジェクトの作成と使用

クラスのインスタンスの作成方法および使用方法について説明します。

クラスとオブジェクト

オブジェクトとは、1 つの単位として扱うことのできるコードとデータの組み合わせです。オブジェクトは、コントロールやフォームのようなアプリケーションの一部である場合もあります。また、アプリケーション全体が 1 つのオブジェクトである場合もあります。

コードの再利用

オブジェクトを使うと、一度宣言した変数やプロシージャをいつでも必要なときに再利用できます。たとえば、アプリケーションにスペル チェック機能を追加する場合は、スペル チェック機能に必要なすべての変数やサポート関数を定義することになります。スペル チェック プログラムをクラスとして作成すると、コンパイル済みアセンブリへの参照を追加するだけで、このスペル チェック プログラムを他のアプリケーションで再利用できるようになります。さらに、既に他の人によって開発されたスペル チェック クラスを使うと、その分の開発の手間を省くことができます。

クラス

Visual Basic の各オブジェクトは、クラスによって定義されます。クラスによって、オブジェクトの変数、プロパティ、プロシージャ、およびイベントが説明されます。オブジェクトは、クラスのインスタンスです。クラスを定義すると、必要なだけオブジェクトを作成できます。

オブジェクトとクラスの関係は、cookie と cookie の抜き型にたとえることができます。cookie の抜き型はクラスです。抜き型は、大きさや形など、それぞれの cookie の特徴を定義します。このクラスを基にオブジェクトを作成します。このオブジェクトに相当するのが cookie です。

クラスとオブジェクトとの関係を説明する、Visual Basic の 2 つの例を示します。

- Visual Basic のツールボックスにあるコントロールは、クラスを表します。ツールボックスからフォームにコントロールをドラッグすると、クラスのインスタンスであるオブジェクトが作成されます。
- デザイン時に使うフォームはクラスです。実行時に、Visual Basic によって、フォームのクラスのインスタンスが作成されます。これがオブジェクトです。

複数のインスタンス

クラスから新しく作成されたオブジェクトは、互いに同一であることがよくあります。ただし、個々のオブジェクトとして存在した後は、他のインスタンスと関係なく、変数およびプロパティを変更できます。たとえば、フォームに 3 つのチェック ボックスを追加した場合、各チェック ボックスのオブジェクトは CheckBox クラスのインスタンスです。各 **CheckBox** オブジェクトは、クラスで定義されている同じ特性や機能 (プロパティ、変数、プロシージャ、およびイベント) を共有します。その一方で、各オブジェクトはそれぞれ固有の名前を持ち、個別に有効または無効にしたり、フォーム上の異なる場所に配置したりできます。

参照

概念

オブジェクト間の関係

Visual Basic のオブジェクトの概要

その他の技術情報

オブジェクトの作成と使用

クラスについて

方法：オブジェクトを作成する

クラスのインスタンスはオブジェクトです。クラスのメンバを使用するには、まずそのクラスからオブジェクトを作成する必要があります。

クラスからオブジェクトを作成するには

1. オブジェクトの作成元のクラスを決定します。
2. Dim ステートメント (Visual Basic) を記述して、クラス インスタンスを代入する変数を作成します。変数の型は、目的のクラスの型にする必要があります。

```
Dim nextCustomer As customer
```

3. New (Visual Basic) キーワードを追加して、変数をそのクラスの新しいインスタンスに初期化します。

```
Dim nextCustomer As New customer
```

4. これで、オブジェクト変数を介してそのクラスのメンバにアクセスできます。

```
nextCustomer.accountNumber = lastAccountNumber + 1
```

堅牢性の高いプログラム

可能な限り、変数は代入するクラスの型を使用して宣言してください。これは、事前バインディングと呼ばれます。コンパイル時にクラス型がわからない場合は、変数を オブジェクト型 (Object) で宣言しておき、遅延バインディングを行うことで対応できます。ただし、遅延バインディングではパフォーマンスが遅くなり、また、ランタイム オブジェクトのメンバに対するアクセスが制限されます。詳細については、「オブジェクト変数の宣言」を参照してください。

参照

処理手順

方法 : 作業用コンポーネントを再利用する
方法 : 既存クラスのメンバを使用するクラスを定義する
方法 : オブジェクトの共有メンバおよび非共有メンバにアクセスする

概念

クラスとオブジェクト

方法：作業用コンポーネントを再利用する

すでにコンポーネントが存在し、デバッグもされて、動作中の場合は、同じ機能の別のコンポーネントを開発する代わりにそのコンポーネントをコード内で使用すると便利です。そのようなコンポーネントは、通常、クラスとして公開されます。それを再利用するには、そのクラスからオブジェクトを作成します。

使用例

.NET Framework には、利用可能なコンポーネントの例が数多くあります。そのようなコンポーネントの 1 つに System 名前空間の TimeZone クラスがあります。**TimeZone** には、現在のコンピュータ システムのタイム ゾーンに関する情報を取得できるメンバがあります。

```
Public Sub examineTimeZone()
    Dim tz As System.TimeZone = System.TimeZone.CurrentTimeZone
    Dim s As String = "Current time zone is "
    s &= CStr(tz.GetUtcOffset(Now).Hours) & " hours and "
    s &= CStr(tz.GetUtcOffset(Now).Minutes) & " minutes "
    s &= "different from UTC (coordinated universal time)"
    s &= vbCrLf & "and is currently "
    If tz.IsDaylightSavingTime(Now) = False Then s &= "not "
    s &= "on ""summer time""."
    MsgBox(s)
End Sub
```

最初の Dim ステートメント (Visual Basic) では、**TimeZone** 型のオブジェクト変数を宣言し、これを CurrentTimeZone プロパティで返される **TimeZone** オブジェクトに割り当てます。

参照

処理手順

方法 : オブジェクトを作成する
方法 : 既存クラスのメンバを使用するクラスを定義する
方法 : オブジェクトの共有メンバおよび非共有メンバにアクセスする

概念

クラスとオブジェクト

方法：既存クラスのメンバを使用するクラスを定義する

既存クラスのメンバは、そのクラスから派生した別のクラスでも使用できます。

次の例では、通常の Button のように動作し、前景色と背景色とを反転させるメソッドを公開する、特殊な **Button** を定義します。

既存クラスのメンバを使用するクラスを定義するには

1. Class ステートメント (Visual Basic) を使用して、必要なオブジェクトの作成元となるクラスを定義します。

   ```
   Public Class reversibleButton
   ```

   クラスの最後のコード行の後には、必ず **End Class** ステートメントを指定してください。既定では、**Class** ステートメントを入力したときに、統合開発環境 (IDE: Integrated Development Environment) によって自動的に **End Class** が挿入されます。

2. **Class** ステートメントの直後に Inherits ステートメント を指定します。新しいクラスの派生元のクラスを指定します。

   ```
   Inherits System.Windows.Forms.Button
   ```

   新しいクラスには、基本クラスで定義されているすべてのメンバが継承されます。

3. 派生クラスで公開する追加のメンバ用のコードを追加します。たとえば、`reverseColors` メソッドを追加する場合、派生クラスは次のようになります。

   ```
   Public Class reversibleButton
       Inherits System.Windows.Forms.Button
       Public Sub reverseColors()
           Dim saveColor As System.Drawing.Color = Me.BackColor
           Me.BackColor = Me.ForeColor
           Me.ForeColor = saveColor
       End Sub
   End Class
   ```

   この `reversibleButton` クラスのオブジェクトを作成すると、作成したオブジェクトでは、**Button** クラスのすべてのメンバに加え、`reversibleButton` で定義した `reverseColors` メソッドなどの新しいメンバにアクセスできます。

コードのコンパイル方法

コンパイラでは、新規クラスの派生元とするクラスにアクセスできる必要があります。このためには、上の例で示したように名前を完全に修飾することや、Imports ステートメント で名前空間を指定することが必要になる場合があります。クラスが別のプロジェクト内にある場合は、そのプロジェクトへの参照を追加することも必要になる場合があります。詳細については、「名前空間およびコンポーネントの参照」を参照してください。

参照

処理手順

方法 : オブジェクトを作成する
方法 : 作業用コンポーネントを再利用する
方法 : オブジェクトの共有メンバおよび非共有メンバにアクセスする
方法 : 派生クラスを作成する

概念

クラスとオブジェクト
継承の基本

オブジェクト間の関係

オブジェクトを互いに関連付ける方法はいくつかあります。主な関係として階層とコンテインメントがあります。

階層関係

より基本的なクラスから別のクラスが派生している場合、これらのクラスの間には階層関係があるといいます。クラスの階層構造は、より一般的なクラスの内部処理形式である項目を記述する時に便利です。たとえば、System.Windows.Forms 名前空間では、Label クラスと TextBox クラスは両方とも Control クラスから派生しています。派生クラスは、派生元のクラスからメンバを継承するため、クラスの階層構造のレベルが深くなるにつれてより複雑なクラスを作成できます。

コンテインメント関係

オブジェクト間の別の関係として、コンテインメント関係があります。コンテナ オブジェクトとは、論理的に他のオブジェクトをカプセル化するオブジェクトです。たとえば、OperatingSystem オブジェクトは論理的に Version オブジェクト (Version プロパティによって返される) を含んでいます。コンテナ オブジェクトが物理的に他のオブジェクトを含んでいるわけではないことに注意してください。

コレクション

オブジェクトの実際のコンテインメントを表すものの 1 つがコレクションです。コレクションは、列挙できる、類似したオブジェクトのグループです。Visual Basic の For Each...Next ステートメント (Visual Basic) には、コレクションの項目を反復処理するための特別な構文があります。さらに、コレクションでは、多くの場合、Item プロパティ (Collection オブジェクト) を使って、インデックスや関連付けた一意の文字列で要素を取得できます。コレクションは、インデックスを使わずに項目を追加したり削除したりできるため、配列よりも使い方が簡単です。使い方が簡単なため、フォームやコントロールの格納によく使われます。

参照

概念

オブジェクト メンバ
Visual Basic におけるコレクション
Visual Basic のオブジェクトとその他のオブジェクト

その他の技術情報

継承階層のデザイン
オブジェクトの作成と使用

オブジェクト メンバ

オブジェクトは、オブジェクト指向プログラミングの基本単位です。オブジェクトは、アプリケーションの要素であり、クラスのインスタンスです。フィールド、プロパティ、メソッド、およびイベントは、オブジェクトの構成要素であり、オブジェクトのメンバを構成します。

オブジェクト

オブジェクトは、Form または Label などのクラスのインスタンスを表します。非共有メンバにアクセスする前に、オブジェクトを作成する必要があります。これを行うには、New キーワードを使用して、オブジェクトを作成するクラスを指定します。その後、新しいオブジェクトをオブジェクト変数に代入します。

```
Dim warningLabel As New System.Windows.Forms.Label
```

詳細については、「方法 : オブジェクトを作成する」を参照してください。

メンバ アクセス

オブジェクト変数の名前、ピリオド (.)、およびメンバの名前の順で指定して、オブジェクトのメンバにアクセスします。**Label** オブジェクトの Text プロパティを設定する例を次に示します。

```
warningLabel.Text = "Data not saved"
```

フィールドとプロパティ

プロパティとフィールドは、オブジェクトに格納されている情報を表します。プロシージャでローカル変数を取得および設定する場合と同じ方法で、代入ステートメントを使用して値を取得および設定します。**Label** オブジェクトの Width プロパティを取得し、ForeColor プロパティを設定する例を次に示します。

```
Dim warningWidth As Integer = warningLabel.Width
warningLabel.ForeColor = System.Drawing.Color.Red
```

フィールドもメンバ変数と呼ばれます。

詳細については、「プロパティ プロシージャとフィールド」を参照してください。

メソッド

メソッドは、オブジェクトが実行できる処理です。たとえば、Add は、コンボ ボックスに新しいエントリを追加する、ComboBox オブジェクトのメソッドです。

Timer オブジェクトの Start メソッドの例を次に示します。

```
Dim safetyTimer As New System.Windows.Forms.Timer
safetyTimer.Start()
```

メソッドとは、オブジェクトによって公開されるプロシージャです。

詳細については、「方法 : メソッドを使用してアクションを実行する」を参照してください。

イベント

イベントとは、オブジェクトによって認識されるアクション (マウス クリックやキー入力など) であり、それに応答するためのコードを記述できます。イベントは、ユーザーによる操作やプログラム コードの結果として発生する場合と、システムによって発生する場合があります。イベントを通知するコードの場合はイベントを発生させると言い、それに応答するコードの場合は処理すると言います。

オブジェクトが発生させるカスタム イベントを独自に作成し、他のオブジェクトに処理させることもできます。詳細については、「イベントとイベント ハンドラ」を参照してください。

インスタンス メンバおよび共有メンバ

クラスからオブジェクトを作成するとき、結果はそのクラスのインスタンスになります。Shared (Visual Basic) キーワードを使用して宣言されていないメンバは、厳密な意味で特定のインスタンスに属しているインスタンス メンバです。あるインスタンスのインスタンス メンバは、同じクラスの他のイ

ンスタンスにある同じメンバとは無関係です。たとえば、インスタンス メンバ変数は異なるインスタンスで異なる値を持つことができます。

Shared キーワードを使用して宣言されたメンバは、特定のインスタンスにではなく、クラス全体に属している共有メンバです。作成したクラスのインスタンスの数にかかわらず、またインスタンスを作成していない場合でも、共有メンバは 1 つだけ存在します。たとえば、共有メンバ変数は、クラスにアクセスできるすべてのコードが利用できる値を 1 つだけ持っています。

メンバを一覧表示する IntelliSense

[メンバの一覧] オプションを起動すると、たとえば、メンバ アクセス演算子としてピリオド (.) を入力したときに、IntelliSense によってクラスのメンバが一覧表示されます。クラスのインスタンスとして宣言された変数の名前の後にピリオドを入力した場合、IntelliSense によってすべてのインスタンス メンバが一覧表示されますが、共有メンバは表示されません。クラス名 自体の後にピリオドを入力した場合、IntelliSense によってすべての共有メンバが一覧表示されますが、インスタンス メンバは表示されません。詳細については、「IntelliSense の使用方法」を参照してください。

参照

概念

オブジェクト間の関係

その他の技術情報

Visual Basic におけるオブジェクト

方法：オブジェクトの共有メンバおよび非共有メンバにアクセスする

オブジェクトが作成されると、オブジェクトの変数を使用して、オブジェクトのフィールド、プロパティ、メソッド、およびイベントにアクセスできます。メンバが Shared (Visual Basic) である場合、メンバにアクセスするためのオブジェクトを作成する必要はありません。

非共有メンバへのアクセス

オブジェクトの非共有メンバにアクセスするには

1. オブジェクトがそのクラスから作成され、オブジェクト変数に割り当てられていることを確認してください。

```
Dim secondForm As New System.Windows.Forms.Form
```

2. メンバにアクセスするためのステートメントで、オブジェクト変数名の後にメンバ アクセス演算子 (.)、次にメンバ名を続けて記述します。

```
secondForm.Show()
```

共有メンバへのアクセス

オブジェクトの共有メンバにアクセスするには

- クラス名の後にメンバ アクセス演算子 (.)、次にメンバ名を続けて記述します。常にクラス名を使用して、オブジェクトの **Shared** メンバに直接アクセスする必要があります。

```
MsgBox("This computer is called " & Environment.MachineName)
```

- 既にクラスからオブジェクトを作成してある場合、オブジェクト変数を使用して、**Shared** メンバにアクセスすることもできます。

参照

処理手順

方法 : オブジェクトを作成する
方法 : 作業用コンポーネントを再利用する
方法 : 既存クラスのメンバを使用するクラスを定義する
方法 : オブジェクトのメンバにアクセスする

概念

オブジェクト メンバ

Visual Basic における文字列

ここでは、Visual Basic での文字列の使用における基本的な考え方について説明します。

このセクションの内容

Visual Basic の文字列の概要

Visual Basic での文字列の使用における基本的な考え方について説明したトピックをいくつか紹介します。

Visual Basic における文字列の作成

新しい文字列の作成方法について説明したトピックをいくつか紹介します。

Visual Basic で、文字列型とその他のデータ型との変換を行う

文字列を他のデータ型に変換する方法について説明したトピックをいくつか紹介します。

Visual Basic の文字列に関するチュートリアル

文字列の操作に関するチュートリアルの一覧です。

Visual Basic における文字列の検証

文字列の検証方法について説明したトピックをいくつか紹介します。

Visual Basic における文字列の検索と置換

文字列の検索と置換を使用する方法について説明したトピックをいくつか紹介します。

Visual Basic における文字列の解析

文字列の解析方法について説明したトピックをいくつか紹介します。

Visual Basic における文字列のトラブルシューティング

文字列操作におけるトラブルシューティングの方法についてのアドバイスを示したトピックをいくつか紹介します。

参照

その他の技術情報

Visual Basic 言語の機能

Visual Basic の文字列の概要

ここでは、Visual Basic における文字列の使用に関する基本概念を説明します。

このセクションの内容

Visual Basic における文字列の基本

文字列と文字列変数の使用に関する基本概念を説明します。

Visual Basic における文字列操作メソッドの種類

文字列を分析および操作するさまざまな方法を説明します。

Visual Basic の Nothing と文字列

Visual Basic ランタイムと .NET Framework では、文字列と組み合わせたときに **Nothing** の評価方法が異なることを説明します。

Visual Basic においてカルチャが文字列に与える影響

Visual Basic で文字列の変換と比較にカルチャ情報がどのように使用されるかを説明します。

Visual Basic における文字列アクセスのインデックス番号

文字列に含まれる文字にアクセスする方法を Visual Basic と .NET Framework で比較します。

参照

その他の技術情報

Visual Basic における文字列
Visual Basic における文字列の作成

Visual Basic における文字列の基本

文字列型 (**String**) は、一連の文字を表します。文字列型のそれぞれの文字は、char 型 (**Char**) のインスタンスを表します。ここでは、Visual Basic における文字列の基本的な考え方について説明します。

文字列変数

文字列型のインスタンスには、一連の文字を表すリテラル値を代入できます。次に例を示します。

VB

```
Dim MyString As String
MyString = "This is an example of the String data type"
```

文字列型 (**String**) の変数には、文字列に評価される式を代入することもできます。次に例を示します。

VB

```
Dim OneString As String
Dim TwoString As String
OneString = "one, two, three, four, five"
' Evaluates to "two".
TwoString = OneString.Substring(5, 3)
OneString = "1"
' Evaluates to "11".
TwoString = OneString & "1"
```

文字列型 (**String**) の変数にリテラルを代入する場合は、二重引用符 ("") で囲む必要があります。このため、文字列内の二重引用符は、二重引用符で表現できません。たとえば、次のコードはコンパイラ エラーになります。

VB

```
Dim myString As String

' This line would cause an error.
' myString = "He said, "Look at this example!""
```

このコードがエラーになるのは、2 つ目の二重引用符の後でコンパイラが文字列を終了し、残りの文字列がコードとして解釈されるためです。この問題を解決するため、Visual Basic は二重引用符が文字列リテラルに 2 つ続けて出現した場合に、文字列内の 1 つの二重引用符として解釈します。次の例は、文字列に二重引用符を含める正しい方法を示しています。

VB

```
' The value of myString is: He said, "Look at this example!"
myString = "He said, ""Look at this example!"""
```

次の例では、`Look` という単語の前にある 2 つの二重引用符が、文字列内の 1 つの二重引用符になります。行末の 3 つの二重引用符は、文字列内の 1 つの二重引用符と文字列の終了文字を表しています。

文字列の文字

文字列は一連の char 型 (**Char**) の値と見なすことができるため、文字列型 (**String**) には、文字列を配列と同じように操作できる組み込み関数があります。.NET Framework のすべての配列と同様に、これらの配列でもインデックス番号が 0 から始まります。**Chars** プロパティで文字列内の位置を指定すると、文字列内の特定の文字を参照できます。次に例を示します。

VB

```
Dim myString As String = "ABCDE"
Dim myChar As Char
' The value of myChar is "C".
myChar = myString.Chars(3)
```

上の例では、**Chars** プロパティによって文字列の 4 番目の文字 `D` が返され、`myChar` に代入されています。また、**Length** プロパティを使うと、特定の文字列の長さを調べることができます。文字列に対して配列型の操作をいくつも実行する必要がある場合は、文字列の **ToCharArray** 関数を使用して char 型 (**Char**) のインスタンスの配列に文字列を変換します。次に例を示します。

VB

```
Dim myString As String = "abcdefghijklmnop"
Dim myArray As Char() = myString.ToCharArray
```

これにより、変数 `myArray` の内容は、`myString` の各文字を表す char 型 (**Char**) の値の配列になります。

文字列の不変性

文字列は不変です。つまり、いったん作成してしまうと、後から値を変更できません。ただし、だからといって文字列変数に 2 つ以上の値を代入できないわけではありません。次に例を示します。

VB

```
Dim myString As String = "This string is immutable"
myString = "Or is it?"
```

この例では文字列変数が作成され、値が代入された後、その値が変更されています。

具体的に説明すると、最初の行で文字列型 (**String**) のインスタンスが作成され、`This string is immutable` という値が代入されています。2 行目では、新しいインスタンスが作成されて `Or is it?` という値が代入されます。その後、文字列変数は最初のインスタンスへの参照を破棄し、新しいインスタンスへの参照を格納します。

ほかの組み込みデータ型とは違って、文字列型 (**String**) は参照型です。参照型の変数が引数として関数やサブルーチンに渡されると、文字列の実際の値ではなく、データが格納されているメモリアドレスへの参照が渡されます。したがって、上の例では、変数の名前は同じですが、参照されているのは **String** クラスの新しい別のインスタンスであり、この新しいインスタンスが新しい値を保持しています。

参照

関連項目

文字列型 (String) (Visual Basic)
文字型 (Char) (Visual Basic)

その他の技術情報

Visual Basic の文字列の概要
基本的な文字列操作

Visual Basic における文字列操作メソッドの種類

文字列を分析し、操作する方法はいくつかあります。ここで紹介するメソッドには、Visual Basic 言語の一部であるメソッドと、**String** クラス固有のメソッドがあります。

Visual Basic 言語と .NET Framework

Visual Basic のメソッドは、言語に固有の関数として使用されます。したがって、コード内で修飾子を付けずに使用できます。次のコードは、Visual Basic の文字列操作コマンドの典型的な使用例です。

VB

```
Dim aString As String = "SomeString"
Dim bString As String
bString = Mid(aString, 3, 3)
```

この例では、**Mid** 関数が `aString` を直接処理し、値を `bString` に代入しています。

共有メソッドとインスタンス メソッド

String クラスのメソッドを使って文字列を操作することもできます。**String** には、共有メソッドとインスタンス メソッドの 2 種類のメソッドがあります。

共有メソッド

共有メソッドは、**String** クラス自体のメソッドであり、クラスのインスタンスを必要としません。これらのメソッドを修飾するには、**String** クラスのインスタンスではなく、クラスの名前 (**String**) を使います。たとえば、次のようにします。

VB

```
Dim aString As String = String.Copy("A literal string")
```

この例の System.String.Copy(System.String) メソッドは静的メソッドであり、指定された式を処理し、結果値を `bString` に代入します。

インスタンス メソッド

これに対して、インスタンス メソッドは、**String**の特定のインスタンスのメソッドであり、インスタンス名で修飾する必要があります。たとえば、次のようにします。

VB

```
Dim aString As String = "A String"
Dim bString As String
' Assign "String" to bString.
bString = aString.Substring(2, 6)
```

この例の System.String.Substring(System.Int32) メソッドは、**String** のインスタンス (`aString`) のメソッドです。`aString` に対して処理を実行し、値を `bString` に代入します。

参照

その他の技術情報

Visual Basic の文字列の概要

Visual Basic の Nothing と文字列

Visual Basic ランタイムと .NET Framework とでは、文字列に格納された **Nothing** の扱い方が違います。

Visual Basic ランタイムと .NET Framework

次に例を示します。

VB

```
Dim MyString As String = "This is my string"
Dim stringLength As Integer
' Explicitly set the string to Nothing.
MyString = Nothing
' stringLength = 0
stringLength = Len(MyString)
' This line, however, causes an exception to be thrown.
stringLength = MyString.Length
```

Visual Basic ランタイムは通常、**Nothing** を空の文字列 ("") として扱います。一方、.NET Framework では、**Nothing** に対して文字列操作を実行しようとすると、例外がスローされます。

参照

その他の技術情報

Visual Basic の文字列の概要

Visual Basic においてカルチャが文字列に与える影響

ここでは、Visual Basic が文字列の変換と比較を実行するときにカルチャ情報をどのように使用するかを説明します。

カルチャ固有文字列を使用する状況

通常は、ユーザーとやり取りするデータにはカルチャ固有文字列を使用し、アプリケーションの内部データにはカルチャ不変文字列を使用します。

たとえば、日付を文字列として入力するようユーザーに要求する場合、アプリケーションは、ユーザーが自分のカルチャに従った形式で文字列を入力するだろうということを予測し、その文字列を適切に変換する必要があります。さらに、その日付をユーザー インターフェイス上に表示するときには、ユーザーのカルチャに従った形式で表示する必要があります。

しかし、その日付を中央のサーバーにアップロードする場合には、日付形式の相違による混乱を防ぐために、その文字列を特定のカルチャに従った形式にする必要があります。

カルチャに依存する関数

Visual Basic のすべての文字列変換関数 (**Str** 関数と **Val** 関数を除く) は、アプリケーションのカルチャ情報に基づいて、アプリケーション ユーザーのカルチャに適した変換および比較を行います。

さまざまなカルチャ設定がされたコンピュータ上で実行されるアプリケーションを開発するときに、そのアプリケーション内で文字列変換関数を正しく使用するための鍵は、どの関数が特定のカルチャ設定を使用して、どの関数が現在のカルチャ設定を使用するかを理解することです。アプリケーションのカルチャ設定は、既定ではオペレーティング システムのカルチャ設定を継承するという点に注意してください。詳細については、「Asc 関数、AscW 関数」、「Chr 関数、ChrW 関数」、「Format 関数」、「Hex 関数 (Visual Basic)」、「Oct 関数」、および「データ型変換関数」を参照してください。

Str 関数 (数値を文字列に変換) と **Val** 関数 (文字列を数値に変換) は、文字列と数値の間で変換を行うときにアプリケーションのカルチャ情報を使用しません。その代わりに、ピリオド (.) だけを有効な小数点の記号として認識します。これらの関数と同様の処理を行うカルチャ対応関数には次のものがあります。

- **現在のカルチャを使用する変換。CStr** 関数と **Format** 関数は数値を文字列に変換し、**CDbl** 関数と **CInt** 関数は文字列を数値に変換します。
- **特定のカルチャを使用する変換。**それぞれの数値オブジェクトには、数値を文字列に変換する `ToString(IFormatProvider)` メソッドと、文字列を数値に変換する `Parse(String, IFormatProvider)` メソッドがあります。たとえば **Double** 型には ToString メソッドと Parse メソッドがあります。

詳細については、「Str 関数」および「Val 関数」を参照してください。

特定のカルチャの使用

たとえば、日付を文字列形式にして Web サービスに送信するアプリケーションを開発しているとします。このアプリケーションでは、文字列変換の際に特定のカルチャを使用する必要があります。この理由を説明するために、日付の ToString メソッドを使用した場合の結果を考えてみましょう。このメソッドを使用して 20005 年 7 月 4 日という日付を変換した場合、United States English (en-US) のカルチャで実行すると "7/4/2005 12:00:00 AM" という結果になりますが、German (de-DE) のカルチャで実行すると "04.07.2005 00:00:00" という結果になります。

特定カルチャの形式で文字列変換を実行する必要があるときは、.NET Framework に組み込まれている **CultureInfo** クラスを使用します。特定のカルチャに関する **CultureInfo** オブジェクトを新規作成するには、CultureInfo コンストラクタにカルチャの名前を渡します。サポートされるカルチャの名前については、CultureInfo クラスのページを参照してください。

別の方法として、System.Globalization.CultureInfo.InvariantCulture プロパティからインバリアント カルチャのインスタンスを取得することもできます。インバリアント カルチャは English カルチャに基づいていますが、いくつかの相違点があります。たとえば、インバリアント カルチャでは、12 時間制ではなく 24 時間制が指定されます。

日付を特定カルチャの文字列に変換するには、その日付オブジェクトの ToString メソッドに **CultureInfo** オブジェクトを渡します。たとえば次のコードでは、日付はアプリケーションのカルチャ設定に関係なく常に "07/04/2005 00:00:00" と表示されます。

```
Dim d As Date = #7/4/2005#
MsgBox(d.ToString(System.Globalization.CultureInfo.InvariantCulture))
```

メモ：

日付リテラルは常に English カルチャに従って解釈されます。

文字列の比較

文字列の比較が必要な状況としては、主に次の 2 つがあります。

- **ユーザーに表示するデータを並べ替える。**現在のカルチャに基づく演算を行い、文字列が適切に並べ替えられるようにします。
- **アプリケーション内部の 2 つの文字列が完全に一致するかどうかを確認する (通常はセキュリティ目的で使用)。**現在のカルチャを無視した演算を行います。

Visual Basic の StrComp 関数では、両方のタイプの比較を実行できます。比較の種類は、オプションの *Compare* 引数を指定することによって制御できます。通常の入出力では Text を使用し、厳密な比較が要求される場合は Binary を使用します。

StrComp 関数は、2 つの文字列を並べ替え順序に基づいて比較した結果を表す整数を返します。結果が正の場合は、1 番目の文字列の方が 2 番目の文字列より大きいことになります。結果が負の場合は 1 番目の文字列の方が小さく、0 の場合は 2 つの文字列が等しいことになります。

VB

```
' Defines variables.
Dim TestStr1 As String = "ABCD"
Dim TestStr2 As String = "abcd"
Dim TestComp As Integer
' The two strings sort equally. Returns 0.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Text)
' TestStr1 sorts after TestStr2. Returns -1.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Binary)
' TestStr2 sorts before TestStr1. Returns 1.
TestComp = StrComp(TestStr2, TestStr1)
```

.NET Framework に用意されている、**StrComp** 関数と同様の機能を持つ System.String.Compare(System.String,System.String) メソッドを使用することもできます。このメソッドは、基本文字列クラスのオーバーロードされた静的メソッドです。次のコードは、このメソッドの使用例です。

VB

```
Dim myString As String = "Alphabetical"
Dim secondString As String = "Order"
Dim result As Integer
result = String.Compare(myString, secondString)
```

比較の実行方法をより細かく制御するには、**Compare** メソッドのオーバーロードを使用します。System.String.Compare(System.String,System.String,System.StringComparison) メソッドでは、*comparisonType* 引数を使用して比較の種類を指定できます。

***comparisonType* 引数の値**	**比較の種類**	**使用する状況**
Ordinal	文字列のコンポーネント バイトに基づいて比較します。	大文字と小文字を区別する識別子、セキュリティ関連の設定、またはバイトが正確に一致する必要があるその他の非言語的識別子を比較するときに使用します。
OrdinalIgnoreCase	文字列のコンポーネント バイトに基づいて比較します。 **OrdinalIgnoreCase** では、2 つの文字の大文字と小文字だけが違っているときに、インバリアント カルチャ情報を使用して比較結果を判断します。	大文字と小文字を区別する識別子、セキュリティ関連の設定、および Windows に格納されているデータを比較するときに使用します。
CurrentCulture または CurrentCultureIgnoreCase	現在のカルチャでの文字列の解釈に基づいて比較します。	ユーザーに表示するデータ、ほとんどのユーザー入力、および言語的解釈を必要とするその他のデータを比較するときに使用します。

InvariantCulture または InvariantCultureIgnoreCase	インバリアント カルチャでの文字列の解釈に基づいて比較します。 これは **Ordinal** や **OrdinalIgnoreCase** とは異なります。インバリアント カルチャは、自身の許容範囲外の文字を等価のインバリアント文字として扱うからです。	永続データを比較するときや、固定の並べ替え順序を必要とする言語関連のデータを表示するときにのみ使用します。

セキュリティの考慮事項

比較処理または大文字/小文字変換処理の結果に基づいてアプリケーションのセキュリティ関連の決定を下す場合は、その処理を **System.String.Compare(System.String,System.String,System.StringComparison)** メソッドで実行し、*comparisonType* 引数に **Ordinal** または **OrdinalIgnoreCase** を渡してください。

参照

関連項目

データ型変換関数

CultureInfo

その他の技術情報

Visual Basic の文字列の概要

Visual Basic における文字列アクセスのインデックス番号

ここでは、Visual Basic と .NET Framework における、文字列内の文字へのアクセス方法の違いについて説明します。.NET Framework では常に、文字列内の文字へのアクセスに 0 から始まるインデックス番号を使うのに対し、Visual Basic の場合は、関数によってインデックス番号が 0 から始まる場合と 1 から始まる場合があります。

1 から始まるインデックス番号

インデックス番号が 1 から始まる Visual Basic の関数の例に、**Mid** 関数があります。この関数の引数では、文字列内の部分文字列の開始位置を 1 から始まるインデックス番号で示します。一方、.NET Framework の System.String.Substring(System.Int32) メソッドの引数では、同じ開始位置を 0 から始まるインデックス番号で示します。したがって、文字列 "ABCDE" の各文字の番号は、**Mid** 関数では 1、2、3、4、5 になるのに対し、**System.String.Substring(System.Int32)** メソッドでは 0、1、2、3、4 になります。

0 から始まるインデックス番号

インデックス番号が 0 から始まる Visual Basic の関数の例に、**Split** 関数があります。この関数は文字列を分割し、部分文字列を格納した配列を返します。.NET Framework の System.String.Split(System.Char[]) メソッドも文字列を分割し、部分文字列を格納した配列を返します。**Split** 関数と **Split** メソッドは .NET Framework の配列を返すため、インデックス番号はどちらも 0 から始まる必要があります。

参照

処理手順

コレクションのトラブルシューティング

関連項目

Mid 関数 (Visual Basic)
Split 関数 (Visual Basic)
Substring
Split

その他の技術情報

Visual Basic の文字列の概要

Visual Basic における文字列の作成

ここでは、Visual Basic で文字列を作成する方法について説明します。

このセクションの内容

方法 : Visual Basic の StringBuilder を使用して文字列を作成する

StringBuilder クラスを使用して、多数の短い文字列から長い文字列を作成します。

方法 : 複数行のリテラル文字列を生成する (Visual Basic)

3 行のリテラル文字列を作成します。

参照

概念

StringBuilder クラスの使用

方法：Visual Basic の StringBuilder を使用して文字列を作成する

この例では、StringBuilder クラスを使って、短い文字列から長い文字列を作成します。**StringBuilder** クラスは、多くの文字列を結合する場合、**&=** 演算子よりも効率的です。

使用例

次の例では、**StringBuilder** クラスのインスタンスを作成し、1000 の文字列をそのインスタンスに結合し、最終的な文字列を返します。

VB

```
Private Function StringBuilderTest() As String
    Dim builder As New System.Text.StringBuilder
    For i As Integer = 1 To 1000
        builder.Append("Step " & i & vbCrLf)
    Next
    Return builder.ToString
End Function
```

参照

関連項目

&= 演算子 (Visual Basic)

概念

StringBuilder クラスの使用
新しい文字列の作成
String のサンプル

その他の技術情報

Visual Basic における文字列
文字列の操作
Visual Basic における文字列の作成

方法：複数行のリテラル文字列を生成する (Visual Basic)

次に示すのは、3 行のリテラル文字列を構築する例です。

使用例

VB

```
Dim MyString As String
MyString = "This is the first line of my string." & VbCrLf & _
           "This is the second line of my string." & VbCrLf & _
           "This is the third line of my string."
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [データ型 - Visual Basic によって定義済み] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System 名前空間を指定する **Imports** ステートメントです。詳細については、「Imports ステートメント」を参照してください。

参照

概念

文字列の .NET Framework データ型への変換
新しい文字列の作成
String のサンプル

その他の技術情報

Visual Basic における文字列
文字列の解析
文字列の操作
Visual Basic における文字列の作成

Visual Basic で、文字列型とその他のデータ型との変換を行う

このセクションでは、文字列をその他のデータ型に変換する方法について説明します。

このセクションの内容

方法 : Visual Basic でバイトの配列を文字列に変換する

バイト配列のバイトを文字列に変換する方法。

方法 : Visual Basic で文字列をバイトの配列に変換する

文字列をバイト配列に変換する方法。

方法 : Char 値の配列から文字列を作成する (Visual Basic)

個別の文字から文字列 "abcd" を作成する方法。

方法 : 16 進文字列を数値に変換する

16 進の文字列を整数に変換する方法。

方法 : Visual Basic でバイトの配列を文字列に変換する

このトピックでは、バイト配列内のバイトを文字列に変換する方法を解説します。

使用例

この例では、System.Text.Encoding.Unicode エンコーディング クラスの GetString メソッドを使用して、バイト配列内のすべてのバイトを文字列に変換します。

VB

```
Private Function UnicodeBytesToString( _
    ByVal bytes() As Byte) _
    As String

    Return System.Text.Encoding.Unicode.GetString(bytes)
End Function
```

バイト配列を文字列に変換する場合、エンコーディング オプションを選択できます。

- System.Text.Encoding.ASCII : ASCII (7 ビット) 文字セットのエンコーディングを取得します。
- System.Text.Encoding.BigEndianUnicode : ビッグ エンディアンのバイト順を使用する UTF-16 形式のエンコーディングを取得します。
- System.Text.Encoding.Default : システムの現在の ANSI コード ページのエンコーディングを取得します。
- **System.Text.Encoding.Unicode** : リトル エンディアンのバイト順を使用する UTF-16 形式のエンコーディングを取得します。
- System.Text.Encoding.UTF32 : リトル エンディアンのバイト順を使用する UTF-32 形式のエンコーディングを取得します。
- System.Text.Encoding.UTF7 : UTF-7 形式のエンコーディングを取得します。
- System.Text.Encoding.UTF8 : UTF-8 形式のエンコーディングを取得します。

参照

処理手順

方法 : Visual Basic で文字列をバイトの配列に変換する

関連項目

System.Text.Encoding

GetString

方法 : Visual Basic で文字列をバイトの配列に変換する

このトピックでは、文字列をバイトの配列に変換する方法を示します。

使用例

この例では、System.Text.Encoding.Unicode エンコーディング クラスの GetBytes メソッドを使用して、文字列をバイトの配列に変換します。

VB

```
Private Function UnicodeStringToBytes( _
    ByVal str As String) _
    As Byte()

    Return System.Text.Encoding.Unicode.GetBytes(str)
End Function
```

文字列をバイト配列に変換する場合、エンコーディング オプションを選択できます。

- System.Text.Encoding.ASCII : ASCII (7 ビット) 文字セットのエンコーディングを取得します。
- System.Text.Encoding.BigEndianUnicode : ビッグ エンディアンのバイト順を使用する UTF-16 形式のエンコーディングを取得します。
- System.Text.Encoding.Default : システムの現在の ANSI コード ページのエンコーディングを取得します。
- **System.Text.Encoding.Unicode** : リトル エンディアンのバイト順を使用する UTF-16 形式のエンコーディングを取得します。
- System.Text.Encoding.UTF32 : リトル エンディアンのバイト順を使用する UTF-32 形式のエンコーディングを取得します。
- System.Text.Encoding.UTF7 : UTF-7 形式のエンコーディングを取得します。
- System.Text.Encoding.UTF8 : UTF-8 形式のエンコーディングを取得します。

参照

処理手順

方法 : Visual Basic でバイトの配列を文字列に変換する

関連項目

System.Text.Encoding

GetBytes

方法：Char 値の配列から文字列を作成する (Visual Basic)

個別の文字から文字列 "abcd" を作成する例を次に示します。

使用例

VB

```
Private Sub MakeStringFromCharacters()
    Dim characters() As Char = {"a"c, "b"c, "c"c, "d"c}
    Dim alphabet As New String(characters)
End Sub
```

コードのコンパイル方法

このメソッドには、特別な要件はありません。

単一文字 `c` が二重引用符に囲まれた単一文字の後に続く構文 `"a"c` は、文字リテラルを作成するために使用されます。

堅牢性の高いプログラム

文字列の中に `Chr(0)` に相当する null 文字があると、文字列を使用するときに予想外の結果になります。文字列に null 文字が含まれる場合、状況によっては null 文字の後の文字が表示されないことがあります。

参照

関連項目

文字型 (Char) (Visual Basic)

String

概念

Visual Basic におけるデータ型

方法：16 進文字列を数値に変換する

次の例は、ToInt32 メソッドを使用して、16 進文字列を整数に変換します。

16 進文字列を数値に変換するには

- **ToInt32** メソッドを使用して、ベース 16 で表現された数値を整数に変換します。

  ToInt32 メソッドの最初の引数が変換する文字列です。2 番目の引数には、数値がどのベースで表現されているかを指定します。16 進であればベース 16 です。

  VB

```
' Assign the value 49153 to i.
Dim i As Integer = Convert.ToInt32("c001", 16)
```

参照

関連項目

Hex 関数 (Visual Basic)
ToInt32

方法：Visual Basic で文字列を文字の配列に変換する

文字列を解析する際などに、文字列内の文字、およびそれらの文字の文字列内の位置に関するデータが取得できると便利な場合があります。次の例では、文字列の ToCharArray メソッドを呼び出して、文字列内の文字の配列を取得する方法を示します。

使用例

文字列を分割して **Char** 型の配列に格納する方法、および文字列を分割して Unicode テキスト文字の **String** 型の配列に格納する方法は次の例のようになります。このような違いがあるのは、Unicode テキスト文字は 2 つ以上の **Char** 型の文字 (サロゲート ペアまたは組み合わせ文字の並びなど) で構成される場合があるからです。詳細については、www.unicode.org の「TextElementEnumerator」および「The Unicode Standard」を参照してください。

VB

```
Dim testString1 As String = "ABC"
' Create an array containing "A", "B", and "C".
Dim charArray() As Char = testString1.ToCharArray
```

文字列を Unicode テキスト文字に分割することは非常に困難ですが、これは文字列のビジュアル表現に関する情報が必要な場合には必要です。次の例では、SubstringByTextElements メソッドを使用して、文字列を構成する Unicode テキスト文字に関する情報を取得します。

VB

```
' This string is made up of a surrogate pair (high surrogate
' U+D800 and low surrogate U+DC00) and a combining character
' sequence (the letter "a" with the combining grave accent).
Dim testString2 As String = ChrW(&HD800) & ChrW(&HDC00) & "a" & ChrW(&H300)

' Create and initialize a StringInfo object for the string.
Dim si As New System.Globalization.StringInfo(testString2)

' Create and populate the array.
Dim unicodeTestArray(si.LengthInTextElements) As String
For i As Integer = 0 To si.LengthInTextElements - 1
    unicodeTestArray(i) = si.SubstringByTextElements(i, 1)
Next
```

参照

処理手順

方法 : Visual Basic で文字列の文字にアクセスする

関連項目

Chars

System.Globalization.StringInfo

その他の技術情報

Visual Basic で、文字列型とその他のデータ型との変換を行う

Visual Basic における文字列

方法 : Visual Basic で文字列の文字にアクセスする

Chars プロパティを使用して、文字列の指定された位置にある文字にアクセスする方法は次のようになります。

使用例

文字列内の文字に関するデータを取得したり、文字列内の文字の位置を調べたりできます。文字列は、文字 (**Char** 型のインスタンス) の配列と見なすことができるため、**Chars** プロパティを使って文字のインデックスを参照すると、特定の文字を取得できます。

VB

```
Dim myString As String = "ABCDE"
Dim myChar As Char
' Assign "D" to myChar.
myChar = myString.Chars(3)
```

Chars プロパティの *index* パラメータは、0 から始まります。

堅牢性の高いプログラム

Chars プロパティは、指定された位置にある文字を返します。ただし、Unicode 文字は複数の文字で表現される場合もあります。Unicode 文字の詳細については、「方法 : Visual Basic で文字列を文字の配列に変換する」を参照してください。

Chars プロパティは、*index* パラメータが文字列の長さ以上である場合、またはゼロよりも小さい場合に IndexOutOfRangeException 例外をスローします。

参照

処理手順

方法 : Visual Basic で文字列を文字の配列に変換する

関連項目

Chars

その他の技術情報

Visual Basic で、文字列型とその他のデータ型との変換を行う

Visual Basic における文字列

Visual Basic の文字列に関するチュートリアル

ここでは、文字列の操作に関するチュートリアルを紹介します。

このセクションの内容

チュートリアル : Visual Basic での文字列の暗号化と複号化

DES (Data Encryption Standard) アルゴリズムを使用した、文字列の暗号化と復号化の方法について説明します。

参照

その他の技術情報

Visual Basic における文字列

チュートリアル：Visual Basic での文字列の暗号化と複号化

このチュートリアルでは DESCryptoServiceProvider クラスを使用して、TDES (Triple Data Encryption Standard) アルゴリズム (TripleDES) の暗号化サービス プロバイダ (CSP: Cryptographic Service Provider) バージョンを用いた文字列の暗号化と復号化を行う方法について説明します。最初に、3DES アルゴリズムをカプセル化し、暗号化されたデータを ベース 64 でエンコードされた文字列として格納する簡単なラッパー クラスを作成します。次に、そのラッパーを使用して、プライベートなユーザー データをパブリックにアクセス可能なテキスト ファイルに安全に格納します。

暗号化を使用すると、ユーザーのシークレット (たとえばパスワードなど) を保護したり、未承認のユーザーによる資格情報の解読を不可能にしたりできます。これにより、承認されたユーザーの ID が盗まれないように保護できるため、ユーザーの資産が保護されます。また、否認が不可能になります。暗号化を行うと、ユーザー データが未承認のユーザーからアクセスされないように保護することも可能になります。

詳細については、「暗号化の概要」を参照してください。

セキュリティに関するメモ：

Rijndael (最近では AES (Advanced Encryption Standard) とも呼ばれる) アルゴリズムや 3DES (Triple Data Encryption Standard) アルゴリズムは、暗号化のための計算量が非常に多いことから、DES よりもかなり高いセキュリティを実現します。詳細については、「DES」および「Rijndael」を参照してください。

暗号化のラッパーを作成するには

1. ファイルの先頭に、暗号化の名前空間のインポートを追加します。

   VB

   ```
   Imports System.Security.Cryptography
   ```

2. 暗号化メソッドと復号化メソッドをカプセル化するクラスを作成します。

   VB

   ```
   Public NotInheritable Class Simple3Des
   End Class
   ```

3. 3DES 暗号化サービス プロバイダを格納するためのプライベート フィールドを追加します。

   VB

   ```
   Private TripleDes As New TripleDESCryptoServiceProvider
   ```

4. 指定されたキーのハッシュから指定された長さのバイト配列を作成するプライベート メソッドを追加します。

   VB

   ```
   Private Function TruncateHash( _
       ByVal key As String, _
       ByVal length As Integer) _
       As Byte()

       Dim sha1 As New SHA1CryptoServiceProvider

       ' Hash the key.
       Dim keyBytes() As Byte = _
           System.Text.Encoding.Unicode.GetBytes(key)
       Dim hash() As Byte = sha1.ComputeHash(keyBytes)

       ' Truncate or pad the hash.
   ```

```
        ReDim Preserve hash(length - 1)
        Return hash
    End Function
```

5. 3DES 暗号化サービス プロバイダを初期化するためのコンストラクタを追加します。

   key パラメータは `EncryptData` メソッドと `DecryptData` メソッドを制御します。

   VB

```
    Sub New(ByVal key As String)
        ' Initialize the crypto provider.
        TripleDes.Key = TruncateHash(key, TripleDes.KeySize \ 8)
        TripleDes.IV = TruncateHash("", TripleDes.BlockSize \ 8)
    End Sub
```

6. 文字列を暗号化するパブリック メソッドを追加します。

   VB

```
    Public Function EncryptData( _
        ByVal plaintext As String) _
        As String

        ' Convert the plaintext string to a byte array.
        Dim plaintextBytes() As Byte = _
            System.Text.Encoding.Unicode.GetBytes(plaintext)

        ' Create the stream.
        Dim ms As New System.IO.MemoryStream
        ' Create the encoder to write to the stream.
        Dim encStream As New CryptoStream(ms, _
            TripleDes.CreateEncryptor(), _
            System.Security.Cryptography.CryptoStreamMode.Write)

        ' Use the crypto stream to write the byte array to the stream.
        encStream.Write(plaintextBytes, 0, plaintextBytes.Length)
        encStream.FlushFinalBlock()

        ' Convert the encrypted stream to a printable string.
        Return Convert.ToBase64String(ms.ToArray)
    End Function
```

7. 文字列を復号化するパブリック メソッドを追加します。

   VB

```
    Public Function DecryptData( _
        ByVal encryptedtext As String) _
        As String

        ' Convert the encrypted text string to a byte array.
        Dim encryptedBytes() As Byte = Convert.FromBase64String(encryptedtext)

        ' Create the stream.
        Dim ms As New System.IO.MemoryStream
        ' Create the decoder to write to the stream.
        Dim decStream As New CryptoStream(ms, _
```

```
            TripleDes.CreateDecryptor(), _
            System.Security.Cryptography.CryptoStreamMode.Write)

        ' Use the crypto stream to write the byte array to the stream.
        decStream.Write(encryptedBytes, 0, encryptedBytes.Length)
        decStream.FlushFinalBlock()

        ' Convert the plaintext stream to a string.
        Return System.Text.Encoding.Unicode.GetString(ms.ToArray)
    End Function
```

これで、ラッパークラスを使用してユーザーの資産を保護できるようになりました。ラッパー クラスを使用して、プライベートなユーザー データをパブリックにアクセス可能なテキスト ファイルに安全に格納する例を次に示します。

暗号化のラッパーをテストするには

1. 別のクラスで、ラッパーの `EncryptData` メソッドを使用して文字列を暗号化し、それをユーザーのマイ ドキュメント フォルダに書き込むメソッドを追加します。

   VB

   ```
   Sub TestEncoding()
       Dim plainText As String = InputBox("Enter the plain text:")
       Dim password As String = InputBox("Enter the password:")

       Dim wrapper As New Simple3Des(password)
       Dim cipherText As String = wrapper.EncryptData(plainText)

       MsgBox("The cipher text is: " & cipherText)
       My.Computer.FileSystem.WriteAllText( _
           My.Computer.FileSystem.SpecialDirectories.MyDocuments & _
           "\cipherText.txt", cipherText, False)
   End Sub
   ```

2. 暗号化された文字列を、ユーザーのマイ ドキュメント フォルダから読み取り、その文字列をラッパーの `DecryptData` メソッドを使って復号化するメソッドを追加します。

   VB

   ```
   Sub TestDecoding()
       Dim cipherText As String = My.Computer.FileSystem.ReadAllText( _
           My.Computer.FileSystem.SpecialDirectories.MyDocuments & _
               "\cipherText.txt")
       Dim password As String = InputBox("Enter the password:")
       Dim wrapper As New Simple3Des(password)

       ' DecryptData throws if the wrong password is used.
       Try
           Dim plainText As String = wrapper.DecryptData(cipherText)
           MsgBox("The plain text is: " & plainText)
       Catch ex As System.Security.Cryptography.CryptographicException
           MsgBox("The data could not be decrypted with the password.")
       End Try
   End Sub
   ```

3. `TestEncoding` メソッドと `TestDecoding` メソッドを呼び出すためのユーザー インターフェイス コードを追加します。

4. アプリケーションを実行します。

   アプリケーションをテストするとき、間違ったパスワードを入力するとデータが復号化されないので注意してください。

参照

関連項目

System.Security.Cryptography
DESCryptoServiceProvider
DES
TripleDES
Rijndael

概念

暗号化の概要

その他の技術情報

Visual Basic の文字列に関するチュートリアル

Visual Basic における文字列の検証

ここでは、Visual Basic で文字列を検証する方法について説明します。

このセクションの内容

Visual Basic における検証関数の構築

検証関数のプロパティおよび検証関数の作成方法と使い方について説明します。

方法 : Visual Basic でファイル名とパスを検証する

文字列がファイル名とパスのどちらを表すのかを確認する方法を説明します。

方法 : 日付または時刻を表す文字列を検証する (Visual Basic)

文字列が有効な日付を表すかどうかを確認する方法を説明します。

方法 : 電子メール アドレスを表す文字列を検証する (Visual Basic)

文字列が有効な電子メール アドレスを表すかどうかを確認する方法を説明します。

方法 : Visual Basic で Web ブラウザに表示するテキストを検証する

テキストがブラウザに正しく表示されることを確認する方法を説明します。

Visual Basic の MaskedTextBox コントロールによる正規表現を使用する

簡単な正規表現を変換して MaskedTextBox コントロールで使用できるようにする方法を説明します。

正規表現と Like 演算子

Like 演算子と正規表現を比較し、違いを説明します。

チュートリアル : パスワードの複雑さの検証 (Visual Basic)

文字列が強力なパスワードの特徴を備えているかどうかを確認する方法を説明します。

参照

その他の技術情報

Visual Basic における文字列
Visual Basic における文字列の解析
Visual Basic における文字列の検索と置換
MaskedTextBox コントロール (Windows フォーム)

Visual Basic における検証関数の構築

検証関数は、文字列が一定の要件を満たしているかどうかを調べます。ここでは検証関数の特徴を紹介し、検証関数を作成して使用する方法について説明します。

検証関数の種類

検証の対象となる文字列は、以下の 3 つのカテゴリに分類されます。

1. おそらく有効。
2. おそらく無効。
3. 有効か無効かがまったくわからない。

1 つ目と 2 つ目のカテゴリでは、検証関数が文字列を使ってどのような処理を行うかは明確ですが、3 つ目のカテゴリは少し面倒です。

3 つ目のカテゴリで検証関数が文字列をどのように扱うかによって、検証関数が次の 2 つのカテゴリに分類されます。

- **慎重な検証関数**これらの関数は、有効であることが証明できる文字列だけを検証します。

  アプリケーションが文字列をセキュリティの目的に使用する場合は、慎重な検証関数を使う必要があります。たとえば、不適切な入力をすべて予想することは困難なので、許可しない要素だけを除去するようなフィルタは作成しないでください。代わりに、フィルタを作成する場合は、受け入れ可能な入力のリストを定義してください。

- **寛大な検証関数**これらの関数は「おそらく無効」な文字列を除くすべての文字列を検証します。

  ユーザーのプロファイルの格納など、セキュリティに関連しないすべての状況において、寛大な検証関数を使用できます。この検証関数は、控えめな検証関数よりも柔軟でユーザー フレンドリです。

検証関数での正規表現

.NET Framework の Regex 正規表現クラスを使用すると、文字列が特定のパターンに一致しているか、または特定のパターンを含んでいるかを判断できます。

文字列の検証では、正規表現は ^ の文字で始める必要があります。これによって正規表現エンジンは、指定されたパターンとの照合を文字列の先頭から開始します。

参照

処理手順

方法 : 文字列が有効な電子メール形式であるかどうかを検証する

その他の技術情報

Visual Basic における文字列の検証

.NET Framework の正規表現

方法：Visual Basic でファイル名とパスを検証する

この例では、文字列がファイル名またはパスを表しているかどうかを示す **Boolean** 値を返します。この検証処理では、ファイル システムで許可されていない文字が名前に含まれているかどうかを調べます。

使用例

VB

```
Function IsValidFileNameOrPath(ByVal name As String) As Boolean
    ' Determines if the name is Nothing.
    If name Is Nothing Then
        Return False
    End If

    ' Determines if there are bad characters in the name.
    For Each badChar As Char In System.IO.Path.GetInvalidPathChars
        If InStr(name, badChar) > 0 Then
            Return False
        End If
    Next

    ' The name passes basic validation.
    Return True
End Function
```

この例では、名前中のコロンの位置が正しくないとか、ディレクトリ名がない、または名前の長さがシステム定義の最大長を超えている、というチェックは行いません。また、アプリケーションが指定の名前のファイル システム リソースにアクセスするアクセス許可を持っているかどうかのチェックも行いません。

参照

関連項目

GetInvalidPathChars

その他の技術情報

Visual Basic における文字列の検証

方法：日付または時刻を表す文字列を検証する (Visual Basic)

次のコード例は、文字列が有効な日付または時刻を表しているかどうかを示す **Boolean** 値を設定します。

使用例

VB

```
Dim isValidDate As Boolean = IsDate("01/01/03")
Dim isValidTime As Boolean = IsDate("9:30 PM")
```

コードのコンパイル方法

`("01/01/03")` および `"9:30 PM"` は、実際に検証する日付と時刻に置き換えてください。この文字列は、別のハードコーディングされた文字列、**String** 変数、または文字列を返す **InputBox** などのメソッドで置き換えることができます。

堅牢性の高いプログラム

このメソッドを使うと、**String** を **DateTime** 変数に変換する前に文字列を検証できます。日付または時刻を最初にチェックすることにより、実行時に例外が生成されることを回避できます。

参照

関連項目

IsDate 関数 (Visual Basic)

InputBox 関数 (Visual Basic)

その他の技術情報

Visual Basic における文字列の検証

方法：電子メール アドレスを表す文字列を検証する (Visual Basic)

次のコード例では、文字列が有効な電子メール アドレスを表しているかどうかを示すブール型変数を設定します。

使用例

VB

```
Function ValidateEmail(ByVal email As String) As Boolean
    Dim emailRegex As _
        New System.Text.RegularExpressions.Regex( _
        "^(?<user>[^@]+)@(?<host>.+)$")
    Dim emailMatch As _
        System.Text.RegularExpressions.Match = emailRegex.Match(email)
    Return emailMatch.Success
End Function
```

コードのコンパイル方法

このメソッドを呼び出すときには、電子メール アドレスを含んでいる文字列を渡します。

堅牢性の高いプログラム

このメソッドは、与えられた電子メール アドレスが "someone@microsoft.com" のような形式になっているかどうかを検証します。

このコードを使用すると、文字列を電子メール アドレスとして使用する前に、その文字列を検証できます。これにより、実行時のその他のエラーを防止できます。

参照

関連項目

Regex

方法：Visual Basic で Web ブラウザに表示するテキストを検証する

この例では HtmlEncode メソッドを使用して、ブラウザがテキストを正しく表示していることを確認する方法について示します。文字列内の <、>、& などの文字は、<、>、& のようにエスケープしなければ正しく表示されません。

使用例

次のコード例は、ブラウザに表示する文字列をエンコードします。This is a <Test String> というテキストが格納された文字列 TestString をエンコードし、それを文字列 EncodedString に This is a <Test String> としてコピーします。

```
Dim TestString As String = "This is a <Test String>"
Dim EncodedString As String = Server.HtmlEncode(TestString)
```

この例では System.Web.dll を参照する必要があります。

参照

関連項目

HtmlEncode

その他の技術情報

Visual Basic における文字列の検証

Visual Basic の MaskedTextBox コントロールによる正規表現を使用する

単純な正規表現を変換して MaskedTextBox コントロールで処理する方法は、次のようになります。

マスク言語について

標準の **MaskedTextBox** のマスク言語は、Visual Basic 6.0 の **Masked Edit** コントロールで使用されているものをベースとしているため、そのプラットフォームから移行しているユーザーには見覚えがあるものです。

MaskedTextBox コントロールの Mask プロパティには、使用する入力マスクを指定します。マスクは次の表にある 1 つ以上のマスク要素で構成される文字列であることが必要です。

マスク要素	説明	正規表現要素
0	0 ～ 9 の単一の数字。入力されていることが必要です。	\d
9	数字または空白。入力されていなくてもかまいません。	[\d]?
#	数字または空白。入力されていなくてもかまいません。この位置がマスク内で空白になっている場合は、空白として表示されます。正符号 (+) と負符号 (-) が許可されます。	[\d+-]?
L	ASCII 文字。入力されていることが必要です。	[a-zA-Z]
?	ASCII 文字。入力されていなくてもかまいません。	[a-zA-Z]?
&	文字入力されていることが必要です。	[\p{Ll}\p{Lu}\p{Lt}\p{Lm}\p{Lo}]
C	文字入力されていなくてもかまいません。	[\p{Ll}\p{Lu}\p{Lt}\p{Lm}\p{Lo}]?
A	英数字。入力されていなくてもかまいません。	\W
.	カルチャに応じた小数点記号。	使用できません。
,	カルチャに応じた桁区切り記号。	使用できません。
:	カルチャに応じた時刻の区切り記号。	使用できません。
/	カルチャに応じた日付の区切り記号。	使用できません。
$	カルチャに応じた通貨記号。	使用できません。
<	次に続くすべての文字を小文字に変換します。	使用できません。
>	次に続くすべての文字を大文字に変換します。	使用できません。
|	直前に行ったシフト アップまたはシフト ダウンを元に戻します。	使用できません。
\	マスク文字をエスケープし、リテラルに変えます。"\\" は円記号 (\) のエスケープ シーケンスです。	\
上記以外のすべての文字。	リテラル。マスク以外の要素は、**MaskedTextBox** にそのまま表示されます。	上記以外のすべての文字。

小数点 (.)、桁区切り (,)、時刻 (:)、日付 (/)、および通貨 ($) の記号は、既定ではアプリケーションのカルチャで定義された記号で表示されます。FormatProvider プロパティを使用すると、別のカルチャの記号で表示させることができます。

正規表現とマスク

正規表現とマスクのどちらを使ってもユーザー入力を検証できますが、この 2 つは完全に同等ではありません。正規表現を使うとマスクよりも複雑なパターンを表現できますが、マスクを使うと同じ情報をより簡潔に、またカルチャに応じた形式で表現できます。

正規表現と、それに相当するマスクとの比較を次の表に示します。

正規表現	マスク	メモ
`\d{2}/\d{2}/\d{4}`	`00/00/0000`	マスク内の / の文字は、論理的な日付の区切り記号であり、ユーザーにはアプリケーションの現在のカルチャに応じた日付の区切り記号が表示されます。
`\d{2}-[A-Z][a-z]{2}-\d{4}`	`00->L<LL-0000`	米国形式の日付 (日、月の省略形、年) が表示されます。月の省略形は先頭が大文字、後続の 2 文字が小文字の 3 文字で表示されます。
`(\(\d{3}\)-)?\d{3}-d{4}`	`(999)-000-0000`	米国形式の電話番号です。エリア コードは省略可能です。省略可能な文字を入力しない場合は、空白を入力するか、マスク内で最初に 0 で表現されている位置に直接マウス ポインタを置きます。
`$\d{6}.00`	`$999,999.00`	0 ～ 999999 の範囲の通貨の値です。通貨記号、区切り記号、小数点記号は実行時にそれぞれのカルチャ固有の記号で置き換えられます。

参照

関連項目

Mask

MaskedTextBox

その他の技術情報

Visual Basic における文字列の検証

MaskedTextBox コントロール (Windows フォーム)

正規表現と Like 演算子

このトピックでは Visual Basic の **Like** 演算子と .NET Framework の正規表現とを比較します。

構文の違い

Like 演算子のパターン指定言語の構文と、正規表現の構文との比較を次の表にまとめます。

Like 演算子の構文	**正規表現の構文**
Like 演算子の動作は、Option Compare ステートメントによって決まります。各ソース ファイルの既定の文字列比較メソッドは **Option Compare Binary** です。	正規表現は **Option Compare** に関係なく同様に動作します。
charlist 内の任意の単一文字と照合する場合は、**[***charlist***]** を使います。	*charlist* 内の任意の単一文字と照合する場合は、**[***charlist***]** を使います。
charlist にない任意の単一文字と照合する場合は、**[!***charlist***]** を使います。	*charlist* にない任意の単一文字と照合する場合は、**[^***charlist***]** を使います。
任意の単一数字 (0–9) と照合する場合は、**#** を使用します。	任意の単一数字 (0–9) と照合する場合は、10 進数字の文字クラス **\d** を使用します。
任意の単一文字と照合する場合は、**?** を使います。	任意の単一文字と照合する場合は、**[***charlist***]** の*charlist* に、同時に指定できない文字クラスを指定します。たとえば、**[\s\S]** などです。
0 個以上の文字と照合する場合は、***** を使用します。	0 個以上の文字と照合する場合は、**[***charlist***]*** の *charlist* に、同時に指定できない文字クラスを指定します。たとえば、**[\s\S]*** などです。
特殊文字 *char* と照合する場合は、**[***char***]** のように文字を角かっこで囲みます。	特殊文字 *char* と照合する場合は、*****char のように文字の前に円記号 (\) を付けます。
ある範囲内の任意の文字と照合する場合は、*charlist* 内でハイフン (–) を使用して範囲の下限と上限を区切ります。	ある範囲内の任意の文字と照合する場合は、*charlist* 内でハイフン (–) を使用して範囲の下限と上限を区切ります。

参照

関連項目

Like 演算子

その他の技術情報

Visual Basic における文字列の検証

.NET Framework の正規表現

チュートリアル：パスワードの複雑さの検証 (Visual Basic)

ここでは、強力なパスワードの特性をチェックし、パスワードの複雑さのチェックに関する情報を使って文字列パラメータを更新するための方法について説明します。

パスワードは、セキュリティ保護されたシステムでユーザーを承認するために使用します。しかし、パスワードは許可されないユーザーから簡単に推測されないために、複雑であることが必要です。攻撃者は、ある辞書内 (または、さまざな言語の複数の辞書内) のすべての語を反復処理して、その中にユーザーのパスワードと一致するものがあるかどうかを調べる辞書攻撃プログラムを使用する場合があります。"Yankees" や "Mustang" などの脆弱なパスワードでは、簡単に推測される可能性があります。"?You'L1N3vaFiNdMeyeP@sSWerd!" などのより強力なパスワードであれば、推測される可能性はかなり低くなります。システムをパスワードで保護する場合は、ユーザーが強力なパスワードを選択していることを確認する必要があります。

強力なパスワードとは、複雑 (大文字と小文字、数字、および特殊文字を組み合わせていること) で、単語ではないパスワードです。次の例は、複雑さを検査する方法を示しています。

サンプル
VB

```
''' <summary>Determines if a password is sufficiently complex.</summary>
''' <param name="pwd">Password to validate</param>
''' <param name="minLength">Minimum number of password characters.</param>
''' <param name="numUpper">Minimum number of uppercase characters.</param>
''' <param name="numLower">Minimum number of lowercase characters.</param>
''' <param name="numNumbers">Minimum number of numeric characters.</param>
''' <param name="numSpecial">Minimum number of special characters.</param>
''' <returns>True if the password is sufficiently complex.</returns>
Function ValidatePassword(ByVal pwd As String, _
    Optional ByVal minLength As Integer = 8, _
    Optional ByVal numUpper As Integer = 2, _
    Optional ByVal numLower As Integer = 2, _
    Optional ByVal numNumbers As Integer = 2, _
    Optional ByVal numSpecial As Integer = 2) _
    As Boolean

    ' Replace [A-Z] with \p{Lu}, to allow for Unicode uppercase letters.
    Dim upper As New System.Text.RegularExpressions.Regex("[A-Z]")
    Dim lower As New System.Text.RegularExpressions.Regex("[a-z]")
    Dim number As New System.Text.RegularExpressions.Regex("[0-9]")
    ' Special is "none of the above".
    Dim special As New System.Text.RegularExpressions.Regex("[^a-zA-Z0-9]")

    ' Check the length.
    If Len(pwd) < minLength Then Return False
    ' Check for minimum number of occurrences.
    If upper.Matches(pwd).Count < numUpper Then Return False
    If lower.Matches(pwd).Count < numLower Then Return False
    If number.Matches(pwd).Count < numNumbers Then Return False
    If special.Matches(pwd).Count < numSpecial Then Return False

    ' Passed all checks.
    Return True
End Function

Sub TestValidatePassword()
    Dim password As String = "Password"
    ' Demonstrate that "Password" is not complex.
    MsgBox(password & " is complex: " & ValidatePassword(password))

    password = "Z9f%a>2kQ"
    ' Demonstrate that "Z9f%a>2kQ" is not complex.
    MsgBox(password & " is complex: " & ValidatePassword(password))
End Sub
```

コードのコンパイル

このメソッドを呼び出すには、パスワードを格納する文字列を渡します。

この例で必要な要素は次のとおりです。

- System.Text.RegularExpressions 名前空間のメンバへのアクセス許可。メンバ名を完全に修飾しないでコードに記述する場合は、**Imports** ステートメントを追加してください。詳細については、「Imports ステートメント」を参照してください。

セキュリティ

ネットワーク経由でパスワードを渡す場合は、安全な方法を使用してデータを転送する必要があります。詳細については、「ASP.NET Web アプリケーションのセキュリティ」を参照してください。

次に示す方法で複雑さのチェックを追加すると、`ValidatePassword` 関数の精度を高めることができます。

- パスワードとその部分文字列を、ユーザー名、ユーザー ID、およびアプリケーション定義の辞書と比較します。また、比較を実行するときには、外見が似ている文字を同等として扱ってください。たとえば、"l" と "e" の文字は、"1" と "3" の数字と同等として扱います。
- 大文字を 1 つしか使っていない場合は、それがパスワードの先頭の文字でないことを確認します。
- パスワードの最後の 2 文字がアルファベット文字であることを確認します。
- すべての記号がキーワードの一番上の行から入力されているパスワードを禁止します。

参照

関連項目

Regex

その他の技術情報

ASP.NET Web アプリケーションのセキュリティ

Visual Basic の文字列に関するチュートリアル

Visual Basic における文字列の検索と置換

ここでは、Visual Basic で文字列を検索および置換する方法について説明します。

このセクションの内容

方法 : 文字列の一部を削除する (Visual Basic)

ある文字列の中に出現する特定の文字列をすべて削除します。

方法 : 文字列の配列内で文字列を検索する (Visual Basic)

配列内で指定の部分文字列を含んでいる最初の文字列のインデックスと、その文字列内での部分文字列のインデックスを取得します。

方法 : 文字列内を検索する (Visual Basic)

ある文字列の中に出現する最初の部分文字列のインデックスを取得します。

参照

概念

Visual Basic における文字列操作メソッドの種類

その他の技術情報

Visual Basic における文字列

方法：文字列の配列内で文字列を検索する (Visual Basic)

次の例は、文字列の配列内で各文字列をループ処理し、指定された部分文字列がどの文字列に含まれるかを調べます。部分文字列が見つかるたびに、文字列内の部分文字列のインデックスが表示されます。

使用例

次の例では、**String** オブジェクトの Contains メソッドと IndexOf メソッドが使用されています。

Contains メソッドは、指定された部分文字列が文字列に含まれるかどうかを示します。

IndexOf メソッドは、最初に見つかった部分文字列の 1 番目の文字の場所を報告します。インデックスはゼロベースです。つまり、文字列の最初の文字のインデックスは 0 になります。部分文字列が見つからなければ、**IndexOf** は -1 を返します。

VB

```
Dim StrArray() As String = {"ABCDEFG", "HIJKLMNOP"}
Dim FindThisString As String = "JKL"
For Each Str As String In StrArray
    If Str.Contains(FindThisString) Then
        MsgBox("Found " & FindThisString & " at index " & _
          Str.IndexOf(FindThisString))
    End If
Next
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [データ型 - Visual Basic によって定義済み] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例には、次の項目が必要です。

- System 名前空間を指定する **Imports** ステートメント。詳細については、「Imports ステートメント」を参照してください。

堅牢性の高いプログラム

IndexOf メソッドでは大文字と小文字が区別され、現在のカルチャが使用されます。

最適なエラー制御としては、文字列の検索を Try...Catch...Finally ステートメント (Visual Basic) 構造の **Try** ブロックで囲む方法があります。

参照

処理手順

方法 : 文字列内を検索する (Visual Basic)

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

IndexOf

その他の技術情報

Visual Basic の文字列の概要

方法：文字列内を検索する (Visual Basic)

次のコード例では、String オブジェクト上で IndexOf メソッドを呼び出して、最初に見つかった部分文字列のインデックスを報告します。

使用例

VB

```
Dim SearchWithinThis As String = "ABCDEFGHIJKLMNOP"
Dim SearchForThis As String = "DEF"
Dim FirstCharacter As Integer = SearchWithinThis.IndexOf(SearchForThis)
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [データ型 - Visual Basic によって定義済み] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System 名前空間を指定する **Imports** ステートメントです。詳細については、「Imports ステートメント」を参照してください。

堅牢性の高いプログラム

IndexOf メソッドは、最初に見つかった部分文字列の 1 番目の文字の場所を報告します。インデックスは、0 から始まります。つまり、文字列の最初の文字はインデックス 0 になります。

IndexOf が部分文字列を検出できない場合は、-1 が返されます。

IndexOf メソッドでは大文字と小文字が区別され、現在のカルチャが使用されます。

最適なエラー制御としては、文字列の検索を Try...Catch...Finally ステートメント (Visual Basic) 構造の **Try** ブロックで囲む方法があります。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)
IndexOf

その他の技術情報

Visual Basic の文字列の概要
Visual Basic における文字列の検索と置換

方法：文字列の一部を削除する (Visual Basic)

次のコード例は、ある文字列を別の文字列からすべて削除します。

使用例

この例では **Replace** 関数を使用して、出現する部分文字列を、それぞれ空の文字列に置き換えています。その結果、部分文字列が削除された新しい文字列に変更されます。

次の例は、**Replace** 関数の戻り値を、文字列変数 `withoutParts` に代入しています。`withParts` に格納された元の文字列は変更されません。

VB

```
Dim withParts As String = "Books and Chapters and Pages"
Dim withoutParts As String = Replace(withParts, "and ", "")
```

コードのコンパイル方法

`"Books and Chapters and Pages"` を好きな文字列で置き換えてください。

参照

関連項目

Replace 関数 (Visual Basic)

文字列型 (String) (Visual Basic)

Visual Basic における文字列の解析

ここでは、Visual Basic で文字列を解析する方法について説明します。

このセクションの内容

方法 : Visual Basic で URI を解析する

System.Uri クラスを使って URI (Uniform Resource Identifier) を解析する方法を示します。

方法 : Visual Basic で電子メール アドレスを解析する

電子メール アドレスを解析するための簡単な正規表現を示します。

方法 : Visual Basic の文字列に含まれている URI を識別する

簡単な正規表現を使って文字列内の URI を識別する方法を示します。

方法 : Visual Basic で HTML 文字列内のテキストを特定する

簡単な正規表現を使って HTML ドキュメントからタグを削除する方法を示します。

方法 : Visual Basic の HTML 文字列に含まれているハイパーリンクを識別する

HTML ドキュメント内のハイパーリンクを識別するための簡単な正規表現を示します。

参照

その他の技術情報

Visual Basic における文字列
Visual Basic の文字列の概要

方法：Visual Basic で URI を解析する

System.Uri クラスを使用して URI を解析する方法は、次の例のようになります。

使用例

この例では、**Uri** オブジェクトの複数のメソッドを使用して、URI に関する情報を取得しています。

```
Dim uriString As String = "http://www.contoso.com/index.htm?date=today"
Dim uriObject As New Uri(uriString)
' Display "Scheme: http"
MsgBox("Scheme: " & uriObject.Scheme)
' Display "Host: www.contoso.com"
MsgBox("Host: " & uriObject.Host)
' Display "Local path: /index.htm"
MsgBox("Local path: " & uriObject.LocalPath)
' Display "Query: ?date=today"
MsgBox("Query: " & uriObject.Query)
```

参照

関連項目

Uri

その他の技術情報

Visual Basic における文字列の解析

方法：Visual Basic で電子メール アドレスを解析する

電子メール アドレスを解析するための簡単な正規表現を次の例に示します。

使用例

この例で使用されている正規表現、`(\S+)@([^\.\s]+)(?:\.([^\.\s]+))+` は次の意味を持ちます。

1. 1 つ以上の (キャプチャされた) 空白ではない文字の後に、以下の順番で文字が続く
2. "@" の文字
3. 1 つ以上の空白でもピリオドでもない (キャプチャされた) 文字
4. 以下の順で 1 つ以上の文字が続く
    a. "." の文字
    b. 1 つ以上の空白でもピリオドでもない (キャプチャされた) 文字。

正規表現の Match メソッドは、入力文字列のどの部分が正規表現と一致するかに関する情報が格納された Match オブジェクトを返します。

```
''' <summary>
''' Parses an e-mail address into its parts.
''' </summary>
''' <param name="emailString">E-mail address to parse.</param>
''' <remarks> For example, this method displays the following
''' text when called with "someone@mail.contoso.com":
''' User name: someone
''' Address part: mail
''' Address part: contoso
''' Address part: com
''' </remarks>
Sub ParseEmailAddress(ByVal emailString As String)
    Dim emailRegEx As New Regex("(\S+)@([^\.\s]+)(?:\.([^\.\s]+))+")
    Dim m As Match = emailRegEx.Match(emailString)
    If m.Success Then
        Dim output As String = ""
        output &= "User name: " & m.Groups(1).Value & vbCrLf
        For i As Integer = 2 To m.Groups.Count - 1
            Dim g As Group = m.Groups(i)
            For Each c As Capture In g.Captures
                output &= "Address part: " & c.Value & vbCrLf
            Next
        Next
        MsgBox(output)
    Else
        MsgBox("The e-mail address cannot be parsed.")
    End If
End Sub
```

この例では、**Imports** ステートメントを使用して System.Text.RegularExpressions 名前空間をインポートする必要があります。詳細については、「Imports ステートメント」を参照してください。

参照

処理手順

方法 : 文字列が有効な電子メール形式であるかどうかを検証する

その他の技術情報

Visual Basic における文字列の解析

方法：Visual Basic の文字列に含まれている URI を識別する

ここでは、簡単な正規表現を使って文字列内の URI (Uniform Resource Identifier) を識別する方法を示します。誤認識 (誤って URI と識別されるテキスト) の数を減らすために、URI が厳密な形式であると仮定します。このため、有効な URI が識別されない認識漏れが起こる可能性があります。

使用例

厳密な URI 形式は、`([^=":\s]+:)?//[^\s"]+` という正規表現に一致します。これは、次の条件を意味します。

1. 以下の文字が現れる可能性があります。
    a. =、"、:、または空白文字ではない 1 つ以上の文字
    b. それに続く : 文字
2. 文字列 `//` がこれに続きます。
3. 引用符または空白文字ではない 1 つ以上の文字があります。

Regex オブジェクトが、この正規表現によって初期化されます。

Regex オブジェクトの Matches メソッドは、入力文字列で正規表現に一致したすべての部分を含むMatchCollection オブジェクトを返します。

```
''' <summary>Identifies URIs in text.</summary>
''' <param name="text">Text to parse.</param>
''' <remarks>Displays each URI in the input text.</remarks>
Sub IdentifyURIs(ByVal text As String)
    Dim uriRegex As New Regex("([^="":\s]+:)?//[^\s""]+")
    Dim output As String = ""
    For Each m As Match In uriRegex.Matches(text)
        output &= m.Value & vbCrLf
    Next
    MsgBox(output)
End Sub
```

この例では、**Imports** ステートメントを使って System.Text.RegularExpressions 名前空間をインポートすることを前提としています。詳細については、「Imports ステートメント」を参照してください。

参照

処理手順

方法 : Visual Basic の HTML 文字列に含まれているハイパーリンクを識別する
方法 : 文字列から無効な文字を取り除く

その他の技術情報

Visual Basic における文字列の解析

方法：Visual Basic で HTML 文字列内のテキストを特定する

この例では、単純な正規表現を使用して HTML ドキュメントからタグを削除する方法を説明します。

使用例

HTML タグは、`\<[^\>]+\>` という正規表現に一致します。この正規表現の意味は次のとおりです。

1. 先頭の文字が "<" で、その後に
2. 文字 ">" 以外の 1 つまたは複数の文字が続き、
3. 最後の文字が ">" となる

この例では、System.Text.RegularExpressions.Regex.Replace(System.String,System.String,System.String) 共有メソッドを使用して、タグの正規表現に一致する部分をすべて空の文字列に置換します。

```
''' <summary>Removes the tags from an HTML document.</summary>
''' <param name="htmlText">HTML text to parse.</param>
''' <returns>The text of an HTML document without tags.</returns>
''' <remarks></remarks>
Function GetTextFromHtml(ByVal htmlText As String) As String
    Dim output As String = Regex.Replace(htmlText, "\<[^\>]+\>", "")
    Return output
End Function
```

この例を実行するには、**Imports** ステートメントを使用して System.Text.RegularExpressions 名前空間をインポートすることが必要です。詳細については、「Imports ステートメント」を参照してください。

参照

処理手順

方法 : Visual Basic の HTML 文字列に含まれているハイパーリンクを識別する

方法 : 文字列から無効な文字を取り除く

その他の技術情報

Visual Basic における文字列の解析

方法：Visual Basic の HTML 文字列に含まれているハイパーリンクを識別する

次の例は、HTML ドキュメント内のハイパーリンクを識別するための簡単な正規表現を示しています。

使用例

この例では、正規表現 `<A[^>]*?HREF\s*=\s*"([^"]+)"[^>]*?>([\s\S]*?)<\/A>` を使用しています。この正規表現の意味は、以下のとおりです。

1. 文字列 "<A" が先頭にある。
2. その後に、">" 以外の文字が 0 個以上連続する。
3. その後に、文字列 "HREF" がある。
4. その後に、空白文字が 0 個以上ある。
5. その後に、文字列 "=" がある。
6. その後に、空白文字が 0 個以上ある。
7. その後に、引用符がある。
8. その後に、(キャプチャされた) 引用符以外の文字が連続する。
9. その後に、引用符がある。
10. その後に、">" 以外の文字が 0 個以上連続する。
11. その後に、文字列 ">" がある。
12. その後に、(キャプチャされた) 文字が 0 個以上ある。
13. 最後に文字列 "</A>" がある。

Regex オブジェクトは、この正規表現によって初期化され、大文字と小文字が区別されるように設定されます。

Regex オブジェクトの Matches メソッドは、入力文字列で正規表現に一致したすべての部分を含む MatchCollection オブジェクトを返します。

```
    ''' <summary>Identifies hyperlinks in HTML text.</summary>
    ''' <param name="htmlText">HTML text to parse.</param>
    ''' <remarks>This method displays the label and destination for
    ''' each link in the input text.</remarks>
    Sub IdentifyLinks(ByVal htmlText As String)
        Dim hrefRegex As New Regex( _
            "<A[^>]*?HREF\s*=\s*""([^""]+)""[^>]*?>([\s\S]*?)<\/A>", _
            RegexOptions.IgnoreCase)
        Dim output As String = ""
        For Each m As Match In hrefRegex.Matches(htmlText)
            output &= "Link label: " & m.Groups(2).Value & vbCrLf
            output &= "Link destination: " & m.Groups(1).Value & vbCrLf
        Next
        MsgBox(output)
    End Sub
```

この例では、**Imports** ステートメントを使って System.Text.RegularExpressions 名前空間をインポートすることを前提としています。詳細については、「Imports ステートメント」を参照してください。

参照

概念

例 : HREFS のスキャン

その他の技術情報

Visual Basic における文字列のトラブルシューティング

このセクションでは Visual Basic での文字列操作について、トラブルシューティングに関するアドバイスを提供します。

このセクションの内容

Visual Basic における正規表現のトラブルシューティング

Visual Basic での文字列の使用における基本的な考え方について説明したトピックをいくつか紹介します。

関連項目

Visual Basic の Nothing と文字列

Visual Basic ランタイムと .NET Framework が、文字列に格納された **Nothing** を評価する方法の違いについて説明します。

Visual Basic においてカルチャが文字列に与える影響

Visual Basic がカルチャ情報を使用して、文字列の変換と比較を実行する方法について説明します。

Visual Basic における文字列アクセスのインデックス番号

Visual Basic と .NET Framework における、文字列内の文字へのアクセス方法の違いについて説明します。

参照

その他の技術情報

Visual Basic における文字列
Visual Basic の文字列に関するチュートリアル
Visual Basic の文字列の概要

Visual Basic における正規表現のトラブルシューティング

ここでは、正規表現を扱うときによく起こる問題について説明し、そのような問題を解決するためのヒントを示し、キャプチャされた文字列にアクセスする手順を説明します。

予期したパターンに一致しない

正規表現を使って実行できる一般的なタスクと、予期した結果が正規表現で得られない場合に役立つトラブルシューティングのヒントを以下に示します。

- **文字列の検証**。文字列を検証する正規表現は、^ 文字で始まる必要があります。この文字があると、正規表現のエンジンは、文字列の先頭から指定のパターンに一致する文字列の検索を開始します。詳細については、「Visual Basic における検証関数の構築」および「アトミック ゼロ幅アサーション」を参照してください。
- **量指定子を使った一致。**量指定子 (*、+、?、および {}) は、最長一致です。最長一致は、条件を満たし得る最も長い文字列に一致することを指します。ただし、パターンによっては、最短一致を使って、条件を満たし得る最も短い文字列を取得する必要があります。最短一致の正規表現量指定子は、*?、+?、??、および {}? です。詳細については、「量指定子」を参照してください。
- **入れ子の量指定子を使った一致。**入れ子の量指定子を使う場合には、すべての量指定子を最長一致または最短一致で統一してください。混在していると、どの文字列に一致するか予測が困難になります。

キャプチャされた文字列へのアクセス

予期した文字列が正規表現によって検出された場合、それらの文字列をキャプチャしてから、各文字列にアクセスする必要があります。正規表現に含まれるサブ式のグループに一致した文字列をキャプチャするには、グループ化構成体を使用できます。詳細については、「グループ化構成体」を参照してください。

入れ子になったグループを使ってキャプチャされた文字列にアクセスするには、次の手順に従います。

キャプチャされたテキストにアクセスするには

1. 正規表現に使用する Regex オブジェクトを作成します。
2. Match メソッドを呼び出して、Match オブジェクトを取得します。

   Match オブジェクトには、正規表現が文字列にどのように一致したかについての情報が格納されます。
3. **Match** オブジェクトの Groups コレクションに格納された Group オブジェクトを反復処理します。

   Group オブジェクトには、1 つのキャプチャ グループによるキャプチャの結果が格納されます。
4. 各 **Group** オブジェクトの Captures コレクションに格納された Capture オブジェクトを反復処理します。

   各 **Capture** オブジェクトには、キャプチャされた 1 つの部分式に関する情報が、一致した部分文字列と場所を含め、格納されます。

たとえば、次のコード例は、3 つのキャプチャ グループが含まれる正規表現でキャプチャされた文字列にアクセスする方法を示しています。

```
    ''' <summary>
    ''' Parses an e-mail address into its parts.
    ''' </summary>
    ''' <param name="emailString">E-mail address to parse.</param>
    ''' <remarks> For example, this method displays the following
    ''' text when called with "someone@mail.contoso.com":
    ''' User name: someone
    ''' Address part: mail
    ''' Address part: contoso
    ''' Address part: com
    ''' </remarks>
    Sub ParseEmailAddress(ByVal emailString As String)
        Dim emailRegEx As New Regex("(\S+)@([^\.\s]+)(?:\.([^\.\s]+))+")
        Dim m As Match = emailRegEx.Match(emailString)
        If m.Success Then
            Dim output As String = ""
            output &= "User name: " & m.Groups(1).Value & vbCrLf
            For i As Integer = 2 To m.Groups.Count - 1
                Dim g As Group = m.Groups(i)
                For Each c As Capture In g.Captures
```

```
                    output &= "Address part: " & c.Value & vbCrLf
                Next
            Next
            MsgBox(output)
        Else
            MsgBox("The e-mail address cannot be parsed.")
        End If
    End Sub
```

参照

処理手順

Visual Basic における文字列のトラブルシューティング

概念

Visual Basic における検証関数の構築

Visual Basic の定数と列挙体

定数は、変化しない値の代わりにわかりやすい名前を使用するための方法です。定数に格納された値は、その名が示すとおり、アプリケーションの実行中に変わることはありません。定数を使うと、数値の代わりにわかりやすい名前を使用できるので、コードが読みやすくなります。

複数の関連する定数を操作する場合や、複数の定数値に名前を関連付ける場合は、列挙型を使うと便利です。たとえば、一連の整数型の定数を曜日に関連付けて列挙型として宣言すると、整数値の代わりに曜日名を使うことができます。

このセクションの内容

Visual Basic の定数

このセクションのトピックでは、定数とその使用法について説明します。

列挙型の概要

このセクションのトピックでは、列挙体とその使用法について説明します。

組み込み定数と組み込み列挙型

Visual Basic 2005 が提供する定数と列挙体について説明します。

関連するセクション

Const ステートメント (Visual Basic)

定数の宣言に使用される **Const** ステートメントの構文と使用上の規則について説明します。

Enum ステートメント (Visual Basic)

列挙型の作成に使用される **Enum** ステートメントの構文と使用上の規則について説明します。

Option Explicit ステートメント (Visual Basic)

Option Explicit ステートメントの構文と使用上の規則について説明します。このステートメントはモジュール レベルで使用され、モジュール内のすべての変数を明示的に宣言することを要求します。

Option Strict ステートメント

Option Strict ステートメントの構文と使用上の規則について説明します。このステートメントは、暗黙のデータ型変換を拡大変換だけに制限します。

Visual Basic の定数

定数に格納された値は、その名が示すとおり、アプリケーションの実行中に変わることはありません。定数は、コード内で繰り返し使う値がある場合や、コードを読みやすくする場合に使用します。

このセクションの内容

定数の概要

定数を定義する方法と、どのような場合に定数を使用するべきかを説明します。

Visual Basic によって宣言された定数

Visual Basic に定義済みの定数を一覧で示します。

方法 : 定数を宣言する

Const ステートメントを使用して定数を宣言し、定数に値を設定する方法を説明します。定数を宣言する際には、値を予想しやすい名前を割り当てます。

ユーザー定義関数

独自の定数を作成する方法について説明します。スコープについて、および循環参照を避ける方法についても説明します。

定数とリテラルのデータ型

Option Explicit がオフの場合に、Visual Basic コンパイラが定数を初期化する方法について説明します。

方法 : 関連する定数値をまとめてグループ化する

関連する定数の値をグループ化する方法を示します。

参照

Const ステートメント (Visual Basic)

Const ステートメントとその使用方法について説明します。

Option Strict ステートメント

Option Strict ステートメントとその使用方法について説明します。

関連するセクション

列挙型の概要

列挙定数とその使用法について、列挙定数に関連する一般的なタスクを含めて説明します。

印刷と表示の定数

Visual Basic で定義済みの、印刷および表示に使用可能な定数を一覧で示します。

組み込み定数と組み込み列挙型

Visual Basic に用意されている定数と列挙定数を紹介します。

Visual Basic の定数と列挙体

最新バージョンの Visual Basic における、定数と列挙定数の変更点を説明します。

定数の概要

定数とは、値が変化しない数字または文字列の代わりに使用される名前です。定数に格納された値は、その名が示すとおり、アプリケーションの実行中に変わることはありません。定数を使うとコードが読みやすくなり、保守も簡単になります。定数は、コード内で、同じ値を繰り返し使用する場合や、覚えにくいか意味が明白でない特定の数値を参照する場合に使用します。

定数の作成および使用方法

Visual Basic には多数の定義済み定数があり、主として印刷用および表示用に使用されます。定数は、変数名を作成するときと同じガイドラインに従って、**Const** ステートメントで宣言します。**Option Strict** が **On** の場合、明示的に定数の型を宣言する必要があります。

定数のスコープ、つまり名前に修飾子を付けずに定数を参照できるコードの範囲は、同じ場所で変数を宣言した場合の変数のスコープと同じです。特定のプロシージャのスコープ内で存在する定数を作成するには、定数をそのプロシージャ内で宣言します。アプリケーション全体で使用できる定数の宣言は、クラスの宣言セクションで **Public** キーワードを使って行います。

メモ :
定数は変数と似ていますが、変数のように変更することや新しい値を代入することはできません。

コードで使用する定数には、使用するコントロールやコンポーネントのオブジェクト モデルにあらかじめ用意されている定数と、ユーザー定義の定数 (つまり、自分で作成する定数) とがあります。

コンパイル時定数および実行時定数

コンパイル時定数は、コードがコンパイルされる時点で計算されます。一方、実行時定数は、アプリケーションの実行時にならないと計算できません。コンパイル時定数の値はアプリケーションを何回実行しても常に同じですが、実行時定数の値はアプリケーション実行のたびに変わる可能性があります。コンパイル時定数は、配列の上下限、ケース式、列挙子の初期化子などに必要です。

参照

処理手順

方法 : 定数を宣言する

関連項目

Const ステートメント (Visual Basic)

Public (Visual Basic)

概念

定数とリテラルのデータ型

Visual Basic によって宣言された定数

その他の技術情報

定数と列挙型 (Visual Basic)

Visual Basic の定数

Visual Basic によって宣言された定数

Visual Basic には、条件付きコンパイルや、データの印刷および表示の際に開発者が使用できるように、いくつかの定義済みの定数が用意されています。

条件付きコンパイル定数

条件付きコンパイルで利用できる定義済みの定数の一覧を次の表に示します。

定数	説明
CONFIG	構成マネージャの [アクティブ ソリューション構成] ボックスの、現在の設定に対応する文字列
DEBUG	[プロジェクトのプロパティ] ダイアログ ボックスで設定できる **Boolean** 値既定では、プロジェクトの Debug 構成により、**DEBUG** が定義されます。**DEBUG** を定義すると、Debug クラスのメソッドは出力ウィンドウに出力を生成します。DEBUG を定義しない場合、**Debug** クラスのメソッドはコンパイルされず、デバッグ出力も生成されません。
TARGET	プロジェクトの出力の種類、またはコマンド ラインの **/target** オプションの設定を表す文字列**TARGET** には、次の値を指定できます。 • Windows アプリケーション用の "winexe" • コンソール アプリケーション用の "exe" • クラス ライブラリ用の "library" • モジュール用の "module" • Visual Studio 統合開発環境では、**/target** オプションを設定できます。詳細については、「/target (Visual Basic)」を参照してください。
TRACE	[プロジェクトのプロパティ] ダイアログ ボックスで設定できる **Boolean** 値既定では、プロジェクトのすべての構成により、**TRACE** が定義されます。**TRACE** を定義すると、Trace クラスのメソッドは出力ウィンドウに出力を生成します。定義しない場合、**Trace** クラスのメソッドはコンパイルされず、**Trace** 出力も生成されません。
VBC_VER	*major.minor* 形式で、Visual Basic バージョンを表す数Visual Basic 2005 のバージョン番号は 8.0 です。

印刷と表示の定数

印刷および表示の関数を呼び出すときに、実際の値の代わりにコード内で次の定数を使用できます。

定数	説明
vbCrLf	キャリッジ リターン文字とライン フィード文字の組み合わせ。
vbCr	キャリッジ リターン文字。
vbLf	ライン フィード文字。
vbNewLine	改行文字
vbNullChar	値 0 を持つ文字。
vbNullString	長さ 0 の文字列 ("") とは異なります。外部プロシージャを呼び出す場合に使用します。

vbObjectError	エラー番号。ユーザー定義エラーの番号は、これより大きい値にします。たとえば次のように指定します。たとえば、次のようになります。 `Err.Raise(Number) = vbObjectError + 1000`
vbTab	タブ文字。
vbBack	バックスペース文字。
vbFormFeed	Microsoft Windows では使用されていません。
vbVerticalTab	Microsoft Windows では使用できません。

参照

処理手順

方法 : 定数を宣言する

概念

Visual Basic で宣言されている列挙型

定数の概要

ユーザー定義関数

定数とリテラルのデータ型

方法：定数を宣言する

定数を宣言して値を設定するには、**Const** ステートメントを使います。定数を宣言すると、値にわかりやすい名前を割り当てることができます。一度宣言した定数は、変更したり新しい値を代入したりはできません。

定数は、プロシージャ内で宣言するか、モジュール、クラス、または構造体の宣言セクションで宣言します。クラスまたは構造体のレベルの定数は、既定では **Private** ですが、コード アクセスの必要なレベルに応じて **Public**、**Friend**、**Protected**、または **Protected Friend** として宣言することもできます。

定数には、有効なシンボル名 (変数の名前付け規則と同じ規則に従う)、および数値や文字列の定数と演算子とを組み合わせた式を指定する必要があります。関数呼び出しを含むことはできません。

メモ：

使用している設定またはエディションによっては、ダイアログ ボックスで使用可能なオプションや、メニュー コマンドの名前や位置がヘルプに記載されている内容と異なる場合があります。このヘルプ ページは、全般的な開発設定を考慮して記述されています。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

定数を宣言するには

- 次の例に示すように、アクセス指定子、**Const** キーワード、および式を含む宣言を記述します。

  VB

  ```
  Public Const DaysInYear = 365
  Private Const WorkDays = 250
  ```

  Option Strict が **On** である場合、データ型
  (**Boolean**、**Byte**、**Char**、**DateTime**、**Decimal**、**Double**、**Integer**、**Long**、**Short**、**Single**、または **String**) を指定して、定数を明示的に宣言する必要があります。**Option Strict** が **Off** である場合、定数のデータ型はコンパイラによって代入されます。詳細については、「定数とリテラルのデータ型」を参照してください。

Option Strict が On の場合に定数を宣言するには

- 次の例に示すように、**Option Strict On** を使用して、**As** キーワードおよび明示的なデータ型が含まれる宣言を記述します。

  VB

  ```
  Public Const MyInteger As Integer = 42
  Private Const DaysInWeek As Short = 7
  Protected Friend Const Funday As String = "Sunday"
  ```

  複数の定数を 1 行で宣言することもできますが、1 行で宣言する定数は 1 つだけにした方がコードが読みやすくなります。1 行で複数の定数を宣言した場合、すべてが同じアクセス レベル (**Public**、**Private**、**Friend**、**Protected**、または **Protected Friend**) を持っている必要があります。

複数の定数を 1 行で宣言するには

- 次の例に示すように、宣言をカンマと空白で区切ります。

  ```
  Public Const Four As Integer = 4, Five As Integer = 5, Six As Integer = 44
  ```

参照

処理手順

方法 : 列挙型を宣言する

関連項目

Const ステートメント (Visual Basic)

Option Strict ステートメント

概念

定数とリテラルのデータ型

列挙型の概要

定数の概要

列挙型と名前の修飾

組み込み定数と組み込み列挙型

その他の技術情報

Visual Basic の定数と列挙体

ユーザー定義関数

定数とは、数字または文字列の代わりに使用される名前のことであり、その値は不変です。定数に格納された値は、その名が示すとおり、アプリケーションの実行中に変わることはありません。使用するコントロールやコンポーネントによって定義されている定数を使用するか、または独自の定数を作成できます。自分で作成した定数は、ユーザー定義と呼ばれます。

定数は、変数名を作成するときと同じガイドラインに従って、**Const** ステートメントで宣言します。**Option Strict** が **On** である場合、定数の型を明示的に宣言する必要があります。

Const ステートメントの使用方法

Const ステートメントでは、数値および日付や時刻を表す値を定数の値にできます。

VB

```
Const conPi = 3.14159265358979
Public Const conMaxPlanets As Integer = 9
Const conReleaseDate = #1/1/1995#
```

また、文字列型 (**String**) の定数も定義できます。

VB

```
Public Const conVersion = "07.10.A"
Const conCodeName = "Enigma"
```

通常、等号 (=) の右側には数値またはリテラル文字列を指定しますが、数値または文字列に評価される式を指定することもできます。ただし、指定する式の中に関数の呼び出しを含めることはできません。また、既に定義された定数を使って別の定数を定義することもできます。次に例を示します。

VB

```
Const conPi2 = conPi * 2
```

ユーザー定義定数のスコープ

Const ステートメントのスコープは、同じ場所で宣言される変数のスコープと同じです。スコープは、次のいずれかの方法で指定できます。

- プロシージャ内だけで存在する定数の宣言は、そのプロシージャ内で行います。
- 同じクラスのすべてのプロシージャで使用でき、ほかのモジュールでは使用できない定数の宣言は、クラスの宣言セクションで行います。
- アセンブリのすべてのメンバが使用でき、アセンブリの外部のクライアントは使用できない定数の宣言は、クラスの宣言セクションで **Friend** キーワードを使って行います。
- アプリケーション全体で使用できる定数の宣言は、クラスの宣言セクションで **Public** キーワードを使って行います。

詳細については、「方法 : 定数を宣言する」を参照してください。

循環参照の回避

定数の値は、他の定数を使って定義できるため、複数の定数の間で循環参照が発生する可能性があります。循環参照は、次のように、複数のパブリック定数が互いに参照し合うと発生します。

VB

```
    Public Const conA = conB * 2
...    Public Const conB = conA / 2
```

循環参照があると、Visual Basic でコンパイラ エラーが発生します。

参照

処理手順
方法 : 列挙型を宣言する
関連項目
Const ステートメント (Visual Basic)
Option Strict ステートメント
概念
定数とリテラルのデータ型
列挙型の概要
定数の概要
列挙型と名前の修飾
組み込み定数と組み込み列挙型
定数の概要
その他の技術情報
Visual Basic の定数と列挙体

定数とリテラルのデータ型

リテラルは、変数の値または式の結果としてではなく、数字の 3 や文字列 "Hello" などのように、そのままの形で表される値です。定数は、リテラルの代わりに使用される有効な名前であり、値が変化する変数とは対照的に、プログラム全体をとおして同じ値を保持します。

Option Strict が **On** の場合、明示的にデータ型を指定してすべての定数を宣言する必要があります。次の例では、`MyByte` のデータ型が、データ型 **Byte** として明示的に宣言されています。

```
Option Strict On
Public Const MyByte As Byte = 2
```

データ型を明示的に宣言すると、あいまいさがなくなり、型を指定しないコードよりもわかりやすくなります。ただし、**Option Strict** が **Off** の場合、コンパイラは定数を初期化する式の型を使用します。整数リテラルは、既定で整数型 (**Integer**) にキャストされます。浮動小数点数の既定のデータ型は **Double** であり、キーワード **True** と **False** は **Boolean** 定数を指定します。

リテラルと型の自動変換

大きな整数リテラル値を **Decimal** の変数に代入する場合など、リテラルを特定のデータ型に強制的に変換することが必要な場合があります。次の例では、エラーが発生します。

```
Dim myDecimal as Decimal
myDecimal = 100000000000000000000   ' This causes a compiler error.
```

このエラーの原因は、リテラルの表現にあります。**Decimal** データ型はこの大きさの値を格納できますが、リテラルは暗黙的に、この大きさの値を格納できない **Long** として表されています。

リテラルを特定のデータ型に自動的に変換するには、2 つの方法があります。リテラルに型文字を追加する方法と、リテラルを囲み文字の間に配置する方法です。型文字や囲み文字は定数の直前、直後に指定する必要があります。間にはどのような文字も空白も指定しないでください。

前の例を修正するには、リテラルの末尾に型文字 `D` を追加します。これにより、リテラルが **Decimal** として表されるようになります。

```
MyDecimal = 100000000000000000000D
```

型文字と囲み文字の正しい使用方法を次の例に示します。

```
Option Strict Off
Public Const DefaultInteger = 100   ' Default is Integer.
Public Const DefaultDouble = 54.3345612   ' Default is Double.
Public Const MyCharacter = "a"C   ' Forces constant to be a Char type.
Public Const MyDate = #01/15/01#   ' Demonstrates DateTime constants.
Public Const MyTime = #1:15:59 AM#
Public Const MyLong = 45L   ' Forces data type to be a Long.
Public Const MySingle = 45.55!   ' Forces data type to be a Single.
```

次の表は、Visual Basic で使用できる囲み文字と型文字の一覧です。

データ型	囲み文字	末尾の型文字
Boolean	(なし)	(なし)
Byte	(なし)	(なし)
Char	"	C
Date	#	(なし)
Decimal	(なし)	D または @

Double	(なし)	R または #
Integer	(なし)	I または %
Long	(なし)	L または &
Short	(なし)	S
Single	(なし)	F または !
String	"	(なし)

参照

処理手順

方法 : 定数を宣言する
方法 : 列挙型を宣言する

関連項目

Option Strict ステートメント
Option Explicit ステートメント (Visual Basic)
データ型の概要 (Visual Basic)

概念

ユーザー定義関数
定数の概要
列挙型の概要
列挙型と名前の修飾
組み込み定数と組み込み列挙型

方法：関連する定数値をまとめてグループ化する

関連する定数値をまとめてグループ化するには、列挙型が最適です。列挙型を作成するには、クラスまたはモジュールの宣言セクションで **Enum** ステートメントを使用します。詳細については、「方法 : 列挙型を宣言する」を参照してください。

関連する定数値をグループ化するには

1. コードのアクセス レベル、**Enum** キーワード、および有効な名前を含む宣言を作成します。この例では、**Private** 列挙型、`temperatureValues` を作成します。

   VB

   ```
   Private Enum temperatureValues
   ```

2. 列挙型の定数を定義します。この例では **Public** 列挙型 `temperatureValues` を作成し、値を割り当てます。

   VB

   ```
   Public Enum temperatureValues
     Scorching
     Hot
     Lukewarm
     Cool
     Cold
   End Enum
   ```

参照

処理手順

方法 : 列挙型のメンバを参照する

概念

列挙型と名前の修飾
列挙型を使用する状況
組み込み定数と組み込み列挙型
定数の概要
定数とリテラルのデータ型

列挙型の概要

複数の関連する定数を操作する場合や、複数の定数値に名前を関連付ける場合は、列挙型を使うと便利です。たとえば、一連の整数型の定数を曜日に関連付けて列挙型として宣言すると、整数値の代わりに曜日名を使うことができます。

列挙型に関するタスク

次の表は、列挙型に関する一般的なタスクの一覧です。

目的	参照項目
定義済みの列挙型を検索する	Visual Basic で宣言されている列挙型
列挙型を宣言する	方法 : 列挙型を宣言する
列挙型の名前を完全修飾する	列挙型と名前の修飾
列挙型のメンバを参照する	方法 : 列挙型のメンバを参照する
列挙型を反復処理する	方法 : Visual Basic で列挙型を反復処理する
列挙値に関連付けられている文字列を確認する	方法 : 列挙値に関連付けられている文字列を確認する
列挙型を使用する状況を決定する	列挙型を使用する状況

参照

処理手順

方法 : 定数を宣言する

関連項目

Enum ステートメント (Visual Basic)

概念

定数の概要
ユーザー定義関数
定数とリテラルのデータ型
組み込み定数と組み込み列挙型

Visual Basic で宣言されている列挙型

複数の関連する定数を扱う場合や、複数の定数値に名前を関連付ける場合は、列挙型を使うと便利です。Visual Basic には、多数の便利な列挙型が用意されています。

定義済み列挙型

Visual Basic で提供されている列挙型について、次の表で説明します。

列挙型	説明
AppWinStyle 列挙型	**Shell** 関数を呼び出すときに、実行されるプログラムで使用するウィンドウ スタイルを示します。
AudioPlayMode 列挙型	オーディオ関連のメソッドを呼び出すときに、サウンドの再生方法を示します。
BuiltInRole 列挙型	My.User.IsInRole メソッドを呼び出すときに、チェックするロールの種類を示します。
CallType 列挙型	**CallByName** 関数を呼び出すときに、起動するプロシージャの種類を示します。
CompareMethod 列挙型	比較関数を呼び出すときに、文字列を比較する方法を示します。
DateFormat 列挙型	**FormatDateTime** 関数を呼び出すときに、日付の表示方法を示します。
DateInterval 列挙型	日付関連の関数を呼び出すときに、日付の間隔の決定方法および形式を示します。
DeleteDirectoryOption 列挙型	削除対象のディレクトリ内にファイルまたはディレクトリが存在する場合の処理を指定します。
DueDate 列挙型	財務関連のメソッドを呼び出すときに、支払い期日を示します。
FieldType 列挙型	テキスト フィールドが区切られているのか、固定幅なのかを示します。
FileAttribute 列挙型	ファイル アクセス用の関数を呼び出すときに、使用するファイル属性を示します。
FirstDayOfWeek 列挙型	日付関連の関数を呼び出すときに使用する、1 週間の最初の日を示します。
FirstWeekOfYear 列挙型	日付関連の関数を呼び出すときに使用する、1 年の最初の週を示します。
MsgBoxResult 列挙型	メッセージ ボックスで押されたボタンを示します。**MsgBox** 関数から返されます。
MsgBoxStyle 列挙型	**MsgBox** 関数を呼び出すときに、表示するボタンを示します。
OpenAccess 列挙型	ファイル アクセス用の関数を呼び出すときに、アクセスの種類を示します。
OpenMode 列挙型	ファイル アクセス用の関数を呼び出すときに、使用するモードを示します。
OpenShare 列挙型	ファイル アクセス用の関数を呼び出すときに、そのファイルを他のプロセスで使用できるようにするかどうかを示します。
RecycleOption 列挙型	ファイルを完全に削除するのか [ごみ箱] に入れるのかを指定します。
SearchOption 列挙型	すべてのディレクトリを検索するのか、トップ レベルのディレクトリのみを検索するのかを指定します。
TriState 列挙型	数値書式指定関数を呼び出すときに、**Boolean** 値と既定値のどちらを使用するのかを示します。

UICancelOption 列挙型	操作中にユーザーが [キャンセル] ボタンをクリックしたときに実行する処理を指定します。
UIOption 列挙型	ファイルまたはディレクトリをコピー、削除、または移動するときに、プログレス ダイアログ ボックスを表示するかどうかを指定します。
VariantType 列挙型	**VarType** 関数が返すバリアント オブジェクトの型を示します。
VbStrConv 列挙型	**StrConv** 関数を呼び出すときに、実行する変換の種類を示します。
DueDate 列挙型	財務関連のメソッドを呼び出すときに、支払い期日を示します。

参照

処理手順

方法 : 列挙型を宣言する
方法 : 列挙型のメンバを参照する
方法 : Visual Basic で列挙型を反復処理する
方法 : 列挙値に関連付けられている文字列を確認する

概念

列挙型と名前の修飾
列挙型を使用する状況

方法：列挙型を宣言する

列挙型を作成するには、クラスまたはモジュールの宣言セクションで **Enum** ステートメントを使います。列挙型をメソッド内で宣言することはできません。適切なレベルのアクセスを指定するには、**Private**、**Protected**、**Friend**、または **Public** を使用します。

Enum 型には、名前、基になる型、およびフィールドのセットがあり、各フィールドは定数を表します。名前は、有効な Visual Basic 2005 修飾子である必要があります。基になる型は、**Byte**、**Short**、**Long** または **Integer** の整数型のいずれかである必要があります。**Integer** が既定です。列挙型は常に厳密に型指定され、整数の数値型と互いに置き換えることはできません。

列挙型は、浮動小数点値を持つことはできません。**Option Strict On** を使用して、列挙型に浮動小数点値を代入すると、コンパイラ エラーになります。**Option Strict** が **Off** である場合、値が自動的に **Enum** 型に変換されます。

名前について、および **Imports** ステートメントを使用して名前の修飾をしなくて済む方法については、「列挙型と名前の修飾」を参照してください。

列挙型を宣言するには

1. それぞれで異なる **Enum** が宣言されている次のコード例のように、コード アクセス レベル、**Enum** キーワード、および有効な名前が含まれる宣言を記述します。

   VB

   ```
   Private Enum SampleEnum
           SampleMember
       End Enum
       Public Enum SampleEnum2
           SampleMember
       End Enum
       Protected Enum SampleEnum3
           SampleMember
       End Enum
       Friend Enum SampleEnum4
           SampleMember
       End Enum
       Protected Friend Enum SampleEnum5
           SampleMember
       End Enum
   ```

2. 列挙型で定数を定義します。既定では、列挙型の中の最初の定数は `0` に初期化され、それに続く定数は、それぞれ前の定数に 1 を加えた値に初期化されます。たとえば、次の列挙型 `Days` では、`Sunday` の値は `0`、`Monday` の値は `1`、`Tuesday` の値は `2` になります。

   VB

   ```
   Public Enum Days
     Sunday
     Monday
     Tuesday
     Wednesday
     Thursday
     Friday
     Saturday
   End Enum
   ```

3. 代入ステートメントを使うと、列挙型の中の定数に明示的に値を代入できます。定数には、負の数を含む任意の整数値を代入できます。たとえば、エラーの状況を表すために、0 よりも小さな値の定数を使うことができます。次の列挙型では、定数 `Invalid` に値 `-1` が明示的に代入されます。また、定数 `Sunday` に、値 `0` が代入されます。これは列挙型の最初の定数であるため、`Saturday` も値 `0` に初期化されます。`Monday` の値は `1` です (`Sunday` の値よりも 1 つ多い)。また、`Tuesday` の値は `2`、のように続きます。

VB

```
Public Enum WorkDays
  Saturday
  Sunday = 0
  Monday
  Tuesday
  Wednesday
  Thursday
  Friday
  Invalid = -1
End Enum
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

列挙型を明示的に宣言するには

- 次の構文を使用して宣言を記述します。

VB

```
Public Enum MyEnum As Byte
  Zero
  One
  Two
End Enum
```

参照

処理手順

方法 : 列挙型のメンバを参照する
方法 : Visual Basic で列挙型を反復処理する
方法 : 列挙値に関連付けられている文字列を確認する

概念

列挙型と名前の修飾
列挙型を使用する状況
組み込み定数と組み込み列挙型
定数とリテラルのデータ型

その他の技術情報

Visual Basic の定数

方法：列挙型のメンバを参照する

複数の関連する定数を操作する場合や、複数の定数値に名前を関連付ける場合は、列挙型を使うと便利です。たとえば、一連の整数型の定数を曜日に関連付けて列挙型として宣言すると、整数値の代わりに曜日名を使うことができます。

Imports ステートメントによって、完全修飾名の使用を回避できます。詳細については、「列挙型と名前の修飾」を参照してください。

列挙型のメンバを参照するには

- メンバ名を列挙型で修飾します。たとえば次の例では、変数 `DayValue` に、**FirstDayOfWeek** 列挙型の **Saturday** メンバを割り当てます。

  VB

```
DayValue = FirstDayOfWeek.Saturday
```

参照

処理手順

方法 : 列挙型を宣言する
方法 : Visual Basic で列挙型を反復処理する
方法 : 列挙値に関連付けられている文字列を確認する

概念

Visual Basic で宣言されている列挙型
列挙型と名前の修飾
列挙型を使用する状況

列挙型と名前の修飾

通常、列挙型のメンバを参照する場合は、その列挙型の名前で修飾する必要があります。たとえば、列挙型 `Days` のメンバ `Sunday` を参照するには、次の構文を使います。

VB

```
X = Days.Sunday
```

Imports ステートメントの使用

次の例に示すように、コードの名前空間宣言セクションに **Imports** ステートメントを追加すると、完全限定名を使わなくても済むようになります。

VB

```
Imports WindowsApplication1.Form1.Days
Imports WindowsApplication1.Form1.WorkDays
```

Imports ステートメントは、参照されるプロジェクトおよびアセンブリから、およびステートメントが使用されるモジュールと同じプロジェクト内から、名前空間の名前をインポートします。このステートメントを追加すると、次のように、修飾子を付けずに列挙型のメンバを参照できるようになります。

VB

```
X = Sunday
```

関連する定数を列挙型にまとめると、同じ定数名を異なるコンテキストで使うことができます。たとえば、列挙型 `Days` と列挙型 `WorkDays` の両方で、曜日の定数に対して同じ名前を使用できます。列挙型で **Imports** ステートメントを使う場合は、あいまいな参照にならないように注意する必要があります。次に例を示します。

VB

```
Imports WindowsApplication1.Form1.Days
Imports WindowsApplication1.Form1.WorkDays
```

VB

```
Public Sub New()
  ' Insert code to implement constructor.
  X = Monday
End Sub
```

`Monday` が `Days` と `Workdays` の両方の列挙型のメンバになっている場合、このコードはコンパイラ エラーになります。定数を個別に参照するときにあいまいな参照を回避するには、定数名を列挙型で修飾します。次のコードは、列挙型 `Days` と列挙型 `WorkDays` の定数 `Saturday` を参照します。

VB

```
X = Days.Saturday
Y = WorkDays.Saturday
```

参照

処理手順

方法 : 列挙型を宣言する
方法 : 列挙型のメンバを参照する
方法 : Visual Basic で列挙型を反復処理する

方法 : 列挙値に関連付けられている文字列を確認する

関連項目

Enum ステートメント (Visual Basic)

Imports ステートメント

データ型の概要 (Visual Basic)

概念

Visual Basic で宣言されている列挙型

列挙型を使用する状況

定数とリテラルのデータ型

方法：Visual Basic で列挙型を反復処理する

複数の関連する定数を操作する場合や、複数の定数値に名前を関連付ける場合は、列挙型を使うと便利です。列挙型を反復処理するために、GetValues メソッドを使用して、列挙型を配列に移動できます。また、**For...Each** ステートメントを使用したり、GetNames メソッドまたは **GetValues** メソッドを使用して文字列や数値を抽出したりして、列挙型を反復処理することもできます。

列挙型を反復処理するには

- 他の変数で行うように、配列を渡す前に、配列を宣言し、**GetValues** メソッドを使用して列挙型を変換します。次の例は、列挙体 `MyEnum` を繰り返し処理して、列挙体の各メンバを表示します。

  VB

```
Dim items As Array
items = System.Enum.GetValues(GetType(FirstDayOfWeek))
Dim item As String
For Each item In items
    MsgBox(item)
Next
```

参照

処理手順

方法 : 列挙型を宣言する
方法 : 列挙値に関連付けられている文字列を確認する
方法 : 列挙型のメンバを参照する
方法 : プロシージャまたはプロパティに配列を渡す

概念

列挙型の概要
列挙型を使用する状況
列挙型と名前の修飾
Visual Basic で宣言されている列挙型
Visual Basic の配列の概要

方法：列挙値に関連付けられている文字列を確認する

GetValues および GetNames メソッドを使用すると、列挙値のメンバに関連付けられている文字列と値を確認できます。

列挙値に関連付けられている文字列を確認するには

- **GetNames** メソッドを使用して、列挙値のメンバに関連付けられている文字列を取得します。この例では、列挙型 `flavorEnum` を宣言し、次に **GetNames** メソッドを使用して各メンバに関連付けられている文字列を表示します。

  VB

```
Public Enum flavorEnum
  salty
  sweet
  sour
  bitter
End Enum

Private Sub TestMethod()
  MsgBox("The strings in the flavorEnum are:")
  Dim i As String
  For Each i In [Enum].GetNames(GetType(flavorEnum))
      MsgBox(i)
  Next
End Sub
```

参照

処理手順

方法 : 列挙型を宣言する
方法 : 列挙型のメンバを参照する
方法 : Visual Basic で列挙型を反復処理する

関連項目

Enum ステートメント (Visual Basic)
GetValues
GetNames
Enum

概念

Visual Basic で宣言されている列挙型
列挙型と名前の修飾
列挙型を使用する状況

列挙型を使用する状況

列挙型を使用すると、関連する定数のセットを扱うのが簡単になります。列挙型 (**Enum**) は、値のセットを表すシンボル名です。列挙型はデータ型として扱われ、これを使用して変数およびプロパティで使用する定数のセットを作成できます。

列挙型を使用する状況

限定された変数のセットをプロシージャが受け取ったときは、列挙型を使用してください。特に意味のある名前が使用されている場合は、列挙型によってわかりやすく読みやすいコードになります。

列挙型を使用する利点は、次のとおりです。

- 数値の置き換えやタイプミスによるエラーが減少する
- 将来、値を変更しやすい
- コードが読みやすくなり、エラーが発生しにくくなる
- 上位互換性が確保できる列挙型を使用すると、メンバの名前に対応する値を将来だれかが変更した場合、コードがエラーになる可能性が低くなります。

列挙型の名前付け

列挙型メンバの名前付け規則を使用します。Visual Basic によって列挙型のメンバ名が検出されると、他の参照される型のライブラリに同じ名前が含まれている場合、例外がスローされます。アプリケーションまたはコンポーネントの値を識別する一意のプレフィックスを使用します。

列挙型のメンバを参照するとき、列挙型の名前でメンバ名を修飾するか、または **Imports** ステートメントを使用する必要があります。詳細については、「列挙型と名前の修飾」を参照してください。

定義済み列挙型

Visual Basic には、コードの記述を簡単にするために、**FirstDayOfWeek** および **MsgBoxResul** などの、いくつかの定義済み列挙型が用意されています。これらの一覧については、「Visual Basic で宣言されている列挙型」を参照してください。

参照

処理手順

方法 : 列挙型を宣言する
方法 : 列挙型のメンバを参照する
方法 : Visual Basic で列挙型を反復処理する
方法 : 列挙値に関連付けられている文字列を確認する

関連項目

Enum ステートメント (Visual Basic)

概念

列挙型と名前の修飾
組み込み定数と組み込み列挙型

組み込み定数と組み込み列挙型

Visual Basic には、プログラミングを簡単にするために、いくつかの組み込み定数と組み込み列挙型が用意されています。

解説

定数に格納された値は、アプリケーションの実行中に変わることはありません。定数は数値や文字列の代わりに使用されるわかりやすい名前であり、それによってコードが読みやすくなります。列挙型を使用すると、関連する定数のセットを扱うのが簡単になります。列挙型 (Enum) は、値のセットを表すシンボル名です。

「ユーザー定義関数」の説明に従って独自の定数および列挙型を作成できます。または、Visual Basic に用意されている組み込み定数および組み込み列挙型を使用できます。「オブジェクト ブラウザ」に一覧があります。

Visual Basic で使用可能な組み込み列挙型について、次の表で説明します。

列挙型	説明
AppWinStyle 列挙型	**Shell** 関数を呼び出したときに実行されるプログラムで使用するウィンドウ スタイルを示します。
AudioPlayMode 列挙型	オーディオ メソッドを呼び出すときの再生方法を示します。
BuiltInRole 列挙型	My.User.IsInRole メソッドを呼び出すときにチェックするロールの種類を示します。
CallType 列挙型	**CallByName** 関数を呼び出したときに実行されるプロシージャの種類を示します。
CompareMethod 列挙型	比較関数を呼び出したときに文字列を比較する方法を示します。
DateFormat 列挙型	**FormatDateTime** 関数を呼び出したときに日付を表示する方法を示します。
DateInterval 列挙型	日付関連の関数を呼び出したときの日付の間隔の決定方法と書式指定方法を示します。
DeleteDirectoryOption 列挙型	削除対象のディレクトリ内にファイルまたはディレクトリが存在する場合の処理を指定します。
DueDate 列挙型	財務関係のメソッドを呼び出すときに支払い期日を示します。
FieldType 列挙型	テキスト フィールドを区切るか、または固定長にするのかを指定します。
FileAttribute 列挙型	ファイル アクセス用の関数を呼び出すときに使用するファイル属性を示します。
FirstDayOfWeek 列挙型	日付関連の関数を呼び出すときに使用する週の最初の曜日を示します。
FirstWeekOfYear 列挙型	日付関連の関数を呼び出すときに使用する年度の最初の週を示します。
MsgBoxResult 列挙型	**MsgBox** 関数によって返される場合、メッセージ ボックスでどのボタンがクリックされたのかを示します。
MsgBoxStyle 列挙型	**MsgBox** 関数を呼び出したときに表示されるボタンを示します。
OpenAccess 列挙型	ファイルが読み取り可能、書き込み可能、その両方、またはそのいずれかであるかを示します。
OpenMode 列挙型	ファイルにアクセスするときに使用するアクセス モードを示します。
OpenShare 列挙型	他の処理で、開いているファイルにアクセスできるかどうかを示します。
RecycleOption 列挙型	ファイルを完全に削除するか、ごみ箱に移動するかを指定します。
SearchOption 列挙型	すべてのディレクトリまたは最上位レベルのディレクトリの、いずれを探索するのかを指定します。

TriState 列挙型	数値書式指定関数を呼び出すときに、Boolean 値または既定のいずれを使用するのかを示します。
UICancelOption 列挙型	操作中にユーザーが [キャンセル] ボタンをクリックしたときに実行する処理を指定します。
UIOption 列挙型	ファイルまたはディレクトリをコピー、削除、または移動するときに、進行状況を示すダイアログ ボックスを表示するかどうかを指定します。
VariantType 列挙型	**VarType** 関数によって返される、バリアント オブジェクトのタイプを示します。
VbStrConv 列挙型	**StrConv** 関数を呼び出したときに実行する変換の型を示します。

参照

処理手順

方法 : 定数を宣言する

関連項目

Enum ステートメント (Visual Basic)

オブジェクト ブラウザ

概念

定数の概要

定数とリテラルのデータ型

列挙型の概要

ユーザー定義関数

その他の技術情報

定数と列挙型 (Visual Basic)

Visual Basic の演算子および式

演算子は、値が格納されている 1 つ以上のコード要素に対して演算を実行するコード要素です。値要素は、変数、定数、リテラル、プロパティ、**Function** プロシージャおよび **Operator** プロシージャからの戻り値、式などです。

式は、演算子で結合され、新しい値を生成する一連の値要素です。演算子は、値要素に対して、計算、比較、またはその他の演算を実行します。

演算子の型

Visual Basic には、次のような種類の演算子があります。

- 算術演算子は、数値に対して一般的な計算を実行します (ビット パターンのシフトも含まれます)。
- 比較演算子は、2 つの式を比較し、比較の結果を表す **Boolean** 値を返します。
- 連結演算子は、複数の文字列を結合して 1 つの文字列にします。
- Visual Basic の論理演算子およびビット処理演算子は、**Boolean** 値または数値を組み合わせて、同じデータ型の結果を値として返します。

演算子で結合される値要素は、演算子のオペランドと呼ばれます。演算子は値要素と結合されて式を構成します。ただし、代入演算子は例外で、これはステートメントを構成します。詳細については、「代入ステートメント」を参照してください。

式の評価

式の最終的な結果は値で表されます。通常、この値は **Boolean**、**String**、または数値型です。

次に式の例をいくつか示します。

```
5 + 4

' The preceding expression evaluates to 9.

15 * System.Math.Sqrt(9) + x

' The preceding expression evaluates to 45 plus the value of x.

"Concat" & "ena" & "tion"

' The preceding expression evaluates to "Concatenation".

763 < 23

' The preceding expression evaluates to False.
```

次の例ように、1 つの式またはステートメントで複数の演算子を使うこともできます。

VB

```
x = 45 + y * z ^ 2
```

この例では、Visual Basic によって代入演算子 (=) の右側にある式の演算が実行され、次にその結果の値が左側にある変数 `x` に代入されます。1 つの式で使用できる演算子の数には、事実上制限はありません。ただし、正しい結果を得るためには、Visual Basic における演算子の優先順位を理解する必要があります。

参照

概念

演算子の効率のよい組み合わせ

その他の技術情報

演算子 (Visual Basic)

Visual Basic におけるステートメント

Visual Basic の演算子で実行される一般的な演算

演算子は、オペランドと呼ばれる 1 つまたは複数の式を使用してさまざまな演算を実行します。

算術演算子とビット シフト演算子

使用可能な算術演算子とビット シフト演算子を次の表に使します。

目的	参照項目
一方の数値を他方の数値に足す	+ 演算子 (Visual Basic)
一方の数値を他方の数値から引く	- 演算子 (Visual Basic)
数値の符号を反転させる	- 演算子 (Visual Basic)
一方の数値に他方の数値を掛ける	* 演算子 (Visual Basic)
一方の数値を他方の数値で割る	/ 演算子 (Visual Basic)
一方の数値を他方の数値で割ったときの商を求める (剰余は求めない)	\ 演算子
一方の数値を他方の数値で割ったときの剰余を求める (商は求めない)	Mod 演算子 (Visual Basic)
一方の数値を他方の数値で累乗する	^ 演算子 (Visual Basic)
数値のビット パターンを左にシフトする	<< 演算子 (Visual Basic)
数値のビット パターンを右にシフトする	>> 演算子 (Visual Basic)

比較演算子

使用可能な比較演算子を次の表に示します。

目的	参照項目
2 つの値が等しいことを確認する	= 演算子 (Visual Basic における比較演算子)
2 つの値が等しくないことを確認する	<> 演算子 (Visual Basic における比較演算子)
一方の値が他方の値より小さいことを確認する	< 演算子 (Visual Basic における比較演算子)
一方の値が他方の値より大きいことを確認する	> 演算子 (Visual Basic における比較演算子)
一方の値が他方の値以下であることを確認する	<= 演算子 (Visual Basic における比較演算子)
一方の値が他方の値以上であることを確認する	>= 演算子 (Visual Basic における比較演算子)
2 つのオブジェクト変数が同じオブジェクト インスタンスを参照していることを確認する	Is 演算子 (Visual Basic)
2 つのオブジェクト変数が別のオブジェクト インスタンスを参照していることを確認する	IsNot 演算子
オブジェクトが特定の型であることを確認する	TypeOf 演算子 (Visual Basic)

連結演算子

使用可能な連結演算子を次の表に示します。

目的	参照項目
複数の文字列を 1 つの文字列に結合する	**&** 演算子 (Visual Basic の連結演算子)
数値を文字列値に結合する	**+** 演算子 (Visual Basic の連結演算子)

論理演算子とビット処理演算子

使用可能な論理演算子とビット処理演算子を次の表に示します。

目的	参照項目
ブール値の論理否定を求める	Not 演算子 (Visual Basic)
2 つのブール値の論理積を求める	And 演算子 (Visual Basic)
2 つのブール値の包括的論理和を求める	Or 演算子 (Visual Basic)
2 つのブール値の排他的論理和を求める	Xor 演算子 (Visual Basic)
2 つのブール値の論理積をショートサーキットで求める	AndAlso 演算子
2 つのブール値の包括的論理和をショート サーキットで求める	OrElse 演算子
2 つの整数値のビットごとの論理積を求める	And 演算子 (Visual Basic)
2 つの整数値のビットごとの包括的論理和を求める	Or 演算子 (Visual Basic)
2 つの整数値のビットごとの排他的論理和を求める	Xor 演算子 (Visual Basic)
整数値のビットごとの論理否定を求める	Not 演算子 (Visual Basic)

参照

関連項目

機能別の演算子一覧

概念

Visual Basic の演算子および式

Visual Basic における算術演算子

算術演算子を使うと、リテラル、変数、その他の式、関数やプロパティの呼び出し、および定数によって表される数値の計算を含む一般的な算術演算を実行できます。また、ビット シフト演算子も算術演算子に分類されます。ビット シフト演算子はオペランドの個々のビットを操作し、オペランドのビット パターンを左または右にシフトします。

算術演算

式の中の 2 つの値を足す場合は + 演算子 (Visual Basic) を使い、一方の値からもう一方の値を引く場合は - 演算子 (Visual Basic) を使います。次に例を示します。

VB

```
Dim x As Integer
x = 67 + 34
x = 32 - 12
```

否定にも - 演算子 (Visual Basic) を使いますが、1 つのオペランドにのみ使います。次に例を使用します。

VB

```
Dim x As Integer = 65
Dim y As Integer
y = -x
```

乗算には * 演算子 (Visual Basic) を使い、除算には / 演算子 (Visual Basic) を使います。次に例を示します。

VB

```
Dim y As Double
y = 45 * 55.23
y = 32 / 23
```

指数演算には ^ 演算子 (Visual Basic) を使います。次に例を示します。

VB

```
Dim z As Double
z = 23 ^ 3
' The preceding statement sets z to 12167 (the cube of 23).
```

整数の除算には \ 演算子 を使用します。整数の除算は商を返します。商とは、被除数を除数で割った値を、余りを無視して表した整数です。この演算子の場合、除数と被除数はどちらも整数型 (**SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、**ULong**) であることが必要です。他のデータ型を使用する場合は、先に整数型に変換する必要があります。次に整数の除算の例を示します。

VB

```
Dim k As Integer
k = 23 \ 5
' The preceding statement sets k to 4.
```

剰余演算には Mod 演算子 (Visual Basic) を使います。この演算子は、被除数を除数で割った余りを返します。除数と被除数が両方とも整数型の場合、戻り値は整数型になります。除数と被除数が両方とも浮動小数点型の場合、戻り値は浮動小数点型になります。次にコード例を示します。

VB

```
Dim x As Integer = 100
Dim y As Integer = 6
Dim z As Integer
```

```
z = x Mod y
' The preceding statement sets z to 4.
```

VB

```
Dim a As Double = 100.3
Dim b As Double = 4.13
Dim c As Double
c = a Mod b
' The preceding statement sets c to 1.18.
```

0 による除算

0 による除算の結果は、含まれているデータ型によって異なります。整数の除算 (**SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、**ULong**) では、.NET Framework は DivideByZeroException 例外をスローします。除算演算子を **Decimal** データ型や **Single** データ型に使用した場合も、.NET Framework は **DivideByZeroException** 例外をスローします。

Double データ型を含む浮動小数点の除算では、例外はスローされず、結果は被除数に応じて NaN、PositiveInfinity、または NegativeInfinity のいずれかを表すクラス メンバになります。**Double** 値を 0 で除算しようとした結果を表にまとめると、次のようになります。

被除数のデータ型	除数のデータ型	被除数の値	結果
Double	**Double**	0	**NaN** (数学的に定義された数値ではない)。
Double	**Double**	> 0	**PositiveInfinity**
Double	**Double**	< 0	**NegativeInfinity**

DivideByZeroException 例外をキャッチした場合は、そのメンバを例外の処理に利用できます。たとえば、Message プロパティには例外のメッセージ テキストが格納されています。詳細については、「Visual Basic の構造化例外処理の概要」を参照してください。

ビット シフト演算

ビット シフト演算は、ビット パターン上で算術シフトを実行します。パターンは左側のオペランドに指定し、右側のオペランドにはパターンをシフトする位置の数を指定します。パターンを右へシフトする場合は >> 演算子 (Visual Basic) を使い、左へシフトする場合は << 演算子 (Visual Basic) を使います。

左側のオペランドのデータ型は **SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、**ULong** である必要があります。右側のオペランドのデータ型は **Integer** または **Integer** を拡張したデータ型であることが必要です。

数値のシフトは、循環的には行われません。つまり、一方の端からはみ出したビットが、もう一方の端に補われることはありません。シフトによって空いたビット位置は、次のように設定されます。

- 算術左シフトでは 0
- 正の数の算術右シフトでは 0
- 符号のないデータ型の算術右シフトでは 0 (**Byte**、**UShort**、**UInteger**、**ULong**)
- 負の数 (**SByte**、**Short**、**Integer**、または **Long**) の算術右シフトでは 1

次の例は **Integer** 値を左シフトおよび右シフトします。

VB

```
Dim lResult, rResult As Integer
Dim pattern As Integer = 12
' The low-order bits of pattern are 0000 1100.
lResult = pattern << 3
' A left shift of 3 bits produces a value of 96.
rResult = pattern >> 2
' A right shift of 2 bits produces value of 3.
```

数値のシフトではオーバーフロー例外は発生しません。

ビット処理演算

Not、**Or**、**And**、**Xor** の各演算子は、論理演算子として機能する他に、数値に対して使用された場合はビットごとの算術演算子としても機能します。詳細については、「Visual Basic の論理演算子とビット処理演算子」の「ビット処理演算」を参照してください。

タイプ セーフ

オペランドは、通常同じ型である必要があります。たとえば、**Integer** 型の変数を加算するときには、この変数を別の **Integer** 変数に加算し、結果も同じく **Integer** 型の変数に代入する必要があります。

コードをタイプ セーフに保つための 1 つの方法として、Option Strict ステートメント を使用できます。**Option Strict On** を設定すると、Visual Basic はタイプ セーフな変換を自動的に実行します。たとえば、**Integer** 型の変数を **Double** 型の変数に加算し、その値を **Double** 型の変数に代入する場合は、**Integer** 値から **Double** 型への変換ではデータが失われる可能性がないため、演算は正常に処理されます。一方、タイプ セーフではない変換の場合は、**Option Strict On** を設定してしているとコンパイラ エラーが発生します。たとえば、**Integer**型 の変数を **Double** 型の変数に加算し、その値を **Integer** 型の変数に代入しようとすると、**Double** の変数を暗黙的に **Integer** 型に変換できないため、コンパイラ エラーが発生します。

一方、**Option Strict Off** を設定していると、Visual Basic は暗黙の縮小変換を許可します。ただし、これによって予測不可能なデータまたは精度の欠落が起こる可能性があります。このため、実稼動用のコードを作成する際には、**Option Strict On** を設定することをお勧めします。詳細については、「拡大変換と縮小変換」を参照してください。

参照

関連項目

算術演算子 (Visual Basic)
ビット シフト演算子

概念

Visual Basic における比較演算子
Visual Basic の連結演算子
Visual Basic の論理演算子とビット処理演算子
演算子の効率のよい組み合わせ

Visual Basic における比較演算子

比較演算子は 2 つの式を比較し、それらの値の関係を表す **Boolean** 値を返します。比較演算子には、数値を比較する演算子、文字列を比較する演算子、およびオブジェクトを比較する演算子があります。ここでは、これら 3 種類の演算子についてそれぞれ説明します。

数値の比較

Visual Basic では、6 つの数値比較演算子を使って数値を比較します。どの演算子も、数値に評価される 2 つの式をオペランドとして受け取ります。次の表に、これらの演算子とそれぞれの使用例を示します。

演算子	テスト条件	例

= (等しい)	1 番目の式の値と 2 番目の式の値が等しいかどうか。	`23 = 33 ' False`
		`23 = 23 ' True`
		`23 = 12 ' False`

<> (等しくない)	1 番目の式の値と 2 番目の式の値が等しくないかどうか。	`23 <> 33 ' True`
		`23 <> 23 ' False`
		`23 <> 12 ' True`

< (より小さい)	1 番目の式の値が 2 番目の式の値よりも小さいかどうか。	`23 < 33 ' True`
		`23 < 23 ' False`
		`23 < 12 ' False`

> (より大きい)	1 番目の式の値が 2 番目の式の値よりも大きいかどうか。	`23 > 33 ' False`
		`23 > 23 ' False`
		`23 > 12 ' True`

<= (以下)	1 番目の式の値が 2 番目の式の値よりも小さい、または等しいかどうか。	`23 <= 33 ' True`
		`23 <= 23 ' True`
		`23 <= 12 ' False`

>= (以上)	1 番目の式の値が 2 番目の式の値よりも大きい、または等しいかどうか。	`23 >= 33 ' False`
		`23 >= 23 ' True`
		`23 >= 12 ' True`

文字列の比較

Visual Basic では、数値比較演算子の他に Like 演算子を使用して文字列を比較します。**Like** を使うと、パターンが指定できます。このパターンと文字列とを比較し、一致した場合に結果が **True** になります。それ以外の場合、結果は **False** になります。数値演算子を使うと、次の例のように、並べ替え順序に基づいて文字列型 (**String**) の値を比較できます。

```
"73" < "9"

' The result of the preceding comparison is True.
```

この例では、1 番目の文字列の最初の文字の並べ替え順序が 2 番目の文字列の最初の文字より前であるため、結果は **True** になります。最初の文字が同じ文字であった場合は、両方の文字列の次の文字が比較されます。また、次の例に示すように、等号演算子を使って文字列が等しいかどうかを調べることもできます。

```
"734" = "734"

' The result of the preceding comparison is True.
```

一方の文字列全体がもう一方の文字列の先頭部分と一致している場合 ("aa" と "aaa" など) は、長い方の文字列が短い方の文字列よりも大きいと見なされます。次に例を示します。

```
"aaa" > "aa"

' The result of the preceding comparison is True.
```

並べ替え順序は、**Option Compare** の設定により、バイナリ モードの比較またはテキスト モードの比較が基になります。詳細については、

「Option Compare ステートメント」を参照してください。

オブジェクトの比較

Visual Basic では、2 つのオブジェクト参照変数を Is 演算子 (Visual Basic) と IsNot 演算子を使って比較します。この 2 つの演算子のどちらを使っても、2 つの参照変数が同じオブジェクトのインスタンスを参照しているかどうかを調べることができます。次に例を示します。

VB

```
Dim x As testClass
Dim y As New testClass()
x = y
If x Is y Then
    ' Insert code to run if x and y point to the same instance.
End If
```

この例では、両方の変数が同じインスタンスを参照しているので、`x Is y` は **True** に評価されます。この結果を次の例と比べてみてください。

VB

```
Dim x As New customer()
Dim y As New customer()
If x Is y Then
    ' Insert code to run if x and y point to the same instance.
End If
```

この例では、`x Is y` は **False** になります。これは、変数は同じ型のオブジェクトを参照していますが、参照しているインスタンスが異なるためです。

2 つのオブジェクトが同じインスタンスをポイントしていないことを調べる場合は、**IsNot** 演算子を使うと **Not** と **Is** を合わせた不恰好な文法を使わずに済みます。次に例を示します。

VB

```
Dim a As New classA()
Dim b As New classB()
If a IsNot b Then
    ' Insert code to run if a and b point to different instances.
End If
```

この例の `If a IsNot b` は、`If Not a Is b` と同じ意味になります。

オブジェクト型の比較

オブジェクトが特定の型かどうかを調べるには、**TypeOf…Is** の式を使います。構文は次のとおりです。

```
TypeOf <objectexpression> Is <typename>
```

`typename` にインターフェイスの型を指定すると、オブジェクトがそのインターフェイス型を実装している場合に、**TypeOf…Is** の式が **True** を返します。`typename` にクラス型を指定すると、オブジェクトがそのクラスのインスタンスであるか、またはそのクラスの派生クラスのインスタンスである場合に **True** が返されます。次に例を示します。

VB

```
Dim x As System.Windows.Forms.Button
x = New System.Windows.Forms.Button()
If TypeOf x Is System.Windows.Forms.Control Then
    ' Insert code to run if x is of type System.Windows.Forms.Control.
End If
```

この例では、`x` の型が `Button` であり、`Control` から継承しているので、`TypeOf x Is Control` の式は **True** になります。

詳細については、「TypeOf 演算子 (Visual Basic)」を参照してください。

参照

関連項目

比較演算子 (Visual Basic)

概念

値の比較

Visual Basic における算術演算子

Visual Basic の連結演算子

Visual Basic の論理演算子とビット処理演算子

その他の技術情報

演算子 (Visual Basic)

方法：2 つのオブジェクトが等しいかどうかをテストする

オブジェクトを参照する 2 つの変数がある場合は、**Is** または **IsNot** 演算子、またはその両方を使用して、2 つの変数が同じインスタンスを参照しているかどうかを確認できます。

2 つのオブジェクトが等しいかどうかをテストするには

- 2 つの変数をオペランドとして、Is 演算子 (Visual Basic) または IsNot 演算子を使用します。

  VB

```
Public Sub processControl(ByVal f As System.Windows.Forms.Form, _
    ByVal c As System.Windows.Forms.Control)
    Dim active As System.Windows.Forms.Control = f.ActiveControl
    If (active IsNot Nothing) And (c Is active) Then
        ' Insert code to process control c
    End If
    Return
End Sub
```

2 つのオブジェクトが同じインスタンスを参照しているかどうかに応じて、特定のアクションを実行することがあります。前述の例では、`c` というコントロールをフォーム `f` 上のアクティブなコントロールと比較しています。アクティブなコントロールがない場合、またはアクティブなコントロールが `c` と同じインスタンスでない場合は、**If** ステートメントが失敗し、プロシージャはこれ以上の処理を行わずに制御を返します。

Is と **IsNot** のどちらを使用するかは、個人的な好みの問題です。一部の式では、どちらかの方が他方より読みやすいコードになることがあります。

参照

概念

Visual Basic における比較演算子

方法：文字列がパターンに一致するかを調べる

文字列型 (String) (Visual Basic) の式がパターンと一致しているかどうかを調べるには、Like 演算子を使います。

Like は 2 つのオペランドを受け取ります。左のオペランドは文字列式で、右のオペランドは一致するかどうかを調べるパターンを格納する文字列です。**Like** は文字列式がパターンと一致するかどうかを示す **Boolean** 値を返します。

文字列式の各文字が、特定の文字、ワイルドカード文字、文字リスト、または文字範囲と一致するかを調べることができます。パターン文字列に指定された位置が、文字列式の一致するかどうかを調べる文字の位置に対応します。

文字列式の文字が特定の文字と一致するかを調べるには

- 特定の文字をパターン文字列に直接配置します。その特定の文字は、角かっこ (**[ ]**) で囲む必要があります。詳細については、「Like 演算子」を参照してください。

  次の例では、`myString` が単一の文字 `H` で構成されるかどうかを調べます。

  VB

```
Dim sMatch As Boolean = myString Like "H"
```

文字列式の文字がワイルドカード文字と一致するか調べるには

- 疑問符 (**?**) をパターン文字列に入れます。この位置に、有効などの文字が入っていても、一致すると見なされます。

  次の例は、`myString` が単一の文字 `W` と、それに続く任意の 2 文字で構成されるかどうかを調べます。

  VB

```
Dim sMatch As Boolean = myString Like "W??"
```

文字列式の文字が文字のリストと一致するか調べるには

- パターン文字列内に角かっこ (**[ ]**) を置き、その内部に文字のリストを入れます。文字をコンマやその他の区切り記号で区切らないでください。リスト内のどの 1 文字が見つかっても、一致すると見なされます。

  次の例は、`myString` が有効な任意の文字と、それに続く 1 文字 (`A`、`C`、または `E` のいずれか) で構成されるかどうかを調べます。

  VB

```
Dim sMatch As Boolean = myString Like "?[ACE]"
```

  この一致では、大文字と小文字が区別されます。

文字列式の文字が文字の範囲と一致するか調べるには

- パターン文字列内に角かっこ (**[ ]**) を置き、その内部に範囲の開始文字と終了文字をハイフン (**-**) で区切って指定します。範囲内のどの 1 文字が見つかっても、一致すると見なされます。

  次の例は、`myString` が文字列 `num` と、それに続く 1 文字 (`i`、`j`、`k`、`l`、`m`、または `n` のいずれか) で構成されるかどうかを調べます。

  VB

```
Dim sMatch As Boolean = myString Like "num[i-m]"
```

  この一致では、大文字と小文字が区別されます。

空の文字列との一致

Like は `[]` を長さ 0 の文字列 (**""**) として扱います。`[]` を使うと文字列式全体が空かどうかを調べることができます。ただし、文字列式の特定の位置が空かどうかを調べることはできません。特定の位置が空である場合も含めて調べる必要がある場合は、**Like** を 2 度以上使用します。

文字列式の文字が文字のリストと一致するか、または空かを調べるには

1. **Like** 演算子を同じ文字列式で 2 度呼び出して、2 つの呼び出しを Or 演算子 (Visual Basic) または OrElse 演算子 でつなぎます。
2. 1 つ目の **Like** 句のパターン文字列に、文字リストを角かっこ (**[ ]**) で囲んで指定します。
3. 2 つ目の **Like** 句のパターン文字列では、調べる位置に文字を入れずに指定します。

   次の例では、7 桁の電話番号 `phoneNum` が、3 桁の数字の後に空白、ハイフン (-)、ピリオド (.) のいずれかが続くか、または文字が何も入らずに 4 桁の数字がそのまま続くかを調べます。

   VB

```
Dim sMatch As Boolean = _
  (phoneNum Like "###[ -.]####") OrElse (phoneNum Like "#######")
```

参照

関連項目

比較演算子 (Visual Basic)
Like 演算子
文字列型 (String) (Visual Basic)

概念

Visual Basic の演算子および式

Visual Basic の連結演算子

連結演算子は、複数の文字列を結合して 1 つの文字列にします。連結演算子には、**+** と **&** の 2 つがあります。どちらの演算子も、次の例に示すように基本的な連結演算を行います。

VB

```
Dim x As String = "Con" & "caten" & "ation"
Dim y As String = "Con" + "caten" + "ation"
' The preceding statements set both x and y to "Concatenation".
```

これらの演算子は、次のように **String** 型の変数を連結することもできます。

VB

```
Dim a As String = "abc"
Dim d As String = "def"
Dim z As String = a & d
Dim w As String = a + d
' The preceding statements set both z and w to "abcdef".
```

2 つの連結演算子の相違点

+ 演算子 (Visual Basic) の基本的な機能は、2 つの数値を加算することです。しかし、数値オペランドを文字列オペランドに連結することもできます。**+** 演算子は、一連の複雑な規則に従って、加算、連結、コンパイル エラーの発生、ランタイム InvalidCastException 例外のスローのどれを行うかを決定します。

& 演算子 (Visual Basic) は、**String** オペランドに対してだけ定義されており、**Option Strict** の設定にかかわらず、常にオペランドを **String** に拡大変換します。文字列の連結には **&** 演算子を使用することをお勧めします。この演算子は文字列だけに定義されているので、意図しない変換が発生する可能性を減らすことができます。

パフォーマンス : 文字列と StringBuilder

連結、削除、置換などの文字列操作を何度も行う場合は、System.Text 名前空間の StringBuilder クラスを使用するとパフォーマンスが向上します。**StringBuilder** オブジェクトを作成して初期化するには追加の命令が必要であり、最終的な値を **String** に変換するにはまた別の命令が必要になりますが、**StringBuilder** は高速に実行できるので、その点はカバーできます。

参照

関連項目

Option Strict ステートメント

概念

Visual Basic における文字列操作メソッドの種類
Visual Basic における算術演算子
Visual Basic における比較演算子
Visual Basic の論理演算子とビット処理演算子

Visual Basic の論理演算子とビット処理演算子

論理演算子は、**Boolean** 式を比較し、結果として **Boolean** 値を返します。**And**、**Or**、**AndAlso**、**OrElse**、および **Xor** 演算子は、オペランドを 2 つ取るので二項演算子と呼ばれます。一方、**Not** 演算子は、オペランドを 1 つ取るので単項演算子と呼ばれます。これらの演算子の中には、整数値に対してビットごとの論理演算を実行できるものもあります。

単項論理演算子

Not 演算子 (Visual Basic) は、**Boolean** 式の論理否定を求めます。これにより、オペランドの論理上の逆の値が得られます。式の評価が **True** の場合、**Not** は **False** を返します。式の評価が **False** の場合、**Not** は **True** を返します。次に例を示します。

VB

```
Dim x, y As Boolean
x = Not 23 > 14
y = Not 23 > 67
' The preceding statements set x to False and y to True.
```

二項論理演算子

And 演算子 (Visual Basic) は、2 つの **Boolean** 式の論理積を求めます。両方の式の評価が **True** の場合、**And** は **True** を返します。少なくとも一方の式の評価が **False** の場合、**And** は **False** を返します。

Or 演算子 (Visual Basic) は、2 つの **Boolean** 式の論理和または包含を実行します。いずれかの式の評価が **True** の場合、または両方の式の評価が **True** の場合は、**Or** は **True** を返します。どちらの式の評価も **True** でない場合は、**Or** は **False** を返します。

Xor 演算子 (Visual Basic) は、2 つの **Boolean** 式の排他的論理和を求めます。どちらか一方の式の評価が **True** で、もう一方はそうでない場合は、**Xor** は **True** を返します。両方の式の評価が **True** の場合、または両方の式の評価が **False** の場合は、**Xor** は **False** を返します。

And、**Or**、および **Xor** 演算子の例を次に示します。

VB

```
Dim a, b, c, d, e, f, g As Boolean

a = 23 > 14 And 11 > 8
b = 14 > 23 And 11 > 8
' The preceding statements set a to True and b to False.

c = 23 > 14 Or 8 > 11
d = 23 > 67 Or 8 > 11
' The preceding statements set c to True and d to False.

e = 23 > 67 Xor 11 > 8
f = 23 > 14 Xor 11 > 8
g = 14 > 23 Xor 8 > 11
' The preceding statements set e to True, f to False, and g to False.
```

論理演算子のショートサーキット

AndAlso 演算子は、2 つの **Boolean** 式の論理積を求めるという点では **And** 演算子によく似ています。両者の大きな違いは、**AndAlso** はショートサーキットの動作を示すという点にあります。**AndAlso** 式の 1 番目の式の評価が **False** の場合は、2 番目の式の評価がどうなっても最終的な結果は変わらないので、2 番目の式は評価されないまま、**AndAlso** は **False** を返します。

同様に、OrElse 演算子は、ショートサーキット評価で 2 つの **Boolean** 式の論理和を求めます。**OrElse** 式の 1 番目の式の評価が **True** の場合は、2 番目の式の評価がどうなっても最終的な結果は変わらないので、2 番目の式は評価されないまま、**OrElse** は **True** を返します。

ショートサーキットの長所と短所

ショートサーキットを使用すると、論理演算の最終的な結果に影響しない式は評価されないので、パフォーマンスが向上します。しかし、その式で追加の処理を実行していた場合は、ショートサーキットによりその処理がスキップされます。たとえば、その式に **Function** プロシージャの呼び出しが含まれている場合は、その式がショートサーキットされるとプロシージャが呼び出されないので、**Function** プロシージャ内の追加コードが実行されません。プログラムのロジックがその追加コードに依存している場合は、ショートサーキット演算子の使用を避けてください。

次の例は、**And** と **Or**、さらにそのショートサーキット版の演算子の違いを示しています。

VB

```
Dim amount As Integer = 12
Dim highestAllowed As Integer = 45
Dim grandTotal As Integer
```

VB

```
If amount > highestAllowed And checkIfValid(amount) Then
    ' The preceding statement calls checkIfValid().
End If
If amount > highestAllowed AndAlso checkIfValid(amount) Then
    ' The preceding statement does not call checkIfValid().
End If
If amount < highestAllowed Or checkIfValid(amount) Then
    ' The preceding statement calls checkIfValid().
End If
If amount < highestAllowed OrElse checkIfValid(amount) Then
    ' The preceding statement does not call checkIfValid().
End If
```

VB

```
Function checkIfValid(ByVal checkValue As Integer) As Boolean
    If checkValue > 15 Then
        MsgBox(CStr(checkValue) & " is not a valid value.")
        ' The MsgBox warning is not displayed if the call to
        ' checkIfValid() is part of a short-circuited expression.
        Return False
    Else
        grandTotal += checkValue
        ' The grandTotal value is not updated if the call to
        ' checkIfValid() is part of a short-circuited expression.
        Return True
    End If
End Function
```

上記の例では、`checkIfValid()` の呼び出しがショートサーキットされると、このプロシージャ内の重要なコードが実行されないという点に注意してください。1 つ目の **If** ステートメントは、`12 > 45` が **False** を返すときでも `checkIfValid()` を呼び出します。なぜなら、**And** はショートサーキットを行わないからです。2 つ目の **If** ステートメントは、`checkIfValid()` を呼び出しません。`12 > 45` が **False** を返すので、**AndAlso** が 2 つ目の式をショートサーキットするからです。3 つ目の **If** ステートメントは、`12 < 45` が **True** を返すときでも `checkIfValid()` を呼び出します。なぜなら、**Or** はショートサーキットを行わないからです。4 つ目の **If** ステートメントは、`checkIfValid()` を呼び出しません。`12 < 45` が **True** を返すので、**OrElse** が 2 つ目の式をショートサーキットするからです。

ビット処理演算

ビット処理演算子は、2 つの整数値をバイナリ (基数 2) の形式で評価します。対応する位置のビットを比較し、比較結果に基づいて値を割り当てます。次に **And** 演算子の例を示します。

```
Dim x As Integer
x = 3 And 5
```

この例では、`x` の値が 1 に設定されます。このような結果になる理由は次のとおりです。

- 2 つの値がバイナリとして扱われます。

  バイナリ形式の 3 = 011

  バイナリ形式の 5 = 101

- **And** 演算子は、このバイナリ表現を 1 桁 (1 ビット) ずつ比較します。ある位置のビットが両方とも 1 だった場合は、結果の同じ位置に 1

が配置されます。いずれかが 0 だった場合は、その位置に 0 が配置されます。したがって、上の例は次のように評価されます。

011 (バイナリ形式の 3)

101 (バイナリ形式の 5)

001 (バイナリ形式の結果)

- この結果は 10 進数として扱われます。001 という値は 1 のバイナリ表現なので、x = 1 となります。

ビットごとの **Or** 演算も同様です。ただし、その場合は、比較するビットの一方または両方が 1 のときに結果ビットが 1 になります。**Xor** では、比較するビットのいずれか一方だけが 1 のときのみ、結果ビットが 1 になります。**Not** のオペランドは 1 つだけであり、そのすべてのビット (符号ビットを含む) を反転させた値が結果になります。つまり、符号付き正数を与えると **Not** は必ず負数を返し、負数を与えると **Not** は必ず正数または 0 の値を返します。

AndAlso 演算子と **OrElse** 演算子はビット処理演算をサポートしません。

メモ：
ビット処理演算を実行できるのは整数型だけです。浮動小数点値に対してビット処理演算を実行するには、事前に整数型に変換する必要があります。

参照

関連項目

論理/ビット演算子

概念

Boolean 式
Visual Basic における算術演算子
Visual Basic における比較演算子
Visual Basic の連結演算子
演算子の効率のよい組み合わせ

演算子の効率のよい組み合わせ

複合式では、さまざまな演算子を使用できます。次に例を示します。

```
x = (45 * (y + z)) ^ (2 / 85) * 5 + z
```

演算子の優先順位の規則を完全に理解していないと、この例に示したような複合式は作成できません。詳細については、「Visual Basic における演算子の優先順位」を参照してください。

かっこを使った式

演算子の優先順位によって決まっている順序とは違う順序で演算を実行することもできます。例を次に示します。

```
x = z * y + 4
```

この例では、`z` を `y` と掛けた後、その結果を `4` に足しています。しかし、`y` と `4` を先に足してから、その結果を `z` に掛ける必要がある場合は、かっこを使って通常の演算子の優先順位をオーバーライドできます。式をかっこで囲むと、演算子の優先順位に関係なく、かっこで囲んだ式が最初に計算されます。上記の例で加算を先に実行するには、次のように式を書き直します。

```
x = z * (y + 4)
```

この例では、`y` と `4` を足した後で、その結果を `z` に掛けます。

かっこで入れ子にした式

かっこを複数のレベルで使って式を入れ子にすると、優先順位をさらにオーバーライドできます。最も深いかっこの中にある式が最初に評価され、2 番目に深いかっこ内の式が次に評価されます。この手順が、最も外側のかっこまで続けられ、最後にかっこの外部にある式が評価されます。次に例を示します。

```
x = (z * 4) ^ (y * (z + 2))
```

この例では、`z + 2` が最初に評価され、次に他のかっこ内の式が評価されます。指数演算は、通常ならば加算や乗算より優先順位は上ですが、他の式がすべてかっこで囲まれているため、この例では最後に計算されます。

参照

処理手順

方法 : 数値を計算する

関連項目

論理/ビット演算子
Visual Basic における演算子の優先順位

概念

Visual Basic における算術演算子
Visual Basic における比較演算子
Visual Basic の論理演算子とビット処理演算子
Boolean 式
値の比較

方法：数値を計算する

数式を使って数値を計算できます。数式とは、数値を表すリテラル、定数、変数、およびそれらの値に対して働く演算子を含む式です。

数値の計算

数値を計算するには

- 数値リテラル、定数、および変数を結合して数式にします。有効な数式の例を次に示します。

  ```
  93.217

  System.Math.PI

  counter

  4 * (67 + i)
  ```

  最初の 3 行は、順にリテラル、定数、および変数です。どの行も単独で有効な式です。最後の行は、1 つの変数と 2 つのリテラルを組み合わせた式です。

  数式は単独では完全な Visual Basic ステートメントにならないことに注意してください。式は完全なステートメントの一部として使用する必要があります。

数値を格納するには

- 数式で表される値を変数に代入するには、次のコード例のように代入ステートメントを使用します。

  VB

  ```
  Dim i As Integer = 2
  Dim j As Integer
  j = 4 * (67 + i)
  ```

  上記の例では、等値演算子 (=) の右側にある式の値が、演算子の左側にある変数 `j` に代入されるので、`j` の値は 276 となります。

  詳細については、「代入ステートメント」を参照してください。

複数の演算子

数式に複数の演算子が含まれている場合、演算子の計算順序は、演算子の優先規則によって決まっています。この順序を変更するには、上の例のように式をかっこで囲みます。かっこで囲まれた式が最初に計算されます。

演算子の通常の優先順位をオーバーライドするには

- 最初に実行する演算子をかっこで囲みます。次のコード例は、同じオペランドと演算子で結果が異なる 2 つの式を示しています。

  VB

  ```
  Dim i As Integer = 2
  Dim j, k As Integer
  j = 4 * (67 + i)
  k = 4 * 67 + i
  ```

  この例では、`j` を計算する際に、`(67 + i)` を囲むかっこにより通常の優先順位がオーバーライドされるため加算演算子 (+) が最初に実行され、`j` に代入される値は 276 (4 掛ける 69) になります。`k` を計算する際には、演算子は通常の優先順位 (+ の前に *) で実行され、`k` に代入される値は 270 (268 足す 2) になります。

  詳細については、「Visual Basic における演算子の優先順位」を参照してください。

参照

関連項目

Visual Basic における演算子の優先順位
算術演算子 (Visual Basic)

概念

Visual Basic の演算子および式
値の比較
代入ステートメント
演算子の効率のよい組み合わせ

値の比較

比較演算子を使うと、数値変数の値を比較する式を作成できます。比較演算子を使って作成した式は、比較の真偽に基づく **Boolean** 値を返します。このような式の例は次のようになります。

```
45 > 26
```

```
26 > 45
```

45 は 26 より大きいため、1 番目の式は **True** になります。26 は 45 より小さいため、2 番目の式は **False** になります。

この方法で、数式も比較できます。比較の対象となる式は、次のような複合式であってもかまいません。

```
x / 45 * (y +17) >= System.Math.Sqrt(z) / (p - (x * 16))
```

この複合式には、リテラル、変数、および関数呼び出しが含まれています。まず比較演算子の両側の式を計算し、その結果値を比較演算子 **>=** を使って比較します。左側の式の値が右側の式の値以上である場合は、上の式全体が **True** になり、それ以外の場合は **False** になります。

値を比較する式は、次のような **If...Then** 構築で最もよく使用されます。

VB

```
If x > 50 Then
    ' Insert code to run if x is greater than 50.
Else
    ' Insert code to run if x is less than or equal to 50.
End If
```

等号 (=) は、代入演算子としてだけでなく、比較演算子としても使用されます。比較演算子として使用された場合は、左側の値が右側の値と等しいかどうかを評価します。たとえば、次のように使います。

VB

```
If x = 50 Then
    ' Insert code to continue program.
End If
```

比較式は、この他にも、**Boolean** 値が必要な場合 (**If**、**While**、**Loop**、または **ElseIf** などのステートメントの中など) や、**Boolean** 変数に値を代入したり渡したりする場合などに使用できます。次の例では、比較式によって返される値が **Boolean** 変数に代入されています。

VB

```
Dim x As Boolean
x = 50 < 30
' The preceding statement assigns False to x.
```

参照

処理手順

方法 : 数値を計算する

関連項目

Visual Basic における演算子の優先順位

概念

Boolean 式

Visual Basic の演算子および式

Visual Basic における比較演算子

Boolean 式

Boolean 式とは、評価結果が Boolean データ型の値になる式です。**Boolean** 式にはいくつかの形式があります。最も単純な形式では、次のように、**Boolean** 変数の値を **Boolean** リテラルと直接比較します。

VB

```
If newCustomer = True Then
    ' Insert code to execute if newCustomer = True.
Else
    ' Insert code to execute if newCustomer = False.
End If
```

= 演算子の 2 つの意味

`newCustomer = True` という代入ステートメントは、前述の例に出てきた式と同じに見えますが、機能も使い方も異なります。前述の例では、`newCustomer = True` という式は Boolean 値を表しているため、= 記号は比較演算子として解釈されます。スタンドアロンのステートメントでは、= 記号は代入演算子として解釈され、右の値が左の変数に代入されます。次に例を示します。

VB

```
If newCustomer = True Then
    newCustomer = False
End If
```

詳細については、「値の比較」および「代入ステートメント」を参照してください。

比較演算子

=、<、>、<>、<=、>= などの比較演算子は、演算子の左側の式と右側の式を比較し、その結果を **True** または **False** として評価することで Boolean 式を形成します。次に例を示します。

```
42 < 81
```

42 は 81 より小さいので、この Boolean 式の評価は **True** になります。この種類の式の詳細については、「値の比較」を参照してください。

比較演算子と論理演算子の組み合わせ

論理演算子を使って比較式を組み合わせると、より複雑な Boolean 式を作成できます。次の例では、比較演算子と論理演算子を組み合わせて使用しています。

```
x > y And x < 1000
```

この式全体の値は、**And** 演算子の両側の式の値によって決まります。両方の式が **True** の場合は、式全体の評価も **True** になります。いずれかの式が **False** の場合は、式全体の評価も **False** になります。

演算子のショートサーキット

AndAlso と **OrElse** という論理演算子は、ショートサーキットと呼ばれる動作を示します。ショートサーキット演算子は、左のオペランドを最初に評価します。左のオペランドによって式全体の値が決まる場合は、右の式が評価されないまま、プログラムの実行が次に進みます。次に例を示します。

VB

```
If 45 < 12 AndAlso testFunction(3) = 81 Then
    ' Add code to continue execution.
End If
```

この例では、まず左の式 `45 < 12` が評価されます。左の式の評価は **False** になるので、この論理式全体の評価は **False** になります。したがって、右の式 `testFunction(3)` は評価されないまま、**If** ブロック内のコードの実行がスキップされます。この例では、左の式を評価しただけで式全体が false になることが決まるので、`testFunction()` は呼び出されません。

同様に、**OrElse** を使った論理式の左の式が **True** になる場合も、右の式は評価されないまま次のコード行が実行されます。これは、左の式を評価しただけで式全体が true になることが決まるからです。

非ショートサーキット演算子との比較

これに対して、**And** 演算子または **Or** 演算子を使用した場合は、論理演算子の両側が評価されます。次に例を示します。

VB

```
If 45 < 12 And testFunction(3) = 81 Then
    ' Add code to continue execution.
End If
```

この例では、左の式の評価が **False** になる場合でも、`testFunction()` が呼び出されます。

かっこを使った式

かっこを使うと、Boolean 式を評価する順序を制御できます。かっこで囲まれている式が最初に評価されます。複数のレベルで入れ子になっている場合は、最も内側の式から評価されます。かっこの中の式は、演算子の優先順位の規則に従って評価されます。詳細については、「Visual Basic における演算子の優先順位」を参照してください。

参照

関連項目

比較演算子 (Visual Basic)
Visual Basic における演算子の優先順位
ブール型 (Boolean) (Visual Basic)

概念

Visual Basic の論理演算子とビット処理演算子
値の比較
代入ステートメント
演算子の効率のよい組み合わせ

Visual Basic におけるステートメント

Visual Basic のステートメントは、キーワード、演算子、変数、定数、および式を含むことのできる完結した命令です。すべてのステートメントは 2 つのカテゴリに分類されます。宣言ステートメントは、変数、定数、またはプロシージャの名前を設定し、データ型も指定できます。実行可能なステートメントは、アクションを実行します。

このセクションの内容

ステートメントの概要

複数のステートメントを 1 行に記述する方法、1 つのステートメントを複数行にわたって継続する方法、コメントの追加、コンパイル エラーの確認など、ステートメントの高度な概要について説明します。

代入ステートメント

演算子の右辺の値を左辺の変数に代入する代入演算を実行するステートメントについて説明します。

Visual Basic の宣言ステートメント

プロシージャ、変数、配列、および定数に名前を付けて定義するステートメントについて説明します。

実行可能なステートメント

メソッドを実行することによってアクションを実行するステートメントについて説明します。実行可能なステートメントでは、複数のコード ブロックのループや分岐が可能で、多くの場合、算術演算子や条件演算子を使用します。

関連するセクション

代入演算子

=、*=、&= などの代入演算子について説明する言語リファレンス関連のトピックへのリンクが用意されています。

Visual Basic の演算子および式

要素と演算子を組み合わせて新しい値を作成する方法について説明します。

ステートメントの概要

Visual Basic のステートメントは、完結した命令です。ステートメントには、キーワード、演算子、変数、定数、および式を含めることができます。各ステートメントは、次のカテゴリのいずれかに属します。

- 宣言ステートメント。宣言ステートメントは、変数、定数、またはプロシージャの名前を指定します。一緒にデータ型を指定することもできます。
- 実行可能なステートメント。実行可能なステートメントは、アクションを実行します。メソッドまたは関数の呼び出しや、コード ブロックのループや分岐を行うことができます。値や式を変数や定数に代入する代入ステートメントも実行可能なステートメントに含まれます。

複数のステートメントを 1 行に記述する方法

各ステートメントをコロン (:) で区切ると、1 行に複数のステートメントを記述できます。次に例を示します。

VB

```
Dim sampleString As String = "Hello World" : MsgBox(sampleString)
```

複数のステートメントを 1 行に記述すると便利な場合もありますが、コードが読みにくくなって管理がたいへんになります。1 行に記述するステートメントは 1 つにすることをお勧めします。

複数行にまたがるステートメント

ステートメントは、1 行に収まるのが普通ですが、長すぎて収まらない場合には、行連結シーケンスを使って次の行に続けることができます。行連結シーケンスは、空白とそれに続くアンダースコア (_)、さらにそれに続く復帰で構成されています。次の例では、実行可能なステートメント **MsgBox** を 2 行にまたがって記述しています。

VB

```
Public Sub demoBox()
    Dim nameVar As String
    nameVar = "John"
    MsgBox("Hello " & nameVar & _
        ". How are you?")
End Sub
```

コメントの追加

ソース コードの意味は、記述した本人にとってさえ必ずしも自明であるとは限りません。このため、ほとんどのプログラマは、たくさんのコメントを挿入してコードに注釈を付けています。コード内のコメントでは、後でそのコードを読んだり作業したりする人のために、プロシージャまたは特定の命令についての説明を記述します。Visual Basic はコンパイル時にコメントを無視するので、コンパイルされたコードにコメントが影響を与えることはありません。

コメント行の先頭には、アポストロフィ (') または **REM** とそれに続く空白を指定します。コメントはコード内のどこにでも追加できますが、文字列の中には指定できません。ステートメントにコメントを付け加えるには、ステートメントの後にアポストロフィまたは **REM** を挿入し、続けてコメントを記述します。この他、独立した行としてコメントを挿入することもできます。具体的な例を次に示します。

VB

```
' This is a comment on a separate code line.
REM This is another comment on a separate code line.
x += a(i) * b(i) ' Add this amount to total.
MsgBox(statusMessage) REM Inform operator of status.
```

コンパイル エラーの確認

入力したステートメントに構文エラーがあると、コード行の下に青い波線が表示されます。場合によっては、エラー メッセージも一緒に表示されます。このような場合は、ステートメントの問題を見つけて修正する必要があります。エラーの内容は、タスク一覧や、エラー箇所の上にマウス ポインタを置くと表示されるエラー メッセージで確認できます。コード内の構文エラーをすべて修正しないと、プログラムは正しくコンパイルされません。

参照

処理手順

方法 : コード内でステートメントを分割および連結する
方法 : ステートメントにラベル付けする

概念

代入ステートメント
Visual Basic の宣言ステートメント
実行可能なステートメント

代入ステートメント

代入ステートメントは代入演算を行います。つまり、次の例のように、代入演算子 (=) の右側の値を左側の要素に格納します。

VB

```
v = 42
```

この例の代入ステートメントでは、42 というリテラル値を `v` 変数に格納しています。

使用できるプログラミング要素

代入演算子の左側に指定するプログラミング要素は、値を受け取って格納できるものにする必要があります。したがって、ReadOnly (Visual Basic) ではない変数またはプロパティを指定するか、配列要素を指定します。代入ステートメントのコンテキストでは、このような要素を左辺値、つまり "左側の値" と呼ぶことがあります。

代入演算子の右側の値は、式によって生成されます。この式は、リテラル、定数、変数、プロパティ、配列要素、他の式、関数呼び出しの任意の組み合わせによって構成されます。次に例を示します。

VB

```
x = y + z + findResult(3)
```

この例では、`y` 変数の値を `z` 変数の値に加算し、さらに `findResult` 関数の戻り値を加算しています。その後、この式の合計値を `x` 変数に格納しています。

代入ステートメントのデータ型

次の例に示すように、代入演算子では、数値だけでなく **String** 値も代入できます。

VB

```
Dim a, b As String
a = "String variable assignment"
b = "Con" & "cat" & "enation"
' The preceding statement assigns the value "Concatenation" to b.
```

また、次の例のように **Boolean** リテラルまたは **Boolean** 式を使用して、**Boolean** 値を代入することもできます。

VB

```
Dim r, s, t As Boolean
r = True
s = 45 > 1003
t = 45 > 1003 Or 45 > 17
' The preceding statements assign False to s and True to t.
```

同様に、**Char**、**Date**、または **Object** データ型のプログラミング要素に適切な値を代入することができます。また、オブジェクト インスタンスを、そのインスタンスの作成元になったクラスとして宣言された要素に代入することもできます。

複合代入ステートメント

複合代入ステートメントは、まず式の演算を実行してから、その結果をプログラミング要素に代入します。次の例では、このような演算子の 1 つである += を使用しています。この演算子は、左側の変数の値に右側の式の値を加算します。

VB

```
n += 1
```

この例では、`n` 変数の値に 1 を加算し、その新しい値を `n` 変数に格納します。これは、次のステートメントを短縮したものと考えることができま

す。

VB

```
n = n + 1
```

この種類の演算子を使うと、さまざまな複合代入演算を行うことができます。このような演算子の一覧と、それぞれの詳細については、「代入演算子」を参照してください。

既存の文字列の末尾に文字列を追加するには、次の例のように連結代入演算子 (**&=**) を使うと便利です。

VB

```
Dim q As String = "Sample "
q &= "String"
' q now contains "Sample String".
```

代入ステートメントでの型変換

変数、プロパティ、または配列要素に代入する値は、代入先の要素に適したデータ型にする必要があります。一般的には、代入先の要素と同じデータ型の値を生成するよう努力します。しかし、型によっては代入時に別の型に変換できるものもあります。

データ型の変換の詳細については、「Visual Basic における型変換」を参照してください。簡単に説明すると、Visual Basic は特定の型の値を、拡大変換した他の任意の型へと自動的に変換します。拡大変換は実行時に必ず成功する変換であり、データを一切失いません。たとえば、Visual Basic は必要に応じて **Integer** 値を **Double** に変換します。**Integer** は **Double** へと拡大変換されるからです。詳細については、「拡大変換と縮小変換」を参照してください。

縮小変換 (拡大変換を行わない変換) は、実行時に失敗したりデータを失ったりする可能性があります。縮小変換は型変換関数を使用して明示的に実行できます。また、**Option Strict Off** を設定するとすべての変換をコンパイラに暗黙的に実行させることができます。詳細については、「暗黙の型変換と明示的な型変換」を参照してください。

参照

関連項目

代入演算子

概念

ステートメントの概要
Visual Basic の宣言ステートメント
実行可能なステートメント
拡大変換と縮小変換

Visual Basic の宣言ステートメント

プロシージャ、変数、プロパティ、配列、および定数に名前を付けて定義するには、宣言ステートメントを使います。プログラミング要素を宣言するときには、データ型、アクセス レベル、スコープも宣言します。詳細については、「宣言された要素の特性」を参照してください。

次の例には、3 つの宣言が含まれています。

VB

```
Public Sub applyFormat()
    Const limit As Integer = 33
    Dim thisWidget As New widget
    ' Insert code to implement the procedure.
End Sub
```

最初の宣言は **Sub** ステートメントです。`applyFormat` という名前のプロシージャが、対応する **End Sub** ステートメントと一緒に宣言されています。また、`applyFormat` に **Public** が指定されています。これは、参照可能なすべてのコードに呼び出しが許可されることを意味します。

次に、**Const** ステートメントによって、定数 `limit` が宣言されています。この宣言では、データ型として整数 (**Integer**) が指定され、値として 33 が指定されています。

3 つ目の宣言である **Dim** ステートメントは、変数 `thisWidget` を宣言しています。データ型にはオブジェクト (具体的には `Widget` クラスから作成されるオブジェクト) が指定されています。変数はすべての基本データ型、または使用するアプリケーションに公開されているどのオブジェクト型でも宣言できます。

初期値

宣言ステートメントを含むコードが実行されると、Visual Basic は宣言された要素に必要なメモリを確保します。宣言された要素が値を保持する場合、Visual Basic はそのデータ型の既定値で要素を初期化します。詳細については、「Dim ステートメント (Visual Basic)」の "動作" を参照してください。

宣言時に変数に初期値を代入することもできます。次に例を示します。

VB

```
Dim m As Integer = 45
' The preceding declaration creates m and assigns the value 45 to it.
```

オブジェクト変数の場合は、次のように New (Visual Basic) キーワードを使うと、変数を宣言するときにそのクラスのインスタンスを明示的に作成できます。

VB

```
Dim f As New System.Windows.Forms.Form()
```

宣言ステートメントに指定した初期値は、実行がその宣言ステートメントに到達するまで変数に代入されません。それまでは、変数にそのデータ型の既定値が格納されます。

参照

処理手順

方法 : 定数を宣言する

概念

ステートメントの概要

代入ステートメント

実行可能なステートメント

実行可能なステートメント

実行可能なステートメントは、処理を実行します。プロシージャを呼び出したり、コード内の別の場所に分岐したり、いくつかのステートメントをループ実行したり、式を評価したりします。代入ステートメントは特殊な形の実行可能ステートメントです。

次の例では、**If...Then...Else** 制御構造を使用して、変数の値に基づいて異なるコード ブロックを実行します。各コード ブロックの中では、**For...Next** ループを指定の回数だけ実行します。

VB

```
Public Sub startWidget(ByVal aWidget As widget, _
    ByVal clockwise As Boolean, ByVal revolutions As Integer)
    Dim counter As Integer
    If clockwise = True Then
        For counter = 1 To revolutions
            aWidget.spinClockwise()
        Next counter
    Else
        For counter = 1 To revolutions
            aWidget.spinCounterClockwise()
        Next counter
    End If
End Sub
```

この例の **If** ステートメントは、`clockwise` パラメータの値をチェックします。値が **True** の場合は、`aWidget` の `spinClockwise` メソッドを呼び出します。値が **False** の場合は、`aWidget` の `spinCounterClockwise` メソッドを呼び出します。**If...Then...Else** 制御構造は **End If** で終わります。

各ブロック内の **For...Next** ループは、適切なメソッドを `revolutions` パラメータの値の回数だけ呼び出します。

参照

概念

ステートメントの概要
代入ステートメント
Visual Basic の宣言ステートメント

Visual Basic におけるプロシージャ

プロシージャは、宣言ステートメント (**Function**、**Sub**、**Operator**、**Get**、**Set**) とそれに対応する **End** 宣言に囲まれた Visual Basic ステートメントのブロックです。Visual Basic では、すべての実行可能なステートメントはプロシージャ内部に収める必要があります。

プロシージャの呼び出し

プロシージャは、コード内の他の場所で呼び出します。これをプロシージャ呼び出しと呼びます。実行が完了すると、そのプロシージャを呼び出したコードに制御を返します。このコードは、呼び出し元のコードと呼ばれます。呼び出し元のコードは、ステートメント、またはステートメント内の式であり、プロシージャを名前で指定して制御を渡します。

プロシージャからの復帰

プロシージャは、実行の終了時に制御を呼び出し元のコードに返します。これを行うには、Return ステートメント (Visual Basic)、プロシージャの適切な Exit ステートメント (Visual Basic) ステートメント、またはプロシージャの End (Visual Basic) ステートメントを使用します。呼び出し元のコードのプロシージャ呼び出しの直後に制御が返されます。

- **Return** ステートメントを使うと、制御はすぐに呼び出し元のコードに返されます。**Return** ステートメントの後にあるステートメントは実行されません。同じプロシージャ内に、複数の **Return** ステートメントを定義できます。
- **Exit Sub** ステートメントまたは **Exit Function** を使うと、制御はすぐに呼び出し元のコードに返されます。**Exit** ステートメントの後にあるステートメントは実行されません。同じプロシージャに複数の **Exit** ステートメントを定義できます。また、**Return** ステートメントと **Exit** ステートメントを同じプロシージャに混在させることもできます。
- プロシージャに **Return** ステートメントまたは **Exit** ステートメントがない場合、プロシージャ本体にある最後のステートメントの後に **End Sub** または **End Function**、**End Get**、または **End Set** が追加されます。**End** ステートメントは、すぐに制御を呼び出し元のコードに返します。プロシージャ内には **End** ステートメントを 1 つだけ指定できます。

パラメータと引数

ほとんどの場合、プロシージャは、呼び出されるたびに異なるデータを操作する必要があります。この情報は、プロシージャ呼び出しの一部としてプロシージャに渡します。プロシージャには 0 個以上のパラメータを定義します。各パラメータは、プロシージャに渡す値を表します。プロシージャ定義における各パラメータに対応するのが、プロシージャ呼び出しにおける引数です。引数は、プロシージャ呼び出しでそれに対応するパラメータに渡す値を表します。

プロシージャの種類

Visual Basic で使用されるプロシージャには、次の種類があります。

- Sub プロシージャ は、アクションを実行しますが、呼び出し元のコードに値を返すことはありません。
- イベント処理プロシージャは、**Sub** プロシージャの一種であり、ユーザーによる操作やプログラムの動作によって発生したイベントに応答して実行されます。
- Function プロシージャ は、呼び出し元のコードに値を返します。呼び出し元に戻る前に、他のアクションを実行できます。
- Property プロシージャ は、オブジェクトまたはモジュールのプロパティの値を取得および設定します。
- 演算子プロシージャ は、オペランドの 1 つまたは両方が新しく定義されたクラスまたは構造体である場合に、標準の動作を定義します。
- Visual Basic におけるジェネリック プロシージャ では、通常のパラメータに加え、1 つ以上の型パラメータが定義されるため、呼び出し元のコードでは呼び出しのたびに特定のデータ型を渡すことができます。

プロシージャと構造化コード

アプリケーションの実行可能コードのすべての行は、なんらかのプロシージャ (`Main`、`calculate`、`Button1_Click` など) の中に記述する必要があります。大きなプロシージャがある場合は、いくつかに分割すると、アプリケーションのコードが読みやすくなります。

プロシージャは、頻繁に使用される計算、テキストやコントロールの操作、データベースの操作など、繰り返し実行されるタスクや共有されているタスクを実行するのに便利です。プロシージャは、コード内のさまざまな場所から呼び出すことができるため、アプリケーションの基本的な構成要素として使用できます。

プロシージャでコードを構成すると、次の利点があります。

- プロシージャを使うと、プログラムを個別の論理単位に分割できます。プロシージャごとにデバッグする方が、プロシージャに分割されていないプログラム全体をデバッグするよりも簡単です。

- あるプログラム用に開発したプロシージャは、多くの場合、ほとんど変更せずに他のプログラムで使用できます。同じコードを何度も書かなくて済みます。

参照

処理手順

方法 : プロシージャを作成する

概念

Sub プロシージャ
Function プロシージャ
Property プロシージャ
演算子プロシージャ
プロシージャのパラメータと引数
再帰プロシージャ
プロシージャのオーバーロード
Visual Basic におけるジェネリック プロシージャ

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

方法：長いコードを複数のコードに分割する

Visual Basic 内のすべての実行可能なステートメントは、`Main` や `Form1_Load`、`calculateTotal` など、何らかのプロシージャの内部にあることが必要です。アプリケーション全体を単一の大きなプロシージャとして作成することも可能ですが、複数のプロシージャに分割した方が読みやすいコードになります。

構造化プログラミングは、アプリケーション内部のプログラムのモジュール性と階層構造を重視した手法です。Visual Basic で構造化プログラミングを実現するには、プロシージャを賢く利用し、アプリケーションを個々の論理単位に分割するという方法が最も簡単です。プログラム全体をデバッグするよりも、各単位を個別にデバッグする方が簡単です。また、あるプログラム用に開発したプロシージャは、多くの場合、ほとんど変更せずに他のプログラムでも使用できます。

大きなプロシージャを分割する

大きなプロシージャを独立したプロシージャに分割するには

1. コードの中から 1 つ以上の独立したセクションを特定します。
2. 独立した各セクションに対して、ソース コードを大きなプロシージャの外側に移動し、それを **Sub** ステートメントと **End Sub** ステートメントで囲みます。
3. 大きなプロシージャ内の、コード セクションが移動した場所に、**Sub** プロシージャを呼び出すステートメントを追加します。

大きなプロシージャに値を戻す

新しいプロシージャが大きなプロシージャに値を返すようにする場合は、**Function** プロシージャを定義します。

値を返すようにセクションを作成するには

1. 外側に移動したソース コードを、**Sub** ステートメントと **End Sub** ステートメントではなく、**Function** ステートメントと **End Function** ステートメントで囲みます。
2. **Function** プロシージャで、呼び出し元のコードに値を返せる状態になる場所に、**Return** ステートメントを追加します。
3. 大きなプロシージャ内の、コード セクションを外部に移動した場所で、呼び出し元のステートメントが戻り値を使って何らかの処理を実行していることを確認します。戻り値は変数に格納したり、式の中で使用したりできます。

参照

処理手順

方法 : プロシージャを作成する

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Function プロシージャ
Property プロシージャ
演算子プロシージャ
プロシージャのパラメータと引数
再帰プロシージャ
プロシージャのオーバーロード

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

方法：プロシージャを作成する

プロシージャは、開始宣言ステートメント (**Sub** または **Function**) と終了宣言ステートメント (**End Sub** または **End Function**) で囲みます。プロシージャのコードは、すべてこれらのステートメントの間に作成します。

プロシージャの内部に別のプロシージャを含めることはできません。つまり、開始ステートメントと終了ステートメントは他のプロシージャの外部に置く必要があります。

同じタスクを複数の場所で実行するコードがある場合は、そのタスクをプロシージャとして一度記述し、コード内の複数の場所からそれを呼び出します。

値を返さないプロシージャを作成するには

1. 他のプロシージャの外側で **Sub** ステートメントを記述し、続けて **End Sub** ステートメントを記述します。
2. **Sub** ステートメント内で、**Sub** キーワードの後ろにプロシージャ名を指定し、続けてパラメータ リストをかっこで囲んで記述します。
3. **Sub** ステートメントと **End Sub** ステートメントの間にプロシージャのコード ステートメントを記述します。

値を返すプロシージャを作成するには

1. 他のプロシージャの外側で **Function** ステートメントを記述し、続けて **End Function** ステートメントを記述します。
2. **Function** ステートメント内で、**Function** キーワードの後にプロシージャ名を定義し、パラメータリストをかっこで囲んで記述してから、戻り値のデータ型を指定する **As** 句を記述します。
3. **Function** ステートメントと **End Function** ステートメントの間にプロシージャのコード ステートメントを記述します。
4. 呼び出し元のコードに値を返すには、**Return** ステートメントを使用します。

作成したプロシージャを、ブロックが繰り返し出現する古いコードにリンクさせるには

1. 作成したコードが、古いコードからアクセス可能な場所にあることを確認します。
2. ブロックが繰り返し出現する古いコードで、同じタスクを実行しているステートメントを、**Sub** プロシージャまたは **Function** プロシージャを呼び出す単一のステートメントに置き換えます。
3. 値を返す **Function** プロシージャで置き換える場合は、呼び出しステートメントが戻り値を使ってアクション (変数に格納するなど) を実行していることを確認します。実行していなければ、値は失われます。

使用例

次の **Function** プロシージャは、直角三角形の最も長い辺 (斜辺) を、他の 2 つの辺の値を基に計算します。

VB

```
Function hypotenuse(ByVal side1 As Single, ByVal side2 As Single) As Single
    Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
End Function
```

参照

処理手順

方法 : 長いコードを複数のコードに分割する

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Function プロシージャ
Property プロシージャ
演算子プロシージャ
プロシージャのパラメータと引数
再帰プロシージャ
プロシージャのオーバーロード

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

Sub プロシージャ

Sub プロシージャは、**Sub** ステートメントと **End Sub** ステートメントで囲まれた一連の Visual Basic ステートメントです。**Sub** プロシージャはタスクを実行した後、呼び出し元のコードに制御を返しますが、呼び出し元のコードに値を返すことはありません。

プロシージャが呼び出されるたびに、**Sub** ステートメント後の最初の実行可能なステートメントから、最初の **End Sub**、**Exit Sub**、または **Return** ステートメントまでの一連のステートメントが実行されます。

Sub プロシージャは、モジュール、クラス、および構造体に定義できます。既定では **Public** に設定されます。つまり、プロシージャを定義したモジュール、クラス、または構造体へのアクセスが可能なアプリケーション内のどこからでも呼び出すことができます。メソッドとは、それが定義されているモジュール、クラス、または構造体の外部からアクセスすることのできる **Sub** プロシージャまたは **Function** プロシージャのことです。詳細については、「クラス メソッド」を参照してください。

また、**Sub** プロシージャは、呼び出し元のコードによって渡される定数、変数、式などの引数を受け取ることができます。

宣言の構文

Sub プロシージャを宣言する構文は次のとおりです。

```
[ modifiers ] Sub subname [( parameterlist )]
' Statements of the Sub procedure.
End Sub
```

modifiers にはアクセス レベルの他、オーバーロード、オーバーライド、共有、およびシャドウに関する情報を指定できます。詳細については、「Sub ステートメント (Visual Basic)」を参照してください。

パラメータ宣言

プロシージャの各パラメータは、パラメータ名とデータ型を指定するという、変数を宣言するのと似た方法で宣言します。また、引数渡しの方法およびパラメータを省略できるかどうか、パラメータ配列かどうかも指定できます。

パラメータ リストの各パラメータの構文は次のとおりです。

```
[Optional] [ByVal | ByRef] [ParamArray] parametername As datatype
```

パラメータを省略可能にする場合は、宣言内で既定値を指定する必要があります。既定値を指定する構文は次のとおりです。

```
Optional [ByVal | ByRef] parametername As datatype = defaultvalue
```

ローカル変数として扱われるパラメータ

制御がプロシージャに渡ると、各パラメータはローカル変数として扱われます。つまり、その有効期間はプロシージャの有効期間と同じで、スコープはプロシージャ全体になります。

呼び出し構文

Sub プロシージャは、スタンドアロンの呼び出しステートメントを使って明示的に呼び出す必要があります。式の中で名前を使って呼び出すことはできません。省略できないすべての引数の値を指定し、引数リストをかっこで囲む必要があります。指定する引数がない場合は、かっこを省略することもできます。**Call** キーワードは省略できますが、指定することが推奨されています。

Sub プロシージャを呼び出す構文は次のとおりです。

```
[Call] subname [( argumentlist )]
```

Sub メソッドは、それが定義されているクラスの外部から呼び出すことができます。まず、**New** キーワードを使用してクラスのインスタンスを作成するか、クラスのインスタンスを返すメソッドを呼び出す必要があります。詳細については、「方法 : New キーワードを使用する」を参照してください。その後、次の構文を使用することで、インスタンス オブジェクトの **Sub** メソッドを呼び出すことができます。

```
Object.methodname[(argumentlist)]
```

詳細については、「方法 : メソッドを使用してアクションを実行する」を参照してください。

宣言と呼び出しの例

次の **Sub** プロシージャは、アプリケーションが実行しようとしているタスクとタイム スタンプを表示します。アプリケーションでは、さまざまな場所から `tellOperator` を呼び出すことができるため、すべてのタスクの先頭にこのコードを複製する必要はありません。呼び出し時には、引数 `task` で文字列が渡されます。この文字列によって、開始されようとしているタスクが確認されます。

VB

```
Sub tellOperator(ByVal task As String)
    Dim stamp As Date
    stamp = TimeOfDay()
    MsgBox("Starting " & task & " at " & CStr(stamp))
End Sub
```

`tellOperator` を呼び出す一般的な例を次に示します。

VB

```
Call tellOperator("file update")
```

参照

処理手順

方法 : 値を返さないプロシージャを呼び出す

方法 : Visual Basic でイベント ハンドラを呼び出す

関連項目

Sub ステートメント (Visual Basic)

概念

Visual Basic におけるプロシージャ

Function プロシージャ

Property プロシージャ

演算子プロシージャ

プロシージャのパラメータと引数

方法：値を返さないプロシージャを呼び出す

Sub プロシージャは、呼び出し元のコードに値を返しません。このプロシージャは、スタンドアロンの呼び出しステートメントを使って明示的に呼び出す必要があります。式の中で名前を指定するだけでは、呼び出すことができません。

呼び出しステートメントには、オプションで **Call** キーワードを使用できます。このキーワードを使うとコードが読みやすくなるので、使用することをお勧めします。

Sub プロシージャを呼び出すには

1. **Call** キーワードに続けて、**Sub** プロシージャの名前を指定した **Call** ステートメントを記述します。
2. プロシージャ名の後に、引数リストを囲むためのかっこを記述します。引数がない場合は、かっこを省略することもできますが、かっこを記述するとコードが読みやすくなります。
3. かっこ内の引数リストに、引数をコンマで区切って指定します。引数は、**Sub** プロシージャのパラメータの定義と同じ順番で指定する必要があります。

   Visual Basic の AppActivate 関数を呼び出して、アプリケーション ウィンドウをアクティブにする例を次に示します。**AppActivate** はウィンドウ タイトルを唯一の引数として受け取ります。呼び出し元のコードに値は返されません。メモ帳のプロセスが実行中でない場合、この例は ArgumentException をスローします。アプリケーションの実行に **Shell** プロシージャを使用していますが、指定されたパスにアプリケーションの実行プログラムが保存されていない場合は、アプリケーションを実行することはできません。

   VB

```
Dim notepadID As Integer
' Activate a running Notepad process.
AppActivate("Untitled - Notepad")
' AppActivate can also use the return value of the Shell function.
' Shell runs a new instance of Notepad.
notepadID = Shell("C:\WINNT\NOTEPAD.EXE", AppWinStyle.NormalFocus)
' Activate the new instance of Notepad.
AppActivate(notepadID)
```

参照

処理手順

方法 : プロシージャを作成する
方法 : 値を返すプロシージャを呼び出す
方法 : Visual Basic でイベント ハンドラを呼び出す

関連項目

Sub ステートメント (Visual Basic)
Shell 関数
ArgumentException

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
プロシージャのパラメータと引数

方法：Visual Basic でイベント ハンドラを呼び出す

イベントとは、なんらかのプログラム コンポーネントによって認識される動作または事象で、たとえば、マウス クリックやクレジットの上限への到達などがあります。このようなイベントに応答するためのコードを記述できます。イベント ハンドラは、イベントに応答するために作成するコードです。

Visual Basic では、イベント ハンドラは **Sub** プロシージャとして作成します。ただし、他の **Sub** プロシージャと同じ方法で呼び出すのではなく、通常はイベントに対するハンドラとしてこのプロシージャを指定します。これには Handles 句と WithEvents 変数を使うか、または AddHandler ステートメント を使います。Visual Basic の既定の方法では、**Handles** 句を使ってイベント ハンドラを宣言します。統合開発環境 (IDE) でプログラムを作成する場合は、この方法でイベント ハンドラを記述することになります。**AddHandler** ステートメントは、実行時にイベントを動的に発生させるのに適しています。

イベントが発生すると、Visual Basic はイベント ハンドラのプロシージャを自動的に呼び出します。イベントにアクセスできるコードであればどこからでも、RaiseEvent ステートメント を実行してイベントを発生させることができます。

同じイベントに複数のイベント ハンドラを関連付けることも可能です。場合によっては、ハンドラをイベントから切り離すこともできます。詳細については、「Visual Basic におけるイベント」を参照してください。

Handles と WithEvents を使用してイベント ハンドラを呼び出すには

1. イベントが Event ステートメント を使って宣言されていることを確認します。
2. WithEvents キーワードを使用して、オブジェクト変数をモジュール レベルまたはクラス レベルで宣言します。この変数の **As** 句では、イベントを発生させるクラスを指定する必要があります。
3. イベントを処理する **Sub** プロシージャの宣言で、**WithEvents** 変数とイベント名を指定する Handles 句を追加します。
4. イベントが発生すると、Visual Basic はこの **Sub** プロシージャを自動的に呼び出します。イベントを発生させるには、**RaiseEvent** ステートメントを使用してコードを記述します。

   イベントの定義例と、イベントを発生させるクラスを参照する **WithEvents** 変数の定義例を次に示します。イベントを処理する **Sub** プロシージャでは **Handles** 句を使用して、クラスおよびプロシージャが処理するイベントが指定されています。

   VB

```
Public Class raisesEvent
    Public Event somethingHappened()
    Dim WithEvents happenObj As New raisesEvent
    Public Sub processHappen() Handles happenObj.somethingHappened
        ' Insert code to handle somethingHappened event.
    End Sub
End Class
```

AddHandler を使用してイベント ハンドラを呼び出すには

1. イベントが **Event** ステートメントを使って宣言されていることを確認します。
2. AddHandler ステートメントを実行して、イベントを処理する **Sub** プロシージャをイベントに動的に関連付けます。
3. イベントが発生すると、Visual Basic はこの **Sub** プロシージャを自動的に呼び出します。イベントを発生させるには、**RaiseEvent** ステートメントを使用してコードを記述します。

   次の例に定義された **Sub** プロシージャは、フォームの Closing イベントを処理します。次に、AddHandler ステートメントを使って `catchClose` プロシージャを **Closing** のイベント ハンドラとして関連付けています。

   VB

```
' Place these procedures inside a Form class definition.
Private Sub catchClose(ByVal sender As Object, ByVal e As System.ComponentModel.Cancel
EventArgs)
    ' Insert code to deal with impending closure of this form.
End Sub
Public Sub formOpened()
    AddHandler Me.Closing, AddressOf catchClose
```

```
    End Sub

```

RemoveHandler ステートメントを実行すると、イベント ハンドラをイベントから切り離すことができます。

参照

処理手順

方法 : プロシージャを作成する

方法 : 値を返さないプロシージャを呼び出す

関連項目

Sub ステートメント (Visual Basic)

AddressOf 演算子

概念

Visual Basic におけるプロシージャ

Sub プロシージャ

Function プロシージャ

Function プロシージャは、**Function** ステートメントと **End Function** ステートメントで囲まれた一連の Visual Basic ステートメントです。**Function** プロシージャはタスクを実行した後、呼び出し元のコードに制御を戻します。制御を戻すとき、値も呼び出し元のコードに返します。

プロシージャが呼び出されるたびに、**Function** ステートメント後の最初の実行可能なステートメントから、最初の **End Function**、**Exit Function**、または **Return** ステートメントまでの一連のステートメントが実行されます。

Function プロシージャはモジュール、クラス、または構造体に定義できます。既定で **Public** に設定されます。つまり、プロシージャを定義したモジュール、クラス、または構造体へのアクセスが可能なアプリケーション内のどこからでも呼び出すことができます。

また、**Function** プロシージャは、呼び出し元のコードによって渡される定数、変数、式などの引数を受け取ることができます。

宣言の構文

Function プロシージャを宣言する構文は次のとおりです。

```
[ modifiers ] Function functionname [( parameterlist )] As returntype
' Statements of the Function procedure.
End Function
```

modifiers にはアクセス レベルの他、オーバーロード、オーバーライド、共有、およびシャドウに関する情報を指定できます。詳細については、「Function ステートメント (Visual Basic)」を参照してください。

各パラメータは、Sub プロシージャ の場合と同じ方法で宣言します。

データ型

すべての **Function** プロシージャには、変数と同じようにデータ型があります。このデータ型は **Function** ステートメント内の **As** 句で指定し、関数が呼び出し元のコードに返す値のデータ型を決定します。たとえば、次のようになります。

```
Function yesterday() As Date
End Function
Function findSqrt(ByVal radicand As Single) As Single
End Function
```

詳細については、「Function ステートメント (Visual Basic)」の "指定項目" を参照してください。

戻り値

Function プロシージャによって呼び出し元のコードに返される値は、戻り値と呼ばれます。プロシージャが値を返すには、次の 2 とおりの方法があります。

- プロシージャのステートメントの自身の関数名に値を代入します。**Exit Function** ステートメントまたは **End Function** ステートメントが実行されるまで、呼び出し元のプログラムに制御が戻されません。次に例を示します。

  ```
  Function functionname [( parameterlist )] As returntype
  ' The following statement does not transfer control back to the calling code.
  functionname = expression
  ' When control returns to the calling code, expression is the return value.
  End Function
  ```

- **Return** ステートメントを使用して戻り値を指定します。この場合は、呼び出し元のプログラムに制御がすぐに戻されます。次に例を示します。

  ```
  Function functionname [( parameterlist )] As returntype
  ' The following statement immediately transfers control back to the calling code and returns the value
  of expression .
  Return expression
  ```

```
End Function
```

戻り値を関数名に代入する方法の利点は、**Exit Function** ステートメントまたは **End Function** ステートメントが実行されるまで、プロシージャから制御が戻されないという点にあります。これにより、一時的な値を代入して、後で必要に応じて調整できます。

詳細については、「Function ステートメント (Visual Basic)」の "戻り値" を参照してください。

配列を返す

Function プロシージャが配列データ型を返す場合、関数内で配列の要素に個別にアクセスできません。要素に個別にアクセスしようとすると、コンパイラによってプロシージャの再帰呼び出しと解釈されます。次に例を示します。

```
Function allOnes(ByVal n As Integer) As Integer()

For i As Integer = 1 To n - 1

' The following statement generates a COMPILER ERROR .

allOnes(i) = 1

Next i

' The following statement generates a COMPILER ERROR .

Return allOnes()

End Function
```

この例では、代入ステートメント `allOnes(i) = 1` が、コンパイラによってステートメントの左側にある `allOnes` の呼び出しと解釈されます。`Return allOnes()` は、引数を指定せずに呼び出したものと解釈されます。どちらのステートメントでも、コンパイル エラーが生成されます。

呼び出し構文

Function プロシージャを呼び出すには、代入ステートメントの右側または式の中に、名前と引数を指定します。省略できないすべての引数の値を指定し、引数リストをかっこで囲む必要があります。指定する引数がない場合は、かっこを省略することもできます。

Function プロシージャを呼び出す構文は次のとおりです。

lvalue = *functionname* [(*argumentlist*)]

If ((*functionname* [(*argumentlist*)] / 3) <= *expression*) Then

Function プロシージャを呼び出すときには、必ずしも戻り値を使用する必要はありません。戻り値を使用しない場合も、戻り値が無視されるだけで、プロシージャのアクションはすべて実行されます。MsgBox はたびたびこの方法で呼び出されます。

宣言と呼び出しの例

次の **Function** プロシージャは、直角三角形の最も長い辺 (斜辺) を、他の 2 つの辺の値を基に計算します。

VB

```
Function hypotenuse(ByVal side1 As Single, ByVal side2 As Single) As Single
    Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
End Function
```

`hypotenuse` を呼び出す一般的な例を次に示します。

VB

```
Dim testLength, testHypotenuse As Single
testHypotenuse = hypotenuse(testLength, 10.7)
```

Visual Basic のランタイム関数

Visual Basic のランタイムでは、Microsoft.VisualBasic 名前空間に置かれた多数の関数が利用できます。たとえば、Beep 関数、MsgBox 関数 (Visual Basic)、StrComp 関数 (Visual Basic) などの一般的な関数を使用できます。これらの関数は、独自に作成した **Function** プロシージャと同じように呼び出すことができます。

参照

処理手順

方法 : 値を返すプロシージャを作成する
方法 : プロシージャから値を返す

方法 : 値を返すプロシージャを呼び出す

関連項目

Function ステートメント (Visual Basic)

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Property プロシージャ
演算子プロシージャ
プロシージャのパラメータと引数

方法：値を返すプロシージャを作成する

呼び出し元のコードに体を返すには、**Function** プロシージャを使用します。

値を返すプロシージャを作成するには

1. その他のプロシージャの外部で、**Function** ステートメントと **End Function** ステートメントを使用します。
2. **Function** ステートメントの中では、**Function** キーワードの後にプロシージャの名前とかっこで囲んだパラメータ リストを指定します。
3. かっこの後には **As** 句で、返された値のデータ型を指定します。
4. プロシージャのコード ステートメントは、**Function** と **End Function** ステートメントの間に記述します。
5. **Return** ステートメントを使用して、呼び出し元のコードに値を返します。

   次の **Function** プロシージャは、直角三角形の最も長い辺 (斜辺) を他の 2 つの辺の値を基に計算します。

   VB

   ```
   Function hypotenuse(ByVal side1 As Single, ByVal side2 As Single) As Single
       Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
   End Function
   ```

   `hypotenuse` の一般的な呼び出しを次に示します。

   VB

   ```
   Dim testLength, testHypotenuse As Single
   testHypotenuse = hypotenuse(testLength, 10.7)
   ```

参照

処理手順

方法 : プロシージャから値を返す
方法 : 値を返すプロシージャを呼び出す

関連項目

Function ステートメント (Visual Basic)

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Property プロシージャ
演算子プロシージャ
プロシージャのパラメータと引数

方法：プロシージャから値を返す

Function プロシージャは、**Return** ステートメントを実行したとき、または **Exit Function** ステートメントか **End Function** ステートメントを実行したときに、呼び出し元のコードに値を返します。

Return ステートメントを使って値を返すには

1. プロシージャのタスクが完了するポイントに **Return** ステートメントを記述します。
2. キーワード **Return** に続けて、呼び出し元のコードに返す値を取得する式を定義します。
3. 同じプロシージャ内に、複数の **Return** ステートメントを定義できます。

   次の **Function** プロシージャは、直角三角形の最も長い辺 (斜辺) を計算し、結果を呼び出し元のコードに返します。

   VB

   ```
   Function hypotenuse(ByVal side1 As Single, ByVal side2 As Single) As Single
       Return Math.Sqrt((side1 ^ 2) + (side2 ^ 2))
   End Function
   ```

   `hypotenuse` を呼び出す一般的な例は次のようになります。このコードは戻り値を格納します。

   VB

   ```
   Dim testLength, testHypotenuse As Single
   testHypotenuse = hypotenuse(testLength, 10.7)
   ```

Exit Function または End Function を使用して値を返すには

1. **Function** プロシージャ内の少なくとも 1 か所で、値をプロシージャ名に代入します。
2. **Exit Function** ステートメントまたは **End Function** ステートメントを実行すると、Visual Basic は直前にプロシージャ名に代入された値を返します。
3. 同じプロシージャに複数の **Exit Function** ステートメントを定義できます。また、**Return** ステートメントと **Exit Function** ステートメントを同じプロシージャに混在させることもできます。
4. **Function** プロシージャには、**End Function** ステートメントを 1 つだけ定義できます。

   詳細および使用例については、「Function ステートメント (Visual Basic)」の「戻り値」を参照してください。

参照

処理手順

方法 : 値を返すプロシージャを作成する
方法 : 値を返すプロシージャを呼び出す

関連項目

Function ステートメント (Visual Basic)
Return ステートメント (Visual Basic)

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Property プロシージャ
演算子プロシージャ
プロシージャのパラメータと引数

方法：値を返すプロシージャを呼び出す

Function プロシージャは呼び出し元のコードに値を返します。これを呼び出すには、代入ステートメントの右側または式の中に、名前と引数を指定します。

Call キーワードを使って **Function** プロシージャを呼び出すこともできます。ただし、この方法ではプロシージャから返された値が無視されます。

Function プロシージャを式内で呼び出すには

1. 変数の場合と同様に、**Function** プロシージャの名前を使用します。**Function** プロシージャ呼び出しは、式内の、変数または定数を使用できる場所で使用できます。
2. プロシージャ名に、かっこで囲んだ引数リストを指定します。指定する引数がない場合は、かっこを省略することもできます。しかし、かっこを使用した方がコードが読みやすくなります。
3. かっこ内の引数リストに、引数をコンマで区切って指定します。引数は、**Function** プロシージャがパラメータを定義したのと同じ順序で渡します。

   また、1 つまたは複数の引数を名前で渡すこともできます。詳細については、「位置と名前による引数渡し」を参照してください。
4. プロシージャから返される値は、変数や定数の値と同じように式の一部となります。

代入ステートメントで Function プロシージャを呼び出すには

1. 代入ステートメントの等号 (=) 記号の後ろに **Function** プロシージャの名前を使用します。
2. プロシージャ名に、かっこで囲んだ引数リストを指定します。指定する引数がない場合は、かっこを省略することもできます。しかし、かっこを使用した方がコードが読みやすくなります。
3. かっこ内の引数リストに、引数をコンマで区切って指定します。名前で渡す場合を除き、引数は、**Function** プロシージャがパラメータを定義したのと同じ順序で渡します。
4. プロシージャから返される値は、代入ステートメントの左側の変数またはプロパティに格納されます。

使用例

次の例では、Visual Basic の Environ 関数を呼び出して、OS の環境変数の値を取得します。最初の行では **Environ** を式から呼び出し、2 行目ではこれを代入ステートメントから呼び出します。**Environ** は唯一の引数として変数の名前を取ります。変数の値は呼び出し元のコードに返されます。

VB

```
MsgBox("Value of PATH is " & Environ("PATH"))
Dim currentPath As String = Environ("PATH")
```

参照

処理手順

方法 : 値を返すプロシージャを作成する
方法 : プロシージャから値を返す
方法 : 値を返さないプロシージャを呼び出す

関連項目

Function ステートメント (Visual Basic)

概念

Function プロシージャ
プロシージャのパラメータと引数

Property プロシージャ

Property プロシージャは、モジュール、クラス、または構造体のカスタム プロパティを操作する一連の Visual Basic ステートメントです。Property プロシージャは、プロパティ アクセサとも呼ばれます。

Visual Basic には次の Property プロシージャが用意されています。

- **Get** プロシージャは、プロパティの値を返します。式の中でプロパティにアクセスするときに呼び出されます。
- **Set** プロシージャは、プロパティに値 (オブジェクト参照を含む) を設定します。プロパティに値を代入するときに呼び出されます。

Property プロシージャは、通常は **Get**ステートメントと **Set** ステートメントを使ってペアで定義しますが、プロパティが読み取り専用 (Get ステートメント) または書き込み専用 (Set ステートメント (Visual Basic)) の場合は、一方のプロシージャだけを定義できます。

プロパティは、クラス、構造体、およびモジュールで定義できます。プロパティは既定で **Public** になります。つまり、プロパティのコンテナにアクセスできるアプリケーションであればどこからでも、プロパティを呼び出すことができます。

プロパティと変数の比較については、「Visual Basic のプロパティと変数の違い」を参照してください。

宣言の構文

プロパティ自体は、Property ステートメントと **End Property** ステートメントに囲まれたコード ブロックで定義されます。このブロック内に、宣言ステートメント (**Get** または **Set**) とそれに対応する **End** の宣言で囲まれた内部ブロックとして、各 Property プロシージャを記述します。

プロパティとそのプロシージャを宣言する構文は、次のとおりです。

```
[Default] [modifiers] Property propertyname[(parameterlist)] As datatype

[accesslevel] Get

' Statements of the Get procedure.

' The following statement returns expression as the property's value.

Return expression

End Get

[accesslevel] Set[(ByVal newvalue As datatype)]

' Statements of the Set procedure.

' The following statement assigns newvalue as the property's value.

lvalue = newvalue

End Set

End Property
```

modifiers はアクセス レベルの他、オーバーロード、オーバーライド、共有、シャドウに関する情報、またはプロパティが読み取り専用か、書き込み専用かを指定します。**Get** プロシージャ、または **Set** プロシージャの *access level* には、プロパティそのものに指定されたアクセス レベルよりも制限の多いレベルをどれでも指定できます。詳細については、「Property ステートメント」を参照してください。

データ型

プロパティのデータ型とアクセス レベルは、Property プロシージャではなく、**Property** ステートメントに定義します。プロパティに定義できるデータ型は 1 つだけです。たとえば、**Decimal** 型で値を格納するが、**Double** 型の値を取得するようなプロパティを定義することはできません。

アクセス レベル

プロパティ自体に定義したアクセス レベルよりも制限の多いアクセスレベルを、Property プロシージャに定義できます。たとえば、**Public** プロパティを定義しておいて、**Private Set** プロシージャを定義するなどが可能です。このとき、一方の **Get** プロシージャは **Public** のままです。一方の Property プロシージャのアクセス レベルだけを変更できます。また、プロパティ自体のアクセス レベルよりも制限の多いレベルにしか変更できません。詳細については、「方法 : 複数のアクセス レベルを持つプロパティを宣言する」を参照してください。

パラメータ宣言

各パラメータの宣言は、Sub プロシージャの場合と同じ方法で行います。ただし、**ByVal** で渡す必要がある点が異なります。

パラメータ リストの各パラメータの構文は次のとおりです。

```
[Optional] ByVal [ParamArray] parametername As datatype
```

パラメータを省略可能にする場合は、宣言内で既定値を指定する必要があります。既定値を指定する構文は次のとおりです。

```
Optional ByVal parametername As datatype = defaultvalue
```

プロパティ値

Get プロシージャでは、戻り値がプロパティの値として呼び出し元の式に返されます。

Set プロシージャでは、新しいプロパティ値を **Set** ステートメントのプロパティに渡します。パラメータを明示的に宣言する場合は、プロパティと同じデータ型で宣言する必要があります。パラメータを宣言しなければ、コンパイラはプロパティに代入する新しい値を表すために、暗黙のパラメータ `Value` を使用します。

呼び出し構文

Property プロシージャは、プロパティを参照することによって暗黙的に呼び出されます。変数の名前を使用するのと同じように、プロパティの名前を使用します。ただし、省略できないすべての引数の値を指定し、引数のリストをかっこで囲む必要があります。指定する引数がない場合は、かっこを省略することもできます。

Set プロシージャを暗黙的に呼び出す構文は次のとおりです。

```
propertyname[(argumentlist)] = expression
```

Get プロシージャを暗黙的に呼び出す構文は次のとおりです。

```
lvalue = propertyname[(argumentlist)]
```

```
Do While (propertyname[(argumentlist)] > expression)
```

宣言と呼び出しの例

次のプロパティは、フル ネームをファースト ネームとラスト ネームの 2 つの部分に分けて格納します。呼び出しコードが `fullName` を読み込むと、**Get** プロシージャが 2 つの部分を組み合わせてフル ネームを返します。呼び出しコードが新しいフル ネームを代入すると、**Set** プロシージャはそれを 2 つの部分に分割します。フル ネームに空白が含まれない場合は、全体をファースト ネームとして格納します。

VB

```
Dim firstName, lastName As String
Property fullName() As String
    Get
      If lastName = "" Then
          Return firstName
      Else
          Return firstName & " " & lastName
      End If

    End Get
    Set(ByVal Value As String)
        Dim space As Integer = Value.IndexOf(" ")
        If space < 0 Then
            firstName = Value
            lastName = ""
        Else
            firstName = Value.Substring(0, space)
            lastName = Value.Substring(space + 1)
        End If
    End Set
End Property
```

`fullName` の Property プロシージャを呼び出す一般的な例は次のとおりです。

VB

```
fullName = "MyFirstName MyLastName"
MsgBox(fullName)
```

参照

処理手順

方法 : プロパティを作成する
方法 : プロパティ プロシージャを呼び出す
方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)

方法 : プロパティに値を格納する
方法 : プロパティから値を取得する

概念

Visual Basic におけるプロシージャ
Function プロシージャ
演算子プロシージャ
プロシージャのパラメータと引数
Visual Basic のプロパティと変数の違い

Visual Basic のプロパティと変数の違い

変数とプロパティは、いずれもアクセス可能な値を表します。しかし、格納と実装が異なります。

変数

変数は、メモリ内の場所に直接対応します。変数は、1 つの宣言ステートメントで定義します。変数は、プロシージャ内で定義され、そのプロシージャ内でのみ使用できるローカル変数と、モジュール、クラス、または構造体の中で定義され、プロシージャ内では定義されないメンバ変数のいずれかです。メンバ変数はフィールドとも呼ばれます。

プロパティ

プロパティは、モジュール、クラス、構造体で定義されるデータ要素です。プロパティは、**Property** と **End Property** ステートメントの間のコード ブロックで定義します。**Get** プロシージャか **Set** プロシージャ、またはこの両方を含むコード ブロックです。これらのプロシージャは、プロパティ プロシージャまたは プロパティ アクセサと呼ばれます。プロパティの値を取得および格納するだけでなく、アクセス カウンタの更新などのカスタム動作を実行することもできます。

相違点

次の表に、変数とプロパティの主な違いをまとめます。

異なる点	変数	プロパティ
宣言	1 つの宣言ステートメント	コード ブロック内の一連のステートメント
実装	1 つの格納場所	実行可能コード (プロパティ プロシージャ)
Storage	変数の値に直接関連付けられる	一般的に、プロパティに含まれるクラスまたはモジュールの外部からはアクセスできない内部ストレージを持つ プロパティの値は 1 つの格納された要素として存在する場合もそうでない場合もある [1]
実行可能コード	なし	最低でも 1 つのプロシージャを持つ必要がある
読み取り/書き込みアクセス	読み取り/書き込みまたは読み取り専用	読み取り/書き込み、読み取り専用、書き込み専用
カスタム動作 (値を受け取ったり返したりする操作以外)	不可能	プロパティの値の設定または取得操作の一部として実行可能

[1] 変数とは異なり、プロパティの値はストレージの単独のアイテムに直接対応しないこともあります。ストレージが利便性やセキュリティのために分割されたり、値が暗号化されたフォームに格納されたりすることがあります。このような場合、**Get** プロシージャが分割された部分をアセンブルしたり、格納された値を復号化したりし、**Set** プロシージャが新しい値を暗号化したり、値をストレージに分割したりします。プロパティの値は、時刻のようにすぐ変わるものである場合もあります。このような場合、プロパティにアクセスすると、**Get** プロシージャが実行時に値を計算します。

参照

処理手順

方法 : プロパティを作成する
方法 : 複数のアクセス レベルを持つプロパティを宣言する
方法 : プロパティ プロシージャを呼び出す
方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)
方法 : プロパティに値を格納する
方法 : プロパティから値を取得する

関連項目

Property ステートメント
Dim ステートメント (Visual Basic)

概念

Property プロシージャ
プロシージャのパラメータと引数

方法：プロパティを作成する

プロパティは **Property** ステートメントと **End Property** ステートメントで囲まれた部分に定義します。この内部に、**Get** プロシージャか **Set** プロシージャ、またはその両方を定義します。プロパティのコードは、すべてこれらのプロシージャの内部に作成します。

Get プロシージャはプロパティの値を取得し、**Set** プロシージャは値を格納します。プロパティのアクセスを読み取り/書き込みにする場合は、両方のプロシージャを定義する必要があります。読み取り専用のプロパティにするには **Get** だけを定義し、書き込み専用のプロパティにするには **Set** だけを定義します。

プロパティを作成するには

1. プロパティやプロシージャの外側で Property ステートメント を記述し、続けて **End Property** ステートメントを記述します。
2. プロパティがパラメータを受け取る場合は、**Property** キーワードの後ろにプロシージャ名を指定し、続けてパラメータ リストをかっこで囲んで記述します。
3. かっこの後ろに **As** 句を記述して、プロパティの値のデータ型を指定します。データ型は、書き込み専用のプロパティの場合でも指定する必要があります。
4. **Get** プロシージャと **Set** プロシージャを、必要に応じて追加します。方法については、以降を参照してください。

プロパティ値を取得する Get プロシージャを作成するには

1. **Property** ステートメントと **End Property** ステートメントの間に Get ステートメント を記述し、続けて **End Get** ステートメントを記述します。**Get** プロシージャにパラメータを定義する必要はありません。
2. **Get** ステートメントと **End Get** ステートメントの間に、プロパティ値を取得するコードを定義します。このコードには、プロパティ値を生成して返す処理の他に、計算を行ったりデータを操作したりする処理を含めることができます。
3. プロパティの値を呼び出し元のコードに返すには、**Return** ステートメントを使用します。

読み取り/書き込みプロパティ、および読み取り専用プロパティを作成する場合は、**Get** プロシージャを記述する必要があります。書き込み専用プロパティに **Get** プロシージャを定義することはできません。

プロパティ値を書き込む Set プロシージャを作成するには

1. **Property** ステートメントと **End Property** ステートメントの間に Set ステートメント (Visual Basic) を記述し、続けて **End Set** ステートメントを記述します。
2. **Set** ステートメント内で、キーワード **Set** の後ろにパラメータ リストをかっこで囲んで指定します。このパラメータ リストには、呼び出し元のコードから渡される値を受け取るための値パラメータを、最低でも 1 つ定義する必要があります。この値パラメータの既定の名前は `Value` ですが、必要であれば別の名前を使うこともできます。値パラメータのデータ型は、プロパティ自体と同じであることが必要です。
3. **Set** ステートメントと **End Set** ステートメントの間に、値をプロパティに格納するコードを定義します。このコードには、プロパティ値を検査して格納する処理の他に、計算を行ったりデータを操作したりする処理を含めることができます。
4. 呼び出し元のコードから渡された値を受け取るには、値パラメータを使用します。この値は、代入ステートメントに直接格納することも、格納する値を内部的に計算するために式の中で使用することもできます。

読み取り/書き込みプロパティ、および書き込み専用プロパティを作成する場合は、**Set** プロシージャを記述する必要があります。読み取り専用プロパティに **Set** プロシージャを定義することはできません。

使用例

次の例は、読み取り/書き込みプロパティを作成して、ファースト ネームとラスト ネームの 2 つの部分で構成されるフル ネームを格納します。呼び出しコードが `fullName` を読み込むと、**Get** プロシージャが 2 つの部分を組み合わせてフル ネームを返します。呼び出しコードが新しいフル ネームを代入すると、**Set** プロシージャはそれを 2 つの部分に分割します。フル ネームに空白が含まれない場合は、全体をファースト ネームとして格納します。

VB

```
Dim firstName, lastName As String
Property fullName() As String
    Get
      If lastName = "" Then
          Return firstName
```

```
        Else
            Return firstName & " " & lastName
        End If

    End Get
    Set(ByVal Value As String)
        Dim space As Integer = Value.IndexOf(" ")
        If space < 0 Then
            firstName = Value
            lastName = ""
        Else
            firstName = Value.Substring(0, space)
            lastName = Value.Substring(space + 1)
        End If
    End Set
End Property
```

`fullName` の Property プロシージャを呼び出す一般的な例は次のとおりです。最初の呼び出しでプロパティ値が設定され、2 番目の呼び出しで値が取得されます。

VB

```
fullName = "MyFirstName MyLastName"
MsgBox(fullName)
```

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する
方法 : プロパティ プロシージャを呼び出す
方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)
方法 : プロパティに値を格納する
方法 : プロパティから値を取得する

概念

Visual Basic におけるプロシージャ
Property プロシージャ
プロシージャのパラメータと引数
Visual Basic のプロパティと変数の違い

方法：複数のアクセス レベルを持つプロパティを宣言する

プロパティ上の **Get** および **Set** プロシージャに異なるアクセス レベルを割り当てるには、**Property** ステートメントでは制限のレベルを低くし、**Get** または **Set** ステートメントで制限のレベルを高くします。混合アクセス レベルを使用すると、コード内の特定の部分がプロパティの値を取得できるようにし、他の特定の部分でその値を変更できるようにすることができます。

アクセス レベルの詳細については、「Visual Basic でのアクセス レベル」を参照してください。

混合アクセス レベルでプロパティを宣言するには

1. 通常どおりプロパティを宣言し、**Property** ステートメントで制限の緩いアクセス レベル (**Public** など) を指定します。
2. **Get** または **Set** プロシージャは、より制限の厳しいアクセス レベル (**Friend** など) を指定して宣言します。
3. その他のプロパティ プロシージャではアクセス レベルを指定しません。**Property** ステートメントで宣言されたアクセス レベルが使用されます。アクセスを制限できるのは、プロパティ プロシージャのうち 1 つだけです。

   VB

   ```
   Public Class employee
       Private salaryValue As Double
       Protected Property salary() As Double
           Get
               Return salaryValue
           End Get
           Private Set(ByVal value As Double)
               salaryValue = value
           End Set
       End Property
   End Class
   ```

   上の例では、**Get** プロシージャはプロパティと同じ **Protected** アクセスを持ち、**Set** プロシージャは **Private** アクセスを持ちます。`employee` から派生したクラスは、`salary` の値を読み取ることができますが、この値を設定できるのは `employee` クラスのみです。

参照

処理手順

方法 : プロパティを作成する
方法 : プロパティ プロシージャを呼び出す
方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)
方法 : プロパティに値を格納する
方法 : プロパティから値を取得する

関連項目

Property ステートメント

概念

Visual Basic におけるプロシージャ
Property プロシージャ
プロシージャのパラメータと引数
Visual Basic のプロパティと変数の違い

方法：プロパティ プロシージャを呼び出す

プロパティ プロシージャは、プロパティに値を格納するか、プロパティの値を取得することによって呼び出します。プロパティのアクセス方法は、変数へのアクセス方法と同じです。

プロパティの **Set** プロシージャは値を格納し、**Get** プロシージャは値を取得します。しかし、これらのプロシージャを名前で明示的に呼び出すことはありません。変数の値を格納および取得するのと同じように、代入ステートメントまたは式の中でプロパティを使用します。Visual Basic がプロパティのプロシージャを呼び出します。

プロパティの Get プロシージャを呼び出すには

1. プロパティ名を、変数名と同様に式の中で使用します。プロパティは、変数や定数を使用できる場所で使用できます。

   または

   代入ステートメントの等号 (=) 記号の後ろにプロパティ名を使用します。

   次の例では、Now プロパティ の **Get** プロシージャを呼び出して、値を読み取ります。

   VB

```
Dim ThisMoment As Date
' The following statement calls the Get procedure of the Visual Basic Now property.
ThisMoment = Now
```

2. プロパティが引数を取る場合、プロパティ名に続けて、かっこで囲んだ引数のリストを指定します。指定する引数がない場合は、かっこを省略することもできます。

3. かっこ内の引数リストに、引数をコンマで区切って指定します。引数は、プロパティがパラメータを定義したのと同じ順序で渡します。

プロパティの値は、変数や定数を指定した場合と同じように式の中で使用されます。式の結果は、代入ステートメントの左側にある変数またはプロパティに格納されます。

プロパティの Set プロシージャを呼び出すには

1. 代入ステートメントの左側にプロパティ名を指定します。

   次の例では、TimeOfDay プロパティ の **Set** プロシージャを呼び出して、値を設定します。

   VB

```
' The following statement calls the Set procedure of the Visual Basic TimeOfDay proper
ty.
TimeOfDay = #12:00:00 PM#
```

2. プロパティが引数を取る場合、プロパティ名に続けて、かっこで囲んだ引数のリストを指定します。指定する引数がない場合は、かっこを省略することもできます。

3. かっこ内の引数リストに、引数をコンマで区切って指定します。引数は、プロパティがパラメータを定義したのと同じ順序で渡します。

代入ステートメントの右側で生成された値が、プロパティに格納されます。

参照

処理手順

方法 : プロパティを作成する
方法 : 複数のアクセス レベルを持つプロパティを宣言する
方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)
方法 : プロパティに値を格納する
方法 : プロパティから値を取得する

関連項目

Property ステートメント
Get ステートメント

Set ステートメント (Visual Basic)
概念
Property プロシージャ
プロシージャのパラメータと引数
Visual Basic のプロパティと変数の違い

方法：既定のプロパティを宣言する/呼び出す (Visual Basic)

既定のプロパティ は、コードで指定しなくてもアクセスできる、クラスまたは構造体のプロパティです。呼び出し元のコードでクラスや構造体を指定し、プロパティを指定しなければ、コンテキストにおいてプロパティへのアクセスが許可された場合に Visual Basic がそのクラスや構造体の既定のプロパティ (存在する場合) にアクセスを解決します。

クラスや構造体には、既定のプロパティを 1 つまで定義できます。ただし、既定のプロパティはオーバーロードできるので、複数の形式の定義が可能です。

詳細については、「既定のプロパティ」を参照してください。

既定のプロパティを宣言するには

1. 通常の方法でプロパティを宣言します。キーワード **Shared** または **Private** を指定しないでください。
2. 宣言に **Default** キーワードを含めます。
3. プロパティにパラメータを少なくとも 1 つ指定します。引数を 1 つも受け取らない既定のプロパティは定義できません。

   VB

   ```
   Default Property myProperty(ByVal index As Integer) As String
   ```

既定のプロパティを呼び出すには

1. 既定のプロパティを含むクラスや構造体の型を使って変数を宣言します。

   VB

   ```
   Dim x As New class1(3)
   ```

2. 通常はプロパティ名を指定する式に、変数名だけを指定します。

   VB

   ```
   MsgBox(x)
   ```

3. 変数名に続けて、引数をかっこで囲んで記述します。既定のプロパティは、少なくとも 1 つの引数を受け取る必要があります。

   VB

   ```
   MsgBox(x(1))
   ```

4. 既定のプロパティの値を取得するには、引数リストを指定した変数名を式の中に記述する、つまり代入ステートメントの等号 (=) 記号の後に記述します。

   VB

   ```
   MsgBox(x(1) & x(2) & x(3))
   ```

5. 既定のプロパティに値を設定するには、引数リストを指定した変数名を代入ステートメントの左側に記述します。

   VB

   ```
   x(1) = "Hello"
   x(2) = " "
   x(3) = "World"
   ```

6. 既定のプロパティ名は、他のプロパティにアクセスする場合と同じように、変数名を使っていつでも指定できます。

VB

```
x.myProperty(1) = "Hello"
x.myProperty(2) = " "
x.myProperty(3) = "World"
```

使用例

クラスに既定のプロパティを宣言するコード例を次に示します。

VB

```
Public Class class1
    Private myStrings() As String
    Sub New(ByVal size As Integer)
        ReDim myStrings(size)
    End Sub
    Default Property myProperty(ByVal index As Integer) As String
        Get
            ' The Get property procedure is called when the value
            ' of the property is retrieved.
            Return myStrings(index)
        End Get
        Set(ByVal Value As String)
            ' The Set property procedure is called when the value
            ' of the property is modified.
            ' The value to be assigned is passed in the argument
            ' to Set.
            myStrings(index) = Value
        End Set
    End Property
End Class
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

`class1` クラスの既定のプロパティ `myProperty` を呼び出す方法は、次の例のようになります。3 つの代入ステートメントが `myProperty` に値を格納し、MsgBox の呼び出しによって値が読み込まれます。

VB

```
Sub Test()
    Dim x As New class1(3)
    x(1) = "Hello"
    x(2) = " "
    x(3) = "World"
    MsgBox(x(1) & x(2) & x(3))
End Sub
```

既定のプロパティが最もよく使用されるのは、さまざまなコレクション クラスの Item プロパティ (Collection オブジェクト) です。

堅牢性の高いプログラム

既定のプロパティを使用すると、ソース コードに記述する文字の量が少し減りますが、コードの読みやすさが低下します。呼び出しコードからクラスや構造体が明確に区別できない場合にクラス名または構造体名を使って参照すると、その参照がクラスと構造体のどちらに対するものなのか、また既定のプロパティを参照しているかどうかもはっきりしません。このような場合、コンパイル エラーまたはランタイムの論理エラーが発生する可能性があります。

Option Strict ステートメントを使ってコンパイラの型チェックを常に **On** に設定しておくことで、既定のプロパティのエラーが発生する可能性をいくらか低下できます。

定義済みのクラスまたは構造体を使ってコードを作成する場合には、既定のプロパティが設定されているかどうかを調べ、設定されていればその名前を確認しておくことが必要です。

以上のような難点があるため、既定のプロパティは定義しないことをお勧めします。コードの読みやすさの点からも、すべてのプロパティを、既定のプロパティであっても明示的に参照することを心がけてください。

参照

処理手順

方法 : プロパティを作成する
方法 : 複数のアクセス レベルを持つプロパティを宣言する
方法 : プロパティ プロシージャを呼び出す
方法 : プロパティに値を格納する
方法 : プロパティから値を取得する

関連項目

Property ステートメント
Default (Visual Basic)

概念

Property プロシージャ
プロシージャのパラメータと引数
既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)
Visual Basic のプロパティと変数の違い

方法：プロパティに値を格納する

プロパティにデータを格納するには、代入ステートメントの左側にプロパティ名を置きます。

プロパティの **Set** プロシージャは値を格納しますが、このプロシージャを明示的に名前で呼び出すことはありません。プロパティは、変数と同じように使用します。Visual Basic はプロパティのプロシージャを呼び出します。

プロパティに値を格納するには

1. 代入ステートメントの左側にプロパティ名を指定します。

   次の例では、Visual Basic **TimeOfDay** プロパティの **Set** プロシージャを明示的に呼び出して、このプロパティを正午に設定します。

   VB

   ```
   ' The following statement calls the Set procedure of the Visual Basic TimeOfDay proper
   ty.
   TimeOfDay = #12:00:00 PM#
   ```

2. プロパティが引数を取る場合、プロパティ名に続けて、かっこで囲んだ引数のリストを指定します。指定する引数がない場合は、かっこを省略することもできます。

3. かっこ内の引数リストに、引数をコンマで区切って指定します。引数は、プロパティがパラメータを定義したのと同じ順序で渡します。

4. 代入ステートメントの右側で生成された値が、プロパティに格納されます。

参照

処理手順

方法 : プロパティを作成する
方法 : 複数のアクセス レベルを持つプロパティを宣言する
方法 : プロパティ プロシージャを呼び出す
方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)
方法 : プロパティから値を取得する

関連項目

Property ステートメント
TimeOfDay プロパティ

概念

Property プロシージャ
プロシージャのパラメータと引数
Visual Basic のプロパティと変数の違い

方法：プロパティから値を取得する

プロパティの値を取得するには、式にプロパティ名を含めます。

値を取得するのはプロパティの **Get** プロシージャですが、このプロシージャの名前を指定して呼び出すのではありません。プロパティは変数とまったく同じように使用し、Visual Basic がプロパティのプロシージャを呼び出します。

プロパティから値を取得するには

1. 式の中で、変数名を使用するのと同じようにプロパティ名を使用します。プロパティは、変数や定数を使用できるところであればどこでも使用できます。

   または

   代入ステートメントの等号 (=) 記号の後ろにプロパティ名を使用します。

   次の例は、Visual Basic の **Now** プロパティの値を読み込みます。**Get** プロシージャが暗黙的に呼び出されています。

   VB

   ```
   Dim ThisMoment As Date
   ' The following statement calls the Get procedure of the Visual Basic Now property.
   ThisMoment = Now
   ```

2. プロパティが引数を受け取る場合は、プロパティ名の後ろに引数リストを囲むためのかっこを記述します。引数がない場合は、かっこを省略することもできます。

3. かっこ内の引数リストに、引数をコンマで区切って指定します。引数は、プロパティのパラメータの定義と同じ順番で指定する必要があります。

プロパティの値は、変数や定数を指定した場合と同じように式の中で使用されます。または、代入ステートメントの左側にある変数またはプロパティに格納されます。

参照

処理手順

方法 : プロパティを作成する
方法 : 複数のアクセス レベルを持つプロパティを宣言する
方法 : プロパティ プロシージャを呼び出す
方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)
方法 : プロパティに値を格納する

関連項目

Property ステートメント

概念

Visual Basic におけるプロシージャ
Property プロシージャ
プロシージャのパラメータと引数
Visual Basic のプロパティと変数の違い

演算子プロシージャ

演算子プロシージャは、定義したクラスまたは構造体で、標準の演算子 (*、<>、**And** など) の動作を定義する一連の Visual Basic ステートメントです。これは、演算子のオーバーロードともいいます。

演算子プロシージャを定義する場合

クラスまたは構造体を定義する場合、そのクラスまたは構造体の型で変数を宣言できます。このような変数は、式の一部として演算にかかわる必要のある場合があります。このためには、この変数は演算子のオペランドである必要があります。

Visual Basic は、基本データ型でのみ演算子を定義します。演算子の動作は、オペランドのいずれか、または両方が、そのクラスまたは構造体の型である場合に定義できます。

詳細については、「Operator ステートメント」を参照してください。

演算子プロシージャの種類

演算子プロシージャは次のいずれかの種類です。

- 引数がクラスまたは構造体の型である、単項演算子の定義。
- 少なくとも 1 つの引数がクラスまたは構造体の型である、二項演算子の定義。
- 引数がクラスまたは構造体の型である、変換演算子の定義。
- クラスまたは構造体の型を返す変換演算子の定義。

変換演算子は常に単項であり、定義する演算子として常に **CType** を使用します。

宣言の構文

演算子プロシージャを宣言する構文は次のとおりです。

```
Public Shared [Widening | Narrowing] Operator operatorsymbol ( operand1 [, operand2 ]) As datatype
' Statements of the operator procedure.
End Operator
```

Widening または **Narrowing** キーワードは、型変換演算子にのみ使用します。型変換演算子の演算子記号は常に CType 関数です。

二項演算子を定義するには 2 つのオペランドを宣言し、型変換演算子を含む単項演算子を定義するには 1 つのオペランドを宣言します。すべてのオペランドは **ByVal** で宣言する必要があります。

各オペランドの宣言は、Sub プロシージャにパラメータを宣言する場合と同じです。

データ型

定義済みのクラスまたは構造体で演算子を定義するので、最低でも 1 つのオペランドはそのクラスまたは構造体の型である必要があります。型変換演算子では、オペランドまたは戻り値の型が、クラスまたは構造体のデータ型である必要があります。

詳細については、「Operator ステートメント」を参照してください。

呼び出し構文

式の中で演算子記号を使用することで、暗黙的に演算子プロシージャを呼び出すことができます。オペランドは、定義済みの演算子の場合と同じように指定します。

演算子プロシージャを暗黙的に呼び出す構文は次のとおりです。

```
Dim testStruct As structurename
Dim testNewStruct As structurename = testStruct operatorsymbol 10
```

宣言と呼び出しの説明

次の構造体は、符号付き 128 ビットの整数値を、上位と下位の構成部分として格納します。+ 演算子を定義し、2 つの `veryLong` 値を追加し、最終的な `veryLong` 値を生成します。

VB

```
Public Structure veryLong
```

```
    Dim highOrder As Long
    Dim lowOrder As Long
    Public Shared Operator +(ByVal v As veryLong, _
                             ByVal w As veryLong) As veryLong
        Dim sum As New veryLong
        sum = v
        Try
            sum.lowOrder += w.lowOrder
        Catch ex As System.OverflowException
            sum.lowOrder -= (Long.MaxValue - w.lowOrder + 1)
            sum.highOrder += 1
        End Try
        sum.highOrder += w.highOrder
        Return sum
    End Operator
End Structure
```

次の例では、 `veryLong` で定義された + 演算子の一般的な呼び出しを示します。

VB

```
Dim v1, v2, v3 As veryLong
v1.highOrder = 1
v1.lowOrder = Long.MaxValue
v2.highOrder = 0
v2.lowOrder = 4
v3 = v1 + v2
```

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する
方法 : 演算子プロシージャを呼び出す
方法 : 演算子を定義するクラスを使用する

関連項目

Operator ステートメント

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Function プロシージャ
Property プロシージャ
プロシージャのパラメータと引数

方法：演算子を定義する

クラスまたは構造体を定義してあれば、オペランドの一方または両方が、このクラスまたは構造体のデータ型であるときの、標準の演算子 (*、<>、**And** など) の動作を定義できます。

標準の演算子を、クラスまたは構造体の演算子プロシージャとして定義します。すべての演算子プロシージャは **Public Shared** である必要があります。

クラスまたは構造体で演算子を定義することを、演算子をオーバーロードするといいます。

使用例

次の例では、構造体 `height` の + 演算子を定義しています。この構造体では、フィート単位とインチ単位の高さを扱います。1 インチは 2.54 センチメートル、1 フィートは 12 インチです。値の正規化 (12.0 以下のインチ) を確実にするために、コンストラクタは モジュロ 12 の演算を実行します。+ 演算子は、コンストラクタを使って正規化された値を生成します。

VB

```
Public Shadows Structure height
    ' Need Shadows because System.Windows.Forms.Form also defines property Height.
    Private feet As Integer
    Private inches As Double
    Public Sub New(ByVal f As Integer, ByVal i As Double)
        Me.feet = f + (CInt(i) \ 12)
        Me.inches = i Mod 12.0
    End Sub
    Public Overloads Function ToString() As String
        Return Me.feet & "' " & Me.inches & """"
    End Function
    Public Shared Operator +(ByVal h1 As height, ByVal h2 As height) _
        As height
        Return New height(h1.feet + h2.feet, h1.inches + h2.inches)
    End Operator
End Structure
```

`height` 構造体をテストするためには、次のコードを使用してください。

VB

```
Public Sub consumeHeight()
    Dim p1 As New height(3, 10)
    Dim p2 As New height(4, 8)
    Dim p3 As height = p1 + p2
    Dim s As String = p1.ToString() & " + " & p2.ToString() _
        & " = " & p3.ToString() & " (= 8' 6"" ?)"
    Dim p4 As New height(2, 14)
    s &= vbCrLf & "2' 14"" = " & p4.ToString() & " (= 3' 2"" ?)"
    Dim p5 As New height(4, 24)
    s &= vbCrLf & "4' 24"" = " & p5.ToString() & " (= 6' 0"" ?)"
    MsgBox(s)
End Sub
```

参照

処理手順

方法 : 変換演算子を定義する
方法 : 演算子プロシージャを呼び出す
方法 : 演算子を定義するクラスを使用する
方法 : 構造体を宣言する

関連項目

Operator ステートメント
Structure ステートメント

概念

演算子プロシージャ

方法：変換演算子を定義する

定義されたクラスまたは構造体がある場合、その型と別のデータ型 (**Integer**、**Double**、または **String**) の間の型変換演算子を定義できます。

型変換はクラスまたは構造体の内部に CType 関数 プロシージャとして定義します。すべての変換プロシージャは **Public Shared** であることが必要です。また、それぞれに Widening または Narrowing を指定する必要があります。

クラスや構造体に演算子を定義することを、演算子をオーバーロードするともいいます。

使用例

構造体 `digit` と **Byte** の間の変換演算子を定義する例は次のようになります。

VB

```
Public Structure digit
Private dig As Byte
    Public Sub New(ByVal b As Byte)
        If (b < 0 OrElse b > 9) Then Throw New _
            System.ArgumentException("Argument outside range for Byte")
        Me.dig = b
    End Sub
    Public Shared Widening Operator CType(ByVal d As digit) As Byte
        Return d.dig
    End Operator
    Public Shared Narrowing Operator CType(ByVal b As Byte) As digit
        Return New digit(b)
    End Operator
End Structure
```

構造体 `digit` を、次のコードを使用してテストしてください。

VB

```
Public Sub consumeDigit()
    Dim d1 As New digit(4)
    Dim d2 As New digit(7)
    Dim d3 As digit = CType(CByte(3), digit)
    Dim s As String = "Initial 4 generates " & CStr(CType(d1, Byte)) _
        & vbCrLf & "Initial 7 generates " & CStr(CType(d2, Byte)) _
        & vbCrLf & "Converted 3 generates " & CStr(CType(d3, Byte))
    Try
        Dim d4 As digit
        d4 = CType(CType(d1, Byte) + CType(d2, Byte), digit)
    Catch e4 As System.Exception
        s &= vbCrLf & "4 + 7 generates " & """ & e4.Message & """
    End Try
    Try
        Dim d5 As digit = CType(CByte(10), digit)
    Catch e5 As System.Exception
        s &= vbCrLf & "Initial 10 generates " & """ & e5.Message & """
    End Try
    MsgBox(s)
End Sub
```

参照

処理手順

方法 : 演算子を定義する
方法 : 演算子プロシージャを呼び出す
方法 : 演算子を定義するクラスを使用する
方法 : 構造体を宣言する

関連項目

Operator ステートメント
Structure ステートメント
概念
演算子プロシージャ
暗黙の型変換と明示的な型変換
拡大変換と縮小変換

方法：演算子プロシージャを呼び出す

演算子プロシージャを呼び出すには、式の中で演算子記号を使用します。変換演算子のプロシージャを呼び出す場合は、CType 関数を呼び出して、あるデータ型の値を別のデータ型に変換します。

演算子プロシージャを明示的に呼び出すことはありません。演算子を通常どおりに使用するのと同じ方法で、演算子や **CType** 関数を代入ステートメントまたは式に定義すると、Visual Basic によって演算子プロシージャが呼び出されます。

クラスや構造体に演算子を定義することを、演算子をオーバーロードするともいいます。

演算子プロシージャを呼び出すには

1. 通常の方法で、演算子記号を式の中に記述します。
2. 演算子のオペランドのデータ型が適切であること、またオペランドの順序が正しいことを確認します。
3. 演算子によって、式の値が適切に処理されます。

変換演算子プロシージャを呼び出す

1. 式の中で **CType** を使用します。
2. 変換処理のオペランドのデータ型が適切であること、またオペランドの順序が正しいことを確認します。
3. **CType** が変換演算子プロシージャを呼び出し、変換された値を返します。

使用例

次の例は、2 つの TimeSpan 構造体を作成し、この 2 つを足し合わせて、結果を 3 つ目の **TimeSpan** 構造体に格納します。**TimeSpan** 構造体には演算子プロシージャが定義され、複数の標準演算子をオーバーロードしています。

VB

```
Dim firstSpan As New TimeSpan(3, 30, 0)
Dim secondSpan As New TimeSpan(1, 30, 30)
Dim combinedSpan As TimeSpan = firstSpan + secondSpan
Dim s As String = firstSpan.ToString() & _
          " + " & secondSpan.ToString() & _
          " = " & combinedSpan.ToString()
MsgBox(s)
```

上記の例では **TimeSpan** で標準の + 演算子がオーバーロードされているため、`combinedSpan` の値を計算するときに演算子プロシージャが呼び出されます。

変換演算子プロシージャを呼び出す例については、「方法 : 演算子を定義するクラスを使用する」を参照してください。

コードのコンパイル方法

使用するクラスまたは構造体に、使用する演算子が定義されていることを確認してください。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する
方法 : 構造体を宣言する

関連項目

Operator ステートメント
Widening
Narrowing
Structure ステートメント

概念

演算子プロシージャ
暗黙の型変換と明示的な型変換
拡大変換と縮小変換

方法：演算子を定義するクラスを使用する

独自の演算子を定義するクラスまたは構造体を使用している場合、Visual Basic でこれらの演算子にアクセスできます。

クラスまたは構造体で演算子を定義することを、演算子をオーバーロードするといいます。

使用例

次の例では、SQL 文字列と Visual Basic 文字列を相互に変換する変換演算子 (CType 関数) を定義する SQL 構造体 SqlString にアクセスします。SQL 文字列を Visual Basic 文字列に変換するには **CType(***SQL string expression*, **String)** を使用し、逆の変換を行うには **CType(***Visual Basic string expression*, **SqlString)** を使用します。

VB

```
' Insert the following line at the beginning of your source file.
Imports System.Data.SqlTypes
```

VB

```
Public Sub setJobString(ByVal g As Integer)
    Dim title As String
    Dim jobTitle As System.Data.SqlTypes.SqlString
    Select Case g
        Case 1
            title = "President"
        Case 2
            title = "Vice President"
        Case 3
            title = "Director"
        Case 4
            title = "Manager"
        Case Else
            title = "Worker"
    End Select
    jobTitle = CType(title, SqlString)
    MsgBox("Group " & CStr(g) & " generates title """ _
        & CType(jobTitle, String) & """")
End Sub
```

SqlString 構造体は、**String** から **SqlString** へ、そして **SqlString** から **String** への変換を行う変換演算子 (CType 関数) を定義します。`title` を `jobTitle` に代入するステートメントは最初の演算子を使用し、MsgBox 関数 (Visual Basic) の呼び出しは 2 つ目の演算子を使用します。

コードのコンパイル方法

使用しているクラスまたは構造体が、必要な演算子を定義していることを確認してください。クラスまたは構造体が、オーバーロード可能なすべての演算子を定義しているとは限りません。利用可能な演算子の一覧については、「Operator ステートメント」を参照してください。

ソース ファイルの先頭に、SQL 文字列用の適切な **Imports** ステートメントを含めます (この場合は、System.Data.SqlTypes)。

プロジェクトには、System.Data および System.XML への参照が必要です。追加する方法については、「方法 : Visual Studio (C#、J#) のリファレンスを追加および削除する」を参照してください。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する
方法 : 演算子プロシージャを呼び出す
方法 : 構造体を宣言する

関連項目

Widening
Narrowing
Structure ステートメント

概念

演算子プロシージャ
暗黙の型変換と明示的な型変換
拡大変換と縮小変換

プロシージャのパラメータと引数

プロシージャは、ほとんどの場合、呼び出されたときの状況に関する情報を必要とします。繰り返し実行されるタスクや共有されているタスクを実行するプロシージャは、呼び出されるたびに異なる情報を使用します。この情報は、プロシージャを呼び出すときに渡される変数、定数、および式から構成されています。

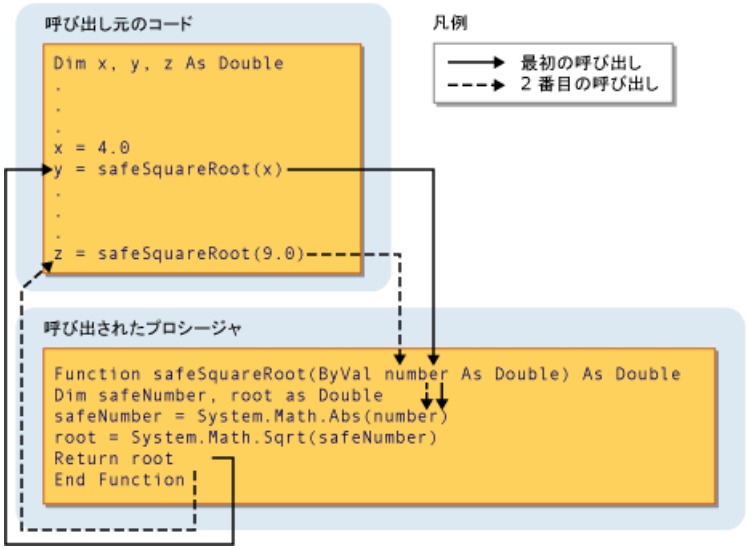
パラメータは、プロシージャが呼び出されるときに期待する値を表します。パラメータは、プロシージャの宣言で定義されます。

プロシージャは、パラメータなしでも、1 つのパラメータでも、複数のパラメータでも定義できます。プロシージャの定義のうち、パラメータを指定する部分をパラメータ リストと呼びます。

引数は、プロシージャを呼び出すときに、プロシージャのパラメータに渡す値を表します。呼び出し元のコードは、プロシージャを呼び出すときに引数を渡します。プロシージャ呼び出しのうち、引数を指定する部分を引数リストと呼びます。

次の図は、プロシージャ `safeSquareRoot` を 2 つの場所から呼び出すコードを示したものです。最初の呼び出しでは、変数 `x` の値 (4.0) をパラメータ `number` に渡し、`root` の戻り値 (2.0) が変数 `y` に代入されます。2 番目の呼び出しでは、リテラル値 9.0 を `number` に渡し、戻り値 (3.0) を変数 `z` に代入します。

パラメータに引数を渡す



詳細については、「パラメータと引数の違い」を参照してください。

パラメータのデータ型

パラメータのデータ型を宣言するには、パラメータの宣言の中で **As** 句を使用します。たとえば、次の関数は、文字列型と整数型の引数を受け取ります。

VB

```
Function appointment(ByVal day As String, ByVal hour As Integer) As String
    ' Insert code to return any appointment for the given day and time.
    Return "appointment"
End Function
```

型チェック スイッチ (Option Strict ステートメント) が **Off,** の場合、**As** 句は省略可能です。ただし、いずれかのパラメータがこれを使用している場合は、すべてのパラメータで使用する必要があります。型チェックが **On** になっている場合は、プロシージャのすべての引数に **As** 句が必要です。

String パラメータに **Byte** を渡すなど、呼び出し元のコードが、対応するパラメータとは異なるデータ型の引数を渡す場合、次のいずれかの作業が必要です。

- パラメータのデータ型に拡大変換されるデータ型の引数だけを渡す
- 暗黙の縮小変換を許可する **Option Strict Off** を設定する
- 変換キーワードを使用してデータ型を明示的に変換する

型パラメータ。

ジェネリック プロシージャ も、通常のパラメータの他に 1 つ以上の型パラメータを定義します。ジェネリック プロシージャを使うと、呼び出し元のコードは、プロシージャを呼び出すたびに異なるデータ型を渡すことができるため、個々の呼び出しの要件に合わせてデータ型を変更できます。Visual Basic におけるジェネリック プロシージャ を参照してください。

参照

処理手順

方法 : プロシージャにパラメータを定義する
方法 : プロシージャに引数を渡す

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Function プロシージャ
Property プロシージャ
演算子プロシージャ
引数の値渡しおよび参照渡し
プロシージャのオーバーロード

その他の技術情報

Visual Basic における型変換

パラメータと引数の違い

プロシージャは、ほとんどの場合、呼び出されたときの状況に関する情報を必要とします。繰り返し実行されるタスクや共有されているタスクを実行するプロシージャは、呼び出されるたびに異なる情報を使用します。この情報は、プロシージャを呼び出すときに渡される変数、定数、および式から構成されています。

この情報をプロシージャに渡すために、プロシージャにはパラメータが定義され、呼び出し元のコードはそのパラメータに引数を渡します。パラメータは駐車場、引数は自動車であると考えてみてください。駐車場にはさまざまな車が駐車できるのと同様に、呼び出し元のコードは、パラメータを呼び出すたびに、同じパラメータに別の引数を渡すことができます。

パラメータ

パラメータは、プロシージャが呼び出されるときに期待する値を表します。パラメータは、プロシージャの宣言で定義されます。

Function または **Sub** プロシージャを定義するときは、プロシージャ名のすぐ後に、かっこで囲んだパラメータ リストを指定します。各パラメータには、名前、データ型、引き渡し方法 (ByVal または ByRef) を指定します。また、パラメータが省略可能 (呼び出し元のコードが値を渡す必要がない) かどうかを示すこともできます。

各パラメータの名前は、プロシージャ内でローカル変数として扱われます。パラメータ名は、他の変数と同じように使用できます。

引数

引数は、プロシージャを呼び出すときに、プロシージャのパラメータに渡す値を表します。呼び出し元のコードは、プロシージャを呼び出すときに引数を渡します。

Function または **Sub** プロシージャを定義するときは、プロシージャ名のすぐ後にかっこで囲んだ引数リストを含めます。リスト内の各引数は、パラメータ リスト内の同じ位置にあるパラメータに対応します。

パラメータの定義とは異なり、引数には名前がありません。引数は式で、0 以上の変数、定数、リテラルを含めることができます。評価済みの式のデータ型は、通常は、対応するパラメータに定義されたデータ型と一致します。それ以外の場合でも、パラメータのデータ型と互換性がある必要があります。

参照

処理手順

方法 : プロシージャにパラメータを定義する
方法 : プロシージャに引数を渡す

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Function プロシージャ
Property プロシージャ
演算子プロシージャ
引数の値渡しおよび参照渡し
再帰プロシージャ
プロシージャのオーバーロード

方法：プロシージャにパラメータを定義する

パラメータ は、呼び出し元のコードが値をプロシージャに渡すために使用します。プロシージャのパラメータを宣言する場合には、変数を宣言するときと同じように、パラメータの名前とデータ型を指定します。また、渡す方法や、パラメータを省略できるかどうかも指定できます。

詳細については、「プロシージャのパラメータと引数」を参照してください。

プロシージャのパラメータを定義するには

1. プロシージャの宣言中に、プロシージャのパラメーター覧に、他のパラメータとの間をコンマで区切ってパラメータ名を追加します。
2. パラメータのデータ型を決定します。
3. パラメータ名の後に **As** 句を入力し、データ型を指定します。
4. パラメータの渡し方を決定します。通常、パラメータは値によって渡されますが、呼び出し元のコード内の値をプロシージャが変更できるように指定することもできます。
5. パラメータ名の前には ByVal または ByRef を指定して、渡し方を指定します。詳細については、「引数の値渡しと参照渡しの違い」を参照してください。
6. パラメータが省略可能である場合、渡し方の前に Optional (Visual Basic) を指定して、パラメータのデータ型の後に等号 (=) と既定値を指定します。

   次の例では、3 つのパラメータを持つ **Sub** プロシージャの骨組みを定義します。最初の 2 つのパラメータは必須で、3 つ目は省略可能です。パラメータの一覧の中で、各パラメータの定義はコンマで区切られます。

   VB

   ```
   Sub updateCustomer(ByRef c As customer, ByVal region As String, _
     Optional ByVal level As Integer = 0)
     ' Insert code to update a customer object.
   End Sub
   ```

   最初のパラメータは `customer` オブジェクトを受け入れ、引数は ByRef で渡されているので、`updateCustomer` は `c` に渡された変数を直接更新できます。最後の 2 つの引数は ByVal で渡されているので、プロシージャはこれらの値を変更することはできません。

   呼び出し元のコードに `level` パラメータの値が指定されていない場合、Visual Basic はこれを既定値である 0 に設定します。

   型チェックのスイッチ (Option Strict ステートメント) が **Off** の場合、パラメータの定義時に **As** 句を省略できます。しかし、いずれかのパラメータが **As** 句を使用している場合は、すべてのパラメータがこれを使用する必要があります。型チェックのスイッチが **On** の場合、**As** 句はすべてのパラメータ定義で必須です。

   すべてのプログラミング要素についてデータ型を指定することは、厳密な型指定と呼ばれます。**Option Strict On** を設定した場合、Visual Basic は厳密な型指定を強制します。これは、次の理由で強く推奨されています。

   - 変数に対する IntelliSense サポートが有効になります。これにより、コードの入力時に、プロパティや他のメンバを表示できます。
   - コンパイラで型チェックを実行できます。これは、実行時にオーバーフローなどのエラーで失敗するステートメントを検出するのに役立ちます。また、オブジェクトでサポートされていないメソッドの呼び出しも検出されます。
   - コードの実行速度が速くなります。この理由の 1 つは、プログラミング要素にデータ型を指定しなかった場合、Visual Basic コンパイラが **Object** タイプを割り当てるためです。コンパイル済みのコードが、**Object** および他のデータ型との間で変換を行う必要がある場合、パフォーマンスが低下します。

参照

処理手順

方法 : プロシージャに引数を渡す

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Function プロシージャ
引数の値渡しおよび参照渡し

再帰プロシージャ
プロシージャのオーバーロード
その他の技術情報
Visual Basic におけるオブジェクト指向プログラミング

方法：プロシージャに引数を渡す

プロシージャを呼び出すときには、プロシージャ名に続けて、引数をかっこ内に並べて指定します。プロシージャに定義されたすべての必須パラメータに対応する引数を指定します。**Optional** のパラメータへの引数の指定は省略することもできます。**Optional** のパラメータの指定を省略した場合、その後に続けて別の引数を指定するのであれば、リスト内で引数を省略した位置を示すためにコンマを入力する必要があります。

String 型のパラメータに **Byte** 型を渡すなど、パラメータの型とは異なるデータ型の引数を渡す場合は、型チェックのスイッチ (Option Strict ステートメント) に **Off** を設定してください。**Option Strict** が **On** である場合は、拡大変換を使用するか、型変換のキーワードを明示的に使用する必要があります。詳細については、「拡大変換と縮小変換」および「データ型変換関数」を参照してください。

詳細については、「プロシージャのパラメータと引数」を参照してください。

1 つ以上の引数をプロシージャに渡すには

1. 呼び出しを行っているステートメントで、プロシージャ名に続けてかっこを入力します。
2. かっこの内側に引数リストを入力します。プロシージャに定義された各必須パラメータに対する引数を、コンマで区切って入力します。
3. 各引数が、対応するパラメータのデータ型 (プロシージャで定義された型) に変換可能な型に評価される、有効な式であることを確認してください。
4. パラメータが Optional (Visual Basic) で定義されている場合は、引数リストに指定することも省略することもできます。省略した場合は、そのパラメータに定義された既定値がプロシージャで使用されます。
5. **Optional** のパラメータの引数を省略し、その後に別のパラメータを指定する場合は、引数リスト内にコンマを追加して引数を省略した位置を示すようにします。

   Visual Basic の MsgBox 関数 (Visual Basic) を呼び出す例を次に示します。

   VB

   ```
   Dim mbResult As MsgBoxResult
   Dim displayString As String = "Show this string to the user"
   mbResult = MsgBox(displayString, , "Put this in the title bar")
   ```

   この例では、必須である最初の引数が指定されています。この引数は表示されるメッセージの文字列です。オプションである 2 つ目のパラメータは省略されています。これはメッセージ ボックスに表示するボタンを指定する引数です。値を指定せずに呼び出しているため、**MsgBox** は既定値である **MsgBoxStyle.OKOnly** を使用し、[OK] ボタンだけを表示します。

   引数リストにある 2 つ目のコンマが、省略された 2 番目の引数の位置を示し、最後の文字列が **MsgBox** の 3 番目のパラメータ (オプション) に渡されます。これはタイトル バーに表示されるテキストです。

参照

処理手順

方法 : プロシージャにパラメータを定義する

概念

Sub プロシージャ
Function プロシージャ
Property プロシージャ
演算子プロシージャ
引数の値渡しおよび参照渡し
再帰プロシージャ
プロシージャのオーバーロード

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

引数の値渡しおよび参照渡し

Visual Basic では、値渡しまたは参照渡しで引数をプロシージャに渡すことができます。これは引渡し方法と呼ばれ、引数の基となる、呼び出し元のコードのプログラミング要素をプロシージャが変更できるかどうかが決まります。プロシージャの宣言では、ByVal または ByRef キーワードを指定することで、各パラメータの引き渡し方法を決定します。

引き渡し方法の違い

引数をプロシージャに渡す場合、2 つの引き渡し方法の違いに注意してください。

- 基になるプログラミング要素が変更可能か変更不可能か
- 引数自体が変更可能か変更不可能か
- 引数が値渡しか参照渡しか
- 引数のデータ型が値型か参照型か

詳細については、「変更できる引数と変更できない引数の違い」および「引数の値渡しと参照渡しの違い」を参照してください。

引数渡しの方法の選択

各引数の引き渡し方法は慎重に決定してください。

- **保護**.引数を渡す方法を選択するときに最も重要な基準となるのは、呼び出し元のコードの変数を変更できるようにするかどうかです。**ByRef** で引数を渡す場合の利点は、プロシージャから呼び出し元のコードに引数を通じて値を返すことができるという点にあります。一方、**ByVal** を使用する利点は、プロシージャによって変数が変更されるのを防ぐことができるという点にあります。
- **パフォーマンス**.どちらの方法を使用するかによってコードのパフォーマンスが変わってきますが、その差は一般にごくわずかです。ただし、値型を **ByVal** で渡す場合は例外です。この場合は、引数の内容がすべてコピーされます。このため、構造体などの大きな値型では、**ByRef** で渡す方が効率的です。

  参照型では、データへのポインタだけがコピーされます (32 ビット プラットフォームでは 4 バイト、64 ビット プラットフォームでは 8 バイト)。したがって、パフォーマンスを損なうことなく、**String** 型や **Object** 型の引数を値で渡すことができます。

引数渡しの方法の決定

プロシージャの宣言では、各パラメータの引き渡し方法を指定します。宣言で **ByVal** が使用されている場合、呼び出し元のコードでそれを変更することはできません。しかし、**ByRef** を使って引数が宣言されている場合は、呼び出し元のコードで引数名をかっこで囲むことによって、**ByVal** で渡すように変更できます。

Visual Basic の既定の設定では、値渡しで引数が渡されます。**ByVal** キーワードを使うとコードが読みやすくなります。パラメータを宣言するときには、**ByVal** または **ByRef** のいずれかのキーワードを常に含めるようにすることをお勧めします。

引数を値で渡す場合

- 引数の基になる呼び出し元のコード要素が変更不可能である場合は、これに対応するパラメータは ByVal で宣言します。変更不可能な要素の値は、どのようなコードでも変更できません。
- 基になる要素が変更可能であっても、プロシージャからはその値を変更できないようにするには、パラメータを **ByVal** で宣言します。変更可能な要素が値渡しされた場合、その値を変更できるのは呼び出し元のコードだけです。

引数を参照で渡す場合

- プロシージャが、呼び出し元のコード内にある基になる要素を変更する必要が本当にある場合、対応するパラメータを ByRef で宣言します。
- プロシージャが正しく実行されるかどうかが、呼び出し元のコード内の基になる要素を変更するプロシージャに依存する場合、パラメータを **ByRef** で宣言します。これを値渡しした場合、または、呼び出し元のコードが引数をかっこで囲んで **ByRef** の引き渡し方法をオーバーライドした場合、プロシージャは予期しない結果になることがあります。

参照

処理手順

方法 : プロシージャに引数を渡す
方法 : プロシージャ引数の値を変更する

方法 : プロシージャ引数の値が変化しないようにする
方法 : 引数の値渡しを強制する

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
位置と名前による引数渡し
値型と参照型

変更できる引数と変更できない引数の違い

プロシージャを呼び出すとき、通常はいくつかの引数を渡します。各引数は、基底にあるプログラミング要素に対応します。基底にある要素と引数の両方が、変更できる場合とできない場合があります。

変更できる要素と変更できない要素

プログラミング要素は、値を変更できる変更できる要素と、作成時から値が変更されない 変更できない要素の 2 つに分けられます。

次の表に、変更できるプログラミング要素と変更できないプログラミング要素を挙げます。

変更できる要素	変更できない要素
オブジェクト変数を含むローカル変数 (プロシージャ内で宣言されたもの) 読み取り専用を除く	読み取り専用変数、フィールド、プロパティ
フィールド (モジュール、クラス、構造体のメンバ変数) 読み取り専用を除く	定数とリテラル
プロパティ、読み取り専用を除く	列挙型メンバ
配列要素	式 (式内の要素が変更可能な場合も)

変更できる引数と変更できない引数

変更できる引数は、基底にある要素が変更できるものです。呼び出し元のコードは、いつでも新しい値を格納でき、引数 ByRef を渡すと、プロシージャ内のコードが、呼び出し元のコードにある基底の要素を変更できます。

変更できない引数は、変更できない要素が基底にあるか、ByVal で渡されています。呼び出し元のコードの基底の要素が変更可能な要素であっても、プロシージャはこれを変更できません。基底の要素が変更できない要素である場合、呼び出し元のコード自身もこれを変更できません。

呼び出したプロシージャによって変更できない引数のコピーが変更されることはあっても、その変更が呼び出し元のコードの基の要素に影響することはありません。

参照

処理手順

方法 : プロシージャに引数を渡す
方法 : プロシージャ引数の値を変更する
方法 : プロシージャ引数の値が変化しないようにする
方法 : 引数の値渡しを強制する

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
引数の値渡しと参照渡しの違い
位置と名前による引数渡し
値型と参照型

引数の値渡しと参照渡しの違い

プロシージャに 1 つ以上の引数を渡す場合、各引数は呼び出し元のコードにある基のプログラミング要素に対応付けられます。この基になる要素の値を渡すこともあれば、要素への参照を渡すこともあります。これを、値渡しと参照渡しと呼びます。

値渡し

引数を値渡しで渡すには、プロシージャ定義内で対応するパラメータに ByVal キーワードを指定します。値渡しを使用すると、Visual Basic は基になるプログラミング要素の値をプロシージャ内のローカル変数にコピーします。プロシージャ コードから呼び出し元のコードにある基の要素にアクセスすることはできません。

参照渡し

引数を参照渡しで渡すには、プロシージャ定義内で対応するパラメータに ByRef キーワードを指定します。参照渡しを使用すると、Visual Basic は呼び出し元のコードにある基のプログラミング要素を、プロシージャから直接参照できるようにします。

引数渡しの方法と要素の型

値渡しと参照渡しのどちらを選ぶかは、基になる要素の型に応じて決めるわけではありません。値渡しか参照渡しかによって、Visual Basic がプロシージャ コードに何を提供するかが変わります。一方、値型か参照型かによって、プログラミング要素がメモリにどのように格納されるかが変わります。

しかし、値渡し/参照渡しと要素の型は無関係ではありません。参照型の値は、メモリ内の他の場所にあるデータへのポインタです。つまり、参照型を値渡しで渡すと、プロシージャ コードは基の要素自体にはアクセスできませんが、基の要素のデータをポイントできるようになります。たとえば、要素が配列変数であった場合、プロシージャ コードは変数そのものにはアクセスできませんが、配列のメンバにはアクセスできます。

要素の変更

不変要素を引数として渡す場合、**ByVal** と **ByRef** のどちらを使用するかに関係なく、プロシージャは呼び出し元のコードにある要素を変更できません。

可変要素の場合の要素のデータ型と引数渡しの方法との関係を次の表に示します。

要素の型	ByVal で渡す場合	ByRef で渡す場合
値型 (格納されるのは値のみ)	プロシージャは、変数およびそのメンバを一切変更できません。	プロシージャは、変数およびそのメンバを変更できます。
参照型 (クラスまたは構造体のインスタンスへのポインタを格納)	プロシージャは、変数を変更することはできませんが、変数が指すインスタンスのメンバを変更できます。	プロシージャは、変数および変数が指すインスタンスのメンバを変更できます。

参照

処理手順

方法 : プロシージャに引数を渡す
方法 : プロシージャ引数の値を変更する
方法 : プロシージャ引数の値が変化しないようにする
方法 : 引数の値渡しを強制する

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
変更できる引数と変更できない引数の違い
位置と名前による引数渡し
値型と参照型

方法：プロシージャ引数の値を変更する

プロシージャを呼び出すときに指定する各引数は、プロシージャに定義されたパラメータのいずれかに対応します。場合によっては、プロシージャのコードで、呼び出し元のコードにある引数の基の値が変更されることもあります。または、引数のローカル コピーだけがプロシージャで変更される場合もあります。

プロシージャを呼び出すと、Visual Basic は ByVal で渡されたすべての引数のローカル コピーを作成します。ByRef で渡された各引数に対しては、プロシージャ コードから呼び出し元のコードにある引数の基のプログラミング要素を直接参照できるように、Visual Basic が操作します。

呼び出し元のコードにある基の要素が可変であり、引数が **ByRef** で渡されていれば、プロシージャ コードから呼び出し元のコードにある要素の値を直接参照して変更できます。

基の値を変更する

呼び出し元のコードにある、プロシージャ引数の基の値を変更するには

1. プロシージャ宣言で、引数に対応するパラメータに ByRef を指定します。
2. 呼び出し元のコードで、可変のプログラミング要素を引数として渡します。
3. 呼び出し元のコードで、引数を引数リストのかっこで囲まないでください。
4. プロシージャ コードでパラメータ名を使用して、呼び出し元のコードにある基の要素に値を代入します。

詳しくは、後に示すコード例を参照してください。

ローカル コピーの変更

呼び出し元のコードにある基の要素が不変であるか、または引数が **ByVal** で渡されている場合、プロシージャから呼び出し元のコードにある基の値を変更できません。ただし、プロシージャからこの引数のローカル コピーを変更できます。

プロシージャ コードにある、プロシージャ引数のコピーを変更するには

1. プロシージャ宣言で、引数に対応するパラメータに ByVal を指定します。

   または

   呼び出し元のコードで、引数を引数リストのかっこで囲みます。こうすると、Visual Basic は引数に対応するパラメータに **ByRef** が指定されている場合でも、引数を値渡しで渡します。

2. プロシージャ コードでパラメータ名を使用して、引数のローカル コピーに値を代入します。呼び出し元のコードにある基の値は変更されません。

使用例

次の例には、配列変数を受け取ってその要素を操作する 2 つのプロシージャがあります。`increase` プロシージャは、各要素に単純に 1 を加算します。`replace` プロシージャは、パラメータ `a()` に新しい配列を代入してから各要素に 1 を加算します。

VB

```
Public Sub increase(ByVal a() As Long)
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

VB

```
Public Sub replace(ByRef a() As Long)
    Dim k() As Long = {100, 200, 300}
    a = k
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

VB

```
Dim n() As Long = {10, 20, 30, 40}
Call increase(n)
MsgBox("After increase(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
Call replace(n)
MsgBox("After replace(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
```

最初に **MsgBox** を呼び出すと、"After increase(n): 11, 21, 31, 41" が表示されます。配列 `n` が参照型なので、引数渡しの方法が **ByVal** であっても `replace` からそのメンバを変更できます。

2 度目に **MsgBox** を呼び出すと、"After replace(n): 101, 201, 301" が表示されます。`n` が **ByRef** で渡されたため、 `replace` から呼び出し元のコードにある変数 `n` を変更でき、新しい配列を変数に代入できます。 `n` が参照型なので、 `replace` からそのメンバを変更することもできます。

プロシージャから呼び出し元のコードにある変数そのものを変更しないようにできます。「方法 : プロシージャ引数の値が変化しないようにする」を参照してください。

コードのコンパイル方法

参照渡しで変数を渡すときには、**ByRef** キーワードを使って明示的に指定する必要があります。

Visual Basic の既定の設定では、値渡しで引数が渡されます。しかし、パラメータを宣言するときには、ByVal または ByRef のいずれかのキーワードを常に指定することをお勧めします。これによって、コードが読みやすくなります。

セキュリティ

呼び出し元のコードにある引数の基の値をプロシージャから変更できるようにすると、必ず危険が伴います。変更すべき値が変更されていることを確認し、検証用のコードを作成して値を使用する前にチェックしてください。

参照

処理手順

方法 : プロシージャに引数を渡す
方法 : プロシージャ引数の値が変化しないようにする
方法 : 引数の値渡しを強制する

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
変更できる引数と変更できない引数の違い
引数の値渡しと参照渡しの違い
位置と名前による引数渡し
値型と参照型

方法：プロシージャ引数の値が変化しないようにする

プロシージャがパラメータを ByRef で宣言すると、Visual Basic はプロシージャ コードに、呼び出し元のコードの引数の基となるプログラミング要素への直接の参照を追加します。これにより、プロシージャが呼び出し元のコードの引数の基となる値を変更できます。場合によっては、呼び出し元のコードがこのような変更を禁止する場合もあります。

引数が変更されるのを防ぐには、プロシージャ内で、対応するパラメータを ByVal で宣言します。特定の引数を、特定の場合のみ変更するには、この引数を **ByRef** で宣言し、呼び出し元のコードが、呼び出しのたびに引き渡し方法を決定できるようにします。対応する引数をかっこで囲むと値で渡され、かっこで囲まないと参照が渡されます。詳細については、「方法 : 引数の値渡しを強制する」を参照してください。

使用例

次の例には、配列変数を受け取ってその要素を操作する 2 つのプロシージャがあります。`increase` プロシージャは、各要素に単純に 1 を加算します。`replace` プロシージャは、新しい配列をパラメータ `a()` 煮割り当て、各要素に 1 を加算します。ただし、この新しい配列の代入は、呼び出し元のコードの基の配列変数には反映されません。

VB

```
Public Sub increase(ByVal a() As Long)
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

VB

```
Public Sub replace(ByVal a() As Long)
    Dim k() As Long = {100, 200, 300}
    a = k
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

VB

```
Dim n() As Long = {10, 20, 30, 40}
Call increase(n)
MsgBox("After increase(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
Call replace(n)
MsgBox("After replace(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
```

最初の **MsgBox** の呼び出しでは、"After increase(n): 11, 21, 31, 41" と表示されます。配列 `n` は参照型なので、引き渡し方法が **ByVal** でも、 `replace` によってこのメンバを変更できます。

2 番目の **MsgBox** の呼び出しでは、"After replace(n): 11, 21, 31, 41" と表示されます。 `n` は **ByVal** で渡されるので、 `replace` は呼び出し元のコードで新しい配列を割り当てても変数 `n` を変更できません。 `replace` が新しい配列インスタンス `k` を作成して、これをローカル変数 `a` に割り当てた場合、呼び出し元のコードが渡した `n` への参照は失われます。 `a` のメンバを変更すると、ローカル配列 `k` のみが影響を受けます。したがって、 `replace` は、呼び出し元のコードにある配列 `n` の値をインクリメントしません。

コードのコンパイル方法

Visual Basic の既定の設定では、値渡しで引数が渡されます。パラメータを宣言するときには、ByVal または ByRef のいずれかのキーワードを常に含めるようにすることをお勧めします。これによって、コードが読みやすくなります。

参照

処理手順

方法 : プロシージャに引数を渡す
方法 : プロシージャ引数の値を変更する
方法 : 引数の値渡しを強制する

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
変更できる引数と変更できない引数の違い
引数の値渡しと参照渡しの違い
位置と名前による引数渡し
値型と参照型

方法：引数の値渡しを強制する

引数を渡す方法は、プロシージャの宣言によって決まります。パラメータが ByRef に宣言されている場合、Visual Basic は対応する引数が参照渡しで渡されると予測します。この場合、プロシージャは呼び出し元のコードにある引数の基のプログラミング要素の値を変更できます。基の要素をこの方法で変更しないように保護する場合は、プロシージャの呼び出し時に引数名をかっこで囲むことによって、**ByRef** の引数渡しの方法をオーバーライドします。このかっこは、呼び出し時に引数リストを囲むかっこに追加して記述します。

呼び出し元のコードで ByVal の引数渡しの方法をオーバーライドすることはできません。

引数の値渡しを強制するには

- プロシージャ内で対応するパラメータが **ByVal** で宣言されている場合、Visual Basic は引数が値渡しで渡されると予測するため、それ以上何もする必要がありません。
- プロシージャ内で対応するパラメータが **ByRef** で宣言されている場合は、プロシージャの呼び出し時に引数をかっこで囲みます。

使用例

ByRef のパラメータ宣言をオーバーライドする例を次に示します。**ByVal** を強制している呼び出しの部分で、かっこが二重に記述されている点に注意してください。

VB

```
Sub setNewString(ByRef inString As String)
    inString = "This is a new value for the inString argument."
    MsgBox(inString)
End Sub
```

VB

```
Dim str As String = "Cannot be replaced if passed ByVal"

' The following call passes str ByVal even though it is declared ByRef.
Call setNewString((str))
' The parentheses around str protect it from change.
MsgBox(str)

' The following call allows str to be passed ByRef as declared.
Call setNewString(str)
' Variable str is not protected from change.
MsgBox(str)
```

引数リストの内部で `str` が二重のかっこに囲まれると、`setNewString` プロシージャは呼び出し元のコードにあるその値を変更できず、**MsgBox** に "Cannot be replaced if passed ByVal" と表示します。`str` が二重のかっこで囲まれなければ、プロシージャは値を変更でき、**MsgBox** には "This is a new value for the inString argument" と表示されます。

コードのコンパイル方法

参照渡しで変数を渡すときには、**ByRef** キーワードを使って明示的に指定する必要があります。

Visual Basic の既定の設定では、値渡しで引数が渡されます。しかし、パラメータを宣言するときには、ByVal または ByRef のいずれかのキーワードを常に指定することをお勧めします。これによって、コードが読みやすくなります。

堅牢性の高いプログラム

プロシージャにパラメータが ByRef で宣言されている場合、そのコードを正しく実行できるかどうかは、呼び出し元のコードにある基の要素を変更できるかどうかに左右されます。呼び出し元のコードで引数をかっこで囲むことによって、この引数渡しの方法がオーバーライドされた場合、または変更不可能な引数が渡された場合は、プロシージャから基の要素を変更できません。これによって、呼び出し元のコードで予期しない結果になる場合があります。

セキュリティ

呼び出し元のコードにある引数の基の値をプロシージャから変更できるようにすると、必ず危険が伴います。変更すべき値が変更されていることを確認し、検証用のコードを作成して値を使用する前にチェックしてください。

参照

処理手順

方法 : プロシージャに引数を渡す
方法 : プロシージャ引数の値を変更する
方法 : プロシージャ引数の値が変化しないようにする

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
変更できる引数と変更できない引数の違い
引数の値渡しと参照渡しの違い
位置と名前による引数渡し
値型と参照型

位置と名前による引数渡し

Sub プロシージャまたは **Function** プロシージャを呼び出すときには、引数を位置で渡したり名前で渡したりできます。位置で渡す場合は、プロシージャの定義で宣言されている順に引数を指定します。名前で渡す場合は、指定する位置 (順序) は関係なくなります。

引数を名前で渡すときには、宣言されている引数の名前、コロンと等号 (:=)、引数の値という順に指定します。引数はどのような順序でも指定できます。

たとえば、次の **Sub** プロシージャでは 3 つの引数を使用します。

VB

```
Sub studentInfo(ByVal name As String, _
       Optional ByVal age As Short = 0, _
       Optional ByVal birth As Date = #1/1/2000#)

  Debug.WriteLine("Name = " & name & _
                  "; age = " & CStr(age) & _
                  "; birth date = " & CStr(birth))
End Sub
```

このプロシージャを呼び出すときには、引数を位置で指定することも、名前で指定することも、両方を一緒に使って指定することもできます。

位置による引数渡し

引数を位置で渡してプロシージャ `studentInfo` を呼び出すには、次のようにコンマで区切って指定します。

VB

```
Call studentInfo("Mary", 19, #9/21/1981#)
```

位置で指定する引数リストで省略可能な引数を省略する場合は、省略する引数の場所にコンマを置く必要があります。 `age` 引数を指定せずに `studentInfo` を呼び出す例を次に示します。

VB

```
Call studentInfo("Mary", , #9/21/1981#)
```

名前による引数渡し

引数を名前で渡してプロシージャ `studentInfo` を呼び出すこともできます。この場合も、次のようにコンマで区切って指定します。

VB

```
Call studentInfo(age:=19, birth:=#9/21/1981#, name:="Mary")
```

位置と名前の両方による引数渡し

次に示す例のように、1 つのプロシージャ呼び出しで、位置と名前の両方を使って引数を指定することもできます。

VB

```
Call studentInfo("Mary", birth:=#9/21/1981#)
```

上の例では、引数 `age` が省略されていますが、 `birth` が名前で指定されているため、 `age` の場所にコンマを置く必要はありません。

位置と名前の両方を使って引数を指定する場合は、位置で指定する引数をすべて先に指定する必要があります。いったん名前で引数を指定したら、残りの引数はすべて名前で指定する必要があります。

名前による省略可能な引数の指定

名前による引数渡しは、省略可能な引数が複数あるプロシージャを呼び出す場合に便利です。引数を名前で指定する場合は、省略する引数の位置を示すためのコンマは必要はありません。また、引数を名前で指定すると、どの引数を渡してどの引数を省略したのかが把握しやすくなります。

名前による引数渡しの制限事項

引数を名前で渡しても、必要な引数を省略することはできません。省略できるのは、省略可能な引数だけです。

パラメータ配列を名前で渡すことはできません。これは、プロシージャの呼び出し時に、パラメータ配列ではコンマで区切られた不特定多数の引数が指定されますが、コンパイラは複数の引数を 1 つの名前に関連付けることができないためです。

参照

処理手順

方法 : プロシージャに引数を渡す
方法 : プロシージャに引数を名前で渡す

関連項目

Optional (Visual Basic)
ParamArray

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
省略可能なパラメータ
パラメータ配列

方法：プロシージャに引数を名前で渡す

Sub または **Function** プロシージャを呼び出す場合、対応するパラメータの、プロシージャの定義における順序に関係なく、引数を名前渡しできます。

引数を名前渡しすると、プロシージャ呼び出し内で引数値に意味を持たせることができるので、呼び出し元のコードが読みやすくなります。プロシージャに省略可能なパラメータがある場合、名前渡しすることで、どの引数を渡し、どの引数を省略しているのかを確認しやすくなります。

引数の名前渡しの規則と制限については、「位置と名前による引数渡し」を参照してください。

引数を名前で渡すには

1. プロシージャの宣言のソース コードで、パラメータ名の正確な綴りを確認します。
2. 呼び出し元のコードで、通常どおりにプロシージャ呼び出しを準備し、プロシージャ名に続いてかっこで囲んだ引数リストを指定します。
3. 名前渡しするすべての引数に対して、宣言されたパラメータ名、コロンと等号 (:=)、引数に渡す値の順で指定します。
4. 名前で指定する引数の順序は自由ですが、位置で指定する引数はすべて、名前で指定する引数の前に指定する必要があります。

使用例

次の例では、3 つのパラメータを持つ **Sub** プロシージャと、これらのパラメータに引数を名前渡しする Call ステートメント (Visual Basic) を示します。

VB

```
Sub studentInfo(ByVal name As String, _
       Optional ByVal age As Short = 0, _
       Optional ByVal birth As Date = #1/1/2000#)

  Debug.WriteLine("Name = " & name & _
                  "; age = " & CStr(age) & _
                  "; birth date = " & CStr(birth))
End Sub
```

VB

```
Call studentInfo(age:=19, birth:=#9/21/1981#, name:="Mary")
```

引数を名前で渡す場合、プロシージャでの宣言時と同じ順序で指定する必要はありません。

コードのコンパイル方法

引数リスト内のパラメータ名が、プロシージャで宣言されたものと完全に一致していることを確認してください。

参照

処理手順

方法 : プロシージャに引数を渡す

関連項目

Optional (Visual Basic)

ParamArray

概念

Visual Basic におけるプロシージャ

プロシージャのパラメータと引数

引数の値渡しおよび参照渡し

位置と名前による引数渡し

省略可能なパラメータ

パラメータ配列

省略可能なパラメータ

プロシージャのパラメータを省略可能にすると、呼び出し時に引数を指定する必要がなくなります。省略可能なパラメータ にするには、プロシージャ定義で **Optional** キーワードを使用します。次の規則が適用されます。

- プロシージャ定義のすべての省略可能なパラメータについて、既定値を指定する必要があります。
- 省略可能なパラメータの既定値には、定数式を指定する必要があります。
- プロシージャ定義で省略可能なパラメータの後に続くパラメータは、すべて省略可能であることが必要です。

次の構文は、省略可能なパラメータを含むプロシージャ宣言を示しています。

```
Sub sub name(ByVal parameter 1 As data type 1, Optional ByVal parameter 2 As data type 2 =
default value)
```

省略可能なパラメータを使ったプロシージャ呼び出し

省略可能なパラメータを使ってプロシージャを呼び出すときには、引数を指定するかどうかを選択できます。引数を指定しない場合は、そのパラメータに対して宣言されている既定値が使用されます。

引数リストで省略可能な引数を省略する場合は、コンマを続けて、省略する引数の位置を表します。次の例では、1 番目と 4 番目の引数は指定されていますが、2 番目と 3 番目の引数は省略されています。

```
Call sub name(argument 1, , , argument 4)
```

省略可能な引数があるかどうかの確認

引数が省略されているのか、呼び出し元のコードで既定値が明示的に指定されているのかについて、プロシージャで実行時に検出することはできません。この区別が必要な場合は、ありそうにない値を既定値に設定します。次のプロシージャでは、省略可能なパラメータ `office` を定義し、その既定値 `QJZ` をテストして、呼び出しの際にこの引数が省略されているかどうかを確認します。

VB

```
Sub notify(ByVal company As String, Optional ByVal office As String = "QJZ")
    If office = "QJZ" Then
        Debug.WriteLine("office not supplied -- using Headquarters")
        office = "Headquarters"
    End If
    ' Insert code to notify headquarters or specified office.
End Sub
```

省略可能なパラメータが **String** などの参照型の場合は、**Nothing** を既定値として使用できます。ただし、**Nothing** が引数の値として使用されることが予想される場合を除きます。

省略可能なパラメータとオーバーロード

省略可能なパラメータを持つプロシージャを定義するには、オーバーロードを使用する方法もあります。省略可能なパラメータが 1 つあるとすると、パラメータを受け取る場合と受け取らない場合の、2 つのオーバーロードされたバージョンのプロシージャを定義できます。この方法は、省略可能なパラメータの数が増えるにつれて複雑になります。しかし、それぞれの省略可能な引数が呼び出しプログラムによって指定されているかどうかを確実に把握できるという利点があります。

参照

処理手順

方法 : 省略可能なパラメータをプロシージャに定義する
方法 : 省略可能なパラメータを受け取るプロシージャを呼び出す
方法 : 省略可能なパラメータが使用されているかどうかを確認する

関連項目

Optional (Visual Basic)
ParamArray

概念

Visual Basic におけるプロシージャ

方法：省略可能なパラメータをプロシージャに定義する

プロシージャのパラメータを省略可能に指定して、プロシージャを呼び出すときに、コードがそのパラメータに引数を渡さなくてもよいようにできます。省略可能に指定する場合は、引数が指定されなかった場合にプロシージャが使用する既定値を定義します。

省略可能なパラメータは複数定義できますが、省略可能なパラメータはすべてパラメータ リストの末尾に置く必要があります。すべての必須パラメータを、すべての省略可能なパラメータの前に定義することが必要です。

省略可能なパラメータを定義するには

1. プロシージャ宣言のパラメータ リスト内で、パラメータ名の前に **Optional** キーワードを記述します。
2. パラメータ名の後に、通常と同じように **As** 句を記述し、**As** 句の後に等号 (=) を入力します。
3. 等号の後に、パラメータの既定値を指定します。既定値は定数式であることが必要です。そうしないと、コンパイラが既定値をコンパイル時に完全にコンパイルできません。
4. 省略可能なパラメータ以降のパラメータは、すべて **Optional** で宣言する必要があります。

使用例

省略可能なパラメータを含むプロシージャ宣言は、次の例のようになります。

VB

```
Sub notify(ByVal company As String, Optional ByVal office As String = "QJZ")
    If office = "QJZ" Then
        Debug.WriteLine("office not supplied -- using Headquarters")
        office = "Headquarters"
    End If
    ' Insert code to notify headquarters or specified office.
End Sub
```

呼び出しを行っているコードが引数リスト内の `office` に値を渡さなければ、Visual Basic が既定値 "QJZ" を渡します。

コードのコンパイル方法

すべての省略可能なパラメータに対して、プロシージャ宣言内で既定値を指定する必要があります。コンパイラがコンパイル時に評価できるように、各既定値には必ず定数を指定してください。

参照

処理手順

方法 : 省略可能なパラメータを受け取るプロシージャを呼び出す
方法 : 省略可能なパラメータが使用されているかどうかを確認する

関連項目

Optional (Visual Basic)
ParamArray

概念

プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
位置と名前による引数渡し
省略可能なパラメータ
パラメータ配列
プロシージャのオーバーロード

方法：省略可能なパラメータを受け取るプロシージャを呼び出す

省略可能なパラメータを使ってプロシージャを呼び出すときには、対応する引数を指定するかどうかを選択できます。引数を指定しない場合は、そのパラメータに対して宣言されている既定値が使用されます。

- 引数を使用する場合は、通常どおり、コンマで区切って引数リストに含めます。
- 引数を省略する場合は、引数リストの中でコンマを続けて、その引数の省略を示します。
- 引数を省略し、引数を名前で渡す場合、省略する引数を名前やコンマで示す必要はありません。

使用例

次の例では、**MsgBox** 関数を数回呼び出します。**MsgBox** には、必須パラメータ 1 つと省略可能なパラメータが 2 つあります。

VB

```
MsgBox("Important message", MsgBoxStyle.Critical, "MsgBox Example")
MsgBox("Just display this message.")
MsgBox("Test message", , "Title bar text")
MsgBox(Title:="Title bar text", Prompt:="Test message")
```

MsgBox の最初の呼び出しでは、**MsgBox** で定義された順番で 3 つの引数を指定します。2 番目の呼び出しでは、必須の引数だけを指定します。3 番目と 4 番目の呼び出しでは、1 つ目と 3 つ目の引数を指定します。3 番目の呼び出しでは引数を位置で指定し、4 番目の呼び出しでは引数を名前で指定します。

コードのコンパイル方法

引数リストから引数を省略する前に、対応するパラメータが省略可能であること、また、プロシージャがそのパラメータに既定値を使用することを確認してください。

引数を名前で渡す場合、引数リスト内の名前が、宣言されたパラメータ名と完全に一致する必要があります。

参照

処理手順

方法 : 省略可能なパラメータをプロシージャに定義する
方法 : 省略可能なパラメータが使用されているかどうかを確認する

関連項目

Optional (Visual Basic)
ParamArray

概念

プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
位置と名前による引数渡し
省略可能なパラメータ
パラメータ配列
プロシージャのオーバーロード

方法：省略可能なパラメータが使用されているかどうかを確認する

プロシージャが省略可能なパラメータを定義する場合、呼び出し元のコードがそれらに対応する引数を渡しているかどうかを確認する必要がある場合があります。

パラメータの値が既定値に等しい場合は、次のいずれかの状況です。

- 呼び出し元のコードがプロシージャの呼び出しで引数を省略した
- 呼び出し元のコードがパラメータの既定値に等しい値を引数として渡した

プロシージャ コードは、この 2 つの状況を区別できません。これが問題にならない場合も多くありますが、この2 つのケースでプロシージャが別々のアクションを実行する場合も考えられます。この対処法としては、あり得ない値を既定値として設定する方法が考えられますが、呼び出し元のコードが絶対にこの値を渡さないという保証はありません。

呼び出し元のプログラムが省略可能な引数を渡しているということを確実にする必要がある場合は、プロシージャのオーバーロードされたバージョンを定義するのが最も安全です。「方法 : プロシージャの複数のバージョンを定義する」と「プロシージャのオーバーロードに関する注意事項」を参照してください。

省略可能なパラメータに引数が渡されたかどうかを確認するには

1. パラメータの既定値に、あり得ない値を定義する
2. 省略可能なパラメータが **String** などの参照型の場合は、**Nothing** を既定値として使用できます。ただし、**Nothing** が引数の値として使用されることが予想される場合を除きます。
3. プロシージャ コード内で、パラメータを既定値と比較して適切なアクションを実行します。

省略可能なパラメータのあるものとないものでプロシージャをオーバーロードする

省略可能なパラメータを持つプロシージャを定義するには、オーバーロードを使用する方法もあります。省略可能なパラメータが 1 つある場合は、パラメータがあるバージョンとないバージョンの、2 つのオーバーロードされたバージョンのプロシージャを定義できます。この方法は、省略可能なパラメータの数が増えるにつれて複雑になります。しかし、それぞれの省略可能な引数が呼び出しプログラムによって指定されているかどうかを確実に把握できるという利点があります。

引数の有無を確認するためにプロシージャの別バージョンを定義する

1. 引数リスト内のパラメータを含むバージョンのプロシージャを定義します。パラメータは **Optional** として宣言しません。
2. もう 1 つのバージョンはパラメータなしで定義します。宣言の他の部分は 1 つ目のバージョンと同一にします。
3. プロシージャの各バージョンに、呼び出しの種類に適したコードを配置します。

使用例

次のプロシージャでは、省略可能なパラメータ `office` を定義し、既定値 `QJZ` をテストし、呼び出しでこのパラメータが省略されているかどうかを確認します。

VB

```
Sub notify(ByVal company As String, Optional ByVal office As String = "QJZ")
    If office = "QJZ" Then
        Debug.WriteLine("office not supplied -- using Headquarters")
        office = "Headquarters"
    End If
    ' Insert code to notify headquarters or specified office.
End Sub
```

呼び出し元のコードが引数リスト内で `office` に値を渡していない場合、Visual Basic は既定値 "QJZ" を返します。

省略可能なパラメータが **String** などの参照型の場合は、Nothing (Visual Basic) を既定値として使用できます。ただし、Nothing (Visual Basic) が引数の値として使用されることが予想される場合を除きます。

オーバーロードを使用して、省略可能なパラメータが渡されているかどうかを確認する例については、「方法 : 省略可能なパラメータを受け取るプロシージャをオーバーロードする」を参照してください。

参照

処理手順

方法 : 省略可能なパラメータをプロシージャに定義する

方法 : 省略可能なパラメータを受け取るプロシージャを呼び出す

関連項目

Optional (Visual Basic)

ParamArray

概念

プロシージャのパラメータと引数

引数の値渡しおよび参照渡し

位置と名前による引数渡し

省略可能なパラメータ

パラメータ配列

プロシージャのオーバーロード

パラメータ配列

通常は、プロシージャ宣言で指定されているより多くの引数を使ってプロシージャを呼び出すことはできません。不特定多数の引数が必要な場合は、パラメータ配列を宣言すると、値の配列をプロシージャのパラメータとして渡すことができます。プロシージャを定義するときには、パラメータ配列の要素の数がわからなくてもかまいません。配列のサイズは、プロシージャの呼び出しごとに個別に決定されます。

ParamArray を宣言する

パラメータ リストでパラメータ配列を指定するには、ParamArray キーワードを使用します。次の規則が適用されます。

- プロシージャはパラメータ配列を 1 つだけ定義でき、これはプロシージャの定義の最後のパラメータである必要があります。
- パラメータ配列は値渡しで渡す必要があります。プロシージャ定義で ByVal キーワードを使って明示的に指定することをお勧めします。
- パラメータ配列は自動的に省略可能になります。既定値は、パラメータ配列の要素型の空の 1 次元配列です。
- パラメータ配列より前には、必須のパラメータだけを指定します。省略可能なパラメータはパラメータ配列だけであることが必要です。

詳細については、「方法 : 不特定数のパラメータを受け取るプロシージャを定義する」を参照してください。

ParamArray を呼び出す

パラメータ配列を定義するプロシージャを呼び出す場合、引数は次のいずれかの方法で渡します。

- なし。ParamArray 引数は省略できます。この場合は、空の配列がプロシージャに渡されます。Nothing (Visual Basic) キーワードを渡しても同じ結果になります。
- コンマで区切った任意の数の引数のリスト。各引数のデータ型は、暗黙的に **ParamArray** 要素型に変換できる必要があります。
- パラメータ配列と同じ要素型の配列

いずれの場合も、プロシージャ内のコードでは、各要素が **ParamArray** データ型と同じデータ型の 1 次元配列として、パラメータ配列を扱う必要があります。

詳細については、「方法 : 不特定数のパラメータを受け取るプロシージャを呼び出す」を参照してください。

セキュリティに関するメモ :
無限に増大する配列を扱う場合、アプリケーション内部の容量を超過してしまう可能性があります。パラメータ配列を受け取る場合は、呼び出し元のコードが渡した配列のサイズをテストする必要があります。このサイズがアプリケーションには大きすぎる場合、適切な手順を行う必要があります。詳細については、「方法 : 配列のサイズを決定する」を参照してください。

参照

関連項目

Optional (Visual Basic)
UBound 関数 (Visual Basic)

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
位置と名前による引数渡し
省略可能なパラメータ
プロシージャのオーバーロード
Visual Basic における型チェック

その他の技術情報

Visual Basic における配列

方法：不特定数のパラメータを受け取るプロシージャを定義する

プロシージャのパラメータ リスト内で最後のエントリには、パラメータ配列を宣言できます。これを宣言すると、プロシージャのパラメータに対して単一の値だけでなく、値のセットを渡せるようになります。プロシージャを定義するとき、このセットに含まれる値の数を決める必要はありません。プロシージャを呼び出すたびに、セットが新しく判断されます。また、呼び出しのたびに異なる数の値を渡すこともできます。

詳細については、「パラメータ配列」を参照してください。

最後のパラメータに不特定数の値を渡すことができるプロシージャを定義するには

1. プロシージャ宣言で、パラメータ リストを通常の方法で定義します。最後のパラメータを除くすべてのパラメータが必須であること (Optional (Visual Basic) ではないこと) が必要です。
2. 最後のパラメータ名の前に、キーワード **ByVal ParamArray** を指定します。このパラメータは自動的に省略可能になります。キーワード **Optional** を指定しないでください。
3. パラメータ配列の名前に続けて、空のかっこを記述します。
4. 空のかっこの後に、通常どおり **As** 句を定義します。
5. **As** 句の後に既定値を定義しないでください。パラメータ配列の既定値は、**As** 句に指定したデータを持つ、空の 1 次元配列に自動的に設定されます。

パラメータ配列の値の操作

プロシージャ内のコードでは、各要素が **ParamArray** データ型と同じデータ型の 1 次元配列として、パラメータ配列を扱う必要があります。

パラメータ配列の値の 1 つにアクセスするには

1. プロシージャコードで、パラメータ配列の名前を指定して UBound 関数 (Visual Basic) を呼び出し、パラメータ配列に渡す配列の長さを決定します。
2. プロシージャ コード内の実行可能なステートメントで、パラメータ配列の名前の後に添字をかっこで囲んで指定します。この添字は 0 ～上限 (**UBound** が返した値) の範囲の値にしてください。

セキュリティに関するメモ：
サイズ制限のない配列を扱うと、配列がアプリケーションの内部容量を超過する危険があります。呼び出し元のコードからパラメータ配列を受け取ったときは配列の長さを調べ、自分のアプリケーションに大きすぎると判断した場合は適切な方法で処理することが必要です。

使用例

パラメータ配列を受け取るプロシージャを定義し、パラメータ配列に渡されたすべての要素の値を出力する例を次に示します。

VB

```
Sub studentScores(ByVal name As String, ByVal ParamArray scores() As String)
    Debug.WriteLine("Scores for " & name & ":" & vbCrLf)
    ' Use UBound to determine largest subscript of the array.
    For i As Integer = 0 To UBound(scores, 1)
        Debug.WriteLine("Score " & i & ": " & scores(i))
    Next i
End Sub
```

`studentScores` の呼び出しは、たとえば次のようになります。

VB

```
Call studentScores("Anne", "10", "26", "32", "15", "22", "24", "16")
Call studentScores("Mary", "High", "Low", "Average", "High")
Dim JohnScores() As String = {"35", "Absent", "21", "30"}
Call studentScores("John", JohnScores)
```

コードのコンパイル方法

ParamArray がリスト内で最後のパラメータであること、および、それよりも前にあるどのパラメータにも **Optional** が宣言されていないことを確認してください。

参照

処理手順

方法 : 不特定数のパラメータを受け取るプロシージャを呼び出す

関連項目

ByVal

ParamArray

概念

プロシージャのパラメータと引数

引数の値渡しおよび参照渡し

位置と名前による引数渡し

省略可能なパラメータ

プロシージャのオーバーロード

Visual Basic における型チェック

その他の技術情報

Visual Basic における配列

方法：不特定数のパラメータを受け取るプロシージャを呼び出す

プロシージャのパラメータ リストには、最後のエントリとしてパラメータ配列を宣言できます。これにより、そのパラメータに対して単一の値だけでなく、不特定数の値を渡せるようになります。

詳細については、「パラメータ配列」を参照してください。

パラメータ配列を使用してプロシージャを呼び出し、対応する引数を省略するには

1. 通常の方法で、プロシージャ呼び出しを記述します。パラメータ配列は最後の引数として指定する必要があります。
2. 最後から 2 番目の引数を指定して、引数リストを終了します。パラメータ配列は省略できます。パラメータ配列よりも前にあるすべてのパラメータは、必須であることが必要です。

   または

   パラメータ配列の引数として、キーワード **Nothing** を指定します。
3. Visual Basic は、このパラメータ配列に対して空の 1 次元配列をプロシージャに渡します。

パラメータ配列を使ってプロシージャを呼び出し、引数のリストを指定するには

1. 通常の方法で、プロシージャ呼び出しを記述します。パラメータ配列は最後の引数として指定する必要があります。
2. パラメータ配列に格納する引数を、任意の数だけコンマで区切って指定します。各引数のデータ型は、暗黙的に **ParamArray** 要素型に変換できる必要があります。
3. Visual Basic は、指定されたすべての値を格納する 1 次元配列をプロシージャに渡します。

パラメータ配列を使ってプロシージャを呼び出し、引数の配列を指定するには

1. 通常の方法で、プロシージャ呼び出しを記述します。パラメータ配列は最後の引数として指定する必要があります。
2. パラメータ配列に対して、パラメータ配列の要素型と同じ要素型を持つ 1 次元配列を指定します。
3. 指定された配列を、Visual Basic がプロシージャに渡します。

使用例

「方法 : 不特定数のパラメータを受け取るプロシージャを定義する」で定義した `studentScores` プロシージャを呼び出す方法は、一般に次の例のようになります。

VB

```
    Call studentScores("George")
...    Call studentScores("Anne", "10", "26", "32", "15", "22", "24", "16")
    Call studentScores("Mary", "High", "Low", "Average", "High")
    Dim JohnScores() As String = {"35", "Absent", "21", "30"}
    Call studentScores("John", JohnScores)
```

最初の呼び出しでは、パラメータ配列が完全に省略され、最初の引数 (必須) だけが指定されています。 `studentScores` プロシージャは、この呼び出しが空の配列を渡しているものとして処理します。

2 番目と 3 番目の呼び出しには、長さの異なる引数リストがパラメータ配列に指定されています。このようなリストは、それぞれ値の配列として渡されます。

4 番目の呼び出しは、パラメータ配列に配列を渡しています。

参照

関連項目

Optional (Visual Basic)
ParamArray
ByVal
UBound 関数 (Visual Basic)

概念

再帰プロシージャ

再帰プロシージャとは、自分自身を呼び出すプロシージャです。一般的に、これは Visual Basic コード作成方法として効率的ではありません。

次のプロシージャは、再帰を使って元の引数の階乗を計算します。

VB

```
Function factorial(ByVal n As Integer) As Integer
    If n <= 1 Then
        Return 1
    Else
        Return factorial(n - 1) * n
    End If
End Function
```

再帰プロシージャの考慮事項

制限条件。再帰プロシージャでは、再帰を終了する条件を最低 1 つテストする必要があります。また、妥当な回数の再帰呼び出しを行ってもこの条件が満たされない場合の処理も必要です。必ず満たされる条件を最低 1 つ用意しないと、プロシージャが無限ループに陥る可能性が高くなります。

メモリ使用状況。アプリケーションがローカル変数に使用できる領域は限られています。プロシージャが自分自身を呼び出す際、ローカル変数のコピーが毎回作成され、領域を消費します。このプロセスがいつまでも続くと、最終的には StackOverflowException エラーが発生します。

効率。ほとんどの場合、再帰はループで代替できます。ループでは引数を渡したり、追加の領域を初期化したり、値を返したりするオーバーヘッドは発生しません。再帰呼び出しを使用しないほうが、パフォーマンスは大きく向上します。

相互再帰。2 つのプロシージャがお互いを呼び出すと、パフォーマンスが大きく低下したり、無限ループが発生したりします。このようなデザインでは、単独の再帰プロシージャと同じ問題が発生しますが、検出とデバッグはずっと困難です。

かっこを使った呼び出し。**Function** プロシージャを再帰的に呼び出すときには、引数リストがない場合でも、プロシージャ名の後にかっこを付ける必要があります。そうしないと、関数名が関数の戻り値を表していると見なされてしまいます。

テスト。再帰プロシージャを作成した場合、最低 1 つの制限条件を満たしていることを必ずテストする必要があります。また、再帰呼び出しが多すぎるためにメモリを使い果たすことがないことを確認する必要があります。

参照

処理手順

プロシージャのトラブルシューティング
例外のトラブルシューティング : System.StackOverflowException

関連項目

StackOverflowException

概念

Visual Basic におけるプロシージャ
Sub プロシージャ
Function プロシージャ
Property プロシージャ
演算子プロシージャ
プロシージャのパラメータと引数
プロシージャのオーバーロード
ループ構造

プロシージャのオーバーロード

プロシージャのオーバーロードとは、名前が同じでパラメータ リストが異なる、複数のバージョンのプロシージャを定義することです。プロシージャをオーバーロードする目的は、互いに密接な関係にある複数のバージョンのプロシージャを、名前で区別せずに定義することにあります。バージョンごとに異なるパラメータ リストを使用することによって、これが可能になります。

オーバーロードの規則

プロシージャをオーバーロードする場合は、次のような規則があります。

- **同じ名前**。オーバーロードされた各バージョンは、同じプロシージャ名を使用する必要があります。
- **別のシグネチャ**。オーバーロードされた各バージョンは、次のうちの少なくとも 1 つが他のすべてのバージョンと異なっている必要があります。
    - パラメータの数
    - パラメータの順序
    - パラメータのデータ型
    - 型パラメータの数 (ジェネリック プロシージャの場合)
    - 戻り値の型 (変換演算子のみ)

  プロシージャ名と共に、上記の項目はまとめて、プロシージャのシグネチャと呼ばれます。オーバーロードされたプロシージャを呼び出す場合、コンパイラはシグネチャを使って、呼び出しが定義と一致しているかどうかを確認します。
- **シグネチャに含まれない項目**。プロシージャをオーバーロードするには、シグネチャを変更する必要があります。以下の要素を変更するだけでは、プロシージャをオーバーロードすることはできません。
    - **Public**、**Shared**、**Static** などのプロシージャ修飾子キーワード
    - パラメータまたは型パラメータの名前
    - 型パラメータの制約 (ジェネリック プロシージャの場合)
    - **ByRef** や **Optional** などのパラメータ修飾子キーワード
    - 値を返すかどうか
    - 戻り値のデータ型 (変換演算子以外)

  上記のリスト項目は、シグネチャの一部ではありません。これらを使って複数のオーバーロードされたバージョンを区別することはできませんが、シグネチャによって適切に差別化されたオーバーロードされたバージョンのそれぞれで、これらの要素が異なっていてもかまいません。
- **遅延バインディングの引数**。遅延バインディング オブジェクト変数をオーバーロードされたバージョンに渡す場合は、適切なパラメータを Object として宣言する必要があります。

プロシージャの複数のバージョン

たとえば、顧客の残高に対してトランザクションをポストする **Sub** プロシージャを記述する場合に、名前と口座番号のどちらでも顧客を参照できるようにするとします。これを行うには、次のように、2 つの異なる **Sub** プロシージャを定義する方法があります。

VB

```
Sub postName(ByVal custName As String, ByVal amount As Single)
    ' Insert code to access customer record by customer name.
End Sub
Sub postAcct(ByVal custAcct As Integer, ByVal amount As Single)
    ' Insert code to access customer record by account number.
End Sub
```

オーバーロードされたバージョン

この他に、1 つのプロシージャ名をオーバーロードする方法もあります。Overloads キーワードを使って、次のように、各パラメータ リストごとにプロシージャのバージョンを定義します。

VB

```
Overloads Sub post(ByVal custName As String, ByVal amount As Single)
    ' Insert code to access customer record by customer name.
End Sub
Overloads Sub post(ByVal custAcct As Integer, ByVal amount As Single)
    ' Insert code to access customer record by account number.
End Sub
```

追加のオーバーロード

トランザクションの額を **Decimal** と **Single** のどちらで受け取ることもできるようにするには、`post` をさらにオーバーロードします。上の例のオーバーロードされた各バージョンをさらにオーバーロードすると、名前が同じでシグネチャがそれぞれ異なる **Sub** プロシージャが 4 つできることになります。

オーバーロードの利点

プロシージャのオーバーロードの利点は、呼び出しの柔軟性にあります。上の例で宣言した `post` プロシージャを使用する場合、呼び出し元のコードでは、顧客の ID を **String** として取得することも **Integer** として取得することもできます。いずれの場合も、呼び出すプロシージャは同じです。次に例を示します。

VB

```
Imports MSVB = Microsoft.VisualBasic
```

VB

```
Dim customer As String
Dim accountNum As Integer
Dim amount As Single
customer = MSVB.Interaction.InputBox("Enter customer name or number")
amount = MSVB.Interaction.InputBox("Enter transaction amount")
Try
    accountNum = CInt(customer)
    Call post(accountNum, amount)
Catch
    Call post(customer, amount)
End Try
```

参照

処理手順

方法 : プロシージャの複数のバージョンを定義する
方法 : オーバーロードされたプロシージャを呼び出す
方法 : 省略可能なパラメータを受け取るプロシージャをオーバーロードする
方法 : 不特定数のパラメータを受け取るプロシージャをオーバーロードする

関連項目

Overloads

概念

Visual Basic におけるプロシージャ
プロシージャのオーバーロードに関する注意事項
オーバーロードの解決法
Visual Basic におけるジェネリック型

方法：プロシージャの複数のバージョンを定義する

プロシージャはオーバーロードすることによって、複数のバージョンを定義できます。オーバーロードでは各バージョンとも同じ名前を使用しますが、パラメータ リストはそれぞれ異なります。プロシージャをオーバーロードする目的は、互いに密接な関係にある複数のバージョンのプロシージャを、名前で区別せずに定義することにあります。

詳細については、「プロシージャのオーバーロード」を参照してください。

プロシージャの複数のバージョンを定義するには

1. 定義するプロシージャの各バージョンに、**Sub** 宣言ステートメントまたは **Function** 宣言ステートメントを記述します。すべての宣言に同じプロシージャ名を使います。
2. 各宣言の **Sub** キーワードまたは **Function** キーワードの前に、Overloads キーワードを指定します。宣言内の **Overloads** の指定は省略可能ですが、いずれかの宣言に指定した場合、すべての宣言に指定することが必要になります。
3. 各宣言ステートメントの後に、特定のケースを処理するプロシージャ コードを記述します。呼び出し元のコードで指定される引数が、そのバージョンのパラメータ リストと一致するようにしてください。呼び出し元のコードでどのパラメータが指定されたかを調べるコードは必要ありません。Visual Basic によって、一致するプロシージャのバージョンに制御が渡されます。
4. プロシージャの各バージョンを **End Sub** ステートメントまたは **End Function** ステートメントで適切に終了します。

使用例

Sub プロシージャを定義し、顧客の残高に対してトランザクションをポストする例を次に示します。**Overloads** キーワードを指定して、プロシージャのバージョンを 2 つ定義しています。1 つは顧客を名前で受け取り、もう 1 つは口座番号で受け取ります。

VB

```
Overloads Sub post(ByVal custName As String, ByVal amount As Single)
    ' Insert code to access customer record by customer name.
End Sub
Overloads Sub post(ByVal custAcct As Integer, ByVal amount As Single)
    ' Insert code to access customer record by account number.
End Sub
```

呼び出し元のコードでは、顧客 ID を文字列 (**String**) または数値 (**Integer**) で取得しますが、どちらの場合でも同じ呼び出しステートメントを使用できます。

これらのバージョンの `post` プロシージャを呼び出す方法については、「方法 : オーバーロードされたプロシージャを呼び出す」を参照してください。

コードのコンパイル方法

オーバーロードされた各バージョンは、プロシージャ名は同じでもパラメータ リストが違う点に注意してください。

参照

処理手順

プロシージャのトラブルシューティング
方法 : 省略可能なパラメータを受け取るプロシージャをオーバーロードする
方法 : 不特定数のパラメータを受け取るプロシージャをオーバーロードする

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
プロシージャのオーバーロードに関する注意事項
オーバーロードの解決法

方法：オーバーロードされたプロシージャを呼び出す

プロシージャのオーバーロードの利点は、呼び出しの柔軟性にあります。呼び出しを行うコードは、どの引数を渡す場合でも、プロシージャに渡すために必要な情報を取得し、単一のプロシージャ名で呼び出すことができます。

複数の形式が定義されたプロシージャを呼び出すには

1. 呼び出しを行うコードで、プロシージャにどのデータを渡すかを決定します。
2. プロシージャ呼び出しを通常の方法で作成し、引数リストにデータを定義します。プロシージャに定義されている、いずれかの形式のパラメータ リストと一致するように、引数を指定してください。
3. どの形式のプロシージャを呼び出すかを判断する必要はありません。Visual Basic によって、引数リストと一致する形式に制御が渡されます。

   方法 : プロシージャの複数のバージョンを定義する で宣言した `post` プロシージャを呼び出す例を次に示します。顧客 ID を取得して、それが文字列型 (**String**) か整数型 (**Integer**) かを判断し、どちらの場合でも同じプロシージャを呼び出します。

   VB

   ```
   Imports MSVB = Microsoft.VisualBasic
   ```

   VB

   ```
   Dim customer As String
   Dim accountNum As Integer
   Dim amount As Single
   customer = MSVB.Interaction.InputBox("Enter customer name or number")
   amount = MSVB.Interaction.InputBox("Enter transaction amount")
   Try
       accountNum = CInt(customer)
       Call post(accountNum, amount)
   Catch
       Call post(customer, amount)
   End Try
   ```

参照

処理手順

プロシージャのトラブルシューティング
方法 : プロシージャの複数のバージョンを定義する
方法 : 省略可能なパラメータを受け取るプロシージャをオーバーロードする
方法 : 不特定数のパラメータを受け取るプロシージャをオーバーロードする

関連項目

Overloads

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
プロシージャのオーバーロード
プロシージャのオーバーロードに関する注意事項
オーバーロードの解決法

方法：省略可能なパラメータを受け取るプロシージャをオーバーロードする

プロシージャに 1 つ以上の Optional (Visual Basic) パラメータがある場合、いずれかの暗黙のオーバーロードに対応するオーバーロードされたバージョンを定義することはできません。詳細については、「プロシージャのオーバーロードに関する注意事項」の「省略可能なパラメータの暗黙のオーバーロード」を参照してください。

省略可能なパラメータが 1 つの場合

省略可能なパラメータを 1 つ取るプロシージャをオーバーロードするには

1. パラメータ リストに省略可能なパラメータを含む **Sub** または **Function** の宣言ステートメントを記述します。このオーバーロードされたバージョンでは **Optional** キーワードを使用しないでください。
2. **Sub** または **Function** キーワードの前に Overloads キーワードを指定します。
3. 呼び出し元のコードが省略可能な引数を渡してきた場合に実行するプロシージャ コードを記述します。
4. 状況に応じて **End Sub** ステートメントか **End Function** ステートメントでプロシージャを終了します。
5. 2 つ目の宣言ステートメントは、パラメータ リストに省略可能なパラメータを含めず、その他は最初の宣言と同じように記述します。
6. 呼び出し元のコードが省略可能な引数を渡さなかった場合に実行するプロシージャ コードを記述します。状況に応じて **End Sub** ステートメントか **End Function** ステートメントでプロシージャを終了します。

   次の例では、省略可能なパラメータを使用するよう定義されたプロシージャ、オーバーロードされた同等の 2 つのプロシージャ、そして最後に、無効なオーバーロードされたバージョンと有効なオーバーロードされたバージョンを示します。

   VB

```
Sub q(ByVal b As Byte, Optional ByVal j As Long = 6)
```

   VB

```
' The preceding definition is equivalent to the following two overloads.
' Overloads Sub q(ByVal b As Byte)
' Overloads Sub q(ByVal b As Byte, ByVal j As Long)
```

   VB

```
' Therefore, the following overload is not valid because the signature is already in u
se.
' Overloads Sub q(ByVal c As Byte, ByVal k As Long)
' The following overload uses a different signature and is valid.
Overloads Sub q(ByVal b As Byte, ByVal j As Long, ByVal s As Single)
```

省略可能なパラメータが複数ある場合

省略可能なパラメータが複数あるパラメータの場合、通常は 2 つ以上のオーバーロードされたバージョンが必要です。たとえば、省略可能なパラメータが 2 つあり、呼び出し元のコードがいずれかのパラメータを渡すか渡さないかを、もう 1 つのパラメータの有無に関係なく決める場合、オーバーロードされたバージョンは 4 つ (渡される引数の組み合わせの数) 必要です。

省略可能なパラメータの数が増えると、オーバーロードはより複雑になります。渡される引数の組み合わせの中にあり得ないものはないとすると、省略可能なパラメータの数が *N* のとき、オーバーロードされたバージョンは *2 ^ N* 個必要です。オーバーロードされたバージョンをすべて定義する意味があるかどうかは、プロシージャの性質によって判断します。

省略可能なパラメータを複数取るプロシージャをオーバーロードするには

1. 省略可能な引数を、どのような組み合わせで受け取ることができるかをプロシージャのロジックから判断します。1 つの省略可能なパラメータが、他の省略可能なパラメータに依存している場合、受け入れられない組み合わせが出てくることがあります。たとえば、配偶者の名前

を受け取るパラメータと、配偶者の年齢を受け取るパラメータがある場合、年齢は含まれるが名前が含まれない組み合わせは受け入れられません。

2. 省略可能な引数の、受け入れ可能な組み合わせそれぞれに対し、対応するパラメータ リストを定義する **Sub** または **Function** 宣言ステートメントを記述します。**Optional** キーワードは使用しないでください。

3. 各定義では、**Sub** または **Function** キーワードの前に Overloads キーワードを指定します。

4. 各宣言に続いて、呼び出し元のコードが、宣言のパラメータ リストに対応する引数のリストを渡してきた場合に実行するプロシージャ コードを記述します。

5. 状況に応じて **End Sub** ステートメントか **End Function** ステートメントで各プロシージャを終了します。

参照

処理手順

プロシージャのトラブルシューティング
方法 : プロシージャの複数のバージョンを定義する
方法 : オーバーロードされたプロシージャを呼び出す
方法 : 不特定数のパラメータを受け取るプロシージャをオーバーロードする

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
省略可能なパラメータ
パラメータ配列
プロシージャのオーバーロード
オーバーロードの解決法

方法：不特定数のパラメータを受け取るプロシージャをオーバーロードする

プロシージャに ParamArray パラメータがある場合、パラメータ配列として 1 次元配列を使用するオーバーロードされたバージョンを定義することはできません。詳細については、「プロシージャのオーバーロードに関する注意事項」の「パラメータ ParamArray の暗黙のオーバーロード」を参照してください。

可変数のパラメータを持つプロシージャをオーバーロードするには

1. プロシージャと呼び出し元のコード ロジックが、**ParamArray** パラメータよりもオーバーロードされたバージョンを有効に活用していることを確認します。「プロシージャのオーバーロードに関する注意事項」の「オーバーロードと ParamArray」を参照してください。
2. パラメータ リストの可変部分で、プロシージャが受け取る値の数を確認します。これには、値がない場合も、単独の 1 次元配列の場合も含まれます。
3. 受け入れ可能な値のそれぞれの数に対し、対応するパラメータ リストを定義する **Sub** または **Function** 宣言ステートメントを記述します。このオーバーロードされたバージョンでは **Optional** または **ParamArray** キーワードを使用しないでください。
4. 各定義では、**Sub** または **Function** キーワードの前に Overloads キーワードを指定します。
5. 各宣言に続いて、呼び出し元のコードが、宣言のパラメータ リストに対応する値を渡してきた場合に実行するプロシージャ コードを記述します。
6. 状況に応じて **End Sub** ステートメントか **End Function** ステートメントで各プロシージャを終了します。

使用例

次の例では、ParamArray パラメータで定義されたプロシージャを示し、これに相当する一連のオーバーロードされたプロシージャを示します。

VB

```
Sub p(ByVal d As Date, ByVal ParamArray c() As Char)
```

VB

```
' The preceding definition is equivalent to the following overloads.
' Overloads Sub p(ByVal d As Date)
' Overloads Sub p(ByVal d As Date, ByVal c() As Char)
' Overloads Sub p(ByVal d As Date, ByVal c1 As Char)
' Overloads Sub p(ByVal d As Date, ByVal c1 As Char, ByVal c2 As Char)
' And so on, with an additional Char argument in each successive overload.
```

このようなプロシージャは、パラメータ配列として 1 次元配列を受け取るパラメータ リストではオーバーロードできません。ただし、他の暗黙のオーバーロードのシグネチャは使用できます。次の宣言はこのことを示しています。

VB

```
' The following overload is not valid because it takes an array for the parameter array.
' Overloads Sub p(ByVal x As Date, ByVal y() As Char)
' The following overload takes a single value for the parameter array and is valid.
Overloads Sub p(ByVal z As Date, ByVal w As Char)
```

オーバーロードされたバージョンのコードでは、呼び出し元のコードが **ParamArray** パラメータに 1 つ以上の値を渡しているか、また、いくつ渡しているかをテストする必要はありません。Visual Basic は呼び出し元の引数リストに一致するバージョンにコントロールを渡します。

コードのコンパイル方法

ParamArray パラメータを持つプロシージャは、一連のオーバーロードされたバージョンと同等なので、このプロシージャを、これらの暗黙のオーバーロードのいずれかに対応するパラメータ リストでオーバーロードすることはできません。詳細については、「プロシージャのオーバーロードに関する注意事項」を参照してください。

セキュリティ

無限に増大する配列を扱う場合、アプリケーション内部の容量を超過してしまう可能性があります。パラメータの配列を受け取る場合、呼び出

し元のコードが渡す配列の長さをテストし、アプリケーションに対して大きすぎるようであれば、適切な手順を実行してください。

参照

処理手順

プロシージャのトラブルシューティング
方法 : プロシージャの複数のバージョンを定義する
方法 : オーバーロードされたプロシージャを呼び出す
方法 : 省略可能なパラメータを受け取るプロシージャをオーバーロードする

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
省略可能なパラメータ
パラメータ配列
プロシージャのオーバーロード
オーバーロードの解決法

プロシージャのオーバーロードに関する注意事項

プロシージャをオーバーロードする場合、オーバーロードされる各バージョンに別々のシグネチャを使用する必要があります。通常は、各バージョンに別々のパラメータ リストを指定します。詳細については、「プロシージャのオーバーロード」の「別のシグネチャ」を参照してください。

シグネチャが異なる場合は、**Function** プロシージャと **Sub** プロシージャとの間のオーバーロードも可能です。戻り値を持つか持たないかという点だけが異なる 2 つのオーバーロードが共存することはできません。

プロパティは、プロシージャと同じようにオーバーロードでき、同じ制約を受けます。ただし、プロシージャによるプロパティのオーバーロード、またはその逆のオーバーロードを行うことはできません。

オーバーロードされたバージョンの代替手段

特に、引数が省略可能な場合や引数の数が可変である場合、オーバーロードされたバージョンの代替手段を利用できる場合があります。

ただし、省略可能な引数はすべての言語でサポートされているとは限りません。また、パラメータの配列は Visual Basic のみに限定されています。他のいくつかの言語で書かれているコードから呼び出されるプロシージャを作成する場合、オーバーロードされたバージョンによって柔軟性が向上します。

オーバーロードと省略可能な引数

呼び出し元のコードが 1 つ以上の引数を指定または省略できる場合は、複数のオーバーロードされたバージョンを定義するか、省略可能なパラメータを使用できます。

オーバーロードされたバージョンを使用する場合

オーバーロードされたバージョンを定義するのは、次のような場合です。

- 呼び出し元のコードが省略可能な引数を使用するかどうかによって、プロシージャ コード内のロジックが大きく異なる
- プロシージャのコードが、呼び出し元のコードが省略可能な引数を使っているかどうかを確認する有効な手段がないこれは、呼び出し元のコードからの提供が期待できない引数に、既定値として使用できる値がない場合などです。

省略可能なパラメータを使用する場合

1 つ以上の省略可能なパラメータを使用すると有効なのは、次のような場合です。

- 呼び出し元のコードが省略可能な引数を使用していない場合に、パラメータを既定値に設定するこの場合、1 つ以上の **Optional** パラメータで 1 つのバージョンを定義すると、プロシージャのコードを単純化できます。

詳細については、「省略可能なパラメータ」を参照してください。

オーバーロードと ParamArray

呼び出し元のコードが使用する引数の数が可変の場合、複数のオーバーロードされたバージョンを定義するか、パラメータの配列を使用します。

オーバーロードされたバージョンを使用する場合

オーバーロードされたバージョンを定義するのは、次のような場合です。

- 呼び出し元のコードがパラメータの配列に格納する値の数が、少ないことが明らかな場合
- プロシージャのコード内のロジックが、呼び出し元のコードから提供される値の数によって大きく異なる
- 呼び出し元のコードから、データ型の異なる値が提供されることがある

パラメータの配列を使用する場合

次の場合は、**ParamArray** を使用するのが適しています。

- 呼び出し元のコードからパラメータの配列に提供される値の数が予測できず、多数になる可能性がある場合
- プロシージャのロジックが、呼び出し元のコードから提供されたすべての値を反復処理し、すべての値にほぼ同じ操作を行う場合

詳細については、「パラメータ配列」を参照してください。

省略可能なパラメータの暗黙のオーバーロード

Optional (Visual Basic) パラメータを持つプロシージャは、省略可能なパラメータを持つバージョンと持たないバージョンの 2 つのオーバーロード

されたプロシージャと同じです。このようなプロシージャは、これらのいずれかに対応するパラメータのリストではオーバーロードできません。これを表す宣言を次に示します。

VB

```
Overloads Sub q(ByVal b As Byte, Optional ByVal j As Long = 6)
```

VB

```
' The preceding definition is equivalent to the following two overloads.
' Overloads Sub q(ByVal b As Byte)
' Overloads Sub q(ByVal b As Byte, ByVal j As Long)
```

VB

```
' Therefore, the following overload is not valid because the signature is already in use.
' Overloads Sub q(ByVal c As Byte, ByVal k As Long)
' The following overload uses a different signature and is valid.
Overloads Sub q(ByVal b As Byte, ByVal j As Long, ByVal s As Single)
```

省略可能なパラメータが複数あるプロシージャでは、上の例と同様の理由で、暗黙のオーバーロードがいくつかあることになります。

パラメータ ParamArray の暗黙のオーバーロード

コンパイラは、ParamArray パラメータを持つプロシージャが、パラメータ配列に渡される内容によって異なる無限の数のオーバーロードを持つと見なします。次に示すのは、それらのオーバーロードを大別したものです。

- 呼び出し元のコードが **ParamArray** に引数を渡さない場合のオーバーロード
- 呼び出し元のコードが **ParamArray** 要素型の 1 次元配列を渡す場合のオーバーロード
- 呼び出し元のコードが **ParamArray** 要素型の引数をその数だけ渡す場合のオーバーロード (すべての正の整数に対して 1 つ)

次の宣言は、これらの暗黙のオーバーロードを示しています。

VB

```
Overloads Sub p(ByVal d As Date, ByVal ParamArray c() As Char)
```

VB

```
' The preceding definition is equivalent to the following overloads.
' Overloads Sub p(ByVal d As Date)
' Overloads Sub p(ByVal d As Date, ByVal c() As Char)
' Overloads Sub p(ByVal d As Date, ByVal c1 As Char)
' Overloads Sub p(ByVal d As Date, ByVal c1 As Char, ByVal c2 As Char)
' And so on, with an additional Char argument in each successive overload.
```

このようなプロシージャは、パラメータ配列として 1 次元配列を受け取るパラメータ リストではオーバーロードできません。ただし、他の暗黙のオーバーロードのシグネチャは使用できます。次の宣言はこのことを示しています。

VB

```
' The following overload is not valid because it takes an array for the parameter array.
' Overloads Sub p(ByVal x As Date, ByVal y() As Char)
' The following overload takes a single value for the parameter array and is valid.
Overloads Sub p(ByVal z As Date, ByVal w As Char)
```

オーバーロードの代わりとしての型宣言を省略したプログラミング

呼び出し元のコードが、パラメータに異なるデータ型を渡せるようにするには、Visual Basic での型宣言を省略したプログラミングというアプローチ

を利用します。Option Strict ステートメントまたは /optionstrict のいずれかのコンパイラ オプションを使用すると、型チェックのスイッチを **Off** にできます。これで、パラメータのデータ型を宣言する必要がなくなります。ただし、この方法には、オーバーロードとは違って次のような問題があります。

- 型宣言を省略したプログラミングでは、実行コードが非効率的になります。
- 渡される可能性があるすべてのデータ型をプロシージャでテストする必要があります。
- プロシージャがサポートしていないデータ型を呼び出し元のコードが渡しても、コンパイラでエラーが生成されません。

参照

処理手順

プロシージャのトラブルシューティング
方法 : プロシージャの複数のバージョンを定義する
方法 : オーバーロードされたプロシージャを呼び出す
方法 : 省略可能なパラメータを受け取るプロシージャをオーバーロードする
方法 : 不特定数のパラメータを受け取るプロシージャをオーバーロードする

関連項目

Overloads

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
オーバーロードの解決法

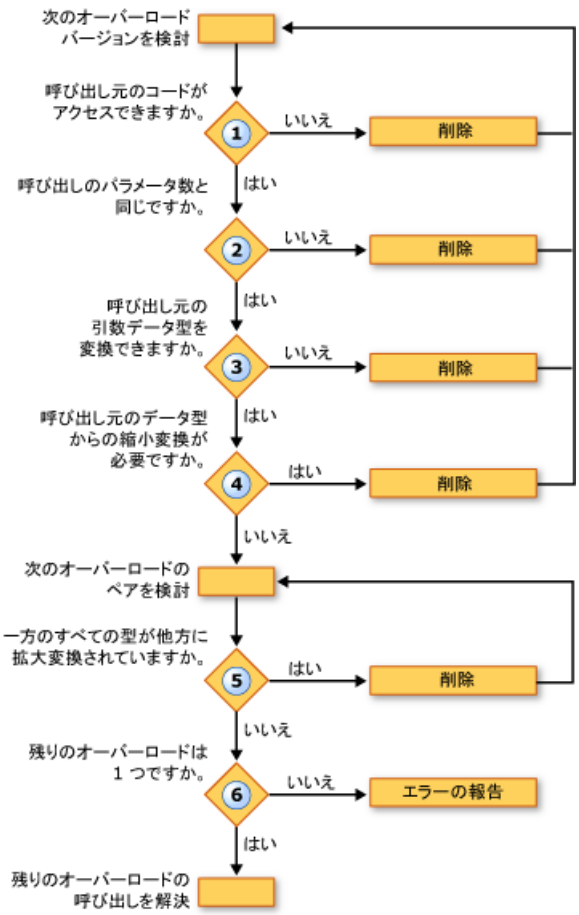
オーバーロードの解決法

オーバーロードされたバージョンが複数定義されているプロシージャが呼び出された場合、Visual Basic のコンパイラは、どのオーバーロードを呼び出すのかを判断する必要があります。この判断は、次の手順で行われます。

1. **アクセシビリティ。** 呼び出し元のコードから呼び出すことができないアクセス レベルを持つオーバーロードを除外します。
2. **パラメータの数。** 呼び出しで指定されている数と異なる数のパラメータが定義されているオーバーロードを除外します。
3. **パラメータのデータ型。** 呼び出し元の引数のデータ型を定義されているパラメータ型に変換できないオーバーロードを除外します。
4. **縮小変換。** 呼び出し元の引数の型から定義されているパラメータの型への下位変換が必要なオーバーロードを除外します。これは型チェックのスイッチ (Option Strict ステートメント) が **On** でも **Off** でも同じです。
5. **最小の拡大。** 残りのオーバーロードをペアと見なします。各ペアごとに、定義されているパラメータのデータ型を比較します。一方のオーバーロードのすべての型がもう一方のオーバーロードの対応する型に上位変換される場合は、後者を除外します。つまり、上位変換が最も少なくて済むオーバーロードが残されます。
6. **1 つの候補。** 残りのオーバーロードが 1 つだけになるまでこの比較を続け、残った 1 つのオーバーロードへの呼び出しを解決します。コンパイラがオーバーロードを 1 つに絞りきれない場合は、エラーが発生します。

次の図は、オーバーロードされたバージョンの、どのセットを呼び出すかを判断するプロセスを示しています。

オーバーロードされたバージョンの解決



次の例は、上で説明したオーバーロード解決のプロセスを示しています。

VB

```
Overloads Sub z(ByVal x As Byte, ByVal y As Double)
End Sub
Overloads Sub z(ByVal x As Short, ByVal y As Single)
End Sub
Overloads Sub z(ByVal x As Integer, ByVal y As Single)
End Sub
```

VB

```
Dim r, s As Short
Call z(r, s)
Dim p As Byte, q As Short
' The following statement causes an overload resolution error.
Call z(p, q)
```

1 回目の呼び出しでは、コンパイラは最初のオーバーロードを除外します。1 つ目の引数の型 (**Short**) が、対応するパラメータの型 (**Byte**) よりも大きいからです。次に、3 番目のオーバーロードを除去します。2 番目のオーバーロードの各引数型 (**Short** と **Single**) よりも、3 番目のオーバーロードで対応する引数型 (**Integer** と **Single**) の方が大きいからです。2 番目のオーバーロードの方が拡大変換が少なくて済むため、コンパイラは 2 番目を使用して呼び出します。

2 回目の呼び出しでは、コンパイラは引数の型が大きすぎるという理由でオーバーロードを除外できません。3 番目のオーバーロードは、1 回目の呼び出しのときと同じ理由で除外されます。2 番目のオーバーロードを呼び出した方が引数の型の拡大変換が少なくて済むからです。しかし、コンパイラは 1 番目と 2 番目のオーバーロードのいずれかに解決できません。どちらにも、対応する型に拡大変換されるパラメータ型が 1 つ定義されています (**Byte** から **Short**、そして **Single** から **Double**)。このため、オーバーロード解決エラーが生成されます。

オーバーロードされた Optional 引数と ParamArray 引数

プロシージャの 2 つのオーバーロードのシグネチャが同じで、唯一の違いが最後のパラメータの宣言 (一方が Optional (Visual Basic) で、他方は ParamArray) である場合、コンパイラはそのプロシージャの呼び出しを次の方法で解決します。

呼び出しの最後の引数	コンパイラがオーバーロードの呼び出しを解決するときの最後の引数の宣言
値なし (引数を省略)	**Optional**
単一の値	**Optional**
コンマで区切られた複数の値のリスト	**ParamArray**
任意の長さの配列 (空の配列を含む)	**ParamArray**

参照

処理手順

プロシージャのトラブルシューティング
方法 : プロシージャの複数のバージョンを定義する
方法 : オーバーロードされたプロシージャを呼び出す
方法 : 省略可能なパラメータを受け取るプロシージャをオーバーロードする
方法 : 不特定数のパラメータを受け取るプロシージャをオーバーロードする

関連項目

Overloads

概念

省略可能なパラメータ
パラメータ配列
プロシージャのオーバーロード
プロシージャのオーバーロードに関する注意事項

プロシージャのトラブルシューティング

ここでは、プロシージャを使用する際に起きる一般的な問題についていくつか説明します。

Function プロシージャから配列型を返す

Function プロシージャが配列型を返す場合は、**Function** 名を使って配列の各要素に値を格納することはできません。格納しようとすると、コンパイラによって **Function** の呼び出しと解釈されます。たとえば、次のコードはコンパイル エラーになります。

```
Function allOnes(ByVal n As Integer) As Integer()
For i As Integer = 1 To n - 1
' The following statement generates a COMPILER ERROR .
allOnes(i) = 1
Next i
' The following statement generates a COMPILER ERROR .
Return allOnes()
End Function
```

`allOnes(i) = 1` ステートメントは、間違った型の引数 (**Integer** 配列ではなく単一の **Integer**) を指定して `allOnes` を呼び出していると解釈されるため、ここでコンパイル エラーが発生します。`Return allOnes()` ステートメントはコンパイル エラーを発生します。引数を指定せずに `allOnes` を呼び出していると解釈されるからです。

正しい方法 : 返される配列の要素を変更できるようにするには、内部配列をローカル変数として定義します。次の例はコンパイル エラーが発生しません。

VB

```
Function allOnes(ByVal n As Integer) As Integer()
    Dim i As Integer, iArray(n) As Integer
    For i = 0 To n - 1
        iArray(i) = 1
    Next i
    Return iArray
End Function
```

引数がプロシージャ呼び出しによって変更されない

呼び出しコードの引数の基になるプログラミング要素を、プロシージャで変更できるようにする場合は、参照渡しで渡す必要があります。ただし、値渡しで渡した場合でも、プロシージャはその参照型の引数の要素にアクセスできます。

- **基になる変数** 基になる変数の要素の値そのものをプロシージャで変更できるようにするには、プロシージャのパラメータを ByRef で宣言する必要があります。また、呼び出しコードで引数をかっこで囲んで記述しないでください。こうすると、**ByRef** で引数を渡す機能がオーバーライドされるからです。
- **型参照要素** パラメータを ByVal で宣言すると、基になる変数要素そのものをプロシージャで変更できなくなります。しかし、引数が参照型であれば、プロシージャは変数の値を置き換えることはできませんが、引数が指すオブジェクトのメンバを変更できます。たとえば、引数が配列変数であった場合、プロシージャは新しい配列を代入することはできませんが、その 1 つ以上の要素を変更できます。変更した要素は、呼び出し元のコードの基の配列変数に反映されます。

次の例には、配列変数を値渡しで受け取ってその要素を操作する 2 つのプロシージャが定義されています。`increase` プロシージャは、各要素に単純に 1 を加算します。`replace` プロシージャは、パラメータ `a()` に新しい配列を代入してから各要素に 1 を加算します。ただし、この新しい配列の代入は、呼び出し元のコードの基の配列変数には反映されません。なぜなら、`a()` が **ByVal** で宣言されているからです。

VB

```
Public Sub increase(ByVal a() As Long)
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

VB

```
Public Sub replace(ByVal a() As Long)
    Dim k() As Long = {100, 200, 300}
    a = k
    For j As Integer = 0 To UBound(a)
        a(j) = a(j) + 1
    Next j
End Sub
```

次に、`increase` と `replace` を呼び出す例を示します。

VB

```
Dim n() As Long = {10, 20, 30, 40}
Call increase(n)
MsgBox("After increase(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
Call replace(n)
MsgBox("After replace(n): " & CStr(n(0)) & ", " & _
    CStr(n(1)) & ", " & CStr(n(2)) & ", " & CStr(n(3)))
```

最初に **MsgBox** を呼び出すと、"After increase(n): 11, 21, 31, 41" が表示されます。`n` が参照型なので、**ByVal** で渡されていても `increase` はそのメンバを変更できます。

2 度目に **MsgBox** を呼び出すと、"After replace(n): 11, 21, 31, 41" が表示されます。`n` が **ByVal** で渡されたため、`replace` は変数 `n` を、新しい配列を代入するという方法で変更できません。`replace` が新しい配列インスタンス `k` を作成し、そこにローカル変数 `a` を代入すると、呼び出しコードによって渡された `n` への参照が失われます。`a` のメンバをインクリメントすると、ローカル変数 `k` だけに反映されます。

正しい方法：基になる変数要素そのものを変更可能にするには、参照渡しで渡します。`replace` の宣言を変更し、呼び出しコードで配列を別の配列に置き換えることができるようにする例を次に示します。

VB

```
Public Sub replace(ByRef a() As Long)
```

オーバーロードを定義できない

プロシージャのオーバーロードを定義する場合は、名前を同じにしてシグネチャを変える必要があります。シグネチャが同じで、コンパイラがプロシージャ宣言をオーバーロードと区別できない場合は、エラーが発生します。

プロシージャのシグネチャは、プロシージャ名とパラメータ リストによって決まります。各オーバーロードは名前がすべて同じである必要がありますが、シグネチャのその他のコンポーネントについては、少なくとも 1 つが他のいずれとも同じでないことが必要です。詳細については、「プロシージャのオーバーロード」を参照してください。

次の項目は、パラメータ リストに記述されますが、プロシージャのシグネチャのコンポーネントには含まれません。

- **Public**、**Shared**、**Static** などのプロシージャ修飾子キーワード
- パラメータ名
- **ByRef** や **Optional** などのパラメータ修飾子キーワード
- 戻り値のデータ型 (変換演算子の場合を除く)

これらの項目を変更するだけでは、プロシージャをオーバーロードすることはできません。

正しい方法：プロシージャのオーバーロードを定義するには、シグネチャを変える必要があります。名前は同じでなくてはならないため、パラメータの数、順序、またはデータ型を変えることが必要です。ジェネリック プロシージャの場合は、型パラメータの数を変更できます。変換演算子 (CType 関数) の場合は、戻り値の型を変更できます。

省略可能な引数やパラメータ配列の引数を持つオーバーロードの解決

Optional (Visual Basic) パラメータまたは ParamArray パラメータを持つプロシージャをオーバーロードする場合は、暗黙のオーバーロードを重複させないよう注意する必要があります。詳細については、「プロシージャのオーバーロードに関する注意事項」を参照してください。

オーバーロードされたプロシージャのうち、正しい形式を呼び出すことができない

プロシージャが複数の形式でオーバーロードされている場合は、それらすべてのパラメータ リストをよく確認することと、Visual Basic が複数のオーバーロードの中からどのように呼び出しを解決するかを理解することが必要です。そうしないと、意図した形式と異なるオーバーロードを呼び出す可能性があります。

どのオーバーロードを呼び出すか決定したら、次の規則に注意して従ってください。

- 正しい数の引数を、正しい順序で指定する。
- 最も望ましいのは、引数の型が対応するパラメータの型とまったく同じであることです。そうでない場合でも、各引数のデータ型は対応するパラメータの型よりも小さいことが必要です。これは Option Strict ステートメント に **Off** が設定されている場合でも同じです。オーバーロードが引数リストによって縮小変換する必要が生じれば、そのオーバーロードは呼び出しの対象から外されます。
- 拡大変換を必要とする引数を指定する場合は、対応するパラメータにデータ型をできるだけ近づけるようにしてください。引数のデータ型を許容するオーバーロードが 2 つ以上ある場合、コンパイラは拡大変換の量が最も少ないオーバーロードに呼び出しを解決します。

引数を指定するとき、変換キーワード CType 関数 を使用すると、データ型の不一致の可能性を減らすことができます。

オーバーロードの解決エラー

オーバーロードされたプロシージャを呼び出すと、コンパイラは 1 つを除くすべてのオーバーロードの除外を試みます。うまく除外できれば、呼び出しをそのオーバーロードに解決します。すべてのオーバーロードが除外された場合、またはオーバーロードの候補を 1 つに絞ることができない場合は、エラーが発生します。

次の例は、オーバーロード解決のプロセスを示しています。

VB

```
Overloads Sub z(ByVal x As Byte, ByVal y As Double)
End Sub
Overloads Sub z(ByVal x As Short, ByVal y As Single)
End Sub
Overloads Sub z(ByVal x As Integer, ByVal y As Single)
End Sub
```

VB

```
Dim r, s As Short
Call z(r, s)
Dim p As Byte, q As Short
' The following statement causes an overload resolution error.
Call z(p, q)
```

1 回目の呼び出しでは、コンパイラは最初のオーバーロードを除外します。1 つ目の引数の型 (**Short**) が、対応するパラメータの型 (**Byte**) よりも大きいからです。次に、3 番目のオーバーロードを除去します。2 番目のオーバーロードの各引数型 (**Short** と **Single**) よりも、3 番目のオーバーロードで対応する引数型 (**Integer** と **Single**) の方が大きいからです。2 番目のオーバーロードの方が拡大変換が少なくて済むため、コンパイラは 2 番目を使用して呼び出します。

2 回目の呼び出しでは、コンパイラは引数の型が大きすぎるという理由でオーバーロードを除外できません。3 番目のオーバーロードは、1 回目の呼び出しのときと同じ理由で除外されます。2 番目のオーバーロードを呼び出した方が引数の型の拡大変換が少なくて済むからです。しかし、コンパイラは 1 番目と 2 番目のオーバーロードのいずれかに解決できません。どちらにも、対応する型に拡大変換されるパラメータ型が 1 つ定義されています (**Byte** から **Short**、そして **Single** から **Double**)。このため、オーバーロード解決エラーが生成されます。

正しい方法 : オーバーロードされたプロシージャを明確に呼び出すには、CType 関数 を使用して引数のデータ型をパラメータの型に一致させます。次の例で呼び出される `z` は、必ず 2 番目のオーバーロードに解決します。

VB

```
Call z(CType(p, Short), CType(q, Single))
```

省略可能な引数やパラメータ配列の引数を持つオーバーロードの解決

プロシージャの 2 つのオーバーロードのシグネチャが同じで、唯一の違いが最後のパラメータの宣言 (一方が Optional (Visual Basic) で、他方は ParamArray) である場合、コンパイラはそのプロシージャの呼び出しを、より厳密に一致している方に解決します。詳細については、「オーバーロードの解決法」を参照してください。

参照

概念

Visual Basic における制御フロー

何も制御を加えないと、プログラムは最初から最後まで順番にステートメントを実行します。このように一定方向の流れだけで記述できる単純なプログラムもあります。しかし、プログラミング言語の能力と有用性のほとんどは、制御ステートメントやループを使った実行順序の変更の機能に由来します。

制御構造では、プログラムの実行の流れを制御できます。制御構造を使用すると、条件判断や繰り返し処理をする Visual Basic コードを記述できます。その他にも、リソースの破棄や、同じオブジェクト参照に対する一連のステートメントの実行を保証する制御構造があります。

このセクションの内容

条件判断構造

分岐に使う制御構造について説明します。

ループ構造

処理の繰り返しに使う制御構造について説明します。

その他の制御構造

リソースの破棄とオブジェクトへのアクセスに使う制御構造について説明します。

入れ子になった制御構造

他の制御構造の内部にある制御構造について説明します。

方法 : 制御構造から制御を移す

制御を直接制御構造から移す方法を説明します。

関連するセクション

制御ステートメント (Visual Basic 6.0 ユーザー向け)

制御構造の動作および構文で、以前のバージョンの Visual Basic から変更があった点の一覧を示します。

制御フローの概要

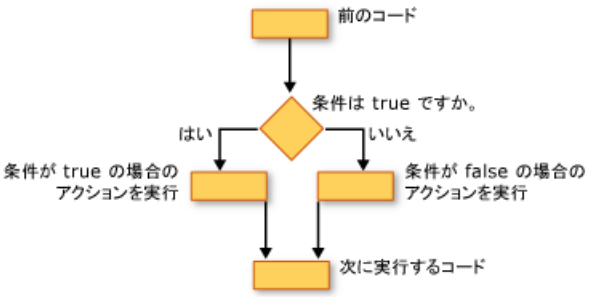
このテーマについての言語参照ページへのリンクです。

条件判断構造

Visual Basic では、条件を調べ、その結果に応じた処理を実行できます。条件のテスト結果は、真偽値、式のさまざまな値、または一連のステートメントを実行したときに生成されるさまざまな例外として得ることができます。

次の図は、条件が真かどうかをテストし、真か偽かによって異なる処理を実行する条件判断構造を示しています。

条件が真である場合と偽である場合とで異なる処理を実行



If...Then...Else 構造

If...Then...Else 構造では、1 つ以上の条件をテストし、各条件に応じて 1 つ以上のステートメントを実行できます。次の方法で条件をテストし、処理を実行できます。

- 条件が **True** の場合に、1 つ以上のステートメントを実行します。
- 条件が **False** の場合に、1 つ以上のステートメントを実行します。
- ある条件が **True** で、他の条件が **False** の場合に、いくつかのステートメントを実行します。
- 先行する条件が **False** の場合に、追加の条件をテストします。

これらのすべての機能を持つ制御構造が、If...Then...Else ステートメント (Visual Basic) です。1 つのテストおよび 1 つのステートメントだけを実行する場合は、単一行のバージョンを使用できます。条件およびアクションがより複雑な場合は、複数行のバージョンを使用します。

Select...Case 構造

Select...Case 構造では、式を 1 回評価し、想定されるさまざまな値に基づいて異なる一連のステートメントを実行できます。詳細については、「Select...Case ステートメント (Visual Basic)」を参照してください。

Try...Catch...Finally 構造

Try...Catch...Finally 構造では、何らかのステートメントで例外が発生した場合に、制御を継続できる環境で一連のステートメントを実行できます。さまざまな例外に応じて異なるアクションを実行できます。オプションで、どのような状況でも関係なく **Try...Catch...Finally** 構造全体を終了する前に必ず実行するコード ブロックを指定できます。詳細については、「Try...Catch...Finally ステートメント (Visual Basic)」を参照してください。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : 1 つ以上の条件に基づいて、ステートメントを実行する
方法 : 式を複数の値と比較する
方法 : エラー発生時に制御を継続する

概念

ループ構造
その他の制御構造
入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

方法：1 つ以上の条件に基づいて、ステートメントを実行する

If...Then...Else ステートメント (Visual Basic) を使うと、条件の ブール型 (Boolean) (Visual Basic) 値に基づいて特定のステートメントまたはステートメント ブロックを実行できます。条件は、通常、2 つの値の比較によって判断されますが、**Boolean** 値 (**True** または **False**) として評価できる式の場合どのような式でも指定できます。指定できる値には、数値型などのその他のデータ型を **Boolean** に変換した値も含まれます。

条件が True である場合に、1 つ以上のステートメントを実行する

- 実行するステートメントが 1 つだけある場合、**If**...**Then**...**Else** 構造の単一行の構文を使用します。**Else** または **End If** ステートメントは必要ありません。次に例を示します。

```
Sub fixDate()
    Dim myDate As Date = #2/13/1973#
    If myDate < Now Then myDate = Now
End Sub
```

または

- 条件が **True** の場合に実行するコードが複数行にわたる場合は、**End If** ステートメントを含む複数行の構文を使用します。条件が **False** のときに実行するコードがない場合、**Else** ステートメントを省略します。次に例を示します。

```
Dim alertLabel As New System.Windows.Forms.Label
Sub alertUser(ByVal value As Long)
    If value = 0 Then
        alertLabel.ForeColor = System.Drawing.Color.Red
        alertLabel.Font = New Font(alertLabel.Font, _
            FontStyle.Bold Or FontStyle.Italic)
    End If
End Sub
```

条件が True である場合にいくつかのステートメントを実行し、False の場合に他を実行する

- Else (Visual Basic) ステートメントと共に **If**...**Then**...**Else** 構造を使用して、ステートメントの 2 つのブロックを定義します。条件が **True** である場合に、あるブロックが Visual Basic によって実行され、**False** である場合に、他のブロックが実行されます。次に例を示します。

```
Dim alertLabel As New System.Windows.Forms.Label
Sub alertUser(ByVal value As Long)
    If value = 0 Then
        alertLabel.ForeColor = System.Drawing.Color.Red
        alertLabel.Font = New Font(alertLabel.Font, _
            FontStyle.Bold Or FontStyle.Italic)
    Else
        alertLabel.Forecolor = System.Drawing.Color.Black
        alertLabel.Font = New Font(alertLabel.Font, _
            FontStyle.Regular)
    End If
End Sub
```

最初の条件が False である場合に、追加の条件をテストする

- 1 つ以上の ElseIf (Visual Basic) ステートメントと共に **If**...**Then**...**Else** 構造を使用して、最初の条件が **False** である場合に、追加の条件をテストします。次の例では、**Function** プロシージャは、業績評価に基づいて従業員のボーナスを計算します。**If** ステートメントおよび **ElseIf** ステートメントの条件が、すべて **False** である場合だけ、**Else** ステートメントの後で、ステートメント ブロックが実行されます。

```
Function bonus(ByVal performance As Integer, ByVal salary As Decimal) _
    As Decimal
```

```
    If performance = 1 Then
        Return salary * 0.1
    ElseIf performance = 2 Then
        Return salary * 0.09
    ElseIf performance = 3 Then
        Return salary * 0.07
    Else
        Return 0
    End If
End Function
```

If...**Then**...**Else** ステートメントに表示される順序で、Visual Basic によって条件がテストされます。**True** 条件または **Else** ステートメントがあると、対応するステートメント ブロックが実行されます。その後、**End If** ステートメントの後のステートメントに、コントロールが渡されます。

ElseIf ステートメントは、必要に応じていくつでも追加できます。また、省略することもできます。**Else** ステートメントは、**ElseIf** ステートメントがあるかどうかに関係なく、含めることも省略することもできます。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : 式を複数の値と比較する
方法 : エラー発生時に制御を継続する

概念

条件判断構造
ループ構造
その他の制御構造
入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

方法：式を複数の値と比較する

同じ式を複数の値と比較する場合は、If...Then...Else ステートメント (Visual Basic) 構造の代わりに Select...Case ステートメント (Visual Basic) 構造を使用できます。**If** ステートメントと Elself (Visual Basic) ステートメントは、各ステートメントで異なる式を評価できるのに対し、**Select** ステートメントは、1 つの式を 1 回だけ評価し、その式をすべての比較に使用します。

1 つの式を 1 回だけ評価し、複数の値と比較するには

- **Select**...**Case** 構造を使用して、式および複数の値を指定します。各 **Case** ステートメントには、1 つまたは複数の値、値の範囲、または値と比較演算子の組み合わせを指定できます。**Case Else** ステートメントを使用すると、それ以前の **Case** ステートメントで検証されなかったすべての値を処理できます。このプロセスを説明する例を次に示します。

```
Function bonus(ByVal performance As Integer, ByVal salary As Decimal) _
    As Decimal
    Select performance
        Case 1
            Return salary * 0.1
        Case 2, 3
            Return salary * 0.09
        Case 5 To 7
            Return salary * 0.07
        Case 4, 8 To 10
            Return salary * 0.05
        Case Is < 15
            Return 100
        Case Else
            Return 0
   End Select
End Function
```

  Visual Basic は、式の値と、**Case** ステートメントの値とを、**Select**...**Case** で指定されている順序で比較します。値が一致するか、**Case Else** ステートメントが実行されると、対応するステートメント ブロックが実行されます。いずれの場合も、その後、**End Select** ステートメントに続くステートメントに分岐します。

  Case ステートメントは、任意の数指定できます。また、**Case** ステートメントがあるかどうかにかかわらず、**Case Else** ステートメントを 1 つ指定でき、省くこともできます。

Select...**Case** 構造のコード例は、IntelliSense コード スニペットにもあります。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : 1 つ以上の条件に基づいて、ステートメントを実行する
方法 : エラー発生時に制御を継続する

概念

条件判断構造
ループ構造
その他の制御構造
入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

方法：エラー発生時に制御を継続する

構造化例外処理に対して Try...Catch...Finally ステートメント (Visual Basic) 構造を使用できます。これにより、コードの実行中に特定の例外が発生した場合に特定のステートメント ブロックを実行できます。このように例外が発生することをコードが例外をスローすると言い、スローされた例外を **Catch** ステートメントで処理することを例外をキャッチすると言います。

コードによって例外が発生した場合に一連のステートメントを実行するには

- **Try**...**Catch**...**Finally** 構造を使用して、例外を引き起こす可能性のあるコードを囲みます。次に、例外が発生した場合に実行するコードを指定し、任意で、**Try**...**Catch**...**Finally** 構造から制御が移る前に実行する一連のステートメントを指定します。

  次の例では、**Object** 変数 `givenDate` で提供される値からちょうど 100 年後の日時を計算しています。

```
Dim givenDate As Object
Dim nextCentury As Date
Try
    nextCentury = Microsoft.VisualBasic.DateAdd("yyyy", 100, givenDate)
    Catch thisExcep As System.ArgumentOutOfRangeException
    ' The resulting date/time is later than December 31, 9999.
    Catch thisExcep As System.ArgumentException
    ' At least one argument has an invalid value.
    Catch thisExcep As System.InvalidCastException
    ' The value in givenDate cannot be interpreted as a date/time.
    Catch
    ' An unforeseen exception has occurred.
    Finally
    ' This block is always run before leaving the Try structure.
End Try
```

  最初の 3 つの **Catch** ブロックは、DateAdd 関数 (Visual Basic) から予測可能な例外を処理します。予期しない例外は、最後の **Catch** ブロックで処理できます。

  どのような場合でも、**Try**...**Catch**...**Finally** 構造から制御が移る前に、必ず最後に **Finally** ブロックで最後のコードが実行されます。**Try** または **Catch** ブロックで、オブジェクトまたはデータベース接続などのリソースを作成するまたは開く場合、必要に応じて **Finally** ブロックを使用してこれらを閉じたり、破棄したりできます。

  例外変数 `thisExcep` が、**Dim** などの宣言ステートメントで指定されていない場合は、**As** 句を持つ **Catch** ステートメントが宣言として機能します。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : 1 つ以上の条件に基づいて、ステートメントを実行する
方法 : 式を複数の値と比較する

概念

条件判断構造
ループ構造
その他の制御構造
入れ子になった制御構造

その他の技術情報

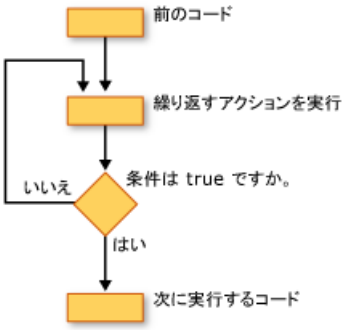
Visual Basic における制御フロー

ループ構造

Visual Basic のループ構造を使うと、1 行または複数行のコードを繰り返し実行できます。ループ構造のステートメントは、条件が **True** になるまで、条件が **False** になるまで、指定した回数、またはコレクションの各要素について 1 回ずつ繰り返すことができます。

次の図は、一連のステートメントを、条件が真になるまで実行するループ構造を示しています。

条件が真になるまで一連のステートメントを実行



WHILE ループ

While ステートメントで指定した条件が **True** である限り、**While**...**End While** 構造によって、一連のステートメントが実行されます。詳細については、「While...End While ステートメント (Visual Basic)」を参照してください。

Do ループ

Do...**Loop** 構造により、ループ構造の先頭または末尾のいずれかにある条件をテストできます。また、条件が **True** のままである間、または **True** になるまで、ループを繰り返すかどうかを指定することもできます。詳細については、「Do...Loop ステートメント (Visual Basic)」を参照してください。

FOR ループ

For...**Next** 構造では、設定した回数のループが実行されます。繰り返しの追跡には、カウンタと呼ばれるループ制御変数が使用されます。このカウンタに開始値および終了値を指定します。また、オプションで、1 回の繰り返しから次回の繰り返しへの増分を指定できます。詳細については、「For...Next ステートメント (Visual Basic)」を参照してください。

For Each ループ

For Each...**Next** 構造により、コレクション内の各要素に対して、一連のステートメントが 1 回実行されます。ループ制御変数を指定しますが、開始値または終了値を決める必要はありません。詳細については、「For Each...Next ステートメント (Visual Basic)」を参照してください。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : 複数のステートメントを繰り返し実行する
方法 : コレクションまたは配列で、各要素の複数のステートメントを実行する
方法 : ループのパフォーマンスを改善する
方法 : ループの次の反復処理にスキップする

概念

条件判断構造
その他の制御構造
入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

方法：複数のステートメントを繰り返し実行する

ループ構造を使用すると、ステートメント ブロックを繰り返し実行できます。ループは、条件の **Boolean** 値にしたがって不特定の回数実行することも、特別な変数で制御される指定回数だけ実行することもできます。

不特定回数ループを実行する

条件が True である限り一連のステートメントを実行するには

- While...End While ステートメント (Visual Basic) を使用してループの繰り返しを制御する条件を指定します。次の例では、`number` が 6 より大きい間ステートメント ブロックを繰り返します。

```
Sub checkWhile()
    Dim counter As Integer = 0
    Dim number As Integer = 10
    While number > 6
        number -= 1
        counter += 1
    End While
    MsgBox("The loop ran " & counter & " times.")
End Sub
```

While ステートメントは、ループを開始する前に必ず条件を確認します。`number` を 10 ではなく 6 に初期化すると、ループ内のステートメントは実行されません。

条件が True である間一連のステートメントを実行するには

- Do...Loop ステートメント (Visual Basic) を使用し、ループの先頭か最後にテスト条件を指定します。While (Visual Basic) キーワードを指定する場所によって、条件をテストする場所が決まります。次に例を示します。

```
Sub checkWhileFirst()
    Dim counter As Integer = 0
    Dim number As Integer = 10
    Do While number > 6
        number -= 1
        counter += 1
    Loop
    MsgBox("The loop ran " & counter & " times.")
End Sub
Sub checkWhileLast()
    Dim counter As Integer = 0
    Dim number As Integer = 5
    Do
        number -= 1
        counter += 1
    Loop While number > 6
    MsgBox("The loop ran " & counter & " times.")
End Sub
```

上の例では、最初の **Do** ループは 4 回実行され、2 番目の **Do** ループは 1 回実行されます。

条件が True になるまで一連のステートメントを実行するには

- **Do**...**Loop** 構造で、**While** キーワードの代わりに Until キーワードを使用します。**While** と同様、キーワードを指定する場所によって条件をテストする場所が決まります。次に例を示します。

```
Sub checkUntilFirst()
```

```
    Dim counter As Integer = 0
    Dim number As Integer = 20
    Do Until number = 15
        number -= 1
        counter += 1
    Loop
    MsgBox("The loop ran " & counter & " times.")
End Sub
Sub checkUntilLast()
    Dim counter As Integer = 0
    Dim number As Integer = 20
    Do
        number -= 1
        counter += 1
    Loop Until number = 15
    MsgBox("The loop ran " & counter & " times.")
End Sub
```

上の例では、各 **Do** ループは 5 回実行されます。

指定された回数だけループを実行する

ループ内のステートメントを何度実行する必要があるのかがわからない場合には、**While** と **Do** によるループが便利です。一方、指定の回数だけループを実行する場合は、For...Next ステートメント (Visual Basic) ループが便利です。**While** や **Do** ループとは違って、**For**...**Next** ループには制御変数という変数があり、ループが繰り返されるたびにこの変数の値が増加または減少します。

指定回数だけ、一連のステートメントを実行するには

1. 制御変数の開始値と終了値を決定し、**For** ステートメントを使用してこれを指定します。

   ```
   For i As Integer = 1 To 10
   ```

   制御変数がループの外側で宣言されていない場合は、**As** 句を使用して、**For** ステートメントの一部として制御変数を宣言できます。

2. 反復処理のたびに制御変数を増加させる量を指定するには、Step キーワードを使用します。特に指定しない場合は 1 ずつ増加します。制御変数を減らすには負の値を使用します。

   ```
   For i As Integer = 10 To 1 Step -1
   ```

3. **For**...**Next** 構造を閉じるには、繰り返す最後のステートメントの次に **Next** ステートメントを指定します。**Next** ステートメントに制御変数を指定することもできます。

   ```
   Function addBackward(ByVal highest As Integer) As Integer
       Dim total As Integer = 0
       For i As Integer = highest To 1 Step -1
           total += i
       Next i
       Return total
   End Function
   ```

   上の例では、1 から `highest` パラメータに渡された値までのすべての数字の合計を返します。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : コレクションまたは配列で、各要素の複数のステートメントを実行する
方法 : ループのパフォーマンスを改善する
方法 : ループの次の反復処理にスキップする

関連項目

方法：コレクションまたは配列で、各要素の複数のステートメントを実行する

For Each ステートメント構造は、**For**...**Next** ループに似ていますが、指定した回数だけステートメント ブロックを実行するのではなく、コレクションの各要素に対してステートメント ブロックを実行します。**For Each**...**Next** ループでは、ループの各繰り返しのときに、コレクションの異なる要素を表す要素変数が使用されます。

コレクションのステートメント ブロックの繰り返し

コレクションの各要素に対して、一連のステートメントを実行するには

1. ステートメント ブロックを実行するコレクションを識別し、For Each...Next ステートメント (Visual Basic) を使用して、要素変数とコレクションの両方を指定します。

   ```
   For Each thisControl As System.Windows.Forms.Control In thisForm.Contr
   ols
   ```

   要素変数がループの外側で宣言されていない場合は、**As** 句を使用して、**For Each** ステートメントの一部として要素変数を宣言できます。

   Visual Basic では、それぞれの反復処理に対して、コレクションの異なる要素が自動的に置換されます。

2. 繰り返される最後のステートメントの後で **Next** ステートメントを使用して、**For Each**...**Next** 構造を完成します。**Next** ステートメントで要素変数を指定することもできます。

   ```
   Sub lightBlueBackground(ByVal thisForm As System.Windows.Forms.Form)
       For Each thisControl As System.Windows.Forms.Control In thisForm.Controls
           thisControl.BackColor = System.Drawing.Color.LightBlue
       Next thisControl
   End Sub
   ```

   コレクションの要素を見ることはできますが、コレクション自体に、要素の追加、削除、または置き換えなどの変更を加えることはできません。しかし、要素が参照型である場合、アクセスしてメンバを設定できます。各 Control 要素が参照型であるため、上の例のコードの BackColor プロパティを変更できます。

配列のステートメント ブロックの繰り返し

For Each...**Next** ループを使用して、配列を反復処理することもできます。ただし、コレクションと同様に、配列要素を見るだけで変更はできません。

配列の各要素に対して、一連のステートメントを実行するには

1. **For Each** ステートメントを使用して、要素変数および配列の両方を指定します。配列名の後にかっこを指定しないでください。

   Visual Basic では、配列がコレクションと同様に扱われます。

2. ステートメント ブロックの後に、**Next** ステートメントを指定します。**Next** ステートメントで要素変数を指定することもできます。

   次に示すのは、配列の要素の合計を求め、各要素をゼロにリセットするプロシージャの例です。

   ```
   Function sumAndReset(ByRef numbers() As Integer) As Integer
       Dim sum As Integer = 0
       For Each elt As Integer In numbers
           sum += elt
           ' The following statement works only on the local copy
           ' of the array, not on the original array.
           elt = 0
       Next elt
   ```

```
    Return sum
End Function
```

合計の計算は、要素の読み取りだけを使用して行われるため、正常に機能します。しかし、ゼロへのリセットは機能しません。各要素のローカル コピーだけがゼロにリセットされ、元の要素は元の配列で変更されないまま残るためです。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : 複数のステートメントを繰り返し実行する
方法 : ループのパフォーマンスを改善する
方法 : ループの次の反復処理にスキップする

関連項目

For...Next ステートメント (Visual Basic)

概念

条件判断構造
ループ構造
その他の制御構造
入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

方法：ループのパフォーマンスを改善する

最も効率的なデータ型を使用することによって、ループのパフォーマンスを最適化できます。実行回数の少ない短いループの場合には、その差はごくわずかなものです。しかし、実行回数の多いループの場合、パフォーマンスの改善はかなり大きなものとなります。

Integer および **UInteger** は、現在のプラットフォームでは最も効率的な型です。**Short**、**Long**、**UShort**、および **ULong** はあまり効率的ではなりません。**Decimal** はかなり遅くなります。詳細については、「数値データ型」を参照してください。

For...Next ループのパフォーマンスを最適化するには

- コントロール変数に対して最も効率的なデータ型を使用します。たとえば、ループに関して次のようなバリエーションが考えられます。絶対的なタイミングはプラットフォームに依存しますが、実行時の比較はそのまま有効です。

```
For fastest As Integer = 0 to 1000000
    ' Insert statements to execute for each value of fastest.
Next fastest
For notAsFast As Long = 0 to 1000000
    ' Insert statements to execute for each value of notAsFast.
Next notAsFast
For muchSlower As Decimal = 0 to 1000000
    ' Insert statements to execute for each value of muchSlower.
Next muchSlower
```

  1 番目の例は、2 番目の例よりほんの少しだけ実行時間が短くなります。ただし、**Integer** は最高 2,147,483,647 まで、**UInteger** は最高 4,294,967,295 までしか処理できません。2 番目と 3 番目の例では、**Long** と **Decimal** の両方とも、より広い範囲の整数を受け入れるためより大きな値を処理できますが、実行には時間がかかります。使用するデータ型の処理速度と能力を検討し、設計上の判断を下す必要があります。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : 複数のステートメントを繰り返し実行する
方法 : コレクションまたは配列で、各要素の複数のステートメントを実行する
方法 : ループの次の反復処理にスキップする

関連項目

For...Next ステートメント (Visual Basic)

概念

ループ構造

その他の技術情報

Visual Basic における制御フロー

方法：ループの次の反復処理にスキップする

Do ループ、**For** ループ、または **While** ループの現在の反復処理を完了している場合、Continue ステートメント (Visual Basic) を使用して、次の反復処理にすぐにスキップできます。

次の反復処理へのスキップ

For...Next ループの次の反復処理へスキップするには

1. 通常の方法で、**For**...**Next** ループを記述します。
2. 現在の反復処理を終了し、次の反復処理をすぐに続行する場所で、**Continue For** を使用します。

```
Public Function findLargestRatio(ByVal high() As Double, _
    ByVal low() As Double) As Double
    Dim ratio As Double
    Dim largestRatio As Double = Double.MinValue
    For counter As Integer = 0 To low.GetUpperBound(0)
        If Math.Abs(low(counter)) < System.Double.Epsilon _
            Then Continue For
        ratio = high(counter) / low(counter)
        If Double.IsInfinity(ratio) OrElse Double.IsNaN(ratio) _
            Then Continue For
        If ratio > largestRatio Then largestRatio = ratio
    Next counter
    Return largestRatio
End Function
```

入れ子になったループ内からのスキップ

互いに入れ子になっている **Do** ループ、**For** ループまたは **While** ループがある場合、次の反復処理が入れ子内のどのレベルにあっても、そこにすぐにスキップできます。しかし、これはループが異なる種類である場合だけです。たとえば、入れ子になった **While** ループなど、同じ種類のループの入れ子になっている場合は、**Continue While** によって、最も内側の **While** ループの次の反復処理にスキップします。

入れ子になった For ループ内から、Do ループの次の反復処理にスキップするには

1. 通常の方法で、入れ子になったループを記述します。
2. 内部 **For** ループの現在の反復処理を終了する場所で **Continue Do** を使用し、外側の **Do** ループの次の反復処理にスキップします。

```
Public Sub divideElements(ByRef matrix(,) As Double)
    Dim i As Integer = -1
    Do Until i > matrix.GetUpperBound(0)
        i += 1
        For j As Integer = 0 To matrix.GetUpperBound(1)
            If matrix(j, j) = 0 Then Continue Do
            matrix(i, j) /= matrix(j, j)
        Next j
    Loop
End Sub
```

参照

処理手順

方法 : 制御構造から制御を移す
方法 : 複数のステートメントを繰り返し実行する
方法 : コレクションまたは配列で、各要素の複数のステートメントを実行する
方法 : ループのパフォーマンスを改善する

関連項目

While...End While ステートメント (Visual Basic)
Do...Loop ステートメント (Visual Basic)
For...Next ステートメント (Visual Basic)

概念

ループ構造
入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

その他の制御構造

Visual Basic には、リソースの破棄するために、または 1 つのオブジェクトを繰り返し参照する回数を減らすために役立つ制御構造があります。

Using...End Using 構造

Using...End Using 構造は、ステートメント ブロックを構築し、SQL 接続などのリソースをそのブロック内で利用できるようにします。**Using** ステートメントでは、オプションでリソースを取得することもできます。**Using** ブロックを終了すると、Visual Basic によって自動的にリソースが解放され、他のコードでそのリソースを使用できるようになります。リソースは、ローカルであり、破棄可能である必要があります。詳細については、「Using ステートメント (Visual Basic)」を参照してください。

With...End With 構造

With...End With 構造では、オブジェクト参照を 1 回だけ指定してから、そのメンバにアクセスする一連のステートメントを実行できます。これによりコードが簡単になります。また、オブジェクトにアクセスするステートメントのたびに Visual Basic が参照を再度確立する必要がなくなるため、パフォーマンスを向上できます。詳細については、「With...End With ステートメント (Visual Basic)」を参照してください。

参照

処理手順

方法 : 制御構造から制御を移す

関連項目

Using ステートメント (Visual Basic)

With...End With ステートメント (Visual Basic)

概念

条件判断構造

ループ構造

入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

方法：システム リソースを破棄する

Using ブロックを使用して、コードがブロックを終了するときに、必ずシステムでリソースが破棄されるようにできます。これは、大量のメモリを消費するシステム リソースを使用している場合、または他のコンポーネントでもシステム リソースを使用できるようにする場合に便利です。

コードがデータベース処理を終了するときにデータベース接続を切断するには

1. ソース ファイルの先頭に、データベース接続用の適切な Imports ステートメント が含まれていることを確認します (この場合は、System.Data.SqlClient)。
2. **Using** および **End Using** ステートメントを使用して、**Using** ブロックを作成します。ブロック内部には、データベース接続を処理するコードを含めます。
3. 接続を宣言し、**Using** ステートメントの一部として、インスタンスを作成します。

```
' Insert the following line at the beginning of your source file.
Imports System.Data.SqlClient
Public Sub AccessSql(ByVal s As String)
    Using sqc As New System.Data.SqlClient.SqlConnection(s)
        MsgBox("Connected with string """ & sqc.ConnectionString & """")
    End Using
End Sub
```

システムは、未処理の例外も含めて、どのような場合であってもブロックを終了するときに、リソースを破棄します。

`sqc` の範囲はブロックに限定されているため、**Using** ブロックの外側からはこれにアクセスできないことに注意してください。

ファイル処理または COM ラッパーなどのシステム リソースに対しても、これと同じ技法を使用できます。**Using** ブロックを終了した後、他のコンポーネントでそのリソースを確実に使用できるようにするには、**Using** ブロックを使用します。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : オブジェクトに対して複数のアクションを実行する

関連項目

Using ステートメント (Visual Basic)
SqlConnection

概念

条件判断構造
ループ構造
その他の制御構造
入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

方法：オブジェクトに対して複数のアクションを実行する

通常、Visual Basic では、オブジェクトのメソッドを呼び出したりオブジェクトのプロパティにアクセスしたりするすべてのステートメントでオブジェクトを指定する必要があります。しかし、一連のステートメントがすべて同じオブジェクトを対象とする場合は、**With...End With** 構造を使用して、1 回ですべてのステートメントに対してそのオブジェクトを指定できます。これにより、プロシージャの実行速度が向上し、入力の手間も省くことができます。

使用例

次の例では、プロシージャ引数の値に従って Label の前景色とフォント スタイルを設定します。

```
Imports draw = System.Drawing
' The preceding statement must appear at the beginning of the source file.
Dim alertLabel As New System.Windows.Forms.Label
Sub alertUser(ByVal value As Long)
    With alertLabel
        If value = 0 Then
            .ForeColor = draw.Color.Red
            .Font = New draw.Font(.Font, draw.FontStyle.Bold Or draw.FontStyle.Italic)
        Else
            .Forecolor = draw.Color.Black
            .Font = New draw.Font(.Font, draw.FontStyle.Regular)
        End If
    End With
End Sub
```

複数のフォント スタイルを組み合わせるには、Or 演算子 (Visual Basic) を使用します。これによって、ビット フラグの目的の組み合わせを指定します。FontStyle 列挙型のメンバはすべて異なるビットを使用しているため、And 演算子 (Visual Basic) の結果は 0 になります。

また、Imports ステートメント を使用してインポート エイリアス `draw` を構築しています。これによって、`System.Drawing` メンバへの各参照が短くなり、読みやすくなります。

With...End With ステートメントの中に別の With...End WIth ステートメントを記述して入れ子にすることもできます。コードの例を次に示します。

VB

```
Sub setupForm()
    Dim anotherForm As New System.Windows.Forms.Form
    Dim button1 As New System.Windows.Forms.Button
    With anotherForm
        .Show()
        .Top = 250
        .Left = 250
        .ForeColor = System.Drawing.Color.LightBlue
        .BackColor = System.Drawing.Color.DarkBlue
        .Controls.Add(button1)
        With .Controls.Item(1)
            .BackColor = System.Drawing.Color.Thistle
            .Text = "Text on button1"
        End With
    End With
End Sub
```

ただし、入れ子になった **With** ステートメントの構文では、入れ子の内側で指定されている側のオブジェクトが対象となります。外側の **With** ステートメントにあるオブジェクトのプロパティは設定されません。

参照

処理手順

方法 : 制御構造から制御を移す
方法 : システム リソースを破棄する
方法 : 長い修飾パスをもつオブジェクトへのアクセス時間を短縮する

関連項目

With...End With ステートメント (Visual Basic)

概念

条件判断構造

ループ構造

その他の制御構造

入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

入れ子になった制御構造

For...Next ループの中に **If...Then...Else** ブロックを配置するなど、制御ステートメントの中に別の制御ステートメントを配置できます。制御ステートメントの中に別の制御ステートメントを配置することを入れ子にすると言います。

入れ子のレベル

Visual Basic には、制御構造を入れ子にするときのレベルに制限はありません。構造を入れ子にする場合は、コードを読みやすくするために、レベルごとにインデントして記述するのが一般的です。これは、統合開発環境 (IDE: Integrated Development Environment) エディタによって、自動的に行われます。

次の例では、`sumRows` プロシージャによって、行列の各行の正の要素が合計されます。

```
Public Sub sumRows(ByVal a(,) As Double, ByRef r() As Double)
    Dim i, j As Integer
    For i = 0 To UBound(a, 1)
        r(i) = 0
        For j = 0 To UBound(a, 2)
            If a(i, j) > 0 Then
                r(i) = r(i) + a(i, j)
            End If
        Next j
    Next i
End Sub
```

上の例では、最初の **Next** ステートメントによって、内部 **For** ループが閉じられ、最後の **Next** ステートメントによって、外部 **For** ループが閉じられます。

同様に、入れ子になった **If** ステートメントの **End If** ステートメントは、先行する最も近い **If** ステートメントに対して自動的に適用されます。入れ子になった **Do** ループの動作も同様であり、最も内側の **Loop** ステートメントが最も内側の **Do** ステートメントに対応します。

異なる種類の制御構造を入れ子にする

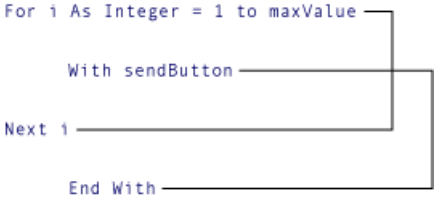
他の種類の制御構造内で、ある種類の制御構造を入れ子にできます。**For Each** ループ内部で **With** ブロックを使用し、**With** ブロック内部で、入れ子になった **If** ブロックを使用しているコード例を次に示します。

```
For Each ctl As System.Windows.Forms.Control In Me.Controls
    With ctl
        .BackColor = System.Drawing.Color.Yellow
        .ForeColor = System.Drawing.Color.Black
        If .CanFocus Then
            .Text = "Colors changed"
            If Not .Focus() Then
                ' Insert code to process failed focus.
            End If
        End If
    End With
Next ctl
```

制御構造の重なり

制御構造は重ねることはできません。つまり、入れ子になった構造体は、次の最も内側の構造体の中に完全に含まれる必要があります。たとえば、次の配置は、内部の **With** ブロックが終了する前に **For** ループが終了するため、無効です。

For および With 構造体の無効な入れ子



Visual Basic コンパイラによって、そのような制御構造の重なりが検出され、コンパイル エラーが通知されます。

参照

処理手順

方法 : 制御構造から制御を移す

概念

条件判断構造

ループ構造

その他の制御構造

その他の技術情報

Visual Basic における制御フロー

方法：制御構造から制御を移す

Exit ステートメント (Visual Basic) を使用すると、制御構造から直接抜け出ることができます。**Exit** は、直ちに実行を制御構造の最後のステートメントの次のステートメントに移します。**Exit** ステートメントの構文では、終了する制御構造の種類を指定します。次のような **Exit** ステートメントが可能です。

- **Exit Select**
- **Exit Try**
- **Exit While**
- **Exit Do**
- **Exit For**

Exit ステートメントは、これがサポートされている制御構造内で、必要に応じて何度でも指定できます。**Exit** は、その制御構造で必要な処理がすべて完了し、これ以上ステートメントを実行する必要がない場合に便利です。

Exit をサポートしない制御構造。If、**Using**、および **With** ブロックでは、**Exit** ステートメントを使用して制御を移すことはできません。そのような結果を得るには、ブロックの **End** ステートメントにステートメント ラベルを付け、**GoTo** ステートメントを使用して制御を移します。ステートメント ラベルの詳細については、「方法 : ステートメントにラベル付けする」を参照してください。

使用例

入れ子になった制御構造の中で **Exit** ステートメントを使用すると、**Exit** ステートメントで指定した種類の最も内側の構造の最後に続くステートメントに制御が渡されます。次に例を示します。

```
Public Sub invertElements(ByRef a(,) As Double)
    For i As Integer = 0 To UBound(a, 1)
        For j As Integer = 0 To UBound(a, 2)
            If a(i, j) = 0 Then
                ' Cannot complete this row; resume outer loop.
                Exit For
            Else
                a(i, j) = 1.0 / a(i, j)
            End If
        Next j
        ' Control comes here directly from the Exit For statement.
    Next i
End Sub
```

上の例では、**Exit For** ステートメントが内側の **For** ループの中に配置されているため、このループに続くステートメントに制御が渡され、外側の **For** ループが実行されます。

参照

処理手順

方法 : ステートメントにラベル付けする

関連項目

Exit ステートメント (Visual Basic)
GoTo ステートメント

概念

条件判断構造
ループ構造
その他の制御構造
入れ子になった制御構造

その他の技術情報

Visual Basic における制御フロー

Visual Basic におけるオブジェクト指向プログラミング

オブジェクトは Visual Basic でのプログラミングの柱となるものです。フォームやコントロールはオブジェクトです。また、データベースもオブジェクトです。Visual Basic を使用したことがある、またはこのドキュメントの例を実践済みの場合は、オブジェクト指向プログラミングを経験したことになります。しかし、オブジェクトには、それ以上に多くの側面があります。

以下のトピックでは、定義したクラスから独自のオブジェクトを簡単に作成する方法や、オブジェクトを使ってすっきりとしたコードを作成したり、コードを再利用しやすくしたりする方法を説明します。

このセクションの内容

Visual Basic のオブジェクトの概要

オブジェクト指向プログラミングで使用される用語や概念を紹介します。

事前バインディングと遅延バインディング

オブジェクトがオブジェクト変数に代入されるときにコンパイラによって実行されるバインディングについて、および事前バンディング オブジェクトと遅延バインディング オブジェクトの違いについて説明します。

Visual Basic の共有メンバ

インスタンスを作成しないで使用できるメンバについて説明します。

オブジェクトの作成と使用

クラスのインスタンスの作成方法および使用方法について説明します。

オブジェクトのグループの管理

オブジェクトの配列やコレクションを使用する方法を紹介します。

実行時のクラス情報の取得

オブジェクトが属するクラスの決定方法を説明します。

クラスについて

オブジェクトの作成と有効期間について、詳しい手順を交えて説明します。

Visual Basic におけるイベント

イベントを宣言および使用する方法について説明します。

Visual Basic でのデリゲート

デリゲートを宣言および使用する方法について説明します。

Visual Basic におけるインターフェイス

インターフェイスの概要とアプリケーションでの使用方法を説明します。

Visual Basic の継承

他のクラスの基本クラスとして機能するクラスの定義方法について説明します。

関連するセクション

Visual Basic におけるオブジェクト

オブジェクトの概要と使用方法を説明します。

コンポーネントによるプログラミング

他のアプリケーションによって提供されるオブジェクトを Visual Basic を使って制御する方法を説明します。

Visual Basic のオブジェクトの概要

オブジェクトは、データおよびそのデータを操作するメソッドを含む構造体のことです。Visual Basic で行うほとんどすべての作業は、オブジェクトにかかわっています。オブジェクト指向プログラミングの経験がない場合は、まず以下の用語と概念の説明を参照してください。

クラスとオブジェクト

"クラス" と "オブジェクト" という用語は、オブジェクト指向プログラミングでは頻繁に使用されるため、混同しやすい用語であると言えます。一般に、*class*はある対象の抽象表現であり、オブジェクトはそのクラスを表す実例です。ただし共有クラス メンバは例外で、クラスのインスタンスで使用することも、クラスの型として宣言されたオブジェクト変数で使用することもできます。

フィールド、プロパティ、メソッド、およびイベント

クラスは、フィールド、プロパティ、メソッド、およびイベントから構成されています。フィールドとプロパティは、オブジェクトに格納されている情報を表します。フィールドは、変数と同じように直接読み取ったり設定したりできます。たとえば、"Car" というオブジェクトがある場合、その色を "Color" というフィールドに格納できます。

プロパティは、フィールドと同じように取得と設定ができます。ただし、プロパティ **Get** プロシージャとプロパティ **Set**プロシージャを使用して実装すると、設定値や戻り値をより細かく制御できます。これらのプロシージャは、格納される値とその値を使用するプロシージャとの仲介層として機能するため、データを切り離して、割り当ておよび取得の前に値を確認できます。

メソッドは、オブジェクトが実行できる処理を表します。たとえば、"Car" オブジェクトのメソッドとしては、"StartEngine"、"Drive"、"Stop" などが考えられます。メソッドを定義するには、プロシージャ (**Sub** ルーチンまたは関数) をクラスに追加します。

イベントは、オブジェクトがほかのオブジェクトやアプリケーションとの間で送受信する通知です。イベントを使うと、ある特定の動作が起こるたびにオブジェクトに処理を実行させることができます。たとえば、"Car" クラスのイベントとしては、"Check_Engine" イベントなどが考えられます。Microsoft Windows はイベント ドリブンのオペレーティング システムなので、ほかのオブジェクト、アプリケーション、マウス クリックやキー入力などのユーザー入力によってイベントが発生します。

カプセル化、継承、およびポリモーフィズム

上で説明したフィールド、プロパティ、メソッド、およびイベントは、オブジェクト指向プログラミングの法則を半分しか伝えていません。真のオブジェクト指向プログラミングは、カプセル化、継承、およびポリモーフィズムの 3 つの特性をオブジェクトでサポートする必要があります。

カプセル化とは、関連するプロパティ、メソッド、およびその他のメンバのグループが 1 つの単位またはオブジェクトとして扱われることを意味します。オブジェクトは、プロパティの変更やメソッドの実行を制御できます。たとえば、プロパティが変更される前に値を確認できます。また、カプセル化により、オブジェクトの実装の詳細を隠すことができるため、実装を後で変更するのが簡単になります。これをデータの隠ぺいと呼びます。

継承は、既存のクラスに基づいて新しいクラスを作成する機能です。新しいクラスは、基本クラスのすべてのプロパティ、メソッド、およびイベントを継承します。また、プロパティやメソッドを追加することによってカスタマイズできます。たとえば、"Car" クラスを基にして "Truck" という新しいクラスを作成できます。"Truck" クラスは、"Car" クラスの "Color" プロパティを継承します。また、"FourWheelDrive" などのプロパティを追加できます。

ポリモーフィズムを採用すると、複数のクラスがあり、各クラスが同一プロパティや同一メソッドを異なる方法で実装していたとしても、交換して使用できます。ポリモーフィズムは、オブジェクト指向プログラミングには欠かせない特性です。ポリモーフィズムによって、その時点で使用されているオブジェクトの型に関係なく、同じ名前の項目を使用できるようになります。たとえば、"Car" という基本クラスがある場合、ポリモーフィズムが実現されていると、複数の派生クラスにそれぞれ異なる "StartEngine" メソッドを定義できます。"DieselCar" という派生クラスの "StartEngine" メソッドは、基本クラスの "StartEngine" メソッドとまったく異なっていてもかまいません。他のプロシージャやメソッドは、その時点で使用されている "Car" オブジェクトの型に関係なく、まったく同じ方法で派生クラスの "StartEngine" メソッドを使用できます。

オーバーロード、オーバーライド、およびシャドウ

オーバーロード、オーバーライド、およびシャドウの概念は、似ているためにしばしば混同されます。これらは 3 つともすべて同じ名前のメンバを作成するための方法ですが、それぞれ重要な違いがあります。

- オーバーロードされたメンバを使うと、同じ名前を持つプロパティまたはメソッドの別のバージョンを提供できます。ただし、名前は同じでも、パラメータの数またはパラメータのデータ型は異なっている必要があります。
- オーバーライドされたプロパティやメソッドを使うと、継承したプロパティやメソッドを派生クラスで置き換えることができます。オーバーライドされたメンバの引数のデータ型と数は同じである必要があります。派生クラスは、オーバーライドされたメンバを継承します。
- シャドウされたメンバは、ローカルな範囲内で、より広いスコープを持つメンバの代わりに使用されます。シャドウの対象となる型に制約はありません。たとえば、継承した同じ名前のメソッドをシャドウするプロパティを宣言できます。シャドウされたメンバは継承できません。

参照

概念

クラス : オブジェクトの設計図
オーバーロードされたプロパティとメソッド
プロパティとメソッドのオーバーライド
Visual Basic におけるシャドウ

その他の技術情報

Visual Basic におけるオブジェクト
オブジェクトの作成と使用
Visual Basic の継承

クラスについて

オブジェクト指向プログラミングでは、クラスが重要な役割を示します。クラスを利用して、関連する項目を 1 つの単位にグループ化したり、他のプロシージャからの参照可能範囲やアクセシビリティを制御したりできます。また、他のクラスで定義されたコードをクラスが継承して再利用することもできます。

このセクションの内容

クラス : オブジェクトの設計図

カプセル化、継承、および共有メンバの概要を示します。

チュートリアル : クラスの定義

クラスを作成する手順について説明します。

クラスとモジュール

クラスと標準モジュールの違いについて説明します。

オブジェクトの有効期間 : オブジェクトの作成と破棄

クラス インスタンスの作成と破棄について説明します。

方法 : Dispose Finalize パターンを実装する (Visual Basic)

オブジェクトのリソースが不要になったときに、そのリソースを確実に解放する方法について説明します。

コンストラクタとデストラクタの使用方法

クラスの新しいインスタンスを初期化する方法と、不要になったリソースを破棄する方法について説明します。

クラス、プロパティ、フィールド、およびメソッド

クラスを構成するフィールド、プロパティ、およびメソッドについて説明します。

関連するセクション

オブジェクトの作成と使用

クラスのインスタンスを作成および使用する方法について説明します。

Visual Basic におけるインターフェイス

インターフェイスの概要を示し、アプリケーションでの使用方法について説明します。

Visual Basic の継承

他のクラスの基本クラスとして機能するクラスの定義方法について説明します。

クラス : オブジェクトの設計図

クラスはオブジェクトのシンボリックな表現です。設計図が建築物を構成する項目を説明するものであるのと同じように、クラスはオブジェクトを構成するプロパティ、フィールド、メソッド、およびイベントを説明するものです。1 枚の設計図から複数の建築物を建造できるように、1 つのクラスから必要な数のオブジェクトを作成できます。建築物を使用する人の出入り (アクセス) できる部分が設計図によって定義されるように、クラスは、カプセル化を通じて、オブジェクト項目へのユーザー アクセスを制御できます。

クラスとオブジェクト

クラスという用語とオブジェクトという用語は同じ意味で使用される場合がありますが、実際には、クラスはオブジェクトの構造を表すのに対し、オブジェクトはクラスの使用可能なインスタンスです。各インスタンスは、クラスの独立したコピーです。オブジェクトはクラスのインスタンスであるため、オブジェクトを作成する操作はインスタンス化と呼ばれます。

設計図との対比を使って説明すると、クラスは設計図であり、オブジェクトはその設計図を基にした建築物です。通常は、1 つのオブジェクト内のデータを変更しても、他のオブジェクト内のデータは変更されません(ただし、共有メンバの場合は例外です。共有メンバは、**Shared** 修飾子を指定して宣言されたクラス メンバであり、クラスの特定のインスタンスとは独立に存在します)。

カプセル化

カプセル化は、関連付けられた複数の項目から成るグループを包含し、グループへのアクセスを制御する機能です。クラスは、項目をカプセル化する最も一般的な方法の 1 つです。下の例の `BankAccount` クラスは、銀行の口座を表すメソッド、フィールド、およびプロパティをカプセル化します。

カプセル化を使用しない場合、たとえば、銀行口座の情報を保持および管理するためのプロシージャや変数を個別に宣言する必要があります。この方法では、一度に複数の口座を処理するのは困難です。カプセル化を使用すると、データおよびプロシージャを `BankAccount` クラスにまとめ、1 つのユニットとして扱うことができます。各口座はクラスの一意のインスタンスで表現されるため、混乱することなく、一度に複数の銀行口座を処理できます。

カプセル化により、データとプロシージャの使用も制御できます。**Private** や **Protected** などのアクセス修飾子を使用することで、外部のプロシージャによるクラスのメソッドの実行や、プロパティやフィールドのデータの読み取りおよび変更を禁止できます。クラスの外部で使用されるのを防ぐためには、クラスの内部的な詳細を **Private** として宣言しておく必要があります。この技法は、データの隠ぺいと呼ばれ、口座残高などの顧客情報を保護するための方法です。

カプセル化の基本的な規則として、クラス データは **Property** プロシージャまたはメソッドだけを通じて変更または取得します。クラスの実装の詳細を隠ぺいすることで、望まない方法でクラスが使用されるのを防ぎます。また、互換性の問題を発生させることなく、後でこれらの項目を変更できます。たとえば、`BankAccount` クラスの将来のバージョンでは、`AccountBalance` フィールドのデータ型に依存する他のアプリケーションに影響を与えることなく、このフィールドのデータ型を変更できます。

継承

Visual Basic の構造体と同様に、関連する項目のグループをカプセル化するデータ型をクラスを使用して定義できます。しかし、構造体とは異なり、Visual Basic のクラスでは、他のクラスの特性を継承および拡張できます。新しいクラスの基になるクラスは、基本クラスと呼ばれます。基本クラスから派生したクラスは、派生クラスと呼ばれます。派生クラスは、基本クラスのプロパティ、メソッド、およびイベントをすべて継承します。つまり、一度開発およびデバッグしたクラスは、他のクラスの基本クラスとして再利用できます。

次の例は、汎用の銀行口座を表す基本クラスを定義しています。また、基本クラスのプロパティを継承し、当座預金を表すようにカスタマイズされたクラスも定義しています。

VB

```
Class BankAccount
    Private AccountNumber As String
    Private AccountBalance As Decimal
    Private HoldOnAccount As Boolean = False
    Public Sub PostInterest()
        ' Add code to calculate the interest for this account.
    End Sub
    ReadOnly Property Balance() As Decimal
        Get
            ' Return the available balance.
            Return AccountBalance
        End Get
    End Property
End Class

Class CheckingAccount
```

```
    Inherits BankAccount
    Sub ProcessCheck()
        ' Add code to process a check drawn on this account.
    End Sub
End Class
```

継承の詳細については、「継承の基本」を参照してください。

共有メンバ

既定では、クラスのデータはクラスの各インスタンスに固有になります。ただし、1 つのクラスから作成したすべてのオブジェクト間で、1 つのデータ項目を共有する方が都合のよい場合もあります。このような場合には、**Shared** 修飾子を使用して、変数の同じ値をクラス内のすべてのインスタンスで共有できるようにします (共有メンバは、他のプログラミング言語では "静的メンバ" と呼ばれる場合もあります)。共有メソッドは、事前にクラスのインスタンスを作成しなくても、クラス名を使用して直接呼び出すことができます。

共有メンバの詳細については、「Visual Basic の共有メンバ」を参照してください。

シャドウ

派生クラスで **Shadows** キーワードを使用すると、継承したメンバと同じ名前のメンバを宣言できます。シャドウされるメンバとシャドウするメンバが、同じデータ型である必要はありません。たとえば、プロパティは **Integer** 型の変数をシャドウできます。

共有メンバの詳細については、「Visual Basic におけるシャドウ」を参照してください。

参照

関連項目

Shadows
Shared (Visual Basic)

概念

構造体とクラス
継承の基本
Visual Basic の共有メンバ
Visual Basic におけるシャドウ

その他の技術情報

オブジェクトの作成と使用
Visual Basic の継承

チュートリアル：クラスの定義

このチュートリアルでは、クラスを定義する方法を説明します。クラスを定義したら、オブジェクトの作成に使用できます。作成したクラスにプロパティとメソッドを追加する方法、およびオブジェクトを初期化する方法についても説明します。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

クラスを定義するには

1. [ファイル] メニューの [新しいプロジェクト] をクリックしてプロジェクトを作成します。[新しいプロジェクト] ダイアログ ボックスが表示されます。
2. Visual Basic プロジェクトの一覧の [Windows アプリケーション] を選択して、新しいプロジェクトを表示します。
3. [プロジェクト] メニューの [クラスの追加] をクリックして、新規クラスをプロジェクトに追加します。[新しい項目の追加] ダイアログ ボックスが表示されます。
4. [クラス] テンプレートを選択します。
5. 新しいクラス `UserNameInfo.vb` に名前を指定し、[追加] をクリックして新しいクラスのコードを表示します。

   VB

   ```
   Public Class UserNameInfo
   End Class
   ```

   メモ：

   Visual Basic の コード エディタ を使用して、クラスをスタートアップ フォームに追加できます。クラスを追加するには、**Class** キーワードに続けて新規クラスの名前を入力します。コード エディタ によって、対応する **End Class** ステートメントが自動的に入力されます。

6. **Class** ステートメントと **End Class** ステートメントの間に次のコードを記述して、クラスのプライベート フィールドを定義します。

   VB

   ```
   Private userNameValue As String
   ```

   フィールドに **Private** を宣言した場合、そのクラスの内部でのみ使用可能になります。これらのフィールドをクラスの外部からアクセスできるようにするには、**Public** などのより広いアクセス範囲を提供するアクセス修飾子を使用します。詳細については、「Visual Basic でのアクセス レベル」を参照してください。

7. 次のコードを追加してクラスのプロパティを定義します。

   VB

   ```
   Public Property UserName() As String
       Get
           ' Gets the property value.
           Return userNameValue
       End Get
       Set(ByVal Value As String)
           ' Sets the property value.
           userNameValue = Value
       End Set
   End Property
   ```

8. 次のコードを追加してクラスのメソッドを定義します。

   VB

```
Public Sub Capitalize()
    ' Capitalize the value of the property.
    userNameValue = UCase(userNameValue)
End Sub
```

9. `Sub New` というプロシージャを追加して新規クラスのパラメータ化されたコンストラクタを定義します。

   VB

```
Public Sub New(ByVal UserName As String)
    ' Set the property value.
    Me.UserName = UserName
End Sub
```

   `Sub New` コンストラクタは、このクラスからオブジェクトを作成するとき自動的に呼び出されます。このコンストラクタは、ユーザー名を保持するフィールドの値を設定します。

クラスをテストするボタンを作成するには

1. スタートアップ フォームをデザイン モードに変更します。変更するには、ソリューション エクスプローラ でスタートアップ フォームの名前を右クリックし、[デザイナの表示] をクリックします。Windows アプリケーション プロジェクトのスタートアップ フォームは、既定で Form1.vb という名前が付きます。メイン フォームが表示されます。

2. メイン フォームにボタンを追加し、それをダブルクリックして、`Button1_Click` イベント ハンドラのコードを表示します。テスト プロシージャを呼び出すために次のコードを追加します。

   VB

```
' Create an instance of the class.
Dim user As New UserNameInfo("Moore, Bobby")
' Capitalize the value of the property.
user.Capitalize()
' Display the value of the property.
MsgBox("The original UserName is: " & user.UserName)
' Change the value of the property.
user.UserName = "Worden, Joe"
' Redisplay the value of the property.
MsgBox("The new UserName is: " & user.UserName)
```

アプリケーションを実行するには

1. F5 キーを押してアプリケーションを実行します。フォームのボタンをクリックしてテスト プロシージャを呼び出します。プロシージャがオブジェクトの `Capitalize` メソッドを呼び出したため、元の `UserName` が "MOORE, BOBBY" であるというメッセージが表示されます。

2. [OK] をクリックしてメッセージ ボックスを閉じます。`Button1 Click` プロシージャは `UserName` プロパティの値を変更し、変更後の `UserName` の値が "Worden, Joe" であるというメッセージを表示します。

参照
処理手順
方法 : イベントをクラスに追加する
関連項目
Imports ステートメント
概念
クラスとモジュール

その他の技術情報

クラスについて

クラス、プロパティ、フィールド、およびメソッド

クラスとモジュール

クラスとモジュールはどちらも、内部に定義された項目をカプセル化する参照型ですが、他のプロシージャから項目にアクセスする方法が異なります。

クラスとモジュールの違い

クラスとモジュールの主な違いは、クラスはオブジェクトとしてインスタンス化でき、標準モジュールはインスタンス化できないことです。標準モジュールのデータには 1 つのコピーしか存在しません。したがって、プログラムのある部分で標準モジュールのパブリック変数を変更した後、プログラムの別の部分で同じ変数を読み取ると、変更後の値が取得されます。一方、オブジェクト データは、オブジェクトのインスタンスごとに存在します。また、標準モジュールとは異なり、クラスはインターフェイスを実装できます。

メモ：

Shared 修飾子をクラス メンバに適用すると、そのクラス メンバは、クラスの特定のインスタンスではなくクラス自身に関連付けられます。そのメンバに対しては、モジュール メンバにアクセスするのと同じ方法で、クラス名によって直接アクセスできます。共有メンバの詳細については、「Visual Basic の共有メンバ」を参照してください。

クラスとモジュールでは、メンバに使用されるスコープも異なります。クラス内で定義されたメンバは、クラスの特定のインスタンス内がスコープとなり、そのオブジェクトの有効期間にだけ存在します。クラスの外側からクラス メンバにアクセスするには、*Object.Member* の形式の完全修飾名を使用する必要があります。

一方、モジュール内で宣言したメンバは既定でパブリック アクセスとなり、そのモジュールにアクセスできる任意のコードからアクセスできます。つまり、標準モジュールの変数は、プロジェクトのどこからでも参照でき、プログラムが実行されている間ずっと存在するため、事実上のグローバル変数です。

参照

関連項目

Shared (Visual Basic)

概念

構造体とクラス

Visual Basic の共有メンバ

その他の技術情報

クラスについて

オブジェクトの有効期間：オブジェクトの作成と破棄

クラスのインスタンス、つまりオブジェクトを作成するには **New** キーワードを使用します。通常は、新しいオブジェクトを使用する前に初期化する必要があります。一般的な初期化処理としては、ファイルを開く、データベースに接続する、レジストリ キーの値を読み取るなどがあります。Microsoft Visual Basic 2005 では、新しいオブジェクトの初期化処理をコンストラクタと呼ばれるプロシージャによって制御します。コンストラクタとは、初期化処理を制御するための特殊なメソッドです。

スコープを離れたオブジェクトは、共通言語ランタイム (CLR) によって解放されます。Visual Basic 2005 では、システム リソースの解放をデストラクタと呼ばれるプロシージャによって制御します。コンストラクタとデストラクタは、堅牢で予測可能なクラス ライブラリを作成するのに役立ちます。

Sub New と Sub Finalize

Visual Basic 2005 では、**Sub New** プロシージャと **Sub Finalize** プロシージャがオブジェクトの初期化と破棄を行います。これは、Visual Basic 6.0 以前のバージョンで使用されていた **Class_Initialize** メソッドと **Class_Terminate** メソッドの代わりになります。**Class_Initialize** とは異なり、**Sub New** コンストラクタはクラスが作成されるときに 1 回だけ実行されます。同じクラスまたは派生クラスの別のコンストラクタの 1 行目に記述して呼び出す以外に、これを明示的に呼び出す方法はありません。また、**Sub New** メソッド内のコードはクラス内の他のどのコードよりも先に実行されます。クラスの **Sub New** プロシージャを明示的に定義しなかった場合、Visual Basic 2005 は実行時に **Sub New** コンストラクタを暗黙的に作成します。

オブジェクトを解放する前に、CLR は **Sub Finalize** プロシージャを定義するオブジェクトの **Finalize** メソッドを自動的に呼び出します。**Finalize** メソッドには、オブジェクトを破棄する直前に実行する必要があるコードを記述できます。これらのコードでは、ファイルを閉じたり、状態情報を保存したりします。**Sub Finalize** の実行には、わずかながらパフォーマンス上の不利があります。そのため、オブジェクトを明示的に解放する必要がある場合にのみ **Sub Finalize** メソッドを定義してください。

メモ：
オペレーティング システムが CLR 環境の外部で直接実行しているオブジェクトをアンマネージ オブジェクトといいますが、CLR 内のガベージ コレクタはアンマネージ オブジェクトを破棄しません (できません)。これは、アンマネージ オブジェクトは種類ごとに異なる方法で破棄する必要があるからです。そのような情報はアンマネージ オブジェクトに直接関連付けられているわけではなく、オブジェクトのドキュメントで調べる必要があります。アンマネージ オブジェクトを使用するクラスは、**Finalize** メソッドを使用してオブジェクトを破棄する必要があります。

Finalize デストラクタは、属しているクラスまたは派生クラスからのみ呼び出し可能なプロテクト メソッドです。オブジェクトが破棄されるときに、システムは **Finalize** を自動的に呼び出します。したがって、派生クラスの **Finalize** 実装の外部から、明示的に **Finalize** を呼び出す必要はありません。

Class_Terminate はオブジェクトが nothing に設定されるとすぐに実行されましたが、Visual Basic 2005 では、オブジェクトがスコープ外になってから **Finalize** デストラクタが呼び出されるまでには遅延があります。Visual Basic 2005 には、Dispose というもう 1 種類のデスクトラクタがあります。このデストラクタは、リソースをすぐに解放する必要があるときにいつでも明示的に呼び出すことができます。

メモ：
Finalize デストラクタからは例外をスローしません。アプリケーションは例外を処理できないので、それが原因で異常終了する可能性があります。

IDisposable インターフェイス

クラス インスタンスは、Windows ハンドルやデータベース接続など、CLR では管理されないリソースを制御する場合があります。このようなリソースはクラスの **Finalize** メソッド内で破棄する必要があります。これにより、オブジェクトがガベージ コレクタによって破棄されるときにリソースが解放されます。ただし、ガベージ コレクタがオブジェクトを破棄するのは、CLR がより多くの空きメモリを必要とするときに限られます。そのため、オブジェクトがスコープ外になった後、しばらく経ってもリソースが解放されない可能性があります。

クラスが IDisposable インターフェイスを実装している場合は、ガベージ コレクションを補うために、システム リソースをアクティブに管理するしくみを用意できます。**IDisposable** には、クライアントがオブジェクトの使用を終了するときに呼び出す **Dispose** メソッドが含まれています。**Dispose** メソッドを使用すると、即座にリソースを解放して、ファイルやデータベース接続を閉じるなどのタスクを実行できます。**Finalize** デストラクタとは異なり、**Dispose** メソッドは自動的には呼び出されません。クラスのクライアントは、リソースを即座に解放するときに、明示的に **Dispose** を呼び出す必要があります。

IDisposable の実装

IDisposable インターフェイスを実装するクラスには、次のコード セクションを含める必要があります。

- オブジェクトが破棄されたかどうかを追跡するためのフィールド。

```
Protected disposed As Boolean = False
```

- クラスのリソースを解放するオーバーロードされた **Dispose** メソッド。このメソッドは基本クラスの **Dispose** メソッドおよび **Finalize** メソッドから呼び出す必要があります。

```
Protected Overridable Sub Dispose(ByVal disposing As Boolean)
    If Not Me.disposed Then
        If disposing Then
            ' Insert code to free unmanaged resources.
        End If
        ' Insert code to free shared resources.
    End If
    Me.disposed = True
End Sub
```

- **Dispose** の実装。次のコードだけを含みます。

```
Public Sub Dispose() Implements IDisposable.Dispose
    Dispose(True)
    GC.SuppressFinalize(Me)
End Sub
```

- オーバーライドされた **Finalize** メソッド。次のコードだけを含みます。

```
Protected Overrides Sub Finalize()
    Dispose(False)
    MyBase.Finalize()
End Sub
```

IDisposable を実装するクラスからの派生

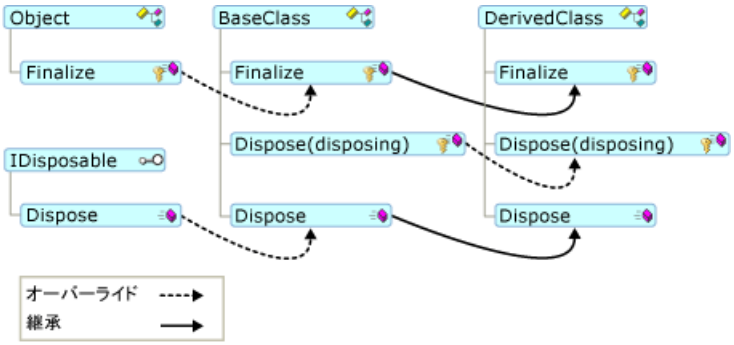
IDisposable インターフェイスを実装する基本クラスから派生したクラスは、基本メソッドをオーバーライドする必要はありません。ただし、追加リソースを使用していて、そのリソースを破棄する必要がある場合は別です。その場合は、派生クラスは基本クラスの `Dispose(disposing)` メソッドをオーバーライドして、派生クラスのリソースを破棄するようにします。このオーバーライドされたメソッドでは、基本クラスの `Dispose(disposing)` メソッドを呼び出す必要があります。

```
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If Not Me.disposed Then
        If disposing Then
            ' Insert code to free unmanaged resources.
        End If
        ' Insert code to free shared resources.
    End If
    MyBase.Dispose(disposing)
End Sub
```

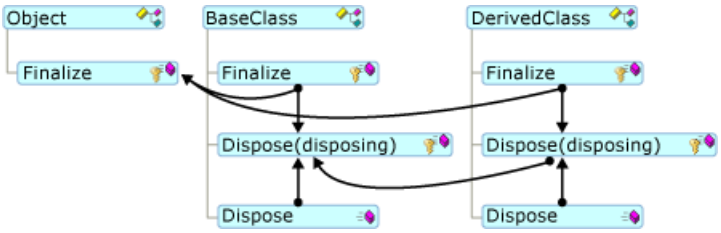
派生クラスでは、基本クラスの **Dispose** および **Finalize** メソッドをオーバーライドしないでください。派生クラスのインスタンスからこれらのメソッドを呼び出したときは、これらのメソッドの基本クラスの実装が、派生クラスのオーバーライドされた `Dispose(disposing)` メソッドを呼び出します。

ビジュアル化

次の図は、派生クラスの中でどのメソッドが継承され、どのメソッドがオーバーライドされているかを示しています。



この **Dispose Finalize** パターンに従えば、派生クラスおよび基本クラスのリソースが正しく破棄されます。次の図は、クラスが破棄されて終了されるときにどのメソッドが呼び出されるかを示しています。



ガベージ コレクションと Finalize デストラクタ

.NET Framework では、使われていないリソースを定期的に解放するために参照トレース ガベージ コレクションというシステムを使用しています。Visual Basic 6.0 以前のバージョンでは、参照カウントと呼ばれる別のシステムを使用してリソースを管理していました。どちらのシステムも同じ機能を自動的に実行しますが、いくつかの重要な違いがあります。

CLR は、システムが不要と判断したオブジェクトを定期的に破棄します。オブジェクトは、システム リソースが不足する場合は短い周期で解放され、そうでない場合は長い周期で解放されます。オブジェクトがスコープを失ってから CLR が解放するまでの遅延があるため、Visual Basic 6.0 以前のバージョンのオブジェクトとは異なり、オブジェクトがいつ破棄されるかを正確には判断できません。このような場合、オブジェクトは未決定の有効期間を持つことになります。ほとんどの場合、オブジェクトがスコープを失っても **Finalize** デストラクタがすぐに実行されない場合があることを覚えている限り、未決定の有効期間を考慮してアプリケーションの記述方法を変える必要はありません。

ガベージ コレクション システムとの違いは、**Nothing** の使用方法にもあります。Visual Basic 6.0 以前の参照カウント システムを利用するには、オブジェクト変数に **Nothing** を代入することで、その変数に保持されている参照を解放するという方法があります。この変数がオブジェクトへの最後の参照を保持している場合、オブジェクトのリソースは直ちに解放されます。Visual Basic 2005 では、この方法が有効な場合もありますが、参照されたオブジェクトのリソースはすぐには解放されません。リソースを直ちに解放するためには、オブジェクトの **Dispose** メソッドを使用します (使用可能な場合)。変数を **Nothing** に設定する必要があるのは、ガベージ コレクタが孤立したオブジェクトを検出するまでにかかる時間よりも、変数の有効期間が長い場合だけです。

参照

処理手順

方法 : Dispose Finalize パターンを実装する (Visual Basic)

関連項目

コンストラクタとデストラクタの使用方法
New (Visual Basic)
Nothing (Visual Basic)
Dispose

概念

コンポーネントの初期化と終了
Finalize メソッドおよびデストラクタ

その他の技術情報

言語の変更点 (Visual Basic 6.0 ユーザー向け)

方法：Dispose Finalize パターンを実装する (Visual Basic)

Dispose Finalize パターンは、オブジェクトが必要なくなった時点でリソースが確実に解放されるようにします。

使用例

以下に示すコード例の `ResourceClass` クラスは、マネージ リソースとアンマネージ リソースを利用した後で、**Dispose Finalize** パターンを使ってこれらのリソースを適切に破棄します。リソースと関数は、以下のとおりです。

- Dispose メソッドの実装。これは、このクラスのユーザーがクラス インスタンスを破棄できるようにします。このメソッドは、オブジェクトのリソースを破棄するために `Dispose(True)` を呼び出した後で、終了コードが 2 回実行されるのを防ぐために SuppressFinalize を呼び出します。
- 基本 Finalize メソッドのオーバーライド。これは、共通言語ランタイム (CLR: Common Language Runtime) のガベージ コレクタがクラス インスタンスを破棄できるようにします。このメソッドは、オブジェクトのリソースを破棄するために `Dispose(False)` を呼び出します。オブジェクトに対して **Dispose** が以前に呼び出されていると、そこから呼び出される **SuppressFinalize** により、ガベージ コレクタが **Finalize** メソッドを呼び出せなくなることに注意してください。
- **Dispose** メソッドのオーバーロード。これは、リソース破棄の処理を実行します。このメソッドは、`disposing` というブール値パラメータを受け取ります。この値は、オブジェクトの破棄が開始されたかどうかを示します。オブジェクトを破棄するときは、オブジェクトのすべてのリソースを破棄する必要があります。CLR のガベージ コレクタがオブジェクトを破棄するときには、アンマネージ リソースだけが破棄されます。ガベージ コレクタでは、マネージ リソースの破棄は必要な時点で自動的に実行されます。

詳細については、「オブジェクトの有効期間 : オブジェクトの作成と破棄」を参照してください。

VB

```
    Public Class ResourceClass
        Implements IDisposable

        Private managedResource As System.ComponentModel.Component
        Private unmanagedResource As IntPtr
        Protected disposed As Boolean = False

        Public Sub New()
            ' Insert appropriate constructor code here.
        End Sub

        Protected Overridable Overloads Sub Dispose( _
            ByVal disposing As Boolean)
            If Not Me.disposed Then
                If disposing Then
                    managedResource.Dispose()
                End If
                ' Add code here to release the unmanaged resource.
                unmanagedResource = IntPtr.Zero
                ' Note that this is not thread safe.
            End If
            Me.disposed = True
        End Sub

        Public Sub AnyOtherMethods()
            If Me.disposed Then
                Throw New ObjectDisposedException(Me.GetType().ToString, _
                    "This object has been disposed.")
            End If
        End Sub

#Region " IDisposable Support "
        ' Do not change or add Overridable to these methods.
        ' Put cleanup code in Dispose(ByVal disposing As Boolean).
        Public Overloads Sub Dispose() Implements IDisposable.Dispose
            Dispose(True)
            GC.SuppressFinalize(Me)
        End Sub
```

```
        Protected Overrides Sub Finalize()
            Dispose(False)
            MyBase.Finalize()
        End Sub
#End Region
    End Class
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System 名前空間および System.ComponentModel 名前空間のメンバへのアクセス。コード内でメンバ名を完全に修飾していない場合は、**Imports** ステートメントを追加します。詳細については、「Imports ステートメント」を参照してください。

コードには以下の変更を加える必要があります。

- `ResourceClass` を、**IDisposable** を実装するクラスの名前に置き換えます。
- 破棄される可能性のあるリソースを使用するすべてのメソッドで、`AnyOtherMethods` 内のテストを使用します。
- `managedResource` 宣言を、破棄する必要があるクラス内でマネージ オブジェクトの宣言に置き換えます。**IDisposable** を実装するか **Close** メソッドを持つクラスは、多くの場合破棄する必要があります。**Dispose** メソッドでは、これらのオブジェクトを閉じるか破棄します。
- `unManagedResource` 宣言を、破棄する必要があるクラス内でアンマネージ オブジェクトの宣言に置き換えます。これらのオブジェクトを破棄するために使用するメソッドは、オブジェクトの定義によって異なります。詳細については、オブジェクトに関するドキュメントを参照してください。

堅牢性の高いプログラム

Dispose メソッドを呼び出した後は、コレクションに含まれているオブジェクトは無効になります。オブジェクトになんらかの操作を実行する前に、`disposed` フィールドをテストする必要があります。具体的な方法については、コード例の `AnyOtherMethods` メソッドを参照してください。

参照

関連項目

IDisposable

概念

Dispose メソッドの実装

その他の技術情報

ガベージ コレクション

コンストラクタとデストラクタの使用方法

コンストラクタとデストラクタは、オブジェクトの構築と破棄を制御します。

コンストラクタ

クラスのコンストラクタを作成するには、クラス定義の任意の場所に **Sub New** というプロシージャを作成します。パラメータ化されたコンストラクタを作成するには、他のプロシージャと同じように、**Sub New** への引数の名前とデータ型を指定します。次に、コード例を示します。

VB

```
Sub New(ByVal s As String)
```

コンストラクタは、次のコードに示すように、オーバーロードされる場合があります。

VB

```
Sub New(ByVal s As String, i As Integer)
```

パラメータを受け取らないコンストラクタに基本クラスからアクセス可能な場合を除き、他のクラスから派生したクラスを定義するときには、コンストラクタの 1 行目で基本クラスのコンストラクタを呼び出す必要があります。上のコンストラクタを持つ基本クラスの呼び出しは、`MyBase.New(s)` のようになります。それ以外の場合、`MyBase.New` は省略可能で、Visual Basic のランタイムが暗黙的に呼び出します。

親オブジェクトのコンストラクタを呼び出すコードを記述したら、**Sub New** プロシージャに初期化コードを自由に追加できます。**Sub New** はパラメータ化されたコンストラクタとして呼び出されると、引数を受け取ることができます。これらのパラメータは、コンストラクタを呼び出すプロシージャから渡されます。たとえば、`Dim AnObject As New ThisClass(X)` のように指定します。

デストラクタ

Dispose と **Finalize** を使用して基本クラスのリソースを解放する方法は、次のコードのようになります。

メモ：

IDisposable を実装する際には、オブジェクトの有効期間 : オブジェクトの作成と破棄 に紹介しているガイドラインに従う必要があります。

VB

```
      ' Design pattern for a base class.
      Public Class Base
          Implements IDisposable

          ' Keep track of when the object is disposed.
          Protected disposed As Boolean = False

          ' This method disposes the base object's resources.
          Protected Overridable Sub Dispose(ByVal disposing As Boolean)
              If Not Me.disposed Then
                  If disposing Then
                      ' Insert code to free unmanaged resources.
                  End If
                  ' Insert code to free shared resources.
              End If
              Me.disposed = True
          End Sub

  #Region " IDisposable Support "
          ' Do not change or add Overridable to these methods.
          ' Put cleanup code in Dispose(ByVal disposing As Boolean).
          Public Sub Dispose() Implements IDisposable.Dispose
              Dispose(True)
              GC.SuppressFinalize(Me)
          End Sub
          Protected Overrides Sub Finalize()
              Dispose(False)
```

```
            MyBase.Finalize()
        End Sub
#End Region
    End Class
```

Dispose と **Finalize** を使用して派生クラスのリソースを解放する方法は、次のコードのようになります。

VB

```
' Design pattern for a derived class.
Public Class Derived
    Inherits Base

    ' This method disposes the derived object's resources.
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If Not Me.disposed Then
            If disposing Then
                ' Insert code to free unmanaged resources.
            End If
            ' Insert code to free shared resources.
        End If
        MyBase.Dispose(disposing)
    End Sub

    ' The derived class does not have a Finalize method
    ' or a Dispose method with parameters because it inherits
    ' them from the base class.
End Class
```

次のコードは、同じ機能を持つ **Using** ブロックと **Try...Finally** ブロックを使用して、**Dispose** デストラクタに共通するデザイン パターンを示しています。

VB

```
Sub DemonstrateUsing()
    Using d As New Derived
        ' Code to use the Derived object goes here.
    End Using
End Sub

Sub DemonstrateTry()
    Dim d As Derived = Nothing
    Try
        d = New Derived
        ' Code to use the Derived object goes here.
    Finally
        ' Call the Dispose method when done, even if there is an exception.
        If Not d Is Nothing Then
            d.Dispose()
        End If
    End Try
End Sub
```

次の例では、パラメータ化されたコンストラクタを使用してオブジェクトを作成し、オブジェクトが不要になったときにデストラクタを呼び出しています。

メモ：

この例では、ガベージ コレクタがどのメソッドを呼び出してメソッドを破棄するかを示すために Collect を使用していますが、通常は共通言語ランタイム (CLR: Common Language Runtime) がガベージ コレクションを管理します。

VB

```
Sub TestConstructorsAndDestructors()
```

```
    ' Demonstrate how the Using statement calls the Dispose method.
    Using AnObject As New ThisClass(6)
        ' Place statements here that use the object.
        MsgBox("The value of ThisProperty after being initialized " & _
        " by the constructor is " & AnObject.ThisProperty & ".")
    End Using

    ' Demonstrate how the garbage collector calls the Finalize method.
    Dim AnObject2 As New ThisClass(6)
    AnObject2 = Nothing
    GC.Collect()
End Sub

Public Class BaseClass
    Sub New()
        MsgBox("BaseClass is initializing with Sub New.")
    End Sub

    Protected Overrides Sub Finalize()
        MsgBox("BaseClass is shutting down with Sub Finalize.")
        ' Place final cleanup tasks here.
        MyBase.Finalize()
    End Sub
End Class

Public Class ThisClass
    Inherits BaseClass
    Implements IDisposable

    Sub New(ByVal SomeValue As Integer)
        ' Call MyBase.New if this is a derived class.
        MyBase.New()
        MsgBox("ThisClass is initializing with Sub New.")
        ' Place initialization statements here.
        ThisPropertyValue = SomeValue
    End Sub

    Private ThisPropertyValue As Integer
    Property ThisProperty() As Integer
        Get
            CheckIfDisposed()
            ThisProperty = ThisPropertyValue
        End Get
        Set(ByVal Value As Integer)
            CheckIfDisposed()
            ThisPropertyValue = Value
        End Set
    End Property

    Protected Overrides Sub Finalize()
        MsgBox("ThisClass is shutting down with Sub Finalize.")
        Dispose(False)
    End Sub

    ' Do not add Overridable to this method.
    Public Overloads Sub Dispose() Implements IDisposable.Dispose
        MsgBox("ThisClass is shutting down with Sub Dispose.")
        Dispose(True)
        GC.SuppressFinalize(Me)
    End Sub

    Private disposed As Boolean = False
    Public Sub CheckIfDisposed()
        If Me.disposed Then
            Throw New ObjectDisposedException(Me.GetType().ToString, _
            "This object has been disposed.")
        End If
    End Sub
```

```
    Protected Overridable Overloads Sub Dispose( _
    ByVal disposing As Boolean)
        MsgBox("ThisClass is shutting down with the Sub Dispose overload.")
        ' Place final cleanup tasks here.
        If Not Me.disposed Then
            If disposing Then
                ' Dispose of any managed resources.
            End If
            ' Dispose of any unmanaged resource.

            ' Call MyBase.Finalize if this is a derived class,
            ' and the base class does not implement Dispose.
            MyBase.Finalize()
        End If
        Me.disposed = True
    End Sub

End Class
```

この例を実行すると、ThisClass クラスは BaseClass の **Sub New** コンストラクタを呼び出します。基本クラスのコンストラクタの実行が終了すると、ThisClass クラスは **Sub New** の残りのステートメントを実行し、ThisProperty プロパティの値を初期化します。

クラスが不要になると、ThisClass で **Dispose** デストラクタが呼び出されます。

この例では次のように表示されます。

```
BaseClass is initializing with Sub New.

ThisClass is initializing with Sub New.

The value of ThisProperty after being initialized by the constructor is 6.

ThisClass is shutting down with Sub Dispose.

ThisClass is shutting down with the Sub Dispose overload.

BaseClass is shutting down with Sub Finalize.

BaseClass is initializing with Sub New.

ThisClass is initializing with Sub New.

ThisClass is shutting down with Sub Finalize.

ThisClass is shutting down with the Sub Dispose overload.

BaseClass is shutting down with Sub Finalize.
```

参照

関連項目

Dispose

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄

Finalize メソッドおよびデストラクタ

クラスの階層構造における New メソッドと Finalize メソッドの動作

クラス、プロパティ、フィールド、およびメソッド

通常、フィールドとプロパティはオブジェクトに関する情報を表し、メソッドはオブジェクトが実行できる動作を表します。

次に示す各トピックでは、クラスにプロパティ、フィールド、およびメソッドを追加する方法について説明し、これらの項目に関連する問題の解決策について説明します。

このセクションの内容

方法 : フィールドおよびプロパティをクラスに追加する

フィールドおよびプロパティの宣言について説明します。

プロパティとプロパティ プロシージャ

Property プロシージャの動作について、および一般的なプロパティ型を実装する方法について説明します。

プロパティ プロシージャとフィールド

クラスのデータを保持するために、フィールドとプロパティのどちらを使用するかについて説明します。

クラス メソッド

クラスに追加するパブリック プロシージャについて説明します。

プロパティとメソッド

必要な機能をプロパティとメソッドのどちらで実装するかについて説明します。

既定のプロパティ

特定のプロパティが指定されていないときに使用されるプロパティについて説明します。

オーバーロードされたプロパティとメソッド

同じ名前を持ち、異なるデータ型で処理される、複数のプロパティまたはメソッドを定義する方法について説明します。

プロパティとメソッドのオーバーライド

継承したプロパティやメソッドの再定義について説明します。

関連するセクション

オブジェクトの作成と使用

クラスのインスタンスを作成および使用する方法について説明します。

Visual Basic におけるイベント

イベントを宣言および使用する方法について説明します。

Visual Basic でのデリゲート

デリゲートを宣言および使用する方法について説明します。

コレクションによるオブジェクトの管理

コレクションを使用してオブジェクトを格納および取得する方法について説明します。

方法：フィールドおよびプロパティをクラスに追加する

フィールドおよびプロパティを使用して、オブジェクトに情報を格納できます。フィールドおよびプロパティは、クライアント アプリケーションからはほとんど見分けることができませんが、クラス内部では個別に宣言されます。フィールドは、クラスによって公開されるパブリック変数です。プロパティは、**Property** プロシージャを使用して、値を設定する方法や値を返す方法を制御します。

フィールドをクラスに追加するには

- クラス定義でパブリック変数を宣言します。コード例を次に示します。

  VB

```
Class ThisClass
    Public ThisField As String
End Class
```

プロパティをクラスに追加するには

1. クラス内部で、プロパティの値を格納するローカル変数を定義します。この手順は、プロパティ自体にストレージが割り当てられないために必要となります。値が直接変更されないようにするには、プロパティ値を格納する変数を **Private** で宣言します。
2. プロパティの宣言に、**Public** や **Shared** などの適切な修飾子を指定します。**Property** キーワードを使用して、プロパティの名前と、プロパティが格納および返すデータ型を宣言します。
3. プロパティ定義で、**Get** プロパティ プロシージャおよび **Set** プロパティ プロシージャを定義します。**Get** プロパティ プロシージャは、プロパティの値を返すために使用され、構文上は関数とほぼ同じです。これらは、引数を受け取りません。また、プロパティ値を格納するために使用されるクラス内で宣言されたプライベート ローカル変数の値を返すために使用できます。**Set** プロパティ プロシージャは、プロパティの値を設定するために使用され、プロパティ自体と同じデータ型を持つ、通常 `Value` と呼ばれるパラメータを持っています。プロパティの値を変更するときには、**Set** プロパティ プロシージャに `Value` が渡され、妥当性を検査されてローカル変数に格納されます。
4. **End Get** ステートメントおよび **End Set** ステートメントを必要に応じて使用して、**Get** プロパティ プロシージャおよび **Set** プロパティ プロシージャを終了します。
5. **End Property** ステートメントでプロパティ ブロックを終了します。

> **メモ：**
>
> Visual Studio 統合開発環境 (IDE: Integrated Development Environment) で作業している場合は、空の **Get** プロパティ プロシージャおよび **Set** プロパティ プロシージャを作成できます。「**Property** *PropName* **As** *DataType* (*PropName* はプロパティの名前を表し、*DataType* は **Integer** などの特定のデータ型を表します)」と入力すると、プロパティ プロシージャがコード エディタに表示されます。

クラス内でプロパティを宣言するコード例を次に示します。

VB

```
Class ThisClass
    Private m_PropVal As String
    Public Property One() As String
        Get
            ' Return the value stored in the local variable.
            Return m_PropVal
        End Get
        Set(ByVal Value As String)
            ' Store the value in a local variable.
            m_PropVal = Value
        End Set
    End Property
End Class
```

`ThisClass` のインスタンスを作成して、プロパティの値を `One` に設定するときに、**Set** プロパティ プロシージャが呼び出され、値が `Value` パラメータに渡されます。渡された値は、`m_PropVal` というローカル変数に格納されます。このプロパティの値を取得するときは、**Get** プロパティ プロシージャが関数のように呼び出され、ローカル変数 `m_PropVal`. に格納されている値が返されます。

参照

処理手順

方法 : イベントをクラスに追加する

関連項目

Property ステートメント

Public (Visual Basic)

Shared (Visual Basic)

概念

プロパティとプロパティ プロシージャ

プロパティとプロパティ プロシージャ

プロパティとフィールドのどちらを使用しても、オブジェクトに情報を格納できます。フィールドは単なるパブリック変数ですが、プロパティは、プロパティ プロシージャを使用して、値を設定する方法や値を返す方法を制御します。プロパティ プロシージャは、プロパティ定義内で宣言されるコードのブロックです。プロパティ値が設定または取得されるときにコードを実行する場合に使用できます。

Visual Basic には、2 種類のプロパティ プロシージャがあります。1 つはプロパティ値を取得するための **Get** プロパティ プロシージャ、もう 1 つはプロパティに値を代入するための **Set** プロパティ プロシージャです。たとえば、銀行口座の残高を格納するプロパティで **Get** プロパティ プロシージャのコードを使用すると、利子を加え、サービス料を確認してから、利用可能な残高を返すことができます。また、**Set** プロパティ プロシージャを使用すると、残高を確認することや、不正確に更新されるのを防ぐことができます。このように、プロパティ プロシージャによって、オブジェクトのデータを保護および検証できます。

フィールドの値およびプロパティの値を検証する方法の比較を次のコード例に示します。

VB

```
Protected Sub TestFieldsAndProperties()
    ' Assume, for this example, that the only valid values for
    ' the field and property are numbers less than 10.
    Dim NewClass As New ThisClass

    ' Test data validation.

    ' Works because there is no data validation.
    NewClass.ThisField = 36
    ' Will print 36.
    MsgBox("ThisField = " & NewClass.ThisField)

    ' The attempt to set the field to a value greater than 10 will silently fail.
    NewClass.ThisProperty = 36
    ' The next statement will print the old value of 0 instead.
    MsgBox("ThisProperty = " & NewClass.ThisProperty)
End Sub

Public Class ThisClass
    ' Declare a field.
    Public ThisField As Integer
    ' Field used for Property Set operations.
    Private thisPropertyValue As Integer = 0
    ' Declare a property.
    Public Property ThisProperty() As Integer
        Get
            Return thisPropertyValue
        End Get
        Set(ByVal Value As Integer)
            ' Only allow Set operation for values less than 10.
            If Value < 10 Then thisPropertyValue = Value
        End Set
    End Property
End Class
```

`TestFieldsAndProperties` プロシージャは、クラスのインスタンスを作成し、フィールドおよびプロパティの値を設定および取得します。この例では、有効な数値は 10 未満です。フィールドの場合、代入された値を検証する方法がないため、フィールドの値は 36 に設定されてしまいます。プロパティの場合は、10 未満の数値に対してだけ代入が実行されるため、36 に設定しようとする操作は無視されます。

読み取り専用プロパティと書き込み専用プロパティ

ほとんどのプロパティには、**Get** プロパティ プロシージャと **Set** プロパティ プロシージャの両方があり、格納されている値の読み取りと変更に使用できます。ただし、**ReadOnly** 修飾子または **WriteOnly** 修飾子を使用すると、プロパティ値の変更または読み取りを制限できます。

読み取り専用プロパティには **Set** プロパティ プロシージャを指定できません。読み取り専用プロパティは、公開はするが変更は許可しない項目の場合に有用です。読み取り専用プロパティは、たとえば、コンピュータのプロセッサの速度を示す場合に使用できます。

書き込み専用プロパティには **Get** プロパティ プロシージャを指定できません。書き込み専用プロパティは、そのオブジェクト内に保存しておくことが望ましくないデータや保存できないデータを扱うオブジェクトを設定する場合に便利です。書き込み専用プロパティは、たとえば、パスワードを要

求するオブジェクトで、パスワードを格納せずにオブジェクトの状態のみ変更する場合に使用できます。

メモ：
Visual Basic の以前のバージョンでは、オブジェクトをプロパティに代入するときに使用する **Let** プロパティ プロシージャをサポートしていました。Visual Basic では、オブジェクトの代入は他の代入処理と同様に処理されるようになったため **Let** プロパティ プロシージャは必要なくなりました。

参照

関連項目

Get ステートメント
Set ステートメント (Visual Basic)
ReadOnly (Visual Basic)
WriteOnly

概念

Property プロシージャ
プロパティ プロシージャとフィールド

プロパティ プロシージャとフィールド

プロパティとフィールドはどちらも、オブジェクトの情報を格納および取得します。これらの類似性により、特定の状況でどちらを使用する方が適切であるかを判断するのが難しい場合があります。

次の場合はプロパティ プロシージャを使用します。

- 値を設定または取得する時期と方法を制御する必要がある場合。
- プロパティに評価する必要のある明確な値のセットがある場合。
- 値の設定により、`IsVisible` のプロパティなど、オブジェクトの状態が大きく変化する場合。
- プロパティの設定により、他の内部変数が変更されたり、他のプロパティの値が変更される場合。
- プロパティを設定または取得する前に、一連の手順を実行する必要がある場合。

次の場合はフィールドを使用します。

- 自己検証型の値である場合。たとえば、**True** や **False** 以外の値を **Boolean** 変数に代入すると、エラーが発生するか、データ変換が自動的に実行されます。
- データ型でサポートされる範囲にある値がすべて有効である場合。**Single** 型または **Double** 型のプロパティの多くがこれに当てはまります。
- プロパティが **String** データ型であり、文字列のサイズや値に制限がない場合。

参照

概念

クラスとモジュール
プロパティとプロパティ プロシージャ
Visual Basic におけるデータ型

クラス メソッド

クラスのメソッドとは、クラスで宣言されている **Sub** プロシージャまたは **Function** プロシージャです。たとえば、`Account` というクラスの `Withdrawal` メソッドを作成するには、次の **Public** 関数をクラス モジュールに追加します。

VB

```
Public Function WithDrawal(ByVal Amount As Decimal, _
      ByVal TransactionCode As Byte) As Double
    ' Add code here to perform the withdrawal,
    ' return a transaction code,
    ' or to raise an overdraft error.
End Function
```

共有メソッド

共有メソッドは、事前にクラスのインスタンスを作成しなくても、直接にクラスから呼び出すことができます。このため、メソッドをクラスの特定のインスタンスに関連付けたくない場合に便利です。共有メソッドの宣言では、**Overridable**、**NotOverridable**、および **MustOverride** の各修飾子は使用できません。また、モジュールで宣言されたメソッドは暗黙的に共有されるため、**Shared** 修飾子を明示的に使用することはできません。

例

VB

```
Class ShareClass
    Shared Sub SharedSub()
        MsgBox("Shared method.")
    End Sub
End Class

Sub Test()
    ' Call the method.
    ShareClass.SharedSub()
End Sub
```

実装の詳細の保護

クラスが内部で使用するユーティリティ プロシージャは、**Private**、**Protected**、または **Friend** として宣言する必要があります。このようなメソッドへのアクセスを制限することにより、オブジェクトに変更を加えた場合にオブジェクトを使用するコードに影響が及ぶのを防ぐことができるため、オブジェクトを使用する開発者を保護できます。

オブジェクトの実装の詳細を保護する機能は、カプセル化のもう 1 つの側面です。カプセル化を利用すると、メソッドを使用するコードを変更することなく、メソッドのパフォーマンスを強化したり、メソッドの実装方法を完全に変更したりできます。

参照

処理手順

方法 : イベントをクラスに追加する

関連項目

Overridable
NotOverridable
MustOverride
Shared (Visual Basic)
Public (Visual Basic)
Private (Visual Basic)
Protected (Visual Basic)
Friend (Visual Basic)

概念

プロパティとメソッド
Visual Basic の共有メンバ

プロパティとメソッド

プロパティとメソッドは、どちらも引数を受け取るプロシージャとして実装されるという点で似ています。一般に、プロパティはオブジェクトの格納データであり、メソッドはオブジェクトに実行を求めることができる処理です。いくつかのオブジェクトの特性は、`Name` のように明らかにプロパティであったり、`Move` や `Show` のように明らかにメソッドであったりします。また、どのクラス メンバをプロパティにし、どのクラス メンバをメソッドにしたらよいのかがわかりにくい場合もあります。たとえば、データの格納と取得を行うコレクション クラスの `Item` メソッドは、インデックス付きプロパティとして実装できます。その一方で、`Item` をメソッドとして実装することも十分考えられます。

プロパティの構文とメソッドの構文

クラス メンバの実装方法を決定するときには、まずその使い方について検討する必要があります。パラメータ化したプロパティから情報を取得する構文は、関数として実装されたメソッドで使用される構文とほとんど変わりませんが、値を変更するための構文は多少異なります。たとえば、クラス メンバをプロパティとして実装した場合の構文は、次のようになります。

ThisObject.ThisProperty(*Index*) = *NewValue*

一方、クラス メンバをメソッドとして実装する場合は、変更する値を引数にする必要があります。したがって、この場合の構文は次のようになります。

ThisObject.ThisProperty(*Index, NewValue*)

エラー メッセージ

クラス メンバの実装方法を選択する場合は、クラスが間違った方法で使われたときに生成されるメッセージの種類についても検討する必要があります。読み取り専用のプロパティに間違って値が割り当てられた場合に返されるエラー メッセージは、メソッドに対して同様の間違った呼び出しが行われた場合に返されるエラー メッセージとは異なります。クラス メンバを正しく実装すると、わかりやすいエラー メッセージを返すことができます。

参照

処理手順

方法 : フィールドおよびプロパティをクラスに追加する

関連項目

Item プロパティ (Collection オブジェクト)

概念

クラス メソッド

既定のプロパティ

引数を受け取るプロパティは、クラスの既定のプロパティとして宣言できます。既定のプロパティとは、オブジェクトに対して特定のプロパティが指定されていない場合に Visual Basic が使用するプロパティです。既定のプロパティを使うと、頻繁に使用するプロパティ名を省略してソース コードをコンパクトにできます。

既定のプロパティにするのに最も適しているのは、パラメータを受け取るプロパティのうちで最もよく使用するプロパティです。たとえば、コレクション クラスの `Item` プロパティは頻繁に使用されるので、既定のプロパティにするのに適しています。

既定のプロパティには、次のような規則があります。

- 1 つの型に指定できる既定のプロパティは、基本クラスから継承するプロパティも含めて 1 つだけです。この規則には例外が 1 つあります。基本クラスで定義された既定のプロパティを、派生クラスで定義された別の既定のプロパティでシャドウできます。
- 基本クラスの既定のプロパティが、派生クラスの既定以外のプロパティでシャドウされる場合でも、既定のプロパティの構文を使用することによって既定のプロパティにアクセスできます。
- 既定のプロパティを **Shared** や **Private** にすることはできません。
- オーバーロードされたプロパティを既定のプロパティにする場合は、オーバーロードされた同じ名前のプロパティをすべて **Default** に指定する必要があります。
- 既定のプロパティは、少なくとも 1 つの引数を受け取る必要があります。

例

次の例では、文字列配列を含むプロパティをクラスの既定のプロパティとして宣言しています。

VB

```
Class Class2
    ' Define a local variable to store the property value.
    Private PropertyValues As String()
    ' Define the default property.
    Default Public Property Prop1(ByVal Index As Integer) As String
        Get
            Return PropertyValues(Index)
        End Get
        Set(ByVal Value As String)
            If PropertyValues Is Nothing Then
                ' The array contains Nothing when first accessed.
                ReDim PropertyValues(0)
            Else
                ' Re-dimension the array to hold the new element.
                ReDim Preserve PropertyValues(UBound(PropertyValues) + 1)
            End If
            PropertyValues(Index) = Value
        End Set
    End Property
End Class
```

既定のプロパティへのアクセス

既定のプロパティにアクセスするときは、省略した構文を使用できます。次のコードでは、標準のプロパティの構文と既定のプロパティの構文の両方が使われています。

VB

```
Dim C As New Class2
' The first two lines of code access a property the standard way.

' Property assignment.
C.Prop1(0) = "Value One"
' Property retrieval.
MsgBox(C.Prop1(0))
```

```
' The following two lines of code use default property syntax.

' Property assignment.
C(1) = "Value Two"
' Property retrieval.
MsgBox(C(1))
```

参照

関連項目

Default (Visual Basic)

概念

アップグレードに関する推奨事項 : パラメータのない既定プロパティの解決

既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)

オーバーロードされたプロパティとメソッド

オーバーロードとは、異なる型の引数を持つ同じ名前のプロシージャ、インスタンス コンストラクタ、またはプロパティをクラスに複数作成することです。

オーバーロードの使用方法

別のデータ型を処理するプロシージャに同じ名前を付けるオブジェクト モデルを使用している場合は、特にオーバーロードが有効です。たとえば、複数のデータ型を表示できるクラスの `Display` プロシージャは、次のようになります。

VB

```
Overloads Sub Display(ByVal theChar As Char)
    ' Add code that displays Char data.
End Sub
Overloads Sub Display(ByVal theInteger As Integer)
    ' Add code that displays Integer data.
End Sub
Overloads Sub Display(ByVal theDouble As Double)
    ' Add code that displays Double data.
End Sub
```

オーバーロードを利用しない場合は、各プロシージャの動作が同じであっても、次のようにそれぞれに別の名前を付ける必要があります。

VB

```
Sub DisplayChar(ByVal theChar As Char)
    ' Add code that displays Char data.
End Sub
Sub DisplayInt(ByVal theInteger As Integer)
    ' Add code that displays Integer data.
End Sub
Sub DisplayDouble(ByVal theDouble As Double)
    ' Add code that displays Double data.
End Sub
```

オーバーロードを利用すると、使用するデータ型を選択できるようになるため、プロパティやメソッドの使い方が簡単になります。たとえば、上のオーバーロードされた `Display` メソッドを呼び出すには、以下のどのコード行でも使用できます。

VB

```
' Call Display with a literal of type Char.
Display("9"c)
' Call Display with a literal of type Integer.
Display(9)
' Call Display with a literal of type Double.
Display(9.9R)
```

実行時には、指定したパラメータのデータ型に基づいて正しいプロシージャが Visual Basic で呼び出されます。

メモ：

オーバーロード、オーバーライド、およびシャドウの概念は、似ているためにしばしば混同されます。詳細については、「Visual Basic のオブジェクトの概要」を参照してください。

オーバーロードの規則

オーバーロードされたクラスのメンバを作成するには、同じ名前を持つ複数のプロパティまたはメソッドを追加します。オーバーロードされた派生メンバを除き、オーバーロードされた各メンバは、それぞれパラメータ リストが異なっている必要があります。また、プロパティやプロシージャをオーバーロードするときには、以下の項目でそれぞれを区別することはできません。

- メンバに適用される **ByVal** や **ByRef** などの修飾子、またはメンバのパラメータ
- パラメータの名前
- プロシージャの戻り値の型

オーバーロードを行うときに **Overloads** キーワードは省略可能ですが、オーバーロードされるメンバのいずれかで **Overloads** キーワードを使用する場合には、同じ名前を持つ他のすべてのオーバーロードされるメンバにもこのキーワードを指定する必要があります。

派生クラスは、継承したメンバを同じパラメータ (および同じパラメータ型) を持つメンバでオーバーロードできます。これは名前とシグネチャによるシャドウと呼ばれます。名前とシグネチャによるシャドウで **Overloads** キーワードを使用した場合は、基本クラス内の実装の代わりにメンバの派生クラスの実装が使用され、そのメンバに対する他のすべてのオーバーロードが派生クラスのインスタンスで使用できるようになります。

継承したメンバを同じパラメータ (および同じパラメータ型) を持つメンバでオーバーロードするときに **Overloads** キーワードを省略した場合、そのオーバーロードは名前によるシャドウと呼ばれます。名前によるシャドウを行うと、メンバの継承した実装が置き換わります。派生クラスのインスタンスとその子孫は、他のすべてのオーバーロードを使用できなくなります。

Overloads 修飾子と **Shadows** 修飾子の両方を同じプロパティまたはメソッドで使用することはできません。

例

次の例で作成するオーバーロードされたメソッドは、金額を **String** または **Decimal** の値として受け取って、消費税を加えた額を文字列として返します。

この例を使ってオーバーロードされたメソッドを作成するには

1. 新しいプロジェクトを開き、`TaxClass` という名前のクラスを追加します。
2. `TaxClass` クラスに次のコードを追加します。

   VB

```
Public Class TaxClass
    Overloads Function TaxAmount(ByVal decPrice As Decimal, _
         ByVal TaxRate As Single) As String
        TaxAmount = "Price is a Decimal. Tax is $" & _
           (CStr(decPrice * TaxRate))
    End Function

    Overloads Function TaxAmount(ByVal strPrice As String, _
          ByVal TaxRate As Single) As String
        TaxAmount = "Price is a String. Tax is $" & _
           CStr((CDec(strPrice) * TaxRate))
    End Function
End Class
```

3. フォームに次のプロシージャを追加します。

   VB

```
Sub ShowTax()
    ' 8% tax rate.
    Const TaxRate As Single = 0.08
    ' $64.00 Purchase as a String.
    Dim strPrice As String = "64.00"
    ' $64.00 Purchase as a Decimal.
    Dim decPrice As Decimal = 64
    Dim aclass As New TaxClass
    'Call the same method with two different kinds of data.
    MsgBox(aclass.TaxAmount(strPrice, TaxRate))
    MsgBox(aclass.TaxAmount(decPrice, TaxRate))
End Sub
```

4. フォームにボタンを追加し、ボタンの `Button1_Click` イベントから `ShowTax` プロシージャを呼び出します。

5. プロジェクトを実行し、フォームのボタンをクリックして、オーバーロードされた `ShowTax` プロシージャをテストします。

実行時には、使用されているパラメータに対応する適切な関数がコンパイラによって選択されます。ボタンをクリックすると、オーバーロードされたメソッドがまず文字列の `Price` パラメータで呼び出され、"Price is a String.Tax is $5.12" というメッセージが表示されます。次に、**Decimal** 値で `TaxAmount` が呼び出され、"Price is a Decimal.Tax is $5.12" というメッセージが表示されます。

参照

関連項目

Sub ステートメント (Visual Basic)
Shadows
ByVal
ByRef
Overloads
Shadows

概念

Visual Basic におけるシャドウ
Visual Basic のオブジェクトの概要
プロパティとメソッドのオーバーライド

プロパティとメソッドのオーバーライド

派生クラスは、基本クラスで定義されているプロパティとメソッドを継承します。このことにより、派生クラスに適したプロパティやメソッドを必要に応じて再利用できるので便利です。基本クラスで **Overridable** キーワードが付けられているプロパティまたはメソッドの場合、派生クラスではそのメンバに対して新しい実装を定義できます。**Overrides** キーワードは、派生クラスでメンバを再定義することによってシャドウする場合に使用します。これは、メンバを "そのまま" 使用できない場合に便利です。

オーバーライドしたメンバは、実際にはポリモーフィズムを実装する目的でよく使われます。詳細については、「ポリモーフィズム」を参照してください。

メソッドをオーバーライドするときには、次の規則があります。

- オーバーライドできるメンバは、基本クラスで **Overridable** キーワードが使用されているメンバだけです。
- プロパティとメソッドは、既定で **NotOverridable** です。
- オーバーライドしたメンバは、基本クラスから継承したメンバと同じ引数を持っている必要があります。
- メンバの新しい実装で、メソッド名の前に **MyBase** を指定すると、親クラスの元の実装を呼び出すことができます。

> **メモ：**
>
> オーバーロード、オーバーライド、およびシャドウの概念は、似ているためにしばしば混同されます。詳細については、「Visual Basic のオブジェクトの概要」を参照してください。

例

給与支払いを処理するクラスを定義するとします。まず、通常の 1 週間分の給与を計算する `RunPayroll` メソッドを含む汎用の `Payroll` クラスを定義します。次に、`Payroll` を基本クラスとして、より特化した `BonusPayroll` クラスを作成します。これはボーナスの支給時に使用します。

`BonusPayroll` クラスでは、基本の `Payroll` クラスで定義されている `PayEmployee` メソッドを継承し、オーバーライドできます。

次の例では、基本クラス `Payroll` と派生クラス `BonusPayroll` を定義しています。`BonusPayroll` は、継承したメソッド `PayEmployee` をオーバーライドします。`RunPayroll` プロシージャは、`Payroll` オブジェクトと `BonusPayroll` オブジェクトを作成し、`Pay` 関数に渡します。この関数は、この 2 つのオブジェクトの `PayEmployee` メソッドを実行します。

VB

```
Const BonusRate As Decimal = 1.45D
Const PayRate As Decimal = 14.75D

Class Payroll
    Overridable Function PayEmployee( _
        ByVal HoursWorked As Decimal, _
        ByVal PayRate As Decimal) _
        As Decimal

        PayEmployee = HoursWorked * PayRate
    End Function
End Class

Class BonusPayroll
    Inherits Payroll
    Overrides Function PayEmployee( _
        ByVal HoursWorked As Decimal, _
        ByVal PayRate As Decimal) _
        As Decimal

        ' The following code calls the original method in the base
        ' class, and then modifies the returned value.
        PayEmployee = MyBase.PayEmployee(HoursWorked, PayRate) * BonusRate
    End Function
End Class

Sub RunPayroll()
```

```
        Dim PayrollItem As Payroll = New Payroll
        Dim BonusPayrollItem As New BonusPayroll
        Dim HoursWorked As Decimal = 40

        MsgBox("Normal pay is: " & _
            PayrollItem.PayEmployee(HoursWorked, PayRate))
        MsgBox("Pay with bonus is: " & _
            BonusPayrollItem.PayEmployee(HoursWorked, PayRate))
    End Sub
```

参照

概念

オーバーロードされたプロパティとメソッド

オーバーライド修飾子

Visual Basic におけるシャドウ

その他の技術情報

ポリモーフィズム

オーバーライド修飾子

基本クラスで **NotOverridable** 修飾子および **MustOverride** 修飾子を使用すると、プロパティおよびメソッドを派生クラスでオーバーライドする方法を制御できます。

NotOverridable 修飾子は、派生クラスでオーバーライドできない基本クラスのメソッドを定義します。**Overridable** 修飾子を指定しない限り、メソッドはすべて **NotOverridable** になります。**NotOverridable** 修飾子を使用すると、オーバーライドしたメソッドが派生クラスで再度オーバーライドされるのを防ぐことができます。

MustOverride 修飾子を使用して定義されているメソッドは、基本クラスには実装がないため、派生クラスで実装する必要があります。**MustOverride** メソッドを持つクラスには、**MustInherit** 修飾子を指定する必要があります。

例

VB

```
MustInherit Class BaseClass
    Public MustOverride Sub aProcedure()
End Class

Class DerivedClass
    Inherits BaseClass
    Public NotOverridable Overrides Sub aProcedure()
        ' Override a procedure inherited from the base class
        ' and mark it with the NotOverridable modifier so that
        ' it cannot be overridden in classes derived from this class.
    End Sub
End Class
```

参照

概念

オーバーロードされたプロパティとメソッド

オブジェクトの作成と使用

オブジェクトとは、1 つの単位として扱うことのできる、コードとデータの組み合わせです。オブジェクトは、Visual Basic のほとんどすべての作業で使用されます。たとえば、コントロールやコントロールを配置するフォームは、すべてオブジェクトです。ここでは、オブジェクトを何から生成してどのように使用するのかについて説明します。

このセクションの内容

Visual Basic のオブジェクトとその他のオブジェクト

Microsoft Word や Microsoft Excel などのアプリケーションのオブジェクトの使用方法を説明します。

方法 : プロパティを設定および取得する

プロパティ値の変更方法および取得方法を説明します。

方法 : メソッドを使用してアクションを実行する

メソッドの概要と使用方法を説明します。

方法 : フォームをオブジェクトとして処理する

フォームを通常のオブジェクトと同じように使用する方法を説明します。

方法 : フォームにアクセスする

変数を作成することなく vbprvb のフォームのメンバにアクセスする方法を示します。

方法 : New キーワードを使用する

フォームのインスタンス、クラス モジュールで定義されているクラスのインスタンス、およびコレクションのインスタンスを作成する方法を説明します。

リソースの管理

オブジェクトへの参照を解放してメモリやシステム リソースを節約する方法を説明します。

方法 : オブジェクトをプロシージャに渡す

オブジェクトを引数としてプロシージャに渡す方法を説明します。

チュートリアル : Visual Basic でのオブジェクトの永続化

シリアル化を使ってインスタンス間でオブジェクトのデータを永続化する方法を紹介します。これにより、値を格納し、オブジェクトが次にインスタンス化されたときにその値を取得できます。

方法 : Visual Basic でクラスを継承する

基本クラスを定義し、基本クラスを拡張したクラスを宣言する方法を示します。

関連するセクション

Visual Basic におけるオブジェクト

オブジェクト指向プログラミングの基本事項を説明します。

オブジェクトのグループの管理

オブジェクトの配列やコレクションを使用する方法を紹介します。

方法 : オブジェクトに対して複数のアクションを実行する

With...End With ステートメントを使ってプロパティに簡単にアクセスする方法を説明します。

Visual Basic のオブジェクトとその他のオブジェクト

Visual Basic で使うオブジェクトには、内部オブジェクトと外部オブジェクトがあります。たとえば、組み込みオブジェクトやプロジェクト内のクラスは内部オブジェクトであり、アセンブリや COM オブジェクトは外部オブジェクトです。

内部オブジェクト

組み込み (内蔵) オブジェクトとは、Visual Basic に初めから用意されているオブジェクトです。たとえば、整数型 (**Integer**) や倍精度浮動小数点数型 (**Double**) などのプリミティブなスカラ型や、配列型 (**Array**) や文字列型 (**String**) がこれに当たります。内部オブジェクトをプロジェクトで使う場合には、事前に参照を作成する必要はありません。

このほか、現在のプロジェクトのクラスのインスタンスも内部オブジェクトです。このようなクラスは、必要に応じてプロジェクト内のどこででも使用できます。また、アセンブリを作成する時に、ほかのアプリケーションから使用できるようにすることもできます。

外部オブジェクト

外部オブジェクトとは、既定の状態ではプロジェクトで使用できないほかのプロジェクトやアセンブリのオブジェクトです。外部オブジェクトをプロジェクトで使用するには、事前にプロジェクト参照を作成する必要があります。

アセンブリは、Visual Basic アプリケーションのオブジェクトの最も一般的なソースです。.NET Framework には、よく使用されるオブジェクトを含むアセンブリが含まれています。.NET Framework のオブジェクトの中には、組み込みオブジェクトとして扱われるものもありますが、ほとんどのアセンブリは、**Imports** ステートメントを使って明示的にインポートしないと使用できません。アセンブリは、Visual Basic や Visual C# など、共通言語仕様 (CLS: Common Language Specification) に準拠しているすべての言語で作成および使用できます。詳細については、「アセンブリ」を参照してください。

従来の Visual Basic プログラミングでは、オブジェクトのソースとして COM コンポーネントを使うのが一般的でしたが、現在では、新しいオブジェクトに対しては .NET Framework アセンブリを使うことをお勧めします。既存の COM コンポーネントをアプリケーションで使うこともできますが、COM オブジェクトにアクセスするには .NET Framework 相互運用性クラスを使う必要があります。COM ライブラリにアクセスするには、COM ライブラリ内で定義されている各 COM クラスの相互運用性クラスを含む相互運用性アセンブリを使う必要があります。詳細については、「COM 相互運用」を参照してください。

ネイティブな .NET Framework クラスおよび COM クラスにアクセスできるのに加え、たとえば Win32 API の関数のような、ダイナミック リンク ライブラリ (DLL: dynamic-link library) で定義されている関数を呼び出すこともできます。Visual Basic では、**Declare** ステートメントで宣言しておいた、DLL の中の関数を呼び出すことができます。また、Visual Basic では、DllImportAttribute 属性を使用して、CallingConvention、ExactSpelling、および SetLastError などに対して既定値を指定できるように、**Declare** ステートメンの機能が拡張されています。**Declare** ステートメントのパラメータには、MarshalAsAttribute 属性で注釈を付けることができます。この属性は、以前のバージョンの Visual Basic では不可能だった方法によるパラメータの変換をサポートします。

参照

処理手順

チュートリアル : COM オブジェクトによる継承の実装

方法 : プロパティを設定および取得する

関連項目

Declare ステートメント

Imports ステートメント

整数型 (Integer) (Visual Basic)

倍精度浮動小数点数型 (Double) (Visual Basic)

文字列型 (String) (Visual Basic)

DllImportAttribute

MarshalAsAttribute

概念

オブジェクト間の関係

その他の技術情報

オブジェクトの作成と使用

方法：プロパティを設定および取得する

Visual Basic では、フォームとコントロールのプロパティをプログラムによって実行時に設定できます。また、[プロパティ] ウィンドウを使用してデザインモードで設定することもできます。アセンブリからのオブジェクトなど、フォームやコントロール以外のオブジェクトのほとんどは、プログラムでしか設定できません。

設定と取得ができるプロパティは、読み取り/書き込みプロパティと呼ばれます。取得はできるが変更はできないプロパティは、読み取り専用プロパティと呼ばれます。書き込みはできるが取得はできないプロパティは、書き込み専用プロパティと呼ばれます。

オブジェクトの外観または動作を変更する場合に、プロパティの値を設定します。たとえば、テキスト ボックスの内容を変更するときは、テキスト ボックス コントロールの `Text` プロパティを変更します。

他のオブジェクトに値を割り当てるなど、次の処理に移る前にオブジェクトの状態を調べる場合は、プロパティの値を取得します。たとえば、テキスト ボックスの内容を変更する可能性のあるコードを実行する前に、テキスト ボックス コントロールの `Text` プロパティを使用して、テキスト ボックスの内容を取得できます。

プロパティ値を設定するには

- 次の構文を使います。

  Object.*property* = *expression*

  プロパティを設定するステートメントの例を次に示します。

  VB

```
' Set the Top property to 200 twips.
TextBox1.Top = 200
' Display the text box.
TextBox1.Visible = True
' Display 'hello' in the text box.
TextBox1.Text = "hello"
```

メモ：

プロパティを **ByRef** のパラメータに渡して設定することもできます。この場合、プロパティは、**ByRef** のパラメータが返す結果によって更新されます。

プロパティ値を取得するには

- 次の構文を使います。

  variable = *Object*.*property*

  変数にプロパティを割り当てずに、プロパティの値をより複雑な式の一部として取得することもできます。オプション ボタン コントロールの **Top** プロパティを変更するコードの例を次に示します。

  VB

```
RadioButton1.Top += 20
```

参照

処理手順

方法 : メソッドを使用してアクションを実行する

概念

Visual Basic のオブジェクトとその他のオブジェクト

その他の技術情報

オブジェクトの作成と使用

方法：メソッドを使用してアクションを実行する

メソッドは、オブジェクトに関連付けられたプロシージャです。フィールドやプロパティはオブジェクトに格納できる情報を表しますが、メソッドはオブジェクトが実行できるアクションを表します。メソッドでプロパティの値を操作することもできます。たとえば、ラジオの音量調整の動作を例にとると、`SetVolume` メソッドで `Volume` プロパティの値を変更できます。同様に、Visual Basic には、`Clear` メソッドおよび `Add` メソッドを使用して変更できる `List` プロパティが、リスト ボックスの項目にあります。

コード内でメソッドを使用する場合、メソッドが必要とする引数の数、およびそのメソッドで値を返すかどうかによって、ステートメントの記述方法が異なります。メソッドは、通常、サブルーチンや関数呼び出しと同じように使用します。つまり、モジュールのプロシージャと同じ方法でメソッドを呼び出すことができます。ただし、メソッドの場合は、メソッドが呼び出されるオブジェクト インスタンスを指定する式で修飾できます。修飾しない場合、インスタンスは暗黙的に **Me** 変数となります。

引数を必要としないメソッドを使用するには

- 次の構文を使います。

  Object.method()

  ピクチャ ボックスを再描画する `Refresh` メソッドの例を次に示します。

  VB

```
' Force the control to repaint.
PictureBox1.Refresh()
```

> **メモ：**
>
> `Refresh` メソッドなど、引数を使用せず、値も返さないメソッドもあります。

複数の引数を必要とするメソッドを使用するには

- かっこの中に、引数をコンマで区切って指定します。表示するメッセージを指定する引数とメッセージ ボックスのスタイルを指定する引数を使用する、**MsgBox** メソッドの例を次に示します。

  VB

```
MsgBox("Database update complete", _
       MsgBoxStyle.OKOnly Or MsgBoxStyle.Exclamation, _
       "My Application")
```

値を返すメソッドを使用するには

- 戻り値を変数に代入するか、またはメソッドの呼び出しを直接他の呼び出しのパラメータとして使用します。次のコードは、戻り値を格納します。

  VB

```
Dim Response As MsgBoxResult
Response = MsgBox("Do you want to exit?", _
                  MsgBoxStyle.YesNo Or MsgBoxStyle.Question, _
                  "My Application")
```

  この例では、**MsgBox** に対する引数として、**Len** メソッドから返された値を使用します。

  VB

```
Dim TestStr As String = "Some String"
' Display the string "String length is : 11".
```

```
MsgBox("String length is : " & Len(TestStr))
```

参照

処理手順

方法 : プロパティを設定および取得する

概念

オブジェクト間の関係

その他の技術情報

オブジェクトの作成と使用

方法：フォームをオブジェクトとして処理する

フォームは、アプリケーションのユーザー インターフェイスを構成するグラフィカル オブジェクトです。Visual Basic では、クラスがフォームの表示形式および機能を定義します。実行時にフォームが表示されると、Visual Basic では、Form クラスのインスタンスが作成されます。このインスタンスは、他のオブジェクトと同じように使用できます。フォームに独自のメソッドとプロパティを追加して、アプリケーションの他のフォームやクラスからアクセスすることもできます。

フォームに新しいメソッドを作成するには

- **Public** 宣言したプロシージャを追加します。コードの例を次に示します。

  VB

```
' Create a custom method on a form.
Public Sub PrintMyJob()
    ' Insert the code for your method here.
End Sub
```

フォームに新しいフィールドを追加するには

- フォーム モジュールでパブリック変数を宣言します。コードの例を次に示します。

  VB

```
Public IDNumber As Integer
```

異なるフォームのメソッドにアクセスするには

1. アクセスするメソッドが登録されているフォームの新しいインスタンスを作成します。フォーム名の参照は、フォームが属しているクラスの参照となります。オブジェクト自体の参照ではありません。

   メモ：

   Visual Basic には、各フォーム クラスごとにフォームと同じ名前の暗黙的グローバル変数が用意されています。詳細については、「方法：フォームにアクセスする」を参照してください。

2. フォームをオブジェクト変数に割り当てます。オブジェクト変数は、フォーム クラスの新しいインスタンスを参照します。

   `PrintMyJob` プロシージャを正しく呼び出すコードの例を次に示します。

   VB

```
Dim newForm1 As New Form1
newForm1.PrintMyJob()
```

   上記の例では、新しいフォームは表示されません。フォーム オブジェクトのメソッドを使用するときに、フォーム オブジェクトを表示する必要はありません。新しいフォームを表示するには、次のコードを追加します。

   VB

```
newForm1.Show()
```

参照

処理手順

方法：New キーワードを使用する

概念

Property プロシージャ

方法：フォームにアクセスする

Visual Basic のフォームのメンバには、変数を作成することなくアクセスできます。次の例では、フォームの色を変えることによってこれを示します。

フォームへのアクセス

Form1 にアクセスするには

1. プロジェクトに System.Drawing 名前空間への参照が含まれていることを確認します。これは、フォームにアクセスするためにではなく、色を設定するために必要です。
2. `Form1` に対して色を直接変更します。
3. `Form1` の Show メソッドを直接呼び出します。

   VB

   ```
   Public Sub ChangeForm1Colors()
       Form1.ForeColor = System.Drawing.Color.Coral
       Form1.BackColor = System.Drawing.Color.Cyan
       Form1.Show()
   End Sub
   ```

   `Form1` がまだ存在しない場合は、Visual Basic によって作成されます。変数を宣言する必要はありません。

フォームの追加のインスタンスを作成する

既存のフォームにアクセスするのではなく、フォームを新規作成する場合には、変数を宣言し、**New** キーワードを使用して変数を初期化することによって、新規作成できます。

Form1 の追加のコピーを作成するには

1. プロジェクトに **System.Drawing** 名前空間への参照が含まれていることを確認します。これは、フォームにアクセスするためにではなく、色を設定するために必要です。
2. `New Form1` を変数に代入します。

   VB

   ```
   Public Sub GetSecondInstance()
       Dim newForm1 As New Form1
       newForm1.BackColor = System.Drawing.Color.YellowGreen
       newForm1.Show()
   End Sub
   ```

   同じフォームのコピーを複数表示する場合は、追加のコピーを作成する必要があります。上の例では、`Form1` の 2 つ目のコピーを作成し、これを別の色で塗りつぶしています。その後は、`Form1` を使用すると元のコピーにアクセスでき、`newForm1` を使用すると 2 つ目のコピーにアクセスできます。

参照

関連項目

New (Visual Basic)
System.Drawing
Form

方法：New キーワードを使用する

クラスのインスタンスを作成するには、**New** キーワードを使用します。整数型 (**Integer**) や倍精度浮動小数点型 (**Double**) などの値型とは異なり、オブジェクトは参照型であり、使用する前に明示的に作成しておく必要があります。2 つのコード例を次に示します。

VB

```
Dim Button1 As System.Windows.Forms.Button
Dim Button2 As New System.Windows.Forms.Button()
```

最初のステートメントは、ボタン オブジェクトへの参照を格納できるオブジェクト変数を宣言しています。ただし、Button 型オブジェクトが割り当てられるまで、`Button1` 変数の値は **Nothing** です。2 番目のステートメントも、ボタン オブジェクトを格納できる変数を定義しています。ただし、**New** キーワードでボタン オブジェクトを作成し、そのボタン オブジェクトを `Button2` 変数に割り当てます。

フォームとコントロールは実際にはクラスです。このため、必要に応じて、**New** キーワードを使用してフォームとコントロールの新しいインスタンスを作成できます。

New キーワードでクラスの新しいインスタンスを作成するには

1. 新しい Windows アプリケーション プロジェクトを開き、コマンド ボタンなどのコントロールを `Form1` という名前のフォームに配置します。
2. コマンド ボタンの Click イベント プロシージャに次のコードを追加します。

   VB

   ```
   Dim f As New Form1
   f.Show()
   ```

3. アプリケーションを実行し、コマンド ボタンを数回クリックします。
4. 前面のフォームを脇へ移動します。フォームはインターフェイス表示の設定されたクラスなので、フォームの複製を見ることができます。デザイン時には、それぞれの複製に同じコントロールが、元のフォームと同じ位置に配置されています。

New キーワードを使用して、クラス内でオブジェクトを作成できます。次に例を示します。

New キーワードによってクラスのインスタンスがどのように作成されるかを確認するには

1. 新しいプロジェクトを開き、`Form1` という名前のフォームにコマンド ボタンを配置します。
2. [プロジェクト] メニューの [クラスの追加] を選択し、プロジェクトにクラスを追加します。
3. 新しいクラスに `ShowMe.vb` という名前を付けます。
4. `ShowMe` に次のプロシージャを追加します。

   VB

   ```
   Public Class ShowMe
       Sub ShowFrm()
           Dim frmNew As Form1
           frmNew = New Form1
           frmNew.Show()
           frmNew.WindowState = FormWindowState.Minimized
       End Sub
   End Class
   ```

5. `Button1` の **Click** イベントを処理するために、フォームに次のコードを追加します。

   VB

```
Protected Sub Button1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles Button1.Click
    Dim clsNew As New ShowMe
    clsNew.ShowFrm()
End Sub
```

6. このコード例を使用するには、アプリケーションを実行し、コマンド ボタンを数回クリックします。ShowMe クラスの新しいインスタンスを作成するたびに、最小化されたフォームがタスクバーに表示されます。

参照

その他の技術情報

オブジェクトの作成と使用

リソースの管理

すべてのオブジェクトは、メモリ、ファイル ハンドル、データベース接続などのシステム リソースを使用します。リソースは、共通言語ランタイム (CLR: Common Language Runtime) によって自動的に管理されるため、通常は不要なオブジェクトの解放について気にする必要はありません。しかし、リソースがどのように管理されているのかを理解しておくと、より効率的なアプリケーションをデザインできるようになります。

ガベージ コレクション

CLR は、ガベージ コレクションというシステムを使って、割り当てられたリソースを管理します。システムのガベージ コレクタは、既にアプリケーションの実行中のコードから利用できなくなったオブジェクトのリソースを解放します。ガベージ コレクションのアルゴリズムは非決定的なので、CLR がオブジェクトのリソースをいつ解放するかは不明確です。以降では、Visual Basic におけるリソースの管理方法の変更点をいくつか説明します。

Nothing へのオブジェクトの割り当て

Nothing は、Visual Basic でオブジェクト変数がオブジェクトへの参照を含んでいないことを示すために使用されるキーワードです。以前のバージョンの Visual Basic では、使わないオブジェクトを **Nothing** に割り当てることによって、オブジェクト変数をオブジェクトから切り離し、リソースを解放していました。現在の Visual Basic でも、使わないオブジェクトを **Nothing** に割り当てることはできますが、リソースの管理方法が変更されているため、オブジェクトがすぐに解放されるとは限りません。一般に、オブジェクトを **Nothing** に割り当てる必要があるのは、共有メンバやグローバル変数などの寿命の長いオブジェクトの場合だけです。

破棄

一部のオブジェクトは、Dispose というメソッドをサポートしています。このメソッドを使用すると、システム リソースをより迅速に解放できます。**Dispose** メソッドをサポートするクラスは、IDisposable インターフェイスを実装する必要があります。**Dispose** メソッドは、オブジェクトのリソースを解放する時に明示的に呼び出します。たとえば、次のようにします。

ThisObject.**Dispose**()

Finalize

この他に、Finalize メソッドをサポートしているクラスもあります。このメソッドは、オブジェクトが解放されるときに自動的に実行されるので、これを使ってその他のクリーンアップ タスクを実行できます。**Finalize** メソッドは、以前のバージョンの Visual Basic で使用されていた **Class_Terminate()** メソッドに似ています。オブジェクトがアクセス不能になると、CLR は最終的に、そのオブジェクトの **Finalize** メソッドを呼び出します。ガベージ コレクションのアルゴリズムは非決定的なので、**Finalize** メソッドは直ちに呼び出されることもあれば、数時間後に呼び出されることもあります。

参照

関連項目

Nothing (Visual Basic)
Dispose
IDisposable

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄
コンポーネントの初期化と終了

方法：オブジェクトをプロシージャに渡す

Visual Basic では、オブジェクトを引数として、他の型の引数と同様にプロシージャに渡すことができます。手順を次に示します。

フォームの新しいインスタンスをプロシージャに渡すには

1. プロジェクトを開き、`Form1` という名前の新しいフォームを作成し、`Button1` という名前のコマンド ボタンをフォームに追加します。
2. フォームに次のコードをコピーします。

   VB

```
Private Sub Button1_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    Dim newForm As New Form1
    newForm.Show()
    CenterForm(newForm)
End Sub

Sub CenterForm(ByVal TheForm As Form)
    ' Centers the form on the screen.
    Dim RecForm As Rectangle = Screen.GetBounds(TheForm)
    TheForm.Left = CInt((RecForm.Width - TheForm.Width) / 2)
    TheForm.Top = CInt((RecForm.Height - TheForm.Height) / 2)
End Sub
```

   参照によってオブジェクトを引数として渡し、プロシージャの中で引数を新しいオブジェクトに設定することもできます。

他のフォームのプロシージャにオブジェクト参照を渡すには

1. プロジェクトを開き、`Form1` という名前のフォームを作成します。
2. `Form2` という名前で 2 つ目のフォームを追加します。
3. 各フォームにピクチャ ボックス コントロールを配置します。
4. Form1 のピクチャ ボックスに `PictureBox1` という名前を付けます。
5. Form2 のピクチャ ボックスに `PictureBox2` という名前を付けます。
6. [プロパティ] ウィンドウの [ImageUrl] プロパティをクリックして、`PictureBox2` にイメージを割り当てます。サイズの小さいイメージを割り当てます。Windows ディレクトリの中の .bmp ファイルや .jpg ファイルを使用することもできます。
7. `Form2` に次のコードを追加します。

   VB

```
Public Sub GetPicture(ByVal x As PictureBox)
    Dim objX As PictureBox
    ' Assign the passed-in picture box to an object variable.
    objX = x
    ' Assign the value of the Picture property to the Form1 picture box.
    objX.Image = PictureBox2.Image
End Sub
```

8. `Form1` の `Form1_Click` イベントに次のコードを追加します。

   VB

```
Protected Sub Form1_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs)
    Dim newForm2 As New Form2
    newForm2.GetPicture(PictureBox1)
End Sub
```

9. アプリケーションを実行し、Form1 をクリックします。Form1 のピクチャ ボックスに、Form2 のイメージが表示されます。

   Form1_Click イベント プロシージャは、Form2 の GetPicture プロシージャを呼び出し、空のピクチャ ボックスを渡します。Form2 の GetPicture プロシージャは、Form2 のピクチャ ボックスの Image プロパティを Form1 の空のピクチャ ボックスに割り当てます。Form1 には Form2 のイメージが表示されます。

参照

概念

リソースの管理

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

コンポーネントによるプログラミング

チュートリアル：Visual Basic でのオブジェクトの永続化

デザイン時にオブジェクトのプロパティを既定値に設定できますが、実行時に入力した値はオブジェクトが破棄されるとすべて失われます。Visual Basic では、シリアル化を使用してインスタンス間でオブジェクトのデータを永続化することで、次にそのオブジェクトのインスタンスを生成するときに値を格納および取得できます。

名前や数字などの単純なデータを格納する場合、**My.Settings** オブジェクトを使用します。詳細については、「My.Settings オブジェクト」を参照してください。

このチュートリアルでは、簡単な `Loan` オブジェクトを作成し、そのデータをファイルに永続化して、オブジェクトの再作成時にファイルからデータを取得します。次に、コードを変更して SOAP 形式でオブジェクトを永続化します。

セキュリティに関するメモ：

次のコード例では、ファイルが存在しない場合は新規にファイルを作成します。アプリケーションがファイルを作成する必要がある場合は、フォルダの **Create** のアクセス許可が必要になります。アクセス許可は、アクセス制御リストを使用して設定します。ファイルが既に存在する場合、アプリケーションに必要なのは **Write** のアクセス許可だけです。可能な場合は、配置時にファイルを作成し、フォルダに対する作成の権限を付与するのではなく 1 つのファイルに対する **Read** の権限だけを付与する方が安全です。また、ルート フォルダや Program Files フォルダにデータを書き込むよりも、ユーザー フォルダに書き込む方が安全です。

セキュリティに関するメモ：

この例では、バイナリ形式または SOAP 形式のファイルにデータを格納します。これらの形式は、パスワードやクレジットカード情報などの重要情報には使用しないでください。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Loan オブジェクトの作成

まず、`Loan` クラスとそのクラスを使用するテスト アプリケーションを作成します。

Loan クラスを作成するには

1. 新しいクラス ライブラリ プロジェクトを作成して、LoanClass という名前を付けます。詳細については、「方法：ソリューションおよびプロジェクトを作成する」を参照してください。
2. コード エディタで、クラスの名前を Class1 から Loan に変更します。
3. クラスに次のパブリック メンバを追加します。

   VB

   ```
   Public LoanAmount As Double = 10000.0
   Public InterestRate As Double = 7.5
   Public Term As Integer = 36
   Public Customer As String
   ```

次に、`Loan` クラスを使用する簡単なアプリケーションを作成する必要があります。

テスト アプリケーションを作成するには

1. [ファイル] メニューで、[追加] をポイントして [新しいプロジェクト] をクリックし、ソリューションに Windows Application プロジェクトを追加します。
2. [新しいプロジェクトの追加] ダイアログ ボックスで、プロジェクト名として「**LoanApp**」と入力し、[OK] をクリックしてダイアログ ボックスを閉じます。
3. ソリューション エクスプローラでプロジェクトを選択します。

4. [プロジェクト] メニューの [スタートアップ プロジェクトに設定] をクリックします。
5. [プロジェクト] メニューの [参照の追加] をクリックします。
6. [参照の追加] ダイアログ ボックスで、[プロジェクト] タブをクリックし、[LoanClass] プロジェクトをクリックします。
7. [OK] をクリックし、ダイアログ ボックスを閉じます。
8. デザイナで、フォームに 4 つの TextBox コントロールを追加します。
9. コード エディタで、次のコードを追加します。

VB

```
Private TestLoan As New LoanClass.Loan
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
    TextBox1.Text = TestLoan.LoanAmount.ToString
    TextBox2.Text = TestLoan.InterestRate.ToString
    TextBox3.Text = TestLoan.Term.ToString
    TextBox4.Text = TestLoan.Customer
End Sub
```

この時点で、アプリケーションをビルドして実行できます。`Loan` クラスの既定値が、テキスト ボックスに表示されます。利率の値を 7.5 から 7.1 に変更し、アプリケーションをいったん閉じてから再び実行してください。値が既定値の 7.5 に戻ります。

実際には利率は定期的に変わりますが、アプリケーションを実行するたびに変わるとは限りません。アプリケーションを実行するたびにユーザーが利率を更新するのではなく、アプリケーションのインスタンス間で最新の利率を保持できるようにすると便利です。次の手順では、Loan クラスにシリアル化を追加して利率を保持できるようにします。

シリアル化を使用したオブジェクトの永続化

Loan クラスの値を永続化するには、まずクラスを **Serializable** 属性でマークする必要があります。

シリアルできるクラスとしてクラスをマークするには

- Loan クラスのクラス宣言を次のように変更します。

VB

```
<Serializable()> Public Class Loan
```

Serializable 属性は、クラス内のすべての要素がファイルに永続化できることをコンパイラに示します。この場合、永続化するのは `InterestRate` メンバのみで、`Customer`、`LoanAmount`、`Period` メンバは永続化の必要がない可能性があります。永続化しないクラス メンバは、**NonSerialized** 属性でマークできます。この例では、簡略化のために `Customer` 以外のすべてのメンバが永続化されます。

シリアル化の対象からメンバを除外するには

- `Customer` メンバの宣言を次のように変更します。

VB

```
<NonSerialized()> Public Customer As String
```

次に、LoanApp アプリケーションにシリアル化コードを追加します。クラスをシリアル化してファイルに書き込むには、System.IO および System.Xml.Serialization 名前空間を使用します。**Imports** ステートメントを使用すると、完全修飾名の入力が不要です。

名前空間に参照を追加するには

- `Form1` クラスの先頭に、次の **Imports** ステートメントを追加します。

VB

```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary
```

この場合は、バイナリ フォーマッタを使用してバイナリ形式でオブジェクトを保存します。後でこのコードを変更して、SOAP 形式でオブジェクトを保存します。

次の手順では、オブジェクトの作成時にファイルからオブジェクトを逆シリアル化するコードを追加します。

オブジェクトを逆シリアル化するには

1. シリアル化されたデータのファイル名として、クラスに定数を追加します。

   VB

   ```
   Const FileName As String = "SavedLoan.bin"
   ```

2. `Form1_Load` イベント プロシージャのコードを次のように変更します。

   VB

   ```
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
   System.EventArgs) Handles MyBase.Load
       If File.Exists(FileName) Then
           Dim TestFileStream As Stream = File.OpenRead(FileName)
           Dim deserializer As New BinaryFormatter
           TestLoan = CType(deserializer.Deserialize(TestFileStream), LoanClass.Loan)
           TestFileStream.Close()
       End If
       TextBox1.Text = TestLoan.LoanAmount.ToString
       TextBox2.Text = TestLoan.InterestRate.ToString
       TextBox3.Text = TestLoan.Term.ToString
       TextBox4.Text = TestLoan.Customer
   End Sub
   ```

   まずファイルが存在することを確認する必要があります。ファイルが存在する場合は、バイナリ ファイルを読み取る Stream クラスと、ファイルを変換する BinaryFormatter クラスを作成します。ストリームから Loan オブジェクト型への変換には、**CType** メソッドを使用します。

次に、テキスト ボックスに入力されたデータを `Loan` クラスに保存して、クラスをファイルにシリアル化するコードを追加します。

データを保存してクラスをシリアル化するには

- `Form1_Closing` イベント プロシージャに次のコードを追加します。

  VB

  ```
  Private Sub Form1_Closing(ByVal sender As System.Object, ByVal e As _
  System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
      TestLoan.LoanAmount = CType(TextBox1.Text, Double)
      TestLoan.InterestRate = CType(TextBox2.Text, Double)
      TestLoan.Term = CType(TextBox3.Text, Integer)
      TestLoan.Customer = TextBox4.Text

      Dim TestFileStream As Stream = File.Create(FileName)
      Dim serializer As New BinaryFormatter
      serializer.Serialize(TestFileStream, TestLoan)
      TestFileStream.Close()
  End Sub
  ```

この時点で、アプリケーションを再度ビルドして実行できます。最初に既定値がテキスト ボックスに表示されます。値を変更して、4 番目のテキスト ボックスに名前を入力します。いったんアプリケーションを閉じて、再び実行します。新しい値がテキスト ボックスに表示されますが、**NonSerialized** としてマークした顧客名の値は表示されません。

SOAP 形式を使用したオブジェクトの永続化

これまで紹介した例では、バイナリ形式を使用してテキスト ファイルにオブジェクトを永続化する方法を説明しました。バイナリ形式は、ほとんどの Windows アプリケーションで問題なく使用できます。Web アプリケーションまたは XML Web サービスの場合は、SOAP 形式を使用して XML ファイルにオブジェクトを永続化した方がオブジェクトの共有が簡単になります。

オブジェクトを SOAP 形式で永続化するには、まず SoapFormatter クラスを参照する必要があります。**SoapFormatter** クラスは、自らの名前空間 System.Runtime.Serialization.Formatters.Soap にあります。

SOAP 形式を使用してオブジェクトを永続化するには

1. ソリューション エクスプローラでプロジェクトを選択します。
2. [プロジェクト] メニューの [参照の追加] をクリックします。
3. [参照の追加] ダイアログ ボックスで、[.NET] タブをクリックし、**System.Runtime.Serialization.Formatters.Soap** コンポーネントをクリックします。
4. [OK] をクリックし、ダイアログ ボックスを閉じます。
5. コード エディタで、`Form1` モジュールの先頭に、次の **Imports** ステートメントを追加します。

   VB

   ```
   Imports System.Runtime.Serialization.Formatters.Soap
   ```

6. ファイル名を `SavedLoan.bin` から `SavedLoan.xml` に変更します。
7. `Form1_Load` イベント プロシージャで、次の **Dim** ステートメントを、`Dim deserializer As New BinaryFormatter` から次のように変更します。

   VB

   ```
   Dim deserializer As New SoapFormatter
   ```

8. `Form1_Closing` イベント プロシージャで、次の **Dim** ステートメントを、`Dim serializer As New BinaryFormatter` から次のように変更します。

   VB

   ```
   Dim serializer As New SoapFormatter
   ```

この時点で、アプリケーションをビルドしてテストできます。アプリケーションを初めて実行したときに、SavedLoan.xml ファイルが作成されます。このファイルを表示するには、ソリューション エクスプローラの [すべてのファイルを表示] をクリックします。ファイルは、Windows アプリケーション プロジェクトの Bin ノードにあります。

メモ：

既に [すべてのファイルを表示] モードになっている場合、ファイルを表示するには、[表示] メニューの [最新の情報に更新] をクリックして最新の内容に更新する必要があります。

`LoanClass` の 3 つのメンバは、XML 形式で表示されます。XML ファイルで InterestRate の値を変更して保存してから、アプリケーションを再度実行してください。新しい利率が、2 番目のテキスト ボックスに表示されます。

参照
概念
PropertyBag の対応関係 (Visual Basic 6.0 ユーザー向け)

方法 : Visual Basic でクラスを継承する

この例では、いずれも `Shape` クラスを継承する、`Circle` クラスと `Rectangle` クラスを定義します。また、`Rectangle` クラスを継承する `Square` クラスを定義します。

使用例

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

VB

```
Public Class Shape
    ' Definitions of properties, methods, fields, and events.
End Class
Public Class Circle : Inherits Shape
    ' Specialized properties, methods, fields, events for Circle.
End Class
Public Class Rectangle : Inherits Shape
    ' Specialized properties, methods, fields, events for Rectangle.
End Class
Public Class Square : Inherits Rectangle
    ' Specialized properties, methods, fields, events for Square.
End Class
```

コードのコンパイル方法

必要な条件は次のとおりです。

- System 名前空間への参照

> **メモ :**
> 継承元のクラスが **NotInheritable** として定義されていないことを確認してください。

参照

処理手順

方法 : 形状のアウトラインを描画する

関連項目

NotInheritable

その他の技術情報

Visual Basic の継承

オブジェクトの作成と使用

方法：Visual Basic で XML ファイルからオブジェクト データを読み込む

XmlSerializer クラスを使用して XML ファイルに書き込んだオブジェクト データを読み込む例を次に示します。

使用例

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [XML] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

VB

```
Public Class Book
    Public Title As String
    Public Sub New()
    End Sub
End Class

Public Sub ReadXML()
    Dim reader As New System.Xml.Serialization.XmlSerializer(GetType(Book))
    Dim file As New System.IO.StreamReader("c:\IntroToVB.xml")
    Dim introToVB As Book
    introToVB = CType(reader.Deserialize(file), Book)
End Sub
```

コードのコンパイル方法

ファイル名 "c:\IntroToVB.xml" を、シリアル化されたデータを含むファイルの名前に置き換えます。データのシリアル化の詳細については、「方法 : Visual Basic で XML ファイルにオブジェクト データを書き込む」を参照してください。

クラスには、パラメータのないパブリック コンストラクタが必要です。

パブリックなプロパティとフィールドだけが逆シリアル化されます。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- シリアル化されるクラスにパブリックなパラメータなしのコンストラクタがない場合
- ファイルのデータが、逆シリアル化されるクラスのデータを表していない場合
- ファイルが存在しない場合 (IOException)

セキュリティ

入力を常に検証し、信頼関係のないソースからのデータは絶対に逆シリアル化しないでください。再作成されたオブジェクトは、このオブジェクトを逆シリアル化したコードと同じアクセス権で、ローカル コンピュータで実行されます。アプリケーションでデータを使用する前に、入力をすべて検証してください。

参照

処理手順

方法 : Visual Basic で XML ファイルにオブジェクト データを書き込む

方法 : StreamReader を使用してファイルからテキストを読み取る (Visual Basic)

関連項目

StreamWriter

方法：Visual Basic で XML ファイルにオブジェクト データを書き込む

XmlSerializer クラスを使用して、クラスから XML ファイルにオブジェクトを書き込む例を次に示します。

使用例

このコード例は、`Book` という名前のクラスを定義し、そのクラスのインスタンスを作成してから、XML シリアル化を使用してインスタンスを XML ファイルに書き込みます。

これに似たコードを、IntelliSense コード スニペットとして利用できます。コード スニペット ピッカーでは、これは [XML] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

VB

```
Public Class Book
    Public Title As String
    Public Sub New()
    End Sub
End Class

Public Sub WriteXML()
    Dim introToVB As New Book
    introToVB.Title = "Intro to Visual Basic"
    Dim writer As New System.Xml.Serialization.XmlSerializer(GetType(Book))
    Dim file As New System.IO.StreamWriter("c:\IntroToVB.xml")
    writer.Serialize(file, introToVB)
    file.Close()
End Sub
```

コードのコンパイル方法

クラスには、パラメータのないパブリック コンストラクタが必要です。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- シリアル化されるクラスにパブリックなパラメータなしのコンストラクタがない場合
- ファイルが存在するものの、読み取り専用の場合 (IOException)
- パスが長すぎる場合 (PathTooLongException)
- ディスクの空き領域がない場合 (**IOException**)

セキュリティ

次のコード例では、ファイルが存在しない場合は新規にファイルを作成します。アプリケーションでファイルを作成する必要がある場合、そのアプリケーションにはフォルダに対する **Create** アクセスが必要です。ファイルが既に存在する場合、アプリケーションに必要なのは、より低い権限である **Write** アクセスだけです。フォルダに対して **Create** アクセスを許可するのではなく、可能な限りアプリケーションの配置時にファイルを作成しておき、1 つのファイルに対してのみ **Read** アクセスを許可する方が安全です。

参照

処理手順

方法 : Visual Basic で XML ファイルからオブジェクト データを読み込む

関連項目

StreamWriter

その他の技術情報

Visual Basic におけるファイル アクセス

Visual Basic におけるコレクション

一般に、コレクションという用語は、関連するオブジェクトのグループ化および管理に使用するオブジェクトを指します。たとえば、どの Form にもコントロールのコレクションがあります。このコレクションにアクセスするには、フォームの Controls プロパティを使用します。このコレクションは、フォーム上のすべてのコントロールを表すオブジェクトです。コレクション内のコントロールをインデックスを使って取得したり、For Each...Next ステートメント (Visual Basic) を使ってコレクションの要素をループしたりできます。

ただし、コレクションにはいくつかの種類があり、さまざまな点で異なります。

コレクションの種類

Visual Basic には、**Collection** クラスも用意されています。このクラスは、独自のコレクションを定義し、作成するために使用できます。**Collection** クラスにも、フォームの **Controls** コレクションと同じように、**For Each...Next** を使って要素をループしたりインデックスで要素を取得したりするための組み込み機能が用意されています。詳細については、「Collection オブジェクト (Visual Basic)」を参照してください。

ただし、これらの 2 つの種類のコレクションは相互運用できません。たとえば、次のコードはコンパイラ エラーになります。

```
Dim localControls As Collection

' The following line generates a COMPILER ERROR.

localControls = Me.Controls()
```

Controls コレクションは .NET Framework コレクションですが、変数 `localControls` は Visual Basic の**Collection**なので、これらのコレクションに互換性はありません。これらの 2 種類のコレクションは、異なるクラスから実装されます。両者のメソッドは似ていますが同一ではありません。インデックスの方式も異なります。

インデックス番号が 0 から始まるコレクションと 1 から始まるコレクション

コレクションには、項目のインデックス番号が 0 から始まるコレクションと 1 から始まるコレクションがあります。0 から始まるコレクションではコレクション内の最初の項目を指すインデックスは 0 で、1 から始まるコレクションでは 1 です。前記の .NET Framework の **Controls** コレクションは、0 から始まるコレクションの例です。Visual Basic の **Collection** オブジェクトは、1 から始まるコレクションの例です。

1 から始まるコレクションは、インデックスの範囲が 1 ～ Count プロパティ (Collection オブジェクト) (コレクション内の項目数を返すプロパティ) であるため、Visual Basic ユーザーはより直感的に使用できます。一方、インデックス番号が 0 から始まるコレクションの範囲は、0 ～ (**Count** プロパティの値から 1 を引いた値) です。この方式は、インデックス値が基本値からのオフセットである場合や、0 から始まる列挙のメンバに相当する場合に適切です。

.NET Framework コレクションのインデックスが 0 から始まるのは、標準化のためです。Visual Basic の **Collection** クラスのインデックスが 1 から始まるのは、旧バージョンとの互換性維持のためです。

インデックスおよびキー値

Visual Basic の **Collection** クラスのインスタンスでは、数値インデックスまたは **String** キーを使って項目にアクセスできます。キーを指定してもしなくても、項目を Visual Basic の **Collection** オブジェクトに追加できます。キーを使わずに項目を追加した場合は、その項目にアクセスするときに数値インデックスを使う必要があります。

一方、一部のコレクション (System.Collections.ArrayList など) では、数値インデックスだけを使用できます。キーを格納した **String** 配列などに基づいて独自の対応付けを作成しない限り、このようなコレクションの要素にキーを関連付けることはできません。

項目の追加と削除

コレクションには項目を追加できるものとできないものがあり、項目を追加できるコレクションの間でもその方法に違いがあります。Visual Basic の **Collection** オブジェクトは、汎用プログラミング ツールなので、他のコレクションに比べて柔軟性に富んでいます。項目をコレクションに追加する Add メソッド (Collection オブジェクト) や、項目をコレクションから削除する Remove メソッド (Collection オブジェクト) があります。

一方、特定の目的で使用されるコレクションでは、コードを使って要素を追加したり削除したりすることはできません。たとえば、System.Windows.Forms.CheckedListBox.CheckedItems プロパティは、項目への参照をインデックスで返しますが、コードでコレクションの項目を追加したり削除したりすることはできません。そのような操作は、ユーザー インターフェイスの該当するチェック ボックスをオンまたはオフにすることにより、ユーザーだけが実行できます。したがって、このコレクションには **Add** メソッドも **Remove** メソッドもありません。

参照

処理手順

方法 : オブジェクトのコレクションを作成する
方法 : オブジェクトの配列を作成する
方法 : コレクションの項目を追加、削除、および取得する
方法 : クラス内にコレクションを定義する

方法 : Visual Basic でコレクションを反復処理する
コレクションのトラブルシューティング
概念
オブジェクトのグループの管理
Visual Basic のコレクション クラス
コレクションによるオブジェクトの管理

オブジェクトのグループの管理

多くのアプリケーションでは、関連するオブジェクトのグループの作成および管理が必要になります。オブジェクトをグループ化するには、オブジェクトの配列を作成する方法と、オブジェクトのコレクションを作成する方法があります。

オブジェクトの配列

配列は、比較的柔軟性に欠ける構造体です。配列のサイズを実行時に変更するには、ReDim ステートメント (Visual Basic) を使って宣言し直す必要があります。また、配列の要素はすべて同じデータ型である必要があります。一方、配列には、すべての要素を連続処理したり、空の要素を保持したりできるという利点があります。このような理由から、配列は、数が固定されている厳密に型指定されたオブジェクトの作成および処理に最も適しています。

詳細については、「方法 : オブジェクトの配列を作成する」を参照してください。

オブジェクトのコレクション

コレクションでは、オブジェクトのグループをより柔軟に処理できます。コレクションはクラスなので、コレクションに要素を追加するには、事前に新しいコレクションを宣言する必要があります。配列の場合とは違って、コレクションで扱うオブジェクトのグループは、アプリケーションの変更に伴う必要に応じて動的に拡大および縮小できます。コレクションに追加する任意のオブジェクトにはキーを割り当てることができ、そのキーに基づいてオブジェクトを取得および操作できます。

一般化されたコレクションは、Visual Basic のコレクション クラス を使用して作成できます。コレクションにディクショナリやリンク リストなどの特定の機能が必要な場合は、.NET Framework の System.Collections 名前空間のいずれかのクラスから作成できます。System.Collections.Specialized 名前空間には、より特殊なコレクション クラスが用意されています。

要素が 1 つのデータ型だけに限定されているコレクションの場合は、System.Collections.Generic 名前空間のクラスのいずれかを使用できます。ジェネリック コレクションでは、タイプ セーフが強制されるため、他のデータ型を追加することはできません。ジェネリック コレクションから要素を取得する場合は、データ型を判断したり、変換したりする必要はありません。

詳細については、「方法 : オブジェクトのコレクションを作成する」を参照してください。

参照

処理手順

方法 : コレクションの項目を追加、削除、および取得する

関連項目

System.Collections
System.Collections.Generic
System.Collections.Specialized

概念

Visual Basic におけるコレクション
Visual Basic のコレクション クラス

その他の技術情報

Visual Basic における配列
オブジェクトの作成と使用

コレクションによるオブジェクトの管理

コレクションは、多様なオブジェクトを管理するのに最適な手段です。コレクションでは、オブジェクトの追加と削除、インデックスやキーに基づくオブジェクトの取得、および For Each...Next ステートメント (Visual Basic) を使った項目の反復処理ができます。

タイプ セーフでないコレクション

しかし、コレクションは、その柔軟性ゆえにかえってクラスの信頼性を損ねてしまう可能性があります。たとえば、Visual Basic で提供されるコレクションには、すべての要素が **Object** 型として格納されるため、任意のデータ型の項目を追加できます。不適切なデータ型が追加されることに対する保護機能はなく、要素にアクセスするときには、データ型を **Object** 型から目的のデータ型に変換する必要があります。

専用コレクション

.NET Framework には、Visual Basic のコレクションに対するいくつかの代替方法が用意されています。System.Collections 名前空間には、キューや並べ替えリストなどの特別な機能を持ったコレクション クラスが含まれています。System.Collections.Specialized 名前空間には、HybridDictionary など、特別な性質のコレクション クラスが含まれています。

タイプ セーフなコレクション

System.Collections.Generic 名前空間のジェネリック コレクションを使用すると、**Object** 型の要素の短所を回避できます。これらのコレクションは、タイプ セーフであり、コレクションの要素を 1 つの特定のデータ型だけに制限できます。

コレクションの使用方法

コレクションを使ってオブジェクト管理を実装するには、一般的な方法が 3 つあります。`widget` オブジェクトを編成し、クライアント コンポーネントに公開する、`widgetRepository` クラスが定義してあるアプリケーションを考えます。コレクションを使用して `widgetRepository` を実装する場合、次のいずれかの方法を使用できます。

- **コレクション クラスを使用する。** `widgetRepository` クラスで、Visual Basic のコレクション クラス のインスタンスか、**System.Collections** 名前空間、**System.Collections.Generic** 名前空間、または **System.Collections.Specialized** 名前空間のいずれかのクラスのインスタンスとして `widgetsColl` 変数を宣言します。変数をパブリックにし、New (Visual Basic) キーワードを使用してコレクションのインスタンスを作成します。詳細については、「方法 : クラス内にコレクションを定義する」を参照してください。
- **コレクション基本クラスを継承する。** CollectionBase クラスを継承して独自の `widgetsColl` クラスを実装します。`widgetRepository` クラスで、`widgetsColl` クラスのインスタンスを定義し、そのインスタンスを返すプロパティを定義します。詳細については、「方法 : クラス内にコレクションを定義する」を参照してください。
- **独自のコレクションを記述する。** 適切なクラスおよびプロシージャを記述して、`widgetRepository` クラスにコレクション機能を実装します。クラスにコレクションの機能が必要だが既存のコレクション クラスを継承できない場合には、この方法が最も便利です。たとえば、まれにしかありませんが、コレクション クラス以外のクラスを継承する必要がある場合が該当します。クラスは複数のクラスを継承できないため、この場合はコレクション メンバを定義および実装する必要があります。

参照

関連項目

System.Collections
System.Collections.Generic
System.Collections.Specialized

概念

オブジェクトのグループの管理
Visual Basic のコレクション クラス

Visual Basic のコレクション クラス

コレクションは、関連項目の集約をグループ化する方法の 1 つです。コレクションにはさまざまな種類があります。定義済みのコレクションは、Visual Basic アプリケーションでさまざまな目的に使用されます。たとえば、フォームの Controls プロパティによって返される Form の ControlCollection などです。また、独自のコレクションを作成して、オブジェクトを整理したり、操作できます。

たとえば、アプリケーションが動的に作成したり破棄したりするオブジェクトを追跡するにはコレクションが適しています。次のコードは、Visual Basic コレクション オブジェクトの Add メソッドを使用して、ユーザーが作成した `widget` オブジェクトの一覧を維持する方法を示しています。

```
' Declare and create the Collection object.
Public widgetColl As New Microsoft.VisualBasic.Collection()
' Create a new widget and add it to the widgetColl collection.
Private Sub makeAWidget()
    Dim tempWidget As New widget()
    widgetColl.Add(tempWidget)
End Sub
```

前の例で、`widgetColl` コレクションは、`makeAWidget` プロシージャを使用して作成されたすべての `widget` オブジェクトを整理し、公開しています。各 `widget` へのオブジェクト参照は、コレクションのインデックスを通じて取得できます。コレクションのサイズは、新しい `widget` オブジェクトが追加されるたびに自動的に調整されます。For Each...Next ステートメント (Visual Basic) ステートメントを使用すると、コレクションを反復処理できます。`widget` オブジェクトにキーを割り当てて、そのオブジェクトを取得できるようにするには、**Add** メソッドの 2 番目のパラメータとして文字列を指定します。

Visual Basic **Collection** オブジェクトは、すべての要素を **Object** 型として保存するため、任意のデータ型の項目を追加できます。不適切なデータ型が追加されないようにする保護機能はありません。このような制限を回避するには、System.Collections.Generic 名前空間のジェネリック コレクションを使用できます。詳細については、「方法 : オブジェクトのコレクションを作成する」を参照してください。

コレクション オブジェクトの作成と破棄

変数 `widgetColl` の宣言の New (Visual Basic) キーワードでは、宣言ステートメントに制御が渡されるときに **Collection** オブジェクトが作成されます。**Collection** は値型というよりはクラスであるため、インスタンスを作成し、そのインスタンス (オブジェクト) への参照を変数に維持する必要があります。このインスタンスは、Visual Basic コレクション オブジェクトです。

他のオブジェクトと同様に、**Collection** オブジェクトは、このオブジェクトへの参照を含む最後の変数が Nothing (Visual Basic) に設定されているか、またはオブジェクトに対する参照がスコープ外に出ると、ガベージ コレクション (GC) のマークが付けられます。ガベージ コレクションによって参照が再要求されると、これを含むすべてのオブジェクト参照が解除されます。このような理由から、前の例の変数 `widgetColl` は、プログラムの実行期間全体を通じて存続するように親クラスで宣言されています。

コレクションは、制御しているオブジェクトへの参照を保持しますが、オブジェクト自身は含みません。したがって、**Collection** オブジェクトを破棄しても、制御しているオブジェクトを破棄するわけではありません。コレクションの要素であった個々のオブジェクトは、ガベージ コレクションとして個々にマークされるまで存在し続けます。

要素の操作

コレクションの要素の追加、削除、および取得という基本的なサービスは、キーやインデックスに基づいています。キーとは、**String**の値です。キーは、名前、運転免許証番号、電話番号、または単に文字列に変換した整数でもかまいません。**Add** メソッドでは、「方法 : コレクションの項目を追加、削除、および取得する」に説明されているように、キーを要素に関連付けることができます。

Collection クラスのインデックスは、1 からコレクション内の項目数までの整数です。Count プロパティ (Collection オブジェクト) は、現在の項目数を返します。項目のインデックスの初期値は、**Add** を呼び出したときの *Before* または *After* パラメータを使用して制御できますが、他の項目が追加されたり削除されたりすることによってインデックスの値が変化する可能性があります。詳細については、「Add メソッド (Collection オブジェクト)」を参照してください。

要素のキーまたはインデックスのいずれかを Remove メソッド (Collection オブジェクト) に渡すことで、コレクションから単一の要素を削除できます。Clear メソッド (Collection オブジェクト) を使用して、コレクションを空にしてすべての要素を削除できます。

要素へのアクセス

キーの値を Contains メソッド (Collection オブジェクト) に渡して、コレクションにそのキーを持った要素が含まれているかどうかをテストできます。要素のキーまたはインデックスのいずれかを Item プロパティ (Collection オブジェクト) に渡して、要素を取得できます。

コレクション内の項目を反復処理するには、インデックスの値および **Item** プロパティを使用するか、または For Each...Next ステートメント (Visual Basic) を使用できます。次の例は、`employee` オブジェクトのコレクションに含まれるすべての従業員の給料を 10 パーセント上げるための 2 つの方法を示しています。なお、変数 `employeesColl` には **Collection** オブジェクトへの参照が含まれていることを前提とします。

```
Option Strict On
' The following alternative uses the Count and Item properties.
Dim emp As employee
For counter As Integer = 1 To employeesColl.Count
    emp = CType(employeesColl.Item(counter), employee)
    emp.payRate *= 1.1
Next counter
' The following alternative uses the For Each...Next statements.
For Each emp As employee In employeesColl
    emp.payRate *= 1.1
Next emp
```

ただし、1 つまたは複数の要素を、型が `employee` ではない `employeesColl` に追加すると、**For Each** ループは実行時に ArgumentException 例外をスローします。

要素のデータ型

Visual Basic コレクション オブジェクトでは、各項目のデータ型が **Object** として保存されます。したがって、**Collection** オブジェクトに追加できるデータ型の範囲は、**Object** 変数に保存できるデータ型の範囲と同じになります。これには、基本データ型、オブジェクト、配列、およびユーザー定義の構造体、クラスのインスタンスが含まれます。

Collection オブジェクトは、各要素を**Object**として保存するため、**Item** プロパティは**Object**の値を返します。コード内で項目を指定するには、通常 **Object** を項目の実行時のデータ型に変換する必要があります。変換の方法は、Option Strict ステートメントで指定されている型チェック スイッチの設定によって異なります。

オブジェクトの暗黙の変換

Option Strict が **Off** の場合、次の例で示されているように、**Collection**の要素を暗黙的に適切なデータ型に変更できます。

```
Option Strict Off
Dim sampleColl As New Microsoft.VisualBasic.Collection()
Dim sampleString As String = "This is a string"
Dim aString As String
sampleColl.Add(sampleString)
' The following statements convert the collection item to a string.
Try
    aString = sampleColl.Item(1)
Catch ex As Exception
    ' Insert code to run if the collection item cannot be converted to String.
End Try
```

オブジェクトの明示的変換

Option Strict が **On** の場合、**Object** を要素の実行時のデータ型に明示的に変換できます。この方法で **Item** から要素を取得するには、次の例で示されているように、CType 関数を使用して変換を実行できます。

```
Option Strict On
Dim sampleColl As New Microsoft.VisualBasic.Collection()
Dim sampleString As String = "This is a string"
Dim aString As String
sampleColl.Add(sampleString)
' The following statements convert the collection item to a string.
Try
    aString = CType(sampleColl.Item(1), String)
Catch ex As Exception
    ' Insert code to run if the collection item cannot be converted to String.
End Try
```

追加サービス

Collection オブジェクトのプロパティやメソッドは、コレクションの最も基本的なサービスだけを提供します。たとえば、**Add** メソッドでは、コレクションに追加する要素の型をチェックすることはできません。コレクションに含まれている要素の種類がすべて同じになるようにするには、このチェックが必要になります。**Add** メソッドで確実にこれを実行できる場合、厳密に型指定されたコレクションを持っていることになるので、その場合には戻り値を **Item** プロパティから実行時のデータ型に変換する必要はありません。これにより、パフォーマンスが向上します。

独自のコレクション クラスを作成することによって、追加プロパティ、メソッド、およびイベントなど、より信頼性の高い機能を提供できます。詳細に

ついては、「方法 : クラス内にコレクションを定義する」を参照してください。

参照

関連項目

Collection オブジェクト (Visual Basic)

概念

配列の代わりとしてのコレクション

Visual Basic におけるコレクション

方法：クラス内にコレクションを定義する

クラスにコレクションを追加すると、クラスで使用するオブジェクトのグループを管理できます。最も簡単な方法としては、**Collection** 型のパブリック変数をクラスに追加します。たとえば、`widget` オブジェクトを管理および公開する `widgetRepository` というクラスがあるとします。次の手順で説明するように、`widget` コレクションを参照する `widgetColl` 変数を作成できます。

簡単なコレクションを定義する

クラス内に簡単なコレクションを定義するには

- オブジェクトのコレクションとして動作するパブリック変数を作成します。

```
Public Class widgetRepository
    Public widgetColl As New Microsoft.VisualBasic.Collection()
    ' Insert code to implement additional functionality.
End Class
```

これで、`widgetRepository` クラスに、`widget` オブジェクトを追加するためのパブリック コレクションができました。次のコードに示すように、For Each...Next ステートメント (Visual Basic) を使用してコレクション要素を処理できます。

```
For Each aWidget As widget In widgetColl
    ' Insert code to process widgetColl elements
Next aWidget
```

上の例で定義した `widgetColl` コレクションでは、厳密な型指定が行われません。つまり、`widget` オブジェクトだけでなく、任意の型のオブジェクトをコレクションに追加できます。このことは、タイプ セーフの問題につながる可能性があります。たとえば、次のコードのように、**String** 型の変数をコレクションに追加するとします。

```
Dim notWidget As String = "This is not a widget object!"
widgetColl.Add(notWidget)
```

これを実行すると、コレクションの要素の 1 つが `widget` 型ではないため、前のプロシージャの **For Each** ループは、実行時に ArgumentException 例外をスローします。

タイプ セーフを保証するために、ジェネリック クラスを定義することを推奨します。使用例を含む詳細については、「方法 : タイプ セーフなコレクションの定義」を参照してください。

参照

処理手順

方法 : タイプ セーフなコレクションの定義

関連項目

Option Strict ステートメント

概念

Visual Basic のコレクション クラス
Visual Basic におけるコレクション
Visual Basic におけるジェネリック型

方法：タイプ セーフなコレクションの定義

Visual Basic に用意された **Collection** クラスを使って、次の例に示すようにコレクションを定義および作成できます。

VB

```
Public Class widgetRepository
    Public widgetColl As New Microsoft.VisualBasic.Collection()
    ' Insert code to implement additional functionality.
End Class
```

ただし、この `widgetColl` コレクションは厳密に型指定されません。`widget` オブジェクトに限らず、どのような型の要素でもこのコレクションに追加できます。要素を取得するときには、取得した要素を `widget` に変換する必要があります。このことは、タイプ セーフの問題につながる可能性があります。たとえば、次のコードを使用して **String** 型の変数をコレクションに追加するとします。

VB

```
Dim notWidget As String = "This is not a widget object!"
widgetColl.Add(notWidget)
```

これを実行すると、コレクションの要素が `widget` 型ではないため、後続のコードで要素を取得しようとすると実行時に ArgumentException 例外がスローされます。

タイプ セーフの問題からの保護

ジェネリック クラスを定義することにより、タイプ セーフを最大限にすることをお勧めします。厳密な型指定が強制されると同時に、操作する特定のデータ型に柔軟性を与えることができます。詳細については、「Visual Basic におけるジェネリック型」を参照してください。

クラス内にタイプ セーフなコレクションを定義するには

- System.Collections.Generic 名前空間内のジェネリック クラスの 1 つ、たとえば List をコレクション クラスの代わりに使用します。これで、`widget` メンバのみに制限されるコレクションを作成できます。次のコード例は、前記の例の宣言を変更してジェネリック コレクションを作成する方法を示します。

  VB

  ```
  Public widgetColl As New System.Collections.Generic.List(Of widget)
  ```

  このようにすると厳密な型指定が強制されるため、`widget` アイテムだけをコレクションに追加できるようになり、`Item` プロパティを使って取得される要素は常に `widget` オブジェクトです。また、厳密な型指定があるので、取得元のコードでは、`widget` から公開されるすべてのプロパティとメソッドを使用できます。

  または

- 定義済みのクラスを使わずに、独自のコレクション クラスを作成します。**Add** メソッドが `widget` オブジェクトだけを受け付けるように制限し、戻り値の型を `widget` として **Item** プロパティを実装します。詳細については、「方法：タイプ セーフなコレクションの定義」を参照してください。

  もし **Item** が **Object** 型の要素を返すように実装したとすると、明示的にアクセスできるのは **Object** クラスに定義されたプロパティとメソッドだけになります。`widget` メンバにアクセスするには、**Option Strict Off** に設定するか、次のコードに示すように CType 関数 を使って、返された要素を明示的に `widget` に変換します。

  VB

  ```
  Dim nextWidget As widget
  Try
      nextWidget = CType(widgetColl.Item(1), widget)
  Catch ex As Exception
      ' Insert code to run if the collection item is not a widget.
  End Try
  ```

いずれの場合にも、**Object** 要素をこの方法で使うと、Visual Basic では遅延バインディングが使用され、パフォーマンスが低下します。

Visual Basic の **Collection** クラスは **Object** 要素を受け取り、**Object** 要素を返すので、弱い型指定および遅延バインディングの欠点があります。

参照

関連項目

Option Strict ステートメント
System.Collections
System.Collections.Generic
System.Collections.Specialized

概念

Visual Basic のコレクション クラス
Visual Basic におけるコレクション
Visual Basic におけるジェネリック型

方法：オブジェクトのコレクションを作成する

他のオブジェクトの場合と同じように、オブジェクトを保持する変数を宣言し、コレクション オブジェクトを作成して変数に代入します。

コレクション オブジェクトには、Visual Basic のコレクション クラスと .NET Framework コレクション クラスのいずれかを使用できます。特に、System.Collections.Generic 名前空間のいずれかのクラスを使用すると、ジェネリック コレクションを作成できます。ジェネリック コレクションは、コレクション内のすべての項目が同じデータ型である場合に便利です。ジェネリック コレクションには該当するデータ型しか追加できないので、厳密な型指定が適用されます。詳細については、「方法 : タイプ セーフなコレクションの定義」を参照してください。

コレクション オブジェクトを作成した後は、コレクション内の項目を追加、削除およびアクセスできます。

コレクションを作成する 2 つの例を示します。各コレクションは **String** 型の項目を保持し、**String** 型のキーを各項目に関連付けます。最初の 2 つのプロシージャは、Visual Basic コレクション クラスを使用してコレクションを作成します。後の 2 つのプロシージャは、.NET Framework ジェネリック コレクション クラスを使用してコレクションを作成します。

Visual Basic コレクション クラスを使用してコレクションを作成するには

1. 次の例のようにして、Visual Basic **Collection** 変数を宣言および作成します。

```
Dim sampleVisualBasicColl As New Microsoft.VisualBasic.Collection()
```

   `sampleVisualBasicColl` のコレクションには、任意のデータ型の項目を格納できます。

2. コレクションに要素を追加するには、Add メソッド (Collection オブジェクト) を使用します。次の例では、4 つの **String** 要素を作成し、これらをコレクションに追加します。新規の各要素のキーとして一意の **String** 値を作成し、この値を **Add** メソッドに渡します。

```
Dim item1, item2, item3, item4 As String
item1 = "Items"
item2 = "In"
item3 = "A"
item4 = "Collection"
sampleVisualBasicColl.Add(item1, "firstkey")
sampleVisualBasicColl.Add(item2, "secondkey")
sampleVisualBasicColl.Add(item3, "thirdkey")
sampleVisualBasicColl.Add(item4, "fourthkey")
```

   Key 引数は、Visual Basic コレクションではオプションです。

3. コレクションから要素を削除する場合は、位置インデックスかオプションのキーのいずれによって要素を指定し、Remove メソッド (Collection オブジェクト) を使用して削除します。次に例を示します。

```
' Remove the first element of the Visual Basic collection.
sampleVisualBasicColl.Remove(1)
' Remove the element with the key "secondkey".
sampleVisualBasicColl.Remove("secondkey")
```

   Visual Basic **Collection** から要素を削除すると、インデックス値には 1 から Count プロパティ (Collection オブジェクト) の値までの番号が振り直されることに注意してください。

For Each...Next を使って Visual Basic コレクションの要素を処理するには

1. コレクションに格納されている型の変数を宣言します。上の例に対して、次の例に示すように、**String** 型の変数を宣言します。

```
' Insert code from the preceding example.
Dim aString As String
```

2. For Each...Next ステートメント (Visual Basic) を使用してコレクションの各要素を確認します。次の例では、特定の文字列を検索し、見つかった場合にはこれを表示します。

```
For Each aString in sampleVisualBasicColl
    If aString = "Collection" Then
        MsgBox(aString)
    End If
Next aString
```

ジェネリック コレクション クラスを使用してコレクションを作成するには

1. 次の例のようにして、.NET Framework System.Collections.Generic.Dictionary 変数を宣言および作成します。

```
Dim sampleGenericColl As New System.Collections.Generic.Dictionary(Of String, String)
```

   `sampleGenericColl` 変数は、**String**型の項目とキーだけを受け付けるタイプ セーフ コレクションを保持します。

2. コレクションに要素を追加するには、System.Collections.Generic.Dictionary.Add(メソッドを使用します。次の例では、4 つの **String** 要素を作成し、これらをコレクションに追加します。新規の各要素のキーとして一意の **String** 値を作成し、この値を **Add** メソッドに渡します。

```
Dim item1, item2, item3, item4 As String
item1 = "Items"
item2 = "In"
item3 = "A"
item4 = "Collection"
sampleGenericColl.Add("firstkey", item1)
sampleGenericColl.Add("secondkey", item2)
sampleGenericColl.Add("thirdkey", item3)
sampleGenericColl.Add("fourthkey", item4)
```

   Key 引数は、このジェネリック コレクションでは必須です。

3. コレクションから要素を削除するには、System.Collections.Generic.IDictionary.Remove(メソッドを使用します。削除する要素を識別するためのキーを指定する必要があります。次に例を示します。

```
If Not sampleGenericColl.Remove("thirdkey")
    ' Insert code to handle "thirdkey" not found in collection.
End If
```

   次のプロシージャで示すように、**For Each...Next** ステートメントを使用すると、コレクション内の要素をループして処理できます。

For Each...Next を使ってジェネリック コレクションの要素を処理するには

1. コレクションに格納されている型の変数を宣言します。上の例に対して、次の例に示すように、**String** 型の変数を宣言します。

```
' Insert code from the preceding example.
Dim aPair As KeyValuePair(Of String, String)
```

2. For Each...Next ステートメント (Visual Basic) を使用してコレクションの各要素を確認します。次の例では、特定の文字列を検索し、見つかった場合にはこれを表示します。

```
For Each aPair In sampleGenericColl
    If aPair.Value = "Items" Then
        MsgBox(aPair.Key & " -- " & aPair.Value)
    End If
Next aPair
```

参照

処理手順
方法 : オブジェクトの配列を作成する
関連項目
Collection オブジェクト (Visual Basic)
System.Collections
System.Collections.Generic
System.Collections.Specialized
概念
Visual Basic のコレクション クラス

方法：オブジェクトの配列を作成する

すべてのオブジェクトは参照型です。参照型の配列を宣言および使用する方法は、その他のデータ型の配列の場合と同様です。オブジェクト型の配列の要素はインデックスで取得できます。取得した要素は、その型の通常のオブジェクトと同じように操作できます。

また、配列には、配列変数を通じて利用できる検索および並べ替えの組み込み機能があります。これらのメソッドの詳細については、「Array」を参照してください。

オブジェクトの配列を作成するには

1. 次の例のように配列を宣言します。配列のインデックス番号は 0 から始まるので、格納される要素の数は、上限として宣言する値より 1 つ多くなります。

```
Dim x(10) As widget
' x now contains 11 elements of type widget, x(0) through x(10).
```

2. 配列の各要素を作成するか、または各要素に既存のオブジェクトへの参照を代入します。次にコード例を示します。

```
' Create each element of an array by using a loop.
For q As Integer = 0 To 10
    x(q) = New widget()
Next q
' Assign a reference to an existing object to two array elements.
Dim specialWidget As New widget()
x(0) = specialWidget
x(1) = specialWidget
```

1 つのオブジェクトへの参照を配列内の複数の要素に代入することもできます。

参照

処理手順

方法 : オブジェクトのコレクションを作成する

概念

オブジェクトのグループの管理

値型と参照型

その他の技術情報

Visual Basic における配列

方法：コレクションの項目を追加、削除、および取得する

Visual Basic **Collection** クラスには、項目の追加、削除、および取得のための組み込み機能が含まれています。

- 新規顧客など、新たに項目が作成または取得されたときは、すぐにコレクションに追加できます。
- 従業員が退職した場合など、項目がコレクションに属さなくなったときは、コレクションからその項目を削除できます。
- 学生の電話番号が変わった場合などは、コレクションから項目を取得してその内容を編集できます。

メモ：
要素を追加したり削除したりすると、**Collection** オブジェクトによって数値インデックス番号が自動的に更新されます。したがって、ある特定の要素を指す数値インデックスがたびたび変わる場合もあります。このため、数値インデックスの値を保存しても、プログラムでそれを使って後から同じ要素を取得できるとは限りません。この目的には、キーを使用します。

コレクションに項目を追加するには

- Add メソッド (Collection オブジェクト) を使用し、*Key* によって項目を指定します。

  ```
  object.Add(Item, Key [, {Before | After}])
  ```

  たとえば、作業指示の `ID` プロパティをキーとして使って、作業指示のコレクションに作業指示オブジェクトを追加するには、次のように呼び出しを行います。

  ```
  workOrders.Add(woNew, woNew.ID)
  ```

  上の呼び出しでは、`ID` プロパティが文字列であることを前提としています。このプロパティが数値 (**Long** 型の整数など) の場合は、ToString メソッドを使って **String** 型の値に変換する必要があります。これは、*Key* 引数が文字列値である必要があるためです。

  ```
  workOrders.Add(woNew, woNew.ID.ToString())
  ```

  キーは省略できます。コレクション内のオブジェクトにキーを関連付けない場合は、キーを指定しないでコレクションに追加することもできます。

  ```
  workOrders.Add(woNew)
  ```

  コレクション内のオブジェクトの順序を維持するには、オプションの *Before* 引数と *After* 引数を使用します。*Before* を指定してコレクションに追加した項目は、指定した項目の前に配置され、*After* を指定した場合は指定した項目の後に配置されます。たとえば、*Before* を 1 に設定すると、**Collection** のインデックス番号は 1 から始まるため、項目はコレクションの先頭に挿入されます。

  ```
  workOrders.Add(woNew, woNew.ID, 1)
  ```

  同様に、*After* 引数では、指定したインデックスの後に項目が追加されます。次の例では、項目が 3 番目の要素として追加されます。

  ```
  workOrders.Add(woNew, woNew.ID,,2)
  ```

  Before と *After* には、いずれか一方に値を指定することはできますが、一度に両方の値を指定することはできません。

コレクションから項目を削除するには

- Remove メソッド (Collection オブジェクト) を使用し、*Index* または *Key* のいずれかによって項目を指定します。

  ```
  object.Remove({Index | Key})
  ```

  Index 引数は、削除する項目の位置を示します。*Key* 引数は、コレクションに項目を追加するときに使用した文字列と同じ文字列です。コレクションの 3 番目にある要素のキーが "W017493" の場合、削除するには次の 2 つのステートメントのいずれも使用できます。

```
workOrders.Remove(3)
workOrders.Remove("W017493")
```

コレクションからすべての項目を削除するには

- Clear メソッド (Collection オブジェクト) を使用します。

  ```
  object.Clear()
  ```

  Clear メソッドはコレクションを空にします。

コレクションから項目を取得するには

1. Item プロパティ (Collection オブジェクト) を使用し、*Index* または *Key* のいずれかによって項目を指定します。

   ```
   variable = object.Item({Index | Key})
   ```

   Remove メソッドと同様、*Index* 引数はコレクション内の項目の位置を示し、*Key* 引数は項目を追加したときに使用した文字列です。**Remove** メソッドと同じ例を使って説明すると、次のステートメントのいずれを使用しても、コレクションの 3 番目の要素を取得できます。

   ```
   woCurrent = workOrders.Item(3)
   woCurrent = workOrders.Item("W017493")
   ```

   メモ：

   キーとして数値を使用する場合は、ToString メソッドを使用してキーを文字列に変換してから、**Add** メソッドや **Remove** メソッド、または **Item** プロパティに渡す必要があります。Visual Basic **Collection** オブジェクトでは、数値は必ず、キーの文字列ではなく、インデックスであると見なされます。

2. コンパイル時にキーがわかっている場合は、別の方法として、ディクショナリ アクセス演算子 (**!**) を使用して、キーを引用符やかっこで囲まないでコレクションの要素にアクセスできます。その場合、上の呼び出しは次のように記述することもできます。

   ```
   woCurrent = workOrders!W017493
   ```

参照

関連項目

Collection オブジェクトのメンバ
Add メソッド (Collection オブジェクト)
Remove メソッド (Collection オブジェクト)
Clear メソッド (Collection オブジェクト)
Item プロパティ (Collection オブジェクト)

概念

Visual Basic のコレクション クラス
配列の代わりとしてのコレクション
Visual Basic におけるコレクション

方法：Visual Basic でコレクションを反復処理する

For Each ループを使用して、コレクションのすべての要素を反復処理できます。

使用例

For Each...Next ステートメント (Visual Basic) を使用してコレクションのすべての項目にアクセスするコード例を次に示します。

```
Dim testCollection As New Microsoft.VisualBasic.Collection()
' The collection is empty until you add one or more items to it.
For Each collectionItem As Object In testCollection
    ' Perform desired processing on each item.
Next collectionItem
```

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System 名前空間へのアクセス

参照

処理手順

方法 : コレクションの項目を追加、削除、および取得する

関連項目

For Each...Next ステートメント (Visual Basic)

概念

Visual Basic におけるコレクション

コレクションのトラブルシューティング

ここでは、コレクションを使用する際に起きる一般的な問題についていくつか説明します。

間違った型のコレクションの使用

Visual Basic で開発を行う場合、Visual Basic の **Collection** クラスと .NET Framework に用意されているコレクション クラスを合わせた、さまざまな型のコレクションを使用できます。これらのクラスは互いに互換性がありません。つまり、変数をあるコレクション型で宣言した場合、その変数には別の型のオブジェクトを代入できません。また、自分が宣言したコレクション型のメソッドとプロパティにしかアクセスできません。

Visual Basic のコレクション クラスと .NET Framework のコレクション クラスには、次のような重要な違いがあります。

- **インデックス番号** .NET Framework のコレクションは 0 ベースですが、Visual Basic のコレクションは 1 ベースです。つまり、Visual Basic のコレクションの要素は、1 から Count プロパティ (Collection オブジェクト) の値までのインデックス値を持ち、.NET Framework のコレクションの要素は、0 からコレクションの `Count` プロパティの値 -1 までのインデックス値を持ちます。
- **要素の型** Visual Basic のコレクションは、**Object** 型の要素をサポートしますが、これにはどのデータ型の要素でも追加できるため、タイプ セーフではありません。通常、これはパフォーマンスの低下を招きます。なぜなら、コンパイラが要素を ボックス化およびボックス化解除して、オブジェクト型 (Object)と相互に変換することが必要になるからです。.NET Framework のコレクションには **Object** 型の要素を持つものもありますが、その他の多くは厳密に型指定されおり、特定の型の要素をサポートします。これによって要素がタイプ セーフになり、通常はパフォーマンスが最適化されます。
- **キー付きの要素** Visual Basic のコレクションを使用すると、要素をコレクションに追加するときにキーを指定できます。キーは一意の **String** 値です。作成後にこれを使用して、特定の要素にアクセスできます。.NET Framework のコレクションは、キーに関してこれと同じではなく、キーをサポートするものもあれば、しないものもあります。

各種コレクション クラスの定義を含む名前空間を次に示します。

- Microsoft.VisualBasic — Visual Basic の **Collection** クラス
- System.Collections — リスト、キュー、ビット配列、ハッシュ テーブル、およびディクショナリなどの特殊なコレクション クラス
- System.Collections.Generic — ジェネリック コレクション クラス。厳密に型指定されたコレクションの作成と、コレクション作成時の要素のデータ型の指定が可能
- System.Collections.Specialized — 特殊かつ厳密に型指定されたコレクション クラス。リンクされたリストとハイブリッド ディクショナリ、ビット ベクタと名前オブジェクトのコレクション、および文字列のみのコレクションなど

正しい方法

どの型のコレクションが自分の要件に最も適しているかを判断します。コレクション変数をその型で宣言します。オブジェクトも必ず同じ型で作成してください。目的のコレクション型を確実に指定するために、コレクションを完全に修飾します。2 つのコレクションを完全に修飾して宣言する例を、次に示します。

VB

```
Dim customers As New Microsoft.VisualBasic.Collection()
Dim stringQueue As New System.Collections.Generic.Queue(Of String)
```

特定の型のコレクションを作成したら、その型に定義されたメソッドとプロパティだけを使用していることを確認します。**Option Strict On** を設定すると、不正なオブジェクトの代入やメンバへのアクセスを、コンパイル時にキャッチできます。

参照

関連項目

Option Strict ステートメント

概念

Visual Basic におけるコレクション

Visual Basic のコレクション クラス

Visual Basic におけるイベント

イベントは、何かが発生したことを伝えるためにオブジェクトが送信するメッセージです。イベントはデリゲートを使用して実装されます。デリゲートは、関数の参照を通じて関数を間接的に呼び出すことができるオブジェクト指向関数ポインタの形式をとります。

このセクションの内容

イベントとイベント ハンドラ

イベント、イベント ハンドラ、および関連用語の概要を示します。

WithEvents と Handles 句

WithEvents キーワードと **Handles** 句を使用してイベントをイベント ハンドラに関連付ける方法を示します。

AddHandler と RemoveHandler

AddHandler ステートメントと **RemoveHandler** ステートメントを使用してイベントをイベント ハンドラに動的に関連付ける方法を示します。

方法 : イベントをクラスに追加する

イベントを定義するプロセスについて説明します。

方法 : Visual Basic コード エディタでのイベント ハンドラの作成

Visual Basic コード エディタを使用して、イベントに応答するコードを作成する手順を示します。

方法 : イベントを発生させる (Visual Basic)

イベントの定義方法について説明し、**RaiseEvent** ステートメントを使用してイベントを発生させます。

方法 : イベントとハンドラを作成する (Visual Basic)

イベントおよびイベント ハンドラの定義方法について説明し、**AddHandler** ステートメントを使用してイベントとイベント ハンドラを関連付ける手順を示します。

方法 : イベント ハンドラを記述する

Handles 句または **AddHandler** ステートメントを使用してイベント ハンドラを記述する方法を示します。

方法 : Visual Basic でイベントを処理する

CauseEvent メソッドを呼び出すときにイベントを発生させるクラスを定義します。

チュートリアル : イベントの宣言と発生

クラスのイベントの宣言および発生のプロセスを順をおって説明します。

チュートリアル : イベントの処理

イベント ハンドラ プロシージャの記述方法を示します。

方法 : ブロックを回避するイベントを宣言する

イベント ハンドラを非同期に呼び出すことができるカスタム イベントの定義方法を示します。

方法 : メモリの使用量を節約するイベントを宣言する

イベントを処理するときにだけメモリを消費するカスタム イベントの定義方法を示します。

Visual Basic 2005 における継承されたイベント ハンドラのトラブルシューティング

継承コンポーネントのイベント ハンドラで発生する一般的な問題について説明します。

関連するセクション

Visual Basic でのデリゲート

Visual Basic におけるデリゲートの概要を説明します。

イベントの処理と発生

.NET Framework のイベント モデルについて概説します。

Windows フォーム内でのイベント ハンドラの作成

Windows フォーム オブジェクトに関連付けられているイベントの処理方法について説明します。

イベントとイベント ハンドラ

一連のプロシージャが順番に実行されるのが Visual Studio プロジェクトであると思っているかもしれませんが、実際には、ほとんどのプログラムはイベントによって駆動されています。つまり、実行の流れを決定するのは、イベントと呼ばれる外部の事象です。

イベントは重要な出来事が発生したことをアプリケーションに伝えるシグナルです。たとえば、ユーザーがフォームのコントロールをクリックすると、フォームは **Click** イベントを発生させてイベントを処理するプロシージャを呼び出すことができます。イベントは、個別のタスクの通信も確立できます。たとえば、アプリケーションが、並べ替えタスクをメイン アプリケーションとは別に実行するとします。ユーザーが並べ替えを取り消した場合、アプリケーションは並べ替えプロセスに停止を指示するキャンセル イベントを送信できます。

イベントの用語と概念

このセクションでは、Visual Basic のイベントに関連して使用される用語と概念について説明します。

イベントの宣言

次の例に示すように、イベントはクラス、構造体、モジュール、およびインターフェイス内で **Event** キーワードを使用して宣言します。

VB

```
Event AnEvent(ByVal EventNumber As Integer)
```

イベントの発生

イベントは、重要な出来事が発生したことを通知するメッセージと同じです。メッセージのブロードキャストの動作は、イベントの発生と呼ばれます。Visual Basic では、次の例に示すように **RaiseEvent** ステートメントを使用してイベントを発生させます。

VB

```
RaiseEvent AnEvent(EventNumber)
```

イベントが発生する範囲は、イベントを宣言したクラス、モジュール、または構造体の内部に限られます。たとえば、派生クラスで基本クラスから継承したイベントを発生させることはできません。

イベント センダ

イベントを発生させる機能を持つオブジェクトがイベント センダです。イベント ソースとも呼ばれます。フォーム、コントロール、およびユーザー定義オブジェクトは、イベント センダの例です。

イベント ハンドラ

イベント ハンドラは、対応するイベントが発生したときに呼び出されるプロシージャです。シグネチャの一致する任意の有効なサブルーチンをイベント ハンドラとして使用できます。ただし、イベント ハンドラはイベント ソースに値を返すことができないため、関数をイベント ハンドラとして使用することはできません。

Visual Basic は、イベント ハンドラに対して、イベント センダ名、アンダースコア、およびイベント名を組み合わせた標準名前付け規則を使用します。たとえば、`button1` という名前のボタンの **Click** イベントは `Sub button1_Click` という名前になります。

メモ：

独自のイベントのイベント ハンドラを定義する場合は、この名前付け規則を使用することをお勧めしますが、名前付け規則の使用は必須ではありません。任意の有効なサブルーチン名を使用できます。

イベントとイベント ハンドラの関連付け

イベント ハンドラを使用できるようにするには、**Handles** ステートメントまたは **AddHandler** ステートメントを使用してイベント ハンドラをイベントに関連付ける必要があります。

WithEvents ステートメントと **Handles** 句を使用すると、イベント ハンドラの指定を宣言できます。**WithEvents** で宣言されたオブジェクトが発生させたイベントは、このイベントの名前を指定する **Handles** 句を持つ任意のサブルーチンで処理できます。**Handles** 句はイベントをイベント ハンドラに関連付ける標準的な方法ですが、関連付けができるのはコンパイル時だけです。

AddHandler ステートメントと **RemoveHandler** ステートメントは、**Handles** 句よりも柔軟性があります。これらのステートメントを使用すると、実行時に 1 つ以上のイベント ハンドラに対してイベントを動的に接続および接続解除できます。また、**WithEvents** を使用してオブジェクト変数を宣言する必要はありません。ただし、**WithEvents** の使用には制限事項があります。詳細については、

「WithEvents と Handles 句」を参照してください。

フォームやコントロールに関連付けられているイベントの場合、Visual Basic が空のイベント ハンドラを自動的にスタブとして作成し、それをイベントに関連付けることがあります。たとえば、デザイン モードでフォームのコマンド ボタンをダブルクリックすると、次のコードに示すように、Visual Basic はコマンド ボタンに対して空のイベント ハンドラと **WithEvents** 変数を作成します。

VB

```
Friend WithEvents Button1 As System.Windows.Forms.Button
Protected Sub Button1_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles Button1.Click
End Sub
```

参照

処理手順

方法 : イベント ハンドラを記述する

関連項目

Handles

AddHandler ステートメント

概念

AddHandler と RemoveHandler

WithEvents と Handles 句

WithEvents と Handles 句

WithEvents ステートメントと **Handles** 句を使用すると、イベント ハンドラの指定を宣言できます。**WithEvents** キーワードで宣言されたオブジェクトが発生させたイベントは、次の例に示すように、そのイベントの **Handles** ステートメントを持つ任意のプロシージャで処理できます。

VB

```
' Declare a WithEvents variable.
Dim WithEvents EClass As New EventClass

' Call the method that raises the object's events.
Sub TestEvents()
    EClass.RaiseEvents()
End Sub

' Declare an event handler that handles multiple events.
Sub EClass_EventHandler() Handles EClass.XEvent, EClass.YEvent
    MsgBox("Received Event.")
End Sub

Class EventClass
    Public Event XEvent()
    Public Event YEvent()
    ' RaiseEvents raises both events.
    Sub RaiseEvents()
        RaiseEvent XEvent()
        RaiseEvent YEvent()
    End Sub
End Class
```

多くの場合、イベント ハンドラに最も適しているのは **WithEvents** ステートメントと **Handles** 句です。これは、**WithEvents** ステートメントと **Handles** 句が使用する宣言構文によってイベント処理のコーディング、読み取り、およびデバッグが簡単になるためです。ただし、**WithEvents** 変数の使用には以下の制限があります。

- **WithEvents** 変数を、オブジェクト変数として使用することはできません。つまり、この変数をオブジェクト型 (**Object**) として宣言することはできません。変数を宣言するときは、クラス名を指定する必要があります。
- 共有イベントはクラスのインスタンスに関連付けられないため、**WithEvents** を使用して、共有イベントを宣言によって処理することはできません。同様に、**WithEvents** または **Handles** を使用して、イベントを **Structure** から処理することもできません。どちらの場合にも、**AddHandler** ステートメントを使ってそれらのイベントを処理するようにしてください。
- **WithEvents** 変数の配列は作成できません。
- **WithEvents** 変数では、1 つのイベント ハンドラが 1 つ以上の種類のイベント、または 1 つ以上のイベント ハンドラが同じ種類のイベントを処理することも可能です。

参照

関連項目

Handles
WithEvents
AddHandler ステートメント

概念

AddHandler と RemoveHandler

AddHandler と RemoveHandler

AddHandler ステートメントは、イベント ハンドラを指定できるという点で **Handles** 句と似ています。ただし、**AddHandler** を **RemoveHandler** と一緒に使用すると、**Handles** 句よりも柔軟性が増し、イベントに関連付けられたイベント ハンドラを動的に追加、削除、変更できます。共有イベントまたは構造体からのイベントを処理する場合は、**AddHandler** を使用する必要があります。

AddHandler は 2 つの引数 (コントロールなどのイベント センダから渡されるイベント名およびデリゲートを評価する式) を使用します。**AddressOf** ステートメントは常にデリゲートへの参照を返すため、**AddHandler** を使用する場合はデリゲート クラスを明示的に指定する必要がありません。オブジェクトが発生させたイベントにイベント ハンドラを関連付ける方法を次の例に示します。

VB

```
AddHandler Obj.XEvent, AddressOf Me.XEventHandler
```

イベント ハンドラとイベントの接続を解除する **RemoveHandler** では、**AddHandler** と同じ構文を使用します。たとえば、次のようにします。

VB

```
RemoveHandler Obj.XEvent, AddressOf Me.XEventHandler
```

参照

処理手順

方法 : イベント ハンドラを記述する

関連項目

AddHandler ステートメント

概念

イベントとイベント ハンドラ

WithEvents と Handles 句

方法：イベントをクラスに追加する

イベントをクラスに追加するには、そのイベントを **Events** ステートメントで宣言します。宣言にはイベントの名前とイベントで使用される引数が含まれます。

イベントをクラスに追加すると、そのクラスのオブジェクトが特定のイベントを発生させることができるように指定されます。イベントを実際に発生させるには、**RaiseEvent** ステートメントを使用する必要があります。**Handles** キーワードまたは **AddHandler** ステートメントを使用してイベント ハンドラ プロシージャにイベントを関連付けることができます。イベントは、そのイベントが宣言されたスコープ内で発生させる必要があります。たとえば、派生クラスで基本クラスから継承したイベントを発生させることはできません。

メモ：
イベントには、戻り値、省略可能な引数、および引数 **ParamArray** を指定できません。

イベントをクラスに追加するには

- クラスを定義するクラス モジュールの宣言セクションで **Event** ステートメントを使用し、必要な引数を指定してイベントを宣言します。たとえば、次のようにします。

  VB

```
Public Event PercentDone(ByVal Percent As Single, _
                         ByRef Cancel As Boolean)
```

参照

処理手順

チュートリアル：イベントの宣言と発生

チュートリアル：イベントの処理

方法：イベント ハンドラを記述する

関連項目

RaiseEvent ステートメント

Handles

AddHandler ステートメント

概念

イベントとイベント ハンドラ

デリゲートと AddressOf 演算子

AddHandler と RemoveHandler

その他の技術情報

ポリモーフィズム

方法：Visual Basic コード エディタでのイベント ハンドラの作成

Visual Basic コード エディタ を使用して、Windows フォームのイベント ハンドラを簡単に作成できます。Visual Basic コード エディタでは、1 つのイベント ハンドラに複数のイベントを接続することはできませんが
(「方法 : Windows フォームの 1 つのイベント ハンドラに複数のイベントを関連付ける」を参照)、フォームのコードの編集中に、Windows フォーム デザイナ内でそのフォームを開く必要がないため、イベント ハンドラをすばやく簡単に作成できます

Visual Basic コード エディタを使用して、イベント ハンドラを作成するには

1. コード エディタの [クラス名] ボックスで、イベント ハンドラを作成するフォームまたはコントロールを選択します。
2. コード エディタの上部にある [メソッド名] リストで、イベント ハンドラを作成するイベントを選択します。

   イベント ハンドラが作成され、作業中のフォームのクラスに追加されます。

参照

処理手順

方法 : デザイナを使用してイベント ハンドラを作成する
Visual Basic 2005 における継承されたイベント ハンドラのトラブルシューティング

概念

イベント ハンドラの概要 (Windows フォーム)

その他の技術情報

Windows フォーム内でのイベント ハンドラの作成
新しい Windows フォームの作成

方法 : イベントを発生させる (Visual Basic)

次に示すのは、イベント (`TimeExpired`) を定義し、**RaiseEvent** ステートメントを使用して、このイベントを発生させる例です。

使用例

VB

```
Public Event TimeExpired(ByVal Status As String)
Public Sub RaiseTimeExpiredEvent()
    RaiseEvent TimeExpired("Your time has run out")
End Sub
```

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System 名前空間のメンバへのアクセス許可。メンバ名を完全に修飾しないでコードに記述する場合は、**Imports** ステートメントを追加してください。詳細については、「Imports ステートメント」を参照してください。
- **Event** ステートメントは、プロシージャ内ではなく、クラス レベルで記述する必要があります。
- **RaiseEvent** ステートメントは、アプリケーションのプロシージャ内に記述する必要があります。

参照

処理手順

方法 : イベントとハンドラを作成する (Visual Basic)

関連項目

Event ステートメント

RaiseEvent ステートメント

概念

イベントとイベント ハンドラ

その他の技術情報

Visual Basic におけるイベント

方法：イベントとハンドラを作成する (Visual Basic)

次に示すのは、イベント `TimeExpired` とイベント ハンドラ `HandleTimeExpired` を定義し、**AddHandler** ステートメントを使用してこれらを関連付ける例です。

使用例

VB

```
Public Event TimeExpired(ByVal Status As String)
Public Sub HandleTimeExpired(ByVal Status As String)
    ' Perform desired processing for when time has expired.
    MsgBox("HandleTimeExpired caught the TimeExpired event" & _
           vbCrLf & "Status = " & Status)
End Sub
Public Sub SetUpEventHandler()
    AddHandler TimeExpired, AddressOf HandleTimeExpired
End Sub
```

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System 名前空間のメンバへのアクセス許可。メンバ名を完全に修飾しないでコードに記述する場合は、**Imports** ステートメントを追加してください。詳細については、「Imports ステートメント」を参照してください。
- **Event** ステートメントは、プロシージャ内ではなく、クラス レベルで記述する必要があります。
- **Event** ステートメント、および `HandleTimeExpired` と `SetUpEventHandler` の両プロシージャは、同じクラスまたはモジュール内に定義する必要があります。そのように定義しない場合は、**AddHandler** ステートメントによって、イベントとハンドラをそれらが定義されているオブジェクトに限定する必要があります。

参照

処理手順

方法 : イベントを発生させる (Visual Basic)

関連項目

Event ステートメント

AddHandler ステートメント

概念

イベントとイベント ハンドラ

その他の技術情報

Visual Basic におけるイベント

方法：イベント ハンドラを記述する

イベント ハンドラの生成方法は、そのイベント ハンドラをイベントにどのように関連付けるかによって決定されます。標準の方法でイベント ハンドラを作成するには、**Handles** キーワードと **WithEvents** キーワードを使用しますが、Visual Basic でイベントを処理する場合は **AddHandler** ステートメントを使うというもう 1 つの方法も選択できます。**AddHandler** と **RemoveHandler** を使うと、特定のイベントに対するイベント処理を動的に開始および停止できます。どちらのキーワードを使用してもかまいませんが、同じイベントに **WithEvents** と **AddHandler** の両方を使用することはできません。

WithEvents を使用したイベントの処理

WithEvents キーワードを使用すると、イベント ハンドラの **Handles** 句で使用できるクラス レベルまたはモジュール レベルのオブジェクト変数を作成できます。

WithEvents と Handles 句を使用してイベントを処理するには

1. イベントを含む単純なクラスを作成します。

   VB

   ```
   Class Class1
       Public Event AnEvent(ByVal EventNumber As Integer)
   End Class
   ```

2. イベントを処理するクラスまたはモジュールで、次の例に示すように **WithEvents** キーワードを使用してイベントのソースのオブジェクト変数を宣言します。

   VB

   ```
   Public WithEvents ClassInst As Class1
   ```

3. コード エディタで、左側の [クラス名] メニューから宣言した **WithEvents** 変数を選択します。

4. 右側の [メソッド名] メニューから処理するイベントを選択します。コード エディタは **Handles** 句を持つ空のイベント ハンドラ プロシージャを作成します。

   メモ：

   この手順は省略できます。作成するプロシージャがサブルーチンであり、処理されるイベントと一致する適切な引数リストおよび処理されるイベントを指定する **Handles** 句があれば、イベント ハンドラ プロシージャを手動で作成できます。

5. 指定された引数を使用して、イベント処理コードをイベント ハンドラ プロシージャに追加します。次に例を示します。

   VB

   ```
   Public Sub ClassInst_AnEvent(ByVal EventNumber As Integer) _
         Handles ClassInst.AnEvent
       MsgBox("Received event number: " & CStr(EventNumber))
   End Sub
   ```

AddHandler を使用したイベントの処理

AddHandler ステートメントを使用すると、イベントとイベント ハンドラ プロシージャを動的に接続できます。

AddHandler を使用してイベントを処理するには

1. イベントを処理するサブルーチンを、次の例のように作成します。

   VB

```
Public Sub EHandler(ByVal EventNumber As Integer)
    MsgBox("Received event number " & CStr(EventNumber))
End Sub
```

2. 処理するイベントのソースであるクラスのオブジェクト変数を宣言します。**WithEvents** 変数とは異なり、このオブジェクト変数はプロシージャ内のローカル変数にできます。たとえば、次のようにします。

   VB

   ```
   Public Sub TestAddHandler()
       Dim CI As New Class1
   End Sub
   ```

3. **AddHandler** ステートメントを使用してイベント センダの名前を指定し、**AddressOf** ステートメントを使用してイベント ハンドラの名前を指定します。たとえば、`TestAddHandler` サブルーチンの末尾に、次のコードを追加します。

   VB

   ```
   AddHandler CI.AnEvent, AddressOf EHandler
   ```

   処理されるイベントに対して適切な引数をサポートしていれば、すべてのプロシージャがイベント ハンドラとして機能できます。

RemoveHandler を使用したイベントの処理の停止

RemoveHandler ステートメントを使用すると、イベント ハンドラ プロシージャからイベントの接続を動的に解除できます。

RemoveHandler を使用してイベントの処理を停止するには

- **RemoveHandler** ステートメントを使用してイベント センダの名前を指定し、**AddressOf** ステートメントを使用してイベント ハンドラの名前を指定します。**RemoveHandler** ステートメントの構文は、イベント処理の開始に使用する **AddHandler** ステートメントと常に厳密に対応します。次に例を示します。

  VB

  ```
  RemoveHandler CI.AnEvent, AddressOf EHandler
  ```

基本クラスから継承されたイベントの処理

派生クラスは基本クラスから特性を継承したクラスであり、**Handles MyBase** ステートメントを使用して基本クラスが発生させたイベントを処理できます。

基本クラスから継承されたイベントを処理するには

- イベント ハンドラ プロシージャの宣言の行に **Handles MyBase.***eventname* ステートメントを追加して、派生クラスにイベント ハンドラを宣言します。*eventname* は処理する基本クラスのイベント名です。次に例を示します。

  VB

  ```
  Public Class BaseClass
      Public Event BaseEvent(ByVal i As Integer)
      ' Place methods and properties here.
  End Class

  Public Class DerivedClass
      Inherits BaseClass
      Sub EventHandler(ByVal x As Integer) Handles MyBase.BaseEvent
          ' Place code to handle events from BaseClass here.
      End Sub
  ```

```
End Class
```

参照

関連項目

Handles
AddHandler ステートメント

概念

イベントとイベント ハンドラ
デリゲートと AddressOf 演算子
AddHandler と RemoveHandler

方法：Visual Basic でイベントを処理する

`CauseEvent` メソッドを呼び出すときにイベントを発生させるクラスを定義する例を次に示します。このイベントは、`EventHandler` というイベント ハンドラ プロシージャによって処理されます。

例

この例を実行するには、次のコードを Visual Basic Windows アプリケーション プロジェクトのフォーム クラスに追加し、整数の引数を指定して `TestEvents` プロシージャを呼び出します。

VB

```
Public Class Class1
    ' Declare an event for this class.
    Public Event Event1(ByVal EventNumber As Integer)
    ' Define a method that raises an event.
    Sub CauseEvent(ByVal EventNumber As Integer)
        RaiseEvent Event1(EventNumber)
    End Sub
End Class

Protected Sub TestEvents(ByVal EventNumber As Integer)
    Dim Obj As New Class1
    AddHandler Obj.Event1, AddressOf Me.EventHandler
    ' Cause the object to raise an event.
    Obj.CauseEvent(EventNumber)
End Sub

Sub EventHandler(ByVal EventNumber As Integer)
    MsgBox("Received event number " & EventNumber.ToString)
End Sub
```

参照

処理手順

方法 : イベント ハンドラを記述する

概念

イベントとイベント ハンドラ

デリゲートと AddressOf 演算子

AddHandler と RemoveHandler

チュートリアル：イベントの宣言と発生

このチュートリアルでは、Widget というクラスのイベントを宣言および発生させる方法を説明します。ここでの手順が完了したら、関連トピックの「チュートリアル：イベントの処理」を参照してください。このトピックでは、Widget オブジェクトのイベントを使用して、アプリケーションのステータス情報を提供する方法を説明しています。

ウィジェット クラス

ここでは Widget クラスが既にあると仮定します。Widget クラスには、実行に時間がかかるメソッドが 1 つあるので、アプリケーションに完了のインジケータを用意することにします。

Widget オブジェクトに、完了した割合を示すダイアログ ボックスを表示させることもできますが、ここでは、Widget クラスを使用するすべてのプロジェクトでダイアログ ボックスを表示する処理を行います。オブジェクトの目的がフォームやダイアログ ボックスを制御することでない場合、オブジェクト設計の原則は、オブジェクトを使用するアプリケーションにユーザー インターフェイスを処理させることです。

Widget の目的は他のタスクを実行することです。したがって、PercentDone イベントを追加して、Widget のメソッドを呼び出すプロシージャにイベントの処理と更新状態の表示を実行させる方が適切です。PercentDone イベントは、タスクをキャンセルするしくみも提供できます。

このトピックのコード例を作成するには

1. 新しい Visual Basic Windows Application プロジェクトを開き、Form1 の名前でフォームを作成します。
2. Form1 に、2 つのボタンと 1 つのラベルを追加します。
3. 次の表のとおりにオブジェクトに名前を付けます。

オブジェクト	プロパティ	設定
Button1	Text	Start Task
Button2	Text	Cancel
Label	(Name), Text	lblPercentDone, 0

4. [プロジェクト] メニューの [クラスの追加] を選択し、プロジェクトに Widget.vb というクラスを追加します。

ウィジェット クラスのイベントを宣言するには

- **Event** キーワードを使って Widget クラス内にイベントを宣言します。イベントは **ByVal** および **ByRef** 引数を受け取ることができます。Widget の PercentDone イベントの例を示します。

  VB

```
Public Event PercentDone(ByVal Percent As Single, _
                         ByRef Cancel As Boolean)
```

呼び出し元のオブジェクトが PercentDone イベントを受け取るとき、Percent 引数には完了したタスクの割合が含まれています。Cancel 引数を **True** に設定すると、イベントを発生させるメソッドをキャンセルできます。

メモ：

イベントの引数は、プロシージャの引数と同様に宣言できます。ただし、イベントに対して **Optional** 引数や **ParamArray** 引数は指定できず、イベントには戻り値がありません。

PercentDone イベントは、Widget クラスの LongTask メソッドによって発生します。LongTask は 2 つの引数を受け取ります。1 つはメソッドの動作時間を表し、もう 1 つは PercentDone イベントを発生させるために LongTask が一時停止するまでの最小の間隔を表します。

PercentDone イベントを発生させるには

1. このクラスによって使用される **Timer** プロパティへのアクセスを簡単にするために、クラス モジュールの宣言セクションの先頭で、Class Widget ステートメントの上に **Imports** ステートメントを追加します。

VB

```
Imports Microsoft.VisualBasic.DateAndTime
```

2. `Widget` クラスに次のコードを追加します。

VB

```
Public Sub LongTask(ByVal Duration As Single, _
                    ByVal MinimumInterval As Single)
    Dim Threshold As Single
    Dim Start As Single
    Dim blnCancel As Boolean

    ' The Timer property of the DateAndTime object returns the seconds
    ' and milliseconds that have passed since midnight.
    Start = CSng(Timer)
    Threshold = MinimumInterval

    Do While CSng(Timer) < (Start + Duration)
        ' In a real application, some unit of work would
        ' be done here each time through the loop.
        If CSng(Timer) > (Start + Threshold) Then
            RaiseEvent PercentDone( _
            Threshold / Duration, blnCancel)
            ' Check to see if the operation was canceled.
            If blnCancel Then Exit Sub
            Threshold = Threshold + MinimumInterval
        End If
    Loop
End Sub
```

アプリケーションで `LongTask` メソッドを呼び出すと、`Widget` クラスは `PercentDone` イベントを `MinimumInterval` 秒間隔で発生させます。イベントが返されると、`LongTask` は `Cancel` 引数が **True** に設定されたかどうかを確認します。

ここで、次の点に注意してください。簡略化のために、`LongTask` プロシージャでは、タスクにかかる時間があらかじめわかっているとします。実際には、このような場合はほとんどありません。タスクを均等なサイズのチャンクに分割することは難しい場合があり、ユーザーにとって最も重要なことは、処理が進行中であることを知らされるまでの時間である場合も多くあります。

このサンプルに別の欠点があります。**Timer** プロパティは、深夜 0 時からの経過時間を秒単位で返します。したがって、アプリケーションが 0 時直前に開始された場合、アプリケーションは正しく動作しなくなります。時間を慎重に測定するには、このような境界条件を考慮に入れるか、**Now** などのプロパティを使用して問題が発生するのを避けます。

これで `Widget` クラスはイベントを発生できるようになったので、次のチュートリアルに進むことができます。「チュートリアル : イベントの処理」は、**WithEvents** を使ってイベント ハンドラを `PercentDone` イベントに関連付ける方法を示します。

参照

処理手順

チュートリアル : イベントの処理

方法 : イベント ハンドラを記述する

関連項目

Timer プロパティ

WithEvents

Now プロパティ

Event ステートメント

ByVal

ByRef

Imports ステートメント

概念

イベントとイベント ハンドラ

デリゲートと AddressOf 演算子
AddHandler と RemoveHandler

チュートリアル：イベントの処理

このトピックは、イベントの処理方法を示す 2 番目のトピックです。最初のトピック「チュートリアル：イベントの宣言と発生」では、イベントの宣言方法と発生方法について説明しました。ここでは、最初のトピックのチュートリアルのフォームとクラスを使用して、発生したイベントを処理する方法を示します。

`Widget` クラスの例では、従来のイベント処理ステートメントを使用します。Visual Basic には、これ以外のイベント処理テクニックも用意されています。この例を変更して **AddHandler** ステートメントと **Handles** ステートメントを使用することもできます。

Widget クラスの PercentDone イベントを処理するには

- `Form1` に次のコードを記述します。

  VB

  ```
  Private WithEvents mWidget As Widget
  Private mblnCancel As Boolean
  ```

  WithEvents キーワードは、変数 `mWidget` がオブジェクトのイベントを処理するために使用されることを指定します。オブジェクトの作成元のクラス名を指定して、オブジェクトの種類を指定します。

  `mWidget` 変数は `Form1` 内で宣言します。**WithEvents** の変数はクラス レベルにする必要があるからです。別のクラスの場合も同様です。

  変数 `mblnCancel` は、`LongTask` メソッドをキャンセルするために使用されます。

イベントを処理するコードの記述

WithEvents を使用して変数を宣言すると、クラスのコード エディタの左側のドロップダウン リストに変数名が表示されます。`mWidget` を選択すると、右側のドロップダウン リストに `Widget` クラスのイベントが表示されます。イベントを選択すると、対応するイベント プロシージャが表示されます。イベント プロシージャの名前は、先頭に `mWidget` およびアンダースコアが付きます。**WithEvents** 変数に関連付けられたイベント プロシージャの名前は、すべて先頭に変数名が付きます。

イベントを処理するには

1. コード エディタで、左側のドロップダウン リストの `mWidget` をクリックします。
2. 右側のドロップダウン リストの `PercentDone` イベントをクリックします。コード エディタ内に `mWidget_PercentDone` イベント プロシージャが表示されます。

   メモ：

   新しいイベント ハンドラを挿入するときにはコード エディタを使用すると便利ですが、必須ではありません。このチュートリアルでは、イベント ハンドラをコード内に直接コピーした方が簡単です。

3. `mWidget_PercentDone` イベント ハンドラに次のコードを追加します。

   VB

   ```
   Private Sub mWidget_PercentDone( _
       ByVal Percent As Single, _
       ByRef Cancel As Boolean _
   ) Handles mWidget.PercentDone
       lblPercentDone.Text = CInt(100 * Percent) & "%"
       My.Application.DoEvents()
       If mblnCancel Then Cancel = True
   End Sub
   ```

   `PercentDone` イベントが発生するたびに、イベント プロシージャは完了したパーセントを **Label** コントロールに表示します。**DoEvents** メ

ソッドにより、ラベルが再描画され、ユーザーに [キャンセル] をクリックする機会が与えられます。

4. `Button2_Click` イベント ハンドラに次のコードを追加します。

   VB

```
Private Sub Button2_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs _
) Handles Button2.Click
    mblnCancel = True
End Sub
```

`LongTask` の実行中にユーザーが [キャンセル] をクリックした場合は、**DoEvents** ステートメントによってイベント処理が許可されるとすぐに、`Button2_Click` イベントが実行されます。クラス レベル変数 `mblnCancel` が **True** に設定され、`mWidget_PercentDone` イベントがこれをテストして、`ByRef Cancel` 引数を **True** に設定します。

WithEvents 変数のオブジェクトへの接続

`Form1` は、`Widget` オブジェクトのイベントを処理するように設定されます。後は、`Widget` を見つけるだけです。

デザイン時に変数 **WithEvents** を宣言するときは、どのオブジェクトも関連付けられていません。**WithEvents** 変数は、他のオブジェクト変数と同じです。オブジェクトを作成し、**WithEvents** 変数を使用してオブジェクトへの参照を代入する必要があります。

オブジェクトを作成し、オブジェクトへの参照を代入するには

1. コード エディタで、左側のドロップダウン リストの [(Form1 イベント)] をクリックします。
2. 右側のドロップダウン リストの `Load` イベントをクリックします。コード エディタ内に `Form1_Load` イベント プロシージャが表示されます。
3. `Form1_Load` イベント プロシージャに次のコードを追加して、`Widget` を作成します。

   VB

```
Private Sub Form1_Load( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs _
) Handles MyBase.Load
    mWidget = New Widget
End Sub
```

このコードが実行されると、Visual Basic は `Widget` オブジェクトを作成し、そのイベントを `mWidget` に関連付けられたイベント プロシージャに接続します。これ以後、`Widget` が `PercentDone` イベントを発生させるたびに、`mWidget_PercentDone` イベント プロシージャが実行されます。

LongTask メソッドを呼び出すには

- `Button1_Click` イベント ハンドラに次のコードを追加します。

  VB

```
Private Sub Button1_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs _
) Handles Button1.Click
    mblnCancel = False
    lblPercentDone.Text = "0%"
    lblPercentDone.Refresh()
    mWidget.LongTask(12.2, 0.33)
    If Not mblnCancel Then lblPercentDone.Text = CStr(100) & "%"
End Sub
```

LongTask メソッドを呼び出す前に、完了したパーセントを表示するラベルを初期化する必要があります。また、メソッドをキャンセルするためのクラス レベルの **Boolean** フラグを **False** に設定する必要があります。

LongTask は、12.2 秒のタスク存続期間で呼び出されます。PercentDone イベントは、1/3 秒ごとに発生します。イベントが発生するたびに、mWidget_PercentDone イベント プロシージャが実行されます。

LongTask の実行が完了すると mblnCancel がテストされ、LongTask が正常に終了したかどうか、または mblnCancel が **True** に設定されたために実行が停止したかどうかが確認されます。完了したパーセントは、正常に終了したかどうかを確認する場合にだけ更新されます。

プログラムを実行するには

1. F5 キーを押してプロジェクトを実行モードにします。
2. [タスクの開始] をクリックします。PercentDone イベントが発生するたびに、タスクの完了したパーセント値でラベルが更新されます。
3. [キャンセル] をクリックして、タスクを停止します。[キャンセル] ボタンの外観は、クリックしてもすぐには変化しません。**My.Application.DoEvents** ステートメントによってイベント処理が許可されるまで、**Click** イベントは発生しません。

メモ :
My.Application.DoEvents メソッドは、フォームとまったく同じ方法でイベントを処理するわけではありません。たとえばこのチュートリアルでは、[キャンセル] を 2 回クリックする必要があります。フォームがイベントを直接処理するようにするには、マルチスレッドを使用します。詳細については、「Visual Basic におけるマルチスレッド」を参照してください。

プログラムを実行するときに、**F11** キーを押してコードを 1 行ずつ処理すると便利な場合があります。実行が LongTask に移り、PercentDone イベントが発生するたびに一時的に実行が Form1 に移るようすを明確に確認できます。

実行が Form1 のコードに戻っているときに、LongTask メソッドが再び呼び出された場合、どうなるか考えてください。最悪の場合、イベントが発生するたびに LongTask が呼び出されると、スタック オーバーフローが発生します。

新しい Widget への参照を mWidget 変数に代入すると、この mWidget 変数に異なる Widget オブジェクトのイベントを処理させることができます。実際、ボタンをクリックするたびに Button1_Click のコードを実行させることができます。

異なる Widget のイベントを処理するには

- Button1_Click プロシージャの mWidget.LongTask(12.2, 0.33) という行の直前に、次のコードを追加します。

  VB

```
mWidget = New Widget
' Create a new Widget object.
```

上記のコードは、ボタンがクリックされるごとに新しい Widget を作成します。LongTask メソッドが完了すると、すぐに Widget への参照が解放され、Widget は破棄されます。

WithEvents 変数には、一度に 1 つのオブジェクト参照しか保持できません。したがって、mWidget に別の Widget オブジェクトを代入すると、前の Widget オブジェクトのイベントは処理されなくなります。mWidget が、前の Widget への参照を含む唯一のオブジェクト変数である場合、オブジェクトは破棄されます。複数の Widget オブジェクトからのイベントを処理する場合は、**AddHandler** ステートメントを使用して、各オブジェクトからのイベントを個別に処理します。

メモ :
WithEvents 変数は必要な数だけ宣言できますが、**WithEvents** 変数の配列はサポートされません。

参照

処理手順

チュートリアル : イベントの宣言と発生

方法 : イベント ハンドラを記述する

関連項目

Handles

WithEvents

概念

イベントとイベント ハンドラ

デリゲートと AddressOf 演算子

AddHandler と RemoveHandler
その他の技術情報
ポリモーフィズム

方法：ブロックを回避するイベントを宣言する

イベント ハンドラが後続のイベント ハンドラをブロックしないことが重視される状況があります。カスタム イベントを使うと、イベントから自身のイベント ハンドラを非同期に呼び出すことができます。

既定では、イベント宣言のバッキング ストア フィールドは、すべてのイベント ハンドラが連続的に結合されたマルチキャスト デリゲートです。つまり、実行の完了に時間がかかるハンドラがあると、それが完了するまで他のハンドラはブロックされます。(実行に時間がかかったり、操作をブロックする可能性があったりするイベント ハンドラは、設計を見直すことをお勧めします。)

Visual Basic に用意された既定のイベント実装を使う代わりに、カスタム イベントを使うと、イベント ハンドラを非同期に実行できます。

使用例

次のコード例では、**AddHandler** アクセッサによって、`Click` イベントの各ハンドラのデリゲートが `EventHandlerList` フィールドに格納されている ArrayList に追加されます。

コードで `Click` イベントが生成されると、**RaiseEvent** アクセッサから、BeginInvoke メソッドを使ってすべてのイベント ハンドラ デリゲートが非同期に呼び出されます。このメソッドは、各ハンドラをワーカー スレッドに呼び出してから、すぐに制御を返すので、ハンドラが別のハンドラをブロックすることはありません。

VB

```
Public NotInheritable Class ReliabilityOptimizedControl
    'Defines a list for storing the delegates
    Private EventHandlerList As New ArrayList

    'Defines the Click event using the custom event syntax.
    'The RaiseEvent always invokes the delegates asynchronously
    Public Custom Event Click As EventHandler
        AddHandler(ByVal value As EventHandler)
            EventHandlerList.Add(value)
        End AddHandler
        RemoveHandler(ByVal value As EventHandler)
            EventHandlerList.Remove(value)
        End RemoveHandler
        RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
            For Each handler As EventHandler In EventHandlerList
                If handler IsNot Nothing Then
                    handler.BeginInvoke(sender, e, Nothing, Nothing)
                End If
            Next
        End RaiseEvent
    End Event
End Class
```

参照

処理手順

方法 : メモリの使用量を節約するイベントを宣言する

関連項目

Event ステートメント

ArrayList

BeginInvoke

方法：メモリの使用量を節約するイベントを宣言する

アプリケーションによるメモリの使用量を低く抑えることが重要になる場面はいくつもあります。カスタム イベントを使うと、アプリケーションは自分が処理するイベントに対してのみメモリを使用できます。

既定では、クラスでイベントが宣言されると、コンパイラはイベント情報を格納するフィールドのためにメモリを割り当てます。クラスに未使用のイベントが多数含まれると、メモリが不必要に消費されます。

Visual Basic に用意されている既定のイベントの実装を使うのではなく、カスタム イベントを使用することで、メモリの使用量を注意深く管理できます。

使用例

次の例では、クラスが EventHandlerList クラスの 1 つのインスタンスを `Events` フィールドに格納して使用します。ここには使用中のイベントに関する情報が格納されます。**EventHandlerList** クラスはデリゲートを保持するための、最適化されたリスト クラスです。

このクラスのすべてのイベントは、`Events` フィールドを使用して、各イベントをどのメソッドが処理しているかを持続的に追跡します。

VB

```
Public Class MemoryOptimizedBaseControl
    ' Define a delegate store for all event handlers.
    Private Events As New System.ComponentModel.EventHandlerList

    ' Define the Click event to use the delegate store.
    Public Custom Event Click As EventHandler
        AddHandler(ByVal value As EventHandler)
            Events.AddHandler("ClickEvent", value)
        End AddHandler
        RemoveHandler(ByVal value As EventHandler)
            Events.RemoveHandler("ClickEvent", value)
        End RemoveHandler
        RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
            CType(Events("ClickEvent"), EventHandler).Invoke(sender, e)
        End RaiseEvent
    End Event

    ' Define the Click event to use the same delegate store.
    Public Custom Event DoubleClick As EventHandler
        AddHandler(ByVal value As EventHandler)
            Events.AddHandler("DoubleClickEvent", value)
        End AddHandler
        RemoveHandler(ByVal value As EventHandler)
            Events.RemoveHandler("DoubleClickEvent", value)
        End RemoveHandler
        RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
            CType(Events("DoubleClickEvent"), EventHandler).Invoke(sender, e)
        End RaiseEvent
    End Event

    ' Define additional events to use the same delegate store.
    ' ...
End Class
```

参照

処理手順

方法 : ブロックを回避するイベントを宣言する

関連項目

Event ステートメント

EventHandlerList

Visual Basic 2005 における継承されたイベント ハンドラのトラブルシューティング

このトピックでは、継承コンポーネントのイベント ハンドラで発生する一般的な問題について説明します。

プロシージャ

イベント ハンドラのコードが呼び出しのたびに 2 回実行される

- 継承されたイベント ハンドラに Handles 句が含まれてないことを確認してください。基本クラス内のメソッドが既にイベントと関連付けられている場合は、そのイベントが発生します。継承されたメソッドから **Handles** 句を削除してください。

  VB

```
' INCORRECT
Protected Overrides Sub Button1_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click

    ' The Handles clause will cause all code
    ' in this block to be executed twice.
End Sub
```

- 継承されたメソッドに **Handles** キーワードが含まれていない場合は、コードに余分な AddHandler ステートメントがないこと、または同じイベントを処理する追加のメソッドがないことを確認してください。

参照

処理手順

方法 : イベント ハンドラを記述する

概念

イベントとイベント ハンドラ

方法 : Visual Basic でイベントを処理する

その他の技術情報

Visual Basic におけるイベント

Visual Basic でのデリゲート

デリゲートは、関数の参照を通じて関数を間接的に呼び出すことを可能にする、オブジェクト指向関数ポインタの形式をとります。デリゲートを使用すると、イベント ハンドラをフックして、あるプロシージャから別のプロシージャへプロシージャを渡すことができます。

このセクションの内容

デリゲートと AddressOf 演算子

デリゲートの概要とその使用方法について説明します。

方法 : デリゲート メソッドを呼び出す

クラスに並べ替えプロシージャを作成できます。このプロシージャは、ほとんどのリスト ボックスに対応する標準的なアルファベット順の並べ替えを使用しますが、実行時にカスタム並べ替えプロシージャに切り替えることができます。このプロシージャを作成するには、デリゲートを使用して実行時にカスタム並べ替えプロシージャを並べ替えクラスに渡します。

方法 : Visual Basic でプロシージャを別のプロシージャに渡す

デリゲートを使用してプロシージャを別のプロシージャに渡す方法について説明します。

関連するセクション

Visual Basic におけるイベント

Visual Basic におけるイベントの概要を説明します。

イベントの処理と発生

.NET Framework のイベント モデルについて概説します。

Windows フォーム内でのイベント ハンドラの作成

Windows フォーム オブジェクトに関連付けられているイベントの処理方法について説明します。

デリゲートと AddressOf 演算子

デリゲートは、ほかのオブジェクトのメソッドを呼び出すために使用できるオブジェクトです。デリゲートは他のプログラミング言語で使用される関数ポインタに似ているため、タイプ セーフ関数ポインタと説明されることがあります。しかし、関数のポインタとは異なり、Visual Basic のデリゲートは、System.Delegate クラスに基づく参照型です。デリゲートは、共有メソッド (特定のクラスのインスタンスがなくても呼び出すことのできるメソッド) とインスタンス メソッドの両方を参照できます。

デリゲートとイベント

デリゲートは、呼び出し側プロシージャと呼び出されるプロシージャの間の媒介手段が必要な状況で役立ちます。たとえば、イベントを発生させるオブジェクトが異なる状況で別個のイベント ハンドラを呼び出さなければならない場合があります。残念ながら、イベントを発生させるオブジェクトからは、どのイベント ハンドラがどのイベントを処理するのかをあらかじめ把握できません。Visual Basic では、**AddHandler** ステートメントを使用する場合、デリゲートを作成して、イベント ハンドラをイベントに動的に関連付けることができます。実行時に、デリゲートによって適切なイベント ハンドラに呼び出しが転送されます。

独自のデリゲートを作成することもできますが、ほとんどの場合、デリゲートの作成と詳細の管理は Visual Basic によって自動的に行われます。たとえば、**Event** ステートメントは、**Event** ステートメントを含むクラスの入れ子のクラスとして、`<EventName>EventHandler` という名前のデリゲート クラスを暗黙に定義します。定義の時はイベントと同じシグネチャを使用します。**AddressOf** ステートメントは、デリゲートのインスタンスを暗黙に作成します。たとえば、次の 2 つのコード行は同等です。

VB

```
AddHandler Button1.Click, AddressOf Me.Button1_Click
' The previous line of code is shorthand for the next line of code.
AddHandler Button1.Click, New EventHandler(AddressOf Button1_Click)
```

デリゲートの種類をコンパイラでコンテキストから判断できる場合は、簡単なデリゲートの作成方法を使用できます。

既存の種類のデリゲートを使用するイベントの宣言

状況によっては、基になるデリゲートとして既存の種類のデリゲートを使用するイベントを宣言しなければならない場合があります。その方法は次のとおりです。

VB

```
Delegate Sub DelegateType()
Event AnEvent As DelegateType
```

これは、同じハンドラに複数のイベントをルーティングする場合に役立ちます。

デリゲート変数とパラメータ

フリー スレッドまたはコンパイル時にさまざまな関数を呼び出す必要のあるプロシージャなど、イベントに関連しないほかのタスクにデリゲートを使用できます。

たとえば、自動車の名前のリスト ボックスを含む、項目別広告のアプリケーションがあるとします。広告はタイトルで並べ替えられます。通常、タイトルは自動車の型式です。問題が発生する可能性があるのは、型式の前に年式が含まれている自動車があるときです。問題になるのは、リスト ボックスに組み込まれた並べ替え機能の並べ替えの基準が文字コードだけであることです。このため、日付で始まるすべての広告が最初に配置され、その後に型式で始まる広告が配置されます。

この問題を解決するために、クラスに並べ替えプロシージャを作成できます。このプロシージャは、ほとんどのリスト ボックスに対応する標準的なアルファベット順の並べ替えを使用しますが、実行時に自動車広告用のカスタム並べ替えプロシージャに切り替えることができます。このプロシージャを作成するには、デリゲートを使用して実行時にカスタム並べ替えプロシージャを並べ替えクラスに渡します。

各デリゲート クラスでは、オブジェクト メソッドの仕様を引数とするコンストラクタを定義します。このようなデリゲート コンストラクタのパラメータ例を示す構文を次に示します。

AddressOf [*expression*.]*methodname*

コンパイル時の *expression* の型は、デリゲート クラスのシグネチャと同じシグネチャの、指定された名前のメソッドを持つクラスまたはインターフェイスである必要があります。メソッドは、共有 (クラス) メソッドまたはインスタンス メソッドのいずれかにできます。

参照

処理手順

方法 : Visual Basic でプロシージャを別のプロシージャに渡す
方法 : デリゲート メソッドを呼び出す
方法 : イベント ハンドラを記述する

関連項目

Delegate ステートメント
AddressOf 演算子

概念

イベントとイベント ハンドラ
AddHandler と RemoveHandler
マルチスレッド アプリケーション

方法：デリゲート メソッドを呼び出す

この例では、メソッドとデリゲートを関連付け、デリゲートによってメソッドを呼び出す方法について説明します。

デリゲートおよび一致プロシージャの作成

1. `MySubDelegate` という名前のデリゲートを作成します。

```
Delegate Sub MySubDelegate(ByVal x As Integer)
```

2. デリゲートと同じシグネチャを持つメソッドが含まれるクラスを宣言します。

```
Class class1
    Sub Sub1(ByVal x As Integer)
        MsgBox("The value of x is: " & CStr(x))
    End Sub
End Class
```

3. デリゲートのインスタンスが作成されるメソッドを定義し、組み込みの **Invoke** メソッドを呼び出して、デリゲートに関連付けられたメソッドを呼び出します。

```
Protected Sub DelegateTest()
    Dim c1 As New class1
    ' Create an instance of the delegate.
    Dim msd As MySubDelegate = AddressOf c1.Sub1
    ' Call the method.
    msd.Invoke(10)
End Sub
```

参照

処理手順

方法 : イベント ハンドラを記述する

関連項目

Delegate ステートメント

概念

デリゲートと AddressOf 演算子

イベントとイベント ハンドラ

AddHandler と RemoveHandler

マルチスレッド アプリケーション

方法：Visual Basic でプロシージャを別のプロシージャに渡す

この例では、プロシージャを他のプロシージャに渡すためにデリゲートを使用する方法について説明します。

デリゲートは、Visual Basic の他の型と同様に使用できる型です。**AddressOf** 演算子は、プロシージャ名に適用される場合、デリゲート オブジェクトを返します。

この例には、**AddressOf** 演算子を指定して取得した他のプロシージャへの参照を取得できるデリゲート パラメータを使ったプロシージャが含まれています。

デリゲートおよびマッチング プロシージャを作成する

1. `MathOperator` という名前のデリゲートを作成する

   VB

   ```
   Delegate Function MathOperator( _
       ByVal x As Double, _
       ByVal y As Double _
   ) As Double
   ```

2. `MathOperator` と一致するパラメータと戻り値を持つ `AddNumbers` という名前のプロシージャを、シグネチャが一致するように作成します。

   VB

   ```
   Function AddNumbers( _
       ByVal x As Double, _
       ByVal y As Double _
   ) As Double
       Return x + y
   End Function
   ```

3. `MathOperator` と一致するシグネチャを持つ `SubtractNumbers` という名前のプロシージャを作成します。

   VB

   ```
   Function SubtractNumbers( _
       ByVal x As Double, _
       ByVal y As Double _
   ) As Double
       Return x - y
   End Function
   ```

4. パラメータとしてデリゲートを取得する `DelegateTest` という名前のプロシージャを作成します。

   `AddNumbers` または `SubtactNumbers` のシグネチャが `MathOperator` のシグネチャと一致するため、このプロシージャでは、`AddNumbers` または `SubtactNumbers` への参照を受け入れることができます。

   VB

   ```
   Sub DelegateTest( _
       ByVal x As Double, _
       ByVal op As MathOperator, _
       ByVal y As Double _
   )
       Dim ret As Double
       ret = op.Invoke(x, y) ' Call the method.
       MsgBox(ret)
   ```

```
End Sub
```

5. パラメータとして `AddNumbers` に対するデリゲートを指定し、再びパラメータとして `SubtractNumbers` へのデリゲートが指定された `DelegateTest` を一度呼び出す `Test` という名前のプロシージャを作成します。

   VB

   ```
   Protected Sub Test()
       DelegateTest(5, AddressOf AddNumbers, 3)
       DelegateTest(9, AddressOf SubtractNumbers, 3)
   End Sub
   ```

   `Test` が呼び出されると、初めに `5` と `3` の加算を実行する `AddNumbers` の結果である 8 が表示されます。次に、`9` から `3` の減算を実行する `SubtractNumbers` の結果である 6 が表示されます。

参照

処理手順

方法 : デリゲート メソッドを呼び出す

関連項目

AddressOf 演算子

Delegate ステートメント

その他の技術情報

Visual Basic でのデリゲート

Visual Basic におけるインターフェイス

インターフェイスは、クラスで実装できるプロパティ、メソッド、およびイベントを定義します。インターフェイスでは機能を密接な関係を持つプロパティ、メソッド、およびイベントの小さなグループとして定義できます。これにより、既存のコードに影響を与えずにインターフェイスの拡張実装を開発できるため、互換性の問題が削減されます。追加のインターフェイスと実装を開発することにより、いつでも新機能を追加できます。

前のバージョンの Visual Basic では、インターフェイスを使用できましたが、直接作成することはできませんでした。今回のバージョンでは **Interface** ステートメントを使用できます。これにより、真のインターフェイスをクラスの別のエンティティとして定義したり、**Implements** キーワードを改良した形式を使用してインターフェイスを実装したりできます。

このセクションの内容

インターフェイスの概要

インターフェイスについて概説し、Visual Basic での実装方法を簡単に説明します。

インターフェイス定義

Interface ステートメントと **End Interface** ステートメントでインターフェイスを定義する方法について説明します。

Implements キーワードおよび Implements ステートメント

コードのセクションが特定のインターフェイスを実装することを示す方法を説明します。

Visual Basic でのインターフェイス実装例

インターフェイス実装の 3 つの例を示します。

インターフェイスを使用する状況

継承階層よりインターフェイスを使用した方がよい場合について説明します。

方法 : インターフェイスを作成および実装する

インターフェイスの定義と実装に関連する手順、および例を示します。

チュートリアル : インターフェイスの作成と実装

独自のインターフェイスを定義および実装するプロセスの詳細な手順を示します。

関連するセクション

Visual Basic の継承

Visual Basic が継承をどのようにサポートし、派生クラスの基礎として機能するクラスを定義できるようにしているかを説明します。

インターフェイスの概要

インターフェイスは、クラスと同様にプロパティ、メソッド、およびイベントのセットを定義します。ただし、クラスとは異なり、インターフェイスは実装を提供しません。インターフェイスはクラスによって実装され、クラスとは別のエンティティとして定義されます。

インターフェイスを実装するクラスは定義時にそのインターフェイスのあらゆる機能を厳密に実装する必要があるため、インターフェイスは「コントラクト」であると考えられます。

インターフェイスでは、機能を密接な関係を持つメンバの小さなグループとして定義できます。既存のコードに影響を与えずにインターフェイスの拡張実装を開発できるため、互換性の問題を最小限に抑えることができます。追加のインターフェイスと実装を開発することにより、いつでも新機能を追加できます。

インターフェイスの実装は拡張できますが、インターフェイス自体は公開後に変更できません。公開されたインターフェイスを変更すると既存のコードが無効になる可能性があります。インターフェイスを「コントラクト」と考えると、コントラクトを交わす両者の役割が明確になります。インターフェイスの公開側はそのインターフェイスを決して変更せず、実装側はデザインに従って正確にインターフェイスを実装することに合意します。

前のバージョンの Visual Basic では、インターフェイスを使用できましたが、直接作成することはできませんでした。現在のバージョンでは、**Interface** ステートメントを使用して実際のインターフェイスを定義できます。また、拡張バージョンの **Implements** キーワードを使用して、インターフェイスを実装できます。

参照

処理手順

方法 : インターフェイスを作成および実装する

チュートリアル : インターフェイスの作成と実装

関連項目

Interface ステートメント (Visual Basic)

概念

インターフェイス定義

Implements キーワードおよび Implements ステートメント

Visual Basic でのインターフェイス実装例

インターフェイスを使用する状況

その他の技術情報

Visual Basic の継承

インターフェイス定義

インターフェイス定義は、**Interface** と **End Interface** ステートメントで囲まれます。**Interface** ステートメントの後には、1 つ以上の継承インターフェイスの一覧を示す **Inherits** ステートメント (省略可能) を追加できます。**Inherits** ステートメントは、宣言内でコメント以外の他のすべてのステートメントよりも前にある必要があります。インターフェイス定義に使用するその他のステートメントは、**Event**、**Sub**、**Function**、**Property**、**Interface**、**Class**、**Structure**、および **Enum** ステートメントです。インターフェイスには、実装コードまたは実装コードに関連付けられた **End Sub** や **End Property** などのステートメントを含めることはできません。

名前空間では、インターフェイス ステートメントは既定で **Friend** となりますが、明示的に **Public** または **Friend** として宣言することもできます。クラス、モジュール、インターフェイス、および構造体の中で定義されているインターフェイスは、既定では **Public** ですが、明示的に **Public**、**Friend**、**Protected**、または **Private** として宣言することもできます。

メモ :

Shadows キーワードは、すべてのインターフェイス メンバに対して使用できます。**Overloads** キーワードは、インターフェイス定義に宣言された **Sub**、**Function**、および **Property** のステートメントに使用できます。また、**Property** ステートメントには **Default**、**ReadOnly**、または **WriteOnly** の修飾子を指定できます。その他の修飾子 (**Public**、**Private**、**Friend**、**Protected**、**Shared**、**Overrides**、**MustOverride**、または **Overridable**) は許可されません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

参照

処理手順

方法 : インターフェイスを作成および実装する
チュートリアル : インターフェイスの作成と実装

関連項目

Interface ステートメント (Visual Basic)
Inherits ステートメント
Overloads
Default (Visual Basic)

概念

インターフェイスの概要
Implements キーワードおよび Implements ステートメント
Visual Basic でのインターフェイス実装例
インターフェイスを使用する状況

その他の技術情報

Visual Basic の継承

Implements キーワードおよび Implements ステートメント

Visual Basic 予約語である **Implements** は、2 つの方法で使用されます。**Implements** ステートメントは、クラスまたは構造体によってインターフェイスが実装されていることを表します。**Implements** キーワードは、クラス メンバまたは構造体メンバによって、特定のインターフェイス メンバが実装されていることを表します。

ステートメントの実装

クラスまたは構造体によって、1 つ以上のインターフェイスが実装されている場合、**Class** ステートメントまたは **Structure** ステートメントの直後に、**Implements** ステートメントを含める必要があります。**Implements** ステートメントには、クラスによって実装されるインターフェイスのコンマ区切りの一覧が必要です。クラスまたは構造体には、**Implements** キーワードを使用して、すべてのインターフェイス メンバが実装される必要があります。

Implements キーワード

Implements キーワードには、実装されるインターフェイス メンバのコンマ区切りの一覧が必要です。通常、1 つのインターフェイス メンバだけが指定されますが、複数のメンバを指定することもできます。インターフェイス メンバの仕様は、クラス内の Implements ステートメントで指定する必要のあるインターフェイス名、ピリオド、および実装するメンバ関数、プロパティ、またはイベントの名前から構成されます。インターフェイス メンバを実装するメンバの名前には有効な識別子を使用できます。以前のバージョンの Visual Basic で使用されていた **InterfaceName_MethodName** 規則には制限されません。

たとえば、次のコードはインターフェイスのメソッドを実装する `Sub1` という名前のサブルーチンの宣言方法を示しています。

VB

```
Class Class1
    Implements interfaceclass.interface2

    Sub Sub1(ByVal i As Integer) Implements interfaceclass.interface2.Sub1
    End Sub
End Class
```

メンバを実装するパラメータの型と戻り値の型は、インターフェイス内のインターフェイス プロパティまたはメンバ宣言と一致する必要があります。インターフェイスの要素を実装する最も一般的な方法は、前の例で示したようにインターフェイスと同じ名前のメンバを使用することです。

インターフェイス メソッドの実装を宣言するため
に、**Overloads**、**Overrides**、**Overridable**、**Public**、**Private**、**Protected**、**Friend**、**Protected Friend**、**MustOverride**、**Default**、および **Static** などのインスタンス メソッド宣言に対して有効な属性を使用できます。**Shared** 属性はインスタンス メソッドではなくクラスを定義するため、使用できません。

Implements を使用すると、次の例に示すようにインターフェイスで定義されている複数のメソッドを実装する単一のメソッドを記述することもできます。

VB

```
Class Class2
    Implements I1, I2

    Protected Sub M1() Implements I1.M1, I1.M2, I2.M3, I2.M4
    End Sub
End Class
```

プライベート メンバを使用すると、インターフェイス メンバを実装できます。プライベート メンバがインターフェイスのメンバを実装する場合は、そのインターフェイス メンバがクラスのオブジェクト変数で直接使用できない場合でも、インターフェイスを通じて使用できるようになります。

参照

処理手順

方法 : インターフェイスを作成および実装する
チュートリアル : インターフェイスの作成と実装

関連項目

Implements ステートメント
Implements (Visual Basic)

概念

インターフェイスの概要
インターフェイス定義
Visual Basic でのインターフェイス実装例
インターフェイスを使用する状況

その他の技術情報

Visual Basic の継承

Visual Basic でのインターフェイス実装例

インターフェイスを実装するクラスでは、プロパティ、メソッド、およびイベントをすべて実装する必要があります。

2 つのインターフェイスの定義を次の例に示します。2 番目のインターフェイス `Interface2` は `Interface1` を継承し、追加のプロパティとメソッドを定義します。

VB

```
Interface Interface1
    Sub sub1(ByVal i As Integer)
End Interface

' Demonstrates interface inheritance.
Interface Interface2
    Inherits Interface1
    Sub M1(ByVal y As Integer)
    ReadOnly Property Num() As Integer
End Interface
```

次の例では、前の例で定義したインターフェイス `Interface1` を実装します。

VB

```
Public Class ImplementationClass1
    Implements Interface1
    Sub Sub1(ByVal i As Integer) Implements Interface1.sub1
        ' Insert code here to implement this method.
    End Sub
End Class
```

最後の例では、`Interface1` から継承されたメソッドを含む `Interface2` を実装します。

VB

```
Public Class ImplementationClass2
    Implements Interface2
    Dim INum As Integer = 0
    Sub sub1(ByVal i As Integer) Implements Interface2.sub1
        ' Insert code here that implements this method.
    End Sub
    Sub M1(ByVal x As Integer) Implements Interface2.M1
        ' Insert code here to implement this method.
    End Sub

    ReadOnly Property Num() As Integer Implements _
       Interface2.Num
        Get
            Num = INum
        End Get
    End Property
End Class
```

参照

処理手順

方法 : インターフェイスを作成および実装する

チュートリアル : インターフェイスの作成と実装

関連項目

Interface ステートメント (Visual Basic)

Implements ステートメント

概念

インターフェイスの概要

インターフェイスを使用する状況

インターフェイスは、オブジェクトの定義を実装から切り離すことができる非常に強力なプログラミング ツールです。インターフェイスとクラス継承にはそれぞれ長所と短所があり、プロジェクト内で結局両方を組み合わせて使用することもあります。このページおよび「継承を使用する状況」は、状況に最も適した方法を判断するために役立ちます。

実装における柔軟性

クラス継承ではなくインターフェイスを使用する理由は他にもあります。

- インターフェイスは、特定の機能を提供するために、関連性のない多数のオブジェクト型がアプリケーションで必要な場合に適しています。
- 複数のインターフェイスを実装できる単一の実装を定義できるので、インターフェイスは基本クラスよりも柔軟性があります。
- インターフェイスは基本クラスから実装を継承する必要のない場合に適しています。
- インターフェイスはクラス継承を使用できない場合に役立ちます。たとえば、構造体はクラスから継承できませんが、インターフェイスを実装できます。

参照

処理手順

方法 : インターフェイスを作成および実装する
チュートリアル : インターフェイスの作成と実装

概念

継承を使用する状況
インターフェイスの概要
インターフェイス定義
Implements キーワードおよび Implements ステートメント
Visual Basic でのインターフェイス実装例

その他の技術情報

Visual Basic の継承

方法：インターフェイスを作成および実装する

「インターフェイスの概要」で説明したように、インターフェイスは実装を提供せずにクラスのプロパティ、メソッド、およびイベントを記述します。

インターフェイスを作成するには

1. **Interface** キーワードとインターフェイスの名前で始まり **End Interface** ステートメントで終わるコードを追加することにより、インターフェイスを定義します。たとえば、次のコードは、`IAsset` という名前のインターフェイスを定義します。

   VB

   ```
   Interface IAsset
   End Interface
   ```

2. インターフェイスでサポートするプロパティ、メソッド、およびイベントを定義するステートメントを追加します。たとえば、次のコードは 1 つの関数、1 つのプロパティ、および 1 つのイベントを定義します。

   VB

   ```
   Interface IAsset
       Event ComittedChange(ByVal Success As Boolean)
       Property Division() As String
       Function GetID() As Integer
   End Interface
   ```

インターフェイスを実装するには

1. 実装するインターフェイスがプロジェクトの一部でない場合は、インターフェイスを含むアセンブリへの参照を追加します。
2. インターフェイスを実装する新規クラスを作成し、クラス名の後の行に **Implements** キーワードを指定します。たとえば、次のコードでは、`IAsset` インターフェイスを実装する際に、実装クラスの名前を `Computer` としています。

   VB

   ```
   Class Computer
       Implements IAsset
   End Class
   ```

3. 次のコードに示すように、クラスのプロパティ、メソッド、およびイベントを実装するプロシージャを追加します。

   VB

   ```
   Class Computer
       Implements IAsset

       Public Event ComittedChange(ByVal Success As Boolean) _
          Implements IAsset.ComittedChange

       Private divisionValue As String

       Public Property Division() As String _
           Implements IAsset.Division

           Get
               Return divisionValue
           End Get
           Set(ByVal value As String)
   ```

```
            divisionValue = value
            RaiseEvent ComittedChange(True)
        End Set
    End Property

    Private IDValue As Integer

    Public Function GetID() As Integer _
        Implements IAsset.GetID

        Return IDValue
    End Function

    Public Sub New(ByVal Division As String, ByVal ID As Integer)
        Me.divisionValue = Division
        Me.IDValue = ID
    End Sub
End Class
```

参照

処理手順

チュートリアル : インターフェイスの作成と実装

関連項目

Interface ステートメント (Visual Basic)

概念

インターフェイスの概要

インターフェイス定義

Implements キーワードおよび Implements ステートメント

Visual Basic でのインターフェイス実装例

インターフェイスを使用する状況

その他の技術情報

Visual Basic の継承

チュートリアル：インターフェイスの作成と実装

インターフェイスはプロパティ、メソッド、およびイベントの特性を記述しますが、その実装の詳細は構造体やクラスに依存しています。

このチュートリアルでは、インターフェイスの宣言および実装の方法を説明します。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

インターフェイスを定義するには

1. 新しい Visual Basic Windows アプリケーション プロジェクトを開きます。
2. [プロジェクト] メニューの [標準モジュールの追加] をクリックして、新規モジュールをプロジェクトに追加します。
3. 新規モジュールに `Module1.vb` という名前を付け、[追加] をクリックします。新規モジュールのコードが表示されます。
4. **Module** ステートメントと **End Module** ステートメントの間に「`Interface TestInterface`」と入力し、Enter を押して、`TestInterface` という名前のインターフェイスを `Module1` で定義します。コード エディタによって **Interface** キーワードがインデントされ、**End Interface** ステートメントが追加されてコード ブロックが形成されます。
5. **Interface** ステートメントと **End Interface** ステートメントの間に次のコードを記述して、インターフェイスのプロパティ、メソッド、およびイベントを定義します。

   VB

   ```
   Property Prop1() As Integer
   Sub Method1(ByVal X As Integer)
   Event Event1()
   ```

実装

インターフェイス メンバを宣言する構文は、クラス メンバを宣言する構文と異なります。この違いは、インターフェイスには実装コードを含めることができないという事実を表しています。

インターフェイスを実装するには

1. `Module1` に `ImplementationClass` という名前のクラスを追加します。`End Interface` ステートメントと `End Module` ステートメントの間に次のステートメントを追加し、Enter キーを押します。

   VB

   ```
   Class ImplementationClass
   ```

   統合開発環境で作業している場合、Enter キー を押すことによって、対応する `End Class` ステートメントがコード エディタに追加されます。

2. 次の **Implements** ステートメントを `ImplementationClass` に追加します。これによって、クラスが実装するインターフェイスが指定されます。

   VB

   ```
   Implements TestInterface
   ```

   Implements ステートメントがクラスまたは構造体の先頭で他の項目と分かれて記述されている場合は、クラスまたは構造体がインター

フェイスを実装することを意味します。

統合開発環境で作業している場合、Enter を押すとコード エディタによって `TestInterface` で必要なクラス メンバが実装され、次の手順は省略できます。

3. 統合開発環境で作業している場合、インターフェイス `MyInterface` のすべてのメンバを実装する必要があります。次のコードを `ImplementationClass` に追加して `Event1`、`Method1`、`Prop1` を実装します。

   VB

   ```
   Event Event1() Implements TestInterface.Event1

   Public Sub Method1(ByVal X As Integer) Implements TestInterface.Method1
   End Sub

   Public Property Prop1() As Integer Implements TestInterface.Prop1
       Get
       End Get
       Set(ByVal value As Integer)
       End Set
   End Property
   ```

   Implements ステートメントは、実装されるインターフェイスとインターフェイスのメンバの名前を指定します。

4. プロパティの値を格納したクラスにプライベート フィールドを追加することで、`Prop1` の定義を完了します。

   VB

   ```
   ' Holds the value of the property.
   Private pval As Integer
   ```

   `pval` の値を、プロパティの get アクセサから返します。

   VB

   ```
   Return pval
   ```

   `pval` の値を、プロパティの set アクセサに設定します。

   VB

   ```
   pval = value
   ```

5. 次のコードを追加して、`Method1` の定義を完了します。

   VB

   ```
   MsgBox("The X parameter for Method1 is " & X)
   RaiseEvent Event1()
   ```

インターフェイスの実装をテストするには

1. ソリューション エクスプローラでプロジェクトのスタートアップ フォームを右クリックし、[コードの表示] をクリックします。スタートアップ フォームのクラスがエディタに表示されます。スタートアップ フォームの名前は既定で `Form1` となります。

2. 次の `testInstance` フィールドを `Form1` クラスに追加します。

   VB

```
Dim WithEvents testInstance As TestInterface
```

`testInstance` を **WithEvents** として宣言することで、`Form1` クラスはそのイベントを処理できるようになります。

3. `testInstance` によって発生するイベントを処理するために、次のイベント ハンドラを `Form1` に追加します。

VB

```
Sub EventHandler() Handles testInstance.Event1
    MsgBox("The event handler caught the event.")
End Sub
```

4. `Test` というサブルーチンを `Form1` クラスに追加して、実装クラスをテストします。

VB

```
Sub Test()
    '  Create an instance of the class.
    Dim T As New ImplementationClass
    ' Assign the class instance to the interface.
    ' Calls to the interface members are
    ' executed through the class instance.
    testInstance = T
    ' Set a property.
    testInstance.Prop1 = 9
    ' Read the property.
    MsgBox("Prop1 was set to " & testInstance.Prop1)
    '  Test the method and raise an event.
    testInstance.Method1(5)
End Sub
```

`Test` プロシージャは、`MyInterface` を実装するクラスのインスタンスを作成し、そのインスタンスを `testInstance` フィールドに代入し、プロパティを設定し、そのインターフェイスを経由してメソッドを実行します。

5. スタートアップ フォームの `Form1 Load` プロシージャから `Test` プロシージャを呼び出すコードを追加します。

VB

```
Private Sub Form1_Load(ByVal sender As System.Object, _
                       ByVal e As System.EventArgs) _
                       Handles MyBase.Load
    Test() ' Test the class.
End Sub
```

6. F5 キーを押して `Test` プロシージャを実行します。"Prop1 was set to 9" というメッセージが表示されます。[OK] をクリックすると、"The X parameter for Method1 is 5" というメッセージが表示されます。[OK] をクリックすると、"The event handler caught the event" というメッセージが表示されます。

参照

処理手順

方法 : インターフェイスを作成および実装する

関連項目

Implements ステートメント

Interface ステートメント (Visual Basic)

Event ステートメント

概念

インターフェイスを使用する状況

その他の技術情報

Visual Basic におけるインターフェイス

Visual Basic の継承

Visual Basic では継承がサポートされているため、派生クラスの基礎として機能するクラスを定義できます。派生クラスは、基本クラスのプロパティ、メソッド、およびイベントを継承します。また、派生クラスで、基本クラスのプロパティ、メソッド、およびイベントを拡張することもできます。さらに、継承したメソッドを新しい実装でオーバーライドすることもできます。既定では、Visual Basic を使用して作成したすべてのクラスを継承できます。

継承を使用すると、一度記述およびデバッグしたクラスのコードを、新しいクラスの基礎として何度でも再利用できます。また、継承ベースのポリモーフィズムを利用できるようになります。ポリモーフィズムが実現されていると、機能的に異なる同じ名前のメソッドやプロパティを持つ複数のクラスを定義し、実行時にクライアント コードで使用できます。

このセクションの内容

継承の基本

継承の修飾子、メソッドとプロパティのオーバーライド、**MyClass**、および **MyBase** について説明します。

方法 : 派生クラスを作成する

派生クラスを作成する手順を説明します。

継承を使用する状況

インターフェイスではなく継承を使用する場合について説明します。

継承と .NET Framework

.NET Framework がどのように継承をサポートしているかを説明します。

チュートリアル : COM オブジェクトによる継承の実装

既存の COM オブジェクトを新しいオブジェクトの基礎として使用する方法について説明します。

継承と基本オブジェクト クラス

他のすべてのクラスの基礎として機能するクラスの概要について説明します。

クラスの階層構造における New メソッドと Finalize メソッドの動作

クラスの階層構造でコンストラクタやデストラクタが呼び出されるしくみについて説明します。

ポリモーフィズム

ポリモーフィズムの概要と使用方法について説明します。

継承階層のデザイン

クラスの階層構造のデザインと実装について説明します。

関連するセクション

Visual Basic におけるインターフェイス

インターフェイスの概要と使用方法について説明します。

Visual Basic におけるオブジェクト指向プログラミング

Visual Basic におけるオブジェクト指向プログラミングの概念を紹介します。

継承の基本

Inherits ステートメントは、既存のクラス (基本クラス) に基づいて新しいクラス (派生クラス) を宣言するときに使用します。派生クラスは、基本クラスで定義されているプロパティ、メソッド、イベント、フィールド、および定数を継承します。また、派生クラスでこれらのプロパティ、メソッド、イベント、フィールド、および定数を拡張することもできます。ここでは、継承の規則の一部と、クラスの継承方法を変更するための修飾子について説明します。

- 既定では、**NotInheritable** キーワードが指定されていない限り、すべてのクラスを継承できます。継承するクラスは、プロジェクト内の他のクラスでも、プロジェクトが参照している他のアセンブリのクラスでもかまいません。
- 多重継承をサポートしている言語もありますが、Visual Basic がサポートしているのは単一継承だけです。したがって、派生クラスの基本クラスは常に 1 つだけです。クラスでは多重継承することはできませんが、複数のインターフェイスを実装することはできます。複数のインターフェイスを実装すると、多重継承と同様の結果が得られます。
- 基本クラスで制限されている項目が公開されるのを防ぐため、派生クラスのアクセス レベルは、基本クラスと同等またはそれより制限の厳しいレベルにする必要があります。たとえば、**Public** クラスは **Friend** クラスまたは **Private** クラスを継承できません。また、**Friend** クラスは **Private** クラスを継承できません。

継承の修飾子

Visual Basic では、継承をサポートするために、次に示すクラス レベルのステートメントと修飾子が導入されています。

- **Inherits** ステートメント — 基本クラスを指定します。
- **NotInheritable** 修飾子 — クラスを基本クラスとして使用できないように指定します。
- **MustInherit** 修飾子 — クラスを基本クラスとしてだけ使用するように指定します。**MustInherit** クラスのインスタンスは直接作成できません。派生クラスの基本クラス インスタンスとして作成する必要があります。このようなクラスは、C++ や C# などの他のプログラミング言語での抽象クラスに相当します。

派生クラスにおけるプロパティとメソッドのオーバーライド

既定では、派生クラスは基本クラスのプロパティとメソッドを継承します。継承したプロパティやメソッドの動作を派生クラスで変更する必要がある場合は、継承したプロパティやメソッドをオーバーライドします。つまり、派生クラスでメソッドの新しい実装を定義できます。プロパティやメソッドのオーバーライド方法を制御するには、次の修飾子を使用します。

- **Overridable** — 派生クラスにおけるプロパティやメソッドのオーバーライドを許可します。
- **Overrides** — 基本クラスで定義されている **Overridable** のプロパティやメソッドをオーバーライドします。
- **NotOverridable** — 派生クラスにおけるプロパティやメソッドのオーバーライドを禁止します。既定では、**Public** メソッドは **NotOverridable** です。
- **MustOverride** — 派生クラスにおけるプロパティやメソッドのオーバーライドを必須にします。**MustOverride** キーワードを使用する場合、メソッドの定義で使用できるステートメントは、**Sub**、**Function**、および **Property** の各ステートメントだけです。その他のステートメントは使用できません。**End Sub** ステートメントと **End Function** ステートメントも使用しません。**MustInherit** クラスで **MustOverride** メソッドを宣言する必要があります。

メソッドのオーバーライドの詳細については、「プロパティとメソッドのオーバーライド」を参照してください。

MyBase キーワード

派生クラスでメソッドをオーバーライドしている場合に基本クラスのメソッドを呼び出すには、**MyBase** キーワードを使用します。たとえば、基本クラスから継承したメソッドをオーバーライドする派生クラスをデザインするとします。この場合は、次の例で示すように、オーバーライドしたメソッドで基本クラスのメソッドを呼び出し、戻り値を変更できます。

VB

```
Class DerivedClass
    Inherits BaseClass
    Public Overrides Function CalculateShipping( _
        ByVal Dist As Double, _
        ByVal Rate As Double) _
        As Double
        ' Call the method in the base class and modify the return value.
        Return MyBase.CalculateShipping(Dist, Rate) * 2
```

```
    End Function
End Class
```

MyBase を使用するときには、次の制約があります。

- **MyBase** が参照するのは、直接の基本クラスとそこで継承されているメンバです。クラスの **Private** メンバへのアクセスには使用できません。
- **MyBase** はキーワードであり、実際のオブジェクトではありません。**MyBase** は変数に代入したり、プロシージャに渡したりできません。また、**Is** 比較で使用できません。
- **MyBase** で修飾するメソッドは、必ずしも直接の基本クラスで定義される必要はありません。間接的に継承されている基本クラスで定義することもできます。**MyBase** で修飾した参照が正しくコンパイルされるためには、呼び出しで使用されているパラメータの名前と型に対応するメソッドが、いずれかの基本クラスに含まれている必要があります。
- **MyBase** を使って基本クラスの **MustOverride** メソッドを呼び出すことはできません。
- **MyBase** を **MyBase** で修飾することはできません。したがって、次のコードは無効です。

  ```
  MyBase.MyBase.BtnOK_Click()
  ```

- **MyBase** をモジュールで使用することはできません。
- 基本クラスが異なるアセンブリ内にある場合は、**Friend** と指定されている基本クラスのメンバに **MyBase** を使ってアクセスすることはできません。

MyClass キーワード

クラスに実装されている **Overridable** メソッドを呼び出すときに、派生クラスでオーバーライドされたメソッドではなく、このクラスのメソッドの実装が呼び出されるようにするには、**MyClass** キーワードを使用します。

- **MyClass** はキーワードであり、実際のオブジェクトではありません。**MyClass** は変数に代入したり、プロシージャに渡したりできません。また、**Is** 比較で使用できません。
- **MyClass** が参照するのは、外側のクラスとそこで継承されているメンバです。
- **MyClass** は、**Shared** メンバの修飾子として使用できます。
- **MyClass** を標準モジュールで使用することはできません。
- 基本クラスで定義されていて、クラスで実装が提供されていないメソッドを、**MyClass** で修飾することもできます。このような参照は、**MyBase.***Method* と同じ意味になります。

参照

処理手順

方法 : 派生クラスを作成する

関連項目

Inherits ステートメント

概念

プロパティとメソッドのオーバーライド

その他の技術情報

言語の変更点 (Visual Basic 6.0 ユーザー向け)

方法：派生クラスを作成する

Inherits ステートメントを使用すると、指定したクラスのプライベート メンバ以外のすべてのメンバが、クラスに継承されます。

他のクラスを継承するには

- 基本クラスとして使用するクラスの名前を指定した **Inherits** ステートメントを、派生クラスの最初のステートメントとして追加します。**Inherits** ステートメントは、クラス ステートメントの後の最初の非コメント ステートメントであることが必要です。

使用例

次の例では、2 つのクラスを定義しています。最初のクラスは、2 つのメソッドを持つ基本クラスです。2 番目のクラスは、基本クラスの 2 つのメソッドを継承し、2 つ目のメソッドをオーバーライドして、`Field` という名前のフィールドを定義しています。

VB

```
Class Class1
    Sub Method1()
        MsgBox("This is a method in the base class.")
    End Sub
    Overridable Sub Method2()
        MsgBox("This is another method in the base class.")
    End Sub
End Class

Class Class2
    Inherits Class1
    Public Field2 As Integer
    Overrides Sub Method2()
        MsgBox("This is a method in a derived class.")
    End Sub
End Class

Protected Sub TestInheritance()
    Dim C1 As New Class1
    Dim C2 As New Class2
    C1.Method1() ' Calls a method in the base class.
    C1.Method2() ' Calls another method from the base class.
    C2.Method1() ' Calls an inherited method from the base class.
    C2.Method2() ' Calls a method from the derived class.
End Sub
```

プロシージャ `TestInheritance` を実行すると、次のメッセージが表示されます。

```
This is a method in the base class.

This is another method in the base class.

This is a method in the base class.

This is a method in a derived class.
```

参照

概念

プロパティとメソッドのオーバーライド

オーバーライド修飾子

その他の技術情報

Visual Basic の継承

クラス、プロパティ、フィールド、およびメソッド

継承を使用する状況

継承というプログラミング概念は、便利である反面、使い方を間違いやすくもあります。継承よりインターフェイスを使用した方がよい場合もよくあります。このトピックと「インターフェイスを使用する状況」では、継承とインターフェイスを場合によってどのように使い分けるかを説明します。

次の場合には、継承の方が適しています。

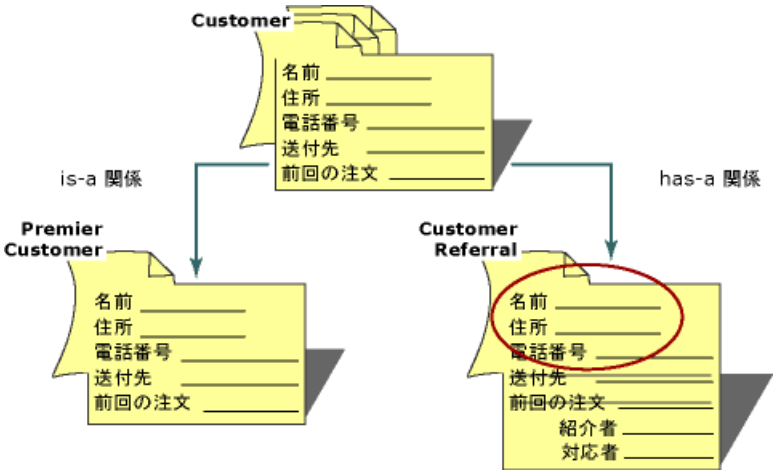
- 継承の階層が has-a 関係ではなく is-a 関係を表している場合。
- 基本クラスのコードを再利用できる場合。
- 同じクラスやメソッドを別のデータ型に適用する必要がある場合。
- クラスの階層構造が浅く、他の開発者によって多くのレベルが追加される可能性も低い場合。
- 基本クラスの変更をすべての派生クラスに反映させる場合。

次に、上の各項目について順番に詳しく説明します。

継承と is-a 関係

オブジェクト指向プログラミングにおけるクラスの関係は、is-a と has-a の 2 つの関係で表されます。is-a 関係では、派生クラスは確実に基本クラスの一種です。たとえば、`PremierCustomer` というクラスは `Customer` という基本クラスの関係は is-a 関係です。これは、重要顧客 (premier customer) は顧客 (customer) の一種であるためです。一方、`CustomerReferral` というクラスと `Customer` クラスの関係は has-a 関係です。これは、顧客紹介 (customer referral) には顧客 (customer) という語が含まれてはいますが、顧客の一種ではないためです。

継承階層のオブジェクトは、基本クラスで定義されているフィールド、プロパティ、メソッド、およびイベントを継承するため、基本クラスとの間に必ず is-a 関係を持ちます。他のクラスとの間に has-a 関係を持つクラスは、不適切なプロパティやメソッドを継承してしまう可能性があるため、継承階層には向きません。たとえば、`CustomerReferral` クラスを上述の `Customer` クラスから派生させたとすると、`ShippingPrefs` や `LastOrderPlaced` などの無用なプロパティも継承されます。このような has-a 関係を表すには、関係のないクラスやインターフェイスを使用します。次の図は、is-a 関係と has-a 関係の例です。



基本クラスとコードの再利用

継承には、コードを再利用できるという利点もあります。適切にデザインされたクラスは、一度デバッグするだけで、新しいクラスの基本クラスとして何度でも再利用できます。

コードの効果的な再利用の例は、データ構造体を管理するライブラリによく見られます。たとえば、数種類のインメモリリストを管理する大規模なビジネス アプリケーションがあったとします。リストの 1 つは顧客データベースのインメモリ コピーであり、処理を高速化するためにセッションの開始時にデータベースから読み込まれます。このデータ構造体は、たとえば次のようになります。

VB

```
Class CustomerInfo
    Protected PreviousCustomer As CustomerInfo
    Protected NextCustomer As CustomerInfo
    Public ID As Integer
    Public FullName As String

    Public Sub InsertCustomer(ByVal FullName As String)
        ' Insert code to add a CustomerInfo item to the list.
```

```
    End Sub

    Public Sub DeleteCustomer()
        ' Insert code to remove a CustomerInfo item from the list.
    End Sub

    Public Function GetNextCustomer() As CustomerInfo
        ' Insert code to get the next CustomerInfo item from the list.
        Return NextCustomer
    End Function

    Public Function GetPrevCustomer() As CustomerInfo
        'Insert code to get the previous CustomerInfo item from the list.
        Return PreviousCustomer
    End Function
End Class
```

このアプリケーションには、次の例に示すように、ユーザーが買い物カゴに追加した製品のリストもあります。

VB

```
Class ShoppingCartItem
    Protected PreviousItem As ShoppingCartItem
    Protected NextItem As ShoppingCartItem
    Public ProductCode As Integer
    Public Function GetNextItem() As ShoppingCartItem
        ' Insert code to get the next ShoppingCartItem from the list.
        Return NextItem
    End Function
End Class
```

上の 2 つのリストには同じパターンがあります。この 2 つのリストは、対象となるデータ型が違うだけで、動作は同じ (挿入、削除、および取得) です。本質的に同じ機能を持つ 2 つのコード ベースを管理することは、非効率です。この場合は、次に示すように、リストの管理専用のクラスを作成し、データ型ごとにそのクラスを継承するのが最も効率的なソリューションです。

VB

```
Class ListItem
    Protected PreviousItem As ListItem
    Protected NextItem As ListItem
    Public Function GetNextItem() As ListItem
        ' Insert code to get the next item in the list.
        Return NextItem
    End Function
    Public Sub InsertNextItem()
        ' Insert code to add a item to the list.
    End Sub

    Public Sub DeleteNextItem()
        ' Insert code to remove a item from the list.
    End Sub

    Public Function GetPrevItem() As ListItem
        'Insert code to get the previous item from the list.
        Return PreviousItem
    End Function
End Class
```

`ListItem` クラスのデバッグは一度で済みます。`ListItem` クラスを使用して他のクラスを作成するときには、リストの管理について考える必要はありません。たとえば、次のようになります。

VB

```
Class CustomerInfo
    Inherits ListItem
```

```
    Public ID As Integer
    Public FullName As String
End Class
Class ShoppingCartItem
    Inherits ListItem
    Public ProductCode As Integer
End Class
```

継承ベースのコードの再利用は強力なツールとなる反面、リスクも付随します。どれほど優れたデザインに基づくシステムであっても、予想外の変更が起こることはあります。既存のクラスの階層構造を変更すると、予想外の結果が生じることがあります。このような例については、「配置後の基本クラス デザインの変更」の「壊れやすい基本クラスの問題」を参照してください。

交換できる派生クラス

クラスの階層構造内の派生クラスを基本クラスと交換して使用できる場合もあります。このプロセスは、継承ベースのポリモーフィズムと呼ばれます。継承ベースのポリモーフィズムでは、インターフェイス ベースのポリモーフィズムの利点に、基本クラスのコードの再利用やオーバーライドという継承の利点が組み合わされています。

継承ベースのポリモーフィズムは、たとえば、描画パッケージで有効に利用できます。次のコードでは、継承は使用されていません。

VB

```
Sub Draw(ByVal Shape As DrawingShape, ByVal X As Integer, _
    ByVal Y As Integer, ByVal Size As Integer)

    Select Case Shape.type
        Case shpCircle
            ' Insert circle drawing code here.
        Case shpLine
            ' Insert line drawing code here.
    End Select
End Sub
```

この方法には、多少問題があります。たとえば、他のユーザーが後で楕円のオプションを追加するにはソース コードを変更する必要がありますが、そのユーザーがソース コードにアクセスすることさえできない場合も考えられます。また、これほどはっきりした問題ではありませんが、楕円を描画するには、(楕円には大小 2 つの直径があるため) 線のケースとは関係のない別のパラメータが必要になります。さらに折れ線 (接続された複数の線) を追加する場合は、その他のケースには関係のない別のパラメータがまた追加されます。

継承を使用すると、これらの問題をほとんど解決できます。適切にデザインされた基本クラスでは、特定のメソッドの実装は派生クラスに委ねられているため、どのような図形にも対応できます。他の開発者は、基本クラスのドキュメントを使用して、派生クラスでメソッドを実装できます。クラスのその他の項目 (x 座標や y 座標など) は、すべての派生クラスが使用するため、基本クラスに組み込むことができます。たとえば、次に示すように、`Draw` を **MustOverride** メソッドにします。

VB

```
MustInherit Class Shape
    Public X As Integer
    Public Y As Integer
    MustOverride Sub Draw()
End Class
```

次に、それぞれの図形に応じた要素をクラスに追加します。たとえば、`Line` クラスに必要なのは `Length` フィールドだけです。

VB

```
Class Line
    Inherits Shape
    Public Length As Integer
    Overrides Sub Draw()
        ' Insert code here to implement Draw for this shape.
    End Sub
End Class
```

この方法を使用すると、ソース コードにアクセスできない他の開発者も、必要に応じて、新しい派生クラスを使用して基本クラスを拡張できます。たとえば、`Line` クラスから `Rectangle` というクラスを派生させることができます。

VB

```
Class Rectangle
    Inherits Line
    Public Width As Integer
    Overrides Sub Draw()
        ' Insert code here to implement Draw for the Rectangle shape.
    End Sub
End Class
```

この例は、レベルごとに実装の詳細を追加することによって、汎用のクラスから特殊なクラスを作成する方法を示しています。

この段階で、派生クラスが本当に is-a 関係を表しているか、誤って has-a 関係を使用していないかを再度確認することをお勧めします。たとえば、新しい四角形のクラスが線だけで構成されている場合は、継承が最適な方法とは言えません。しかし、新しい四角形が幅のプロパティを持つ線である場合は、is-a 関係が維持されます。

クラスの浅い階層構造

継承は、クラスの階層構造が比較的浅い場合に最も適しています。クラスの階層構造が極端に深く複雑になると、開発が難しくなります。クラスの階層構造を使用するかどうかを決定するときには、クラスの階層構造を使用するメリットとそれに伴う複雑さとを考慮する必要があります。階層数の上限は、6 レベルとするのが一般的です。ただし、クラスの階層数の上限は、各レベルの複雑さの程度など、多くの要因に左右されます。

基本クラスを通じたすべての派生クラスの変更

継承の最も強力な機能の 1 つは、基本クラスに対する変更を派生クラスに反映させることです。これをうまく利用すると、1 つのメソッドの実装を更新するだけで、数十、数百もの派生クラスで新しいコードを使用できます。ただし、他のユーザーがデザインした派生クラスで問題が発生する可能性があるため、この操作には注意が必要です。この操作を実行するときには、元の基本クラスを使用しているクラスと新しい基本クラスとの互換性を確認する必要があります。特に、基本クラスのメンバの名前や型を変更するのは避けてください。

たとえば、**Integer** 型のフィールドに郵便番号を格納する基本クラスをデザインしたとします。他の開発者は、この郵便番号フィールドを継承して派生クラスを作成しています。この郵便番号フィールドには 5 桁の数値を格納できますが、今度は郵便番号が拡張され、新たにハイフン (-) と 4 桁の数値を加えることになったとします。この場合、10 文字の文字列を格納できるように基本クラスのフィールドを変更してしまうと、他の開発者は、新しいサイズとデータ型を使用するために、派生クラスを変更および再コンパイルする必要があります。これは最悪のシナリオです。

基本クラスを変更するときには、新しいメンバを追加するのが最も安全です。たとえば、上の例に新しいフィールドを追加して、郵便番号の追加の 4 桁をそこに格納するようにします。これにより、新しいフィールドを使用するようにクライアント アプリケーションを更新できます。既存のアプリケーションが使用できなくなることはありません。基本クラスを拡張できる機能は、インターフェイスにはない継承階層の大きな利点です。

参照

概念

インターフェイスを使用する状況

配置後の基本クラス デザインの変更

継承と .NET Framework

Visual Studio アプリケーションでは、.NET Framework で見つかった多くのクラスを基本クラスとして使用できます。たとえば、属性クラスは、Attribute クラスに常に基づいています。他の便利な .NET Framework 基本クラスには、コンポーネントを定義するために継承された Component、およびカスタム例外クラスを定義するために継承された Exception が含まれています。

.NET Framework は、基本クラスのソースにとどまりません。異なるプログラミング言語で作成されたクラス間での互換性が保証されます。Visual Basic クラスは、Visual Studio にサポートされているどの言語で作成したアセンブリのクラスからでも継承できます。Visual Basic で作成されたクラスは、他の .NET Framework プログラミング言語でも、基本クラスとして使用できます。

参照

処理手順

チュートリアル : COM オブジェクトによる継承の実装

概念

継承と基本オブジェクト クラス

継承を使用する状況

チュートリアル：COM オブジェクトによる継承の実装

Visual Basic 2005 のクラスは、COM オブジェクトの **Public** クラス (Visual Basic の以前のバージョンで作成されたものも含む) から派生できます。COM オブジェクトから継承されたクラスのプロパティとメソッドは、その他の基本クラスのプロパティとメソッドをオーバーライドまたはオーバーロードするのと同じ方法で、オーバーライドまたはオーバーロードできます。COM オブジェクトからの継承は、再コンパイルしない既存のクラス ライブラリがあるときに利用すると便利です。

次の手順では、Visual Basic 6.0 を使用してクラスを含む COM オブジェクトを作成し、作成した COM オブジェクトを Visual Basic 2005 で基本クラスとして使用する方法を示します。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

このチュートリアルで使用する COM オブジェクトを作成するには

1. Visual Basic 6.0 で、新しい ActiveX DLL プロジェクトを開きます。`Project1` という名前のプロジェクトが、`Class1` という名前のクラスと共に作成されます。
2. プロジェクト エクスプローラで、[Project1] を右クリックし、[Project1 のプロパティ] をクリックします。[プロジェクト プロパティ] ダイアログ ボックスが表示されます。
3. [プロジェクト プロパティ] ダイアログ ボックスの [全般] タブの [プロジェクト名] フィールドに「`ComObject1`」と入力してプロジェクトの名前を変更します。
4. プロジェクト エクスプローラで [`Class1`] を右クリックし、[プロパティ] をクリックします。このクラスの [プロパティ] ウィンドウが表示されます。
5. **Name** プロパティを `MathFunctions` に変更します。
6. プロジェクト エクスプローラで [`MathFunctions`] を右クリックし、[コードの表示] をクリックします。コード エディタが表示されます。
7. プロパティの値を保持するローカル変数を追加します。

```
' Local variable to hold property value
Private mvarProp1 As Integer
```

8. Property `Let` および Property `Get` プロパティ プロシージャを追加します。

```
Public Property Let Prop1(ByVal vData As Integer)
    'Used when assigning a value to the property.
    mvarProp1 = vData
End Property
Public Property Get Prop1() As Integer
    'Used when retrieving a property's value.
    Prop1 = mvarProp1
End Property
```

9. 関数を追加します。

```
Function AddNumbers( _
    ByVal SomeNumber As Integer, _
    ByVal AnotherNumber As Integer) _
    As Integer

    AddNumbers = SomeNumber + AnotherNumber
End Function
```

10. [ファイル] メニューの [ComObject1.dll の作成] をクリックして、COM オブジェクトを作成および登録します。

> **メモ：**
>
> Visual Basic で作成したクラスも COM オブジェクトとして公開できますが、これは本当の COM オブジェクトではないので、このチュートリアルでは使用できません。詳細については、「.NET Framework アプリケーションにおける COM 相互運用性」を参照してください。

相互運用アセンブリ

次の手順では、相互運用機能アセンブリを作成します。相互運用機能アセンブリは、アンマネージ コード (COM オブジェクトなど) と、Visual Studio が使用するマネージ コードの仲介役として機能します。Visual Basic によって作成される相互運用機能アセンブリは、相互運用マーシャリングなど、COM オブジェクトの操作で発生する数多くの詳細な作業を処理します。相互運用マーシャリングとは、COM オブジェクトとの間の変換のときに、パラメータと戻り値を対応するデータ型にパッケージ化する処理です。Visual Basic アプリケーション内の参照は、実際の COM オブジェクトではなく、相互運用機能アセンブリを指します。

Visual Basic 2005 で COM オブジェクトを使用するには

1. 新しい Visual Basic Windows アプリケーション プロジェクトを開きます。
2. [プロジェクト] メニューの [参照の追加] をクリックします。

   [参照の追加] ダイアログ ボックスが表示されます。
3. [COM] タブの [コンポーネント名] ボックスの一覧で `ComObject1` をダブルクリックし、[OK] をクリックします。
4. [プロジェクト] メニューの [新しい項目の追加] をクリックします。

   [新しい項目の追加] ダイアログ ボックスが表示されます。
5. テンプレート ペインの [クラス] をクリックします。

   既定のファイル名 `Class1.vb` が [ファイル名] フィールドに表示されます。このフィールドを「MathClass.vb」に変更し、[追加] をクリックします。これで `MathClass` という名前のクラスが作成され、コードが表示されます。
6. `MathClass` の先頭に次のコードを追加して、COM クラスを継承します。

   VB

```
' The inherited class is called MathFunctions in the base class,
' but the interop assembly appends the word Class to the name.
Inherits ComObject1.MathFunctionsClass
```

7. 次のコードを `MathClass` に追加して、基本クラスのパブリック メソッドをオーバーロードします。

   VB

```
'  This method overloads the method AddNumbers from the base class.
Overloads Function AddNumbers( _
    ByVal SomeNumber As Integer, _
    ByVal AnotherNumber As Integer) _
    As Integer

    Return SomeNumber + AnotherNumber
End Function
```

8. 次のコードを `MathClass` に追加して、継承されるクラスを拡張します。

   VB

```
'  The following function extends the inherited class.
Function SubtractNumbers( _
    ByVal SomeNumber As Integer, _
    ByVal AnotherNumber As Integer) _
```

```
    As Integer

    Return AnotherNumber - SomeNumber
End Function
```

新しいクラスは、COM オブジェクト内の基本クラスのプロパティを継承し、メソッドをオーバーロードし、新しいメソッドを定義してクラスを拡張します。

継承したクラスをテストするには

1. スタートアップ フォームにボタンを追加し、そのボタンをダブルクリックしてコードを表示します。
2. ボタンの **Click** イベント ハンドラ プロシージャに、次のコードを追加します。このコードは、`MathClass` のインスタンスを作成し、オーバーロードされたメソッドを呼び出します。

   VB

```
Dim Result1 As Short
Dim Result2 As Integer
Dim Result3 As Integer
Dim MathObject As New MathClass
Result1 = MathObject.AddNumbers(4S, 2S) ' Add two Shorts.
Result2 = MathObject.AddNumbers(4, 2) 'Add two Integers.
Result3 = MathObject.SubtractNumbers(2, 4) ' Subtract 2 from 4.
MathObject.Prop1 = 6 ' Set an inherited property.

MsgBox("Calling the AddNumbers method in the base class " & _
       "using Short type numbers 4 and 2 = " & Result1)
MsgBox("Calling the overloaded AddNumbers method using " & _
       "Integer type numbers 4 and 2 = " & Result2)
MsgBox("Calling the SubtractNumbers method " & _
       "subtracting 2 from 4 = " & Result3)
MsgBox("The value of the inherited property is " & _
        MathObject.Prop1)
```

3. F5 キーを押してプロジェクトを実行します。

フォーム上のボタンをクリックすると、**Short** 型の数値を使用して `AddNumbers` メソッドが呼び出され、Visual Basic によって基本クラスの適切なメソッドが選択されます。`AddNumbers` の 2 回目の呼び出しは `MathClass` のオーバーロード メソッドに渡されます。3 回目の呼び出しでは `SubtractNumbers` メソッドが呼び出され、クラスが拡張されます。基本クラス内のプロパティが設定され、値が表示されます。

次の手順

オーバーロードされた `AddNumbers` 関数は、COM オブジェクトの基本クラスから継承されたメソッドと同じデータ型を持つように見えます。これは、基本クラスのメソッドの引数とパラメータが Visual Basic 6.0 では 16 ビットの整数として定義されているのに対し、Visual Basic 2005 では **Short** 型の 16 ビットの整数として公開されるためです。新しい関数は、32 ビットの整数を受け取り、基本クラスの関数をオーバーロードします。

COM オブジェクトを操作するときは、パラメータのサイズとデータ型を確認することが重要です。たとえば、Visual Basic 6.0 のコレクション オブジェクトを引数として受け取る COM オブジェクトを使用するときは、Visual Basic 2005 のコレクションを提供できません。データ型の変更の詳細については、「言語の変更点 (Visual Basic 6.0 ユーザー向け)」を参照してください。

COM クラスから継承されたプロパティとメソッドをオーバーライドできます。これは、COM の基本クラスから継承されたプロパティやメソッドを置き換えるローカルのプロパティやメソッドを宣言できることを意味します。継承された COM プロパティをオーバーライドする場合の規則は、その他のプロパティやメソッドをオーバーライドする場合の規則と同じですが、次の例外もあります。

- COM クラスから継承されたプロパティやメソッドをオーバーライドする場合は、その他の継承されたプロパティとメソッドをすべてオーバーライドする必要があります。
- **ByRef** パラメータを使用するプロパティをオーバーライドできません。

参照

処理手順
方法 : 派生クラスを作成する
関連項目
Inherits ステートメント
Short 型 (Visual Basic)
概念
プロパティとメソッドのオーバーライド
その他の技術情報
.NET Framework アプリケーションにおける COM 相互運用性
言語の変更点 (Visual Basic 6.0 ユーザー向け)

継承と基本オブジェクト クラス

オブジェクトは、Visual Basic のほとんどの作業で使用されます。中には、文字列や整数のように、通常はオブジェクトと見なされない要素もあります。Visual Basic のオブジェクトはすべて、System.Object という基本クラスから派生し、この基本クラスのメソッドを継承しています。**System.Object** クラスには、.NET Framework のすべてのオブジェクトに共通の基本クラスが用意されており、Visual Studio を使用して開発するオブジェクトの相互運用性を保証しています。新しいクラスは暗黙的に **System.Object** クラスを継承するため、**Inherits** ステートメントでこのクラスの名前を明示的に指定する必要はありません。

オブジェクトが **System.Object** から継承するメソッドの中で最も便利なメソッドの 1 つが GetType です。このメソッドを使用すると、現在のオブジェクトの厳密な型を返すことができます。

参照

関連項目

Object
GetType

概念

プロパティとメソッドのオーバーライド
継承の基本

クラスの階層構造における New メソッドと Finalize メソッドの動作

クラスのインスタンスが作成されると、そのオブジェクト内に存在する場合、共通言語ランタイム (CLR: Common Language Runtime) は **New** という名前のプロシージャを実行しようとします。**New** は、*constructor* と呼ばれるプロシージャの一種です。これは、オブジェクト内の他のコードを実行する前に、新しいオブジェクトを初期化するために使用されます。**New** コンストラクタを使用すると、ファイルを開いたり、データベースに接続したり、変数を初期化するなど、オブジェクトを使用できるようにするために事前に実行する必要があるタスクを処理できます。

派生クラスのインスタンスが作成されるときには、まず基本クラスの **Sub New** コンストラクタが実行され、その後に派生クラスのコンストラクタが実行されます。これは、**Sub New** コンストラクタのコードの最初の行が、`MyBase.New()` 構文を使用して、クラスの階層構造でこのクラスのすぐ上のクラスのコンストラクタを呼び出すためです。基本クラスのコンストラクタにたどり着くまで、クラスの階層構造の各クラスに対して **Sub New** コンストラクタが呼び出されます。基本クラスにたどり着くと、基本クラスのコンストラクタのコードが実行され、続いてすべての派生クラスの各コンストラクタのコードが実行されます。最後に実行されるのは、最派生クラスのコードです。

Sub New コンストラクタ

基本クラス

```
Sub New()
    MyBase.New()'Call constructor for base class last.
    'Add code here to perform initialization tasks in the base class.
End Sub 'New
```

派生クラス

```
Sub New()

    MyBase.New()'Call constructor for derived class next.
    'Add code here to perform initialization tasks in the derived class.

End Sub 'New
```

最派生クラス

```
Sub New()
    MyBase.New()'Call constructor for most derived class first.
    'Add code here to perform initialization tasks in the most derived class.
End Sub 'New
```

オブジェクトが不要になると、CLR は、メモリを解放する前にそのオブジェクトの Finalize メソッドを呼び出します。**Finalize** メソッドは、状態情報を保存したり、ファイルとデータベースへの接続を閉じるなど、オブジェクトを解放する前に実行する必要があるクリーンアップ タスクを処理するため、*destructor* と呼ばれます。

Finalize デストラクタ

基本クラス

```
Sub Finalize()
    'Add code here to perform cleanup tasks
    'such as closing files and saving state information.
    MyBase.Finalize()'Call destructor for base class last.
End Sub
```

派生クラス

```
Sub Finalize()
    'Add code here to perform cleanup tasks in the derived class.
    MyBase.Finalize()'Call destructor for base class last.
End Sub
```

最派生クラス

```
Sub Finalize()
    Close DBConnections()'Call a procedure to close database connections.
    SaveState()'Call a procedure to save state information.
    MyBase.Finalize()'Call destructor for base class last.
End Sub
```

参照

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄

ポリモーフィズム

ポリモーフィズムが実現されていると、機能的に異なる同じ名前のメソッドやプロパティを持つ複数のクラスを定義し、実行時にクライアント コードで交換して使用できます。

このセクションの内容

Visual Basic のポリモーフィズムのしくみ

ポリモーフィズムを実現する 2 つの方法を紹介します。

継承ベースのポリモーフィズム

継承を使用してポリモーフィズムを実現する方法について説明します。

インターフェイス ベースのポリモーフィズム

インターフェイスを使用してポリモーフィズムを実現する方法について説明します。

関連するセクション

インターフェイスを使用する状況

継承階層よりインターフェイスを使用した方がよい場合について説明します。

継承を使用する状況

インターフェイスより継承を使用した方がよい場合について説明します。

Visual Basic のポリモーフィズムのしくみ

従来の Visual Basic のポリモーフィズムは、インターフェイスによって実現されていました。インターフェイスによるポリモーフィズムは、現在でも利用できます。しかし、Visual Basic では、継承を使用してポリモーフィズムを実現することもできます。

どちらの方法を使用するかは、オブジェクト指向プログラミングのその他の問題と同様に、個別の要件によって決まります。一般的には、派生クラスで拡張できる基本的な機能を作成する場合には継承を使用し、共通要素がほとんどない複数の実装で類似の機能を提供する必要がある場合にはインターフェイスを使用します。

参照

概念

継承ベースのポリモーフィズム

インターフェイス ベースのポリモーフィズム

その他の技術情報

継承階層のデザイン

継承ベースのポリモーフィズム

ほとんどのオブジェクト指向プログラミング システムは、継承を通じてポリモーフィズムを提供しています。継承ベースのポリモーフィズムでは、基本クラスでメソッドを定義し、派生クラスで新しい実装を使用してメソッドをオーバーライドします。

たとえば、消費税を計算するための基本機能を提供する `BaseTax` というクラスを定義したとします。`CountyTax` や `CityTax` などの `BaseTax` の派生クラスでは、必要に応じて `CalculateTax` などのメソッドを実装できます。

この場合は、オブジェクトが属しているクラスがわからなくても、`BaseTax` の任意の派生クラスに属するオブジェクトの `CalculateTax` メソッドを呼び出すことができるため、ポリモーフィズムが実現されています。

次に示す `TestPoly` プロシージャは、継承ベースのポリモーフィズムの例です。

VB

```
' %5.3 State tax
Const StateRate As Double = 0.053
' %2.8 City tax
Const CityRate As Double = 0.028
Public Class BaseTax
    Overridable Function CalculateTax(ByVal Amount As Double) As Double
        ' Calculate state tax.
        Return Amount * StateRate
    End Function
End Class

Public Class CityTax
    ' This method calls a method in the base class
    ' and modifies the returned value.
    Inherits BaseTax
    Private BaseAmount As Double
    Overrides Function CalculateTax(ByVal Amount As Double) As Double
        ' Some cities apply a tax to the total cost of purchases,
        ' including other taxes.
        BaseAmount = MyBase.CalculateTax(Amount)
        Return CityRate * (BaseAmount + Amount) + BaseAmount
    End Function
End Class

Sub TestPoly()
    Dim Item1 As New BaseTax
    Dim Item2 As New CityTax
    ' $22.74 normal purchase.
    ShowTax(Item1, 22.74)
    ' $22.74 city purchase.
    ShowTax(Item2, 22.74)
End Sub

Sub ShowTax(ByVal Item As BaseTax, ByVal SaleAmount As Double)
    ' Item is declared as BaseTax, but you can
    ' pass an item of type CityTax instead.
    Dim TaxAmount As Double
    TaxAmount = Item.CalculateTax(SaleAmount)
    MsgBox("The tax is: " & Format(TaxAmount, "C"))
End Sub
```

この例では、`ShowTax` プロシージャは `BaseTax` 型の `Item` というパラメータを受け取っていますが、`BaseTax` から派生した任意のクラス (`CityTax` など) を渡すこともできます。このデザインの利点は、`ShowTax` プロシージャのクライアント コードを変更することなく、`BaseTax` クラスから派生した新しいクラスを追加できるという点にあります。

参照

概念

インターフェイス ベースのポリモーフィズム

その他の技術情報

継承階層のデザイン

インターフェイス ベースのポリモーフィズム

Visual Basic では、インターフェイスを使用してポリモーフィズムを実現することもできます。インターフェイスには、クラスのようにプロパティやメソッドが記述されています。しかし、クラスとは異なり、実装を提供することはできません。複数のインターフェイスを使用すると、既存のコードを壊すことなく、ソフトウェア コンポーネントのシステムを発展させていくことができます。

インターフェイスを使用してポリモーフィズムを実現するには、複数のクラスにそれぞれ違った方法でインターフェイスを実装します。クライアント アプリケーションは、古い実装と新しい実装のどちらでも、まったく同じ方法で使用できます。インターフェイス ベースのポリモーフィズムの利点は、既存のクライアント アプリケーションを新しいインターフェイスの実装で動作させるために、再コンパイルする必要がないことです。

次の例では、`Shape2` というインターフェイスを定義しています。`Shape2` は、`RightTriangleClass2` と `RectangleClass2` というクラスに実装されています。`ProcessShape2` という名前のプロシージャによって、`RightTriangleClass2` または `RectangleClass2` のインスタンスの `CalculateArea` メソッドが呼び出されます。

VB

```
Sub TestInterface()
    Dim RectangleObject2 As New RectangleClass2
    Dim RightTriangleObject2 As New RightTriangleClass2
    ProcessShape2(RightTriangleObject2, 3, 14)
    ProcessShape2(RectangleObject2, 3, 5)
End Sub

Sub ProcessShape2(ByVal Shape2 As Shape2, ByVal X As Double, _
      ByVal Y As Double)
    MsgBox("The area of the object is " _
       & Shape2.CalculateArea(X, Y))
End Sub

Public Interface Shape2
    Function CalculateArea(ByVal X As Double, ByVal Y As Double) As Double
End Interface

Public Class RightTriangleClass2
    Implements Shape2
    Function CalculateArea(ByVal X As Double, _
          ByVal Y As Double) As Double Implements Shape2.CalculateArea
        ' Calculate the area of a right triangle.
        Return 0.5 * (X * Y)
    End Function
End Class

Public Class RectangleClass2
    Implements Shape2
    Function CalculateArea(ByVal X As Double, _
          ByVal Y As Double) As Double Implements Shape2.CalculateArea
        ' Calculate the area of a rectangle.
        Return X * Y
    End Function
End Class
```

参照

処理手順

方法 : インターフェイスを作成および実装する

概念

継承ベースのポリモーフィズム

継承階層のデザイン

継承階層は、デザインするより実装する方が簡単なため、要件を明確にしないままコーディングを始めてしまうのは危険です。クラスの階層構造のデザイン上の誤りを実装後に修正するには、既存のアプリケーションが無効になるようなコード変更が必要になる場合もあります。このセクションでは、継承階層をデザインする際の注意点について説明し、このような失敗を回避するための情報を提供します。

このセクションの内容

拡張性に関するクラスの階層構造の考慮事項

他の開発者による更新や拡張が可能なクラスの階層構造をデザインする方法について説明します。

メソッドに対するアクセス レベルの選択に関する考慮事項

クラス内のアクセシビリティ レベルの正しい使い方について説明します。

配置後の基本クラス デザインの変更

クラスの階層構造を変更する場合の問題について説明します。

関連するセクション

インターフェイスを使用する状況

継承階層よりインターフェイスを使用した方がよい場合について説明します。

Visual Basic におけるインターフェイス

インターフェイスをデザインおよび実装する方法について説明します。

拡張性に関するクラスの階層構造の考慮事項

クラスの階層構造をどれほど綿密にデザインしても、時が経つにつれて発展させていく必要があります。クラスの階層構造をデザインする際の初期段階での選択によって、後の作業が簡単になります。

クラスの階層構造を拡張する

クラスの階層構造を簡単に拡張できるようにするためのヒントを次に示します。

- クラスの階層構造は、一般的なクラスから特殊なクラスへという順に定義します。継承階層の各レベルでクラスを定義するときには、できるだけ一般的になるようにします。派生クラスは、基本クラスのメソッドを継承し、継承したメソッドを再利用したり拡張したりできます。

  たとえば、コンピュータのハードウェアをモデルにしてクラスの階層構造をデザインするとします。出力デバイスのモデリングを開始する段階で、まず、`Display`、`Printer`、および `File` の各クラスを定義します。次に、基本クラスで定義されているメソッドを実装するクラスを定義します。たとえば、`Display` からは、`LCDDisplay` というクラスを派生させ、`EnterPowerSaveMode` という名前のメソッドを実装できます。

- データ型やストレージを定義するときには、余裕を見て定義しておくと、後の変更が簡単になります。たとえば、現在のデータでは標準の **Integer** 変数で十分な場合でも、**Long** 型の変数を使用します。

- 派生クラスで必要な項目だけを公開します。**Private** なフィールドとメソッドでは、名前の競合が起こりにくく、後で変更する可能性のある項目が他のユーザーによって使用されるのを防ぐことができます。

- 派生クラスだけが必要とするメンバは、**Protected** に設定します。これによって、派生クラスだけがこれらのメンバに依存することとなり、開発中にこれらのメンバを更新しやすくなります。

- 基本クラスのメソッドが **Overridable** メンバに依存しないようにします。それらのメンバの機能は、継承するクラスによって変更される場合があります。

参照

関連項目

MustInherit
MustOverride

概念

メソッドに対するアクセス レベルの選択に関する考慮事項
配置後の基本クラス デザインの変更

メソッドに対するアクセス レベルの選択に関する考慮事項

クラスの階層構造のメンバに対して最適なアクセス レベルを適用すると、メンバの使用を制御できるため、階層構造が管理しやすくなります。

原則として、クラス メンバを宣言するときには、できるだけ制限の厳しいアクセス修飾子を使用します。クラス メンバへのアクセスを制限すると、名前の競合が起こりにくくなり、メソッドが予想外の方法で使用されるのを防ぐことができます。

クラス内部のメンバは、**Private** として宣言します。Private として宣言したメンバには、定義されているクラスの中からしかアクセスできません。

そのクラスと派生クラスでしか使用されないメソッドには、**Protected** アクセス修飾子を使用します。**Protected** メンバには、そのメンバが宣言されたクラス、またはその派生クラスからしかアクセスできません。

Friend データ メンバには、クラスの外部からもアクセスできます。ただし、クラスが定義されているプロジェクトに含まれているモジュールを使用しないとアクセスできません。

Public データ メンバには、だれでもアクセスできます。**Public** データ メンバは、多くの場合、クラスの階層構造の一番下で使用されます。

参照

概念

拡張性に関するクラスの階層構造の考慮事項
配置後の基本クラス デザインの変更

配置後の基本クラス デザインの変更

クラスの階層構造は、些細な変更でも予想外の結果を引き起こす可能性があるため、配置後には変更しないのが理想的です。しかし、現実には変更せざるを得ない場合もあります。たとえば、製品の要件が変更されたり、デザインの仕様に重要な要素が漏れていたりする場合などが考えられます。継承には、壊れやすい基本クラスの問題などの問題があります。

壊れやすい基本クラスの問題

継承階層を使用する場合の最大の欠点となるのが壊れやすい基本クラスという問題です。基本クラスを変更するときには、基本クラス、および派生クラスのすべてのコードの変更、再コンパイル、および再配布が必要になることがよくあります。基本クラスと派生クラスが複数の開発者によって作成されている場合は、この問題がさらに複雑になります。これは、それぞれの開発者が自分のソース コードへのアクセスを拒否する可能性があるためです。最悪の場合には、コンパイル済みバイナリ形式の更新された基本クラスを、それとは互換性がない元のバイナリ バージョンの派生クラスと一緒に使用するユーザーが出てきてしまいます。

たとえば、基本クラスのメンバのデータ型をオーバーライドしたり変更したりすると、派生クラスが使用できなくなる可能性があります。

壊れやすい基本クラスの問題への対策

壊れやすい基本クラスの問題を回避するための最善の方法は、派生クラスだけに変更を加えることです。しかし、基本クラスを最初にリリースする時点であらゆる可能性を考慮に入れる必要があるため、必ずしも可能ではありません。最善の努力を尽くしても、予想外の事態が発生して基本クラスの変更を余儀なくされることはあります。

MustInherit クラスと **MustOverride** メソッドを使用すると、実装の詳細が派生クラスに移り、基本クラスを変更する必要が少なくなるため、壊れやすい基本クラスの問題を軽減できます。しかし、この方法の場合でも、どれほど綿密な計画を立てても必ずしも可能とは限りません。

この他、派生クラスでシャドウされたデータ メンバを使用するのも有効です。この場合は、基本クラスのメンバとの名前の競合が起こりにくくなります。

基本クラスの機能を拡張するには、新しいメンバを追加するのが最も安全です。新しいメンバによって基本クラスが壊れてしまう可能性があるのは、派生クラスで **Overloads** キーワードを使用して基本クラスのメソッドを継承し、同じ名前の新しいメソッドを導入している場合だけです。この問題を回避するには、継承するメソッドを再定義して代わりに処理させることにより、継承する基本クラスのメソッドを派生クラスで明示的に指定します。

ある意味では、すべての基本クラスはある程度壊れやすいものです。結局のところ、壊れやすい基本クラスの問題を解消することはできません。しかし、できるだけ基本クラスを変更しなくても済むようにクラスの階層構造を慎重にデザインし、基本クラスを変更せざるを得ないときには徹底したテストを行うことで、この問題を最小限にとどめることはできます。

参照

関連項目

Shadows

その他の技術情報

継承階層のデザイン

Visual Basic の共有メンバ

共有メンバは、1 つのクラスまたは構造体のすべてのインスタンスで共有されるプロパティ、プロシージャ、およびフィールドです。プログラミング言語によっては、静的メンバとも呼ばれます。

共有フィールドと共有プロパティ

インスタンスのいずれかに固有ではなく、クラスの一部となっている情報がある場合は、共有されたフィールドやプロパティが有効です。共有フィールドおよび共有プロパティの値を変更すると、クラスとそのクラスのすべてのインスタンスに関連付けられた値も変更されます。

一方、インスタンスのいずれかに関連付けられた非共有フィールドやプロパティの値を変更しても、同じクラスの他のインスタンスのフィールドやプロパティが影響を受けることはありません。非共有フィールドやプロパティは、クラスのインスタンスごとに独立して存在します。

このように、共有フィールドや共有プロパティは、クラスのインスタンスからだけ、またはクラス名で修飾することによってアクセスできる、グローバル変数のように動作します。共有フィールドや共有プロパティを使用せずに同じ機能を使用するには、モジュール レベルの変数を使用する必要があります。ただし、モジュール レベルの変数は、クラスの理解や保守を難しくする場合があります。また、モジュール レベルの変数をこのように使用すると、クラスが表現するカプセル化の概念に反することになります。

共有プロシージャ

共有プロシージャは、クラスの特定のインスタンスに関連付けられていないクラス メソッドです。たとえば、Math クラスで定義されている Cos メソッドは共有メソッドです。共有プロシージャは、オブジェクトのメソッドとして呼び出すか、クラスから直接呼び出すことができます。

共有プロシージャと共有プロパティは、そのクラスのインスタンスにはアクセスできません。このため、共有メソッドでは、非共有データ メンバには修飾参照だけが許可されています。

メモ：

クラスのインスタンス経由で共有メンバにアクセスするコードを作成するのはお勧めできません。これは、コンパイラが共有メンバの修飾を無視し、クラス経由で直接アクセスされているものとして扱うためです。一部のコードを実行するためにオブジェクトを修飾する場合も考えられるため、Visual Basic コンパイラは、クラスのインスタンス経由で共有メンバにアクセスしようとすると警告を表示します。IntelliSense では、クラスのインスタンスに対して共有メンバを表示しません。

共有メンバの例

次の例では、共有フィールド、2 つのインスタンス フィールド、および共有メソッドを作成して、コード中における共有メンバの動作を示します。

VB

```
Public Class Item
    Public Shared Count As Integer = 1
    Public Shared Sub ShareMethod()
        MsgBox("Current value of Count: " & Count)
    End Sub

    Public Sub New(ByVal Name As String)
        ' Use Count to initialize SerialNumber.
        Me.SerialNumber = Count
        Me.Name = Name
        ' Increment the shared variable
        Count += 1
    End Sub
    Public SerialNumber As Integer
    Public Name As String
    Public Sub InstanceMethod()
        MsgBox("Information in the first object: " & _
            Me.SerialNumber & vbTab & Me.Name)
    End Sub
End Class

Sub TestShared()
    ' Create two instances of the class.
    Dim part1 As New Item("keyboard")
    Dim part2 As New Item("monitor")
```

```
        part1.InstanceMethod()
        part2.InstanceMethod()
        Item.ShareMethod()
    End Sub
```

TestShared プロシージャを実行すると、クラスのインスタンスが 2 つ作成されます。コンストラクタは、共有フィールド Count を使用して、インスタンス フィールド SerialNumber を初期化し、Count をインクリメントします。このテクニックにより、各インスタンスに自動的に異なるシリアル番号が割り当てられます。

2 つのインスタンスを作成した後で、インスタンス メソッド InstanceMethod が両方のオブジェクトで呼び出され、共有メソッド ShareMethod も呼び出されます。出力は次のとおりです。

```
Information in the first object: 1 keyboard

Information in the second object: 2 monitor

Current value of the shared Count field: 3
```

参照

関連項目

Dim ステートメント (Visual Basic)

Cos

概念

構造体とクラス

その他の技術情報

クラス、プロパティ、フィールド、およびメソッド

構造体 : 独自のデータ型

事前バインディングと遅延バインディング

Visual Basic コンパイラは、オブジェクトがオブジェクト変数に代入されるときに *binding* と呼ばれる処理を実行します。オブジェクトが特定のオブジェクト型として宣言された変数に代入される場合、オブジェクトは事前バインディングされます。事前バインディングされたオブジェクトでは、アプリケーションが実行される前に、コンパイラによってメモリの割り当てとその他の最適化が実行されます。たとえば、次のコード例では FileStream 型の変数を宣言しています。

VB

```
'  Create a variable to hold a new object.
Dim FS As System.IO.FileStream
' Assign a new object to the variable.
FS = New System.IO.FileStream("C:\tmp.txt", _
    System.IO.FileMode.Open)
```

FileStreamは特定のオブジェクト型であるため、`FS` に代入されるインスタンスは事前バインディングされます。

対照的に、オブジェクトが **Object** 型として宣言された変数に代入される場合は、遅延バインディングされます。この型のオブジェクトは、任意のオブジェクトへの参照を保持できますが、事前バインディングされたオブジェクトの利点をほとんど持ちません。たとえば、次のコード例では、**CreateObject** 関数で返されたオブジェクトを保持するオブジェクト変数を宣言しています。

VB

```
' To use this example, you must have Microsoft Excel installed on your computer.
' Compile with Option Strict Off to allow late binding.
Sub TestLateBinding()
    Dim xlApp As Object
    Dim xlBook As Object
    Dim xlSheet As Object
    xlApp = CreateObject("Excel.Application")
    ' Late bind an instance of an Excel workbook.
    xlBook = xlApp.Workbooks.Add
    ' Late bind an instance of an Excel worksheet.
    xlSheet = xlBook.Worksheets(1)
    xlSheet.Activate()
    ' Show the application.
    xlSheet.Application.Visible = True
    ' Place some text in the second row of the sheet.
    xlSheet.Cells(2, 2) = "This is column B row 2"
End Sub
```

事前バインディングの利点

可能な場合は、事前バインディングされたオブジェクトを使用してください。これによって、コンパイラは、アプリケーションをより効率的にする重要な最適化を実行できます。事前バインディングされたオブジェクトは遅延バインディングされたオブジェクトよりも処理が高速です。また、使用されているオブジェクトの種類が明確になるため、コードがより読みやすくなり、保守も簡単になります。事前バインディングのその他の利点として、自動コード補完やダイナミック ヘルプなどの便利な機能が有効になります。これは、Visual Studio の統合開発環境 (IDE) により、コードの編集中に作業しているオブジェクトの種類が正確に判断されるためです。事前バインディングを使用すると、プログラムのコンパイル時にコンパイラがエラーを報告できるため、ランタイム エラーの数が減少し、エラーの重大性も低下します。

メモ：

遅延バインディングを使用できるのは、**Public** として宣言されている型メンバにアクセスするときだけです。**Friend** または **Protected Friend** として宣言されたメンバにアクセスすると、ランタイム エラーになります。

参照

関連項目

オブジェクト型 (Object)

CreateObject 関数 (Visual Basic)

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄

実行時のクラス情報の取得

オブジェクトの型を厳密に特定せずに、オブジェクト (**Object**) 型として定義された変数を使用することが必要な場合もあります。オブジェクト型として定義した変数は、整数型、ダブル型、オブジェクト データ型など、他のすべての型のオブジェクトを含みます。

このセクションの内容

オブジェクトの型の決定

オブジェクトが属するクラスを決定する方法について説明します。

文字列名によるプロパティまたはメソッドの呼び出し

メソッドを呼び出すときに位置でなく名前によって引数を指定するプロセスについて、順をおって説明します。

関連するセクション

Visual Basic におけるオブジェクト

オブジェクト指向プログラミングの基本事項を説明します。

オブジェクトの型の決定

汎用オブジェクト変数 (**Object** として宣言する変数) は、あらゆるクラスのオブジェクトを保持できます。**Object** 型の変数を使用するとき、オブジェクトのクラスに基づいて異なるアクションを行う必要があります。たとえば、特定のプロパティまたはメソッドをサポートしないオブジェクトもあります。Visual Basic には、どの型のオブジェクトがオブジェクト変数に格納されるのかを決める、2 つの方法が用意されています。**TypeName** 関数および **TypeOf...Is** 演算子です。

TypeName および TypeOf...Is

TypeName 関数は文字列を返します。このため、たとえば次のコードのように、オブジェクトのクラス名を格納したり表示したりする場合に適しています。

VB

```
Dim Ctrl As Control = New TextBox
MsgBox(TypeName(Ctrl))
```

TypeOf...Is 演算子は、**TypeName** を使った同等の文字列比較に比べ、処理が大幅に速くなります。このため、オブジェクトの型をテストする場合に適しています。次のコード片では、**If...Then...Else** ステートメントの中で **TypeOf...Is** が使用されています。

VB

```
If TypeOf Ctrl Is Button Then
    MsgBox("The control is a button.")
End If
```

TypeOf...Is 演算子を使用する場合には注意が必要です。**TypeOf...Is** 演算子は、オブジェクトが特定の型である場合、または特定の型から派生してる場合に **True** を返します。オブジェクトは、Visual Basic のほとんどの作業で使用されます。中には、文字列や整数のように、通常はオブジェクトと見なされない要素もあります。これらのオブジェクトは、Object から派生し、メソッドを継承しています。**Integer** が渡されて **Object** で評価されると、**TypeOf...Is** 演算子は **True** を返します。次の例では、パラメータ `InParam` は **Object** と **Integer** の両方であると報告されます。

VB

```
Sub CheckType(ByVal InParam As Object)
    ' Both If statements evaluate to True when an
    ' Integer is passed to this procedure.
    If TypeOf InParam Is Object Then
        MsgBox("InParam is an Object")
    End If
    If TypeOf InParam Is Integer Then
        MsgBox("InParam is an Integer")
    End If
End Sub
```

次の例は、**TypeOf...Is** と **TypeName** の両方を使って、引数 `Ctrl` で渡されるオブジェクトの型を調べます。`TestObject` プロシージャは、3 種類のコントロールで `ShowType` を呼び出します。

例を実行するには

1. 新しい Windows アプリケーション プロジェクトを作成し、Button、CheckBox、および RadioButton の各コントロールをフォームに追加します。
2. フォームのボタンから `TestObject` プロシージャを呼び出します。
3. フォームに次のコードを追加します。

   VB

   ```
   Sub ShowType(ByVal Ctrl As Object)
       'Use the TypeName function to display the class name as text.
   ```

```
        MsgBox(TypeName(Ctrl))
        'Use the TypeOf function to determine the object's type.
        If TypeOf Ctrl Is Button Then
            MsgBox("The control is a button.")
        ElseIf TypeOf Ctrl Is CheckBox Then
            MsgBox("The control is a check box.")
        Else
            MsgBox("The object is some other type of control.")
        End If
    End Sub

    Protected Sub TestObject()
        'Test the ShowType procedure with three kinds of objects.
        ShowType(Me.Button1)
        ShowType(Me.CheckBox1)
        ShowType(Me.RadioButton1)
    End Sub
```

参照

関連項目

オブジェクト型 (Object)
TypeName 関数 (Visual Basic)
If...Then...Else ステートメント (Visual Basic)
文字列型 (String) (Visual Basic)
整数型 (Integer) (Visual Basic)

概念

文字列名によるプロパティまたはメソッドの呼び出し

その他の技術情報

実行時のクラス情報の取得

文字列名によるプロパティまたはメソッドの呼び出し

ほとんどの場合、デザイン時にオブジェクトのプロパティとメソッドを調べ、オブジェクトに適切なコードを記述できます。ただし、オブジェクトのプロパティとメソッドをあらかじめ確認できない場合があります。また、実行時にプロパティの指定やメソッドの実行ができるような、柔軟なコードを記述することが必要な場合もあります。

CallByName 関数

たとえば、COM コンポーネントに演算子を渡すことによってユーザーが入力した式を評価する、クライアント アプリケーションなどが考えられます。新しい演算子を必要とするコンポーネントに新しい関数を繰り返し追加するとします。標準のオブジェクト アクセス方法では、クライアント アプリケーションの再コンパイルと再配布を行って、新しい演算子を使用できるようにする必要があります。これを避けるには、**CallByName** 関数を使用します。この関数を使用すると、アプリケーションを変更せずに、新しい演算子を文字列として渡すことができます。

CallByName 関数を使用すると、実行時にプロパティやメソッドを文字列で指定できます。**CallByName** 関数の使用方法を次に示します。

Result = **CallByName**(*Object*, *ProcedureName*, *CallType*, *Arguments*())

1 番目の引数 *Object* には、対象とするオブジェクトの名前を指定します。引数 *ProcedureName* には、呼び出すメソッドまたはプロパティ プロシージャの名前を文字列で指定します。引数 *CallType* には、呼び出すプロシージャの型を表す定数を指定します。メソッド呼び出しプロシージャの場合は `Microsoft.VisualBasic.CallType.Method`、プロパティ取得プロシージャの場合は `Microsoft.VisualBasic.CallType.Get`、プロパティ設定プロシージャの場合は `Microsoft.VisualBasic.CallType.Set` を指定します。任意で指定する引数 *Arguments* には、プロシージャに渡す引数を格納している **Object** 型の配列を指定します。

CallByName 関数は、現在のソリューションのクラスに対して使用できますが、一般的には、COM オブジェクトまたは .NET Framework アセンブリからのオブジェクトにアクセスするときに使用します。

たとえば、次のコードに示すように、新しい関数 `SquareRoot` を持つ `MathClass` クラスを含むアセンブリへの参照を追加するとします。

VB

```
Class MathClass
    Function SquareRoot(ByVal X As Double) As Double
        Return Math.Sqrt(X)
    End Function
    Function InverseSine(ByVal X As Double) As Double
        Return Math.Atan(X / Math.Sqrt(-X * X + 1))
    End Function
    Function Acos(ByVal X As Double) As Double
        Return Math.Atan(-X / Math.Sqrt(-X * X + 1)) + 2 * Math.Atan(1)
    End Function
End Class
```

アプリケーションは、テキスト ボックス コントロールを使用して、呼び出すメソッドとその引数を制御できます。たとえば、`TextBox1` に評価する式を入力し、`TextBox2` に関数の名前を入力する場合は、次のコードを使用して `TextBox1` の式に対して `SquareRoot` 関数を呼び出します。

VB

```
Private Sub CallMath()
    Dim Math As New MathClass
    Me.TextBox1.Text = CStr(CallByName(Math, Me.TextBox2.Text, _
       Microsoft.VisualBasic.CallType.Method, TextBox1.Text))
End Sub
```

`TextBox1` に「64」を入力し、`TextBox2` に「SquareRoot」を入力してから、`CallMath` プロシージャを呼び出した場合は、`TextBox1` の数値の平方根が評価されます。この例のコードは `SquareRoot` 関数を呼び出し、`TextBox1` に "8" (64 の平方根) を返します。この関数の引数は必須で、評価する式を含む文字列を指定します。`TextBox2` に不正な文字列を入力した場合、文字列にメソッドではなくプロパティの名前を指定した場合、またはメソッドに渡す必須引数が不足していた場合は、ランタイム エラーが発生します。**CallByName** 関数を使用する場合は、信頼性の高いエラー処理コードを追加して、このようなエラーに対処する必要があります。

メモ :

CallByName 関数を使用すると便利な場合もありますが、**CallByName** 関数を使用したプロシージャ呼び出しは、遅延バインディングによる呼び出しよりも速度が若干遅くなります。この関数を使用する場合には、利便性と性能面への影響とを比較検討する必要があります。ループ内などで、繰り返し実行される関数を呼び出す場合は、**CallByName** 関数の使用によってパフォーマンスが大幅に低下する可能性があります。

参照

処理手順

方法 : オブジェクトに対して複数のアクションを実行する

関連項目

CallByName 関数

概念

オブジェクトの型の決定

その他の技術情報

実行時のクラス情報の取得

COM 相互運用

コンポーネント オブジェクト モデル (COM: Component Object Model) を使用することによって、オブジェクトはその機能をその他のコンポーネントやホスト アプリケーションに公開できます。今日のほとんどのソフトウェアでは、COM オブジェクトが使用されています。.NET アセンブリは新規アプリケーションにとって最善の選択肢ですが、COM オブジェクトを使用する必要のある場合もあります。ここでは、Visual Basic での COM オブジェクトの作成と使用に関連する問題について説明します。

このセクションの内容

COM 相互運用の概要

COM 相互運用性の概要を説明します。

方法 : Visual Basic から COM オブジェクトを参照する

タイプ ライブラリのある COM オブジェクトに参照を追加する方法を説明します。

方法 : ActiveX コントロールを操作する

既存の ActiveX コントロールを使用して Visual Studio ツールボックスに機能を追加する方法を説明します。

チュートリアル : Windows API の呼び出し

Windows オペレーティング システムに含まれる API を呼び出す手順について説明します。

方法 : Windows API を呼び出す (Visual Basic)

User32.dll の **MessageBox** 関数を定義し、呼び出す方法を説明します。

チュートリアル : Visual Basic 2005 での COM オブジェクトの作成

COM クラス テンプレートを使用する場合と使用しない場合の COM オブジェクトの作成手順を説明します。

相互運用性のトラブルシューティング

COM を使用する際、発生する可能性のある問題について説明します。

.NET Framework アプリケーションにおける COM 相互運用性

1 つのアプリケーションで COM オブジェクトと .NET Framework オブジェクトを同時に使用する方法について簡単に説明します。

関連するセクション

チュートリアル : COM オブジェクトによる継承の実装

Visual Basic での継承の基本として、COM オブジェクト内でクラスを使用する方法を説明します。

アンマネージ コードとの相互運用

共通言語ランタイムによって提供される相互運用サービスについて説明します。

.NET Framework への COM コンポーネントの公開

COM 相互運用機能を通じて COM の型を呼び出す手順を説明します。

COM への .NET Framework コンポーネントの公開

COM からのマネージ型の準備および使用方法について説明します。

相互運用固有の属性の適用

アンマネージ コードと相互運用するときに使用できる属性について説明します。

COM 相互運用の概要

コンポーネント オブジェクト モデル (COM: Component Object Model) を使用することによって、オブジェクトはその機能をその他のコンポーネントやホスト アプリケーションに公開できます。COM オブジェクトは長年にわたり Windows プログラミングの基盤として使用されてきましたが、共通言語ランタイム (CLR: Common Language Runtime) 用に設計されたアプリケーションには多数の利点があります。

COM で開発されたアプリケーションは今後置き換えられ、最終的には .NET Framework アプリケーションが使われるようになると考えられます。しかし、それまでは Visual Studio で COM オブジェクトを使用または作成することが必要な場合があります。COM との相互運用性、つまり COM 相互運用機能によって、既存の COM オブジェクトを使用しながら、独自のペースで .NET Framework に移行していくことができます。

.NET Framework を使用して COM コンポーネントを作成すると、登録を必要としない COM 相互運用機能を利用できます。これによって、複数のバージョンの DLL がコンピュータにインストールされている場合に、どの DLL バージョンをアクティブにするのかを制御できます。エンド ユーザーは、XCOPY または FTP を使用して、アプリケーションを実行できる自分のコンピュータ上の適切なディレクトリにコピーできます。詳細については、「登録を必要としない COM 相互運用機能」を参照してください。

マネージ コードとマネージ データ

.NET Framework 用に開発されたコードをマネージ コードと呼びます。マネージ コードには、CLR によって使用されるメタデータが含まれます。.NET Framework アプリケーションによって使用されるデータをマネージ データと呼びます。これは、メモリの割り当ておよび再要求、型チェックの実行などのデータ関連のタスクが、ランタイムによって管理されるためです。Visual Basic 2005 では既定でマネージ コードとマネージ データが使用されますが、相互運用機能アセンブリ (このページの後半で説明します) を使用することによって、COM オブジェクトのアンマネージ コードとアンマネージ データにアクセスできます。

アセンブリ

アセンブリとは、.NET Framework アプリケーションの主要なビルド ブロックです。アセンブリは機能の集合体であり、1 つ以上のファイルで構成される単一の実装ユニットとしてビルド、バージョン管理、および配置されます。各アセンブリにはアセンブリ マニフェストが含まれます。

タイプ ライブラリとアセンブリ マニフェスト

タイプ ライブラリは、メンバの名前やデータ型など、COM オブジェクトの特性を記述します。アセンブリ マニフェストは、.NET Framework アプリケーションについて同様の機能を果たします。アセンブリ マニフェストには次の情報が含まれます。

- アセンブリの ID、バージョン、カルチャ、およびデジタル署名
- アセンブリ実装を構成するファイル
- アセンブリを構成する型およびリソース (アセンブリからエクスポートされる型およびリソースも含む)
- その他のアセンブリへのコンパイル時の依存関係
- アセンブリを正常に実行するために必要なアクセス許可

アセンブリおよびアセンブリ マニフェストの詳細については、「アセンブリ」を参照してください。

タイプ ライブラリのインポートとエクスポート

Visual Studio には Tlbimp ユーティリティがあります。Tlbimp を使用すると、タイプ ライブラリから .NET Framework アプリケーションに情報をインポートできます。また、Tlbexp ユーティリティを使用して、アセンブリからタイプ ライブラリを生成できます。

Tlbimp および Tlbexp の詳細については、「タイプ ライブラリ インポータ (Tlbimp.exe)」および「タイプ ライブラリ エクスポータ (Tlbexp.exe)」を参照してください。

相互運用機能アセンブリ

相互運用機能アセンブリは、マネージ コードとアンマネージ コードの仲介役として機能する .NET Framework アセンブリであり、それに対応する .NET Framework マネージ メンバに COM オブジェクト メンバを割り当てます。Visual Basic 2005 によって作成される相互運用機能アセンブリは、相互運用性のマーシャリングなど、COM オブジェクトの操作で発生する詳細な作業の多くを処理します。

相互運用性のマーシャリング

すべての .NET Framework アプリケーションは、使用するプログラミング言語の種類に関係なく、オブジェクトを相互運用できるようにするための共通の型セットを共有します。COM オブジェクトのパラメータおよび戻り値の型は、マネージ コードで使用されるデータ型とは異なることがあります。相互運用性のマーシャリングとは、COM オブジェクトからマネージ コードへの変換およびマネージ コードから COM オブジェクトへの変換のときに、パラメータと戻り値を対応するデータ型にパッケージ化する処理です。詳細については、「相互運用マーシャリングの概要」を参照してください。

参照

処理手順

チュートリアル : COM オブジェクトによる継承の実装

相互運用性のトラブルシューティング

関連項目

タイプ ライブラリ インポータ (Tlbimp.exe)

タイプ ライブラリ エクスポータ (Tlbexp.exe)

概念

アセンブリ

相互運用マーシャリングの概要

その他の技術情報

COM 相互運用

アンマネージ コードとの相互運用

登録を必要としない COM 相互運用機能

方法：Visual Basic から COM オブジェクトを参照する

Visual Basic でタイプ ライブラリを持つ COM オブジェクトへの参照を追加する作業は、以前のバージョンの場合と似ています。ただし、Visual Basic では、相互運用機能アセンブリを作成する作業が加わります。COM オブジェクトのメンバへの参照は、相互運用機能アセンブリを経由して、実際の COM オブジェクトに転送されます。COM オブジェクトからの応答は、相互運用機能アセンブリを経由して .NET Framework アプリケーションにルーティングされます。

COM オブジェクトへの参照を追加するには

1. [プロジェクト] メニューの [参照の追加] をクリックし、[COM] タブをクリックします。
2. 使用するコンポーネントを COM オブジェクトのリストから選択します。
3. 相互運用機能アセンブリへのアクセスを単純化するために、COM オブジェクトを使用するクラスまたはモジュールの先頭に **Imports** ステートメントを追加します。

   Visual Basic では、統合開発環境 (IDE: Integrated Development Environment) でタイプ ライブラリへの参照を追加すると、相互運用機能アセンブリが自動的に作成されます。コマンド ラインから操作するときは、Tlbimp ユーティリティを使用して、手動で相互運用機能アセンブリを作成できます。

Tlbimp を使用して相互運用機能アセンブリを作成するには

1. Tlbimp の格納ディレクトリが検索パスに指定されておらず、格納ディレクトリ以外の作業ディレクトリにいる場合は、Tlbimp の格納ディレクトリを検索パスに追加します。
2. コマンド プロンプトから Tlbimp を起動し、次の情報を入力します。

   - タイプ ライブラリを含む DLL の名前と場所
   - 情報を格納する名前空間の名前と場所
   - 作成する相互運用機能アセンブリの名前と場所

   次に例を示します。

   ```
   Tlbimp test3.dll /out:NameSpace1 /out:Interop1.dll
   ```

   Tlbimp を使用すると、タイプ ライブラリおよび未登録の COM オブジェクトについても相互運用機能アセンブリを作成できます。ただし、相互運用機能アセンブリを経由して参照される COM オブジェクトは、使用先のコンピュータで正しく登録されている必要があります。COM オブジェクトを登録するには、Windows オペレーティング システムに付属の Regsvr32 ユーティリティを使用します。

参照

処理手順

チュートリアル : COM オブジェクトによる継承の実装
相互運用性のトラブルシューティング

関連項目

タイプ ライブラリ インポータ (Tlbimp.exe)
タイプ ライブラリ エクスポータ (Tlbexp.exe)
Imports ステートメント

その他の技術情報

COM 相互運用

方法：ActiveX コントロールを操作する

ActiveX コントロールは、Web ページやその他のアプリケーションに挿入できる COM コンポーネントまたは COM オブジェクトです。これによって、他者によってプログラミングされた、パッケージ化された機能を再利用できます。Visual Basic 6.0 以前のバージョン用に開発された ActiveX コントロールを使用して、Visual Studio のツールボックスに機能を追加できます。

ActiveX コントロールをツールボックスに追加するには

1. [ツール] メニューの [ツールボックス アイテムの選択] をクリックします。

   ツールボックス アイテムの選択 ダイアログ ボックスが表示されます。

2. [COM コンポーネント] タブをクリックします。

3. 使用する ActiveX コントロールの横にあるチェック ボックスをオンにし、[OK] をクリックします。

   新しいコントロールがその他のツールと共にツールボックスに表示されます。

メモ：
Aximp ユーティリティを使用して、ActiveX コントロールの相互運用機能アセンブリを手動で作成できます。詳細については、「Windows フォーム ActiveX コントロール インポータ (Aximp.exe)」を参照してください。

参照

処理手順

方法 : Windows フォームに ActiveX コントロールを追加する

相互運用性のトラブルシューティング

関連項目

[ツールボックス アイテムの選択] ダイアログ ボックス (Visual Studio)

Windows フォーム ActiveX コントロール インポータ (Aximp.exe)

概念

Windows フォームで ActiveX コントロールをホストする場合の考慮事項

その他の技術情報

COM 相互運用

チュートリアル：Windows API の呼び出し

Windows API は、Windows オペレーティング システムの一部であるダイナミック リンク ライブラリ (DLL: Dynamic Link Library) です。同等のプロシージャを独自に作成するのが困難なときに、Windows API を利用してタスクを実行します。たとえば、Windows には **FlashWindowEx** という名前の関数があります。この関数を使用すると、アプリケーションのタイトル バーを明るい網かけと暗い網かけに交互に変化させることができます。

コードで Windows API を使用する利点は、開発時間を短縮できることです。Windows API には、作成済みの、すぐに利用できる便利な関数が数多く用意されているため、開発時間を短縮できます。Windows API の短所としては、操作が難しい場合があることや、適切に動作しないときに修正できない場合があることが挙げられます。

Windows API は、相互運用性の特別なカテゴリを表します。Windows API ではマネージ コードを使用せず、タイプ ライブラリが組み込まれていません。また、Visual Studio で使用されるデータ型とは異なるデータ型を使用します。このような相違点、および Windows API は COM オブジェクトではないという理由により、Windows API と .NET Framework の相互運用は、PInvoke (プラットフォーム呼び出し) を使用して実行されます。プラットフォーム呼び出しは、DLL に実装されたアンマネージ関数をマネージ コードが呼び出すことができるサービスです。詳細については、「アンマネージ DLL 関数の処理」を参照してください。Visual Basic で PInvoke を使用するには、**Declare** ステートメントを使用するか、または **DllImport** 属性を空のプロシージャに適用します。

過去において Windows API 呼び出しは Visual Basic プログラミングの重要な部分を占めていましたが、Visual Basic 2005 ではほとんど必要がなくなりました。Windows API 呼び出しを使用する代わりに、できる限り .NET Framework からマネージ コードを使用してタスクを実行する必要があります。このチュートリアルでは、Windows API を使用する必要のある状況についての情報を提供します。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Declare による API 呼び出し

Windows API を呼び出す最も一般的な方法は、**Declare** ステートメントを使用する方法です。

DLL プロシージャを宣言するには

1. 呼び出しの対象となる関数の名前、引数、引数の型、戻り値、およびその関数を含む DLL の名前と場所を確認します。

   メモ：

   Windows API の詳細については、プラットフォーム SDK の「Windows API」で、Win32 SDK に関する記述を参照してください。Windows API で使用される定数の詳細については、プラットフォーム SDK に組み込まれている Windows.h などのヘッダー ファイルを参照してください。

2. [ファイル] メニューの [新規作成] をクリックし、[プロジェクト] をクリックして、新規の Windows アプリケーション プロジェクトを開きます。[新しいプロジェクト] ダイアログ ボックスが表示されます。
3. Visual Basic プロジェクト テンプレートの一覧の [Windows アプリケーション] を選択します。新しいプロジェクトが表示されます。
4. 次の **Declare** 関数を、DLL を使用するクラスまたはモジュールに追加します。

   VB

```
Declare Auto Function MBox Lib "user32.dll" Alias "MessageBox" ( _
    ByVal hWnd As Integer, _
    ByVal txt As String, _
    ByVal caption As String, _
    ByVal Typ As Integer) _
    As Integer
```

Declare ステートメントの構成要素

Declare ステートメントは次の要素で構成されます。

Auto 修飾子

Auto 修飾子は、共通言語ランタイムの実行時規則に従い、メソッド名またはエイリアス名 (指定されている場合) に基づいて、文字列を変換するようにランタイムに指示します。

Lib キーワードと Alias キーワード

Function キーワードに続く名前は、インポートした関数にアクセスするためにプログラムで使用される名前です。この名前には、呼び出しの対象となる関数の実際の名前を使用できます。また、任意の有効なプロシージャ名を使用してから、**Alias** キーワードを使用して呼び出しの対象となる関数の実際の名前を指定することもできます。

Lib キーワードは、呼び出しの対象となる関数を含む DLL の名前と場所の前に指定します。Windows システム ディレクトリに格納されたファイルのパスを指定する必要はありません。

呼び出しの対象となる関数の名前が、有効な Visual Basic プロシージャ名ではない場合、またはアプリケーション内のその他の項目の名前と競合する場合は、**Alias** キーワードを使用します。**Alias** は呼び出し対象の関数の真の名前を示します。

引数とデータ型の宣言

引数とそのデータ型を宣言します。この部分は、Windows が使用するデータ型が Visual Studio のデータ型に対応していないため、難しい部分です。Visual Basic では、マーシャリング と呼ばれるプロセスで、引数が互換性のあるデータ型に変換されます。System.Runtime.InteropServices 名前空間で定義されている MarshalAsAttribute 属性を使用することによって、引数がマーシャリングされる方法を明示的に制御できます。

> **メモ :**
>
> Visual Basic の以前のバージョンでは、どのデータ型のデータでも使用可能とするパラメータ **As Any** を宣言できました。Visual Basic では、すべての **Declare** ステートメントに特定のデータ型を使用する必要があります。

Windows API の定数

一部の引数は定数の組み合わせです。たとえば、このチュートリアルで示す `MessageBox` API は、メッセージ ボックスの表示方法を制御する `Typ` と呼ばれる整数の引数を受け取ります。これらの定数の数値を確認するには、WinUser.h ファイル内の **#define** ステートメントを調べます。数値は通常 16 進数で示されるため、数値を加算したり 10 進数に変換したりする場合は、計算機を使用することをお勧めします。たとえば、感嘆符スタイルの定数 `MB_ICONEXCLAMATION` 0x00000030 と Yes/No スタイルの定数 `MB_YESNO` 0x00000004 を組み合わせる場合は、数値を加算して、0x00000034 (10 進数では 52) の結果を得ることができます。10 進数の結果を直接使用することもできますが、アプリケーション内でこれらの値を定数として宣言し、**Or** 演算子を使用して組み合わせることをお勧めします。

Windows API 呼び出しの定数を宣言するには

1. 呼び出す Windows 関数のマニュアルを参照してください。使用する定数の名前と、これらの定数に対応する数値を含む .h ファイルの名前を決定します。
2. メモ帳などのテキスト エディタを使用して、ヘッダー (.h) ファイルの内容を表示し、使用する定数に関連付けられた値を探します。たとえば、`MessageBox` API は、定数 `MB_ICONQUESTION` を使用して、メッセージ ボックスに疑問符を表示します。`MB_ICONQUESTION` の定義は WinUser.h にあり、次のように表示されます。

   ```
   #define MB_ICONQUESTION 0x00000020L
   ```

3. クラスまたはモジュールに同等の **Const** ステートメントを追加して、これらの定数をアプリケーションで使用できるようにします。以下に例を示します。

   VB

   ```
   Const MB_ICONQUESTION As Integer = &H20
   Const MB_YESNO As Integer = &H4
   Const IDYES As Integer = 6
   Const IDNO As Integer = 7
   ```

DLL プロシージャを呼び出すには

1. `Button1` という名前のボタンをプロジェクトのスタートアップ フォームに追加し、ボタンをダブルクリックしてコードを表示します。ボタンのイベント ハンドラが表示されます。
2. 追加したボタンの `Click` イベント ハンドラにプロシージャを呼び出すためのコードを追加し、適切な引数を指定します。

VB

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click

    ' Stores the return value.
    Dim RetVal As Integer
    RetVal = MBox(0, "Declare DLL Test", "Windows API MessageBox", _
        MB_ICONQUESTION Or MB_YESNO)

    ' Check the return value.
    If RetVal = IDYES Then
        MsgBox("You chose Yes")
    Else
        MsgBox("You chose No")
    End If
End Sub
```

3. F5 キーを押してプロジェクトを実行します。[はい] 応答ボタンと [いいえ] 応答ボタンのあるメッセージ ボックスが表示されます。いずれかのボタンをクリックします。

データのマーシャリング

Visual Basic では、Windows API 呼び出しで使用されるパラメータと戻り値のデータ型が自動的に変換されますが、**MarshalAs** 属性を使用して、API が受け取るアンマネージ データ型を明示的に指定できます。相互運用マーシャリングの詳細については、「相互運用マーシャリング」を参照してください。

API 呼び出しで Declare および MarshalAs を使用するには

1. 呼び出しの対象となる関数の名前、引数、データ型、および戻り値を確認します。
2. **MarshalAs** 属性へのアクセスを簡単にするために、クラスまたはモジュールのコードの先頭に **Imports** ステートメントを追加します。次に例を示します。

   VB

   ```
   Imports System.Runtime.InteropServices
   ```

3. インポートした関数の関数プロトタイプを、使用するクラスまたはモジュールに追加します。また、パラメータまたは戻り値に **MarshalAs** 属性を適用します。次の例では、**void*** 型を受け取る API 呼び出しを AsAny としてマーシャリングしています。

   VB

   ```
   Declare Sub SetData Lib "..\LIB\UnmgdLib.dll" ( _
       ByVal x As Short, _
       <MarshalAsAttribute(UnmanagedType.AsAny)> _
           ByVal o As Object)
   ```

DllImport による API 呼び出し

DllImport 属性では、もう 1 つの方法でタイプ ライブラリを持たない DLL 内の関数を呼び出すことができます。**DllImport** は、**Declare** ステートメントを使用する場合とほぼ同じ働きをしますが、関数の呼び出し方をより細かく制御できます。

呼び出しが共有 (static と呼ばれる場合もあります) メソッドを参照している限り、ほとんどの Windows API 呼び出しで **DllImport** を使用できます。クラスのインスタンスが必要なメソッドは使用できません。**Declare** ステートメントとは異なり、**DllImport** では **MarshalAs** 属性を使用できません。

DllImport 属性を使用して Windows API を呼び出すには

1. [ファイル] メニューの [新規作成] をクリックし、[プロジェクト] をクリックして、新規の Windows アプリケーション プロジェクトを開きます。[新

しいプロジェクト] ダイアログ ボックスが表示されます。

2. Visual Basic プロジェクト テンプレートの一覧の [Windows アプリケーション] を選択します。新しいプロジェクトが表示されます。
3. `Button2` という名前のボタンをスタートアップ フォームに追加します。
4. `Button2` をダブルクリックしてフォームのコードを表示します。
5. **DllImport** へのアクセスを簡単にするために、スタートアップ フォーム クラスのコードの先頭に **Imports** ステートメントを追加します。

   VB

   ```
   Imports System.Runtime.InteropServices
   ```

6. フォームの **End Class** ステートメントの前に空の関数を宣言し、`MoveFile` という名前を付けます。
7. 関数宣言に **Public** 修飾子および **Shared** 修飾子を適用し、Windows API 関数で使用される引数に基づいて `MoveFile` のパラメータを設定します。

   VB

   ```
   Public Shared Function MoveFile( _
       ByVal src As String, _
       ByVal dst As String) _
       As Boolean
       ' Leave the body of the function empty.
   End Function
   ```

   関数は任意の有効なプロシージャ名を持つことができます。**DllImport** 属性によって、この名前が DLL で指定されます。また、パラメータと戻り値の相互運用性マーシャリングも処理されるため、API によって使用されるデータ型と同様の Visual Studio データ型を選択できます。

8. **DllImport** 属性を空の関数に適用します。最初のパラメータは、呼び出しの対象となる関数を含む DLL の名前と場所です。Windows システム ディレクトリに格納されたファイルのパスを指定する必要はありません。2 番目のパラメータは、Windows API 内の関数の名前を指定する名前付き引数です。この例では、**DllImport** 属性は `MoveFile` の呼び出しを強制的に KERNEL32.DLL 内の `MoveFileW` に転送します。`MoveFileW` メソッドは、パス `src` からパス `dst` にファイルをコピーします。

   VB

   ```
   <DllImport("KERNEL32.DLL", EntryPoint:="MoveFileW", SetLastError:=True, _
       CharSet:=CharSet.Unicode, ExactSpelling:=True, _
       CallingConvention:=CallingConvention.StdCall)> _
   Public Shared Function MoveFile( _
       ByVal src As String, _
       ByVal dst As String) _
       As Boolean
       ' Leave the body of the function empty.
   End Function
   ```

9. 関数を呼び出すには、コードを `Button2_Click` イベント ハンドラに追加します。

   VB

   ```
   Private Sub Button2_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles Button2.Click

       Dim RetVal As Boolean = MoveFile("c:\tmp\Test.txt", "c:\Test.txt")
       If RetVal = True Then
           MsgBox("The file was moved successfully.")
       Else
   ```

```
            MsgBox("The file could not be moved.")
        End If
    End Sub
```

10. Test.txt という名前のファイルを作成し、ハード ドライブ上の C:\Tmp ディレクトリに配置します。必要に応じて Tmp ディレクトリを作成します。
11. F5 キーを押してアプリケーションを起動します。メイン フォームが表示されます。
12. [Button2] をクリックします。ファイルを移動できる場合は、"The file was moved successfully" というメッセージが表示されます。

参照

関連項目

Declare ステートメント
Auto
Alias
Lib
DllImportAttribute
MarshalAsAttribute

概念

マネージ コードでのプロトタイプの作成
Callback のサンプル

その他の技術情報

COM 相互運用

方法：Windows API を呼び出す (Visual Basic)

次に示すのは、user32.dll の **MessageBox** 関数を定義および呼び出して、その関数に文字列を渡す例です。

使用例

このコードの例は、IntelliSense コード スニペットとしても利用できます。コード スニペット ピッカーでは、これは [Visual Basic Language] にあります。詳細については、「方法 : コードにスニペットを挿入する (Visual Basic)」を参照してください。

VB

```
' Defines the MessageBox function.
Public Class Win32
    Declare Auto Function MessageBox Lib "user32.dll" ( _
        ByVal hWnd As Integer, ByVal txt As String, _
        ByVal caption As String, ByVal Type As Integer) _
        As Integer
End Class

' Calls the MessageBox function.
Public Class DemoMessageBox
    Public Shared Sub Main()
        Win32.MessageBox(0, "Here's a MessageBox", "Platform Invoke Sample", 0)
    End Sub
End Class
```

コードのコンパイル方法

この例で必要な要素は次のとおりです。

- System 名前空間への参照。

堅牢性の高いプログラム

次の条件を満たす場合は、例外が発生する可能性があります。

- メソッドが静的でないか、抽象であるか、定義済みである場合。親の型がインターフェイスであるか、*name* または *dllName* の長さがゼロである場合 (ArgumentException)
- *name* または *dllName* が **Nothing** である場合 (ArgumentNullException)
- コンテナの型が `CreateType` を使用して作成済みの型である場合 (InvalidOperationException)

参照

処理手順

チュートリアル : Windows API の呼び出し

概念

プラットフォーム呼び出しの詳細
プラットフォーム呼び出しの例
アンマネージ DLL 関数の処理
リフレクション出力によるメソッドの定義

その他の技術情報

COM 相互運用

チュートリアル : Visual Basic 2005 での COM オブジェクトの作成

新しいアプリケーションまたはコンポーネントを作成する場合は、.NET Framework アセンブリを作成します。また、Visual Basic 2005 を使用することで、.NET Framework コンポーネントを COM に公開することもできます。これにより、COM コンポーネントを必要とするレガシ アプリケーション スイートに新しいコンポーネントを作成できるようになります。このチュートリアルでは、Visual Basic を使用して .NET Framework オブジェクトを COM オブジェクトとして公開する方法を、COM クラス テンプレートを使う場合と使わない場合の両方で紹介します。

COM オブジェクトを公開するための最も簡単な方法は、COM クラス テンプレートを使用する方法です。COM クラス テンプレートは、新規クラスを作成し、クラスと相互運用レイヤが COM オブジェクトとして生成されるようにプロジェクトを設定し、生成された COM オブジェクトをオペレーティング システムに登録します。

メモ :

Visual Basic で作成したクラスも、アンマネージ コードで使用する COM オブジェクトとして公開できますが、これは本当の COM オブジェクトではないため、Visual Basic では使用できません。詳細については、「.NET Framework アプリケーションにおける COM 相互運用性」を参照してください。

メモ :

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

COM クラス テンプレートを使用して COM オブジェクトを作成するには

1. [ファイル] メニューで [新しいプロジェクト] をクリックして、新しい Windows アプリケーション プロジェクトを開きます。
2. [プロジェクトの種類] フィールドの [新しいプロジェクト] ダイアログ ボックスで、Windows が選択されていることを確認します。テンプレート ペインの一覧の [クラス ライブラリ] をクリックし、[OK] をクリックします。新しいプロジェクトが表示されます。
3. [プロジェクト] メニューの [新しい項目の追加] をクリックします。[新しい項目の追加] ダイアログ ボックスが表示されます。
4. テンプレート ペインの一覧の [COM クラス] を選択し、[追加] をクリックします。Visual Basic によって新しいクラスが追加され、COM 相互運用に新しいプロジェクトが構成されます。
5. プロパティ、メソッド、イベントなどのコードを COM クラスに追加します。
6. [ビルド] メニューで [ClassLibrary1 のビルド] を選択します。Visual Basic によってアセンブリが作成され、COM オブジェクトが OS に登録されます。

COM クラス テンプレートを使用しない COM オブジェクトの作成

COM クラスは、COM クラス テンプレートを使用しなくても、手動で作成することもできます。コマンド ラインから操作するときや、COM オブジェクトの定義方法を制御するときに、手動による作業手順が役に立ちます。

COM オブジェクトが生成されるようにプロジェクトを設定するには

1. [ファイル] メニューで [新しいプロジェクト] をクリックして、新しい Windows アプリケーション プロジェクトを開きます。
2. [プロジェクトの種類] フィールドの [新しいプロジェクト] ダイアログ ボックスで、Windows が選択されていることを確認します。テンプレート ペインの一覧の [クラス ライブラリ] をクリックし、[OK] をクリックします。新しいプロジェクトが表示されます。
3. ソリューション エクスプローラで、プロジェクトを右クリックし、[プロパティ] をクリックします。プロジェクト デザイナ が表示されます。
4. [コンパイル] タブをクリックします。
5. [COM の相互運用機能に登録] チェック ボックスをオンにします。

クラス内のコードを設定して COM オブジェクトを作成するには

1. ソリューション エクスプローラで、Class1.vb をダブルクリックして、そのコードを表示します。
2. クラスの名前を `ComClass1` に変更します。

3. `ComClass1` に次の定数を追加します。これにより、COM オブジェクトに必要な グローバル一意識別子 (GUID) 定数が格納されます。

VB
```
Public Const ClassId As String = ""
Public Const InterfaceId As String = ""
Public Const EventsId As String = ""
```

4. グローバル一意識別子 (GUID) を取得するには、Guidgen.exe を起動して Guidgen ユーティリティを開始します。Guidgen アプリケーションによって表示される形式のリストで、[Registry Format] をクリックします。[New GUID] ボタンをクリックして GUID を生成し、[Copy] ボタンをクリックして GUID をクリップボードにコピーします。

5. `ClassId` の空白の文字列を GUID で置き換え、前後にある中かっこを削除します。たとえば、Guidgen によって提供された GUID が "`{2C8B0AEE-02C9-486e-B809-C780A11530FE}`" である場合、コードは次のようになります。

VB
```
Public Const ClassId As String = "2C8B0AEE-02C9-486e-B809-C780A11530FE"
```

6. 次の例のように、`InterfaceId` 定数と `EventsId` 定数に対してこれまでの手順を繰り返します。

VB
```
Public Const InterfaceId As String = "3D8B5BA4-FB8C-5ff8-8468-11BF6BD5CF91"
Public Const EventsId As String = "2B691787-6ED7-401e-90A4-B3B9C0360E31"
```

メモ：

COM コンポーネントが他の COM コンポーネントと競合しないよう、GUID が新しく、一意であることを確認する必要があります。

7. **ComClass** 属性を `ComClass1` に追加して、クラス ID、インターフェイス ID、およびイベント ID のグローバル一意識別子 (GUID: global unique identifier) を指定します。次に例を示します。

VB
```
<ComClass(ComClass1.ClassId, ComClass1.InterfaceId, ComClass1.EventsId)> _
Public Class ComClass1
```

8. COM クラスにはパラメータなしの `Public Sub New()` コンストラクタを指定する必要があります。指定しなかった場合、クラスが正しく登録されません。パラメータなしのコンストラクタをクラスに追加します。

VB
```
Public Sub New()
    MyBase.New()
End Sub
```

9. プロパティ、メソッド、およびイベントをクラスに追加して、最後に **End Class** ステートメントを記述します。[ビルド] メニューで [ソリューションのビルド] を選択します。Visual Basic によってアセンブリが作成され、COM オブジェクトが OS に登録されます。

メモ：

Visual Basic 2005 で生成した COM オブジェクトは本当の COM オブジェクトではないため、他の Visual Basic 2005 アプリケーションでは使用できません。そのような COM オブジェクトへの参照を追加しようとすると、エラーが発生します。詳細については、「.NET Framework アプリケーションにおける COM 相互運用性」を参照してください。

参照

処理手順

チュートリアル : COM オブジェクトによる継承の実装

相互運用性のトラブルシューティング

関連項目

ComClassAttribute クラス

#Region ディレクティブ

その他の技術情報

COM 相互運用

.NET Framework アプリケーションにおける COM 相互運用性

相互運用性のトラブルシューティング

COM オブジェクトと .NET Framework のマネージ コードの間で相互運用するときには、以下のような一般的な問題が発生することがあります。

相互運用マーシャリング

.NET Framework に含まれないデータ型を使用するケースも考えられます。相互運用アセンブリは COM オブジェクトの大部分の操作を処理しますが、マネージ オブジェクトを COM に公開するときは、使用するデータ型を制御する必要があります。たとえば、クラス ライブラリ内の構造体では、Visual Basic 6.0 で作成された COM オブジェクトに送信される文字列に対して、BStr アンマネージ型を指定する必要があります。そのような場合は、MarshalAsAttribute 属性を使用して、マネージ型がアンマネージ型として公開されるように指定できます。

固定長文字列のアンマネージ コードへのエクスポート

6.0 以前の Visual Basic では、文字列は null 終了文字のないバイト シーケンスとして COM オブジェクトにエクスポートされます。Visual Basic 2005 では、その他の言語との互換性を考慮して、文字列をエクスポートするときに終了文字が追加されます。この非互換性の問題に対処する最善の方法は、終了文字のない文字列をバイト型 (**Byte**) または char 型 (**Char**) の配列としてエクスポートする方法です。

継承階層のエクスポート

マネージ クラスの階層は、COM オブジェクトとして公開されるときに平坦化されます。たとえば、メンバを含む基本クラスを定義し、COM オブジェクトとして公開される派生クラス内でその基本クラスを継承すると、COM オブジェクト内の派生クラスを使用するクライアントは、継承されるメンバを使用できなくなります。基本クラスのメンバに COM オブジェクトからアクセスできるのは、基本クラスのインスタンスとしてだけであり、その基本クラスも COM オブジェクトとして作成されている場合に限られます。

オーバーロードされたメソッド

Visual Basic ではオーバーロードされたメソッドを作成できますが、COM ではオーバーロードされたメソッドはサポートされていません。オーバーロードされたメソッドを含むクラスを COM オブジェクトとして公開した場合、オーバーロードされたメソッドに対して新しいメソッド名が生成されます。

たとえば、`Synch` メソッドについて、2 つのオーバーロードを持つクラスがあるとします。このクラスを COM オブジェクトとして公開した場合、`Synch` と `Synch_2` というメソッド名が新たに生成されます。

COM オブジェクトのコンシューマから見ると、この名前変更には 2 つの問題点があります。

1. クライアントが想定していないメソッド名が生成される可能性があります。
2. COM オブジェクトとして公開されたクラスまたはその基本クラスに新しいオーバーロードが追加された場合、いったん生成されたメソッド名が変わることがあり、バージョン管理の問題が生じる可能性があります。

この 2 つの問題を解決するには、COM オブジェクトとして公開されるオブジェクトを開発する場合は、オーバーロードの使用を避け、各メソッドに一意の名前を指定するようにします。

相互運用アセンブリを通じた COM オブジェクトの使用

相互運用アセンブリを使用するときは、それらが表す COM オブジェクトに置き換わるマネージ コードとして扱います。ただし、相互運用アセンブリはラッパーであり、実際の COM オブジェクトではないため、相互運用アセンブリと標準アセンブリの使用には多少の違いがあります。相違点としては、クラスの公開、パラメータおよび戻り値のデータ型などがあります。

インターフェイスおよびクラスとして公開されるクラス

COM クラスは、標準アセンブリのクラスとは異なり、COM クラスを表すインターフェイスおよびクラスの両方として相互運用アセンブリで公開されます。インターフェイスの名前は COM クラスの名前と同じです。相互運用クラスの名前は元の COM クラスと同じですが、"Class" という語が末尾に追加されます。たとえば、COM オブジェクトの相互運用アセンブリへの参照を持つプロジェクトがあるとします。COM クラスの名前が `MyComClass` である場合、IntelliSense およびオブジェクト ブラウザには、`MyComClass` という名前のインターフェイスと `MyComClassClass` という名前のクラスが表示されます。

.NET Framework クラスのインスタンスの作成

通常、.NET Framework クラスのインスタンスを作成するときは、**New** ステートメントとクラス名を使用します。相互運用アセンブリによって表される COM クラスを持つということは、**New** ステートメントとインターフェイスを使用できる 1 つのケースです。COM クラスと **Inherits** ステートメントを使用する場合を除いて、インターフェイスはクラスの場合と同様に使用できます。次のコードは、`Command` オブジェクトを、Microsoft ActiveX データ オブジェクト 2.8 ライブラリの COM オブジェクトへの参照があるプロジェクト内で作成する方法を示します。

VB

```
Dim cmd As New ADODB.Command
```

ただし、COM クラスを派生クラスの基本クラスとして使用する場合は、COM クラスを表す相互運用クラスを使用する必要があります。次に例を示します。

VB

```
Class DerivedCommand
    Inherits ADODB.CommandClass
End Class
```

メモ：

相互運用機能アセンブリは、COM クラスを表すインターフェイスを暗黙的に実装します。**Implements** ステートメントを使用してそれらのインターフェイスを実装しないでください。エラーが発生します。

パラメータおよび戻り値のデータ型

標準アセンブリのメンバと異なり、相互運用機能アセンブリのメンバは、オブジェクトの元の宣言で使用されているデータ型とは異なるデータ型を持つ場合があります。相互運用アセンブリは COM 型を互換性のある共通言語ランタイム型に暗黙的に変換しますが、実行時エラーが発生しないように、両方の側で使用されるデータ型に注意する必要があります。たとえば、Visual Basic 6.0 以前で作成された COM オブジェクトでは、整数型 (**Integer**) の値に、対応する.NET Framework の短整数型 (**Short**) が使用されます。インポートしたメンバを使用する場合は、オブジェクト ブラウザでその特性を調べてから使用することをお勧めします。

モジュール レベルの COM メソッド

ほとんどの COM オブジェクトは、**New** キーワードを使用して COM クラスのインスタンスを作成し、そのオブジェクトのメソッドを呼び出すことによって使用されます。この規則の例外としては、**AppObj** または **GlobalMultiUse** COM クラスを含む COM オブジェクトがあります。このようなクラスは、Visual Basic 2005 のクラスのモジュール レベルのメソッドと同様です。Visual Basic 6.0 以前のバージョンでは、このようなオブジェクトのメソッドのいずれかを最初に呼び出したときに、インスタンスが暗黙的に作成されます。たとえば、Visual Basic 6.0 では、最初にインスタンスを作成しなくても、Microsoft DAO 3.6 オブジェクト ライブラリへの参照を追加し、`DBEngine` メソッドを呼び出すことができます。

```
Dim db As DAO.Database
' Open the database.
Set db = DBEngine.OpenDatabase("C:\nwind.mdb")
' Use the database object.
```

Visual Basic 2005 では、必ず COM オブジェクトのインスタンスを作成してから、それらのメソッドを使用する必要があります。Visual Basic 2005 でこれらのメソッドを使用するには、目的のクラスの変数を宣言し、New キーワードを使用して、オブジェクト変数にオブジェクトを割り当てます。クラスのインスタンスが 1 つだけ作成されていることを確認するときには、**Shared** キーワードを使用できます。

VB

```
' Class level variable.
Shared DBEngine As New DAO.DBEngine

Sub DAOOpenRecordset()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim fld As DAO.Field
    ' Open the database.
    db = DBEngine.OpenDatabase("C:\nwind.mdb")

    ' Open the Recordset.
    rst = db.OpenRecordset( _
        "SELECT * FROM Customers WHERE Region = 'WA'", _
        DAO.RecordsetTypeEnum.dbOpenForwardOnly, _
        DAO.RecordsetOptionEnum.dbReadOnly)
    ' Print the values for the fields in the debug window.
    For Each fld In rst.Fields
        Debug.WriteLine(fld.Value.ToString & ";")
    Next
    Debug.WriteLine("")
    ' Close the Recordset.
    rst.Close()
```

```
End Sub
```

イベント ハンドラで処理されないエラー

相互運用の一般的な問題の 1 つに、COM オブジェクトが発生したイベントを処理するイベント ハンドラのエラーがあります。**On Error** ステートメントまたは **Try...Catch...Finally** ステートメントを使用してエラーを明確にチェックしない限り、このようなエラーは無視されます。次の例は Microsoft ActiveX データ オブジェクト 2.8 ライブラリの COM オブジェクトへの参照を含む Visual Basic 2005 プロジェクトの一部です。

VB

```
' To use this example, add a reference to the
'     Microsoft ActiveX Data Objects 2.8 Library
' from the COM tab of the project references page.
Dim WithEvents cn As New ADODB.Connection
Sub ADODBConnect()
    cn.ConnectionString = _
    "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=C:\NWIND.MDB"
    cn.Open()
    MsgBox(cn.ConnectionString)
End Sub

Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    ADODBConnect()
End Sub

Private Sub cn_ConnectComplete( _
    ByVal pError As ADODB.Error, _
    ByRef adStatus As ADODB.EventStatusEnum, _
    ByVal pConnection As ADODB.Connection) _
    Handles cn.ConnectComplete

    '  This is the event handler for the cn_ConnectComplete event raised
    '  by the ADODB.Connection object when a database is opened.
    Dim x As Integer = 6
    Dim y As Integer = 0
    Try
        x = CInt(x / y) ' Attempt to divide by zero.
        ' This procedure would fail silently without exception handling.
    Catch ex As Exception
        MsgBox("There was an error: " & ex.Message)
    End Try
End Sub
```

この例のコードは、エラーが通知されるように記述されています。しかし、**Try...Catch...Finally** ブロックを使用せずにこのコードを実行すると、**OnError Resume Next** ステートメントを使用した場合と同じようにエラーが無視されます。エラー処理がなければ、ゼロによる除算は通知なしで失敗します。この種のエラーが未処理の例外エラーとして通知されることはないため、COM オブジェクトからのイベントを処理するイベント ハンドラで例外処理の手段を講じることが不可欠です。

COM 相互運用エラーについて

相互運用の呼び出しでは、エラー処理を組み込んでいないと、エラーが発生してもそれに関する情報がほとんど得られません。可能な場合は、必ず構造化されたエラー処理を使用して、エラー発生時の問題に関する情報が通知されるようにしてください。これは、特にアプリケーションのデバッグ時に役立ちます。次に例を示します。

VB

```
Try
    ' Place call to COM object here.
Catch ex As Exception
    ' Display information about the failed call.
End Try
```

例外オブジェクトの内容を調べることにより、エラーの説明 (HRESULT) や、COM エラーのソースなどの情報が得られます。

ActiveX コントロールの問題

Visual Basic 6.0 で動作するほとんどの ActiveX コントロールは、Visual Basic 2005 でも問題なく動作します。主な例外としては、コンテナ コントロール、つまり別のコントロールを視覚的に含むようなコントロールがあります。Visual Studio で正常に動作しない古いコントロールの例としては、次のようなものが挙げられます。

- Microsoft Forms 2.0 フレーム コントロール
- スピン コントロールとも呼ばれるアップダウン コントロール
- Sheridan タブ コントロール

サポートされていない ActiveX コントロールに関する問題の回避方法はいくつかしかありません。元のソース コードがある場合は、既存のコントロールを Visual Studio に移行できます。元のソース コードがない場合は、ソフトウェアの販売元に問い合わせて、サポートされていない ActiveX コントロールを、更新された .NET 互換バージョンのコントロールに置き換えてください。

ByRef を使用した、コントロールの ReadOnly プロパティの受け渡し

Visual Basic 2005 では、古い ActiveX コントロールの **ReadOnly** プロパティを **ByRef** パラメータとして別のプロシージャに渡すときに、"Error 0x800A017F CTL_E_SETNOTSUPPORTED" などの COM エラーが発生することがあります。Visual Basic 6.0 では、同様にプロシージャを呼び出してもエラーは発生せず、パラメータは値で渡す場合と同様に扱われます。Visual Basic 2005 では、Property **Set** プロシージャを持たないプロパティを変更しようとしているという内容のエラー メッセージが COM オブジェクトとして表示されます。

呼び出し先のプロシージャにアクセスできる場合は、**ByVal** キーワードを使用して **ReadOnly** プロパティを受け入れるパラメータを宣言することにより、このエラーを阻止できます。以下に例を示します。

VB

```
Sub ProcessParams(ByVal c As Object)
    'Use the arguments here.
End Sub
```

呼び出し先のプロシージャのソース コードにアクセスできない場合は、呼び出し元プロシージャをさらにかっこで囲むことにより、プロパティを強制的に値で渡すことができます。たとえば、Microsoft ActiveX データ オブジェクト 2.8 ライブラリの COM オブジェクトへの参照を含むプロジェクトでは、次のように指定できます。

VB

```
Sub PassByVal(ByVal pError As ADODB.Error)
    ' The extra set of parentheses around the arguments
    ' forces them to be passed by value.
    ProcessParams((pError.Description))
End Sub
```

相互運用機能を公開するアセンブリの配置

COM インターフェイスを公開するアセンブリの配置については、いくつかの考慮事項があります。たとえば、別のアプリケーションが同じ COM アセンブリを参照する場合、問題が発生する可能性があります。このような状況として一般的なのは、新しいバージョンのアセンブリをインストールした後、別のアプリケーションが依然として古いバージョンのアセンブリを使用している場合です。同じ dll を共有するアセンブリのうちのいずれかでもアンインストールすると、他のアセンブリでその dll が使用できなくなる可能性があります。

この問題を回避するには、各共有アセンブリをグローバル アセンブリ キャッシュ (GAC: Global Assembly Cache) にインストールし、そのコンポーネントに対して MergeModule を使用します。アプリケーションを GAC にインストールできない場合は、バージョン固有のサブディレクトリにある CommonFilesFolder にインストールします。

共有されないアセンブリは、呼び出し元のアプリケーションと同じディレクトリに格納してください。

参照

処理手順

チュートリアル : COM オブジェクトによる継承の実装

関連項目

タイプ ライブラリ インポータ (Tlbimp.exe)
タイプ ライブラリ エクスポータ (Tlbexp.exe)
Inherits ステートメント
MarshalAsAttribute

概念

データ型の変更点 (Visual Basic 6.0 ユーザー向け)

マージ モジュールの概要

マージ モジュール プロジェクト

グローバル アセンブリ キャッシュ

その他の技術情報

COM 相互運用

相互運用マーシャリング

.NET Framework アプリケーションにおける COM 相互運用性

COM オブジェクトと .NET Framework オブジェクトを同じアプリケーションで使用する場合は、オブジェクトをメモリに格納する方法の違いに対処する必要があります。.NET Framework オブジェクトは、共通言語ランタイムが制御するメモリである、マネージ メモリに格納されます。また、必要に応じて、共通言語ランタイムによって移動される場合があります。COM オブジェクトは、アンマネージ メモリ内に配置され、他のメモリ位置には移動しません。Visual Studio および .NET Framework には、これらのマネージ コンポーネントとアンマネージ コンポーネントの対話を制御するためのツールが用意されています。マネージ コードの詳細については、「共通言語ランタイム」を参照してください。

.NET アプリケーションで COM オブジェクトを利用する以外に、Visual Basic を使用して、COM を通じてアンマネージ コードからアクセスできるオブジェクトを作成できます。

COM と .NET Framework オブジェクトの間の対話の詳細については、このページ内のリンクを参照してください。

関連するセクション

COM 相互運用

COM オブジェクト、ActiveX コントロール、Win32 DLL、マネージ オブジェクト、COM オブジェクトの継承など、Visual Basic での COM 相互運用性に関するトピックへのリンクを提供します。

COM 相互運用ラッパー エラー

プロジェクト システムが特定のコンポーネントに対して COM 相互運用ラッパーを作成できない場合の結果とオプションについて説明します。

アンマネージ コードとの相互運用

マネージ コードとアンマネージ コードの間のやり取りに関するいくつかの問題を簡単に説明し、詳細な情報へのリンクを提供します。

COM ラッパー

マネージ コードで COM メソッドを呼び出すためのランタイム呼び出し可能ラッパー、および COM クライアントで .NET オブジェクト メソッドを呼び出すための COM 呼び出し可能ラッパーについて説明します。

高度な COM 相互運用性

ラッパー、例外、継承、スレッド処理、イベント、変換、およびマーシャリングについて説明している、COM 相互運用に関連するトピックへのリンクを提供します。

Office オブジェクト モデルを使用したアプリケーションのオートメーション

Microsoft Office と Visual Studio をビジネス アプリケーションの一部として使用する方法について説明します。

タイプ ライブラリ インポータ (Tlbimp.exe)

COM タイプ ライブラリに含まれる型定義を共通言語ランタイム アセンブリ内の同等の定義に変換するためのツールについて説明します。

Visual Basic におけるマルチスレッド

Visual Basic アプリケーションは、マルチスレッド (またはフリー スレッド) を使用して複数のタスクを同時に実行できます。マルチスレッドとはそれぞれのタスクを異なるスレッドで実行するプロセスです。マルチスレッドによって、アプリケーションのパフォーマンスと応答性を向上させることができます。

このセクションの内容

マルチスレッド アプリケーション

スレッドの作成方法と使用方法について説明します。

スレッド状態

スレッドの状態の判断方法と変更方法について説明します。

マルチスレッド プロシージャのパラメータと戻り値

マルチスレッド アプリケーションでのパラメータの受け渡し方法について説明します。

フォームとコントロールでのマルチスレッド

マルチスレッド、フォーム、およびコントロールを操作するときの特別な注意事項を説明します。

チュートリアル : マルチスレッド

単純なマルチスレッド アプリケーションの作成方法を示します。

関連するセクション

Visual Basic での高度なマルチスレッド処理

マルチスレッド アプリケーションでスレッドを管理する方法についてのトピックの一覧を示します。

コンポーネントのマルチスレッド

コンポーネント プログラミングでマルチスレッドを使用する方法についてのトピックへのリンクが用意されています。

チュートリアル : Visual Basic による簡単なマルチスレッド コンポーネントの作成

マルチスレッド コンポーネントの作成方法を示します。

マルチスレッド アプリケーション

Visual Basic では、複数のタスクを同時に実行するアプリケーションを作成できます。他のタスクを停止させる可能性のあるタスクは別のスレッドで実行できます。これは、マルチスレッドまたはフリー スレッドと呼ばれるプロセスです。

マルチスレッドを使用するアプリケーションでは、プロセッサ集中型のタスクを別のスレッドで実行している間もユーザー インターフェイスがアクティブに保たれるため、ユーザー入力に対して応答性がより高くなります。マルチスレッドでは作業負荷が増加するとスレッドを追加できるため、スケーラブルなアプリケーションを作成する場合にも役立ちます。

BackgroundWorker コンポーネントの使用

マルチスレッド アプリケーションを作成する最も信頼性の高い方法は、BackgroundWorker コンポーネントを使うことです。このクラスは、指定したメソッドのみを処理する独立したスレッドを管理します。カスタマイズ例については、「チュートリアル : マルチスレッド」を参照してください。

処理をバックグラウンドで開始するには、**BackgroundWorker** を作成し、処理の進行の報告と、処理が完了した時点の通知をするイベントを待機します。**BackgroundWorker** オブジェクトは、プログラムで作成できます。また、ツールボックスの [コンポーネント] タブからフォーム上にドラッグする方法でも作成できます。**BackgroundWorker** をフォーム デザイナで作成すると、このオブジェクトがコンポーネント トレイに表示され、そのプロパティが [プロパティ] ウィンドウに表示されます。

バックグラウンド処理のセットアップ

バックグラウンド処理をセットアップするには、DoWork イベントにイベント ハンドラを追加します。時間のかかる処理は、このイベント ハンドラの内部で呼び出します。

処理を開始するには、RunWorkerAsync を呼び出します。更新の進行状況に関する通知を受け取るには、ProgressChanged イベントを処理します。処理完了の通知を受け取るには、RunWorkerCompleted イベントを処理します。

ProgressChanged イベントと **RunWorkerCompleted** イベントは **RunWorkerAsync** メソッドの呼び出し元のスレッドに生成されるので、これらのイベントを処理するメソッドは、アプリケーションのユーザー インターフェイスにアクセスできます。ただし、**DoWork** イベント ハンドラは、バックグラウンド スレッドで動作するので、ユーザー インターフェイス オブジェクトを操作できません。

スレッドの作成と使用

アプリケーションのスレッドの動作をもっと細かく制御する必要がある場合は、スレッドそのものを管理します。ただし、マルチスレッド アプリケーションを正しく作成することは簡単ではありません。競合状態の発生によってアプリケーションが応答しなくなることや一時的なエラーが発生することがあります。詳細については、「スレッドセーフ コンポーネント」を参照してください。

Visual Basic で新規スレッドを作成するには、Thread 型の変数を宣言し、**AddressOf** ステートメントおよび新規スレッドで実行するプロシージャ名またはメソッド名を使用してコンストラクタを呼び出します。次に例を示します。

VB

```
Dim TestThread As New System.Threading.Thread(AddressOf TestSub)
```

スレッドの開始と停止

新規スレッドの実行を開始するには、次のコードに示すように Start メソッドを使用します。

VB

```
TestThread.Start()
```

スレッドの実行を停止するには、次のコードに示すように Abort メソッドを使用します

VB

```
TestThread.Abort()
```

スレッドの開始と停止の他、次のコードに示すように Sleep メソッドや Suspend メソッドを呼び出してスレッドを一時停止したり、中断しているスレッドを Resume メソッドで再開したり、**Abort** メソッドを使用してスレッドを破棄したりできます。

VB

```
  TestThread.Sleep(1000)
  TestThread.Abort()
```

スレッドのメソッド

個別のスレッドを制御するために使用できるメソッドの一部を次の表に示します。

メソッド	動作
Start	スレッドの実行を開始します。
Sleep	スレッドを一定の時間中断します。
Suspend	セーフ ポイントに達したスレッドを一時中断します。
Abort	セーフ ポイントに達したスレッドを中断します。
Resume	中断しているスレッドを再開します。
Join	別のスレッドが終了するまで現在のスレッドを待機させます。タイムアウト値を指定してこのメソッドを使用すると、指定の時間内でスレッドが終了した場合、**True** が返されます。

セーフ ポイント

これらのメソッドの大部分は自明ですが、セーフ ポイントは新しい概念です。セーフ ポイントとは、共通言語ランタイムが自動ガベージ コレクション、つまり不要な変数を解放し、メモリを再要求するプロセスを安全に実行できるコード内の場所です。スレッドの **Abort** メソッドまたは **Suspend** メソッドを呼び出すと、共通言語ランタイムがコードを分析し、スレッドの実行を停止してもよい場所を探します。

スレッドのプロパティ

スレッドには、次の表に示すように、数多くの役に立つプロパティが用意されています。

プロパティ	値
IsAlive	スレッドがアクティブな場合、値は **True** です。
IsBackground	スレッドがバックグランウンド スレッドであるかどうかを示す、ブール型を取得または設定します。バックグラウンド スレッドは、フォアグラウンド スレッドと似ていますが、プロセスの終了は回避しません。プロセスに属するフォアグラウンド スレッドがすべて終了すると、共通言語ランタイムは、まだ動作しているフォアグラウンド スレッドの **Abort** メソッドを呼び出してプロセスを終了します。
Name	スレッドの名前を取得または設定します。デバッグ時の個別のスレッドの検出に、最もよく使用されます。
Priority	オペレーティング システムがスレッドのスケジュールに優先順位を示す値を取得または設定します。
ApartmentState	特定のスレッドに使用するスレッド モデルを取得または設定します。スレッド モデルは、スレッドがアンマネージ コードを呼び出すときに重要です。
ThreadState	スレッドの状態を記述する値です。

スレッドの状態とメソッドの詳細については、「スレッド状態」を参照してください。

スレッドの優先順位

各スレッドには、実行するプロセッサ時間のスライスの大小を指定する Priority プロパティがあります。オペレーティング システムは、優先順位の高いスレッドに長いタイム スライスを割り当て、優先順位の低いスレッドに短いスライスを割り当てます。新しいスレッドは Normal 値で作成されますが、**Priority** プロパティは ThreadPriority 列挙の任意の値に変更できます。

さまざまなスレッド優先順位の詳細については、「**ThreadPriority**」を参照してください。

フォアグラウンド スレッドとバックグラウンド スレッド

フォアグラウンド スレッドは無期限に実行されますが、バックグラウンド スレッドは最後のフォアグラウンド スレッドが停止すると終了します。**IsBackground** プロパティを使用すると、スレッドのバックグラウンド ステータスを指定または変更できます。

参照

概念

スレッド状態
マルチスレッド プロシージャのパラメータと戻り値
スレッドの同期
フォームとコントロールでのマルチスレッド

その他の技術情報

コンポーネントのマルチスレッド

スレッド状態

ThreadState プロパティは、スレッドのステータスに関する情報を提供します。スレッドは複数の状態を持つことがあるため、**ThreadState** に格納されている値は ThreadState 列挙型の値の組み合わせの場合があります。たとえば、スレッドが Wait の呼び出しからブロックされ、別のスレッドが同じスレッドで Abort メソッドを呼び出した場合、スレッドは同時に WaitSleepJoin 状態と AbortRequested 状態になります。

スレッドの状態の変更

スレッドの開始後に、スレッドのメソッドを呼び出してその状態を変更できます。たとえば、System.Threading.Thread.Sleep(System.Int32) を呼び出して、スレッドをミリ秒単位で一定の時間だけ一時中断できます。**Sleep** メソッドは、パラメータとしてタイムアウトを受け取ります。これはスレッドをブロックされた状態にする数値で、単位はミリ秒です。

引数 Infinite で **Sleep** を呼び出すと、スレッドは、Interrupt を呼び出した別のスレッドに割り込まれるまで、スリープ状態になります。**Interrupt** メソッドは、指定のスレッドの待機状態を解除し、例外を生成します。

また、Suspend を呼び出すと、スレッドを一時中断することもできます。スレッドがそれ自身に **Suspend** を呼び出すと、その呼び出しは、別のスレッドが Resume を呼び出してそのスレッドを再開するまでそのスレッドをブロックします。スレッドが別のスレッドで **Suspend** を呼び出した場合、その呼び出しは非ブロッキング呼び出しであり、この呼び出しによって他のスレッドは一時中断されます。**Resume** を呼び出すと、別のスレッドの中断状態が終了し、実行が再開されます。**Sleep** とは異なり、**Suspend** はスレッドをすぐには停止しません。中断されたスレッドは、共通言語ランタイムによって安全なポイントに達したと判断されるまでは一時停止しません。

Abort メソッドは、スレッドを終了させる ThreadAbortException 例外を発生させることにより、実行中のスレッドを停止します。

これらのメソッドの詳細については、「Thread」を参照してください。

参照

関連項目

SyncLock ステートメント
System.Threading

概念

マルチスレッド アプリケーション
スレッドの同期
マルチスレッド プロシージャのパラメータと戻り値
フォームとコントロールでのマルチスレッド
デリゲートと AddressOf 演算子

その他の技術情報

コンポーネントのマルチスレッド

マルチスレッド プロシージャのパラメータと戻り値

スレッド クラスのコンストラクタは、引数を取らず値を返さないプロシージャに参照を渡す必要があるため、マルチスレッド アプリケーションでの値の受け渡しは複雑です。以下のセクションでは、異なるスレッドのプロシージャからパラメータを指定して値を返す単純な方法を示します。

マルチスレッド プロシージャのパラメータの指定

マルチスレッド メソッドの呼び出しにパラメータを指定する最良の方法は、クラスにターゲット メソッドをラップし、新規スレッドのパラメータとして機能するフィールドをそのクラスに対して定義することです。この方法の利点は、新規スレッドを開始するたびに独自のパラメータでクラスの新規インスタンスを作成できることです。たとえば、次のコードに示すように三角形の面積を計算する関数があるとします。

VB

```
Function CalcArea(ByVal Base As Double, ByVal Height As Double) As Double
    CalcArea = 0.5 * Base * Height
End Function
```

`CalcArea` 関数をラップして入力パラメータを格納するフィールドを作成するクラスを記述するコードは次のとおりです。

VB

```
Class AreaClass
    Public Base As Double
    Public Height As Double
    Public Area As Double
    Sub CalcArea()
        Area = 0.5 * Base * Height
        MsgBox("The area is: " & Area)
    End Sub
End Class
```

`AreaClass` を使用するには、次のコードに示すように `AreaClass` オブジェクトを作成し、`Base` プロパティと `Height` プロパティを設定します。

VB

```
Protected Sub TestArea()
    Dim AreaObject As New AreaClass
    Dim Thread As New System.Threading.Thread _
                      (AddressOf AreaObject.CalcArea)
    AreaObject.Base = 30
    AreaObject.Height = 40
    Thread.Start()
End Sub
```

`TestArea` プロシージャは `CalcArea` メソッドの呼び出し後に `Area` フィールドの値を確認しないことに注意してください。`CalcArea` は別のスレッドで実行されるため、`Thread.Start` を呼び出した直後に確認する場合は、`Area` フィールドが設定されているとは限りません。次のセクションでは、マルチスレッド プロシージャから値を返す適切な方法について説明します。

マルチスレッド プロシージャからの戻り値

異なるスレッドで実行されるプロシージャの場合、値を返す動作は複雑になります。プロシージャは関数にすることができず、**ByRef** 引数を使用できないからです。値を返す最も簡単な方法は、BackgroundWorker コンポーネントを使ってスレッドを管理し、タスクが完了したときにイベントを生成し、結果をイベント ハンドラで処理することです。

次の例では、別のスレッドで実行中のプロシージャからイベントを発生させて値を返します。

VB

```
Private Class AreaClass2
    Public Base As Double
    Public Height As Double
    Function CalcArea() As Double
        ' Calculate the area of a triangle.
```

```
            Return 0.5 * Base * Height
        End Function
    End Class

    Private WithEvents BackgroundWorker1 As New System.ComponentModel.BackgroundWorker

    Private Sub TestArea2()
        Dim AreaObject2 As New AreaClass2
        AreaObject2.Base = 30
        AreaObject2.Height = 40

        ' Start the asynchronous operation.
        BackgroundWorker1.RunWorkerAsync(AreaObject2)
    End Sub

    ' This method runs on the background thread when it starts.
    Private Sub BackgroundWorker1_DoWork(ByVal sender As Object, _
        ByVal e As System.ComponentModel.DoWorkEventArgs) _
        Handles BackgroundWorker1.DoWork

        Dim AreaObject2 As AreaClass2 = CType(e.Argument, AreaClass2)
        ' Return the value through the Result property.
        e.Result = AreaObject2.CalcArea()
    End Sub

    ' This method runs on the main thread when the background thread finishes.
    Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, _
        ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
        Handles BackgroundWorker1.RunWorkerCompleted

        ' Access the result through the Result property.
        Dim Area As Double = CDbl(e.Result)
        MsgBox("The area is: " & Area)
    End Sub
```

スレッド プールのスレッドにパラメータと戻り値を用意するには、QueueUserWorkItem メソッドの省略可能な状態オブジェクト変数 (**ByVal**) を使用します。スレッド タイマのスレッドもこの目的で状態オブジェクトをサポートしています。スレッド プールおよびスレッド タイマの詳細については、「スレッド プール」および「スレッド タイマ」を参照してください。

参照

概念

スレッド状態
スレッド プール
スレッドの同期
イベントとイベント ハンドラ
マルチスレッド アプリケーション
フォームとコントロールでのマルチスレッド
デリゲートと AddressOf 演算子

その他の技術情報

コンポーネントのマルチスレッド

フォームとコントロールでのマルチスレッド

マルチスレッドはプロシージャとクラス メソッドの実行に最適ですが、フォームとコントロールでも使用できます。その場合は、以下の点に注意してください。

- コントロールのメソッドは、できる限りそのコントロールが作成されたスレッドだけで実行してください。コントロールのメソッドを別のスレッドから呼び出す必要がある場合は、Invoke を使用してメソッドを呼び出す必要があります。
- **SyncLock** ステートメントを使用してコントロールまたはフォームを操作するスレッドをロックしないでください。コントロールとフォームのメソッドは呼び出しプロシージャをコールバックする場合があるため、誤ってデッドロック状態が発生することがあります。デッドロック状態になると 2 つのスレッドが互いにロックの解放を待機するため、アプリケーションが停止します。

参照

関連項目

Invoke
InvokeRequired

概念

スレッド状態
マルチスレッド プロシージャのパラメータと戻り値
マルチスレッド アプリケーション

その他の技術情報

コンポーネントのマルチスレッド

チュートリアル：マルチスレッド

このチュートリアルでは、テキスト ファイルで語句を検索する、マルチスレッド アプリケーションの作成方法を説明します。次の方法を学ぶことができます。

- BackgroundWorker コンポーネントから呼び出し可能なメソッドを含むクラスの定義。
- **BackgroundWorker** コンポーネントで発生したイベントの処理。
- メソッドを実行するための **BackgroundWorker** コンポーネントの開始。
- **BackgroundWorker** コンポーネントを停止する `Cancel` ボタンの実装。

このトピックのコード例を作成するには

1. 新しい Visual Basic Windows Application プロジェクトを開き、`Form1` の名前でフォームを作成します。
2. `Form1` に 2 つのボタンと 4 つのテキスト ボックスを追加します。
3. 次の表のとおりにオブジェクトに名前を付けます。

オブジェクト	プロパティ	設定
1 番目のボタン	**Name**, **Text**	Start、Start
2 番目のボタン	**Name**, **Text**	Cancel、Cancel
1 番目のテキスト ボックス	**Name**, **Text**	SourceFile、""
2 番目のテキスト ボックス	**Name**, **Text**	CompareString、""
3 番目のテキスト ボックス	**Name**, **Text**	WordsCounted、"0"
4 番目のテキスト ボックス	**Name**, **Text**	LinesCounted、"0"

4. 各テキスト ボックスの横にラベルを追加します。次の表に示すように、各ラベルの **Text** プロパティを設定します。

オブジェクト	プロパティ	設定
1 番目のラベル	**Text**	Source File
2 番目のラベル	**Text**	Compare String
3 番目のラベル	**Text**	Matching Words
4 番目のラベル	**Text**	Lines Counted

5. [ツールボックス] の [コンポーネント] にある**BackgroundWorker** コンポーネントをフォームに追加します。追加したコンポーネントがフォームのコンポーネント トレイに表示されます。
6. `BackgroundWorker1` オブジェクトに対して以下のプロパティを設定します。

プロパティ	設定
WorkerReportsProgress	True
WorkerSupportsCancellation	True

7.

個別のスレッドで動作するメソッドを定義するには

1. [プロジェクト] メニューの [クラスの追加] を選択し、プロジェクトにクラスを追加します。[新しい項目の追加] ダイアログ ボックスが表示されます。
2. テンプレート ウィンドウの [クラス] を選択し、名前フィールドに「`Words.vb`」と入力します。
3. [追加] をクリックします。`Words` クラスが表示されます。
4. `Words` クラスの一番上の **Class** ステートメントの上に、**Option Compare** ステートメントを追加します。

   VB

```
Option Compare Text ' Case insensitive search.
' Use Option Compare Binary for case sensitive search.
```

5. `Words` クラスに次のコードを追加します。

   VB

```
Public Class Words
    ' Object to store the current state, for passing to the caller.
    Public Class CurrentState
        Public LinesCounted As Integer
        Public WordsMatched As Integer
    End Class

    Public SourceFile As String
    Public CompareString As String
    Private WordCount As Integer = 0
    Private LinesCounted As Integer = 0

    Public Sub CountWords( _
        ByVal worker As System.ComponentModel.BackgroundWorker, _
        ByVal e As System.ComponentModel.DoWorkEventArgs _
    )
        ' Initialize the variables.
        Dim mystream As System.IO.StreamReader = Nothing
        Dim line As String = ""

        If CompareString Is Nothing Or _
                CompareString = System.String.Empty Then
            Throw New Exception("CompareString not specified.")
        End If

        Try
            ' Open a new stream.
            mystream = My.Computer.FileSystem.OpenTextFileReader(SourceFile)
            ' Do until the stream returns Nothing at end of file.
            Do Until line Is Nothing
                If worker.CancellationPending Then
                    e.Cancel = True
                    Exit Do
                Else
                    line = mystream.ReadLine
                    WordCount += CountInString(line, CompareString)
                    LinesCounted += 1 ' Increment line count.

                    ' Raise an event so the form can monitor progress.
                    Dim state As New CurrentState
                    state.LinesCounted = LinesCounted
                    state.WordsMatched = WordCount
```

```
                    worker.ReportProgress(0, state)
                End If
            Loop
        Finally
            If mystream IsNot Nothing Then
                ' Close the file.
                mystream.Close()
            End If
        End Try
    End Sub

    Private Function CountInString( _
        ByVal SourceString As String, _
        ByVal CompareString As String _
    ) As Integer
        ' This function counts the number of times
        ' a word is found in a line.
        If SourceString Is Nothing Then
            Return 0
        End If

        Dim regex As New System.Text.RegularExpressions.Regex( _
            System.Text.RegularExpressions.Regex.Escape(CompareString))
        Dim matches As System.Text.RegularExpressions.MatchCollection
        matches = regex.Matches(SourceString)
        Return matches.Count
    End Function
End Class
```

スレッドからイベントを処理するには

- メイン フォームに次のイベント ハンドラを追加します。

VB

```
Private Sub BackgroundWorker1_RunWorkerCompleted( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
    Handles BackgroundWorker1.RunWorkerCompleted

    ' This event handler is called when the background thread finishes.
    ' This method runs on the main thread.

    If e.Error IsNot Nothing Then
        MsgBox("Error: " & e.Error.Message)
    ElseIf e.Cancelled Then
        MsgBox("Word counting canceled.")
    Else
        MsgBox("Finished counting words.")
    End If
End Sub

Private Sub BackgroundWorker1_ProgressChanged( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.ProgressChangedEventArgs) _
    Handles BackgroundWorker1.ProgressChanged

    ' This event handler is called after the background thread
```

```
    ' reads a line from the source file.
    ' This method runs on the main thread.

    Dim state As Words.CurrentState = _
        CType(e.UserState, Words.CurrentState)
    Me.LinesCounted.Text = state.LinesCounted.ToString
    Me.WordsCounted.Text = state.WordsMatched.ToString
End Sub
```

WordCount メソッドを実行する新しいスレッドを開始して呼び出すには

1. プログラムに次のプロシージャを追加します。

   VB

```
Private Sub BackgroundWorker1_DoWork( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.DoWorkEventArgs) _
    Handles BackgroundWorker1.DoWork

    ' This event handler is where the actual work is done.
    ' This method runs on the background thread.

    ' Get the BackgroundWorker object that raised this event.
    Dim worker As System.ComponentModel.BackgroundWorker
    worker = CType(sender, System.ComponentModel.BackgroundWorker)

    ' Get the Works object and call the main method.
    Dim WC As Words = CType(e.Argument, Words)
    WC.CountWords(worker, e)
End Sub

Sub StartThread()
    ' This method runs on the main thread.

    Me.WordsCounted.Text = "0"

    ' Initialize the object that the background worker calls.
    Dim WC As New Words
    WC.CompareString = Me.CompareString.Text
    WC.SourceFile = Me.SourceFile.Text

    ' Start the asynchronous operation.
    BackgroundWorker1.RunWorkerAsync(WC)
End Sub
```

2. フォームの `Start` ボタンから `StartThread` メソッドを呼び出します。

   VB

```
Private Sub Start_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Start.Click

    StartThread()
End Sub
```

スレッドを停止する Cancel ボタンを実装するには

- `Cancel` ボタンの `Click` イベント ハンドラから `StopThread` プロシージャを呼び出します。

  VB

```
Private Sub Cancel_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Cancel.Click

    ' Cancel the asynchronous operation.
    Me.BackgroundWorker1.CancelAsync()
End Sub
```

テスト

アプリケーションをテストして、正常に動作することを確認します。

アプリケーションをテストするには

1. F5 キーを押してアプリケーションを実行します。
2. フォームが表示されたら、テストするファイルのファイル パスを `sourceFile` ボックスに入力します。たとえば、テスト ファイルの名前が Test.txt の場合は、「C:\Test.txt」と入力します。
3. 2 番目のテキスト ボックスに、アプリケーションがテキスト ファイル内を検索する語句またはフレーズを入力します。
4. [`Start`] をクリックします。`LinesCounted` ボックスで、直ちにインクリメントが開始されます。完了すると、アプリケーションは "Finished Counting" というメッセージを表示します。

Cancel ボタンをテストするには

1. F5 キーを押して、アプリケーションを開始します。前の手順で説明したように、ファイル名と検索語句を入力します。動作が完了する前にプロシージャをキャンセルするだけの時間があるように、サイズの大きいファイルを選択します。
2. `Start` をクリックして、アプリケーションを起動します。
3. [`Cancel`] をクリックします。アプリケーションは直ちにカウントを中止します。

次の手順

このアプリケーションには、基本的なエラー処理が含まれています。空白の検索文字列が検出されます。語句の数やカウントする行数が最大値を超えた場合など、その他のエラーを処理することで、このプログラムの信頼性をより高くできます。

参照

処理手順

チュートリアル : Visual Basic による簡単なマルチスレッド コンポーネントの作成

その他の技術情報

Visual Basic におけるマルチスレッド

リファレンス (Visual Basic)

このセクションには、Visual Basic プログラミングに関するさまざまなリファレンスへのリンクがあります。

このセクションの内容

Visual Basic リファレンス

Visual Basic 言語のさまざまな側面に関するリファレンス情報を提供します。

Visual Basic のコマンド ライン ツール

Visual Basic のコマンド ライン コンパイラ、コンパイラ オプション、およびキーワード アップグレード ツールについて説明します。

Visual Basic のアップグレード リファレンス

アプリケーションで Visual Basic を以前のバージョンから現在のバージョンへアップグレードするための情報へのリンクを示します。

.NET Framework の参照情報

.NET Framework クラス ライブラリの使用方法に関する情報へのリンクがあります。

Visual Basic 言語仕様

完全な Visual Basic 言語仕様へのリンクを示します。ここには、この言語のあらゆる側面に関する詳細な情報が掲載されています。

関連するセクション

一般的なユーザー インターフェイス要素 (Visual Studio)

Visual Studio のダイアログ ボックスとウィンドウに関するトピックです。

XML スキーマのリファレンス

Visual Studio で使用可能な、さまざまな XML スキーマに関するトピックへのリンクを示します。

各言語の比較

複数のプログラミング言語間で、キーワード、プログラミング概念、データ型、演算子、コントロール、およびプログラミング可能なオブジェクトを比較します。

オートメーションと機能拡張のリファレンス

共有コンポーネントと言語固有コンポーネントの両方について Visual Studio でのオートメーションと機能拡張に関するトピックへのリンクを示しています。

Visual Basic リファレンス

ここでは、Visual Basic 言語のさまざまな側面に関するリファレンス情報を提供します。

このセクションの内容

表記規則とコード規則

Visual Basic ドキュメントでのキーワード、プレースホルダ、およびその他の言語要素の表記法について説明します。

Visual Basic 言語のキーワード

Visual Basic 言語の予約済みキーワードと予約済みでないキーワードについて説明します。

Visual Basic ランタイム ライブラリのメンバ

Microsoft.VisualBasic 名前空間のクラスとモジュールを一覧で示し、それぞれのメンバ関数、メソッド、プロパティ、定数、および列挙値へのリンクを提供します。

キーワードとメンバ (タスク別)

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを目的と使用方法ごとにまとめて示します。

Visual Basic の属性

Visual Basic で利用できる属性について説明します。

定数と列挙型 (Visual Basic)

Visual Basic で利用できる定数と列挙値について説明します。

データ型の概要 (Visual Basic)

Visual Basic で利用できるデータ型について説明します。

ディレクティブ (Visual Basic)

Visual Basic で利用できるコンパイラ ディレクティブについて説明します。

関数 (Visual Basic)

Visual Basic で利用できるランタイム関数について説明します。

キーワード (Visual Basic)

Visual Basic で利用できる言語キーワードについて説明します。

モジュール (Visual Basic)

Visual Basic で利用できるモジュールと、それぞれのメンバについて説明します。

オブジェクト (Visual Basic)

Visual Basic で利用できるオブジェクトと、それぞれのメンバについて説明します。

演算子 (Visual Basic)

Visual Basic で利用できる演算子について説明します。

プロパティ (Visual Basic)

Visual Basic で利用できるプロパティについて説明します。

ステートメント (Visual Basic)

Visual Basic で利用できる宣言と実行ステートメントについて説明します。

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

Visual Basic コード エディタ内で IntelliSenseTM が提供されているドキュメント コメントについて説明します。

エラー メッセージ (Visual Basic)

Visual Basic コンパイラとランタイム エラー メッセージ、およびそれらの処理方法に関するヘルプの一覧を示します。

関連するセクション

Visual Basic

Visual Basic 言語の全分野についての包括的なヘルプです。

Visual Basic コンパイラ

Visual Studio 統合開発環境 (IDE: integrated development environment) でプログラムをコンパイルするのに代わる方法として、コマンド ライン コンパイラの使用方法を説明します。

表記規則とコード規則

Visual Basic のヘルプは、次の表記規則とコード規則に従って記述されています。

表記規則

表記	説明
Sub, **If**, **ChDir**, **Print**, **True**, **Debug**	Visual Basic 特有のキーワードやランタイム メンバは、太字で先頭を大文字にして表記します。
Module ステートメント	下線付きで、色付きフォントの語および句はハイパーリンクで、クリックすると関連するヘルプ ページにすぐに移動できます。
object, *variablename*, *argumentlist*	構文内の斜体は、ファイル名など、実際に入力するときには、該当する文字に置き換える必要のあるプレースホルダを示します。
[Shadows], [*expressionlist*]	構文内の角かっこ ([ ]) は、囲まれた項目が省略できることを示します。
{ **Public** | **Friend** | **Private** }	構文内の中かっこ ({ }) と縦棒の組み合わせは、縦棒で区切られた複数の項目から 1 つを選択する必要があることを示します。1 つの項目だけを選択する必要があります。
[**Protected** | **Friend**]	構文内の角かっこ ([ ]) と縦棒の組み合わせは、2 つ以上の独立したオプションを示します。これらの項目を任意に組み合わせて選択するか、または何も選択しないこともできます。
[{ **ByVal** | **ByRef** }]	構文内の中かっこを囲む角かっこ ([{ }]) と縦棒の組み合わせは、項目を 1 つしか選択できないことを示します。ただし、項目を何も選択しないこともできます。
membername $_1$, *membername*$_2$, *membername*$_3$	添字は、同じプレースホルダの複数のインスタンスを区別します。
memerbname1 ... *membernamen*	構文内の省略記号 (...) は、省略記号の直前の種類の項目が無限に続くことを示します。 コードでは、わかりやすくするために省略されたコードを示します。
ESC、ENTER	大文字の単語は、キーボードのキー名やキー シーケンスを示します。
Alt + **F1**、**Ctrl** + **R**	プラス記号 (+) は、キーの組み合わせを表します。たとえば、**Alt** + **F1** は **Alt** キーを押しながら、**F1** キーを押すことを表します。

コード規則

サンプル コード	説明
`sampleString = "Hello, world!"` このステートメントは、`sampleString` の値に "Hello, world!" を設定します。	固定ピッチ フォントは、コード サンプルおよび解説文に含まれるコード要素に使用します。
`' This is a comment.` `REM This is also a comment.`	アポストロフィ (') や REM キーワードは、コードのコメントの先頭を示します。

`sampleVar = "This is an " _`
`& "example" _`
`& " of how to continue code."`
	行末の空白とアンダースコア (_) は、ソース コードのステートメントが継続することを示します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

Visual Basic 言語のキーワード

Visual Basic ランタイム ライブラリのメンバ

概念

Visual Basic の名前付け規則

コード内のコメント

その他の技術情報

Visual Basic リファレンス

Visual Basic 言語のキーワード

次の表は、Visual Basic 言語のすべてのキーワードの一覧です。

予約済みのキーワード

次のキーワードは予約済みであり、プログラミングの要素 (変数やプロシージャなど) の名前として使用できません。ただし、名前を角かっこ ([ ]) で囲むと、この制約を回避できます。詳細については、「宣言された要素の名前」の「エスケープされた名前」を参照してください。

メモ：
エスケープされた名前を使用すると、コードの可読性が低下し、検出しにくい微妙なエラーの原因となる可能性があるため、使用を避けてください。

AddHandler	AddressOf	Alias	And
AndAlso	As	Boolean	ByRef
Byte	ByVal	Call	Case
Catch	CBool	CByte	CChar
CDate	CDec	CDbl	Char
CInt	Class	CLng	CObj
Const	Continue	CSByte	CShort
CSng	CStr	CType	CUInt
CULng	CUShort	Date	Decimal
Declare	Default	Delegate	Dim
DirectCast	Do	Double	Each
Else	ElseIf	End	**EndIf**
Enum	Erase	Error	Event
Exit	False	Finally	For
Friend	Function	Get	GetType
Global	**GoSub**	GoTo	Handles
If	Implements	Imports	In
Inherits	Integer	Interface	Is
IsNot	**Let**	Lib	Like
Long	Loop	Me	Mod

Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	Narrowing	New
Next	Not	Nothing	NotInheritable
NotOverridable	Object	Of	On
Operator	Option	Optional	Or
OrElse	Overloads	Overridable	Overrides
ParamArray	Partial	Private	Property
Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	RemoveHandler	Resume
Return	SByte	Select	Set
Shadows	Shared	Short	Single
Static	Step	Stop	String
Structure	Sub	SyncLock	Then
Throw	To	True	Try
TryCast	TypeOf	**Variant**	**Wend**
UInteger	ULong	UShort	Using
When	While	Widening	With
WithEvents	WriteOnly	Xor	#Const
#Else	#ElseIf	#End	#If
-	&	&=	*
*=	/	/=	\
\=	^	^=	+
+=	=	-=	

メモ：

EndIf、**GoSub**、**Let**、**Variant**、および **Wend** は、Visual Basic では使用されなくなりましたが、予約済みのキーワードとして残っています。

予約されていないキーワード

次のキーワードは予約されていません。したがって、プログラミングの要素の名前として使用できます。ただし、コードの可読性が低下し、検出しにくい微妙なエラーの原因となる可能性があるため、これらのキーワードを要素の名前として使用することは避けてください。

Ansi	Assembly	Auto	Binary
Compare	Custom	Explicit	IsFalse
IsTrue	Mid	Off	Preserve
Strict	Text	Unicode	Until
#ExternalSource	#Region		

参照

関連項目

Visual Basic ランタイム ライブラリのメンバ
キーワードとメンバ (タスク別)
各言語のキーワードの比較

Visual Basic ランタイム ライブラリのメンバ

Microsoft.VisualBasic 名前空間には、Visual Basic ランタイム ライブラリを構成するクラス、モジュール、定数、および列挙体が含まれています。これらのライブラリ メンバには、コードに使用できるプロシージャ、プロパティ、および定数値が用意されています。各モジュールおよび各クラスは、特定のカテゴリの機能を表します。

Microsoft.VisualBasic.Collection クラス

Add	Clear	Contains	Count
GetEnumerator	Item	Remove	

Microsoft.VisualBasic.ComClassAttribute クラス

ClassID	EventID	InterfaceID	InterfaceShadows

Microsoft.VisualBasic.ControlChars クラス

Back	Cr	CrLf	FormFeed
Lf	NewLine	NullChar	Quote
Tab	VerticalTab		

Microsoft.VisualBasic.Conversion モジュール

ErrorToString	Fix	Hex	Int
Oct	Str	Val	

Microsoft.VisualBasic.DateAndTime モジュール

DateAdd	DateDiff	DatePart	DateSerial
DateString	DateValue	Day	Hour
Minute	Month	MonthName	Now
Second	TimeOfDay	Timer	TimeSerial
TimeString	TimeValue	Today	WeekDay
WeekDayName	Year		

Microsoft.VisualBasic.ErrObject クラス

Clear	Description	Erl	GetException
HelpContext	HelpFile	LastDLLError	Number
Raise	Source		

Microsoft.VisualBasic.FileSystem モジュール

ChDir	ChDrive	CurDir	Dir
EOF	FileAttr	FileClose	FileCopy
FileDateTime	FileGet	FileGetObject	FileLen

FileOpen	FilePut	FilePutObject	FileWidth
FreeFile	GetAttr	Input	InputString
Kill	LineInput	Loc	Lock
LOF	MkDir	Print	PrintLine
Rename	Reset	RmDir	Seek
SetAttr	SPC	Tab	Unlock
Write	WriteLine		

Microsoft.VisualBasic.Financial モジュール

DDB	FV	IPmt	IRR
MIRR	NPer	NPV	Pmt
PPmt	PV	Rate	SLN
SYD			

Microsoft.VisualBasic.Globals モジュール

ScriptEngine	ScriptEngineBuildVersion	ScriptEngineMajorVersion	ScriptEngineMinorVersion

Microsoft.VisualBasic.Information モジュール

Err	IsArray	IsDate	IsDBNull
IsError	IsNothing	IsNumeric	IsReference
LBound	QBColor	RGB	SystemTypeName
TypeName	UBound	VarType	VbTypeName

Microsoft.VisualBasic.Interaction モジュール

AppActivate	Beep	CallByName	Choose
Command	CreateObject	DeleteSetting	Environ
GetAllSettings	GetObject	GetSetting	IIf
InputBox	MsgBox	Partition	SaveSetting
Shell	Switch		

Microsoft.VisualBasic.Strings モジュール

Asc	AscW	Chr	ChrW
Filter	Format	FormatCurrency	FormatDateTime
FormatNumber	FormatPercent	GetChar	InStr

InStrRev	Join	LCase	Left
Len	LSet	LTrim	Mid
Replace	Right	RSet	RTrim
Space	Split	StrComp	StrConv
StrDup	StrReverse	Trim	UCase

Microsoft.VisualBasic.VBFixedArrayAttribute クラス

Bounds	Length		

Microsoft.VisualBasic.VBFixedStringAttribute クラス

Length			

Microsoft.VisualBasic.VbMath モジュール

Randomize	Rnd		

Microsoft.VisualBasic の定数と列挙体

Microsoft.VisualBasic 名前空間には、Visual Basic ランタイム ライブラリの一部である定数と列挙体が用意されています。これらの定数値をコードに使用できます。各列挙体は、特定のカテゴリの機能を表します。詳細については、「定数と列挙型 (Visual Basic)」を参照してください。

参照

関連項目

Visual Basic 言語のキーワード

キーワードとメンバ (タスク別)

その他の技術情報

Visual Studio の Visual Basic ランタイムの名前空間

定数と列挙型 (Visual Basic)

キーワードとメンバ (タスク別)

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

カテゴリ	説明
配列の概要	配列の作成、定義、および使用
Collection オブジェクトの概要	コレクションの使用
コンパイラ ディレクティブの概要	コンパイラの動作の制御
制御フローの概要	ループとプロシージャのフロー制御
変換の概要	数値とデータ型の変換
データ型の概要	データ型とオブジェクトのサブタイプ
日付と時刻の概要	日付式と時刻式の変換と使用
宣言と定数の概要	プログラミング要素の宣言と定数の定義
ディレクトリとファイルの概要	ファイル システムの制御とファイルの処理
エラーの概要	エラー値のトラップと取得
財務処理の概要	財務計算の実行
入出力の概要	入力の受け取りおよび出力の表示と印刷
情報と対話の概要	他のアプリケーションの起動とイベントの処理
数値演算の概要	三角関数やその他の数値演算の実行
My の参照	**My** について説明します。これは、アプリケーションを実行しているコンピュータや、現在のアプリケーション、アプリケーションのリソース、アプリケーションの設定などに対してプログラミングを行うときによく使用されるメソッド、プロパティ、イベントへの直観的なアクセスを提供することで、アプリケーション開発を容易かつ迅速にするための機能です。
演算子の概要	式の比較およびその他の演算
レジストリの概要	プログラム設定の削除、読み取り、および保存
文字列操作の概要	文字列および文字列型のデータの操作

参照

関連項目

Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

配列の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

操作	言語要素
配列かどうかの確認	IsArray
配列の宣言と初期化	Dim、Private、Public、および ReDim
配列の上限の確認	LBound および UBound
配列の再初期化	Erase および ReDim

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

Collection オブジェクトの概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

操作	言語要素
Collection オブジェクトの作成	Collection
コレクションへの項目の追加	Add
コレクションのオブジェクトの削除	Remove
コレクション内の項目の参照	Item
IEnumerator インターフェイスに対する参照の取得	GetEnumerator

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

コンパイラ ディレクティブの概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

処理	言語要素
コンパイラ定数の定義	#Const ディレクティブ
選択したコード ブロックのコンパイル	#If...Then...#Else ディレクティブ
コード セクションの折りたたみと非表示	#Region ディレクティブ
ソース行とソース外部のテキスト間のマップの指定	#ExternalSource ディレクティブ

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

制御フローの概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

動作	言語要素
分岐	GoTo および On Error
プログラムの終了または一時停止	End、Exit、および Stop
ループ	Do...Loop、For...Next、For Each...Next、While...End While、および With
条件判断	Choose、If...Then...Else、Select Case、および Switch
プロシージャの使用	Call、Function、Property、および Sub

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

変換の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

動作	言語要素
ANSI 値から文字列への変換	Chr および ChrW
文字列をすべて小文字または大文字に変換	Format、LCase および UCase
日付をシリアル値に変換	DateSerial および DateValue
10 進数を 16 進数または 8 進数に変換	Hex および Oct
数値を文字列に変換	Format および Str
データ型間の変換	CBool、CByte、CDate、CDbl、CDec、CInt、CLng、CSng、CShort、CStr、CType、Fix、および Int
日付を日、月、曜日、または年に変換	Day、Month、Weekday、および Year
時刻を時、分、または秒に変換	Hour、Minute、および Second
文字列を ASCII 値に変換	Asc および AscW
文字列を数値に変換	Val
時刻をシリアル値に変換	TimeSerial および TimeValue

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

データ型の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

動作	言語要素
データ型間の変換	CBool、CByte、CChar、CDate、CDbl、CDec、CInt、CLng、CObj、CShort、CSng、CStr、Fix、および Int
組み込みデータ型の設定	Boolean、Byte、Char、Date、Decimal、Double、Integer、Long、Object、Short、Single、および String
データ型の確認	IsArray、IsDate、IsDbNull、IsError、IsNothing、IsNumeric、および IsReference

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

日付と時刻の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

操作	言語要素
現在の日付または時刻の取得	Today、Now、および TimeOfDay
日付計算の実行	DateAdd、DateDiff、および DatePart
日付の取得	DateSerial、DateValue、MonthName、および WeekDayName
時刻の取得	TimeSerial および TimeValue
日付または時刻の設定	DateString、TimeOfDay、TimeString、および Today
処理時間の計測	Timer

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

宣言と定数の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

動作	言語要素
値の代入	Get および Property
変数や定数の宣言	Const、Dim、Private、Protected、Public、Shadows、Shared、および Static
クラス、デリゲート、列挙体、モジュール、名前空間、または構造体の宣言	Class、Delegate、Enum、Module、Namespace、および Structure
オブジェクトの作成	CreateObject、GetObject、および New
オブジェクトに関する情報の取得	GetType、IsArray、IsDate、IsDbNull、IsError、IsNothing、IsNumeric、IsReference、SystemTypeNa、TypeName、VarType、および VbTypeName
現在のオブジェクトの参照	Me
変数の明示的宣言を要求	Option Explicit および Option Strict
イベントの処理	AddHandler、Event、RaiseEvent、および RemoveHandler
継承の実装	Inherits、MustInherit、MustOverride、MyBase、MyClass、New、NotInheritable、NotOverridable、Overloads、Overridable、および Overrides

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

ディレクトリとファイルの概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

My 機能を使用すると、これらのメンバを使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

処理	言語要素
ディレクトリまたはフォルダの変更	ChDir
ドライブの変更	ChDrive
ファイルのコピー	FileCopy
ディレクトリまたはフォルダの作成	MkDir
ディレクトリまたはフォルダの削除	RmDir
ファイル、ディレクトリ、またはフォルダの名前変更	Rename
現在のパスの取得	CurDir
ファイルのタイム スタンプの取得	FileDateTime
ファイル、ディレクトリ、およびラベルの属性の取得	GetAttr
ファイルのサイズの取得	FileLen
ファイル名またはボリューム ラベルの取得	Dir
ファイルの属性情報の設定	SetAttr

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

その他の技術情報

Visual Basic でのファイルの読み取り
Visual Basic でのファイルへの書き込み
Visual Basic でのファイルおよびディレクトリの作成、削除、および移動
Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

エラーの概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

動作	言語要素
ランタイム エラーの生成	Clear、Error、および Raise
例外の取得	GetException
エラー情報の提供	Err
ランタイム エラーのトラップ	On Error、Resume、および Try...Catch...Finally
エラーになった行番号の提供	Erl
システム エラー コードの提供	LastDLLError

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

財務処理の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

処理	言語要素
減価償却費の計算	DDB、SLN、および SYD
将来価値の計算	FV
利率の計算	Rate
内部利益率の計算	IRR および MIRR
期間の計算	NPer
支払い額の計算	IPmt、Pmt、および PPmt
正味現在価値の計算	NPV および PV

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

情報と対話の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

動作	言語要素
他のプログラムの実行	AppActivate および Shell
メソッドやプロパティの呼び出し	CallByName
コンピュータのビープ音	Beep
コマンド ライン文字列の提供	Command
COM オブジェクトの操作	CreateObject および GetObject
色情報の取得	QBColor および RGB
ダイアログ ボックスの制御	InputBox および MsgBox

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

入出力の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

動作	言語要素
ファイルへのアクセスまたはファイルの作成	FileOpen
ファイルのクローズ	FileClose および Reset
出力形式の制御	Format、Print、SPC、TAB、および FileWidth
ファイルのコピー	FileCopy
ファイルに関する情報の取得	EOF、FileAttr、FileDateTime、FileLen、FreeFile、GetAttr、Loc、LOF、および Seek
コントロール ダイアログ ボックスを使用して、情報をユーザーから取得するか、ユーザーに提供します。	InputBox および MsgBox
ファイルの管理	Dir、Kill、Lock、および Unlock
ファイルのデータの読み取り	FileGet、FileGetObject、Input、InputString、および LineInput
ファイル サイズの取得	FileLen
ファイル属性の設定または取得	FileAttr、GetAttr、および SetAttr
ファイルの読み書きの位置の設定	Seek
ファイルへの書き込み	FilePut、FilePutObject、Print、Write 関数、WriteLine 関数および Write 関数、WriteLine 関数

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

数値演算の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

処理	言語要素
三角関数の計算	Atn、Cos、Sin、および Tan
一般的な計算	Exp、Log、および Sqr
乱数の生成	Randomize および Rnd
絶対値の取得	Abs
式の符号の取得	Sgn
数値の変換	Fix および Int

参照

関連項目

数値演算関数の導出 (Visual Basic)
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

数値演算関数の導出 (Visual Basic)

次の表に、Visual Basic に組み込まれていない数値演算関数の導出式を示します。これらの数値演算関数は、いずれも Visual Basic の組み込み関数から導くことができます。

関数	組み込み関数を使った導出式
セカント (Sec(x))	*1 / Cos(x)*
コセカント (Csc(x))	*1 / Sin(x)*
コタンジェント (Ctan(x))	*1 / Tan(x)*
アークサイン (Asin(x))	*Atan(x / Sqrt(-x * x + 1))*
アークコサイン (Acos(x))	*Atan(-x / Sqrt(-x * x + 1)) + 2 * Atan(1)*
アークセカント (Asec(x))	*2 * Atan(1) – Atan(Sign(x) / Sqrt(x * x – 1))*
アークコセカント (Acsc(x))	*Atan(Sign(x) / Sqrt(x * x – 1))*
アークコタンジェント (Acot(x))	*2 * Atan(1) - Atan(x)*
ハイパーボリック サイン (Sinh(x))	*(Exp(x) – Exp(-x)) / 2*
ハイパーボリック コサイン (Cosh(x))	*(Exp(x) + Exp(-x)) / 2*
ハイパーボリック タンジェント (Tanh(x))	*(Exp(x) – Exp(-x)) / (Exp(x) + Exp(-x))*
ハイパーボリック セカント (Sech(x))	*2 / (Exp(x) + Exp(-x))*
ハイパーボリック コセカント (Csch(x))	*2 / (Exp(x) – Exp(-x))*
ハイパーボリック コタンジェント (Coth(x))	*(Exp(x) + Exp(-x)) / (Exp(x) – Exp(-x))*
ハイパーボリック アークサイン (Asinh(x))	*Log(x + Sqrt(x * x + 1))*
ハイパーボリック アークコサイン (Acosh(x))	*Log(x + Sqrt(x * x – 1))*
ハイパーボリック アークタンジェント (Atanh(x))	*Log((1 + x) / (1 – x)) / 2*
ハイパーボリック アークセカント (AsecH(x))	*Log((Sqrt(-x * x + 1) + 1) / x)*
ハイパーボリック アークコセカント (Acsch(x))	*Log((Sign(x) * Sqrt(x * x + 1) + 1) / x)*
ハイパーボリック アークコタンジェント (Acoth(x))	*Log((x + 1) / (x – 1)) / 2*

必要条件

名前空間 : Math

アセンブリ : mscorlib (mscorlib.dll 内)

参照
関連項目
数値演算関数 (Visual Basic)

My の参照

My 機能を使うと、頻繁に使用するメソッド、プロパティ、およびイベントにわかりやすい方法でアクセスできるため、プログラムを短い時間で簡単に作成できます。次の表に、**My** に含まれるオブジェクトと、各オブジェクトを使って実行できる操作をまとめます。

操作	オブジェクト
アプリケーションの情報とサービスにアクセスします。	My.Application オブジェクト
ホスト コンピュータとそのリソース、サービス、およびデータにアクセスします。	My.Computer オブジェクト
現在のプロジェクトのフォームにアクセスします。	My.Forms オブジェクト
アプリケーション ログにアクセスします。	My.Log オブジェクト
現在の Web 要求にアクセスします。	My.Request オブジェクト
リソース要素にアクセスします。	My.Resources オブジェクト
現在の Web 応答にアクセスします。	My.Response オブジェクト
ユーザーとアプリケーションのレベル設定にアクセスします。	My.Settings オブジェクト
現在のユーザーのセキュリティ コンテキストにアクセスします。	My.User オブジェクト
現在のプロジェクトから参照される XML Web サービスにアクセスします。	My.WebServices オブジェクト

参照

概念

Visual Basic の新機能

Visual Basic アプリケーション モデルの概要

My による開発

演算子の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

操作	言語要素
算術	^、–、*、/、\、Mod、+、=
代入	=, ^=, *=, /=, \=, +=, -=, &=
比較	=、<>、<、>、<=、>=、Like、Is
連結	&, +
論理/ビット処理演算子	Not、And、Or、Xor、AndAlso、および OrElse
その他の演算子	AddressOf および GetType

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

レジストリの概要

Visual Studio 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

My 機能を使用すると、これらの要素を使用するよりもレジストリ操作の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

処理	言語要素
プログラム設定の削除	DeleteSetting
プログラム設定の読み込み	GetSetting および GetAllSettings
プログラム設定の保存	SaveSetting

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

概念

一般的なレジストリ タスク

文字列操作の概要

Visual Basic 言語のキーワードとランタイム ライブラリ メンバを、目的と使用方法に応じて編成しました。

処理	言語要素
2 つの文字列の比較	StrComp
文字列の変換	StrConv
文字列の反転	InStrRev および StrReverse
文字列をすべて小文字または大文字に変換	Format、LCase および UCase
文字の繰り返しから成る文字列の作成	Space および StrDup
文字列の長さの確認	Len
文字列の書式指定	Format、FormatCurrency、FormatDateTime、FormatNumber、および FormatPercent
文字列の操作	InStr、Left、LTrim、Mid、Right、RTrim、および Trim
文字列比較の規則の設定	[Option Compare]
ASCII 値および ANSI 値の操作	Asc、AscW、Chr、および ChrW
指定した部分文字列の置換	Replace
フィルター ベースの文字列配列の取得	Filter
指定した数の部分文字列の取得	Split および Join

参照

関連項目

キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

Visual Basic の属性

Visual Basic には、オブジェクトでのアンマネージ コードとの相互運用を可能にする属性がいくつか用意されています。また、モジュールのメンバをモジュール名を指定せずにアクセスできるようにする属性もあります。

このセクションの内容

ComClassAttribute クラス

HideModuleNameAttribute クラス

VBFixedArrayAttribute クラス

VBFixedStringAttribute クラス

関連するセクション

Visual Basic における属性

ComClassAttribute クラス

ComClassAttribute 属性は、クラスを COM オブジェクトとして公開できるようにするメタデータの追加をコンパイラに指示します。

```
<System.AttributeUsage(System.AttributeTargets.Class, _
   Inherited := False, AllowMultiple := False)> _
Public NotInheritable Class ComClassAttribute
   Inherits System.Attribute
```

解説

ComClassAttribute を使うと、Visual Basic から COM コンポーネントを公開するプロセスを簡略化できます。COM オブジェクトと .NET Framework アセンブリはまったく異なるため、**ComClassAttribute** を使わずに Visual Basic から COM オブジェクトを生成するには、多くの手順が必要になります。**ComClassAttribute** が適用されているクラスでは、これらの手順の多くがコンパイラによって自動的に実行されます。

メモ :

この属性を使用すると、COM オブジェクトを簡単に作成できます。クラスを COM オブジェクトとして公開するには、[構成プロパティ] ダイアログ ボックスの [ビルド] セクションの、[COM の相互運用機能に登録] チェック ボックスをオンにして、プロジェクトをコンパイルする必要があります。

メモ :

Visual Basic で作成したクラスをアンマネージ コード用の COM オブジェクトとして公開することも可能ですが、そうすると本来の COM オブジェクトではなくなります。詳細については、「.NET Framework アプリケーションにおける COM 相互運用性」を参照してください。

使用例

この例を実行するには、新しい Class Library アプリケーションを作成し、以下のコードをクラス モジュールに追加します。

VB

```
    <ComClass(ComClass1.ClassId, ComClass1.InterfaceId, ComClass1.EventsId)> _
    Public Class ComClass1
        ' Use the Region directive to define a section named COM Guids.
#Region "COM GUIDs"
        ' These  GUIDs provide the COM identity for this class
        ' and its COM interfaces. You can generate
        ' these guids using guidgen.exe
        Public Const ClassId As String = "7666AC25-855F-4534-BC55-27BF09D49D46"
        Public Const InterfaceId As String = "54388137-8A76-491e-AA3A-853E23AC1217"
        Public Const EventsId As String = "EA329A13-16A0-478d-B41F-47583A761FF2"
#End Region

        Public Sub New()
            MyBase.New()
        End Sub

        Function AddNumbers(ByVal X As Integer, ByVal Y As Integer)
            AddNumbers = X + Y
        End Function
    End Class
```

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

チュートリアル : Visual Basic 2005 での COM オブジェクトの作成

関連項目

ComClassAttribute クラス メンバ
VBFixedArrayAttribute クラス
VBFixedStringAttribute クラス
System.Runtime.InteropServices
概念
相互運用固有の属性の適用
Visual Basic で使用される属性
その他の技術情報
Visual Basic における属性

ComClassAttribute クラス メンバ

ComClassAttribute クラス には、オブジェクトを識別し、そのオブジェクトの名前が別のメンバをシャドウするかどうかを示すプロパティがあります。

プロパティ

ClassID プロパティ	クラスを一意に識別するクラス ID を格納します。
EventID プロパティ	イベントを一意に識別するイベント ID を格納します。
InterfaceID プロパティ	インターフェイスを一意に識別するインターフェイス ID を格納します。
InterfaceShadows プロパティ	生成した COM インターフェイスの名前がクラスまたは基本クラスの別のメンバをシャドウするように指定します。

コンストラクタ

ComClassAttribute コンストラクタ	**ComClassAttribute** クラスの新しいインスタンスを初期化します。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

チュートリアル : Visual Basic 2005 での COM オブジェクトの作成

関連項目

VBFixedArrayAttribute クラス
VBFixedStringAttribute クラス
ComClassAttribute クラス
Shadows
System.Runtime.InteropServices

概念

相互運用固有の属性の適用
Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

ComClassAttribute コンストラクタ

ComClassAttribute クラスの新しいインスタンスを初期化します。

```
Public Sub New()
' -or-
Public Sub New( _
   ByVal _ClassID As String _
)
' -or-
Public Sub New( _
   ByVal _ClassID As String, _
   ByVal _InterfaceID As String _
)
' -or-
Public Sub New( _
   ByVal _ClassID As String, _
   ByVal _InterfaceID As String, _
   ByVal _EventId As String _
)
```

パラメータ

_ClassID

クラスを一意に識別する **ClassID** プロパティの値を初期化します。

_InterfaceID

インターフェイスを一意に識別する **InterfaceID** プロパティの値を初期化します。

_EventId

イベントを一意に識別する **EventID** プロパティの値を初期化します。

解説

ComClassAttribute をクラスに適用するときに **ClassID**、**InterfaceID**、または **EventID** プロパティを設定するには、**ComClassAttribute** クラスで構造体を使用します。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **ComClassAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

チュートリアル : Visual Basic 2005 での COM オブジェクトの作成

関連項目

ComClassAttribute クラス

System.Runtime.InteropServices

概念

Visual Basic で使用される属性

オブジェクトの有効期間 : オブジェクトの作成と破棄

その他の技術情報

Visual Basic における属性

ComClassAttribute.ClassID プロパティ

クラスを一意に識別するクラス ID を取得します。

```
Public ReadOnly Property ClassID As String
```

戻り値

読み取り専用です。COM オブジェクトの作成時にコンパイラがクラスを一意に識別するために使う文字列です。

解説

ComClassAttribute がクラスに適用されるときに、コンストラクタがこのプロパティを設定します。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **ComClassAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

ComClassAttribute クラス

System.Runtime.InteropServices

概念

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

ComClassAttribute.EventID プロパティ

イベントを一意に識別するイベント ID を取得します。

```
Public ReadOnly Property EventID As String
```

戻り値

読み取り専用。COM オブジェクトの作成時にコンパイラがクラスのイベントを一意に識別するために使う文字列です。

解説

ComClassAttribute がクラスに適用されるときに、コンストラクタがこのプロパティを設定します。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **ComClassAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

ComClassAttribute クラス

System.Runtime.InteropServices

概念

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

ComClassAttribute.InterfaceID プロパティ

インターフェイスを一意に識別するインターフェイス ID を取得します。

```
Public ReadOnly Property InterfaceID As String
```

戻り値

読み取り専用です。●COM オブジェクトの作成時にコンパイラがクラスのインターフェイスを一意に識別するために使う文字列です。

解説

ComClassAttribute がクラスに適用されるときに、コンストラクタがこのプロパティを設定します。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **ComClassAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

ComClassAttribute クラス

System.Runtime.InteropServices

概念

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

ComClassAttribute.InterfaceShadows プロパティ

COM インターフェイスの名前がクラスまたは基本クラスの別のメンバをシャドウするように指定します。

```
Public Property InterfaceShadows As Boolean
```

戻り値

COM インターフェイスの名前がクラスまたは基本クラスの別のメンバをシャドウするように指定する **Boolean** 値です。

解説

シャドウとは、あるメンバが別のメンバと同じ名前を使っている状態です。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **ComClassAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

ComClassAttribute クラス

System.Runtime.InteropServices

概念

Visual Basic におけるシャドウ

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

HideModuleNameAttribute クラス

HideModuleNameAttribute 属性をモジュールに適用すると、そのモジュールに必要な修飾子を使用した場合のみモジュール メンバにアクセスできるようになります。

```
' Usage
<HideModuleName> Module moduleName
' Declaration
<System.AttributeUsage(System.AttributeTargets.Class, _
   Inherited := False, AllowMultiple := False)> _
Public NotInheritable Class HideModuleNameAttribute
   Inherits System.Attribute
```

解説

この属性は、モジュールそのものは公開せずに、モジュール メンバのみを公開する場合に役立ちます。Visual Studio 統合開発環境 (IDE) の Visual Basic で IntelliSense を使用しているときは、このモジュールはステートメント入力候補リストに表示されません。その代わりに、モジュール メンバが表示されます。

モジュール名が非表示になっているときでも、そのモジュールを通じてモジュール メンバにアクセスできます。

メンバ名が他の識別子と競合している場合や、**HideModuleNameAttribute** 属性を持つ別のモジュールのメンバと競合している場合は、そのメンバは IntelliSense の該当モジュールのレベルに表示されません。

使用例

この例では、**HideModuleNameAttribute** 属性を使用して **My** に Database オブジェクトを追加する方法を示しています。このオブジェクトは `My.Database` を通じてアクセスできます。

VB

```
Namespace My
    <HideModuleName()> Module CustomMyDatabase
        Public ReadOnly Property Database() As MyDatabase
            Get
                Return databaseValue
            End Get
        End Property
        Private ReadOnly databaseValue As MyDatabase = New MyDatabase
    End Module
End Namespace

Class MyDatabase
    ' The members of the My.Database object go here.
End Class
```

Namespace ステートメントは、クラスやモジュールの外側に記述する必要があります。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : HideModuleNameAttribute

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

VBFixedArrayAttribute クラス

VBFixedStringAttribute クラス

概念

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

VBFixedArrayAttribute クラス

構造体または非ローカル変数に含まれる配列を固定長の配列として扱うことを示します。

```
<System.AttributeUsage(System.AttributeTargets.Field, _
   Inherited := False, AllowMultiple := False)> _
Public NotInheritable Class VBFixedArrayAttribute
   Inherits System.Attribute
```

解説

Visual Basic の配列は、既定では可変長です。固定サイズの配列を必要とする Visual Basic のファイル入出力関数 (**FileGet** や **FilePut** など) および API 呼び出しを使うときには、この属性を使用します。

使用例
VB

```
Structure Book
    <VBFixedArray(4)> Public Chapter() As Integer
End Structure

Sub WriteData()
    Dim FileNum As Integer = FreeFile()
    Dim MyBook As Book
    ReDim MyBook.Chapter(4)
    ' Add code to populate the array.
    MyBook.Chapter(0) = 1
    MyBook.Chapter(1) = 2
    MyBook.Chapter(2) = 3
    MyBook.Chapter(3) = 4
    MyBook.Chapter(4) = 5
    ' Write the array to a file.
    FileOpen(FileNum, "C:\testfile", OpenMode.Binary, _
             OpenAccess.Write, OpenShare.Default)
    FilePut(FileNum, MyBook) ' Write data.
    FileClose(FileNum)
End Sub
```

メモ :

VBFixedArrayAttribute は情報提供のための属性で、ストレージを割り当てません。この属性の目的は、構造体および非ローカル変数に含まれる配列についての、**VBFixedArrayAttribute** を認識するメソッドや API 呼び出しでの使用方法を変更することにあります。この属性によって可変長の配列が固定長の配列に変換されることはなく、**Dim** ステートメントまたは **ReDim** ステートメントで配列のストレージを割り当てる必要が残っていることに注意してください。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
VBFixedArrayAttribute クラス メンバ
VBFixedStringAttribute クラス
ComClassAttribute クラス
FileGet 関数
FilePut 関数
FileOpen 関数
MarshalAsAttribute
その他の技術情報
Visual Basic における属性

XML シリアル化および SOAP シリアル化

VBFixedArrayAttribute クラス メンバ

VBFixedArrayAttribute クラス には、配列のサイズと上下限を特定するプロパティがあります。

プロパティ

範囲	配列の境界を返します。
長さ	配列のサイズを返します。

コンストラクタ

VBFixedArrayAttribute コンストラクタ	固定サイズの配列の上下限を初期化します。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **VBFixedArrayAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

VBFixedStringAttribute クラス
ComClassAttribute クラス
MarshalAsAttribute

概念

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

VBFixedArrayAttribute コンストラクタ

Bounds プロパティの値を初期化します。

```
Public Sub New( _
   ByVal UpperBound1 As Integer, _
   )
' -or-
Public Sub New( _
   ByVal UpperBound1 As Integer, _
   ByVal UpperBound2 As Integer, _
   )
```

パラメータ

UpperBound1

配列の最初の次元のサイズを表す上限フィールドの値を初期化します。

UpperBound2

配列の最初の次元のサイズを表す上限フィールドの値を初期化します。

解説

このコンストラクタは、**VBFixedArrayAttribute** 属性が配列に適用されているときに実行されます。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **VBFixedArrayAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

VBFixedStringAttribute クラス
ComClassAttribute クラス
MarshalAsAttribute Class

概念

Visual Basic で使用される属性
オブジェクトの有効期間 : オブジェクトの作成と破棄

その他の技術情報

Visual Basic における属性

VBFixedArrayAttribute.Bounds プロパティ

配列の境界を返します。

```
Public ReadOnly Property Bounds() As Integer()
```

戻り値

配列の境界を表す整数型の配列が含まれます。

解説

配列に **VBFixedArrayAttribute** が適用されていると、コンストラクタによってこのプロパティが設定されます。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **VBFixedArrayAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

VBFixedStringAttribute クラス
ComClassAttribute クラス
MarshalAsAttribute

概念

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

VBFixedArrayAttribute.Length プロパティ

配列のサイズを返します。

```
Public ReadOnly Property Length As Integer
```

戻り値

配列の要素数を表す整数が含まれます。

解説

配列に **VBFixedArrayAttribute** が適用されていると、コンストラクタによってこのプロパティが設定されます。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **VBFixedArrayAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

VBFixedStringAttribute クラス
ComClassAttribute クラス
MarshalAsAttribute Class

概念

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

VBFixedStringAttribute クラス

文字列を固定長と見なして扱うことを示します。

```
<System.AttributeUsage(System.AttributeTargets.Field, _
   Inherited := False, AllowMultiple := False)> _
Public NotInheritable Class VBFixedStringAttribute
   Inherits System.Attribute
```

解説

Visual Basic の文字列は、既定では可変長です。固定長文字列を必要とする Visual Basic のファイル入出力関数 (**FileGet** や **FilePut** など) を使うときには、この属性を使用します。

メモ :

VBFixedStringAttribute 属性は、文字列の長さを文字数ではなく、バイト数で指定します。

使用例
VB

```
Structure Person
    Public ID As Integer
    Public MonthlySalary As Decimal
    Public LastReviewDate As Long
    <VBFixedString(15)> Public FirstName As String
    <VBFixedString(15)> Public LastName As String
    <VBFixedString(15)> Public Title As String
    <VBFixedString(150)> Public ReviewComments As String
End Structure
```

メモ :

VBFixedStringAttribute は情報提供のための属性で、可変長の文字列を固定長の文字列に変換するためには使用できません。この属性の目的は、構造体および非ローカル変数に含まれる文字列についての、**VBFixedStringAttribute** を認識するメソッドや API (**Len** 関数や **FilePut** 関数など) の呼び出しでの使用方法を変更することにあります。この属性によって文字列自身の実際の長さが変更されることはない点に注意してください。

スマート デバイス開発者のためのメモ

このクラスはサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
VBFixedStringAttribute クラス メンバ
VBFixedArrayAttribute クラス
ComClassAttribute クラス
Len 関数 (Visual Basic)
FileGet 関数
FilePut 関数
FileOpen 関数
StringBuilder
MarshalAsAttribute
概念
Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

VBFixedStringAttribute クラス メンバ

VBFixedStringAttribute クラス には、文字列の長さを特定するプロパティがあります。

フィールド

VBFixedStringAttribute.Length	文字列の長さを取得します。

コンストラクタ

VBFixedStringAttribute コンストラクタ	**SizeConst** フィールドの値を初期化します。

スマート デバイス開発者のためのメモ

このクラスはサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

VBFixedArrayAttribute クラス
VBFixedStringAttribute クラス
ComClassAttribute クラス
StringBuilder
MarshalAsAttribute

概念

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

VBFixedStringAttribute コンストラクタ

SizeConst フィールドの値を初期化します。

```
Public Sub New( _
   ByVal Length As Integer _
)
```

パラメータ
Length

固定長文字列の長さです。

解説

このコンストラクタは、**VBFixedStringAttribute** 属性が文字列に適用されているときに実行されます。

スマート デバイス開発者のためのメモ

このコンストラクタはサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **VBFixedStringAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
VBFixedStringAttribute クラス
StringBuilder Class
MarshalAsAttribute Class
概念
Visual Basic で使用される属性
オブジェクトの有効期間 : オブジェクトの作成と破棄
その他の技術情報
Visual Basic における属性

VBFixedStringAttribute.Length プロパティ

文字列の長さを取得します。

```
Public ReadOnly Property Length As Integer
```

戻り値

文字列の長さを返します。

解説

VBFixedStringAttribute が文字列に適用されるときに、この属性のコンストラクタがこのプロパティを設定します。

スマート デバイス開発者のためのメモ

このプロパティはサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **VBFixedStringAttribute**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

VBFixedStringAttribute クラス

概念

Visual Basic で使用される属性

その他の技術情報

Visual Basic における属性

定数と列挙型 (Visual Basic)

Visual Basic には、定義済みの定数および列挙型が開発者向けに多数用意されています。定数に格納された値は、アプリケーションの実行中に変わることはありません。複数の関連する定数を操作する場合や、複数の定数値に名前を関連付ける場合は、列挙型を使うと便利です。

このセクションの内容

AppWinStyle 列挙型

Shell 関数を呼び出すときに、プログラムの起動に使用するウィンドウ スタイルを示します。

AudioPlayMode 列挙型

オーディオ関連のメソッドを呼び出すときに、サウンドの再生方法を示します。

BuiltInRole 列挙型

My.User.IsInRole メソッド を呼び出すときに、チェックするロールの種類を示します。

CallType 列挙型

CallByName 関数を呼び出すときに呼び出すプロシージャの種類を示します。

CompareMethod 列挙型

比較関数を呼び出すときに文字列を比較する方法を示します。

条件付きコンパイル定数

条件付きコンパイルで使用可能な定義済み定数の一覧を示します。

DateFormat 列挙型

FormatDateTime 関数を呼び出したときの日付の表示方法を示します。

DateInterval 列挙型

日付関連の関数を呼び出すときに使用する、日付の間隔の測定方法と書式を示します。

DeleteDirectoryOption 列挙型

削除対象のディレクトリ内にファイルまたはディレクトリが存在する場合の処理を指定します。

DueDate 列挙型

財務関係のメソッドを呼び出すときに支払い期日を示します。

FieldType 列挙型

テキスト フィールドが区切られているのか、固定幅なのかを示します。

FileAttribute 列挙型

ファイル アクセス用の関数を呼び出すときに使用するファイル属性を示します。

FirstDayOfWeek 列挙型

日付関連の関数を呼び出すときに使用する、1 年の最初の日を示します。

FirstWeekOfYear 列挙型

日付関連の関数を呼び出すときに使用する、1 年の最初の週を示します。

MsgBoxResult 列挙型

メッセージ ボックスで押されたボタンを示し、**MsgBox** 関数から返されます。

MsgBoxStyle 列挙型

MsgBox 関数を呼び出すときに、表示するボタンを示します。

OpenAccess 列挙型

ファイル アクセス関数を呼び出すときにファイルを開く方法を示します。

OpenMode 列挙型

ファイル アクセス関数を呼び出すときにファイルを開く方法を示します。

OpenShare 列挙型

ファイル アクセス関数を呼び出すときにファイルを開く方法を示します。

印刷と表示の定数

印刷および表示を行う関数を呼び出す際に使用可能な、定義済みの定数を一覧で示します。

RecycleOption 列挙型

ファイルを完全に削除するのか、ごみ箱に入れるのかを指定します。

SearchOption 列挙型

すべてのディレクトリまたは最上位レベルのディレクトリの、いずれを探索するのかを指定します。

TriState 列挙型

数値書式指定関数を呼び出すときに、ブール (**Boolean**) 値、または既定の設定を使用するかどうかを示します。

UICancelOption 列挙型

操作中にユーザーが [キャンセル] ボタンをクリックしたときに実行する処理を指定します。

UIOption 列挙型

ファイルまたはディレクトリをコピー、削除、または移動するときに、プログレス ダイアログ ボックスを表示するかどうかを指定します。

VariantType 列挙型

VarType 関数が返すバリアント オブジェクトの型を示します。

VbStrConv 列挙型

StrConv 関数を呼び出すときに実行する変換の種類を示します。

関連するセクション

Visual Basic リファレンス

Visual Basic

Visual Basic の定数

列挙型の概要

AppWinStyle 列挙型

Shell 関数を呼び出すときに、プログラムの起動に使用するウィンドウ スタイルを示します。

解説

Shell コマンドを発行するとき、実際の値の代わりにコード内で次の列挙型のメンバを使用できます。

Style 引数は、**AppWinStyle** 列挙型のメンバを受け取ります。

Members

メンバ	定数	説明
Hide	**vbHide**	ウィンドウは非表示です。フォーカスはこの非表示のウィンドウに渡されます。
NormalFocus	**vbNormalFocus**	ウィンドウはフォーカスを持ち、元の大きさと位置で表示されています。
MinimizedFocus	**vbMinimizedFocus**	ウィンドウはフォーカスを持った状態で最小化表示されています。
MaximizedFocus	**vbMaximizedFocus**	ウィンドウはフォーカスを持った状態で最大化表示されています。
NormalNoFocus	**vbNormalNoFocus**	ウィンドウは最後にウィンドウを閉じたときの大きさと位置で表示されています。現在アクティブなウィンドウは、アクティブのままです。
MinimizedNoFocus	**vbMinimizedNoFocus**	ウィンドウは最小化表示されています。現在アクティブなウィンドウは、アクティブのままです。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

Shell 関数

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

AudioPlayMode 列挙型

オーディオ関連のメソッドを呼び出すときに、サウンドの再生方法を示します。

解説

AudioPlayMode 列挙型のメンバを使って、My.Computer.Audio.Play メソッド によるサウンドの再生を制御できます。

メンバ

メンバ	説明
WaitToComplete	**My.Computer.Audio.Play** メソッドは、サウンドを再生し、再生の完了を待った後で呼び出し元のコードに戻ります。
Background	**My.Computer.Audio.Play** メソッドは、サウンドをバックグラウンドで再生します。呼び出し元のコードは実行を続行します。
BackgroundLoop	**My.Computer.Audio.Play** メソッドは、My.Computer.Audio.Stop メソッド が呼び出されるまでサウンドをバックグラウンドで再生します。呼び出し元のコードは実行を続行します。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

My.Computer.Audio.Play メソッド

AudioPlayMode

概念

組み込み定数と組み込み列挙型

BuiltInRole 列挙型

My.User.IsInRole メソッド を呼び出すときに、チェックするロールの種類を示します。

解説

この列挙型は、WindowsBuiltInRole 列挙型と同様の機能を提供します。

これらのロールは、Windows NT、Windows 2000、および Windows XP のほとんどのインストールに共通するローカル Windows グループを表しています。

メンバ

メンバ	説明
AccountOperator	コンピュータまたはドメイン上のユーザー アカウントを管理する Account Operators。
Administrator	コンピュータまたはドメインに対する完全な無制限のアクセス権を持つ Administrators。
BackupOperator	ファイルのバックアップまたは復元を唯一の目的としてセキュリティ制限をオーバーライドできる Backup Operators。
Guest	ユーザーよりも制限の厳しい Guests。
PowerUser	ほとんどの管理権限を持ち、一部制限が課されている Power Users。Power Users は、保証されたアプリケーションだけでなくレガシ アプリケーションも実行できます。
PrintOperator	プリンタを制御できる Print Operators。
Replicator	ドメイン内のファイル レプリケーションを支援する Replicators。
SystemOperator	特定のコンピュータを管理する System Operators。
User	システム全体に対する予定外の変更または意図的な変更を禁じられている Users。Users は保証されたアプリケーションを実行できますが、ほとんどのレガシ アプリケーションは実行できません。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照

関連項目

My.User.IsInRole メソッド

Microsoft.VisualBasic.ApplicationServices.BuiltInRole

概念

組み込み定数と組み込み列挙型

CallType 列挙型

CallByName 関数を呼び出すときに呼び出すプロシージャの種類を示します。

解説

CallByName 関数を呼び出すときに、実際の値の代わりにコード内で次の **CallType** 列挙型のメンバを使用できます。*UseCallType* 引数は、**CallType** 列挙型のメンバを受け取ります。

メンバ

メンバ	定数	説明
Method	**vbMethod**	メソッドを呼び出します。
Get	**vbGet**	プロパティ値を取得します。
Set	**vbSet**	プロパティ値を設定します。
Let	**vbLet**	オブジェクトのプロパティ値を設定します。

スマート デバイス開発者のためのメモ

この列挙型はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

CallByName 関数

CallType 定数 (Visual Basic 6.0 ユーザー向け)

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

CompareMethod 列挙型

比較関数を呼び出すときに文字列を比較する方法を示します。

解説

比較関数を呼び出すときに、実際の値の代わりにコード内で **CompareMethod** 列挙型を使用できます。*Compare* 引数は、**CompareMethod** 列挙定数のメンバを受け取ります。

Members

メンバ	定数	説明
Binary	**vbBinaryCompare**	バイナリ モードで比較を行います。
Text	**vbTextCompare**	テキスト モードで比較を行います。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

Filter 関数 (Visual Basic)
InStr 関数 (Visual Basic)
InStrRev 関数 (Visual Basic)
Replace 関数 (Visual Basic)
Split 関数 (Visual Basic)
StrComp 関数 (Visual Basic)

概念

組み込み定数と組み込み列挙型
列挙型を使用する状況

条件付きコンパイル定数

条件付きコンパイルを使用すると、プログラムにどのコードを含めるかを、コンパイル時に簡単に制御できます。

条件付きコンパイルで利用できる定義済みの定数の一覧を次の表に示します。

定数	説明
CONFIG	構成マネージャの [アクティブ ソリューション構成] ボックスの、現在の設定に対応する文字列
DEBUG	[プロジェクトのプロパティ] ダイアログ ボックスで設定できる **Boolean** 値既定では、プロジェクトの Debug 構成により、**DEBUG** が定義されます。**DEBUG** を定義すると、Debug クラスのメソッドは出力ウィンドウに出力を生成します。DEBUG を定義しない場合、**Debug** クラスのメソッドはコンパイルされず、デバッグ出力も生成されません。
TARGET	プロジェクトの出力の種類、またはコマンド ラインの **/target** オプションの設定を表す文字列TARGET に指定可能な値は、Windows アプリケーションの "winexe"、コンソール アプリケーションの "exe"、クラス ライブラリの "library" 、そしてモジュールの "module" です。**/target** オプションは、Visual Studio の統合開発環境 (IDE: Integrated Development Environment) で設定できます。詳細については、「/target」を参照してください。
TRACE	[プロジェクトのプロパティ] ダイアログ ボックスで設定できる **Boolean** 値既定では、プロジェクトのすべての構成により、**TRACE** が定義されます。**TRACE** を定義すると、Trace クラスのメソッドは出力ウィンドウに出力を生成します。定義しない場合、**Trace** クラスのメソッドはコンパイルされず、**Trace** 出力も生成されません。
VBC_VER	*major.minor* 形式で、Visual Basic のバージョンを表す数。Visual Basic 2005 のバージョン番号は 8.0 です。
_MYTYPE	ビルド対象のプロジェクトの種類を表す文字列。これを使用して、**My** オブジェクトのどれをコード内で利用可能にするかが制御できます。詳細については、「プロジェクトの種類に応じた My の機能」を参照してください。

これらの定数が使用できるのは条件付きコンパイルだけです。実行可能コードでは使用できません。

使用例

特定のステートメントをコンパイルするかどうかを、条件付きコンパイル定数 **TARGET** を使用して決定する例を次に示します。

VB

```
#If TARGET = "winexe" Then
        ' Insert code to be compiled for a Windows application.
#ElseIf TARGET = "exe" Then
        ' Insert code to be compiled for a console application.
#End If
```

必要条件

Visual Studio の統合開発環境には、**CONFIG**、**DEBUG**、**TRACE**、および **_MYTYPE** の条件付きコンパイル定数が定義されています。

Visual Basic コンパイラで定義されている条件付きコンパイル定数は **TARGET** と **VBC_VER** です。**TARGET**、**VBC_VER**、および **_MYTYPE** の定数は、Visual Basic 2005 よりも前のバージョンのコンパイラでは利用できません。

参照

処理手順

方法 : 条件付きコンパイル定数を宣言する

関連項目

#If...Then...#Else ディレクティブ

#Const ディレクティブ

/target (Visual Basic)

概念

条件付きコンパイルの概要

プロジェクトの種類に応じた My の機能

DateFormat 列挙型

FormatDateTime 関数を呼び出したときの日付の表示方法を示します。

解説

FormatDateTime 関数を呼び出すときに、実際の値の代わりにコード内で次の列挙体のメンバを使用できます。

メモ：
使用しているコンピュータの地域設定にアクセスするには、[コントロール パネル] の [地域と言語のオプション] をダブルクリックします。

メンバ

メンバ	定数	説明
GeneralDate	**vbGeneralDate**	実数では、日付と時刻を表示します。小数部がない場合は、日付のみを表示します。整数部がない場合は、時刻のみを表示します。日付および時刻の表示は、使用するコンピュータの地域設定に従います。
LongDate	**vbLongDate**	[地域のオプション] で指定されている長い形式で日付を表示します。
ShortDate	**vbShortDate**	[地域のオプション] で指定されている短い形式で日付を表示します。
LongTime	**vbLongTime**	コントロール パネルの [地域のオプション] で指定されている長い形式で時刻を表示します。
ShortTime	**vbShortTime**	コントロール パネルの [地域のオプション] で指定されている短い形式で時刻を表示します。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

FormatDateTime 関数 (Visual Basic)

概念

カスタム DateTime 書式指定文字列

列挙型を使用する状況

DateInterval 列挙型

日付関連の関数を呼び出すときに使用する、日付の間隔の測定方法と書式を示します。

解説

日付関連の関数を呼び出すときに、実際の値の代わりにコード内で列挙型メンバを使用できます。

DateInterval 列挙型で定義されている定数を日付関連の関数で使うと、日付の間隔の測定方法と書式を特定できます。次の表は、**DateInterval** 列挙型メンバの一覧です。

メンバ

メンバ	説明
Day	日付 (1 ～ 31)
DayOfYear	年間通算日 (1 ～ 366)
Hour	時 (1 ～ 24)
Minute	分 (1 ～ 60)
Month	月 (1 ～ 12)
Quarter	四半期 (1 ～ 4)
Second	秒 (1 ～ 60)
Weekday	曜日 (1 ～ 7)
WeekOfYear	週 (1 ～ 53)
Year	年

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

DateFormat 列挙型
DueDate 列挙型
DatePart 関数 (Visual Basic)
DateAdd 関数 (Visual Basic)
DateDiff 関数 (Visual Basic)

概念

組み込み定数と組み込み列挙型
列挙型を使用する状況

DeleteDirectoryOption 列挙型

削除対象のディレクトリ内にファイルまたはディレクトリが存在する場合の処理を指定します。

解説

この列挙型は My.Computer.FileSystem.DeleteDirectory メソッド で使用します。

メンバ

メンバ	説明
DeleteAllContents	ディレクトリと一緒にディレクトリの内容も削除します。既定値です。
ThrowIfDirectoryNonEmpty	ディレクトリが空でない場合に IOException をスローします。例外の **Data** プロパティに、削除できなかったファイルの一覧が入っています。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : DeleteDirectoryOption

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でディレクトリを削除する

関連項目

My.Computer.FileSystem.DeleteDirectory メソッド

Microsoft.VisualBasic.FileIO.DeleteDirectoryOption

DueDate 列挙型

財務関係のメソッドを呼び出すときに支払い期日を示します。

解説

財務関係の関数を呼び出すときに、実際の値の代わりにコード内で列挙型のメンバを使用できます。

DueDate 列挙型には、支払い期日を示す定数が定義されています。

Members

メンバ	説明
BegOfPeriod	日付の間隔の最初の日が期日になります。
EndOfPeriod	日付の間隔の最後の日が期日になります。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

DateInterval 列挙型

財務処理の概要

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

FieldType 列挙型

テキスト フィールドが区切られているのか、固定幅なのかを示します。

解説

この列挙型は TextFieldParser オブジェクト で使用します。

メンバ

メンバ名	説明
Delimited	フィールドが区切られていることを示します。
FixedWidth	フィールドが固定幅であることを示します。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で固定幅のテキスト ファイルを読み取る

関連項目

TextFieldParser.TextFieldType プロパティ
TextFieldParser
OpenTextFieldParser

FileAttribute 列挙型

ファイル アクセス用の関数を呼び出すときに使用するファイル属性を示します。

解説

Dir 関数、**GetAttr** 関数、または **SetAttr** 関数を呼び出すときに、実際の値の代わりにコード内で **FileAttribute** 列挙型を使用できます。

Attributes 引数は、**FileAttribute** 列挙型のメンバを受け取ります。

ファイル I/O 操作を実行するときに **My.Computer.FileSystem** オブジェクトを使用すると、レガシのファイル I/O メソッドより簡単に使用でき、パフォーマンスもよくなります。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

メンバ

メンバ	定数	説明
Normal	**vbNormal**	通常ファイル (**Dir** 関数および **SetAttr** 関数の既定値)。このファイルには特別な特性は適用されません。
ReadOnly	**vbReadOnly**	読み取り専用。
Hidden	**vbHidden**	隠しファイル。
System	**vbSystem**	システム ファイル。
Volume	**vbVolume**	ボリューム ラベルこの属性は **SetAttr** では使用できません。
Directory	**vbDirectory**	ディレクトリまたはフォルダ。
Archive	**vbArchive**	前回のバックアップ以降に変更されているファイル。

スマート デバイス開発者のためのメモ

この列挙型はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

Dir 関数

GetAttr 関数

SetAttr 関数

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

FirstDayOfWeek 列挙型

日付関連の関数を呼び出すときに使用する、1 年の最初の日を示します。

解説

日付関連の関数を呼び出すときに、実際の値の代わりにコード内で次の列挙型のメンバを使用できます。

一部の日付関連関数は、引数として *DayOfWeek*、*WeekOfYear*、またはその両方を使用します。**FirstDayOfWeek** 列挙体は、*DayOfWeek* 引数の有効な値と、日付関連関数からの戻り値を指定します。

DayOfWeek 引数は、**FirstDayOfWeek** 列挙体のメンバを受け取ります。

また、**System** を除く **FirstDayOfWeek** の値は、日付関連関数の戻り値にも使用されます。

メンバ

メンバ	定数	説明
System	**vbUseSystemDayOfWeek**	週の最初の曜日としてシステムで指定された曜日
Sunday	**vbSunday**	日曜日 (既定値)
Monday	**vbMonday**	月曜日
Tuesday	**vbTuesday**	火曜日
Wednesday	**vbWednesday**	水曜日
Thursday	**vbThursday**	木曜日
Friday	**vbFriday**	金曜日
Saturday	**vbSaturday**	土曜日

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

FirstWeekOfYear 列挙型
DateFormat 列挙型
DateDiff 関数 (Visual Basic)
DatePart 関数 (Visual Basic)
Format 関数
Weekday 関数 (Visual Basic)

概念

組み込み定数と組み込み列挙型
列挙型を使用する状況

FirstWeekOfYear 列挙型

日付関連の関数を呼び出すときに使用する、1 年の最初の週を示します。

解説

日付関連の関数を呼び出すときに、実際の値の代わりにコード内で次の列挙型のメンバを使用できます。

一部の日付関連関数は、引数として *DayOfWeek*、*WeekOfYear*、またはその両方を使用します。**FirstWeekOfYear** 列挙型は、*WeekOfYear* 引数の有効な値と、日付関連関数からの戻り値を指定します。

WeekOfYear 引数は、**FirstWeekOfYear** 列挙型のメンバを受け取ります。

メンバ

メンバ	定数	説明
System	**vbUseSystem**	週の最初の曜日としてシステム設定で指定された曜日
Jan1	**vbFirstJan1**	1 月 1 日を含む週 (既定値)
FirstFourDays	**vbFirstFourDays**	7 日のうち少なくとも 4 日が新年度に含まれる週
FirstFullWeek	**vbFirstFullWeek**	1 週間全体が新年度に含まれる最初の週

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

FirstDayOfWeek 列挙型
DateFormat 列挙型
DateDiff 関数 (Visual Basic)
DatePart 関数 (Visual Basic)
Format 関数

概念

組み込み定数と組み込み列挙型
列挙型を使用する状況

MsgBoxResult 列挙型

メッセージ ボックスで押されたボタンを示し、**MsgBox** 関数から返されます。

解説

MsgBox 関数を呼び出すときに、実際の値の代わりにコード内で **MsgBoxResult** 列挙型を使用できます。

MsgBox 関数は、**MsgBoxResult** 列挙型の値を返します。

メンバ

メンバ	定数	説明
OK	**vbOK**	[OK] ボタン
Cancel	**vbCancel**	[キャンセル] ボタン
Abort	**vbAbort**	[中止] ボタン
Retry	**vbRetry**	[再試行] ボタン
Ignore	**vbIgnore**	[無視] ボタン
Yes	**vbYes**	[はい] ボタン
No	**vbNo**	[いいえ] ボタン

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

MsgBoxStyle 列挙型

MsgBox 関数 (Visual Basic)

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

MsgBoxStyle 列挙型

MsgBox 関数を呼び出すときに、表示するボタンを示します。

解説

MsgBox 関数を呼び出すときに、実際の値の代わりにコード内で **MsgBoxStyle** 列挙型を使用できます。*Buttons* 引数は、**MsgBoxStyle** 列挙体のメンバを受け取ります。

メンバ

メンバ	定数	説明
OKOnly	**vbOKOnly**	[OK] ボタンのみを表示します (既定)。
OKCancel	**vbOKCancel**	[OK] ボタンと [キャンセル] ボタンを表示します。
AbortRetryIgnore	**vbAbortRetryIgnore**	[中止]、[再試行]、[無視] の 3 つのボタンを表示します。
YesNoCancel	**vbYesNoCancel**	[はい]、[いいえ]、[キャンセル] の 3 つのボタンを表示します。
YesNo	**vbYesNo**	[はい] ボタンと [いいえ] ボタンを表示します。
RetryCancel	**vbRetryCancel**	[再試行] ボタンと [キャンセル] ボタンを表示します。
Critical	**vbCritical**	警告メッセージ アイコンを表示します。
Question	**vbQuestion**	問い合わせメッセージ アイコンを表示します。
Exclamation	**vbExclamation**	注意メッセージ アイコンを表示します。
Information	**vbInformation**	情報メッセージを表示します。
DefaultButton1	**vbDefaultButton1**	第 1 ボタンを既定ボタンにします。
DefaultButton2	**vbDefaultButton2**	第 2 ボタンを既定ボタンにします。
DefaultButton3	**vbDefaultButton3**	第 3 ボタンを既定ボタンにします。
ApplicationModal	**vbApplicationModal**	アプリケーション モーダルに設定します。
SystemModal	**vbSystemModal**	システム モーダルに設定します。
MsgBoxSetForeground	**vbMsgBoxSetForeground**	メッセージ ボックス ウィンドウを前面に表示します。
MsgBoxRight	**vbMsgBoxRight**	右揃え。
MsgBoxRtlReading	**vbMsgBoxRtlReading**	右から左の方向でテキストを表示します (ヘブライ語およびアラビア語のシステム)。
MsgBoxHelp	**vbMsgBoxHelp**	ヘルプ テキストを表示します。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

MsgBoxResult 列挙型

MsgBox 関数 (Visual Basic)

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

OpenAccess 列挙型

ファイル アクセス用の関数を呼び出すときにファイルを開く方法を示します。

解説

ファイル アクセス関連の関数を呼び出すときには、実際の値を指定する代わりに、コード内で列挙定数のメンバを使用できます。

OpenAccess 列挙型には、ファイルを読み込むことができる (**Read**) のか、ファイルに書き込むことができる (**Write**) のか、両方可能なのか (**ReadWrite**) を示す定数が定義してあります。**Read** アクセスで開いたファイルは変更できません。読み込みだけを実行できます。次の表は、**OpenAccess** 列挙型メンバの一覧です。

ファイル I/O 操作を実行するときに **My.Computer.FileSystem** オブジェクトを使用すると、レガシのファイル I/O メソッドより簡単に使用でき、パフォーマンスもよくなります。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

メンバ

メンバ	説明
Default	読み取りおよび書き込みアクセスを許可します。これは、既定の設定です。
Read	読み取りアクセスを許可します。
ReadWrite	読み取りおよび書き込みアクセスを許可します。
Write	書き込みアクセスを許可します。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

OpenMode 列挙型

OpenShare 列挙型

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

OpenMode 列挙型

ファイル アクセス関数を呼び出すときにファイルを開く方法を示します。

解説

ファイル アクセス関連の関数を呼び出すときには、実際の値を指定する代わりに、コード内で列挙型メンバを使用できます。

OpenMode 列挙型には、ファイル アクセス モードを設定するための定数が定義されています。次の表は、**OpenMode** 列挙型メンバの一覧です。

テキスト ファイルなどのファイルに順次アクセスする場合、**Input**、**Output**、および **Append** を使用します。バイナリ ファイルの場合は **Binary** を、ランダム アクセス ファイルの場合は **Random** を使用します。

ファイルに順次アクセスする場合、ファイルのデータを変更できません。データの読み込み、追加、または新しいデータによる上書きを行うことができます。入力用に開いた場合は、自分で直接書き込まなくても、ファイルの内容が上書きされます。

ファイル I/O 操作を実行するときに **My.Computer.FileSystem** オブジェクトを使用すると、レガシのファイル I/O メソッドより簡単に使用でき、パフォーマンスもよくなります。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

メンバ

メンバ	説明
Append	追加用にファイルを開きます。既定値です。
Binary	バイナリ アクセス用にファイルを開きます。
Input	読み取りアクセス用にファイルを開きます。
Output	書き込みアクセス用にファイルを開きます。
Random	ランダム アクセス用にファイルを開きます。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

OpenShare 列挙型

OpenAccess 列挙型

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

OpenShare 列挙型

ファイル アクセス関数を呼び出すときにファイルを開く方法を示します。

解説

ファイル アクセス関連の関数を呼び出すときには、実際の値を指定する代わりに、コード内で列挙体のメンバを使用できます。

OpenShare 列挙体には、このアプリケーションでファイルを開いている間に、他のプロセスがファイルにアクセスできるかどうかを指定するための定数が定義されています。次の表は、**OpenShare** 列挙型メンバの一覧です。

ファイル I/O 操作を実行するときに **My.Computer.FileSystem** オブジェクトを使用すると、レガシなファイル I/O メソッドよりもパフォーマンスがよくなります。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

メンバ

メンバ	説明
Default	**LockReadWrite** これは、既定の設定です。
LockRead	他のプロセスはこのファイルを読み取ることができません。
LockReadWrite	他のプロセスはこのファイルに対して読み取りも書き込みもできません。
LockWrite	他のプロセスはこのファイルに書き込むことができません。
Shared	すべてのプロセスがこのファイルに対して読み取りおよび書き込みできます。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

OpenAccess 列挙型

OpenMode 列挙型

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

印刷と表示の定数

印刷および表示の関数を呼び出すときに、実際の値の代わりにコード内で次の定数を使用できます。

ControlChars モジュールのメンバ

メンバ	定数	等価な値	説明
CrLf	**vbCrLf**	**Chr(**13**) + Chr(**10**)**	復帰と改行の組み合わせ。
Cr	**vbCr**	**Chr(**13**)**	復帰文字。
Lf	**vbLf**	**Chr(**10**)**	ライン フィード文字。
NewLine	**vbNewLine**	**Chr(**13**) + Chr(**10**)**	改行文字。
NullChar	**vbNullChar**	**Chr(**0**)**	値 0 を持つ文字。
n/a	**vbNullString**	値 0 を持つ文字列	長さ 0 の文字列 ("") とは異なります。外部プロシージャを呼び出す場合に使用します。
n/a	**vbObjectError**	-2147221504	エラー番号。ユーザー定義エラーの番号は、これより大きい値にします。たとえば次のように指定します。 `Err.Raise(Number) = vbObjectError + 1000`
Tab	**vbTab**	**Chr(**9**)**	タブ文字。
Back	**vbBack**	**Chr(**8**)**	バックスペース文字。
FormFeed	**vbFormFeed**	**Chr(**12**)**	Microsoft Windows では使用できません。
VerticalTab	**vbVerticalTab**	**Chr(**11**)**	Microsoft Windows では使用できません。
Quote	n/a	**Chr(**34**)**	値を囲むのに使用する引用符文字 (" または ')。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Constants**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Print 関数、PrintLine 関数

Write 関数、WriteLine 関数

概念

組み込み定数と組み込み列挙型

定数の概要

Visual Basic によって宣言された定数

RecycleOption 列挙型

ファイルを完全に削除するのか [ごみ箱] に入れるのかを指定します。

解説

この列挙型は My.Computer.FileSystem.DeleteDirectory メソッド および My.Computer.FileSystem.DeleteFile メソッド で使用します。

削除したファイルを [ごみ箱] に入れておくと、ファイルを復元することができるので、ユーザーは重要なファイルを誤って削除する事態を防ぐことができます。ファイルを完全に削除するには、[ゴミ箱] からファイルを明示的に削除する必要があります。ファイルを [ごみ箱] から完全に削除するまでは、これらのファイルが使用しているディスク容量を他で使用することはできません。

全く必要のない、大量のファイル (または少量の非常に大きなファイル) を削除して、ディスク容量を増やす場合など、[ごみ箱] を経由しないほうが便利なことがあります。

メンバ

メンバ	説明
DeletePermanently	ファイルまたはディレクトリを完全に削除します。既定値です。
SendToRecycleBin	ファイルまたはディレクトリを [ごみ箱] に送ります。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : RecycleOption

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でファイルを削除する
方法 : Visual Basic でディレクトリを削除する

関連項目

My.Computer.FileSystem.DeleteDirectory メソッド
My.Computer.FileSystem.DeleteFile メソッド
Microsoft.VisualBasic.FileIO.RecycleOption

SearchOption 列挙型

すべてのディレクトリまたは最上位レベルのディレクトリの、いずれを探索するのかを指定します。

解説

この列挙型は My.Computer.FileSystem.GetFiles メソッド、My.Computer.FileSystem.GetDirectoryInfo メソッド、および My.Computer.FileSystem.FindInFiles メソッド で使用します。

メンバ

メンバ	説明
SearchAllSubDirectories	指定したディレクトリおよびそのすべてのサブディレクトリを検索します。既定値です。
SearchTopLevelOnly	指定したディレクトリのみを検索し、サブディレクトリは検索しません。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : SearchOption

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

My.Computer.FileSystem.FindInFiles メソッド
My.Computer.FileSystem.GetDirectoryInfo メソッド
My.Computer.FileSystem.GetFiles メソッド
Microsoft.VisualBasic.FileIO.SearchOption

TriState 列挙型

数値書式指定関数を呼び出すときに、ブール値を示すか、既定の値を使用する必要があるのかを示します。

解説

数値書式指定関数を呼び出すときに、実際の値の代わりにコード内で次の列挙定数のメンバを使用できます。

メンバ

メンバ	説明
True	True。このメンバの数値は -1 です。
False	False。このメンバの数値は 0 です。
UseDefault	既定の設定。このメンバの数値は -2 です。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

FormatNumber 関数 (Visual Basic)
FormatCurrency 関数 (Visual Basic)
FormatPercent 関数 (Visual Basic)

概念

列挙型を使用する状況

UICancelOption 列挙型

ユーザーが操作中に [キャンセル] をクリックした場合に、例外をスローするかどうかを指定します。

解説

この列挙型は、ユーザーが操作中に [キャンセル] をクリックした場合に、IOException をスローするかどうかを決定します。いずれの場合でも、操作はキャンセルされます。

Members

メンバ名	説明
DoNothing	ユーザーが [キャンセル] をクリックした場合に何も実行しません。
ThrowException	ユーザーが [キャンセル] をクリックした場合に例外をスローします。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : UICancelOption

プラットフォーム : Windows 95、Windows 98、Windows 98 Second Edition、Windows Millennium Edition、Windows NT 4.0、Windows NT Workstation 4.0、Windows NT Server 4.0、Windows 2000、Windows 2000 Server、Windows 2000 Advanced Server、Windows XP Home Edition、Windows XP Professional、Windows Server 2003、.NET Compact Framework - Mobile Phone、.NET Compact Framework、共通言語基盤 (CLI: Common Language Infrastructure) 規格

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

My.Computer.FileSystem.CopyDirectory メソッド
My.Computer.FileSystem.CopyFile メソッド
My.Computer.FileSystem.DeleteDirectory メソッド
My.Computer.FileSystem.DeleteFile メソッド
My.Computer.Network.DownloadFile メソッド
My.Computer.FileSystem.MoveDirectory メソッド
My.Computer.FileSystem.MoveFile メソッド
My.Computer.Network.UploadFile メソッド
UICancelOption

UIOption 列挙型

ファイルやディレクトリのコピー、削除、または移動のときに使用するダイアログ ボックスを指定します。

解説

この列挙型は
My.Computer.FileSystem.CopyDirectory メソッド、My.Computer.FileSystem.CopyFile メソッド、My.Computer.FileSystem.DeleteDirectory メソッド、My.Computer.FileSystem.DeleteFile メソッド、My.Computer.FileSyst
および My.Computer.FileSystem.MoveFile メソッド で使用します。

メンバ

メンバ	説明
OnlyErrorDialogs	エラー ダイアログ ボックスのみを表示し、プログレス ダイアログ ボックスは非表示にします。既定値です。
AllDialogs	プログレス ダイアログ ボックスおよびすべてのエラー ダイアログ ボックスを表示します。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : UIOption

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

My.Computer.FileSystem.CopyDirectory メソッド
My.Computer.FileSystem.CopyFile メソッド
My.Computer.FileSystem.DeleteDirectory メソッド
My.Computer.FileSystem.DeleteFile メソッド
My.Computer.FileSystem.MoveDirectory メソッド
My.Computer.FileSystem.MoveFile メソッド
Microsoft.VisualBasic.FileIO.UIOption

VariantType 列挙型

VarType 関数が返すバリアント オブジェクトの型を示します。

解説

VarType 関数を呼び出すときに、実際の値の代わりにコード内で列挙型メンバを使用できます。

VariantType 列挙型には、設定可能な **Variant** 型を識別するための定数が定義されています。次の表は、**VariantType** 列挙型メンバの一覧です。

メンバ

メンバ	定数	説明
Array	**vbArray**	配列。
Boolean	**vbBoolean**	**Boolean** (**True** または **False**)
Byte	**vbByte**	長整数型 (**Byte**)。0 ～ 255。
Char	**vbChar**	長整数型 (**Char**)。0 ～ 65535。
Currency	**vbCurrency**	通貨型 (**Currency**)。
DataObject		DataObject。
Date	**vbDate**	**Date** (0001 年 1 月 1 日 0:00:00 ～ 9999 年 12 月 31 日 11:59:59 PM)
Decimal	**vbDecimal**	**Decimal** (小数部分を持たない数値の場合、-79,228,162,514,264,337,593,543,950,335 ～ 79,228,162,514,264,337,593,543,950,335 の範囲の値をとります。小数点以下 28 桁の数値の場合、-7.9228162514264337593543950335 ～ 7.9228162514264337593543950335 の範囲の値をとります。絶対値の最小値は 0 を除いた場合、0.0000000000000000000000000001 です。)
Double	**vbDouble**	**Double** (負の値の場合、-1.79769313486231E+308 ～ -4.94065645841247E-324。正の値の場合、4.94065645841247E-324 ～ 1.79769313486231E+308。)
Empty	**vbEmpty**	NULL 参照。
Error		System.Exception
Integer	**vbInteger**	長整数型 (**Integer**)。-2,147,483,648 ～ 2,147,483,647。
Long	**vbLong**	長整数型 (**Long**)。-9,223,372,036,854,775,808 ～ 9,223,372,036,854,775,807。

Null	**vbNull**	NULL オブジェクト。
Object	**vbObject**	オブジェクト型 (**Object**) の変数には任意の型を格納できます。
Short		長整数型 (**Short**)。-32,768 ～ 32,767。
Single	**vbSingle**	**Single** (負の値の場合、-3.402823E+38 ～ -1.401298E-45。正の値の場合、1.401298E-45 ～ 3.402823E+38。)
String	**vbString**	**String** (0 個 ～ 約 20 億個の Unicode 文字。)
UserDefinedType	**vbUserDefinedType**	ユーザー定義型。構造体の各メンバの範囲はデータ型によって決まり、他のメンバの範囲とは関係しません。
Variant	**vbVariant**	**Variant**.

スマート デバイス開発者のためのメモ

この列挙型はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

VarType 関数 (Visual Basic)

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

VbStrConv 列挙型

StrConv 関数を呼び出すときに実行する変換の種類を示します。

解説

StrConv 関数を呼び出すときに、実際の値の代わりにコード内で次の列挙型のメンバを使用できます。*Conversion* 引数は、**VbStrConv** 列挙型のメンバを受け取ります。

メンバ

メンバ	定数	説明
UpperCase	**vbUpperCase**	文字列を大文字に変換します。
LowerCase	**vbLowerCase**	文字列を小文字に変換します。
ProperCase	**vbProperCase**	文字列の各単語の先頭の文字を大文字に変換します。
Wide	**vbWide**	文字列内の半角文字 (1 バイト) を全角文字 (2 バイト) に変換します。アジア ロケールに適用されます。
Narrow	**vbNarrow**	文字列内の全角文字 (2 バイト) を半角文字 (1 バイト) に変換します。アジア ロケールに適用されます。
なし		変換は行われません。
Katakana	**vbKatakana**	文字列内のひらがなをカタカナに変換します。ロケールの設定が日本の場合のみ有効です。
Hiragana	**vbHiragana**	文字列内のカタカナをひらがなに変換します。ロケールの設定が日本の場合のみ有効です。
SimplifiedChinese	**vbSimplifiedChinese**	中国語の繁体字を簡体字に変換します。
TraditionalChinese	**vbTraditionalChinese**	中国語の簡体字を繁体字に変換します。
LinguisticCasing	**vbLinguisticCasing**	大文字小文字の処理規則をファイル システムから文法に変換します。

スマート デバイス開発者のためのメモ

この列挙型はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 列挙型のメンバを参照する

関連項目

StrConv 関数

概念

組み込み定数と組み込み列挙型

列挙型を使用する状況

その他の技術情報

Visual Basic の文字列の概要

データ型の概要 (Visual Basic)

Visual Basic のデータ型の一覧を次の表に示します。それぞれのデータ型について、サポートされている共通言語ランタイムの型、ストレージ割り当ての公称サイズ、および値の範囲を示しています。

Visual Basic のデータ型	**共通言語ランタイムの型構造**	**ストレージ割り当ての公称サイズ**	**値の範囲**
ブール型	Boolean	実装するプラットフォームに依存	**True** または、次のようにも指定できます。**False**
バイト型	Byte	1 バイト	0 ～ 255 (符号なし)
Char (1 文字)	Char	2 バイト	0 ～ 65535 (符号なし)
日付	DateTime	8 バイト	0001 年 1 月 1 日 0:00:00 (午前 0 時) ～ 9999 年 12 月 31 日 11:59:59 PM
10 進数型	Decimal	16 バイト	0 ～ +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) † (小数点なし)、0 ～ +/-7.9228162514264337593543950335 (小数点以下 28 桁) 0 以外の最小数は +/-0.0000000000000000000000000001 (+/-1E-28) †
Double (倍精度浮動小数点数型)	Double	8 バイト	-1.79769313486231570E+308 ～ -4.94065645841246544E-324 † (負の値) 4.94065645841246544E-324 ～ 1.79769313486231570E+308 † (正の値)
整数型	Int32	4 バイト	-2,147,483,648 ～ 2,147,483,647 (符号付き)
Long (長整数型)	Int64	8 バイト	-9,223,372,036,854,775,808 ～ 9,223,372,036,854,775,807 (9.2...E+18 †) (符号付き)
オブジェクト型	Object (クラス)	32 ビット プラットフォームでは 4 バイト 64 ビット プラットフォームでは 8 バイト	オブジェクト型 (**Object**) の変数には任意の型を格納できます。
SByte	SByte	1 バイト	-128 ～ 127 (符号付き)
Short (短整数型)	Int16	2 バイト	-32,768 ～ 32,767 (符号付き)
Single (単精度浮動小数点数)	Single	4 バイト	-3.4028235E+38 ～ -1.401298E-45 † (負の値) 1.401298E-45 ～ 3.4028235E+38 † (正の値)
String (可変長)	String (クラス)	実装するプラットフォームに依存	0 個 ～ 約 20 億個の Unicode 文字
UInteger	UInt32	4 バイト	0 ～ 4,294,967,295 (符号なし)
ulong	UInt64	8 バイト	0 ～ 18,446,744,073,709,551,615 (1.8...E+19 †) (符号なし)
ユーザー定義 (構造体)	(ValueType から継承)	実装するプラットフォームに依存	構造体の各メンバの範囲はデータ型によって決まり、他のメンバの範囲とは関係しません。

ushort	UInt16	2 バイト	0 ～ 65,535 (符号なし)

†指数表記では、"E" は 10 のべき乗を表します。つまり、3.56E+2 は *3.56 x* 10^2 または 356 を表し、3.56E-2 は *3.56 / * 10^2 または 0.0356 を表します。

メモ：
テキストを含む文字列の場合は、StrConv 関数を使用してテキスト形式を変換できます。

メモリの使用量

基本データ型を宣言する場合、そのメモリの使用量がストレージ割り当ての公称サイズと同じであると仮定するのは安全ではありません。これは、次の考慮事項によるものです。

- **ストレージの割り当て** 共通言語ランタイムは、アプリケーションが実行されるプラットフォームの現在の特性に基づいてストレージを割り当てます。メモリがほぼ満杯である場合、宣言された要素を可能な限り 1 つにまとめます。その他の場合、メモリ アドレスを自然なハードウェアの境界まで配置し、パフォーマンスを最適化します。
- **プラットフォームの幅** ストレージ割り当ては、64 ビット プラットフォームの場合と 32 ビット プラットフォームの場合とで異なります。

複合データ型

構造体や配列などの複合データ型の各メンバに対しても、同じことが当てはまります。単に型のメンバのストレージ割り当ての公称サイズを合計するだけでは不十分です。さらに、次のような考慮事項もあります。

- **オーバーヘッド** 一部の複合型には、別のメモリ要件もあります。たとえば配列は、配列それ自体に対してと、各次元に対して、それぞれ別個のメモリを使用します。32 ビット プラットフォームでは、現在このオーバーヘッドは 12 バイトに加えて各次元について 8 バイトです。64 ビットでは、この要件が 2 倍になります。
- **ストレージのレイアウト** さらに、メモリ内に格納される順序が宣言の順序と同じであると仮定するのも安全ではありません。2 バイトまたは 4 バイトの境界など、バイトの配置を仮定することもできません。クラスまたは構造体を定義するときに、そのメンバのストレージのレイアウトを制御する必要がある場合、StructLayoutAttribute 属性をそのクラスまたは構造体に割り当てます。

オブジェクトのオーバーヘッド

基本データ型または複合データ型を参照するオブジェクト型 (**Object**) は、データ型に含まれるデータのほかに、さらに 4 バイトを使用します。

参照

関連項目

データ型変換関数
変換の概要
StrConv 関数
StructLayoutAttribute Class

概念

型文字
データ型の有効な使用方法

ブール型 (Boolean) (Visual Basic)

True または **False** の値だけを格納します。キーワード **True** および **False** は、**Boolean** 変数の 2 つの状態に相当します。

解説

ブール型 (**Boolean**) は、true/false、yes/no、または on/off など、2 つの状態を表す値を格納するために使用します。

ブール型 (**Boolean**) の既定値は **False** です。

型変換

Visual Basic で数値型の値をブール型 (**Boolean**) に変換すると、0 は **False** になり、その他の値はすべて **True** になります。Visual Basic でブール型 (**Boolean**) の値を数値型に変換すると、**False** は 0 になり、**True** は -1 になります。

Boolean 値と数値型との間で変換を行う場合は、.NET フレームワークの変換メソッドが必ずしも Visual Basic の変換キーワードと同じ結果を出力するとは限らないことに注意してください。これは、Visual Basic の変換処理が、前バージョンと互換性のある動作を保持しているからです。詳細については、「データ型のトラブルシューティング」を参照してください。

プログラミングのヒント

- **負の数。** ブール型 (**Boolean**) は数値型ではないので、負の値を表現できません。どのような場合でも、**Boolean** を使って数値を格納しないでください。
- **型文字。** ブール型 (**Boolean**) にはリテラルの型文字も、識別子の型文字もありません。
- **Framework の型。** .NET Framework において対応する型は、System.Boolean 構造体です。

使用例

次の例では、`runningVB` はブール型 (**Boolean**) の変数であり、単純な Yes/No の設定が格納されます。

```
Dim runningVB As Boolean
' Check to see if program is running on Visual Basic engine.
If scriptEngine = "VB" Then
    runningVB = True
End If
```

参照

処理手順

データ型のトラブルシューティング

関連項目

データ型の概要 (Visual Basic)
データ型変換関数
変換の概要
CType 関数
System.Boolean

概念

データ型の有効な使用方法

バイト型 (Byte) (Visual Basic)

0 から 255 までの符号なし 8 ビット (1 バイト) の整数です。

解説

Byte データ型はバイナリ データの格納に使用します。

Byte の既定値は 0 です。

プログラミングのヒント

- **負数Byte** は符号なしのデータ型なので、負数を表すことはできません。バイト型 (**Byte**) を評価する式で単項マイナス演算子 (-) を使用すると、Visual Basic では、最初に式が **Short** 型に変換されます。
- **形式の変換**Visual Basic がファイルを読み書きするとき、または DLL、メソッド、プロパティを呼び出すとき、データ形式が自動的に変換されます。バイト型 (**Byte**) の変数および配列に格納されるバイナリ データは、形式変換中に保存されます。バイナリ データに文字列型 (**String**) の変数を使用しないでください。これは、ANSI 形式と Unicode 形式の変換時にデータの内容が破損する場合があるためです。
- **拡大Byte** データ型は、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、**ULong**、**Decimal**、**Single**、または **Double** に拡大されます。これは、System.OverflowException エラーを発生させることなく、これらの型のいずれかに **Byte** を変換できることを意味します。
- **型宣言文字** **Byte** 型には、リテラルの型文字も識別子の型文字もありません。
- **Framework のデータ型**.NET Framework において対応する型は、System.Byte 構造体です。

参照

関連項目

データ型の概要 (Visual Basic)
データ型変換関数
変換の概要
System.Byte

概念

データ型の有効な使用方法

文字型 (Char) (Visual Basic)

0 から 65535 までの値を持つ符号なしの 16 ビット (2 バイト) コード ポイントを保持します。各コード ポイント (文字コード) は、単一の Unicode 文字を表します。

解説

単独の文字を扱う場合に **String** のオーバーヘッドを避けるには **Char** データ型を使用します。複数の文字を扱う場合に、**Char** 要素の配列である **Char()** を使用することもできます。

Char の既定値は、コード ポイント 0 を持つ文字です。

Unicode 文字

Unicode の最初の 128 個のコード ポイント (0 ～ 127) は、標準の英語版キーボードの文字と記号に対応しています。これら最初の 128 個のコード ポイントは、ASCII 文字セットで定義された文字と同じです。次の 128 個のコード ポイント (128 ～ 255) は、ラテン語系のアルファベット、アクセント記号、通貨記号、分数などの特殊文字を表します。Unicodeはその他のコード ポイント (256-65535) を使用して、各言語の文字、発音符、数学記号、技術記号などのさまざまな記号を表します。

IsDigit や IsPunctuation などのメソッドを **Char** 型の変数で使用すると、Unicode の分類を判別できます。

型変換

Visual Basic では、**Char** 型と数値型の直接変換は行われません。**Char** の値を、対応するコード ポイントを表す **Integer** に変換するには、Asc 関数、AscW 関数 を使用します。**Integer** の値を、このコード ポイントを持つ **Char** に変換するには、Chr 関数、ChrW 関数 を使用します。

型チェックのスイッチ (Option Strict ステートメント) がオンの場合、これが **Char** データ型であることを識別するために、リテラル型の文字を 1 文字の文字列に追加します。次に例を示します。

```
Option Strict On
Dim charVar As Char
' The following statement attempts to convert a String literal to Char.
' Because Option Strict is On, it generates a compiler error.
charVar = "Z"
' The following statement succeeds because it specifies a Char literal.
charVar = "Z"C
```

プログラミングのヒント

- **負数 Char** は符号なしのデータ型で、負の値を表すことはできません。数値を格納するのに **Char** は使用しません。
- **相互運用の考慮事項**オートメーション オブジェクトや COM オブジェクトなど、.NET Framework 向けに作成されていないコンポーネントを使用する場合、他の環境では文字型のデータ幅が異なる (8 ビット) ことに注意してください。そのようなコンポーネントに 8 ビットの引数を渡す場合は、新しい Visual Basic コードで、文字型 (**Char**) ではなく短整数型 (**Byte**) として宣言します。
- **拡大Char** データ型は、**String** に拡大されます。つまり、**Char** は System.OverflowException エラーにならずに **String** に変換できます。
- **型宣言文字**リテラルの型文字 **C** を 1 文字の文字列リテラルに追加すると、これは **Char** データ型に変換されます。**Char** には識別子の型文字はありません。
- **Framework のデータ型**.NET Framework において対応する型は、System.Char 構造体です。

参照

処理手順

方法 : unsigned 型を使用して正の整数のストレージを最適化する
方法 : 符号なしの型を使用する Windows の機能を呼び出す

関連項目

データ型の概要 (Visual Basic)
文字列型 (String) (Visual Basic)
データ型変換関数
変換の概要
Asc 関数、AscW 関数
Chr 関数、ChrW 関数
System.Char

概念

データ型の有効な使用方法

日付型 (Date) (Visual Basic)

IEEE 64 ビット (8 バイト) の整数として格納され、西暦 0001 年 1 月 1 日から西暦 9999 年 12 月 31 日までの日付と、午前 12:00:00 から午後 11:59:59.9999999 までの時刻を表します。増分は、グレゴリオ暦の西暦 1 年 1 月 1 日からの経過時間の 100 ナノ秒を表します。最大値は、西暦 10000 年 1 月 1 日の 100 ナノ秒前です。

解説

Date データ型は、日付、時刻、およびその両方の値を格納するのに使用します。

Date の既定値は 0001 年 1 月 1 日の 0:00:00 (午前 0 時) です。

書式の要件

Date リテラルは、シャープ記号 (**# #**) で囲む必要があります。日付の値は、`#5/31/1993#` のようにM/d/yyyy の形式で指定します。この要件は、ロケールやコンピュータの日付と時刻の設定とは無関係です。

この要件は、アプリケーションが実行されているロケールによってコードの意味が変わってしまうことがないようにするために設けられています。`#3/4/1998#` という **Date** リテラルをハードコーディングし、これを 1998 年 3 月 4 日の意味で使用したい場合、mm/dd/yyyy という形式を使用するロケールでは「3/4/1998」が意図したとおりにコンパイルされます。しかし、このアプリケーションが複数の国で使用される場合を考えてみましょう。dd/mm/yyyy という形式を使用するロケールでは、ハードコーディングされたリテラルは 1998 年 4 月 3 日としてコンパイルされます。yyyy/mm/dd という形式を使用するロケールでは、このリテラルは無効な値 (0003 年 4 月 1998 日) となり、コンパイラ エラーとなります。

問題回避

Date リテラルを、使用されるロケールの形式やカスタム形式に変換するには、リテラルを Format 関数 に渡し、定義済みの日付/時刻書式 (Format 関数) または ユーザー定義の日付/時刻書式 (Format 関数) を指定します。次にコード例を示します。

```
MsgBox("The formatted date is " & Format(#5/31/1993#, "dddd, d MMM yyyy"))
```

または、DateTime 構造体のいずれかのオーバーロードされたコンストラクタを使用して、日付と時刻の値をアセンブルします。次の例では、1993 年 5 月 31 日午後 12:14 を表す値を作成します。

```
Dim dateInMay As New System.DateTime(1993, 5, 31, 12, 14, 0)
```

時刻の形式

時刻の値は 12 時間制または 24 時間制で指定できます。たとえば、`#1:15:30 PM#` または `#13:15:30#` のようになります。ただし、分や秒を指定しない場合は、AM または PM を指定する必要があります。

日付と時刻の既定値

日時リテラルに日付が含まれない場合は、Visual Basic によって日付の部分の値が 0001 年 1 月 1 日に設定されます。日時リテラルに時刻が含まれない場合は、時刻の部分の値は 1 日の始まり、つまり深夜 0 時に設定されます。

型変換

Date の値を **String** 型に変換する場合、日付は実行時ロケールで指定された短い日付形式で表示され、時間は実行時ロケールで指定された時刻形式 (12 時間制または 24 時間制) で表示されます。

プログラミングのヒント

- **相互運用の考慮事項** オートメーションまたは COM オブジェクトのように、.NET Framework 向けに作成されていないコンポーネントとやり取りする場合、他の環境の日付/時刻の型は Visual Basic の **Date** 型と互換性がないことに注意してください。そのようなコンポーネントに日付/時刻の引数を渡す場合は、新しい Visual Basic のコードで、**Date** 型ではなく **Double** として宣言し、System.DateTime.FromOADate(System.Double) および System.DateTime.ToOADate の変換メソッドを使用します。
- **型宣言文字** **Date** 型には、リテラルの型文字も識別子の型文字もありません。ただし、コンパイラでは、**# #** (シャープ記号) で囲まれたリテラルは日付型 (**Date**) として処理されます。
- **Framework のデータ型**.NET Framework において対応する型は、**System.DateTime** 構造体です。

使用例

Date データ型の変数または定数は、日付と時刻の両方を格納します。次に例を示します。

```
Dim someDateAndTime As Date = #8/13/2002 12:14 PM#
```

参照

関連項目

データ型の概要 (Visual Basic)
データ型変換関数
変換の概要
System.DateTime

概念

データ型の有効な使用方法
データ型の変更点 (Visual Basic 6.0 ユーザー向け)

10 進型 (Decimal) (Visual Basic)

96 ビット (12 バイト) の整数に 10 の累乗 (可変) をスケーリングした値を表す 128 ビット (16 バイト) の符号付きの値を保持します。スケール ファクタには、小数点の右側の桁数を 0 ～ 28 の範囲で指定します。スケール ファクタが 0 (小数点なし) であれば、最大値は +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9228162514264337593543950335E+28) になります。小数点以下 28 桁の数値の場合、最大値は +/-7.9228162514264337593543950335 で、ゼロを除く最小値は +/-0.0000000000000000000000000001 (+/-1E-28) です。

解説

Decimal 型では、桁数が非常に大きい数値を扱うことができます。有効桁数は最大 29 で、*7.9228 x 10^28* を超える値を表現できます。特に、多くの桁数を必要とし、丸め誤差が許容されない財務計算などに適しています。

Decimal の既定値は 0 です。

プログラミングのヒント

- **精度。Decimal** は浮動小数点型のデータ型ではありません。**Decimal** 構造体には、バイナリ整数値と共に符号ビット、および値のどの部分が小数かを指定するスケール ファクタが整数で格納されます。このため、**Decimal** では、メモリ内で浮動小数点型 (**Single** および **Double**) よりも正確に数値を表現できます。
- **パフォーマンス。Decimal** は、すべての数値型の中で最もパフォーマンスの低いデータ型です。データ型を選択する前に、精度の重要性とパフォーマンスとを比較検討する必要があります。
- **拡大変換。Decimal** データ型は **Single** または **Double** に拡大変換されます。これは、System.OverflowException エラーを発生させることなく、これらの型のいずれかに **Decimal** を変換できることを意味します。
- **後続のゼロ** Visual Basic では、**Decimal** のリテラルに後続ゼロが格納されません。ただし、**Decimal** 型の変数は、計算に必要なすべての後続ゼロを保持します。次に例を示します。

  ```
  Dim d1, d2, d3, d4 As Decimal
  d1 = 2.375D
  d2 = 1.625D
  d3 = d1 + d2
  d4 = 4.000D
  MsgBox("d1 = " & CStr(d1) & ", d2 = " & CStr(d2) _
      & ", d3 = " & CStr(d3) & ", d4 = " & CStr(d4))
  ```

  この例で、**MsgBox** の出力は次のようになります。

  d1 = 2.375, d2 = 1.625, d3 = 4.000, d4 = 4

- **型文字** あるリテラルに型文字 **D** を付けると、そのリテラルは整数型 (**Decimal**) に変換されます。ある識別子に識別子の型文字 **@** を付けると、その識別子は整数型 (**Decimal**) に変換されます。
- **Framework の型** .NET Framework において対応する型は、System.Decimal 構造体です。

範囲

D の型文字を使用して、**Decimal** 型の変数または定数に大きな値を代入することが必要になる場合があります。次に例を示します。

```
Dim bigDec1 As Decimal = 9223372036854775807   ' No overflow.
Dim bigDec2 As Decimal = 9223372036854775808   ' Overflow.
Dim bigDec3 As Decimal = 9223372036854775808D  ' No overflow.
```

リテラルの後に型文字を指定しなければ、コンパイラはリテラルを **Long** 型と解釈します。`bigDec1` の宣言では、値が **Long** 型の範囲内にあるため、オーバーフローが発生しません。しかし、`bigDec2` の値は **Long** 型には大きすぎるため、コンパイル エラーが発生します。`bigDec3` では、リテラルの型文字 **D** を指定して、コンパイラにリテラルを **Decimal** と解釈させているため、エラーが発生しません。

参照

関連項目

データ型の概要 (Visual Basic)
単精度浮動小数点型 (Single) (Visual Basic)
倍精度浮動小数点数型 (Double) (Visual Basic)

データ型変換関数
変換の概要
System.Decimal
System.Decimal.#ctor(System.Int32[])
概念
データ型の有効な使用方法

倍精度浮動小数点数型 (Double) (Visual Basic)

IEEE 64 ビット (8 バイト) の符号付き浮動小数点数を格納します。負の値は -1.79769313486231570E+308 ～ -4.94065645841246544E-324、正の値は 4.94065645841246544E-324 ～ 1.79769313486231570E+308 の範囲の値をとります。倍精度の数値には、実数の近似値が格納されます。

解説

Double データ型に格納できる数値の範囲は、最大値はどのデータ型よりも大きく、最小値はどのデータ型よりも小さいという広範なものです。

Double の既定値は 0 です。

プログラミングのヒント

- **有効桁数** 浮動小数点数を操作する場合は、メモリ内に必ずしも正確に表現されるわけではないので注意してください。このため、値の比較や **Mod** 演算子など、特定の処理で予期しない結果が返される可能性があります。詳細については、「データ型のトラブルシューティング」を参照してください。
- **後続のゼロ** 浮動小数点のデータ型には、後続のゼロの文字に対する内部表現がありません。たとえば、4.2000 と 4.2 は区別されません。したがって、浮動小数点数の値を表示または印刷するとき、後続のゼロの文字は出力されません。
- **型文字** あるリテラルにリテラルの型文字 **R** を付けると、そのリテラルは倍精度浮動小数点型 (**Double**) に変換されます。ある識別子に識別子の型文字 **#** を付けると、その識別子は整数型 (**Double**) に変換されます。
- **Framework の型**.NET Framework において対応する型は、System.Double 構造体です。

参照

処理手順

データ型のトラブルシューティング

関連項目

データ型の概要 (Visual Basic)
10 進型 (Decimal) (Visual Basic)
単精度浮動小数点型 (Single) (Visual Basic)
データ型変換関数
変換の概要
System.Double

概念

データ型の有効な使用方法

整数型 (Integer) (Visual Basic)

-2,147,483,648 から 2,147,483,647 までの符号付き 32 ビット (4 バイト) の整数です。

解説

Integer データ型は、32 ビットのプロセッサでパフォーマンスが最大になります。他の整数型では、メモリとの間の読み込みと格納により長い時間がかかります。

Integer の既定値は 0 です。

プログラミングのヒント

- **相互運用の考慮事項**オートメーション オブジェクトや COM オブジェクトなど、.NET Framework 向けに作成されていないコンポーネントを使用する場合、他の環境では整数型 (**Integer**) のデータ幅が異なる (16 ビット) ことに注意してください。そのようなコンポーネントに 16 ビットの引数を渡す場合は、新しい Visual Basic のコードで、整数型 (**Integer**) ではなく短整数型 (**Short**) として宣言します。
- **拡大**Integer データ型は、**Long**、**Decimal**、**Single**、または **Double** に拡大されます。これは、System.OverflowException エラーを発生させることなく、これらの型のいずれかに **Integer** を変換できることを意味します。
- **型宣言文字**あるリテラルにリテラルの型文字 **I** を付けると、そのリテラルは **Integer** に変換されます。ある識別子に識別子の型文字 **%** を付けると、その識別子は整数型 (**Integer**) に変換されます。
- **Framework のデータ型**.NET Framework において対応する型は、System.Int32 構造体です。

範囲

整数型の変数をその型の範囲外の数値に設定しようとすると、エラーが発生します。小数に設定しようとすると、その数字は丸められます。次に例を示します。

```
' The valid range of an Integer variable is -2147483648 through +2147483647.
Dim k As Integer
' The following statement causes an error because the value is too large.
k = 2147483648
' The following statement sets k to 6.
k = CInt(5.9)
```

参照

関連項目

データ型の概要 (Visual Basic)
Long データ型 (Visual Basic)
Short 型 (Visual Basic)
データ型変換関数
変換の概要
System.Int32

概念

整数型 (Integer) (Visual Basic 6.0 ユーザー向け)
データ型の有効な使用方法

Long データ型 (Visual Basic)

-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 (9.2...E+18) までの符号付き 64 ビット (8 バイト) の整数です。

解説

大きすぎて **Integer** データ型に格納できない整数値を格納するには、**Long** データ型を使用します。

Long の既定値は 0 です。

プログラミングのヒント

- **相互運用の考慮事項**オートメーション オブジェクトや COM オブジェクトなど、.NET Framework 向けに作成されていないコンポーネントを使用する場合、他の環境では整数型 (**Long**) のデータ幅が異なる (32 ビット) ことに注意してください。そのようなコンポーネントに 32 ビットの引数を渡す場合は、新しい Visual Basic のコードで、整数型 (**Long**) ではなく短整数型 (**Integer**) として宣言します。

  また、Windows 95、Windows 98、Windows ME、Windows 2000 などでは、オートメーションは 64 ビット整数型をサポートしていません。これらのプラットフォームでは、Visual Basic の長整数型 (**Long**) の引数をオートメーション コンポーネントに渡すことはできません。

- **拡大Long** データ型は、**Decimal**、**Single**、または **Double** に拡大されます。これは、System.OverflowException エラーを発生させることなく、これらの型のいずれかに **Byte** を変換できることを意味します。

- **型宣言文字**あるリテラルにリテラルの型文字 **L** を付けると、そのリテラルは **Long** に変換されます。ある識別子に識別子の型文字 **&** を付けると、その識別子は整数型 (**Long**) に変換されます。

- **Framework のデータ型**.NET Framework において対応する型は、System.Int64 構造体です。

参照

関連項目

データ型の概要 (Visual Basic)
Int64 Structure
整数型 (Integer) (Visual Basic)
Short 型 (Visual Basic)
データ型変換関数
変換の概要

概念

整数型 (Integer) (Visual Basic 6.0 ユーザー向け)
データ型の有効な使用方法

オブジェクト型 (Object)

オブジェクトを参照する 32 ビット (4 バイト) のアドレスを格納します。オブジェクト (**Object**) 変数には、任意の参照型 (文字列、配列、クラス、またはインターフェイス) を割り当てることができます。また、オブジェクト型 (**Object**) の変数は、任意の値型 (数値、ブール型 (**Boolean**)、char 型 (**Char**)、日付型 (**Date**)、構造体、または列挙型) のデータを参照できます。

解説

オブジェクト型 (**Object**) は、任意のデータ型のデータをポイントできます。これにはアプリケーションが認識する任意のオブジェクト インスタンスも含まれます。オブジェクト型 (**Object**) は、コンパイル時に変数がどのデータ型をポイントするかわからない場合に使用します。

Object の既定値は **Nothing** (null 参照) です。

データ型

すべてのデータ型の変数、定数、または式を、オブジェクト型 (**Object**) の変数に代入できます。**Object** 変数が現在参照しているデータ型を調べるには、System.Type クラスの GetTypeCode メソッドを使用します。次に例を示します。

```
Dim myObject As Object
' Suppose myObject has now had something assigned to it.
Dim datTyp As Integer
datTyp = Type.GetTypeCode(myObject.GetType())
```

オブジェクト型 (**Object**) は参照型です。ただし、**Object** 変数が値型のデータを参照する場合は、値型として扱われます。

Storage

Object 変数はどのデータ型を参照する場合でも、データの値そのものを格納するのではなく、値へのポインタを格納します。このポインタはコンピュータのメモリ内で常に 4 バイトを占めますが、変数の値を表すデータの記憶領域は別に割り当てられます。コードはポインタを使用してデータを探すため、値型を保持する**オブジェクト型 (Object)** の変数は、明示的に型指定された変数よりも多少アクセスに時間がかかります。

プログラミングのヒント

- **相互運用のための注意事項** たとえば Automation オブジェクトや COM オブジェクトなど、.NET Framework 用に作成されていないコンポーネントを使用する場合は、他の環境のポインタ型に Visual Basic のオブジェクト型 (**Object**) との互換性がないことを頭に入れておく必要があります。
- **パフォーマンス。** オブジェクト型 (**Object**) を使って定義した変数は非常に柔軟であり、あらゆるオブジェクトへの参照を格納できます。しかし、このような変数からメソッドまたはプロパティを呼び出すと、必ず遅延バインディング (実行時の連結) が発生してしまいます。事前バインディング (コンパイル時の連結) を強制し、パフォーマンスを向上させるには、特定のクラス名を使って変数を宣言するか、または変数を特定のデータ型にキャストする必要があります。

  オブジェクト変数を宣言する場合は、汎用のオブジェクト型 (**Object**) 使用するのではなく、たとえば OperatingSystem など、できるだけ具体的なクラス型を使うようにしてください。また、Control ではなく TextBox など、最も具体的なクラスを使用してください。そうすることで、クラスのプロパティやメソッドにアクセスできます。通常、[オブジェクト ブラウザ] の [クラス] ボックスから、使用できるクラス名を探すことができます。
- **拡大変換。** すべてのデータ型とすべての参照型が、オブジェクト型 (**Object**) に拡大変換されます。これは、System.OverflowException エラーを発生させることなく、任意の型をオブジェクト型 (**Object**) に変換できることを意味します。

  ただし、値型とオブジェクト型 (**Object**) の間で変換を行うと、Visual Basic はボックス化およびボックス化解除と呼ばれる処理を実行するため、実行時間が長くなります。
- **型文字。** SByte 型 (**Object**) にはリテラルの型文字も、識別子の型文字もありません。
- **Framework の型。** .NET Framework において対応する型は、System.Object クラスです。

使用例

オブジェクト インスタンスをポイントする **Object** 変数の例は、次のようになります。

```
Dim objDb As Object
Dim myCollection As New Collection()
' Suppose myCollection has now been populated.
objDb = myCollection.Item(1)
```

参照

処理手順

方法 : 2 つのオブジェクトが関連しているかどうかを決める
方法 : 2 つのオブジェクトが同一であるかどうか判別する

関連項目

データ型の概要 (Visual Basic)
Object Class
データ型変換関数
変換の概要

概念

データ型の有効な使用方法
汎用データ型としてのオブジェクト型 (Object)

SByte 型 (Visual Basic)

-128 ～ 127 の範囲の、8 ビット (1 バイト) の符号付き整数値を格納します。

解説

SByte 型は、**Integer** のデータ幅全体、または **Short** のデータ幅の半分も必要としない整数値を格納する場合に使用します。場合によっては、共通言語ランタイムが **SByte** 変数をメモリ内で間を空けないようにパックし、メモリの消費量を減らすことも可能です。

SByte の既定値は 0 です。

プログラミングのヒント

- **CLS への準拠SByte** 型は、共通言語仕様 (CLS: Common Language Specification) の一部ではありません。したがって、CLS 準拠のコードでは、このデータ型を使用しているコンポーネントを使用できません。
- **拡大変換。SByte** データ型は、**Short**、**Integer**、**Long**、**Decimal**、**Single**、および **Double** に拡大変換されます。これは、System.OverflowException エラーを発生させることなく、これらの型のいずれかに **SByte** を変換できることを意味します。
- **型文字。** SByte 型 (**SByte**) にはリテラルの型文字も、識別子の型文字もありません。
- **Framework の型。**.NET Framework において対応する型は System.SByte 構造体です。

参照

関連項目

データ型の概要 (Visual Basic)
データ型変換関数
変換の概要
Short 型 (Visual Basic)
整数型 (Integer) (Visual Basic)
Long データ型 (Visual Basic)
System.SByte

概念

整数型 (Integer) (Visual Basic 6.0 ユーザー向け)
データ型の有効な使用方法

Short 型 (Visual Basic)

-32,768 ～ 32,767 の範囲の、16 ビット (2 バイト) の符号付き整数値を格納します。

解説

Short データ型は、整数型 (**Integer**) ほどのデータ 幅を必要としない整数値を格納するために使用します。場合によっては、共通言語ランタイムが **Short** 変数をメモリ内で間を空けないようにパックし、メモリの消費量を減らすことも可能です。

Short の既定値は 0 です。

プログラミングのヒント

- **拡大変換。Short** データ型は、**Integer**、**Long**、**Decimal**、**Single**、および **Double** に拡大変換されます。これは、System.OverflowException エラーを発生させることなく、これらの型のいずれかに **Short** を変換できることを意味します。
- **型文字。**あるリテラルにリテラルの型文字 **S** を付けると、そのリテラルは **Short** データ型に変換されます。**Short** には識別子の型文字がありません。
- **Framework の型。**.NET Framework において対応する型は System.Int16 構造体です。

参照

関連項目

データ型の概要 (Visual Basic)
データ型変換関数
変換の概要
整数型 (Integer) (Visual Basic)
Long データ型 (Visual Basic)
System.Int16

概念

整数型 (Integer) (Visual Basic 6.0 ユーザー向け)
データ型の有効な使用方法

単精度浮動小数点型 (Single) (Visual Basic)

IEEE 32 ビット (4 バイト) の符号付き単精度浮動小数点数を格納します。負の値は -3.4028235E+38 ～ -1.401298E-45、正の値は 1.401298E-45 ～ 3.4028235E+38 の範囲の値をとります。単精度の数値には、実数の近似値が格納されます。

解説

Single データ型は、倍精度浮動小数点型 (**Double**) ほどのデータ 幅を必要としない浮動小数点数を格納するために使用します。場合によっては、共通言語ランタイムが **Single** 変数をメモリ内で間を空けないようにパックし、メモリの消費量を減らすことも可能です。

Single の既定値は 0 です。

プログラミングのヒント

- **精度。** 浮動小数点数を操作する場合は、メモリ内に必ずしも正確に表現されるわけではないので注意してください。このため、値の比較や **Mod** 演算子など、特定の処理で予期しない結果が返される可能性があります。詳細については、「データ型のトラブルシューティング」を参照してください。
- **拡大変換。** **Single** データ型は **Double** に拡大変換されます。これは、System.OverflowException エラーを発生させることなく、**Single** を **Double** に変換できることを意味します。
- **後続のゼロ。** 浮動小数点のデータ型には、後続のゼロの文字に対する内部表現がありません。たとえば、4.2000 と 4.2 は区別されません。したがって、浮動小数点数の値を表示または印刷するとき、後続のゼロの文字は出力されません。
- **型文字。** あるリテラルに型文字 **F** を付けると、そのリテラルは単精度浮動小数点型 (**Single**) に変換されます。ある識別子に識別子の型文字 **!** を付けると、その識別子は整数型 (**Single**) に変換されます。
- **Framework の型。** .NET Framework において対応する型は、System.Single 構造体です。

参照

処理手順

データ型のトラブルシューティング

関連項目

データ型の概要 (Visual Basic)
10 進型 (Decimal) (Visual Basic)
倍精度浮動小数点数型 (Double) (Visual Basic)
データ型変換関数
変換の概要
System.Single

概念

データ型の有効な使用方法

文字列型 (String) (Visual Basic)

値の範囲が 0 ～ 65535 である、16 ビット (2 バイト) の符号なしコード ポイントのシーケンスを格納します。各コード ポイント (文字コード) は、単一の Unicode 文字を表します。文字列型 には、0 ～約 20 億個 (2^31) までの Unicode 文字を格納できます。

解説

String データ型を使用すると、**Char()** の配列管理のオーバーヘッド、つまり **Char** 要素の配列を管理するオーバーヘッドをかけることなく、複数の文字を格納できます。

String の既定値は **Nothing** (null 参照) です。これは空の文字列 (値 "") とは異なるので注意してください。

Unicode 文字

Unicode の最初の 128 個のコード ポイント (0 ～ 127) は、標準の英語版キーボードの文字と記号に対応しています。これら最初の 128 個のコード ポイントは、ASCII 文字セットで定義された文字と同じです。その次の 128 個のコード ポイント (128 ～ 255) は、ラテン語系のアルファベット、アクセント記号、通貨記号、分数などの特殊文字を表します。その他のコード ポイント (256-65535) は、各言語の文字、発音符、数学記号、技術記号などのさまざまな記号を表します。

String 変数に含まれる個々の文字に対して IsDigit や IsPunctuation などのメソッドを使用すると、Unicode の分類を判別できます。

書式

String のリテラルは、二重引用符 (" ") で囲む必要があります。文字列の 1 文字として二重引用符を含める場合は、二重引用符を 2 つ続けて ("") 記述します。次に例を示します。

```
Dim j As String = "Joe said ""Hello"" to me."
Dim h As String = "Hello"
' The following messages all display the same thing:
' "Joe said "Hello" to me."
MsgBox(j)
MsgBox("Joe said " & """ & h & """ & " to me.")
MsgBox("Joe said """ & h & """ to me.")
```

連続した二重引用符のうち、文字列に含まれる二重引用符は、**String** のリテラルを開始および終了する二重引用符と関係ないことに注意してください。

文字列操作の概要

文字列を **String** 変数に代入すると、その文字列は変更不可になります。つまり、文字列の長さや内容を変更できなくなります。何らかの方法で文字列を変更すると、Visual Basic は新しい文字列を作成し、変更前の文字列を破棄します。これにより、**String** 変数は新しい文字列をポイントするようになります。

String 変数の内容は、さまざまな文字列関数を使って操作できます。次に Left 関数 (Visual Basic) の例を示します。

```
Dim S As String = "Database"
' The following statement sets S to a new string containing "Data".
S = Microsoft.VisualBasic.Left(S, 4)
```

別のコンポーネントで作成された文字列には、先頭または末尾に空白が埋め込まれていることがあります。このような文字列を受け取った場合に、Trim、LTrim、RTrim 関数 を使用して空白を削除できます。

文字列操作の詳細については、「Visual Basic における文字列」を参照してください。

プログラミングのヒント

- **負の数 String** に格納される文字には符号がないため、負の値を表現できないことを覚えておいてください。どのような場合でも、**String** を使って数値を格納しないでください。
- **相互運用のための注意事項** たとえば Automation オブジェクトや COM オブジェクトなど、.NET Framework 用に作成されていないコンポーネントを使用する場合は、文字列の各文字のデータ幅 (8 ビット) が環境によって異なる場合があることを覚えておく必要があります。Visual Basic コードを作成する際、8 ビット文字の文字列引数をこのようなコンポーネントに渡す場合は、引数を **String** ではなく、**Byte()** として、つまり **Byte** 要素の配列として宣言してください。
- **型文字** 任意の識別子に識別子の型文字 **$** を付けると、その識別子は **String** データ型に変換されます。**String** にはリテラルの型文字がありません。ただし、コンパイラでは、二重引用符 (" ") で囲まれたリテラルは文字列型 (**String**) として処理されます。

- **Framework の型** .NET Framework において対応する型は、System.String クラスです。

参照

処理手順

方法 : unsigned 型を使用して正の整数のストレージを最適化する
方法 : 符号なしの型を使用する Windows の機能を呼び出す

関連項目

データ型の概要 (Visual Basic)
文字型 (Char) (Visual Basic)
データ型変換関数
変換の概要
System.String

概念

データ型の有効な使用方法

UInteger データ型

0 から 4,294,967,295 までの符号なし 32 ビット (4 バイト) の整数です。

解説

UInteger データ型では、符号なしの最大値を最も効率的なデータ幅で表示します。

UInteger の既定値は 0 です。

プログラミングのヒント

UInteger および **Integer** データ型は、32 ビット プロセッサで最良のパフォーマンスを発揮します。より小さな整数型 (**UShort**、**Short**、**Byte**、**SByte**) では、読み込み、格納、フェッチの際に、使用するビットは少なくて済みますが、より多くの時間がかかるためです。

参照

処理手順

方法 : unsigned 型を使用して正の整数のストレージを最適化する

方法 : 符号なしの型を使用する Windows の機能を呼び出す

関連項目

データ型の概要 (Visual Basic)

UInt32 Structure

データ型変換関数

変換の概要

概念

データ型の有効な使用方法

ULong データ型 (Visual Basic)

0 から 18,446,744,073,709,551,615 (1.84 の 10 ^ 18 倍よりも大きい) までの範囲の、符号なし 64 ビット (8 バイト) 整数値を格納します。

解説

ULong データ型は、**UInteger** には大きすぎるバイナリ データや、非常に大きい符号なし整数値を格納する場合に使用します。

ULong の既定値は 0 です。

プログラミングのヒント

- **負の数ULong** 型は符号を持たないので、負の数を表現できません。**ULong** 型を評価する式で単項マイナス演算子 (-) を使用すると、Visual Basic では、最初に式が **Decimal** 型に変換されます。
- **CLS への準拠ULong** データ型は、共通言語仕様 (CLS: Common Language Specification) の一部ではありません。したがって、CLS 準拠のコードでは、このデータ型を使用しているコンポーネントを使用できません。
- **相互運用のための注意事項**たとえば Automation オブジェクトや COM オブジェクトなど、.NET Framework 用に作成されていないコンポーネントを使用する場合は、**ulong** などの型のデータ幅 (32 ビット) が環境によって異なる場合があることを覚えておく必要があります。そのようなコンポーネントに 32 ビットの引数を渡す場合は、Visual Basic のマネージ コードで、**ULong** 型ではなく **UInteger** 型で宣言してください。

  また、Windows 95、Windows 98、Windows ME、Windows 2000 などでは、オートメーションは 64 ビット整数型をサポートしていません。これらのプラットフォームでは、Visual Basic の長整数型 (**ULong**) の引数をオートメーション コンポーネントに渡すことはできません。
- **拡大変換ULong** データ型は、**Decimal**、**Single**、および **Double** に拡大変換されます。これは、System.OverflowException エラーを発生させることなく、これらの型のいずれかに **ULong** を変換できることを意味します。
- **型宣言文字**あるリテラルにリテラルの型宣言文字 **UL** を付けると、そのリテラルは **ULong** データ型に変換されます。**ULong** には識別子の型宣言文字がありません。
- **Framework の型** .NET Framework において対応する型は、System.UInt64 構造体です。

参照

処理手順

方法 : 符号なしの型を使用する Windows の機能を呼び出す

関連項目

データ型の概要 (Visual Basic)
UInt64 Structure
データ型変換関数
変換の概要

概念

データ型の有効な使用方法

ユーザー定義型

定義した形式でデータを保持します。**Structure** ステートメントは形式を定義します。

Visual Basic の以前のバージョンでは、ユーザー定義型 (UDT) がサポートされていました。現在のバージョンでは、UDT を構造体に拡張したものが使用されています。構造体は、さまざまなデータ型の 1 つ以上のメンバを連結したものです。Visual Basic では、構造体は単独のユニットとして扱われますが、そのメンバに個別にアクセスすることもできます。

解説

複数のデータ型を結合して単一のユニットにする場合、または、どの基本データ型も要件を満たしていない場合に、構造体データ型を定義および使用します。

構造体データ型の既定値は、各メンバの既定値を組み合わせたものです。

宣言の形式

構造体宣言は、Structure ステートメント ステートメントと **End Structure** ステートメントの間に記述します。**Structure** ステートメントで構造体の名前を指定します。この名前は、構造体によって定義されるデータ型の識別子にもなります。コードの他の部分では、この識別子を使用して、この構造体のデータ型で変数、パラメータ、関数の戻り値を宣言できます。

Structure ステートメントと **End Structure** ステートメントの間の宣言によって、構造体のメンバが定義されます。

メンバのアクセス レベル

Dim ステートメント (Visual Basic) または、Public (Visual Basic)、Friend (Visual Basic)、Private (Visual Basic) などのアクセス レベルを指定するステートメントを使用して、すべてのメンバを宣言する必要があります。**Dim** ステートメントを使用すると、アクセス レベルは既定の public になります。

プログラミングのヒント

- **メモリの使用量。** 他のすべての複合データ型と同様に、構造体の総メモリ使用量を計算する場合、各メンバのストレージ割り当ての公称サイズを単に合計しただけでは安全ではありません。さらに、メモリ内に格納される順序が宣言の順序と同じであると仮定するのも安全ではありません。構造体のストレージ レイアウトを制御する必要がある場合は、StructLayoutAttribute 属性を **Structure** ステートメントに適用します。
- **相互運用のための注意事項** オートメーションまたは COM オブジェクトのように、.NET Framework 向けに作成されていないコンポーネントとやり取りする場合、他の環境のユーザー定義型は Visual Basic の構造体型と互換性がないことに注意してください。
- **拡大** 構造体データ型から、または構造体データ型に自動変換することはできません。Operator ステートメント を使用して構造体に変換演算子を定義し、各変換演算子を **Widening** または **Narrowing** として定義します。
- **型宣言文字** 構造体データ型には、リテラルの型文字も識別子の型文字もありません。
- **Framework のデータ型** .NET Framework にはこれに対応するデータ型はありません。すべての構造体は .NET Framework のクラス System.ValueType から継承されていますが、**System.ValueType** に対応する個別の構造体はありません。

使用例

構造体の宣言の概要を次に示します。

```
[Public | Protected | Friend | Protected Friend | Private] Structure structname
    {Dim | Public | Friend | Private} member1 As datatype1
    '  ...
    {Dim | Public | Friend | Private} memberN As datatypeN
End Structure
```

参照

関連項目

データ型の概要 (Visual Basic)
ValueType Class
データ型変換関数
変換の概要
Structure ステートメント
Widening
Narrowing

StructLayoutAttribute
概念
データ型の有効な使用方法
その他の技術情報
構造体 : 独自のデータ型

UShort 型 (Visual Basic)

0 ～ 65,535 の範囲の、16 ビット (2 バイト) の符号なし整数値を格納します。

解説

UShort データ型は、**Byte** を使うには大きすぎるバイナリ データを格納する際に使用します。

UShort の既定値は 0 です。

プログラミングのヒント

- **負の数UShort** 型は符号を持たないので、負の数を表現できません。**UShort** 型を評価する式で単項マイナス演算子 (-) を使用すると、Visual Basic では、最初に式が **Integer** 型に変換されます。
- **CLS への準拠UShort** データ型は、共通言語仕様 (CLS: Common Language Specification) の一部ではありません。したがって、CLS 準拠のコードでは、このデータ型を使用しているコンポーネントを使用できません。
- **拡大変換UShort** データ型は、**Integer**、**UInteger**、**Long**、**ULong**、**Decimal**、**Single**、および **Double** に拡大変換されます。これは、System.OverflowException エラーを発生させることなく、これらの型のいずれかに **UShort** を変換できることを意味します。
- **型文字**あるリテラルにリテラルの型文字 **US** を付けると、そのリテラルは **UShort** データ型に変換されます。**UShort** には識別子の型文字がありません。
- **Framework の型**.NET Framework において対応する型は、System.UInt16 構造体です。

参照

処理手順

方法 : 符号なしの型を使用する Windows の機能を呼び出す

関連項目

データ型の概要 (Visual Basic)

UInt16 Structure

データ型変換関数

変換の概要

概念

データ型の有効な使用方法

ディレクティブ (Visual Basic)

The topics in this section document the Visual Basic source code compiler directives.

このセクションの内容

#Const ディレクティブ

#ExternalSource ディレクティブ

#If...Then...#Else ディレクティブ

#Region ディレクティブ

関連するセクション

Visual Basic リファレンス

Visual Basic

#Const ディレクティブ

Visual Basic の条件付きコンパイル定数を定義します。

```
#Const constname = expression
```

指定項目

constname

必ず指定します。定義する定数の名前を文字列 (**String**) で指定します。

expression

必ず指定します。リテラル値、その他の条件付きコンパイル定数、算術演算子や論理演算子 (**Is** を除く) を任意に組み合わせた式を指定します。

解説

条件付きコンパイル定数は、定義されているファイル内でだけ参照できるプライベート定数です。**#Const** ディレクティブを使ってパブリックなコンパイル定数を作成することはできません。パブリックなコンパイル定数はユーザー インターフェイス内かまたは **/define** コンパイラ オプションを使ってしか作成できません。

expression に使用できるのは、条件付きコンパイル定数とリテラル値だけです。**Const** で定義された通常の定数を使用すると、エラーが発生します。逆に言えば、**#Const** キーワードで定義された定数は、条件付きコンパイル以外には使えません。定数を未定義のまま使用することもできます。未定義の定数の値は **Nothing** になります。

使用例

#Const ディレクティブの使用例を次に示します。

VB

```
#Const MyLocation = "USA"
#Const Version = "8.0.0012"
#Const CustomerNumber = 36
```

参照

処理手順

方法 : 条件付きコンパイル定数を宣言する

関連項目

/define (Visual Basic)

#If...Then...#Else ディレクティブ

Const ステートメント (Visual Basic)

If...Then...Else ステートメント (Visual Basic)

概念

条件付きコンパイルの概要

#ExternalSource ディレクティブ

ソース コードの特定行とソース外部のテキストと間の対応付けを指定します。

```
#ExternalSource( StringLiteral , IntLiteral )
    [ LogicalLine+ ]
#End ExternalSource
```

指定項目
StringLiteral

外部ソースのパス。

IntLiteral

外部ソースの最初の行の行番号。

LogicalLine

外部ソースでエラーが発生する行。

#End ExternalSource

#ExternalSource ブロックを終了します。

解説

このディレクティブは、コンパイラとデバッガだけに使用されます。

ソース ファイルには、外部ソース ディレクティブを含めることができます。ソース ディレクティブは、ソース ファイル内のコードの特定行と、.aspx ファイルなど、ソース外部のテキストとの対応付けを指定します。指定されたソース コードでコンパイル中にエラーが発生した場合、エラーは外部ソースからのものであると示されます。

外部ソース ディレクティブは、コンパイルには影響を与えず、入れ子になることもありません。これらのディレクティブは、アプリケーションの内部で使用するためだけに用意されています。

参照
その他の技術情報
条件付きコンパイル (Visual Basic)

#If...Then...#Else ディレクティブ

式の値に基づいて、条件付きのコンパイルを行います。

```
#If expression Then
      statements
[ #ElseIf expression Then
   [ statements ]
...
#ElseIf expression Then
   [ statements ] ]
[ #Else
   [ statements ] ]
#End If
```

指定項目

expression

#If ステートメントおよび **#ElseIf** ステートメント に対しては必ず指定します。それ以外の場合は省略できます。真 (**True**) または偽 (**False**) を評価する、1 つ以上の条件付きコンパイル定数、リテラル値、および演算子だけで構成される任意の式を指定します。

statements

#If ステートメント ブロックに対しては必ず指定します。それ以外の場合は省略できます。関連付けられた式が真 (**True**) に評価される場合にコンパイルされる、Visual Basic のプログラム行またはコンパイラ ディレクティブを指定します。

#End If

#If ステートメント ブロックを終了します。

解説

#If...Then...#Else ディレクティブの動作は、**If...Then...Else** ステートメントの動作と表面上は同じに見えます。しかし、**#If...Then...#Else** ディレクティブはコンパイラが何をコンパイルするかを評価するのに対し、**If...Then...Else** ステートメントは実行時に条件を評価する点が異なります。

通常、条件付きコンパイルは、同じプログラムを異なるシステムでコンパイルするために使用します。また、デバッグ コードが実行可能ファイルに表示されるのを防ぐためにも使用します。条件付きコンパイルによって省略されたコードは、コンパイル後の実行可能ファイルからは取り除かれるため、プログラムのサイズや動作に影響しません。

評価結果に関係なく、すべての式が **Option Compare Binary** を使って評価されます。**Option Compare** ステートメントは、**#If** ステートメントと **#ElseIf** ステートメントの式に影響しません。

メモ：

#If、**#Else**、**#ElseIf**、および **#End If** ディレクティブを単体で使用することはできません。他のコードをこれらのディレクティブと同じ行に記述することはできません。

使用例

この例では、**#If...Then...#Else** ブロックを使用して、特定のステートメントをコンパイルするかどうかを決定します。

VB

```
#Const CustomerNumber = 36
#If CustomerNumber = 35 Then
        ' Insert code to be compiled for customer # 35.
#ElseIf CustomerNumber = 36 Then
        ' Insert code to be compiled for customer # 36.
#Else
        ' Insert code to be compiled for all other customers.
#End If
```

参照

関連項目

#Const ディレクティブ

If...Then...Else ステートメント (Visual Basic)

条件付きコンパイル定数

概念

条件付きコンパイルの概要

#Region ディレクティブ

Visual Basic ファイルのコードをセクションごとに折りたたんで非表示にします。

```
#Region "identifier_string"
#End Region
```

指定項目
identifier_string

必ず指定します。領域が折りたたまれたときにその領域のタイトルとして機能する文字列です。既定では、領域は折りたたまれています。

#End Region

#Region ブロックを終了します。

解説

Visual Studio コード エディタのアウトライン機能を使うときに展開または折りたたみされるコード ブロックを指定するには、**#Region** ディレクティブを使用します。**#Region** ステートメントは、ブロック セマンティクス (**#If**...**#End If** など) をサポートしており、開始点と終了点が同じコード ブロックの中にある必要があります。

使用例

#Region ディレクティブの使用例を次に示します。

VB

```
#Region "MathFunctions"
    ' Insert code for the Math functions here.
#End Region
```

参照
処理手順
方法 : コードをアウトライン表示する/非表示にする
方法 : コードのセクションを折りたたんで非表示にする
関連項目
#If...Then...#Else ディレクティブ

関数 (Visual Basic)

ここで紹介するトピックには、Visual Basic のランタイム メンバ関数の表が掲載されています。

このセクションの内容

関連するセクション

Visual Basic リファレンス

Visual Basic

変換関数 (Visual Basic)

Asc 関数

AscW 関数

CBool 関数

CByte 関数

CChar 関数

CDate 関数

CDbl 関数

CDec 関数

Chr 関数

ChrW 関数

CInt 関数

CLng 関数

CObj 関数

CSByte 関数

CShort 関数

CSng 関数

CStr 関数

CType 関数

CUInt 関数

CULng 関数

CUShort 関数

Format 関数

Hex 関数

Oct 関数

Str 関数

Val 関数

参照

関連項目

データ型変換関数

数値演算関数 (Visual Basic)

Visual Basic 6 の数値演算関数は、.NET Framework の System.Math クラスに含まれる同等なメソッドに置き換えられています。

解説

.NET Framework の数値演算関数は、Visual Basic 6 の関数と機能的には同じですが、名前が一部異なっているものがあります。たとえば、.NET Framework で Visual Basic 6 の **Atn** 関数に相当するのは Atan です。Visual Basic 6 の数値演算関数名とそれに対応する .NET Framework メソッド名を次の表に示します。

Visual Basic 6 の関数	.NET Framework メソッド	説明
Abs	Abs	指定された数値の絶対値を返します。
Atn	**Atan**	指定された数値をタンジェントとする角度を倍精度浮動小数点数型 (**Double**) の値で返します。
Cos	Cos	指定した角度のコサインを含む倍精度浮動小数点数型 (**Double**) の値を返します。
Exp	Exp	指定された数値を指数とする *e* (自然対数の底) の累乗を含む **Double** の値を返します。
Log	Log	指定された数値の対数を含む倍精度浮動小数点数型 (**Double**) の値を返します。このメソッドはオーバーロードされ、指定された数値の自然対数 (定数 *e* を底とする)、または指定された底における指定された数値の対数を返します。
Round	Round	指定された値に最も近い数値を含む倍精度浮動小数点数型 (**Double**) の値を返します。round に関するその他の関数は、Round など組み込み型のメソッドとして利用できます。
Sgn	Sign	引数に指定された数式の符号を表す **Integer** の値を返します。
Sin	Sin	指定した角度のサインを倍精度浮動小数点数型 (**Double**) の値で返します。
Sqr	Sqrt	数式の平方根を倍精度浮動小数点数型 (**Double**) の値で返します。
Tan	Tan	指定した角度のタンジェントを倍精度浮動小数点数型 (**Double**) の値で返します。

また、.NET Framework の数値演算クラスには、三角関数、対数関数、およびその他の数値演算関数に対応する定数やその他の静的なメソッドがあります。これらはすべて Visual Basic プログラムで使用できます。

これらの関数を修飾子なしで使用するには、次のコードをソース コードの先頭に追加して、**System.Math** 名前空間をプロジェクトにインポートします。

```
'Imports System.Math
```

使用例

この例では、**Math** クラスの **Abs** メソッドを使って、数値の絶対値を計算します。

```
' Returns 50.3.
Dim MyNumber1 As Double = Math.Abs(50.3)
' Returns 50.3.
Dim MyNumber2 As Double = Math.Abs(-50.3)
```

この例では、**Math** クラスの **Atan** メソッドを使って、πの値を計算します。

```
Public Function GetPi() As Double
    ' Calculate the value of pi.
```

```
    Return 4.0 * Math.Atan(1.0)
End Function
```

この例では、**Math** クラスの **Cos** メソッドを使って、角度のコサインを返します。

```
Public Function Sec(ByVal angle As Double) As Double
    ' Calculate the secant of angle, in radians.
    Return 1.0 / Math.Cos(angle)
End Function
```

この例では、**Math** クラスの **Exp** メソッドを使って、指定した数値を指数とする *e* の累乗を返します。

```
Public Function Sinh(ByVal angle As Double) As Double
    ' Calculate hyperbolic sine of an angle, in radians.
    Return (Math.Exp(angle) - Math.Exp(-angle)) / 2.0
End Function
```

この例では、**Math** クラスの **Log** メソッドを使って、数値の自然対数を返します。

```
Public Function Asinh(ByVal value As Double) As Double
    ' Calculate inverse hyperbolic sine, in radians.
    Return Math.Log(value + Math.Sqrt(value * value + 1.0))
End Function
```

この例では、**Math** クラスの **Round** メソッドを使って、数値を最も近い整数に丸めます。

```
' Returns 3.
Dim MyVar2 As Double = Math.Round(2.8)
```

この例では、**Math** クラスの **Sign** メソッドを使って、数の符号を判定します。

```
' Returns 1.
Dim MySign1 As Integer = Math.Sign(12)
' Returns -1.
Dim MySign2 As Integer = Math.Sign(-2.4)
' Returns 0.
Dim MySign3 As Integer = Math.Sign(0)
```

この例では、**Math** クラスの **Sin** メソッドを使って、角度のサイン値を返します。

```
Public Function Csc(ByVal angle As Double) As Double
    ' Calculate cosecant of an angle, in radians.
    Return 1.0 / Math.Sin(angle)
End Function
```

この例では、**Math** クラスの **Sqrt** メソッドを使って、数値の平方根を計算します。

```
' Returns 2.
Dim MySqr1 As Double = Math.Sqrt(4)
' Returns 4.79583152331272.
Dim MySqr2 As Double = Math.Sqrt(23)
' Returns 0.
Dim MySqr3 As Double = Math.Sqrt(0)
' Returns NaN (not a number).
Dim MySqr4 As Double = Math.Sqrt(-4)
```

この例では、**Math** クラスの **Tan** メソッドを使って、角度のタンジェントを返します。

```
Public Function Ctan(ByVal angle As Double) As Double
    ' Calculate cotangent of an angle, in radians.
```

```
        Return 1.0 / Math.Tan(angle)
    End Function
```

必要条件

クラス : **Math**

名前空間 : System

アセンブリ : mscorlib (mscorlib.dll 内)

参照
関連項目
Rnd 関数 (Visual Basic)
Randomize 関数 (Visual Basic)
数値演算関数の導出 (Visual Basic)
NaN

データ型変換関数

データ型変換関数は、インラインでコンパイルされます。つまり、変換コードは、式を評価するコードに含まれます。変換を実行するためにプロシージャへの呼び出しを行わないようにすると、パフォーマンスが向上することがあります。各関数は式を特定のデータ型に変換します。

```
CBool(expression)
CByte(expression)
CChar(expression)
CDate(expression)
CDbl(expression)
CDec(expression)
CInt(expression)
CLng(expression)
CObj(expression)
CSByte(expression)
CShort(expression)
CSng(expression)
CStr(expression)
CUInt(expression)
CULng(expression)
CUShort(expression)
```

指定項目
expression

必ず指定します。変換前のデータ型の任意の式です。

戻り値のデータ型

返される値のデータ型は、次に示すように、関数名によって異なります。

関数名	戻り値の型	引数 *expression* の範囲
CBool	ブール型 (Boolean) (Visual Basic)	任意の有効な **Char**、**String**、または数式。
CByte	バイト型 (Byte) (Visual Basic)	0 から 255 (符号なし)。小数部分は丸められます[1]。
CChar	文字型 (Char) (Visual Basic)	任意の有効な **Char** または **String** 式。**String** の最初の文字だけが変換されます。値は 0 から 65535 (符号なし)。
CDate	日付型 (Date) (Visual Basic)	任意の有効な日付と時刻の表現。
CDbl	倍精度浮動小数点数型 (Double) (Visual Basic)	-1.79769313486231570E+308 ～ -4.94065645841246544E-324 (負の値)。 4.94065645841246544E-324 ～ 1.79769313486231570E+308 (正の値)。

CDec	10 進型 (Decimal) (Visual Basic)	小数点以下が 0 桁 (小数部分を持たない数値) の場合、-79,228,162,514,264,337,593,543,950,335 ～ 79,228,162,514,264,337,593,543,950,335。小数点以下 28 桁の数値の場合、-7.9228162514264337593543950335 ～ 7.9228162514264337593543950335。絶対値の最小値は 0 を除いた場合、0.0000000000000000000000000001 (+/-1E-28) です。
CInt	整数型 (Integer) (Visual Basic)	-2,147,483,648 から 2,147,483,647。小数部分は丸められます[1]。
CLng	Long データ型 (Visual Basic)	-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807。小数部分は丸められます[1]。
CObj	オブジェクト型 (Object)	任意の有効な式。
CSByte	SByte 型 (Visual Basic)	-128 から 127。小数部分は丸められます[1]。
CShort	Short 型 (Visual Basic)	-32,768 から 32,767。小数部分は丸められます[1]。
CSng	単精度浮動小数点型 (Single) (Visual Basic)	-3.402823E+38 ～ -1.401298E-45 (負の値)。1.401298E-45 ～ 3.402823E+38 (正の値)。
CStr	文字列型 (String) (Visual Basic)	**CStr** 関数の戻り値は引数 *expression* により異なります。CStr 関数の戻り値 を参照してください。
CUInt	UInteger データ型	0 から 4,294,967,295 (符号なし)。小数部分は丸められます[1]。
CULng	ULong データ型 (Visual Basic)	0 から 18,446,744,073,709,551,615 (符号なし)。小数部分は丸められます[1]。
CUShort	UShort 型 (Visual Basic)	0 から 65,535 (符号なし)。小数部分は丸められます[1]。

[1] 小数部分は、銀行型丸めと呼ばれる特別な丸めの対象になります。詳細については、「解説」を参照してください。

解説

規則として、Visual Basic の型変換関数は、Convert クラスと個々の型構造体またはクラスの両方で `ToString()` などの .NET Framework メソッドよりも優先して使用します。Visual Basic 関数は、Visual Basic コードへの対応に最適化されており、ソース コードをより短く、より読みやすくします。さらに、**Boolean** を **Integer** に変換する場合など、.NET Framework の変換メソッドは Visual Basic 関数と同じ結果を生成するとは限りません。詳細については、「データ型のトラブルシューティング」を参照してください。

動作

- **強制型変換。** 一般に、ある操作結果を既定のデータ型ではなく特定のデータ型に強制的に変換する場合は、データ型変換関数を使用します。たとえば、単精度、倍精度、または整数で計算を行うような場合に、**CDec** 関数を使用して 10 進の演算を強制的に行います。

- **変換の失敗。** 関数に渡された引数 *expression* の値が変換されるデータ型の範囲を超えている場合、OverflowException が発生します。

- **小数部分。** 非整数型の値を整数型に変換する場合、整数変換関数 (**CByte**、**CInt**、**CLng**、**CSByte**、**CShort**、**CUInt**、**CULng**、**CUShort**) は、最も近い整数に値を丸めて小数部分を取り除きます。

  小数部分がちょうど 0.5 のとき、整数変換関数はこれを最も近い偶数に丸めます。たとえば、0.5 は 0 に、1.5 と 2.5 はどちらも 2 になります。これは銀行型丸めとも呼ばれ、このような数値を大量に加算するときに蓄積する偏りを調整する目的で使用されます。

  Int 関数、Fix 関数 (Visual Basic) は小数部分を丸めるのではなく切り捨てるので、**CInt** と **CLng** とは異なります。また、**Fix** 関数および **Int** 関数は引き渡された値と同じデータ型で常に値を返します。

- **日付/時刻の変換。** 値を日付または時刻に変換できるかどうかを確認するには、IsDate 関数 (Visual Basic) を使用します。**CDate** は日付リテラルと時刻リテラルを認識しますが、数値は認識しません。Visual Basic 6.0 の **Date** 値を Visual Basic 2005 の **Date** 値に変換するには、System.DateTime.FromOADate(System.Double) メソッドを使用します。

- **日付/時刻の基準値。** 日付型 (Date) (Visual Basic) には、常に日付と時刻の両方の情報が含まれます。型変換のために、Visual Basic は 1/1/0001 (西暦 1 年 1 月 1 日) を日付の基準値と見なし、00:00:00 (午前 0 時) を時刻の基準値と見なします。日付型 (**Date**) の値を文字列に変換する場合、**CStr** は結果の文字列に基準値を含めません。たとえば、`#January 1, 0001 9:30:00#` を文字列に変換すると、結果は "9:30:00 AM" となり、日付情報が省略されます。ただし、元の日付型 (**Date**) の値には日付情報が残っており、DatePart 関数 (Visual Basic) などの関数を使って復元できます。

- **カルチャの区別。** 文字列を扱う型変換関数は、アプリケーションの現在のカルチャ設定に基づいて変換を実行します。たとえば、**CDate** は、システムのロケール情報に基づいて日付の形式を認識します。日、月、および年は、ロケールに対応した正しい順序で指定する必要があります。そうしないと、日付が正しく解釈されない場合があります。日付の形式が、曜日文字列 ("Wednesday" など) を含む長い形式の場合も認識できません。

  ロケールで指定されているもの以外の書式の、値の文字列形式を扱った変換では、Visual Basic の型変換関数は使用できません。この変換では、その値の型の `ToString(IFormatProvider)` および `Parse(String, IFormatProvider)` メソッドを使用します。たとえば、文字列を **Double** に変換するには System.Double.Parse(System.String,System.IFormatProvider) を使用します。**Double** 型の値を文字列に変換するには System.Double.ToString(System.IFormatProvider) を使用します。

CType 関数

CType 関数 は、2 番目の引数として *typename* を取り、*expression* を強制的に *typename* に変換します (*typename* は、有効な変換が可能な任意のデータ型、構造体、クラス、インターフェイス)。

CType とその他の型変換キーワードとの比較については、「DirectCast」および「TryCast」を参照してください。

CBool の例

CBool 関数を使って式をブール型 (**Boolean**) の値に変換する例を次に示します。式が 0 でない値のときは、**CBool** 関数は **True** を返します。それ以外は **False** を返します。

VB

```
Dim a, b, c As Integer
Dim check As Boolean
a = 5
b = 5
' The following line of code sets check to True.
check = CBool(a = b)
c = 0
' The following line of code sets check to False.
check = CBool(c)
```

CByte の例

CByte 関数を使って式を (**Byte**) に変換する例を次に示します。

VB

```
Dim aDouble As Double
Dim aByte As Byte
aDouble = 125.5678
' The following line of code sets aByte to 126.
aByte = CByte(aDouble)
```

CChar の例

CChar 関数を使って、文字列型 (**String**) の式の最初の文字を char 型 (**Char**) に変換する例を次に示します。

VB

```
Dim aString As String
Dim aChar As Char
' CChar converts only the first character of the string.
aString = "BCD"
' The following line of code sets aChar to "B".
aChar = CChar(aString)
```

CChar の入力引数は**Char** または **String** データ型である必要があります。**CChar** には数値データ型を入力できないため、**CChar** を使って数値を文字に変換することはできません。次の例では、コード ポイント (文字コード) を表す数値を取得し、それを対応する文字に変換します。ここでは、InputBox 関数 (Visual Basic) を使って数字の文字列を取得し、**CInt** を使って文字列を整数型 (**Integer**) に変換し、**ChrW** を使ってその数値を **Char** に変換しています。

VB

```
Dim someDigits As String
Dim codePoint As Integer
Dim thisChar As Char
someDigits = InputBox("Enter code point of character:")
codePoint = CInt(someDigits)
' The following line of code sets thisChar to the Char value of codePoint.
thisChar = ChrW(codePoint)
```

CDate の例

CDate 関数を使って文字列を日付型 (**Date**) の値に変換する例を次に示します。一般的には、この例のように文字列で日付/時刻を表すことはお勧めできません。文字列の代わりに、日付リテラルや時刻リテラル (#Feb 12, 1969# や #4:45:23 PM# など) を使ってください。

VB

```
Dim aDateString, aTimeString As String
Dim aDate, aTime As Date
aDateString = "February 12, 1969"
aTimeString = "4:35:47 PM"
' The following line of code sets aDate to a Date value.
aDate = CDate(aDateString)
' The following line of code sets aTime to Date value.
aTime = CDate(aTimeString)
```

CDbl の例

VB

```
Dim aDec As Decimal
Dim aDbl As Double
' The following line of code uses the literal type character D to make aDec a Decimal.
aDec = 234.456784D
```

```
' The following line of code sets aDbl to 1.9225456288E+1.
aDbl = CDbl(aDec * 8.2D * 0.01D)
```

CDec の例

CDec 関数を使って数値を **Decimal** に変換する例を次に示します。

VB

```
Dim aDouble As Double
Dim aDecimal As Decimal
aDouble = 10000000.0587
' The following line of code sets aDecimal to 10000000.0587.
aDecimal = CDec(aDouble)
```

CInt の例

CInt 関数を使って値を **Integer** に変換する例を次に示します。

VB

```
Dim aDbl As Double
Dim anInt As Integer
aDbl = 2345.5678
' The following line of code sets anInt to 2346.
anInt = CInt(aDbl)
```

CLng の例

CLng 関数を使って値を **Long** に変換する例を次に示します。

VB

```
Dim aDbl1, aDbl2 As Double
Dim aLng1, aLng2 As Long
aDbl1 = 25427.45
aDbl2 = 25427.55
' The following line of code sets aLng1 to 25427.
aLng1 = CLng(aDbl1)
' The following line of code sets aLng2 to 25428.
aLng2 = CLng(aDbl2)
```

CObj の例

CObj 関数を使って数値を **Object** に変換する例を次に示します。オブジェクト型 (**Object**) の変数自体には、4 バイトのポインタだけが含まれます。このポインタは、変数に代入された倍精度浮動小数点数型 (**Double**) の値を指しています。

VB

```
Dim aDouble As Double
Dim anObject As Object
aDouble = 2.7182818284
' The following line of code sets anObject to a pointer to aDouble.
anObject = CObj(aDouble)
```

CSByte の例

CSByte 関数を使って数値を **SByte** に変換する例を次に示します。

VB

```
Dim aDouble As Double
Dim anSByte As SByte
aDouble = 39.501
```

```
' The following line of code sets anSByte to 40.
anSByte = CSByte(aDouble)
```

CShort の例

CShort 関数を使って数値を **Short** に変換する例を次に示します。

VB

```
Dim aByte As Byte
Dim aShort As Short
aByte = 100
' The following line of code sets aShort to 100.
aShort = CShort(aByte)
```

CSng の例

CSng 関数を使って値を **Single** に変換する例を次に示します。

VB

```
Dim aDouble1, aDouble2 As Double
Dim aSingle1, aSingle2 As Single
aDouble1 = 75.3421105
aDouble2 = 75.3421567
' The following line of code sets aSingle1 to 75.34211.
aSingle1 = CSng(aDouble1)
' The following line of code sets aSingle2 to 75.34216.
aSingle2 = CSng(aDouble2)
```

CStr の例

CStr 関数を使って数値を **String** に変換する例を次に示します。

VB

```
Dim aDouble As Double
Dim aString As String
aDouble = 437.324
' The following line of code sets aString to "437.324".
aString = CStr(aDouble)
```

CStr 関数を使って **Date** の値を **String** の値に変換する例を次に示します。

VB

```
Dim aDate As Date
Dim aString As String
' The following line of code generates a COMPILER ERROR because of invalid format.
' aDate = #February 12, 1969 00:00:00#
' Date literals must be in the format #m/d/yyyy# or they are invalid.
' The following line of code sets the time component of aDate to midnight.
aDate = #2/12/1969#
' The following conversion suppresses the neutral time value of 00:00:00.
' The following line of code sets aString to "2/12/1969".
aString = CStr(aDate)
' The following line of code sets the time component of aDate to one second past midnight.
aDate = #2/12/1969 12:00:01 AM#
' The time component becomes part of the converted value.
' The following line of code sets aString to "2/12/1969 12:00:01 AM".
aString = CStr(aDate)
```

CStr は常に、現在のロケールに対する標準の短い形式 ("6/15/2003 4:35:47 PM " など) で日付型 (**Date**) の値を表示します。しか

し、**CStr** は 1/1/0001 (日付) および 00:00:00 (時刻) の基準値を表示しません。

CStr が返す値の詳細については、「CStr 関数の戻り値」を参照してください。

CUInt の例

CUInt 関数を使って数値を **UInteger** に変換する例を次に示します。

VB

```
Dim aDouble As Double
Dim aUInteger As UInteger
aDouble = 39.501
' The following line of code sets aUInteger to 40.
aUInteger = CUInt(aDouble)
```

CULng の例

CULng 関数を使って数値を **ULong** に変換する例を次に示します。

VB

```
Dim aDouble As Double
Dim aULong As ULong
aDouble = 39.501
' The following line of code sets aULong to 40.
aULong = CULng(aDouble)
```

CUShort の例

CUShort 関数を使って数値を **UShort** に変換する例を次に示します。

VB

```
Dim aDouble As Double
Dim aUShort As UShort
aDouble = 39.501
' The following line of code sets aUShort to 40.
aUShort = CUShort(aDouble)
```

参照

関連項目

変換関数 (Visual Basic)
Asc 関数、AscW 関数
Chr 関数、ChrW 関数
Int 関数、Fix 関数 (Visual Basic)
Format 関数
Hex 関数 (Visual Basic)
Oct 関数
Str 関数
Val 関数

その他の技術情報

Visual Basic における型変換

CStr 関数の戻り値

異なる型の *expression* に対する **CStr** の戻り値を次の表に示します。

***expression* の型**	**CStr の戻り値**
ブール型 (Boolean) (Visual Basic)	"True" または "False" を表す文字列
日付型 (Date) (Visual Basic)	システムで設定されている短い形式の日付で **Date** 型の値 (日付と時刻) を表す文字列
数値データ型	数値を表す文字列

CStr と日付データ型

日付データ型 (**Date**) には、常に日付と時刻の両方の情報が含まれます。型変換のために、Visual Basic は 1/1/0001 (西暦 1 年 1 月 1 日) を日付の基準値と見なし、00:00:00 (午前 0 時) を時刻の基準値と見なします。**CStr** では、基準値が結果の文字列に含まれません。たとえば、`#January 1, 0001 9:30:00#` を文字列に変換すると、結果は "9:30:00 AM" となり、日付情報が省略されます。ただし、元の日付型 (**Date**) の値には日付情報が残っており、DatePart 関数 (Visual Basic) などの関数を使って復元できます。

メモ：

CStr 関数は、アプリケーションにおける現在のカルチャ設定に基づいて変換処理を実行します。特定のカルチャにおける数の文字表現を調べるには、その数の `ToString(IFormatProvider)` メソッドを使用します。たとえば、倍精度浮動小数点型 (**Double**) の値を文字列型 (**String**) に変換するときには、System.Double.ToString(System.IFormatProvider) を使用します。

参照

関連項目

データ型変換関数
ブール型 (Boolean) (Visual Basic)
日付型 (Date) (Visual Basic)
DatePart 関数 (Visual Basic)

関数 A ～ C

次の表は、Visual Basic のランタイム メンバ関数の一覧です。

AppActivate	Asc	AscW	Beep
CallByName	CBool	CByte	CChar
CDate	CDbl	CDec	ChDir
ChDrive	Choose	Chr	ChrW
CInt	CLng	CObj	Command
CreateObject	CShort	CSng	CStr
CType	CurDir		

バージョン 6.0 以前に Visual Basic のランタイム メンバ関数として利用できた数値演算関数 **Abs**、**Atn**、および **Cos** は、.NET Framework クラス ライブラリの Math クラスのメソッド (Abs、Atan、および Cos) に置き換えられました。詳細については、「数値演算関数 (Visual Basic)」を参照してください。

スマート デバイス開発者のためのメモ

スマート デバイス アプリケーションでは、**AppActivate**、**Beep**、**CallByName**、**ChDir**、**ChDrive**、**CreateObject**、および **CurDir** はサポートされません。

参照

関連項目

関数 D ～ G
関数 H ～ L
関数 M ～ R
関数 S ～ Z

その他の技術情報

Visual Basic リファレンス

AppActivate 関数

既に動作しているアプリケーションをアクティブにします。

```
Public Overloads Sub AppActivate( _
      ByVal { Title As String | ProcessId As Integer } _
)
```

パラメータ

Title

アクティブにするアプリケーションのタイトル バーに表示されるタイトルを文字列 (**String**) 式で指定します。アプリケーションを起動したときアプリケーションに割り当てたタイトルを使用します。

ProcessId

このプロセスに割り当てられる Win32 プロセス ID 番号を表す整数 (**Integer**) を指定します。Shell 関数 から返された ID を使用します。ただし、ゼロでないことが必要です。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Title* または *ProcessId* が見つかりません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

AppActivate を使用すると、アプリケーションのアクティブなウィンドウにフォーカスを移すことができます。アクティブなウィンドウへのハンドルや参照がない場合もあれば、特定の瞬間にどのウィンドウがアクティブであるかわからない場合さえあります。そのような場合は、Focus メソッドを使用できません。

AppActivate 関数は、指定したアプリケーションやウィンドウにフォーカスを移します。このとき、フォーカスが移っても、指定したウィンドウの状態は変化しません。たとえば、最小化されているウィンドウにフォーカスを移しても、そのウィンドウは最小化されたままです。ユーザーがフォーカスを変更するか、またはウィンドウを閉じると、他のウィンドウへフォーカスが移ります。アプリケーションを起動するには、**Shell** 関数を使用します。また、ウィンドウ スタイルの設定をする場合にも **Shell** 関数を使用します。

Title パラメータを使用した場合、**AppActivate** は大文字小文字を区別せずに比較しますが、使用しない場合はタイトル バーの文字とまったく同じであることが必要です。最初にトップ レベル ウィンドウが検索され、次に子ウィンドウが検索されます。一致するウィンドウが見つからない場合は、**ArgumentException** がスローされます。

AppActivate は、ウィンドウを持つプロセスに対してだけ使用できます。ほとんどのコンソール アプリケーションはウィンドウを持っていません。つまり、そのようなアプリケーションは **AppActivate** が検索するプロセスの一覧には含まれません。コンソール アプリケーションから実行する場合、システムはアプリケーションを実行するために別のプロセスを作成し、コンソール プロセスに出力を返します。したがって、現在のプロセス ID を要求すると、コンソール アプリケーションのプロセス ID ではなく、この別のプロセスの ID が返されます。

実行時に、**AppActivate** 関数は、タイトルが *Title* に一致する、またはプロセス ID が *ProcessId* に一致する実行中のアプリケーションをすべてアクティブにします。完全に一致するものが見つからないときは、アプリケーション ウィンドウのタイトル バーの文字列が *Title* で終わるアプリケーションをアクティブにします。*Title* に一致するアプリケーション ウィンドウが複数ある場合、**AppActivate** 関数はその中の 1 つを任意に選択してアクティブにします。

メモ :

AppActivate 関数には SafeTopLevelWindows レベルの **UIPermission** が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「アクセス許可の要求」および「UIPermission Class」を参照してください。

使用例

AppActivate 関数を使って、アプリケーション ウィンドウをアクティブにする例を次に示します。メモ帳のプロセスが実行中でない場合、この例は **ArgumentException** をスローします。アプリケーションの実行に **Shell** プロシージャを使用していますが、指定されたパスにアプリケーションの実行プログラムが保存されていない場合は、アプリケーションを実行することはできません。

VB

```vb
Dim notepadID As Integer
' Activate a running Notepad process.
AppActivate("Untitled - Notepad")
' AppActivate can also use the return value of the Shell function.
' Shell runs a new instance of Notepad.
notepadID = Shell("C:\WINNT\NOTEPAD.EXE", AppWinStyle.NormalFocus)
' Activate the new instance of Notepad.
AppActivate(notepadID)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
Shell 関数
Focus
ArgumentException

Asc 関数、AscW 関数

文字に対応する文字コードを表す整数型 (**Integer**) の値を返します。

```
Public Overloads Function Asc(ByVal String As Char) As Integer
Public Overloads Function AscW(ByVal String As Char) As Integer
' -or-
Public Overloads Function Asc(ByVal String As String) As Integer
Public Overloads Function AscW(ByVal String As String) As Integer
```

パラメータ

String

必ず指定します。任意の有効な文字 (**Char**) または文字列 (**String**) 式。*String* が文字列 (**String**) 式である場合は、文字列の最初の文字だけが入力として使用されます。*String* が **Nothing** であるか、または文字を含んでいない場合は、ArgumentException エラーが発生します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*String* が空または長さが 0 です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Asc は、入力文字に対するコード ポイント (文字コード) を返します。これは、1 バイト文字セット (SBCS: Single-Byte Character Set) 値に対しては 0 ～ 255、2 バイト文字セット (DBCS: Double-Byte Character Set) 値に対しては -32768 ～ 32767 です。1 バイト文字セットである ASCII の表については、「ASCII 文字コード」を参照してください。

返される値は、System.Globalization 名前空間にある TextInfo クラスの ANSICodePage プロパティに含まれている現在のスレッドのコード ページによって異なります。**ANSICodePage** を取得するには、`System.Globalization.CultureInfo.CurrentCulture.TextInfo.ANSICodePage` を指定します。

AscW は、入力文字に対する Unicode コード ポイントを返します。これは 0 ～ 65535 の値です。返される値は、現在のスレッドのカルチャやコード ページ設定には依存しません。

メモ :

以前のバージョンの Visual Basic の **AscB** 関数は、文字ではなくバイトに対するコードを返していました。これは主に、2 バイト文字セット (DBCS) アプリケーションで文字列を変換するために使用します。Visual Basic 2005 のすべての文字列は Unicode で、**AscB** は現在サポートされていません。

使用例

Asc 関数を使って、各文字列の最初の文字に対応する 整数型 (Integer) (Visual Basic) の文字コードを返す例を次に示します。

VB

```
Dim codeInt As Integer
' The following line of code sets myInt to 65.
codeInt = Asc("A")
' The following line of code sets myInt to 97.
codeInt = Asc("a")
' The following line of code sets myInt to 65.
codeInt = Asc("Apple")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Chr 関数、ChrW 関数
変換関数 (Visual Basic)
データ型変換関数
整数型 (Integer) (Visual Basic)
System.Globalization
CultureInfo
ANSICodePage
ArgumentException

その他の技術情報

ASCII 文字コード

Beep 関数

コンピュータのスピーカーを鳴らします。

```
Public Sub Beep()
```

解説

ビープ音の鳴る時間と音程はハードウェアとシステム ソフトウェアに依存するため、コンピュータの機種によって異なります。

メモ：

Beep 関数には SafeTopLevelWindows レベルの **UIPermission** が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「アクセス許可の要求」および「UIPermission Class」を参照してください。

使用例

Beep 関数を使って、コンピュータのスピーカーから途切れのない長い音を鳴らす例を次に示します。

VB

```
Dim I As Integer
For I = 1 To 100   ' Loop 100 times.
   Beep   ' Sound a tone.
Next I
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Visual Basic ランタイム ライブラリのメンバ

その他の技術情報

サウンドの再生

CallByName 関数

オブジェクトに対してメソッドを実行します。また、オブジェクトのプロパティを設定または取得します。

```
Public Function CallByName( _
   ByVal ObjectRef As System.Object, _
   ByVal ProcName As String, _
   ByVal UseCallType As CallType, _
   ByVal Args() As Object _
) As Object
```

パラメータ

ObjectRef

必ず指定します。オブジェクト型 (**Object**) です。プロパティまたはメソッドを公開するオブジェクトへのポインタを指定します。

ProcName

必ず指定します。文字列型 (**String**) です。オブジェクトのプロパティまたはメソッドの名前を含む文字列式を指定します。

UseCallType

必ず指定します。呼び出されるプロシージャの種類を表す CallType 列挙型 型の列挙体のメンバです。**CallType** の値は、**Method**、**Get**、**Set** のいずれかです。

Args

(省略可能。) パラメータ配列型 (**ParamArray**) です。呼び出されるプロパティまたはメソッドに渡す引数を含むパラメータ配列を指定します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*UseCallType* の値が無効です。**Method**、**Get**、または **Set** のいずれかを指定してください。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

CallByName 関数は、実行時に使用され、プロパティの取得、プロパティの設定、またはメソッドの呼び出しを実行します。

使用例

CallByName 関数の次の使用例では、テキスト ボックスの **Text** プロパティを設定し、2 行目では **Text** プロパティの値を取得します。3 行目では、**Move** メソッドを呼び出してテキスト ボックスを移動します。

VB

```
' Imports statements must be at the top of a module.
Imports Microsoft.VisualBasic.CallType
```

VB

```
Sub TestCallByName1()
    'Set a property.
    CallByName(TextBox1, "Text", CallType.Set, "New Text")

    'Retrieve the value of a property.
    MsgBox(CallByName(TextBox1, "Text", CallType.Get))

    'Call a method.
    CallByName(TextBox1, "Hide", CallType.Method)
End Sub
```

CallByName 関数を使ってコレクション オブジェクトの **Add** メソッドと **Item** メソッドを呼び出す例を次に示します。

VB

```
Public Sub TestCallByName2()
    Dim col As New Collection()

    'Store the string "Item One" in a collection by
    'calling the Add method.
    CallByName(col, "Add", CallType.Method, "Item One")

    'Retrieve the first entry from the collection using the
    'Item property and display it using MsgBox().
    MsgBox(CallByName(col, "Item", CallType.Get, 1))
End Sub
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

CallType 列挙型

ArgumentException Class

概念

パラメータ配列

文字列名によるプロパティまたはメソッドの呼び出し

ChDir 関数

現在のディレクトリまたはフォルダを変更します。

My 機能により、**ChDir** 関数よりも、ファイルの I/O 操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.FileSystem.CurrentDirectory プロパティ」を参照してください。

```
Public Sub ChDir(ByVal Path As String)
```

パラメータ

Path

必ず指定します。新しい既定のディレクトリやフォルダになるディレクトリやフォルダを識別する文字列 (**String**) 式を指定します。*Path* にはドライブが含まれる場合もあります。ドライブを指定しない場合、**ChDir** 関数は現在のドライブ上の既定ディレクトリまたは既定フォルダを変更します。

例外

例外の種類	エラー番号	条件
ArgumentException	52	*Path* は空です。
FileNotFoundException	76	指定されたドライブが無効です。または利用できません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

ChDir 関数は既定ディレクトリを変更しますが、既定ドライブは変更しません。たとえば、既定ドライブが C の場合は、次のステートメントにより D ドライブの既定ディレクトリが変更されますが、C ドライブは引き続き既定ドライブです。

VB

```
ChDir("D:\TMP")
```

相対的なディレクトリの変更は、次のように 2 つのピリオドを入力することで行います。

VB

```
ChDir("..") ' Moves up one directory.
```

セキュリティに関するメモ：

ChDir 関数にはアンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission クラス」と「コード アクセス許可」を参照してください。

使用例

ChDir 関数を使って、現在のディレクトリまたはフォルダを変更するコード例を次に示します。

VB

```
' Change current directory or folder to "MYDIR".
ChDir("MYDIR")

' Assume "C:" is the current drive. The following statement changes
' the default directory on drive "D:". "C:" remains the current drive.
ChDir("D:\WINDOWS\SYSTEM")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

ChDrive 関数
CurDir 関数
Dir 関数
MkDir 関数
RmDir 関数
ArgumentException Class
FileNotFoundException Class

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

ChDrive 関数

現在のドライブを変更します。

```
Public Overloads Sub ChDrive(ByVal Drive As { Char | String })
```

パラメータ
Drive

必ず指定します。既存のドライブを表す文字列式を指定します。長さ 0 の文字列 ("") を指定すると、現在のドライブは変更されません。引数 *Drive* が複数の文字で構成される文字列の場合、**ChDrive** 関数は先頭の文字だけを使用します。

例外

例外の種類	エラー番号	条件
IOException	68	指定されたドライブが無効です。または利用できません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

ChDrive 関数にはアンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、SecurityPermission Class と コード アクセス許可 についての説明を参照してください。

使用例

ChDrive 関数を使って、現在のドライブを変更するコード例を次に示します。ドライブが存在しない場合、この関数は例外をスローします。

VB

```
ChDrive("D")   ' Make "D" the current drive.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
ChDir 関数
CurDir 関数
MkDir 関数
RmDir 関数
IOException Class
その他の技術情報
Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

Choose 関数

引数のリストから値を選択して返します。

```
Public Function Choose( _
   ByVal Index As Double, _
   ByVal ParamArray Choice() As Object _
) As Object
```

パラメータ

Index

必ず指定します。倍精度浮動小数点型 (**Double**) です。1 から引数 *Choice* で渡される要素数の範囲内の値を返す数式を指定します。

Choice

必ず指定します。オブジェクト型 (**Object**) のパラメータ配列です。単一の変数を指定するか、またはオブジェクト (**Object**) データ型、コンマで区切られたオブジェクト型 (**Object**) の変数や式のリスト、要素がオブジェクト型 (**Object**) の 1 次配列を返す式を指定できます。

解説

Choose 関数は、*Choice()* で渡されるリストのメンバを *Index* の値に基づいて返します。*Index* が 1 であれば、リストの最初のメンバが選択され、*Index* が **UBound**(*Choice()*) であれば、リストの最後のメンバが選択されます。*Index* が範囲外の場合、**Choose** は **Nothing** 値を返します。

引数 *Index* が整数でない場合は、最も近い整数に丸められてから評価されます。

Choose 関数を使用すると、選択可能な値のリストから特定の値を検索できます。

メモ :

引数リスト内の式には、関数呼び出しが含まれることがあります。**Choose** の呼び出しに使用する引数リストを用意するために、Visual Basic コンパイラは、すべての式のすべての関数を呼び出します。このため、異なる式が *Index* で選択される場合は、呼び出されていない特定の関数に依存できません。

使用例

Choose 関数を使ってプロシージャに渡されたインデックス (パラメータ `Ind`) に対応する名前を返すコード例は、次のとおりです。

VB

```
Function GetChoice(ByVal Ind As Integer) As String
    GetChoice = CStr(Choose(Ind, "Speedy", "United", "Federal"))
End Function
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

IIf 関数

Select...Case ステートメント (Visual Basic)

Switch 関数

Chr 関数、ChrW 関数

指定された文字コードに対応する文字を返します。

```
Public Function Chr(ByVal CharCode As Integer) As Char
Public Function ChrW(ByVal CharCode As Integer) As Char
```

パラメータ

CharCode

必ず指定します。文字に対するコード ポイント、つまり文字コードを表す整数型 (**Integer**) の式です。*CharCode* に無効な範囲のコードを指定すると、ArgumentException エラーが発生します。**Chr** に指定可能な範囲は 0 ～ 255、**ChrW** に指定可能な範囲は -32768 ～ 65535 です。

例外

例外の種類	エラー番号	条件
ArgumentException	5	**ChrW** の *CharCode* が -32768 より小さいか、65535 より大きい。
ArgumentException	5	**Chr** の *CharCode* が 0 より小さいか、255 より大きい。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

CharCode に設定できる値の範囲は非対称であり、短整数型 (Short 型 (Visual Basic)) と整数型 (整数型 (Integer) (Visual Basic)) の間にあるストレージ範囲の違いがこれによって吸収されます。たとえば、-29183 は **Short** 型の値ですが、+36353 は **Integer** 型の値です。これにより、Visual Basic 6.0 との互換性も容易になります。

Chr は、System.Text 名前空間の Encoding クラスを使って、現在のスレッドが 1 バイト文字セット (SBCS) と 2 バイト文字セット (DBCS) のどちらを使用するのかを判断します。それに基づいて、*CharCode* を適切なセットのコード ポイントとして受け取ります。SBCS 文字の範囲は 0 ～ 255 で、DBCS 文字の範囲は -32768 ～ 65535 です。1 バイト文字セットである ASCII の表については、「ASCII 文字コード」を参照してください。

返される値は、System.Globalization 名前空間にある TextInfo クラスの ANSICodePage プロパティに含まれている現在のスレッドのコード ページによって異なります。**ANSICodePage** を取得するには、`System.Globalization.CultureInfo.CurrentCulture.TextInfo.ANSICodePage` を指定します。

ChrW は、*CharCode* を Unicode のコード ポイントとして受け取ります。コードの範囲は、文化や現在のスレッドに対するコード ページの設定には依存しません。-32768 ～ -1 の範囲の値は、+32768 ～ +65535 の範囲の値と同じように扱われます。

0 ～ 31 の範囲の値は、標準の ASCII コードと同じで表示できません。たとえば、`Chr(10)` はライン フィード文字を返します。

メモ：

旧バージョンの Visual Basic に含まれる **ChrB** 関数は、1 バイト文字を返します。これは主に、2 バイト文字セット (DBCS) アプリケーションで文字列を変換するために使用します。Visual Basic および .NET Framework のすべての文字列は Unicode で、**ChrB** はサポートされていません。

使用例

次のコード例では、**Chr** 関数を使って、指定された文字コードに対応する文字を返します。

VB

```
Dim associatedChar As Char
' Returns "A".
associatedChar = Chr(65)
' Returns "a".
associatedChar = Chr(97)
' Returns ">".
associatedChar = Chr(62)
```

```
' Returns "%".
associatedChar = Chr(37)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
Asc 関数、AscW 関数
Str 関数
変換関数 (Visual Basic)
データ型変換関数
CultureInfo
ArgumentException

その他の技術情報

ASCII 文字コード

Command 関数

Visual Basic 自体または Visual Basic で開発した実行可能プログラムを起動させるために使用するコマンド ラインの引数の部分を返します。

My 機能は、**Command** 関数よりも高い生産性とパフォーマンスを実現します。詳細については、「My.Application.CommandLineArgs プロパティ」を参照してください。

```
Public Function Command() As String
```

解説

引数が返された後で、空白、バックスラッシュ、スラッシュ、ハイフン、引用符などの一般的なデリミタを検索し、各パラメータを分割、または文字列を検索します。

アプリケーションを Visual Basic で開発したり、EXE ファイルをコンパイルする場合、**Command** 関数は `MyApp(cmdlineargs)` のように、コマンド ライン上のアプリケーションの名前の後の引数を返します。

使用例

次に示すのは、**Command** 関数を使用して、配列を格納したオブジェクトのコマンド ライン引数を返す例です。

VB

```
Function GetCommandLineArgs() As String()
    ' Declare variables.
    Dim separators As String = " "
    Dim commands As String = Microsoft.VisualBasic.Interaction.Command()
    Dim args() As String = commands.Split(separators.ToCharArray)
    Return args
End Function
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Visual Basic ランタイム ライブラリのメンバ
Visual Basic プログラムの構造
Environment.GetCommandLineArgs Method
My.Application.CommandLineArgs プロパティ

概念

Visual Basic バージョンの Hello World!

その他の技術情報

Visual Basic コンパイラ

CreateObject 関数 (Visual Basic)

COM オブジェクトへの参照を作成して返します。**CreateObject** 関数を使って Visual Basic 内のクラスのインスタンスを作成するには、そのクラスを COM コンポーネントとして明示的に示す必要があります。

```
Public Shared Function CreateObject( _
   ByVal ProgId As String, _
   Optional ByVal ServerName As String = "" _
) As Object
```

パラメータ

ProgId

必ず指定します。文字列型 (**String**) です。作成するオブジェクトのプログラム ID です。

ServerName

省略可能です。文字列型 (**String**) です。オブジェクトが作成されるネットワーク サーバーの名前を指定します。引数 *ServerName* が空の文字列 ("") の場合、ローカル コンピュータが使用されます。

例外

例外の種類	エラー番号	条件
Exception	429	*ProgId* が見つかりません。または指定されていません。 または *ServerName* が DnsValidateName 関数の実行に失敗する場合の原因のほとんどは、63 文字より長かったり、無効な文字が含まれていたりすることです。
Exception	462	サーバーを利用できません。
FileNotFoundException	53	指定された型のオブジェクトが存在していません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

COM コンポーネントのインスタンスを作成するには、**CreateObject** 関数の戻り値をオブジェクト変数に代入します。

```
Sub CreateADODB()
   Dim adoApp As Object
   adoApp = CreateObject("ADODB.Connection")
End Sub
```

戻り値の格納に使用するオブジェクト変数の型は、アプリケーションのパフォーマンスに影響する場合があります。**As Object** 句を使用してオブジェクト変数を宣言すると、任意の型のオブジェクトへの参照を含む変数が作成されます。ただし、その変数を経由したオブジェクトへのアクセスは、遅延バインディングであり、プログラムの実行時にバインディングされます。アプリケーションのパフォーマンス低下などのさまざまな理由から、遅延バインディングは使用しないことを推奨します。

プログラムがコンパイルされたときにバインディングする事前バインディングを行うオブジェクト変数を作成できます。このためには、[プロジェクト] メニューの [参照の追加] ダイアログ ボックスの [COM] タブで、オブジェクトのタイプ ライブラリへの参照を追加します。次に、オブジェクトの特定の型のオブジェクト変数を宣言します。ほとんどの場合、**CreateObject** 関数を使用するよりも、**Dim** ステートメントとプライマリ相互運用機能アセンブリを使用してオブジェクトを作成する方が効果的です。

アンマネージ コードへの対応

さらに、COM オブジェクトがアンマネージ コードを使用するという問題があります。アンマネージ コードとは、共通言語ランタイムの利点を持たないコードです。Visual Basic のマネージ コードに COM のアンマネージ コードを混在させると、非常に複雑な状態になります。COM オブジェクトへの参照を追加すると、ライブラリのプライマリ相互運用機能アセンブリ (PIA) が検索され、見つかった場合は使用されます。PIA が見つからない

場合、Visual Basic は COM ライブラリ内の各クラスに対するローカルな相互運用性クラスを含む相互運用性アセンブリを作成します。詳細については、「.NET Framework アプリケーションにおける COM 相互運用性」を参照してください。

一般的には、できる限り、厳密にバインドされたオブジェクトとプライマリ相互運用機能アセンブリを使用する必要があります。**CreateObject** 関数を Microsoft Office オブジェクトと共に使用する例を次に示します。この例は、デモンストレーションのためだけに作成されています。ただし、これらのオブジェクトを、適切なプライマリ相互運用機能アセンブリと共に使用すると、より簡単でより信頼性が高まります。

リモート コンピュータでオブジェクトを作成する

コンピュータ名を **CreateObject** 関数の引数 *ServerName* に渡すことで、リモート ネットワークで接続されたコンピュータ上にオブジェクトを作成できます。コンピュータ名は、共有名のコンピュータ名の部分と同じです。つまり、共有名が "\\MyServer\Public" の場合は、*ServerName* は "MyServer" になります。

メモ：

アプリケーションをリモート ネットワーク コンピュータでアクセスできるようにする方法の詳細については、COM に関するドキュメント (Microsoft Developer Network) を参照してください。アプリケーションのレジストリ キーの追加が必要な場合もあります。

`MyServer` というリモート コンピュータ上で実行されている Excel のインスタンスのバージョン番号を取得する例を次に示します。

```
Sub CreateRemoteExcelObj()
    Dim xlApp As Object
    ' Replace string "\\MyServer" with name of the remote computer.
    xlApp = CreateObject("Excel.Application", "\\MyServer")
    MsgBox(xlApp.Version)
End Sub
```

リモート サーバー名が正しくないか、または利用できない場合は、実行時エラーが発生します。

メモ：

オブジェクトの現在のインスタンスがない場合は **CreateObject** 関数を使用します。オブジェクトのインスタンスが既に実行中の場合は、新規インスタンスが起動し、指定した型のオブジェクトが作成されます。現在のインスタンスを使用する、またはアプリケーションを起動してファイルを読み込ませるには、**GetObject** 関数を使用します。複数のインスタンスを作成できないオブジェクトの場合は、何度 **CreateObject** 関数を実行しても、そのオブジェクトのインスタンスは 1 つしか作成されません。

Framework オブジェクトを作成する

CreateObject 関数は、COM オブジェクトの作成のみに使用できます。.NET Framework オブジェクトを作成するまったく同様の方法はありませんが、System 名前空間の Activator には、ローカルまたはリモート オブジェクトを作成するメソッドが含まれています。具体的には、CreateInstance メソッドまたは CreateInstanceFrom メソッドが使用できます。

セキュリティに関するメモ：

CreateObject 関数にはアンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

使用例

次の例では、**CreateObject** 関数を使用して Microsoft Excel のワークシートを作成し、作成したワークシートをファイルに保存します。この例を利用するには、プログラムを実行するコンピュータに Excel がインストールされている必要があります。さらに、[プロジェクト] メニューの [参照の追加] ダイアログ ボックスの [COM] タブで、オブジェクトのタイプ ライブラリへの参照を追加する必要があります。タイプ ライブラリの名前は、コンピュータにインストールされている Excel のバージョンによって異なります。たとえば、Microsoft Excel 2002 のタイプ ライブラリ名は Microsoft Excel 10.0 Object Library です。

VB

```
Sub TestExcel()
    Dim xlApp As Microsoft.Office.Interop.Excel.Application
    Dim xlBook As Microsoft.Office.Interop.Excel.Workbook
    Dim xlSheet As Microsoft.Office.Interop.Excel.Worksheet

    xlApp = CType(CreateObject("Excel.Application"), _
                Microsoft.Office.Interop.Excel.Application)
    xlBook = CType(xlApp.Workbooks.Add, _
                Microsoft.Office.Interop.Excel.Workbook)
```

```
    xlSheet = CType(xlBook.Worksheets(1), _
                Microsoft.Office.Interop.Excel.Worksheet)

    ' The following statement puts text in the second row of the sheet.
    xlSheet.Cells(2, 2) = "This is column B row 2"
    ' The following statement shows the sheet.
    xlSheet.Application.Visible = True
    ' The following statement saves the sheet to the C:\Test.xls directory.
    xlSheet.SaveAs("C:\Test.xls")
    ' Optionally, you can call xlApp.Quit to close the workbook.
End Sub
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : Interaction

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

GetObject 関数 (Visual Basic)
Dim ステートメント (Visual Basic)
Declare ステートメント
Exception
FileNotFoundException
Activator
CreateInstance
CreateInstanceFrom

その他の技術情報

.NET Framework アプリケーションにおける COM 相互運用性
アンマネージ コードとの相互運用

CType 関数

任意の式を、指定されたデータ型、オブジェクト、構造体、クラス、またはインターフェイスに明示的に変換し、その結果を返します。

```
CType(expression, typename)
```

指定項目

expression

任意の有効な式。*expression* の値が *typename* で許可されている範囲内でない場合、Visual Basic が例外をスローします。

typename

Dim ステートメントの **As** 句で有効な任意の式。つまり、任意のデータ型、オブジェクト、構造体、クラス、またはインターフェイスの名前を指定します。

解説

CType は、インラインでコンパイルされます。つまり、変換コードは、式を評価するコードに含まれます。変換を完了するためにプロシージャへの呼び出しを行わないようにすると、実行速度がが向上することがあります。

Integer から **Date** など、*expression* から *typename* への変換が定義されていない場合、Visual Basic はコンパイル時のエラー メッセージを表示します。

実行時に変換が失敗すると、適切な例外がスローされます。縮小変換が失敗した場合、最もよくスローされるのは OverflowException です。変換が定義されていない場合、InvalidCastException が発生します。これは、たとえば *expression* が **Object** 型で、実行時の型が *typename* への変換を持たない場合に起こります。

expression または *typename* のデータ型が、定義したクラスまたは構造体の場合、そのクラスまたは構造体に **CType** を変換演算子として定義できます。これにより、**CType** はオーバーロードされた演算子として機能します。この方法を利用する場合、定義したクラスまたは構造体からの変換、またはこのクラスまたは構造体への変換の動作 (スローする例外など) を制御できます。

オーバーロード

CType 演算子も、コードの外部で定義されたクラスまたは構造体でオーバーロードできます。このようなクラスまたは構造体からの変換、またはこのクラスまたは構造体への変換を行う場合は、その **CType** 演算子の動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

CType 関数を使って、指定したデータ型に式を変換する例を次に示します。

VB

```
Dim testNumber As Long = 1000
' The following line of code sets testNewType to 1000.0.
Dim testNewType As Single = CType(testNumber, Single)
```

参照

処理手順

方法 : 変換演算子を定義する

関連項目

データ型変換関数

変換関数 (Visual Basic)

Operator ステートメント

OverflowException

InvalidCastException

CurDir 関数

現在のパスを表す文字列を返します。

My.Computer.FileSystem オブジェクト では、**CurDir** よりも、ファイルの I/O 操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.FileSystem.CurrentDirectory プロパティ」を参照してください。

```
Public Overloads Function CurDir([ ByVal Drive As Char ]) As String
```

パラメータ

Drive

省略できます。既存のドライブを表す **Char** 式を指定します。ドライブを指定しない場合、または引数 *Drive* が長さ 0 の文字列 ("") の場合、**CurDir** 関数は現在のドライブのパスを返します。

例外

例外の種類	エラー番号	条件
IOException	68	*Drive* が見つかりません。
ArgumentException	68	指定された *Drive* が無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

この関数は、現在のパスを表す文字列を返します。

使用例

CurDir 関数を使って、現在のパスを取得するコード例を次に示します。

VB

```
' Assume current path on C drive is "C:\WINDOWS\SYSTEM".
' Assume current path on D drive is "D:\EXCEL".
' Assume C is the current drive.
Dim MyPath As String
MyPath = CurDir()    ' Returns "C:\WINDOWS\SYSTEM".
MyPath = CurDir("C"c)    ' Returns "C:\WINDOWS\SYSTEM".
MyPath = CurDir("D"c)    ' Returns "D:\EXCEL".
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

ChDir 関数
ChDrive 関数
MkDir 関数
RmDir 関数
IOException Class
ArgumentException Class

関数 D ～ G

次の表は、Visual Basic のランタイム メンバ関数の一覧です。

DateAdd	DateDiff	DatePart	DateSerial
DateValue	Day	DDB	DeleteSetting
Derived Math	Dir	Environ	EOF
ErrorToString	FileAttr	FileClose	FileCopy
FileDateTime	FileGet	FileGetObject	FileLen
FileOpen	FilePut	FilePutObject	FileWidth
Filter	Fix	Format	FormatCurrency
FormatDateTime	FormatNumber	FormatPercent	FreeFile
FV	GetAllSettings	GetAttr	GetChar
GetException	GetObject	GetSetting	

バージョン 6.0 以前に Visual Basic のランタイム メンバ関数として利用できた数値演算関数 **Exp** は、.NET Framework クラス ライブラリの Math クラスのメソッド (Exp) に置き換えられました。詳細については、「数値演算関数 (Visual Basic)」を参照してください。

スマート デバイス開発者のためのメモ

DeleteSetting、**Dir**、**Environ**、**EOF**、**FileAttr**、**FileClose**、**FileCopy**、**FileDateTime**、**FileGet**、**FileGetObject**、**FileLen**、**FileOpen**、**FilePut**、**FilePutObject**、**FileWidth**、**FreeFile**、**GetAllSettings**、**GetAttr** および **GetSetting** は、スマート デバイス アプリケーションではサポートされません。

参照

関連項目

関数 A ～ C
関数 H ～ L
関数 M ～ R
関数 S ～ Z

その他の技術情報

Visual Basic リファレンス

DateAdd 関数 (Visual Basic)

指定された時間間隔を加算した日付と時刻を日付型 (**Date**) の値で返します。

```
Public Overloads Function DateAdd( _
   ByVal Interval As DateInterval, _
   ByVal Number As Double, _
   ByVal DateValue As DateTime _
) As DateTime
' -or-
Public Overloads Function DateAdd( _
   ByVal Interval As String, _
   ByVal Number As Double, _
   ByVal DateValue As Object _
) As DateTime
```

パラメータ

Interval

必ず指定します。加算する時間間隔を表す **DateInterval** 列挙値または文字列 (**String**) 式です。

Number

必ず指定します。倍精度浮動小数点数型 (**Double**)。加算する間隔数を表す浮動小数点数式です。*Number* には、将来の日時を取得する場合は正の数を指定し、過去の日時を取得する場合は負の数を指定します。*Number* の小数部分は無視されます。

DateValue

必ず指定します。日付型 (**Date**) 間隔を追加する日時を表す式です。*DateValue* 自体は、呼び出しの実行中に変更されません。

設定

Interval 引数の設定値は次のいずれかです。

列挙値	数字列	加算する時間間隔の単位
DateInterval.Day	d	日付 : 整数。小数点以下は切り捨てられます。
DateInterval.DayOfYear	y	日付 : 整数。小数点以下は切り捨てられます。
DateInterval.Hour	h	時 : 整数。小数点以下は切り捨てられます。
DateInterval.Minute	n	分 : 整数。小数点以下は切り捨てられます。
DateInterval.Month	m	日付 : 整数。小数点以下は切り捨てられます。
DateInterval.Quarter	q	四半期 : 整数値に切り捨てられます。
DateInterval.Second	s	秒 : 整数。小数点以下は切り捨てられます。
DateInterval.Weekday	w	日付 : 整数。小数点以下は切り捨てられます。
DateInterval.WeekOfYear	ww	週 : 整数値に切り捨てられます。
DateInterval.Year	yyyy	日付 : 整数。小数点以下は切り捨てられます。

例外

例外の種類	エラー番号	条件

InvalidCastException	13	*DateValue* を日付型 (**Date**) に強制変換できません。
ArgumentException	5	Interval が無効です。
ArgumentOutOfRangeException	9	計算された日付が西暦 1 年 1 月 1 日 00:00:00 より前であるか、西暦 9999 年 12 月 31 日 23:59:59 より後です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

DateAdd 関数を使用すると、ある日付に対して、指定した時間間隔を加えたり引いたりできます。たとえば、現在から 30 日後の日付や、現在から 45 分前の時刻などを計算できます。

DateValue に日を追加するには、**DateInterval.Day**、**DateInterval.DayOfYear**、または **DateInterval.Weekday** を使用します。**DayOfYear** と **Weekday** は間隔とは関係ないので、上記 3 つは同様に扱われます。

DateAdd 関数は、無効な日付を返しません。結果の日付部分は、必要に応じて、結果の年における月の最後の日付になるように調整されます。次の例では、1 月 31 日に 1 か月を加えています。

```
Dim NextMonth As Date = DateAdd(DateInterval.Month, 1, #1/31/1995#)
```

この例では、**DateAdd** は `#2/31/1995#` ではなく `#2/28/1995#` を返します。*DateValue* が `#31-Jan-1996#` の場合は、1996 年が閏年であるため `#29-Feb-1996#` を返します。

メモ :

DateAdd は、System.Globalization 名前空間にある CultureInfo クラスの CurrentCulture プロパティに設定された現在のカレンダーを使用します。**CurrentCulture** の既定値は、[コントロール パネル] の設定によって決まります。

すべての日付型 (**Date**) の値が DateTime 構造体を使用できるため、メソッドでは時間間隔を加算する際に追加のオプションを指定できます。たとえば、次に示すように、最も近いミリ秒に丸められる小数の日付を日付型 (**Date**) の変数に加算できます。

```
Dim NextTime As Date = Now        ' Current date and time.
NextTime = NextTime.AddDays(3.4)  ' Increment by 3 2/5 days.
```

使用例

次の例は、ユーザーに日付とそれに加算する月数の入力を求めた後、**DateAdd** 関数を使って加算後の日付を表示します。

VB

```
Dim Msg, Number, StartDate As String   'Declare variables.
Dim Months As Double
Dim SecondDate As Date
Dim IntervalType As DateInterval
IntervalType = DateInterval.Month   ' Specifies months as interval.
StartDate = InputBox("Enter a date")
SecondDate = CDate(StartDate)
Number = InputBox("Enter number of months to add")
Months = Val(Number)
Msg = "New date: " & DateAdd(IntervalType, Months, SecondDate)
MsgBox(Msg)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DateDiff 関数 (Visual Basic)
DatePart 関数 (Visual Basic)
Day 関数 (Visual Basic)
Format 関数
Now プロパティ
Weekday 関数 (Visual Basic)
Year 関数 (Visual Basic)
日付型 (Date) (Visual Basic)
System

DateDiff 関数 (Visual Basic)

2 つの日付型 (**Date**) 値の間の時間間隔数を指定する長整数型 (**Long**) の値を返します。

```
Public Overloads Function DateDiff( _
    ByVal Interval As [ DateInterval | String ], _
    ByVal Date1 As DateTime, _
    ByVal Date2 As DateTime, _
    Optional ByVal DayOfWeek As FirstDayOfWeek = FirstDayOfWeek.Sunday, _
    Optional ByVal  WeekOfYear As FirstWeekOfYear = FirstWeekOfYear.Jan1 _
) As Long
```

パラメータ

Interval

必ず指定します。*Date1* と *Date2* の間の差異の単位として使用する時間間隔を表す **DateInterval** 列挙値または文字列 (**String**) 式。

Date1

必ず指定します。日付型 (**Date**)。計算で使用する最初の日時の値を指定します。

Date2

必ず指定します。日付型 (**Date**)。計算で使用する 2 番目の日時の値を指定します。

DayOfWeek

省略可能です。**FirstDayOfWeek** 列挙値から値を選択し、週の最初の曜日を指定します。指定しない場合、**FirstDayOfWeek.Sunday** が使用されます。

WeekOfYear

省略可能です。年の第 1 週を指定する、**FirstWeekOfYear** 列挙型から選択した値。指定しない場合、**FirstWeekOfYear.Jan1** が使用されます。

設定

Interval 引数の設定値は次のいずれかです。

列挙値	文字列値	時間の差異の単位
DateInterval.Day	"d"	Day
DateInterval.DayOfYear	"y"	Day
DateInterval.Hour	"h"	Hour
DateInterval.Minute	"n"	Minute
DateInterval.Month	"m"	Month
DateInterval.Quarter	"q"	四半期
DateInterval.Second	"s"	Second
DateInterval.Weekday	"w"	Week
DateInterval.WeekOfYear	"ww"	カレンダー週
DateInterval.Year	"yyyy"	Year

DayOfWeek 引数の設定値は次のいずれかです。

列挙値	値	説明
FirstDayOfWeek.System	0	システムで設定されている週の最初の曜日
FirstDayOfWeek.Sunday	1	日曜日 (既定値)
FirstDayOfWeek.Monday	2	月曜日 (ISO 規格 8601、3.17 項に準拠)
FirstDayOfWeek.Tuesday	3	火曜日
FirstDayOfWeek.Wednesday	4	水曜日
FirstDayOfWeek.Thursday	5	木曜日
FirstDayOfWeek.Friday	6	金曜日
FirstDayOfWeek.Saturday	7	土曜日

WeekOfYear 引数の設定値は次のいずれかです。

列挙値	値	説明
FirstWeekOfYear.System	0	システム設定で指定された年の第 1 週
FirstWeekOfYear.Jan1	1	1 月 1 日を含む週 (既定値)
FirstWeekOfYear.FirstFourDays	2	新しい年の少なくとも 4 日間を含む週 (ISO 規格 8601、3.17 項に準拠)
FirstWeekOfYear.FirstFullWeek	3	新しい年の 1 週間を含む最初の週

例外

例外の種類	エラー番号	条件
ArgumentException	5	無効な *Interval*.
ArgumentException	5	該当する *Date1*、*Date2*、*DayofWeek* がありません。
InvalidCastException	13	*Date1* または *Date2* が無効な型です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

DateDiff 関数を使用すると、指定した時間間隔が 2 つの日時の値の間にいくつ存在するかを判断できます。たとえば、**DateDiff** を使って、2 つの日付の間の日数や、現在から年末までの週の数などを求めることができます。

動作

- **パラメータの扱い DateDiff** は *Date1* の値を *Date2* の値から減算して差を求めます。どちらの値も、呼び出し側プログラムでは変更されません。
- **戻り値** *Date1* と *Date2* は日付型 (**Date**) であるため、日付と時刻の値はシステム タイマの 100 ナノ秒目盛りに正確に保たれます。ただし、**DateDiff** は、時間間隔数を常に長整数型 (**Long**) の値として返します。

  Date1 が *Date2* より後の日付と時刻を表す場合、**DateDiff** は負の数値を返します。
- **日間隔DayOfYear** は有効な時間間隔ではないため、*Interval* が **DateInterval.DayOfYear** に設定されている場合、**DateInterval.Day** と同じように扱われます。

- **週間隔***Interval* が **DateInterval.WeekOfYear** に設定されていると、戻り値は *Date1* を含む週の最初の日から、*Date2* を含む週の最初の日までの週数を表します。**DateInterval.Weekday** からさまざまな結果がどのように生成されるかを次に示します。

  VB

```
' The following statements set datTim1 to a Thursday
' and datTim2 to the following Tuesday.
Dim datTim1 As Date = #1/4/2001#
Dim datTim2 As Date = #1/9/2001#
' Assume Sunday is specified as first day of the week.
Dim wD As Long = DateDiff(DateInterval.Weekday, datTim1, datTim2)
Dim wY As Long = DateDiff(DateInterval.WeekOfYear, datTim1, datTim2)
```

  前の例では、2 つの日付間の差異が 7 日より少ないため、**DateDiff** は `wD` に 0 を返しますが、それぞれのカレンダー週の最初の日の間には 7 日の差異があるため、`wY` には 1 を返します。

  注意

  Date1 の時刻部分が *Date2* の時刻部分より大きく、*Interval* が **DateInterval.WeekOfYear** に設定されている場合、**DateDiff** 関数は、正確な値よりも 1 つ少ない値を返します。

- **それ以上の間隔***Interval* が **DateInterval.Year** に設定されていると、戻り値は単純に *Date1* の年と *Date2* の年から求められます。同様に、**DateInterval.Month** の戻り値は、引数の年と月の部分から求められ、**DateInterval.Quarter** の戻り値は、2 つの日付を含む四半期から求められます。

  たとえば、12 月 31 日と翌年の 1 月 1 日を比較する場合、差は 1 日ですが、**DateDiff** は **DateInterval.Year**、**DateInterval.Quarter**、**DateInterval.Month** に対して 1 を返します。

- **その他の間隔**すべての日付型 (**Date**) の値は DateTime 構造体でサポートされているため、メソッドには時間間隔を判断するときの追加のオプションがあります。たとえば、**Subtract** メソッドを、オーバーロードされたいずれかの形式で使用できます。System.DateTime.Subtract(System.TimeSpan) は **Date** 変数からTimeSpan を減算し、別の **Date** 値を求め、System.DateTime.Subtract(System.DateTime) は **Date** 値を減算して **TimeSpan** を返します。次に示すように、プロセスにかかった時間をミリ秒単位で調べることができます。

  VB

```
Dim startTime As Date = Now
' Run the process that is to be timed.
Dim runLength As Global.System.TimeSpan = Now.Subtract(startTime)
Dim millisecs As Integer = runLength.Milliseconds
```

使用例

次の例は、**DateDiff** 関数を使って、指定された日付までの日数を取得します。

VB

```
Dim firstDate, msg As String
Dim secondDate As Date
firstDate = InputBox("Enter a date")
secondDate = CDate(firstDate)
msg = "Days from today: " & DateDiff(DateInterval.Day, Now, secondDate)
MsgBox(msg)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DateAdd 関数 (Visual Basic)
DatePart 関数 (Visual Basic)
Day 関数 (Visual Basic)
Format 関数
Now プロパティ
Weekday 関数 (Visual Basic)
Year 関数 (Visual Basic)
日付型 (Date) (Visual Basic)
DateTime
TimeSpan

DatePart 関数 (Visual Basic)

特定の日付型 (**Date**) の値の指定コンポーネントを含む整数型 (**Integer**) の値を返します。

```
Public Overloads Function DatePart( _
   ByVal Interval As DateInterval, _
   ByVal DateValue As DateTime, _
   Optional ByVal FirstDayOfWeekValue As FirstDayOfWeek = VbSunday, _
   Optional ByVal FirstWeekOfYearValue As FirstWeekOfYear = VbFirstJan1 _
) As Integer
' -or-
Public Overloads Function DatePart( _
    ByVal Interval As String, _
   ByVal DateValue As Object, _
   Optional ByVal DayOfWeek As FirstDayOfWeek = FirstDayOfWeek.Sunday, _
   Optional ByVal WeekOfYear As FirstWeekOfYear = FirstWeekOfYear.Jan1 _
) As Integer
```

パラメータ

Interval

必ず指定します。戻り値とする日時の値の項目を表す **DateInterval** 列挙値または文字列 (**String**) 式です。

DateValue

必ず指定します。評価する日付型 (**Date**) の値。

FirstDayOfWeekValue, *DayOfWeek*

省略可能です。週の最初の曜日を指定する、**FirstDayOfWeek** 列挙値から選択した値です。省略すると、**FirstDayOfWeek.Sunday** が使用されます。

FirstWeekOfYearValue, *WeekOfYear*

省略可能です。年の第 1 週を指定する、**FirstWeekOfYear** 列挙型から選択した値です。省略すると、**FirstWeekOfYear.Jan1** が使用されます。

設定

Interval 引数の設定値は次のいずれかです。

列挙値	文字列	戻り値とする日時の値の項目
DateInterval.Day	d	日付 (1 ～ 31)
DateInterval.DayOfYear	y	年間通算日 (1 ～ 366)
DateInterval.Hour	h	時間
DateInterval.Minute	n	分
DateInterval.Month	m	月
DateInterval.Quarter	q	四半期
DateInterval.Second	s	秒
DateInterval.Weekday	w	曜日 (1 ～ 7)
DateInterval.WeekOfYear	ww	週 (1 ～ 53)

DateInterval.Year	yyyy	年

FirstDayOfWeekValue 引数の設定値は次のいずれかです。

列挙値	値	説明
FirstDayOfWeek.System	0	システムで設定されている週の最初の曜日
FirstDayOfWeek.Sunday	1	日曜日 (既定値)
FirstDayOfWeek.Monday	2	月曜日 (ISO 規格 8601、3.17 項に準拠)
FirstDayOfWeek.Tuesday	3	火曜日
FirstDayOfWeek.Wednesday	4	水曜日
FirstDayOfWeek.Thursday	5	木曜日
FirstDayOfWeek.Friday	6	金曜日
FirstDayOfWeek.Saturday	7	土曜日

FirstWeekOfYearValue 引数の設定値は次のいずれかです。

列挙値	値	説明
FirstWeekOfYear.System	0	システム設定で指定された年の第 1 週
FirstWeekOfYear.Jan1	1	1 月 1 日を含む週 (既定値)
FirstWeekOfYear.FirstFourDays	2	新しい年の少なくとも 4 日間を含む週 (ISO 規格 8601、3.17 項に準拠)
FirstWeekOfYear.FirstFullWeek	3	新しい年の 1 週間を含む最初の週

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Interval* の型が無効です。
InvalidCastException	13	*DateValue* を日付型 (**Date**) に強制変換できません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

DatePart 関数を使用すると、日時の値を評価し、特定のコンポーネントを返すことができます。たとえば、**DatePart** を使用して、曜日や現在の時刻などを計算できます。

Interval 引数に **DateInterval.Weekday** を選択した場合、戻り値は **FirstDayOfWeek** 列挙型の値で構成されます。**DateInterval.WeekOfYear** を選択すると、**DatePart** は System.Globalization 名前空間の Calendar クラスと CultureInfo クラスを使用して、現在の設定を判断します。

FirstDayOfWeekValue 引数は、**DateInterval.Weekday**、**DateInterval.WeekOfYear**、および *Interval* の設定を使用する計算に影響します。*FirstWeekOfYearValue* 引数は、*Interval* に **DateInterval.WeekOfYear** を指定した計算に影響します。

すべての日付型 (**Date**) の値は DateTime 構造体でサポートされているため、日付型のメソッドでは日時部分を取得するときに追加のオプションを使用できます。たとえば、次に示すように、時刻値が午前 0 時に設定された日付型 (**Date**) の変数の日付値全体を取得できます。

```
Dim CurrDatTim As Date = Now   ' Current date and time.
```

```
Dim LastMidnight As Date = CurrDatTim.Date    ' At midnight.
```

使用例

次の例は、ユーザーに日付の入力を求めた後、**DatePart** 関数を使って、その日付がどの四半期に当たるかを調べます。

VB

```
Dim FirstDate, Msg As String   'Declare variables.
Dim SecondDate As Date
FirstDate = InputBox("Enter a date:")
SecondDate = CDate(FirstDate)
Msg = "Quarter: " & DatePart(DateInterval.Quarter, SecondDate)
MsgBox(Msg)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DateAdd 関数 (Visual Basic)
DateDiff 関数 (Visual Basic)
Day 関数 (Visual Basic)
Format 関数
Now プロパティ
Weekday 関数 (Visual Basic)
Year 関数 (Visual Basic)
日付型 (Date) (Visual Basic)
System

DateSerial 関数 (Visual Basic)

時刻情報が午前 0 時 (00:00:00) に設定された、指定の年、月、および日を表す日付型 (**Date**) 型の値を返します。

```
Public Function DateSerial( _
   ByVal [Year] As Integer, _
   ByVal [Month] As Integer, _
   ByVal [Day] As Integer _
) As DateTime
```

パラメータ

Year

必ず指定します。1 ～ 9999 の整数型 (**Integer**) の式です。ただし、この範囲より小さい値も指定できます。*Year* が 0 ～ 99 の場合は、後の「解説」で説明するように 1930 ～ 2029 として解釈されます。*Year* が 1 より小さい場合は、現在の年から引かれます。

Month

必ず指定します。1 ～ 12 の整数型 (**Integer**) の式です。ただし、この範囲外の値も指定できます。*Month* の値から 1 が引かれ、計算された年の 1 月に加えられます。つまり、(*Month* - 1) が 1 月に加算されます。年は必要に応じて再計算されます。次の結果はこの効果を示しています。

- *Month* が 1 の場合、結果は計算された年の 1 月です。
- *Month* が 0 の場合、結果は前年の 12 月です。
- *Month* が -1 の場合、結果は前年の 11 月です。
- *Month* が 13 の場合、結果は翌年の 1 月です。

Day

必ず指定します。1 ～ 31 の整数型 (**Integer**) の式です。ただし、この範囲外の値も指定できます。*Day* の値から 1 が引かれ、計算された月の第 1 日に加えられます。つまり、(*Day* - 1) が月の 1 日に加算されます。月と年は、必要に応じて再計算されます。次の結果はこの効果を示しています。

- *Day* が 1 の場合、結果は計算された月の 1 日です。
- *Day* が 0 の場合、結果は前月の末日です。
- *Day* が -1 の場合、結果は前月の最後から 2 日目です。
- *Day* が当月の末日を過ぎている場合、結果は翌月の該当日です。たとえば、*Month* が 4 で *Day* が 31 の場合、結果は 5 月 1 日です。

解説

Windows 98 または Windows 2000 では、引数 *Year* の 2 桁の年は、ユーザー定義のコンピュータ設定に基づいて解釈されます。既定の設定では 0 ～ 29 の値は 2000 ～ 2029 年、30 ～ 99 の値は 1930 ～ 1999 年と解釈されます。これ以外の *Year* 引数では、4 桁の数値 (たとえば 1924) を使用してください。

以前のバージョンの Windows では、2 桁の年はこの既定に基づいて解釈されます。関数が必ず正しい値を返すようにするには、4 桁の *Year* を使用してください。

負の値、0、および正の値の引数を次に示します。ここでは、**DateSerial** 関数は、本年より 10 年前の年の 3 月 1 日の前日 (10 年前の 2 月末日) を表す日付型 (**Date**) の値を返します。

```
Dim EndFeb As Date = DateSerial(-10, 3, 0)
```

Month または *Day* が通常の範囲を超えた場合は、その上の単位に繰り上げられます。たとえば、32 日を指定すると、*Month* の値に応じて、1 か月と 1 ～ 4 日として評価されます。*Year* が 9999 より大きい場合、または引数のいずれかが -2,147,483,648 ～ 2,147,483,647 の範囲外の場合は、ArgumentException エラーが発生します。3 つの引数で指定された日付が 1 年 1 月 1 日 00:00:00 よりも前の場合、または 9999 年 12 月 31 日 23:59:59 よりも後の場合は、ArgumentOutOfRangeException エラーが発生します。

日付型 (**Date**) には時刻コンポーネントが含まれます。**DateSerial** は、これらすべてのコンポーネントを 0 に設定するため、返される値は計算

された日の始まりを表します。

すべての日付型 (**Date**) の値は DateTime 構造体でサポートされているため、メソッドには日付型 (**Date**) の値をアセンブルするときの追加のオプションがあります。たとえば、オーバーロードされた DateTime コンストラクタの 1 つを使用し、コンポーネントを任意に組み合わせて日付型 (**Date**) の変数を設定できます。`NewDateTime` に 1978 年 5 月 6 日午前 8:30 の 0.1 秒前を設定する例を次に示します。

```
Dim NewDateTime As Date = New Date(1978, 5, 6, 8, 29, 59, 900)
```

使用例

次の例は、**DateSerial** 関数を使って、指定された年、月、日に対応する日付を取得します。

VB

```
Dim MyDate As Date
' MyDate contains the date for February 12, 1969.
MyDate = DateSerial(1969, 2, 12)   ' Return a date.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DateValue 関数 (Visual Basic)
Day 関数 (Visual Basic)
Month 関数 (Visual Basic)
Now プロパティ
TimeSerial 関数 (Visual Basic)
TimeValue 関数 (Visual Basic)
Weekday 関数 (Visual Basic)
Year 関数 (Visual Basic)
日付型 (Date) (Visual Basic)
System

DateValue 関数 (Visual Basic)

時刻情報が午前 0 時 (00:00:00) に設定された、文字列で表される日付情報を含む日付型 (**Date**) の値を返します。

```
Public Function DateValue(ByVal StringDate As String) As DateTime
```

パラメータ

StringDate

必ず指定します。1 年 1 月 1 日 00:00:00 から 9999 年 12 月 31 日 23:59:59 までの日時の値を表す文字列 (**String**) 式。

例外

例外の種類	エラー番号	条件
InvalidCastException	13	*StringDate* に無効な時刻情報が含まれています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

StringDate に、有効な日付の区切り記号で区切られた 1 から 12 までの数字だけが含まれている場合、**DateValue** は、システムで設定された短い日付書式に従って月、日、年の順を認識します。**DateValue** は、System.Globalization 名前空間内の CultureInfo クラスの CurrentCulture プロパティの現在のカレンダーの設定を使用します。**CurrentCulture** の既定値は、コントロール パネルの設定によって決まります。短い日付書式は、**System.Globalization** 名前空間内の DateTimeFormatInfo クラスの ShortDatePattern プロパティを設定することでオーバーライドできます。

DateValue は、長い書式、省略形、および数値書式の月名を認識します。たとえば、**DateValue** は "1991/12/30" や "91/12/30" だけでなく、"平成 3 年 12 月 30 日" や "H3 - 12 - 30" などの形式も認識します。

引数 *StringDate* の年の部分を省略すると、**DateValue** 関数はシステムの日付を現在の年として使用します。

StringDate 引数に時刻情報が含まれる場合、**DateValue** は返される値に時刻情報を含めません。ただし、引数 *StringDate* に不正な時刻情報 (たとえば "89:98") を指定した場合は、**InvalidCastException** エラーが発生します。

使用例

次の例は、**DateValue** 関数を使って、文字列を日付に変換します。日付リテラルを使って、オブジェクト (**Object**) や日付型 (**Date**) の変数に日付を直接代入することもできます (たとえば `oldDate = #2/12/69#`)。

VB

```
Dim oldDate As Date
oldDate = DateValue("February 12, 1969")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DateSerial 関数 (Visual Basic)
Day 関数 (Visual Basic)
Month 関数 (Visual Basic)
Now プロパティ
TimeSerial 関数 (Visual Basic)
TimeValue 関数 (Visual Basic)
Weekday 関数 (Visual Basic)
Year 関数 (Visual Basic)

日付型 (Date) (Visual Basic)
DateTime

Day 関数 (Visual Basic)

日付を表す 1 ～ 31 の整数型 (**Integer**) の値を返します。

```
Public Function Day(ByVal DateValue As DateTime) As Integer
```

パラメータ
DateValue

必ず指定します。日付を抽出する日付型 (**Date**) の値です。

解説

Day 関数を使用する際、関数を **Microsoft.VisualBasic** 名前空間で修飾することが必要になる場合があります。この理由は、System.Windows.Forms 名前空間では、Day が列挙体として定義されているからです。`Day` で修飾することにより、このあいまいさがどのように解決されるかを次に示します。

```
Dim thisDay As Integer = Microsoft.VisualBasic.DateAndTime.Day(Now)
```

DatePart を呼び出し、*Interval* 引数に `DateInterval.Day` を指定することによっても日付を取得できます。

使用例

次の例は、**Day** 関数を使って、指定された日付の日にちを取得します。開発環境では、日付リテラルは、コード記述時のロケールで設定されている標準の短い形式 ("02/12/1969" など) で表示されます。

VB

```
Dim oldDate As Date
Dim oldDay As Integer
' Assign a date using standard short format.
oldDate = #2/12/1969#
oldDay = Microsoft.VisualBasic.DateAndTime.Day(oldDate)
' oldDay now contains 12.
```

System.Windows.Forms.Day の列挙体と区別するために、`Day` が修飾されています。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
DatePart 関数 (Visual Basic)
Month 関数 (Visual Basic)
Now プロパティ
Weekday 関数 (Visual Basic)
Year 関数 (Visual Basic)
DateTime
ArgumentException
ArgumentOutOfRangeException

DDB 関数

倍精度浮動小数点数型 (**Double**) の値を返します。倍率法などの指定した方法を使って特定の期における資産の減価償却費を返します。

```
Function DDB( _
   ByVal Cost As Double, _
   ByVal Salvage As Double, _
   ByVal Life As Double, _
   ByVal Period As Double, _
   Optional ByVal Factor As Double = 2.0 _
) As Double
```

パラメータ

Cost

必ず指定します。資産を購入した時点での価格を示す倍精度浮動小数点数型 (**Double**) の値を指定します。

Salvage

必ず指定します。耐用年数が終了した時点での資産の価格を示す倍精度浮動小数点数型 (**Double**) の値を指定します。

Life

必ず指定します。資産の耐用年数を示す倍精度浮動小数点数型 (**Double**) の値を指定します。

Period

必ず指定します。減価償却費を計算する期を示す倍精度浮動小数点数型 (**Double**) の値を指定します。

Factor

省略可能です。減価償却率を示す倍精度浮動小数点数型 (**Double**) の値を指定します。引数 factor を省略すると、2 が指定されたものと見なされ、倍率逓減法で計算されます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Factor* <= 0、*Salvage* < 0、*Period* <= 0、または *Period* > *Life*.

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

倍率法では、指定した償却率で減価償却費が計算されます。減価償却費は最初の期が最も高く、その後の期では減少していきます。

引数 *Life* および *Period* は、同じ単位で指定する必要があります。たとえば、*Life* が月単位で指定されている場合は、*Period* も月単位で指定する必要があります。すべての引数には必ず正の値を指定してください。

DDB 関数を使って指定した期に対する減価償却費を求めるには、次の式を使います。

*Depreciation / Period = ((Cost – Salvage) * Factor) / Life*

使用例

次の例は、**DDB** 関数を使って、指定した期の資産の減価償却費を返します。資産購入時点の価格 (`InitCost`)、資産の耐用年数を経た後での残存価額 (`SalvageVal`)、資産の耐用年数 (`LifeTime`)、減価償却費を計算する年 (`Depr`) を指定します。

VB

```
Dim InitCost, SalvageVal, LifeTime, DepYear As Double
Dim Fmt As String = "###,##0.00"

InitCost = CDbl(InputBox("What's the initial cost of the asset?"))
SalvageVal = CDbl(InputBox("Enter the asset's value at end of its life."))
```

```
LifeTime = CDbl(InputBox("What's the asset's useful life in years?"))

' Use the SLN function to calculate the deprecation per year.
Dim SlnDepr As Double = SLN(InitCost, SalvageVal, LifeTime)
Dim msg As String = "The depreciation per year: " & Format(SlnDepr, Fmt)
msg &= vbCrLf & "Year" & vbTab & "Linear" & vbTab & "Doubling" & vbCrLf

' Use the SYD and DDB functions to calculate the deprecation for each year.
For DepYear = 1 To LifeTime
    msg &= DepYear & vbTab & _
        Format(SYD(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbTab & _
        Format(DDB(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbCrLf
Next
MsgBox(msg)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

SLN 関数
SYD 関数
財務処理の概要
ArgumentException

DeleteSetting 関数

Windows レジストリのアプリケーションの初期設定ファイルから、セクションまたはキー設定を削除します。

My 機能では、**DeleteSetting** 関数よりも、レジストリ操作の生産性とパフォーマンスが高くなっています。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

```
Public Sub DeleteSetting( _
   ByVal AppName As String, _
   Optional ByVal Section As String = Nothing, _
   Optional ByVal Key As String = Nothing _
)
```

パラメータ

AppName

必ず指定します。セクションまたはキー設定を適用するアプリケーション名またはプロジェクト名を含む文字列型 (**String**) の式を指定します。

Section

必ず指定します。キー設定を削除するセクション名を含む文字列型 (**String**) の式を指定します。*AppName* および *Section* だけを指定した場合、指定されたセクションは関連付けられたすべてのキー設定と共に削除されます。

Key

省略できます。削除するキー設定名を含む文字列型 (**String**) の式を指定します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Section*、*AppName*、または *Key* の設定値が存在していません。
ArgumentException	5	ユーザーがログインしていません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

引数をすべて指定した場合、指定したキー設定が削除されます。ただし、指定したセクションまたはキー設定が存在しない場合に **DeleteSetting** を使用すると、ランタイム エラーが発生します。

DeleteSetting は、ユーザーが対話形式でログオンするまでアクティブにならない **HKEY_LOCAL_USER** レジストリ キーのもとで動作するため、ユーザーがログオンしていることが必要条件です。

双方向でないプロセス (Mtx.exe など) からアクセスするレジストリの設定は、**HKEY_LOCAL_MACHINE\Software** または **HKEY_USER\DEFAULT\Software** レジストリ キーのいずれかに格納されます。

使用例

次の例は、最初に **SaveSetting** プロシージャを使用して、Windows のレジストリに `MyApp` アプリケーションのエントリを作成します。次に、**DeleteSetting** 関数を使用して、作成したエントリを削除します。引数 *Key* を指定していないため、セクション名とセクションのすべてのキーを含む `Startup` セクション全体が削除されます。

VB

```
' Place some settings in the registry.
SaveSetting("MyApp", "Startup", "Top", "75")
SaveSetting("MyApp","Startup", "Left", "50")
' Remove section and all its settings from registry.
DeleteSetting ("MyApp", "Startup")
' Remove MyApp from the registry.
DeleteSetting ("MyApp")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

GetAllSettings 関数
GetSetting 関数
SaveSetting 関数
ArgumentException Class

Dir 関数

指定したパターン、ファイル属性、またはドライブのボリューム ラベルに一致する、ファイル、ディレクトリ、フォルダの名前を表す文字列を返します。

My.Computer.FileSystem オブジェクト を使用すると、**Dir** 関数を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem.GetDirectoryInfo メソッド」を参照してください。

```
Public Overloads Function Dir() As String
' -or-
Public Overloads Function Dir( _
       ByVal PathName As String, _
       Optional ByVal Attributes As FileAttribute = FileAttribute.Normal _
) As String
```

パラメータ

PathName

省略可能です。**String** 式にはファイル名、フォルダ名やディレクトリ名、またはドライブのボリューム ラベルを指定します。引数 *PathName* が見つからない場合は、長さ 0 の文字列 ("") が返されます。

Attributes

省略可能です。ファイル属性を指定する列挙型または数式を指定します。省略すると、**Dir** は引数 *PathName* に一致し、属性を持たないファイルを返します。

設定

引数 *Attributes* の列挙型値は次のようになります。

値	定数	説明
Normal	**vbnormal**	既定値です。属性のないファイルを指定します。
ReadOnly	**vbReadOnly**	属性のないファイルと読み取り専用ファイルを指定します。
Hidden	**vbHidden**	属性のないファイルと隠しファイルを指定します。
System	**vbSystem**	属性のないファイルとシステム ファイルを指定します。
Volume	**vbVolume**	ボリューム ラベルを指定します。他の属性を指定すると、**vbVolume** は無視されます。
Directory	**vbDirectory**	属性のないファイルとディレクトリまたはフォルダを指定します。
Archive	**vbArchive**	前回のバックアップ以降に変更されているファイル。
Alias	**vbAlias**	他の名前が付いているファイル。

メモ：

ここで示した列挙型は Visual Basic 言語によって指定され、実際の値の代わりにコード内の任意の場所で使用できます。

解説

Dir 関数は、複数文字 (*) および単一文字 (?) のワイルドカードに対応しており、複数のファイルを一度に指定できます。

VbVolume はファイル名ではなく、ドライブのボリューム ラベルを返します。

Dir 関数を初めて呼び出すときは、*PathName* を指定する必要があります。次の項目を取得する場合は、**Dir** 関数をパラメータを指定せずに続けて呼び出します。

セキュリティに関するメモ：

Dir 関数を正しく実行するには、FileIOPermission の Read フラグと PathDiscovery フラグに、コードの実行を許可するよう設定されていることが必要です。詳細については、「**FileIOPermission**」、「SecurityException」、および「コード アクセス許可」を参照してください。

使用例

Dir 関数を使って、特定のファイルおよびディレクトリが存在するかどうかを調べるコード例を次に示します。

VB

```
Dim MyFile, MyPath, MyName As String
' Returns "WIN.INI" if it exists.
MyFile = Dir("C:\WINDOWS\WIN.INI")

' Returns filename with specified extension. If more than one *.INI
' file exists, the first file found is returned.
MyFile = Dir("C:\WINDOWS\*.INI")

' Call Dir again without arguments to return the next *.INI file in the
' same directory.
MyFile = Dir()

' Return first *.TXT file, including files with a set hidden attribute.
MyFile = Dir("*.TXT", vbHidden)

' Display the names in C:\ that represent directories.
MyPath = "c:\"   ' Set the path.
MyName = Dir(MyPath, vbDirectory)   ' Retrieve the first entry.
Do While MyName <> ""   ' Start the loop.
      ' Use bitwise comparison to make sure MyName is a directory.
      If (GetAttr(MyPath & MyName) And vbDirectory) = vbDirectory Then
         ' Display entry only if it's a directory.
         MsgBox(MyName)
      End If
   MyName = Dir()   ' Get next entry.
Loop
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

ChDir 関数
CurDir 関数
FileAttribute 列挙型

Environ 関数

オペレーティング システムの環境変数に関連付けられた文字列を返します。

```
Overloads Function Environ(ByVal Expression As Integer) As String
' -or-
Overloads Function Environ(ByVal Expression As String) As String
```

パラメータ

Expression

必ず指定します。環境変数名を含む文字列か、または環境ストリング テーブルでの環境文字列の番号に対応する整数のどちらかを指す式を指定します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Expression* が指定されていません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

引数 *Expression* に文字列が含まれる場合、Environ 関数は、指定された環境文字列に設定されたテキストを返します。これは、該当する環境変数の環境ストリング テーブルで、等号 (=) の後に続くテキストです。引数 *Expression* 内の文字列が環境ストリング テーブルに見つからない場合は、長さ 0 の文字列 ("") が返されます。

引数 *Expression* に整数が含まれる場合は、環境ストリング テーブルでその数値が示す位置にある文字列が返されます。この場合、**Environ** 関数は、環境変数名を含むテキスト全体を返します。指定された位置に環境文字列がない場合、**Environ** 関数は長さ 0 の文字列を返します。

セキュリティに関するメモ：

Environ 関数には環境変数へのアクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

使用例

Environ 関数を使って、環境ストリング テーブルから `PATH` ステートメントのエントリ番号と長さを取得するコード例を次に示します。

VB

```
Sub tenv()
   Dim envString As String
   Dim found As Boolean = False
   Dim index As Integer = 1
   Dim pathLength As Integer
   Dim message As String

   envString = Environ(index)
   While Not found And (envString <> "")
      If (envString.Substring(0, 5) = "Path=") Then
         found = True
      Else
         index += 1
         envString = Environ(index)
      End If
   End While

   If found Then
      pathLength = Environ("PATH").Length
      message = "PATH entry = " & index & " and length = " & pathLength
```

```
    Else
       message = "No PATH environment variable exists."
    End If

    MsgBox(message)
End Sub
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
Visual Basic ランタイム ライブラリのメンバ
ArgumentException

EOF 関数

Random またはシーケンシャル **Input** で開いたファイルの現在位置がファイルの末尾に達している場合、ブール値の **True** を返します。

```
Public Function EOF(ByVal FileNumber As Integer) As Boolean
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号を表す整数型 (**Integer**) の値。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

EOF 関数は、ファイルの末尾に達する直前の入力を取得しようとするときにエラーが発生するのを防ぐために使用します。

EOF 関数はファイルの末尾に達していない場合は、**False** を返します。**Random** または **Binary** モードでファイルを開いた場合、**EOF** 関数は最後に実行された **FileGet** 関数でレコード全体が読み込みができなくなるまで **False** を返します。

Binary モードでファイルを開いた場合、**Input** 関数を使用して **EOF** 関数が **True** を返すまでファイルを読み込もうとすると、エラーが発生します。**Input** 関数を使用してバイナリ ファイルを読み込む場合は、**EOF** 関数の代わりに **LOF** 関数および **Loc** 関数を使用します。**EOF** 関数を使用する場合は、**Get** ステートメントを使用します。**Output** モードで開いたファイルの場合は、**EOF** は常に **True** を返します。

使用例

EOF 関数を使って、ファイルの末尾に達したかどうかを調べる例を次に示します。この例では、ファイル `Testfile` は、複数行のテキストを含むテキスト ファイルと仮定します。

VB

```
Dim TextLine As String
' Open file.
FileOpen(1, "TESTFILE", OpenMode.Input)
' Loop until end of file.
Do While Not EOF(1)
' Read line into variable.
   TextLine = LineInput(1)
' Display result in a message box.
   MsgBox("End of file reached at " & TextLine)
Loop
FileClose(1)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileGet 関数

Loc 関数

LOF 関数

FileOpen 関数

IOException

ErrorToString 関数

指定したエラー番号に対応するエラー メッセージを返します。

```
Public Shared Function ErrorToString(ByVal ErrorNumber As Integer) As String
```

パラメータ

ErrorNumber

(省略可能。) 任意の有効なエラー番号。

例外

例外の種類	エラー番号	条件
ArgumentException	5	該当する *ErrorNumber* がありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

ErrorToString 関数は、**Err** オブジェクトのプロパティの設定値を調べ、直前に発生したランタイム エラーを特定します。**ErrorToString** 関数の戻り値は、**Err** オブジェクトの **Description** プロパティと同じです。*ErrorNumber* が有効なエラー番号であり、しかし定義されていないという場合、**ErrorToString** は 文字列 `"Application-defined or object-defined error."` を返します。*ErrorNumber* が有効でない場合は、エラーが発生します。*ErrorNumber* を省略すると、直前に発生したランタイム エラーに対応するメッセージが返されます。ランタイム エラーがまったく発生していない場合や、*ErrorNumber* が 0 の場合、**ErrorToString** は長さ 0 の文字列 ("") を返します。

バージョン 6.0 以前の Visual Basic では、**Error** 関数がこの機能を提供していました。

使用例

次のコードは、**ErrorToString** 関数を使って、指定したエラー番号に対応するエラー メッセージを表示します。

VB

```
Dim ErrorNumber As Integer
For ErrorNumber = 61 To 64   ' Loop through values 61 - 64.
   MsgBox(ErrorToString(ErrorNumber))   ' Display error names in message box.
Next ErrorNumber
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Conversion**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Err オブジェクト (Visual Basic)
Description プロパティ (Err オブジェクト)
ArgumentException

概念

Visual Basic の構造化例外処理の概要
非構造化例外処理の概要

FileAttr 関数

FileOpen 関数で開いたファイルのファイル モードを表す列挙定数を返します。

My.Computer.FileSystem オブジェクト により、**FileAttr** 機能よりも、ファイルの I/O 操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.FileSystem.GetFileInfo メソッド」を参照してください。

```
Public Function FileAttr(ByVal FileNumber As Integer) As OpenMode
```

パラメータ

FileNumber

必ず指定します。**Integer** です。有効なファイル番号です。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

戻り値

次の列挙型値は、ファイルのアクセス モードを示します。

値	モード
1	**OpenMode.Input**
2	**OpenMode.Output**
4	**OpenMode.Random**
8	**OpenMode.Append**
32	**OpenMode.Binary**

解説

FileOpen 関数で開いたファイルのファイル モードを表す列挙定数を返します。

使用例

FileAttr 関数を使って、開いているファイルのファイル モードを返す例を次に示します。

VB

```
Dim mode As OpenMode
FileOpen(1, "c:\TESTFILE.TXT", OpenMode.Input)
mode = FileAttr(1)
MsgBox("The file mode is " & mode.ToString())
FileClose(1)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

GetAttr 関数
FileOpen 関数
SetAttr 関数
IOException Class

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

FileClose 関数

FileOpen 関数で開いたファイルへの入出力を終了して、ファイルを閉じます。

My を使用すると、ファイルの I/O 操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Sub FileClose(ParamArray FileNumbers() As Integer)
```

パラメータ
FileNumbers

省略可能です。閉じる対象となる、0 以上のチャネルで構成されるパラメータ配列。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在しません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

FileClose 関数は、下位互換性のために用意されており、パフォーマンスに悪影響を与える可能性があります。非レガシ アプリケーションに対しては、**My.Computer.FileSystem** オブジェクトはより優れたパフォーマンスを発揮します。詳細については、「Visual Basic におけるファイル アクセス」を参照してください。

引数 *FileNumbers* を省略すると、**FileOpen** 関数で開いたすべてのファイルが閉じられます。

Output モードまたは **Append** モードで開いたファイルを閉じると、出力バッファに残っているデータは、そのファイルに対するオペレーティング システム バッファに書き込まれます。閉じたファイルが使用していたバッファ領域はすべて解放されます。

FileClose 関数でファイルを閉じると、そのファイルに割り当てられていたファイル番号は解放されます。

使用例

FileClose 関数を使って、**Input** モードで開いている 3 つのファイルをすべて閉じる例を次に示します。

VB

```
Dim TextLine As String
FileOpen(1, "TESTFILE", OpenMode.Input)   ' Open file.
Do While Not EOF(1)   ' Loop until end of file.
   TextLine = LineInput(1)   ' Read line into variable.
   MsgBox(TextLine)   ' Display the line
Loop
FileClose(1)   ' Close file.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
End ステートメント

FileOpen 関数
Reset 関数
Stop ステートメント (Visual Basic)

その他の技術情報

Visual Basic におけるファイル アクセス

FileCopy 関数

ファイルをコピーします。

My.Computer.FileSystem オブジェクト を使用すると、**FileCopy** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem.CopyFile メソッド」を参照してください。

```
Public Sub FileCopy( _
   ByVal Source As String, _
   ByVal Destination As String _
)
```

パラメータ

Source

必ず指定します。コピーするファイル名を指定する **String** 式です。*Source* には、コピー元ファイルのディレクトリ名やフォルダ名、およびドライブ名を指定できます。

Destination

必ず指定します。コピー先のファイル名を指定する **String** 式です。*Destination* にはコピー先ファイルのディレクトリ名やフォルダ名、およびドライブ名を指定できます。

例外

例外の種類	エラー番号	条件
ArgumentException	52	*Source* または *Destination* が無効であるか、指定されていません。
IOException	55	ファイルが既に開かれています。
FileNotFoundException	53	ファイルが存在していません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

現在開いているファイルに対して **FileCopy** 関数を使用しようとすると、エラーが発生します。

FileCopy をローカル ドライブで使用するには、完全な信頼が必要です。

使用例

FileCopy 関数を使って、ファイルを別のファイルにコピーするコード例を次に示します。この例では、`SrcFile` はデータを含むファイルであると仮定します。

VB

```
Dim SourceFile, DestinationFile As String
SourceFile = "SRCFILE"    ' Define source file name.
DestinationFile = "DESTFILE"    ' Define target file name.
FileCopy(SourceFile, DestinationFile)    ' Copy source to target.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でファイルのコピーを別のディレクトリに作成する
方法 : Visual Basic でファイルのコピーを同じディレクトリに作成する
方法 : Visual Basic でディレクトリを別のディレクトリにコピーする

関連項目

Kill 関数
IOException Class
FileNotFoundException Class
ArgumentException

FileDateTime 関数

ファイルの作成日時または最終変更日時を示す日付型 (**Date**) の値を返します。

My 機能を使用すると、**FileDateTime** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem.GetFileInfo メソッド」を参照してください。

```
Public Function FileDateTime(ByVal PathName As String) As DateTime
```

パラメータ

PathName

必ず指定します。ファイル名を指定する文字列 (**String**) 式です。*PathName* にはディレクトリ名またはフォルダ名、およびドライブ名も含めて指定できます。

例外

例外の種類	エラー番号	条件
ArgumentException	52	*PathName* が無効です。またはワイルドカードを含んでいます。
FileNotFoundException	53	対象のファイルが存在しません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

この関数は、ファイルの作成日時または最終変更日時を示す日付型 (**Date**) の値を返します。

使用例

FileDateTime 関数を使って、ファイルの作成日時や最終変更日時を求めるコード例を次に示します。表示される日時の形式は、システムのロケール設定に基づいています。

VB

```
Dim MyStamp As Date
' Assume TESTFILE was last modified on October 12, 2001 at 4:35:47 PM.
' Assume English/U.S. locale settings.
' Returns "10/12/2001 4:35:47 PM".
MyStamp = FileDateTime("C:\TESTFILE.txt")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileLen 関数
GetAttr 関数
ArgumentException Class
FileNotFoundException Class

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

FileGet 関数

ディスク ファイルからデータを読み込み、それを変数に格納します。

My 機能を使用すると、**FileGet** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Object, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Short, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Integer, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Single, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Double, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Decimal, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Byte, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Boolean, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As Date, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FileGet( _
   ByVal FileNumber As Integer, _
   ByRef Value As System.Array, _
```

```
    Optional RecordNumber As Integer = -1, _
    Optional ArrayIsDynamic As Boolean = False _
)
' -or-
Public Overloads Sub FileGet(
    ByVal FileNumber As Integer, _
    ByRef Value As String, _
    Optional RecordNumber As Integer = -1, _
    Optional StringIsFixedLength As Boolean = False _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

Value

必ず指定します。読み込んだデータを格納する変数名。

RecordNumber

(省略可能。) 読み込みを始めるレコード番号 (**Random** モードのファイル) またはバイト位置 (**Binary** モードのファイル)。

ArrayIsDynamic

(省略可能。) 配列を書き込む場合にだけ適用されます。配列を動的に処理するかどうか、および配列のサイズと境界を表す配列記述子が必要かどうかを指定します。

StringIsFixedLength

(省略可能。) 文字列を書き込む場合にだけ適用されます。文字列の長さを表す 2 バイトの記述子を書き込むかどうかを指定します。既定では **False** です。

例外

例外の種類	エラー番号	条件
ArgumentException	63	*RecordNumber* が -1 ではない 1 未満の値です。
IOException	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

FileGet は、**Random** モードおよび **Binary** モードの場合にだけ有効です。

通常、**FileGet** 関数を使用して読み込んだデータは **FilePut** 関数を使用して書き込みます。

ファイルの中で先頭のレコード番号またはバイト位置は 1 になり、2 番目のレコード番号またはバイト位置は 2 になります。引数 *RecordNumber* を省略すると、最後に呼び出した **FileGet** 関数または **FilePut** 関数、あるいは最後に呼び出した **Seek** 関数が示す位置の次のレコードまたはデータが読み込まれます。

セキュリティに関するメモ :
ファイルからデータを読み取るときは、ファイル名の拡張子に基づいてファイルの内容を判断しないでください。たとえば、Form1.vb という名前のファイルが Visual Basic のソース ファイルとは限りません。

Random モード

Random モードで開いたファイルに対しては、次の規則が適用されます。

- 読み込むデータの長さが **FileOpen** 関数ステートメントの *RecordLength* 句で指定したレコード長よりも短ければ、**FileGet** 関数はレコード長の終端から次のレコードを読み込むことができます。レコードの終わりと次のレコードの先頭の間には、ファイル バッファの内容が埋め込まれます。埋め込まれるデータの量は正確に指定できないため、通常はレコード長を読み込むデータ長に合わせます。

- 文字列変数を使う場合、**FileGet** 関数は、既定では文字列の長さを示す 2 バイトの記述子を読み込んでから、変数に格納するデータを読み込みます。したがって、**FileOpen** 関数の *RecordLength* 句では、文字列の実際の長さより 2 バイト以上大きい値を指定する必要があります。Visual Basic 6.0 以前のバージョンは固定長文字列をサポートしており、ファイルに書き込む場合、長さ記述子は書き込まれません。記述子なしで文字列を読み込む場合は、パラメータ *StringIsFixedLength* に **True** を渡します。この場合は、読み込む文字列の長さは正しいことが必要です。

- 配列型の変数を使う場合は、配列のサイズと次元を表す記述子を読み込むかどうかを選択できます。記述子を書き込むには、パラメータ *ArrayIsDynamic* を **True** に設定します。配列を読み取る場合は、配列の書き込み方法に合わせる必要があります。記述子を書き込む場合は、記述子を読み込む必要があります。記述子を使用しない場合は、**FileGet** 関数に渡される配列のサイズと境界によって読み込む対象が決定します。

  記述子は、配列のランク、サイズ、および各ランクの最小値を指定します。長さは、2 に 8 と次元数の積を加えた値 (*(2 + 8 * NumberOfDimensions)*) になります。**FileOpen** 関数のパラメータ *RecordLength* で指定するレコード長は、データと配列の記述子を書き込むために必要なバイト数の合計、またはそれ以上である必要があります。たとえば、次の配列を定義すると、配列がディスクに書き込まれる場合に 118 バイトが必要です。

  VB

```
Dim MyArray(4,9) As Integer
```

  118 バイトは次のように算出されます。

  - 記述子に 18 バイト (*(2 + 8 * 2)*)
  - データに 100 バイト (*(5 * 10 * 2)*)

- 記述子は読み込みません。その他の型の変数 (可変長文字列変数とオブジェクト型の変数以外) を使う場合、**FileGet** 関数は変数データだけを読み込みます。**FileOpen** 関数の *RecordLength* 句では、読み込むデータの長さ以上の値を指定する必要があります。

- 構造体の要素を **FileGet** 関数を使って読み込む場合、各要素は独立しているように読み込まれます。ただし、各要素の間には何も埋め込まれません。**FilePut** 関数を使用してユーザー定義型に含まれる動的配列をディスクに書き込むと、記述子が前置されます。この記述子の長さは、2 に 8 と次元数の積を加えた値 (*(2 + 8 * NumberOfDimensions)*) に等しくなります。**FileOpen** 関数の *RecordLength* 句で指定するレコード長は、データと配列の記述子を読み込むために必要なバイト数の合計、またはそれ以上である必要があります。**VBFixedString** 属性を構造体内の文字列フィールドに適用すると、ディスクに書き込むときに文字列のサイズを示すことができます。

Binary モード

ファイルを **Binary** モードで開いた場合も、いくつかの例外を除いて **Random** モードと同じ規則が適用されます。ファイルを **Binary** モードで開いた場合に適用される、**Random** モードとは異なる規則を次に示します。

- **FileOpen** 関数の *RecordLength* 句は何もしません。**FileGet** はすべての変数をディスクから連続して読み込みます。つまり、レコードの間にデータを埋め込みません。

- 構造体配列以外の配列の場合、**FileGet** 関数はデータだけを読み込みます。記述子は読み込みません。

- 構造体の要素でない可変長文字列を使用する場合、**FileGet** 関数は、2 バイトの記述子を読み込みません。読み込まれるバイト数は、文字列内の文字数と同じです。

セキュリティに関するメモ：
FileGet 関数を使ってファイルを読み込むには、FileIOPermissionAccess 列挙体の **Read** アクセスが必要です。

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileOpen 関数
FilePut 関数
Seek 関数
FileGetObject 関数

その他の技術情報

Visual Basic でのファイルの読み取り
Visual Basic でのファイルへの書き込み

FileGetObject 関数

ディスク ファイルからデータを読み込み、それを変数に格納します。

My 機能を使用すると、**FileGetObject** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Object, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Short, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Integer, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Single, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Double, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Decimal, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Byte, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Boolean, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As Date, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FileGetObject( _
   ByVal FileNumber As Integer, _
   ByRef Value As System.Array, _
```

```
    Optional RecordNumber As Integer = -1, _
    Optional ArrayIsDynamic As Boolean = False _
)
' -or-
Overloads Public Sub FileGetObject( _
    ByVal FileNumber As Integer, _
    ByRef Value As String, _
    Optional RecordNumber As Integer = -1, _
    Optional StringIsFixedLength As Boolean = False _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

Value

必ず指定します。読み込んだデータを格納する変数名。

RecordNumber

省略可能です。読み込みを始めるレコード番号 (**Random** モードのファイル) またはバイト位置 (**Binary** モードのファイル)。

ArrayIsDynamic

省略可能です。配列を書き込む場合にだけ適用されます。配列を動的に処理するかどうか、および配列のサイズと境界を表す配列記述子を書き込むかどうかを指定します。

StringIsFixedLength

省略可能です。文字列を書き込む場合にだけ適用されます。文字列の長さを表す 2 バイトの記述子を書き込むかどうかを指定します。既定では **False** です。

解説

整数型 (**Integer**)、長整数型 (**Long**)、Short 型 (**Short**) などではなくオブジェクト型 (**Object**) が渡される場合は、**FileGet** 関数の代わりに **FileGetObject** 関数を使って、コンパイル時のあいまいさを回避します。

バリアント型 (**Variant**) を書き出す場合は、**FileGetObject** を使う必要があります。判断に迷った場合、2 番目のパラメータにオブジェクトを使っているならば **FilePutObject** と **FileGetObject** を使った方が安全です。

FileGetObject は、**Random** モードおよび **Binary** モードの場合にだけ有効です。

通常、**FileGetObject** 関数を使用して読み込んだデータは、**FilePutObject** 関数を使用して書き込みます。

ファイルの中で先頭のレコード番号またはバイト位置は 1 になり、2 番目のレコード番号またはバイト位置は 2 になります。引数 *RecordNumber* を省略すると、**FileGetObject** は最後に呼び出した **FileGetObject** 関数または **FilePutObject** 関数、または最後に呼び出した **Seek** 関数が示す位置の次のレコードまたはバイトを読み込みます。

Random モード

Random モードで開いたファイルに対しては、次の規則が適用されます。

- 読み込むデータの長さが **FileOpen** 関数ステートメントの *RecordLength* 句で指定したレコード長よりも短ければ、**FileGetObject** 関数はレコード長の終端から次のレコードを読み込むことができます。レコードの終わりと次のレコードの先頭の間には、ファイル バッファの内容が埋め込まれます。埋め込まれるデータ量は正確にわからないので、読み込まれているデータの長さにレコード長を合わせます。
- 文字列変数を使う場合、**FileGetObject** 関数は、既定では文字列の長さを示す 2 バイトの記述子を読み込んでから、変数に格納するデータを読み込みます。したがって、**FileOpen** 関数の *RecordLength* 句では、文字列の実際の長さより 2 バイト以上大きい値を指定する必要があります。Visual Basic 6.0 以前のバージョンは固定長文字列をサポートしており、ファイルに書き込む場合、長さ記述子は書き込まれません。記述子なしで文字列を読み込む場合は、パラメータ *StringIsFixedLength* に **True** を渡します。この場合は、読み込む文字列の長さは正しいことが必要です。
- 配列型の変数を使う場合は、配列のサイズと次元を表す記述子を読み込むかどうかを選択できます。記述子を読み取るには、パラメータ *ArrayIsDynamic* を **True** に設定します。配列を読み取る場合は、配列の書き込み方法に合わせる必要があります。記述子を書き込む場合は、記述子を読み込む必要があります。記述子を使用しない場合は、**FileGetObject** 関数に渡される配列のサイズと境界を使って、読み込む対象を決定します。

  記述子は、配列のランク、サイズ、および各ランクの最小値を指定します。長さは、2 に 8 と次元数の積を加えた値 (*2 + 8 * NumberOfDimensions*) になります。**FileOpen** 関数のパラメータ *RecordLength* で指定するレコード長は、データと配列の記述子を

書き込むために必要なバイト数の合計、またはそれ以上である必要があります。たとえば、次の配列を定義した場合は、配列がディスクに書き込まれるときに 118 バイトが必要になります。

VB

```
Dim MyArray(4,9) As Integer
```

118 バイトは、記述子のための 18 バイト (2 + 8 * 2)、データのための 100 バイト (5 * 10 * 2) を合わせた値です。

- 構造体の要素を **FileGetObject** 関数を使って読み込む場合、各要素は独立しているように読み込まれます。ただし、各要素の間には何も埋め込まれません。**FilePutObject** 関数を使用してユーザー定義型に含まれる動的配列をディスクに書き込むと、記述子が前置されます。この記述子の長さは、2 に 8 と次元数の積を加えた値 (*2 + 8 * NumberOfDimensions*) に等しくなります。**FileOpen** 関数の *RecordLength* 句で指定するレコード長は、データと配列の記述子を読み込むために必要なバイト数の合計、またはそれ以上である必要があります。VBFixedStringAttribute クラス を構造体内の文字列フィールドに適用すると、ディスクに書き込むときに文字列のサイズを示すことができます。

Binary モード

ファイルを **Binary** モードで開いた場合も、次に示す例外を除いて、**Random** モードとすべて同じ規則が適用されます。

- **FileOpen** 関数の *RecordLength* 句は何もしません。**FileGetObject** はすべての変数をディスクから連続して読み込みます。つまり、レコードの間に何も埋め込みません。
- 構造体配列以外の配列の場合、**FileGetObject** 関数はデータだけを読み込みます。記述子は読み込みません。

構造体の要素でない可変長文字列を使用する場合、**FileGetObject** 関数は、2 バイトの記述子を読み込みません。読み込まれるバイト数は、文字列内の文字数と同じです。

セキュリティに関するメモ：
ファイルからデータを読み取るときは、ファイル名の拡張子に基づいてファイルの内容を判断しないでください。たとえば、Form1.vb という名前のファイルが Visual Basic のソース ファイルとは限りません。

使用例

次に示すのは、テスト用のファイルにレコードを読み込んでからそれを取得する例です。

VB

```
Dim c As Object = "test"
FileSystem.FileOpen(1, "test.dat", OpenMode.Binary)
FileSystem.FilePutObject(1, "ABCDEF")
FileSystem.Seek(1, 1)
FileSystem.FileGetObject(1, c)
MsgBox(c)
FileSystem.FileClose(1)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FilePut 関数
FileOpen 関数
Seek 関数
FileGet 関数

その他の技術情報

FileLen 関数

ファイルの長さをバイト単位で示す長整数型 (**Long**) の値を返します。

My 機能を使用すると、**FileLen** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem.GetFileInfo メソッド」を参照してください。

```
Public Function FileLen(ByVal PathName As String) As Long
```

パラメータ

PathName

必ず指定します。ファイル名を指定する文字列 (**String**) 式です。*PathName* にはディレクトリ名またはフォルダ名、およびドライブ名も含めて指定できます。

例外

例外の種類	エラー番号	条件
FileNotFoundException	53	ファイルが存在していません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

指定したファイルが **FileLen** 関数の呼び出し時に開いていた場合は、ファイルが開かれた時点でのファイルのサイズを表す値を返します。

メモ :
開いているファイルの現在の長さを取得するには、**LOF** 関数を使用します。

使用例

FileLen 関数を使って、ファイルの長さをバイト単位で取得するコード例を次に示します。この例では、`TestFile` はデータを含むファイルであると仮定します。

VB

```
Dim MySize As Long
' Returns file length (bytes).
MySize = FileLen("TESTFILE")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileDateTime 関数
GetAttr 関数
LOF 関数
FileNotFoundException Class

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

FileOpen 関数

ファイルを開いて入出力を行います。

My 機能を使用すると、**FileOpen** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Sub FileOpen( _
   ByVal FileNumber As Integer, _
   ByVal FileName As String, _
   ByVal Mode As OpenMode, _
   Optional ByVal Access As OpenAccess = OpenAccess.Default, _
   Optional ByVal Share As OpenShare = OpenShare.Default, _
   Optional ByVal RecordLength As Integer = -1 _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。**FreeFile** 関数を使用して、使用できる次のファイル番号を取得します。

FileName

必ず指定します。ファイル名を指定する文字列 (**String**) 式です。ディレクトリ名またはフォルダ名、およびドライブ名も含めて指定できます。

Mode

必ず指定します。ファイル モード (**Append**、**Binary**、**Input**、**Output**、または **Random**) を指定する列挙体です。詳細については、「OpenMode 列挙型」を参照してください。

Access

省略可能です。開いているファイルに対して許可された操作 (**Read**、**Write**、または **ReadWrite**) を指定する列挙体です。既定値は **ReadWrite** です。詳細については、「OpenAccess 列挙型」を参照してください。

Share

省略可能です。開いているファイルに対する、他のプロセスによる許可されない操作 (**Shared**、**Lock Read**、**Lock Write**、および **Lock Read Write**) を指定する列挙体です。既定値は **Lock Read Write** です。詳細については、「OpenShare 列挙型」を参照してください。

RecordLength

省略可能です。32,767 バイト以下の数値。ランダム アクセス ファイルの場合は、レコード長を表します。シーケンシャル ファイルの場合は、バッファの容量を表します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	**Access**、**Share**、または **Mode** が無効です。
ArgumentException	5	**WriteOnly** のファイルが **Input** で開かれています。
ArgumentException	5	**ReadOnly** のファイルが **Output** で開かれています。
ArgumentException	5	**ReadOnly** のファイルが **Append** で開かれています。
ArgumentException	5	レコード長が、-1 ではない負の値です。
IOException	52	*FileNumber* が無効です (< -1 または > 255)。または *FileNumber* が既に使われています。
IOException	55	*FileName* が既に開かれています。または *FileName* が無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番

号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

FileOpen 関数は下位互換性のために提供されており、パフォーマンスを低下させる可能性があります。非レガシ アプリケーションに対しては、**My.Computer.FileSystem** オブジェクトを使用した方が、パフォーマンスが高くなります。詳細については、「Visual Basic におけるファイル アクセス」を参照してください。

ファイルに対して I/O 処理を実行するには、その前にファイルを開く必要があります。**FileOpen** はファイルへの I/O 用にバッファを割り当て、そのバッファでどのアクセス モードを使用するかを判断します。

セキュリティに関するメモ：

ファイルへの書き込みを行うとき、書き込み先のファイルが存在しなければ、アプリケーションでファイルを作成することが必要になる場合もあります。その処理を行うには、ファイルの作成先となるディレクトリへのアクセス許可が必要です。ただし、*FileName* で指定されたファイルが存在すると、アプリケーションに必要な権限は、ファイルそのものだけへの **Write** アクセス許可になります。できれば、配置時にファイルを作成し、ディレクトリ全体ではなくそのファイルだけへの **Write** アクセス許可を与える方が安全です。安全性を高めるために、ルート ディレクトリや Program Files ディレクトリではなく、ユーザー ディレクトリへデータを書き込むようにしてください。

開くチャネルは、**FreeFile()** 関数を使って検索できます。

セキュリティに関するメモ：

FileOpen 関数には **FileIOPermissionAccess** 列挙体の **Read**アクセスが必要です。ただし、部分的に信頼されている状況でこれを使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、FileIOPermissionAccess 列挙体および「アクセス許可の要求」を参照してください。

使用例

FileOpen 関数を使ってファイルへの入出力を可能にする、さまざまな例を示します。

ファイル `TestFile` を **Input** モードで開くコード例は、次のとおりです。

VB

```
FileOpen(1, "TESTFILE", OpenMode.Input)
' Close before reopening in another mode.
FileClose(1)
```

ファイルを書き込み専用の **Binary** モードで開く例を次に示します。

VB

```
FileOpen(1, "TESTFILE", OpenMode.Binary,OpenAccess.Write)
' Close before reopening in another mode.
FileClose(1)
```

ファイルを **Random** モードで開く例を次に示します。ファイルには、構造体 `Person` のレコードが含まれているものと仮定します。

VB

```
Structure Person
    <VBFixedString(30)> Dim Name As String
    Dim ID As Integer
End Structure
Public Sub ExampleMethod()
    ' Count 30 for the string, plus 4 for the integer.
    FileOpen(1, "TESTFILE", OpenMode.Random, , , 34)
    ' Close before reopening in another mode.
    FileClose(1)
End Sub
```

ファイルを **Output** モードで開く例を次に示します。どのプロセスからでも、ファイルの読み書きができます。

VB

```
FileOpen(1, "TESTFILE", OpenMode.Output, OpenAccess.Default, OpenShare.Shared)
' Close before reopening in another mode.
FileClose(1)
```

ファイルを読み込み専用の **Binary** モードで開く例を次に示します。他のプロセスから、このファイルを読むことはできません。

VB

```
FileOpen(1, "TESTFILE", OpenMode.Binary, OpenAccess.Read, _
   OpenShare.LockRead)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileClose 関数

FreeFile 関数

その他の技術情報

Visual Basic でのファイルの読み取り

Visual Basic でのファイルへの書き込み

FilePut 関数

変数の内容をディスク上のファイルに書き込みます。

My 機能により、**FilePut** よりも、ファイルの I/O 操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Short, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Integer, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Single, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Double, _
      RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Decimal, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Byte, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Boolean, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As Date, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Public Overloads Sub FilePut( _
      FileNumber As Integer, _
      Value As System.Array, _
   Optional RecordNumber As Integer = -1, _
   Optional ArrayIsDynamic As Boolean = False _
)
' -or-
Public Overloads Sub FilePut (
      FileNumber As Integer,
```

```
        Value As String, _
    Optional RecordNumber As Integer = -1,
    Optional StringIsFixedLength As Boolean = False
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

Value

必ず指定します。ディスクに書き込むデータが格納されている有効な変数の名前を指定します。

RecordNumber

省略可能です。書き込みを行うレコード番号 (**Random** モード ファイル) またはバイト位置 (**Binary** モード ファイル)。

ArrayIsDynamic

省略可能です。配列を書き込む場合にだけ適用されます。配列を動的に処理するかどうか、および配列の長さを表す文字列に配列記述子を書き込むかどうかを指定します。

StringIsFixedLength

省略可能です。文字列を書き込む場合にだけ適用されます。文字列に関する 2 バイトの記述子をファイルに書き込むかどうかを指定します。既定のクラスは **False** です。

例外

例外の種類	エラー番号	条件
ArgumentException	63	*RecordNumber* が -1 ではない 1 未満の値です。
IOExceptio	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

FilePut は、**Random** モードおよび **Binary** モードの場合にだけ有効です。

通常、**FilePut** 関数を使用して書き込んだデータは、**FileGet** 関数で読み込みます。

ファイルの中で先頭のレコード番号またはバイト位置は 1 になり、2 番目のレコード番号またはバイト位置は 2 になります。引数 *RecordNumber* を省略すると、最後に呼び出した **FileGet** 関数または **FilePut** 関数、あるいは最後に呼び出した **Seek** 関数が示す位置の次のレコードまたはバイトが書き込まれます。

StringIsFixedLength 引数は、関数が文字列を可変長と固定長どちらで解釈するかを制御します。引数が **True** のとき、**FilePut** は長さ記述子を書き込みません。**FilePut** で *StringIsFixedLength* = **True** を指定した場合は、**FileGet** でも同じように指定する必要があります。また、文字列が正しい長さに初期化されているかどうかを確認する必要もあります。

ランダム モード

Random モードで開いたファイルの場合は、次の規則が適用されます。

- 書き込むデータの長さが **FileOpen** 関数の *RecordLength* 句で指定した長さを超えない限り、**FilePut** 関数は次のレコードを直前のレコードの終端から書き込みます。レコードの終わりと次のレコードの先頭の間には、ファイル バッファの内容が埋め込まれます。埋め込まれるデータの量は指定できないため、通常はレコード長を書き込むデータ長に合わせます。書き込むデータの長さが **FileOpen** 関数の *RecordLength* 句で指定した長さを超えると、例外が発生します。
- 可変長文字列変数を使う場合、**FilePut** 関数は文字列の長さを示す 2 バイトの記述子を書き込んでから、変数にデータを書き込みます。**FileOpen** 関数の *RecordLength* 句では、文字列の実際の長さより 2 バイト以上大きい値を指定する必要があります。
- 書き込む変数が数値型を格納するオブジェクトである場合、**FilePut** 関数は、オブジェクトの **VarType** を表す 2 バイトのデータを書き込み、次に変数のデータを書き込みます。たとえば、整数型を格納するオブジェクトを書き込む場合、**FilePut** 関数は 6 バイトを書き込

みます。内訳は、**VarType(3)** (**Integer**) を示す 2 バイトと、データを格納する 4 バイトです。**FileOpen** 関数のパラメータ *RecordLength*には、実際にデータを格納するために必要なバイト数より 2 バイト以上大きい値を指定する必要があります。

- 文字列を格納するオブジェクト型の変数を使う場合、**FilePut** 関数は、オブジェクトの **VarType(8)** を示す 2 バイトの記述子を書き込んでから、文字列の長さを示す 2 バイトの記述子を書き込み、その後、文字列データを書き込みます。**FileOpen** 関数のパラメータ *RecordLength* で指定するレコード長には、文字列の実際の長さより 4 バイト以上大きい値を指定する必要があります。記述子なしで文字列を書き込む場合は、パラメータ *StringIsFixedLength* に **True** を渡します。この場合は、正しい長さの文字列を読み込む必要があります。

- 配列型の変数を使う場合は、配列のサイズと次元を表す記述子を書き込むかどうかを選択できます。Visual Basic 6.0 以前のバージョンでは、動的配列にファイル記述子を書き込みます。固定サイズの配列には書き込みません。Visual Basic 2005 の既定設定では、記述子を書き込みません。記述子を書き込むには、パラメータ *ArrayIsDynamic* を **True** に設定します。配列を書き込むときは、配列の読み取り方法に合わせる必要があります。記述子を使って読み取る場合は、記述子を書き込む必要があります。記述子は、配列のランク、サイズ、および各ランクの最小値を指定します。長さが 2 + 次元の数 X 8 に等しい場合は *(2 + 8 * NumberOfDimensions)* です。**FileOpen** 関数の *RecordLength* 句で指定するレコード長は、データと配列の記述子を書き込むために必要なバイト数の合計、またはそれ以上である必要があります。たとえば、次の配列を定義すると、配列がディスクに書き込まれる場合に 118 バイトが必要です。

  VB
  ```
  Dim MyArray(4,9) As Integer
  ```

- その他の型の変数 (可変長文字列変数とオブジェクト型変数以外) を使う場合、**FilePut** 関数は、変数のデータだけを書き込みます。**FileOpen** 関数の *RecordLength* 句では、書き込むデータの長さ以上の値を指定する必要があります。

- 構造体の要素を **FilePut** 関数を使って書き込む場合、各要素は独立しているかのように書き込まれますが、各要素の間には何も埋め込まれません。**VBFixedString** 属性を構造体内の文字列フィールドに適用すると、ディスクに書き込むときに文字列のサイズを示すことができます。

  メモ：

  文字列フィールドのバイト数が、**VBFixedString** 属性に指定した値よりも大きい場合は、ディスクに書き込むときに文字列が切り取られます。

バイナリ モード

Binary モードで開かれているファイルでは、**Random** モードの規則のほとんどが適用されますが、いくつか例外があります。**Binary** モードで開かれたファイルの次の規則は、**Random** モードの規則と異なります。

- **FileOpen** 関数の *RecordLength* 句は無効です。**FilePut** はすべての変数を連続して (レコードの間にパディングを入れずに) ディスクに書き込みます。

- 構造体配列以外の配列の場合、**FilePut** 関数は、データだけを書き込みます。記述子は書き込みません。

- 構造体の要素でない可変長文字列を使用する場合、**FilePut** 関数は、2 バイトの記述子を書き込みません。書き込まれるバイト数は、文字列内の文字数と同じです。たとえば、次のステートメントは、ファイル番号 1 のファイルに 11 バイトのデータを書き込みます。

  VB
  ```
  Dim hellow As String = "Hello World"
  FilePut(1,hellow)
  ```

- **FilePut** 関数を使用してファイルにデータを書き込むには、FileIOPermissionAccess 列挙体からの **Write** アクセスが必要です。

使用例

FilePut 関数を使って、データをファイルに書き込む例を次に示します。この例では、構造体 `Person` の 5 つのレコードをファイルに書き込みます。

VB
```
Structure Person
   Public ID As Integer
   Public Name As String
```

```
End Structure

Sub WriteData()
   Dim PatientRecord As Person
   Dim recordNumber As Integer
'    Open file for random access.
   FileOpen(1, "C:\TESTFILE.txt", OpenMode.Binary)
   ' Loop 5 times.
   For recordNumber = 1 To 5
      ' Define ID.
      PatientRecord.ID = recordNumber
      ' Create a string.
      PatientRecord.Name = "Name " & recordNumber
      ' Write record to file.
      FilePut(1, PatientRecord)
   Next recordNumber
   FileClose(1)
End Sub
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileGet 関数
FileOpen 関数
Seek 関数
FileGetObject 関数
VBFixedStringAttribute クラス
Len 関数 (Visual Basic)
ArgumentException
IOException

その他の技術情報

Visual Basic でのファイルへの書き込み

FilePutObject 関数

変数の内容をディスク上のファイルに書き込みます。

My 機能により、**FilePutObject** よりも、ファイルの I/O 操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Object, _
      RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Short, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Integer, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Single, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Double, _
      RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Decimal, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Byte, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Boolean, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As Date, _
   Optional RecordNumber As Integer = -1 _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As System.Array, _
```

```
    Optional RecordNumber As Integer = -1, _
    Optional ArrayIsDynamic As Boolean = False _
)
' -or-
Overloads Public Sub FilePutObject( _
      FileNumber As Integer, _
      Value As String, _
    Optional RecordNumber As Integer = -1, _
    Optional StringIsFixedLength As Boolean = False _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

Value

必ず指定します。ディスクに書き込むデータが格納されている有効な変数の名前を指定します。

RecordNumber

省略可能です。書き込みを行うレコード番号 (**Random** モード ファイル) またはバイト位置 (**Binary** モード ファイル)。

ArrayIsDynamic

省略可能です。配列を書き込む場合にだけ適用されます。配列を動的に処理するかどうか、および配列の長さを表す文字列に配列記述子を書き込むかどうかを指定します。

StringIsFixedLength

省略可能です。文字列を書き込む場合にだけ適用されます。文字列の長さを表す記述子を書き込むかどうかを指定します。既定は **False** です。

解説

整数型 (**Integer**)、長整数型 (**Long**)、Short 型 (**Short**) などではなくオブジェクト型 (**Object**) が渡される場合は、**FilePut** 関数の代わりに **FilePutObject** 関数を使って、コンパイル時のあいまいさを回避します。

FilePutObject は、オブジェクトを記述する記述子の書き込みおよび読み取りを行います。バリアント型 (**Variant**) を書き出す場合は、**FilePutObject** を使う必要があります。判断に迷った場合は、2 番目のパラメータにオブジェクトを使っているときは **FilePutObject** と **FileGetObject** を使った方が安全です。

FilePutObject は、**Random** モードおよび **Binary** モードの場合にだけ有効です。

通常、**FilePutObject** 関数を使用して書き込んだデータは、**FileGetObject** 関数で読み込みます。

ファイルの中で先頭のレコード番号またはバイト位置は 1 になり、2 番目のレコード番号またはバイト位置は 2 になります。引数 *RecordNumber* を省略すると、最後に呼び出した **FileGetObject** 関数または **FilePutObject** 関数、あるいは最後に呼び出した **Seek** 関数が示す位置の次のレコードまたはバイトが書き込まれます。

StringIsFixedLength 引数は、関数が文字列を可変長と固定長どちらで解釈するかを制御します。引数が **True** のとき、**FilePutObject** は長さの記述子を書き込みません。**FilePutObject** で *StringIsFixedLength* = **True** を指定した場合は、**FileGetObject** でも同じように指定する必要があります。また、文字列が正しい長さに初期化されているかどうかを確認する必要もあります。

ランダム モード

Random モードで開いたファイルの場合は、次の規則が適用されます。

- 書き込むデータの長さが **FileOpen** 関数の *RecordLength* 句で指定した長さを超えない限り、**FilePutObject** 関数は次のレコードを直前のレコードの終端から書き込みます。レコードの終わりと次のレコードの先頭の間には、ファイル バッファの内容が埋め込まれます。埋め込まれるデータ量は正確にわからないので、通常は書き込まれているデータの長さにレコード長を合わせます。書き込むデータの長さが **FileOpen** 関数の *RecordLength* 句で指定した長さを超えると、例外が発生します。
- 書き込む変数が数値型を格納するオブジェクトである場合、**FilePutObject** 関数は、オブジェクトの **VarType** を表す 2 バイトのデータを書き込み、次に変数のデータを書き込みます。たとえば、整数型を格納するオブジェクトを書き込む場合、**FilePutObject** 関数は 6 バイトを書き込みます。内訳は、**VarType(**3**)** (**Integer**) を示す 2 バイトと、データを格納する 4 バイトです。**FileOpen** 関数のパラメータ *RecordLength*には、実際にデータを格納するために必要なバイト数より 2 バイト以上大きい値を指定する必要があります。
- 文字列を格納するオブジェクト型の変数を使う場合、**FilePutObject** 関数は、オブジェクトの **VarType(**8**)** を示す 2 バイトの記述子を書き込んでから、文字列の長さを示す 2 バイトの記述子を書き込み、その後、文字列データを書き込みます。**FileOpen** 関数のパラ

メータ *RecordLength* で指定するレコード長には、文字列の実際の長さより 4 バイト以上大きい値を指定する必要があります。記述子なしで文字列を書き込む場合は、パラメータ *StringIsFixedLength* に **True** を渡します。この場合は、正しい長さの文字列を読み込む必要があります。

- 配列型の変数を使う場合は、配列のサイズと次元を表す記述子を書き込むかどうかを選択できます。Visual Basic 6.0 以前のバージョンでは、動的配列にファイル記述子を書き込みますが、固定サイズの配列には書き込みません。Visual Basic では、既定では記述子は書き込まれません。記述子を書き込むには、パラメータ *ArrayIsDynamic* を **True** に設定します。配列を書き込むときは、配列の読み込み方法に合わせる必要があります。記述子を読み込んだ場合は、記述子を書き込む必要があります。記述子は、配列のランク、サイズ、および各ランクの最小値を指定します。長さが 2 + 次元の数×8 に等しい場合は *(2 + 8 * NumberOfDimensions)* です。**FileOpen** 関数の *RecordLength* 句で指定するレコード長は、データと配列の記述子を書き込むために必要なバイト数の合計、またはそれ以上である必要があります。

バイナリ モード

Binary モードで開いたファイルの場合は、次に示す規則を除いて、**Random** モードの規則がすべて適用されます。

- **FileOpen** 関数の *RecordLength* 句は無効です。**FilePutObject** はすべての変数を連続して (レコードの間にパディングを入れずに) ディスクに書き込みます。

使用例

FilePutObject 関数を使って文字列をファイルに書き込む例を次に示します。

VB

```
Sub WriteData()
    Dim text As String = "test"
    FileOpen(1, "test.bin", OpenMode.Binary)
    FilePutObject(1, text)
    FileClose(1)
End Sub
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileGet 関数
FileOpen 関数
Seek 関数
FilePut 関数
My.Computer.FileSystem オブジェクト

その他の技術情報

Visual Basic でのファイルへの書き込み

FileWidth 関数

FileOpen 関数で開いたファイルに出力する桁数を割り当てます。

```
Public Sub FileWidth( _
      FileNumber As Integer, _
      RecordWidth As Integer _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

RecordWidth

必ず指定します。1 行に表示される文字数を表す 0 ～ 255 の範囲の数式。引数 *RecordWidth* に 0 を指定すると、行の長さは制限されません。*RecordWidth* の既定値は 0 です。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

この関数は、**FileOpen** 関数で開いたファイルに出力する行の桁数を割り当てます。

使用例

FileWidth 関数を使って、ファイルの出力行の幅を設定する例を次に示します。

VB

```
Dim i As Integer
FileOpen(1, "TESTFILE", OpenMode.Output) ' Open file for output.
FileWidth(1, 5)   ' Set output line width to 5.
For i = 0 To 9   ' Loop 10 times.
   Print(1, Chr(48 + I))   ' Prints five characters per line.
Next
FileClose(1)   ' Close file.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileOpen 関数

Print 関数、PrintLine 関数

IOException Class

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

Filter 関数 (Visual Basic)

指定されたフィルタ条件に基づいた文字列 (**String**) 配列のサブセットを含むゼロ ベースの配列を返します。

```
Function Filter(
   ByVal Source() As { Object | String },
   ByVal Match As String,
   Optional ByVal Include As Boolean = True,
   Optional ByVal Compare As CompareMethod = CompareMethod.Binary
)  As String()
```

パラメータ

Source

必ず指定します。検索先の 1 次元配列の文字列を指定します。

Match

必ず指定します。検索する文字列を指定します。

Include

省略可能です。*Match* を含む、または含まない部分文字列を返すかどうかを表すブール (**Boolean**) 値です。*Include* が **True** の場合、**Filter** 関数は、配列の各要素の文字列の中で、*Match* が含まれる配列のサブセットを返します。*Include* が **False** の場合、**Filter** 関数は、配列の各要素の文字列の中で、*Match* が含まれない配列のサブセットを返します。

Compare

省略可能です。文字列式を評価するときに使用する文字列比較のモードを表す数値を指定します。値については、「設定」を参照してください。

設定

引数 *Compare* の設定値は次のとおりです。

定数	説明
CompareMethod.Binary	バイナリ モードで比較を行います。
CompareMethod.Text	テキスト モードで比較を行います。

例外

例外の種類	エラー番号	条件
ArgumentException	9	*Source* が **Nothing** であるか、1 次元の配列ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

Match に一致するものが *Source* にない場合、**Filter** 関数は空の配列を返します。*Source* が **Nothing** であるか、1 次元配列でない場合は、エラーになります。

Filter 関数が返す配列は、一致した項目数分だけの要素が含まれています。

使用例

Filter 関数の使用例を次に示します。

VB

```
Dim TestStrings(2) As String
TestStrings(0) = "This"
TestStrings(1) = "Is"
TestStrings(2) = "It"
```

```
Dim subStrings() As String
' Returns ["This", "Is"].
subStrings = Filter(TestStrings, "is", True, CompareMethod.Text)
' Returns ["This"].
subStrings = Filter(TestStrings, "is", True, CompareMethod.Binary)
' Returns ["Is", "It"].
subStrings = Filter(TestStrings, "is", False, CompareMethod.Binary)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要

Replace 関数 (Visual Basic)

ArgumentException

Int 関数、Fix 関数 (Visual Basic)

指定した数値の整数部分を返します。

```
Public Shared Function Int( _
    ByVal Number As { Double | Integer | Long | Object | Short | Single | Decimal }) _
    As { Double | Integer | Long | Object | Short | Single | Decimal }
Public Shared Function Fix( _
    ByVal Number As { Double | Integer | Long | Object | Short | Single | Decimal }) _
    As { Double | Integer | Long | Object | Short | Single | Decimal }
```

パラメータ

Number

必ず指定します。倍精度浮動小数点数型 (**Double**) の数値または任意の数式を指定します。*Number* に **Nothing** が含まれている場合は、**Nothing** を返します。

例外

例外の種類	エラー番号	条件
ArgumentNullException	5	Number が指定されていません。
ArgumentException	5	Number が数値型ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Int 関数と **Fix** 関数は、どちらも *Number* の小数部分を取り除いた整数値を返します。

Int と **Fix** の違いは、*Number* に負の値を指定した場合に現れます。**Int** 関数が *Number* 以下の最大の負の整数を返すのに対して、**Fix** 関数は *Number* 以上の最小の負の整数を返します。たとえば、-8.4 を指定した場合、**Int** 関数は -9、**Fix** 関数は -8 をそれぞれ返します。

`Fix(number)` これは、次のコードと同じです。`Sign(number) * Int(Abs(number))`.

使用例

Int 関数と **Fix** 関数を使って、数の整数部分を返す例を次に示します。負の値を指定した場合、**Int** 関数がその数値以下の最大の負の整数を返すのに対して、**Fix** 関数はその数値以上の最小の負の整数を返します。この例では、**Option Strict Off** を指定する必要があります。これは、**Option Strict On** だと、倍精度浮動小数点数型 (**Double**) から整数型 (**Integer**) への暗黙の型変換が許可されないためです。

VB

```
' This code requires Option Strict Off
Dim MyNumber As Integer
MyNumber = Int(99.8)   ' Returns 99.
MyNumber = Fix(99.8)   ' Returns 99.

MyNumber = Int(-99.8)  ' Returns -100.
MyNumber = Fix(-99.8)  ' Returns -99.

MyNumber = Int(-99.2)  ' Returns -100.
MyNumber = Fix(-99.2)  ' Returns -99.
```

CInt 関数を使用すると、**Option Strict Off** の場合に、他のデータ型を明示的に整数型 (**Integer**) に変換できます。ただし、**CInt** は、数値の小数部を切り捨てるのではなく、最も近い整数に丸めます。以下に例を示します。

VB

```
MyNumber = CInt(99.8)    ' Returns 100.
MyNumber = CInt(-99.8)   ' Returns -100.
```

```
MyNumber = CInt(-99.2)    ' Returns -99.
```

Fix または **Int** の呼び出しの結果に対して **CInt** 関数を使用すると、値を丸めずに明示的に整数に変換できます。以下に例を示します。

VB

```
MyNumber = CInt(Fix(99.8))    ' Returns 99.
MyNumber = CInt(Int(99.8))    ' Returns 99.
```

CInt の詳細については、「データ型変換関数」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Conversion**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

データ型変換関数
整数型 (Integer) (Visual Basic)
数値演算の概要
数値演算関数 (Visual Basic)
変換の概要
ArgumentNullException

Format 関数

書式指定文字列 (**String**) 式に含まれる指示に従って書式設定された文字列を返します。

```
Public Shared Function Format( _
   ByVal Expression As Object, _
   Optional ByVal Style As String = "" _
) As String
```

パラメータ

Expression

必ず指定します。任意の有効な式。

Style

省略可能です。既定またはユーザー定義の書式指定文字列 (**String**) 式を指定します。

設定

Style 引数の作成方法については、次の各トピックを参照してください。

書式指定の対象	方法
数値	定義済み数値書式 (Format 関数)を使用するか、ユーザー定義の数値書式 (Format 関数)を作成します。
日付/時刻	定義済みの日付/時刻書式 (Format 関数)を使用するか、ユーザー定義の日付/時刻書式 (Format 関数)を作成します。
日付/時刻のシリアル番号	日付/時刻書式または数値書式を使用します。

Style を指定せずに数値書式を指定する場合、ロケール情報に対応している点以外は、**Format** 関数は **Str** 関数と同様に機能します。ただし、**Format** 関数で文字列として設定された正の数値には、先頭にプラス記号を表示するためのスペースは含まれません。**Str** 関数を使って変換された正の数値には先頭にスペースがあります。

解説

非ローカライズ数値文字列の書式を指定する場合は、ユーザー定義の数値書式設定を使って、目的の書式になるようにします。

System.String.Format メソッドも同様の機能を提供します。

使用例

String の書式指定とユーザー定義の書式指定の両方を使って値の書式を指定する、**Format** 関数のさまざまな使用例を次に示します。日付の区切り記号 (**/**)、時刻の区切り記号 (**:**)、および午前/午後を示す文字 (**t** および **tt**) について、システムで実際に表示される書式は、コードが使用するロケール設定によって決まります。時刻と日付を開発環境で表示する場合は、コード ロケールの短い時刻書式と短い日付書式が使用されます。

メモ :
24 時間制を使用するロケールでは、午前/午後を示す記号 (**t** および **tt**) では何も表示されません。

VB

```
Dim TestDateTime As Date = #1/27/2001 5:04:23 PM#
Dim TestStr As String
' Returns current system time in the system-defined long time format.
TestStr = Format(Now(), "Long Time")
' Returns current system date in the system-defined long date format.
TestStr = Format(Now(), "Long Date")
' Also returns current system date in the system-defined long date
' format, using the single letter code for the format.
TestStr = Format(Now(), "D")
```

```
' Returns the value of TestDateTime in user-defined date/time formats.
' Returns "5:4:23".
TestStr = Format(TestDateTime, "h:m:s")
' Returns "05:04:23 PM".
TestStr = Format(TestDateTime, "hh:mm:ss tt")
' Returns "Saturday, Jan 27 2001".
TestStr = Format(TestDateTime, "dddd, MMM d yyyy")
' Returns "17:04:23".
TestStr = Format(TestDateTime, "HH:mm:ss")
' Returns "23".
TestStr = Format(23)

' User-defined numeric formats.
' Returns "5,459.40".
TestStr = Format(5459.4, "##,##0.00")
' Returns "334.90".
TestStr = Format(334.9, "###0.00")
' Returns "500.00%".
TestStr = Format(5, "0.00%")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
数値ごとに異なる書式指定 (Format 関数)
定義済みの日付/時刻書式 (Format 関数)
定義済み数値書式 (Format 関数)
Str 関数
データ型変換関数
ユーザー定義の日付/時刻書式 (Format 関数)
ユーザー定義の数値書式 (Format 関数)
Format

数値ごとに異なる書式指定 (Format 関数)

数値に使用するユーザー定義の書式指定式は、セミコロンで区切られたセクションを 1 つから 3 つまで指定できます。**Format** 関数の *Style* 引数に定義済みの数値書式が含まれる場合は、1 つ以上のセクションを持つことはできません。

使用するセクション数	結果
1 つ	書式指定式はすべての値に適用されます。
2 つ	最初のセクションが正の値とゼロに適用され、2 番目のセクションが負の値に適用されます。
3 つ	最初のセクションが正の値、2 番目のセクションが負の値、3 番目のセクションがゼロに適用されます。

次の例には 2 つのセクションがあります。最初のセクションは正の値とゼロの書式を定義し、2 番目のセクションは負の値の書式を定義します。**Format** 関数の *Style* 引数は文字列なので、引用符で囲みます。

VB

```
Dim Style1 As String = "$#,##0;($#,##0)"
```

セミコロンを入れて、その間に何も指定しない場合、そのセクションは正の値の書式を使って表示されます。たとえば、次の書式では、最初のセクションの書式指定を使って正と負の値が表示され、値がゼロの場合は `Zero` と表示されます。

VB

```
Dim Style2 As String = "$#,##0;;\Z\e\r\o"
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

定義済み数値書式 (Format 関数)

Format 関数

文字列操作の概要

定義済み数値書式 (Format 関数)

定義済み数値書式名を次の表に示します。これらは **Format** 関数の style 引数として名前で指定できます。

書式名	説明
General Number、**G**、または **g**	3 桁ごとの区切り記号を付けずに数値を表示します。
Currency、**C**、または **c**	3 桁ごとに区切り記号を付けて数値を表示します。小数部の数値がある場合、小数部 2 桁まで表示します。出力はシステムのロケール設定に基づきます。
Fixed、**F**、または **f**	少なくとも整数部 1 桁、小数部 2 桁を表示します。
Standard、**N**、または **n**	3 桁ごとに区切り記号を付けて、数値を表示します。少なくとも整数部 1 桁、小数部 2 桁を表示します。
Percent	数値を 100 倍して、右側にパーセント記号 (%) を付けて表示します。小数部は常に 2 桁です。
P、または **p**	数値を 100 倍して、右側に空白 1 文字とパーセント記号 (%) を付け、3 桁ごとに区切り記号を付けて数値を表示します。小数部は常に 2 桁です。
Scientific	上位 2 桁を表示する標準の指数表記を使用します。
E、または **e**	上位 6 桁を表示する標準の指数表記を使用します。
D、または **d**	数値を 10 進数形式で文字列として表示します。このオプションは整数型 (**Byte**、**Short**、**Integer**、**Long**) だけをサポートします。
X、または **x**	数値を 16 進数形式で文字列として表示します。このオプションは整数型 (**Byte**、**Short**、**Integer**、**Long**) だけをサポートします。
Yes/No	数値が 0 の場合は `No` を表示します。それ以外の場合は `Yes` を表示します。
True/False	数値が 0 の場合は `False` を表示します。それ以外の場合は `True` を表示します。
On/Off	数値が 0 の場合は `Off` を表示します。それ以外の場合は `On` を表示します。

スマート デバイス開発者のためのメモ

Yes/No 形式、**True/False** 形式、**On/Off** 形式はサポートされません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
変換の概要
Format 関数
定義済みの日付/時刻書式 (Format 関数)
ユーザー定義の数値書式 (Format 関数)

定義済みの日付/時刻書式 (Format 関数)

定義済みの日付/時刻書式名を次の表に示します。これらは **Format** 関数の style 引数として名前で指定できます。

書式名	説明
General Date、または **G**	日付か時刻、または両方を表示します。たとえば、`4/3/93 05:34 PM` とします。日付表示は、アプリケーションの現在のカルチャの値で決まります。
Long Date、**Medium Date**、または **D**	現在のカルチャの長い日付書式に従って日付を表示します。
Short Date、または **d**	現在のカルチャの短い日付書式に従って日付を表示します。
Long Time、**Medium Time**、または **T**	現在のカルチャの長い日付書式に従って、通常、時分秒を含む時刻を表示します。
Short Time、または **t**	現在のカルチャの短い時刻の形式に従って時刻を表示します。
f	現在のカルチャの書式に従って、長い日付と短い時刻を表示します。
F	現在のカルチャの書式に従って、長い日付と長い時刻を表示します。
g	現在のカルチャの書式に従って、短い日付と短い時刻を表示します。
M, **m**	月と日を表示します。
R, **r**	日付と時刻をグリニッジ標準時 (GMT) で表します。
s	日付と時刻を並べ替え可能なインデックスとして設定します。
u	日付と時刻を GMT 形式の並べ替え可能なインデックスとして設定します。
U	日付と時刻を GMT 形式の長い日付と長い時刻で設定します。
Y, **y**	日付を年と月として設定します。

アプリケーションの現在のカルチャ情報については、「Visual Basic においてカルチャが文字列に与える影響」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
変換の概要
Format 関数
定義済み数値書式 (Format 関数)
ユーザー定義の日付/時刻書式 (Format 関数)

ユーザー定義の日付/時刻書式 (Format 関数)

ユーザー定義の日付/時刻書式を作成するときに使用できる文字を次の表に示します。以前のバージョンの Visual Basic とは異なり、これらの形式指定文字の大文字と小文字は区別されます。

文字	説明
(**:**)	時刻の区切り記号。ロケールによっては、時刻の区切り記号を表すのに別の記号が使用されます。時刻の区切り記号は、時刻値を表すときに、時、分、および秒を区切ります。書式指定された出力で、実際に時刻の区切り記号として使用される記号は、アプリケーションの現在のカルチャ値によって決まります。
(**/**)	日付の区切り記号。ロケールによっては、日付の区切り記号を表すのに別の記号が使用されます。日付の区切り記号は、日付値を表すときに、年月日を区切ります。書式指定された出力で、実際に日付の区切り記号として使用される記号は、アプリケーションの現在のカルチャ値によって決まります。
(**%**)	後に続く文字には関係なく、次の文字を 1 文字書式指定として読み取る必要があることを示すために使用します。1 文字書式指定をユーザー定義の書式として読むことを示すためにも使用します。詳細については、以下の説明を参照してください。
d	先行するゼロを付けずに日付を表示します (たとえば、`1`)。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%d** を使用します。
dd	先行するゼロを付けて日付を表示します (たとえば、`01`)。
ddd	曜日を短縮形で表示します (たとえば、`Sun`)。
dddd	曜日をそのまま表示します (たとえば、`Sunday`)。
M	先行するゼロを付けずに月を表示します (たとえば、1 月は `1`)。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%M** を使用します。
MM	先行するゼロを付けて月を表示します (たとえば、`01/12/01`)。
MMM	月を短縮形で表示します (たとえば、`Jan`)。
MMMM	月をそのまま表示します (たとえば、`January`)。
gg	時代/年号を示す文字列を表示します (たとえば、`A.D.`)。
h	先行するゼロを付けずに、時を 12 時間制で表示します (たとえば、`1:15:15 PM`)。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%h** を使用します。
hh	先行するゼロを付けて、時を 12 時間制で表示します (たとえば、`01:15:15 PM`)。
H	先行するゼロを付けずに、時を 24 時間制で表示します (たとえば、`1:15:15`)。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%H** を使用します。

HH	先行するゼロを付けて、時を 24 時間制で表示します (たとえば、`01:15:15`)。
m	先行するゼロを付けずに、分を表示します (たとえば、`12:1:15`)。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%m** を使用します。
mm	先行するゼロを付けて、分を表示します (たとえば、`12:01:15`)。
s	先行するゼロを付けずに、秒を表示します (たとえば、`12:15:5`)。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%s** を使用します。
ss	先行するゼロを付けて、秒を表示します (たとえば、`12:15:05`)。
f	秒の小数部を表示します。たとえば、**ff** は 100 分の 1 秒を表示し、**ffff** は 10,000 分の 1 秒を表示します。ユーザー定義の書式では **f** 記号を 7 つまで使用できます。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%f** を使用します。
t	12 時間制を使用し、午前は大文字の `A`、午後 (11:59 P.M. まで) は大文字の `P` を表示します。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%t** を使用します。
tt	12 時間制を使用し、午前は大文字の `AM`、午後 (11:59 P.M. まで) は大文字の `PM` を表示します。 24 時間制を使用するロケールの場合、何も表示されません。
y	先行するゼロを付けずに、年を 1 桁の値 (0 ～ 9) で表示します。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%y** を使用します。
yy	先行するゼロを付けずに、年を 2 桁の数値で表示します。
yyy	年を 4 桁の数値書式で表示します。
yyyy	年を 4 桁の数値書式で表示します。
z	標準時との時差を先頭にゼロのない数値として表示します (たとえば、`-8`)。ユーザー定義の数値書式でこれが唯一の文字である場合は、**%z** を使用します。
zz	標準時との時差を先頭にゼロのある数値として表示します (たとえば、`-08`)。
zzz	標準時との時差を完全な形で表示します (たとえば、`-08:00`)。

アプリケーションの現在のカルチャ情報については、「Visual Basic においてカルチャが文字列に与える影響」を参照してください。

使用例

`December 7, 1958, 8:50 PM, 35 seconds` を使ったユーザー定義の日付/時刻書式の例を次に示します。

形式	表示
`M/d/yy`	`12/7/58`
`d-MMM`	`7-Dec`
`d-MMMM-yy`	`7-December-58`
`d MMMM`	`7 December`
`MMMM yy`	`December 58`

hh:mm tt	08:50 PM
h:mm:ss t	8:50:35 P
H:mm	20:50
H:mm:ss	20:50:35
M/d/yyyy H:mm	12/7/1958 20:50

スマート デバイス開発者のためのメモ

デバイスの最小時間精度は、デバイスのメーカーによって異なります。時間の分解能が低いデバイスで使用した場合、形式指定文字 **f** は 0 を返します。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
変換の概要
Format 関数
定義済みの日付/時刻書式 (Format 関数)
ユーザー定義の数値書式 (Format 関数)

ユーザー定義の数値書式 (Format 関数)

ユーザー定義の数値書式を作成するときに使用できる文字を次の表に示します。これらは **Format** 関数の *Style* 引数の構築に使用できます。

文字	説明
なし	書式指定なしで数値を表示します。
(**0**)	桁のプレースホルダです。数字またはゼロを表示します。式が、書式指定文字列でゼロが指定されている場所を使っている場合は、該当する数字がその桁に表示されます。それ以外の場合は、その場所にゼロを表示します。 数値の整数部または小数部の桁数が、書式指定式内の 0 の桁数に満たない場合は、その桁位置には 0 が付加されます。数値の小数部の桁数が、書式指定式の小数部で指定されているゼロの数より多い場合は、ゼロの数と同じ桁数に数値が丸められます。数値の整数部の桁数が、書式指定式の整数部のゼロの数より多い場合は、桁をそのまま表示します。
(**#**)	桁のプレースホルダです。数字を表示するか、何も表示しません。書式指定文字列で **#** が指定されている場所に該当する桁が、式の結果にある場合は、該当する数字が表示されます。それ以外の場合は、その場所には何も表示されません。 この記号は **0** 桁プレースホルダと同様の機能を持ちます。ただし、数値の桁数が、書式指定式の整数部および小数部にある **#** 記号よりも少ない場合でも、先頭および末尾にゼロが表示されません。
(**.**)	小数点のプレースホルダです。小数点のプレースホルダにより、整数部および小数部に表示する桁数が決まります。書式指定式でこの記号の左に **#** 記号だけがある場合、1 未満の数値の先頭は小数点区切り記号になります。小数値の先頭にゼロを表示するには、整数部の最初の桁プレースホルダとしてゼロを使用します。ロケールによっては、小数点の区切り記号としてコンマが使用されます。書式指定結果で小数点のプレースホルダとして実際に使用される記号は、システムで認識される数値書式によって異なります。したがって、小数点のプレースホルダとしてコンマを使用するロケールであっても、書式指定の中ではピリオドを小数点プレースホルダとして使用する必要があります。書式指定された文字列は、ロケールに対して適切な書式で表現されます。
(**%**)	パーセントのプレースホルダです。式を 100 倍します。書式指定文字列の表示位置にパーセント記号 (**%**) が挿入されます。
(**,**)	3 桁ごとの区切り記号です。3 桁ごとの区切り記号は、整数部に 4 桁以上ある数値の百の位と千の位を区切ります。書式にある 3 桁ごとの桁区切り記号が桁のプレースホルダ (**0** または **#**) で囲まれている場合は、3 桁とごの桁区切り記号の標準使用が指定されます。 小数部の指定の有無にかかわらず、小数点のすぐ左に 3 桁ごとの区切り記号がある場合、または文字列の一番右に桁区切り記号がある場合は、"数値を 1,000 で割って、必要に応じて丸める" ことを意味します。500 以上で 1,000 未満の数値は `1` として表示され、500 未満の数値は `0` として表示されます。同じ位置で 3 桁ごとの区切り記号が 2 つ隣接する場合は、100 万で割って変換します。このように、桁区切り記号が増えるごとに 1,000 倍の係数で変換されます。 小数点のすぐ左、または文字列の一番右以外の位置に複数の区切り記号がある場合は、通常の 3 桁ごとの区切り記号を指定したものとして単純に処理されます。ロケールによっては、3 桁ごとの区切り記号としてピリオドが使用されます。書式指定された出力で、実際に 3 桁ごとの区切り記号として使用される記号は、システムで認識される数値書式によって異なります。したがって、桁区切り記号としてピリオドを使用するロケールであっても、書式指定の中ではコンマを 3 桁の桁区切り記号として使用する必要があります。書式指定された文字列は、ロケールに対して適切な書式で表現されます。 たとえば、次の 3 つの書式指定文字列があるとします。 ● `"#,0."` とした場合、1 億という数値は、桁区切り記号を用いて "100,000,000" という文字列形式で表記されます。 ● `"#0,."` とした場合、1 億という数値は、1000 で割って位取りした数値、つまり、"100000" という文字列になります。 ● `"#,0,."` とした場合、1 億という数値は、1000 で割って位取りした結果が、桁区切り記号を用いて "100,000" という文字列形式で表記されます。
(**:**)	時刻の区切り記号。ロケールによっては、時刻の区切り記号を表すのに別の記号が使用されます。時刻の区切り記号は、時刻値を表すときに、時、分、および秒を区切ります。書式指定された出力で、実際に時刻の区切り記号として使用される記号は、システム設定によって異なります。

(**/**)	日付の区切り記号。ロケールによっては、日付の区切り記号を表すのに別の記号が使用されます。日付の区切り記号は、日付値を表すときに、年月日を区切ります。書式指定された出力で、実際に日付の区切り記号として使用される記号は、システム設定によって異なります。
(**E-** **E+** **e-** **e+**)	指数形式。書式指定式で、**E-**、**E+**、**e-**、または **e+** の左に少なくとも 1 つの桁プレースホルダー (**0** または **#**) がある場合、数値は、数値と指数部の間に `E` または `e` を挿入して指数形式で表示されます。左にある桁プレースホルダの数で、指数の桁数が決まります。負の指数の左にマイナス記号を挿入するには、**E-** または **e-** を使用します。負の指数の左にマイナス記号を入れて、正の指数の左にプラス記号を入れるには、**E+** または **e+** を使用します。書式が適切に設定されるように、この記号の右には桁プレースホルダも入れておく必要があります。
- **+** **$** ( )	リテラル文字です。これらの文字は書式指定文字列に入力したとおりに表示されます。一覧にない文字を表示するには、その文字の前に円記号 (\) を付けるか、または二重引用符記号 (" ") で囲みます。
(****)	書式指定文字列内の次の文字を表示します。特殊な意味を持つ文字をリテラル文字として表示するには、その文字の前に円記号 (\) を付けます。円記号自体は表示されません。円記号を使用するのは、その文字を二重引用符記号で囲むことと同じです。円記号を表示するには、円記号を 2 つ (\\) 使用します。 リテラル文字として表示できない文字には次のような文字があります。日付書式文字および時刻書式文字 (**a**、**c**、**d**、**h**、**m**、**n**、**p**、**q**、**s**、**t**、**w**、**y**、**/**、および **:**)、数値書式文字 (**#**、**0**、**%**、**E**、**e**、コンマ、およびピリオド)、および文字列書式文字 (**@**、**&**、**<**、**>**、および **!**)。
("*ABC*")	二重引用符 (") で囲まれた文字列を表示します。コードからスタイル引数に文字列を含めるには、**Chr(34)** を使ってテキストを囲みます (**34** は引用符 (") を表す文字コードです)。

使用例

数値の書式指定式の例を次に示します。以下の例は、システムのロケール設定が English-U.S. であることを前提としています。最初の列は、**Format** 関数の *Style* 引数に対応する書式指定文字列です。その他の列は、書式指定されたデータに列見出しの値が設定されている場合の結果出力です。

書式 (*Style*)	**"5" の出力結果**	**"-5" の出力結果**	**"0.5" の出力結果**
`Zero-length string ("")`	`5`	`-5`	`0.5`
`0`	`5`	`-5`	`1`
`0.00`	`5.00`	`-5.00`	`0.50`
`#,##0`	`5`	`-5`	`1`
`$#,##0;($#,##0)`	`$5`	`($5)`	`$1`
`$#,##0.00;($#,##0.00)`	`$5.00`	`($5.00)`	`$0.50`
`0%`	`500%`	`-500%`	`50%`
`0.00%`	`500.00%`	`-500.00%`	`50.00%`
`0.00E+00`	`5.00E+00`	`-5.00E+00`	`5.00E-01`
`0.00E-00`	`5.00E00`	`-5.00E00`	`5.00E-01`

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
変換の概要
数値ごとに異なる書式指定 (Format 関数)
Format 関数
定義済み数値書式 (Format 関数)
ユーザー定義の日付/時刻書式 (Format 関数)

FormatCurrency 関数 (Visual Basic)

システムの [コントロール パネル] で定義されている通貨記号を使って通貨形式の文字列に書式設定して返す文字列処理関数です。

```
Function FormatCurrency(
   ByVal Expression As Object,
   Optional ByVal NumDigitsAfterDecimal As Integer = -1,
   Optional ByVal IncludeLeadingDigit As TriState = TriState.UseDefault,
   Optional ByVal UseParensForNegativeNumbers As TriState = TriState.UseDefault,
   Optional ByVal GroupDigits As TriState = TriState.UseDefault
) As String
```

パラメータ

Expression

必ず指定します。書式を変換する式を指定します。

NumDigitsAfterDecimal

省略可能です。小数点以下に表示する桁数を表す数値を指定します。既定値は -1 で、コンピュータの地域設定を使用します。

IncludeLeadingDigit

省略可能です。小数点の左側のゼロを表示するかどうかを表す列挙型を指定します。値については、「設定」を参照してください。

UseParensForNegativeNumbers

省略可能です。負の値をかっこで囲むかどうかを表す列挙型を指定します。値については、「設定」を参照してください。

GroupDigits

省略可能です。[地域のオプション] で指定されている桁区切り記号を使用して、数値を区切るかどうかを表す列挙型を指定します。値については、「設定」を参照してください。

設定

IncludeLeadingDigit、*UseParensForNegativeNumbers*、および *GroupDigits* の各引数の列挙値は次のとおりです。

値	説明
TriState.True	True
TriState.False	False
TriState.UseDefault	コンピュータの地域設定

例外

例外の種類	エラー番号	条件
ArgumentException	5	小数点以下の桁数が 99 を超えています。
InvalidCastException	13	型が数値型ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

オプションの引数を省略すると、コンピュータの地域設定と一致する値が代わりに使用されます。

通貨記号の位置は、[地域のオプション] の設定値で決まります。

メモ：

これらの設定値の情報は、アプリケーションのロケールに基づいています。既定では、コントロール パネルのロケール設定になります。これは .NET Framework を使ってプログラムで変更可能です。ただし、ゼロの表示については、**Number** タブに基づいています。

使用例

FormatCurrency 関数の使用例を次に示します。

VB

```
Dim TestDebt As Double = -4456.43
Dim TestString As String
' Returns "($4,456.43)".
TestString = FormatCurrency(TestDebt, , , TriState.True, TriState.True)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
FormatDateTime 関数 (Visual Basic)
FormatNumber 関数 (Visual Basic)
FormatPercent 関数 (Visual Basic)
TriState 列挙型
ArgumentException
InvalidCastException

FormatDateTime 関数 (Visual Basic)

日時の値を表す文字列式を返します。

```
Function FormatDateTime(
   ByVal Expression As DateTime,
   Optional ByVal NamedFormat As DateFormat = DateFormat.GeneralDate
) As String
```

パラメータ

Expression

必ず指定します。書式を変換する日付 (**Date**) を指定します。

NamedFormat

省略可能です。使用されている日付または時刻形式を表す数値を指定します。この引数を省略すると、**DateFormat.GeneralDate** が指定されます。

設定

引数 *NamedFormat* の設定値は次のとおりです。

定数	説明
DateFormat.GeneralDate	日付か時刻、または両方を表示します。日付部を短い形式で表示します。時刻部がある場合は、時刻を長い形式で表示します。両方ある場合は、両方とも表示します。
DateFormat.LongDate	[地域のオプション] で指定されている長い形式で日付を表示します。
DateFormat.ShortDate	[地域のオプション] で指定されている短い形式で日付を表示します。
DateFormat.LongTime	[地域のオプション] で指定されている形式で時刻を表示します。
DateFormat.ShortTime	24 時間形式 (hh:mm) で時刻を表示します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*NamedFormat* の設定が有効ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

日付データ型 (**Date**) には、常に日付と時刻の両方の情報が含まれます。型変換のために、Visual Basic は 1/1/1 (西暦 1 年 1 月 1 日) を日付の基準値と見なし、00:00:00 (午前 0 時) を時刻の基準値と見なします。日付型 (**Date**) の値を日時の文字列として書式設定する場合、**FormatDateTime** は結果の文字列に基準値を含めません。たとえば、#1/1/0001 9:30:00# を文字列に変換すると、結果は "9:30:00 AM" となり、日付情報が省略されます。ただし、元の日付型 (**Date**) の値には日付情報が残っており、**DatePart** などの関数を使って復元できます。

メモ :

引数 *Expression* を **String** リテラルとして渡す場合、**FormatDateTime** はアプリケーションの CurrentCulture 設定に基づいて、この引数を解釈します。ただし、引数を **Date** リテラルとして渡す場合は、**FormatDateTime** が必ず English (US) カルチャに基づいて **Date** リテラルを解釈するため、#mm/dd/yyyy# の形式を使用してください。1 つのカルチャの **Date** リテラルを使用して開発およびコーディングされたアプリケーションを別のカルチャのプラットフォームで実行した場合には、**Date** リテラルが正しく解析されないことがあるため、この形式にする必要があります。

使用例

FormatDateTime 関数の使用例を次に示します。

VB

```
' English (US) format.
Dim TestDate As DateTime = #3/12/1999#

' FormatDateTime returns "Friday, March 12, 1999".
' The time information is neutral (00:00:00) and therefore suppressed.
Dim TestString As String = FormatDateTime(TestDate, DateFormat.LongDate)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FormatCurrency 関数 (Visual Basic)
FormatNumber 関数 (Visual Basic)
FormatPercent 関数 (Visual Basic)
DatePart 関数 (Visual Basic)
文字列操作の概要
ArgumentException

FormatNumber 関数 (Visual Basic)

数値形式の文字列に書式設定して返す文字列処理関数です。

```
Function FormatNumber(
   ByVal Expression As Object,
   Optional ByVal NumDigitsAfterDecimal As Integer = -1,
   Optional ByVal IncludeLeadingDigit As TriState = TriState.UseDefault,
   Optional ByVal UseParensForNegativeNumbers As TriState = TriState.UseDefault,
   Optional ByVal GroupDigits As TriState = TriState.UseDefault
) As String
```

パラメータ

Expression

必ず指定します。書式を変換する式を指定します。

NumDigitsAfterDecimal

省略可能です。小数点以下に表示する桁数を表す数値を指定します。既定値は -1 で、[地域のオプション] の値を使用します。

IncludeLeadingDigit

省略可能です。小数点の左側の 0 を表示するかどうかを表す定数を指定します。値については、「設定」を参照してください。

UseParensForNegativeNumbers

省略可能です。負の値をかっこで囲むかどうかを表す定数を指定します。値については、「設定」を参照してください。

GroupDigits

省略可能です。ロケール設定で指定されている桁区切り記号を使用して、数値を区切るかどうかを表す定数を指定します。値については、「設定」を参照してください。

設定

IncludeLeadingDigit、*UseParensForNegativeNumbers*、および *GroupDigits* の各引数の設定値は次のとおりです。

定数	説明
TriState.True	True
TriState.False	False
TriState.UseDefault	コンピュータの地域設定

例外

例外の種類	エラー番号	条件
InvalidCastException	13	型が数値型ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

引数を省略すると、ロケール設定の対応する値を使用します。

メモ：
これらの設定値の情報は、アプリケーションのロケールに基づいています。既定では、コントロール パネルのロケール設定になります。ただし、.NET Framework を使ってプログラムで変更可能です。

使用例

FormatNumber 関数の使用例を次に示します。

VB

```
Dim TestNumber As Integer = 45600
' Returns "45,600.00".
Dim TestString As String = FormatNumber(TestNumber, 2, , , TriState.True)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
FormatCurrency 関数 (Visual Basic)
FormatDateTime 関数 (Visual Basic)
FormatPercent 関数 (Visual Basic)
TriState 列挙型
InvalidCastException

FormatPercent 関数 (Visual Basic)

パーセント記号 (%) が付加されたパーセント形式 (100 で乗算した) の文字列に書式設定して返す文字列処理関数です。

```
Function FormatPercent(
   ByVal Expression As Object,
   Optional ByVal NumDigitsAfterDecimal As Integer = -1,
   Optional ByVal IncludeLeadingDigit As TriState = TriState.UseDefault,
   Optional ByVal UseParensForNegativeNumbers As TriState = TriState.UseDefault,
   Optional ByVal GroupDigits As TriState = TriState.UseDefault
) As String
```

パラメータ

Expression

必ず指定します。書式を変換する式を指定します。

NumDigitsAfterDecimal

省略可能です。小数点以下に表示する桁数を表す数値を指定します。既定値は -1 で、ロケール設定の値を使用します。

IncludeLeadingDigit

省略可能です。小数点の左側のゼロを表示するかどうかを表す **TriState** 定数を指定します。値については、「設定」を参照してください。

UseParensForNegativeNumbers

省略可能です。負の値をかっこで囲むかどうかを表す **TriState** 定数を指定します。値については、「設定」を参照してください。

GroupDigits

省略可能です。ロケール設定で指定されている桁区切り記号を使用して、数値を区切るかどうかを表す **TriState** 定数を指定します。値については、「設定」を参照してください。

設定

IncludeLeadingDigit、*UseParensForNegativeNumbers*、および *GroupDigits* の各引数の設定値は次のとおりです。

定数	説明
TriState.True	True
TriState.False	False
TriState.Default	コンピュータの地域設定

例外

例外の種類	エラー番号	条件
InvalidCastException	13	型が数値型ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

引数を省略すると、ロケール設定の対応する値を使用します。

メモ：
これらの設定値の情報は、アプリケーションのロケールに基づいています。既定では、コントロール パネルのロケール設定になります。ただし、.NET Framework を使ってプログラムで変更可能です。

使用例

FormatPercent 関数の使用例を次に示します。

VB

```
Dim TestNumber As Single = 0.76
' Returns "76.00%".
Dim TestString As String = FormatPercent(TestNumber)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
FormatCurrency 関数 (Visual Basic)
FormatDateTime 関数 (Visual Basic)
FormatNumber 関数 (Visual Basic)
TriState 列挙型
InvalidCastException

FreeFile 関数

FileOpen 関数で使用できる次のファイル番号を表す整数型 (**Integer**) の値を返します。

```
Public Function FreeFile() As Integer
```

例外

例外の種類	エラー番号	条件
IOException	67	使用中のファイルが 255 個を超えています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

FreeFile 関数は、まだ使用されていないファイル番号を取得するために使用します。

使用例

FreeFile 関数を使って、次に使用できるファイル番号を返す例を次に示します。この例では、ループ内で 5 つのファイルが出力モードで開かれており、各ファイルにはサンプル データが書き込まれています。

VB

```
Dim count As Integer
Dim fileNumber As Integer
For count = 1 To 5
   fileNumber = FreeFile()
   FileOpen(fileNumber, "TEST" & count & ".TXT", OpenMode.Output)
   PrintLine(fileNumber, "This is a sample.")
   FileClose(fileNumber)
Next
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileOpen 関数
IOException Class

その他の技術情報

Visual Basic でのファイルへの書き込み

FV 関数

倍精度浮動小数点数型 (Double) の値を返します。定額の支払いを定期的に行い、利率が一定であると仮定して、投資の将来価値を返します。

```
Function FV( _
   ByVal Rate As Double, _
   ByVal NPer As Double, _
   ByVal Pmt As Double, _
   Optional ByVal PV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

パラメータ

Rate

投資期間を通じて一定の利率を示す倍精度浮動小数点数型 (**Double**) の値を指定します。たとえば、年利 (APR) 10% の車のローンを月払いで返済する場合、各期間の利率は 0.1/12 または 0.0083 になります。

NPer

必ず指定します。投資期間全体での支払い回数の合計を示す倍精度浮動小数点数型 (**Double**) の値。たとえば、4 年の車のローンを月払いで返済する場合、支払い回数は 48 回 (12 回×4 年) になります。

Pmt

必ず指定します。毎回の支払額を示す倍精度浮動小数点数型 (**Double**) の値を指定します。通常、支払額には元金と利息が含まれます。支払額を投資期間内に変更することはできません。

PV

省略可能です。現在の投資額、つまり将来行われる一連の支払いを一括支払いした場合の合計金額を示す倍精度浮動小数点数型 (**Double**) の値を指定します。たとえば、車を購入するために資金を借りる場合、現在価値は毎月支払うローンの総額になります。このパラメータを省略すると、0 を指定したものと見なされます。

Due

省略可能です。支払期日を示すオブジェクト型 DueDate 列挙型 の値を指定します。各期の期末に支払う場合は **DueDate.EndOfPeriod** を、各期の期首に支払う場合は **DueDate.BegOfPeriod** をそれぞれ引数に指定します。この引数を省略すると、**DueDate.EndOfPeriod** を指定したものと見なされます。

解説

投資とは、一連の定額の支払いを長期間行うことです。たとえば、住宅ローンなどのローンまたは毎月の貯蓄プランなどの出資を指します。

引数 *Rate* および *NPer* は、単位が同じ支払い期日を使用して計算する必要があります。たとえば、*Rate* を月単位で計算する場合は、*NPer* も月単位で計算する必要があります。

出金 (定額預金の支払いなど) を表す引数には負の値を指定し、入金 (配当金など) を表す引数には正の値を指定します。

使用例

次の例は、将来の貯蓄総額を **FV** 関数で計算します。年利 (`APR / 12`)、貯蓄回数 (`TotPmts`)、毎月の貯蓄額 (`Payment`)、現在の貯蓄額 (`PVal`)、貯蓄期日 (`PayType`) を指定します。`Payment` は出金を表すので、負の数を指定しています。

VB

```
Sub TestFV()
    Dim TotPmts As Integer
    Dim Payment, APR, PVal, Fval As Double
    Dim PayType As DueDate
    Dim Response As MsgBoxResult

    ' Define money format.
    Dim Fmt As String = "###,###,##0.00"
    Payment = CDbl(InputBox("How much do you plan to save each month?"))
    APR = CDbl(InputBox("Enter the expected interest annual percentage rate."))
```

```
    ' Ensure proper form.
    If APR > 1 Then APR = APR / 100
    TotPmts = CInt(InputBox("For how many months do you expect to save?"))
    Response = MsgBox("Do you make payments at the end of month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    PVal = CDbl(InputBox("How much is in this savings account now?"))
    Fval = FV(APR / 12, TotPmts, -Payment, -PVal, PayType)
    MsgBox("Your savings will be worth " & Format(Fval, Fmt) & ".")
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
財務処理の概要

GetAllSettings 関数

Windows レジストリのアプリケーションの初期設定ファイルから、キー設定とその設定値 (**SaveSetting** により作成) の一覧を返します。

My 機能を使用すると、**GetAllSettings** よりも、レジストリ操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

```
Public Function GetAllSettings( _
   ByVal AppName As String, _
   ByVal Section As String _
) As String(,)
```

パラメータ

AppName

必ず指定します。キー設定を取得するアプリケーション名またはプロジェクト名を含む文字列型 (**String**) の式を指定します。

Section

必ず指定します。キー設定を要求されているセクションの名前を含む **String** 式です。**GetAllSettings** は文字列の 2 次元配列を含むオブジェクトを返します。文字列には、指定したセクションのすべてのキー設定と、その設定に対応する値が含まれます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	ユーザーがログインしていません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

AppName または *Section* が存在しない場合、**GetAllSettings** は初期化されていない **Object** を返します。

GetAllSettings は、ユーザーが対話形式でログオンするまでアクティブにならない **HKEY_LOCAL_USER** レジストリ キーのもとで動作するため、ユーザーがログオンしていることが必要条件です。

双方向でないプロセス (Mtx.exe など) からアクセスするレジストリの設定は、**HKEY_LOCAL_MACHINE\Software** または **HKEY_USER\DEFAULT\Software** レジストリ キーのいずれかに格納されます。

使用例

次の例は、最初に **SaveSetting** 関数を使用して、*AppName* として指定したアプリケーションのエントリを Windows のレジストリに作成します。次に、**GetAllSettings** 関数を使用して、設定を表示します。**GetAllSettings** 関数では、アプリケーション名および *Section* 名は取得できません。最後に、**DeleteSetting** 関数でアプリケーションのエントリを削除します。

VB

```
' Object to hold 2-dimensional array returned by GetAllSettings.
' Integer to hold counter.
Dim MySettings(,) As String
Dim intSettings As Integer
' Place some settings in the registry.
SaveSetting("MyApp", "Startup", "Top", "75")
SaveSetting("MyApp", "Startup", "Left", "50")
' Retrieve the settings.
MySettings = GetAllSettings("MyApp", "Startup")
For intSettings = LBound(MySettings, 1) To UBound(MySettings, 1)
   WriteLine(1, MySettings(intSettings, 0))
   WriteLine(1, MySettings(intSettings, 1))
Next intSettings
DeleteSetting("MyApp")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DeleteSetting 関数
GetSetting 関数
SaveSetting 関数
ArgumentException Class

概念

一般的なレジストリ タスク

GetAttr 関数

ファイル、ディレクトリ、またはフォルダの属性を表す **FileAttribute** 型の値を返します。

My 機能を使用すると、**FileAttribute** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Function GetAttr(ByVal PathName As String) As FileAttribute
```

パラメータ

PathName

必ず指定します。ファイル名、ディレクトリ名、フォルダ名を指定する文字列 (**String**) 式です。*PathName* にはディレクトリ名またはフォルダ名、およびドライブ名も含めて指定できます。

戻り値

GetAttr によって返される値は、次の表に示す列挙型値の合計です。

値	定数	説明
Normal	**vbNormal**	通常。
ReadOnly	**vbReadOnly**	読み取り専用です。
Hidden	**vbHidden**	隠しファイル。
System	**vbSystem**	システム ファイル。
Directory	**vbDirectory**	ディレクトリまたはフォルダ。
Archive	**vbArchive**	前回のバックアップ以降に変更されているファイル。
Alias	**vbAlias**	他の名前が付いているファイル。

メモ：

ここに示した列挙型値は、Visual Basic 言語で設定されています。これらの名前は、実際の値の代わりにコード内のどの部分でも使用できます。

例外

例外の種類	エラー番号	条件
IOException	52	*Pathname* が無効です。またはワイルドカードを含んでいます。
FileNotFoundException	53	対象のファイルが存在しません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

どの属性が設定されているかを調べるには、**And** 演算子を使って、**GetAttr** 関数によって返される値と、調べる対象となる個々のファイル属性の値をビットごとに比較します。結果が 0 以外の場合は、その属性がファイルに設定されています。たとえば、**Archive** 属性が設定されていない場合、次の **And** 式の戻り値は 0 です。

```
Result = GetAttr(FName) And vbArchive
```

Archive 属性が設定されている場合は、0 以外の値が返されます。

使用例

GetAttr 関数を使って、ファイルおよびディレクトリまたはフォルダの属性を調べるコード例を次に示します。

VB

```
Dim MyAttr As FileAttribute
' Assume file TESTFILE is normal and readonly.
MyAttr = GetAttr("C:\TESTFILE.txt")   ' Returns vbNormal.

' Test for normal.
If (MyAttr And FileAttribute.Normal) = FileAttribute.Normal Then
   MsgBox("This file is normal.")
End If

' Test for normal and readonly.
Dim normalReadonly As FileAttribute
normalReadonly = FileAttribute.Normal Or FileAttribute.ReadOnly
If (MyAttr And normalReadonly) = normalReadonly Then
   MsgBox("This file is normal and readonly.")
End If

' Assume MYDIR is a directory or folder.
MyAttr = GetAttr("C:\MYDIR")
If (MyAttr And FileAttribute.Directory) = FileAttribute.Directory Then
   MsgBox("MYDIR is a directory")
End If
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : FileSystem

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

And 演算子 (Visual Basic)
FileAttr 関数
SetAttr 関数
IOException Class
FileNotFoundException Class
FileAttribute 列挙型

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

GetChar 関数

指定された文字列内の指定されたインデックス位置にある文字を表す **Char** 型の値を返します。

```
Public Shared Function GetChar( _
   ByVal str As String, _
   ByVal Index As Integer _
) As Char
```

パラメータ

str

必ず指定します。任意の有効な文字列型 (**String**) の式を指定します。

Index

必ず指定します。整数型 (**Integer**) の式です。*str* から取り出す部分の文字を指定するインデックスです。インデックスは 1 から始まります。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*str* が **Nothing**、*Index* < 1、または *Index* が *str* の最後の文字のインデックスよりも大きい

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

Index が 1 よりも小さいか、*str* の最後の文字のインデックスよりも大きい場合、**ArgumentException** がスローされます。is thrown.

使用例

GetChar 関数を使って、文字列内で指定されたインデックス位置にある文字を取得する方法の例を次に示します。

VB

```
Dim TestString As String = "ABCDE"
Dim TestChar As Char
' Returns "D"
TestChar = GetChar(TestString, 4)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
Mid 関数 (Visual Basic)
ArgumentException

GetException 関数

発生したエラーを表す例外を返します。

```
Overridable Public Function GetException() As Exception
```

解説

GetException 関数は、**Err** オブジェクト クラスからしか使用できません。発生したエラーを表示するために、**Err** オブジェクトの **Exception** プロパティを使用します。

使用例

次のコードは、**Err** オブジェクトの例外に割り当てられているメッセージを表示します。

VB

```
On Error Resume Next
Dim myError As System.Exception
' Generate an overflow exception.
Err.Raise(6)
' Assigns the exception from the Err object to myError.
myError = Err.GetException()
' Displays the message associated with the exception.
MsgBox(myError.Message)
```

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **Err Object**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : エラー オブジェクトから情報を取得する

関連項目

Err オブジェクト (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

非構造化例外処理の概要

GetObject 関数 (Visual Basic)

COM コンポーネントによって提供されるオブジェクトへの参照を返します。

```
Public Function GetObject( _
    Optional ByVal PathName As String = Nothing, _
    Optional ByVal [Class] As String = Nothing _
) As Object
```

パラメータ

パラメータ	説明
PathName	省略可能です。取得するオブジェクトが含まれているファイル名までの完全パスを文字列 (**String**) で指定します。*PathName* を省略するか、長さ 0 の文字列 ("") を指定した場合、*Class* は省略できません。
Class	*PathName* が省略されている場合は、必ず指定します。オブジェクトのクラスを表す文字列 (**String**) を指定します。*Class* の構文と指定項目は次のとおりです。 *appname* . *objecttype* **パラメータ** / **説明** *appname* — 必ず指定します。オブジェクトを提供するアプリケーションの名前を文字列 (**String**) で指定します。 *objecttype* — 必ず指定します。作成するオブジェクトの型またはクラスを文字列 (**String**) で指定します。

例外

例外の種類	エラー番号	条件
Exception	429	指定されたクラス型のオブジェクトが存在しません。
FileNotFoundException	432	指定されたパスおよびファイル名を持つオブジェクトが存在しません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

ファイルから COM コンポーネントのインスタンスを読み込むには、**GetObject** 関数を使用します。次に例を示します。

```
Dim CADObject As Object
CADObject = GetObject("C:\CAD\schema.cad")
```

このコード例が実行されると、指定された引数 *PathName* に対応するアプリケーションが起動し、指定されたファイルのオブジェクトがアクティブになります。

既定の動作

引数 *PathName* が長さ 0 の文字列 ("") の場合、**GetObject** 関数は、指定されたクラス型の新しいオブジェクト インスタンスを返します。引数 *PathName* を省略すると、**GetObject** 関数は、*Class* に指定されたクラス型の現在アクティブなオブジェクトを返します。指定した種類のオブジェクトが存在しない場合はエラーが発生します。

サブオブジェクトの使用

アプリケーションによっては、ファイルに関連付けられたサブオブジェクトをアクティブにできる場合もあります。これには、ファイル名の最後に感嘆符 (**!**) を付け、続けてアクティブにするファイルの部分を表す文字列を指定します。このような文字列を作成する方法については、オブジェクトを作成したアプリケーションのドキュメントを参照してください。

たとえば、描画のアプリケーションでは、1 つのファイルの複数のレイヤに描画を保存する場合があります。`schema.cad` という描画ファイルの 1 つのレイヤをアクティブにするコード例を次に示します。

```
layerObject = GetObject("C:\CAD\schema.cad!Layer3")
```

クラスの指定

オブジェクトの *Class* が指定されていない場合は、指定されたファイル名を基に、オートメーションが起動するアプリケーションとアクティブにするオブジェクトを決定します。ただし、中には複数のクラスのオブジェクトをサポートしているファイルもあります。たとえば、1 つの描画で `Application` オブジェクト、`Drawing` オブジェクト、および `Toolbar` オブジェクトという 3 つの異なる型のオブジェクトをサポートし、3 つとも同一ファイルに含まれることがあります。ファイルのどのオブジェクトをアクティブにするかは、引数 *Class* で指定します。次に例を示します。

```
Dim drawObj As Object
drawObj = GetObject("C:\Drawings\sample.drw", "Figment.Drawing")
```

このコード例では、`Figment` が描画のアプリケーションの名前で、`Drawing` がそのアプリケーションによってサポートされるオブジェクトの種類です。

オブジェクトの使用

アクティブになったオブジェクトをコードで参照するには、宣言したオブジェクト変数を使います。上の例では、オブジェクト変数 `drawObj` を使って新規オブジェクトのプロパティとメソッドにアクセスします。次に例を示します。

```
drawObj.Line(9, 90)
drawObj.InsertText(9, 100, "Hello, world.")
drawObj.SaveAs("C:\Drawings\sample.drw")
```

メモ：

GetObject 関数は、オブジェクトの現在のインスタンスがある場合や、読み込んだファイルからオブジェクトを作成する場合に使います。現在のインスタンスがなく、ファイルを読み込んでもオブジェクトを開始しないときは、CreateObject 関数 (Visual Basic) 関数を使用します。

複数のインスタンスを作成できない ActiveX オブジェクトの場合は、**CreateObject** 関数を何度呼び出しても、そのオブジェクトのインスタンスは 1 つしか作成されません。単一インスタンス オブジェクトの場合、長さ 0 の文字列 ("") を指定して **GetObject** 関数を呼び出すと、常に同じインスタンスを返します。また、引数 *PathName* を省略すると、エラーになります。**GetObject** 関数では、Visual Basic で作成したクラスへの参照を取得できません。

セキュリティに関するメモ：

GetObject 関数にはアンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

オブジェクトの使用後は、そのオブジェクトのすべての参照を **Nothing** に設定します。これにより、ランタイムが COM コンポーネントを破棄できるようになります。

使用例

次の例では、Excel が実行中であるかどうかを **GetObject** 関数を使用して調べています。

VB

```
' Test to see if a copy of Excel is already running.
Private Sub testExcelRunning()
    On Error Resume Next
    ' GetObject called without the first argument returns a
    ' reference to an instance of the application. If the
    ' application is not already running, an error occurs.
    Dim excelObj As Object = GetObject(, "Excel.Application")
    If Err.Number = 0 Then
        MsgBox("Excel is running")
    Else
        MsgBox("Excel is not running")
    End If
    Err.Clear()
    excelObj = Nothing
End Sub
```

GetObject 関数を使って、Microsoft Excel の特定のワークシートへの参照 (`excelObj`) を取得するコード例を次に示します。**GetObject** を呼び出すと、指定したファイル (`test.xls`) で表されるワークシートへの参照が返されます。次に、Excel を表示し、指定されたワークシートのウィンドウを表示するコード例を示します。

この例では **Option Strict Off** の指定が必要です。その理由は、遅延バインディングを使用して、オブジェクトが **Object** 型の変数に代入されるからです。Excel タイプ ライブラリへのプロジェクト参照を追加した場合は、**Option Strict On** を指定して、特定のオブジェクト型のオブジェクトを宣言できます。プロジェクト参照を追加するには、Visual Studio で [プロジェクト] メニューの [参照の追加] ダイアログ ボックスを開き、[COM] タブで Excel タイプ ライブラリを選択します。

VB

```
' Add Option Strict Off to the top of your program.
Option Strict Off
```

VB

```
Private Sub getExcel()
    Dim fileName As String = "c:\vb\test.xls"

    If Not My.Computer.FileSystem.FileExists(fileName) Then
        MsgBox(fileName & " does not exist")
        Exit Sub
    End If

    ' Set the object variable to refer to the file you want to use.
    Dim excelObj As Object = GetObject(fileName)
    ' Show Excel through its Application property.
    excelObj.Application.Visible = True
    ' Show the window containing the file.
    Dim winCount As Integer = excelObj.Parent.Windows.Count()
    excelObj.Parent.Windows(winCount).Visible = True

    ' Insert additional code to manipulate the test.xls file here.
    ' ...

    excelObj = Nothing
End Sub
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

CreateObject 関数 (Visual Basic)
Declare ステートメント
Option Strict ステートメント
Exception
FileNotFoundException

GetSetting 関数

Windows レジストリのアプリケーションの初期設定ファイルから、キー設定値を返します。

My 機能では、**GetAllSettings** よりも、レジストリ操作の生産性とパフォーマンスが高くなっています。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

```
Public Function GetSetting( _
   ByVal AppName As String, _
   ByVal Section As String, _
   ByVal Key As String, _
   Optional ByVal Default As String = "" _
) As String
```

パラメータ

AppName

必ず指定します。キー設定を取得するアプリケーション名またはプロジェクト名を含む文字列型 (**String**) の式を指定します。

Section

必ず指定します。対象となるキー設定があるセクション名を含む文字列型 (**String**) の式を指定します。

Key

必ず指定します。返すキー設定名を含む文字列型 (**String**) の式を指定します。

Default

省略可能です。*Key* 設定に値が設定されていない場合に返す値を含む式。省略すると、*Default* は長さ 0 の文字列 ("") になります。

例外

例外の種類	エラー番号	条件
ArgumentException	5	文字列型 (**String**) の式でない引数が含まれるか、ユーザーがログインしていません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

GetSetting 関数の引数のいずれかを指定していない場合、**GetSetting** 関数は、*Default* の値を返します。

GetSetting は、ユーザーが対話形式でログオンするまでアクティブにならない **HKEY_LOCAL_USER** レジストリ キーのもとで動作するため、ユーザーがログオンしていることが必要条件です。

双方向でないプロセス (Mtx.exe など) からアクセスするレジストリの設定は、**HKEY_LOCAL_MACHINE\Software** または **HKEY_USER\DEFAULT\Software** レジストリ キーのいずれかに格納されます。

GetSetting には、**Read** レジストリ アクセス許可 が必要です。

使用例

次の例は、最初に **SaveSetting** 関数を使用して、*AppName* として指定したアプリケーションのエントリを Windows のレジストリに作成します。次に、**GetSetting** 関数を使用して、設定を表示します。引数 *Default* を指定すると、必ず値が返されます。*Section* 名は **GetSetting** では取得できないことに注意してください。最後に、**DeleteSetting** 関数でアプリケーションのすべてのエントリを削除します。

VB

```
' Place some settings in the registry.
SaveSetting("MyApp", "Startup", "Top", "75")
SaveSetting("MyApp", "Startup", "Left", "50")
Console.WriteLine(GetSetting("MyApp", "Startup", "Left", "25"))
DeleteSetting("MyApp")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DeleteSetting 関数
GetAllSettings 関数
SaveSetting 関数
ArgumentException Class
RegistryPermission Class

概念

一般的なレジストリ タスク

関数 H ～ L

次の表は、Visual Basic のランタイム メンバ関数の一覧です。

Hex	Hour	IIf	Input
InputBox	InputString	InStr	InStrRev
Int	IPmt	IRR	IsArray
IsDate	IsDBNull	IsError	IsNothing
IsNumeric	IsReference	Join	Kill
LBound	LCase	Left	Len
LineInput	Loc	Lock	LOF
LSet	LTrim		

バージョン 6.0 以前に Visual Basic のランタイム メンバ関数として利用できた数値演算関数 **Log** は、.NET Framework クラス ライブラリの Math クラスのメソッド (Log) に置き換えられました。詳細については、「数値演算関数 (Visual Basic)」を参照してください。

スマート デバイス開発者のためのメモ

Input、**InputString**、**Kill**、**LineInput**、**Loc**、**Lock**、および **LOF** は、スマート デバイス アプリケーションではサポートされません。

参照

関連項目

関数 A ～ C
関数 D ～ G
関数 M ～ R
関数 S ～ Z

その他の技術情報

Visual Basic リファレンス

Hex 関数 (Visual Basic)

指定された値を 16 進数で表した文字列で返します。

```
Public Shared Function Hex( _
   ByVal Number As { Byte | SByte | Short | UShort |
   Integer | UInteger | Long | ULong | Object } _
) As String
```

パラメータ

Number

必ず指定します。任意の有効な数式または文字列型 (**String**) の式です。

例外

例外の種類	エラー番号	条件
ArgumentNullException	5	*Number* が指定されていません。
ArgumentException	5	*Number* が数値型ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

引数 *Number* が整数でない場合は、最も近い整数に丸められてから評価されます。

***Number* の値**	**Hex の戻り値**
空	ゼロ (0)
任意の数式	16 進数を表す最大 16 桁の文字列

適切な範囲内の数字の前に **&H** を付けることで、16 進数を直接表すことができます。たとえば、`&H10` は、10 進数の 16 を 16 進数で表します。

使用例

Hex 関数を使って 16 進数の数値を返す例を次に示します。

VB

```
Dim TestHex As String
' Returns 5.
TestHex = Hex(5)
' Returns A.
TestHex = Hex(10)
' Returns 1CB.
TestHex = Hex(459)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Conversion**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 16 進文字列を数値に変換する

関連項目

Oct 関数
データ型変換関数
ArgumentNullException

Hour 関数 (Visual Basic)

時刻を表す 0 ～ 23 の整数型 (**Integer**) の値を返します。

```
Public Function Hour(ByVal TimeValue As DateTime) As Integer
```

パラメータ

TimeValue

必ず指定します。"時" を抽出する日付型 (**Date**) の値です。

解説

DatePart を呼び出し、*Interval* 引数に **DateInterval.Hour** を指定することによっても時を取得できます。

使用例

次の例は、**Hour** 関数を使って、指定された時刻の "時" の部分を取得します。開発環境では、時刻リテラルは、コード記述時のロケールで設定されている短い形式で表示されます。

VB

```
Dim someTime As Date
Dim someHour As Integer
someTime = #4:35:17 PM#
someHour = Hour(someTime)
' someHour now contains 16.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Day 関数 (Visual Basic)
Minute 関数
Now プロパティ
Second 関数 (Visual Basic)
TimeOfDay プロパティ
DatePart 関数 (Visual Basic)
System
DateTime
ArgumentException
ArgumentOutOfRangeException

IIf 関数

式の評価結果によって、2 つのオブジェクトのうち 1 つを返します。

```
Public Function IIf( _
   ByVal Expression As Boolean, _
   ByVal TruePart As Object, _
   ByVal FalsePart As Object _
) As Object
```

パラメータ

Expression

必ず指定します。評価する式をブール型 (**Boolean**) で指定します。

TruePart

必ず指定します。*Expression* が **True** に評価された場合に返すオブジェクト (**Object**) を指定します。

FalsePart

必ず指定します。*Expression* が **False** に評価された場合に返すオブジェクト (**Object**) を指定します。

解説

IIf 関数は、Visual C++ の三項の Conditional Operator: ? : と同じように利用できます。

使用例

IIf 関数を使って、プロシージャ名 `checkIt` のパラメータ `testMe` を評価するコード例は、次のとおりです。パラメータ `testMe` の値が 1,000 よりも大きい場合は "Large" という文字列を返し、1,000 以下の場合は "Small" という文字列を返します。

VB

```
Function checkIt(ByVal testMe As Integer) As String
    Return CStr(IIf(testMe > 1000, "Large", "Small"))
End Function
```

Option Strict が **On** の場合、キーワード **CStr** を使って戻り値を **Object** から **String** に明示的に変換しなければならないことに注意してください。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Option Strict ステートメント
データ型変換関数
Choose 関数
If...Then...Else ステートメント (Visual Basic)
Select...Case ステートメント (Visual Basic)
Switch 関数

Input 関数

シーケンシャル入力モードで開いたファイルからデータを読み込んで、それを変数に格納します。

```
Public Sub Input( _
   FileNumber As Integer, _
   ByRef Value As Object _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

Value

必ず指定します。ファイルから読み込んだ値を格納するための変数です。配列変数、またはオブジェクト変数を指定することはできません。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Input 関数は、下位互換性を保つために提供されており、パフォーマンスに影響を与える可能性があります。非レガシ アプリケーションに対しては、**My.Computer.FileSystem** オブジェクトを使用した方が、パフォーマンスが高くなります。詳細については、「Visual Basic におけるファイル アクセス」を参照してください。

通常、**Input** 関数を使用して読み込んだデータは **Write** 関数を使用して書き込みます。この関数は、**Input** モードまたは **Binary** モードで開いたファイルに対してだけ使用します。

セキュリティに関するメモ：

ファイルからデータを読み取るときは、ファイル名の拡張子に基づいてファイルの内容を判断しないでください。たとえば、Form1.vb という名前のファイルが Visual Basic 2005 のソース ファイルとは限りません。

ファイルからデータを読み込む場合、通常、文字列データは文字列型 (**String**)、数値データは数値データ型として格納されます。これ以外のデータを読み込んだ場合、次に示すようにデータによって変数に格納される値が異なります。

データ	変数に格納される値
コンマのみ、または空白行	Empty
#NULL#	**DBNull**
#TRUE# または #FALSE#	**True** または、次のようにも指定できます。**False**
#yyyy-mm-dd hh:mm:ss#	式によって表された日付と時刻
#ERROR *errornumber*#	*errornumber* (エラー値として格納されたオブジェクト型)

データの入力中にファイルの末尾に達した場合は、入力が終了されてエラーが発生します。

メモ：

Input 関数はローカライズされません。たとえば、ドイツ語バージョンでもコンマは小数点ではなく変数の区切り文字として扱われるため、3,14159 と入力すると 3 だけが返ります。

セキュリティに関するメモ：

Input 関数を使ってファイルを読み込むには、**FileIOPermissionAccess** 列挙値の **Read** アクセスが必要です。詳細については、「FileIOPermissionAccess 列挙体」を参照してください。

使用例

Input 関数を使ってファイルからデータを読み込み、2 つの変数に代入する例を次に示します。この例のファイル `TestFile` は、数行のデータが **Write** 関数を使って書き込まれているものと仮定します。各行は "Hello", 234 のように、文字列が二重引用符で囲まれ、数値はコンマで区切られています。

VB

```
FileOpen(1, "TESTFILE", OpenMode.Output)
Write(1, "hello")
Write(1, 14)
FileClose(1)
Dim s As String = "teststring"
Dim i As Integer
FileOpen(1, "TESTFILE", OpenMode.Input)
Input(1, s)
MsgBox(s)
Input(1, i)
MsgBox(i)
FileClose(1)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でテキストをファイルに書き込む

方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

関連項目

InputString 関数

FileOpen 関数

Print 関数、PrintLine 関数

Write 関数、WriteLine 関数

その他の技術情報

Visual Basic におけるファイル アクセス

InputBox 関数 (Visual Basic)

ダイアログ ボックスにメッセージ、テキスト ボックス、またはボタンを表示します。文字列が入力されるか、またはボタンがクリックされると、テキスト ボックスの内容を格納した文字列を返します。

```
Public Function InputBox( _
   ByVal Prompt As String, _
   Optional ByVal Title As String = "", _
   Optional ByVal DefaultResponse As String = "", _
   Optional ByVal Xpos As Integer = -1, _
   Optional ByVal YPos As Integer = -1 _
) As String
```

パラメータ

Prompt

必ず指定します。ダイアログ ボックス内にメッセージとして表示する文字列 (**String**) 式を指定します。*Prompt* に指定できる最大文字数は、1 バイト文字で約 1,024 文字です。ただし、使用する文字の文字幅に依存します。*Prompt* が複数の行から成る場合、復帰文字 (**Chr(**13**)**)、ライン フィード文字 (**Chr(**10**)**)、復帰/ライン フィードの組み合わせ (**Chr(**13**)** & **Chr(**10**)**) を行の間に使用して、行を分割できます。

Title

省略できます。ダイアログ ボックスのタイトル バーに表示する文字列 (**String**) 式を指定します。*Title* を省略すると、タイトル バーにはアプリケーション名が表示されます。

DefaultResponse

省略できます。ユーザーが何も入力しない場合に、テキスト ボックスに既定値として表示する文字列 (**String**) 式を指定します。*DefaultResponse* を省略すると、テキスト ボックスには何も表示されません。

XPos

省略可能です。画面の左端からダイアログ ボックスの左端までの距離をピクセル単位で示す数式を指定します。*XPos* と *YPos* を省略すると、ダイアログ ボックスは画面の中央に配置されます。

YPos

省略可能です。画面の上端からダイアログ ボックスの上端までの距離をピクセル単位で示す数式を指定します。*XPos* と *YPos* を省略すると、ダイアログ ボックスは画面の中央に配置されます。

解説

[キャンセル] がクリックされると、長さ 0 の文字列が返されます。

Prompt 以外の引数も指定する場合は、式の中で **InputBox** 関数を使用する必要があります。引数を省略する場合も、それに対応する位置にコンマ区切り記号は残します。

メモ :

InputBox 関数には SafeTopLevelWindows レベルの **UIPermission** が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「アクセス許可の要求」と「UIPermission クラス」を参照してください。

使用例

次の例は、**InputBox** 関数を使って、ユーザーに値を入力させます。位置 x と y の指定を省略すると、ダイアログ ボックスは、水平軸および垂直軸に対して中央に自動的に配置されます。[OK] がクリックされるか、Enter キーが押されると、変数 `MyValue` には、ユーザーの入力した値が入ります。

VB

```
Dim message, title, defaultValue As String
Dim myValue As Object
' Set prompt.
message = "Enter a value between 1 and 3"
```

```
' Set title.
title = "InputBox Demo"
defaultValue = "1"    ' Set default value.

' Display message, title, and default value.
myValue = InputBox(message, title, defaultValue)
' If user has clicked Cancel, set myValue to defaultValue
If myValue Is "" Then myValue = defaultValue

' Display dialog box at position 100, 100.
myValue = InputBox(message, title, defaultValue, 100, 100)
' If user has clicked Cancel, set myValue to defaultValue
If myValue Is "" Then myValue = defaultValue
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

MsgBox 関数 (Visual Basic)

Chr 関数、ChrW 関数

InputString 関数

Input モードまたは **Binary** モードで開いたファイルから読み取る文字を表す文字列型 (**String**) の値を返します。

My 機能を使用すると、**InputString** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
InputString(_
   ByVal FileNumber As Integer, _
   ByVal CharCount As Integer _
) As String
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

CharCount

必ず指定します。ファイルから読み取る文字数を指定する有効な数式。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在していません。
ArgumentException	5	*CharCount* が 0 より小さいか、または 214 より大きい値です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

InputString 関数は、下位互換性を保つために提供されており、パフォーマンスに影響を与える可能性があります。非レガシ アプリケーションに対しては、**My.Computer.FileSystem** オブジェクトを使用した方が、パフォーマンスが高くなります。詳細については、「Visual Basic におけるファイル アクセス」を参照してください。

通常、**InputString** 関数を使用して読み込んだデータは **Print** 関数または **FilePut** 関数を使用してファイルに書き込みます。この関数は、**Input** モードまたは **Binary** モードで開いたファイルに対してだけ使用します。

Input 関数と異なり、**InputString** 関数は、コンマ、キャリッジ リターン、ライン フィード、二重引用符、および先行空白を含む、読み込んだすべての文字を返します。

Binary モードでファイルを開いた場合、**InputString** 関数を使用して **EOF** 関数が **True** を返すまでファイルを読み込もうとすると、エラーが発生します。**InputString** 関数を使用してバイナリ ファイルを読み込む場合は、**EOF** 関数の代わりに **LOF** 関数および **Loc** 関数を使用します。**EOF** 関数を使用する場合は、**FileGet** を使用してください。

セキュリティに関するメモ：
ファイルからデータを読み取るときは、ファイル名の拡張子に基づいてファイルの内容を判断しないでください。たとえば、Form1.vb という名前のファイルが Visual Basic のソース ファイルとは限りません。

使用例

InputString 関数を使って、ファイルから 1 文字ずつ読み込み、読み込んだ文字を **Output** ウィンドウに表示する例を次に示します。`MyFile` は、複数行のサンプル データを含むテキスト ファイルと仮定します。

VB

```
Dim oneChar As String
' Open file.
FileOpen(1,  "MYFILE.TXT", OpenMode.Input)
' Loop until end of file.
While Not EOF(1)
```

```
    ' Get one character.
    oneChar = (InputString(1, 1))
    ' Print to the output window.
    System.Console.Out.WriteLine(oneChar)
    End While
    FileClose(1)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

方法 : Visual Basic でテキストをファイルに書き込む

関連項目

Input 関数

その他の技術情報

Visual Basic におけるファイル アクセス

InStr 関数 (Visual Basic)

ある文字列の中から指定した文字列を検索し、最初に見つかった文字列の開始位置を示す整数型 (**Integer**) の値を返します。

```
Public Shared Function InStr(_
   ByVal String1 As String, _
   ByVal String2 As String, _
   Optional ByVal Compare As CompareMethod _
) As Integer
' -or-
Public Shared Function InStr(_
   ByVal Start As Integer, _
   ByVal String1 As String, _
   ByVal String2 As String, _
   Optional ByVal Compare As Microsoft.VisualBasic.CompareMethod _
) As Integer
```

パラメータ

Start

省略可能です。各検索の開始位置を設定する数式を指定します。引数 start を省略すると、先頭の文字から検索されます。開始位置のインデックスの値は、1 から始まります。

String1

必ず指定します。検索する文字列型 (**String**) の式を指定します。

String2

必ず指定します。検索される文字列型 (**String**) の式を指定します。

Compare

省略可能です。文字列比較のタイプを指定します。*Compare* を省略すると、**Option Compare** 設定により比較のタイプが決定されます。

設定

引数 *Compare* には、次の値を指定します。

定数	値	説明
Binary	0	バイナリ モードで比較を行います。
Text	1	テキスト モードで比較を行います。

戻り値

条件	InStr の戻り値
String1 の長さが 0、または **Nothing**	0
String2 の長さが 0、または **Nothing**	*start*
String2 が見つからない。	0
String2 が内部にある *String1*	一致する文字列の開始位置
Start > *String2*	0

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Start* < 1.

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

多くの場合、**InStr** 関数は文字列を解析するときに使用します。

メモ :
以前のバージョンの Visual Basic では、**InStrB** 関数は文字の位置ではなくバイト数を返していました。これは主に、2 バイト文字セット (DBCS) アプリケーションで文字列を変換するために使用します。Visual Basic 2005 のすべての文字列は Unicode で、**InStrB** は現在サポートされていません。

使用例

InStr 関数を使って、ある文字列の中から指定された文字列を検索し、最初に見つかった文字位置を返す例を次に示します。

VB

```
' String to search in.
Dim SearchString As String = "XXpXXpXXPXXP"
' Search for "P".
Dim SearchChar As String = "P"

Dim TestPos As Integer
' A textual comparison starting at position 4. Returns 6.
TestPos = InStr(4, SearchString, SearchChar, CompareMethod.Text)

' A binary comparison starting at position 1. Returns 9.
TestPos = InStr(1, SearchString, SearchChar, CompareMethod.Binary)

' If Option Compare is not set, or set to Binary, return 9.
' If Option Compare is set to Text, returns 3.
TestPos = InStr(SearchString, SearchChar)

' Returns 0.
TestPos = InStr(1, SearchString, "W")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

InStrRev 関数 (Visual Basic)
Option Compare ステートメント
StrComp 関数 (Visual Basic)
ArgumentException

概念

プログラミング要素のサポートに関する変更の概要

その他の技術情報

Visual Basic における文字列
Visual Basic の文字列の概要

InStrRev 関数 (Visual Basic)

ある文字列 (*StringCheck*) の中から指定された文字列 (*StringMatch*) を最後の文字位置から検索を開始し、最初に見つかった文字位置 (先頭からその位置までの文字数) を返します。

```
Public Function InStrRev(
   ByVal StringCheck As String,
   ByVal StringMatch As String,
   Optional ByVal Start As Integer = -1,
   Optional ByVal Compare As CompareMethod = CompareMethod.Binary
) As Integer
```

パラメータ

StringCheck

必ず指定します。検索する文字列式を指定します。

StringMatch

必ず指定します。検索する文字列式を指定します。

Start

省略可能です。文字列の先頭を 1 として検索開始位置を設定する数式を指定します。*Start* を省略すると -1 が使用され、最後の文字位置から検索を開始します。検索は右から左へと行われます。

Compare

省略可能です。部分文字列を評価するときに使用する文字列比較のモードを表す数値を指定します。compare を省略すると、バイナリモードで比較が行われます。値については、「設定」を参照してください。

設定

引数 *Compare* の設定値は次のとおりです。

定数	説明
Binary	バイナリ モードで比較を行います。
Text	テキスト モードで比較を行います。

戻り値

InStrRev 関数の戻り値は次のとおりです。

条件	InStrRev の戻り値
StringCheck が長さ 0 の文字列 ("") のとき	0
StringMatch が長さ 0 の文字列 ("") のとき	*Start*
StringMatch が見つからないとき	0
StringMatch が内部にある *StringCheck*	文字列の先頭から検索して最初に文字列が見つかった位置
Start が *StringMatch* の長さよりも大きいとき	0

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Start* = 0 または *Start* < -1 であるとき

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は

Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

InStrRev 関数の構文は、**InStr** 関数の構文とは異なります。

使用例

InStrRev 関数の使用例を次に示します。

VB

```
Dim TestString As String = "the quick brown fox jumps over the lazy dog"
Dim TestNumber As Integer
' Returns 32.
TestNumber = InStrRev(TestString, "the")
' Returns 1.
TestNumber = InStrRev(TestString, "the", 16)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

InStr 関数 (Visual Basic)

その他の技術情報

Visual Basic における文字列

Visual Basic の文字列の概要

Int 関数、Fix 関数 (Visual Basic)

指定した数値の整数部分を返します。

```
Public Shared Function Int( _
    ByVal Number As { Double | Integer | Long | Object | Short | Single | Decimal }) _
    As { Double | Integer | Long | Object | Short | Single | Decimal }
Public Shared Function Fix( _
    ByVal Number As { Double | Integer | Long | Object | Short | Single | Decimal }) _
    As { Double | Integer | Long | Object | Short | Single | Decimal }
```

パラメータ

Number

必ず指定します。倍精度浮動小数点数型 (**Double**) の数値または任意の数式を指定します。*Number* に **Nothing** が含まれている場合は、**Nothing** を返します。

例外

例外の種類	エラー番号	条件
ArgumentNullException	5	Number が指定されていません。
ArgumentException	5	Number が数値型ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Int 関数と **Fix** 関数は、どちらも *Number* の小数部分を取り除いた整数値を返します。

Int と **Fix** の違いは、*Number* に負の値を指定した場合に現れます。**Int** 関数が *Number* 以下の最大の負の整数を返すのに対して、**Fix** 関数は *Number* 以上の最小の負の整数を返します。たとえば、-8.4 を指定した場合、**Int** 関数は -9、**Fix** 関数は -8 をそれぞれ返します。

`Fix(number)` これは、次のコードと同じです。`Sign(number) * Int(Abs(number))`.

使用例

Int 関数と **Fix** 関数を使って、数の整数部分を返す例を次に示します。負の値を指定した場合、**Int** 関数がその数値以下の最大の負の整数を返すのに対して、**Fix** 関数はその数値以上の最小の負の整数を返します。この例では、**Option Strict Off** を指定する必要があります。これは、**Option Strict On** だと、倍精度浮動小数点数型 (**Double**) から整数型 (**Integer**) への暗黙の型変換が許可されないためです。

VB

```
' This code requires Option Strict Off
Dim MyNumber As Integer
MyNumber = Int(99.8)   ' Returns 99.
MyNumber = Fix(99.8)   ' Returns 99.

MyNumber = Int(-99.8)  ' Returns -100.
MyNumber = Fix(-99.8)  ' Returns -99.

MyNumber = Int(-99.2)  ' Returns -100.
MyNumber = Fix(-99.2)  ' Returns -99.
```

CInt 関数を使用すると、**Option Strict Off** の場合に、他のデータ型を明示的に整数型 (**Integer**) に変換できます。ただし、**CInt** は、数値の小数部を切り捨てるのではなく、最も近い整数に丸めます。以下に例を示します。

VB

```
MyNumber = CInt(99.8)    ' Returns 100.
MyNumber = CInt(-99.8)   ' Returns -100.
```

```
MyNumber = CInt(-99.2)    ' Returns -99.
```

Fix または **Int** の呼び出しの結果に対して **CInt** 関数を使用すると、値を丸めずに明示的に整数に変換できます。以下に例を示します。

VB

```
MyNumber = CInt(Fix(99.8))    ' Returns 99.
MyNumber = CInt(Int(99.8))    ' Returns 99.
```

CInt の詳細については、「データ型変換関数」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Conversion**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

データ型変換関数
整数型 (Integer) (Visual Basic)
数値演算の概要
数値演算関数 (Visual Basic)
変換の概要
ArgumentNullException

IPmt 関数

倍精度浮動小数点数型 (**Double**) の値を返します。定額の支払いを定期的に行い、利率が一定であると仮定して、投資期間内の指定した期に支払う金利を返します。

```
Function IPmt( _
   ByVal Rate As Double, _
   ByVal Per As Double, _
   ByVal NPer As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

パラメータ

Rate

投資期間を通じて一定の利率を示す倍精度浮動小数点数型 (**Double**) の値を指定します。たとえば、年利 (APR) 10% の車のローンを月払いで返済する場合、各期間の利率は 0.1/12 または 0.0083 になります。

Per

必ず指定します。金利支払額を求める期を示す倍精度浮動小数点数型 (**Double**) の値を 1 ～ *NPer* の範囲で指定します。

NPer

必ず指定します。投資期間全体での支払い回数の合計を示す倍精度浮動小数点数型 (**Double**) の値。たとえば、4 年の車のローンを月払いで返済する場合、支払い回数は 48 回 (12 回×4 年) になります。

PV

必ず指定します。現在の投資額、つまり将来行われる一連の支払いや収益を現時点で一括した場合の合計金額を示す倍精度浮動小数点数型 (**Double**) の値。たとえば、車を購入するために資金を借りる場合、現在価値は毎月支払うローンの総額になります。

FV

省略可能。投資の将来価値、つまり最後の支払いを行った後に残る現金の収支を示す倍精度浮動小数点数型 (**Double**) の値。たとえば、ローンなどの借入の将来価値は 0 になります。また、子供の教育費用として 18 年間で 500,000 円貯蓄する場合は、将来価値が 500,000 円になります。このパラメータを省略すると、0 を指定したものと見なされます。

Due

省略可能です。支払期日を示すオブジェクト型 DueDate 列挙型 の値を指定します。各期の期末に支払う場合は `DueDate.EndOfPeriod` を、各期の期首に支払う場合は `DueDate.BegOfPeriod` をそれぞれ引数に指定します。この引数を省略すると、`DueDate.EndOfPeriod` を指定したものと見なされます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Per* <= 0 または *Per* > *NPer*

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

投資とは、一連の定額の支払いを長期間行うことです。たとえば、住宅ローンなどのローンまたは毎月の貯蓄プランなどの出資を指します。

引数 *Rate* および *NPer* は、単位が同じ支払い期日を使用して計算する必要があります。たとえば、*Rate* を月単位で計算する場合は、*NPer* も月単位で計算する必要があります。

出金 (定額預金の支払いなど) を表す引数には負の値を指定し、入金 (配当金など) を表す引数には正の値を指定します。

使用例

次の例は、**IPmt** 関数を使って、毎月の支払い額が同額の場合の金利を計算します。利率 (`APR / 12`)、金利支払い額を求める期

(Period)、支払い回数 (TotPmts)、現在価値または元金 (PVal)、将来価値 (FVal)、支払い期日 (PayType) を指定します。

VB

```
Sub TestIPMT()
    Dim APR, PVal, Period, IntPmt, TotInt, TotPmts As Double
    Dim PayType As DueDate
    Dim Response As MsgBoxResult

    ' Usually 0 for a loan.
    Dim Fval As Double = 0
    ' Define money format.
    Dim Fmt As String = "###,###,##0.00"
    PVal = CDbl(InputBox("How much do you want to borrow?"))
    APR = CDbl(InputBox("What is the annual percentage rate of your loan?"))
    If APR > 1 Then APR = APR / 100 ' Ensure proper form.
    TotPmts = CInt(InputBox("How many monthly payments?"))
    Response = MsgBox("Do you make payments at end of the month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    For Period = 1 To TotPmts   ' Total all interest.
        IntPmt = IPmt(APR / 12, Period, TotPmts, -PVal, Fval, PayType)
        TotInt = TotInt + IntPmt
    Next Period

    ' Display results.
    MsgBox("You will pay a total of " & Format(TotInt, Fmt) & _
        " in interest for this loan.")
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

財務処理の概要

ArgumentException

IRR 関数

倍精度浮動小数点数型 (Double) の値を返します。一連の定期的なキャッシュ フロー (支払いと収益) に基づいて、内部利益率を返します。

```
Function IRR( _
    ByRef ValueArray() As Double, _
   Optional ByVal Guess As Double = 0.1 _
) As Double
```

パラメータ

ValueArray

必ず指定します。キャッシュ フローの値を**倍精度浮動小数点数型 (Double)** の配列として指定します。配列には、負の値 (支払額) と正の値 (収益額) が少なくとも 1 つずつ含まれている必要があります。

Guess

省略可能です。**IRR** 関数によって返される推定値を示すオブジェクト。省略すると、*Guess* に 0.1 (10%) を指定したものと見なされます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	配列の引数値が無効、または *Guess* <= -1 です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

内部利益率とは、一定間隔で発生する支払いと収益から成る投資に対する受け取り利率のことです。

IRR 関数では、配列で指定した値の順序がキャッシュ フローの順序であると見なされます。支払額と収益額を入力するときは、その順序に注意してください。各期のキャッシュ フローは、投資のキャッシュ フローのように定額である必要はありません。

IRR 関数は、反復計算の手法を用いて利率を計算します。引数 *Guess*、**IRR** の値を初期値とし、計算結果の誤差が 0.00001% 以内になるまで反復計算を行います。**IRR** が反復計算を 20 回行っても適切な解が見つからない場合は、エラーとなります。

使用例

次の例は、**IRR** 関数を使って、配列 `Values()` に指定した 5 つのキャッシュ フローに対する内部利益率を計算します。最初の配列要素は、事業の操業資金を表す負のキャッシュ フローです。他の 4 つの配列要素は、以後 4 年間の収益を表す正のキャッシュ フローです。`Guess` には、内部利益率の推定値を指定します。

VB

```
' Define money format.
Dim MoneyFmt As String = "###,##0.00"
' Define percentage format.
Dim PercentFmt As String = "#0.00"

Dim values(4) As Double
' Business start-up costs.
values(0) = -70000
' Positive cash flows reflecting income for four successive years.
values(1) = 22000
values(2) = 25000
values(3) = 28000
values(4) = 31000

' Use the IRR function to calculate the rate of return.
' Guess starts at 10 percent.
Dim Guess As Double = 0.1
' Calculate internal rate.
```

```
Dim CalcRetRate As Double = IRR(values, Guess) * 100
' Display internal return rate.
MsgBox("The internal rate of return for these cash flows is " & _
    Format(CalcRetRate, CStr(PercentFmt)) & " percent.")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : Financial

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

NPV 関数
MIRR 関数
財務処理の概要
ArgumentException

IsArray 関数 (Visual Basic)

変数が配列かどうかを調べ、結果をブール型 **(Boolean)** の値で返します。

```
Public Function IsArray(ByVal VarName As Object) As Boolean
```

パラメータ

VarName

必ず指定します。オブジェクト型 (**Object**) の変数です。

解説

変数が配列をポイントしていれば、**IsArray** は **True** を返し、そうでなければ **False** を返します。**IsArray** は、配列を含む可能性のあるオブジェクトでよく利用されます。

使用例

この例では、**IsArray** 関数を使って、いくつかの変数が配列を参照するかどうかを調べます。

VB

```
Dim firstArray(4), secondArray(3) As Integer
Dim thisString As String = "Test"
Dim arrayCheck As Boolean
arrayCheck = IsArray(firstArray)
arrayCheck = IsArray(secondArray)
arrayCheck = IsArray(thisString)
' The first two calls to IsArray return True; the third returns False.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : 配列変数を宣言する

関連項目

IsDate 関数 (Visual Basic)
IsDBNull 関数
IsError 関数
IsNothing 関数
IsNumeric 関数 (Visual Basic)
IsReference 関数
オブジェクト型 (Object)
TypeName 関数 (Visual Basic)

IsDate 関数 (Visual Basic)

式が有効な **Date** 型を表すかどうかを示す **Boolean** 値を返します。

```
Public Function IsDate(ByVal Expression As Object) As Boolean
```

パラメータ

Expression

必ず指定します。オブジェクト型 (**Object**) の式です。

解説

IsDate は、*Expression* が日付型である場合、または **Date** に変換できる場合、**True** を返します。それ以外の場合は **False** を返します。

Date データ型は、日付と時刻両方の値を保持します。**IsDate** は、*Expression* が有効な日付、時刻、または日付と時刻の値である場合に **True** を返します。

使用例

次の例では、**IsDate** 関数を使用して複数の変数が有効な **Date** 値を表しているかどうかを確認します。

VB

```
Dim firstDate, secondDate As Date
Dim timeOnly, dateAndTime, noDate As String
Dim dateCheck As Boolean
firstDate = CDate("February 12, 1969")
secondDate = #2/12/1969#
timeOnly = "3:45 PM"
dateAndTime = "March 15, 1981 10:22 AM"
noDate = "Hello"
dateCheck = IsDate(firstDate)
dateCheck = IsDate(secondDate)
dateCheck = IsDate(timeOnly)
dateCheck = IsDate(dateAndTime)
dateCheck = IsDate(noDate)
```

この例では、最初の 4 つの呼び出しには **IsDate** が **True** を返し、最後の呼び出しには **False** を返します。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

IsArray 関数 (Visual Basic)
IsDBNull 関数
IsError 関数
IsNothing 関数
IsNumeric 関数 (Visual Basic)
IsReference 関数
オブジェクト型 (Object)
日付型 (Date) (Visual Basic)
TypeName 関数 (Visual Basic)

IsDBNull 関数

式が System.DBNull クラスとして評価できるかどうかを調べ、結果を **Boolean** 値で返します。

```
Public Function IsDBNull(ByVal Expression As Object) As Boolean
```

パラメータ

Expression

必ず指定します。オブジェクト型 (**Object**) の式です。

解説

IsDBNull は、*Expression* のデータ型が **DBNull** であると評価された場合 **True** を返します。それ以外の場合は **IsDBNull** は **False** を返します。

System.DBNull 値は、**Object** が失われたデータ、または存在しないデータを表していることを示します。**DBNull** は、変数が初期化されていないことを示す **Nothing** とは異なります。また、**DBNull** は長さが 0 の文字列 (**""**) とも異なります。長さが 0 の文字列は、null 文字列として参照されることもあるためです。

使用例

この例では、**IsDBNull** 関数を使って、変数が **DBNull** に評価されるかどうかを調べます。

VB

```
Dim testVar As Object
Dim nullCheck As Boolean
nullCheck = IsDBNull(testVar)
testVar = ""
nullCheck = IsDBNull(testVar)
testVar = System.DBNull.Value
nullCheck = IsDBNull(testVar)
' The first two calls to IsDBNull return False; the third returns True.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

IsArray 関数 (Visual Basic)
IsDate 関数 (Visual Basic)
IsError 関数
IsNothing 関数
IsNumeric 関数 (Visual Basic)
IsReference 関数
オブジェクト型 (Object)
TypeName 関数 (Visual Basic)
DBNull

IsError 関数

式が例外型かどうかを調べ、結果をブール型 (**Boolean**) の値で返します。

```
Public Function IsError(ByVal Expression As Object) As Boolean
```

パラメータ

Expression

必ず指定します。オブジェクト型 (**Object**) の式です。

解説

IsError は、式が System 名前空間の Exception クラスから派生したオブジェクト型 (**Object**) の変数を表す場合に **True** を返します。

System.Exception から派生した例外は、**Try...Catch...Finally** ステートメントを使用してキャッチできます。

使用例

次の例では、**IsError** 関数を使って、式がシステム例外を表すかどうかを調べます。

VB

```
Sub demonstrateIsError(ByVal firstArg As Integer)
    Dim returnVal As New Object
    Dim badArg As String = "Bad argument value"
    Dim errorCheck As Boolean
    If firstArg > 10000 Then
        returnVal = New System.ArgumentOutOfRangeException(badArg)
    End If
    errorCheck = IsError(returnVal)
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

IsArray 関数 (Visual Basic)
IsDate 関数 (Visual Basic)
IsDBNull 関数
IsNothing 関数
IsNumeric 関数 (Visual Basic)
IsReference 関数
オブジェクト型 (Object)
TypeName 関数 (Visual Basic)

IsNothing 関数

式に割り当てられているオブジェクトがないかどうかを調べ、結果をブール型 **(Boolean)** の値で返します。

```
Public Function IsNothing(ByVal Expression As Object) As Boolean
```

パラメータ

Expression

必ず指定します。**Object** を指定します。

解説

IsNothing は、式が現在オブジェクトが割り当てられていないオブジェクト変数であれば **True** を、それ以外の場合は **False** を返します。

IsNothing は参照型に使用します。値型に Nothing の値を格納することはできません。また、値型に **Nothing** を割り当てると、値型は既定値に戻ります。*Expression* に値型を指定すると、**IsNothing** は必ず **False** を返します。

使用例

次の例は、**IsNothing** 関数を使って、オブジェクト変数がオブジェクト インスタンスに関連付けられているかどうかを調べます。

VB

```
Dim testVar As Object
' No instance has been assigned to variable testVar yet.
Dim testCheck As Boolean
' The following call returns True.
testCheck = IsNothing(testVar)
' Assign a string instance to variable testVar.
testVar = "ABCDEF"
' The following call returns False.
testCheck = IsNothing(testVar)
' Disassociate variable testVar from any instance.
testVar = Nothing
' The following call returns True.
testCheck = IsNothing(testVar)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

IsArray 関数 (Visual Basic)
IsDate 関数 (Visual Basic)
IsDBNull 関数
IsError 関数
IsNumeric 関数 (Visual Basic)
IsReference 関数
オブジェクト型 (Object)
TypeName 関数 (Visual Basic)

概念

値型と参照型

IsNumeric 関数 (Visual Basic)

式が数値として評価できるかどうかを調べ、結果をブール型 **(Boolean)** の値で返します。

```
Public Function IsNumeric(ByVal Expression As Object) As Boolean
```

パラメータ
Expression

必ず指定します。**Object** を指定します。

解説

IsNumeric は、*Expression* のデータ型が **Boolean**、**Byte**、**Decimal**、**Double**、**Integer**、**Long**、**SByte**、**Short**、**Single**、**UInteger**、**ULong**、または **UShort** の場合、あるいは、これらのいずれかの数値型を保持する **Object** の場合に **True** を返します。また、*Expression* が、数値に正しく変換できる **Char** または **String** である場合にも **True** を返します。

Expression のデータ型が **Date** の場合、または数値型を保持しない **Object** の場合、**IsNumeric** は **False** を返します。また、*Expression* が、数値に変換できない **Char** または **String** の場合にも、**IsNumeric** は **False** を返します。

使用例

次の例は、**IsNumeric** 関数を使って、変数の内容が数値として評価できるかどうかを調べます。

VB

```
Dim testVar As Object
Dim numericCheck As Boolean
testVar = "53"
' The following call to IsNumeric returns True.
numericCheck = IsNumeric(testVar)
testVar = "459.95"
' The following call to IsNumeric returns True.
numericCheck = IsNumeric(testVar)
testVar = "45 Help"
' The following call to IsNumeric returns False.
numericCheck = IsNumeric(testVar)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
IsArray 関数 (Visual Basic)
IsDate 関数 (Visual Basic)
IsDBNull 関数
IsError 関数
IsNothing 関数
IsReference 関数
オブジェクト型 (Object)
TypeName 関数 (Visual Basic)

IsReference 関数

式が参照型として評価できるかどうかを調べ、結果をブール型 **(Boolean)** の値で返します。

```
Public Function IsReference(ByVal Expression As Object) As Boolean
```

パラメータ

Expression

必ず指定します。オブジェクト (**Object**) を指定します。

解説

IsReference は、*Expression* が参照型 (クラスのインスタンス、文字列型 (**String**)、任意の型の配列など) を表す場合には **True** を返し、それ以外の場合は、**False** を返します。

参照型には、メモリの他の場所に格納されているデータへのポインタが含まれます。数値型には独自のデータが含まれます。

使用例

この例では、**IsReference** 関数を使って、いくつかの変数が参照型を参照するどうかを調べます。

VB

```
Dim testArray(3) As Boolean
Dim testString As String = "Test string"
Dim testObject As Object = New Object()
Dim testNumber As Integer = 12
testArray(0) = IsReference(testArray)
testArray(1) = IsReference(testString)
testArray(2) = IsReference(testObject)
testArray(3) = IsReference(testNumber)
```

次の例では、3 回目までの **IsReference** の呼び出しが **True** を返します。最後の呼び出しは、**Integer** が値型であり、参照型でないため **False** を返します。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

IsArray 関数 (Visual Basic)
IsDate 関数 (Visual Basic)
IsDBNull 関数
IsError 関数
IsNothing 関数
IsNumeric 関数 (Visual Basic)
オブジェクト型 (Object)
TypeName 関数 (Visual Basic)

概念

値型と参照型

Join 関数 (Visual Basic)

配列に含まれる多数の部分文字列を結合して作成される文字列を返します。

```
Function Join(
   ByVal SourceArray() As { Object | String },
   Optional ByVal Delimiter As String = " "
) As String
```

パラメータ

SourceArray

必ず指定します。結合する部分文字列を含む 1 次元配列を指定します。

Delimiter

省略可能です。戻り値となる文字列を部分文字列に区切るのに使用する文字列を指定します。省略すると、スペース (" ") が使用されます。*Delimiter* が長さ 0 の文字列 ("") かまたは **Nothing** である場合は、リスト内のすべての項目が区切り文字なしで連結されます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*SourceArray* が 1 次元ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Join 関数と **Split** 関数の間には類似点があります。**Join** 関数は文字列の配列を受け取り、それらをデリミタ文字列を使って結合し、単一の文字列を返します。**Split** 関数は文字列を受け取って区切り文字の位置で分割し、文字列の配列を返します。ただし、**Join** がデリミタ文字列を使って文字列を結合できるのに対し、**Split** が文字列を分割する際には、デリミタ文字 (1 文字) しか使用できないという重要な違いがあります。

使用例

Join 関数を使用して、複数の文字列からリストを作成する例は次のようになります。

VB

```
Dim TestItem() As String = {"Pickle", "Pineapple", "Papaya"}
' Returns "Pickle, Pineapple, Papaya"
Dim TestShoppingList As String = Join(TestItem, ", ")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
Split 関数 (Visual Basic)
ArgumentException

Kill 関数

ディスクからファイルを削除します。

My 機能により、**Kill** よりも、ファイルの I/O 操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Sub Kill(ByVal PathName As String)
```

パラメータ

PathName

必ず指定します。削除する 1 つ以上のファイル名を指定する **String** 式です。*PathName* には、ディレクトリまたはフォルダ、ドライブを含めることができます。

例外

例外の種類	エラー番号	条件
IOException	55	対象のファイルが開かれています。
FileNotFoundException	53	対象のファイルが見つかりません。
SecurityException	アクセス許可は拒否されました。(Visual Basic)	アクセス許可が拒否された。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

Kill 関数は、複数文字 (*) および単一文字 (**?**) のワイルドカードに対応しており、複数のファイルを一度に指定できます。

セキュリティに関するメモ **Kill** 関数を実行するには、実行されるコードに FileIOPermission の **Read** および **PathDiscovery** フラグが付与されている必要があります。詳細については、「**SecurityException**」および「コード アクセス許可」を参照してください。

使用例

Kill 関数を使って、ディスクからファイルを削除するコード例を次に示します。

VB

```
' Assume TESTFILE is a file containing some data.
Kill("TestFile")   ' Delete file.

' Delete all *.TXT files in current directory.
Kill("*.TXT")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

RmDir 関数
IOException Class
FileNotFoundException Class

LBound 関数 (Visual Basic)

配列の指定された次元で使用できる添字の最小値を返します。

```
Public Function LBound( _
   ByVal Array As System.Array, _
   Optional ByVal Rank As Integer = 1 _
) As Integer
```

パラメータ

Array

必ず指定します。任意のデータ型の配列です。ある次元で使用できる添字の最小値を調べる対象となる配列を指定します。

Rank

省略可能です。**Integer** 型の値です。使用できる添字の最小値を取得する次元を指定します。最初の次元なら 1、2 番目の次元なら 2 というように指定します。*Rank* を省略すると、1 が使用されます。

戻り値

Integer 型の値です。指定された次元に格納できる添字の最小値です。*Array* が初期化されていれば、要素が長さゼロの文字列であるなど、要素がない場合でも **LBound** は必ず 0 を返します。*Array* が **Nothing** であれば、**LBound** は ArgumentNullException をスローします。

例外

例外の種類	エラー番号	条件
ArgumentNullException	9	*Array* が **Nothing** です。
RankException	9	*Rank* < 1 です。または、*Rank* が *Array* のランクより大きな値です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

配列の添字は 0 から始まるため、各次元で使用できる添字の最小値は常に 0 です。

配列の次元と **LBound** 関数が返す値の関係を次に示します。

```
Dim a(100, 5, 4) As Byte
```

LBound の呼び出し	戻り値
`LBound(a, 1)`	0
`LBound(a, 2)`	0
`LBound(a, 3)`	0

使用例

LBound 関数を使って、配列の指定された次元で使用できる添字の最小値を調べるコード例は、次のとおりです。

VB

```
Dim lowest, bigArray(10, 15, 20), littleArray(6) As Integer
lowest = LBound(bigArray, 1)
lowest = LBound(bigArray, 3)
lowest = LBound(littleArray)
' All three calls to LBound return 0.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

UBound 関数 (Visual Basic)
Dim ステートメント (Visual Basic)
ReDim ステートメント (Visual Basic)
ArgumentException
RankException

LCase 関数 (Visual Basic)

小文字に変換した文字列または文字を返します。

```
Public Shared Function LCase(ByVal Value As Char) As Char
' -or-
Public Shared Function LCase(ByVal Value As String) As String
```

パラメータ
Value

必ず指定します。任意の有効な文字列 (**String**) または文字 (**Char**) 式。

解説

大文字だけが小文字に変換されます。大文字のアルファベット以外の文字には影響がありません。

この関数は文字列を操作するときにアプリケーションのカルチャ情報を使用するため、大文字小文字がアプリケーションで使用しているロケールに対して正しく変換されます。

セキュリティに関するメモ：

比較処理または大文字/小文字変換処理の結果に基づいてアプリケーションのセキュリティ関連の決定を下す場合は、その処理を System.String.Compare(System.String,System.String,System.StringComparison) メソッドで実行し、*comparisonType* 引数に Ordinal または OrdinalIgnoreCase を渡してください。詳細については、「Visual Basic においてカルチャが文字列に与える影響」を参照してください。

使用例

LCase 関数を使って文字列を小文字に変換する例を次に示します。

VB

```
' String to convert.
Dim UpperCase As String = "Hello World 1234"
' Returns "hello world 1234".
Dim LowerCase As String = LCase(UpperCase)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
UCase 関数 (Visual Basic)
その他の技術情報
Visual Basic における文字列
Visual Basic の文字列の概要

Left 関数 (Visual Basic)

文字列の左端から指定された文字数分の文字列を返します。

```
Public Shared Function Left( _
   ByVal str As String, _
   ByVal Length As Integer _
) As String
```

パラメータ

str

必ず指定します。文字列型 (**String**) の式です。この左端から文字列が取り出されます。

Length

必ず指定します。整数型 (**Integer**) の式です。取り出す文字列の文字数を示す数式を指定します。0 を指定した場合は、長さ 0 の文字列 ("") を返します。引数 *str* の文字数以上の場合は、文字列全体が返されます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Length* < 0.

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

str の文字数を確認するには、**Len** 関数を使用します。Windows フォーム アプリケーションで使用された場合や、**Left** プロパティを持つクラスで使用された場合、関数を `Microsoft.VisualBasic.Left` で完全修飾する必要があります。

メモ :

以前のバージョンの Visual Basic では、**LeftB** 関数は文字数ではなくバイト数に基づいて文字列を返していました。これは主に、2 バイト文字セット (DBCS) アプリケーションで文字列を変換するために使用します。現在 Visual Basic のすべての文字列は Unicode で、**LeftB** はサポートされていません。

使用例

Left 関数を使って、指定された **String** の部分文字列を返す例を次に示します。**Left** プロパティを持つクラスでは、**Left** 関数を完全修飾しなければならないことがあります。

VB

```
Dim TestString As String = "Hello World!"
' Returns "Hello".
Dim subString As String = Microsoft.VisualBasic.Left(TestString, 5)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
Right 関数 (Visual Basic)
Len 関数 (Visual Basic)

Mid 関数 (Visual Basic)
ArgumentException
概念
プログラミング要素のサポートに関する変更の概要

Len 関数 (Visual Basic)

指定された文字列の文字数、または変数の格納に必要な名目上のバイト数を含む整数型の値を返します。

```
Public Shared Function Len( _
   ByVal Expression As { Boolean | Byte | SByte | Char | Double |
   Integer | UInteger | Long | ULong | Object | Short | UShort |
   Single | String | DateTime | Decimal } _
) As Integer
```

パラメータ

Expression

有効な文字列型 (**String**) の式または変数名を指定します。*Expression* がオブジェクト型 (**Object**) の場合、**Len** 関数はその型が **FilePut** 関数によってファイルに書き込まれるときのサイズを返します。

解説

ユーザー定義の型およびオブジェクト型 (**Object**) の変数の場合、**Len** 関数はその型が **FilePut** 関数によってファイルに書き込まれるときのサイズを返します。**Object** に文字列型 (**String**) が含まれる場合は、文字列の長さを返します。**Object** にその他の型が含まれる場合は、オブジェクトが **FilePut** 関数によってファイルに書き込まれるときのサイズを返します。

VBFixedString 属性をオブジェクト内の文字列フィールドに適用すると、ディスクに書き込むときに文字列のサイズをバイト数で示すことができます。**Len** 関数は、**Object** 変数のサイズを判断するとき **VBFixedString** 属性を (可能であれば) 使用します。

メモ :

Len 関数をユーザー定義のデータ型に含まれる可変長文字列に対して実行する場合は、格納に必要な実際のバイトを決定できない場合があります。

メモ :

以前のバージョンの Visual Basic では、**LenB** 関数は文字数ではなくバイト数を返していました。これは主に、2 バイト文字セット (DBCS) アプリケーションで文字列を変換するために使用します。現在 Visual Basic のすべての文字列は Unicode で、**LenB** はサポートされていません。

使用例

Len 関数を使って文字列の文字数を返す例を次に示します。

VB

```
' Initializes variable.
Dim TestString As String = "Hello World"
' Returns 11.
Dim TestLen As Integer = Len(TestString)
```

スマート デバイス開発者のためのメモ

Len 関数は、**String** 変数と **Object** 変数だけをパラメータとして受け入れます。**Object** に文字列型 (**String**) が含まれる場合は、文字列の長さを返します。パラメータが NULL **Object** 参照である場合は、ゼロを返します。**Object** にその他の型が含まれる場合は、例外がスローされます。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

データ型の概要 (Visual Basic)

FilePut 関数
VBFixedStringAttribute クラス
文字列操作の概要
データ型の概要 (Visual Basic)
InStr 関数 (Visual Basic)
概念
プログラミング要素のサポートに関する変更の概要

LineInput 関数

シーケンシャル入力モードで開いたファイルから行全体を読み込み、文字列型 (**String**) の変数に代入します。

```
Public Function LineInput(ByVal FileNumber As Integer) As String
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

例外

例外の種類	エラー番号	条件
EndOfStreamException	62	ファイルの終端に達しました。
IOException	52	*FileNumber* が存在していません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

LineInput 関数は、下位互換性を保つために提供されており、パフォーマンスに影響を与える可能性があります。非レガシ アプリケーションに対しては、**My.Computer.FileSystem** オブジェクトを使用した方が、パフォーマンスが高くなります。詳細については、「Visual Basic におけるファイル アクセス」を参照してください。

通常、**LineInput** 関数を使用して読み込んだデータは **Print** 関数を使用して書き込みます。

セキュリティに関するメモ：
ファイルからデータを読み取るときは、ファイル名の拡張子に基づいてファイルの内容を判断しないでください。たとえば、Form1.vb という名前のファイルが Visual Basic のソース ファイルとは限りません。

LineInput 関数は、キャリッジ リターン (**Chr(**13**)**) またはキャリッジ リターンとライン フィード (**Chr(**13**)** + **Chr(**10**)**) が見つかるまで、ファイルから一度に 1 文字ずつ読み込みます。キャリッジ リターンとライン フィードは、文字列に追加されるのではなく、読み飛ばされます。

セキュリティに関するメモ：
LineInput 関数を使ってファイルを読み込むには、FileIOPermissionAccess 列挙体の **Read** アクセスが必要です。

使用例

LineInput 関数を使って、シーケンシャル ファイルから 1 行ずつ読み込んで変数に代入する例を次に示します。`TestFile` は、複数行のサンプル データを含むテキスト ファイルと仮定します。

VB

```
Dim TextLine As String
' Open file.
FileOpen(1, "TESTFILE", OpenMode.Input)
' Loop until end of file.
While Not EOF(1)
   ' Read line into variable.
   TextLine = LineInput(1)
   ' Print to the console.
   WriteLine(1, TextLine)
End While
FileClose(1)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

方法 : Visual Basic でテキストをファイルに書き込む

関連項目

Chr 関数、ChrW 関数

Input 関数

その他の技術情報

Visual Basic におけるファイル アクセス

Loc 関数

開いたファイル内の現在の読み込み位置または書き込み位置を指定する長整数型 (**Long**) の値を返します。

```
Public Function Loc(ByVal FileNumber As Integer) As Long
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号を表す整数型 (**Integer**) の値。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Loc 関数はゼロから始まります。つまり、ファイルの先頭のバイトを取得すると 0 が返されます。

Loc 関数は、下位互換性を保つために提供されており、パフォーマンスに影響を与える可能性があります。非レガシ アプリケーションに対しては、**My.Computer.FileSystem** オブジェクトを使用した方が、パフォーマンスが高くなります。詳細については、「Visual Basic におけるファイル アクセス」を参照してください。

各ファイル アクセス モードの戻り値を次に示します。

モード	戻り値
Random	ファイルに対して直前に入出力を行ったレコードの番号を返します。
Sequential	ファイル内の現在のバイト位置を 128 で割った値を返します。シーケンシャル ファイル入力モードのファイルに対する **Loc** 関数の戻り値です。
Binary	直前に入出力を行ったバイト位置を返します。

使用例

Loc 関数を使って、開いているファイル内の現在の読み書き位置を返す例を次に示します。`MyFile` は、複数行のサンプル データを含むテキスト ファイルと仮定します。

VB

```
Dim location As Long
Dim oneChar As Char
FileOpen(1, "C:\TESTFILE.TXT", OpenMode.Binary)
While location < LOF(1)
   Input(1, oneChar)
   location = Loc(1)
   WriteLine(1, location & ControlChars.CrLf)
End While
FileClose(1)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

EOF 関数
LOF 関数
Seek 関数
IOException

Lock 関数、Unlock 関数

Open 関数で開いたファイルの一部または全体に対する他のプロセスからのアクセスを制御します。

My 機能を使用すると、**Lock** および **Unlock** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Overloads Sub Lock(ByVal FileNumber As Integer)
' -or-
Public Overloads Sub Unlock(ByVal FileNumber As Integer)
' -or-
Public Overloads Sub Lock(_
   ByVal FileNumber As Integer, _
   ByVal Record As Long _
)
' -or-
Public Overloads Sub Unlock( _
   ByVal FileNumber As Integer, _
   ByVal Record As Long _
)
' -or-
Public Overloads Sub Lock( _
   ByVal FileNumber As Integer, _
   ByVal FromRecord As Long, _
   ByVal ToRecord As Long _
)
' -or-
Public Overloads Sub Unlock( _
   ByVal FileNumber As Integer, _
   ByVal FromRecord As Long, _
   ByVal ToRecord As Long _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

Record

省略可能です。ロックまたはロックを解除する唯一のレコード番号またはバイト位置。

FromRecord

省略可能です。ロックまたはロックを解除する先頭のレコード番号またはバイト位置。

ToRecord

省略可能です。ロックまたはロックを解除する末尾のレコード番号またはバイト位置。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Lock 関数および **Unlock** 関数は、複数のプロセスが同じファイルにアクセスする環境で使います。

Lock 関数および **Unlock** 関数は、必ず 2 つ 1 組で使用します。**Lock** 関数および **Unlock** 関数に指定する引数は一致している必要があります。

引数の *Record*、または *FromRecord* と *ToRecord* を指定しない場合、ロックはファイル全体に適用されます。*Record* だけが指定された場合は、単一のレコードがロックまたはロック解除されます。

シーケンシャル入力モードまたはシーケンシャル出力モードで開いたファイルでは、*FromRecord* および *ToRecord* の範囲指定にかかわらず、ファイル全体が **Lock** 関数と **Unlock** 関数の対象になります。

使用例

Lock 関数および **Unlock** 関数の使用例を次に示します。ファイル `People.txt` には、構造体 `Person` のレコードが含まれているものと仮定します。

VB

```
Structure Person
   Dim Name As String
   Dim ID As Integer
End Structure

Sub PutInLockedFile(ByVal index As Integer, ByVal onePerson As Person)
   Try
      FileOpen(1, "c:\people.txt", OpenMode.Binary)
      Lock(1)
      FilePut(index, onePerson)
      Unlock(1)
      FileClose(1)
   Catch
      ' Error recovery code here.
   End Try
End Sub
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileOpen 関数

IOException Class

その他の技術情報

Visual Basic におけるファイル アクセス

LOF 関数

FileOpen 関数を使用して開いたファイルのバイト単位のサイズを表す長整数型 (**Long**) の値を返します。

My 機能を使用すると、**LOF** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Function LOF(ByVal FileNumber As Integer) As Long
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号を表す整数型 (**Integer**) の値です。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

開いていないファイルのサイズを取得するには、**FileLen** 関数を使用します。

使用例

LOF 関数を使って、開いたファイルのサイズを調べる例を次に示します。ファイル `TestFile` は、サンプル データを含むテキスト ファイルと仮定します。

VB

```
Dim length As Long
FileOpen(1, "C:\TESTFILE.TXT", OpenMode.Input) ' Open file.
length = LOF(1)   ' Get length of file.
MsgBox(length)
FileClose(1)   ' Close file.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

EOF 関数
FileLen 関数
Loc 関数
FileOpen 関数
IOException Class

その他の技術情報

Visual Basic でのファイルの読み取り
Visual Basic でのファイルへの書き込み

LSet 関数

文字列と長さが指定され、その長さに調整された文字列を左揃えにして文字列を返します。

```
Public Shared Function LSet( _
       ByVal Source As String, _
       ByVal Length As Integer _
) As String
```

パラメータ

Source

必ず指定します。**String** を指定します。調整される文字列変数。

Length

必ず指定します。**Integer** を指定します。返される文字列の長さです。

解説

Source が *Length* より長い場合、**LSet** 関数は、文字列の末尾から *Length* の長さの文字だけを返します。*Source* が *Length* より短い場合、**LSet** は文字列の右にスペースを追加し、適切な長さにします。

使用例

LSet 関数の使用例を次に示します。

VB

```
Dim TestString As String = "Left"
Dim lString As String
' Returns "Left      "
lString = LSet(TestString, 10)
' Returns "Le"
lString = LSet(TestString, 2)
' Returns "Left"
lString = LSet(TestString, 4)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
データ型の概要 (Visual Basic)
RSet 関数

概念

データ型の有効な使用方法

関数 M ～ R

次の表は、Visual Basic のランタイム メンバ関数の一覧です。

Mid	Minute	MIRR	MkDir
Month	MonthName	MsgBox	NPer
NPV	Oct	Partition	Pmt
PPmt	Print	PrintLine	PV
QBColor	Randomize	Rate	Rename
Replace	Reset	RGB	Right
RmDir	Rnd	RSet	RTrim

バージョン 6.0 以前に Visual Basic のランタイム メンバ関数として利用できた数値演算関数 **Round** は、.NET Framework クラス ライブラリの Math クラスのメソッド (Round) に置き換えられました。詳細については、「数値演算関数 (Visual Basic)」を参照してください。

スマート デバイス開発者のためのメモ

MkDir、**Print**、**PrintLine**、**Rename**、**Reset**、および **RmDir** は、スマート デバイス アプリケーションではサポートされません。

参照

関連項目

関数 A ～ C
関数 D ～ G
関数 H ～ L
関数 S ～ Z

その他の技術情報

Visual Basic リファレンス

Mid 関数 (Visual Basic)

文字列から指定された文字数分の文字列を返します。

```
Public Shared Function Mid( _
   ByVal str As String, _
   ByVal Start As Integer, _
   Optional ByVal Length As Integer _
) As String
```

パラメータ

str

必ず指定します。文字列を取り出す対象の文字列型 (**String**) の式を指定します。

Start

必ず指定します。オブジェクト型 (**Integer**) の式です。返す文字の開始点です。引数 *Start* が引数 *str* の文字数を超える場合は、**Mid** 関数は長さ 0 の文字列 ("") を返します。*Start* は 1 から始まります。

Length

省略できます。文字列 (**Integer**) を指定します。取り出す文字数を指定します。引数 *Length* を省略する場合、または文字列内に引数 *Length* より短い文字数しかない場合には、引数 *Start* で指定された位置以降のすべての文字を返します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Start* <= 0 または *Length* < 0 です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

str の文字数を確認するには、**Len** 関数を使用します。

Visual Basic には、**Mid** 関数と **Mid** ステートメントがあります。これらの要素はどちらも、文字列の中の指定された文字数を扱いますが、**Mid** 関数は文字を返すのに対し、**Mid** ステートメントは文字を置き換えます。詳細については、「Mid ステートメント」を参照してください。

メモ :

以前のバージョンの Visual Basic では、**MidB** 関数は文字数ではなくバイト数に基づいて文字列を返していました。これは主に、2 バイト文字セット (DBCS) アプリケーションで文字列を変換するために使用します。Visual Basic のすべての文字列は Unicode で、**MidB** はサポートされていません。

使用例

Mid 関数を使って、文字列から指定された字数を返す例を次に示します。

VB

```
' Creates text string.
Dim TestString As String = "Mid Function Demo"
' Returns "Mid".
Dim FirstWord As String = Mid(TestString, 1, 3)
' Returns "Demo".
Dim LastWord As String = Mid(TestString, 14, 4)
' Returns "Function Demo".
Dim MidWords As String = Mid(TestString, 5)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
Left 関数 (Visual Basic)
Len 関数 (Visual Basic)
Trim、LTrim、RTrim 関数
Mid ステートメント
Right 関数 (Visual Basic)
ArgumentException

概念

プログラミング要素のサポートに関する変更の概要

その他の技術情報

Visual Basic における文字列
Visual Basic の文字列の概要

Minute 関数

分を表す 0 ～ 59 の整数型 (**Integer**) の値を返します。

```
Public Function Minute(ByVal TimeValue As DateTime) As Integer
```

パラメータ

TimeValue

分を抽出する日付型 (**Date**) の値。

解説

DatePart を呼び出し、*Interval* 引数に **DateInterval.Minute** を指定することによっても分を取得できます。

使用例

次の例は、**Minute** 関数を使って、指定された時刻の "分" で表される部分を取得します。開発環境では、時刻リテラルは、コード記述時のロケールで設定されている短い形式で表示されます。

VB

```
Dim thisTime As Date
Dim thisMinute As Integer
thisTime = #4:35:17 PM#
thisMinute = Minute(thisTime)
' thisMinute now contains 35.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Day 関数 (Visual Basic)
Hour 関数 (Visual Basic)
Now プロパティ
Second 関数 (Visual Basic)
TimeOfDay プロパティ
DatePart 関数 (Visual Basic)
System
DateTime
ArgumentException
ArgumentOutOfRangeException

MIRR 関数

倍精度浮動小数点数型 (Double) の値を返します。一連の定期的なキャッシュ フロー (支払いと収益) に基づいて、修正内部利益率を返します。

```
Function MIRR( _
   ByRef ValueArray() As Double, _
   ByVal FinanceRate As Double, _
   ByVal ReinvestRate As Double _
) As Double
```

パラメータ

ValueArray

必ず指定します。キャッシュ フローの値を **Double** の配列として指定します。配列には、負の値 (支払額) と正の値 (収益額) が少なくとも 1 つずつ含まれている必要があります。

FinanceRate

必ず指定します。支払額に対する利率を示す倍精度浮動小数点数型 (**Double**) の値を指定します。

ReinvestRate

収益額に対する利率を示す倍精度浮動小数点数型 (**Double**) の値を指定します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*ValueArray* のランクが 1 ではない、*FinanceRate* = -1、または *ReinvestRate* = -1。
DivideByZeroException	11	0 による除算が行われました。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

修正内部利益率は、支払いと収益を異なる利率で管理する場合の内部利益率を表します。**MIRR** 関数は、投資原価 (*FinanceRate*) と現金の再投資に対する受け取り利率 (*ReinvestRate*) の両方を基に修正内部利益率を求めます。

FinanceRate および *ReinvestRateарguments* は、小数点を用いた百分率で指定します。たとえば、12% は 0.12 と記述します。

MIRR 関数では、配列で指定した値の順序がキャッシュ フローの順序であると見なされます。支払額と収益額を入力するときは、その順序に注意してください。

使用例

次の例は、**MIRR** 関数を使って、配列 `Values()` に指定されたキャッシュ フローに対する修正内部利益率を計算します。`LoanAPR` は借入金に対する利率を表し、`InvAPR` は再投資に対する利率を表します。

VB

```
' Define money format.
Dim MoneyFmt As String = "###,##0.00"
' Define percentage format.
Dim PercentFmt As String = "#0.00"

Dim values(4) As Double
' Business start-up costs.
values(0) = -70000
' Positive cash flows reflecting income for four successive years.
values(1) = 22000
values(2) = 25000
values(3) = 28000
```

```
values(4) = 31000

' Use the MIRR function to calculate the internal return rate.
' Set the loan rate.
Dim LoanAPR As Double = 0.1
' Set the reinvestment rate.
Dim InvAPR As Double = 0.12
' Calculate internal rate.
Dim RetRate As Double = MIRR(values, LoanAPR, InvAPR)
' Display internal return rate.
MsgBox("The modified internal rate of return for these cash flows is " & _
    Format(Math.Abs(RetRate) * 100, CStr(PercentFmt)) & "%.")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
IRR 関数
NPV 関数
財務処理の概要
ArgumentException
DivideByZeroException

MkDir 関数

新しいディレクトリを作成します。

My 機能により、**MkDir** よりも、ファイルの I/O 操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.FileSystem.CreateDirectory メソッド」を参照してください。

```
Public Sub MkDir(ByVal Path As String)
```

パラメータ

Path

作成するディレクトリを表す文字列 (**String**) 式を指定します。引数 *Path* にはドライブを含めることができます。ドライブを指定しない場合、**MkDir** 関数は現在のドライブにディレクトリを新規作成します。

例外

例外の種類	エラー番号	条件
ArgumentException	52	*Path* が指定されていません。または空です。
SecurityException	70	アクセス許可が拒否された。
IOException	75	ディレクトリが既に存在しています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

セキュリティに関するメモ：

ディレクトリまたはフォルダを作成するには、読み取りと書き込み両方のファイル I/O アクセス許可が必要です。詳細については、「FileIOPermission」および「コード アクセス許可」を参照してください。

解説

この関数を使用すると、既存のディレクトリが存在していたり、*Path* が無効である場合を除き、*Path* で指定された新しいディレクトリが作成されます。*Path* パラメータには、ファイルのパスではなく、ディレクトリのパスを指定する必要があります。

使用例

MkDir 関数を使ってディレクトリを新規作成するコード例を次に示します。ドライブを指定しない場合、現在のドライブにディレクトリを新規作成します。

VB

```
' Make new directory.
MkDir("C:\TESTDIR")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でディレクトリを作成する

関連項目

ChDir 関数
CurDir 関数
RmDir 関数
ArgumentException Class
IOException Class

Month 関数 (Visual Basic)

月を表す 1 ～ 12 の整数型 (**Integer**) の値を返します。

```
Public Function Month(ByVal DateValue As DateTime) As Integer
```

パラメータ
DateValue

必ず指定します。月を抽出する日付型 (**Date**) の値です。

解説

DatePart を呼び出し、*Interval* 引数に **DateInterval.Month** を指定することによっても月を取得できます。

使用例

次の例は、**Month** 関数を使って、指定された日付の "月" で表される部分を取得します。開発環境では、日付リテラルは、コード記述時のロケールで設定されている短い日付の形式で表示されます。

VB

```
Dim thisDate As Date
Dim thisMonth As Integer
thisDate = #2/12/1969#
thisMonth = Month(thisDate)
' thisMonth now contains 2.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
Day 関数 (Visual Basic)
Now プロパティ
Weekday 関数 (Visual Basic)
Year 関数 (Visual Basic)
DatePart 関数 (Visual Basic)
System
DateTime
ArgumentException
ArgumentOutOfRangeException

MonthName 関数 (Visual Basic)

指定した月の名前を含む文字列型 (**String**) の値を返します。

```
Public Function MonthName( _
   ByVal Month As Integer, _
   Optional ByVal Abbreviate As Boolean = False _
) As String
```

パラメータ

Month

必ず指定します。整数型 (**Integer**)。1 ～ 13 の数値で指定した月であり、1 は 1 月、12 は 12 月を表します。13 か月カレンダーでは、値 13 を使用できます。システムで 12 か月カレンダーが使用され、*Month* が 13 の場合、**MonthName** は空の文字列を返します。

Abbreviate

省略可能です。月名を省略するかどうかを表すブール型 (**Boolean**) の値。省略すると、既定値の **False** が使用され、月名は短縮されません。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Month* が 1 より小さいか、または 13 を超えています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

MonthName から返される文字列は、入力引数だけではなく、Windows のコントロール パネルで設定される [地域のオプション] にも依存します。

メモ :

MonthName は、System.Globalization 名前空間にある CultureInfo クラスの CurrentCulture プロパティの現在のカレンダーの設定を使用します。**CurrentCulture** の既定値は、コントロール パネルの設定によって決まります。

使用例

次の例は、**MonthName** 関数を使って、指定された整数に対応する月の名前を取得します。ブール型の値によって、完全な名前 (**False**) と省略名 (**True**) のどちらを表示するかが決まります。

VB

```
Dim thisMonth As Integer
Dim name As String
thisMonth = 4
' Set Abbreviate to True to return an abbreviated name.
name = MonthName(thisMonth, True)
' name now contains "Apr".
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

WeekdayName 関数 (Visual Basic)

System
System.Globalization
DateTime
ArgumentException
CultureInfo

MsgBox 関数 (Visual Basic)

ダイアログ ボックスにメッセージを表示します。ボタンがクリックされるのを待って、どのボタンがクリックされたのかを示す、整数型 (**Integer**) の値を返します。

```
Public Function MsgBox( _
   ByVal Prompt As Object, _
   Optional ByVal Buttons As MsgBoxStyle = MsgBoxStyle.OKOnly, _
   Optional ByVal Title As Object = Nothing _
) As MsgBoxResult
```

パラメータ

Prompt

必ず指定します。ダイアログ ボックス内にメッセージとして表示する文字列 (**String**) 式を指定します。*Prompt* に指定できる最大文字数は、1 バイト文字で約 1,024 文字です。ただし、使用する文字の文字幅に依存します。*Prompt* を複数行で構成する場合は、各行をキャリッジ リターン文字 (**Chr(**13**)**)、ライン フィード文字 (**Chr(**10**)**)、またはキャリッジ リターン文字とライン フィード文字の組み合わせ (**Chr(**13**)** & **Chr(**10**)**) を使って区切ります。

Buttons

省略可能です。表示されるボタンの種類と個数、使用するアイコンのスタイル、標準ボタン、メッセージ ボックスがモーダルかどうかなど、それらを表す値の合計値を示す数式を指定します。*Buttons* を省略すると、既定値は 0 になります。

Title

省略可能です。ダイアログ ボックスのタイトル バーに表示する文字列 (**String**) 式を指定します。*Title* を省略すると、タイトル バーにはアプリケーション名が表示されます。

設定

MsgBoxStyle 列挙型の値を次の表に示します。

メンバ	値	説明
OKOnly	0	[OK] ボタンのみを表示します。
OKCancel	1	[OK] ボタンと [キャンセル] ボタンを表示します。
AbortRetryIgnore	2	[中止]、[再試行]、および [無視] の 3 つのボタンを表示します。
YesNoCancel	3	[はい]、[いいえ]、および [キャンセル] の 3 つのボタンを表示します。
YesNo	4	[はい] ボタンと [いいえ] ボタンを表示します。
RetryCancel	5	[再試行] ボタンと [キャンセル] ボタンを表示します。
Critical	16	警告メッセージ アイコンを表示します。
Question	32	問い合わせメッセージ アイコンを表示します。
Exclamation	48	注意メッセージ アイコンを表示します。
Information	64	情報メッセージ アイコンを表示します。
DefaultButton1	0	第 1 ボタンを標準ボタンにします。
DefaultButton2	256	第 2 ボタンを標準ボタンにします。

DefaultButton3	512	第 3 ボタンを標準ボタンにします。
ApplicationModal	0	アプリケーションをモーダルに設定します。メッセージ ボックスに応答するまで、現在選択中のアプリケーションの実行を継続できません。
SystemModal	4096	システムをモーダルに設定します。メッセージ ボックスに応答するまで、すべてのアプリケーションが中断されます。
MsgBoxSetForeground	65536	最前面のウィンドウとして表示します。
MsgBoxRight	524288	テキストを右寄せで表示します。
MsgBoxRtlReading	1048576	ヘブライ語やアラビア語のシステムの場合、テキストを右から左の方向で表示します。

最初の値のグループ (0–5) には、ダイアログ ボックスに表示されるボタンの数と種類が定義されています。2 番目のグループ (16, 32, 48, 64) には、アイコンのスタイルが定義されています。3 番目のグループ (0, 256, 512) は、どれが既定のボタンかを決定します。4 番目のグループ (0, 4096) はメッセージ ボックスがモーダルかどうかを決定し、5 番目のグループはメッセージ ボックス ウィンドウが画面の手前に表示されるかどうか、またテキストの配置や方向などを指定します。引数 *Buttons* の値を設定するには、各グループから値を 1 つずつ選択して加算した合計値を指定します。

戻り値

定数	値
OK	1
Cancel	2
Abort	3
Retry	4
Ignore	5
Yes	6
No	7

例外

例外の種類	エラー番号	状態
ArgumentException	5	*Prompt* が文字列型 (**String**) の式でないか、*Title* が無効です。
InvalidOperationException	5	プロセスがユーザー対話モードで動作していません。
InvalidEnumArgumentException	5	**MsgBoxResult** 列挙型または **MsgBoxStyle** 列挙型のメンバでないパラメータが含まれています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

[キャンセル] ボタンが表示されているダイアログ ボックスでは、Esc キーを押すと、[キャンセル] をクリックしたときと同じ結果になります。ダイアログ

ボックスに [ヘルプ] ボタンが表示されているとき、そのダイアログ ボックスには状況依存のヘルプが設定されています。ただし、[ヘルプ] ボタン以外のボタンがクリックされるまでは、値を返しません。

メモ：
Prompt 以外の引数も指定する場合は、式の中で **MsgBox** 関数を使用する必要があります。引数を省略する場合も、それに対応する位置にコンマ区切り記号は残します。

メモ：
MsgBox 関数には SafeTopLevelWindows レベルの **UIPermission** が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「アクセス許可の要求」および「UIPermission」を参照してください。

使用例

次の例は、**MsgBox** 関数を使って、致命的なエラーが発生したときのエラー メッセージをダイアログ ボックスに表示します。このダイアログ ボックスには、[はい] と [いいえ] があり、[いいえ] が既定のボタンとして指定されています。これを実現するには **MsgBox** 定数の複数の値を 1 つの数式に設定します。この場合、4 ([はい]/[いいえ] ボタンの組み合わせ) と 16 ([警告メッセージ] ウィンドウ) と 256 (既定ボタンが第 2 ボタン) を足すと、合計 276 になります。**MsgBox** 関数の返す値は、ユーザーが選択したボタンによって決まります。つまり、[はい] を選択すると 6、[いいえ] を選択すると 7 が返されます。

VB

```
Dim msg As String
Dim title As String
Dim style As MsgBoxStyle
Dim response As MsgBoxResult
msg = "Do you want to continue?"    ' Define message.
style = MsgBoxStyle.DefaultButton2 Or _
   MsgBoxStyle.Critical Or MsgBoxStyle.YesNo
title = "MsgBox Demonstration"    ' Define title.
' Display message.
response = MsgBox(msg, style, title)
If response = MsgBoxResult.Yes Then    ' User chose Yes.
   ' Perform some action.
Else
   ' Perform some other action.
End If
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

InputBox 関数 (Visual Basic)

NPer 関数

倍精度浮動小数点数型 (**Double**) の値を返します。定額の支払いを定期的に行い、利率が一定であると仮定して、投資に必要な期間を返します。

```
Function NPer( _
   ByVal Rate As Double, _
   ByVal Pmt As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

パラメータ

Rate

必ず指定します。投資期間を通じて一定の利率を示す倍精度浮動小数点数型 (**Double**) の値です。たとえば、年利 (APR) 10% の車のローンを月払いで返済する場合、各期間の利率は 0.1/12 または 0.0083 になります。

Pmt

必ず指定します。毎回の支払額を示す倍精度浮動小数点数型 (**Double**) の値を指定します。通常、支払額には元金と利息が含まれます。支払額を投資期間内に変更することはできません。

PV

必ず指定します。現在の投資額、つまり将来行われる一連の支払いや収益を現時点で一括した場合の合計金額を示す倍精度浮動小数点数型 (**Double**) の値を指定します。たとえば、車を購入するために資金を借りる場合、現在価値は毎月支払うローンの総額になります。

FV

省略可能です。投資の将来価値、つまり最後の支払いを行った後に残る現金の収支を示す倍精度浮動小数点数型 (**Double**) の値です。たとえば、ローンなどの借入の将来価値は 0 になります。また、子供の教育費用として 18 年間で 500,000 円貯蓄する場合は、将来価値が 500,000 円になります。このパラメータを省略すると、0 を指定したものと見なされます。

Due

省略可能です。支払期日を示すオブジェクト型 DueDate 列挙型 の値を指定します。各期の期末に支払う場合は `DueDate.EndOfPeriod` を、各期の期首に支払う場合は `DueDate.BegOfPeriod` をそれぞれ引数に指定します。この引数を省略すると、`DueDate.EndOfPeriod` を指定したものと見なされます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Rate* <= -1.
ArgumentException	5	*Rate* = 0 かつ *Pmt* = 0 です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

投資とは、一連の定額の支払いを一定の期間行うことです。たとえば、住宅ローンなどのローンまたは毎月の貯蓄プランなどの出資を指します。

出金 (定額預金の支払いなど) を表す引数には負の値を指定し、入金 (配当金など) を表す引数には正の値を指定します。

使用例

次の例は、**NPer** 関数を使って、支払い回数を計算します。ローンの総額 (`PVal`)、利率 (`APR / 12`)、毎月の支払い額 (`Payment`)、将来価値 (`FVal`)、支払い期日 (`PayType`) を指定します。

VB

```
Sub TestNPer()
    Dim TotPmts As Double
    Dim PVal, APR, Payment As Double
    Dim PayType As DueDate
    Dim Response As MsgBoxResult

    ' Usually 0 for a loan.
    Dim Fval As Double = 0
    PVal = CDbl(InputBox("How much do you want to borrow?"))
    APR = CDbl(InputBox("What is the annual percentage rate of your loan?"))
    ' Usually 0 for a loan.
    If APR > 1 Then APR = APR / 100
    Payment = CDbl(InputBox("How much do you want to pay each month?"))
    Response = MsgBox("Do you make payments at the end of month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    TotPmts = NPer(APR / 12, -Payment, PVal, FVal, PayType)
    If Int(TotPmts) <> TotPmts Then TotPmts = Int(TotPmts) + 1

    MsgBox("It will take you " & TotPmts & " months to pay off your loan.")
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

財務処理の概要

ArgumentException

NPV 関数

倍精度浮動小数点数型 (Double) の値を返します。一連の定期的なキャッシュ フロー (支払いと収益) と割引率に基づいて、投資の正味現在価値を返します。

```
Function NPV( _
   ByVal Rate As Double, _
   ByRef ValueArray() As Double _
) As Double
```

パラメータ

Rate

必ず指定します。投資期間を通じて一定の割引率を、**Double** 型の 10 進数で指定します。

ValueArray

必ず指定します。キャッシュ フローの値を**倍精度浮動小数点数型 (Double)** の配列として指定します。配列には、負の値 (支払額) と正の値 (収益額) が少なくとも 1 つずつ含まれている必要があります。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*ValueArray* が **Nothing** か *ValueArray* のランクが <> 1、または *Rate* = -1 です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

投資の正味現在価値とは、将来行われる一連の支払いと収益を現時点での現金価値に換算したものです。

NPV 関数では、配列で指定した値の順序がキャッシュ フローの順序であると見なされます。支払額と収益額を入力するときは、その順序に注意してください。

NPV 関数では、投資は配列の最初のキャッシュ フローの日付よりも 1 期前に開始され、配列の最後のキャッシュ フローで終了します。

正味現在価値は、将来のキャッシュ フローを基にして計算されます。最初のキャッシュ フローが第 1 期の期首に発生する場合、その最初の値は引数 *ValueArray* のキャッシュ フローの値に含めず、**NPV** 関数の計算結果に加算する必要があります。

NPV 関数は投資の現在価値を返す **PV** 関数によく似ていますが、**NPV** 関数と **PV** 関数の違いは、**PV** 関数ではキャッシュ フローが期首と期末のどちらで発生してもよい点にあります。また、**NPV** 関数ではキャッシュ フローの金額が一定していませんが、**PV** 関数では投資期間を通じて一定である必要があります。

使用例

NPV 関数を使用して、`values()` 配列に格納された一連のキャッシュ フローの正味現在価値を返す例は、以下のようになります。戻り値は `FixedRetRate` に格納されます。この値は固定内部利益率を表します。

VB

```
' Define money format.
Dim MoneyFmt As String = "###,##0.00"
' Define percentage format.
Dim PercentFmt As String = "#0.00"

Dim values(4) As Double
' Business start-up costs.
values(0) = -70000
' Positive cash flows reflecting income for four successive years.
values(1) = 22000
values(2) = 25000
values(3) = 28000
values(4) = 31000
```

```
' Use the NPV function to calculate the net present value.
' Set fixed internal rate.
Dim FixedRetRate As Double = 0.0625
' Calculate net present value.
Dim NetPVal As Double = NPV(FixedRetRate, values)
' Display net present value.
MsgBox("The net present value of these cash flows is " & _
    Format(NetPVal, MoneyFmt) & ".")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

IRR 関数
MIRR 関数
財務処理の概要
ArgumentException

Oct 関数

指定された値を 8 進数で返します。

```
Public Shared Function Oct( _
   ByVal Number As { Byte | SByte | Short | UShort | _
   Integer | UInteger | Long | ULong | Object } _
) As String
```

パラメータ

Number

必ず指定します。任意の有効な数式または文字列型 (**String**) の式です。

例外

例外の種類	エラー番号	条件
ArgumentNullException	5	*Number* が指定されていません。
ArgumentException	5	*Number* が数値型ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

引数 *Number* が整数でない場合は、最も近い整数に丸められてから評価されます。

Number の値	Oct の戻り値
空	ゼロ (0)
その他の数値	最大 22 桁の 8 進数

適切な範囲内の数字の前に **&O** を付けることで、8 進数を直接表すことができます。たとえば、`&O10` は、10 進数の 8 を 8 進数で表したものです。

使用例

Oct 関数を使って 8 進数の数値を返す例を次に示します。

VB

```
Dim TestOct As String
' Returns "4".
TestOct = Oct(4)
' Returns "10".
TestOct = Oct(8)
' Returns "713".
TestOct = Oct(459)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Conversion**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

変換の概要

Hex 関数 (Visual Basic)
データ型変換関数
ArgumentNullException

Partition 関数

指定された値が含まれる範囲を計算し、その範囲を表す文字列を返します。

```
Public Function Partition( _
   ByVal Number As Long, _
   ByVal Start As Long, _
   ByVal Stop As Long, _
   ByVal Interval As Long _
) As String
```

パラメータ

Number

必ず指定します。長整数型 (**Long**) です。算出された範囲のいずれかに存在する整数を指定します。

Start

必ず指定します。長整数型 (**Long**) です。計算する一連の範囲の開始点を示す整数です。*Start* に 0 以下の数字を指定することはできませんん。

Stop

必ず指定します。長整数型 (**Long**) です。計算する一連の範囲の終了点を示す整数です。*Stop* に *Start* と同じ、またはこれより小さい数字を指定することはできませんん。

Interval

必ず指定します。長整数型 (**Long**) です。*Start* と *Stop* の間にある計算される各範囲のサイズを表す整数です。*Interval* は 1 以下にはしないでください。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Start* < 0、*Stop* <= *Start*、または *Interval* < 1

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

Partition 関数は、数値の範囲を計算します。それぞれの範囲には、*Interval* で指定される値の数が含まれます。最初の範囲は *Start* から始まり、最後の範囲は *Stop* で終わります。**Partition** 関数は、名前付き引数 *Number* に指定した整数が属する範囲を算出し、その範囲を示す文字列を返します。範囲を示す文字列は、範囲内の最小値 (*lowervalue*) と最大値 (*uppervalue*) をコロンで区切って、"*lowervalue*:*uppervalue*" の形式の文字列として返されます。

必要に応じて **Partition** 関数は、*lowervalue* と *uppervalue* の先頭にスペースを挿入します。これにより最小値と最大値には、値の文字列表現と同じ文字数 (*Stop* に 1 を加えた文字数) が設定されます。スペースを挿入することで、*Number* に複数の数値を指定して **Partition** 関数を実行しても、その後の並べ替え処理で適切に文字列を処理できます。

名前付き引数の *Start*、*Stop*、*Interval* の値の 3 とおりの組み合わせを使って、その結果得られる範囲の文字列の例を次の表に示します。"最初の範囲" と "最後の範囲" の列は、指定された *Start* と *Stop* の値で返す最大値と最小値の範囲を表しています。"最初の範囲より前" と "最後の範囲より後" の列はそれぞれ、*Start* より小さい *Number* の値および *Stop* より大きい *Number* の値に対して返される文字列を表しています。

Start	***Stop***	***Interval***	最初の範囲より前	最初の範囲	最後の範囲	最後の範囲より後
0	99	5	" : -1"	" 0: 4"	" 95: 99"	"100: "
20	199	10	" : 19"	" 20: 29"	"190:199"	"200: "
100	1010	20	" : 99"	" 100: 119"	"1000:1010"	"1011: "

この表の 3 行目には、名前付き引数 *Start* と名前付き引数 *Stop* で定義される数列が名前付き引数 *Interval* の値で等分できない場合の結果が示されています。名前付き引数 *Interval* の値は 20 ですが、最後の範囲は名前付き引数 *Stop* に指定された整数値で終わっているため、その長さは 11 になっています。

Interval が 1 の場合、*Start* および *Stop* 引数にかかわらず、範囲は "*Number*:*Number*" になります。たとえば、名前付き引数 *Number* に 267 を指定し、名前付き引数 *Stop* に 1,000 を指定した場合に、名前付き引数 *Interval* に 1 を指定すると、**Partition** は "267:267" という範囲を返します。

Partition は、データベース クエリの作成に役立ちます。たとえば、請求書番号の各範囲 (1 ～ 1,000、1,001 ～ 2,000 など) に含まれる注文の数を示す SELECT クエリの作成が可能です。

使用例

次の例では、1950 年から 2049 年まで、10 年単位で範囲を設定します。`year` の値を適切な範囲に割り当て、範囲を示す**String** の値を返します。たとえば、`year` の値が 1984 の場合、**Partition** は "1980:1989" を返します。

VB

```
Dim year As Long = 1984
' Assume the value of year is provided by data or by user input.
Dim decade As String
decade = Partition(year, 1950, 2049, 10)
MsgBox("Year " & CStr(year) & " is in decade " & decade & ".")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

その他の技術情報

カテゴリから検索 (SQL Server)

Pmt 関数

倍精度浮動小数点数型 (Double) の値を返します。定額の支払いを定期的に行い、利率が一定であると仮定して、投資に必要な定期支払額を返します。

```
Function Pmt( _
   ByVal Rate As Double, _
   ByVal NPer As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

パラメータ

Rate

必ず指定します。投資期間を通じて一定の利率を示す倍精度浮動小数点型 (**Double**) の値です。たとえば、年利 (APR) 10% の車のローンを月払いで返済する場合、各期間の利率は 0.1/12 または 0.0083 になります。

NPer

必ず指定します。投資期間全体での支払い回数の合計を示す倍精度浮動小数点数型 (**Double**) の値です。たとえば、4 年の車のローンを月払いで返済する場合、支払い回数は 48 回 (12 回 × 4 年) になります。

PV

必ず指定します。現在の投資額、つまり将来行われる一連の支払いを現時点で一括支払いした場合の合計金額を示す倍精度浮動小数点型 (**Double**) の値を指定します。たとえば、車を購入するために資金を借りる場合、現在価値は毎月支払うローンの総額になります。

FV

省略可能。投資の将来価値、つまり最後の支払いを行った後に残る現金の収支を示す倍精度浮動小数点数型 (**Double**) の値。たとえば、ローンなどの借入の将来価値は 0 になります。また、子供の教育費用として 18 年間で 500,000 円貯蓄する場合は、将来価値が 500,000 円になります。このパラメータを省略すると、0 を指定したものと見なされます。

Due

省略可能です。支払期日を示すオブジェクト型 DueDate 列挙型 の値を指定します。各期の期末に支払う場合は `DueDate.EndOfPeriod` を、各期の期首に支払う場合は `DueDate.BegOfPeriod` をそれぞれ引数に指定します。この引数を省略すると、`DueDate.EndOfPeriod` を指定したものと見なされます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*NPer* = 0.

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

投資とは、一連の定額の支払いを一定の期間行うことです。たとえば、住宅ローンなどのローンまたは毎月の貯蓄プランなどの出資を指します。

引数 *Rate* および *NPer* は、単位が同じ支払い期日を使用して計算する必要があります。たとえば、*Rate* を月単位で計算する場合は、*NPer* も月単位で計算する必要があります。

出金 (定額預金の支払いなど) を表す引数には負の値を指定し、入金 (配当金など) を表す引数には正の値を指定します。

使用例

次の例は、**Pmt** 関数を使って、一定期間のローンに対する毎月の支払い額を計算します。利率 (`APR / 12`)、支払い回数 (`TotPmts`)、現在価値または元金 (`PVal`)、将来価値 (`FVal`)、支払い期日 (`PayType`) を指定します。

VB

```
Sub TestPMT()
    Dim PVal, APR, Payment, TotPmts As Double
    Dim PayType As DueDate
    Dim Response As MsgBoxResult

    ' Define money format.
    Dim Fmt As String = "###,###,##0.00"
    ' Usually 0 for a loan.
    Dim FVal As Double = 0
    PVal = CDbl(InputBox("How much do you want to borrow?"))
    APR = CDbl(InputBox("What is the annual percentage rate of your loan?"))
    If APR > 1 Then APR = APR / 100 ' Ensure proper form.
    TotPmts = CDbl(InputBox("How many monthly payments will you make?"))
    Response = MsgBox("Do you make payments at the end of month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    Payment = Pmt(APR / 12, TotPmts, -PVal, FVal, PayType)

    MsgBox("Your payment will be " & Format(Payment, Fmt) & " per month.")
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
財務処理の概要
ArgumentException

PPmt 関数

倍精度浮動小数点数型 (**Double**) の値を返します。定額の支払いを定期的に行い、利率が一定であると仮定して、指定した期に支払われる元金を返します。

```
Function PPmt( _
   ByVal Rate As Double, _
   ByVal Per As Double, _
   ByVal NPer As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

パラメータ

Rate

必ず指定します。投資期間を通じて一定の利率を示す倍精度浮動小数点数型 (**Double**) の値を指定します。たとえば、年利 (APR) 10% の車のローンを月払いで返済する場合、各期間の利率は 0.1/12 または 0.0083 になります。

Per

必ず指定します。金利支払額を求める期を示す倍精度浮動小数点数型 (**Double**) の値を 1 ～ *NPer* の範囲で指定します。

NPer

必ず指定します。投資期間全体での支払い回数の合計を示す倍精度浮動小数点数型 (**Double**) の値です。たとえば、4 年の車のローンを月払いで返済する場合、支払い回数は 48 回 (12 回×4 年) になります。

PV

必ず指定します。今後の一連の支払いまたは収益の現在の値を示す倍精度浮動小数点型 (**Double**) の値です。たとえば、車を購入するために資金を借りる場合、現在価値は毎月支払うローンの総額になります。

FV

省略可能。投資の将来価値、つまり最後の支払いを行った後に残る現金の収支を示す倍精度浮動小数点数型 (**Double**) の値。たとえば、ローンなどの借入の将来価値は 0 になります。また、子供の教育費用として 18 年間で 500,000 円貯蓄する場合は、将来価値が 500,000 円になります。このパラメータを省略すると、0 を指定したものと見なされます。

Due

省略可能です。支払期日を示すオブジェクト型 DueDate 列挙型 の値を指定します。各期の期末に支払う場合は `DueDate.EndOfPeriod` を、各期の期首に支払う場合は `DueDate.BegOfPeriod` をそれぞれ引数に指定します。この引数を省略すると、`DueDate.EndOfPeriod` を指定したものと見なされます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Per* <=0 または *Per* > *NPer* です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

投資とは、一連の定額の支払いを一定の期間行うことです。たとえば、住宅ローンなどのローンまたは毎月の貯蓄プランなどの出資を指します。

引数 *Rate* および *NPer* は、単位が同じ支払い期日を使用して計算する必要があります。たとえば、*Rate* を月単位で計算する場合は、*NPer* も月単位で計算する必要があります。

出金 (定額預金の支払いなど) を表す引数には負の値を指定し、入金 (配当金など) を表す引数には正の値を指定します。

使用例

次の例は、**PPmt** 関数を使って、毎月の支払いが同額である場合の特定の期間における元金を計算します。利率 (`APR / 12`)、支払い元金を求める期 (`Period`)、支払い回数 (`TotPmts`)、現在価値または元金 (`PVal`)、将来価値 (`FVal`)、支払い期日 (`PayType`) を指定します。

VB

```
Sub TestPPMT()
    Dim PVal, APR, TotPmts, Payment, Period, P, I As Double
    Dim PayType As DueDate
    Dim Msg As String
    Dim Response As MsgBoxResult

    ' Define money format.
    Dim Fmt As String = "###,###,##0.00"
    ' Usually 0 for a loan.
    Dim Fval As Double = 0
    PVal = CDbl(InputBox("How much do you want to borrow?"))
    APR = CDbl(InputBox("What is the annual percentage rate of your loan?"))
    ' Ensure proper form.
    If APR > 1 Then APR = APR / 100
    TotPmts = CDbl(InputBox("How many monthly payments do you have to make?"))
    Response = MsgBox("Do you make payments at the end of month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    Payment = Math.Abs(-Pmt(APR / 12, TotPmts, PVal, FVal, PayType))
    Msg = "Your monthly payment is " & Format(Payment, Fmt) & ". "
    Msg = Msg & "Would you like a breakdown of your principal and "
    Msg = Msg & "interest per period?"
    ' See if chart is desired.
    Response = MsgBox(Msg, MsgBoxStyle.YesNo)
    If Response <> MsgBoxResult.No Then
        If TotPmts > 12 Then MsgBox("Only first year will be shown.")
        Msg = "Month  Payment  Principal  Interest" & vbNewLine
        For Period = 1 To TotPmts
            ' Show only first 12.
            If Period > 12 Then Exit For
            P = PPmt(APR / 12, Period, TotPmts, -PVal, FVal, PayType)
            ' Round principal.
            P = (Int((P + 0.005) * 100) / 100)
            I = Payment - P
            ' Round interest.
            I = (Int((I + 0.005) * 100) / 100)
            Msg = Msg & Period & vbTab & Format(Payment, Fmt)
            Msg = Msg & vbTab & Format(P, Fmt) & vbTab & Format(I, Fmt) & vbNewLine
        Next Period
        ' Display amortization table.
        MsgBox(Msg)
    End If
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

財務処理の概要

ArgumentException

Print 関数、PrintLine 関数

シーケンシャル出力モードで開いたファイルにデータを書き込むファイル入出力関数です。

```
Public Sub Print( _
   ByVal FileNumber As Integer, _
   ByVal ParamArray Output() As Object _
)
' -or-
Public Sub PrintLine( _
   ByVal FileNumber As Integer, _
   ByVal ParamArray Output() As Object _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号です。

Output

省略可能です。ファイルに書き込む式。ファイルが複数の場合はコンマで区切ります。

引数 *Output* には、次の値を指定します。

設定	説明
SPC(*n* **)**	出力するデータに *n* 個の空白を挿入します。
TAB(*n* **)**	絶対桁数に対する出力位置を *n* 桁目に移動します。**TAB** 関数を引数なしで使用すると、出力位置は次の印字領域の先頭になります。
expression	出力する数式または文字列式を指定します。

例外

例外の種類	エラー番号	条件
IOException	54	ファイル モードが無効です。
IOException	52	*FileNumber* が存在していません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

Print および **PrintLine** 関数は、下位互換性を保つために提供されており、パフォーマンスに影響を与える可能性があります。非レガシ アプリケーションに対しては、**My.Computer.FileSystem** オブジェクトはより優れたパフォーマンスを発揮します。詳細については、「Visual Basic におけるファイル アクセス」を参照してください。

Print は、行末にライン フィードを含めませんが、**PrintLine** はライン フィードを含めます。

通常、**Print** 関数を使用して書き込んだデータは、**LineInput** または **Input** 関数で読み込みます。

PrintLine 関数の引数 *Output* を省略すると、空行がファイルに出力されます。**Print** 関数でこの引数を省略すると、何も出力されません。複数の式をコンマで区切って指定するとタブ境界に合わせて配置されますが、コンマと **TAB** が混在していると結果が乱れる可能性があります。

ブール型 (**Boolean**) のデータは、`True` または `False` という文字列で出力されます。キーワード **True** および **False** はロケールに応じて翻訳されることはありません。

日付型のデータは、コントロール パネルで設定した短い形式で書き込まれます。日付や時間の構成要素がない場合やゼロの場合は、指定されている部分だけのデータがファイルに書き込まれます。

引数 *Output* のデータが空の場合は、ファイルには何も書き込まれません。ただし引数 *Output* のリスト データが **DBNull** の場合は、`Null` という文字列がファイルに書き込まれます。

Error データは、文字列 `Error errorcode` という形式で出力されます。キーワード **Error** はロケールに応じて翻訳されることはありません。

Print 関数を使用して書き込んだデータは、すべて国別情報に対応しています。書き込まれたデータは、適切な小数点記号を使用して書式設定されます。複数のロケールで使用できるようにデータを出力するには、**Write** 関数を使用します。

Print または **PrintLine** 関数を使用してファイルにデータを書き込むには、**FileIOPermissionAccess** 列挙体からの **Write** アクセスが必要です。詳細については、「FileIOPermissionAccess 列挙体」を参照してください。

使用例

Print および **PrintLine** 関数を使って、データをファイルに書き込む例を次に示します。

VB

```
FileOpen(1, "c:\trash.txt", OpenMode.Output)   ' Open file for output.
Print(1, "This is a test.")   ' Print text to file.
PrintLine(1)   ' Print blank line to file.
PrintLine(1, "Zone 1", TAB(), "Zone 2")   ' Print in two print zones.
PrintLine(1, "Hello", "World")   ' Separate strings with a tab.
PrintLine(1, SPC(5), "5 leading spaces ")   ' Print five leading spaces.
PrintLine(1, TAB(10), "Hello")   ' Print word at column 10.

' Assign Boolean, Date, and Error values.
Dim aBool As Boolean
Dim aDate As DateTime
aBool = False
aDate = DateTime.Parse("February 12, 1969")

' Dates and booleans are translated using locale settings of your system.
PrintLine(1, aBool, " is a Boolean value")
PrintLine(1, aDate, " is a date")
FileClose(1)   ' Close file.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でテキストをファイルに書き込む

方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

関連項目

FileOpen 関数

SPC 関数

TAB 関数

Write 関数、WriteLine 関数

その他の技術情報

Visual Basic におけるファイル アクセス

PV 関数

倍精度浮動小数点数型 (Double) の値を返します。定額の支払いを定期的に行い、利率が一定であると仮定して、投資の現在価値を返します。

```
Function PV( _
   ByVal Rate As Double, _
   ByVal NPer As Double, _
   ByVal Pmt As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod _
) As Double
```

パラメータ

Rate

必ず指定します。投資期間を通じて一定の利率を示す倍精度浮動小数点数型 (**Double**) の値を指定します。たとえば、年利 (APR) 10% の車のローンを月払いで返済する場合、各期間の利率は 0.1/12 または 0.0083 になります。

NPer

必ず指定します。投資期間全体での支払い回数の合計を示す倍精度浮動小数点数型 (**Double**) の値です。たとえば、4 年の車のローンを月払いで返済する場合、支払い回数の合計は 48 回 (12 回×4 年) になります。

Pmt

必ず指定します。毎回の支払額を示す倍精度浮動小数点数型 (**Double**) の値を指定します。通常、支払額には元金と利息が含まれます。支払額を投資期間内に変更することはできません。

FV

省略可能です。投資の将来価値、つまり最後の支払いを行った後に残る現金の収支を示す倍精度浮動小数点数型 (**Double**) の値を指定します。たとえば、ローンなどの借入の将来価値は 0 になります。また、子供の教育費用として 18 年間で 500,000 円貯蓄する場合は、将来価値が 500,000 円になります。このパラメータを省略すると、0 を指定したものと見なされます。

Due

省略可能です。支払期日を示すオブジェクト型 DueDate 列挙型 の値を指定します。各期の期末に支払う場合は `DueDate.EndOfPeriod` を、各期の期首に支払う場合は `DueDate.BegOfPeriod` をそれぞれ引数に指定します。この引数を省略すると、`DueDate.EndOfPeriod` を指定したものと見なされます。

解説

投資とは、一連の定額の支払いを一定の期間行うことです。たとえば、住宅ローンなどのローンまたは毎月の貯蓄プランなどの出資を指します。

引数 *Rate* および *NPer* は、単位が同じ支払い期日を使用して計算する必要があります。たとえば、*Rate* を月単位で計算する場合は、*NPer* も月単位で計算する必要があります。

出金 (定額預金の支払いなど) を表す引数には負の値を指定し、入金 (配当金など) を表す引数には正の値を指定します。

使用例

次の例は、**PV** 関数を使って、以後 20 年間にわたって、毎月 5 万円の配当が支払われる証券 (100 万円) の現在価値を計算します。年率 (`APR`)、支払い回数 (`TotPmts`)、支払い金額 (`YrIncome`)、投資の将来価値 (`FVal`)、支払い期日 (`PayType`) を指定します。`YrIncome` は出金を表すので、負の数を指定しています。

VB

```
Sub TestPV()
    ' Define money format.
    Dim Fmt As String = "###,##0.00"
    ' Annual percentage rate.
    Dim APR As Double = 0.0825
    ' Total number of payments.
    Dim TotPmts As Double = 20
    ' Yearly income.
```

```
    Dim YrIncome As Double = 50000
    ' Future value.
    Dim FVal As Double = 1000000
    ' Payment at beginning of month.
    Dim PayType As DueDate = DueDate.BegOfPeriod
    Dim PVal As Double = PV(APR, TotPmts, -YrIncome, FVal, PayType)
    MsgBox("The present value is " & Format(PVal, Fmt) & ".")
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

財務処理の概要

QBColor 関数

指定された色番号に対応する、RGB カラー コードを表す整数型 (**Integer**) の値を返します。

```
QBColor(Color)
```

パラメータ

Color

必ず指定します。0 ～ 15 の範囲内の整数を指定します。

設定

引数 *Color* の設定値は次のとおりです。

数値	色	数値	色
0	黒	8	灰色
1	青	9	明るい青
2	緑	10	明るい緑
3	シアン	11	明るいシアン
4	赤	12	明るい赤
5	マゼンタ	13	明るいマゼンタ
6	黄	14	明るい黄
7	白	15	明るい白

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Color* が 0 ～ 15 (両端を含む) の範囲内の値ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

戻り値は、最下位バイトから赤、緑、青の値を指定します。これらの値を使って、Visual Basic 言語で使用される RGB システムにそれぞれの色を設定します。

使用例

QBColor 関数を使用して、`colorInteger` で指定された色に変更する例を次に示します。**QBColor** には 0 ～ 15 の整数値を指定できます。

VB

```
Dim colorInteger As Integer
' Use 4 for red.
colorInteger = QBColor(4)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
RGB 関数 (Visual Basic)
ArgumentException Class

Randomize 関数 (Visual Basic)

乱数ジェネレータを初期化 (乱数系列を再設定) します。

```
Public Shared Sub Randomize ([ Number ])
```

パラメータ
Number

省略可能です。**Object** または任意の有効な数値式です。

解説

Randomize は *Number* を使用して、**Rnd** 関数の乱数ジェネレータに新しいシード値を指定して初期化します。*Number* を省略した場合、システム タイマから取得した値が新しいシード値として使われます。

Randomize ステートメントを使用しない場合、引数を指定しないで **Rnd** 関数を呼び出すと、最初にこの関数を呼び出したときのシード値と同じ値が使用されます。それ以降は、直前に生成された数がシード値として使用されます。

メモ :

乱数系列を繰り返すには、数値を指定して **Randomize** ステートメントを実行する直前に、負の引数を指定して **Rnd** 関数を呼び出します。*Number* に同じ値を指定して **Randomize** ステートメントを使用しても、前の乱数系列を繰り返すことはできません。

セキュリティに関するメモ :

Random ステートメントと **Rnd** 関数の開始にはシード値が使用され、有限の範囲で減少する値が生成されるため、それらのアルゴリズムを知っている人には結果が予測できる場合があります。このため、暗号化に使用する乱数の生成には、**Random** ステートメントと **Rnd** 関数を使用しないでください。詳細については、「RandomNumberGenerator」を参照してください。

使用例

Randomize ステートメントを使って乱数ジェネレータを初期化する例を次に示します。引数 number を省略したので、**Randomize** ステートメントは新しいシード値として **Timer** 関数からの戻り値を使用します。

VB

```
' Initialize the random-number generator.
Randomize()
' Generate random value between 1 and 6.
Dim value As Integer = CInt(Int((6 * Rnd()) + 1))
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **VBMath**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
数値演算の概要
Rnd 関数 (Visual Basic)

Rate 関数

倍精度浮動小数点数型 (Double) の値を返します。投資期間を通じての利率を返します。

```
Function Rate( _
   ByVal NPer As Double, _
   ByVal Pmt As Double, _
   ByVal PV As Double, _
   Optional ByVal FV As Double = 0, _
   Optional ByVal Due As DueDate = DueDate.EndOfPeriod, _
   Optional ByVal Guess As Double = 0.1 _
) As Double
```

パラメータ

NPer

必ず指定します。投資期間全体での支払い回数の合計を示す倍精度浮動小数点数型 (**Double**) の値です。たとえば、4 年の車のローンを月払いで返済する場合、支払い回数は 48 回 (12 回×4 年) になります。

Pmt

必ず指定します。毎回の支払額を示す倍精度浮動小数点数型 (**Double**) の値です。通常、支払額には元金と利息が含まれます。支払額を投資期間内に変更することはできません。

PV

必ず指定します。現在の投資額、つまり将来行われる一連の支払いや収益を現時点で一括した場合の合計金額を示す倍精度浮動小数点数型 (**Double**) の値を指定します。たとえば、車を購入するために資金を借りる場合、現在価値は毎月支払うローンの総額になります。

FV

省略できます。投資の将来価値、つまり最後の支払いを行った後に残る現金の収支を示す倍精度浮動小数点数型 (**Double**) の値を指定します。たとえば、ローンなどの借入の将来価値は 0 になります。また、子供の教育費用として 18 年間で 500,000 円貯蓄する場合は、将来価値が 500,000 円になります。このパラメータを省略すると、0 を指定したものと見なされます。

Due

省略可能です。支払期日を示すオブジェクト型 DueDate 列挙型 の値を指定します。各期の期末に支払う場合は `DueDate.EndOfPeriod` を、各期の期首に支払う場合は `DueDate.BegOfPeriod` をそれぞれ引数に指定します。この引数を省略すると、`DueDate.EndOfPeriod` を指定したものと見なされます。

Guess

省略できます。**Rate** によって返される値の推定値を倍精度浮動小数点数型 (**Double**) で指定します。省略すると、*Guess* に 0.1 (10%) を指定したものと見なされます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*NPer* <= 0.

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

投資とは、一連の定額の支払いを一定の期間行うことです。たとえば、住宅ローンなどのローンまたは毎月の貯蓄プランなどの出資を指します。

出金 (定額預金の支払いなど) を表す引数には負の値を指定し、入金 (配当金など) を表す引数には正の値を指定します。

Rate 関数は、反復計算の手法を用いて利率を計算します。引数 *Guess*、**Rate** の値を初期値とし、計算結果の誤差が 0.00001% 以内になるまで反復計算を行います。**Rate** が反復計算を 20 回行っても適切な解が見つからない場合は、エラーとなります。推定値に 10% を指定した場合に **Rate** がエラーになったときは、引数 *Guess* の値を変えてみてください。

使用例

次の例は、**Rate** 関数を使って、実際の利率を計算します。支払い回数 (`TotPmts`)、支払い額 (`Payment`)、現在価値または元金 (`PVal`)、将来価値 (`FVal`)、支払い期日 (`PayType`)、利率の推定値 (`Guess`) を指定します。

VB

```
Sub TestRate()
    Dim PVal, Payment, TotPmts, APR As Double
    Dim PayType As DueDate

    ' Define percentage format.
    Dim Fmt As String = "##0.00"
    Dim Response As MsgBoxResult
    ' Usually 0 for a loan.
    Dim FVal As Double = 0
    ' Guess of 10 percent.
    Dim Guess As Double = 0.1
    PVal = CDbl(InputBox("How much did you borrow?"))
    Payment = CDbl(InputBox("What's your monthly payment?"))
    TotPmts = CDbl(InputBox("How many monthly payments do you have to make?"))
    Response = MsgBox("Do you make payments at the end of the month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    APR = (Rate(TotPmts, -Payment, PVal, FVal, PayType, Guess) * 12) * 100

    MsgBox("Your interest rate is " & Format(CInt(APR), Fmt) & " percent.")
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

財務処理の概要

ArgumentException

Rename 関数

ディスク ファイルまたはディレクトリの名前を変更します。

My 機能を使用すると、**Rename** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Sub Rename( _
   ByVal OldPath As String, _
   ByVal NewPath As String _
)
```

パラメータ

OldPath

必ず指定します。既存のファイル名と場所を指定する **String** 式です。*OldPath* にはファイルのディレクトリ名やドライブ名を指定できます。

NewPath

必ず指定します。新しいファイル名と場所を指定する **String** 式です。*NewPath* には移動先の場所のディレクトリ名やドライブ名を指定できます。*NewPath* に既存のファイル名を指定することはできません。

例外

例外の種類	エラー番号	条件
ArgumentException	5	パスが無効です。
FileNotFoundException	53	*OldPath* ファイルが存在しません。
IOException	58	*NewPath* ファイルが既に存在します。
IOException	75	アクセスが無効です。
IOException	74	異なるデバイスには名前を変更できません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Rename 関数はファイルの名前を変更し、必要ならば別のディレクトリに移動します。**Rename** 関数はファイルを別のドライブに移動できますが、*NewPath* と *OldPath* が同じドライブに存在する場合は名前の変更だけを行います。**Rename** を使って新しいファイルやディレクトリを作成することはできません。

開いているファイルに対して **Rename** 関数を実行すると、エラーが発生します。名前を変更する前にファイルを閉じる必要があります。**Rename** 関数の引数には、複数文字 (*) および単一文字 (?) のワイルドカードを含めることはできません。

セキュリティに関するメモ：
Rename を使って、保護されていない場所から保護された場所にファイルをコピーすると、ファイルは制限の少ない方の権限を保持することになります。これによってセキュリティ上の危険を負うことのないよう、確実にチェックする必要があります。

使用例

Rename 関数を使って、ファイルの名前を変更するコード例を次に示します。この例では、指定したディレクトリが既に存在すると仮定します。

VB

```
Dim OldName, NewName As String
OldName = "OLDFILE"
' Define file names.
NewName = "NEWFILE"
```

```
' Rename file.
Rename(OldName, NewName)

OldName = "C:\OLDDIR\OLDFILE"
NewName = "C:\NEWDIR\NEWFILE"
' Move and rename file.
Rename(OldName, NewName)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でファイルの名前を変更する

方法 : Visual Basic でディレクトリの名前を変更する

関連項目

Kill 関数

ArgumentException Class

FileNotFoundException Class

IOException Class

Replace 関数 (Visual Basic)

指定された文字列の一部を指定された回数分別の部分文字列で置換した文字列を返します。

```
Public Function Replace(
   ByVal Expression As String,
   ByVal Find As String,
   ByVal Replacement As String,
   Optional ByVal Start As Integer = 1,
   Optional ByVal Count As Integer = -1,
   Optional ByVal Compare As CompareMethod = CompareMethod.Binary
) As String
```

パラメータ

Expression

必ず指定します。置換する部分文字列を含む文字列式を指定します

Find

必ず指定します。検索する部分文字列を指定します。

Replacement

必ず指定します。置換する部分文字列を指定します。

Start

(省略可能。) *Expression* 内の部分文字列の検索開始位置を指定します。このパラメータを省略すると、1 を指定したものと見なされます。

Count

(省略可能。) 置換する部分文字列数を指定します。このパラメータを省略すると、既定値の -1 が使用され、すべての候補が置換されます。

Compare

(省略可能。) 部分文字列を評価するときに使用する文字列比較のモードを表す数値を指定します。値については、「設定」を参照してください。

設定

引数 *Compare* の設定値は次のとおりです。

定数	説明
Binary	バイナリ モードで比較を行います。
Text	テキスト モードで比較を行います。

戻り値

Replace 関数の戻り値は次のとおりです。

条件	Replace の戻り値
Find の長さが 0、または **Nothing**	*Expression*
Replace の長さが 0	*Find* の出現しない *Expression*
Expression の長さが 0 か **Nothing**、または *Start* の値が *Expression* の長さよりも大きい	**Nothing**
Count が 0 のとき	*Expression*

例外

例外の種類	エラー番号	条件

ArgumentException	5	*Count* < -1 または *Start* <= 0 です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Replace 関数の戻り値は、文字列 *Expression* を *Find* と *Replace* の値に指定された方法で置換処理した、*Start* の位置から末尾までの文字列です。

使用例

Replace 関数の使用例を次に示します。

```
Dim TestString As String = "Shopping List"
' Returns "Shipping List".
Dim aString As String = Replace(TestString, "o", "i")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
Filter 関数 (Visual Basic)
ArgumentException

Reset 関数

FileOpen 関数で開いたすべてのファイルを閉じます。

My 機能を使用すると、**Reset** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Sub Reset()
```

解説

Reset 関数は、**FileOpen** 関数で開かれたすべてのアクティブなファイルを閉じます。**FileClose()** 関数にパラメータを指定しない場合と同じ機能です。

使用例

次の例は、**Reset** 関数を使って、開いているファイルをすべて閉じて、ファイル バッファに残っている内容をディスクに書き込みます。この例では、オブジェクト型 (**Object**) の変数 `FileNumber` を文字列と数値の両方の変数として使っています。

VB

```
' Open 5 files named TEST1, TEST2, etc.
Dim fileNumber As Integer
' Open 5 files.
For fileNumber = 1 To 5
   FileOpen(fileNumber, "TEST" & fileNumber, OpenMode.Output)
   PrintLine(fileNumber, "Hello World")
Next fileNumber
' Close files and write contents to disk.
Reset()
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileClose 関数
End ステートメント
FileOpen 関数

RGB 関数 (Visual Basic)

RGB カラー値を表す整数型 (**Integer**) の値を返します。この値は、色の赤、緑、青の成分を示します。

```
RGB( _
      Red As Integer, _
    Green As Integer, _
      Blue As Integer _
)
```

パラメータ

Red

必ず指定します。目的の色に含まれる赤の成分の強度を表す 0 ～ 255 の範囲内の整数型 (**Integer**) の値を指定します。

Green

必ず指定します。目的の色に含まれる緑の成分の強度を表す 0 ～ 255 の範囲内の整数型 (**Integer**) の値を指定します。

Blue

必ず指定します。目的の色に含まれる青の成分の強度を表す 0 ～ 255 の範囲内の整数型 (**Integer**) の値を指定します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Green*、*Blue*、または *Red* が、0 ～ 255 (両端を含む) の範囲内の値ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

色の指定を受け取るメソッドやプロパティは、RGB カラー値を表す数値として色を受け取ります。RGB カラー値は、赤、緑、および青の相対的な明度を指定することによって色を表示します。

RGB 関数への引数が 255 を超えている場合は、255 が使用されます。

標準色と、それに含まれる赤、緑、および青の値は、次のとおりです。

色	Red 値	Green 値	Blue 値
黒	0	0	0
青	0	0	255
緑	0	255	0
シアン	0	255	255
赤	255	0	0
マゼンタ	255	0	255
黄	255	255	0
白	255	255	255

使用例

RGB 関数を使って、**RGB** カラー値を表す整数を取得するコード例を次に示します。

VB

```
Dim red, rgbValue As Integer
Dim i As Integer = 75
' Return the value for red.
red = RGB(255, 0, 0)
' Same as RGB(75, 139, 203).
rgbValue = RGB(i, 64 + i, 128 + i)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
QBColor 関数
ArgumentException Class

Right 関数 (Visual Basic)

文字列の右端から指定された文字数分の文字列を返します。

```
Public Shared Function Right( _
   ByVal str As String, _
   ByVal Length As Integer _
) As String
```

パラメータ

str

必ず指定します。文字列型 (**String**) の式です。この左端から文字列が取り出されます。

Length

必ず指定します。**Integer** です。取り出す文字列の文字数を示す数式を指定します。0 を指定した場合は、長さ 0 の文字列 ("") を返します。引数 *str* の文字数以上の場合は、文字列全体が返されます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Length* < 0.

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

str の文字数を確認するには、**Len** 関数を使用します。Windows フォームで使用された場合や、**Right** プロパティを持つクラスで使用された場合、関数を **Microsoft.VisualBasic.Right** で完全修飾する必要があります。

メモ :

以前のバージョンの Visual Basic では、**RightB** 関数は文字数ではなくバイト数に基づいて文字列を返していました。これは主に、2 バイト文字セット (DBCS) アプリケーションで文字列を変換するために使用します。現在 Visual Basic のすべての文字列は Unicode で、**RightB** はサポートされていません。

使用例

Right 関数を使って、指定された **String** の部分文字列を返す例を次に示します。**Right** プロパティを持つクラスでは、**Right** 関数を完全修飾しなければならないことがあります。

VB

```
Dim TestString As String = "Hello World!"
' Returns "World!".
Dim subString As String = Microsoft.VisualBasic.Right(TestString, 6)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
Left 関数 (Visual Basic)
Len 関数 (Visual Basic)

Mid 関数 (Visual Basic)
ArgumentException
概念
プログラミング要素のサポートに関する変更の概要

RmDir 関数

既存のディレクトリを削除します。

My 機能を使用すると、**RmDir** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem.DeleteDirectory メソッド」を参照してください。

```
Public Sub RmDir(ByVal Path As String)
```

パラメータ

Path

必ず指定します。削除するディレクトリまたはフォルダを表す文字列 (**String**) 式を指定します。*Path* にはドライブ名を含めることができます。ドライブを指定しない場合、**RmDir** 関数は現在のドライブ上のディレクトリを削除します。

例外

例外の種類	エラー番号	条件
ArgumentException	52	*Path* が指定されていません。または空です。
IOException	75	対象のディレクトリにファイルが存在しています。
FileNotFoundException	76	ディレクトリが存在していません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

ファイルが含まれるディレクトリに対して **RmDir** 関数を実行しようとすると、エラーが発生します。ディレクトリを削除する前に、**Kill** 関数を使ってすべてのファイルを削除してください。

使用例

RmDir 関数を使って、既存のディレクトリを削除するコード例を次に示します。

VB

```
' Assume that MYDIR is an empty directory.
' Remove MYDIR.
RmDir("MYDIR")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でディレクトリを削除する

関連項目

ChDir 関数
CurDir 関数
Kill 関数
MkDir 関数

ArgumentException Class
IOException Class
FileNotFoundException Class

Rnd 関数 (Visual Basic)

単精度浮動小数点数型 (Single) の乱数を返します。

```
Public Shared Function Rnd[(Number)] As Single
```

パラメータ

Number

省略可能です。単精度浮動小数点数型 (**Single**) の値、または任意の有効な単精度浮動小数点数型 (**Single**) の式です。

戻り値

number の値	**Rnd による生成結果**
number < 0	常に、*Number* のシード値によって決まる同じ数値を返します。
number > 0	乱数系列の次の乱数を返します。
number = 0	直前に生成した乱数を返します。
省略したとき	乱数系列の次の乱数を返します。

解説

Rnd 関数は 0 以上、1 未満の範囲の値を返します。

Number の値によって、**Rnd** 関数が返す乱数が決まります。

初期シード値が変わらない限り、一連の **Rnd** 関数が返す乱数系列は同じになります。これは、連続する **Rnd** 関数が乱数系列の中の直前に生成された乱数をシード値として、次の乱数をそれぞれ生成するためです。

システム タイマから取得した新しいシード値を使って、乱数ジェネレータを初期化するには、**Rnd** 関数を呼び出す前に、引数を指定せずに **Randomize** ステートメントを実行してください。

任意の範囲の整数の乱数を生成するには、次の式を使用してください。

VB

```
randomvalue = CInt(Int((upperbound - lowerbound + 1) * Rnd() + lowerbound))
```

ここでは、`upperbound` が範囲内の最大数で、`lowerbound` が範囲内の最小数です。

メモ：

乱数系列を繰り返すには、数値を指定して **Randomize** ステートメントを実行する直前に、負の引数を指定して **Rnd** 関数を呼び出します。*Number* に同じ値を指定して **Randomize** ステートメントを使用しても、前の乱数系列を繰り返すことはできません。

セキュリティに関するメモ：

Random ステートメントと **Rnd** 関数の開始にはシード値が使用され、有限の範囲で減少する値が生成されるため、それらのアルゴリズムを知っている人には結果が予測できる場合があります。このため、暗号化に使用する乱数の生成には、**Random** ステートメントと **Rnd** 関数を使用しないでください。

使用例

Rnd 関数を使って、1 ～ 6 の範囲でランダムな整数を生成する例を次に示します。

VB

```
' Initialize the random-number generator.
Randomize()
' Generate random value between 1 and 6.
```

```
Dim value As Integer = CInt(Int((6 * Rnd()) + 1))
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **VBMath**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Randomize 関数 (Visual Basic)
数値演算の概要
Randomize 関数 (Visual Basic)

RSet 関数

文字列と長さが指定され、その長さに調整された文字列右揃えにして文字列を返します。

```
Public Shared Function RSet( _
       ByVal Source As String, _
       ByVal Length As Integer _
) As String
```

パラメータ

Source

必ず指定します。文字列 (**String**) 式を指定します。調整される文字列。

Length

必ず指定します。整数 (**Integer**) 式を指定します。返される文字列の長さです。

解説

Source が *Length* より長い場合、**RSet** 関数は、文字列の先頭から *Length* の長さの文字だけを返します。*Source* が *Length* より短い場合、**RSet** は文字列の左にスペースを追加し、適切な長さにします。

使用例

RSet 関数の使用例を次に示します。

VB

```
Dim TestString As String = "Right"
' Returns "      Right"
Dim rString As String = RSet(TestString, 11)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
データ型の概要 (Visual Basic)
LSet 関数

概念

データ型の有効な使用方法

Trim、LTrim、RTrim 関数

指定された文字列から先頭のスペースを除いたコピー (**LTrim**)、後続のスペースを除いたコピー (**RTrim**)、または先頭と後続のスペースを除いたコピー (**Trim**) を含む文字列を返します。

```
Public Shared Function LTrim(ByVal str As String) As String
Public Shared Function RTrim(ByVal str As String) As String
Public Shared Function Trim(ByVal str As String) As String
```

パラメータ

str

必ず指定します。任意の有効な文字列型 (**String**) の式を指定します。

解説

LTrim、**RTrim**、および **Trim** 関数は、文字列の端の空白を削除します。タブ文字など、他の形式の空白を削除するには、System.String.Trim メソッドを使用します。

使用例

文字列変数から **LTrim** 関数を使って先頭のスペースを除去し、**RTrim** 関数を使って後続のスペースを除去する例を次に示します。また、**Trim** 関数を使って両方のタイプのスペースを一度に除去します。

VB

```
' Initializes string.
Dim TestString As String = "  <-Trim->  "
Dim TrimString As String
' Returns "<-Trim->  ".
TrimString = LTrim(TestString)
' Returns "  <-Trim->".
TrimString = RTrim(TestString)
' Returns "<-Trim->".
TrimString = LTrim(RTrim(TestString))
' Using the Trim function alone achieves the same result.
' Returns "<-Trim->".
TrimString = Trim(TestString)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
Left 関数 (Visual Basic)
Right 関数 (Visual Basic)
Trim

関数 S ～ Z

次の表は、Visual Basic のランタイム メンバ関数の一覧です。

SaveSetting	Second	Seek	SetAttr
Shell	SLN	Space	SPC
Split	Str	StrComp	StrConv
StrDup	StrReverse	Switch	SYD
SystemTypeName	Tab	TimeSerial	TimeValue
Trim	TypeName	UBound	UCase
Unlock	Val	VarType	VbTypeName
Weekday	WeekdayName	Write	WriteLine
Year			

バージョン 6.0 以前に Visual Basic のランタイム メンバ関数として利用できた数値演算関数 **Sgn**、**Sin**、**Sqr**、および **Tan** は、.NET Framework クラス ライブラリの Math クラスのメソッド (Sign、Sin、Sqrt、および Tan) に置き換えられました。詳細については、「数値演算関数 (Visual Basic)」を参照してください。

スマート デバイス開発者のためのメモ

SaveSetting、**Seek**、**SetAttr**、**Shell**、**SPC**、**StrConv**、**TAB**、**Unlock**、**VarType**、**Write**、および **WriteLine** は、スマート デバイス アプリケーションではサポートされません。

参照

関連項目

関数 A ～ C
関数 D ～ G
関数 H ～ L
関数 M ～ R

その他の技術情報

Visual Basic リファレンス

SaveSetting 関数

Windows レジストリに、アプリケーションの初期ファイルを保存または作成します。

My 機能では、**SaveSetting** よりも、レジストリ操作の生産性とパフォーマンスが高くなっています。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

```
Public Sub SaveSetting( _
   ByVal AppName As String, _
   ByVal Section As String, _
   ByVal Key As String, _
   ByVal Setting As String _
)
```

パラメータ

AppName

必ず指定します。設定を適用するアプリケーション名またはプロジェクト名を含む文字列型 (**String**) の式を指定します。

Section

必ず指定します。キー設定を保存するセクション名を含む文字列型 (**String**) の式を指定します。

Key

必ず指定します。保存するキー設定名を含む文字列型 (**String**) の式を指定します。

Setting

必ず指定します。*Key* が設定される値を含む式です。

例外

例外の種類	エラー番号	条件
ArgumentException	5	キーのレジストリを作成できませんでした。または、ユーザーがログインしていません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

SaveSetting 関数は、キーを **HKEY_CURRENT_USER\Software\VB and VBA Program Settings** に追加します。

キー設定が保存できない場合、エラーが発生します。

SaveSetting は、ユーザーが対話形式でログオンするまでアクティブにならない **HKEY_LOCAL_USER** レジストリ キーのもとで動作するため、ユーザーがログオンしていることが必要条件です。

双方向でないプロセス (Mtx.exe など) からアクセスするレジストリの設定は、**HKEY_LOCAL_MACHINE\Software** または **HKEY_USER\DEFAULT\Software** レジストリ キーのいずれかに格納されます。

SaveSetting には、**Write**Read**Create** のレジストリ アクセス許可が必要です。

使用例

次の例は、最初に **SaveSetting** 関数を使用して、Windows のレジストリに `MyApp` アプリケーションのエントリを作成します。次に、**DeleteSetting** 関数を使用して、作成したエントリを削除します。

VB

```
' Place some settings in the registry.
SaveSetting("MyApp", "Startup", "Top", "75")
SaveSetting("MyApp","Startup", "Left", "50")
' Remove Startup section and all its settings from registry.
DeleteSetting ("MyApp", "Startup")
' Remove MyApp from the registry.
```

```
DeleteSetting ("MyApp")
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DeleteSetting 関数
GetAllSettings 関数
GetSetting 関数
RegistryPermission Class

概念

一般的なレジストリ タスク

Second 関数 (Visual Basic)

秒を表す 0 ～ 59 の整数型 (**Integer**) の値を返します。

```
Public Function Second(ByVal TimeValue As DateTime) As Integer
```

パラメータ

TimeValue

必ず指定します。秒を抽出する日付型 (**Date**) の値です。

解説

DatePart を呼び出し、*Interval* 引数に **DateInterval.Second** を指定することによっても秒を取得できます。

使用例

次の例は、**Second** 関数を使って、指定された時刻の秒を取得します。開発環境では、時刻リテラルは、コード記述時のロケールで設定されている短い形式で表示されます。

VB

```
Dim thisTime As Date
Dim thisSecond As Integer
thisTime = #4:35:17 PM#
thisSecond = Second(thisTime)
' thisSecond now contains 17.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : DateAndTime

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Day 関数 (Visual Basic)
Hour 関数 (Visual Basic)
Minute 関数
Now プロパティ
TimeOfDay プロパティ
DatePart 関数 (Visual Basic)
System
DateTime
ArgumentException
ArgumentOutOfRangeException

Seek 関数

FileOpen 関数で開いたファイルの現在の読み込み位置または書き込み位置を示す長整数型 (**Long**) の値を返します。また、**FileOpen** 関数で開いたファイルの、次に読み込みまたは書き込みを行う位置を設定します。

My 機能により、**Seek** よりも、ファイルの I/O 操作の生産性とパフォーマンスが向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Overloads Function Seek( _
   ByVal FileNumber As Integer _
) As Long
' -or-
Public Overloads Sub Seek( _
   ByVal FileNumber As Integer, _
   ByVal Position As Long _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号を表す整数型 (**Integer**) の値です。

Position

必ず指定します。次に読み込みまたは書き込みを行う位置を表す 1 ～ 2,147,483,647 の範囲の数。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在しません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

Seek 関数は、1 ～ 2,147,483,647 (= 2^31 – 1) の範囲の値を返します。

各ファイルのアクセス モードに応じた戻り値を次に示します。

モード	戻り値
Random	読み込みまたは書き込みが行われる次のレコードの番号。
Binary, **Input**, **Output**, **Append**	次の操作を開始するバイト位置。ファイル内の先頭バイトは 1、次のバイトは 2 になります。

使用例

Seek 関数を使って、ファイル内の現在の読み書き位置を返す例を次に示します。ファイル `TestFile` には、構造体 `Record` のレコードが含まれるものと仮定します。

VB

```
Structure Record   ' Define user-defined type.
   Dim ID As Integer
   Dim Name As String
End Structure
```

Random モードで開いたファイルでは、**Seek** 関数は読み書きできる次のレコード番号を返します。

VB

```
FileOpen(1, "TESTFILE", OpenMode.Random)
Do While Not EOF(1)
   WriteLine(1,Seek(1))    ' Write record number.
   FileGet(1, MyRecord, -1)    ' Read next record.
Loop
FileClose(1)
```

Random モード以外のモードで開いたファイルでは、**Seek** 関数は読み書きする次のバイト位置を返します。ファイル `TestFile` は、複数のテキスト行のあるファイルと仮定します。

VB

```
' Report character position at beginning of each line.
Dim TextLine As String
FileOpen(1, "TESTFILE", OpenMode.Input)    ' Open file for reading.
While Not EOF(1)
' Read next line.
   TextLine = LineInput(1)
   ' Position of next line.
   MsgBox(Seek(1))
End While
FileClose(1)
```

Seek 関数を使って、ファイル内で次に読み書きする位置を設定する例を次に示します。ファイル `People.txt` には、構造体 `Record` のレコードが含まれているものと仮定します。

VB

```
Structure TestRecord
   Dim Name As String
   Dim ID As Integer
End Structure
```

Random モード以外のモードで開いたファイルでは、**Seek** ステートメントは次に読み書きするバイト位置を設定します。ファイル `TestFile` は、複数のテキスト行のあるファイルと仮定します。

VB

```
Dim someText As String = "This is a test string."
' Open file for output.
FileOpen(1, "TESTFILE", OpenMode.Input)
' Move to the third character.
Seek(1, 3)
Input(1, someText)
Console.WriteLine(someText)
FileClose(1)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
FileGet 関数

Loc 関数
FileOpen 関数
FilePut 関数
IOException Class

その他の技術情報

Visual Basic でのファイルの読み取り
Visual Basic でのファイルへの書き込み

SetAttr 関数

ファイルの属性情報を設定します。

My 機能を使用すると、**SetAttr** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

```
Public Sub SetAttr( _
    ByVal PathName As String, _
    ByVal Attributes As FileAttribute _
)
```

パラメータ

PathName

必ず指定します。ファイル名を指定する文字列 (**String**) 式です。*PathName* にはディレクトリ名またはフォルダ名、およびドライブ名も含めて指定できます。

Attributes

必ず指定します。定数または数式を指定します。その合計でファイル属性を指定します。

設定

引数 *Attributes* の列挙型値は次のようになります。

値	定数	説明
Normal	**vbNormal**	標準ファイル (既定値)。
ReadOnly	**vbReadOnly**	読み取り専用です。
Hidden	**vbHidden**	隠しファイル。
System	**vbSystem**	システム ファイル。
Volume	**vbVolume**	ボリューム ラベル。
Directory	**vbDirectory**	ディレクトリまたはフォルダ。
Archive	**vbArchive**	前回のバックアップ以降に変更されているファイル。
Alias	**vbAlias**	他の名前が付いているファイル。

メモ :

ここに示した列挙型値は、Visual Basic 言語で設定されています。これらの名前は、実際の値の代わりにコード内のどの部分でも使用できます。

例外

例外の種類	エラー番号	状態
ArgumentException	52	*PathName* が無効です。または存在していません。
ArgumentException	5	*Attribute* の型が無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

開いたファイルの属性を設定しようとすると、実行時エラーが発生します。

使用例

SetAttr 関数を使って、ファイルの属性を設定するコード例を次に示します。

VB

```
' Set hidden attribute.
SetAttr("TESTFILE", vbHidden)
' Set hidden and read-only attributes.
SetAttr("TESTFILE", vbHidden Or vbReadOnly)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

FileAttr 関数
GetAttr 関数
ArgumentException Class
FileAttribute 列挙型
My.Computer.FileSystem オブジェクト

Shell 関数

実行可能プログラムを実行し、そのプログラムが実行中である場合、プログラムのプロセス ID を格納する整数を返します。

```
Public Function Shell( _
   ByVal PathName As String, _
   Optional ByVal Style As AppWinStyle = AppWinStyle.MinimizedFocus, _
   Optional ByVal Wait As Boolean = False, _
   Optional ByVal Timeout As Integer = -1 _
) As Integer
```

パラメータ

PathName

必ず指定します。**String** です。実行するプログラムの名前と、必要な引数とコマンド ライン スイッチです。*PathName* には、ドライブとディレクトリ パス、またはフォルダを含めることができます。

プログラムのファイル パスが不明な場合、My.Computer.FileSystem.GetFiles メソッド を使って参照できます。たとえば、`My.Computer.FileSystem.GetFiles("C:\", True, "testFile.txt")` を呼び出すことができます。ドライブ C:\ 内にある `testFile.txt` という名前のすべてのファイルのフル パスを返します。

Style

省略可能です。**AppWinStyle** です。プログラムを実行するウィンドウのスタイルを指定する AppWinStyle 列挙型 から選ばれた値です。*Style* を省略すると、**Shell** は **AppWinStyle.MinimizedFocus** を使用し、最小化され、フォーカスを持つプログラムを起動します。

Wait

省略可能です。ブール型 (**Boolean**) です。**Shell** 関数がプログラムの完了を待つかどうかを示す値を指定します。*Wait* を省略すると、**Shell** は **False** を使用します。

Timeout

省略可能です。**Integer** です。*Wait* が **True** の場合に、完了までかかる時間 (ミリ秒) です。引数 *Timeout* を省略すると、**Shell** 関数は -1 を使用します。タイムアウトが無効であり、プログラムの終了まで **Shell** 関数は制御を戻さないことを示します。このため、引数 *Timeout* を省略するか、または -1 に設定すると、**Shell** 関数はプログラムに制御をまったく戻さない可能性があります。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Style* が 0 ～ 9 (両端を含む) の範囲内の値ではありません。
FileNotFoundException	53	**Shell** が *PathName* ファイルを見つけられません。
NullReferenceException	91	*PathName* が **Nothing**。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

Shell 関数の戻り値は、引数 *PathName* に指定されたプログラムが **Shell** 関数が制御を戻す時点で実行中かどうかを示します。引数 *Wait* を **True** に設定し、タイムアウトが満了する前にプログラムが終了する場合、**Shell** 関数は 0 を返します。タイムアウトが満了するか、または引数 *Wait* を省略するか **False** に設定する場合は、**Shell** 関数はプログラムのプロセス ID を返します。プロセス ID は、実行中のプログラムを識別する、重複しない番号です。

起動の失敗

Shell 関数が指定されたプログラムを起動できない場合、**FileNotFoundException** エラーが発生します。これは、たとえば System.Windows.Forms を使っているアプリケーションから `command.com` などの 16 ビットのプログラムを実行しようとしている場合などに起こります。この問題を回避するには、起動したい 16 ビットのプログラムを呼び出す 32 ビットのプログラムを実行します。この場合は、`command.com` の代わりに `cmd.exe` を実行します。

完了の待機

既定では、**Shell** 関数はプログラムを非同期的に実行します。したがって、**Shell** 関数を使用して実行を開始したプログラムが終了しなくても、**Shell** 関数の次のステートメントは実行されます。プログラムの終了を待ってから続行する場合は、引数 *Wait* を **True** に設定します。

終了コードを決定する

プロセスは、完了時に 終了コード を返すことができます。しかし、**Shell** は完了を待っている場合 0 を返すため、また、プロセスは **Shell** とは別のオブジェクトで実行されるため、**Shell** を使用してこの終了コードを取得することはできません。

プロセスから終了コードを取得するには、プロセスを開始し、完了を待つコードを作成する必要があります。次の例では、プロセスを開始して終了を待ち、終了コードを取得する方法を示します。

```
Dim procID As Integer
Dim newProc As Diagnostics.Process
newProc = Diagnostics.Process.Start("C:\WINDOWS\NOTEPAD.EXE")
procID = newProc.Id
newProc.WaitForExit()
Dim procEC As Integer = -1
If newProc.HasExited Then
    procEC = newProc.ExitCode
End If
MsgBox("Process with ID " & CStr(ProcID) & _
    " terminated with exit code " & CStr(procEC))
```

ファイルの指定を保護する

パスとファイルの指定は、次の例のように常に全体を引用符で囲む必要があります。

```
ID = Shell("""C:\Program Files\display.exe"" -a -q", , True, 100000)
```

リテラル文字列で二重引用符が 2 つ連続する (" ") のは、文字列内の 1 つの二重引用符として解釈されます。このため、上の例は次の文字列を **Shell** 関数に渡しています。

```
"C:\Program Files\display.exe" -a -q
```

パスを引用符で囲まない場合、Windows は C:\Program Files ディレクトリの `display.exe` ではなく、C:\ ディレクトリの `Program.exe` というファイルを探します。

セキュリティに関するメモ：

ファイル名やパス ノードに空白が含まれる場合は、パスとファイルの指定を引用符で囲まないとセキュリティ上の危険があります。上記の例では、パス ノード `\Program Files` に空白が含まれます。たとえば不正な方法などにより、`Program.exe` という名前のプログラムが C:\ にインストールされている場合、ファイルとパスの指定を引用符で囲まないと、Windows は `display.exe` ではなく、このプログラムを実行します。

セキュリティに関するメモ：

Shell 関数にはアンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

使用例

Shell 関数を使って、ユーザーが指定したアプリケーションを実行するコード例は、次のとおりです。2 番目の引数に Microsoft.VisualBasic.AppWinStyle.NormalFocus を指定することで、アプリケーションは通常のサイズで開き、フォーカスを持ちます。

VB

```
Dim procID As Integer
' Run calculator.
procID = Shell("C:\Windows\system32\calc.exe", AppWinStyle.NormalFocus)
' The preceding path is for Windows XP.
' The Windows 2000 path is C:\WINNT\system32\calc.exe.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

AppWinStyle 列挙型
My.Computer.FileSystem.GetFiles メソッド
AppActivate 関数
ArgumentException
FileNotFoundException
NullReferenceException

SLN 関数

倍精度浮動小数点数型 (**Double**) の値を返します。定額法を用いて、資産の 1 期あたりの減価償却費を返します。

```
Function SLN( _
   ByVal Cost As Double, _
   ByVal Salvage As Double, _
   ByVal Life As Double _
) As Double
```

パラメータ

Cost

必ず指定します。資産を購入した時点での価格を示す倍精度浮動小数点数型 (**Double**) の値を指定します。

Salvage

必ず指定します。耐用年数が終了した時点での資産の価格を示す倍精度浮動小数点数型 (**Double**) の値を指定します。

Life

必ず指定します。資産の耐用年数を示す倍精度浮動小数点数型 (**Double**) の値を指定します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Life* = 0.

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

減価償却の期間は、引数 *Life* と同じ単位で表されます。すべての引数には必ず正の値を指定してください。

使用例

次の例は、資産 1 期あたりの減価償却費を定額法を使って **SLN** 関数で計算します。資産購入時の価格 (`InitCost`)、耐用年数を経た後での資産の価格 (`SalvageVal`)、資産の耐用年数 (`LifeTime`) を指定します。

VB

```
Dim InitCost, SalvageVal, LifeTime, DepYear As Double
Dim Fmt As String = "###,##0.00"

InitCost = CDbl(InputBox("What's the initial cost of the asset?"))
SalvageVal = CDbl(InputBox("Enter the asset's value at end of its life."))
LifeTime = CDbl(InputBox("What's the asset's useful life in years?"))

' Use the SLN function to calculate the deprecation per year.
Dim SlnDepr As Double = SLN(InitCost, SalvageVal, LifeTime)
Dim msg As String = "The depreciation per year: " & Format(SlnDepr, Fmt)
msg &= vbCrLf & "Year" & vbTab & "Linear" & vbTab & "Doubling" & vbCrLf

' Use the SYD and DDB functions to calculate the deprecation for each year.
For DepYear = 1 To LifeTime
    msg &= DepYear & vbTab & _
        Format(SYD(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbTab & _
        Format(DDB(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbCrLf
Next
MsgBox(msg)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

SYD 関数

DDB 関数

財務処理の概要

ArgumentException

Space 関数 (Visual Basic)

指定された数のスペースから成る文字列を返します。

```
Public Shared Function Space(ByVal Number As Integer) As String
```

パラメータ

Number

必ず指定します。整数 (**Integer**) を指定します。スペースの数を指定します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Number* < 0.

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Space 関数は、出力の書式指定や固定長文字列のデータのクリアを行うのに便利です。

使用例

Space 関数を使って、指定された数のスペースから成る文字列を返す例を次に示します。

VB

```
Dim TestString As String
' Returns a string with 10 spaces.
TestString = Space(10)
' Inserts 10 spaces between two strings.
TestString = "Hello" & Space(10) & "World"
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
SPC 関数
ArgumentException

SPC 関数

Print 関数または **PrintLine** 関数と共に使用し、出力の位置を移動させます。

```
Public Function SPC(ByVal Count As Short) As SPCInfo
```

パラメータ
Count

必ず指定します。リスト内の次の式を表示または印刷するときに、式の前に挿入する空白の数。

解説

引数 *Count* に出力行の桁数未満の値を指定すると、次の出力位置は指定した数の空白の直後になります。引数 *Count* に出力行の桁数を超える値を指定すると、**SPC** 関数は次の出力位置を次の式で計算します。

currentprintposition(+(*Count* **Mod** *width*))

たとえば、現在の出力位置を 24、出力行の桁数を 80 として、**SPC(**90**)** と指定すると、次の出力は 34 の位置 (現在の出力位置 + 90/80 の剰余) から始まります。現在の出力位置と出力行の桁数との差が *Count* (または、*Count* **Mod** *width*) 未満の場合、**SPC** 関数は次の行の先頭に進み、*Count* – (*width* – *currentprintposition*) の式に従って空白の数を計算し、次の出力位置を決めます。

メモ :

幅の広い文字を表示できるだけの十分な桁数があることを確認してください。

使用例

SPC 関数を使って、出力ファイルと [出力] ウィンドウ内の出力位置を移動する例を次に示します。

VB

```
' The SPC function can be used with the Print function.
FileOpen(1, "TESTFILE", OpenMode.Output)   ' Open file for output.
Print(1, "10 spaces between here", SPC(10), "and here.")
FileClose(1)   ' Close file.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
Mod 演算子 (Visual Basic)
Print 関数、PrintLine 関数
Space 関数 (Visual Basic)
TAB 関数
FileWidth 関数

Split 関数 (Visual Basic)

部分文字列ごとに区切られた文字列からゼロ ベースの 1 次元配列を作成し、返します。

```
Function Split(
   ByVal Expression As String,
   Optional ByVal Delimiter As String = " ",
   Optional ByVal Limit As Integer = -1,
   Optional ByVal Compare As CompareMethod = CompareMethod.Binary
) As String()
```

パラメータ

Expression

必ず指定します。部分文字列と区切り文字を含んだ文字列 (**String**) 式を指定します。

Delimiter

省略可能です。文字列の区切りを識別する任意の文字を指定します。*Delimiter* を省略すると、区切り文字にスペース (" ") が使用されます。

Limit

省略可能です。入力文字列を分割した部分文字列の最大数です。既定値の –1 は、*Delimiter* 文字列のある部分で入力文字列を分割することを示します。

Compare

省略可能です。部分文字列を評価するときに使用する文字列比較のモードを表す数値を指定します。値については、設定」を参照してください。

戻り値

String 配列*Expression* が長さ 0 の文字列 ("") である場合、**Split** は長さ 0 の文字列を含む、単一要素の配列を返します。*Delimiter* が長さ 0 の文字列である場合、または *Expression* のどこにもない場合は、**Split** は *Expression* 文字列全体を含む単一要素の配列を返します。

設定

引数 *Compare* の設定値は次のとおりです。

定数	説明	値
CompareMethod.Binary	バイナリ モードで比較を行います。	0
CompareMethod.Text	テキスト モードで比較を行います。	1

解説

既定、または *Limit* が -1 の場合、**Split** 関数はデリミタ文字列で入力文字列を分割し、部分文字列を含む配列を返します。*Limit* パラメータが 0 より大きい場合、**Split** 関数は、最初の *Limit*-1 のデリミタ出現箇所で文字列を分割し、分割後の部分文字列を含む配列を返します。たとえば、`Split("a:b:c", ":")` は配列 `{"a", "b", "c"}` を返し、`Split("a:b:c", ":", 2)` は配列 `{"a", "b:c"}` を返します。

1 行に 2 つのデリミタがある場合、または、文字列の冒頭や末尾にデリミタがある場合、**Split** 関数はこれらのデリミタが空白の文字列 ("") を囲んでいると見なします。たとえば、`Split("xx", "x")` は空白の文字列を 3 つ含む配列を返します。文字列の冒頭と最初の "x" の間にあるもの、2 つの "x" 文字列の間にあるもの、最後の "x" と文字列の末尾の間にあるものの 3 つです。

この表では、省略可能な *Delimiter*、*Limit*、そして *Compare* パラメータを使用して **Split** 関数の動作を変更する方法を示します。

Split の呼び出し	戻り値
`Split("42, 12, 19")`	{"42," , "12," , "19"}
`Split("42, 12, 19", ", ")`	{"42", "12", "19"}

`Split("42, 12, 19", ", ", 2)`	{"42", "12, 19"}
`Split("192.168.0.1", ".")`	{"192", "168", "0", "1"}
`Split("Alice and Bob", " AND ")`	{"Alice and Bob"}
`Split("Alice and Bob", " AND ", ,CompareMethod.Text)`	{"Alice", "Bob"}
`Split("someone@example.com", "@",1)`	{"someone@example.com"}
`Split("someone@example.com", "@",2)`	{"someone", "example.com"}

使用例

次の例では、空白で文字列を区切る方法を示します。

VB

```
Dim TestString As String = "Look at these!"
' Returns an array containing "Look", "at", and "these!".
Dim TestArray() As String = Split(TestString)
```

次の例では、同じ行内の複数のデリミタで文字列を区切り、空白の文字列をフィルタする方法を示します。

VB

```
Dim TestString As String = "apple    pear banana  "
Dim TestArray() As String = Split(TestString)
' TestArray holds {"apple", "", "", "", "pear", "banana", "", ""}
Dim LastNonEmpty As Integer = -1
For i As Integer = 0 To TestArray.Length - 1
    If TestArray(i) <> "" Then
        LastNonEmpty += 1
        TestArray(LastNonEmpty) = TestArray(i)
    End If
Next
ReDim Preserve TestArray(LastNonEmpty)
' TestArray now holds {"apple", "pear", "banana"}
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要

Join 関数 (Visual Basic)

CompareMethod 列挙型

Str 関数

数値を表す文字列を返します。

```
Public Shared Function Str(ByVal Number As Object) As String
```

パラメータ

Number

必ず指定します。任意の有効な数値式を含むオブジェクト型 (**Object**) の値です。

例外

例外の種類	エラー番号	条件
ArgumentNullException	5	*Number* が指定されていません。
InvalidCastException	5	*Number* が数値型ではありません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

数値を文字列に変換する場合、文字列の先頭には *Number* の記号用のスペースが確保されます。*Number* が正である場合、返された文字列には先頭にスペースが含まれます。これは、明示されないプラス記号があることを意味します。負の数にはマイナス記号 (-) が明示され、先頭にスペースは挿入されません。

Format 関数を使用して、数値を日付、時刻、通貨、およびその他のユーザー定義の書式に変換できます。**Str** 関数とは異なり、**Format** 関数には *Number* の先頭に記号用のスペースは含まれません。

メモ :

Str 関数は、ピリオド (.) だけを有効な小数点の記号として認識します。国際対応のアプリケーションなどで、ピリオドと異なる小数点の記号が使われている場合は、**CStr** 関数または **Format** 関数を使用して、数値を文字列に変換してください。特定の文化圏における数値の文字表現を調べるには、その数値の `ToString(IFormatProvider)` メソッドを使用します。たとえば、**Double** 型の値を文字列に変換するには、ToString を使用します。

使用例

Str 関数を使って、数値の **String** 表現を返す例を次に示します。正の数を文字列に変換すると、先頭にはプラス記号用のスペースが常に確保されます。

VB

```
Dim TestString As String
' Returns " 459".
TestString = Str(459)
' Returns "-459.65".
TestString = Str(-459.65)
' Returns " 459.001".
TestString = Str(459.001)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Conversion**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

変換の概要
Format 関数
データ型変換関数
Format 関数
Val 関数
ArgumentNullException
InvalidCastException

StrComp 関数 (Visual Basic)

文字列比較の結果により、-1、0、または 1 のいずれかを返します。

```
Public Shared Function StrComp( _
   ByVal String1 As String, _
   ByVal String2 As String, _
   <Microsoft.VisualBasic.OptionCompareAttribute> _
   Optional ByVal Compare As Microsoft.VisualBasic.CompareMethod _
) As Integer
```

パラメータ

String1

必ず指定します。任意の有効な文字列型 (**String**) の式を指定します。

String2

必ず指定します。任意の有効な文字列型 (**String**) の式を指定します。

Compare

省略可能です。文字列比較のタイプを指定します。*Compare* を省略すると、**Option Compare** 設定により比較のタイプが決定されます。

設定

引数 *Compare* には、次の値を指定します。

定数	説明
Binary	文字の内部バイナリ表現による並べ替え順序に基づいて、バイナリ比較を行います。
Text	アプリケーションの現在のカルチャ情報によって決められた、大文字と小文字が区別されない並べ替え順序に基づいて、テキスト比較を行います。

戻り値

StrComp 関数の戻り値は次のとおりです。

条件	StrComp の戻り値
String1 が *String2* よりも先に来る	-1
String1 が *String2* と等しい	0
String1 が *String2* の後に来る	1

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Compare* の値が無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

文字列は、先頭文字から英数字の並べ替え値に基づいて比較されます。バイナリ比較、テキスト比較、および並べ替え順序の詳細については、「Option Compare ステートメント」を参照してください。

セキュリティに関するメモ：

比較処理または大文字/小文字変換処理の結果に基づいてアプリケーションのセキュリティ関連の決定を下す場合は、その処理を System.String.Compare(System.String,System.String,System.StringComparison) メソッドで実行し、*comparisonType* 引数に Ordinal または OrdinalIgnoreCase を渡してください。詳細については、「Visual Basic においてカルチャが文字列に与える影響」を参照してください。

使用例

StrComp 関数を使って文字列比較の結果を返す例を次に示します。3 番目の引数が省略された場合は、**Option Compare** ステートメントで定義された比較のタイプ、またはプロジェクトの既定の比較のタイプが実行されます。

VB

```
' Defines variables.
Dim TestStr1 As String = "ABCD"
Dim TestStr2 As String = "abcd"
Dim TestComp As Integer
' The two strings sort equally. Returns 0.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Text)
' TestStr1 sorts after TestStr2. Returns -1.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Binary)
' TestStr2 sorts before TestStr1. Returns 1.
TestComp = StrComp(TestStr2, TestStr1)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
InStr 関数 (Visual Basic)
ArgumentException

その他の技術情報

Visual Basic における文字列
Visual Basic の文字列の概要

StrConv 関数

指定に従って変換された文字列型 (**String**) の値を返します。

```
Public Shared Function StrConv( _
   ByVal str As String, _
   ByVal Conversion As Microsoft.VisualBasic.VbStrConv, _
   Optional ByVal LocaleID As Integer,
) As String
```

パラメータ

str

必ず指定します。変換される文字列 (**String**) 式を指定します。

Conversion

必ず指定します。VbStrConv 列挙型 のメンバを指定します。実行する変換のタイプを指定する列挙型の値です。

LocaleID

省略可能です。アプリケーションの現在のカルチャ値と異なる場合は、**LocaleID** 値を指定します。ロケール ID とカルチャ情報の詳細については、「CultureInfo」を参照してください。既定値は、アプリケーションの現在のカルチャ値です。

設定

引数 *Conversion* には、次の値を指定します。

列挙型メンバ	説明
VbStrConv.None	変換は行われません。
VbStrConv.LinguisticCasing	大文字と小文字については、ファイル システム (既定) ではなく言語の規則に従います。**VbStrConv.UpperCase** および **VbStrConv.LowerCase** にのみ使用できます。
VbStrConv.UpperCase	文字列を大文字に変換します。
VbStrConv.LowerCase	文字列を小文字に変換します。
VbStrConv.ProperCase	文字列の各単語の先頭の文字を大文字に変換します。
VbStrConv.Wide *	文字列内の半角文字 (1 バイト) を全角文字 (2 バイト) に変換します。
VbStrConv.Narrow *	文字列内の全角文字 (2 バイト) を半角文字 (1 バイト) に変換します。
VbStrConv.Katakana **	文字列内のひらがなをカタカナに変換します。
VbStrConv.Hiragana **	文字列内のカタカナをひらがなに変換します。
VbStrConv.SimplifiedChinese *	中国語の繁体字を簡体字に変換します。
VbStrConv.TraditionalChinese *	中国語の簡体字を繁体字に変換します。

* アジア ロケールに適用されます。

** ロケール設定が日本の場合のみ有効です。

メモ：

これらの定数は、.NET Framework 共通言語ランタイムに指定されています。したがって、実際の値の代わりにコード内のどの部分でも使用できます。多くの場合、定数は組み合わせて使用できます (`UpperCase + Wide` など)。ただし、互いに両立しない定数 (`VbStrConv.Wide + VbStrConv.Narrow` など) を除きます。

単語の区切り記号には、Null (`Chr$(0)`)、水平タブ (`Chr$(9)`)、ライン フィード (`Chr$(10)`)、垂直タブ (`Chr$(11)`)、フォーム フィード (`Chr$(12)`)、キャリッジ リターン (`Chr$(13)`)、空白 (1 バイト文字セット) (`Chr$(32)`) を使用できます。ただし、大文字と小文字の記述が正しいことが必要です。空白の実際の値は、全角文字と半角文字のどちらでもかまいませんが、東アジア文化圏では国/地域によって異なります。

例外

例外の種類	エラー番号	状態
ArgumentException	5	*LocaleID* の値がサポートされていないか、*Conversion* が 0 未満または 2048 より大きいか、または変換の方法が指定されたロケールではサポートされていません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

定数
VbStrConv.Wide、**VbStrConv.Narrow**、**VbStrConv.SimplifiedChinese**、**VbStrConv.TraditionalChinese**、**VbStrConv.Katakana**、および **VbStrConv.Hiragana** を、これらが適用されないロケールで使用すると、ランタイム エラーを起こす可能性があります (常に起こるわけではありません)。定数 **VbStrConv.Katakana** と **VbStrConv.Hiragana** は、日本語 Language Pack がインストールされた日本語以外のシステムで使用できます。また、定数 **VbStrConv.Wide** および **VbStrConv.Narrow** は、東アジアの言語がインストールされているシステムでサポートされます。

この関数は文字列を操作するときにアプリケーションのカルチャ情報を使用するため、大文字小文字がアプリケーションで使用しているロケールに対して正しく変換されます。

セキュリティに関するメモ：

比較処理または大文字/小文字変換処理の結果に基づいてアプリケーションのセキュリティ関連の決定を下す場合は、その処理を System.String.Compare(System.String,System.String,System.StringComparison) メソッドで実行し、*comparisonType* 引数に Ordinal または OrdinalIgnoreCase を渡してください。詳細については、「Visual Basic においてカルチャが文字列に与える影響」を参照してください。

使用例

テキストをすべて小文字に変換する例を次に示します。

VB

```
Dim sText As String = "Hello World"
' Returns "hello world".
Dim sNewText As String = StrConv(sText, VbStrConv.LowerCase)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
文字列操作の概要
Chr 関数、ChrW 関数
文字列型 (String) (Visual Basic)
データ型変換関数
ArgumentException

StrDup 関数

指定された文字が指定された回数繰り返されている文字列型 (**String**) またはオブジェクト型 (**Object**) の値を返します。

```
Public Shared Function StrDup( _
   ByVal Number As Integer, _
   ByVal Character As { Char | String } _
) As String
' -or-
Public Shared Function StrDup( _
   ByVal Number As Integer, _
   ByVal Character As Object _
) As Object
```

パラメータ

Number

必ず指定します。オブジェクト型 (**Integer**) の式です。取得する文字列の長さを指定します。

Character

必ず指定します。任意の有効な **Char**、**String**、または **Object** 式。式の先頭文字だけが使用されます。文字が **Object** 型の場合、**Char** または **String** 値のいずれかを含む必要があります。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Number* が 0 以下の数字、または *Character* の型が無効。
ArgumentNullException	5	*Character* が **Nothing**。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

この関数は、文字の繰り返しから成る文字列型 (**String**) の値を返します。文字列を構成する文字は、引数 *Character* の先頭文字です。これが *Number* の回数だけ複製されます。

使用例

StrDup 関数を使って、文字の繰り返しで構成される文字列を取得する例を次に示します。

VB

```
Dim aString As String = "Wow! What a string!"
Dim aObject As New Object
Dim TestString As String
aObject = "This is a String contained within an Object"
' Returns "PPPPP"
TestString = StrDup(5, "P")
' Returns "WWWWWWWWWW"
TestString = StrDup(10, aString)
' Returns "TTTTTT"
TestString = CStr(StrDup(6, aObject))
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要

SPC 関数

ArgumentException

ArgumentNullException

StrReverse 関数 (Visual Basic)

指定された文字列の文字の並び順を逆にした文字列を返します。

```
Public Function StrReverse(ByVal Expression As String) As String
```

パラメータ

Expression

必ず指定します。文字の並び順を逆にする文字列式を指定します。*Expression* が長さ 0 の文字列 ("") である場合は、長さ 0 の文字列を返します。

解説

StrReverse 関数は、*Expression* と同じ文字を逆の順序で含む文字列を返します。

使用例

VB

```
Dim TestString As String = "ABCDEFG"
' Returns "GFEDCBA".
Dim revString As String = StrReverse(TestString)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要

InStrRev 関数 (Visual Basic)

Switch 関数

式のリストを評価し、リスト内で最初に **True** になった式に対応する **Object** の値を返します。

```
Public Function Switch( _
    ByVal ParamArray VarExpr() As Object _
) As Object
```

パラメータ

VarExpr

必ず指定します。オブジェクト型 (**Object**) のパラメータ配列です。要素数は偶数である必要があります。コンマで区切られたオブジェクト型 (**Object**) の変数や式のリスト、または要素がオブジェクト型 (**Object**) の 1 次配列を指定できます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	引数の数は奇数です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

VarExpr に渡される引数は、式と値のペアから成ります。**Switch** 関数は、*VarExpr* 内のインデックスの値の小さいものから順に、奇数の式を評価し、**True** と評価された最初の式に関連付けられた偶数の値を返します。たとえば、*VarExpr(0)* が **True** の場合、**Switch** は *VarExpr(1)* を返し、*VarExpr(0)* が **False** で *VarExpr(2)* が **True** の場合、**Switch** は *VarExpr(3)* を返します。

引数 *VarExpr* を指定しない場合、**Switch** は **Nothing** を返します。

メモ :

引数リスト内の式には、関数呼び出しが含まれることがあります。**Switch** の呼び出しに使用する引数リストを用意するために、Visual Basic コンパイラは、すべての式のすべての関数を呼び出します。このため、引数リストで前に位置する式が **True** である場合でも、呼び出されていない特定の関数には依存できません。

使用例

Switch 関数を使って、指定された都市で使用されている言語を返すコード例は、次のとおりです。**Option Strict** が **Off** である必要があります。

VB

```
Function matchLanguage(ByVal cityName As String) As String
    Return CStr(Microsoft.VisualBasic.Switch( _
        cityName = "London", "English", _
        cityName = "Rome", "Italian", _
        cityName = "Paris", "French"))
End Function
```

System.Diagnostics 名前空間にも Switch と呼ばれるクラスがあるため、**Switch** 関数の呼び出しでは Microsoft.VisualBasic 名前空間を指定する必要があります。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Choose 関数
IIf 関数
Select...Case ステートメント (Visual Basic)
概念
パラメータ配列

SYD 関数

倍精度浮動小数点数型 (Double) の値を返します。定額逓減法を使って、指定した期の減価償却費を返します。

```
Function SYD( _
   ByVal Cost As Double, _
   ByVal Salvage As Double, _
   ByVal Life As Double, _
   ByVal Period As Double _
) As Double
```

パラメータ

Cost

必ず指定します。資産を購入した時点での価格を示す倍精度浮動小数点数型 (**Double**) の値です。

Salvage

必ず指定します。耐用年数が終了した時点での資産の価格を示す倍精度浮動小数点数型 (**Double**) の値です。

Life

必ず指定します。資産の耐用年数を示す倍精度浮動小数点数型 (**Double**) の値です。

Period

必ず指定します。減価償却費を計算する期を示す倍精度浮動小数点数型 (**Double**) の値です。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Salvage* < 0、*Period* > *Life*、または *Period* <=0 です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

引数 *Life* および *Period* は、同じ単位で指定する必要があります。たとえば、*Life* を月で指定した場合は、*Period* も月で指定することが必要です。すべての引数には必ず正の値を指定してください。

使用例

次の例は、資産 1 期あたりの減価償却費を **SYD** 関数を使って計算します。資産購入時の価格 (`InitCost`)、耐用年数を経た後での資産の価格 (`SalvageVal`)、資産の耐用年数 (`LifeTime`) を指定します。

VB

```
Dim InitCost, SalvageVal, LifeTime, DepYear As Double
Dim Fmt As String = "###,##0.00"

InitCost = CDbl(InputBox("What's the initial cost of the asset?"))
SalvageVal = CDbl(InputBox("Enter the asset's value at end of its life."))
LifeTime = CDbl(InputBox("What's the asset's useful life in years?"))

' Use the SLN function to calculate the deprecation per year.
Dim SlnDepr As Double = SLN(InitCost, SalvageVal, LifeTime)
Dim msg As String = "The depreciation per year: " & Format(SlnDepr, Fmt)
msg &= vbCrLf & "Year" & vbTab & "Linear" & vbTab & "Doubling" & vbCrLf

' Use the SYD and DDB functions to calculate the deprecation for each year.
For DepYear = 1 To LifeTime
    msg &= DepYear & vbTab & _
        Format(SYD(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbTab & _
        Format(DDB(InitCost, SalvageVal, LifeTime, DepYear), Fmt) & vbCrLf
```

```
Next
MsgBox(msg)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Financial**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DDB 関数
SLN 関数
財務処理の概要
ArgumentException

SystemTypeName 関数

変数のシステム データ型名を含む文字列型 (**String**) の値を返します。

```
Public Function SystemTypeName(ByVal VbName As String) As String
```

パラメータ
VbName

必ず指定します。Visual Basic の型名を含む文字列型 (**String**) の変数です。

解説

SystemTypeName は、Visual Basic の型名に対応する共通言語ランタイム (CLR: Common Language Runtime) の完全限定型名を返します。たとえば、*VbName* に "Date" を指定した場合、**SystemTypeName** は "System.DateTime" を返します。**SystemTypeName** が *VbName* の値を認識できない場合は、**Nothing** (文字列 "Nothing" ではなく) を返します。

使用例

次の例は、**SystemTypeName** 関数を使って、いくつかの変数のデータ型名を返します。

VB

```
Dim vbLongName As String = "Long"
Dim vbDateName As String = "Date"
Dim vbBadName As String = "Number"
Dim testSysName As String
testSysName = SystemTypeName(vbLongName)
' The preceding call returns "System.Int64".
testSysName = SystemTypeName(vbDateName)
' The preceding call returns "System.DateTime".
testSysName = SystemTypeName(vbBadName)
' The preceding call returns Nothing.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
データ型の概要 (Visual Basic)
文字列型 (String) (Visual Basic)
VbTypeName 関数

TAB 関数

Print 関数または **PrintLine** 関数と共に使用し、出力の位置を移動させます。

```
Public Overloads Function TAB() As TABInfo
' -or-
Public Overloads Function TAB(ByVal Column As Short) As TABInfo
```

パラメータ

Column

省略可能です。リスト内の式を表示または印刷する前に移動する列数。省略すると、**TAB** は次の印字領域の先頭にカーソルを移動します。

解説

出力行上の出力位置が引数 *Column* を超える場合、**TAB** 関数は次の出力行の *Column* 桁目に進みます。引数 *Column* に 1 未満の値を指定すると、**TAB** は出力位置を桁位置 1 に移動します。引数 *Column* が出力行の桁数を超える場合、**TAB** 関数は次の式を使って次の出力位置を決めます。

Column Mod width

たとえば、出力行の桁数 (*width*) が 80 で、**TAB(**90**)** と指定すると、次の出力は 10 の位置 (90/80 の剰余) から始まります。指定した桁位置 *Column* が現在の出力位置より前の場合、次の行に改行され、計算した桁位置に出力位置が移動します。計算された出力位置が現在の出力位置より後ろのときは、同じ行上の計算された位置に出力位置が移動します。

出力行上の出力位置は左端を 1 と数えます。**Print** 関数または **PrintLine** 関数を使ってファイルを出力する場合、出力位置の右端は出力ファイルの現在の 1 行の桁数に相当します。出力ファイルの桁数は、**FileWidth** 関数を使って設定します。

TAB 関数は **WriteLine** 関数と一緒に使うこともできます。System.Diagnostics.Debug.WriteLine や System.Console.WriteLine と一緒に使うことはできません。

メモ：

幅の広い文字を表示できるだけの十分な桁数があることを確認してください。

使用例

TAB 関数を使って、出力ファイルと [出力] ウィンドウ内の出力位置を移動する例を次に示します。

VB

```
FileOpen(1, "TESTFILE", OpenMode.Output) ' Open file for output.
' The second word prints at column 20.
Print(1, "Hello", TAB(20), "World.")
' If the argument is omitted, cursor is moved to the next print zone.
Print(1, "Hello", TAB(), "World")
FileClose(1)
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Mod 演算子 (Visual Basic)

Print 関数、PrintLine 関数

Space 関数 (Visual Basic)
SPC 関数
FileWidth 関数

TimeSerial 関数 (Visual Basic)

西暦 1 年 1 月 1 日を基準にして日付情報が設定された、指定の時、分、および秒を表す日付型 (**Date**) の値を返します。

```
Public Function TimeSerial( _
   ByVal Hour As Integer, _
   ByVal Minute As Integer, _
   ByVal Second As Integer _
) As DateTime
```

パラメータ

Hour

必ず指定します。0 ～ 23 の整数型 (**Integer**) の式です。ただし、この範囲外の値も指定できます。

Minute

必ず指定します。0 ～ 59 の整数型 (**Integer**) の式です。ただし、この範囲外の値も指定できます。計算された時刻に *Minute* の値が加算されるため、負の値はその時刻より前の分を指定します。

Second

必ず指定します。0 ～ 59 の整数型 (**Integer**) の式です。ただし、この範囲外の値も指定できます。計算された分に *Second* の値が加算されるため、負の値はその分より前の秒を指定します。

例外

例外の種類	エラー番号	条件
ArgumentException	5	引数が -2,147,483,648 ～ 2,147,483,647 の範囲外です。
ArgumentOutOfRangeException	9	計算された時刻が負の 24 時間より小さくなっています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

負の値、0、および正の値の引数を次に示します。**TimeSerial** 関数は、正午の 3 時間前の 15 分前の時刻 (8:45:00 午前) を返します。

```
Dim alarmTime As Date = TimeSerial(12 - 3, -15, 0)
```

Minute または *Second* が通常の範囲を超えた場合は、その上の単位に繰り上げられます。たとえば、引数 Minute に 75 を指定した場合は、1 時間と 15 分として評価されます。

TimeSerial は、合計秒数に対する 86,400 (1 日の秒数) の剰余をとります。したがって、返される時刻は常に 00:00:00 ～ 23:59:59 の範囲になります。

日付型 (**Date**) には日付コンポーネントが格納されます。**TimeSerial** は、すべての日付コンポーネントを 1 に設定するため、返される値は西暦 1 年の最初の日を表します。ただし、引数の値によって計算される時刻が 24 時間を超える場合は、必要に応じて日がインクリメントされます。次の例では、*Hour* と *Minute* の値を組み合わせた結果が 24 時間を超えています。

```
MsgBox(TimeSerial(23, 75, 0))
' The preceding statement displays "1/2/0001 12:15:00 AM".
```

引数の値によって計算された時刻が負になる場合、日付情報は 1/1/0001 に設定され、時刻情報は 00:00:00 ～ 23:59:59 の範囲に収まるように調整されます。ただし、計算された時刻が負の 24 時間より小さい場合は、**ArgumentOutOfRangeException** エラーが発生します。

すべての日付型 (**Date**) の値は System.DateTime 構造体でサポートされているため、メソッドには日付型 (**Date**) の値をアセンブルするときの追加のオプションがあります。たとえば、オーバーロードされた DateTime コンストラクタの 1 つを使用し、コンポーネントを任意に組み合わせて日付型 (**Date**) の変数を設定できます。`newDateTime` に 1978 年 5 月 6 日午前 8:30 の 0.1 秒前を設定する例を次に示します。

```
Dim newDateTime As Date = New Date(1978, 5, 6, 8, 29, 59, 900)
```

使用例

次の例は、**TimeSerial** 関数を使って、指定された時、分、秒で表される時刻を求めます。

VB

```
Dim thisTime As Date
thisTime = TimeSerial(16, 35, 17)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

DateSerial 関数 (Visual Basic)
DateValue 関数 (Visual Basic)
Hour 関数 (Visual Basic)
Minute 関数
Now プロパティ
Second 関数 (Visual Basic)
TimeValue 関数 (Visual Basic)

TimeValue 関数 (Visual Basic)

日付情報が 1 年 1 月 1 日に設定された、文字列で表される時刻情報を含む日付型 (**Date**) の値を返します。

```
Public Function TimeValue(ByVal StringTime As String) As DateTime
```

パラメータ
StringTime

必ず指定します。1 年 1 月 1 日 00:00:00 から 9999 年 12 月 31 日 23:59:59 までの日時の値を表す文字列 (**String**) 式。

例外

例外の種類	エラー番号	条件
InvalidCastException	13	*StringTime* に無効な日付情報が含まれています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

12 時間制、24 時間制のどちらでも時刻を指定できます。たとえば、"2:24 P.M." と "14:24" は、両方とも有効な *StringTime* 引数です。

StringTime 引数に日付情報が含まれる場合、**TimeValue** は返される値に日付情報を含めません。しかし、*StringTime* に "January 32" などの無効な日付情報が含まれている場合、**InvalidCastException** エラーが発生します。

使用例

この例は、**TimeValue** 関数を使って、文字列を時刻に変換します。日付リテラルを使用して、時刻を日付型 (**Date**) の変数に直接代入することもできます。

VB

```
Dim thisTime As Date
thisTime = TimeValue("4:35:17 PM")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
DateSerial 関数 (Visual Basic)
DateValue 関数 (Visual Basic)
Hour 関数 (Visual Basic)
Minute 関数
Now プロパティ
Second 関数 (Visual Basic)
TimeSerial 関数 (Visual Basic)
DateTime

Trim、LTrim、RTrim 関数

指定された文字列から先頭のスペースを除いたコピー (**LTrim**)、後続のスペースを除いたコピー (**RTrim**)、または先頭と後続のスペースを除いたコピー (**Trim**) を含む文字列を返します。

```
Public Shared Function LTrim(ByVal str As String) As String
Public Shared Function RTrim(ByVal str As String) As String
Public Shared Function Trim(ByVal str As String) As String
```

パラメータ

str

必ず指定します。任意の有効な文字列型 (**String**) の式を指定します。

解説

LTrim、**RTrim**、および **Trim** 関数は、文字列の端の空白を削除します。タブ文字など、他の形式の空白を削除するには、System.String.Trim メソッドを使用します。

使用例

文字列変数から **LTrim** 関数を使って先頭のスペースを除去し、**RTrim** 関数を使って後続のスペースを除去する例を次に示します。また、**Trim** 関数を使って両方のタイプのスペースを一度に除去します。

VB

```
' Initializes string.
Dim TestString As String = "  <-Trim->  "
Dim TrimString As String
' Returns "<-Trim->  ".
TrimString = LTrim(TestString)
' Returns "  <-Trim->".
TrimString = RTrim(TestString)
' Returns "<-Trim->".
TrimString = LTrim(RTrim(TestString))
' Using the Trim function alone achieves the same result.
' Returns "<-Trim->".
TrimString = Trim(TestString)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

文字列操作の概要
Left 関数 (Visual Basic)
Right 関数 (Visual Basic)
Trim

TypeName 関数 (Visual Basic)

変数に関するデータ型情報を含む文字列型 (**String**) の値を返します。

```
Public Function TypeName(ByVal VarName As Object) As String
```

パラメータ

VarName

必ず指定します。オブジェクト型 (**Object**) の変数です。**Option Strict** が **Off** であれば、構造体を除くすべてのデータ型の変数を渡すことができます。

解説

VarName の内容と、**TypeName** から返される文字列 (**String**) 値の関係を次の表に示します。

***VarName* の内容**	**返される文字列**
16 ビットの **True** または **False** の値型	"Boolean"
8 ビットのバイナリの数値型	"Byte"
16 ビットの文字型の数値型	"Char"
64 ビットの日付型と時刻型の値	"Date"
参照不可能なデータまたは存在しないデータを示す参照型	"DBNull"
128 ビットの固定小数点数値型	"Decimal"
64 ビットの浮動小数点数値型	"Double"
32 ビット整数型値	"Integer"
特定されていないオブジェクトを指す参照型	"Object"
objectclass クラスから作成され、特定されているオブジェクトを指す参照型	"*objectclass*"
64 ビット整数型値	"Long"
現在オブジェクトが割り当てられていない参照型	"Nothing"
8 ビット符号付き整数値型	"SByte"
16 ビット整数型値	"Short"
32 ビットの浮動小数点数値型	"Single"
16 ビットの文字列を指す参照型	"String"
32 ビット符号なし整数値型	"UInteger"
64 ビット符号なし整数値型	"ULong"

16 ビット符号なし整数値型	"UShort"

VarName が配列の場合は、上の表のいずれかの文字列に、空のかっこが付いた文字列が返されます。たとえば、*VarName* が整数の配列を指す場合、**TypeName** は "Integer()" を返します。

TypeName がクラスなどの参照型の名前を返す場合、修飾名ではなく名前そのものだけを返します。たとえば、*VarName* が System.Drawing.Printing.PaperSource クラスのオブジェクトを指す場合、**TypeName** は "PaperSource" を返します。変数が特定のクラス型に宣言されているが、オブジェクトが変数に割り当てられていなければ、**TypeName** は "Nothing" を返します。

使用例

次の例は、**TypeName** 関数を使って、いくつかの変数のデータ型情報を返します。

VB

```
Dim testType As String
Dim strVar As String = "String for testing"
Dim decVar As Decimal
Dim intVar, arrayVar(5) As Integer
testType = TypeName(strVar)
' The preceding call returns "String".
testType = TypeName(decVar)
' The preceding call returns "Decimal".
testType = TypeName(intVar)
' The preceding call returns "Integer".
testType = TypeName(arrayVar)
' The preceding call returns "Integer()".
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

データ型の概要 (Visual Basic)
IsArray 関数 (Visual Basic)
IsDate 関数 (Visual Basic)
IsDBNull 関数
IsError 関数
IsNothing 関数
IsNumeric 関数 (Visual Basic)
IsReference 関数

UBound 関数 (Visual Basic)

配列の指定された次元で使用できる添字の最大値を返します。

```
Public Function UBound( _
   ByVal Array As System.Array, _
   Optional ByVal Rank As Integer = 1 _
) As Integer
```

パラメータ

Array

必ず指定します。任意のデータ型の配列です。ある次元で使用できる添字の最大値を調べる対象となる配列を指定します。

Rank

省略可能です。**Integer** 型の値です。使用できる添字の最大値を取得する次元を指定します。最初の次元なら 1、2 番目の次元なら 2 というように指定します。*Rank* を省略すると、1 が使用されます。

戻り値

Integer 型の値です。指定した次元で使用できる添え字の最大値です。*Array* に要素が 1 つしかない場合、**UBound** は 0 を返します。要素が長さゼロの文字列である場合など、*Array* に要素がない場合、**UBound** は -1 を返します。

例外

例外の種類	エラー番号	条件
ArgumentNullException	9	*Array* または **Nothing**
RankException	9	*Rank* < 1 です。または、*Rank* が *Array* のランクより大きな値です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

配列の添字は 0 から始まるため、次元の長さは、その次元で使用できる添字の最大値より 1 だけ大きい数値になります。

配列の次元と **UBound** 関数が返す値の関係を次に示します。

```
Dim a(100, 5, 4) As Byte
```

UBound の呼び出し	戻り値
`UBound(a, 1)`	100
`UBound(a, 2)`	5
`UBound(a, 3)`	4

UBound 関数を使って、配列内の要素の合計数を調べることができます。ただし、添字が 0 から始まることを考慮して、戻り値を調整する必要があります。前の例で使用した配列 `a` の合計サイズを計算するコード例は、次のとおりです。

```
Dim total As Integer
total = (UBound(A, 1) + 1) * (UBound(A, 2) + 1) * (UBound(A, 3) + 1)
```

`total` の計算結果は 3030、つまり、101 x 6 x 5 です。

使用例

UBound 関数を使って、配列の指定された次元で使用できる添字の最大値を調べるコード例は、次のとおりです。

VB

```
Dim highest, bigArray(10, 15, 20), littleArray(6) As Integer
highest = UBound(bigArray, 1)
highest = UBound(bigArray, 3)
highest = UBound(littleArray)
' The three calls to UBound return 10, 20, and 6 respectively.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : Information

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

LBound 関数 (Visual Basic)
Dim ステートメント (Visual Basic)
ReDim ステートメント (Visual Basic)
ArgumentException
RankException

UCase 関数 (Visual Basic)

指定された文字列を大文字に変換して文字列型 (**String**) または char 型 (**Char**) の値を返します。

```
Public Shared Function UCase(ByVal Value As Char) As Char
' -or-
Public Shared Function UCase(ByVal Value As String) As String
```

パラメータ
Value

必ず指定します。任意の有効な **String** または **Char** 式。

解説

小文字だけが大文字に変換されます。小文字のアルファベット以外の文字には影響ありません。

この関数は文字列を操作するときにアプリケーションのカルチャ情報を使用するため、大文字小文字がアプリケーションで使用しているロケールに対して正しく変換されます。

セキュリティに関するメモ：

比較処理または大文字/小文字変換処理の結果に基づいてアプリケーションのセキュリティ関連の決定を下す場合は、その処理を System.String.Compare(System.String,System.String,System.StringComparison) メソッドで実行し、*comparisonType* 引数に Ordinal または OrdinalIgnoreCase を渡してください。詳細については、「Visual Basic においてカルチャが文字列に与える影響」を参照してください。

使用例

UCase 関数を使って文字列を大文字に変換して返す例を次に示します。

VB

```
' String to convert.
Dim LowerCase As String = "Hello World 1234"
' Returns "HELLO WORLD 1234".
Dim UpperCase As String = UCase(LowerCase)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
文字列操作の概要
LCase 関数 (Visual Basic)
その他の技術情報
Visual Basic における文字列
Visual Basic の文字列の概要

Val 関数

指定した文字列に含まれる数値を適切なデータ型に変換して返します。

```
Public Overloads Function Val(ByVal InputStr As String) As Double
' -or-
Public Overloads Function Val(ByVal Expression As Object) As Double
' -or-
Public Overloads Function Val(ByVal Expression As Char) As Integer
```

パラメータ

Expression, *InputStr*

必ず指定します。有効な文字列型 (**String**) の式、オブジェクト型 (**Object**) の変数、または char 型 (**Char**) の値。*Expression* がオブジェクト型 (**Object**) である場合、その値は文字列型 (**String**) に変換できる必要があります。変換できない場合、ArgumentException エラーが発生します。

例外

例外の種類	エラー番号	条件
OverflowException	6	*InputStr* が大きすぎます。
InvalidCastException	13	数値の形式が誤っています。
ArgumentException	438	オブジェクト型 (**Object**) の式を文字列型 (**String**) に変換できません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

文字列中に数字以外の文字が見つかると、**Val** 関数は読み込みを中止します。円記号 (\) やコンマ (,) など、通常は数値の一部と見なされる記号や文字も、**Val** 関数は数値として解釈しません。ただし、**Val** 関数は基数を示すプリフィックス `&O` (8 進数) や `&H` (16 進数) は認識します。引数の文字列中に含まれる空白、タブ、ライン フィードは無視されます。

次のように呼び出すと、1615198 という値を返します。

```
Val(" 1615 198th Street N.E.")
```

次のように呼び出すと、10 進数の -1 という値を返します。

```
Val("&HFFFF")
```

メモ :

Val 関数は、ピリオド (.) だけを有効な小数点の記号として認識します。ピリオドと異なる小数点の記号が使われている可能性があるときは、**Val** 関数で文字列を数値に変換する代わりに、**CDbl** 関数または **CInt** 関数を使用してください。特定のカルチャでの数字の文字列表記を数値に変換するには、数値型の `Parse(String, IFormatProvider)` メソッドを使用します。たとえば、文字列を **Double** に変換する場合は System.Double.Parse(System.String,System.IFormatProvider) を使用します。

使用例

次の例では、**Val** 関数を使用して核文字列に含まれている数字を返します。**Val** は、数値、数値の修飾子、数値の区切り文字、空白として解釈できない文字が最初に現れた時点で変換を中止します。

VB

```
Dim valResult As Double
' The following line of code sets valResult to 2457.
valResult = Val("2457")
' The following line of code sets valResult to 2457.
valResult = Val(" 2 45 7")
' The following line of code sets valResult to 24.
valResult = Val("24 and 57")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Conversion**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
Str 関数
データ型変換関数
OverflowException
InvalidCastException
ArgumentException

VarType 関数 (Visual Basic)

変数のデータ型の分類を含む整数型 (**Integer**) の値を返します。

```
Public Function VarType(ByVal VarName As Object) As VariantType
```

パラメータ

VarName

必ず指定します。**Object** 変数です。**Option Strict** が **Off** の場合、構造体を除く任意のデータ型の変数を渡すことができます。

解説

VarType によって返される整数値は、VariantType 列挙型 のメンバです。

特別な場合の *VarName* に対して、**VarType** によって返される値を次に示します。

VarName **によって表されるデータ型**	**VarType によって返される値**
Nothing (Visual Basic)	**VariantType.Object**
DBNull	**VariantType.Null**
列挙型	基になるデータ型 (**SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、**ULong**)
配列	配列要素型と **VariantType.Array** のビットごとの OR
配列の配列	**VariantType.Object** および **VariantType.Array** のビットごとの OR
構造体 (System.ValueType)	**VariantType.UserDefinedType**
Exception	**VariantType.Error**
不明	**VariantType.Object**

使用例

次の例では、**VarType** 関数を使って、いくつかの変数に関するデータ型分類情報を返します。

VB

```
Dim testString As String = "String for testing"
Dim testObject As New Object
Dim testNumber, testArray(5) As Integer
Dim testVarType As VariantType
testVarType = VarType(testVarType)
' Returns VariantType.Integer.
testVarType = VarType(testString)
' Returns VariantType.String.
testVarType = VarType(testObject)
' Returns VariantType.Object.
testVarType = VarType(testNumber)
' Returns VariantType.Integer.
testVarType = VarType(testArray)
' Returns the bitwise OR of VariantType.Array and VariantType.Integer.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : Information

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

データ型の概要 (Visual Basic)
オブジェクト型 (Object)
VariantType 列挙型

VbTypeName 関数

変数の Visual Basic データ型名を含む文字列型 (**String**) の値を返します。

```
Public Function VbTypeName(ByVal UrtName As String) As String
```

パラメータ
UrtName

必ず指定します。共通言語ランタイムによって使用される型名を含む文字列型 (**String**) の変数です。

解説

VbTypeName は、共通言語ランタイムの型名に対応する Visual Basic の型名を返します。たとえば、*UrtName* に "Int32" または "System.Int32" を指定した場合、**VbTypeName** は "Integer" を返します。**VbTypeName** が *UrtName* の値を認識できない場合は、**Nothing** (文字列 "Nothing" ではなく) を返します。

UrtName に指定する型名は、Type クラスの MemberInfo 配列の Name プロパティなど、さまざまなソースから得ることができます。

使用例

次の例は、**VbTypeName** 関数を使って、いくつかの変数のデータ型名を返します。

VB

```
Dim sysDateName As String = "System.DateTime"
Dim sysShortName As String = "Int16"
Dim sysBadName As String = "Nonsense"
Dim testVbName As String
testVbName = VbTypeName(sysDateName)
' Returns "Date".
testVbName = VbTypeName(sysShortName)
' Returns "Short".
testVbName = VbTypeName(sysBadName)
' Returns Nothing.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Information**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
データ型の概要 (Visual Basic)
文字列型 (String) (Visual Basic)
SystemTypeName 関数

Weekday 関数 (Visual Basic)

曜日を表す数値を含む整数型 (**Integer**) の値を返します。

```
Public Function Weekday( _
   ByVal DateValue As DateTime, _
   Optional ByVal DayOfWeek As FirstDayOfWeek = FirstDayOfWeek.Sunday _
) As Integer
```

パラメータ

DateValue

必ず指定します。曜日を判断する日付型 (**Date**) の値です。

DayOfWeek

省略可能です。週の最初の曜日を指定する、**FirstDayOfWeek** 列挙値から選択した値です。省略すると、**FirstDayOfWeek.Sunday** が使用されます。

設定

DayOfWeek 引数の設定値は次のいずれかです。

列挙値	値	説明
FirstDayOfWeek.System	0	システムで設定されている週の最初の曜日
FirstDayOfWeek.Sunday	1	日曜日 (既定値)
FirstDayOfWeek.Monday	2	月曜日 (ISO 規格 8601、3.17 項に準拠)
FirstDayOfWeek.Tuesday	3	火曜日
FirstDayOfWeek.Wednesday	4	水曜日
FirstDayOfWeek.Thursday	5	木曜日
FirstDayOfWeek.Friday	6	金曜日
FirstDayOfWeek.Saturday	7	土曜日

例外

例外の種類	エラー番号	条件
ArgumentException	5	*DayOfWeek* が 0 より小さいか、または 7 を超えています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Weekday 関数によって返される値は、*DayOfWeek* 値で定義された曜日を週の第 1 日目とする *DateValue* の曜日を表す値です。たとえば、週の第 1 日目が月曜日として指定されているとき、日付が水曜日に該当する場合は 3 が返されます。

メモ :

Weekday は、System.Globalization 名前空間にある CultureInfo クラスの CurrentCulture プロパティに設定された現在のカレンダーを使用します。**CurrentCulture** の既定値は、[コントロール パネル] の設定によって決まります。

使用例

次の例は、**Weekday** 関数を使って、指定された日付の曜日を取得します。

VB

```
Dim oldDate As Date
Dim oldWeekDay As Integer
oldDate = #2/12/1969#
oldWeekDay = Weekday(oldDate)
' oldWeekDay now contains 4 because thisDate represents a Wednesday.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : DateAndTime

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Day 関数 (Visual Basic)
Month 関数 (Visual Basic)
Now プロパティ
WeekdayName 関数 (Visual Basic)
Year 関数 (Visual Basic)
DatePart 関数 (Visual Basic)
DateTime

WeekdayName 関数 (Visual Basic)

指定した曜日の名前を含む文字列型 (**String**) の値を返します。

```
Public Function WeekdayName( _
   ByVal Weekday As Integer, _
   Optional ByVal Abbreviate As Boolean = False, _
   Optional ByVal FirstDayOfWeekValue As FirstDayOfWeek = FirstDayOfWeek.System _
) As String
```

パラメータ

Weekday

必ず指定します。整数型 (**Integer**)。1 ～ 7 の曜日を示す数値で、1 は週の最初の曜日を示し、7 は週の最後の曜日を示します。最初と最後の曜日は、*FirstDayOfWeekValue* の設定に依存します。

Abbreviate

省略可能です。曜日名を短縮するかどうかを示すブール (**Boolean**) 値です。省略すると、既定値の **False** が使用され、曜日名は短縮されません。

FirstDayOfWeekValue

省略可能です。週の最初の曜日を指定する、**FirstDayOfWeek** 列挙値から選択した値です。省略すると、**FirstDayOfWeek.System** が使用されます。

設定

FirstDayOfWeekValue 引数の設定値は次のいずれかです。

列挙値	値	説明
FirstDayOfWeek.System	0	システム設定で指定された週の最初の曜日 (既定)
FirstDayOfWeek.Sunday	1	日曜日
FirstDayOfWeek.Monday	2	月曜日 (ISO 規格 8601、3.17 項に準拠)
FirstDayOfWeek.Tuesday	3	火曜日
FirstDayOfWeek.Wednesday	4	水曜日
FirstDayOfWeek.Thursday	5	木曜日
FirstDayOfWeek.Friday	6	金曜日
FirstDayOfWeek.Saturday	7	土曜日

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Weekday* が 1 より小さいか 7 を超えています。または *FirstDayOfWeekValue* が 0 より小さいか 7 を超えています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

WeekdayName から返される文字列は、入力引数だけではなく、Windows の [コントロール パネル] で設定される [地域のオプション] にも依存します。

メモ：
WeekdayName は、System.Globalization 名前空間にある CultureInfo クラスの CurrentCulture プロパティに設定された現在のカレンダーを使用します。**CurrentCulture** の既定値は、[コントロール パネル] の設定によって決まります。

使用例

次の例は、**Weekday** 関数を使用して、指定された日付から曜日を取得し、その番号を基に **WeekDayName** 関数を使用して曜日の名前を取得します。

VB

```
Dim oldDate As Date
Dim oldWeekDayName As String
oldDate = #2/12/1969#
oldWeekDayName = WeekdayName(Weekday(oldDate))
' oldWeekDayName now contains "Wednesday".
```

必要条件

名前空間： Microsoft.VisualBasic

モジュール：DateAndTime

アセンブリ： Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Day 関数 (Visual Basic)
Month 関数 (Visual Basic)
Now プロパティ
Weekday 関数 (Visual Basic)
MonthName 関数 (Visual Basic)
Year 関数 (Visual Basic)
DatePart 関数 (Visual Basic)
DateTime

Write 関数、WriteLine 関数

シーケンシャル出力モードで開いたファイルにデータを書き込みます。通常、**Write** 関数を使用して書き込んだデータは、**Input** 関数で読み込みます。

```
Public Sub Write( _
   ByVal FileNumber As Integer, _
   ByVal ParamArray Output As Object _
)
' -or-
Public Sub WriteLine( _
   ByVal FileNumber As Integer, _
   ByVal ParamArray Output() As Object _
)
```

パラメータ

FileNumber

必ず指定します。有効なファイル番号を表す整数型 (**Integer**) の式です。

Output

省略可能です。ファイルに書き込む、コンマで区切られた式。

例外

例外の種類	エラー番号	条件
IOException	52	*FileNumber* が存在していません。
IOException	54	ファイル モードが無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

Write および **WriteLine** 関数は、下位互換性を保つために提供されており、パフォーマンスに影響を与える可能性があります。非レガシ アプリケーションに対しては、**My.Computer.FileSystem** オブジェクトはより優れたパフォーマンスを発揮します。詳細については、「Visual Basic におけるファイル アクセス」を参照してください。

引数 *Output* を省略すると、空行がファイルに出力されます。複数の式をコンマで区切って含めることができます。

Write 関数は、**Print** 関数と異なり、ファイルにデータを書き込むときにデータ項目の間にはコンマを挿入し、文字列は引用符で囲みます。引数 outputlist には明示的に区切り文字を指定する必要はありません。**Write** を使用してデータをファイルに書き込む場合、サポートされるのは、数値、**Boolean**、日付、null、**Error** のデータ形式のみです。次の一般的な前提が適用されるので、ロケールに関係なく **Input** を使ってデータを正しく読み取り、解釈できます。

- 数値データ型のデータは、小数点記号としてピリオド (.) が常に使用されます。
- ブール型 (**Boolean**) のデータは、`#TRUE#` または `#FALSE#` という文字列で出力されます。キーワード **True** と **False** は、ロケールに応じて翻訳されることはありません。
- 日付型のデータは、共通日付形式を使用して書き込まれます。日付や時間の構成要素がない場合やゼロの場合は、指定されている部分だけのデータがファイルに書き込まれます。
- 引数 *Output* のデータが空の場合は、ファイルには何も書き込まれません。引数 *Output* のデータが NULL の場合は、文字列 `#NULL#` がファイルに書き込まれます。
- エラー データは、文字列 `#ERROR errorcode#` という形式で出力されます。キーワード **Error** は、ロケールに応じて翻訳されることはありません。

WriteLine 関数はファイルに引数 *Output* の最後の文字を出力した後、復帰/改行文字 (`Chr(13) + Chr(10)`) を挿入します。

引用符を文字列に埋め込むには、二重の引用符 ("") を使用します。たとえば、次のようになります。

VB

```
Dim x As String = "Double quotation marks aren't ""difficult"" to handle."
```

`Double quotation marks aren't "difficult" to handle` という値の文字列を返します。

Write または **WriteLine** 関数を使用してファイルにデータを書き込むには、**FileIOPermissionAccess** 列挙体からの **Append** アクセスが必要です。詳細については、「FileIOPermissionAccess 列挙体」を参照してください。

使用例

Write 関数を使って、シーケンシャル ファイルにデータを書き込みます。

VB

```
FileOpen(1, "TESTFILE", OpenMode.Output) ' Open file for output.
Write(1, "This is a test.")  ' Print text to file.
WriteLine(1)  ' Print blank line to file.
WriteLine(1, "Zone 1", TAB(), "Zone 2")   ' Print in two print zones.
WriteLine(1, "Hello", " ", "World")     ' Separate strings with space.
WriteLine(1, SPC(5), "5 leading spaces ")    ' Print five leading spaces.
WriteLine(1, TAB(10), "Hello")   ' Print word at column 10.

' Assign Boolean, Date, and Error values.
Dim aBool As Boolean
Dim aDate As DateTime
aBool = False
aDate = DateTime.Parse("February 12, 1969")

' Dates and Booleans are translated using locale settings of
' your system.
WriteLine(1, aBool, " is a Boolean value")
WriteLine(1, aDate, " is a date")
FileClose(1)   ' Close file.
```

スマート デバイス開発者のためのメモ

この関数はサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **FileSystem**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic でテキストをファイルに書き込む

方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

関連項目

Input 関数

FileOpen 関数

Print 関数、PrintLine 関数

その他の技術情報

Visual Basic におけるファイル アクセス

Year 関数 (Visual Basic)

年を表す 1 ～ 9999 の整数型 (**Integer**) の値を返します。

```
Public Function Year(ByVal DateValue As DateTime) As Integer
```

パラメータ

DateValue

必ず指定します。年を抽出する日付型 (**Date**) の値。

解説

DatePart を呼び出し、*Interval* 引数に **DateInterval.Year** を指定することによっても年を取得できます。

使用例

次の例は、**Year** 関数を使って、指定された日付の "年" で表される部分を取得します。開発環境では、日付リテラルは、コード記述時のロケールで設定されている短い日付の形式で表示されます。

VB

```
Dim thisDate As Date
Dim thisYear As Integer
thisDate = #2/12/1969#
thisYear = Year(thisDate)
' thisYear now contains 1969.
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : DateAndTime

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Day 関数 (Visual Basic)
Month 関数 (Visual Basic)
Now プロパティ
Weekday 関数 (Visual Basic)
DatePart 関数 (Visual Basic)
DateTime
ArgumentOutOfRangeException

キーワード (Visual Basic)

ここで紹介するトピックには、Visual Basic の言語キーワード (予約済みと予約済みでないものの両方) の表が記載されています。

このセクションの内容

関連するセクション

Visual Basic リファレンス

Visual Basic

キーワード A ～ E

次の表は、Visual Basic 言語のキーワードの一覧です。

Alias	AddHandler	Ansi	As
Assembly	Auto	Binary	ByRef
ByVal	Case	Catch	Class
Custom	Default	DirectCast	Each
Else	ElseIf	End	Error

スマート デバイス開発者のためのメモ

Ansi、**Auto**、および **End** は、スマート デバイス アプリケーションではサポートされません。

参照

関連項目

キーワード F ～ O

キーワード P ～ Z

その他の技術情報

Visual Basic リファレンス

Alias

外部プロシージャが DLL の中では別の名前で宣言されていることを示すキーワードです。

解説

キーワード **Alias** は、次の構文で使用します。

Declare ステートメント

参照

関連項目

Visual Basic 言語のキーワード

AddHandler

AddHandler は、実行時にイベントをイベント ハンドラに関連付けます。または、イベント ハンドラを追加するときにどのコードを実行するかを宣言します。

解説

キーワード **AddHandler** は、次の構文で使用します。

Event ステートメント

キーワード **AddHandler** は、カスタムの **AddHandler** アクセサを宣言します。

AddHandler ステートメント

キーワード **AddHandler** は、イベントをイベント ハンドラに関連付けます。

参照

処理手順

方法 : ブロックを回避するイベントを宣言する

方法 : メモリの使用量を節約するイベントを宣言する

Ansi

宣言されている外部プロシージャの名前に関係なく、すべての文字列を ANSI (American National Standards Institute) 値にマーシャリングすることを指定します。

プロジェクトの外部に定義されたプロシージャを呼び出すと、Visual Basic コンパイラはプロシージャを正しく呼び出すために必要な情報にアクセスできません。この情報には、プロシージャが置かれている場所と識別する方法、呼び出しシーケンス、戻り値の型、および使用する文字列の文字セットが含まれます。Declare ステートメント は外部プロシージャの参照を作成して、この必要な情報を提供します。

Declare ステートメントの *charsetmodifier* の部分では、外部プロシージャの呼び出しの間に文字列をマーシャリングするための文字セット情報を提供します。これは、Visual Basic が指定された外部プロシージャ名を、外部ファイルの中から検索する方法にも影響します。**Ansi** 修飾子を指定すると、Visual Basic はすべての文字列を ANSI 値にマーシャリングし、プロシージャの検索中にプロシージャ名を変更しなくなります。

文字セットに修飾子を指定しない場合は **Ansi** が既定値になります。

解説

Ansi 修飾子は次の構文で使用します。

Declare ステートメント

スマート デバイス開発者のためのメモ

このキーワードはサポートされていません。

参照

関連項目

Auto
Unicode (Visual Basic)
Visual Basic 言語のキーワード

As (Visual Basic)

宣言ステートメント内、またはジェネリック型のパラメータの入力規則リスト内のデータ型を識別する **As** 句を開始します。

解説

キーワード **As** は、次の構文で使用します。

Class ステートメント

Const ステートメント

Declare ステートメント

Delegate ステートメント

Dim ステートメント

Enum ステートメント

Event ステートメント

For...Next ステートメント

For Each...Next ステートメント

Function ステートメント

Interface ステートメント

Operator ステートメント

Property ステートメント

Structure ステートメント

Sub ステートメント

Try...Catch...Finally ステートメント

参照

処理手順

方法 : 新しい変数を作成する

関連項目

型リスト

Visual Basic 言語のキーワード

概念

Visual Basic におけるデータ型

Visual Basic での変数宣言

Visual Basic におけるジェネリック型

Assembly

ソース ファイルの冒頭にある属性がアセンブリ全体に適用されることを示します。

解説

多くの属性は、クラスまたはプロパティなどの個別のプログラミング要素に適用されます。このような属性は、宣言ステートメントに直接、山かっこ (< >) で囲んだ属性ブロックを割り当てることで適用します。

属性が、次の要素だけでなくアセンブリ全体に適用される場合、属性ブロックはソース ファイルの冒頭に記述し、**Assembly** キーワードで属性を明示します。これが現在のアセンブリ モジュールに適用される場合、Module キーワードを使用します。

AssemblyInfo.vb ファイルでは、属性をアセンブリに適用できます。この場合、メインのソース コード ファイルで属性ブロックを使用する必要はありません。

参照

関連項目

Module (Visual Basic)

概念

属性の適用

Auto

Visual Basic で、宣言されている外部プロシージャの外部名に基づいた .NET Framework の規則に従って文字列をマーシャリングするよう指定します。

プロジェクト外部で定義されたプロシージャを呼び出す場合、Visual Basic コンパイラは、そのプロシージャを正しく呼び出すために必要な情報にアクセスできません。この情報には、プロシージャの場所、識別方法、呼び出しシーケンスと戻り値のデータ型、使用する文字列の文字セットが含まれます。Declare ステートメント は、外部プロシージャへの参照を作成し、この必須情報を提供します。

Declare ステートメントの *charsetmodifier* は、外部プロシージャの呼び出し時に文字列をマーシャリングするための文字セットの情報を提供します。また、Visual Basic が外部ファイルで外部プロシージャ名を検索する方法にも影響します。**Auto** 修飾子は、Visual Basic が .NET Framework の規則に従って文字列をマーシャリングすること、また、実行時プラットフォームの基本文字セットを確認し、最初の検索が失敗した場合には外部プロシージャ名を変更することを指定します。詳細については、「Declare ステートメント」の「文字セット」を参照してください。

文字セットの修飾子を指定しない場合は **Ansi** が既定値です。

解説

Auto 修飾子は、次の場合に使用できます。

Declare ステートメント

スマート デバイス開発者のためのメモ

このキーワードはサポートされていません。

参照

関連項目

Ansi
Unicode (Visual Basic)
Visual Basic 言語のキーワード

Binary

文字列を比較するメソッドに、厳密なバイナリの並べ替え順序を設定します。

解説

Option Compare ステートメントを使用する場合は、ソース ファイル内で他のソース ステートメントより前に記述する必要があります。これにより、そのソース ファイルから生成されるすべてのコードの文字列比較に対して設定が有効になります。

Binary を使用する状況

文字列の要素を、実際のバイナリ値に基づいて比較することが必要になる場合があります。具体的には、テキストに変換されない文字が文字列に含まれている場合などです。このような場合、大文字と小文字を区別せずに比較するなどの際に、アルファベット順による文字列の比較に偏りが生じるのは望ましくありません。バイナリのままで比較するための指定には、**Option Compare** に **Binary** を設定します。

キーワード **Binary** は、次の構文で使用します。

Option Compare ステートメント

参照

関連項目

Text

Visual Basic 言語のキーワード

ByRef

呼び出されたプロシージャは、呼び出しコードの引数の基になる変数の値を変更できることを、引数を渡す方法として指定します。

解説

修飾子 **ByRef** は、次の構文で使用します。

Declare ステートメント

Function ステートメント

Sub ステートメント

参照

関連項目

Visual Basic 言語のキーワード

概念

引数の値渡しおよび参照渡し

ByVal

呼び出されたプロシージャまたはプロパティは、呼び出しコードの引数の基になる変数の値を変更できないことを、引数を渡す方法として指定します。

解説

修飾子 **ByVal** は、次の構文で使用します。

Declare ステートメント

Function ステートメント (Visual Basic)

Operator ステートメント

Property ステートメント

Sub ステートメント (Visual Basic)

参照

関連項目

Visual Basic 言語のキーワード

概念

引数の値渡しおよび参照渡し

Case (Visual Basic)

式の値を検査する対象となる値または値の集約を示すキーワードです。

解説

キーワード **Case** は、次の構文で使用します。

Select...Case ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Catch (Visual Basic)

Try ブロック内で特定の例外が発生した場合に実行するステートメント ブロックを開始します。

解説

各 **Try** 構造体には、**Finally** ブロック、または最低 1 つの **Catch** ブロックが含まれている必要があります。

Catch を使用する状況

コード内の特定のセクションで特定の例外が発生することが予想される場合、コードを **Try** ブロック内に記述し、**Catch** ブロックを使用してコントロールを保持して、例外が発生した場合にこれを処理するようにします。**Try** 構造体の中で、**Catch** ブロックは必要に応じていくつでも使用できます。

キーワード **Catch** は、次の構文で使用します。

Try...Catch...Finally ステートメント (Visual Basic)

参照

関連項目

Finally (Visual Basic)
Visual Basic 言語のキーワード

Class (Visual Basic)

ジェネリック型のパラメータに制約を加え、これに渡す型引数を必ず参照型にするか、そうでなければ Class ステートメント (Visual Basic) を使用するように要求します。

解説

ジェネリック型に型パラメータを宣言するとき、制約、つまり 1 つ以上の要件を加えて、その型のパラメータに渡す型引数を制限できます。詳細については、「Visual Basic におけるジェネリック型」の「制約」を参照してください。

制約に定義可能な要件には、型引数が参照型 (**String** 型、配列、デリゲート、クラスから作成されたオブジェクトなど) であることなどが挙げられます。これを指定するには、制約に **Class** キーワードを指定します。

これ以外に、型引数が値型 (構造体、列挙体、基本データ型など) であること、という要件も定義できます。値型の要件を指定するには、制約に Structure (Visual Basic) キーワードを指定します。

Class または **Structure** を、制約に必ず指定する必要はありません。同じ制約に、この 2 つを同時に指定することはできません。

Class 制約は、Class ステートメント (Visual Basic) とは異なります。

参照

関連項目

Structure (Visual Basic)
Class ステートメント (Visual Basic)

概念

Visual Basic におけるジェネリック型
値型と参照型

Custom

イベントに対してハンドラの追加、ハンドラの削除、イベントの発生に関する追加コードが記述されていることを表します。

解説

キーワード **Custom** は、次の構文で使用します。

Event ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Default (Visual Basic)

プロパティがクラス、構造体、またはインターフェイスの既定のプロパティであることを示すキーワードです。

解説

クラス、構造体、インターフェイスでは、最大で 1 つのプロパティを既定のプロパティとして定義できます。ただし、そのプロパティは 1 つ以上のパラメータを受け取るものであることが必要です。Visual Basic で、メンバを指定せずにクラスや構造体を参照するコードを作成すると、既定のプロパティを参照していると判断されます。

既定のプロパティを使用すると、ソース コードに記述する文字の量が少し減りますが、コードの読みやすさが低下します。呼び出しコードからクラスや構造体が明確に区別できない場合にクラス名または構造体名を使って参照すると、その参照がクラスと構造体のどちらに対するものなのか、また既定のプロパティを参照しているかどうかもはっきりしません。このような場合、コンパイル エラーまたはランタイム エラーや論理エラーが発生する可能性があります。

Option Strict ステートメント を使ってコンパイラの型チェックを常に **On** に設定しておくことで、既定のプロパティのエラーが発生する可能性をいくらか低下できます。

定義済みのクラスまたは構造体を使ってコードを作成する場合には、既定のプロパティが設定されているかどうかを調べ、設定されていればその名前を確認しておくことが必要です。

以上のような難点があるため、既定のプロパティは定義しないことをお勧めします。コードの読みやすさの点からも、すべてのプロパティを、既定のプロパティであっても明示的に参照することを心がけてください。

Default 修飾子は次の構文で使用します。

Property ステートメント

参照

処理手順

方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)

関連項目

Visual Basic 言語のキーワード

概念

既定のプロパティ

DirectCast

継承または実装に基づく型変換処理を実行します。

解説

DirectCast は Visual Basic のランタイム ヘルパー ルーチンを変換に使用しません。このため、オブジェクト型 (**Object**) との間で変換を行う場合に、**CType** よりもいくらかパフォーマンスがよくなります。

キーワード **DirectCast** は、CType 関数 およびキーワード TryCast と同じ方法で使用します。1 つ目の引数に式を指定し、2 つ目の引数に変換後の型を指定します。**DirectCast** の場合、この 2 つの引数のデータ型の間に、継承または実装の関係があることが必要です。つまり、一方の型が他方の型を継承または実装している必要があります。

エラーと失敗

DirectCast は、継承または実装の関係が存在しないことを検出すると、コンパイル エラーを生成します。しかし、コンパイル エラーが発生しなければ変換が成功であるとは限りません。縮小変換が指定されている場合は、実行時に失敗する可能性があります。このとき、ランタイムによって InvalidCastException エラーがスローされます。

型変換のキーワード

型変換キーワードの比較は次のようになります。

キーワード	データ型	引数の関係	ランタイム エラー
CType 関数	任意のデータ型	2 つのデータ型の間で、拡大変換または縮小変換が定義される必要がある	**InvalidCastException** をスロー
DirectCast	任意のデータ型	一方の型が他方の型を継承または実装する必要がある	**InvalidCastException** をスロー
TryCast	参照型のみ	一方の型が他方の型を継承または実装する必要がある	Nothing (Visual Basic) を返す

使用例

DirectCast の 2 つの使用例を次に示します。1 つは実行時に失敗し、もう 1 つは成功します。

VB

```
Dim q As Object = 2.37
Dim i As Integer = CType(q, Integer)
' The following conversion fails at run time
Dim j As Integer = DirectCast(q, Integer)
Dim f As New System.Windows.Forms.Form
Dim c As System.Windows.Forms.Control
' The following conversion succeeds.
c = DirectCast(f, System.Windows.Forms.Control)
```

この例では、q のランタイム型は倍精度浮動小数点数型 (**Double**) です。**Double** は **Integer** に変換可能なので、**CType** は成功します。しかし、最初の **DirectCast** は実行時に失敗します。**Double** のランタイム型は **Integer** に変換可能ですが、継承の関係にはないからです。2 番目の **DirectCast** は、Form 型から Control 型への変換です。**Form** は **Control** を継承しているので成功します。

参照

概念

拡大変換と縮小変換

暗黙の型変換と明示的な型変換

Each

For Each ループで使用するループ変数を指定します。

解説

キーワード **Each** は、次の場面で使用します。

For Each...Next ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Else (Visual Basic)

他の条件グループのステートメントがいずれも実行またはコンパイルされなかった場合に、実行またはコンパイルされるステートメント グループを記述します。

解説

キーワード **Else** は、次の構文で使用します。

If...Then...Else ステートメント

Select...Case ステートメント

#If...Then...#Else ディレクティブ

参照

関連項目

Visual Basic 言語のキーワード

ElseIf (Visual Basic)

1 つ前の条件テストが満たされなかった場合にテストする条件を指定します。

解説

キーワード **ElseIf** は、次の構文で使用します。

If...Then...Else ステートメント

#If...Then...#Else ディレクティブ

参照

関連項目

Visual Basic 言語のキーワード

End (Visual Basic)

追加のキーワードを続けて記述すると、そのキーワードが指定されたステートメント ブロックの定義を終了します。

```
End AddHandler
End Class
End Enum
End Event
End Function
End Get
End If
End Interface
End Module
End Namespace
End Operator
End Property
End RaiseEvent
End RemoveHandler
End Select
End Set
End Structure
End Sub
End SyncLock
End Try
End While
End With
```

指定項目

End

必ず指定します。プログラミング要素の定義を終了します。

AddHandler

カスタムの Event ステートメント で、対応する **AddHandler** ステートメントによって開始された **AddHandler** アクセサを終了する場合に必要です。

Class

対応する Class ステートメント (Visual Basic) によって開始されたクラス定義を終了する場合に必要です。

Enum

対応する Enum ステートメント (Visual Basic) によって開始された列挙値の定義を終了する場合に必要です。

Event

対応する Event ステートメント によって開始された Custom イベントの定義を終了する場合に必要です。

Function

対応する Function ステートメント (Visual Basic) によって開始された **Function** プロシージャの定義を終了する場合に必要です。**End Function** ステートメントが実行されると、呼び出しコードに制御が返されます。

Get

対応する Get ステートメント によって開始された **Property** プロシージャの定義を終了する場合に必要です。**End Get** ステートメントが実行されると、プロパティの値を要求したステートメントに制御が返されます。

If

対応する **If** ステートメントによって開始された **If**...**Then**...**Else** ブロックの定義を終了する場合に必要です。If...Then...Else ステートメント (Visual Basic) を参照してください。

Interface

対応する Interface ステートメント (Visual Basic) によって開始されたインターフェイス定義を終了する場合に必要です。

Module

対応する Module ステートメント によって開始されたモジュール定義を終了する場合に必要です。

Namespace

対応する Namespace ステートメント によって開始された名前空間定義を終了する場合に必要です。

Operator

対応する Operator ステートメント によって開始された演算子の定義を終了する場合に必要です。

Property

対応する Property ステートメント によって開始されたプロパティ定義を終了する場合に必要です。

RaiseEvent

カスタムの Event ステートメント で、対応する **RaiseEvent** ステートメントによって開始された **RaiseEvent** アクセサを終了する場合に必要です。

RemoveHandler

カスタムの Event ステートメント で、対応する **RemoveHandler** ステートメントによって開始された **RemoveHandler** アクセサを終了する場合に必要です。

Select

対応する **Select** ステートメントによって開始された **Select**...**Case** ブロックの定義を終了する場合に必要です。Select...Case ステートメント (Visual Basic) を参照してください。

Set

対応する Set ステートメント (Visual Basic) によって開始された **Property** プロシージャの定義を終了する場合に必要です。**End Set** ステートメントが実行されると、プロパティの値を設定するステートメントに制御が返されます。

Structure

対応する Structure ステートメント によって開始された構造体定義を終了する場合に必要です。

Sub

対応する Sub ステートメント (Visual Basic) によって開始された **Sub** プロシージャの定義を終了する場合に必要です。**End Sub** ステートメントが実行されると、呼び出しコードに制御が返されます。

SyncLock

対応する **SyncLock** ステートメントによって開始された **SyncLock** ブロックの定義を終了する場合に必要です。SyncLock ステートメント を参照してください。

Try

対応する **Try** ステートメントによって開始された **Try**...**Catch**...**Finally** ブロックの定義を終了する場合に必要です。Try...Catch...Finally ステートメント (Visual Basic) を参照してください。

While

対応する **While** ステートメントによって開始された **While** ループの定義を終了する場合に必要です。While...End While ステートメント (Visual Basic) を参照してください。

With

対応する **With** ステートメントによって開始された **With** ブロックの定義を終了する場合に必要です。With...End With ステートメント (Visual Basic) を参照してください。

解説

追加のキーワードを指定しないで End ステートメント を使うと、コードの実行がすぐに終了します。

シャープ記号 (#) を前に付けると、**End** キーワードは対応するディレクティブが指定したプリプロセス ブロックを終了します。

#End

必ず指定します。プリプロセス ブロックの定義を終了します。

#ExternalSource

対応する #ExternalSource ディレクティブ によって開始された外部ソース ブロックを終了する場合に必要です。

#If

対応する **#If** ディレクティブによって開始された条件付きコンパイル ブロックを終了する場合に必要です。#If...Then...#Else ディレクティブ を参照してください。

#Region

対応する #Region ディレクティブ によって開始されたソース領域ブロックを終了する場合に必要です。

スマート デバイス開発者のためのメモ

追加のキーワードを指定しない **End** ステートメントは、サポートされていません。

参照
関連項目
End ステートメント

Error (Visual Basic)

ランタイム エラーをシミュレートするか、ランタイムエラーに応答します。

解説

キーワード **Error** は、次の構文で使用します。

Error ステートメント

On Error ステートメント

参照

関連項目

Visual Basic 言語のキーワード

その他の技術情報

Visual Basic での例外およびエラー処理

キーワード F ～ O

次の表は、Visual Basic 言語のキーワードの一覧です。

False	Finally	For	Friend
グローバル	Handles	Implements	In
Is	Lib	Loop	Me
Module	MustInherit	MustOverride	MyBase
MyClass	Narrowing	New	Next
Nothing	NotInheritable	NotOverridable	Of
Off	On	Option	Optional
Overloads	Overridable	Overrides	

参照

関連項目

キーワード A ～ E

キーワード P ～ Z

その他の技術情報

Visual Basic リファレンス

False (Visual Basic)

条件テストに失敗した場合の **Boolean** 値を表します。

解説

ブール型 (Boolean) (Visual Basic) 値は数字としては格納されず、格納された値は数字と等しくなりません。コードを作成する際、**True** および **False** に相当する数値を利用しないようにしてください。ブール型 (**Boolean**) の変数に対しては、できるだけ本来の論理値だけを使用してください。ブール型 (**Boolean**) と数値を混在させる必要がある場合は、選択した変換メソッドを十分理解してから行ってください。詳細については、「データ型のトラブルシューティング」の「Boolean 型が数値型に正確に変換されない」を参照してください。

参照

処理手順

データ型のトラブルシューティング

関連項目

ブール型 (Boolean) (Visual Basic)

True (Visual Basic)

Finally (Visual Basic)

Try 構造体を終了する前に実行するステートメント ブロックを開始します。

解説

各 **Try** 構造体には、**Finally** ブロック、または最低 1 つの **Catch** ブロックが含まれている必要があります。

メモ：
次の場合、コントロールは **Try** または **Catch** ブロックから、対応する **Finally** ブロックに移りません。

- **Try** または **Catch** ブロック内に End ステートメント がある場合
- **Try** または **Catch** ブロックで StackOverflowException がスローされた場合

Finally を使用する状況

Try 構造体を終了する前に実行するステートメントがある場合は、**Finally** ブロックを使用します。コントロールは **Try** 構造体に移る直前に **Finally** ブロックに移ります。これは、**Try** 構造体の中で例外が発生した場合でも同様です。

キーワード **Finally** は、次の構文で使用します。

Try...Catch...Finally ステートメント (Visual Basic)

参照

関連項目

Catch (Visual Basic)
Visual Basic 言語のキーワード

For (Visual Basic)

異なる値のループ変数で繰り返されるループ処理を開始します。

解説

キーワード **For** は、次の構文で使用します。

For...Next ステートメント

For Each...Next ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Friend (Visual Basic)

1 つまたは複数の宣言されたプログラミング要素が、その宣言を含むアセンブリからのみアクセス可能かどうかを示します。

解説

規則

- **宣言コンテキスト** **Friend** は、モジュール レベル、インターフェイス レベル、または名前空間レベルでのみ使用できます。つまり、**Friend** の宣言コンテキストは、ソース ファイル、名前空間、インターフェイス、モジュール、クラス、構造体のいずれかである必要があり、プロシージャでは宣言できません。
- **修飾子の結合** **Friend** 修飾子は、Protected (Visual Basic) 修飾子と組み合わせて同じ宣言内で使用できます。この組み合わせを使うと、宣言される要素にはフレンド アクセスとプロテクト アクセスの両方が設定され、同じアセンブリ、要素自体のクラス、およびすべての派生クラスから要素にアクセスできます。**Protected Friend** は、クラスのメンバでのみ指定できます。

動作

- **アクセス レベル** 宣言コンテキスト内のすべてのコードは、自らの要素にアクセスできます。同じアセンブリにコンパイルされる他のクラス、構造体、モジュール内のコードは、そのアセンブリ内のすべての **Friend** 要素にアクセスできます。

  フレンド アクセスは、プロテクト アクセスのスーパーセットでもサブセットでもありません。
- **アクセス修飾子** アクセス レベルを指定するキーワードは、*access modifiers* と呼ばれます。アクセス修飾子の比較については、「Visual Basic でのアクセス レベル」を参照してください。

フレンドを使用する状況

多くの場合、クラスや構造体のようなプログラミング要素は、それを宣言したコンポーネント内だけでなく、アセンブリ全体で使用できるようにします。しかし、アプリケーションが独自である場合など、アセンブリ外部のコードからはアクセスできないようにする必要がある場合もあります。このような形で要素へのアクセスを制限するには、要素を **Friend** で宣言します。

フレンド アクセスは、アプリケーションのプログラミング要素に望ましいレベルとして多く利用されます。特に宣言しない場合、インターフェイス、モジュール、クラス、構造体の既定のアクセス レベルは **Friend** です。

Friend 修飾子は、次の場合に使用できます。

Class ステートメント

Const ステートメント

Declare ステートメント

Delegate ステートメント

Dim ステートメント

Enum ステートメント

Event ステートメント

Function ステートメント

Interface ステートメント

Module ステートメント

Property ステートメント

Structure ステートメント

Sub ステートメント

参照

関連項目

Public (Visual Basic)
Protected (Visual Basic)
Private (Visual Basic)

概念

Visual Basic でのアクセス レベル
Visual Basic におけるプロシージャ

その他の技術情報

構造体 : 独自のデータ型

クラスについて

Global

.NET Framework プログラミング要素へのアクセスを、名前空間を構成してブロックした場合に、そのアクセスを可能にします。

解説

入れ子になった階層構造の名前空間を定義すると、その階層構造の内部にあるコードが .NET Framework の System 名前空間にアクセスするのを防止できます。`SpecialSpace.System` 名前空間に定義された階層構造が **System** へのアクセスを防止する例は、次のようになります。

```
Namespace SpecialSpace
    Namespace System
        Class abc
            Function getValue() As System.Int32
                Dim n As System.Int32
                Return n
            End Function
        End Class
    End Namespace
End Namespace
```

この場合、`SpecialSpace.System` に `Int32` が定義されていないため、Visual Basic コンパイラは System.Int32 への参照を正しく解決できません。**Global** キーワードを使用すると、修飾チェインを .NET Framework クラス ライブラリの最も外側のレベルで開始できます。これにより、クラス ライブラリの **System** 名前空間や、その他のあらゆる名前空間を指定できるようになります。次に例を示します。

```
Namespace SpecialSpace
    Namespace System
        Class abc
            Function getValue() As Global.System.Int32
                Dim n As Global.System.Int32
                Return n
            End Function
        End Class
    End Namespace
End Namespace
```

Global を指定しているため、他のルート レベルの名前空間 (Microsoft.VisualBasic など)、およびプロジェクトに関連する任意の名前空間にアクセスできます。

キーワード **Global** は、次の構文で使用します。

Class ステートメント (Visual Basic)

Const ステートメント (Visual Basic)

Declare ステートメント

Delegate ステートメント

Dim ステートメント (Visual Basic)

Enum ステートメント (Visual Basic)

Event ステートメント

For...Next ステートメント (Visual Basic)

For Each...Next ステートメント (Visual Basic)

Function ステートメント (Visual Basic)

Interface ステートメント (Visual Basic)

Operator ステートメント

Property ステートメント

Structure ステートメント

Sub ステートメント (Visual Basic)

Try...Catch...Finally ステートメント (Visual Basic)

Using ステートメント (Visual Basic)

参照

関連項目

Namespace ステートメント
.NET Framework クラス ライブラリ リファレンス
System
Microsoft.VisualBasic

Handles

プロシージャが特定のイベントを処理することを宣言します。

```
proceduredeclaration Handles eventlist
```

指定項目
proceduredeclaration

イベントを処理するプロシージャの **Sub** プロシージャ宣言。

eventlist

proceduredeclaration が処理するイベントのリスト。現在のクラスに対する基本クラス、または **WithEvents** キーワードを使って宣言されているオブジェクトのどちらかが発生させるイベントを指定する必要があります。

解説

プロシージャ宣言の末尾にキーワード **Handles** を指定すると、そのプロシージャは、キーワード **WithEvents** を使って宣言されたオブジェクト変数が発生させるイベントを処理します。**Handles** キーワードは、基本クラスからのイベントを処理するために派生クラスで使用することもできます。

プロシージャのシグネチャは、*eventlist* の各イベントのシグネチャと一致している必要があります。

Handles キーワードと **AddHandler** ステートメントのどちらを使用しても、特定のプロシージャが特定のイベントを処理することを指定できますが、この 2 つには違いがあります。プロシージャを定義して、それが特定のイベントを処理することを指定する場合は、**Handles** キーワードを使います。一方、**AddHandler** ステートメントは実行時にプロシージャをイベントに関連付けます。詳細については、「AddHandler ステートメント」を参照してください。

カスタム イベントの場合は、プロシージャがイベント ハンドラとして追加されたときに、アプリケーションがイベントの **AddHandler** アクセサを呼び出します。カスタム イベントの詳細については、「Event ステートメント」を参照してください。

使用例
VB

```
Public Class ContainerClass
    ' Module or class level declaration.
    WithEvents Obj As New Class1

    Public Class Class1
        ' Declare an event.
        Public Event Ev_Event()
        Sub CauseSomeEvent()
            ' Raise an event.
            RaiseEvent Ev_Event()
        End Sub
    End Class

    Sub EventHandler() Handles Obj.Ev_Event
        ' Handle the event.
        MsgBox("EventHandler caught event.")
    End Sub

    ' Call the TestEvents procedure from an instance of the ContainerClass
    ' class to test the Ev_Event event and the event handler.
    Public Sub TestEvents()
        Obj.CauseSomeEvent()
    End Sub
End Class
```

次に示すのは、派生クラスで **Handles** ステートメントを使って基本クラスからのイベントを処理する方法の例です。

VB

```
Public Class BaseClass
    ' Declare an event.
    Event Ev1()
End Class
Class DerivedClass
    Inherits BaseClass
    Sub TestEvents() Handles MyBase.Ev1
        ' Add code to handle this event.
    End Sub
End Class
```

参照

関連項目

WithEvents

AddHandler ステートメント

RemoveHandler ステートメント

Event ステートメント

RaiseEvent ステートメント

概念

イベントとイベント ハンドラ

Implements (Visual Basic)

クラスまたは構造体のメンバが、インターフェイスで定義されているメンバを実装することを示すキーワードです。

解説

Implements キーワードは、Implements ステートメントとは異なります。**Implements** ステートメントは、クラスまたは構造体が 1 つ以上のインターフェイスを実装することを指定するために使用します。そして、各メンバがどのインターフェイスのどのメンバを実装しているかを、**Implements** キーワードを使って指定します。

クラスまたは構造体がインターフェイスを実装する場合は、Class ステートメント (Visual Basic) または Structure ステートメントの直後に **Implements** ステートメントを記述し、インターフェイスで定義されているすべてのメンバを実装する必要があります。

再実装

派生クラスでは、基本クラスで既に実装されているインターフェイスのメンバを再実装できます。これは次の点で基本クラスのメンバのオーバーライドとは異なります。

- 基本クラスのメンバは、再実装するために Overridable する必要がありません。
- メンバを別の名前で再実装できます。

キーワード **Implements** は、次の構文で使用します。

Event ステートメント

Function ステートメント (Visual Basic)

Property ステートメント

Sub ステートメント (Visual Basic)

参照

関連項目

Implements ステートメント
Interface ステートメント (Visual Basic)
Class ステートメント (Visual Basic)
Structure ステートメント

概念

Implements キーワードおよび Implements ステートメント

In (Visual Basic)

ループ変数が **For Each** ループ内で繰り返し処理するグループを指定します。

解説

キーワード **In** は、次の構文で使用します。

For Each...Next ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Is (Visual Basic)

比較を行う **Is** 句の概要を説明します。

解説

Is キーワードは、**TypeOf**...**Is** 式の中で **TypeOf** キーワードと組み合わせて使用され、式の実行時の型が特定の型と互換性があるかどうかをテストします。

キーワード **Is** は、次の構文で使用します。

Is 演算子

Select...Case ステートメント

参照

関連項目

TypeOf 演算子 (Visual Basic)
Visual Basic 言語のキーワード

Lib

Lib 句は、外部プロシージャを含む外部ファイル (DLL またはコード リソース) であることを示します。

解説

キーワード **Lib** は、次の構文で使用します。

Declare ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Loop

Do ステートメントで開始されたループを終了します。

解説

キーワード **Loop** は、次の構文で使用します。

Do...Loop ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Me

コードが実行されているクラスまたは構造体の特定のインスタンスを参照できます。

解説

Me は、現在のインスタンスを参照するオブジェクト変数または構造体変数のいずれかと同様に動作します。特に、別のクラス、構造体、またはモジュールのプロシージャに、クラスや構造体の現在のインスタンスに関する情報を渡す場合などには、**Me** が役立ちます。たとえば、モジュール内に次のプロシージャがあったとします。

```
Sub ChangeFormColor(FormName As Form)
   Randomize()
   FormName.BackColor = Color.FromArgb(Rnd() * 256, Rnd() * 256, Rnd() * 256)
End Sub
```

この場合に次のステートメントを使うと、上のプロシージャを呼び出し、Form クラスの現在のインスタンスを引数として渡すことができます。

```
ChangeFormColor(Me)
```

キーワード **Me** は、次の構文で使用します。

Class ステートメント

Structure ステートメント

参照
関連項目
MyBase
MyClass
概念
オブジェクト変数の代入

Module (Visual Basic)

ソース ファイルの冒頭にある属性が現在のアセンブリ モジュールに適用されることを示します。

解説

多くの属性は、クラスまたはプロパティなどの個別のプログラミング要素に適用されます。このような属性は、宣言ステートメントに直接、山かっこ (< >) で囲んだ属性ブロックを割り当てることで適用します。

属性が、次の要素だけでなく現在のアセンブリ モジュールに適用される場合、属性ブロックはソース ファイルの冒頭に記述し、**Module** キーワードで属性を明示します。これがアセンブリ全体に適用される場合、Assembly キーワードを使用します。

Module キーワードは、Module ステートメントとは異なるものです。

参照

関連項目

Assembly

Module ステートメント

概念

属性の適用

MustInherit

クラスが基本クラスとしてのみ使用でき、オブジェクトを直接作成できないことを示します。

解説

基本クラス (抽象クラスとも呼ばれます) の目的は、ここから派生したすべてのクラスに共通した機能を定義することです。これにより、派生クラスが共通の要素を再定義する必要がなくなります。有用なオブジェクトを作成するために、この共通の機能では十分でない場合は、派生クラスがそれぞれ足りない機能を定義します。この場合、使用する側のコードでは、派生クラスからのみオブジェクトを作成できるようにすると便利です。これを実現するには、基本クラスで **MustInherit** を使用します。

また、**MustInherit** クラスを使用して、変数を関連するクラスのみに制限できます。基本クラスを定義し、これらの関連するクラスをすべてここから派生させます。基本クラスでは、すべての派生クラスに共通の機能を定義する必要はなく、変数に値を割り当てる際のフィルタとして機能させることができます。使用する側のコードで、変数を基本クラスとして宣言すると、その変数には、いずれかの派生クラスのオブジェクトだけを割り当てることができるようになります。

.NET Framework では、いくつかの **MustInherit** クラスが定義されています。たとえば、Array、Enum、ValueType などです。**ValueType** は、変数を制限する基本クラスの 1 つです。すべての値型は **ValueType** から派生します。変数を **ValueType** として宣言すると、その変数には値型のみを割り当てられます。

ルール

- **宣言コンテキストMustInherit** は、**Class** ステートメントでのみ使用できます。
- **修飾子の結合** 同じ変数宣言で **NotInheritable** と **MustInherit** を同時に指定することはできません。

使用例

次の例では、強制的な継承と強制的なオーバーライドを示します。基本クラス `shape` は、変数 `acrossLine` を定義します。`circle` クラスと `square` クラスは、`shape` から継承されます。これらのクラスは、`acrossLine` の定義を継承しますが、図形の種類によって計算が異なるため、`area` 関数を定義する必要があります。

VB

```
Public MustInherit Class shape
    Public acrossLine As Double
    Public MustOverride Function area() As Double
End Class
Public Class circle : Inherits shape
    Public Overrides Function area() As Double
        Return Math.PI * acrossLine
    End Function
End Class
Public Class square : Inherits shape
    Public Overrides Function area() As Double
        Return acrossLine * acrossLine
    End Function
End Class
Public Class consumeShapes
    Public Sub makeShapes()
        Dim shape1, shape2 As shape
        shape1 = New circle
        shape2 = New square
    End Sub
End Class
```

`shape1` と `shape2` は、`shape` 型として定義できます。ただし、`shape` からオブジェクトを作成することはできません。`area` 関数の機能がないことと、**MustInherit** とマークされているためです。

クラスが `shape` として宣言されているので、変数 `shape1` と `shape2` は派生クラス `circle` および `square` のオブジェクトに限定されます。Visual Basic では、高いレベルのタイプ セーフを実現するために、その他のオブジェクトをこれらの変数に割り当てることはできません。

使用法

MustInherit 修飾子は、次の場合に使用できます。

Class ステートメント

参照

関連項目

Inherits ステートメント

NotInheritable

Visual Basic 言語のキーワード

その他の技術情報

Visual Basic の継承

MustOverride

このクラスで実装されておらず、派生クラスでオーバーライドしないと使用できないプロパティやプロシージャであることを示します。

解説

MustOverride は、プロパティまたはプロシージャの宣言ステートメント内でだけ使用できます。**MustOverride** が指定されているプロパティまたはプロシージャは、クラスのメンバであることが必要です。また、そのクラスには MustInherit が指定されている必要があります。

ルール

- **不完全な宣言** **MustOverride** を指定したプロパティまたはプロシージャには、コードを 1 行も追加しません。**End Function**、**End Property**、または **End Sub** のステートメントも追加しないでください。
- **修飾子の結合** 同じ宣言内で **MustOverride** を **NotOverridable**、**Overridable**、または **Shared** と共に指定することはできません。
- **シャドウとオーバーライド** シャドウとオーバーライドはどちらも継承された要素を再定義しますが、この 2 つの方法には重大な違いがあります。詳細については、「Visual Basic におけるシャドウ」を参照してください。
- **別の呼び名** オーバーライドしないと使用できない要素を、純粋仮想要素とも呼びます。

修飾子 **MustOverride** は、次の構文で使用します。

Function ステートメント (Visual Basic)

Property ステートメント

Sub ステートメント (Visual Basic)

参照
関連項目
NotOverridable
Overridable
Overrides
MustInherit
Visual Basic 言語のキーワード
概念
Visual Basic におけるシャドウ

MyBase

現在のクラス インスタンスの基本クラスを参照する手段を提供します。

解説

MyBase キーワードは、クラスの現在のインスタンスの基本クラスを参照するオブジェクト変数と同様に機能します。**MyBase** は一般に、派生クラスでオーバーライドまたはシャドウされている基本クラスのメンバにアクセスする目的で使用されます。特に、派生クラスのコンストラクタから基本クラスのコンストラクタを明示的に呼び出すために、**MyBase.New** がよく使用されます。

MyBase を使って **MustOverride** 基本メソッドを呼び出すことはできません。

使用例を含む詳細については、「方法 : 派生クラスによって非表示になっている変数にアクセスする」を参照してください。

参照

関連項目

Me

MyClass

New (Visual Basic)

概念

継承の基本

MyClass

クラスの現在のインスタンスのメンバを、派生クラスのオーバーライドによって置き換えられていない状態で参照できるようにします。

解説

MyClass キーワードは、クラスの現在のインスタンスをもともと実装されている状態で参照するオブジェクト変数と同じように動作します。**MyClass** は Me と似ていますが、**MyClass** から呼び出されたメソッドおよびプロパティは、NotOverridable であるかのように扱われます。したがって、メソッドまたはプロパティは、派生クラスでのオーバーライドによる影響を受けません。**Me** と **MyClass** を比較した例を次に示します。

```
Class baseClass
    Public Overridable Sub testMethod()
        MsgBox("Base class string")
    End Sub
    Public Sub useMe()
        ' The following call uses the calling class's version, even if
        ' that version is an override.
        Me.testMethod()
    End Sub
    Public Sub useMyClass()
        ' The following call uses this version and not any override.
        MyClass.testMethod()
    End Sub
End Class
Class derivedClass : Inherits baseClass
    Public Overrides Sub testMethod()
        MsgBox("Derived class string")
    End Sub
End Class
Class testClasses
    Sub startHere()
        Dim testObj As derivedClass = New derivedClass()
        ' The following call displays "Derived class string".
        testObj.useMe()
        ' The following call displays "Base class string".
        testObj.useMyClass()
    End Sub
End Class
```

`derivedClass` は `testMethod` をオーバーライドしていますが、`useMyClass` の **MyClass** キーワードによってオーバーライドの影響は無効になり、`testMethod` の基本クラス バージョンに対する呼び出しが行われます。

Shared (Visual Basic) メソッドの内部では **MyClass** を使用できませんが、インスタンス メソッドの中で **MyClass** を使ってクラスの共有メンバにアクセスすることはできます。

参照

関連項目

Me

MyBase

概念

継承の基本

Narrowing

変換演算子 (**CType**) が、元のクラスまたは構造体の値を格納しきれないデータ型にクラスまたは構造体を変換することを示します。

Narrowing キーワードを使用した変換

変換のプロシージャでは、**Narrowing** に加えて **Public Shared** を指定する必要があります。

縮小変換は、実行時に常に正しく実行されるとは限りません。したがって失敗またはデータを消失する可能性があります。たとえば、**Long** から **Integer**、**String** から **Date**、基本型から派生型への変換などです。基本型から派生型への変換が縮小変換なのは、基本型には派生型のすべてのメンバが含まれていない場合があり、派生型のインスタンスではないためです。

Option Strict が **On** の場合、コードではすべての縮小変換に **CType** を使用する必要があります。

キーワード **Narrowing** は、次の構文で使用します。

Operator ステートメント

参照

処理手順

方法 : 演算子を定義する

関連項目

Operator ステートメント
Widening
CType 関数
Option Strict ステートメント

概念

拡大変換と縮小変換

New (Visual Basic)

新しいオブジェクト インスタンスを作成する **New** 句を開始する、または、型パラメータにコンストラクタ制約を指定します。

解説

型パラメータ リストで、**New** 制約は、アクセス可能なパラメータなしのコンストラクタを渡された型が公開する必要があることを示します。型パラメータと制約の詳細については、型リスト を参照してください。

代入ステートメントの宣言で、**New** 句は、インスタンスを作成する定義済みのクラスを指定する必要があります。つまり、クラスは呼び出し元のコードがアクセスできるコンストラクタを公開する必要があります。

New 句は、宣言ステートメントまたは代入ステートメントの中で使用できます。ステートメントが実行されると、指定したクラスのコンストラクタが呼び出されて、指定した引数が渡されます。次にコード例を示します。

```
Dim someObj As Object
someObj = New someClass("String required by constructor")
Dim nextLabel As New Label()
```

配列はクラスであることから、**New** は次のとおり、新しい配列のインスタンスを作成できます。

```
Dim intArray As Integer()
intArray = New Integer() {0, 1, 2, 3}
```

新しいインスタンスを作成するためのメモリが不足している場合、共通言語ランタイム (CLR) は OutOfMemoryException エラーをスローします。

キーワード **New** は、次の構文で使用します。

Dim ステートメント (Visual Basic)

Of

参照

関連項目

Visual Basic 言語のキーワード
型リスト
コンストラクタとデストラクタの使用方法
OutOfMemoryException

概念

Visual Basic におけるジェネリック型
オブジェクトの有効期間 : オブジェクトの作成と破棄

Next

ループを終了し、異なる値のループ変数で反復処理を行うことを指定します。または、エラーを生成したステートメントの次のステートメントから実行を継続するよう指定します。

解説

キーワード **Next** は、次の構文で使用します。

For...Next ステートメント

For Each...Next ステートメント

On Error ステートメント

Resume ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Nothing (Visual Basic)

任意のデータ型の既定値を表します。

解説

変数に **Nothing** を代入すると、変数の宣言された型に対する既定値が変数に設定されます。型に変数のメンバが含まれている場合は、すべてに既定値が設定されます。次に例を示します。

```
Public Structure testStruct
    Public name As String
    Public number As Short
End Structure
Dim ts As testStruct, i As Integer, b As Boolean
ts = Nothing
' The preceding statement sets ts.name to "" and ts.number to 0.
i = Nothing
b = Nothing
' The preceding statements set i to 0 and b to False.
```

変数が参照型である場合、つまりオブジェクト変数である場合は、**Nothing** を代入すると、どのオブジェクトとも関連付けられていない変数になります。次にコード例を示します。

```
Dim testObject As Object
testObject = Nothing
' The preceding statement sets testObject to not refer to any instance.
```

キーワード Nothing をオブジェクト変数に代入すると、その変数はどのオブジェクト インスタンスも参照しなくなります。それまで変数がインスタンスを参照していた場合は、変数に **Nothing** を設定しても、インスタンス自体が終了することはありません。インスタンスが終了したとき、関連付けられていたメモリとシステム リソースが解放されるのは、アクティブな参照が残っていないことをガベージ コレクタ (GC: Garbage Collector) が検出した後だけです。

参照

関連項目

Dim ステートメント (Visual Basic)

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄

Visual Basic における有効期間

NotInheritable

基本クラスとして使用できないクラスを示すキーワードです。

解説

別の呼び名 継承できないクラスは、シール クラスとも呼ばれます。

NotInheritable 修飾子は次の構文で使用します。

Class ステートメント (Visual Basic)

参照

関連項目

Inherits ステートメント
MustInherit
Visual Basic 言語のキーワード

NotOverridable

派生クラスでオーバーライドできないプロパティまたはプロシージャを示すキーワードです。

解説

基本クラスのプロパティまたはプロシージャをオーバーライドしていないプロパティまたはプロシージャの場合は、**NotOverridable** が既定の設定です。

オーバーライドできない要素は、シール要素と呼ばれることもあります。

ルール

- **宣言コンテキスト NotOverridable** は、プロパティまたはプロシージャの宣言ステートメント内でだけ使用できます。**NotOverridable** を指定できるのは、別のプロパティまたはプロシージャをオーバーライドしているプロパティやプロシージャだけです。つまり、**Overrides** と組み合わせた場合のみ使用できます。
- **修飾子の結合** 同じ宣言内で **NotOverridable** を **MustOverride**、**Overridable**、または **Shared** と共に指定することはできません。

修飾子 **NotOverridable** は、次の構文で使用します。

Function ステートメント (Visual Basic)

Property ステートメント

Sub ステートメント (Visual Basic)

参照
関連項目
MustOverride
Overridable
Overrides
Visual Basic 言語のキーワード
概念
Visual Basic におけるシャドウ

Of

Of 句を使用すると、ジェネリックであるクラス、構造体、インターフェイス、デリゲート、またはプロシージャに型パラメータを定義できます。ジェネリック型の詳細については、「Visual Basic におけるジェネリック型」を参照してください。

Of キーワードの使用方法

次のコード例は、キーワード **Of** を使って、2 つの型パラメータを受け取るクラスのアウトラインを定義します。IComparable インターフェイスによって、`keyType` パラメータに 制約 が指定されています。そのため、このクラスを使用するコードは、**IComparable** を実装する型引数を渡す必要があります。これは、`add` プロシージャが System.IComparable.CompareTo(System.Object) メソッドを呼び出すために必要です。制約の詳細については、「型リスト」を参照してください。

```
Public Class Dictionary(Of entryType, keyType As IComparable)
    Public Sub add(ByVal e As entryType, ByVal k As keyType)
        Dim dk As keyType
        If k.CompareTo(dk) = 0 Then
        End If
    End Sub
    Public Function find(ByVal k As keyType) As entryType
    End Function
End Class
```

上記のクラス定義を作成すると、そこからさまざまな `dictionary` クラスが作成できます。`entryType` と `keyType` に指定する型によって、クラスが保持するエントリの型と、クラスが各エントリに関連付けるキーの型が決まります。制約が定義されているため、`keyType` には **IComparable** を実装する型を指定する必要があります。

次のコード例は、文字列 (**String**) のエントリを保持するオブジェクトを作成し、各エントリに整数 (**Integer**) のキーを関連付けます。**Integer** は **IComparable** を実装しているため、`keyType` に対する制約を満たします。

```
Dim d As New dictionary(Of String, Integer)
```

キーワード **Of** は、次の構文で使用します。

Class ステートメント

Delegate ステートメント

Function ステートメント

Interface ステートメント

Structure ステートメント

Sub ステートメント

参照
関連項目
型リスト
IComparable
概念
Visual Basic におけるジェネリック型

Off

コンパイラ オプションを無効にします。

解説

キーワード **Off** は、次の構文で使用します。

Option Explicit ステートメント

Option Strict ステートメント

参照

関連項目

Visual Basic 言語のキーワード

On

ランタイム エラーに対する応答を示すキーワード、またはコンパイラ オプションをオンにするキーワードです。

解説

キーワード **On** は、次の構文で使用します。

On Error ステートメント

Option Explicit ステートメント

Option Strict ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Option (Visual Basic)

ステートメントが、ソース ファイル全体に適用されるコンパイラ オプションを指定することを示します。

解説

コンパイラ オプションを使用すると、すべての変数を明示的に宣言する必要があるか、型の縮小変換を明示的に定義する必要があるか、文字列はテキストとバイナリ値のどちらで比較するのかを制御できます。

キーワード **Option** は、次の構文で使用します。

Option Compare ステートメント

Option Explicit ステートメント (Visual Basic)

Option Strict ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Optional (Visual Basic)

プロシージャを呼び出すとき省略できる引数であることを示すキーワードです。

解説

修飾子 **Optional** は、次の構文で使用します。

Declare ステートメント

Function ステートメント

Property ステートメント

Sub ステートメント

参照

関連項目

Visual Basic 言語のキーワード

Overloads

プロパティまたはプロシージャが、1 つ以上の既存のプロパティまたはプロシージャを、同じ名前で再度宣言することを指定します。

解説

オーバーロードとは、特定のプロパティ名またはプロシージャ名に対し、同じスコープ内で複数の定義を与えることをいいます。プロパティやプロシージャを異なるシグネチャで宣言し直すことを、シグネチャによる隠ぺいと呼ぶ場合があります。

ルール

- **宣言コンテキスト Overloads** は、プロパティまたはプロシージャの宣言ステートメント内でだけ使用できます。
- **修飾子の結合** 同じプロシージャ宣言で **Overloads** と Shadows の両方を指定することはできません。
- **シグネチャの変更** この宣言内のシグネチャは、オーバーロードされるすべてのプロパティまたはプロシージャのシグネチャと異なっている必要があります。シグネチャは、プロパティ名またはプロシージャ名の他に、次の要素で構成されます。
    - パラメータの数
    - パラメータの順序
    - パラメータのデータ型
    - 型パラメータの数 (ジェネリック プロシージャの場合)
    - 戻り値の型 (変換演算子のプロシージャの場合のみ)

  すべてのオーバーロードは同じ名前を持つ必要があります。ただし、各オーバーロードは、上記の 1 つ以上の要素において、他のすべてと異なっていることが必要です。これは、コードがプロパティまたはプロシージャを呼び出すとき、コンパイラがどのバージョンを使えばよいかを区別するために必要です。
- **許可されない変更** 以下の要素を 1 つでも変更した場合、プロパティまたはプロシージャのオーバーロードは有効ではありません。以下の要素はシグニチャには含まれません。
    - 値を返すかどうか (プロシージャの場合)
    - 戻り値のデータ型 (変換演算子の場合を除く)
    - パラメータ名または型パラメータ名
    - 型パラメータに対する制約 (ジェネリック プロシージャの場合)
    - パラメータを修飾するキーワード (**ByRef** や **Optional** など)
    - プロパティやプロシージャを修飾するキーワード (**Public** や **Shared** など)
- **修飾子の省略** 同じクラスに複数のオーバーロード プロパティまたはオーバーロード プロシージャを定義する場合は、**Overloads** 修飾子を指定する必要はありません。ただし、1 つの宣言で **Overloads** を使用する場合は、すべての宣言で **Overloads** を使用する必要があります。
- **シャドウとオーバーロード Overloads** は基本クラスの既存のメンバまたはオーバーロードされたメンバをシャドウするためにも使用できます。この方法で **Overloads** キーワードを使う場合は、基本クラスのメンバと名前もパラメータ リストも同じプロパティまたはメソッドを宣言し、**Shadows** キーワードは指定しません。

修飾子 **Overloads** は、次の構文で使用します。

Function ステートメント

Operator ステートメント

Property ステートメント

Sub ステートメント

参照
処理手順
方法 : 変換演算子を定義する
関連項目
Shadows

概念

プロシージャのオーバーロード
Visual Basic におけるジェネリック型
演算子プロシージャ

Overridable

プロパティまたはプロシージャを、派生クラスで同じ名前のプロパティまたはプロシージャでオーバーライドできることを指定します。

解説

基本クラスのプロパティまたはプロシージャをオーバーライドする、プロパティまたはプロシージャの場合は、**Overridable** が既定の設定です。

シャドウとオーバーライドはどちらも継承された要素を再定義しますが、この 2 つの方法には重大な違いがあります。詳細については、「Visual Basic におけるシャドウ」を参照してください。

オーバーライド可能な要素は、仮想要素と呼ばれることもあります。オーバーライド可能だが、オーバーライドする必要はないという場合は、具象要素とも呼ばれます。

ルール

- **宣言コンテキスト Overridable** は、プロパティまたはプロシージャの宣言ステートメント内でだけ使用できます。
- **修飾子の結合** 同じ宣言内で **Overridable** を **MustOverride**、**NotOverridable**、または **Shared** と共に指定することはできません。オーバーライドする要素は暗黙的にオーバーライド可能であるため、**Overridable** と **Overrides** を同時に指定することはできません。

修飾子 **Overridable** は、次の構文で使用します。

Function ステートメント (Visual Basic)

Property ステートメント

Sub ステートメント (Visual Basic)

参照

関連項目

MustOverride
NotOverridable
Overrides
Visual Basic 言語のキーワード

概念

Visual Basic におけるシャドウ

Overrides

プロパティまたはプロシージャが、基本クラスから継承された、同じ名前のプロパティまたはプロシージャをオーバーライドすることを指定します。

解説
規則

- **宣言コンテキストOverrides** は、プロパティまたはプロシージャの宣言ステートメント内でだけ使用できます。
- **修飾子の結合** 同じ変数宣言で **Overrides** を、**Shadows** または **Shared** と 同時に指定することはできません。オーバーライドする要素は暗黙でオーバーライド可能なので、**Overridable** を **Overrides** と組み合わせることはできません。
- **シグネチャの一致** この宣言のシグネチャは、オーバーライドされるプロパティまたはプロシージャのシグネチャと完全に一致する必要があります。つまり、パラメータ リストには、同じデータ型の、同じ数のパラメータを同じ順序で指定する必要があります。

  シグネチャ以外では、オーバーライドする宣言で次の要素が完全に一致している必要があります。

  - アクセス レベル
  - 戻り値 (あれば) の型
- **ジェネリック シグネチャ** ジェネリック プロシージャでは、シグネチャに型パラメータの数が含まれます。したがって、その点においても、オーバーライドする宣言は基本クラスのバージョンに一致する必要があります。
- **一致が必要なその他の要素** 基本クラスのバージョンのシグネチャ以外に、この宣言では次の点でも一致している必要があります。
  - アクセス レベル修飾子 (Public (Visual Basic) など)
  - 各パラメータの渡し方 (ByVal または ByRef)
  - ジェネリック プロシージャの型パラメータごとの入力規則のリスト
- **シャドウとオーバーライド** シャドウとオーバーライドは、どちらも継承された要素を再定義しますが、そのアプローチは大きく異なります。詳細については、「Visual Basic におけるシャドウ」を参照してください。

Overrides 修飾子は、次の場合に使用できます。

Function ステートメント

Property ステートメント

Sub ステートメント

参照
関連項目
MustOverride
NotOverridable
Overridable
Visual Basic 言語のキーワード
型リスト
概念
Visual Basic におけるシャドウ
Visual Basic におけるジェネリック型

キーワード P ～ Z

次の表は、Visual Basic 言語のキーワードの一覧です。

ParamArray	Partial	Preserve	Private
Protected	Public	RaiseEvent	ReadOnly
Resume	Shadows	Shared	Static
Step	Structure	Text	Then
To	True	TryCast	Unicode
Until	When	While	Widening
WithEvents	WriteOnly		

スマート デバイス開発者のためのメモ

Unicode は、スマート デバイス アプリケーションではサポートされません。

参照

関連項目

キーワード A ～ E

キーワード F ～ O

その他の技術情報

Visual Basic リファレンス

ParamArray

プロシージャのパラメータが、指定されたデータ型の、省略可能な要素の配列を受け取ることを示します。**ParamArray** は、パラメータ リストの最後のパラメータにのみ使用できます。

解説

ParamArray を使うと、任意の数の引数をプロシージャに渡すことができます。**ParamArray** パラメータは常に ByVal を使って宣言されます。

ParamArray パラメータには、適切なデータ型の配列を渡すか、コンマ区切りの値のリストを渡すか、または何も渡さないという方法によって、1 つ以上の引数を指定できます。詳細については、「パラメータ配列」の「ParamArray を呼び出す」を参照してください。

セキュリティに関するメモ：
無限に増大する配列を扱う場合、アプリケーション内部の容量を超過してしまう可能性があります。呼び出し元のコードからパラメータの配列を受け取る場合、その長さをテストし、アプリケーションに対して大きすぎるようであれば、適切な手順を行ってください。

ParamArray 修飾子は、次の場合に使用できます。

Declare ステートメント

Function ステートメント (Visual Basic)

Property ステートメント

Sub ステートメント (Visual Basic)

参照

処理手順

方法 : 不特定数のパラメータを受け取るプロシージャを定義する

方法 : 不特定数のパラメータを受け取るプロシージャを呼び出す

関連項目

Visual Basic 言語のキーワード

概念

パラメータ配列

Partial (Visual Basic)

クラスまたは構造体の宣言が、クラスまたは構造体の部分定義であることを示します。

```
[ <attrlist> ] [ accessmodifier ] [ Shadows ] [ MustInherit | NotInheritable ] _
Partial { Class | Structure } name [ (Of typelist) ]
    [ Inherits classname ]
    [ Implements interfacenames ]
    [ variabledeclarations ]
    [ proceduredeclarations ]
{ End Class | End Structure }
```

指定項目

attrlist

省略可能です。このクラスまたは構造体に適用される属性の一覧を指定します。属性リスト は、山かっこ (< >) で囲む必要があります。

accessmodifier

省略可能です。どのようなコードからこのクラスまたは構造体にアクセスできるのかを指定します。Visual Basic でのアクセス レベル を参照してください。

Shadows

省略可能です。Shadows を参照してください。

MustInherit

省略可能です。MustInherit を参照してください。

NotInheritable

省略可能です。NotInheritable を参照してください。

name

必ず指定します。このクラスまたは構造体の名前です。同じクラスまたは構造体の中の、他のすべての部分宣言で定義されている名前と一致する必要があります。

Of

省略可能です。これがジェネリック クラスまたはジェネリックな構造体であることを指定します。Visual Basic におけるジェネリック型 を参照してください。

typelist

Of を使用する場合は必ず指定します。型リスト を参照してください。

Inherits

省略可能です。Inherits ステートメント を参照してください。

classname

Inherits を使用する場合は必ず指定します。このクラスの派生元のクラスまたはインターフェイスの名前です。

Implements

省略可能です。Implements ステートメント を参照してください。

interfacenames

Implements を使用する場合は必ず指定します。このクラスまたは構造体が実装するインターフェイスの名前を指定します。

variabledeclarations

省略可能です。クラスまたは構造体の、追加の変数やイベントを宣言するステートメントです。

proceduredeclarations

省略可能です。クラスまたは構造体の、追加のプロシージャを宣言および定義するステートメントです。

End Class または、次のようにも指定できます。**End Structure**

この **Class** または **Structure** の部分定義を終了します。

解説

Partial キーワードを使用すると、クラスまたは構造体の定義は、複数の宣言に分割できます。部分宣言は、必要に応じていくつでも、いくつのソース ファイルでも使用できます。しかし、すべての宣言は同じアセンブリと同じ名前空間内にある必要があります。

メモ：
Visual Basic は部分クラスの定義を使用して、生成されたコードとユーザーが作成したコードを、別々のソース ファイルに分割します。たとえば、Windows フォーム デザイナは、Form などのコントロールに部分クラスを定義します。これらのコントロールでは、生成されたコードを変更しないでください。

通常の状況では、単独のクラスまたは構造体の開発を、複数の宣言に分割しないでください。したがって、多くの場合は **Partial** キーワードは必要ありません。

修飾子の利用法と継承など、クラスおよび構造体の作成に関するすべての規則は、部分クラスや部分構造体を作成するときに適用されます。

最適な使用方法

- **キーワードの使用法** 読みやすさのために、クラスまたは構造体の部分宣言にはすべて **Partial** キーワードを含めます。コンパイラでは、1 つの部分宣言でこのキーワードを省略してもエラーになりませんが、複数省略するとエラーになります。

動作

- **宣言の共用体** コンパイラは、クラスまたは構造体を、部分宣言の共用体として扱います。すべての部分定義で使用されているすべての修飾子は、クラスまたは構造体全体に適用され、すべての部分定義のすべてのメンバは、クラスまたは構造体全体で利用可能です。
- **モジュール内の部分型は上位変換不可能** 部分定義がモジュール内にある場合、そのクラスまたは構造体の型の上位変換は自動的に失敗します。この場合、一連の部分定義によって、予期しない結果になったり、場合によってはコンパイラ エラーが発生することがあります。詳細については、「型の上位変換」を参照してください。

  コンパイラは、完全修飾されたパスがまったく同じ場合にのみ、部分定義をマージします。

キーワード **Partial** は、次の構文で使用します。

Class ステートメント

Structure ステートメント

使用例

次の例では、クラス `sampleClass` の定義を 2 つの宣言に分割し、それぞれの部分で別の **Sub** プロシージャを定義します。

VB

```
Partial Public Class sampleClass
    Public Sub sub1()
    End Sub
End Class
Partial Public Class sampleClass
    Public Sub sub2()
    End Sub
End Class
```

この例にある 2 つの部分定義は、同じソース ファイル内にあっても、別々のソース ファイル内にあってもかまいません。

参照

関連項目

Class ステートメント (Visual Basic)
Structure ステートメント
Shadows

概念

型の上位変換
Visual Basic におけるジェネリック型

Preserve

配列の次元が再設定されるときに配列の内容が消去されるのを防ぎます。

解説

キーワード **Preserve** は、次の構文で使用します。

ReDim ステートメント (Visual Basic)

参照

関連項目

Visual Basic 言語のキーワード

Private (Visual Basic)

1 つ以上のプログラミング要素が宣言のコンテキスト内 (内包型など) からのみアクセスできることを示します。

解説

プログラミング要素が独自の機能を持つ場合や機密データを含む場合、その要素へのアクセスは可能な限り厳しく制限する必要があります。その要素を定義したモジュール、クラス、または構造体のみがアクセスできるようにすることで、最大の制限が可能になります。このような形で要素へのアクセスを制限するには、要素を **Private** で宣言します。

規則

- **宣言コンテキスト** **Private** は、モジュール レベルでのみ使用できます。つまり、**Private** 要素の宣言コンテキストは、ソース ファイル、名前空間、インターフェイス、プロシージャではなく、モジュール、クラス、または構造体である必要があります。

動作

- **アクセス レベル** 宣言コンテキスト内のすべてのコードは、自らの **Private** 要素にアクセスできます。これには、ネストされたクラスや、列挙体内の代入式など、内包型のコードも含まれます。宣言コンテキスト外部のコードは、**Private** 要素にはアクセスできません。
- **アクセス修飾子** アクセス レベルを指定するキーワードは、アクセス修飾子と呼ばれます。アクセス修飾子の比較については、「Visual Basic でのアクセス レベル」を参照してください。

Private 修飾子は、次の場合に使用できます。

Class ステートメント

Const ステートメント

Declare ステートメント

Delegate ステートメント

Dim ステートメント

Enum ステートメント

Event ステートメント

Function ステートメント

Interface ステートメント

Property ステートメント

Structure ステートメント

Sub ステートメント

参照

関連項目

Public (Visual Basic)
Protected (Visual Basic)
Friend (Visual Basic)

概念

Visual Basic でのアクセス レベル
Visual Basic におけるプロシージャ

その他の技術情報

構造体 : 独自のデータ型
クラスについて

Protected (Visual Basic)

宣言された 1 つ以上のプログラミング要素が、要素と同じクラスまたは派生クラスからのみアクセス可能であることを指定します。

解説

クラスに宣言されたプログラミング要素に重要情報や制限付きのコードが含まれるために、その要素へのアクセスを制限する場合があります。しかし、それが継承可能なクラスで、派生クラスが階層的に構成されると予想される場合、これらの派生クラスから機密データや制限付きコードにアクセスすることも必要になります。このような場合は、要素を基本クラスとすべての派生クラスからアクセス可能にしてください。**Protected** を使って要素を宣言すると、この方法で要素へのアクセスを制限できます。

ルール

- **宣言コンテキスト Protected** は、クラス レベルでのみ使用できます。つまり、**Protected** 要素の宣言コンテキストはクラスであることが必要で、ソース ファイル、名前空間、インターフェイス、モジュール、構造体、またはプロシージャでは宣言できません。
- **修飾子の結合 Protected** 修飾子は、同じ宣言内で Friend (Visual Basic) 修飾子と共に使用できます。この組み合わせで宣言すると、要素は同じアセンブリ内の任意の場所、要素と同じクラス、そして派生クラスからアクセス可能になります。**Protected Friend** はクラスのメンバに対してのみ指定できます。

動作

- **アクセス レベル** クラス内のすべてのコードがその要素にアクセスできます。基本クラスから派生した任意のクラスのコードが、基本クラスのすべての **Protected** 要素にアクセスできます。これは、派生のすべての世代で同じです、つまり、クラスは基本クラスのそのまた基本クラス、などのように、何世代も上の **Protected** 要素にアクセスできます。

  プロテクト アクセスはフレンド アクセスのスーパーセットでもサブセットでもありません。
- **アクセス修飾子** アクセス レベルを指定するキーワードは、アクセス修飾子と呼ばれます。アクセス修飾子の比較については、「Visual Basic でのアクセス レベル」を参照してください。

修飾子 **Protected** は、次の構文で使用します。

Class ステートメント

Const ステートメント

Declare ステートメント

Delegate ステートメント

Dim ステートメント

Enum ステートメント

Event ステートメント

Function ステートメント

Interface ステートメント

Property ステートメント

Structure ステートメント

Sub ステートメント

参照

関連項目

Public (Visual Basic)
Friend (Visual Basic)
Private (Visual Basic)

概念

Visual Basic でのアクセス レベル
Visual Basic におけるプロシージャ

その他の技術情報

構造体 : 独自のデータ型
クラスについて

Public (Visual Basic)

宣言された 1 つ以上のプログラミング要素に、アクセス制限がないことを指定します。

解説

クラス ライブラリなど、1 つ以上のコンポーネントを発行する場合は、そのプログラミング要素へのアクセスを、自分のアセンブリとやり取りするすべてのコードに対して許可するのが普通です。このような無制限のアクセス許可を与えるには、**Public** を使って要素を宣言します。

アクセスを制限する必要がないプログラミング要素に対しては、パブリック アクセスが通常のアクセス レベルになります。特に宣言しなければ、インターフェイス、モジュール、クラス、または構造体の内部で宣言された要素のアクセス レベルは、既定で **Public** に設定されるので注意が必要です。

ルール

- **宣言コンテキスト** **Public** は、モジュール レベル、インターフェイス レベル、または名前空間レベルでのみ使用できます。つまり、**Public** 要素の宣言コンテキストは、ソース ファイル、名前空間、インターフェイス、モジュール、クラス、または構造体のいずれかである必要があり、プロシージャでは宣言できません。

動作

- **アクセス レベル** モジュール、クラス、構造体にアクセスできるコードであれば、その **Public** 要素にアクセスできます。
- **既定のアクセス** プロシージャ内のローカル変数のアクセス レベルは、既定で public になります。このような変数にはアクセス修飾子を指定できません。
- **アクセス修飾子** アクセス レベルを指定するキーワードは、アクセス修飾子と呼ばれます。アクセス修飾子の比較については、「Visual Basic でのアクセス レベル」を参照してください。

修飾子 **Public** は、次の構文で使用します。

Class ステートメント

Const ステートメント

Declare ステートメント

Delegate ステートメント

Dim ステートメント

Enum ステートメント

Event ステートメント

Function ステートメント

Interface ステートメント

Module ステートメント

Operator ステートメント

Property ステートメント

Structure ステートメント

Sub ステートメント

参照

関連項目

Protected (Visual Basic)
Friend (Visual Basic)
Private (Visual Basic)

概念

Visual Basic でのアクセス レベル
Visual Basic におけるプロシージャ

その他の技術情報

構造体 : 独自のデータ型
クラスについて

RaiseEvent

実行時にイベントを発生させます。または、イベントの発生時にどのコードを実行するかを宣言します。

解説

キーワード **RaiseEvent** は、次の構文で使用します。

Event ステートメント

RaiseEvent キーワードを使って、カスタムな **RaiseEvent** アクセサを宣言します。

RaiseEvent ステートメント

RaiseEvent キーワードを使ってイベントを発生させます。

参照

処理手順

方法 : イベントを発生させる (Visual Basic)
方法 : ブロックを回避するイベントを宣言する
方法 : メモリの使用量を節約するイベントを宣言する

ReadOnly (Visual Basic)

読み取ることはできても書き込むことはできない変数またはプロパティを示します。

解説

ルール

- **宣言コンテキスト** **ReadOnly** は、モジュール レベルでのみ使用できます。つまり、**ReadOnly** 要素の宣言コンテキストは、ソース ファイル、名前空間、インターフェイス、プロシージャではなく、クラス、構造体、またはモジュールである必要があります。
- **結合された修飾子** 同じ変数宣言で **Static** と **ReadOnly** を同時に指定することはできません。
- **値の代入** **ReadOnly** プロパティを利用するコードは値を設定できません。しかし、基になるストレージにアクセスできるコードは、いつでもその値を代入および変更できます。

  ReadOnly 変数に値を代入できるのは、変数の定義時、または、この変数が定義されたクラスまたは構造体のコンストラクタの中のみです。

ReadOnly 変数を使用する場合

Const ステートメント (Visual Basic) を使用して、定数の値を宣言および代入できない場合があります。たとえば、**Const** ステートメントが代入するデータ型を受け入れない場合や、コンパイル時に定数式で値を計算できない場合などです。コンパイル時には値がわからないことも考えられます。このような場合、**ReadOnly** 変数を使用して定数値を格納します。

セキュリティに関するメモ：
この変数のデータ型が配列やクラス インスタンスなどの参照型の場合、変数自体が **ReadOnly** であっても、このメンバは変更できます。次に例を示します。

```
ReadOnly characterArray() As Char = {"x"c, "y"c, "z"c}

Sub changeArrayElement()

characterArray(1) = "M"c

End Sub
```

初期化されたとき、`characterArray()` で参照される配列には "x"、"y"、"z" が格納されています。変数 `characterArray` は **ReadOnly** なので、一度初期化するとこの値は変更できません。つまり、新しい配列をこれに代入することはできません。しかし、配列のメンバ (1 つまたは複数) の値は変更できます。プロシージャ `changeArrayElement` の呼び出し後、`characterArray()` が参照する配列には "x"、"M"、"z" が格納されています。

これは、プロシージャのパラメータを ByVal として宣言するのと同様です。プロシージャは呼び出し元の引数そのものは変更できませんが、そのメンバは変更できます。

使用例

次の例では、従業員が雇用された日付を表す **ReadOnly** プロパティを定義します。このクラスは、プロパティの値を **Private** 変数として内部に格納し、クラス内のコードだけがこの値を変更できます。ただし、このプロパティは **Public** であり、このクラスにアクセスできるすべてのコードがこのプロパティを読み取れます。

VB

```
Class employee
    ' Only code inside class employee can change the value of hireDateValue.
    Private hireDateValue As Date
    ' Any code that can access class employee can read property dateHired.
    Public ReadOnly Property dateHired() As Date
        Get
            Return hireDateValue
        End Get
    End Property
End Class
```

ReadOnly 修飾子は、次の場合に使用できます。

Dim ステートメント

Property ステートメント

参照

関連項目

WriteOnly

Visual Basic 言語のキーワード

RemoveHandler

イベントとイベント ハンドラの関連付けを解除するか、イベント ハンドラを追加するときに実行するコードを宣言します。

解説

キーワード **RemoveHandler** は、次の構文で使用します。

Event ステートメント

RemoveHandler キーワードを使って、カスタムな **RemoveHandler** アクセサを宣言します。

RemoveHandler ステートメント

RemoveHandler キーワードを使って、イベントとイベント ハンドラの関連付けを解除します。

参照

処理手順

方法 : ブロックを回避するイベントを宣言する
方法 : メモリの使用量を節約するイベントを宣言する

Resume

エラー処理の後で実行を再開する場所を示します。

解説

キーワード **Resume** は、次の構文で使用します。

On Error ステートメント (Visual Basic)

Resume ステートメント

参照

関連項目

Visual Basic 言語のキーワード

その他の技術情報

Visual Basic での例外およびエラー処理

Shadows

宣言されたプログラミング要素が、基本クラスにある、同じ名前を持つ要素またはオーバーロードされる要素を宣言し直すことを示します。

解説

シャドウ (名前による隠ぺいとも呼ばれます) の主な目的は、クラス メンバの定義を保持することにあります。基本クラスは、既に定義されている要素と同じ名前の要素を作成するように変更される可能性もあります。このような場合、**Shadows** 修飾子を指定してあると、派生クラスを通じた参照は新しい基本クラスの要素に解決されず、派生クラスで定義したメンバに解決されます。

シャドウとオーバーライドはどちらも継承された要素を再定義しますが、この 2 つの方法には重大な違いがあります。詳細については、「Visual Basic におけるシャドウ」を参照してください。

ルール

- **宣言コンテキスト Shadows** は、クラス レベルでのみ使用できます。つまり、**Shadows** 要素の宣言コンテキストはクラスであることが必要で、ソース ファイル、名前空間、インターフェイス、モジュール、構造体、またはプロシージャでは宣言できません。

  1 つの宣言ステートメントに宣言できるシャドウ要素は 1 つだけです。

- **修飾子の結合** 同じ宣言内で **Shadows** を **Overloads**、**Overrides**、または **Static** と共に指定することはできません。

- **要素の型が合っていること。** 宣言された要素は、他の任意の種類の要素でシャドウできます。プロパティまたはプロシージャを別のプロパティまたはプロシージャでシャドウする場合、パラメータおよび戻り値の型は、基本クラスのプロパティまたはプロシージャのパラメータおよび戻り値の型と一致しなくてもかまいません。

- **アクセス** 通常、シャドウされた基本クラスの要素は、その要素をシャドウする派生クラスからは使用できません。ただし、次の点に注意してください。

  - シャドウする要素が、その要素を参照するコードからアクセス不可能である場合、参照はシャドウされる要素に解決されます。たとえば、**Private** 要素が基本クラスの要素をシャドウした場合、その **Private** 要素へのアクセス許可を持たないコードは、基本クラスの要素へ代わりにアクセスします。

  - 要素をシャドウした場合でも、基本クラスの型で宣言されたオブジェクトを使用すると、シャドウされた要素にアクセスできます。**MyBase** を使用してアクセスすることもできます。

修飾子 **Shadows** は、次の構文で使用します。

Class ステートメント

Const ステートメント

Declare ステートメント

Delegate ステートメント

Dim ステートメント

Enum ステートメント

Event ステートメント

Function ステートメント

Interface ステートメント

Property ステートメント

Structure ステートメント

Sub ステートメント

参照

関連項目

Shared (Visual Basic)
Static (Visual Basic)
Private (Visual Basic)
MyBase
MustOverride
NotOverridable

Overloads
Overridable
Overrides

概念

Visual Basic におけるシャドウ

Shared (Visual Basic)

宣言された 1 つ以上のプログラミング要素が、クラス全体または構造体全体に関連付けられ、クラスまたは構造体の特定のインスタンスに関連付けられないことを指定します。

解説

Shared を使用する状況

非共有であるクラスまたは構造体のメンバを共有すると、各インスタンスがメンバのコピーを別々に保持するのではなく、すべてのインスタンスがそのメンバを使用できます。このことは、たとえば、変数の値をアプリケーション全体で参照する場合に便利です。そのような変数を **Shared** で宣言した場合、すべてのインスタンスがストレージ内の同じ場所にアクセスするため、あるインスタンスが変数の値を変更すると、すべてのインスタンスが変更後の値にアクセスするようになります。

共有メンバとして宣言したからといって、アクセス レベルが変更されるわけではありません。たとえば、クラスのメンバを、共有プライベート メンバとして宣言することも (クラス内でのアクセスのみ可能)、非共有のパブリック メンバとして宣言することも可能です。詳細については、「Visual Basic でのアクセス レベル」を参照してください。

ルール

- **宣言コンテキスト Shared** は、モジュール レベルでのみ使用できます。つまり、**Shared** 要素の宣言コンテキストは、ソース ファイル、名前空間、プロシージャではなく、クラスまたは構造体である必要があります。
- **修飾子の結合** 同じ宣言内で **Shared** を Overrides、Overridable、NotOverridable、MustOverride または Static (Visual Basic) と共に指定することはできません。
- **アクセス** 共有要素にアクセスするには、そのクラスや構造体の特定のインスタンスの変数名ではなく、クラスや構造体の名前を使用して共有要素を修飾します。共有メンバにアクセスするために、そのクラスや構造体のインスタンスを作成する必要もありません。

  次の例は、Double 構造体で公開されている共有プロシージャ IsNaN を呼び出します。

  ```
  If Double.IsNaN(result) Then MsgBox("Result is mathematically undefined.")
  ```

- **暗黙の共有** Const ステートメント (Visual Basic) で **Shared** 修飾子を使うことはできませんが、定数は暗黙的に共有されます。同様に、モジュールやインターフェイスのメンバは **Shared** では宣言できませんが、暗黙的に共有されます。

動作

- **ストレージ** 共有変数や共有イベントは、そのクラスや構造体のインスタンスが何回作成されたとしても、メモリに一度だけ格納されます。同様に、共有プロシージャや共有プロパティは、ローカル変数を 1 セットだけ保持します。
- **インスタンス変数を使用したアクセス** クラスや構造体の特定のインスタンスを格納する変数の名前で修飾して、共有要素にアクセスすることが可能です。この方法で予期しない動作が起きることは通常ありませんが、コンパイラは警告メッセージを表示し、変数名ではなくクラス名や構造体名を使ってアクセスします。
- **インスタンス式を使用したアクセス** クラスや構造体のインスタンスを返す式を使用して共有要素にアクセスした場合、コンパイラは式を評価せず、クラス名や構造体名を使ってアクセスします。この式を使ってインスタンスを返す他に何か別の処理も実行しようとしていた場合は、予期しない結果になります。次に例を示します。

```
Sub main()
    shareTotal.total = 10
    ' The preceding line is the preferred way to access total.
    Dim instanceVar As New shareTotal
    instanceVar.total += 100
    ' The preceding line generates a compiler warning message and
    ' accesses total through class shareTotal instead of through
    ' the variable instanceVar. This works as expected and adds
    ' 100 to total.
    returnClass().total += 1000
    ' The preceding line generates a compiler warning message and
    ' accesses total through class shareTotal instead of calling
    ' returnClass(). This adds 1000 to total but does not work as
    ' expected, because the MsgBox in returnClass() does not run.
    MsgBox("Value of total is " & CStr(shareTotal.total))
End Sub
```

```
Public Function returnClass() As shareTotal
    MsgBox("Function returnClass() called")
    Return New shareTotal
End Function
Public Class shareTotal
    Public Shared total As Integer
End Class
```

このコード例では、インスタンスを使って共有変数 `total` に 2 度アクセスしていますが、コンパイラは 2 度とも警告メッセージを生成します。どちらの場合も、コンパイラは直接 `shareTotal` クラスを使ってアクセスし、インスタンスを使用しません。プロシージャ `returnClass` を呼び出そうとしていた部分では、`returnClass` の呼び出しが行われないため、ここに追加されている "Function returnClass() called" というメッセージの表示は実行されません。

修飾子 **Shared** は、次の構文で使用します。

Dim ステートメント

Event ステートメント

Function ステートメント

Operator ステートメント

Property ステートメント

Sub ステートメント

参照

関連項目

Shadows
Static (Visual Basic)

概念

Visual Basic における有効期間
Visual Basic におけるプロシージャ
Visual Basic の共有メンバ

その他の技術情報

構造体 : 独自のデータ型
クラスについて

Static (Visual Basic)

1 つ以上のローカル変数が、それらが宣言されたプロシージャの終了後も存在し続け、最後に設定された値を保持することを指定します。

解説

通常、プロシージャ内のローカル変数は、プロシージャが終了した直後に削除されます。静的変数はプロシージャの終了後も存在し続け、最後に設定された値を保持します。次回、コードからそのプロシージャを呼び出したとき、変数は初期化し直されることなく、最後に割り当てられた値をそのまま保持します。静的変数は、自らが定義されたクラスやモジュールの有効期間中存在し続けます。

規則

- **宣言コンテキスト Static** はローカル変数にのみ使用できます。つまり、**Static** 変数は、プロシージャまたはプロシージャ内のブロックのコンテキストで宣言される必要があり、ソース ファイル、名前空間、クラス、構造体、またはモジュールのコンテキストでは宣言できません。

  Static は、構造体のプロシージャの内部では使用できません。

- **修飾子の結合** 同じ宣言内で **Static** を **ReadOnly**、**Shadows**、または **Shared** と共に指定することはできません。

動作

ローカル変数の動作は、それが **Shared** プロシージャに宣言されているかどうかによって変わります。プロシージャが **Shared** であれば、その **Static** 変数を含むすべてのローカル変数が自動的に共有されます。このような変数は、アプリケーション全体で 1 つしか作成されません。**Shared** プロシージャはクラス名を使って呼び出してください。クラスのインスタンスをポイントする変数を使って呼び出すことはできません。

プロシージャが **Shared** でなければ、その **Static** 変数を含むローカル変数はインスタンス変数になります。クラスの各インスタンスに独自の変数が作成されます。共有でないプロシージャは、クラスの特定のインスタンスをポイントする変数を使って呼び出してください。そのインスタンス内の変数は、別のインスタンス内の同じ名前を持つ変数とは独立したものです。したがって、これらの変数には異なる値を格納できます。

使用例

次の例は **Static** の使い方を示しています。

VB

```
Function updateSales(ByVal thisSale As Decimal) As Decimal
    Static totalSales As Decimal = 0
    totalSales += thisSale
    Return totalSales
End Function
```

Static 変数 `totalSales` は、一度だけ 0 に初期化されます。`updateSales` を何度入力しても、`totalSales` は最後に計算された値をそのまま保持します。

Static 修飾子は次の構文で使用します。

Dim ステートメント (Visual Basic)

参照

関連項目

Shadows

Shared (Visual Basic)

概念

Visual Basic における有効期間

Visual Basic での変数宣言

その他の技術情報

構造体 : 独自のデータ型

クラスについて

Step

ループ カウンタの増分値を示します。

解説

キーワード **Step** は、次の構文で使用します。

For...Next ステートメント (Visual Basic)

参照

関連項目

Visual Basic 言語のキーワード

Structure (Visual Basic)

ジェネリック型のパラメータに制約を指定し、これに渡す型引数を必ず値型にするか、そうでなければ Structure ステートメントを使用するように要求します。

解説

ジェネリック型に型パラメータを宣言するとき、制約、つまり 1 つ以上の要件を加えて、その型のパラメータに渡す型引数を制限できます。詳細については、「Visual Basic におけるジェネリック型」の「制約」を参照してください。

たとえば、型引数が値型 (たとえば構造体、列挙体、基本データ型など) であること、という要件を制約を使って定義できます。これを指定するには、制約に **Structure** キーワードを使用します。

これ以外の要件では、型引数が参照型 (たとえば文字列 (**String**) 型、配列、デリゲート、クラスから作成されたオブジェクトなど) であることも指定できます。参照型の要件を指定するには、制約に Class (Visual Basic) キーワードを使用します。

Class または **Structure** を、制約に必ず指定する必要はありません。同じ制約に、この 2 つを同時に指定することはできません。

Structure 制約は、Structure ステートメントとは異なります。

参照

関連項目

Class (Visual Basic)
Structure ステートメント

概念

Visual Basic におけるジェネリック型
値型と参照型

Text

文字列の比較メソッドを、等価なテキスト文字を等しいと見なしてソートするように設定します。

解説

Option Compare ステートメントを使用する場合は、ソース ファイル内で他のソース ステートメントより前に記述する必要があります。これにより、そのソース ファイルから生成されるすべてのコードの文字列比較に対して設定が有効になります。

Text を使用する状況

テキスト文字だけが格納された文字列がある場合、アルファベット上等価な文字 (大文字と小文字を区別しないなど)、および関係の深い文字を考慮して文字列を比較することがよくあります。たとえば、`A` と `a` が等しく、`Ä` と `ä` が `B` と `b` の前にくると見なすような場合です。テキストの比較を指定するには、**Option Compare** に **Text** を設定します。

キーワード **Text** は、次の構文で使用します。

Option Compare ステートメント

参照
関連項目
Binary
Visual Basic 言語のキーワード

Then

テストした条件が true の場合にコンパイルまたは実行するステートメント ブロックを示すキーワードです。

解説

キーワード **Then** は、次の構文で使用します。

#If...Then...#Else ディレクティブ

If...Then...Else ステートメント

参照

関連項目

Visual Basic 言語のキーワード

To

ループ カウンタまたは値が一致する範囲の開始と終了の値を分けるキーワードです。

解説

キーワード **To** は、次の構文で使用します。

For...Next ステートメント

Select...Case ステートメント

参照

関連項目

Visual Basic 言語のキーワード

True (Visual Basic)

条件テストを渡す **Boolean** 値を表します。

解説

ブール型 (Boolean) (Visual Basic) 値は数字としては格納されず、格納された値は数字と等しくなりません。コードを作成する際、**True** および **False** に相当する数値を利用しないようにしてください。ブール型 (**Boolean**) の変数に対しては、できるだけ本来の論理値だけを使用してください。ブール型 (**Boolean**) と数値を混在させる必要がある場合は、選択した変換メソッドを十分理解してから行ってください。詳細については、「データ型のトラブルシューティング」の「Boolean 型が数値型に正確に変換されない」を参照してください。

参照

処理手順

データ型のトラブルシューティング

関連項目

ブール型 (Boolean) (Visual Basic)

False (Visual Basic)

TryCast

例外をスローしない型変換演算を開始します。

解説

変換が失敗した場合、**CType** および **DirectCast** の両方から InvalidCastException エラーが発生します。これは、アプリケーションのパフォーマンスに悪影響を与える可能性があります。**TryCast** は、例外を処理するのではなく、返された値が **Nothing** かどうかをテストするだけで済むように Nothing (Visual Basic) を返します。

TryCast キーワードは、CType 関数 や DirectCast キーワードと同じように使用します。最初の引数は式、2 つ目の引数は変換する型です。**TryCast** は、クラスやインターフェイスなどの参照型のみを扱います。2 つの型の間には、継承または実装の関係が必要です。つまり、一方の型が他方を継承しているか、実装している必要があります。

エラー

継承も実装もないことを確認した場合、**TryCast** はコンパイラ エラーを生成します。コンパイラ エラーが発生しなくても、変換が成功したとは限りません。実行する変換が縮小変換である場合、実行時に失敗する可能性があります。この場合、**TryCast** は Nothing (Visual Basic) を返します。

型変換のキーワード

型変換のキーワードを比較したものを次に示します。

キーワード	データ型	引数の関係	実行時のエラー
CType 関数	Any 型	2 つのデータ型の間で拡大変換または縮小変換を定義する必要があります。	**InvalidCastException** をスロー
DirectCast	Any 型	一方の型が他方を継承または実装していることが必要	**InvalidCastException** をスロー
TryCast	参照型のみ	一方の型が他方を継承または実装していることが必要	Nothing (Visual Basic) を返す

使用例

次の例は、**TryCast** を使用する方法を示しています。

VB

```
  Function PrintTypeCode(ByVal obj As Object) As String
      Dim objAsConvertible As IConvertible = TryCast(obj, IConvertible)
      If objAsConvertible Is Nothing Then
          Return obj.ToString() & " does not implement IConvertible"
      Else
          Return "Type code is " & objAsConvertible.GetTypeCode()
      End If
  End Function
```

参照

概念

拡大変換と縮小変換

暗黙の型変換と明示的な型変換

Unicode (Visual Basic)

宣言される外部プロシージャの名前に関係なく、すべての文字列を Unicode 値にマーシャリングするように指定します。

プロジェクトの外部に定義されたプロシージャを呼び出すと、Visual Basic コンパイラはプロシージャを正しく呼び出すために必要な情報にアクセスできません。この情報には、プロシージャが置かれている場所と識別する方法、呼び出しシーケンス、戻り値の型、および使用する文字列の文字セットが含まれます。Declare ステートメント は外部プロシージャの参照を作成して、この必要な情報を提供します。

Declare ステートメントの *charsetmodifier* の部分では、外部プロシージャの呼び出しの間に文字列をマーシャリングするための文字セット情報を提供します。これは、Visual Basic が指定された外部プロシージャ名を、外部ファイルの中から検索する方法にも影響します。**Unicode** 修飾子を指定すると、Visual Basic はすべての文字列を Unicode 値にマーシャリングし、プロシージャの検索中にプロシージャ名を変更しなくなります。

文字セットに修飾子を指定しない場合は **Ansi** が既定値になります。

解説

Unicode 修飾子は次の構文で使用します。

Declare ステートメント

スマート デバイス開発者のためのメモ

このキーワードはサポートされていません。

参照

関連項目

Ansi
Auto
Visual Basic 言語のキーワード

Until

Do ループの実行を終了する条件を指定します。

解説

キーワード **Until** は、次の構文で使用します。

Do...Loop ステートメント

参照

関連項目

Visual Basic 言語のキーワード

When

Catch ステートメントに条件テストを追加します。

解説

キーワード **When** は、次の構文で使用します。

Try...Catch...Finally ステートメント

参照

関連項目

Visual Basic 言語のキーワード

While (Visual Basic)

Do ループの実行を継続する条件を指定します。

解説

While キーワードは、**While...End While** ブロックを開始するステートメントとは異なるものです。

キーワード **While** は、次の構文で使用します。

Do...Loop ステートメント

参照

関連項目

While...End While ステートメント (Visual Basic)

Widening

変換演算子 (**CType**) がクラスまたは構造体を、元の型に格納可能なすべての値を格納できる型に変換することを指定します。

Widening キーワードによる変換

変換プロシージャには、**Widening** の他に **Public Shared** を指定する必要があります。

拡大変換は、実行時には常に正常に行われ、データ消失が発生することはありません。**Single** から **Double**、**Char** から **String**、または派生型から基本型への変換などが例として挙げられます。最後の例が拡大変換なのは、派生型には基本型のすべてのメンバが含まれ、したがって基本型のインスタンスについても同様だからです。

コードでは **Option Strict** が **On** の場合でも、拡大変換に **CType** を使う必要はありません。

キーワード **Widening** は、次の構文で使用します。

Operator ステートメント

参照

処理手順

方法 : 演算子を定義する

関連項目

Operator ステートメント
Narrowing
CType 関数
Option Strict ステートメント

概念

拡大変換と縮小変換

WithEvents

宣言された 1 つ以上のメンバ変数が、イベントを発生させる可能性のあるクラスのインスタンスを参照していることを指定します。

解説

WithEvents を使って変数を定義すると、その変数のイベントを、メソッドが **Handles** キーワードを使用して処理することを宣言によって指定できます。

WithEvents は、クラス レベルまたはモジュール レベルでのみ使用できます。つまり、**WithEvents** 変数の宣言コンテキストは、ソース ファイル、名前空間、構造体、プロシージャではなく、クラスまたはモジュールである必要があります。

WithEvents は、構造体のメンバに対しては使用できません。

WithEvents を使って宣言できるのは、個々の変数だけです (配列は不可)。

規則

- **要素の型が合っていること。WithEvents** はオブジェクト変数に宣言して、変数がクラスのインスタンスを参照できるようにする必要があります。ただし、**Object** 型で宣言することはできません。イベントを発生できる特定のクラスとして宣言する必要があります。

WithEvents 修飾子は、Dim ステートメント (Visual Basic) の構文で使用します。

参照

関連項目

Handles

Visual Basic 言語のキーワード

概念

WithEvents と Handles 句

その他の技術情報

Visual Basic におけるイベント

WriteOnly

プロパティに書き込みはできるが、読み込みはできないことを指定します。

解説

ルール

宣言コンテキスト WriteOnly は、モジュール レベルでのみ使用できます。つまり、**WriteOnly** プロパティの宣言コンテキストは、クラス、構造体、またはモジュールであることが必要で、ソース ファイル、名前空間、プロシージャでは宣言できません。

プロパティは **WriteOnly** で宣言できますが、変数ではできません。

WriteOnly を使用する状況

値の設定はできるが、その値が何かを知ることができないようなコードが必要になる場合があります。たとえば、何らかの登録番号やパスワードなど、それを設定した以外のコンポーネントからアクセスされないよう保護する必要のある重要情報です。このような場合に、**WriteOnly** プロパティを使用して値を設定します。

セキュリティに関するメモ：
WriteOnly プロパティを定義して使用する際には、次のような方法で保護を強化するようにしてください。

- **オーバーライド** プロパティがクラスのメンバである場合は、既定で NotOverridable に設定されるようにし、**Overridable** や **MustOverride** に宣言しないでください。これにより、派生クラスでのオーバーライドによってアクセスが許可されてしまうのを防ぐことができます。
- **アクセス レベル** プロパティの重要情報を 1 つ以上の変数に格納する場合は、変数を Private (Visual Basic) で宣言して、他のコードからアクセスできないようにしてください。
- **暗号化** 重要情報はすべて、平文ではなく暗号化された形式で格納してください。悪意のあるコードが、何らかの方法でメモリ内のその領域にアクセスできた場合でも、データを使用するのが非常に困難になります。また、暗号化は重要情報をシリアル化する必要がある場合にも使用されます。
- **リセット** プロパティが定義されたクラス、構造体、またはモジュールが終了するとき、重要情報を既定値またはその他の意味のない値にリセットしてください。こうすることで、メモリのその領域へのアクセスが一般に解放されたときの保護を強化できます。
- **永続性。** 重要情報は、できればディスクなどに保存しないようにしてください。また、重要情報をクリップボードに書き込むのは避けてください。

WriteOnly 修飾子は次の構文で使用します。

Property ステートメント

参照

関連項目

ReadOnly (Visual Basic)
Private (Visual Basic)
Visual Basic 言語のキーワード

モジュール (Visual Basic)

ここで紹介するトピックでは、Visual Basic のランタイム モジュールについて説明します。また、それぞれのメンバ プロシージャ、プロパティ、定数、および列挙値の表も記載します。

このセクションの内容

Constants モジュール (Visual Basic)

ControlChars モジュール (Visual Basic)

Conversion モジュール (Visual Basic)

DateAndTime モジュール (Visual Basic)

ErrObject モジュール (Visual Basic)

FileSystem モジュール (Visual Basic)

Financial モジュール (Visual Basic)

Globals モジュール (Visual Basic)

Information モジュール (Visual Basic)

Interaction モジュール (Visual Basic)

Strings モジュール (Visual Basic)

VbMath モジュール (Visual Basic)

関連するセクション

Visual Basic リファレンス

Visual Basic

Constants モジュール (Visual Basic)

Constants モジュールには、さまざまな定数が含まれます。この定数は、コード内のどこにでも使用できます。

解説

このモジュールは、Visual Basic のさまざまな列挙体のメンバに対応する Visual Basic 言語定数をサポートしています。

メンバ

vbAbort	vbAbortRetryIgnore	vbApplicationModal	vbArchive
vbArray	vbBack	vbBinaryCompare	vbBoolean
vbByte	vbCancel	vbCr	vbCritical
vbCrLf	vbCurrency	vbDate	vbDecimal
vbDefaultButton1	vbDefaultButton2	vbDefaultButton3	vbDirectory
vbDouble	vbEmpty	vbExclamation	vbFalse
vbFirstFourDays	vbFirstFullWeek	vbFirstJan1	vbFormFeed
vbFriday	vbGeneralDate	vbGet	vbHidden
vbHide	vbHiragana	vbIgnore	vbInformation
vbInteger	vbKatakana	vbLet	vbLf
vbLinguisticCasing	vbLong	vbLongDate	vbLongTime
vbLowerCase	vbMaximizedFocus	vbMethod	vbMinimizedFocus
vbMinimizedNoFocus	vbMonday	vbMsgBoxHelp	vbMsgBoxRight
vbMsgBoxRtlReading	vbMsgBoxSetForeground	vbNarrow	vbNewLine
vbNo	vbNormal	vbNormalFocus	vbNormalNoFocus
vbNull	vbNullChar	vbNullString	vbObject
vbObjectError	vbOK	vbOKCancel	vbOKOnly
vbProperCase	vbQuestion	vbReadOnly	vbRetry
vbRetryCancel	vbSaturday	vbSet	vbShortDate
vbShortTime	vbSimplifiedChinese	vbSingle	vbString
vbSunday	vbSystem	vbSystemModal	vbTab
vbTextCompare	vbThursday	vbTraditionalChinese	vbTrue

vbTuesday	vbUpperCase	vbUseDefault	vbUserDefinedType
vbUseSystem	vbUseSystemDayOfWeek	vbVariant	vbVerticalTab
vbVolume	vbWednesday	vbWide	vbYes
vbYesNo	vbYesNoCancel		

参照

関連項目

AppWinStyle 列挙型
BuiltInRole 列挙型
CallType 列挙型
CompareMethod 列挙型
DateFormat 列挙型
DateInterval 列挙型
DueDate 列挙型
FileAttribute 列挙型
FirstDayOfWeek 列挙型
FirstWeekOfYear 列挙型
MsgBoxResult 列挙型
MsgBoxStyle 列挙型
OpenAccess 列挙型
OpenMode 列挙型
OpenShare 列挙型
印刷と表示の定数
TriState 列挙型
VariantType 列挙型
VbStrConv 列挙型
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

ControlChars モジュール (Visual Basic)

ControlChars モジュールには、制御文字として使用される定数が含まれます。この定数は、コード内のどこにでも使用できます。

解説

このモジュールは、テキストを出力および表示するための定数制御文字を提供する Visual Basic 言語キーワードおよびランタイム ライブラリ メンバをサポートします。詳細については、「印刷と表示の定数」を参照してください。

メンバ

Back	Cr	CrLf	FormFeed
Lf	NewLine	NullChar	Quote
Tab	VerticalTab		

参照

関連項目

印刷と表示の定数
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

Conversion モジュール (Visual Basic)

Conversion モジュールに含まれるプロシージャを使って、さまざまな変換操作を実行します。

解説

このモジュールは、10 進の数値から他のベースへの変換、数値から文字列への変換、文字列から数値への変換、あるデータ型から別のデータ型への変換を行う Visual Basic 言語のキーワードやランタイム ライブラリのメンバをサポートします。

メンバ

ErrorToString	Fix	Hex	Int
Oct	Str	Val	

使用例

Hex 関数を使って 16 進数の数値を返す例を次に示します。

VB

```
Dim TestHex As String
' Returns 5.
TestHex = Hex(5)
' Returns A.
TestHex = Hex(10)
' Returns 1CB.
TestHex = Hex(459)
```

参照

関連項目

変換の概要
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

DateAndTime モジュール (Visual Basic)

DateAndTime モジュールには、日付と時刻の操作で使用されるプロシージャとプロパティが含まれます。

解説

このモジュールは、現在の日時の取得、日付計算の実行、日付または時刻の取得、日付または時刻の設定、または処理期間の記録を扱うための Visual Basic 言語キーワードおよびランタイム ライブラリをサポートします。

メンバ

DateAdd	DateDiff	DatePart	DateSerial
DateString	DateValue	Day	Hour
Minute	Month	MonthName	Now
Second	TimeOfDay	Timer	TimeSerial
TimeString	TimeValue	Today	WeekDay
WeekDayName	Year		

使用例

次の例は、**Today** プロパティを使って、現在のシステムの日付を求めます。

```
Dim thisDate As Date
thisDate = Today
```

参照

関連項目

日付と時刻の概要
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

ErrObject モジュール (Visual Basic)

ErrObject モジュールには、**Err** オブジェクトで実行時エラーを識別および処理するために使われるプロパティとプロシージャが含まれます。

解説

ErrObject モジュールは、**Err** オブジェクトをサポートします。詳細については、「Err オブジェクト (Visual Basic)」を参照してください。

メンバ

Clear	説明	Erl	GetException
HelpContext	HelpFile	LastDLLError	Number
Raise	Source		

使用例

次に示す例では、エラー メッセージ ダイアログ ボックスを作成する処理で **Err** オブジェクトのプロパティを使用しています。**Clear** メソッドを最初に使用したときには、**Raise** メソッドを使って Visual Basic エラーを生成すると、Visual Basic の既定の値が **Err** オブジェクトのプロパティになることに注意してください。

```
Dim Msg As String
' If an error occurs, construct an error message.
On Error Resume Next   ' Defer error handling.
Err.Clear
Err.Raise(6)   ' Generate an "Overflow" error.
' Check for error, then show message.
If Err.Number <> 0 Then
    Msg = "Error # " & Str(Err.Number) & " was generated by " _
        & Err.Source & ControlChars.CrLf & Err.Description
    MsgBox(Msg, MsgBoxStyle.Information, "Error")
End If
```

参照

関連項目

Err オブジェクト (Visual Basic)
変換の概要
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

FileSystem モジュール (Visual Basic)

FileSystem モジュールに含まれるプロシージャを使って、ファイル、ディレクトリまたはフォルダ、およびシステムに対する操作を実行します。

My 機能を使用すると、**FileSystem** を使用するよりもファイル I/O 処理の生産性とパフォーマンスが格段に向上します。詳細については、「My.Computer.FileSystem オブジェクト」を参照してください。

解説

このモジュールは、Visual Basic 言語キーワードと、ファイルおよびフォルダにアクセスするランタイム ライブラリ メンバをサポートします。

メンバ

ChDir	ChDrive	CurDir	Dir
EOF	FileAttr	FileClose	FileCopy
FileDateTime	FileGet	FileGetObject	FileLen
FileOpen	FilePut	FilePutObject	FileWidth
FreeFile	GetAttr	Input	InputString
Kill	LineInput	Loc	Lock
LOF	MkDir	Print	PrintLine
Rename	Reset	RmDir	Seek
SetAttr	SPC	Tab	Unlock
Write	WriteLine		

使用例

GetAttr 関数を使って、ファイルおよびディレクトリまたはフォルダの属性を調べるコード例を次に示します。

VB

```
Dim MyAttr As FileAttribute
' Assume file TESTFILE is normal and readonly.
MyAttr = GetAttr("C:\TESTFILE.txt")   ' Returns vbNormal.

' Test for normal.
If (MyAttr And FileAttribute.Normal) = FileAttribute.Normal Then
   MsgBox("This file is normal.")
End If

' Test for normal and readonly.
Dim normalReadonly As FileAttribute
normalReadonly = FileAttribute.Normal Or FileAttribute.ReadOnly
If (MyAttr And normalReadonly) = normalReadonly Then
   MsgBox("This file is normal and readonly.")
End If

' Assume MYDIR is a directory or folder.
MyAttr = GetAttr("C:\MYDIR")
If (MyAttr And FileAttribute.Directory) = FileAttribute.Directory Then
   MsgBox("MYDIR is a directory")
End If
```

参照

関連項目

ディレクトリとファイルの概要
入出力の概要
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

Financial モジュール (Visual Basic)

Financial モジュールに含まれるプロシージャを使って、財務関連の操作を実行します。

解説

このモジュールは減価償却費、現在価値および将来価値、利率、収益率、および支払額などの財務計算を実行する Visual Basic ランタイム ライブラリ メンバをサポートします。

メンバ

DDB	FV	IPmt	IRR
MIRR	NPer	NPV	Pmt
PPmt	PV	Rate	SLN
SYD			

使用例

次の例は、**Rate** 関数を使って、実際の利率を計算します。支払い回数 (`TotPmts`)、支払い額 (`Payment`)、現在価値または元金 (`PVal`)、将来価値 (`FVal`)、支払い期日 (`PayType`)、利率の推定値 (`Guess`) を指定します。

VB

```
Sub TestRate()
    Dim PVal, Payment, TotPmts, APR As Double
    Dim PayType As DueDate

    ' Define percentage format.
    Dim Fmt As String = "##0.00"
    Dim Response As MsgBoxResult
    ' Usually 0 for a loan.
    Dim FVal As Double = 0
    ' Guess of 10 percent.
    Dim Guess As Double = 0.1
    PVal = CDbl(InputBox("How much did you borrow?"))
    Payment = CDbl(InputBox("What's your monthly payment?"))
    TotPmts = CDbl(InputBox("How many monthly payments do you have to make?"))
    Response = MsgBox("Do you make payments at the end of the month?", MsgBoxStyle.YesNo)
    If Response = MsgBoxResult.No Then
        PayType = DueDate.BegOfPeriod
    Else
        PayType = DueDate.EndOfPeriod
    End If
    APR = (Rate(TotPmts, -Payment, PVal, FVal, PayType, Guess) * 12) * 100

    MsgBox("Your interest rate is " & Format(CInt(APR), Fmt) & " percent.")
End Sub
```

参照

関連項目

財務処理の概要
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

Globals モジュール (Visual Basic)

Globals モジュールには、スクリプト エンジン関数が含まれます。

解説

このモジュールは、現在使用しているランタイムに関する情報を提供する、Visual Basic 言語のキーワードやランタイム ライブラリのメンバをサポートします。

メンバ

ScriptEngine	ScriptEngineBuildVersion	ScriptEngineMajorVersion	ScriptEngineMinorVersion

使用例

次の例では、**ScriptEngineMajorVersion** プロパティおよびその他の関連するプロパティを使用して、現在のランタイム情報を示す文字列を返します。

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

参照

関連項目

ScriptEngine プロパティ
ScriptEngineBuildVersion プロパティ
ScriptEngineMajorVersion プロパティ
ScriptEngineMinorVersion プロパティ
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

Information モジュール (Visual Basic)

Information モジュールに含まれるプロシージャを使って、情報を取得、検査、または確認できます。

解説

このモジュールは、エラー情報の提供、データの検証、配列の範囲の判定、データ型と色情報の取得を行うための Visual Basic の言語キーワードとランタイム ライブラリ メンバをサポートしています。

メンバ

Err	IsArray	IsDate	IsDBNull
IsError	IsNothing	IsNumeric	IsReference
LBound	QBColor	RGB	SystemTypeName
TypeName	UBound	VarType	VbTypeName

使用例

次の例は、**VbTypeName** 関数を使って、いくつかの変数のデータ型名を返します。

```
Dim sysDateName As String = "System.DateTime"
Dim sysShortName As String = "Int16"
Dim sysBadName As String = "Nonsense"
Dim testVbName As String
testVbName = VbTypeName(sysDateName)
' Returns "Date".
testVbName = VbTypeName(sysShortName)
' Returns "Short".
testVbName = VbTypeName(sysBadName)
' Returns Nothing.
```

参照

関連項目

情報と対話の概要
データ型の概要
宣言と定数の概要
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

Interaction モジュール (Visual Basic)

Interaction モジュールに含まれるプロシージャを使って、オブジェクト、アプリケーション、およびシステムと対話します。

解説

このモジュールは他のプログラムの実行、メソッドまたはプロパティの呼び出し、コンピュータからのビープ音の出力、コマンド行文字列の提供、COM オブジェクトの操作、およびダイアログ ボックスの制御を行う Visual Basic 言語キーワードとランタイム ライブラリ メンバをサポートします。

メンバ

AppActivate	Beep	CallByName	Choose
Command	CreateObject	DeleteSetting	Environ
GetAllSettings	GetObject	GetSetting	IIf
InputBox	MsgBox	Partition	SaveSetting
Shell	Switch		

使用例

Shell 関数を使って、ユーザーが指定したアプリケーションを実行するコード例は、次のとおりです。2 番目の引数に Microsoft.VisualBasic.AppWinStyle.NormalFocus を指定することで、アプリケーションは通常のサイズで開き、フォーカスを持ちます。

```
Dim procID As Integer
' Run calculator.
procID = Shell("C:\Windows\system32\calc.exe", AppWinStyle.NormalFocus)
' The preceding path is for Windows XP.
' The Windows 2000 path is C:\WINNT\system32\calc.exe.
```

参照

関連項目

情報と対話の概要
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

Strings モジュール (Visual Basic)

Strings モジュールに含まれるプロシージャを使って、文字列操作を実行します。

解説

このモジュールは、文字列を操作する Visual Basic 言語キーワードとランタイム ライブラリ メンバをサポートします。

メンバ

Asc	AscW	Chr	ChrW
Filter	Format	FormatCurrency	FormatDateTime
FormatNumber	FormatPercent	GetChar	InStr
InStrRev	Join	LCase	Left
Len	LSet	LTrim	Mid
Replace	Right	RSet	RTrim
空白	Split	StrComp	StrConv
StrDup	StrReverse	Trim	UCase

使用例

次の例では、空白で文字列を区切る方法を示します。

VB

```
Dim TestString As String = "Look at these!"
' Returns an array containing "Look", "at", and "these!".
Dim TestArray() As String = Split(TestString)
```

参照

関連項目

文字列操作の概要
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

VbMath モジュール (Visual Basic)

VbMath モジュールに含まれるプロシージャを使って、数値演算を実行します。

解説

このモジュールは、乱数生成用の Visual Basic 言語キーワードとランタイム ライブラリ メンバをサポートします。

メンバ

Randomize	Rnd		

使用例

Rnd 関数を使って、1 ～ 6 の範囲でランダムな整数を生成する例を次に示します。

VB

```
' Initialize the random-number generator.
Randomize()
' Generate random value between 1 and 6.
Dim value As Integer = CInt(Int((6 * Rnd()) + 1))
```

参照

関連項目

数値演算の概要
数値演算関数の導出 (Visual Basic)
キーワードとメンバ (タスク別)
Visual Basic 言語のキーワード
Visual Basic ランタイム ライブラリのメンバ
各言語のキーワードの比較

オブジェクト (Visual Basic)

ここに紹介するトピックでは、Visual Basic のランタイム オブジェクトについて説明します。また、それぞれのメンバ プロシージャ、プロパティ、およびイベントの表も記載します。

このセクションの内容

Collection オブジェクト (Visual Basic)

Collection オブジェクトのメンバ

Err オブジェクト (Visual Basic)

Err オブジェクトのメンバ

関連するセクション

Visual Basic リファレンス

Visual Basic

Collection オブジェクト (Visual Basic)

Visual Basic の **Collection** は、一定の順序で並んだ項目のセットであり、1 つの単位として参照できます。

解説

Visual Basic の **Collection** オブジェクトを利用すると、関連のある項目のグループを単一のオブジェクトとして簡単に参照できます。コレクションの項目、つまり要素に必要な関連性は、コレクション内に存在しているという事実だけです。コレクションの要素は、同じデータ型を共有する必要はありません。

次の例のとおり、コレクションは、その他のオブジェクトと同じように作成できます。

```
Dim coll As New Microsoft.VisualBasic.Collection()
```

コレクションを作成すると、次の作業を行うことができます。

- Add メソッドで要素を追加する
- Remove メソッドで要素を削除する
- Clear メソッドですべての要素を削除する
- Count プロパティでコレクション内の要素数を取得する
- Contains メソッドで特定の要素が存在するかどうかを確認する
- Item プロパティでコレクション内の特定の要素を返す
- For Each...Next ステートメント (Visual Basic)でコレクション全体を反復処理する

メモ：

Visual Basic の **Collection** オブジェクトは、Visual Basic 6.0 の **Collection** オブジェクトと同様の機能を持ちますが、この 2 つは COM 環境で相互運用できません。

注意

Visual Basic の **Collection** の反復処理は、スレッド セーフなプロシージャではありません。コレクションの同期がとられている場合でも、別のスレッドによってそのコレクションを変更できるため、変更の結果として列挙子は例外をスローします。列挙の処理をスレッド セーフにするには、コレクションをロックする方法と、別のスレッドによる変更で発生した例外をキャッチする方法があります。プログラミング要素のロックの詳細については、「SyncLock ステートメント」を参照してください。

使用例

次の例では、**Collection** オブジェクトの `names` と、オブジェクト (名前) をコレクションに追加するためのダイアログ ボックスを作成します。次に、コレクション内の名前を表示し、**Collection** オブジェクト自身は破棄せずにコレクションを空にします。

このプログラムがどのように動作するかを確認するには、[プロジェクト] メニューの [クラスの追加] をクリックし、各インスタンスの名前を保持する変数として、`instanceName` という名前のパブリック変数を `nameClass` のモジュール レベルで宣言します (「`Public instanceName`」と入力)。クラス名は既定の `nameClass` のままにしておきます。以下のコードをコピーして別のモジュールの General セクションに貼り付けた後、別のプロシージャの `classNamer` ステートメントで起動します。この例は、クラスをサポートするホスト アプリケーションでだけ動作します。

VB

```
Public Class nameClass
    Public instanceName As String
End Class
Sub classNamer()
    ' Create a Visual Basic Collection object.
    Dim names As New Microsoft.VisualBasic.Collection()
    Dim key As Integer
    Dim msg As String
    Dim name As String
    Dim nameList As String = ""
    ' 1. Get names from the user to add to the collection.
```

```
    Do
        Dim inst As New nameClass()
        key += 1
        msg = "Please enter a name for this object." & vbCrLf _
            & "Press Cancel to see names in collection."
        name = InputBox(msg, "Name the Collection items")
        inst.instanceName = name
        ' If user entered a name, add it to the collection.
        If inst.instanceName <> "" Then
            names.Add(inst, CStr(key))
        End If
    Loop Until name = ""
    ' 2. Create and display a list of names from the collection.
    For Each oneInst As nameClass In names
        nameList &= oneInst.instanceName & vbCrLf
    Next oneInst
    MsgBox(nameList, , "Instance Names in names Collection")
    ' 3. Remove elements from the collection without disposing of the collection.
    For count As Integer = 1 To names.Count
        names.Remove(1)
        ' Since Visual Basic collections are reindexed automatically,
        ' remove the first member on each iteration.
    Next count
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Collection オブジェクトのメンバ

GetEnumerator メソッド (コレクション オブジェクト)

Collection オブジェクトのメンバ

Visual Basic の **Collection** オブジェクトは、以下のメンバを公開します。

プロパティ

プロパティ	説明
Count プロパティ	読み取り専用です。Visual Basic コレクション内の要素の数を含む整数 (**Integer**) を返します。
Item プロパティ	読み取り専用です。指定した位置またはキーに対応する Visual Basic **Collection** オブジェクトの要素を返します。

メソッド

メソッド	説明
Add メソッド	Visual Basic **Collection** オブジェクトに要素を追加します。
Clear メソッド	Visual Basic **Collection** オブジェクトからすべての要素を削除します。
Contains メソッド (Collection オブジェクト)	指定されたキーを持つ要素が Visual Basic **Collection** オブジェクトに含まれるかどうかを示すブール値 (**Boolean**) を返します。
GetEnumerator メソッド	コレクションを反復処理する列挙子オブジェクトを返します。
Remove メソッド	Visual Basic **Collection** オブジェクトから要素を削除します。

参照

関連項目

Collection オブジェクト (Visual Basic)

Add メソッド (Collection オブジェクト)

Collection オブジェクトに要素を割り当てます。

```
Public Sub Add( _
   ByVal Item As Object, _
   Optional ByVal Key As String, _
   Optional ByVal { Before | After } As Object = Nothing _
)
```

パラメータ

Item

必ず指定します。コレクションに追加する要素を指定する任意の型のオブジェクトです。

Key

省略可能です。位置を示すインデックスの代わりに使用できるキー文字列を表す、一意な文字列 (**String**) 式を指定します。これを使ってコレクション内の新しい要素にアクセスします。

Before

省略可能です。コレクション内の相対位置を式で指定します。追加される要素は、コレクション内の引数 *Before* で識別される要素の前に配置されます。*Before* が数式である場合、1 からコレクションの Count プロパティ (Collection オブジェクト) の値までの範囲の整数を指定する必要があります。文字列 (**String**) 式を使って指定する場合、*Before* には、参照先の要素をコレクションに追加したときに指定したキー文字列に一致する文字列を指定する必要があります。*Before* と *After* の両方を指定することはできません。

After

省略可能です。コレクション内の相対位置を式で指定します。追加されるメンバは、コレクション内の引数 *After* で識別されるメンバの後に配置されます。*After* が数式である場合、1 からコレクションの **Count** プロパティの値までの範囲の整数を指定する必要があります。文字列 (**String**) 式を使って指定する場合、*After* には、参照先の要素をコレクションに追加したときに指定したキー文字列に一致する文字列を指定する必要があります。*Before* と *After* の両方を指定することはできません。

例外/エラー コード

例外の種類	エラー番号	条件
ArgumentException	5	• *Before* および *After* がどちらも指定されている。 • 引数 *Before* または *After* が存在するコレクションの要素を参照していない。 • 指定された *Key* が既に存在する。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

引数の *Before* または *After* は、コレクションの既存の要素を参照する必要があります。それ以外の場合はエラーが発生します。

また、指定した *Key* の値が、コレクションの既存要素のキーと一致する場合にも、エラーが発生します。

使用例

次の例では、**Add** メソッドを使用して `child` オブジェクト (**Public** プロパティ `name` を含む `child` というクラスのインスタンス) を、`family` というコレクションに追加します。この処理を確認するために、2 つの Button コントロールを持つForm を作成し、各ボタンの Text プロパティを **Add** および **List** に設定します。`child` のクラス定義と `family` の宣言をフォーム コードに追加します。Add および List ボタンの `_Click` イベント ハンドラを次のように変更します。Add ボタンをクリックすると子を追加できます。List ボタンをクリックするとすべての子の名前を表示します。

VB

```
Public Class child
    Public name As String
    Sub New(ByVal newName As String)
        name = newName
```

```
    End Sub
End Class
' Create a Collection object.
Private family As New Collection()
Private Sub addChild_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim newName As String
    newName = InputBox("Name of new family member: ")
    If newName <> "" Then
        family.Add(New child(newName), newName)
    End If
End Sub
Private Sub listChild_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button2.Click
    For Each aChild As child In family
        MsgBox(aChild.name)
    Next
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Collection**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Collection オブジェクト (Visual Basic)
Item プロパティ (Collection オブジェクト)
Remove メソッド (Collection オブジェクト)

Clear メソッド (Collection オブジェクト)

Visual Basic **Collection** オブジェクトのすべての要素を削除します。

```
Public Sub Clear()
```

解説

Clear メソッドはコレクションを空にして、コレクションの **Count** プロパティを 0 にリセットします。

使用例

VB

```
Dim customers As New Microsoft.VisualBasic.Collection()
' Insert code that adds customers to collection.
customers.Clear()
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Collection**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Collection オブジェクト (Visual Basic)
Item プロパティ (Collection オブジェクト)
Count プロパティ (Collection オブジェクト)
Contains メソッド (Collection オブジェクト)
Add メソッド (Collection オブジェクト)
Remove メソッド (Collection オブジェクト)

Contains メソッド (Collection オブジェクト)

指定されたキーを持つ要素が Visual Basic **Collection** オブジェクトに含まれるかどうかを示すブール値 (**Boolean**) を返します。

```
Public Function Contains( _
    ByVal Key As String _
) As Boolean
```

パラメータ
Key

必ず指定します。コレクションから要素を検索するためのキーを指定する **String** 式です。

例外/エラー コード

例外の種類	エラー番号	条件
ArgumentException	5	指定された *Key* が **Nothing** です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Contains は、コレクションに *Key* とまったく同じキーを持つ要素が含まれている場合に **True** を返します。それ以外の場合、**Contains** は **False** を返します。

Visual Basic **Collection** に格納される要素には、キーを持つものと持たないものがあります。これは、Add メソッド (Collection オブジェクト) を呼び出すとき、オプションの *Key* パラメータに引数が指定されたかどうかによって決まります。

使用例
VB

```
Dim customers As New Microsoft.VisualBasic.Collection()
Dim accountNumber As String = "1234"
' Insert code that obtains new customer objects.
' Use the new customer's account number as the key.
customers.Add(newCustomer, accountNumber)
' The preceding statements can be repeated for several customers.
Dim searchNumber As String = "1234"
' Insert code to obtain an account number to search for.
If customers.Contains(searchNumber) Then
    MsgBox("The desired customer is in the collection.")
Else
    MsgBox("The desired customer is not in the collection.")
End If
```

コレクションの要素を、キーを使って検索するのであれば、**Add** メソッドを呼び出すとき必ず引数 *Key* を指定することを忘れないでください。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Collection**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
関連項目
Collection オブジェクト (Visual Basic)
Item プロパティ (Collection オブジェクト)
Count プロパティ (Collection オブジェクト)
Add メソッド (Collection オブジェクト)
Remove メソッド (Collection オブジェクト)

Clear メソッド (Collection オブジェクト)

Count プロパティ (Collection オブジェクト)

コレクション内の要素数を含む整数 (**Integer**) を返します。読み取り専用です。

```
Public ReadOnly Property Count() As Integer
```

解説

Count プロパティを使用すると、**Collection** オブジェクトの要素の数を調べることができます。

使用例

Count プロパティを使って、変数 `birthdays` の Collection オブジェクト (Visual Basic) に含まれる要素の数を表示する例を次に示します。

VB

```
    Dim birthdays As New Collection()
    birthdays.Add(New DateTime(2001, 1, 12), "Bill")
    birthdays.Add(New DateTime(2001, 1, 13), "Joe")
    birthdays.Add(New DateTime(2001, 1, 14), "Mike")
    birthdays.Add(New DateTime(2001, 1, 15), "Pete")
...    MsgBox("Number of birthdays in collection: " & CStr(birthdays.Count))
```

Collection オブジェクトでは、要素のインデックス値が 1 から始まるため、インデックスは 1 ～ **Count** プロパティの値の範囲をとります。

必要条件

名前空間 : Microsoft.VisualBasic

クラス : **Collection**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Collection オブジェクト (Visual Basic)
Add メソッド (Collection オブジェクト)
Item プロパティ (Collection オブジェクト)
Remove メソッド (Collection オブジェクト)

GetEnumerator メソッド (コレクション オブジェクト)

Collection オブジェクト (Visual Basic) を反復処理するのに使用される列挙子オブジェクトへの参照を返します。

```
Public Function GetEnumerator() As IEnumerator
```

解説

For Each...Next ステートメント (Visual Basic) は **GetEnumerator** を呼び出し、列挙子オブジェクトを取得して、コレクション内の要素の反復処理をサポートします。通常、コレクションまたは配列を走査するには **For Each...Next** ループを使用し、明示的に **GetEnumerator** を呼び出す必要はありません。

反復処理を、**For Each...Next** ステートメントよりも細かく制御する必要がある場合は、**GetEnumerator** メソッドを使用してカスタマイズされた走査を実行します。このような作業が必要なケースを次に示します。

- 反復処理が完了する前に、コレクションの先頭に戻って反復処理を再実行する必要がある。
- 何らかの理由で一部の要素をスキップする必要がある。
- 走査の実行中にコレクション内の要素に変更を加える必要がある。この場合、列挙子オブジェクトが無効になるため、新しい列挙子オブジェクトを取得する必要があります。

使用例

次の例では、**GetEnumerator** を使用して **Collection** オブジェクトのすべての要素を取得する方法を示します。

VB

```
Dim customers As New Collection
' Insert code to add elements to the customers collection.
Dim custEnum As IEnumerator = customers.GetEnumerator()
custEnum.Reset()
Dim thisCustomer As Object
While custEnum.MoveNext()
    thisCustomer = custEnum.Current()
    ' Insert code to process this element of the collection.
End While
```

GetEnumerator は、System.Collections 名前空間の IEnumerator インターフェイスを実装する列挙子オブジェクトを構築し、これを返します。この列挙子オブジェクトは、Current プロパティと MoveNext および Reset メソッドを公開します。詳細については、「For Each...Next ステートメント (Visual Basic)」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : Collection

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Collection オブジェクト (Visual Basic)

Item プロパティ (Collection オブジェクト)

指定した位置またはキーに対応する **Collection** オブジェクトの要素を返します。読み取り専用です。

```
Default Public ReadOnly Property Item( _
    ByVal { Key As String | Index As Integer | Index As Object } _
) As Object
```

パラメータ

Key

コレクションの要素へアクセスするために使用できる、(位置を示すインデックスではなく) キー文字列を指定する一意な **String** 式です。*Key* は、コレクションに要素を追加するときに指定した *Key* 引数と一致している必要があります。

Index

以下の (A) または (B) です。(A) コレクションの要素の位置を指定する数式です。*Index* は 1 からコレクションの Count プロパティ (Collection オブジェクト) までの値であることが必要です。(B) コレクションの要素の位置またはキー文字列を指定する **Object** 式です。

例外

例外の種類	エラー番号	条件
ArgumentException	5	• *Key* が無効、またはコレクションの既存の要素のいずれにも一致しません。 • *Index* を文字データまたは数値データに解釈できません。
IndexOutOfRangeException	9	• *Key* が **Nothing** です。 • *Index* がコレクションの既存の要素のいずれにも一致しません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Index の型が **Object** の場合、**Item** プロパティはこれを **String**、**Char**、**Char** 配列、または整数値として扱うことを試みます。*Index* を **String** や **Integer** に変換できなければ、**Item** は **ArgumentException** 例外をスローします。

Item プロパティは、コレクションの既定のプロパティです。したがって、次の 2 つのコード行は同じになります。

```
MsgBox(CStr(customers.Item(1)))
MsgBox(CStr(customers(1)))
```

使用例

Item プロパティを使って、コレクションに含まれるオブジェクトへの参照を取得する例を次に示します。`birthdays` を **Collection** オブジェクトとして作成し、*Index* 引数に `"Bill"` というキーを指定して、Bill の誕生日を表すオブジェクトを取得します。

VB

```
    Dim birthdays As New Collection()
    birthdays.Add(New DateTime(2001, 1, 12), "Bill")
    birthdays.Add(New DateTime(2001, 1, 13), "Joe")
    birthdays.Add(New DateTime(2001, 1, 14), "Mike")
    birthdays.Add(New DateTime(2001, 1, 15), "Pete")
...    Dim aBirthday As DateTime
    aBirthday = birthdays.Item("Bill")
    MsgBox(CStr(aBirthday))
    aBirthday = birthdays("Bill")
    MsgBox(CStr(aBirthday))
```

最初の呼び出しでは **Item** プロパティが明示的に指定されていますが、2 回目の呼び出しでは明示的に指定されていないことに注意してください。**Collection** オブジェクトでは、**Item** プロパティが既定となるため、いずれの呼び出し方法でも同じ結果になります。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Collection**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Collection オブジェクト (Visual Basic)
Add メソッド (Collection オブジェクト)
Count プロパティ (Collection オブジェクト)
Remove メソッド (Collection オブジェクト)

Remove メソッド (Collection オブジェクト)

Collection オブジェクトから要素を削除します。

```
Public Overloads Sub Remove(_
    ByVal { Key As String | Index As Integer } _
)
```

パラメータ

Key

コレクションの要素へアクセスするために使用できる、(位置を示すインデックスではなく) キー文字列を指定する一意な **String** 式です。*Key* は、コレクションに要素を追加するときに指定した *Key* 引数と一致している必要があります。

Index

コレクションの要素の位置を特定する数式です。*Index* は 1 からコレクションの Count プロパティ (Collection オブジェクト) までの値であることが必要です。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Key* が無効であるか、指定されていません。
IndexOutOfRangeException	9	*Index* がコレクションの既存の要素のいずれにも一致しません。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

Remove はコレクションから要素を削除すると、コレクションの Count プロパティ (Collection オブジェクト) を 1 ずつ減算します。また、削除された要素の後に続くすべての要素の *Index* 値もデクリメントします。

Key を指定せずにコレクションに追加した要素は、*Index* を使わなければ削除できません。

使用例

Remove メソッドを使って、変数 `birthdays` の Collection オブジェクト (Visual Basic) からオブジェクトを削除する例を次に示します。

VB

```
    Dim birthdays As New Collection()
    birthdays.Add(New DateTime(2001, 1, 12), "Bill")
    birthdays.Add(New DateTime(2001, 1, 13), "Joe")
    birthdays.Add(New DateTime(2001, 1, 14), "Mike")
    birthdays.Add(New DateTime(2001, 1, 15), "Pete")
...    birthdays.Remove(1)
    birthdays.Remove("Mike")
```

Add メソッドを 4 回呼び出した後、**Count** プロパティには 4 が格納され、要素 `"Bill"` のインデックスの値は 1、また要素 `"Pete"` のインデックスの値は 4 になります。

最初に **Remove** を呼び出した後、**Count** は 3 になり、要素 `"Bill"` が削除され、要素 `"Pete"` のインデックスの値は 3 になります。

2 度目の **Remove** 呼び出しの後、**Count** は 2 になり、要素 `"Mike"` が削除され、要素 `"Pete"` のインデックスの値は 2 になります。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Collection**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Collection オブジェクト (Visual Basic)

Add メソッド (Collection オブジェクト)

Err オブジェクト (Visual Basic)

Err オブジェクトは、ランタイム エラーに関する情報を保有しています。

解説

Err オブジェクトのプロパティは、エラーを発生させた Visual Basic、オブジェクト、またはプログラマによって設定されます。

ランタイム エラーが発生すると、そのエラーを一意に識別するための情報と、エラー処理に使用できる情報が、**Err** オブジェクトに格納されます。コード内でランタイム エラーを生成するには、**Raise** メソッドを使います。

Err オブジェクトの各プロパティは、エラー処理ルーチン内の **Exit Sub**、**Exit Function**、**Exit Property**、**Resume Next** ステートメントの後で、0 または長さ 0 の文字列 ("") にリセットされます。エラー処理ルーチンの外側で **Resume** ステートメントを使用した場合は、**Err** オブジェクトのプロパティはリセットされません。**Err** オブジェクトを明示的にリセットするには **Clear** メソッドを使います。

システム エラーおよびクラス モジュールに対してランタイム エラーを生成するには、**Error** ステートメントではなく、**Raise** メソッドを使います。他のコードで **Raise** メソッドを使うかどうかは、返される情報の量によって決まります。

Err オブジェクトは、グローバル スコープの組み込みオブジェクトです。したがって、コード内でオブジェクトのインスタンスを作成する必要はありません。

使用例

次に示す例では、エラー メッセージ ダイアログ ボックスを作成する処理で **Err** オブジェクトのプロパティを使用しています。**Clear** メソッドを最初に使用した場合は、**Raise** メソッドを使って Visual Basic エラーを生成すると、Visual Basic の既定の値が **Err** オブジェクトのプロパティになることに注意してください。

VB

```
Dim Msg As String
' If an error occurs, construct an error message.
On Error Resume Next   ' Defer error handling.
Err.Clear()
Err.Raise(6)   ' Generate an "Overflow" error.
' Check for error, then show message.
If Err.Number <> 0 Then
  Msg = "Error # " & Str(Err.Number) & " was generated by " _
      & Err.Source & ControlChars.CrLf & Err.Description
  MsgBox(Msg, MsgBoxStyle.Information, "Error")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : Visual Basic ランタイム エラーに関する情報を取得する
方法 : エラー オブジェクトから情報を取得する

関連項目

Err オブジェクトのメンバ
Error ステートメント
On Error ステートメント (Visual Basic)
Exit ステートメント (Visual Basic)
Resume ステートメント

Err オブジェクトのメンバ

Err オブジェクトには、発生したエラーによって値が決まるプロパティがあります。**Number** プロパティには、エラーの原因が含まれています。**Description** プロパティには、エラーの詳細を説明するテキスト メッセージが含まれています。**Helpfile** と **HelpContext** を使用すると、ユーザーが **Help** ボタンまたは F1 キーを押したときに、関連するヘルプ ファイルを表示させることができます。**LastDLLError** は、最後に呼び出された DLL と、呼び出しが正常に実行されたかどうかを示します。**Source** は、エラーを生成したオブジェクトまたはプログラムを表す文字列式を示します。

プロパティ

Description プロパティ	読み取り/書き込みプロパティです。エラーと関連付けられたエラーを説明する文字列を取得または設定します。
Erl プロパティ	読み取り専用です。最後に実行されたステートメントの行番号を示す整数を返します。
HelpContext プロパティ	読み取り/書き込みプロパティです。ヘルプ ファイルのトピックを表すコンテキスト ID を含む整数 (**Integer**) を設定します。値の取得も可能です。
HelpFile プロパティ	読み取り/書き込みプロパティです。ヘルプ ファイルへの絶対パスを含む文字列 (**String**) 式を設定します。値の取得も可能です。
LastDLLError プロパティ	読み取り専用です。ダイナミック リンク ライブラリ (DLL) の呼び出しにより作成されたシステム エラー コードを返します。
Number プロパティ	読み取り/書き込みプロパティです。エラーを指定する数値を設定します。値の取得も可能です。
Source プロパティ	読み取り/書き込みプロパティです。最初にエラーを発生させたオブジェクトまたはアプリケーションの名前を示す文字列 (**String**) 式を設定します。値の取得も可能です。

メソッド

Clear メソッド	**Err** オブジェクトのすべてのプロパティの設定値をクリアします。
Raise メソッド	ランタイム エラーを生成します。**Error** ステートメントの代わりに使用できます。

参照

処理手順

方法 : エラー オブジェクトから情報を取得する

関連項目

Err オブジェクト (Visual Basic)

Clear メソッド (Err オブジェクト)

Err オブジェクトのすべてのプロパティの設定値をクリアします。

```
object.Clear
```

パラメータ
object

常に **Err** オブジェクトです。

解説

On Error Resume Next ステートメントを使用した後でエラー処理を行うときなど、エラー処理後に、**Clear** を使って **Err** オブジェクトを明示的にクリアします。**Clear** メソッドが自動的に呼び出されるのは、次のいずれかのステートメントの実行時です。

- 任意の型の **Resume** ステートメント
- **Exit Sub**、**Exit Function**、または **Exit Property**
- 任意の **On Error** ステートメント
- 任意の **Try...Catch...Finally** ステートメント

> **メモ :**
>
> 他のオブジェクトへのアクセス中に生成されたエラーを処理する場合は、**On Error GoTo** よりも **On Error Resume Next** 構成要素の方が適しています。オブジェクトを操作した後の **Err** を毎回調べることで、該当するコードによってアクセスされたオブジェクトが明らかになります。**Err.Number** にエラー コードを設定したオブジェクトと最初にエラーを生成したオブジェクト (**Err.Source** で指定されるオブジェクト) についても確認できます。

使用例

次の例は、**Err** オブジェクトの **Clear** メソッドを使って、**Err** オブジェクトの数値型のプロパティを 0 に、文字列型のプロパティを長さ 0 の文字列にリセットします。**Clear** を呼び出さずに **MsgBox** をもう一度呼び出すと、同じエラー メッセージが表示されます。

VB

```
Sub ClearErr()
  ' Produce overflow error
  On Error Resume Next
  Dim zero As Integer = 0
  Dim result As Integer = 8 / zero
  MsgBox(Err.Description)
  Err.Clear()
  MsgBox(Err.Description)
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **ErrObject**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照
処理手順
方法 : エラー オブジェクトから情報を取得する
関連項目
Err オブジェクト (Visual Basic)
Description プロパティ (Err オブジェクト)
HelpContext プロパティ (Err オブジェクト)

HelpFile プロパティ (Err オブジェクト)
LastDllError プロパティ (Err オブジェクト)
Number プロパティ (Err オブジェクト)
On Error ステートメント (Visual Basic)
Raise メソッド (Err オブジェクト)
Source プロパティ (Err オブジェクト)

Description プロパティ (Err オブジェクト)

エラーと関連付けられたエラーを説明する文字列を含む文字列 (**String**) 式を設定します。値の取得も可能です。読み取り/書き込みプロパティです。

```
Public Property Description() As String
```

解説

Description プロパティには、エラーに関する簡単な説明を定義します。このプロパティを使用して、処理できないエラーまたは処理しないエラーをユーザーに警告してください。ユーザー定義エラーを発生させるときには、エラーに関する簡単な説明を **Description** プロパティに割り当てます。**Description** プロパティを設定せず、Number プロパティの値が Visual Basic 実行時エラーに相当する場合は、ErrorToString 関数によって返された文字列が、エラー発生時に **Description** プロパティに設定されます。

使用例

Err オブジェクトの **Description** プロパティにユーザー定義のメッセージを設定する例を次に示します。

VB

```
On Error Resume Next
Err.Raise(60000)
Err.Description = "Your Widget needs a new Frob!"
MsgBox(Err.Description)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **ErrObject**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Err オブジェクト (Visual Basic)
ErrorToString 関数
HelpContext プロパティ (Err オブジェクト)
HelpFile プロパティ (Err オブジェクト)
LastDllError プロパティ (Err オブジェクト)
Number プロパティ (Err オブジェクト)
Source プロパティ (Err オブジェクト)

Erl プロパティ (Err オブジェクト)

最後に実行されたステートメントの行番号を示す整数を返します。読み取り専用です。

```
Public ReadOnly Property Erl() As Integer
```

解説

Visual Basic が該当する行番号を見つけられない場合は、0 を返します。

使用例

Erl プロパティを使って行番号を表示する例を次に示します。

VB

```
10:     On Error Resume Next
20:     Err.Raise(60000)
' Returns 20.
30:     MsgBox(Erl())
```

スマート デバイス開発者のためのメモ

このプロパティはサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **ErrObject**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Err オブジェクト (Visual Basic)

ErrorToString 関数

HelpContext プロパティ (Err オブジェクト)

ヘルプ ファイルのトピックを表すコンテキスト ID を含む整数 (**Integer**) を設定します。値の取得も可能です。読み取り/書き込みプロパティです。

```
Public Property HelpContext() As Integer
```

解説

HelpContext プロパティを使って、アプリケーションの状況依存のヘルプを表示します。ヘルプ ファイルが **HelpFile** プロパティの中で指定されている場合、**HelpContext** プロパティは識別されたヘルプ ファイルを自動的に表示します。**HelpFile** プロパティと **HelpContext** プロパティが空の場合、Number プロパティの値がチェックされます。**Number** プロパティの値が Visual Basic 実行時エラーの値に対応しているときは、Visual Basic ヘルプのコンテキスト ID が使われます。**Number** プロパティの値が Visual Basic エラーに対応していないときは、Visual Basic ヘルプ ファイルの目次画面が表示されます。

メモ :

一般的なエラーを処理するには、アプリケーションでルーチンを記述する必要があります。オブジェクトを使ってプログラミングする場合、そのオブジェクトのヘルプ ファイルを使って、エラー処理の質を高めたり、またはエラーが回復できないときにユーザーにわかりやすいメッセージを表示したりできます。

使用例

Err オブジェクトの **HelpContext** プロパティを使って、`Overflow` エラーの Visual Basic ヘルプ トピックを表示する例を次に示します。

VB

```
Dim Msg As String
Err.Clear()
On Error Resume Next   ' Suppress errors for demonstration purposes.
Err.Raise(6)   ' Generate "Overflow" error.
If Err.Number <> 0 Then
  Msg = "Press F1 or HELP to see " & Err.HelpFile & " topic for" & _
  " the following HelpContext: " & Err.HelpContext
  MsgBox(Msg, , "Error:")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **ErrObject**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Err オブジェクト (Visual Basic)
Description プロパティ (Err オブジェクト)
ErrorToString 関数
HelpFile プロパティ (Err オブジェクト)
LastDllError プロパティ (Err オブジェクト)
Number プロパティ (Err オブジェクト)
Source プロパティ (Err オブジェクト)

HelpFile プロパティ (Err オブジェクト)

ヘルプ ファイルへの絶対パスを含む文字列 (**String**) 式を設定します。値の取得も可能です。読み取り/書き込みプロパティです。

```
Public Property HelpFile() As String
```

解説

ヘルプ ファイルを **HelpFile** プロパティで指定すると、エラー メッセージが表示されるダイアログ ボックスの [ヘルプ] をクリックするか、または F1 キーを押したときにヘルプ ファイルが自動的に呼び出されます。**HelpContext** プロパティに指定したファイルの有効なコンテキスト ID が格納されている場合、そのトピックが自動的に表示されます。**HelpFile** プロパティを指定していない場合は、Visual Basic のヘルプ ファイルが表示されます。

メモ :

一般的なエラーの処理には、アプリケーションにルーチンを記述してください。オブジェクトを使ってプログラミングする場合、そのオブジェクトのヘルプ ファイルを使って、エラー処理の質を高めたり、またはエラーが回復できないときにユーザーにわかりやすいメッセージを表示したりできます。

使用例

Err オブジェクトの **HelpFile** プロパティを使って、ヘルプ システムを起動する例を次に示します。既定では、**HelpFile** プロパティには Visual Basic ヘルプ ファイルの名前が格納されます。

VB

```
Dim Msg As String
Err.Clear()
On Error Resume Next   ' Suppress errors for demonstration purposes.
Err.Raise(6)   ' Generate "Overflow" error.
If Err.Number <> 0 Then
  Msg = "Press F1 or HELP to see " & Err.HelpFile & " topic for" & _
  " the following HelpContext: " & Err.HelpContext
  MsgBox(Msg, , "Error:")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **ErrObject**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Err オブジェクト (Visual Basic)
Description プロパティ (Err オブジェクト)
HelpContext プロパティ (Err オブジェクト)
LastDllError プロパティ (Err オブジェクト)
Number プロパティ (Err オブジェクト)
Source プロパティ (Err オブジェクト)

LastDllError プロパティ (Err オブジェクト)

ダイナミック リンク ライブラリ (DLL) の呼び出しにより作成されたシステム エラー コードを返します。読み取り専用です。

```
ReadOnly Property LastDllError() As Integer
```

解説

LastDllError プロパティは Visual Basic コードから作成された DLL 呼び出しだけに適用されます。呼び出しが実行されると、呼び出された関数は、通常、成功か失敗かを示すコードを返し、**LastDllError** プロパティを設定します。戻り値が成功と失敗のどちらを示しているかを判断するには、DLL の関数のドキュメントを参照してください。失敗を表すコードが返されると、Visual Basic アプリケーションはすぐに **LastDllError** プロパティを確認します。**LastDllError** プロパティが設定される場合、例外はまったく発生しません。

メモ :

LastDllError プロパティにはアンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

使用例

次の例では、Windows API で関数を呼び出した後で **LastDllError** プロパティを使用する方法を示します。**PrintWindowCoordinates** プロシージャは、ウィンドウのハンドルを取得し、**GetWindowRect** 関数を呼び出します。**GetWindowRect** は RECT データ構造に、ウィンドウを構成する四角形の各辺の長さを格納します。無効なハンドルを渡すとエラーが発生し、このエラー番号は **LastDllError** プロパティで参照できます。

VB

```
    Declare Function GetWindowRect Lib "user32" _
    (ByVal hwnd As Integer, ByRef lpRect As RECT) As Integer
...
    Public Structure RECT
        Public Left As Integer
        Public Top As Integer
        Public Right As Integer
        Public Bottom As Integer
    End Structure
...
    Const ERROR_INVALID_WINDOW_HANDLE As Long = 1400
    Const ERROR_INVALID_WINDOW_HANDLE_DESCR As String = _
    "Invalid window handle."
```

VB

```
  Private Sub PrintWindowCoordinates(ByVal hwnd As Integer)
  ' Prints left, right, top, and bottom positions
  ' of a window in pixels.

    Dim rectWindow As RECT

    ' Pass in window handle and empty the data structure.
    ' If function returns 0, an error occurred.
    If GetWindowRect(hwnd, rectWindow) = 0 Then
        ' Check LastDllError and display a dialog box if the error
        ' occurred because an invalid handle was passed.
        If Err.LastDllError = ERROR_INVALID_WINDOW_HANDLE Then
            MsgBox(ERROR_INVALID_WINDOW_HANDLE_DESCR, Title:="Error!")
        End If
    Else
        Debug.Print(rectWindow.Bottom)
        Debug.Print(rectWindow.Left)
        Debug.Print(rectWindow.Right)
```

```
        Debug.Print(rectWindow.Top)
    End If
End Sub
```

スマート デバイス開発者のためのメモ

このプロパティは常にゼロを返します。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **ErrObject**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Err オブジェクト (Visual Basic)
Declare ステートメント
Description プロパティ (Err オブジェクト)
ErrorToString 関数
HelpContext プロパティ (Err オブジェクト)
HelpFile プロパティ (Err オブジェクト)
Number プロパティ (Err オブジェクト)
Source プロパティ (Err オブジェクト)

Number プロパティ (Err オブジェクト)

エラーを指定する数値を設定します。値の取得も可能です。読み取り/書き込みプロパティです。

```
Public Property Number() As Integer
```

解説

オブジェクトからユーザー定義エラーを返す場合は、エラー コードとして選択した数値を定数 **VbObjectError** に追加して、**Err.Number** を設定します。たとえば、次のコードでは、数値の 1051 をエラー コードとして返します。

VB

```
Err.Raise(Number:=vbObjectError + 1051, Source:="SomeClass")
```

使用例

エラー処理ルーチンにおける **Number** プロパティの通常の使用例を次に示します。

VB

```
' Typical use of Number property.
Sub test()
  On Error GoTo out

  Dim x, y As Integer
  x = 1 / y   ' Create division by zero error.
  Exit Sub
out:
  MsgBox(Err.Number)
  MsgBox(Err.Description)
  ' Check for division by zero error.
  If Err.Number = 11 Then
      y = y + 1
  End If
  Resume Next
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **ErrObject**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Err オブジェクト (Visual Basic)
Description プロパティ (Err オブジェクト)
ErrorToString 関数
HelpContext プロパティ (Err オブジェクト)
HelpFile プロパティ (Err オブジェクト)
LastDllError プロパティ (Err オブジェクト)
Source プロパティ (Err オブジェクト)

Raise メソッド (Err オブジェクト)

ランタイム エラーを生成します。**Error** ステートメントの代わりに使用できます。

```
Public Sub Raise( _
   ByVal Number As Integer, _
   Optional ByVal Source As Object = Nothing, _
   Optional ByVal Description As Object = Nothing, _
   Optional ByVal HelpFile As Object = Nothing, _
   Optional ByVal HelpContext As Object = Nothing _
)
```

パラメータ

Number

必ず指定します。エラーに割り当てられているエラー番号を示す、長整数型 (**Long**) の整数を指定します。Visual Basic の各種エラーは、0 ～ 65535 の範囲内にあります。0 ～ 512 はシステム エラー用に予約されており、513 ～ 65535 はユーザー定義のエラーに利用できます。クラス モジュール内で **Number** プロパティに独自のエラー番号を設定するときには、エラー番号を定数 **vbObjectError** に追加します。たとえば、エラー番号 513 を発生させるには、**Number** プロパティに `vbObjectError + 513` を割り当てます。

Source

エラーを発生させたオブジェクトまたはアプリケーションを指定する文字列 (**String**) 式です。このプロパティをオブジェクトに設定する場合は、*project.class* という形式を使用します。*Source* が指定されていない場合、現在の Visual Basic プロヘクトのプロセス ID が使用されます。

Description

エラー メッセージを表す文字列 (**String**) 式を指定します。省略すると、**Number** プロパティの値が使われます。Visual Basic の実行時エラー番号に対応しているときは、**Error** 関数によって返される文字列を **Description** プロパティとして使用します。**Number** プロパティに対応する Visual Basic エラー番号がないときは、"アプリケーション定義またはオブジェクト定義のエラーです。" メッセージが使用されます。

HelpFile

省略可能です。このエラーに関するヘルプ トピックが含まれるヘルプ ファイルの絶対パスを指定します。ヘルプ ファイルの絶対パスを指定しない場合は、Visual Basic ヘルプ ファイルのドライブ、パス、およびファイル名が使用されます。

HelpContext

省略可能です。*HelpFile* 内の指定したエラーに関するトピックのコンテキスト番号を指定します。このコンテキスト番号を省略すると、**Number** プロパティの値が Visual Basic ヘルプ ファイルのコンテキスト番号に対応しているときは、その番号が使われます。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Number* が 65535 を超えています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

Number を除き、**Raise** メソッドのすべての引数は省略できます。引数を省略する場合は、**Err** オブジェクトの各プロパティの設定値がクリアされていないと、その値がエラーを表す値として使用されます。

Error ステートメントでエラーを生成した場合よりも、**Err** オブジェクトの方がより充実した情報を提供できるので、**Raise** は、クラス モジュールの作成時にエラーを生成する野に役立ちます。たとえば、**Raise** メソッドを使って、エラーを発生させたソースを **Source** プロパティに指定したり、エラーのオンライン ドキュメントを参照したりできます。

使用例

Err オブジェクトの **Raise** メソッドを使って、Visual Basic で記述された関数の中でエラーを発生させる例を次に示します。呼び出し元の関数は、エラーを検出してメッセージ ボックスでユーザーに表示できます。

VB

```
Const WidthError As Integer = 1
Const WidthHelp As Object = 101

Sub TestWidth(ByVal width As Integer)
  If width > 1000 Then
      Err.Raise(vbObjectError + 512 + WidthError, "TestWidth", _
      "Width must be less than 1000.", "HelpFile.hlp", WidthHelp)
  End If
End Sub

Sub CallingProcedure()
  Try
    TestWidth(2000)
  Catch ex As Exception
    MsgBox(ex.Message)
  End Try
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **ErrObject**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Err オブジェクト (Visual Basic)
Clear メソッド (Err オブジェクト)
Description プロパティ (Err オブジェクト)
Error ステートメント
HelpContext プロパティ (Err オブジェクト)
HelpFile プロパティ (Err オブジェクト)
LastDllError プロパティ (Err オブジェクト)
Number プロパティ (Err オブジェクト)
On Error ステートメント (Visual Basic)
Source プロパティ (Err オブジェクト)
ArgumentException Class

Source プロパティ (Err オブジェクト)

最初にエラーを発生させたオブジェクトまたはアプリケーションの名前を示す文字列 (**String**) 式を設定します。値の取得も可能です。読み取り/書き込みプロパティです。

```
Public Property Source() As String
```

解説

Source プロパティは、エラーを発生させたオブジェクトを表す **String** 式を指定します。通常、式はオブジェクトのクラス名またはプロセス ID です。操作したオブジェクトの中で発生したエラーを処理できない場合に、**Source** プロパティを使って情報を提供します。たとえば、Microsoft Excel にアクセスしたときに `Division by zero` エラーが発生すると、**Err.Number** プロパティには Microsoft Excel によってエラー コードが設定され、**Source** プロパティには "Excel.Application" という文字列が設定されます。

アプリケーションがコードからエラーを生成する場合、アプリケーションのプログラム ID は **Source** です。クラス内では、**Source** には *project.class* という形式の名前が含まれます。コード内で予期しないエラーが発生した場合、**Source** プロパティには自動的に値が割り当てられます。モジュール内のエラーについては、**Source** にプロジェクト名が含まれます。

使用例

通常のエラー処理ルーチンにおける **Source** プロパティの使用例を次に示します。エラーが `Class1` から発生する場合、文字列 `"Class1"` は **Err** オブジェクトの **Source** プロパティに割り当てられます。また、この文字列は、エラーの発生元および番号を示す、有益なメッセージに表示されます。

VB

```
Public Class Class1
  Public Sub MySub()
      On Error Resume Next
      Err.Raise(60000, "Class1")
      MsgBox(Err.Source & " caused an error of type " & Err.Number)
  End Sub
End Class
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **ErrObject**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Err オブジェクト (Visual Basic)
Description プロパティ (Err オブジェクト)
ErrorToString 関数
GetObject 関数 (Visual Basic)
HelpContext プロパティ (Err オブジェクト)
HelpContext プロパティ (Err オブジェクト)
HelpFile プロパティ (Err オブジェクト)
LastDllError プロパティ (Err オブジェクト)
Number プロパティ (Err オブジェクト)
On Error ステートメント (Visual Basic)

My.Application オブジェクト

現在のアプリケーションに関するプロパティ、メソッド、およびイベントを提供します。

解説

My.Application オブジェクトから公開されるプロパティは、現在のアプリケーションまたは DLL に関連付けられたデータだけを返します。**My.Application** を使ってシステム レベルの情報を変更することはできません。

一部のメンバは、Windows フォーム アプリケーションまたはコンソール アプリケーションだけで使用できます。

使用例

次のコード例は、**My.Application.CommandLineArgs** プロパティを使用して、アプリケーションのコマンド ライン引数を確認します。`/input=` で始まる引数がある場合は、その引数の後続部分を表示します。

VB

```
Private Sub ParseCommandLineArgs()
    Dim inputArgument As String = "/input="
    Dim inputName As String = ""

    For Each s As String In My.Application.CommandLineArgs
        If s.ToLower.StartsWith(inputArgument) Then
            inputName = s.Remove(0, inputArgument.Length)
        End If
    Next

    If inputName = "" Then
        MsgBox("No input name")
    Else
        MsgBox("Input name: " & inputName)
    End If
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase (基本クラス ConsoleApplicationBase は、コンソール アプリケーションで使用できるメンバを提供し、その基本クラス ApplicationBase は、すべてのプロジェクトで使用できるメンバを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照

関連項目

My.Application オブジェクトのメンバ
My.Application.ApplicationContext プロパティ
My.Application.Info オブジェクト
My.Application.Log オブジェクト
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase
Microsoft.VisualBasic.ApplicationServices.ApplicationBase

My.Application オブジェクトのメンバ

My.Application オブジェクトは、現在のアプリケーションに関連するプロパティ、メソッド、およびイベントを提供します。

プロパティ

プロパティ	説明
ApplicationContext	Windows フォーム アプリケーションの現在のスレッドの ApplicationContext オブジェクトを取得します。 このプロパティは、Windows フォーム アプリケーションだけで使用できます。
CommandLineArgs	現在のアプリケーションのコマンド ライン引数のコレクションを文字列として取得します。
Culture	現在のスレッドで文字列操作や文字列の書式設定に使用されるカルチャを取得します。
Deployment	現在のアプリケーションの ClickOnce 配置オブジェクトを取得します。このオブジェクトには、現在の配置をプログラムによって更新したり、ファイルをオン デマンドでダウンロードしたりする機能があります。
Info	My.Application.Info オブジェクト を返します。このオブジェクトには、バージョン番号、説明など、アプリケーションのアセンブリに関する情報を取得するためのプロパティがあります。
IsNetworkDeployed	アプリケーションがネットワーク経由で配置されるかどうかを示す **Boolean** を取得します。
Log	My.Application.Log オブジェクト オブジェクトを返します。このオブジェクトには、イベントおよび例外の情報をアプリケーションのイベント ログ リスナに書き込むためのプロパティおよびメソッドがあります。
MinimumSplashScreenDisplayTime	スプラッシュ スクリーンを表示するミリ秒単位の最小時間を取得または設定します。
OpenForms	アプリケーションで開いているすべてのフォームのコレクションを取得します。 このプロパティは、Windows フォーム アプリケーションだけで使用できます。
SaveMySettingsOnExit	アプリケーションの終了時にユーザー設定を保存するかどうかを指定します。 このプロパティは、Windows フォーム アプリケーションおよびコンソール アプリケーションだけで使用できます。
SplashScreen	このアプリケーションのスプラッシュ スクリーンを取得または設定します。 このプロパティは、Windows フォーム アプリケーションだけで使用できます。
UICulture	現在のスレッドでカルチャ固有リソースの取得に使用されるカルチャを取得します。

メソッド

メソッド	説明
ChangeCulture	現在のスレッドで文字列操作や文字列の書式設定に使用されるカルチャを変更します。
ChangeUICulture	現在のスレッドでカルチャ固有リソースの取得に使用されるカルチャを変更します。
DoEvents	現在メッセージ キューにあるすべての Windows メッセージを処理します。 このメソッドは、Windows フォーム アプリケーションだけで使用できます。
GetEnvironmentVariable	指定された環境変数の値を返します。

Run	Visual Basic アプリケーション モデルをセットアップし、開始します。 このメソッドは、Windows フォーム アプリケーションだけで使用できます。

イベント

イベント	説明
NetworkAvailabilityChanged	ネットワークが使用可能になった、または使用不可になったときに発生します。 このイベントは、Windows フォーム アプリケーションだけで使用できます。
Shutdown	アプリケーションのシャットダウン時に発生します。 このイベントは、Windows フォーム アプリケーションだけで使用できます。
Startup	アプリケーションの起動時に発生します。 このイベントは、Windows フォーム アプリケーションだけで使用できます。
StartupNextInstance	単一インスタンス アプリケーションを起動したときに既にアプリケーションがアクティブだった場合に発生します。 このイベントは、Windows フォーム アプリケーションだけで使用できます。
UnhandledException	アプリケーションで処理されない例外が起きたときに発生します。 このイベントは、Windows フォーム アプリケーションだけで使用できます。

参照

関連項目

My.Application オブジェクト

My.Application.ApplicationContext プロパティ

Windows フォーム アプリケーションの現在のスレッドに関する ApplicationContext オブジェクトを取得します。

```
' Usage
Dim value As System.Windows.Forms.ApplicationContext = My.Application.ApplicationContext
' Declaration
Public ReadOnly Property ApplicationContext As System.Windows.Forms.ApplicationContext
```

戻り値

このプロパティは、現在のスレッドに関する **ApplicationContext** オブジェクトを返します。このオブジェクトには、スレッドに関する文脈情報が含まれます。

解説

これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Application オブジェクト
System.Windows.Forms.ApplicationContext
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.ApplicationContext

My.Application.ChangeCulture メソッド

現在のスレッドが、文字列の操作と文字列の書式設定のため使用しているカルチャを変更します。

```
' Usage
My.Application.ChangeCulture(cultureName)
' Declaration
Public Sub ChangeCulture( _
   ByVal cultureName As String _
)
```

パラメータ

cultureName

カルチャの名前を指定する **String** です。有効な名前の一覧については、CultureInfo を参照してください。

例外

例外を引き起こす可能性のある状態を次に示します。

- *cultureName* 引数が **Nothing** です (ArgumentNullException)。
- *cultureName* 引数が、有効なカルチャ名ではありません (ArgumentException)。

解説

My.Application.ChangeCulture メソッドは現在のスレッドの CurrentCulture プロパティを変更します。**CurrentCulture** は、そのスレッドのすべての処理における既定の日付、時刻、通貨、および数の書式、テキストのソート順、文字列の比較、大文字小文字の区別を決定します。

現在のカルチャを取得するには、My.Application.Culture プロパティ プロパティか **CurrentCulture** プロパティを使用します。

CurrentCulture の設定は、言語の設定とは違います。このプロパティには、地理的地域の標準設定に関連するデータだけが含まれています。したがって、**CurrentCulture** プロパティは、特定のカルチャまたは InvariantCulture にだけ設定できます。

現在のスレッドがカルチャ固有のリソースを取得するために使用しているカルチャを変更するには、**My.Application.ChangeUICulture** メソッドを使用します。

セキュリティに関するメモ：

My.Application.ChangeCulture メソッドを使用するには、ControlThread が設定された SecurityPermission が必要です。スレッドに関連付けられたセキュリティの状態によっては、スレッドを操作することは危険です。したがって、このアクセス許可は必要な場合にだけ、信頼できるコードに対して付与してください。信頼度の低いコードではスレッドのカルチャを変更できません。

使用例

次の例に、カルチャを変更することによって、日付の文字列表現がどのように変化するかを示します。

VB

```
Private Sub TestChangeCulture()
    ' Store the current culture.
    Dim currentculture As String = My.Application.Culture.Name
    MsgBox("Current culture is " & currentculture)

    Dim jan1 As New Date(2005, 1, 1, 15, 15, 15)

    My.Application.ChangeCulture("en-US")
    MsgBox("Date represented in en-US culture: " & jan1)
    ' 1/1/2005 3:15:15 PM

    My.Application.ChangeCulture("")
    MsgBox("Date represented in invariant culture" & jan1)
    ' 01/01/2005 15:15:15

    ' Restore the culture.
```

```
        My.Application.ChangeCulture(currentculture)
    End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase、ApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
SecurityPermission	コードに適用されたセキュリティ アクセス許可のセットを記述します。関連する列挙値 : **ControlThread**。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application オブジェクト
My.Application.Culture プロパティ
My.Application.ChangeUICulture メソッド
Microsoft.VisualBasic.ApplicationServices.ApplicationBase.ChangeCulture(System.String)
CurrentCulture

My.Application.ChangeUICulture メソッド

現在のスレッドがカルチャ固有のリソースを取得するために使用するカルチャを変更します。

```
' Usage
My.Application.ChangeUICulture(cultureName)
' Declaration
Public Sub ChangeUICulture( _
   ByVal cultureName As String _
)
```

パラメータ

cultureName

String。カルチャの名前を文字列で指定します。有効な名前の一覧については、CultureInfo の解説を参照してください。

例外

例外を引き起こす可能性のある状態を次に示します。

- *cultureName* が **Nothing** である (ArgumentNullException)。
- *cultureName* 引数が有効なカルチャ名でない (ArgumentException)。

解説

My.Application.ChangeUICulture メソッドは、現在のスレッドの CurrentUICulture プロパティを変更します。**CurrentUICulture** プロパティは、リソース マネージャと **My.Resources** オブジェクトで使用されるカルチャを決定します。この情報に基づいて、実行時にカルチャ固有のリソースが検索されます。

現在の UI カルチャを取得するには、My.Application.UICulture プロパティまたは **CurrentUICulture** プロパティを使用します。

現在のスレッドが文字列操作と文字列書式設定を行うときに使用するカルチャを変更するには、**My.Application.ChangeCulture** メソッドを使用します。

使用例

この例では、**My.Application.ChangeUICulture** メソッドを使用して、My.Resources オブジェクトがリソースを取得するときに使用するカルチャを設定しています。

VB

```
Sub ShowLocalizedMessage()
    Dim culture As String = My.Application.UICulture.Name
    My.Application.ChangeUICulture("fr-FR")
    MsgBox(My.Resources.Message)
    My.Application.ChangeUICulture(culture)
End Sub
```

この例を実行するためには、アプリケーションのリソース ファイル内に `Message` という文字列が含まれている必要があり、さらに French カルチャ用のリソース ファイル `Resources.fr-FR.resx` を用意する必要があります。詳細については、「方法 : リソースを追加または削除する」を参照してください。

French カルチャ用のリソース ファイルがない場合は、**My.Resource** オブジェクトは既定カルチャのリソース ファイルからリソースを取得します。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase、ApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性

Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic で、ローカライズされたリソースを取得する

関連項目

My.Application オブジェクト

My.Application.ChangeCulture メソッド

My.Resources オブジェクト

Microsoft.VisualBasic.ApplicationServices.ApplicationBase.ChangeUICulture(System.String)

My.Application.Culture プロパティ

現在のスレッドが文字列の操作と書式設定に使用するカルチャを取得します。

```
' Usage
Dim value As System.Globalization.CultureInfo = My.Application.Culture
' Declaration
Public ReadOnly Property Culture As System.Globalization.CultureInfo
```

戻り値

現在のスレッドが文字列の操作と書式設定に使用するカルチャを表す CultureInfo オブジェクトを返します。

解説

My.Application.CurrentCulture プロパティは、現在のスレッドが文字列の操作と書式設定に使用する **CultureInfo** オブジェクトを取得します。このオブジェクトは CurrentCulture プロパティが返すオブジェクトと同じであり、現在のスレッドで文字列に関する処理の多くを制御します。**CurrentCulture** プロパティは日付、時刻、通貨、および数字の既定の書式を判断します。また、文字列のソートと比較の方法、および大文字小文字の指定も判断します。

カルチャを変更するには、My.Application.ChangeCulture メソッド を使用するか、**CurrentCulture** プロパティに別の **CultureInfo** オブジェクトを割り当てます。

CurrentCulture の設定は、言語の設定とは違います。このプロパティには、地理的地域の標準設定に関連するデータだけが含まれています。

現在のスレッドがカルチャ固有のリソースを取得するために使用しているカルチャを取得するには、**My.Application.CurrentUICulture** プロパティを使用します。

使用例

次の例に、日付の文字列表現がカルチャによってどのように変化するかを示します。

VB

```
Private Sub TestChangeCulture()
    ' Store the current culture.
    Dim currentculture As String = My.Application.Culture.Name
    MsgBox("Current culture is " & currentculture)

    Dim jan1 As New Date(2005, 1, 1, 15, 15, 15)

    My.Application.ChangeCulture("en-US")
    MsgBox("Date represented in en-US culture: " & jan1)
    ' 1/1/2005 3:15:15 PM

    My.Application.ChangeCulture("")
    MsgBox("Date represented in invariant culture" & jan1)
    ' 01/01/2005 15:15:15

    ' Restore the culture.
    My.Application.ChangeCulture(currentculture)
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase、ApplicationBase

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可

Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Application オブジェクト
My.Application.ChangeCulture メソッド
My.Application.UICulture プロパティ
System.Globalization.CultureInfo
CurrentCulture
Microsoft.VisualBasic.ApplicationServices.ApplicationBase.Culture

My.Application.CommandLineArgs プロパティ

現在のアプリケーションのコマンド ライン引数を文字列のコレクションとして取得します。

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Application.
CommandLineArgs
' Declaration
Public ReadOnly Property CommandLineArgs As System.Collections.ObjectModel.ReadOnlyCollecti
on(Of String)
```

戻り値

現在のアプリケーションのコマンド ライン引数が文字列のコレクションとして格納される **String** の ReadOnlyCollection です。

解説

My.Application.CommandLineArgs プロパティは、ClickOnce を使用しないアプリケーションのために、現在のアプリケーションのコマンド ライン引数に読み取り専用でアクセスできるようにします。

単一インスタンス アプリケーションの場合、**My.Application.CommandLineArgs** プロパティは、アプリケーションの最初のインスタンスのコマンド ライン引数を返します。それ以降に単一インスタンス アプリケーションを起動しようとして使用された引数にアクセスするには、My.Application.StartupNextInstance イベント を処理し、StartupEventArgs 引数の CommandLine プロパティを調べる必要があります。

メモ：

My.Application.CommandLineArgs プロパティは、コマンド ライン引数だけを返します。この動作は、引数とアプリケーション名を返す CommandLine プロパティとは異なります。

メモ：

ClickOnce を使用するアプリケーションでは、**My.Application.Deployment** オブジェクトの ActivationUri プロパティを使ってコマンド ライン引数を取得します。詳細については、「My.Application.Deployment プロパティ」を参照してください。

処理手順

My.Application.CommandLineArgs プロパティに関連するタスクの例を次の表に示します。

タスク	参照項目
アプリケーションの起動時に `/batch` という文字列が引数として指定されたかどうかを確認する方法を説明します。	方法 : Windows フォーム アプリケーションのバッチ モードを有効にする
単一インスタンス アプリケーションの 2 回目以降の起動に使用されたコマンド ライン引数をチェックする方法を説明します。	My.Application.StartupNextInstance イベント

使用例

次のコード例は、**My.Application.CommandLineArgs** プロパティを使用して、アプリケーションのコマンド ライン引数を確認します。`/input=` で始まる引数がある場合は、その引数の後続部分を表示します。

VB

```
Private Sub ParseCommandLineArgs()
    Dim inputArgument As String = "/input="
    Dim inputName As String = ""

    For Each s As String In My.Application.CommandLineArgs
        If s.ToLower.StartsWith(inputArgument) Then
            inputName = s.Remove(0, inputArgument.Length)
        End If
    Next
```

```
        If inputName = "" Then
            MsgBox("No input name")
        Else
            MsgBox("Input name: " & inputName)
        End If
    End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : ConsoleApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
EnvironmentPermission	PATH 環境変数へのアクセス許可を制御します。関連する列挙値 : Read。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application オブジェクト
My.Application.StartupNextInstance イベント
ReadOnlyCollection
Microsoft.VisualBasic.ApplicationServices.ConsoleApplicationBase.CommandLineArgs
StartupEventArgs
CommandLine

My.Application.Deployment プロパティ

現在のアプリケーションの ClickOnce 配置オブジェクトを取得します。このオブジェクトは、現在の配置をプログラムから更新する機能と、ファイルのオンデマンド ダウンロードを可能にします。

```
' Usage
Dim value As System.Deployment.Application.ApplicationDeployment = My.Application.Deploymen
t
' Declaration
Public ReadOnly Property Deployment As System.Deployment.Application.ApplicationDeployment
```

戻り値

アプリケーションの ClickOnce 配置に関する ApplicationDeployment オブジェクトを返します。

例外

例外を引き起こす可能性のある状態を次に示します。

- アプリケーションが ClickOnce アプリケーションとして配置されていない (InvalidDeploymentException)。

解説

My.Application.Deployment プロパティは、アプリケーションの ClickOnce **ApplicationDeployment** オブジェクトを返します。**ApplicationDeployment** オブジェクトは、現在の配置をプログラムから更新する機能と、ファイルのオンデマンド ダウンロードを可能にします。ClickOnce アプリケーションとその配置方法の詳細については、「ClickOnce の配置」および「ClickOnce アプリケーションの発行」を参照してください。

My.Application.Deployment プロパティにアクセスする前に、My.Application.IsNetworkDeployed プロパティを調べてください。そうしないと、アプリケーションが ClickOnce を使用して配置されていない場合は、**My.Application.Deployment** プロパティを読み取ろうとしたときに **InvalidDeploymentException** 例外が発生します。ClickOnce アプリケーションの配置の詳細については、「方法 : ClickOnce アプリケーションを発行する」を参照してください。

My.Application.Deployment プロパティの使い方の詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

処理手順

My.Application.Deployment プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
アプリケーションの更新を確認する	方法 : ClickOnce アプリケーションの更新の有無をチェックする
アプリケーションの更新をダウンロードする	方法 : ClickOnce アプリケーションの更新をダウンロードする

使用例

この例では、アプリケーションがネットワークから配置されたものであることを確認したうえで、更新のダウンロードとインストールを行います。Update メソッドは、アプリケーションが最新の状態であるときはアプリケーションを更新しません。更新を使用するためには、アプリケーションを再起動する必要があります。詳細については、「方法 : ClickOnce アプリケーションの更新をダウンロードする」を参照してください。

VB

```
Sub UpdateApplication()
    If My.Application.IsNetworkDeployed Then
        My.Application.Deployment.Update()
    End If
End Sub
```

My.Application.Deployment オブジェクトを使用して更新できるのは、ClickOnce を使用して配置したアプリケーションだけです。ClickOnce アプリケーションの配置の詳細については、「方法 : ClickOnce アプリケーションを発行する」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : ConsoleApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**使用する**
クラス ライブラリ	使用しない
コンソール アプリケーション	**使用する**
Windows コントロール ライブラリ	使用しない
Web コントロール ライブラリ	使用しない
Windows サービス	**使用する**
Web サイト	使用しない

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : ClickOnce アプリケーションの更新の有無をチェックする
方法 : ClickOnce アプリケーションの更新をダウンロードする
方法 : ClickOnce アプリケーションを発行する

関連項目

My.Application オブジェクト
My.Application.IsNetworkDeployed プロパティ
System.Deployment.Application.ApplicationDeployment
Microsoft.VisualBasic.ApplicationServices.ConsoleApplicationBase.Deployment

概念

Visual Basic アプリケーション モデルの概要

その他の技術情報

ClickOnce アプリケーションの発行

My.Application.DoEvents メソッド

メッセージ キューに現在入っているすべての Windows メッセージを処理します。

```
' Usage
My.Application.DoEvents()
' Declaration
Public Sub DoEvents()
```

解説

アプリケーションに **My.Application.DoEvents** メソッドを使用すると、コードの実行中に発生する可能性のある他のイベントを処理できます。**My.Application.DoEvents** メソッドと DoEvents メソッドの動作は同じです。

Windows フォーム アプリケーションを実行すると、新しいフォームが作成され、イベントが処理されるのを待機し始めます。フォームはボタンのクリックなどのイベントを処理するたびに、そのイベントに関連するすべてのコードを処理します。他のすべてのイベントはキューの中で待機します。コードがイベントを処理する間、アプリケーションは応答しません。たとえば、ウィンドウは別のウィンドウが最前面に移動された場合は再描画されません。

コード内で **My.Application.DoEvents** を呼び出すと、アプリケーションは他のイベントを処理できます。たとえば、ループを使って ListBox にデータを追加し、ループの各ステップの後で **My.Application.DoEvents** を呼び出すと、別のウィンドウがフォームの前面にあるときにフォームを再描画できます。**My.Application.DoEvents** をコードから削除すると、ボタンのクリック イベントのハンドラが実行を終了するまで、フォームは再描画されません。

通常、このメソッドはループ内でメッセージを処理するために使います。

メモ：

My.Application.DoEvents メソッドがイベントを処理する方法は、フォームとすべて同じではありません。Use multithreading to make the form directly handle the events.詳細については、「Visual Basic におけるマルチスレッド」を参照してください。

注意

ユーザー インターフェイス (UI) イベントを処理するメソッドが **My.Application.DoEvents** メソッドを呼び出す場合、このメソッドは終了前にもう一度入られる可能性があります。これは、**My.Application.DoEvents** メソッドが Windows メッセージを処理し、Windows メッセージがイベントを発生させることがあるためです。

処理手順

My.Application.DoEvents メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
フォームがビジー状態のときでも UI 入力に応答できるようにする	チュートリアル : イベントの処理

使用例

次の例は **My.Application.DoEvents** メソッドを使用して、`TextBox1` の UI の更新を可能にします。

VB

```
Private Sub TestDoEvents()
    For i As Integer = 0 To 10000
        TextBox1.Text = i.ToString
        My.Application.DoEvents()
    Next
End Sub
```

このコードは、`Text` プロパティを持つ `TextBox1` コンポーネントが定義されたフォームに記述する必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
UIPermission	ユーザー インターフェイスとクリップボードに関連するアクセス許可を制御します。関連する列挙値 : AllWindows。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application オブジェクト

Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.DoEvents

DoEvents

My.Application.GetEnvironmentVariable メソッド

指定された環境変数の値を返します。

```
' Usage
Dim value As String = My.Application.GetEnvironmentVariable(name)
' Declaration
Public Function GetEnvironmentVariable( _
   ByVal name As String _
) As String
```

パラメータ

name

環境変数の名前を指定する **String** です。

戻り値

名前が *name* である環境変数の値を格納する **String** です。

例外

例外を引き起こす可能性のある状態を次に示します。

- *name* 引数が **Nothing** です (ArgumentNullException)。
- *name* で指定された環境変数が存在しません (ArgumentException)。
- 呼び出し元のコードに **Read** アクセスを持つ EnvironmentPermission がありません (SecurityException)。

解説

My.Application.GetEnvironmentVariable メソッドは、名前が *name* である環境変数を返します。このメソッドは System.Environment.GetEnvironmentVariable(System.String) と似ていますが、*name* で指定された環境変数が存在しない場合に例外を発生させる点が違います。

使用例

次の例は **My.Application.GetEnvironmentVariable** メソッドを使用し、PATH 環境変数の値を (取得できれば) 取得して表示します。取得できない場合は、PATH 環境変数が存在しないことを示すメッセージを表示します。

VB

```
Private Sub TestGetEnvironmentVariable()
    Try
        MsgBox("PATH = " & My.Application.GetEnvironmentVariable("PATH"))
    Catch ex As System.ArgumentException
        MsgBox("Environment variable 'PATH' does not exist.")
    End Try
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase、ApplicationBase

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**

コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Application オブジェクト

GetEnvironmentVariable

GetEnvironmentVariable

My.Application.Info プロパティ

アプリケーションのアセンブリに関する情報 (バージョン番号や説明など) を取得するためのプロパティを提供するオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.ApplicationServices.AssemblyInfo = My.Application.Info
' Declaration
Public ReadOnly Property Info As AssemblyInfo
```

戻り値

このプロパティは、現在のアプリケーションに **My.Application.Info** オブジェクトを返します。

解説

このプロパティによって、**My.Application.Info** に簡単にアクセスできます。詳細については、「My.Application.Info オブジェクト」を参照してください。

使用例

次の例では、**My.Application.Info.Version** プロパティを使用して、アプリケーションのバージョンを表示します。

VB

```
MsgBox("Application version: " & My.Application.Info.Version.ToString)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : ApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Application オブジェクト

My.Application.Info オブジェクト

My.Application.IsNetworkDeployed プロパティ

このアプリケーションがネットワークから ClickOnce を使用して配置されたものかどうかを示す **Boolean** 値を取得します。

```
' Usage
Dim value As Boolean = My.Application.IsNetworkDeployed
' Declaration
Public ReadOnly Property IsNetworkDeployed As Boolean
```

戻り値

このアプリケーションがネットワークから配置されたものかどうかを示す **Boolean** 値。ネットワークから配置されたアプリケーションの場合は **True**、それ以外の場合は **False** が返されます。

解説

My.Application.IsNetworkDeployed プロパティは、このアプリケーションがネットワークから ClickOnce を使用して配置されたものかどうかを示します。ClickOnce アプリケーションの詳細については、「ClickOnce の配置」を参照してください。

My.Application.Deployment プロパティにアクセスする前に、このプロパティが **True** であることを確認してください。そうしないと、アプリケーションがネットワークから ClickOnce を使用して配置されていない場合は、**My.Application.Deployment** プロパティを読み取ろうとしたときに InvalidDeploymentException 例外が発生します。

ClickOnce の設定の詳細については、「ClickOnce アプリケーションの発行」を参照してください。ClickOnce アプリケーションの配置の詳細については、「方法 : ClickOnce アプリケーションを発行する」を参照してください。

My.Application.IsNetworkDeployed プロパティの使い方の詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

処理手順

My.Application.IsNetworkDeployed プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
アプリケーションの更新を確認する	方法 : ClickOnce アプリケーションの更新の有無をチェックする
アプリケーションの更新をダウンロードする	方法 : ClickOnce アプリケーションの更新をダウンロードする

使用例

この例では、アプリケーションがネットワークから配置されたものであることを確認したうえで、更新のダウンロードとインストールを行います。Update メソッドは、アプリケーションが最新の状態であるときはアプリケーションを更新しません。更新を使用するためには、アプリケーションを再起動する必要があります。詳細については、「方法 : ClickOnce アプリケーションの更新をダウンロードする」を参照してください。

VB

```
Sub UpdateApplication()
    If My.Application.IsNetworkDeployed Then
        My.Application.Deployment.Update()
    End If
End Sub
```

My.Application.Deployment オブジェクトを使用して更新できるのは、ClickOnce を使用して配置したアプリケーションだけです。ClickOnce アプリケーションの配置の詳細については、「方法 : ClickOnce アプリケーションを発行する」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : ConsoleApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**使用する**
クラス ライブラリ	使用しない
コンソール アプリケーション	**使用する**
Windows コントロール ライブラリ	使用しない
Web コントロール ライブラリ	使用しない
Windows サービス	**使用する**
Web サイト	使用しない

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : ClickOnce アプリケーションの更新の有無をチェックする
方法 : ClickOnce アプリケーションの更新をダウンロードする
方法 : ClickOnce アプリケーションを発行する

関連項目

My.Application オブジェクト
My.Application.Deployment プロパティ
IsNetworkDeployed

概念

Visual Basic アプリケーション モデルの概要

My.Application.Log プロパティ

イベントと例外の情報を作成するためのプロパティとメソッドを、アプリケーション ログのリスナに提供するオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.Logging.Log = My.Application.Log
' Declaration
Public ReadOnly Property Log As MyLog
```

戻り値

このプロパティは、現在のアプリケーションに **My.Application.Log** オブジェクトを返します。

解説

このプロパティを使用して、**My.Application.Log** オブジェクトに簡単にアクセスできます。詳細については、「My.Application.Log オブジェクト」を参照してください。

使用例

My.Application.Log.WriteEntry メソッドを使用して、トレース情報のログを記録する方法は次の例のようになります。詳細については、「方法 : ログ メッセージを書き込む」を参照してください。

VB

```
Public Sub TracingTest(ByVal fileName As String)
    My.Application.Log.WriteEntry( _
        "Entering TracingTest with argument " & _
        fileName & ".")
    ' Code to trace goes here.
    My.Application.Log.WriteEntry( _
        "Exiting TracingTest with argument " & _
        fileName & ".")
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : ApplicationBase

アセンブリ : Microsoft Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可能
Windows アプリケーション	○
クラス ライブラリ	○
コンソール アプリケーション	○
Windows コントロール ライブラリ	○
Web コントロール ライブラリ	×
Windows サービス	○
Web サイト	×

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application オブジェクト

My.Application.Log オブジェクト

My.Application.MinimumSplashScreenDisplayTime プロパティ

スプラッシュ スクリーン を表示する最短時間 (ミリ秒) を決定します。

```
' Usage
Dim value As Integer = My.Application.MinimumSplashScreenDisplayTime
' Declaration
Public Property MinimumSplashScreenDisplayTime As Integer
```

戻り値

スプラッシュ スクリーン を表示する最短時間 (ミリ秒) を格納する **Integer** です。

解説

My.Application.MinimumSplashScreenDisplayTime プロパティを使用すると、アプリケーションのスプラッシュ スクリーンを表示する最短時間を指定できます。このプロパティに指定した時間よりも先にメイン フォームの初期化が終了した場合は、指定の時間が経過するまでスプラッシュ スクリーンが画面に残り、その後でメイン フォームが表示されます。アプリケーションの起動にそれよりも長い時間がかかった場合は、メイン フォームがアクティブになった時点でスプラッシュ スクリーンが閉じます。

プロジェクト デザイナを使ってアプリケーションにスプラッシュ スクリーンを追加すると、**My.Application.MinimumSplashScreenDisplayTime** プロパティに 2000 が設定され、最短で 2 秒間表示されます。

このプロパティは、Visual Basic アプリケーション モデルをサポートします。詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

My.Application.MinimumSplashScreenDisplayTime プロパティは、OnInitialize メソッドまたは OnCreateSplashScreen メソッドをオーバーライドするメソッドで設定する必要があります。WindowsFormsApplicationBase クラスのメソッドをオーバーライドするコードは、ApplicationEvents.vb ファイルに入力する必要があります。このファイルは既定で隠しファイルになっています。

オーバーライドしているメンバのコード エディタ ウィンドウを表示するには、次の操作を行います。

1. ソリューション エクスプローラでプロジェクトを選択した状態で、[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [アプリケーション イベントの表示] をクリックしてコード エディタを開きます。

   詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

使用例

OnInitialize プロパティをオーバーライドすることによって **My.Application.MinimumSplashScreenDisplayTime** プロパティを設定する方法は、次の例のようになります。

VB

```
Protected Overrides Function OnInitialize( _
    ByVal commandLineArgs As _
    System.Collections.ObjectModel.ReadOnlyCollection(Of String) _
) As Boolean
    ' Set the display time to 5000 milliseconds (5 seconds).
    Me.MinimumSplashScreenDisplayTime = 5000
    Return MyBase.OnInitialize(commandLineArgs)
End Function
```

この例は、スプラッシュ スクリーンがプロジェクトにあることを前提としています。

アプリケーション イベントのコードを、コード エディタ ウィンドウに入力する必要があります。詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

アクセス許可

アクセス許可は不要です。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : **WindowsFormsApplicationBase**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	なし
コンソール アプリケーション	なし
Windows コントロール ライブラリ	なし
Web コントロール ライブラリ	なし
Windows サービス	なし
Web サイト	なし

参照

関連項目

My.Application オブジェクト
My.Application.SplashScreen プロパティ
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.MinimumSplashScreenDisplayTime
OnInitialize
OnCreateSplashScreen
WindowsFormsApplicationBase

概念

Visual Basic アプリケーション モデルの拡張

My.Application.NetworkAvailabilityChanged イベント

ネットワークの可用性に変化があったときに発生します。

```
' Usage
Public Sub Me_NetworkAvailabilityChanged( _
   ByVal sender As Object, _
   ByVal e As NetworkAvailableEventArgs _
) Handles Me.NetworkAvailabilityChanged
End Sub
' Declaration
Public Event NetworkAvailabilityChanged( _
   ByVal sender As Object, _
   ByVal e As NetworkAvailableEventArgs _
)
```

パラメータ

sender

必ず指定します。イベントの発生元の **Object**。

e

必ず指定します。ネットワークの可用性に関する情報が格納される NetworkAvailableEventArgs オブジェクトを指定します。

解説

アプリケーションでは、ネットワークの可用性が変化するたびに **NetworkAvailabilityChanged** イベントが生成されます。*e* パラメータの IsNetworkAvailable プロパティを使って、ネットワーク接続の新しい状態を取得できます。ネットワーク接続の現在の状態を取得するには、My.Computer.Network.IsAvailable プロパティ を使用します。

このイベントは、アプリケーションのメイン スレッドで他のユーザー インターフェイス イベントと一緒に発生します。このため、イベント ハンドラはアプリケーションの UI に直接アクセスできます。ただし、このイベントが発生したとき、アプリケーションが別のユーザー インターフェイス イベントの処理でビジーになっている場合は、そのイベント ハンドラが終了するか、または My.Application.DoEvents メソッド を呼び出すまで、このイベントを処理できません。

My.Computer.Network.NetworkAvailabilityChanged イベント はこのイベントと同じ機能を持ちますが、すべての種類のアプリケーションで使用できます。

NetworkAvailabilityChanged イベント ハンドラ用のコードは、ApplicationEvents.vb ファイルに格納されています。このファイルは既定では非表示です。

アプリケーション イベントのコード エディタ ウィンドウにアクセスするには、次の操作を行います。

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [アプリケーション イベントの表示] をクリックしてコード エディタを開きます。

   詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

メモ :

多くのネットワーク ハブが、広域ネットワークからハブが切断された場合でもネットワーク接続を提供します。そのため、回線がつながっている場合、このイベントはコンピュータとハブの間で接続に変化があったことを示します。

メモ :

NetworkAvailabilityChanged イベントは、Windows 95 や Windows 98 で動作しているアプリケーションや、Windows 2000 で管理者以外によって実行されたアプリケーションでは発生しません。このようなプラットフォームでアプリケーションを実行する可能性がある場合は、My.Computer.Network.IsAvailable プロパティ を使用してネットワークが使用可能かどうかを調べてください。

使用例

次の例は、ネットワークの可用性が変化したときに、既定の `Form1` クラスの `SetConnectionStatus` メソッドを呼び出します。

アプリケーション イベントのコードを、コード エディタ ウィンドウに入力する必要があります。このウィンドウを利用するには、このトピックの「解説」にある説明に従ってください。

VB

```
Private Sub MyApplication_NetworkAvailabilityChanged( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.Devices.NetworkAvailableEventArgs _
) Handles Me.NetworkAvailabilityChanged
    My.Forms.Form1.SetConnectionStatus(e.IsNetworkAvailable)
End Sub
```

コードのコンパイル

プロジェクトに `Form1` という名前のフォームが必要です。そのフォームは、**Boolean** 型のパラメータを受け取る `SetConnectionStatus` という名前のメソッドを含む必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Application オブジェクト
My.Forms オブジェクト
My.Computer.Network.NetworkAvailabilityChanged イベント
Microsoft.VisualBasic.Devices.NetworkAvailableEventArgs

My.Application.OpenForms プロパティ

アプリケーションの現在開かれているすべてのフォームのコレクションを取得します。

```
' Usage
Dim value As System.Windows.Forms.FormCollection = My.Application.OpenForms
' Declaration
Public ReadOnly Property OpenForms As System.Windows.Forms.FormCollection
```

戻り値

アプリケーションの現在開かれているすべてのフォームを含む FormCollection オブジェクト

解説

My.Application.OpenForms プロパティは、アプリケーションの現在開かれているすべてのフォームのコレクションを取得します。このプロパティの動作は、System.Windows.Forms.Application.OpenForms プロパティと同じです。

メモ：

My.Application.OpenForms プロパティは、どのスレッドによって開かれたかに関係なく、現在開かれているすべてのフォームを返します。フォームにアクセスするときには、事前に各フォームの InvokeRequired プロパティを調べる必要があります。そうしないと、InvalidOperationException 例外がスローされる可能性があります。詳細については、「方法 : アプリケーションで開いているすべてのフォームにアクセスする」を参照してください。

処理手順

My.Application.OpenForms プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
アプリケーションの現在開かれているすべてのフォームのタイトルを表示する	方法 : アプリケーションで開いているすべてのフォームにアクセスする

使用例

この例では、アプリケーションの現在開かれているフォームをループ処理し、現在のスレッドから直接アクセス可能なものを選択し、そのフォームのタイトルを ListBox コントロールに表示します。開かれているフォームにアクセスする方法については、「方法 : アプリケーションで開いているすべてのフォームにアクセスする」を参照してください。

VB

```
Private Sub GetOpenFormTitles()
    Dim formTitles As New Collection

    Try
        For Each f As Form In My.Application.OpenForms
            If Not f.InvokeRequired Then
                ' Can access the form directly.
                formTitles.Add(f.Text)
            End If
        Next
    Catch ex As Exception
        formTitles.Add("Error: " & ex.Message)
    End Try

    Form1.ListBox1.DataSource = formTitles
End Sub
```

この例を実行するには、Windows フォーム アプリケーション内に、`ListBox1` という名前のリスト ボックスを含んだ `Form1` というフォームを用意する必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
UIPermission	ユーザー インターフェイスとクリップボードに関係するアクセス許可を制御します。関連する列挙値 : AllWindows。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application オブジェクト
System.Windows.Forms.FormCollection
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.OpenForms
System.Windows.Forms.Application.OpenForms

My.Application.Run メソッド

Visual Basic アプリケーション モデルを設定および開始します。

```
' Usage
My.Application.Run(commandLine)
' Declaration
Public Sub Run( _
   ByVal commandLine As String() _
)
```

パラメータ

commandLine

String 型の配列です。**Sub Main** からのコマンド ラインです。

解説

My.Application.Run メソッドは、Visual Basic アプリケーション モデルをサポートします。このメソッドをアプリケーションの **Sub Main** 以外から呼び出すことはできません。既定では、Windows フォーム アプリケーションの **Sub Main** がこのメソッドを呼び出します。詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
EnvironmentPermission	すべての環境変数へのアクセスを許可するかどうかを制御します。関連する列挙値 : Unrestricted。
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : **Unrestricted**。
RegistryPermission	レジストリ変数へのアクセスを許可するかどうかを制御します。関連する列挙値 : **Unrestricted**。
UIPermission	ユーザー インターフェイスとクリップボードに関連するアクセス許可を制御します。関連する列挙値 : AllWindows。

WebPermission	HTTP インターネット リソースにアクセスするための権限を制御します。関連する列挙値 : **Unrestricted**。
SocketPermission	トランスポート アドレスでの接続の確立と承認のための権限を制御します。関連する列挙値 : **Unrestricted**。
PerformanceCounterPermission	Windows NT パフォーマンス カウンタ コンポーネントへのアクセスを制御します。関連する列挙値 : **Unrestricted**。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application オブジェクト

Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Run(System.String[])

概念

Visual Basic アプリケーション モデルの概要

My.Application.SaveMySettingsOnExit プロパティ

アプリケーションが、終了時にユーザー設定を保存するかどうかを決定します。

```
' Usage
Dim value As Boolean = My.Application.SaveMySettingsOnExit
' Declaration
Public Property SaveMySettingsOnExit As Boolean
```

戻り値

Boolean 型。この値が **True** であれば、アプリケーションは終了時にユーザー設定を保存します。そうでない場合、設定は暗黙的に保存されません。

解説

SaveMySettingsOnExit プロパティを使用すると、アプリケーションが実行時に設定を保存する方法を変更できます。**My.Settings** オブジェクトの Save メソッドを使用すると、設定の変更を明示的に保存できます。設定の変更および保存の詳細については、「My.Settings オブジェクト」を参照してください。

このプロパティへの変更は、アプリケーションの終了時に破棄されます。**SaveMySettingsOnExit** プロパティを完全に変更するには、プロジェクト デザイナで設定を変更する必要があります。

プロジェクト デザイナで設定を変更するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [シャットダウン時に My.Settings を保存する] を選択します。

詳細については、「アプリケーション プロパティの管理」を参照してください。

使用例

次の例は 2 つの部分に分かれています。

- `InitializeSaveMySettingsOnExit` サブルーチンは、オンの状態の CheckBox コントロールを、**SaveMySettingsOnExit** プロパティの現在の値に初期化します。
- `SaveMySettingsOnExit_CheckedChanged` サブルーチンは **CheckBox** コントロールの変更を処理して、**My.Application.SaveMySettingsOnExit** プロパティを更新します。

VB

```
Private Sub InitializeSaveMySettingsOnExit()
    SaveMySettingsOnExit.Checked = _
        My.Application.SaveMySettingsOnExit
End Sub
Private Sub SaveMySettingsOnExit_CheckedChanged( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs _
) Handles SaveMySettingsOnExit.CheckedChanged
    My.Application.SaveMySettingsOnExit = _
        SaveMySettingsOnExit.Checked
End Sub
```

この例を実行するには、アプリケーションに `SaveMySettingsOnExit` という名前の **CheckBox** コントロールがあることが必要です。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Application オブジェクト
My.Settings オブジェクト
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.SaveMySettingsOnExit
Save

My.Application.Shutdown イベント

アプリケーションが終了したときに発生します。

```
' Usage
Public Sub Me_Shutdown( _
   ByVal sender As Object, _
   ByVal e As System.EventArgs _
) Handles Me.Shutdown
End Sub
' Declaration
Public Event Shutdown( _
   ByVal sender As Object, _
   ByVal e As System.EventArgs _
)
```

パラメータ

sender

イベントの発生元の **Object**。

e

Empty を格納している EventArgs オブジェクト。

解説

アプリケーションは終了する前に **Shutdown** イベントを発生させます。これを使用して、アプリケーションがリソースを閉じる方法を制御できます。これは Visual Basic アプリケーション モデルに含まれるイベントです。詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

Shutdown イベント ハンドラ用のコードは、ApplicationEvents.vb ファイルに格納されています。このファイルは既定では非表示です。

アプリケーション イベントのコード エディタ ウィンドウにアクセスするには、次の操作を行います。

1. ソリューション エクスプローラにプロジェクトが選択されている状態で、[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [アプリケーション イベントの表示] をクリックしてコード エディタを開きます。

   詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

処理手順

My.Application.Shutdown イベントに関連するタスクの例を次の表に示します。

目的	参照項目
Visual Basic アプリケーション モデルによって提供されるイベントを使用してコードを実行する	方法 : アプリケーションの起動時または終了時にコードを実行する
アプリケーションが終了するときにメッセージをログに記録する	方法 : アプリケーションの起動時または終了時にメッセージをログに記録する

使用例

次の例は、アプリケーションが終了するときにメッセージをログに記録します。

VB

```
Private Sub MyApplication_Shutdown( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs _
) Handles Me.Shutdown
    My.Application.Log.WriteEntry("Application Shut Down.")
End Sub
```

アプリケーション イベントのコードを、コード エディタ ウィンドウに入力する必要があります。このウィンドウを利用するには、このトピックの「解説」にある説明に従ってください。詳細については、「方法 : アプリケーションの起動時または終了時にメッセージをログに記録する」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : アプリケーションの起動時または終了時にメッセージをログに記録する

方法 : アプリケーション イベントを処理する (Visual Basic)

関連項目

My.Application オブジェクト

System.EventArgs

Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Shutdown

概念

Visual Basic アプリケーション モデルの概要

My.Application.SplashScreen プロパティ

このアプリケーションのスプラッシュ スクリーンを取得または設定します。

```
' Usage
Dim value As System.Windows.Forms.Form = My.Application.SplashScreen
' Declaration
Public Property SplashScreen As System.Windows.Forms.Form
```

戻り値

アプリケーションがスプラッシュ スクリーンとして使用する Form オブジェクト。

例外

例外を引き起こす可能性のある状態を次に示します。

- このプロパティと **My.Application.MainForm** プロパティに同じ値が割り当てられている場合 (ArgumentNullException)。

解説

My.Application.SplashScreen プロパティを使用すると、アプリケーションがスプラッシュ スクリーン (アプリケーションが起動する間に表示される最初のグラフィック フォーム) として使用する **Form** オブジェクトを取得または設定できます。

このプロパティは、Visual Basic アプリケーション モデルをサポートします。詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

このプロパティへの変更は、アプリケーションが閉じたときに破棄されます。スプラッシュ スクリーンを完全に変更するには、プロジェクト デザイナで設定を変更する必要があります。詳細については、「方法 : アプリケーションのスプラッシュ スクリーンを指定する (Visual Basic)」を参照してください。

使用例

次の例は **My.Application.SplashScreen** プロパティと **My.Application.Startup** イベントを使用して、スプラッシュ スクリーンをアプリケーション起動時のステータス情報で更新します。

VB

```
Private Sub MyApplication_Startup( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs _
) Handles Me.Startup
    ' Get the splash screen.
    Dim splash As SplashScreen1 = CType(My.Application.SplashScreen, SplashScreen1)
    ' Display current status information.
    splash.Status = "Current user: " & My.User.Name
End Sub
```

この例では、プロジェクトに `SplashScreen1` という名前のスプラッシュ スクリーンが設定されている必要があります。スプラッシュ スクリーンには、そのユーザー インターフェイスを更新する `Status` という名前のプロパティが必要です。

アプリケーション イベントのコードを、コード エディタ ウィンドウに入力する必要があります。詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可

Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Application オブジェクト

My.Application.MinimumSplashScreenDisplayTime プロパティ

System.Windows.Forms.Form

Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.SplashScreen

My.Application.Startup イベント

アプリケーションの起動時に発生します。

```
' Usage
Public Sub Me_Startup( _
   ByVal sender As Object, _
   ByVal e As StartupEventArgs _
) Handles Me.Startup
End Sub
' Declaration
Public Event Startup( _
   ByVal sender As Object, _
   ByVal e As StartupEventArgs _
)
```

パラメータ

sender

イベントの発生元の **Object**。

e

アプリケーションのコマンド ライン引数を含む StartupEventArgs オブジェクト。

解説

通常の (単一インスタンスではない) アプリケーションは、起動のたびに **Startup** イベントを発生させます。単一インスタンス アプリケーションは、既にアクティブでない場合にのみ起動時に **Startup** イベントを発生させます。既にアクティブである場合は、**StartupNextInstance** イベントを発生させます。詳細については、「My.Application.StartupNextInstance イベント」および「方法 : アプリケーションのインスタンス化の動作を指定する」を参照してください。

このイベントは、Visual Basic アプリケーション モデルの一部です。詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

e パラメータの Cancel プロパティを使って、アプリケーションのスタートアップ フォームの読み込みを制御できます。**Cancel** プロパティが **True** に設定されている場合、スタートアップ フォームは起動しません。この場合は、コードで別のスタートアップ コードのパスを呼び出す必要があります。たとえば、「方法 : Windows フォーム アプリケーションのバッチ モードを有効にする」を参照してください。

e パラメータの CommandLine プロパティまたは My.Application.CommandLineArgs プロパティ を使って、アプリケーションのコマンド ライン引数にアクセスできます。

Startup イベント ハンドラのコードは、ApplicationEvents.vb ファイルに格納されています。このファイルは既定では非表示です。

アプリケーション イベントのコード エディタ ウィンドウにアクセスするには、次の操作を行います。

1. ソリューション エクスプローラでプロジェクトが選択されている状態で、[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [アプリケーション イベントの表示] をクリックしてコード エディタを開きます。

詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

処理手順

My.Application.Startup イベントに関連するタスクの例を次の表に示します。

タスク	参照項目
Visual Basic アプリケーション モデルに用意されたイベントを使ってコードを実行する。	方法 : アプリケーションの起動時または終了時にコードを実行する
アプリケーションの起動時に `/batch` という文字列が引数として指定されたかどうかを確認する。	方法 : Windows フォーム アプリケーションのバッチ モードを有効にする

使用例

この例では、**My.Application.SplashScreen** プロパティと **My.Application.Startup** イベントを使って、アプリケーション起動時にスプラッシュ スクリーンのステータス情報を更新します。

VB

```
Private Sub MyApplication_Startup( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs _
) Handles Me.Startup
    ' Get the splash screen.
    Dim splash As SplashScreen1 = CType(My.Application.SplashScreen, SplashScreen1)
    ' Display current status information.
    splash.Status = "Current user: " & My.User.Name
End Sub
```

この例は、`SplashScreen1` というスプラッシュ スクリーンがプロジェクトにあることを前提としています。スプラッシュ スクリーンには、ユーザー インターフェイスを更新する `Status` というプロパティが必要です。

このコードは、アプリケーション イベントのコード エディタ ウィンドウに入力します。このウィンドウにアクセスするには、このトピックの「解説」に記載されている手順に従います。詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
SecurityPermission	このイベントのイベント ハンドラを追加できるかどうかを制御します。関連する列挙値 : System.Security.Permissions.SecurityPermissionFlag.ControlAppDomain。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Windows フォーム アプリケーションのバッチ モードを有効にする
方法 : アプリケーション イベントを処理する (Visual Basic)
方法 : アプリケーションのインスタンス化の動作を指定する

関連項目

My.Application オブジェクト
My.Application.StartupNextInstance イベント
My.Application.CommandLineArgs プロパティ
Microsoft.VisualBasic.ApplicationServices.StartupEventArgs
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.Startup

概念

Visual Basic アプリケーション モデルの概要

My.Application.StartupNextInstance イベント

単一インスタンス アプリケーションを起動しようとしたときに、既にそのアプリケーションがアクティブである場合に発生します。

```
' Usage
Public Sub Me_StartupNextInstance( _
   ByVal sender As Object, _
   ByVal e As StartupNextInstanceEventArgs _
) Handles Me.StartupNextInstance
End Sub
' Declaration
Public Event StartupNextInstance( _
   ByVal sender As Object, _
   ByVal e As StartupNextInstanceEventArgs _
)
```

パラメータ

sender

イベントの発生元の **Object**。

e

アプリケーションのコマンド ライン引数を含んでいる StartupEventArgs オブジェクト。

解説

単一インスタンス アプリケーションは、そのアプリケーションが既にアクティブであるときに再起動しようとすると、StartupNextInstance イベントを発生させます。単一インスタンス アプリケーションを最初に起動したときには、Startup イベントが発生します。詳細については、「My.Application.Startup イベント」および「方法 : アプリケーションのインスタンス化の動作を指定する」を参照してください。

このイベントは Visual Basic アプリケーション モデルの一部です。詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

このイベントは、他のユーザー インターフェイス イベントと一緒にアプリケーションのメイン スレッド上で発生します。そのため、イベント ハンドラはアプリケーションのユーザー インターフェイスに直接アクセスできます。ただし、アプリケーションが他のユーザー インターフェイス イベントを処理している最中にこのイベントが発生した場合は、他のイベント ハンドラが完了するまで、または My.Application.DoEvents メソッドが呼び出されるまでは、このイベントは処理されません。

メモ :

StartupNextInstance イベントは単一インスタンス アプリケーションでのみ発生します。アプリケーションで単一インスタンスの動作を有効にするには、プロジェクト デザイナの [単一インスタンスのアプリケーションを作成する] チェック ボックスをオンにします。詳細については、「方法 : アプリケーションのインスタンス化の動作を指定する」を参照してください。

これ以降、単一インスタンス アプリケーションの起動時の引数にアクセスするには、*e* パラメータの CommandLine プロパティを使用する必要があります。My.Application.CommandLineArgs プロパティを使用すると、単一インスタンス アプリケーションの最初のインスタンスを起動するときに使用する引数にアクセスできます。

StartupNextInstance イベント ハンドラのコードは ApplicationEvents.vb ファイルに格納されます。このファイルは既定では非表示です。

アプリケーション イベントのコード エディタ ウィンドウを表示するには、次の操作を行います。

1. ソリューション エクスプローラでプロジェクトを選択した状態で、[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [アプリケーション イベントの表示] をクリックしてコード エディタを開きます。

   詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

処理手順

My.Application.StartupNextInstance イベントに関連するタスクの例を次の表に示します。

目的	参照項目
Visual Basic アプリケーション モデルに用意されているイベントを使用してコードを実行する	方法 : アプリケーションの起動時または終了時にコードを実行する
最初のアプリケーション インスタンスのコマンド ライン引数を調べる	My.Application.CommandLineArgs プロパティ

使用例

この例では、**StartupNextInstance** イベント ハンドラの *e* パラメータを使用して、アプリケーションのコマンド ライン引数を調べます。`/input=` で始まる引数が見つかった場合は、その引数の残りの部分を表示します。

VB

```
Private Sub MyApplication_StartupNextInstance( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.StartupNextInstanceEventArgs _
) Handles Me.StartupNextInstance
    Dim inputArgument As String = "/input="
    Dim inputName As String = ""

    For Each s As String In e.CommandLine
        If s.ToLower.StartsWith(inputArgument) Then
            inputName = s.Remove(0, inputArgument.Length)
        End If
    Next

    If inputName = "" Then
        MsgBox("No input name")
    Else
        MsgBox("Input name: " & inputName)
    End If
End Sub
```

このコードはアプリケーション イベントのコード エディタ ウィンドウ内に入力する必要があります。このウィンドウにアクセスするには、「解説」で説明した手順に従ってください。詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可能
Windows アプリケーション	○
クラス ライブラリ	X
コンソール アプリケーション	X
Windows コントロール ライブラリ	X
Web コントロール ライブラリ	X
Windows サービス	X
Web サイト	X

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : アプリケーション イベントを処理する (Visual Basic)
方法 : アプリケーションのインスタンス化の動作を指定する
方法 : アプリケーションのインスタンス化の動作を指定する

関連項目

My.Application オブジェクト
My.Application.Startup イベント
My.Application.CommandLineArgs プロパティ
StartupEventArgs
CommandLine
Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase.StartupNextInstance

概念

Visual Basic アプリケーション モデルの概要

My.Application.UICulture プロパティ

現在のスレッドがカルチャ固有のリソースを取得するために使用するカルチャを取得します。

```
' Usage
Dim value As System.Globalization.CultureInfo = My.Application.UICulture
' Declaration
Public ReadOnly Property UICulture As System.Globalization.CultureInfo
```

戻り値

現在のスレッドがカルチャ固有のリソースを取得するために使用するカルチャを表現する CultureInfo オブジェクトを返します。

解説

My.Application.CurrentUICulture プロパティは、現在のスレッドがカルチャ固有のリソースを取得するために使用する **CultureInfo** オブジェクトを取得します。このオブジェクトは CurrentUICulture プロパティから返されるオブジェクトと同じものです。**CurrentUICulture** プロパティはリソース マネージャと **My.Resources** オブジェクトが使用するカルチャ、および実行時にカルチャ固有のリソースを検索するために必要な情報を判断します。

カルチャを変更するには、My.Application.ChangeUICulture メソッド を使用するか、**CurrentUICulture** プロパティに別の **CultureInfo** オブジェクトを割り当てます。

CurrentCulture の設定は、言語の設定とは違います。このプロパティには、地理的地域の標準設定に関連するデータだけが含まれています。

現在のスレッドが文字列の操作と書式設定に使用するカルチャを取得する場合は、**My.Application.CurrentCulture** プロパティを使用してください。

使用例

次の例は、**My.Application.CurrentCulture** プロパティを使用して現在のカルチャをキャッシュし、その後 My.Application.ChangeUICulture メソッド を使用してそれを変更します。**My.Application.ChangeUICulture** メソッドは、My.Resources オブジェクト がリソースを取得するために使うカルチャを設定します。

VB

```
Sub ShowLocalizedMessage()
    Dim culture As String = My.Application.UICulture.Name
    My.Application.ChangeUICulture("fr-FR")
    MsgBox(My.Resources.Message)
    My.Application.ChangeUICulture(culture)
End Sub
```

この例を実行するためには、アプリケーションのリソース ファイルに `Message` という名前の文字列があることと、そのリソース ファイルの French カルチャ版である `Resources.fr-FR.resx` が、アプリケーションに含まれていることが必要です。詳細については、「方法 : リソースを追加または削除する」を参照してください。

アプリケーションにそのリソース ファイルの French カルチャ版がない場合は、**My.Resource** オブジェクトが既定のカルチャのリソース ファイルからリソースを取得します。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase、ApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**

コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic で、ローカライズされたリソースを取得する

関連項目

My.Application オブジェクト
My.Application.ChangeUICulture メソッド
My.Application.Culture プロパティ
System.Globalization.CultureInfo
CurrentUICulture
Microsoft.VisualBasic.ApplicationServices.ApplicationBase.UICulture

My.Application.UnhandledException イベント

アプリケーションが未処理の例外を検出したときに発生します。

```
' Usage
Public Sub Me_UnhandledException( _
   ByVal sender As Object, _
   ByVal e As UnhandledExceptionEventArgs _
) Handles Me.UnhandledException
End Sub
' Declaration
Public Event UnhandledException( _
   ByVal sender As Object, _
   ByVal e As UnhandledExceptionEventArgs _
)
```

パラメータ

sender

イベントの発生元の **Object**。

e

未処理の例外と詳細な情報が格納された UnhandledExceptionEventArgs オブジェクト。

解説

アプリケーションは未処理の例外を検出すると **UnhandledException** イベントを発生させます。これは Visual Basic アプリケーション モデルに含まれるイベントです。詳細については、「Visual Basic アプリケーション モデルの概要」を参照してください。

e パラメータの Exception プロパティを使用すると、このイベントを発生させた未処理の例外にアクセスできます。

e パラメータの ExitApplication プロパティを使用すると、アプリケーションを終了するかどうかを制御できます。既定では、**ExitApplication** は **True** になっているため、アプリケーションは **UnhandledException** のイベント ハンドラが実行された後で終了します。**UnhandledException** イベント ハンドラ内で、この値を **True** に設定すると、アプリケーションの実行を継続し、待機中の状態に戻すことができます。

UnhandledException イベント ハンドラ用のコードは、ApplicationEvents.vb ファイルに格納されています。このファイルは既定では非表示です。

アプリケーション イベントのコード エディタ ウィンドウを表示するには、次の操作を行います。

1. ソリューション エクスプローラ でプロジェクトが選択されている状態で、[プロジェクト] メニューの [プロパティ] をクリックします。
2. [アプリケーション] タブをクリックします。
3. [アプリケーション イベントの表示] をクリックしてコード エディタを開きます。

詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

メモ :
Visual Basic コンパイラでは、デバッグ用にビルドされたアプリケーションがこのイベントを発生させることはありません。これは、デバッガが未処理の例外を処理できるようにするためです。つまり、アプリケーションを Visual Studio の統合開発環境のデバッガで実行してテストしている場合は、**UnhandledException** イベント ハンドラは呼び出されません。デバッグ用のアプリケーションのビルドの詳細については、「/debug (Visual Basic)」を参照してください。

処理手順

My.Application.UnhandledException イベントに関連するタスクの例を次の表に示します。

目的	参照項目
Visual Basic アプリケーション モデルによって提供されるイベントを使用してコードを実行する	方法 : アプリケーションの起動時または終了時にコードを実行する

未処理の例外のログを記録する	方法 : Visual Basic で例外をログに記録する

使用例

次の例は、**My.Application.UnhandledException** イベントを使用して、未処理の例外をすべてログに記録します。

VB

```
Private Sub MyApplication_UnhandledException( _
    ByVal sender As Object, _
    ByVal e As Microsoft.VisualBasic.ApplicationServices.UnhandledExceptionEventArgs _
) Handles Me.UnhandledException
    My.Application.Log.WriteException(e.Exception, _
        TraceEventType.Critical, _
        "Unhandled Exception.")
End Sub
```

アプリケーション イベントのコードを、コード エディタ ウィンドウに入力する必要があります。このウィンドウを利用するには、このトピックの「解説」で説明した手順に従ってください。詳細については、「方法 : アプリケーション イベントを処理する (Visual Basic)」を参照してください。

アプリケーションにデバッガが結合されている場合、**UnhandledException** イベントは発生しないため、この例は Visual Studio の統合開発環境の外部で実行する必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : WindowsFormsApplicationBase

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	不可
コンソール アプリケーション	不可
Windows コントロール ライブラリ	不可
Web コントロール ライブラリ	不可
Windows サービス	不可
Web サイト	不可

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
SecurityPermission	このイベントのイベント ハンドラを追加できるかどうかを制御します。関連する列挙値 : System.Security.Permissions.SecurityPermissionFlag.ControlAppDomain。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application オブジェクト

Microsoft.VisualBasic.ApplicationServices.UnhandledExceptionEventArgs

概念

Visual Basic アプリケーション モデルの概要

My.Application.Info オブジェクト

バージョン番号、説明、ロードされたアセンブリなど、アプリケーションに関する情報を取得するためのプロパティを提供します。

解説

My.Application.Info オブジェクトで公開されるプロパティの中には、アプリケーションのアセンブリ情報を返すものがあります。アプリケーションのアセンブリ内の情報を設定するには、プロジェクト デザイナの アプリケーション ペインにある [アセンブリ情報] ダイアログ ボックスを使う必要があります。詳細については、「方法 : アセンブリ情報を指定する」を参照してください。

My.Application.Info オブジェクトでは、FileVersionInfo クラスと類似した機能を利用できますが、非常に一般的なプロパティにすばやくアクセスするという機能しかありません。

LoadedAssemblies などの他のプロパティを使用すると、アプリケーションの現在の状態にアクセスできます。

使用例

次の例では、**My.Application.Info.Version** プロパティを使用して、アプリケーションのバージョンを表示します。

VB

```
MsgBox("Application version: " & My.Application.Info.Version.ToString)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照

関連項目

My.Application.Info オブジェクトのメンバ
My.Application オブジェクト
Microsoft.VisualBasic.ApplicationServices.AssemblyInfo
FileVersionInfo

My.Application.Info オブジェクトのメンバ

My.Application.Info オブジェクトには、バージョン番号、説明、ロードされたアセンブリなど、アプリケーションに関する情報を取得するためのプロパティが用意されています。

プロパティ

プロパティ	説明
AssemblyName	アプリケーションのアセンブリ ファイルの名前を、拡張子なしで取得します。
CompanyName	アプリケーションに関連付けられた会社名を取得します。
Copyright	アプリケーションに関連付けられた著作権表記を取得します。
Description	アプリケーションに関連付けられた説明を取得します。
DirectoryPath	アプリケーションが格納されているディレクトリを取得します。
LoadedAssemblies	アプリケーションによって読み込まれたすべてのアセンブリのコレクションを取得します。
ProductName	アプリケーションに関連付けられている製品名を取得します。
StackTrace	現在のスタック トレース情報を取得します。
Title	アプリケーションに関連付けられたタイトルを取得します。
[商標]	アプリケーションに関連付けられた商標表記を取得します。
Version	アプリケーションのバージョン番号を取得します。
WorkingSet	プロセスのコンテキストに割り当てられた物理メモリの量を取得します。

参照

関連項目

My.Application.Info オブジェクト

My.Application.Info.AssemblyName プロパティ

アプリケーションのアセンブリ ファイルの名前を拡張子抜きで取得します。

```
' Usage
Dim value As String = My.Application.Info.AssemblyName
' Declaration
Public ReadOnly Property AssemblyName As String
```

戻り値

ファイル名を含んでいる **String**

解説

My.Application.Info.AssemblyName プロパティは、マニフェストを含んでいるファイルの名前を取得します。

使用例

この例では、**My.Application.Info.AssemblyName** プロパティを使用してアプリケーションの名前を表示します。

VB

```
MsgBox("Application assembly name: " & My.Application.Info.AssemblyName)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application.Info.CompanyName プロパティ

アプリケーションに関連付けられた会社名を取得します。

```
' Usage
Dim value As String = My.Application.Info.CompanyName
' Declaration
Public ReadOnly Property CompanyName As String
```

戻り値

アプリケーションに関連付けられた会社名を格納する **String** です。

例外

例外を引き起こす可能性のある状態を次に示します。

- アセンブリに AssemblyCompanyAttribute がありません (InvalidOperationException)。

解説

My.Application.Info.CompanyName プロパティは、アプリケーションに関連付けられた会社名を取得します。

会社名は プロジェクト デザイナ の アプリケーション ペインにある、[アセンブリ情報] ダイアログ ボックスの [会社] の値を変えることによって変更できます。詳細については、「方法 : アセンブリ情報を指定する」を参照してください。

使用例

次の例では、**My.Application.Info.CompanyName** プロパティを使用して、アプリケーションに関連付けられた会社名を表示します。

VB

```
MsgBox("Application created by: " & My.Application.Info.CompanyName)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : アセンブリ情報を指定する

関連項目

My.Application.Info オブジェクト

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.CompanyName

My.Application.Info.Copyright プロパティ

アプリケーションに関連付けられている著作権表記を取得します。

```
' Usage
Dim value As String = My.Application.Info.Copyright
' Declaration
Public ReadOnly Property Copyright As String
```

戻り値

アプリケーションに関連付けられている著作権表記を含んだ **String**

例外

例外を引き起こす可能性のある状態を次に示します。

- アセンブリに AssemblyCopyrightAttribute がない (InvalidOperationException)。

解説

My.Application.Info.CompanyName プロパティは、アプリケーションに関連付けられている著作権表記を取得します。

著作権表記を変更するには、プロジェクト デザイナの [アプリケーション] から [アセンブリ情報] ダイアログ ボックスを表示し、[著作権] の値を変更します。詳細については、「方法 : アセンブリ情報を指定する」を参照してください。

使用例

この例では、**My.Application.Info.CompanyName** プロパティを使用して、アプリケーションに関連付けられている企業名を表示します。

VB

```
MsgBox("Application copyright: " & My.Application.Info.Copyright)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : アセンブリ情報を指定する

関連項目

My.Application.Info オブジェクト

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Copyright

My.Application.Info.Description プロパティ

アプリケーションに関連付けられた説明を取得します。

```
' Usage
Dim value As String = My.Application.Info.Description
' Declaration
Public ReadOnly Property Description As String
```

戻り値

アプリケーションに関連付けられた説明を格納する **String**。

例外

例外を引き起こす可能性のある状態を次に示します。

- アセンブリに AssemblyDescriptionAttribute がありません (InvalidOperationException)。

解説

My.Application.Info.Description プロパティは、アプリケーションに関連付けられた説明を取得します。

説明は プロジェクト デザイナ の アプリケーション ペインにある、[アセンブリ情報] ダイアログ ボックスの [説明] の値を変えることによって変更できます。詳細については、「方法 : アセンブリ情報を指定する」を参照してください。

使用例

次の例では、**My.Application.Info.Description** プロパティを使用して、アプリケーションに関連付けられた説明を表示します。

VB

```
MsgBox("Application description: " & My.Application.Info.Description)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : アセンブリ情報を指定する

関連項目

My.Application.Info オブジェクト

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Description

My.Application.Info.DirectoryPath プロパティ

アプリケーションが格納されているディレクトリを取得します。

```
' Usage
Dim value As String = My.Application.Info.DirectoryPath
' Declaration
Public ReadOnly Property DirectoryPath As String
```

戻り値

アプリケーションが格納されているディレクトリを含む **String** です。

解説

My.Application.Info.DirectoryPath プロパティは、アプリケーションが格納されているディレクトリを取得します。

使用例

次の例では、**My.Application.Info.DirectoryPath** プロパティを使用して、アプリケーションが格納されているディレクトリ パスを表示します。

VB

```
MsgBox("Application directory path: " & My.Application.Info.DirectoryPath)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application.Info オブジェクト
Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.DirectoryPath

My.Application.Info.LoadedAssemblies プロパティ

アプリケーションによって読み込まれたすべてのアセンブリのコレクションを取得します。

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of System.Reflection.Assembl
y) = My.Application.Info.LoadedAssemblies
' Declaration
Public ReadOnly Property LoadedAssemblies As System.Collections.ObjectModel.ReadOnlyCollect
ion(Of System.Reflection.Assembly)
```

戻り値

アプリケーションによって読み込まれたすべてのアセンブリを含む Assembly の ReadOnlyCollection です。

例外

例外を引き起こす可能性のある状態を次に示します。

- アプリケーション ドメインが読み込まれていません (AppDomainUnloadedException)。

解説

My.Application.Info.LoadedAssemblies プロパティは GetAssemblies メソッドとよく似ています。

使用例

次の例では、**My.Application.Info.LoadedAssemblies** プロパティを使用して、アプリケーションによって読み込まれたアセンブリを表示します。

VB

```
ListBox1.DataSource = My.Application.Info.LoadedAssemblies
```

この例では、Windows フォーム アプリケーションに `ListBox1` という名前の ListBox コントロールがあることが必要です。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Application.Info オブジェクト
ReadOnlyCollection
System.Reflection.Assembly
Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.LoadedAssemblies

My.Application.Info.ProductName プロパティ

アプリケーションに関連付けられている製品名を取得します。

```
' Usage
Dim value As String = My.Application.Info.ProductName
' Declaration
Public ReadOnly Property ProductName As String
```

戻り値

アプリケーションに関連付けられている製品名を含んだ **String**

例外

例外を引き起こす可能性のある状態を次に示します。

- アセンブリに AssemblyProductAttribute がない場合 (InvalidOperationException)。

解説

My.Application.Info.ProductName プロパティは、アプリケーションに関連付けられている製品名を取得します。

製品名を変更するには、プロジェクト デザイナの アプリケーション ペインの [アセンブリ情報] ダイアログ ボックスを表示し、[製品] の値を変更します。詳細については、「方法 : アセンブリ情報を指定する」を参照してください。

使用例

この例では、**My.Application.Info.ProductName** プロパティを使用して、アプリケーションに関連付けられている製品名を表示します。

VB

```
MsgBox("Application product name: " & My.Application.Info.ProductName)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : アセンブリ情報を指定する

関連項目

My.Application.Info オブジェクト

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.ProductName

My.Application.Info.StackTrace プロパティ

現在のスタック トレース情報を取得します。

```
' Usage
Dim value As String = My.Application.Info.StackTrace
' Declaration
Public ReadOnly Property StackTrace As String
```

戻り値

現在のスタック トレース情報を格納する **String** です。戻り値が Empty になる場合もあります。

例外

例外を引き起こす可能性のある状態を次に示します。

- 要求されたスタック トレース情報が範囲外である場合 (ArgumentOutOfRangeException)。

解説

My.Application.Info.StackTrace プロパティはメソッド呼び出しを新しい順に記述します。つまり、最新のメソッド呼び出しが最初に記述され、スタック上の各メソッド呼び出しが改行で区切って一覧入力されます。ただし、最適化によってコードが変換されるために、予想される数のメソッド呼び出しを **My.Application.Info.StackTrace** プロパティが報告しない場合もあります。

通常、スタック トレース情報はアプリケーションのデバッグに使用されます。

各メソッド呼び出しのスタック トレース情報は、次のような形式で書き込まれます。

"at *FullClassName.MethodName*(*MethodParams*) in *FileName*:line *LineNumber* "

デバッグ シンボルが使用できない場合は、リテラルの "at" の前に空白が 3 つ置かれ、"in" と ":line" のリテラルが省略されます。*FullClassName*、*MethodName*、*MethodParams*、*FileName*、および *LineNumber* のプレースホルダは実際の値で置き換えます。次にその定義を示します。

指定項目	説明
FullClassName	名前空間を含むクラスの完全名です。
MethodName	メソッドの名前です。
MethodParams	パラメータのリストを型と名前のペアで定義します。各ペアはコンマ (,) で区切ります。*MethodName* がパラメータを受け取らない場合、この情報は省略されます。
FileName	*MethodName* メソッドが宣言されているソース ファイルの名前。デバッグ シンボルが使用できない場合、この情報は省略されます。
LineNumber	呼び出し履歴上の命令に対する *MethodName* のソース コードを含む、*FileName* 内の行番号。デバッグ シンボルが使用できない場合、この情報は省略されます。

My.Applicaiton.Info.StackTrace プロパティは、StackTrace プロパティと同様の機能を提供します。

使用例

次の例は **My.Application.Info.StackTrace** プロパティを使用して、コードが実行されたポイントからの、アプリケーションのスタック トレースを表示します。

VB

```
MsgBox("Stack trace: " & My.Application.Info.StackTrace)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイル パス内の情報へのアクセス許可を制御します。関連する列挙値 : PathDiscovery。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application.Info オブジェクト
StackTrace
Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.StackTrace

My.Application.Info.Title プロパティ

アプリケーションに関連付けられたタイトルを取得します。

```
' Usage
Dim value As String = My.Application.Info.Title
' Declaration
Public ReadOnly Property Title As String
```

戻り値

アプリケーションに関連付けられた AssemblyTitleAttribute を格納する **String**。

例外

例外を引き起こす可能性のある状態を次に示します。

- アセンブリに **AssemblyTitleAttribute** がありません (InvalidOperationException)。

解説

My.Application.Info.Title プロパティは、アプリケーションに関連付けられたタイトルを取得します。

タイトルはプロジェクト デザイナの アプリケーション ペインにある、[アセンブリ情報] ダイアログ ボックスの [タイトル] の値を変えることによって変更できます。詳細については、「方法 : アセンブリ情報を指定する」を参照してください。

使用例

次の例では、**My.Application.Info.Title** プロパティを使用して、アプリケーションに関連付けられたタイトルを表示します。

VB

```
MsgBox("Application title: " & My.Application.Info.Title)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : アセンブリ情報を指定する

関連項目

My.Application.Info オブジェクト

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Title

My.Application.Info.Trademark プロパティ

アプリケーションに関連付けられた商標表記を取得します。

```
' Usage
Dim value As String = My.Application.Info.Trademark
' Declaration
Public ReadOnly Property Trademark As String
```

戻り値

アプリケーションに関連付けられた商標表記を格納する **String**。

例外

例外を引き起こす可能性のある状態を次に示します。

- アセンブリに AssemblyTrademarkAttribute がありません (InvalidOperationException)。

解説

My.Application.Info.Trademark プロパティは、アプリケーションに関連付けられた商標表記を取得します。

商標表記は [プロジェクト デザイナ] の アプリケーション ペインにある、[アセンブリ情報] ダイアログ ボックスの [商標] の値を変えることによって変更できます。詳細については、「方法 : アセンブリ情報を指定する」を参照してください。

使用例

次の例では、**My.Application.Info.Trademark** プロパティを使用して、アプリケーションに関連付けられた商標情報を表示します。

VB

```
MsgBox("Application trademark: " & My.Application.Info.Trademark)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : アセンブリ情報を指定する

関連項目

My.Application.Info オブジェクト

Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Trademark

My.Application.Info.Version プロパティ

アプリケーションのバージョン番号を取得します。

```
' Usage
Dim value As System.Version = My.Application.Info.Version
' Declaration
Public ReadOnly Property Version As System.Version
```

戻り値

アプリケーションのバージョン番号を含んでいる Version オブジェクト

例外

例外を引き起こす可能性のある状態を次に示します。

- アプリケーションがアセンブリのバージョンにアクセスするのに必要なアクセス許可を持っていない (SecurityException)。

解説

My.Application.Info.Version プロパティでは、アプリケーションのバージョン番号を含んでいる **Version** オブジェクトを取得できます。**Version** オブジェクトの Major、Minor、Build、および Revision プロパティを使用すると、アプリケーションの特定のバージョン情報にアクセスできます。

ClickOnce を使用して配置したアプリケーションの場合は、My.Application.Deployment プロパティのCurrentVersion プロパティを使用します。

使用例

この例では、**My.Application.Info.Version** プロパティを使用してアプリケーションのバージョンを表示します。

VB

```
MsgBox("Application version: " & My.Application.Info.Version.ToString)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application.Info オブジェクト
My.Application.Deployment プロパティ
CurrentVersion
System.Version
Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.Version

My.Application.Info.WorkingSet プロパティ

プロセスのコンテキストに割り当てられた物理メモリの量を取得します。

```
' Usage
Dim value As Long = My.Application.Info.WorkingSet
' Declaration
Public ReadOnly Property WorkingSet As Long
```

戻り値

プロセスのコンテキストに割り当てられた物理メモリのバイト数を格納する **Long** です。

例外

例外を引き起こす可能性のある状態を次に示します。

- ユーザーが部分的に信頼されており、必要なアクセス許可を持っていません (SecurityException)。

解説

My.Application.Info.WorkingSet プロパティと WorkingSet プロパティの動作は同じです。

メモ :
このプロパティは、Windows 98 および Windows ME (Millennium Edition) では常に 0 を返します。

使用例

次の例は **My.Application.Info.WorkingSet** プロパティを使用して、アプリケーションに割り当てられた物理メモリのバイト数を表示します。

VB

```
MsgBox("Application working set: " & My.Application.Info.WorkingSet)
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : AssemblyInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	**使用可/不可**
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
EnvironmentPermission	すべての環境変数へのアクセスを許可するかどうかを制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Application.Info オブジェクト
WorkingSet
Microsoft.VisualBasic.ApplicationServices.AssemblyInfo.WorkingSet

My.Application.Log オブジェクト

イベントと例外の情報を作成するためのプロパティとメソッドを、アプリケーション ログのリスナに提供します。

解説

My.Application.Log オブジェクトには、.NET Framework のログ サービスにアクセスするための、わかりやすいエントリ ポイントが用意されています。**WriteEntry** メソッドと **WriteException** メソッドは、アプリケーション ログのリスナにメッセージを書き込みます。リスナはアプリケーションの構成ファイルで構成できます。詳細については、「チュートリアル : My.Application.Log による情報の書き込み先の変更」および「Visual Basic でのアプリケーション ログの使用」を参照してください。

My.Application.Log オブジェクトは、クライアント アプリケーションにのみ使用できます。Web アプリケーションの場合は、**My.Log** を使用します。詳細については、「My.Log オブジェクト」を参照してください。

処理手順

My.Application.Log オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
アプリケーション ログのリスナにイベント情報を書き込む	方法 : ログ メッセージを書き込む
アプリケーション ログのリスナに例外の情報を書き込む	方法 : Visual Basic で例外をログに記録する
My.Application.Log が情報を書き込む場所を調べる	チュートリアル : My.Application.Log による情報の書き込み先の確認

使用例

My.Application.Log.WriteEntry メソッドを使用して、トレース情報のログを記録する方法は次の例のようになります。詳細については、「方法 : ログ メッセージを書き込む」を参照してください。

VB

```
Public Sub TracingTest(ByVal fileName As String)
    My.Application.Log.WriteEntry( _
        "Entering TracingTest with argument " & _
        fileName & ".")
    ' Code to trace goes here.
    My.Application.Log.WriteEntry( _
        "Exiting TracingTest with argument " & _
        fileName & ".")
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic.Logging

クラス : Log

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可

Windows サービス	**可**
Web サイト	**不可**

参照

関連項目

My.Application.Log オブジェクトのメンバ

My.Application オブジェクト

My.Application.Log オブジェクトのメンバ

My.Application.Log オブジェクトは、イベントと例外の情報を作成するためのプロパティとメソッドを、アプリケーション ログのリスナに提供します。

プロパティ

プロパティ	説明
TraceSource	**My.Application.Log** オブジェクトの基礎にある TraceSource オブジェクトを取得します。 これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

メソッド

メソッド	説明
WriteEntry	アプリケーション ログのリスナにメッセージを書き込みます。
WriteException	アプリケーション ログのリスナに例外の情報を書き込みます。

参照

関連項目

My.Application.Log オブジェクト

DefaultFileLogWriter プロパティ (My.Application.Log および My.Log)

Log オブジェクトが使用する FileLogTraceListener オブジェクトを取得します。

```
' Usage
Dim value As Logging.FileLogTraceListener = My.Application.Log.DefaultFileLogWriter
' Web usage
Dim value As Logging.FileLogTraceListener = My.Log.DefaultFileLogWriter
' Declaration
Public ReadOnly Property DefaultFileLogWriter As Logging.FileLogTraceListener
```

戻り値

Log オブジェクトが使用する **FileLogTraceListener** オブジェクトを返します。

解説

このプロパティは、**Log** オブジェクトの **FileLogTraceListener** オブジェクトの現在の構成を確認するために使用できます。

クライアント アプリケーションでは、**My.Application.Log** オブジェクトを通じて **Log** オブジェクトにアクセスできます。Web アプリケーションでは、**My.Log** オブジェクトを通じて **Log** オブジェクトにアクセスできます。

必要条件

名前空間 : Microsoft.VisualBasic.Logging

クラス : Log

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

トラブルシューティング : ログ リスナ

関連項目

My.Application.Log オブジェクト

My.Log オブジェクト

FileLogTraceListener

TraceSource プロパティ (My.Application.Log and My.Log)

Log の基になる TraceSource オブジェクトを取得します。

```
' Usage
Dim value As System.Diagnostics.TraceSource = My.Application.Log.TraceSource
' Web usage
Dim value As System.Diagnostics.TraceSource = My.Log.TraceSource
' Declaration
Public ReadOnly Property TraceSource As System.Diagnostics.TraceSource
```

戻り値

Log オブジェクトの基になる **TraceSource** オブジェクトを返します。

解説

このプロパティは、**Log** オブジェクトの現在の構成を確認するために使用できます。

クライアント アプリケーションでは、**My.Application.Log** オブジェクトをとおして **Log** オブジェクトを使用できます。Web アプリケーションでは、**My.Log** オブジェクトをとおして **Log** オブジェクトを使用できます。

これは詳細メンバであるため、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

処理手順

TraceSource プロパティに関連するタスクの例を次の表に示します。

タスク	参照項目
各 **Log** オブジェクトのログ リスナに関する情報を取得します。	トラブルシューティング : ログ リスナ

必要条件

名前空間 : Microsoft.VisualBasic.Logging

クラス : Log

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類別の可用性

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

処理手順

トラブルシューティング : ログ リスナ

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
System.Diagnostics.TraceSource
Microsoft.VisualBasic.Logging.Log.TraceSource

WriteEntry メソッド (My.Application.Log and My.Log)

メッセージをアプリケーションのログ リスナに書き込みます。

```
' Usage
My.Application.Log.WriteEntry(message)
My.Application.Log.WriteEntry(message ,severity)
My.Application.Log.WriteEntry(message ,severity ,id)
' Web usage
My.Log.WriteEntry(message)
My.Log.WriteEntry(message ,severity)
My.Log.WriteEntry(message ,severity ,id)
' Declaration
Public Sub WriteEntry( _
   ByVal message As String _
)
' -or-
Public Sub WriteEntry( _
   ByVal message As String, _
   ByVal severity As System.Diagnostics.TraceEventType _
)
' -or-
Public Sub WriteEntry( _
   ByVal message As String, _
   ByVal severity As System.Diagnostics.TraceEventType, _
   ByVal id As Integer _
)
```

パラメータ

message

必ず指定します。ログに記録するメッセージ。*message* が **Nothing** である場合は、空の文字列が使用されます。

severity

メッセージの種類を指定します。既定では **TraceEventType.Information** です。

id

メッセージ識別子を指定します。通常は、関連付けのために使用します。既定で、表に示したように *entryType* に関連付けられます。

例外

例外を引き起こす状況を次に示します。

- 例外またはメッセージの型が、**Nothing** である (ArgumentNullException)。
- メッセージの型が、TraceEventType 列挙値の 1 つではない (InvalidEnumArgumentException)。
- 部分的に信頼されるコードがメソッドを呼び出すが、イベント ログ リスナに書き込むには完全に信頼されている必要がある (SecurityException)。

解説

WriteEntry メソッドは、メッセージをアプリケーションのイベント ログ リスナに書き込みます。

クライアント アプリケーションでは、**My.Application.Log** オブジェクトをとおして **Log** オブジェクトを使用できます。Web アプリケーションでは、**My.Log** オブジェクトをとおして **Log** オブジェクトを使用できます。

どのログ リスナが **WriteEntry** メソッドのメッセージを受け取るかについては、「チュートリアル : My.Application.Log による情報の書き込み先の確認」を参照してください。既定のログ リスナは変更できます。詳細については、「Visual Basic でのアプリケーション ログの使用」を参照してください。

id 引数を受け取らないオーバーロードの場合、ログに書き込まれる *id* は次の表のように定義されます。

severity	**既定の *id***

Information	0
Warning	1
Error	2
Critical	3
Start	4
Stop	5
Suspend	6
Resume	7
Verbose	8
Transfer	9

処理手順

WriteEntry メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
イベント情報をアプリケーションのログ リスナに書き込みます。	方法 : ログ メッセージを書き込む
Log が情報を書き込む場所を決定します。	チュートリアル : My.Application.Log による情報の書き込み先の確認

使用例

次のコード例は、**My.Application.Log.WriteEntry** メソッドを使ってトレース情報をログに記録する方法を示します。詳細については、「方法 : ログ メッセージを書き込む」を参照してください。

VB

```
Public Sub TracingTest(ByVal fileName As String)
    My.Application.Log.WriteEntry( _
        "Entering TracingTest with argument " & _
        fileName & ".")
    ' Code to trace goes here.
    My.Application.Log.WriteEntry( _
        "Exiting TracingTest with argument " & _
        fileName & ".")
End Sub
```

このコード例は、クライアント アプリケーション内だけで実行できます。Web アプリケーションで使用するには、**My.Application.Log.WriteEntry** を **My.Log.WriteEntry** に変更します。

必要条件

名前空間 : Microsoft.VisualBasic.Logging

クラス : Log

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類別の可用性

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**

クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	**可**

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : ログ メッセージを書き込む

チュートリアル : My.Application.Log による情報の書き込み先の確認

関連項目

My.Application.Log オブジェクト

My.Log オブジェクト

System.Diagnostics.TraceEventType

Microsoft.VisualBasic.Logging.Log.WriteEntry(System.String)

概念

Visual Basic でのアプリケーション ログの使用

WriteException メソッド (My.Application.Log および My.Log)

例外情報をアプリケーションのログ リスナに書き込みます。

```
' Usage
My.Application.Log.WriteException(ex)
My.Application.Log.WriteException(ex ,severity ,additionalInfo)
My.Application.Log.WriteException(ex ,severity ,additionalInfo ,id)
' Web usage
My.Log.WriteException(ex)
My.Log.WriteException(ex ,severity ,additionalInfo)
My.Log.WriteException(ex ,severity ,additionalInfo ,id)
' Declaration
Public Sub WriteException( _
   ByVal ex As System.Exception _
)
' -or-
Public Sub WriteException( _
   ByVal ex As System.Exception, _
   ByVal severity As System.Diagnostics.TraceEventType, _
   ByVal additionalInfo As String _
)
' -or-
Public Sub WriteException( _
   ByVal ex As System.Exception, _
   ByVal severity As System.Diagnostics.TraceEventType, _
   ByVal additionalInfo As String, _
   ByVal id As Integer _
)
```

パラメータ

ex

必ず指定します。ログに記録する例外を指定します。

severity

メッセージの種類を指定します。既定では Error です。

additionalInfo

メッセージに付け加える文字列を指定します。既定では、空の文字列になります。

id

相関に使用されるメッセージ ID を指定します。既定では、「解説」の表に示されているとおり *entryType* に関連付けられます。

例外

次の条件では例外が発生します。

- 例外またはメッセージの種類が **Nothing** である (ArgumentNullException)。
- メッセージの種類が TraceEventType 列挙値のどれでもない (InvalidEnumArgumentException)。
- 部分信頼を持つコードがこのメソッドを呼び出して、完全信頼を必要とするイベント ログ リスナに書き込みを行っている (SecurityException)。

解説

WriteException メソッドは、例外に含まれている情報をアプリケーションのイベント ログ リスナに書き込みます。

クライアント アプリケーションでは、**My.Application.Log** オブジェクトを通じて **Log** オブジェクトにアクセスできます。Web アプリケーションでは、**My.Log** オブジェクトを通じて **Log** オブジェクトにアクセスできます。

どのログ リスナが **WriteException** メソッドのメッセージを受信するかについては、
「チュートリアル : My.Application.Log による情報の書き込み先の確認」を参照してください。既定のログ リスナは変更可能です。詳細につい

ては、「Visual Basic でのアプリケーション ログの使用」を参照してください。

id 引数を使用しないオーバーロードでは、ログに書き込まれる ID は次の表によって定義されます。

severity	**既定の *id***
Information	0
Warning	1
Error	2
Critical	3
Start	4
Stop	5
Suspend	6
Resume	7
Verbose	8
Transfer	9

処理手順

WriteException メソッドに関連するタスクの例を次の表に示します。

目的	**参照項目**
例外情報をアプリケーションのイベント ログ リスナに書き込む	方法 : Visual Basic で例外をログに記録する
Log が情報をどこに書き込むかを調べる	チュートリアル : My.Application.Log による情報の書き込み先の確認

使用例

この例は、**My.Application.Log.WriteException** メソッドを使用して例外をログに記録する方法を示しています。NullReferenceException 例外を発生させるには、`Dim` 行と `MsgBox` 行をコメントから外してください。詳細については、「方法 : Visual Basic で例外をログに記録する」を参照してください。

VB

```
Public Sub ExceptionLogTest(ByVal fileName As String)
    Try
        ' Code that might generate an exception goes here.
        ' For example:
        '    Dim x As Object
        '    MsgBox(x.ToString)
    Catch ex As Exception
        My.Application.Log.WriteException(ex, _
            TraceEventType.Error, _
            "Exception in ExceptionLogTest " & _
            "with argument " & fileName & ".")
    End Try
End Sub
```

このコード例は、クライアント アプリケーション内でのみ実行できます。Web アプリケーションの場合は、**My.Application.Log.WriteException** を **My.Log.WriteException** に変更します。

必要条件

名前空間 : Microsoft.VisualBasic.Logging

クラス : Log

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic で例外をログに記録する

チュートリアル : My.Application.Log による情報の書き込み先の確認

関連項目

My.Application.Log オブジェクト

My.Log オブジェクト

System.Exception

System.Diagnostics.TraceEventType

Microsoft.VisualBasic.Logging.Log.WriteException(System.Exception)

概念

Visual Basic でのアプリケーション ログの使用

My.Computer オブジェクト

オーディオ、時計、キーボード、ファイル システムなど、コンピュータの構成要素を操作するプロパティを提供します。

解説

My.Computer オブジェクトのプロパティは、アプリケーションの配置先コンピュータに関する実行時の情報を返します。通常は、このデータは開発用コンピュータ上の情報とは異なります。

一部のメンバ (**My.Computer.Audio** オブジェクトなど) は、非サーバー アプリケーションでのみ使用できます。

処理手順

My.Computer オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
接続ステータスを調べる	方法 : Visual Basic で接続ステータスをチェックする
リモート コンピュータが使用可能かどうかを調べる	方法 : Visual Basic でリモート コンピュータが利用可能かどうかを確認する
ファイルをダウンロードする	方法 : Visual Basic でファイルをダウンロードする
ファイルをアップロードする	方法 : Visual Basic でファイルをアップロードする
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
クリップボードを消去する	方法 : Visual Basic でクリップボードを消去する
クリップボードからデータを読み取る	方法 : Visual Basic でクリップボードから読み込む
クリップボードに格納されているデータを確認する	方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する
クリップボードにオーディオを保存する	方法 : Visual Basic でオーディオ ストリームをクリップボードに保存する
クリップボードからイメージを取得する	方法 : Visual Basic でクリップボードからイメージを取得する

使用例

この例では、**My.Computer.Name** プロパティを使用して、このコードを実行しているコンピュータの名前を表示します。

VB

```
MsgBox("Computer name: " & My.Computer.Name)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer (ServerComputer 基本クラスは全プロジェクトで使用可能なメンバを提供)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照

関連項目

My.Computer オブジェクトのメンバ
My.Computer.Audio オブジェクト
My.Computer.Clipboard オブジェクト
My.Computer.Clock オブジェクト
My.Computer.FileSystem オブジェクト
My.Computer.Info オブジェクト
My.Computer.Keyboard オブジェクト

My.Computer.Mouse オブジェクト
My.Computer.Network オブジェクト
My.Computer.Ports オブジェクト
My.Computer.Registry オブジェクト
Microsoft.VisualBasic.Devices.Computer
Microsoft.VisualBasic.Devices.ServerComputer

My.Computer オブジェクトのメンバ

My.Computer オブジェクトは、オーディオ、時計、キーボード、ファイル システムなど、コンピュータの構成要素を操作するプロパティを提供します。

プロパティ

プロパティ	説明
Audio	My.Computer.Audio オブジェクトを返します。このオブジェクトは、コンピュータのオーディオ システムへのアクセスを提供し、.wav ファイルを再生するためのメソッドを公開します。 このオブジェクトは、非サーバー アプリケーションだけで使用できます。
Clipboard	My.Computer.Clipboard オブジェクトを返します。このオブジェクトは、クリップボードを操作するためのメソッドを公開します。 このオブジェクトは、非サーバー アプリケーションだけで使用できます。
Clock	My.Computer.Clock オブジェクトを返します。このオブジェクトは、現在の現地時刻および世界協定時刻 (グリニッジ標準時に等しい) にシステム時計からアクセスするためのプロパティを公開します。
FileSystem	My.Computer.FileSystem オブジェクトを返します。このオブジェクトは、ドライブ、ファイル、ディレクトリを操作するためのプロパティとメソッドを公開します。
Info	My.Computer.Info オブジェクトを返します。このオブジェクトは、コンピュータのメモリ、ロードされているアセンブリ、名前、およびオペレーティング システムについての情報を取得するためのプロパティを公開します。
Keyboard	My.Computer.Keyboard オブジェクトを返します。このオブジェクトは、キーボードの現在の状態 (現在どのキーが押されているかなど) にアクセスするためのプロパティや、アクティブなウィンドウにキーストロークを送るためのメソッドを公開します。 このオブジェクトは、非サーバー アプリケーションだけで使用できます。
Mouse	My.Computer.Mouse オブジェクトを返します。ローカル コンピュータに接続されているマウスの形式および構成の情報を収集するためのプロパティを公開します。 このオブジェクトは、非サーバー アプリケーションだけで使用できます。
Name	コンピュータの名前を取得します。
Network	My.Computer.Network オブジェクトを返します。このオブジェクトは、ネットワークの種類とイベントにアクセスするときに使用します。
Ports	My.Computer.Ports オブジェクトを返します。このオブジェクトは、コンピュータのシリアル ポートにアクセスするためのプロパティとメソッドを公開します。 このオブジェクトは、非サーバー アプリケーションだけで使用できます。
Registry	My.Computer.Registry オブジェクトを返します。このオブジェクトは、レジストリの値を読み書きするときに使用します。
Screen	コンピュータのプライマリ ディスプレイ画面を表す Screen オブジェクトを取得します。 このプロパティは、非サーバー アプリケーションでのみ使用できます。

参照

関連項目

My.Computer オブジェクト

My.Computer.Audio プロパティ

サウンドを再生するメソッドのためのプロパティを備えたオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Audio = My.Computer.Audio
' Declaration
Public ReadOnly Property Audio As MyAudio
```

戻り値

このプロパティは、コンピュータの **My.Computer.Audio** オブジェクトを返します。

解説

このプロパティによって、**My.Computer.Audio** オブジェクトに簡単にアクセスできます。詳細については、「My.Computer.Audio オブジェクト」を参照してください。

使用例

My.Computer.Audio.Play メソッドは、**AudioPlayMode.Background** が指定されたときに指定のサウンドをバックグラウンドで再生します。

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

このコード例は、Windows フォーム アプリケーション内だけで実行できます。

ファイル名は、使用しているシステム上のサウンド ファイルを参照するものである必要があります。

サウンド ファイルの管理を容易にするためには、ファイルをアプリケーション リソースとして格納することをお勧めします。それにより、My.Resources オブジェクト を使ってファイルにアクセスできるようになります。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト

My.Computer.Audio オブジェクト

My.Computer.Clipboard プロパティ

クリップボードを操作するためのメソッドを提供するオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.MyServices.ClipboardProxy = My.Computer.Clipboard
' Declaration
Public ReadOnly Property Clipboard As ClipboardProxy
```

戻り値

このプロパティは、コンピュータの My.Computer.Clipboard オブジェクト を返します。

解説

このプロパティによって、My.Computer.Clipboard オブジェクト に簡単にアクセスできます。

このプロパティは、非サーバー アプリケーションだけで使用できます。

使用例

クリップボードからテキストを読み込んで文字列 `textOnClipboard` に格納する例は、次のようになります。

VB

```
Dim textOnClipboard As String = My.Computer.Clipboard.GetText()
```

この例では、クリップボードにテキストがなければエラーになります。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト

My.Computer.Clipboard オブジェクト

その他の技術情報

My.Computer.Clock プロパティ

現在の現地時刻および世界協定時刻 (グリニッジ標準時に等しい) にシステム時計からアクセスするためのプロパティを提供するオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Clock = My.Computer.Clock
' Declaration
Public ReadOnly Property Clock As MyClock
```

戻り値

このプロパティは、コンピュータの **My.Computer.Clock** オブジェクトを返します。

解説

このプロパティを使用して、**My.Computer.Clock** オブジェクトに簡単にアクセスできます。詳細については、「My.Computer.Clock オブジェクト」を参照してください。

使用例

My.Computer.Audio.Play メソッドは、**AudioPlayMode.Background** が指定されたときに、指定されたサウンドをバックグラウンドで再生します。

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

このコードは、Windows フォーム アプリケーション内だけで実行できます。

ファイル名は、使用中のシステムに置かれた .wav サウンド ファイルの名前であることが必要です。

サウンド ファイルを簡単に管理するために、アプリケーション リソースとして格納することをお勧めします。それにより、My.Resources オブジェクトを使ってファイルにアクセスできるようになります。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer、ServerComputer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**

Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト

My.Computer.Clock オブジェクト

My.Computer.FileSystem プロパティ

ドライブ、ファイル、およびディレクトリを操作するためのプロパティとメソッドを提供するオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.MyServices.FileSystemProxy = My.Computer.FileSystem
' Declaration
Public ReadOnly Property FileSystem As FileSystemProxy
```

戻り値

このプロパティは、コンピュータの My.Computer.FileSystem オブジェクト を返します。

解説

このプロパティによって、My.Computer.FileSystem オブジェクト に簡単にアクセスできます。

使用例

次の例は、 `C:\backup\logs` フォルダが存在するかどうかを判断し、そのプロパティを調べます。

VB

```
Dim logInfo As System.IO.DirectoryInfo
If My.Computer.FileSystem.DirectoryExists("C:\backup\logs") Then
    logInfo = My.Computer.FileSystem.GetDirectoryInfo _
    ("C:\backup\logs")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer、ServerComputer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト

My.Computer.FileSystem オブジェクト

その他の技術情報

Visual Basic でのファイルの読み取り
Visual Basic でのファイルへの書き込み
Visual Basic でのファイルおよびディレクトリの作成、削除、および移動
Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.Info プロパティ

コンピュータのメモリ、ロードされているアセンブリ、名前、およびオペレーティング システムについての情報を取得するためのプロパティを備えたオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.ComputerInfo = My.Computer.Info
' Declaration
Public ReadOnly Property Info As MyComputerInfo
```

戻り値

このプロパティは、コンピュータの **My.Computer.Info** オブジェクトを返します。

解説

このプロパティによって、**My.Computer.Info** オブジェクトに簡単にアクセスできます。詳細については、「My.Computer.Info オブジェクト」を参照してください。

使用例

この例では、**My.Computer.Info.AvailablePhysicalMemory** プロパティを使用して、このコードを実行しているコンピュータの使用可能な物理メモリの量を表示します。

VB

```
MsgBox("Computer's available physical memory: " & _
    My.Computer.Info.AvailablePhysicalMemory)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer、ServerComputer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト
My.Computer.Info オブジェクト

My.Computer.Keyboard プロパティ

キーボードの現在の状態 (現在どのキーが押されているかなど) にアクセスするためのプロパティや、アクティブなウィンドウにキーストロークを送るためのメソッドが用意されているオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Keyboard = My.Computer.Keyboard
' Declaration
Public ReadOnly Property Keyboard As Microsoft.VisualBasic.Devices.Keyboard
```

戻り値

このプロパティは、コンピュータの **My.Computer.Keyboard** を返します。

解説

このプロパティによって、**My.Computer.Keyboard** オブジェクトに簡単にアクセスできます。詳細については、「My.Computer.Keyboard オブジェクト」を参照してください。

使用例

次のコード例は、**My.Computer.Keyboard.CtrlKeyDown** プロパティを使って、このコンピュータの Ctrl キーが押されているかどうかを確認します。

VB

```
If My.Computer.Keyboard.CtrlKeyDown Then
    MsgBox("CTRL key down")
Else
    MsgBox("CTRL key up")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト
My.Computer.Keyboard オブジェクト

My.Computer.Mouse プロパティ

ローカル コンピュータに接続されているマウスの形式および構成の情報を収集するためのプロパティを公開しているオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Mouse = My.Computer.Mouse
' Declaration
Public ReadOnly Property Mouse As Microsoft.VisualBasic.Devices.Mouse
```

戻り値

このプロパティは、コンピュータの **My.Computer.Mouse** オブジェクトを返します。

解説

このプロパティによって、**My.Computer.Mouse** オブジェクトに簡単にアクセスできます。詳細については、「My.Computer.Mouse オブジェクト」を参照してください。

このプロパティは、非サーバー アプリケーションでのみ使用できます。

使用例

この例では、**My.Computer.Mouse.WheelExists** プロパティと **My.Computer.Mouse.WheelScrollLines** プロパティを使用して、マウスにスクロール ホイールがあるかどうかと、ホイール回転時のスクロール量を調べます。

VB

```
If My.Computer.Mouse.WheelExists Then
    Dim lines As Integer = My.Computer.Mouse.WheelScrollLines
    If lines > 0 Then
        MsgBox("Application scrolls " & _
            lines & " line(s) for each wheel turn.")
    Else
        MsgBox("Application scrolls " & _
            (-lines) & " page(s) for each wheel turn.")
    End If
Else
    MsgBox("Mouse has no scroll wheel.")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**

Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト

My.Computer.Mouse オブジェクト

My.Computer.Name プロパティ

コンピュータ名を取得します。

```
' Usage
Dim value As String = My.Computer.Name
' Declaration
Public ReadOnly Property Name As String
```

戻り値

コンピュータの名前を格納した **String** を返します。

解説

My.Computer.Name プロパティは、MachineName プロパティと同様の機能を提供します。

使用例

次のコード例は、**My.Computer.Name** プロパティを使って、このコードが実行されるコンピュータの名前を表示します。

VB

```
MsgBox("Computer name: " & My.Computer.Name)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer、ServerComputer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト
MachineName
Microsoft.VisualBasic.Devices.ServerComputer.Name

My.Computer.Network プロパティ

コンピュータの接続先のネットワークと対話するためのプロパティおよびメソッドを提供するオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Network = My.Computer.Network
' Declaration
Public ReadOnly Property Network As MyNetwork
```

戻り値

このプロパティは、コンピュータの My.Computer.Network オブジェクト を返します。

解説

このプロパティによって、My.Computer.Network オブジェクト に簡単にアクセスできます。

使用例

この例では、`Order.txt` ファイルを http://www.cohowinery.com/uploads にアップロードします。

VB

```
My.Computer.Network.UploadFile ( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/uploads.aspx")
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer、ServerComputer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト

My.Computer.Network オブジェクト

その他の技術情報

ネットワーク操作の実行

My.Computer.Ports プロパティ

コンピュータのシリアル ポートにアクセスするためのプロパティとメソッドが用意されているオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.Devices.Ports = My.Computer.Ports
' Declaration
Public ReadOnly Property Ports As MyPorts
```

戻り値

このプロパティは、**My.Computer.Ports** オブジェクトを返します。

解説

このプロパティによって、**My.Computer.Ports** オブジェクトに簡単にアクセスできます。詳細については、「My.Computer.Ports オブジェクト」を参照してください。

使用例

このコード例は、文字列をコンピュータの COM1 シリアル ポートに送る方法を示しています。

Using ブロックを使用すると、例外が生成される場合でもアプリケーションでシリアル ポートを閉じることができます。シリアル ポートを操作するコードは、このブロックまたは **Try...Catch...Finally** ブロック内に Close メソッドを使う呼び出しと共に記述する必要があります。

WriteLine メソッドは、データをシリアル ポートに送ります。

VB

```
Sub SendSerialData(ByVal data As String)
    ' Send strings to a serial port.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        com1.WriteLine(data)
    End Using
End Sub
```

詳細については、「方法 : Visual Basic でシリアル ポートに文字列を送信する」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**

Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト

My.Computer.Ports オブジェクト

My.Computer.Registry プロパティ

レジストリ操作用のプロパティとメソッドを提供するオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.MyServices.RegistryProxy = My.Computer.Registry
' Declaration
Public ReadOnly Property Registry As RegistryProxy
```

戻り値

このプロパティは、コンピュータの My.Computer.Registry オブジェクト を返します。

解説

このプロパティによって、My.Computer.Registry オブジェクト に簡単にアクセスできます。

使用例

この例では、`HKEY_CURRENT_USER\Software\MyApp` から値 `Name` を読み取り、メッセージ ボックスにこの値を表示します。

VB

```
Dim readValue As Object
        readValue = My.Computer.Registry.GetValue _
        ("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothing)
        MsgBox("The value is " & CStr(readValue))
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer、ServerComputer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer オブジェクト

My.Computer.Registry オブジェクト

概念

一般的なレジストリ タスク

My.Computer.Screen プロパティ

コンピュータのプライマリ ディスプレイ画面を表す Screen オブジェクトを取得します。

```
' Usage
Dim value As System.Windows.Forms.Screen = My.Computer.Screen
' Declaration
Public ReadOnly Property Screen As System.Windows.Forms.Screen
```

戻り値

コンピュータのプライマリ画面を表す **Screen** オブジェクト

解説

My.Computer.Screen プロパティは、PrimaryScreen プロパティと同様の機能を提供します。

使用例

この例では、**My.Computer.Screen** プロパティの WorkingArea プロパティを使用してコンピュータのプライマリ ディスプレイの作業領域を調べ、フォームのサイズを作業領域いっぱいに広げます。

VB

```
Private Sub EnlargeForm()
    Me.Size = My.Computer.Screen.WorkingArea.Size
    Me.Location = New System.Drawing.Point(0, 0)
End Sub
```

この例は Windows フォーム アプリケーション内で実行する必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Computer

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明

FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer オブジェクト
System.Windows.Forms.Screen
PrimaryScreen
Microsoft.VisualBasic.Devices.Computer.Screen

My.Computer.Audio オブジェクト

サウンドを再生するためのメソッドが用意されています。

解説

My.Computer.Audio.Play メソッドと **My.Computer.Audio.PlaySystemSound** メソッドを使って、.wav サウンド ファイルとシステム サウンドを再生できます。

処理手順

My.Computer.Audio オブジェクトに関連するタスクの例を次の表に示します。

タスク	参照項目
サウンドをバックグラウンドで再生します。	方法 : Visual Basic でサウンドを再生する
サウンドを一度再生し、完了するまで待ちます。	方法 : Visual Basic でサウンドを再生し、完了まで待機する
サウンドを何度も再生します。	方法 : Visual Basic でループ サウンドを再生する
バックグラウンドで再生中のサウンドを停止します。	方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する

使用例

My.Computer.Audio.Play メソッドは、**AudioPlayMode.Background** が指定されたときにバックグラウンドで指定されたサウンドを再生します。

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

このコード例は、Windows フォーム アプリケーション内だけで実行できます。

ファイル名は、使用しているシステム上の .wav サウンド ファイルを参照するものである必要があります。

サウンド ファイルの管理を容易にするためには、ファイルをアプリケーション リソースとして格納することをお勧めします。それにより、My.Resources オブジェクト を使ってファイルにアクセスできるようになります。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Audio

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可

Windows サービス	**可**
Web サイト	**不可**

参照

関連項目

My.Computer.Audio オブジェクトのメンバ

My.Computer オブジェクト

My.Computer.Audio オブジェクトのメンバ

My.Computer.Audio オブジェクト には、サウンドを再生するためのメソッドが用意されています。

メソッド

Play	.wav サウンド ファイルを再生します。 このメソッドは、非サーバー アプリケーションだけで使用できます。
PlaySystemSound	システム サウンドを再生します。 このメソッドは、非サーバー アプリケーションだけで使用できます。
Stop	バックグラウンドで再生中のサウンドを停止します。 このメソッドは、非サーバー アプリケーションだけで使用できます。

参照

関連項目

My.Computer.Audio オブジェクト

My.Computer.Audio.Play メソッド

.wav サウンド ファイルを再生します。

```
' Usage
My.Computer.Audio.Play(location)
My.Computer.Audio.Play(location ,playMode)
My.Computer.Audio.Play(data ,playMode)
My.Computer.Audio.Play(stream ,playMode)
' Declaration
Public Sub Play( _
   ByVal location As String _
)
' -or-
Public Sub Play( _
   ByVal location As String, _
   ByVal playMode As AudioPlayMode _
)
' -or-
Public Sub Play( _
   ByVal data As Byte(), _
   ByVal playMode As AudioPlayMode _
)
' -or-
Public Sub Play( _
   ByVal stream As System.IO.Stream, _
   ByVal playMode As AudioPlayMode _
)
```

パラメータ

location

サウンド ファイルの名前を含む **String**。

data

サウンド ファイルを表現する **Byte** 配列。

stream

サウンド ファイルを表現する Stream。

playMode

サウンドの再生モードである AudioPlayMode 列挙型。既定では **AudioPlayMode.Background** です。

例外

例外を引き起こす可能性のある状態を次に示します。

- *data* または *stream* が **Nothing** であるか、*location* が空の文字列です (ArgumentNullException)。
- *playMode* 引数が AudioPlayMode 列挙型 のいずれかの値ではありません (InvalidEnumArgumentException)。
- ユーザーに *location* で指定されたファイルにアクセスするための十分なアクセス許可がありません (IOException)。
- *location* に指定されたファイル パスの形式が誤っています (DirectoryNotFoundException)。
- *location* のパス名が長すぎます (PathTooLongException))。
- ユーザーが部分的に信頼されており、必要なアクセス許可を持っていません (SecurityException)。

解説

Play メソッドは .wav サウンド ファイルを再生します。このファイルは *location* にファイルとして、*data* にバイト配列として、または *stream* にストリームとして格納されています。

location パラメータのみを受け取るオーバーロードが使用された場合、**Play** メソッドはサウンドをバックグラウンドで再生します。それ以外で

は、*playMode* パラメータによってサウンドの再生方法が判断されます。

playMode	**説明**
AudioPlayMode.Background	サウンドをバックグラウンドで再生します。呼び出し元のコードは実行を継続します。
AudioPlayMode.BackgroundLoop	My.Computer.Audio.Stop メソッド が呼び出されるまで、サウンドをバックグラウンドで再生します。呼び出し元のコードは実行を継続します。
AudioPlayMode.WaitToComplete	サウンドを再生し、呼び出し元のコードは再生が終了するまで待機してから実行を継続します。

バックグラウンド再生を利用すると、サウンドの再生中にアプリケーションで他のコードを実行できます。詳細については、「方法 : Visual Basic でループ サウンドを再生する」および「方法 : Visual Basic でサウンドを再生する」を参照してください。

処理手順

My.Computer.Audio.Play メソッドに関連するタスクの例を次の表に示します。

目的	**参照項目**
サウンドを一度だけ再生する	方法 : Visual Basic でサウンドを再生し、完了まで待機する
サウンドを繰り返し再生する	方法 : Visual Basic でループ サウンドを再生する
サウンドをバックグラウンドで再生する	方法 : Visual Basic でサウンドを再生する

使用例

My.Computer.Audio.Play メソッドは、**PlayMode.Background** が指定されたときに、指定されたサウンドをバックグラウンドで再生します。

VB

```
Sub PlayBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.Background)
End Sub
```

このコード例は、Windows フォーム アプリケーション内だけで実行できます。

ファイル名は、使用中のシステムに置かれた .wav サウンド ファイルの名前であることが必要です。

サウンド ファイルを簡単に管理するために、アプリケーション リソースとして格納することをお勧めします。それにより、My.Resources オブジェクトを使ってファイルにアクセスできるようになります。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Audio

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	**使用可/不可**
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**

Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
SecurityPermission	コードに適用されたセキュリティ アクセス許可のセットを記述します。関連する列挙値 : ControlThread。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でサウンドを再生し、完了まで待機する
方法 : Visual Basic でループ サウンドを再生する
方法 : Visual Basic でサウンドを再生する

関連項目

My.Computer.Audio オブジェクト
My.Computer.Audio.Stop メソッド
AudioPlayMode 列挙型
System.IO.Stream
Microsoft.VisualBasic.Devices.Audio.Play(System.String)

My.Computer.Audio.PlaySystemSound メソッド

システム サウンドを再生します。

```
' Usage
My.Computer.Audio.PlaySystemSound(systemSound)
' Declaration
Public Sub PlaySystemSound( _
   ByVal systemSound As System.Media.SystemSound _
)
```

パラメータ

systemSound

再生するシステム サウンドを表す SystemSound オブジェクトです。

例外

例外を引き起こす可能性のある状態を次に示します。

- *systemSound* が **Nothing** です (ArgumentNullException)。

解説

PlaySystemSound メソッドは *systemSound* に指定されたシステム サウンドを再生します。サウンドはバックグラウンドで 1 回再生されます。

systemSound の値は SystemSounds クラスのいずれかの共有メンバであることが必要です。

- Asterisk
- Beep
- Exclamation
- Hand
- Question

処理手順

My.Computer.Audio.PlaySystemSound メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
システム サウンドを再生する	方法 : Visual Basic でシステム サウンドを再生する

使用例

次の例は、**My.Computer.Audio.PlaySystemSound** メソッドを使用して、システム サウンドを再生します。

VB

```
Sub PlaySystemSound()
    My.Computer.Audio.PlaySystemSound( _
        System.Media.SystemSounds.Asterisk)
End Sub
```

このコード例は、Windows フォーム アプリケーション内だけで実行できます。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Audio

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でシステム サウンドを再生する

関連項目

My.Computer.Audio オブジェクト
System.Media.SystemSound
SystemSounds
Microsoft.VisualBasic.Devices.Audio.PlaySystemSound(System.Media.SystemSound)

My.Computer.Audio.Stop メソッド

バックグラウンドでのサウンドの再生を停止します。

```
' Usage
My.Computer.Audio.Stop()
' Declaration
Public Sub Stop()
```

解説

My.Computer.Audio.Stop メソッドは、**My.Computer.Audio.Play** メソッドによって開始されたバックグラウンドでのサウンドの再生を停止します。

バックグラウンド再生を利用すると、サウンドの再生中にアプリケーションで他のコードを実行できます。詳細については、「方法 : Visual Basic でループ サウンドを再生する」および「方法 : Visual Basic でサウンドを再生する」を参照してください。

通常、アプリケーションがループ サウンドを再生したときと同じポイントで、サウンドを停止する必要があります。

処理手順

My.Computer.Audio.Stop メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
バックグラウンドでのサウンドの再生を停止する	方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する

使用例

次の例は、**My.Computer.Audio.Stop** メソッドを使用して現在バックグラウンド サウンドまたはループ サウンドを再生しているアプリケーションを停止し、**My.Computer.Audio.Play** メソッドを使用してループ サウンドをバックグラウンドで再生します。

VB

```
Sub PlayLoopingBackgroundSoundFile()
    My.Computer.Audio.Play("C:\Waterfall.wav", _
        AudioPlayMode.BackgroundLoop)
End Sub
Sub StopBackgroundSound()
    My.Computer.Audio.Stop()
End Sub
```

このコード例は、Windows フォーム アプリケーション内だけで実行できます。

ファイル名は、使用中のシステムに置かれた .wav サウンド ファイルの名前であることが必要です。

サウンド ファイルを簡単に管理するために、アプリケーション リソースとして格納することをお勧めします。それにより、My.Resources オブジェクトを使ってファイルにアクセスできるようになります。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Audio

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**

コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でバックグラウンドでのサウンドの再生を停止する

関連項目

My.Computer.Audio オブジェクト

Microsoft.VisualBasic.Devices.Audio.Stop

My.Computer.Clipboard オブジェクト

クリップボードを操作するためのメソッドを提供します。

解説

クリップボードに移動またはコピーされた項目は、アプリケーションが終了した後も保持されます。

クリップボードに置くデータは、どの形式のものでもかまいません。これはクリップボード形式とも呼ばれます。DataFormats を参照すると、クリップボードで使用できるよう定義済みである形式の一覧が確認できます。クリップボードに項目が移動またはコピーされると、それ以外の形式の項目が消去されます。形式の異なるデータが消去されないようにするには DataObject を使用します。これにより、他のアプリケーションから貼り付けられた項目も含め、その時点でクリップボードに存在するすべてのデータがコピーされます。貼り付け先のアプリケーションでどの形式を使用すればよいかわからないような場合でも、クリップボードに複数の形式でデータを置いておけば、データの貼り付けに失敗する可能性がかなり低くなります。

クリップボードのシステムは、すべての Windows アプリケーションで共有されます。したがって、別のアプリケーションに切り替えたとき、クリップボードの内容が変わる可能性があります。

クラスをクリップボードに置く場合は、クラスをシリアル化する必要があります。詳細については、「シリアル化」を参照してください。

リモートからクリップボードにアクセスした場合、そのスレッドが STA (Single-Threaded Apartment) モードで動作していなければ、ThreadStateException がスローされます。これを回避するには、*ThreadApartmentState* に `STA` を設定します。詳細については、「STAThreadAttribute」を参照してください。

処理手順

My.Computer.Clipboard オブジェクトに関連する操作の例を次の表に示します。

目的	参照項目
クリップボードを消去する	方法 : Visual Basic でクリップボードを消去する
クリップボードからデータを読み取る	方法 : Visual Basic でクリップボードから読み込む
クリップボードに格納されているデータを確認する	方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する
クリップボードにオーディオを保存する	方法 : Visual Basic でオーディオ ストリームをクリップボードに保存する
クリップボードからイメージを取得する	方法 : Visual Basic でクリップボードからイメージを取得する
クリップボードにデータを格納する	方法 : Visual Basic でクリップボードに書き込む

使用例

クリップボードからテキストを読み込んで文字列 `textOnClipboard` に格納する例は、次のようになります。

VB

```
Dim textOnClipboard As String = My.Computer.Clipboard.GetText()
```

この例では、クリップボードにテキストがなければエラーになります。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**

クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

参照

関連項目

My.Computer.Clipboard オブジェクトのメンバ
My.Computer オブジェクト
System.Windows.Forms.Clipboard

My.Computer.Clipboard オブジェクトのメンバ

My.Computer.Clipboard オブジェクト は、クリップボードの作業用のメソッドを提供します。

メソッド

メソッド	説明
Clear	クリップボードをクリアします。 このメソッドは、非サーバー アプリケーションだけで使用できます。
ContainsAudio	クリップボードにオーディオ データが含まれているかどうかを示します。 このメソッドは、非サーバー アプリケーションだけで使用できます。
ContainsData	クリップボードに、指定されたカスタム形式のデータが含まれているかどうかを示します。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
ContainsFileDropList	クリップボードに FileDropList が含まれるかどうかを示します。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
ContainsImage	クリップボードにイメージが含まれるかどうかを示します。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
ContainsText	クリップボードにテキストが含まれるかどうかを示します。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
GetAudioStream	クリップボードからオーディオ ストリームを取得します。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
GetData	クリップボードからデータを取得します。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
GetDataObject	クリップボードからデータ オブジェクトを取得します。このメソッドは、非サーバー アプリケーションでのみ使用できます。
GetFileDropList	クリップボードから FileDropList を取得します。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
GetImage	配列からイメージを取得します。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
GetText	クリップボードからテキストを取得します。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
SetAudio	クリップボードにオーディオ データを書き込みます。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
SetData	データを指定されたカスタム形式でクリップボードに書き込みます。 このメソッドは、非サーバー アプリケーションでのみ使用できます。

SetDataObject	クリップボードにデータ オブジェクトを書き込みます。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
SetFileDropList	クリップボードに FileFropList を書き込みます。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
SetImage	クリップボードにイメージを書き込みます。 このメソッドは、非サーバー アプリケーションでのみ使用できます。
SetText	クリップボードにテキストを書き込みます。 このメソッドは、非サーバー アプリケーションでのみ使用できます。

参照

処理手順

方法 : Visual Basic でクリップボードを消去する
方法 : Visual Basic でクリップボードから読み込む
方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する
方法 : Visual Basic でクリップボードに書き込む
方法 : Visual Basic でクリップボードからイメージを取得する
方法 : Visual Basic でオーディオ ストリームをクリップボードに保存する

関連項目

My.Computer.Clipboard オブジェクト

My.Computer.Clipboard.Clear メソッド

クリップボードを消去します。

```
' Usage
My.Computer.Clipboard.Clear()
' Declaration
Public Sub Clear()
```

解説

UIPermission クラスがクリップボードへのアクセスを制御し、関連付けられた UIPermissionClipboard 列挙値がアクセスのレベルを示します。詳細については、「Windows フォームのセキュリティに関するその他の考慮事項」を参照してください。

処理手順

My.Computer.Clipboard.Clear メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
クリップボードを消去する	方法 : Visual Basic でクリップボードを消去する

使用例

クリップボードを消去する例は次のようになります。

```
My.Computer.Clipboard.Clear()
```

クリップボードからすべてのデータが消去されます。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でクリップボードを消去する

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard オブジェクトのメンバ
System.Windows.Forms.Clipboard.Clear

My.Computer.Clipboard.ContainsAudio メソッド

クリップボードにオーディオ データが含まれているかどうかを示します。

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsAudio()
' Declaration
Public Function ContainsAudio() As Boolean
```

戻り値

オーディオ データがクリップボードに格納されていれば **True**、それ以外の場合は **False** です。

解説
処理手順

My.Computer.Clipboard.ContainsAudio メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
クリップボード内にあるデータの種類を確認します。	方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する

使用例

この例でぇあ、クリップボードにオーディオ データが含まれているかどうかを確認し、結果を表示します。

VB

```
If My.Computer.Clipboard.ContainsAudio() Then
   MsgBox("The clipboard contains audio data.")
Else
   MsgBox("The clipboard does not contain audio data.")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス ClipboardProxy (Clipboard へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する

関連項目

My.Computer.Clipboard オブジェクト

My.Computer.Clipboard.GetAudioStream メソッド

My.Computer.Clipboard.SetAudio メソッド

System.Windows.Forms.Clipboard.ContainsAudio

その他の技術情報

サウンドの再生

My.Computer.Clipboard.ContainsData メソッド

クリップボードに、指定されたカスタム形式のデータが含まれているかどうかを示します。

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsData(format)
' Declaration
Public Function ContainsData( _
   ByVal format As String _
) As Boolean
```

パラメータ

format

String です。チェックするカスタム形式の名前です。必ず指定します。

戻り値

指定されたカスタム形式のデータがクリップボードに含まれていれば **True**、それ以外の場合は **False** です。

例外

このメソッドは例外をスローしません。

解説

カスタム形式のデータがクリップボード内にある場合、このメソッドを使用して、クリップボードにあるその形式のデータをチェックできます。

処理手順

My.Computer.Clipboard.ContainsData メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
クリップボード内にあるデータの種類を確認します。	方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する

使用例

この例では、カスタム形式 `specialFormat` のデータをチェックします。

VB

```
If My.Computer.Clipboard.ContainsData("specialFormat") Then
   MsgBox("Data found.")
End If
```

`specialFormat` の部分を、チェックするカスタム形式の名前に置換してください。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス ClipboardProxy (Clipboard へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**

Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.GetData メソッド
My.Computer.Clipboard.GetDataObject メソッド
My.Computer.Clipboard.SetData メソッド
My.Computer.Clipboard.SetDataObject メソッド
ContainsData

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.ContainsFileDropList メソッド

クリップボードにファイル ドロップ リストが含まれるかどうかを示す **Boolean** を返します。

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsFileDropList()
' Declaration
Public Function ContainsFileDropList() As Boolean
```

戻り値

ファイル ドロップ リストがクリップボードに格納されていれば **True**、それ以外の場合は **False** です。

解説

ファイル ドロップダウン リストは、ファイルのパス情報を含む文字列のコレクションです。

使用例

この例では、クリップボードにファイル ドロップダウン リストがあるかどうかを確認し、ある場合はそのリストを `lstFiles` という **ListBox** に追加します。

VB

```
If My.Computer.Clipboard.ContainsFileDropList Then
   Dim filelist As System.Collections.Specialized.StringCollection
   filelist = My.Computer.Clipboard.GetFileDropList()
   For Each filePath As String In filelist
       lstFiles.Items.Add(filePath)
   Next
End If
```

このコードでは、`lstFiles` という名前の **ListBox** がない場合に例外を作成します。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス ClipboardProxy (Clipboard へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト

My.Computer.Clipboard.GetFileDropList メソッド

My.Computer.Clipboard.SetFileDropList メソッド

System.Windows.Forms.Clipboard.ContainsFileDropList

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.ContainsImage メソッド

イメージがクリップボードに格納されているかどうかを示す **Boolean** を返します。

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsImage()
' Declaration
Public Function ContainsImage() As Boolean
```

戻り値

イメージがクリップボードに格納されていれば **True**、それ以外の場合は **False** です。

例外

このメソッドは例外をスローしません。

解説

このメソッドは AllClipboard を必要とします。

処理手順

My.Computer.Clipboard.ContainsImage メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
クリップボードにイメージが含まれるかどうかを示します。	方法 : Visual Basic でクリップボードからイメージを取得する

使用例

この例では、クリップボードにイメージがあるかどうかを確認し、ある場合はイメージを取得して `PictureBox1` に追加します。

VB

```
If My.Computer.Clipboard.ContainsImage() Then
   Dim grabpicture As System.Drawing.Image
   grabpicture = My.Computer.Clipboard.GetImage()
   PictureBox1.Image = grabpicture
End If
```

この例は、`PictureBox1` という名前の **PictureBox** があることを前提としています。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス ClipboardProxy (Clipboard へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし

Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.GetImage メソッド
My.Computer.Clipboard.SetImage メソッド
System.Windows.Forms.Clipboard.ContainsImage

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.ContainsText メソッド

クリップボード内にテキストがあるかどうかを確認します。

```
' Usage
Dim value As Boolean = My.Computer.Clipboard.ContainsText()
Dim value As Boolean = My.Computer.Clipboard.ContainsText(format)
' Declaration
Public Function ContainsText() As Boolean
' -or-
Public Function ContainsText( _
   ByVal format As System.Windows.Forms.TextDataFormat _
) As Boolean
```

パラメータ

format

TextDataFormat です。指定されている場合、チェックするテキスト形式を識別します。必ず指定します。

戻り値

クリップボードに読み取り専用のテキストが含まれている場合は **True**、それ以外の場合は **False** です。

解説

使用可能な形式は、CommaSeparatedValue、Html、Rtf、および UnicodeText です。

このメソッドは AllClipboard を必要とします。

処理手順

My.Computer.Clipboard.ContainsText メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
クリップボードにテキストが含まれるかどうかを示します。	方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する

使用例

この例では、HTML テキストがクリップボードに格納されているかどうかを確認し、格納されている場合はこれを読み出します。

VB

```
If My.Computer.Clipboard.ContainsText _
(System.Windows.Forms.TextDataFormat.Html) Then
   Dim clipText As String = My.Computer.Clipboard.GetText()
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス ClipboardProxy (Clipboard へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**

Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.GetText メソッド
My.Computer.Clipboard.SetText メソッド
System.Windows.Forms.TextDataFormat
System.Windows.Forms.Clipboard.ContainsText

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.GetAudioStream メソッド

クリップボードからオーディオ ストリームを取得します。

```
' Usage
Dim value As System.IO.Stream = My.Computer.Clipboard.GetAudioStream()
' Declaration
Public Function GetAudioStream() As System.IO.Stream
```

戻り値

Stream

解説
使用例

次の例では、クリップボードからオーディオ ストリームを取得し、再生します。

VB

```
If My.Computer.Clipboard.ContainsAudio Then
   Dim song As System.IO.Stream
   song = My.Computer.Clipboard.GetAudioStream
   My.Computer.Audio.Play(song, AudioPlayMode.WaitToComplete)
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス ClipboardProxy (Clipboard へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照
関連項目
My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.ContainsAudio メソッド
My.Computer.Clipboard.SetAudio メソッド
System.IO.Stream

System.Windows.Forms.Clipboard.GetAudioStream

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.GetData メソッド

カスタム形式のデータをクリップボードから取得します。

```
' Usage
Dim value As Object = My.Computer.Clipboard.GetData(format)
' Declaration
Public Function GetData( _
   ByVal format As String _
) As Object
```

パラメータ

format

String です。データ形式の名前です。必ず指定します。

戻り値

オブジェクト

解説

使用例

この例では、`specialformat` という形式のデータをクリップボードから読み込み、ファイルに書き込みます。

VB

```
Dim someData As Object
someData = My.Computer.Clipboard.GetData("specialformat")
My.Computer.FileSystem.WriteAllBytes("C:\\mylogfile", someData, True)
```

`specialformat` の部分を、取得するカスタム形式の名前で置き換えます。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス ClipboardProxy (Clipboard へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.ContainsData メソッド
My.Computer.Clipboard.GetDataObject メソッド
My.Computer.Clipboard.SetData メソッド
My.Computer.Clipboard.SetDataObject メソッド
GetData

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.GetDataObject メソッド

クリップボードからデータを IDataObject として取得します。

```
' Usage
Dim value As System.Windows.Forms.IDataObject = My.Computer.Clipboard.GetDataObject()
' Declaration
Public Function GetDataObject() As System.Windows.Forms.IDataObject
```

戻り値

IDataObject

解説

これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

使用例

次の例は、クリップボードからデータを **IDataObject** の形式で読み取り、ファイルに書き込みます。

VB

```
Dim someData As Object
someData = My.Computer.Clipboard.GetDataObject()
My.Computer.FileSystem.WriteAllBytes("C:\mylogfile", someData, True)
```

`C:\mylogfile` の部分を、書き込みを行うファイルの名前に置換してください。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト

My.Computer.Clipboard.ContainsData メソッド

My.Computer.Clipboard.GetData メソッド
My.Computer.Clipboard.SetData メソッド
My.Computer.Clipboard.SetDataObject メソッド
System.Windows.Forms.IDataObject
System.Windows.Forms.Clipboard.GetDataObject

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.GetFileDropList メソッド

ファイル名を表す文字列のコレクションをクリップボードから取得します。

```
' Usage
Dim value As System.Collections.Specialized.StringCollection = My.Computer.Clipboard.GetFil
eDropList()
' Declaration
Public Function GetFileDropList() As System.Collections.Specialized.StringCollection
```

戻り値

StringCollection

解説

このメソッドは、非サーバー アプリケーションでのみ使用できます。

ファイル ドロップダウン リストは、ファイルのパス情報を含む文字列のコレクションです。

使用例

この例では、クリップボードにファイル ドロップダウン リストがあるかどうかを確認し、ある場合はそのリストを `lstFiles` という **ListBox** に追加します。

VB

```
If My.Computer.Clipboard.ContainsFileDropList Then
   Dim filelist As System.Collections.Specialized.StringCollection
   filelist = My.Computer.Clipboard.GetFileDropList()
   For Each filePath As String In filelist
       lstFiles.Items.Add(filePath)
   Next
End If
```

このコードでは、`lstFiles` という名前の **ListBox** がない場合に例外を作成します。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス ClipboardProxy (Clipboard へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.ContainsFileDropList メソッド
My.Computer.Clipboard.SetFileDropList メソッド
System.Collections.Specialized.StringCollection
System.Windows.Forms.Clipboard.GetFileDropList

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.GetImage メソッド

クリップボードからイメージを取得します。

```
' Usage
Dim value As System.Drawing.Image = My.Computer.Clipboard.GetImage()
' Declaration
Public Function GetImage() As System.Drawing.Image
```

戻り値

Image

解説

GetImage メソッドは、クリップボードにイメージであるデータが存在しない場合は **Nothing** を返します。

このメソッドは AllClipboard を必要とします。

処理手順

My.Computer.Clipboard.GetImage メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
クリップボードからイメージを取得する	方法 : Visual Basic でクリップボードからイメージを取得する

使用例

次の例は、クリップボードにイメージが存在するかどうかを確認してからイメージを取得し、 `PictureBox1` に代入します。

VB

```
If My.Computer.Clipboard.ContainsImage() Then
   Dim grabpicture As System.Drawing.Image
   grabpicture = My.Computer.Clipboard.GetImage()
   PictureBox1.Image = grabpicture
End If
```

この例が正しく動作するためには、`PictureBox1` という名前の **PictureBox** が、フォームに存在している必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**

Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : クリップボードに格納されているファイルの種類を Visual Basic で判断する

関連項目

My.Computer.Clipboard オブジェクト

My.Computer.Clipboard.ContainsImage メソッド

My.Computer.Clipboard.SetImage メソッド

System.Drawing.Image

System.Windows.Forms.Clipboard.GetImage

My.Computer.Clipboard.GetText メソッド

クリップボードからテキストを取得します。

```
' Usage
Dim value As String = My.Computer.Clipboard.GetText()
Dim value As String = My.Computer.Clipboard.GetText(format)
' Declaration
Public Function GetText() As String
' -or-
Public Function GetText( _
   ByVal format As System.Windows.Forms.TextDataFormat _
) As String
```

パラメータ

format

取得するテキストの形式を TextDataFormat 型で指定します。既定値は CommaSeparatedValue です。必ず指定します。

戻り値

String.

例外

このメソッドでスローされる例外はありません。

解説

指定可能な形式は **CommaSeparatedValue**、Html、Rtf、および UnicodeText です。

このメソッドは AllClipboard を必要とします。

指定された形式に一致するデータがクリップボードになければ、空の文字列が返されます。

処理手順

My.Computer.Clipboard.GetText メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
クリップボードからテキストを読み取る	方法 : Visual Basic でクリップボードから読み込む

使用例

クリップボードからテキストを読み込んで文字列 `textOnClipboard` に格納する例は、次のようになります。

VB

```
Dim textOnClipboard As String = My.Computer.Clipboard.GetText()
```

この例では、クリップボードにテキストがなければエラーになります。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**

クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でクリップボードから読み込む

関連項目

My.Computer.Clipboard オブジェクト

My.Computer.Clipboard.ContainsText メソッド

My.Computer.Clipboard.SetText メソッド

System.Windows.Forms.TextDataFormat

System.Windows.Forms.Clipboard.GetText

My.Computer.Clipboard.SetAudio メソッド

オーディオ データをクリップボードに書き込みます。

```
' Usage
My.Computer.Clipboard.SetAudio(audioBytes)
My.Computer.Clipboard.SetAudio(audioStream)
' Declaration
Public Sub SetAudio( _
   ByVal audioBytes As Byte() _
)
' -or-
Public Sub SetAudio( _
   ByVal audioStream As System.IO.Stream _
)
```

パラメータ

audioBytes

Byte 配列。必ず指定します。クリップボードに書き込むオーディオ データです。

audioStream

必ず指定します。クリップボードに書き込む Stream 型のオーディオ データです。

例外

このメソッドでスローされる例外はありません。

解説

このメソッドを使用すると、オーディオ データをバイト配列またはオーディオ ストリームとして書き込むことができます。

セキュリティに関するメモ :
クリップボードには他のユーザーからもアクセス可能なので、クリップボードを使ってパスワードや極秘データなどの機密情報を格納しないでください。

処理手順

My.Computer.Clipboard.SetAudio メソッドに関連するタスクの例を次の表に示します。

操作	参照項目
オーディオ データをクリップボードに書き込む	方法 : Visual Basic でオーディオ ストリームをクリップボードに保存する

使用例

次の例は、バイト配列 `musicReader` を作成し、ファイル `cool.wav` をそのバイト配列に読み込んでから、クリップボードに書き出します。

VB

```
Dim musicReader As Byte()
musicReader = My.Computer.FileSystem.ReadAllBytes("cool.wav")
My.Computer.Clipboard.SetAudio(musicReader)
```

`cool.wav` の名前とパスを、自分が読み込みたいファイルの名前とパスに置き換えてください。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.ContainsAudio メソッド
My.Computer.Clipboard.GetAudioStream メソッド
System.IO.Stream
System.Windows.Forms.Clipboard.SetAudio

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.SetData メソッド

データをカスタム形式でクリップボードに書き込みます。

```
' Usage
My.Computer.Clipboard.SetData(format ,data)
' Declaration
Public Sub SetData( _
   ByVal format As String, _
   ByVal data As Object _
)
```

パラメータ

format

必ず指定します。データの形式を指定する文字列 (**String**) です。

data

必ず指定します。クリップボードに書き込むデータ オブジェクト (**Object**) です。

例外

このメソッドでスローされる例外はありません。

解説

セキュリティに関するメモ :
クリップボードには他のユーザーからもアクセス可能なので、クリップボードを使ってパスワードや極秘データなどの機密情報を格納しないでください。

使用例

次の例は、**DataObject** `dataChunk` を、カスタム形式 `specialFormat` でクリップボードに書き込みます。

VB

```
My.Computer.Clipboard.SetData("specialFormat", dataChunk)
```

この例を実行するには、カスタム データ形式 `specialFormat` が存在している必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし

Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.ContainsData メソッド
My.Computer.Clipboard.GetData メソッド
My.Computer.Clipboard.GetDataObject メソッド
My.Computer.Clipboard.SetDataObject メソッド
SetData

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.SetDataObject メソッド

DataObject をクリップボードに書き込みます。

```
' Usage
My.Computer.Clipboard.SetDataObject(data)
' Declaration
Public Sub SetDataObject( _
   ByVal data As System.Windows.Forms.DataObject _
)
```

パラメータ

data

必ず指定します。クリップボードに書き込むデータ オブジェクト (**DataObject**) です。

解説

これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

セキュリティに関するメモ :
クリップボードには他のユーザーからもアクセス可能なので、クリップボードを使ってパスワードや極秘データなどの機密情報を格納しないでください。

使用例

データ オブジェクト `dataChunk` をクリップボードに書き込む例は次のとおりです。

VB

```
My.Computer.Clipboard.SetDataObject(dataChunk)
```

この例を実行するには、データ オブジェクト `dataChunk` が存在している必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.ContainsData メソッド
My.Computer.Clipboard.GetData メソッド
My.Computer.Clipboard.GetDataObject メソッド
My.Computer.Clipboard.SetData メソッド
SetDataObject

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.SetFileDropList メソッド

ファイルのパスを表す文字列のコレクションを、クリップボードに書き込みます。

```
' Usage
My.Computer.Clipboard.SetFileDropList(filePaths)
' Declaration
Public Sub SetFileDropList( _
   ByVal filePaths As System.Collections.Specialized.StringCollection _
)
```

パラメータ

filePaths

必ず指定します。ファイル名のリストを指定する StringCollection です。

解説

ファイル ドロップダウン リストは、ファイル名を表す文字列のコレクションです。

セキュリティに関するメモ：

クリップボードには他のユーザーからもアクセス可能なので、クリップボードを使ってパスワードや極秘データなどの機密情報を格納しないでください。

使用例

次の例は、[マイ ドキュメント] からファイル名のコレクションを取得し、ファイル ドロップダウン リストに変換してから、クリップボードに書き込みます。

VB

```
Dim list As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
list = My.Computer.FileSystem.GetFiles _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments)
Dim listReader As New System.Collections.Specialized.StringCollection
For Each item As String In list
   listReader.Add(item)
Next
My.Computer.Clipboard.SetFileDropList(listReader)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし

Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.ContainsFileDropList メソッド
My.Computer.Clipboard.GetFileDropList メソッド
System.Collections.Specialized.StringCollection
SetFileDropList

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.SetImage メソッド

イメージをクリップボードに書き込みます。

```
' Usage
My.Computer.Clipboard.SetImage(image)
' Declaration
Public Sub SetImage( _
   ByVal image As System.Drawing.Image _
)
```

パラメータ

image

必ず指定します。書き込むイメージを指定する Image です。

解説

セキュリティに関するメモ：
クリップボードには他のユーザーからもアクセス可能なので、クリップボードを使ってパスワードや極秘データなどの機密情報を格納しないでください。

使用例

イメージ `coolPicture` をクリップボードに書き込む例は次のとおりです。

VB

```
My.Computer.Clipboard.SetImage(coolPicture)
```

この例を実行するには、イメージ データ `coolPicture` が存在している必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でクリップボードからイメージを取得する

関連項目

My.Computer.Clipboard オブジェクト

My.Computer.Clipboard.ContainsImage メソッド

My.Computer.Clipboard.GetImage メソッド

System.Drawing.Image

System.Windows.Forms.Clipboard.SetImage(System.Drawing.Image)

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clipboard.SetText メソッド

テキストをクリップボードに書き込みます。

```
' Usage
My.Computer.Clipboard.SetText(text)
My.Computer.Clipboard.SetText(text ,format)
' Declaration
Public Sub SetText( _
   ByVal text As String _
)
' -or-
Public Sub SetText( _
   ByVal text As String, _
   ByVal format As System.Windows.Forms.TextDataFormat _
)
```

パラメータ

text

必ず指定します。書き込むテキストを指定する文字列 **String** です。

format

テキストを書き込むときに使う書式を指定する TextDataFormat です。既定値は UnicodeText です。必ず指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- *text* が空の文字列です (ArgumentException)。
- *text* が **Nothing** です (ArgumentNullException)。

解説

指定可能な書式は CommaSeparatedValue、Html、Rtf、および **UnicodeText** です。

このメソッドは AllClipboard を必要とします。

前回指定したクリップボードの書式は保持されません。

セキュリティに関するメモ：
クリップボードには他のユーザーからもアクセス可能なので、クリップボードを使ってパスワードや極秘データなどの機密情報を格納しないでください。

処理手順

My.Computer.Clipboard.SetText メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
テキストをクリップボードに書き込みます。	方法 : Visual Basic でクリップボードに書き込む

使用例

文字列 `This is a test string.` をクリップボードに書き込む例は次のとおりです。

VB

```
My.Computer.Clipboard.SetText("This is a test string.")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : ClipboardProxy (Clipboard へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	なし
Windows サービス	**あり**
Web サイト	なし

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clipboard オブジェクト
My.Computer.Clipboard.ContainsText メソッド
My.Computer.Clipboard.GetText メソッド
System.Windows.Forms.TextDataFormat
System.Windows.Forms.Clipboard.SetText

その他の技術情報

クリップボードのデータの格納と読み込み

My.Computer.Clock オブジェクト

現在の現地時刻および世界協定時刻 (グリニッジ標準時に等しい) にシステム時計からアクセスするためのプロパティを提供します。

解説

My.Computer.Clock オブジェクトには、コンピュータの現在の現地時刻と世界協定時刻を取得するプロパティがあります。また、コンピュータのシステム タイマからミリ秒単位のカウントを公開します。

使用例

次のコード例は、**My.Computer.Clock.LocalTime** プロパティを使って、このコードが実行されるコンピュータの現地時刻を表示します。

VB

```
MsgBox("Current local time: " & My.Computer.Clock.LocalTime)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Clock

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照

関連項目

My.Computer.Clock オブジェクトのメンバ
My.Computer オブジェクト
Microsoft.VisualBasic.Devices.Clock

My.Computer.Clock オブジェクトのメンバ

My.Computer.Clock オブジェクトには、現在の現地時刻および世界協定時刻 (グリニッジ標準時に等しい) にシステム時計からアクセスするためのプロパティが用意されています。

プロパティ

プロパティ	説明
GmtTime	世界協定時刻 (GMT) で表現された、コンピュータの現在の現地時刻を含む **Date** オブジェクトを取得します。
LocalTime	コンピュータの現在の現地時刻を含む **Date** オブジェクトを取得します。
TickCount	コンピュータのシステム タイマからミリ秒単位のカウントを取得します。

参照

関連項目

My.Computer.Clock オブジェクト

My.Computer.Clock.GmtTime プロパティ

コンピュータにおける現在のローカルな日付と時刻 (UTC (GMT) 時で表現) を格納した **Date** オブジェクトを取得します。

```
' Usage
Dim value As Date = My.Computer.Clock.GmtTime
' Declaration
Public ReadOnly Property GmtTime As Date
```

戻り値

UTC (GMT) 時で表現された、現在の日付と時刻を格納した **Date** オブジェクト。

解説

My.Computer.Clock.GmtTime プロパティと System.DateTime.UtcNow プロパティの動作は同じです。

使用例

次の例は、**My.Computer.Clock.GmtTime** プロパティを使用して、コードを実行しているコンピュータの GMT 時刻を表示します。

VB

```
MsgBox("Current GMT time: " & My.Computer.Clock.GmtTime)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Clock

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clock オブジェクト
My.Computer.Clock.LocalTime プロパティ
System.DateTime.UtcNow
Microsoft.VisualBasic.Devices.Clock.GmtTime

My.Computer.Clock.LocalTime プロパティ

このコンピュータにおける現在のローカルな日付と時刻を格納した **Date** オブジェクトを取得します。

```
' Usage
Dim value As Date = My.Computer.Clock.LocalTime
' Declaration
Public ReadOnly Property LocalTime As Date
```

戻り値

現在のローカルな日付と時刻を格納した **Date** オブジェクト。

解説

My.Computer.Clock.LocalTime プロパティと System.DateTime.Now プロパティの動作は同じです。

使用例

次の例は、**My.Computer.Clock.LocalTime** プロパティを使用して、このコードを実行しているコンピュータのローカル時刻を表示します。

VB

```
MsgBox("Current local time: " & My.Computer.Clock.LocalTime)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Clock

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clock オブジェクト
My.Computer.Clock.GmtTime プロパティ
System.DateTime.Now
Microsoft.VisualBasic.Devices.Clock.LocalTime

My.Computer.Clock.TickCount プロパティ

コンピュータのシステム タイマからミリ秒のカウントを取得します。

```
' Usage
Dim value As Integer = My.Computer.Clock.TickCount
' Declaration
Public ReadOnly Property TickCount As Integer
```

戻り値

コンピュータのシステム タイマから取得したミリ秒のカウントを格納する **Integer** です。

解説

TickCount プロパティを使用すると、コンピュータがアクティブになったときに実行されるシステム タイマにアクセスできます。このタイマの解像力は 500 ミリ秒以上です。

このプロパティを使うことによって、アプリケーションが稼動している時間の長さに応じてその動作を変更したり、イベントにラベルを付けたりすることが可能です。また、このどちらもコンピュータのクロックの影響を受けません。

注意

TickCount プロパティの値が整数の最大値 (MaxValue) に達すると、負の数である整数の最小値 (MinValue) にジャンプし、そこから続けてインクリメントします。

コンピュータが継続して動作している場合、**TickCount** が 0 から整数の最大値にインクリメントされるまでには、およそ 24.9 日間かかります。

TickCount プロパティがインクリメントするのは、オペレーティング システムが稼動しているときだけです。コンピュータがスタンバイや休止状態など、特定の省電力モードに入るとインクリメントは停止します。**TickCount** プロパティは、コンピュータのクロック設定とは関係ありません。

このコンピュータにおける現在のローカルな日付と時刻を取得するには、My.Computer.Clock.LocalTime プロパティまたは My.Computer.Clock.GmtTime プロパティを取得します。

My.Computer.Clock.TickCount プロパティと System.Environment.TickCount プロパティの動作は同じです。

使用例

次の例は **My.Computer.Clock.TickCount** プロパティを使用して、ループ内のタスクを指定された秒数だけ実行します。実行中にコンピュータのシステム時刻が変更されても動作に影響はありません。

VB

```
Public Sub LoopTask(ByVal secondsToRun As Integer)
    Dim startTicks As Integer = My.Computer.Clock.TickCount
    Do While IsTimeUp(startTicks, secondsToRun)
        ' Code to run for at least secondsToRun seconds goes here.
    Loop
End Sub

Private Function IsTimeUp( _
    ByVal startTicks As Integer, _
    ByVal seconds As Integer _
) As Boolean
    ' This function throws an overflow exception if the
    ' tick count difference is greater than 2,147,483,647,
    ' about 24 days for My.Computer.Clock.TickCount.

    ' Use UInteger to simplify the code for roll over.
    Dim uStart As UInteger = _
        CUInt(CLng(startTicks) - Integer.MinValue)
    Dim uCurrent As UInteger = _
        CUInt(CLng(My.Computer.Clock.TickCount) - Integer.MinValue)

    ' Calculate the tick count difference.
```

```
        Dim tickCountDifference As UInteger
        If uStart <= uCurrent Then
            tickCountDifference = uCurrent - uStart
        Else
            ' Tick count rolled over.
            tickCountDifference = UInteger.MaxValue - (uStart - uCurrent)
        End If

        ' Convert seconds to milliseconds and compare.
        Return CInt(tickCountDifference) < (seconds * 1000)
    End Function
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Clock

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Clock オブジェクト
System.Environment.TickCount
Microsoft.VisualBasic.Devices.Clock.TickCount

My.Computer.FileSystem オブジェクト

ドライブ、ファイル、およびディレクトリを操作するためのプロパティとメソッドを提供します。

処理手順

My.Computer.FileSystem オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
テキスト ファイルから読み取る	方法 : Visual Basic でテキスト ファイルを読み取る
区切り記号で区切られたテキスト ファイルから読み込む	方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
固定幅のテキスト ファイルから読み込む	方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
複数の形式を持つテキスト ファイルから読み込む	方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る
バイナリ ファイルから読み取る	方法 : Visual Basic でバイナリ ファイルを読み取る
マイ ドキュメント ディレクトリに含まれるテキスト ファイルから読み取る	方法 : My Documents の既存のテキスト ファイルを読み取る (Visual Basic)
StreamReader を使用してテキスト ファイルから読み取る	方法 : StreamReader を使用してファイルからテキストを読み取る (Visual Basic)
テキスト ファイルへ書き込む	方法 : Visual Basic でテキストをファイルに書き込む
テキスト ファイルへ追加する	方法 : Visual Basic でテキスト ファイルに追記する
バイナリ ファイルへ書き込む	方法 : Visual Basic でバイナリ ファイルに書き込む
マイ ドキュメント ディレクトリにあるテキスト ファイルへ書き込む	方法 : Visual Basic で My Documents ディレクトリのファイルにテキストを書き込む
StreamWriter を使用してテキスト ファイルに書き込む	方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む
特定のパターンを持つファイルをコピーする	方法 : Visual Basic で特定のパターンを持つファイルをディレクトリにコピーする
同じディレクトリへファイルをコピーする	方法 : Visual Basic でファイルのコピーを同じディレクトリに作成する
別のディレクトリへファイルをコピーする	方法 : Visual Basic でファイルのコピーを別のディレクトリに作成する
ファイルを作成する	方法 : Visual Basic でファイルを作成する
ファイルを削除する	方法 : Visual Basic でファイルを削除する
ディレクトリ内のすべてのファイルを削除する	方法 : Visual Basic でディレクトリ内のすべてのファイルを削除する
特定のパターンを持つファイルを検索する	方法 : Visual Basic で特定のパターンに一致するファイルを検索する
ファイルを移動する	方法 : Visual Basic でファイルを移動する
ファイルのコレクションを移動する	方法 : Visual Basic でファイルのコレクションを移動する
ファイル名を変更する	方法 : Visual Basic でファイルの名前を変更する

ディレクトリ名を変更する	方法 : Visual Basic でディレクトリの名前を変更する
他のディレクトリへディレクトリをコピーする	方法 : Visual Basic でディレクトリを別のディレクトリにコピーする
ディレクトリを作成する	方法 : Visual Basic でディレクトリを作成する
ディレクトリを削除する	方法 : Visual Basic でディレクトリを削除する
特定のパターンを持つサブディレクトリを検索する	方法 : Visual Basic で特定のパターンに一致するサブディレクトリを検索する
ディレクトリ内のファイルのコレクションを取得する	方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する
ディレクトリに含まれるファイルの数を調べる	方法 : ディレクトリに含まれているファイルの数を確認する
ディレクトリを移動する	方法 : Visual Basic でディレクトリを移動する
ディレクトリのコンテンツを移動する	方法 : Visual Basic でディレクトリの内容を移動する
マイ ドキュメント ディレクトリから読み取る	方法 : Visual Basic で My Documents ディレクトリの内容を取得する
ファイル パスを解析する	方法 : Visual Basic でファイル パスを解析する

使用例

次の例は、 `C:\backup\logs` フォルダが存在するかどうかを判断し、そのプロパティを調べます。

VB

```
Dim logInfo As System.IO.DirectoryInfo
If My.Computer.FileSystem.DirectoryExists("C:\backup\logs") Then
    logInfo = My.Computer.FileSystem.GetDirectoryInfo _
    ("C:\backup\logs")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

参照

関連項目

My.Computer.FileSystem オブジェクトのメンバ
My.Computer.FileSystem.SpecialDirectories オブジェクト
My.Computer オブジェクト
Microsoft.VisualBasic.FileIO.FileSystem

My.Computer.FileSystem オブジェクトのメンバ

My.Computer.FileSystem オブジェクトには、ドライブ、ファイル、およびディレクトリを操作するためのプロパティおよびメソッドが用意されています。

プロパティ

プロパティ	説明
CurrentDirectory	現在のディレクトリを取得します。
Drives	ドライブに関する情報を取得します。
SpecialDirectories	My.Computer.FileSystem.SpecialDirectories オブジェクト を返します。このオブジェクトは、Temp または MyDocuments などの特殊なディレクトリにアクセスするために使用できます。

メソッド

メソッド	説明
CombinePath	適切な形式で連結されたパスを **String** として返します。
CopyDirectory	ディレクトリをコピーします。
CopyFile	ファイルをコピーします。
CreateDirectory	ディレクトリを作成します。
DeleteDirectory	ディレクトリを削除します。
DeleteFile	ファイルを削除します。
DirectoryExists	ディレクトリが存在するかどうかを示す **Boolean** を返します。
FileExists	ファイルが存在するかどうかを示す **Boolean** を返します。
My.Computer.FileSystem.FindInFiles メソッド	指定されたテキストが含まれるファイル名を読み取り専用の文字列コレクションとして返します。
GetDirectories	ディレクトリ内にあるサブディレクトリのパス名を **String** コレクションとして返します。
GetDirectoryInfo	指定されたパスの DirectoryInfo オブジェクトを返します。
GetDriveInfo	指定されたパスの DriveInfo オブジェクトを返します。
GetFileInfo	指定されたパスの FileInfo オブジェクトを返します。
GetFiles	ディレクトリ内にあるファイルの名前を読み取り専用の **String** コレクションとして返します。
GetParentPath	指定されたパスの親の絶対パスを **String** として返します。
MoveDirectory	ディレクトリを移動します。
MoveFile	ファイルを移動します。
OpenTextFieldParser	TextFieldParser を開きます。

OpenTextFileReader	TextReader を開きます。
OpenTextFileWriter	TextWriter を開きます。
ReadAllBytes	バイナリ ファイルの内容を読み取ります。
ReadAllText	テキスト ファイルの内容を読み取ります。
RenameDirectory	ディレクトリの名前を変更します。
RenameFile	ファイルの名前を変更します。
WriteAllBytes	バイナリ ファイルに書き込みます。
WriteAllText	テキスト ファイルに書き込みます。

参照

処理手順

チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

My.Computer.FileSystem オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

その他の技術情報

Visual Basic でのファイルの読み取り

Visual Basic でのファイルへの書き込み

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.FileSystem.CombinePath メソッド

2 つのパスを結合して、正しく書式化されたパスを返します。

```
' Usage
Dim value As String = My.Computer.FileSystem.CombinePath(baseDirectory ,relativePath)
' Declaration
Public Function CombinePath( _
   ByVal baseDirectory As String, _
   ByVal relativePath As String _
) As String
```

パラメータ

baseDirectory

結合する 1 つ目のパスを指定する **String** です。必ず指定します。

relativePath

結合する 2 つ目のパスを指定する **String** です。必ず指定します。

戻り値

String.

例外

例外を引き起こす可能性のある状態を次に示します。

- パスの形式が誤っています (ArgumentException)。

解説

このメソッドは余分なスラッシュ記号の文字をトリムして、正しく書式化されたパスを作成します。

処理手順

My.Computer.FileSystem.CombinePath メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ディレクトリ パスとファイル名を結合する	方法 : Visual Basic でファイル パスを解析する

使用例

次の例は、ディレクトリ パスとファイル名を結合して、正しく書式化されたパスを作成します。

VB

```
Dim fullPath As String
fullPath = My.Computer.FileSystem.CombinePath _
("C:\Documents and Settings\All Users\Documents\My Pictures", "picture.jpg")
```

次の例は、2 つのパスを結合して、正しく書式化されたパスを作成します。

VB

```
Dim fullPath As String
fullPath = My.Computer.FileSystem.CombinePath _
("C:\Dir1\Dir2\Dir3", "..\Dir4\Dir5\File.txt")
```

この例では、`C:\Dir1\Dir2\Dir4\Dir5\File.txt` が返されます。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.GetParentPath メソッド
CombinePath

My.Computer.FileSystem.CopyDirectory メソッド

ディレクトリ間のコピーを実行します。

```
' Usage
My.Computer.FileSystem.CopyDirectory(sourceDirectoryName ,destinationDirectoryName)
My.Computer.FileSystem.CopyDirectory(sourceDirectoryName ,destinationDirectoryName ,overwri
te)
My.Computer.FileSystem.CopyDirectory(sourceDirectoryName ,destinationDirectoryName ,showUI)
My.Computer.FileSystem.CopyDirectory(sourceDirectoryName ,destinationDirectoryName ,showUI
,onUserCancel)
' Declaration
Public Sub CopyDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String _
)
' -or-
Public Sub CopyDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub CopyDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal showUI As UIOption _
)
' -or-
Public Sub CopyDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal showUI As UIOption, _
   ByVal onUserCancel As UICancelOption _
)
```

パラメータ

sourceDirectoryName

String です。コピーするディレクトリを指定します。必ず指定します。

destinationDirectoryName

String です。ディレクトリをコピーする場所を指定します。必ず指定します。

overwrite

Boolean です。既存のファイルを上書きするかどうかを指定します。既定値は **False** です。必ず指定します。

showUI

UIOption です。処理の進行状況を視覚的に表示するかどうかを指定します。既定値は **UIOption.OnlyErrorDialogs** です。必ず指定します。

onUserCancel

UICancelOption です。操作中にユーザーが [キャンセル] ボタンをクリックしたときに実行する処理を指定します。既定値は ThrowException です。必ず指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- ディレクトリとして指定した新しい名前にコロン (:) またはスラッシュ (\ または /) が含まれます (ArgumentException)。
- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである

(\\\\.\\ で開始されている)、のいずれかの理由が考えられます (**ArgumentException**)。

- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- *destinationDirectoryName* は、**Nothing** または空の文字列です (**ArgumentNullException**)。
- コピー元のディレクトリが存在しません (DirectoryNotFoundException)。
- コピー元のディレクトリがルート ディレクトリではありません (IOException)。
- 結合されたパスが、既存のファイルを指しています (**IOException**)。
- コピー元のパスとコピー先のパスが同じです (**IOException**)。
- *ShowUI* が **UIOption.AllDialogs** に設定され、ユーザーが操作をキャンセルしたか、ディレクトリ内の 1 つ以上のファイルをコピーできません (OperationCanceledException)。
- 操作が循環しています (InvalidOperationException)。
- パスにコロン (:) が含まれています (NotSupportedException)。
- パスがシステムで定義されている最大長を超えています (PathTooLongException)。
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効です (**NotSupportedException**)。
- ユーザーがパスを表示するのに必要なアクセス許可がありません (SecurityException)。
- コピー先のファイルは存在しますが、アクセスできません (UnauthorizedAccessException)。

解説

このメソッドは、内容を含めてディレクトリ全体をコピーします。コピー先のディレクトリが存在しないときは、新たに作成されます。同じ名前のディレクトリがコピー先に存在する場合は、2 つのディレクトリの内容が結合されます。コピー時にディレクトリに新しい名前を付けることができます。

ディレクトリ内のファイルをコピーするときに、特定のファイルが原因で例外がスローされることがあります。たとえば、*overwrite* が **False** に設定されていて、マージを実行しているときに、既に存在するファイルなどが原因です。このような例外は、スローされると 1 つの例外に統合され、その `Data` プロパティに IDictionary 形式のエントリが格納されます。このエントリには、ファイル パスまたはディレクトリ パスがキーとなり、特定の例外メッセージがそれに対応する値として格納されます。エントリを列挙するには、**For...Each** を使用します。

処理手順

My.Computer.FileSystem.CopyDirectory メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
ディレクトリをコピーします。	方法 : Visual Basic でディレクトリを別のディレクトリにコピーする

使用例

次のコード例は、 `TestDirectory1` というディレクトリを `TestDirectory2` というディレクトリにコピーし、既存のファイルを上書きします。

VB

```
My.Computer.FileSystem.CopyDirectory("C:\TestDirectory1", "C:\TestDirectory2", True)
```

`C:\TestDirectory1` および `C:\TestDirectory2` は、実際にコピーするディレクトリのパスと名前およびコピー先の場所に置き換えてください。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**

コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
UIPermission	ユーザー インターフェイスおよびクリップボードに関連するアクセス許可を制御します。関連する列挙値 : SafeSubWindows。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する
方法 : Visual Basic でディレクトリを移動する
方法 : Visual Basic でディレクトリの内容を移動する
方法 : Visual Basic でファイル パスを解析する
方法 : Visual Basic でファイルの絶対パスを確認する
方法 : Visual Basic でディレクトリが存在するかどうかを確認する

関連項目

My.Computer.FileSystem オブジェクト
UICancelOption 列挙型

My.Computer.FileSystem.CopyFile メソッド

ファイルを新しい場所にコピーします。

```
' Usage
My.Computer.FileSystem.CopyFile(sourceFileName ,destinationFileName)
My.Computer.FileSystem.CopyFile(sourceFileName ,destinationFileName ,overwrite)
My.Computer.FileSystem.CopyFile(sourceFileName ,destinationFileName ,showUI)
My.Computer.FileSystem.CopyFile(sourceFileName ,destinationFileName ,showUI ,onUserCancel)
' Declaration
Public Sub CopyFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String _
)
' -or-
Public Sub CopyFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub CopyFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal showUI As UIOption _
)
' -or-
Public Sub CopyFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal showUI As UIOption, _
   ByVal onUserCancel As UICancelOption _
)
```

パラメータ

sourceFileName

String。コピーするファイルを指定します。必ず指定します。

destinationFileName

String。ファイルをコピーする場所を指定します。必ず指定します。

overwrite

Boolean。既存のファイルを上書きするかどうかを指定します。既定値は **False** です。必ず指定します。

showUI

UIOption。処理の進行状況を視覚的に表示するかどうかを指定します。既定値は **UIOption.OnlyErrorDialogs** です。必ず指定します。

onUserCancel

UICancelOption。操作中にユーザーが [キャンセル] ボタンをクリックしたときに実行する処理を指定します。既定値は ThrowException です。必ず指定します。

例外

例外がスローされる可能性のある状況を次に示します。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\\\.\\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- 絶対パスが取得できませんでした (**ArgumentException**)。
- *destinationFileName* にパス情報が格納されています (**ArgumentException**)

- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- *destinationFileName* は、**Nothing** または空の文字列です (**ArgumentNullException**)。
- ソース ファイルが有効でないか存在しません (FileNotFoundException)。
- 結合されたパスが、既存のディレクトリを指しています (IOException)。
- コピー先のファイルが存在し、*overwrite* が **False** に設定されています (**IOException**)。
- ユーザーがファイルにアクセスするのに必要なアクセス許可がありません (**IOException**)。
- コピー先のディレクトリにある同じ名前のファイルが使用中です (**IOException**)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- *UICancelOption* が **ThrowException** に設定されており、ユーザーが操作をキャンセルしました (OperationCanceledException)。
- *UICancelOption* が **ThrowException** に設定されており、未指定の I/O エラーが発生しました (**OperationCanceledException**)。
- パスがシステムで定義されている最大長を超えています (PathTooLongException)。
- ユーザーに必要なアクセス許可がありません (UnauthorizedAccessException)。
- ユーザーがパスを表示するのに必要なアクセス許可がありません (SecurityException)。

解説

CopyFile は、アクセス制御エントリ (ACE: Access Control Entries) を保持しません。新たに作成したファイルは、ファイルのあるディレクトリの既定の ACE を継承します。

処理手順

My.Computer.FileSystem.CopyFile メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
ファイルを同じディレクトリにコピーします。	方法 : Visual Basic でファイルのコピーを同じディレクトリに作成する
ファイルを別のディレクトリにコピーします。	方法 : Visual Basic でファイルのコピーを別のディレクトリに作成する

使用例

次のコード例は、 `Test.txt` というファイルを `TestFiles2` というディレクトリにコピーします。既存のファイルの上書きは行いません。

VB

```
My.Computer.FileSystem.CopyFile _
("C:\UserFiles\TestFiles\test.txt", _
"C:\UserFiles\TestFiles2")
```

ファイルのパスは、実際のコードで使用するパスに置き換えてください。

次のコード例は、 `Test.txt` というファイルを `TestFiles2` というディレクトリにコピーし、ファイル名を `NewFile.txt` に変更します。

VB

```
My.Computer.FileSystem.CopyFile _
("C:\UserFiles\TestFiles\test.txt", _
"C:\UserFiles\TestFiles2", "NewFile.txt", FileIO.UICancelOption.DoNothing)
```

ファイルのパスは、実際のコードで使用するパスに置き換えてください。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
EnvironmentPermission	すべての環境変数へのアクセス許可を制御します。関連する列挙値 : Unrestricted。
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : **Unrestricted**。
RegistryPermission	レジストリ変数へのアクセス許可を制御します。関連する列挙値 : **Unrestricted**。
UIPermission	ユーザー インターフェイスおよびクリップボードに関連するアクセス許可を制御します。関連する列挙値 : SafeSubWindows。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic で特定のパターンを持つファイルをディレクトリにコピーする
方法 : Visual Basic でファイルのコピーを同じディレクトリに作成する
方法 : Visual Basic でディレクトリを別のディレクトリにコピーする
方法 : Visual Basic でファイルの名前を変更する

関連項目

My.Computer.FileSystem オブジェクト
UICancelOption 列挙型

My.Computer.FileSystem.CreateDirectory メソッド

ディレクトリを作成します。

```
' Usage
My.Computer.FileSystem.CreateDirectory(directory)
' Declaration
Public Sub CreateDirectory( _
   ByVal directory As String _
)
```

パラメータ

directory

ディレクトリの名前と場所を指定する **String** です。必ず指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- ディレクトリ名が不適切である場合。たとえば、名前に無効な文字が含まれている場合や、空白だけの場合です (ArgumentException クラス)。
- 作成するディレクトリの親ディレクトリが読み取り専用である場合 (IOException クラス)。
- ディレクトリ名が **Nothing** である場合 (ArgumentNullException クラス)。
- ディレクトリ名が長すぎる場合 (PathTooLongException クラス)。
- ディレクトリ名にコロン (":") だけが指定されている場合 (NotSupportedException クラス)。
- ディレクトリを作成するためのアクセス許可がユーザーにない場合 (UnauthorizedAccessException)。
- ユーザーが部分的に信頼されており、十分なアクセス許可を持っていない場合 (SecurityException)。

解説

ディレクトリが既に存在していた場合、例外はスローされません。

処理手順

My.Computer.FileSystem.CreateDirectory メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ディレクトリを作成する	方法 : Visual Basic でディレクトリを作成する

使用例

次の例は、`NewDirectory` ディレクトリを `C:\Documents and Settings\All Users\Documents` に作成します。

VB

```
My.Computer.FileSystem.CreateDirectory _
("C:\Documents and Settings\All Users\Documents\NewDirectory")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可

Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でディレクトリが存在するかどうかを確認する

関連項目

My.Computer.FileSystem オブジェクト

CreateDirectory

My.Computer.FileSystem.CurrentDirectory プロパティ

現在のディレクトリを取得または設定します。

```
' Usage
Dim value As String = My.Computer.FileSystem.CurrentDirectory
' Declaration
Public Property CurrentDirectory As String
```

戻り値

String.

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効 (DirectoryNotFoundException) です。
- ユーザーに必要なアクセス許可がありません (UnauthorizedAccessException)。

解説

CurrentDirectory はシステム全体に影響する環境変数です。

使用例

次の例では、現在のディレクトリを受け取ってメッセージ ボックスに表示します。

VB

```
MsgBox(My.Computer.FileSystem.CurrentDirectory)
```

次の例は、現在のディレクトリに `C:\TestDirectory` を設定します。

VB

```
My.Computer.FileSystem.CurrentDirectory = "C:\TestDirectory"
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**

Web サイト	**可**

アクセス許可

CurrentDirectory を設定するためのアクセス許可は必要ありません。

CurrentDirectory を読み込むために、次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト

Microsoft.VisualBasic.FileIO.FileSystem.CurrentDirectory

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.FileSystem.DeleteDirectory メソッド

ディレクトリを削除します。

```
' Usage
My.Computer.FileSystem.DeleteDirectory(directory ,onDirectoryNotEmpty)
My.Computer.FileSystem.DeleteDirectory(directory ,showUI ,recycle)
My.Computer.FileSystem.DeleteDirectory(directory ,showUI ,recycle ,onUserCancel)
' Declaration
Public Sub DeleteDirectory( _
   ByVal directory As String, _
   ByVal onDirectoryNotEmpty As DeleteDirectoryOption _
)
' -or-
Public Sub DeleteDirectory( _
   ByVal directory As String, _
   ByVal showUI As UIOption, _
   ByVal recycle As RecycleOption _
)
' -or-
Public Sub DeleteDirectory( _
   ByVal directory As String, _
   ByVal showUI As UIOption, _
   ByVal recycle As RecycleOption, _
   ByVal onUserCancel As UICancelOption _
)
```

パラメータ

directory

String です。削除するディレクトリです。必ず指定します。

onDirectoryNotEmpty

DeleteDirectoryOption 列挙型です。削除対象のディレクトリ内にファイルまたはディレクトリが存在する場合の処理を指定します。既定値は **DeleteDirectoryOption.DeleteAllContents** です。

showUI

UIOption 列挙型です。操作の進行状況を視覚的に表示するかどうかを指定します。既定値は **UIOption.OnlyErrorDialogs** です。必ず指定します。

recycle

RecycleOption 列挙型です。削除したファイルを [ごみ箱] に送るかどうかを指定します。既定値は **RecycleOption.DeletePermanently** です。

onUserCancel

UICancelOption 列挙型です。ユーザーが [キャンセル] をクリックした場合に例外をスローするかどうかを指定します。必ず指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが長さ 0 の文字列である、パスの形式に誤りがある、パスに空白だけが含まれている、パスに無効な文字 (ワイルドカード文字を含む) が含まれている (ArgumentException)。
- パスがデバイス パスである (\\.\ で始まる) (**ArgumentException**)。
- パスが **Nothing** である (ArgumentNullException)。
- ディレクトリが存在しないか、ファイルである (DirectoryNotFoundException)。
- 対象のディレクトリが空でなく、かつ、*onDirectoryNotEmpty* が **ThrowIfDirectoryNonEmpty** に設定されている (IOException)。
- 対象のディレクトリまたはサブディレクトリを削除するためのアクセス許可をユーザーが持っていない (**IOException**)。

- 対象のディレクトリまたはサブディレクトリ内のファイルが使用中である (**IOException**)。
- ファイル名またはディレクトリ名にコロン (:) が含まれている (NotSupportedException)。
- ユーザーが操作をキャンセルした、またはディレクトリを削除できなかった (OperationCanceledException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- ユーザーに必要なアクセス許可がない (SecurityException)。

解説

showUI、*recycle*、および *onUserCancel* パラメータは、ユーザー対話型でないアプリケーション (Windows サービスなど) ではサポートされていません。

処理手順

My.Computer.FileSystem.DeleteDirectory メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ディレクトリを削除する。	方法 : Visual Basic でディレクトリを削除する

使用例

この例では、 `OldDirectory` ディレクトリが空の場合のみ、このディレクトリを削除します。

VB

```
My.Computer.FileSystem.DeleteDirectory _
("C:\OldDirectory", FileIO.DeleteDirectoryOption.ThrowIfDirectoryNonEmpty)
```

この例では、 `OldDirectory` ディレクトリとその内容をすべて削除します。

VB

```
My.Computer.FileSystem.DeleteDirectory _
("C:\OldDirectory", FileIO.DeleteDirectoryOption.DeleteAllContents)
```

この例では、削除するかどうかをユーザーに確認したうえで、 `OldDirectory` ディレクトリとその内容をすべて削除します。ただし、削除した内容を [ごみ箱] には送りません。

VB

```
My.Computer.FileSystem.DeleteDirectory _
("C:\OldDirectory", FileIO.UIOption.AllDialogs, FileIO.RecycleOption.DeletePermanently, Fil
eIO.UICancelOption.ThrowException)
```

この例では、 `OldDirectory` ディレクトリとその内容をすべて削除して [ごみ箱] に送りますが、操作の進行状況は表示しません。

VB

```
My.Computer.FileSystem.DeleteDirectory("C:\OldDirectory", _
 FileIO.UIOption.AllDialogs, FileIO.RecycleOption.SendToRecycleBin, FileIO.UICancelOption.T
hrowException)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性

Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
UIPermission	ユーザー インターフェイスとクリップボードに関係するアクセス許可を制御します。関連する列挙値 : SafeSubWindows。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でファイルを削除する
方法 : Visual Basic でディレクトリを削除する

関連項目

My.Computer.FileSystem オブジェクト
UICancelOption 列挙型
RecycleOption 列挙型
DeleteDirectoryOption 列挙型
UICancelOption 列挙型
UIOption 列挙型
Microsoft.VisualBasic.FileIO.FileSystem.DeleteDirectory
UICancelOption

My.Computer.FileSystem.DeleteFile メソッド

ファイルを削除します。

```
' Usage
My.Computer.FileSystem.DeleteFile(file)
My.Computer.FileSystem.DeleteFile(file ,showUI ,recycle)
My.Computer.FileSystem.DeleteFile(file ,showUI ,recycle ,onUserCancel)
' Declaration
Public Sub DeleteFile( _
   ByVal file As String _
)
' -or-
Public Sub DeleteFile( _
   ByVal file As String, _
   ByVal showUI As UIOption, _
   ByVal recycle As RecycleOption _
)
' -or-
Public Sub DeleteFile( _
   ByVal file As String, _
   ByVal showUI As UIOption, _
   ByVal recycle As RecycleOption, _
   ByVal onUserCancel As UICancelOption _
)
```

パラメータ

file

削除するファイルの名前とパスを指定する **String** です。必ず指定します。

showUI

UIOption です。処理の進行状況を画面に表示するかどうかを指定します。既定値は **UIOption.OnlyErrorDialogs** です。必ず指定します。

recycle

RecycleOption です。削除したファイルを [ごみ箱] に入れるかどうかを指定します。既定値は **RecycleOption.DeletePermanently** です。必ず指定します。

onUserCancel

UICancelOption です。ユーザーが処理をキャンセルしたとき例外をスローするかどうかを指定します。既定値は **UICancelOption.ThrowException** です。必ず指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) ファイルを指定する場所に後続のスラッシュが入っている、5) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- パスがシステムで定義されている最大長 (PathTooLongException) を超えています。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- ファイルが使用中です (IOException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていません (SecurityException)。
- ファイルが存在しない場合 (FileNotFoundException)
- ユーザーがファイルを削除するためのアクセス許可を持っていないか、ファイルが読み取り専用です (UnauthorizedAccessException)。

- ユーザーが部分的に信頼されており、十分なアクセス許可を持っていません (**SecurityException**)。
- ユーザーが処理をキャンセルし、*onUserCancel* に Microsoft.VisualBasic.FileIO.UICancelOption.ThrowException が設定されています (OperationCanceledException)。

解説

showUI、*recycle*、および *onUserCancel* のパラメータは、Windows サービスなどのユーザーと対話しないアプリケーションではサポートされていません。

処理手順

My.Computer.FileSystem.DeleteFile メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ファイルを削除するには	方法 : Visual Basic でファイルを削除する
ディレクトリ内のすべてのファイルを削除するには	方法 : Visual Basic でディレクトリ内のすべてのファイルを削除する

使用例

次の例では、ファイル `Test.txt` を削除します。

VB

```
My.Computer.FileSystem.DeleteFile("C:\test.txt")
```

次の例ではファイル `Test.txt` を削除し、ファイルが削除されることをユーザーが確認できるようにします。

VB

```
My.Computer.FileSystem.DeleteFile _
("C:\test.txt", FileIO.UIOption.AllDialogs, FileIO.RecycleOption.SendToRecycleBin, FileIO.U
ICancelOption.ThrowException)
```

次の例ではファイル `Test.txt` を削除して [ごみ箱] に入れます。

VB

```
My.Computer.FileSystem.DeleteFile _
("C:\test.txt", FileIO.UIOption.OnlyErrorDialogs, FileIO.RecycleOption.SendToRecycleBin, Fi
leIO.UICancelOption.ThrowException)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**

Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
UIPermission	ユーザー インターフェイスとクリップボードに関連するアクセス許可を制御します。関連する列挙値 : SafeSubWindows。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でディレクトリを削除する

関連項目

My.Computer.FileSystem オブジェクト
UIOption 列挙型
RecycleOption 列挙型
UICancelOption 列挙型
My.Computer.FileSystem オブジェクト
Microsoft.VisualBasic.FileIO.FileSystem.DeleteFile

My.Computer.FileSystem.DirectoryExists メソッド

指定されたディレクトリが存在するかどうかを示す **Boolean** を返します。

```
' Usage
Dim value As Boolean = My.Computer.FileSystem.DirectoryExists(directory)
' Declaration
Public Function DirectoryExists( _
   ByVal directory As String _
) As Boolean
```

パラメータ

directory

ディレクトリのパスを指定する **String** です。必ず指定します。

戻り値

ディレクトリが存在する場合は **True**、それ以外の場合は **False** です。

例外

このメソッドは例外をスローしません。

解説

DirectoryExists の呼び出しでは、FileIOPermission が必要です。

処理手順

My.Computer.FileSystem.DirectoryExists メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ディレクトリが存在するかどうかを調べる	方法 : Visual Basic でディレクトリが存在するかどうかを確認する

使用例

次の例は、`C:\backup\logs` ディレクトリが存在するかどうかを判断し、そのプロパティを調べます。

VB

```
If My.Computer.FileSystem.DirectoryExists("C:\backup\logs") Then
    Dim logInfo As System.IO.DirectoryInfo
    logInfo = My.Computer.FileSystem.GetDirectoryInfo _
    ("C:\backup\logs")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**

Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でファイルが存在するかどうかを確認する

チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

My.Computer.FileSystem オブジェクト

DirectoryExists

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

My.Computer.FileSystem.Drives プロパティ

すべての使用可能なドライブの名前を読み取り専用コレクションとして返します。

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of System.IO.DriveInfo) = My
.Computer.FileSystem.Drives
' Declaration
Public ReadOnly Property Drives As System.Collections.ObjectModel.ReadOnlyCollection(Of Sys
tem.IO.DriveInfo)
```

戻り値

System.Collections.ObjectModel.ReadOnlyCollection (of **System.IO.DriveInfo**)

例外

このプロパティは例外をスローしません。

解説

このプロパティはすべての論理ドライブを返します。

処理手順

My.Computer.FileSystem.Drives プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
ドライブのボリューム ラベルを調べる	方法 : Visual Basic でドライブのボリューム ラベルを確認する
ドライブの形式を調べる	方法 : Visual Basic でドライブのフォーマットを確認する
ドライブの種類を調べる	方法 : Visual Basic でドライブの種類を確認する
ドライブの総領域を調べる	方法 : Visual Basic でドライブの合計領域を確認する
ドライブの空き領域を調べる	方法 : Visual Basic でドライブの物理空き容量を確認する
ドライブのルート ディレクトリを調べる	方法 : Visual Basic でドライブのルート ディレクトリを確認する

使用例

この例では、使用可能なドライブの名前をメッセージ ボックスに表示します。

VB

```
Dim getInfo As System.IO.DriveInfo()
getInfo = System.IO.DriveInfo.GetDrives
For Each info As System.IO.DriveInfo In getInfo
    MsgBox(info.name)
Next
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性

Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	**あり**
Windows サービス	**あり**
Web サイト	**あり**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト
System.Collections.ObjectModel.ReadOnlyCollection
System.IO.DriveInfo
Microsoft.VisualBasic.FileIO.FileSystem.Drives

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.FileSystem.FileExists メソッド

指定のファイルが存在するかどうかを表す **Boolean** 値を返します。

```
' Usage
Dim value As Boolean = My.Computer.FileSystem.FileExists(file)
' Declaration
Public Function FileExists( _
   ByVal file As String _
) As Boolean
```

パラメータ

file

String です。ファイルの名前とパスを指定します。必ず指定します。

戻り値

ファイルが存在する場合は **True**、それ以外の場合は **False** を返します。

例外

例外を引き起こす可能性のある状態を次に示します。

- ファイルの名前が円記号 (\) で終わっている (ArgumentException)。

解説

指定したファイルの読み取りに必要なアクセス許可がアプリケーションに与えられていない場合、**FileExists** メソッドは目的のパスが存在していても **False** を返します。ただし、例外はスローされません。

処理手順

My.Computer.FileSystem.FileExists メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ファイルが存在するかどうかを調べる	方法 : Visual Basic でファイルが存在するかどうかを確認する

使用例

この例では、ファイル `Check.txt` が存在するかどうかを調べ、その情報をメッセージ ボックスに表示します。

VB

```
If My.Computer.FileSystem.FileExists("c://Check.txt") Then
    MsgBox("File found.")
Else
    MsgBox("File not found.")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**

コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でディレクトリが存在するかどうかを確認する

チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

My.Computer.FileSystem オブジェクト

FileExists

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

My.Computer.FileSystem.FindInFiles メソッド

指定されたテキストを含むファイルの名前を表す文字列のコレクションを、読み取り専用で返します。

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.FindInFiles(directory ,containsText ,ignoreCase ,searchType)
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.FindInFiles(directory ,containsText ,ignoreCase ,searchType ,fileWildcards)
' Declaration
Public Function FindInFiles( _
   ByVal directory As String, _
   ByVal containsText As String, _
   ByVal ignoreCase As Boolean, _
   ByVal searchType As SearchOption _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
' -or-
Public Function FindInFiles( _
   ByVal directory As String, _
   ByVal containsText As String, _
   ByVal ignoreCase As Boolean, _
   ByVal searchType As SearchOption, _
   ByVal fileWildcards As String() _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
```

パラメータ

directory

検索するディレクトリを指定する **String** です。必ず指定します。

containsText

検索文字列を指定する **String** です。必ず指定します。

ignoreCase

大文字と小文字を区別して検索するかどうかを指定する **Boolean** です。既定値は **True** です。必ず指定します。

searchType

SearchOption 列挙型 です。サブフォルダを含めるかどうかを指定します。既定値は **SearchOption.SearchTopLevelOnly** です。必ず指定します。

fileWildcards

String 型の配列です。一致を調べるパターンです。必ず指定します。

戻り値

String 型の読み取り専用のコレクションです。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- *directory* が存在しません (DirectoryNotFoundException)。
- *directory* が既存のファイルをポイントしています (IOException)。
- パスがシステムで定義されている最大長 (PathTooLongException) を超えています。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。

- ユーザーがパスを表示するのに必要なアクセス許可を持っていません (SecurityException)。
- ユーザーに必要なアクセス許可がありません (UnauthorizedAccessException)。

解説

指定されたパターンに一致するファイルが見つからなければ、空のコレクションが返されます。

処理手順

My.Computer.FileSystem.FindInFiles メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
指定された文字列を含むファイルをディレクトリから検索する	チュートリアル : Visual Basic によるファイルとディレクトリの操作

使用例

次の例は、`C:\TestDir` ディレクトリから `"sample string"` の文字列を含むファイルをすべて探し出し、結果を `ListBox1` に表示します。

VB

```
Dim list As System.Collections.ObjectModel.ReadOnlyCollection _
(Of String)
list = My.Computer.FileSystem.FindInFiles("C:\TestDir", _
 "sample string", True, FileIO.SearchOption.SearchTopLevelOnly)
For Each name As String In list
    ListBox1.Items.Add(name)
Next
```

この例を実行するためには、プロジェクトに `ListBox1` という名前の **ListBox** が含まれている必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	**あり**
Windows サービス	**あり**
Web サイト	**あり**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

My.Computer.FileSystem オブジェクト

SearchOption 列挙型

System.Collections.ObjectModel.ReadOnlyCollection

Microsoft.VisualBasic.FileIO.FileSystem.FindInFiles

My.Computer.FileSystem.GetDirectoryInfo メソッド

ディレクトリ内のサブディレクトリのパス名を表す文字列のコレクションを返します。

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.GetDirectories(directory)
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.GetDirectories(directory ,searchType ,wildcards)
' Declaration
Public Function GetDirectories( _
   ByVal directory As String _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
' -or-
Public Function GetDirectories( _
   ByVal directory As String, _
   ByVal searchType As SearchOption, _
   ByVal wildcards As String() _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
```

パラメータ

directory

ディレクトリの名前とパスを指定する **String** です。必ず指定します。

searchType

SearchOption 列挙型 です。サブフォルダを含めるかどうかを指定します。既定値は **SearchOption.SearchTopLevelOnly** です。必ず指定します。

wildcards

名前と照合するパターンを指定する **String** です。必ず指定します。

戻り値

String の読み取り専用のコレクションです。

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- 1 つ以上の指定されたワイルドカード文字が **Nothing**か、空の文字列か、または空白のみを含んでいます (**ArgumentNullException**)。
- *directory* が存在しません (DirectoryNotFoundException)。
- *directory* が既存のファイルをポイントしています (IOException)。
- パスがシステムで定義されている最大長 (PathTooLongException) を超えています。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていません (SecurityException)。
- ユーザーに必要なアクセス許可がありません (UnauthorizedAccessException)。

解説

wildcards パラメータを使用して、特定のパターンを指定できます。サブディレクトリの中身を含めて検索する場合は、*searchType* パラメータに **SearchAllSubDirectories** を設定します。

指定されたパターンに一致するディレクトリが見つからなければ、空のコレクションが返されます。

処理手順

My.Computer.FileSystem.GetDirectories メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
特定のパターンを持つサブディレクトリを検索する	方法 : Visual Basic で特定のパターンに一致するサブディレクトリを検索する

使用例

次の例は、名前に `Logs` という語を含むすべてのディレクトリを、ディレクトリ構造から検索して返し、それらを `ListBox1` に追加します。

VB

```
For Each foundDirectory As String In _
My.Computer.FileSystem.GetDirectories _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments, _
True, "*Logs*")
    ListBox1.Items.Add(foundDirectory)
Next
```

この例では、`ListBox1` という名前の **ListBox** がフォームにあることが必要です。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	**あり**
Windows サービス	**あり**
Web サイト	**あり**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト
SearchOption 列挙型
System.Collections.ObjectModel.ReadOnlyCollection
Microsoft.VisualBasic.FileIO.FileSystem.GetDirectories

My.Computer.FileSystem.GetDirectoryInfo メソッド

指定されたパスの DirectoryInfo オブジェクトを返します。

```
' Usage
Dim value As System.IO.DirectoryInfo = My.Computer.FileSystem.GetDirectoryInfo(directory)
' Declaration
Public Function GetDirectoryInfo( _
   ByVal directory As String _
) As System.IO.DirectoryInfo
```

パラメータ

directory

ディレクトリのパスを指定する **String** です。必ず指定します。

戻り値

DirectoryInfo

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- ディレクトリ パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- ディレクトリ パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- ディレクトリ パスがシステムで定義されている最大長 (PathTooLongException) を超えています。
- ディレクトリ パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- ユーザーがディレクトリ パスを表示するのに必要なアクセス許可を持っていません (SecurityException)。

解説

ディレクトリが存在しない場合、最初に **DirectoryInfo** オブジェクトのプロパティがアクセスされるまで、例外はスローされません。

処理手順

My.Computer.FileSystem.GetDirectoryInfo メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ディレクトリがいつ作成されたかを調べる	方法 : Visual Basic でディレクトリの作成時刻を確認する
ディレクトリが読み取り専用であるかどうかを調べる	方法 : Visual Basic でディレクトリが読み取り専用かどうかを確認する

使用例

次の例は `C:\Documents and Settings` ディレクトリの **DirectoryInfo** オブジェクトを取得して、そのディレクトリの作成日時、最終アクセス日時、および最終更新日時を表示します。

VB

```
Dim getInfo As System.IO.DirectoryInfo
getInfo = My.Computer.FileSystem.GetDirectoryInfo _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments)
MsgBox("The directory was created at " & getInfo.CreationTime)
MsgBox("The directory was last accessed at " & getInfo.LastAccessTime)
MsgBox("The directory was last written to at " & getInfo.LastWriteTime)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト
System.IO.DirectoryInfo
GetDirectoryInfo

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.FileSystem.GetDriveInfo メソッド

指定されたドライブの DriveInfo オブジェクトを返します。

```
' Usage
Dim value As System.IO.DriveInfo = My.Computer.FileSystem.GetDriveInfo(drive)
' Declaration
Public Function GetDriveInfo( _
   ByVal drive As String _
) As System.IO.DriveInfo
```

パラメータ

drive

調べるドライブを指定する **String** です。必ず指定します。

戻り値

DriveInfo

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- パスがシステムで定義されている最大長 (PathTooLongException) を超えています。
- パス内のファイル名またはフォルダ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていません (SecurityException)。

解説

DriveInfo クラスはドライブを表し、ドライブ情報を問い合わせるためのメソッドやプロパティを提供します。**DriveInfo** はどのドライブが利用可能で、そのドライブの種類が何かを判断するために使用します。また、プロパティを問い合わせて、ドライブの容量や使用可能な空き領域を確認できます。

処理手順

My.Computer.FileSystem.GetDriveInfo メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ドライブのボリューム ラベルを確認する	方法 : Visual Basic でドライブのボリューム ラベルを確認する
ドライブの種類を確認する	方法 : Visual Basic でドライブの種類を確認する
ドライブの合計サイズを確認する	方法 : Visual Basic でドライブの合計領域を確認する
ドライブの物理的な空き領域を確認する	方法 : Visual Basic でドライブの物理空き容量を確認する
ドライブのルート フォルダを確認する	方法 : Visual Basic でドライブのルート ディレクトリを確認する

使用例

次の例は、C ドライブの **DriveInfo** オブジェクトを取得し、それを使用してドライブに関する情報を表示します。

VB

```
Dim getInfo As System.IO.DriveInfo
getInfo = My.Computer.FileSystem.GetDriveInfo("C:\")
```

```
MsgBox("The drive's type is " & getInfo.DriveType)
MsgBox("The drive has " & getInfo.TotalFreeSpace & " bytes free.")
```

ドライブの種類については、「DriveType」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト
System.IO.DriveInfo
GetDriveInfo
DriveType

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.FileSystem.GetFileInfo メソッド

指定されたファイルの FileInfo オブジェクトを返します。

```
' Usage
Dim value As System.IO.FileInfo = My.Computer.FileSystem.GetFileInfo(file)
' Declaration
Public Function GetFileInfo( _
   ByVal file As String _
) As System.IO.FileInfo
```

パラメータ

file

ファイルの名前とパスを指定する **String** です。必ず指定します。

戻り値

FileInfo

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パス名の形式に誤りがある場合。たとえば、無効な文字が含まれている場合や、空白だけの場合などです (ArgumentException)。
- ファイル名に後続のスラッシュ記号が指定されています (**ArgumentException**)。
- ファイルが存在しないか **Nothing** です (ArgumentNullException)。
- パスの文字列の間にコロンが含まれています (NotSupportedException)。
- パスが長すぎる場合 (PathTooLongException)
- ユーザーに必要なアクセス許可がありません (SecurityException)。
- ユーザーにファイルへの ACL (アクセス制御リスト) のアクセス権がありません (UnauthorizedAccessException)。

解説

ファイルが存在しなくても例外はスローされません。ただし、そのオブジェクトのプロパティが最初にアクセスされたときに例外がスローされます。

処理手順

My.Computer.FileSystem.GetFileInfo メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ファイルに関する情報の取得	方法 : Visual Basic でファイルについての情報を取得する
ファイルの名前とパスを調べる	方法 : Visual Basic でファイル パスを解析する

使用例

次の例は、 `MyLogFile.log` ファイルの **System.IO.FileInfo** オブジェクトを取得し、それを使ってファイルの完全名、最終アクセス日時、および長さを報告します。

VB

```
Dim information As System.IO.FileInfo
information = My.Computer.FileSystem.GetFileInfo("C:\MyLogFile.log")
MsgBox("The file's full name is " & information.FullName & ".")
MsgBox("Last access time is " & information.LastAccessTime & ".")
MsgBox("The length is " & information.Length & ".")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

My.Computer.FileSystem オブジェクト

System.IO.FileInfo

Microsoft.VisualBasic.FileIO.FileSystem.GetFileInfo(System.String)

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.FileSystem.GetFiles メソッド

ディレクトリ内のファイルの名前を表す文字列のコレクションを、読み取り専用で返します。

```
' Usage
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.GetFiles(directory)
Dim value As System.Collections.ObjectModel.ReadOnlyCollection(Of String) = My.Computer.Fil
eSystem.GetFiles(directory ,searchType ,wildcards)
' Declaration
Public Function GetFiles( _
   ByVal directory As String _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
' -or-
Public Function GetFiles( _
   ByVal directory As String, _
   ByVal searchType As SearchOption, _
   ByVal wildcards As String() _
) As System.Collections.ObjectModel.ReadOnlyCollection(Of String)
```

パラメータ

directory

ファイルを探すディレクトリを指定する **String** です。必ず指定します。

searchType

SearchOption 列挙型 です。サブフォルダを含めるかどうかを指定します。既定値は **SearchOption.SearchTopLevelOnly** です。必ず指定します。

wildcards

一致を調べるパターンを指定する **String** です。必ず指定します。

戻り値

文字列の読み取り専用のコレクションです。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- *directory* が存在しません (DirectoryNotFoundException)。
- *directory* が既存のファイルをポイントしています (IOException)。
- パスがシステムで定義されている最大長 (PathTooLongException) を超えています。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていません (SecurityException)。
- ユーザーに必要なアクセス許可がありません (UnauthorizedAccessException)。

解説

指定されたパターンに一致するファイルが見つからなければ、空のコレクションが返されます。

処理手順

My.Computer.FileSystem.GetFiles メソッドに関連するタスクの例を次の表に示します。

目的	参照項目

ディレクトリ内のファイルのコレクションを取得する	方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する
ディレクトリに含まれる特定のパターンを持つファイルを探す	方法 : Visual Basic で特定のパターンに一致するファイルを検索する

使用例

次の例は、ディレクトリ内のすべてのファイルを返して、 `ListBox1` にそれらを追加します。

VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments)
    ListBox1.Items.Add(foundFile)
Next
```

この例では、`ListBox1` という名前の **ListBox** がフォームにあることが必要です。

次の例は、ディレクトリ内の拡張子が `.dll` であるすべてのファイル返して、それらを `ListBox1`に追加します。

VB

```
For Each foundFile As String In My.Computer.FileSystem.GetFiles _
(My.Computer.FileSystem.SpecialDirectories.MyDocuments, True, "*.dll")
    ListBox1.Items.Add(foundFile)
Next
```

この例では、`ListBox1` という名前の **ListBox** がフォームにあることが必要です。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	**あり**
Windows サービス	**あり**
Web サイト	**あり**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic で特定のパターンに一致するファイルを検索する

方法 : Visual Basic でディレクトリにあるファイルのコレクションを取得する

関連項目

My.Computer.FileSystem オブジェクト

SearchOption 列挙型

System.Collections.ObjectModel.ReadOnlyCollection

Microsoft.VisualBasic.FileIO.FileSystem.GetFiles

My.Computer.FileSystem.GetName メソッド

指定されたパスからファイル名を解析します。

```
' Usage
Dim value As String = My.Computer.FileSystem.GetFileName(path)
' Declaration
Public Function GetFileName( _
   ByVal path As String _
) As String
```

パラメータ

path

必ず指定します。解析するパスです。**String**.

戻り値

String.

例外

このメソッドは例外をスローしません。

解説

これは文字列操作であり、**FileSystem** は調べません。

GetName メソッドはパスの末尾にあるスラッシュ (/) を無視します。

処理手順

My.Computer.FileSystem.GetFileName メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ファイル パスの解析	方法 : Visual Basic でファイル パスを解析する

使用例

次の例はファイル パスを解析して、ファイルの名前を返します。

VB

```
MsgBox("The filename is: " & _
My.Computer.FileSystem.GetName("C:\testdirectory\testfile"))
```

`C:\Testdirectory\Testfile` のパスは、実際に解析を行うパスに置き換えてください。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**

Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.FileSystem オブジェクト

GetFileName

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.FileSystem.GetParentPath メソッド

指定されたパスの親パスを返します。

```
' Usage
Dim value As String = My.Computer.FileSystem.GetParentPath(path)
' Declaration
Public Function GetParentPath( _
   ByVal path As String _
) As String
```

パラメータ

path

String です。確認するパスを指定します。必ず指定します。

戻り値

String.

例外

例外がスローされる可能性のある状態を次に示します。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\\\.\\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- パスはルート パスなので、親パスは存在しません (**ArgumentException**)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- パスがシステムで定義されている最大長を超えています (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- ユーザーがパスを表示するのに必要なアクセス許可がありません (SecurityException)。

解説

文字列操作なので、ファイル システムはチェックされません。

使用例

次のコード例は、 `C:\Backups\Tmp\Test` の親パスを取得します。

VB

```
Dim strPath As String
strPath = My.Computer.FileSystem.GetParentPath("C:\backups\tmp\test")
MsgBox(strPath)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**

コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.FileSystem オブジェクト

GetParentPath

その他の技術情報

Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

My.Computer.FileSystem.GetTempFileName メソッド

0 バイトの一時ファイルを一意な名前でディスクに作成し、そのファイルの完全パスを返します。

```
' Usage
Dim value As String = My.Computer.FileSystem.GetTempFileName()
' Declaration
Public Function GetTempFileName() As String
```

戻り値

一時ファイルの完全パスを格納する **String** です。

解説

このメソッドを使用すると、一時ファイルを作成できます。

使用例

次の例では一時ファイルを作成して、そのパスを返します。

VB

```
MsgBox("The file is located at " & _
My.Computer.FileSystem.GetTempFileName)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト

Microsoft.VisualBasic.FileIO.FileSystem.GetTempFileName

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

My.Computer.FileSystem.MoveDirectory メソッド

ディレクトリを別の場所に移動します。

```
' Usage
My.Computer.FileSystem.MoveDirectory(sourceDirectoryName ,destinationDirectoryName)
My.Computer.FileSystem.MoveDirectory(sourceDirectoryName ,destinationDirectoryName ,overwri
te)
My.Computer.FileSystem.MoveDirectory(sourceDirectoryName ,destinationDirectoryName ,showUI)
My.Computer.FileSystem.MoveDirectory(sourceDirectoryName ,destinationDirectoryName ,showUI
,onUserCancel)

' Declaration
Public Sub MoveDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String _
)
' -or-
Public Sub MoveDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub MoveDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal showUI As UIOption _
)
' -or-
Public Sub MoveDirectory( _
   ByVal sourceDirectoryName As String, _
   ByVal destinationDirectoryName As String, _
   ByVal showUI As UIOption, _
   ByVal onUserCancel As UICancelOption _
)
```

パラメータ

sourceDirectoryName

String です。移動元のディレクトリのパスです。必ず指定します。

destinationDirectoryName

String です。移動先のディレクトリのパスです。必ず指定します。

overwrite

Boolean です。既存のディレクトリを上書きするかどうかを指定します。既定値は **False** です。必ず指定します。

showUI

UIOptionです。操作の進行状況を視覚的に表示するかどうかを指定します。既定値は **UIOption.OnlyErrorDialogs** です。必ず指定します。

onUserCancel

UICancelOption 列挙型 です。ユーザーが操作をキャンセルしたときに例外をスローするかどうかを指定します。既定値は **UICancelOption.ThrowException** です。必ず指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが長さ 0 の文字列である、パスに空白だけが含まれている、パスに無効な文字が含まれている、パスがデバイス パスである (\\.\ で始まる) (ArgumentException)。

- パスが **Nothing** である (ArgumentNullException)。
- パスが無効である (DirectoryNotFoundException)。
- 移動元のディレクトリがルート ディレクトリである (IOException)。
- 組み合わせたパスが既存のファイルを指している (**IOException**)。
- 移動元のパスと移動先のパスが同じである (**IOException**)。
- ファイルが既に存在していて、*overwrite* が **False** に設定されている (**IOException**)。
- *onUserCancel* が **ThrowException** に設定されていて、ファイルのサブディレクトリがコピーできない (**IOException**)。
- 操作が循環的である (InvalidOperationException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれている (NotSupportedException)。
- **UICancelOption** が **ThrowException** に設定されていて、ユーザーが操作をキャンセルしたか、操作が完了できなかった (OperationCanceledException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- **UICancelOption** が **ThrowException** に設定されていて、ユーザーが必要なアクセス許可を持っていない (SecurityException)。
- ユーザーがファイルの修正に必要なアクセス許可を持っていない (UnauthorizedAccessException)。

解説

存在しないディレクトリの中にディレクトリを移動しようとした場合には、移動先の構造が自動的に作成されます。

処理手順

My.Computer.FileSystem.MoveDirectory メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ディレクトリを移動する	方法 : Visual Basic でディレクトリを移動する

使用例

この例では、 `Directory1` を `Directory2` に移動します。

VB

```
My.Computer.FileSystem.MoveDirectory("C:\Directory1", "C:\Directory2")
```

この例では、 `Directory1` を `Directory2` に移動し、既に移動先が存在する場合はそのディレクトリを上書きします。

VB

```
My.Computer.FileSystem.MoveDirectory("C:\Directory1", "C:\Directory2", _
True)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**

コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
UIPermission	ユーザー インターフェイスとクリップボードに関係するアクセス許可を制御します。関連する列挙値 : SafeSubWindows。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト
UIOption 列挙型
UICancelOption 列挙型
My.Computer.FileSystem.MoveFile メソッド
My.Computer.FileSystem.CopyDirectory メソッド

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

My.Computer.FileSystem.MoveFile メソッド

ファイルを新しい位置に移動します。

```
' Usage
My.Computer.FileSystem.MoveFile(sourceFileName ,destinationFileName)
My.Computer.FileSystem.MoveFile(sourceFileName ,destinationFileName ,overwrite)
My.Computer.FileSystem.MoveFile(sourceFileName ,destinationFileName ,showUI)
My.Computer.FileSystem.MoveFile(sourceFileName ,destinationFileName ,showUI ,onUserCancel)
' Declaration
Public Sub MoveFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String _
)
' -or-
Public Sub MoveFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub MoveFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal showUI As UIOption _
)
' -or-
Public Sub MoveFile( _
   ByVal sourceFileName As String, _
   ByVal destinationFileName As String, _
   ByVal showUI As UIOption, _
   ByVal onUserCancel As UICancelOption _
)
```

パラメータ

sourceFileName

String です。移動するファイルのパスです。必ず指定します。

destinationFileName

String です。ファイルの移動先となるディレクトリのパスです。必ず指定します。

overwrite

Boolean です。既存のファイルを上書きするかどうかを指定します。既定値は **False** です。必ず指定します。

showUI

UIOption 列挙型です。操作の進行状況を視覚的に表示するかどうかを指定します。既定値は **UIOption.OnlyErrorDialogs** です。必ず指定します。

onUserCancel

UICancelOption 列挙型 です。ユーザーが操作をキャンセルしたときに例外をスローするかどうかを指定します。既定値は **UICancelOption.ThrowException** です。必ず指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。つまり、パスが長さ 0 の文字列である、パスに空白だけが含まれている、パスに無効な文字が含まれている、またはパスがデバイス パスである (\\.\ で始まる) (ArgumentException)。
- ファイル名の末尾に円記号 (\) が付いている (**ArgumentException**)。
- パスが **Nothing** であるため、有効ではない (ArgumentNullException)。

- *destinationFileName* が **Nothing** または空の文字列である (**ArgumentNullException**)。
- ソース ファイルが有効でないか存在しない (FileNotFoundException)。
- 組み合わせたパスが既存のディレクトリを指していて、ターゲット ファイルが既に存在し、かつ *overwrite* が **False** に設定されている。ターゲット ディレクトリ内の同名のファイルが使用中である。ユーザーがファイルにアクセスするのに必要なアクセス許可を持っていない (IOException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、形式が無効である (NotSupportedException)。
- *onUserCancel* が **ThrowException** に設定されているときに、ユーザーが操作をキャンセルしたか、不特定の I/O エラーが発生した (OperationCanceledException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていない (SecurityException)。
- ユーザーが必要なアクセス許可を持っていない (UnauthorizedAccessException)。

解説

ターゲット構造が存在しないときは、新たに作成されます。

MoveFile メソッドは、同じボリューム内でファイルを移動する場合にのみアクセス制御エントリ (ACE: Access Control Entry) を維持します。これには継承された ACE も含まれており、移動すると、それが直接 ACE になります (直接 ACE は継承された ACE よりも優先されます)。別のボリュームにファイルを移動する場合は、ACE はコピーされません。

処理手順

My.Computer.FileSystem.MoveFile メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ファイルを移動する	方法 : Visual Basic でファイルを移動する
ファイルのコレクションを移動する	方法 : Visual Basic でファイルのコレクションを移動する
ディレクトリの内容を移動する	方法 : Visual Basic でディレクトリの内容を移動する

使用例

この例では、 `Test.txt` というファイルを `TestDir1` から `TestDir2` に移動します。

VB

```
My.Computer.FileSystem.MoveFile("C:\TestDir1\test.txt", "C:\TestDir2\test.txt")
```

この例では、 `Test.txt` というファイルを `TestDir1` から `TestDir2` に移動し、名前を `Test2.txt` に変更します。

VB

```
My.Computer.FileSystem.MoveFile("C:\TestDir1\test.txt", "C:\TestDir2\test2.txt")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**

コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
EnvironmentPermission	すべての環境変数へのアクセスを制御します。関連する列挙値 : Unrestricted。
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : **Unrestricted**。
RegistryPermission	レジストリ変数へのアクセスを制御します。関連する列挙値 : **Unrestricted**。
UIPermission	ユーザー インターフェイスとクリップボードに関係するアクセス許可を制御します。関連する列挙値 : SafeSubWindows。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト

UIOption 列挙型

UICancelOption 列挙型

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

My.Computer.FileSystem.OpenTextFieldParser メソッド

OpenTextFieldParser メソッドを使用すると、TextFieldParser オブジェクトを作成できます。このオブジェクトは、ログなどの構造化されたテキスト ファイルを容易かつ効率的に解析できるようにします。**TextFieldParser** オブジェクトは、区切り記号入りファイルと固定長ファイルの両方を読み取るときに使用できます。

```
' Usage
Dim value As TextFieldParser = My.Computer.FileSystem.OpenTextFieldParser(file)
Dim value As TextFieldParser = My.Computer.FileSystem.OpenTextFieldParser(file ,delimiters)
Dim value As TextFieldParser = My.Computer.FileSystem.OpenTextFieldParser(file ,fieldWidths
)
' Declaration
Public Function OpenTextFieldParser( _
   ByVal file As String _
) As TextFieldParser
' -or-
Public Function OpenTextFieldParser( _
   ByVal file As String, _
   ByVal delimiters As String() _
) As TextFieldParser
' -or-
Public Function OpenTextFieldParser( _
   ByVal file As String, _
   ByVal fieldWidths As Integer() _
) As TextFieldParser
```

パラメータ

file

String です。**TextFieldParser** で開くファイルを指定します。必ず指定します。

delimiters

String() です。フィールドの区切り記号を指定します。必ず指定します。

fieldWidths

Integer() です。フィールドの幅を指定します。必ず指定します。

戻り値

TextFieldParser

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。つまり、パスが長さ 0 の文字列である、パスに空白だけが含まれている、パスに無効な文字が含まれている、またはパスがデバイス パスである (\\.\ で始まる) (ArgumentException)。
- ファイル名の末尾に後続のスラッシュが付いている (**ArgumentException**)。
- パスが **Nothing** であるため、有効ではない (ArgumentNullException)。
- 指定のファイルが存在しない (FileNotFoundException)。
- 指定のファイルが使用中である (IOException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、形式が無効である (NotSupportedException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていない (SecurityException)。
- 行を指定の形式で解析できない (MalformedLineException)。例外メッセージではこの行が例外の原因とされ、TextFieldParser.ErrorLine プロパティにこの行内のテキストが割り当てられる。

- ユーザーがファイルにアクセスするのに必要なアクセス許可を持っていない (UnauthorizedAccessException)。

処理手順

My.Computer.FileSystem.OpenTextFieldParser メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
区切り記号で区切られたテキスト ファイルから読み込む	方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
固定幅のテキスト ファイルから読み込む	方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
複数の形式を持つテキスト ファイルから読み込む	方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

使用例

この例では、`reader` という **TextFieldParser** を開き、それを使用して `C:\TestFolder1\Test1.txt` から読み込みを行います。

VB

```
Dim reader As Microsoft.VisualBasic.FileIO.TextFieldParser
reader = My.Computer.FileSystem.OpenTextFieldParser _
("C:\TestFolder1\test1.txt")
reader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
reader.delimiters = New String() {","}
Dim currentRow As String()
While Not reader.EndOfData
    Try
        currentRow = reader.ReadFields()
        Dim currentField As String
        For Each currentField In currentRow
            MsgBox(currentField)
        Next
    Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
        MsgBox("Line " & ex.Message & _
        "is not valid and will be skipped.")
    End Try
End While
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
SecurityPermission	コードに適用される一連のセキュリティ アクセス許可を表します。関連する列挙値 : ControlEvidence。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でテキスト ファイルを読み取る
方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem オブジェクト
TextFieldParser オブジェクト
OpenTextFieldParser

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

その他の技術情報

Visual Basic でのファイルの読み取り

My.Computer.FileSystem.OpenTextFileReader メソッド

StreamReader を開きます。

```
' Usage
Dim value As System.IO.StreamReader = My.Computer.FileSystem.OpenTextFileReader(file)
Dim value As System.IO.StreamReader = My.Computer.FileSystem.OpenTextFileReader(file ,encod
ing)
' Declaration
Public Function OpenTextFileReader( _
   ByVal file As String _
) As System.IO.StreamReader
' -or-
Public Function OpenTextFileReader( _
   ByVal file As String, _
   ByVal encoding As System.Text.Encoding _
) As System.IO.StreamReader
```

パラメータ

file

読み込むファイルを指定する **String** です。必ず指定します。

encoding

Encoding です (既定値は UTF8)。必ず指定します。

戻り値

StreamReader

例外

例外を引き起こす可能性のある状態を次に示します。

- ファイル名が円記号で (\) 終わっています (ArgumentException)。
- 指定されたファイルが見つかりません (FileNotFoundException)。
- ユーザーがファイルの読み込みに必要なアクセス許可を持っていません (SecurityException)。

解説

StreamReader を使って読み込むことができるのはテキスト ファイルだけです。

処理手順

My.Computer.FileSystem.OpenTextFileReader メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
StreamReader を使用してファイルを開く	方法 : StreamReader を使用してファイルからテキストを読み取る (Visual Basic)

使用例

次の例では、ファイル `Testfile.txt` を開き、1 行読み込んでから、その行を **MessageBox** に表示します。

VB

```
Dim fileReader As System.IO.StreamReader
fileReader = _
My.Computer.FileSystem.OpenTextFileReader("C:\\testfile.txt")
Dim stringReader As String
stringReader = fileReader.ReadLine()
MsgBox("The first line of the file is " & stringReader)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可能
Windows アプリケーション	○
クラス ライブラリ	○
コンソール アプリケーション	○
Windows コントロール ライブラリ	○
Web コントロール ライブラリ	○
Windows サービス	○
Web サイト	○

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : StreamReader を使用してファイルからテキストを読み取る (Visual Basic)

関連項目

My.Computer.FileSystem オブジェクト
System.Text.Encoding
System.IO.StreamReader
Microsoft.VisualBasic.FileIO.FileSystem.OpenTextFileReader

その他の技術情報

Visual Basic でのファイルの読み取り

My.Computer.FileSystem.OpenTextFileWriter メソッド

StreamWriter を開きます。

```
' Usage
Dim value As System.IO.StreamWriter = My.Computer.FileSystem.OpenTextFileWriter(file ,appen
d)
Dim value As System.IO.StreamWriter = My.Computer.FileSystem.OpenTextFileWriter(file ,appen
d ,encoding)
' Declaration
Public Function OpenTextFileWriter( _
   ByVal file As String, _
   ByVal append As Boolean _
) As System.IO.StreamWriter
' -or-
Public Function OpenTextFileWriter( _
   ByVal file As String, _
   ByVal append As Boolean, _
   ByVal encoding As System.Text.Encoding _
) As System.IO.StreamWriter
```

パラメータ

file

必ず指定します。書き込むファイルを指定する文字列 **String** です。

append

情報をファイルに追加するか、ファイル全体を情報で上書きするかを指定する **Boolean** です。既定値は **False** です。必ず指定します。

encoding

ファイルの書き込みに使用するエンコーディングを示す Encoding です。既定値は UTF8 です。必ず指定します。

戻り値

StreamWriter

例外

例外がスローされる可能性のある状態を次に示します。

- ファイル名がスラッシュ (ArgumentException) で終わる。

処理手順

My.Computer.FileSystem.OpenTextFileWriter メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
テキストを **StreamWriter** を使ってファイルに書き込みます。	方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

使用例

このコード例は、**My.Computer.FileSystem.OpenTextFileWriter** メソッドを使って **StreamWriter** を開き、**StreamWriter** クラスの **WriteLine** メソッドを使って文字列をテキスト ファイルに書き込みます。

VB

```
Dim file As System.IO.StreamWriter
file = My.Computer.FileSystem.OpenTextFileWriter("c:\test.txt", True)
file.WriteLine("Here is the first string.")
file.Close()
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の使用可/不可

プロジェクトの種類	使用可能
Windows アプリケーション	○
クラス ライブラリ	○
コンソール アプリケーション	○
Windows コントロール ライブラリ	○
Web コントロール ライブラリ	○
Windows サービス	○
Web サイト	○

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic で StreamWriter を使用してテキストをファイルに書き込む

関連項目

My.Computer.FileSystem オブジェクト
System.Text.Encoding
System.IO.StreamWriter
Microsoft.VisualBasic.FileIO.FileSystem.OpenTextFileWriter

My.Computer.FileSystem.ReadAllBytes メソッド

ファイルの内容をバイト配列で返します。

```
' Usage
Dim value As Byte() = My.Computer.FileSystem.ReadAllBytes(file)
' Declaration
Public Function ReadAllBytes( _
   ByVal file As String _
) As Byte()
```

パラメータ

file

読み込むファイルを指定する **String** です。必ず指定します。

戻り値

ファイルの内容が格納された **Byte** 配列。

例外

例外がスローされる可能性のある状態を次に示します。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- ファイル名が円記号 (\) で終わっています (**ArgumentException**)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- ファイルが存在しない場合 (FileNotFoundException)
- ファイルが別のプロセスで使用中か、または I/O エラーが起きています (IOException)。
- パスがシステムで定義されている最大長 (PathTooLongException) を超えています。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- 文字列をバッファに書き込むための十分なメモリがありません (OutOfMemoryException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていません (SecurityException)。

解説

My.Computer.FileSystem オブジェクトの **ReadAllBytes** メソッドを使うと、バイナリ ファイルを読み取ることができます。ファイルの内容はバイト配列で返されます。

ファイル名からファイルの内容を判断しないでください。たとえば、Form1.vb というファイルが Visual Basic のソース ファイルではない可能性もあります。アプリケーションでデータを使用する前に、入力をすべて検証してください。

処理手順

My.Computer.FileSystem.ReadAllBytes メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
バイナリ ファイルからの読み取り	方法 : Visual Basic でバイナリ ファイルを読み取る

使用例

ファイル `C:/Documents and Settings/selfportrait.jpg.` を読み取る例は次のようになります。

VB

```
My.Computer.FileSystem.ReadAllBytes _
("C:/Documents and Settings/selfportrait.jpg")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト

ReadAllBytes

その他の技術情報

Visual Basic でのファイルの読み取り

My.Computer.FileSystem.ReadAllText メソッド

テキスト ファイルの内容を **String** で返します。

```
' Usage
Dim value As String = My.Computer.FileSystem.ReadAllText(file)
Dim value As String = My.Computer.FileSystem.ReadAllText(file ,encoding)
' Declaration
Public Function ReadAllText( _
   ByVal file As String _
) As String
' -or-
Public Function ReadAllText( _
   ByVal file As String, _
   ByVal encoding As System.Text.Encoding _
) As String
```

パラメータ

file

読み取るファイルの名前とパスを指定する **String** です。必ず指定します。

encoding

ファイルの読み取りに使用する文字エンコーディングを指定する System.Text.Encoding です。必ず指定します。既定値は UTF-8 です。

戻り値

ファイルの内容が格納された **String**。

例外

ファイルの内容が予想どおりでないことがあり、ファイルの内容を読み取るメソッドが失敗する可能性があります。

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- ファイル名が円記号 (\) で終わっています (**ArgumentException**)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- ファイルが存在しない場合 (FileNotFoundException)
- ファイルが別のプロセスで使用中か、または I/O エラーが起きています (IOException)。
- パスがシステムで定義されている最大長 (PathTooLongException) を超えています。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- 文字列をバッファに書き込むための十分なメモリがありません (OutOfMemoryException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていません (SecurityException)。

解説

My.Computer.FileSystem オブジェクトの **ReadAllText** メソッドを使うと、テキスト ファイルを読み取ることができます。ファイルの内容は文字列で返されます。

ファイルの中身が ASCII や UTF-8 などでエンコーディングされている場合は、ファイルのエンコーディングを指定できます。拡張文字を含むファイルを読み取る場合は、ファイルのエンコーディングを指定する必要があります。

ファイル名からファイルの内容を判断しないでください。たとえば、Form1.vb というファイルが Visual Basic のソース ファイルではない可能性もあります。アプリケーションでデータを使用する前に、入力をすべて検証してください。

処理手順

My.Computer.FileSystem.ReadAllText メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
テキスト ファイルからの読み取り	方法 : Visual Basic でテキスト ファイルを読み取る

使用例

次の例は `Test.txt` の内容を文字列に読み取り、メッセージ ボックスに表示します。

VB

```
Dim reader As String
reader = My.Computer.FileSystem.ReadAllText("C:\test.txt")
MsgBox(reader)
```

次の例は、ASCII ファイルである `Test.txt` の内容を文字列に読み取り、メッセージ ボックスに表示します。

VB

```
Dim reader As String
reader = My.Computer.FileSystem.ReadAllText("C:\test.txt", _
   System.Text.Encoding.ASCII)
MsgBox(reader)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : StreamReader を使用してファイルからテキストを読み取る (Visual Basic)
トラブルシューティング : テキスト ファイルの読み取りと書き込み
チュートリアル : Visual Basic によるファイルとディレクトリの操作

関連項目

My.Computer.FileSystem オブジェクト
System.Text.Encoding
Microsoft.VisualBasic.FileIO.FileSystem.ReadAllText
StreamReader

概念

ファイル エンコーディング

その他の技術情報

Visual Basic でのファイルの読み取り

My.Computer.FileSystem.RenameDirectory メソッド

ディレクトリの名前を変更します。

```
' Usage
My.Computer.FileSystem.RenameDirectory(directory ,newName)
' Declaration
Public Sub RenameDirectory( _
   ByVal directory As String, _
   ByVal newName As String _
)
```

パラメータ

directory

名前を変更するディレクトリのパスと名前を指定する **String** です。必ず指定します。

newName

ディレクトリの新しい名前を指定する **String** です。必ず指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- *newName* パラメータにパスの情報が指定されています (**ArgumentException**)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- *newName* パラメータが **Nothing** または空の文字列です (**ArgumentNullException**)。
- ソース ディレクトリが有効でないか存在しません (DirectoryNotFoundException)。
- *newName* に指定された名前のファイルまたはディレクトリが既に存在します (IOException)。
- ディレクトリがルート ディレクトリです (**IOException**)。
- パスが 248 文字を超えています (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていません (SecurityException)。
- ユーザーに必要なアクセス許可がありません (UnauthorizedAccessException)。

解説

このメソッドを使ってディレクトリを移動させることはできません。ディレクトリを移動して名前を変更するには、**MoveDirectory** を使用します。

処理手順

My.Computer.FileSystem.RenameDirectory メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ディレクトリの名前を変更します。	方法 : Visual Basic でディレクトリの名前を変更する

使用例

ディレクトリ名を `Test` から `SecondTest` に変換する例は次のようになります。

VB

```
My.Computer.FileSystem.RenameDirectory("C:MyDocuments\Test", "SecondTest")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.FileSystem オブジェクト

My.Computer.FileSystem.MoveDirectory メソッド

RenameDirectory

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

My.Computer.FileSystem.RenameFile メソッド

ファイル名を変更します。

```
' Usage
My.Computer.FileSystem.RenameFile(file ,newName)
' Declaration
Public Sub RenameFile( _
   ByVal file As String, _
   ByVal newName As String _
)
```

パラメータ

file

名前を変更するファイルを指定する **String** です。必ず指定します。

newName

新しいファイル名を指定する **String** です。必ず指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\.\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- *newName* パラメータにパスの情報が含まれているか、または円記号 (\) で開始されています (**ArgumentException**)。
- パスが **Nothing** であるため、有効ではありません (ArgumentNullException)。
- *newName* パラメータが **Nothing** または空の文字列です (**ArgumentNullException**)。
- ソース ファイルが有効でないか存在しません (FileNotFoundException)。
- *newName* に指定された名前のファイルまたはディレクトリが既に存在します (IOException)。
- パスがシステムで定義されている最大長 (PathTooLongException) を超えています。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- ユーザーがパスを表示するのに必要なアクセス許可がありません (SecurityException)。
- ユーザーに必要なアクセス許可がありません (UnauthorizedAccessException)。

解説

このメソッドを使ってファイルを移動することはできません。ファイルを移動して名前を変更するには、My.Computer.FileSystem.MoveFile メソッド を使用します。

処理手順

My.Computer.FileSystem.RenameFile メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ファイル名の変更	方法 : Visual Basic でファイルの名前を変更する

使用例

ファイル名を `Test.txt` から `SecondTest.txt` に変換する例は次のようになります。

VB

```
My.Computer.FileSystem.RenameFile("C:\Test.txt", "SecondTest.txt")
```

"C:\Test.txt" の部分は、名前を変更するファイルのパスと名前に置き換えてください。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト

Microsoft.VisualBasic.FileIO.FileSystem.RenameFile(System.String,System.String)

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

My.Computer.FileSystem.SpecialDirectories プロパティ

頻繁に参照されるディレクトリにアクセスするためのプロパティを提供するオブジェクトを取得します。

```
' Usage
Dim value As Microsoft.VisualBasic.MyServices.SpecialDirectoriesProxy = My.Computer.FileSys
tem.SpecialDirectories
' Declaration
Public ReadOnly Property SpecialDirectories As MyServices.SpecialDirectoriesProxy
```

戻り値

このプロパティは、コンピュータの My.Computer.FileSystem.SpecialDirectories オブジェクト を返します。

解説

このプロパティによって、My.Computer.FileSystem.SpecialDirectories オブジェクト に簡単にアクセスできます。

使用例

この例では、ユーザーのデスクトップ ディレクトリのファイル パスを返し、それを表示します。

VB

```
Dim filePath As String
filePath = My.Computer.FileSystem.SpecialDirectories.Desktop
MsgBox(filePath)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.SpecialDirectories オブジェクト

My.Computer.FileSystem.WriteAllBytes メソッド

バイナリ ファイルにデータを書き込みます。

```
' Usage
My.Computer.FileSystem.WriteAllBytes(file ,data ,append)
' Declaration
Public Sub WriteAllBytes( _
   ByVal file As String, _
   ByVal data As Byte(), _
   ByVal append As Boolean _
)
```

パラメータ

file

String です。書き込み先のファイルのパスと名前を指定します。必ず指定します。

data

Byte です。ファイルに書き込むデータを指定します。必ず指定します。

append

Boolean です。データを追加するか、上書きするかを指定します。既定値は **False** です。必ず指定します。

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが無効です。1) 長さが 0 の文字列である、2) 空白だけが含まれている、3) 無効な文字が含まれている、4) デバイス パスである (\\\\.\\ で開始されている)、のいずれかの理由が考えられます (ArgumentException)。
- パスが **Nothing** であるか円記号 (\) で終わるため、有効ではありません (ArgumentNullException)。
- *file* パラメータは、存在しないパスを指しています (FileNotFoundException または DirectoryNotFoundException)。
- ファイルが別のプロセスで使用されているか、I/O エラーが起こります (IOException)。
- パスがシステムで定義されている最大長 を超えています (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、または形式が無効です (NotSupportedException)。
- ユーザーがパスを表示するのに必要なアクセス許可がありません (SecurityException)。

解説

append パラメータが **True** の場合にデータはファイルに追加され、それ以外の場合はファイル内のデータが上書きされます。

指定されたパスからファイル名を除外した部分が無効なパスである場合、**DirectoryNotFoundException** 例外がスローされます。パスが有効でもファイルが存在しない場合は、そのファイルが作成されます。

処理手順

My.Computer.FileSystem.WriteAllBytes メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
バイナリ ファイルに書き込みます。	方法 : Visual Basic でバイナリ ファイルに書き込む

使用例

次のコード例は、データ配列 `CustomerData` を `CollectedData` というファイルに追加します。

VB

```
My.Computer.FileSystem.WriteAllBytes _
("C:\MyDocuments\CustomerData", CustomerData, True)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でバイナリ ファイルに書き込む

関連項目

My.Computer.FileSystem オブジェクト

WriteAllBytes

その他の技術情報

Visual Basic でのファイルへの書き込み

My.Computer.FileSystem.WriteAllText メソッド

ファイルにテキストを書き込みます。

```
' Usage
My.Computer.FileSystem.WriteAllText(file ,text ,append)
My.Computer.FileSystem.WriteAllText(file ,text ,append ,encoding)
' Declaration
Public Sub WriteAllText( _
   ByVal file As String, _
   ByVal text As String, _
   ByVal append As Boolean _
)
' -or-
Public Sub WriteAllText( _
   ByVal file As String, _
   ByVal text As String, _
   ByVal append As Boolean, _
   ByVal encoding As System.Text.Encoding _
)
```

パラメータ

file

必ず指定します。書き込み先のファイルを指定する文字列 **String** です。

text

必ず指定します。ファイルに書き込むテキストを指定する文字列 **String** です。

append

Boolean です。既存のテキストに追加するか、既存のテキストを上書きするかを指定します。既定値は **False** です。必ず指定します。

encoding

Encoding です。ファイルへの書き込み時に使用するエンコーディングを指定します。必ず指定します。既定値は UTF-8 です。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが無効である。つまり、パスが長さ 0 の文字列である、パスに空白だけが含まれている、パスに無効な文字が含まれている、末尾に後続のスラッシュが付いている、またはパスがデバイス パスである (\\.\ で始まる) (ArgumentException)。
- パスが **Nothing** であるため、有効ではない (ArgumentNullException)。
- *file* が、存在しないパスを指している (FileNotFoundException または DirectoryNotFoundException)。
- ファイルが別のプロセスで使用されている、または I/O が発生した (IOException)。
- パスがシステムで定義されている最大長を超えている (PathTooLongException)。
- パス内のファイル名またはディレクトリ名にコロン (:) が含まれているか、形式が無効である (NotSupportedException)。
- ユーザーがパスを表示するのに必要なアクセス許可を持っていない (SecurityException)。

部分的に信頼されているコンテキストでプロセスを実行している場合は、権限不足のため例外がスローされることがあります。詳細については、「コード アクセス セキュリティの基礎」を参照してください。

解説

エンコーディングを指定しなかった場合は、UTF-8 が使用されます。エンコーディングのバイト順マーク (BOM: byte order mark) は、システムの現在の ANSI コード ページを使用する System.Text.Encoding.Default を指定しない限り、ファイルに書き込まれます。

指定されたファイルが存在しない場合は、新たに作成されます。

append が **False** に設定されている場合は、ファイル内の既存のテキストが上書きされます。

指定したエンコーディングがファイルの既存のエンコーディングに一致しない場合は、指定したエンコーディングが無視されます。

処理手順

My.Computer.FileSystem.WriteAllText メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ファイルにテキストを書き込む	方法 : Visual Basic でテキストをファイルに書き込む
ファイルにテキストを追加する	方法 : Visual Basic でテキスト ファイルに追記する

使用例

この例では、 `"This is new text to be added."` という行をファイル `Test.txt` に書き込み、このファイル内の既存のテキストをすべて上書きします。

VB

```
My.Computer.FileSystem.WriteAllText("C:\TestFolder1\test.txt", _
"This is new text to be added.", False)
```

この例では、 `Documents and Settings` フォルダに含まれているファイルの名前を `FileList.txt` に書き込み、読みやすくするために各ファイル名の間に復帰を挿入します。

VB

```
For Each foundFile As String In _
My.Computer.FileSystem.GetFiles("C:\Documents and Settings")
    foundFile = foundFile & vbCrLf
    My.Computer.FileSystem.WriteAllText _
    ("C:\Documents and Settings\FileList.txt", foundFile, True)
Next
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : FileSystemProxy (FileSystem へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**使用する**
クラス ライブラリ	**使用する**
コンソール アプリケーション	**使用する**
Windows コントロール ライブラリ	**使用する**
Web コントロール ライブラリ	**使用する**
Windows サービス	**使用する**
Web サイト	**使用する**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明

FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem オブジェクト

System.Text.Encoding

Microsoft.VisualBasic.FileIO.FileSystem.WriteAllText

その他の技術情報

Visual Basic でのファイルへの書き込み

My.Computer.FileSystem.SpecialDirectories オブジェクト

頻繁に参照されるディレクトリへのアクセスに使用するプロパティを提供します。

解説

OS がそのディレクトリをサポートしていないなどの理由で参照先のディレクトリのパスが空白の場合、DirectoryNotFoundException 例外がスローされます。

パスの最後にバックスラッシュ (\) は表示されません。

処理手順

My.Computer.FileSystem.SpecialDirectories オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
MyDocuments ディレクトリから読み取り	方法 : Visual Basic で My Documents ディレクトリの内容を取得する

使用例

この例では、ユーザーのデスクトップ ディレクトリの `filePath` を返し、これを表示します。

VB

```
Dim filePath As String
filePath = My.Computer.FileSystem.SpecialDirectories.Desktop
MsgBox(filePath)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス SpecialDirectoriesProxy (SpecialDirectories へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクトのメンバ
My.Computer.FileSystem.SpecialDirectories.AllUsersApplicationData プロパティ
My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData プロパティ
My.Computer.FileSystem.SpecialDirectories.Desktop プロパティ
My.Computer.FileSystem.SpecialDirectories.MyDocuments プロパティ
My.Computer.FileSystem.SpecialDirectories.MyMusic プロパティ
My.Computer.FileSystem.SpecialDirectories.MyPictures プロパティ
My.Computer.FileSystem.SpecialDirectories.Programs プロパティ
My.Computer.FileSystem.SpecialDirectories.Temp プロパティ
My.Computer.FileSystem オブジェクト
Microsoft.VisualBasic.FileIO.SpecialDirectories

My.Computer.FileSystem.SpecialDirectories オブジェクトのメンバ

My.Computer.FileSystem.SpecialDirectories プロパティ オブジェクトには、頻繁に参照されるディレクトリにアクセスするためのプロパティが用意されています。

プロパティ

AllUsersApplicationData	All Users フォルダの Application Data ディレクトリをポイントするパス名を表す読み取り専用の文字列 (**String**) です。
CurrentUserApplicationData	現在のユーザーの Application Data ディレクトリをポイントするパス名を表す読み取り専用の文字列 (**String**) です。
Desktop	ユーザーのデスクトップ ディレクトリをポイントするパス名を表す読み取り専用の文字列 (**String**) です。
MyDocuments	ユーザーの MyDocuments ディレクトリをポイントするパス名を表す読み取り専用の文字列 (**String**) です。
MyMusic	ユーザーの MyMusic ディレクトリをポイントするパス名を表す読み取り専用の文字列 (**String**) です。
MyPictures	ユーザーの MyPictures ディレクトリをポイントするパス名を表す読み取り専用の文字列 (**String**) です。
[プログラム]	ユーザーの Programs ディレクトリをポイントするパス名を表す読み取り専用の文字列 (**String**) です。
Temp	ユーザーの Temp ディレクトリをポイントするパス名を表す読み取り専用の文字列 (**String**) です。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト

My.Computer.FileSystem.SpecialDirectories.AllUsersApplicationData プロパティ

コンピュータ上のすべてのユーザーで共有されるアプリケーション データの格納場所へのパスを取得します。

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.AllUsersApplicationData
' Declaration
Public ReadOnly Property AllUsersApplicationData As String
```

戻り値

String.

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが空です。通常は、オペレーティング システムでディレクトリ (DirectoryNotFoundException) がサポートされていないことが原因です。

解説

My.Computer.FileSystem.SpecialDirectories オブジェクト には、よく参照されるディレクトリのパスが含まれています。

パスが存在しなかった場合は、"*BasePath**CompanyName**ProductName**Version*" の形式で作成されます。

通常、Windows XP の *BasePath* は C:\Documents and Settings\All Users\Application Data になります。*CompanyName*、*ProductName*、および *Version* は、それぞれ My.Application.Info.CompanyName、My.Application.Info.ProductName、My.Application.Info.Version の各プロパティから取得できます。

メモ :

[アセンブリ情報] ダイアログ ボックスで会社名と製品名が指定されていなかった場合、パスの "*CompanyName**ProductName*" の部分は、アセンブリ名で置き換えられます。アセンブリ情報で名前を設定する方法の詳細については、「方法 : アセンブリ情報を指定する」を参照してください。

使用例

次の例では、共有されるアプリケーション データのパスを MessageBox に表示します。

VB

```
MsgBox _
(My.Computer.FileSystem.SpecialDirectories.AllUsersApplicationData)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス SpecialDirectoriesProxy (SpecialDirectories へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類別の使用可/不可

プロジェクトの種類	使用可能
Windows アプリケーション	○
クラス ライブラリ	○
コンソール アプリケーション	○
Windows コントロール ライブラリ	○
Web コントロール ライブラリ	○

Windows サービス	○
Web サイト	○

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルおよびフォルダへのアクセスを制御します。関連する列挙体 : Unrestricted.

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト

Microsoft.VisualBasic.FileIO.SpecialDirectories.AllUsersApplicationData

My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData プロパティ

現在のユーザーのアプリケーション データが格納されているパスを取得します。

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData
' Declaration
Public ReadOnly Property CurrentUserApplicationData As String
```

戻り値

String.

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが空です。通常は、オペレーティング システムでディレクトリ (DirectoryNotFoundException) がサポートされていないことが原因です。

解説

My.Computer.FileSystem.SpecialDirectories オブジェクト には、よく参照されるディレクトリのパスが含まれています。

パスが存在しなかった場合は、"*BasePath**CompanyName**ProductName**Version*" の形式で作成されます。

通常、Windows XP の場合、*BasePath* は C:\Documents and Settings*username*\Application Data になります。*CompanyName*、*ProductName*、および *Version* は、それぞれ My.Application.Info.CompanyName、My.Application.Info.ProductName、My.Application.Info.Version の各プロパティから取得できます。

メモ :

[アセンブリ情報] ダイアログ ボックスで会社名と製品名が指定されていなかった場合、パスの "*CompanyName**ProductName*" の部分は、アセンブリ名で置き換えられます。アセンブリ情報で名前を設定する方法の詳細については、「方法 : アセンブリ情報を指定する」を参照してください。

このパスには、ユーザー プロファイルの中でも、ローミング (移動) に対応したデータが格納されます。"ローミング ユーザー" は、ネットワーク内の複数のコンピュータで作業を行います。こうしたローミング ユーザーのユーザー プロファイルはネットワーク上のサーバーに格納され、そのユーザーがログオンしたときに、システムに読み込まれます。ユーザー プロファイルをローミング可能として扱うためには、オペレーティング システムが移動プロファイルをサポートしており、その機能が有効に設定されている必要があります。

使用例

次の例は、現在のユーザーのアプリケーション データが保存されるパスを MessageBox に表示します。

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.CurrentUserApplicationData)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス SpecialDirectoriesProxy (SpecialDirectories へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類別の使用可/不可

プロジェクトの種類	使用可能
Windows アプリケーション	○
クラス ライブラリ	○
コンソール アプリケーション	○
Windows コントロール ライブラリ	○
Web コントロール ライブラリ	○

Windows サービス	○
Web サイト	○

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
FileIOPermission	ファイルおよびフォルダへのアクセスを制御します。関連する列挙体 : Unrestricted.

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト

Microsoft.VisualBasic.FileIO.SpecialDirectories.CurrentUserApplicationData

My.Computer.FileSystem.SpecialDirectories.Desktop プロパティ

デスクトップ ディレクトリを指すパス名を取得します。

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.Desktop
' Declaration
Public ReadOnly Property Desktop As String
```

戻り値

String.

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが空です。通常は、オペレーティング システムでディレクトリ (DirectoryNotFoundException) がサポートされていないことが原因です。

解説

My.Computer.FileSystem.SpecialDirectories オブジェクト には、よく参照されるディレクトリのパスが含まれています。

使用例

この例では、デスクトップ ディレクトリのパスを MessageBox に表示します。

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.Desktop)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス SpecialDirectoriesProxy (SpecialDirectories へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
FileIOPermission	ファイルおよびフォルダへのアクセスを制御します。関連する列挙体 : Unrestricted.

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト
Microsoft.VisualBasic.FileIO.SpecialDirectories.Desktop

My.Computer.FileSystem.SpecialDirectories.MyDocuments プロパティ

マイ ドキュメント ディレクトリを指すパス名を取得します。

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.MyDocuments
' Declaration
Public ReadOnly Property MyDocuments As String
```

戻り値

String.

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが空です。通常は、オペレーティング システムでディレクトリ (DirectoryNotFoundException) がサポートされていないことが原因です。

解説

My.Computer.FileSystem.SpecialDirectories オブジェクト には、よく参照されるディレクトリのパスが含まれています。

使用例

次の例は、マイ ドキュメント ディレクトリへのパスを MessageBox に表示します。

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.MyDocuments)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス SpecialDirectoriesProxy (SpecialDirectories へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
FileIOPermission	ファイルおよびフォルダへのアクセスを制御します。関連する列挙体 : Unrestricted.

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト
Microsoft.VisualBasic.FileIO.SpecialDirectories.MyDocuments

My.Computer.FileSystem.SpecialDirectories.MyMusic プロパティ

マイ ミュージック ディレクトリを指すパス名を取得します。

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.MyMusic
' Declaration
Public ReadOnly Property MyMusic As String
```

戻り値

String.

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが空です。通常は、オペレーティング システムでディレクトリ (DirectoryNotFoundException) がサポートされていないことが原因です。

解説

My.Computer.FileSystem.SpecialDirectories オブジェクト には、頻繁に参照されるディレクトリへのパスが格納されます。

使用例

次の例は、マイ ミュージック ディレクトリへのパスを MessageBox に表示します。

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.MyMusic)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : SpecialDirectoriesProxy (SpecialDirectories へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト
Microsoft.VisualBasic.FileIO.SpecialDirectories.MyMusic

My.Computer.FileSystem.SpecialDirectories.MyPictures プロパティ

マイ ピクチャ ディレクトリを指すパス名を取得します。

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.MyPictures
' Declaration
Public ReadOnly Property MyPictures As String
```

戻り値

String.

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが空です。通常は、オペレーティング システムでディレクトリ (DirectoryNotFoundException) がサポートされていないことが原因です。

解説

My.Computer.FileSystem.SpecialDirectories オブジェクト クラスには、よく参照されるディレクトリのパスが含まれています。

使用例

次の例は、マイ ピクチャ ディレクトリへのパスを MessageBox に表示します。

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.MyPictures)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス SpecialDirectoriesProxy (SpecialDirectories へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
FileIOPermission	ファイルおよびフォルダへのアクセスを制御します。関連する列挙体 : Unrestricted.

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト
Microsoft.VisualBasic.FileIO.SpecialDirectories.MyPictures

My.Computer.FileSystem.SpecialDirectories.ProgramFiles プロパティ

Program Files ディレクトリを指すパスを取得します。

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.ProgramFiles
' Declaration
Public ReadOnly Property Programs As String
```

戻り値

String.

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが空です。通常は、オペレーティング システムでディレクトリ (DirectoryNotFoundException) がサポートされていないことが原因です。

解説

My.Computer.FileSystem.SpecialDirectories オブジェクト には、よく参照されるディレクトリのパスが含まれています。Program Files ディレクトリには、コンピュータのプログラム ファイルが含まれています。

使用例

この例では、Program Files ディレクトリのパスを MessageBox に表示します。

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.ProgramFiles)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : SpecialDirectoriesProxy (SpecialDirectories へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト
Microsoft.VisualBasic.FileIO.SpecialDirectories.ProgramFiles

My.Computer.FileSystem.SpecialDirectories.Programs プロパティ

Programs ディレクトリを指すパス名を取得します。

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.Programs
' Declaration
Public ReadOnly Property Programs As String
```

戻り値

String.

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- パスが空です。通常は、オペレーティング システムでディレクトリ (DirectoryNotFoundException) がサポートされていないことが原因です。

解説

My.Computer.FileSystem.SpecialDirectories オブジェクト には、よく参照されるディレクトリのパスが含まれています。Programs ディレクトリには、ユーザーのプログラム グループが含まれます。

使用例

次の例では、Programs ディレクトリのパスを MessageBox に表示します。

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.Programs)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス SpecialDirectoriesProxy (SpecialDirectories へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
FileIOPermission	ファイルおよびフォルダへのアクセスを制御します。関連する列挙体 : Unrestricted.

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト
Microsoft.VisualBasic.FileIO.SpecialDirectories.Programs

My.Computer.FileSystem.SpecialDirectories.Temp プロパティ

システムの現在の一時ディレクトリを指すパス名を取得します。

```
' Usage
Dim value As String = My.Computer.FileSystem.SpecialDirectories.Temp
' Declaration
Public ReadOnly Property Temp As String
```

戻り値

String.

例外

例外を引き起こす可能性のある状態を次に示します。

- パスが空です。通常は、オペレーティング システムでディレクトリ (DirectoryNotFoundException) がサポートされていないことが原因です。

解説

My.Computer.FileSystem.SpecialDirectories オブジェクト には、頻繁に参照されるディレクトリへのパスが格納されます。

使用例

次の例は、システムの現在の一時ディレクトリのパスを MessageBox に表示します。

VB

```
MsgBox(My.Computer.FileSystem.SpecialDirectories.Temp)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : SpecialDirectoriesProxy (SpecialDirectories へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
EnvironmentPermission	システム環境変数とユーザー環境変数へのアクセスを制御します。関連する列挙値 : Unrestricted。
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : **Unrestricted**。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.FileSystem.SpecialDirectories オブジェクト
Microsoft.VisualBasic.FileIO.SpecialDirectories.Temp

My.Computer.Info オブジェクト

コンピュータのメモリ、ロードされているアセンブリ、名前、およびオペレーティング システムについての情報を取得するためのプロパティを提供します。

解説

My.Computer.Info オブジェクトによって公開されるプロパティは、アプリケーションの配置元であるコンピュータ (実行時に決定する) に関する情報を返します。通常、このデータは開発コンピュータで何が利用可能だったかの情報とは異なります。

使用例

次の例は、**My.Computer.Info.AvailablePhysicalMemory** プロパティを使用して、コードを実行しているコンピュータの使用可能な物理メモリの量を表示します。

VB

```
MsgBox("Computer's available physical memory: " & _
    My.Computer.Info.AvailablePhysicalMemory)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : ComputerInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照

関連項目

My.Computer.Info オブジェクトのメンバ

My.Computer.Info オブジェクトのメンバ

My.Computer.Info オブジェクトには、コンピュータのメモリ、ロードされているアセンブリ、名前、およびオペレーティング システムについての情報を取得するためのプロパティが用意されています。

プロパティ

AvailablePhysicalMemory	コンピュータの物理的な空きメモリの合計サイズを取得します。
AvailableVirtualMemory	コンピュータの使用可能な仮想アドレス空間の、空きサイズの合計を取得します。
InstalledUICulture	オペレーティング システムでインストールされている、現在の UI カルチャを取得します。
OSFullName	オペレーティング システムの完全名を取得します。
OSPlatform	コンピュータのオペレーティング システムのプラットフォーム ID を取得します。
OSVersion	コンピュータのオペレーティング システムのバージョンを取得します。
TotalPhysicalMemory	コンピュータの物理メモリの合計サイズを取得します。
TotalVirtualMemory	コンピュータの使用可能な仮想アドレス空間の合計サイズを取得します。

参照

関連項目

My.Computer.Info オブジェクト

My.Computer.Info.AvailablePhysicalMemory プロパティ

コンピュータの空き物理メモリの合計量を取得します。

```
' Usage
Dim value As ULong = My.Computer.Info.AvailablePhysicalMemory
' Declaration
Public ReadOnly Property AvailablePhysicalMemory As ULong
```

戻り値

コンピュータの空き物理メモリのバイト数を表す **ULong**

例外

例外を引き起こす可能性のある状態を次に示します。

- アプリケーションがメモリ ステータスを取得できない (Win32Exception)。

解説

My.Computer.Info.AvailablePhysicalMemory プロパティは、Windows XP、Windows 2000 Professional、Windows Server 2003、または Windows 2000 Server でのみ使用できます。

使用例

この例では、**My.Computer.Info.AvailablePhysicalMemory** プロパティを使用して、このコードを実行しているコンピュータの使用可能な物理メモリの量を表示します。

VB

```
MsgBox("Computer's available physical memory: " & _
    My.Computer.Info.AvailablePhysicalMemory)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : ComputerInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Info オブジェクト

My.Computer.Info.TotalPhysicalMemory プロパティ

My.Computer.Info.AvailableVirtualMemory プロパティ

Microsoft.VisualBasic.Devices.ComputerInfo.AvailablePhysicalMemory

My.Computer.Info.AvailableVirtualMemory プロパティ

コンピュータの仮想アドレス空間の空き領域の合計サイズを取得します。

```
' Usage
Dim value As ULong = My.Computer.Info.AvailableVirtualMemory
' Declaration
Public ReadOnly Property AvailableVirtualMemory As ULong
```

戻り値

コンピュータの仮想アドレス空間の空き領域のバイト数を格納した **ULong** です。

例外

例外を引き起こす可能性のある状態を次に示します。

- アプリケーションがメモリのステータスを取得できません (Win32Exception)。

解説

My.Computer.Info.AvailableVirtualMemory プロパティを使用できるのは、Windows XP、Windows 2000 Professional、Windows Server 2003、または Windows 2000 Server に限られます。

使用例

次の例は、**My.Computer.Info.AvailableVirtualMemory** プロパティを使用して、コードを実行しているコンピュータの使用可能な仮想メモリの量を表示します。

VB

```
MsgBox("Computer's available virtual memory: " & _
    My.Computer.Info.AvailableVirtualMemory)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : ComputerInfo

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Info オブジェクト

My.Computer.Info.TotalVirtualMemory プロパティ

My.Computer.Info.AvailablePhysicalMemory プロパティ

Microsoft.VisualBasic.Devices.ComputerInfo.AvailableVirtualMemory

My.Computer.Info.InstalledUICulture プロパティ

オペレーティング システムと共にインストールされた現在の UI カルチャを取得します。

```
' Usage
Dim value As System.Globalization.CultureInfo = My.Computer.Info.InstalledUICulture
' Declaration
Public ReadOnly Property InstalledUICulture As System.Globalization.CultureInfo
```

戻り値

CultureInfo オブジェクトは、コンピュータにインストールされた UI カルチャを表します。

解説

My.Computer.Info.InstalledUICulture プロパティと InstalledUICulture プロパティの動作は同じです。

日本語版の Windows 2000 Professional など、ローカライズされたオペレーティング システムでは、このプロパティはオペレーティング システムのカルチャを返します。

使用例

次のコード例は、**My.Computer.Info.InstalledUICulture** プロパティを使って、オペレーティング システムの UI カルチャの名前を表示します。

VB

```
MsgBox("Computer's UI culture name: " & _
    My.Computer.Info.InstalledUICulture.DisplayName)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : ComputerInfo

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Info オブジェクト

System.Globalization.CultureInfo
Microsoft.VisualBasic.Devices.ComputerInfo.InstalledUICulture

My.Computer.Info.OSFullName プロパティ

完全なオペレーティング システム名を取得します。

```
' Usage
Dim value As String = My.Computer.Info.OSFullName
' Declaration
Public ReadOnly Property OSFullName As String
```

戻り値

オペレーティング システム名を含む **String** です。

例外

例外を引き起こす可能性のある状態を次に示します。

- 呼び出し元のコードが、完全な信頼を与えられていません (SecurityException)。

解説

このプロパティは、WMI (Windows Management Instrumentation) がコンピュータにインストールされている場合に、オペレーティング システム名に関する詳細な情報を返します。そうでない場合、このプロパティは、**My.Computer.Info.OSPlatform** プロパティと同じ文字列を返します。この文字列に含まれる情報は、WMI から提供される情報ほど詳細ではありません。

使用例

次のコード例は、**My.Computer.Info.OSFullName** プロパティを使って、コンピュータのオペレーティング システムの名前を表示します。

VB

```
MsgBox("Computer's operating system name: " & _
    My.Computer.Info.OSFullName)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : ComputerInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.Info オブジェクト

My.Computer.Info.OSPlatform プロパティ

Microsoft.VisualBasic.Devices.ComputerInfo.OSFullName

My.Computer.Info.OSPlatform プロパティ

コンピュータのオペレーティング システムのプラットフォーム ID を取得します。

```
' Usage
Dim value As String = My.Computer.Info.OSPlatform
' Declaration
Public ReadOnly Property OSPlatform As String
```

戻り値

コンピュータのオペレーティング システムのプラットフォーム ID を含む **String** です。ID は PlatformID 列挙体のメンバ名から選択されます。

例外

例外を引き起こす可能性のある状態を次に示します。

- アプリケーションがオペレーティング システムのプラットフォーム情報を取得できない場合 (ExecutionEngineException)。

解説

My.Computer.Info.OSPlatform プロパティは、OSVersion プロパティから返されるオブジェクトの Platform プロパティに類似した機能を提供します。

使用例

次の例は **My.Computer.Info.OSPlatform** プロパティを使用して、コンピュータのオペレーティング システムのプラットフォームを表示します。

VB

```
MsgBox("Computer's operating system platform: " & _
    My.Computer.Info.OSPlatform)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : ComputerInfo

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Info オブジェクト
Platform
OSVersion
Microsoft.VisualBasic.Devices.ComputerInfo.OSPlatform

My.Computer.Info.OSVersion プロパティ

コンピュータのオペレーティング システムのバージョンを取得します。

```
' Usage
Dim value As String = My.Computer.Info.OSVersion
' Declaration
Public ReadOnly Property OSVersion As String
```

戻り値

オペレーティング システムの現在のバージョン番号を含む **String**

例外

例外を引き起こす可能性のある状態を次に示します。

- アプリケーションがオペレーティング システムのバージョン情報を取得できない (ExecutionEngineException)

解説

My.Computer.Info.OSVersion プロパティは、バージョン情報を "major.minor.build.revision" という形式にします。

My.Computer.Info.OSVersion プロパティは、OSVersion プロパティから返されるオブジェクトの Version プロパティによく似た働きをします。

使用例

この例では、**My.Computer.Info.OSVersion** プロパティを使用してコンピュータのオペレーティング システムのバージョンを表示します。

VB

```
MsgBox("Computer's operating system version: " & _
    My.Computer.Info.OSVersion)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : ComputerInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Info オブジェクト

Microsoft.VisualBasic.Devices.ComputerInfo.OSVersion

My.Computer.Info.TotalPhysicalMemory プロパティ

コンピュータの物理メモリの合計サイズを取得します。

```
' Usage
Dim value As ULong = My.Computer.Info.TotalPhysicalMemory
' Declaration
Public ReadOnly Property TotalPhysicalMemory As ULong
```

戻り値

コンピュータの物理メモリのバイト数を格納する **ULong** です。

例外

例外を引き起こす可能性のある状態を次に示します。

- アプリケーションがメモリのステータスを取得できません (Win32Exception)。

解説

My.Computer.Info.TotalPhysicalMemory プロパティを使用できるのは、Windows XP、Windows 2000 Professional、Windows Server 2003、または Windows 2000 Server に限られます。

使用例

次の例は、**My.Computer.Info.TotalPhysicalMemory** プロパティを使用して、このコードを実行しているコンピュータの物理メモリの合計サイズを表示します。

VB

```
MsgBox("Computer's available physical memory: " & _
    My.Computer.Info.TotalPhysicalMemory)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : ComputerInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Info オブジェクト

My.Computer.Info.AvailablePhysicalMemory プロパティ

My.Computer.Info.TotalVirtualMemory プロパティ

Microsoft.VisualBasic.Devices.ComputerInfo.TotalPhysicalMemory

My.Computer.Info.TotalVirtualMemory プロパティ

コンピュータで使用できる仮想アドレス空間の合計量を取得します。

```
' Usage
Dim value As ULong = My.Computer.Info.TotalVirtualMemory
' Declaration
Public ReadOnly Property TotalVirtualMemory As ULong
```

戻り値

コンピュータで使用できる仮想アドレス空間のバイト数を表す **ULong**

例外

例外を引き起こす可能性のある状態を次に示します。

- アプリケーションがメモリ ステータスを取得できない (Win32Exception)。

解説

My.Computer.Info.TotalVirtualMemory プロパティは、Windows XP、Windows 2000 Professional、Windows Server 2003、または Windows 2000 Server でのみ使用できます。

使用例

この例では、**My.Computer.Info.TotalVirtualMemory** プロパティを使用して、このコードを実行しているコンピュータの仮想メモリの合計量を表示します。

VB

```
MsgBox("Computer's available virtual memory: " & _
    My.Computer.Info.TotalVirtualMemory)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : ComputerInfo

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Info オブジェクト

My.Computer.Info.AvailableVirtualMemory プロパティ

My.Computer.Info.TotalPhysicalMemory プロパティ

Microsoft.VisualBasic.Devices.ComputerInfo.TotalVirtualMemory

My.Computer.Keyboard オブジェクト

キーボードの現在の状態 (現在どのキーが押されているかなど) にアクセスするためのプロパティや、アクティブなウィンドウにキーストロークを送るためのメソッドが用意されています。

解説

My.Computer.Keyboard オブジェクトはコンピュータのキーボードへのインターフェイスを提供します。**My.Computer.Keyboard** のプロパティを使用すると、いくつもの特殊なキーの状態に関する情報を取得できます。**My.Computer.Keyboard.SendKeys** メソッドを使用すると、まるでキーボードから入力されたかのように、アクティブなウィンドウにキーを送ることができます。

タスク

My.Computer.Keyboard オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
CapsLock がオンになっているかどうかをチェックする	方法 : Visual Basic で CapsLock キーがオンかどうかを確認する
アプリケーションにキーストロークを送信する	方法 : アプリケーションを起動してキーストロークを送る (Visual Basic)

使用例

My.Computer.Keyboard.CtrlKeyDown プロパティを使用して、コンピュータの Ctrl キーが押されたかどうかを調べる例は次のようになります。

VB

```
If My.Computer.Keyboard.CtrlKeyDown Then
    MsgBox("CTRL key down")
Else
    MsgBox("CTRL key up")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Keyboard

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

参照

関連項目

My.Computer.Keyboard オブジェクトのメンバ

My.Computer オブジェクト

Microsoft.VisualBasic.Devices.Keyboard

概念

キーボードへのアクセス

My.Computer.Keyboard オブジェクトのメンバ

My.Computer.Keyboard オブジェクトには、キーボードの現在の状態 (現在どのキーが押されているかなど) にアクセスするためのプロパティや、アクティブなウィンドウにキーストロークを送るためのメソッドが用意されています。

プロパティ

AltKeyDown	Alt キーが押されているかどうかを示す **Boolean** を取得します。 このプロパティは、非サーバー アプリケーションだけで使用できます。
CapsLock	CapsLock キーがオンになっているかどうかを示す **Boolean** を取得します。 このプロパティは、非サーバー アプリケーションだけで使用できます。
CtrlKeyDown	Ctrl キーが押されているかどうかを示す **Boolean** を取得します。 このプロパティは、非サーバー アプリケーションだけで使用できます。
NumLock	NumLock キーがオンになっているかどうかを示す **Boolean** を取得します。 このプロパティは、非サーバー アプリケーションだけで使用できます。
ScrollLock	ScrollLock キーがオンになっているかどうかを示す **Boolean** を取得します。 このプロパティは、非サーバー アプリケーションだけで使用できます。
ShiftKeyDown	Shift キーが押されているかどうかを示す **Boolean** を取得します。 このプロパティは、非サーバー アプリケーションだけで使用できます。

メソッド

SendKeys	キーボードで入力されたかのように、1 つ以上のキーストロークをアクティブなウィンドウに送ります。 このメソッドは、非サーバー アプリケーションだけで使用できます。

参照

関連項目

My.Computer.Keyboard オブジェクト

My.Computer.Keyboard.AltKeyDown プロパティ

Alt キーが押されたかどうかを示す **Boolean** を取得します。

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.AltKeyDown
' Declaration
Public ReadOnly Property AltKeyDown As Boolean
```

戻り値

Boolean の値です。Alt キーが押された場合は **True** になり、押されない場合は **False** になります。

解説

My.Computer.Keyboard.AltKeyDown プロパティは、ModifierKeys プロパティと同様の機能を提供します。

使用例

My.Computer.Keyboard.AltKeyDown プロパティを使用して、コンピュータの Alt キーが押されたかどうかを調べる例は次のようになります。

VB

```
If My.Computer.Keyboard.AltKeyDown Then
    MsgBox("ALT key down")
Else
    MsgBox("ALT key up")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Keyboard

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Keyboard オブジェクト

ModifierKeys
Microsoft.VisualBasic.Devices.Keyboard.AltKeyDown

My.Computer.Keyboard.CapsLock プロパティ

CAPS LOCK がオンになっているかどうかを示す **Boolean** 値を取得します。

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.CapsLock
' Declaration
Public ReadOnly Property CapsLock As Boolean
```

戻り値

Boolean 値。CAPS LOCK がオンの場合は **True**、それ以外の場合は **False** になります。

解説

CAPS LOCK の状態のみが返されます。

使用例

この例では、**My.Computer.Keyboard.CapsLock** プロパティを使用して、コンピュータの CAPS LOCK がオンになっているかどうかを確認します。

VB

```
If My.Computer.Keyboard.CapsLock Then
    MsgBox("CAPS LOCK on")
Else
    MsgBox("CAPS LOCK off")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Keyboard

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Keyboard オブジェクト

Microsoft.VisualBasic.Devices.Keyboard.CapsLock

My.Computer.Keyboard.CtrlKeyDown プロパティ

Ctrl キーが押されたかどうかを示す **Boolean** を取得します。

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.CtrlKeyDown
' Declaration
Public ReadOnly Property CtrlKeyDown As Boolean
```

戻り値

Boolean の値です。Ctrl キーが押された場合は **True** になり、押されない場合は **False** になります。

解説

My.Computer.Keyboard.CtrlKeyDown プロパティは、ModifierKeys プロパティと同様の機能を提供します。

使用例

My.Computer.Keyboard.CtrlKeyDown プロパティを使用して、コンピュータの Ctrl キーが押されたかどうかを調べる例は次のようになります。

VB

```
If My.Computer.Keyboard.CtrlKeyDown Then
    MsgBox("CTRL key down")
Else
    MsgBox("CTRL key up")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Keyboard

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Keyboard オブジェクト

ModifierKeys
Microsoft.VisualBasic.Devices.Keyboard.CtrlKeyDown

My.Computer.Keyboard.NumLock プロパティ

NumLock キーがオンになっているかどうかを示す **Boolean** を取得します。

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.NumLock
' Declaration
Public ReadOnly Property NumLock As Boolean
```

戻り値

Boolean の値です。NumLock キーがオンになっている場合は **True** で、それ以外の場合は **False** です。

解説

NumLock キーの状態だけを返します。

使用例

次のコード例は、**My.Computer.Keyboard.NumLock** プロパティを使って、このコンピュータの NumLock キーがオンになっているかどうかを確認します。

VB

```
If My.Computer.Keyboard.NumLock Then
    MsgBox("NUM LOCK on")
Else
    MsgBox("NUM LOCK off")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Keyboard

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Keyboard オブジェクト

Microsoft.VisualBasic.Devices.Keyboard.NumLock

My.Computer.Keyboard.ScrollLock プロパティ

ScrollLock キーがオンになっているかどうかを示す **Boolean** を取得します。

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.ScrollLock
' Declaration
Public ReadOnly Property ScrollLock As Boolean
```

戻り値

Boolean 値。ScrollLock がオンになっている場合は **True**、そうでない場合は **False** を返します。

解説

ScrollLock キーの状態だけが返されます。

使用例

次の例は **My.Computer.Keyboard.ScrollLock** プロパティを使用して、コンピュータの ScrollLock キーがオンになっているかどうかを調べます。

VB

```
If My.Computer.Keyboard.ScrollLock Then
    MsgBox("SCROLL LOCK on")
Else
    MsgBox("SCROLL LOCK off")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Keyboard

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Keyboard オブジェクト

Microsoft.VisualBasic.Devices.Keyboard.ScrollLock

My.Computer.Keyboard.SendKeys メソッド

キーボードで入力しているかのように、1 つ以上のキーストロークをアクティブなウィンドウに送信します。

```
' Usage
My.Computer.Keyboard.SendKeys(keys)
My.Computer.Keyboard.SendKeys(keys ,wait)
' Declaration
Public Sub SendKeys( _
   ByVal keys As String _
)
' -or-
Public Sub SendKeys( _
   ByVal keys As String, _
   ByVal wait As Boolean _
)
```

パラメータ

keys

送信するキーを定義する **String** です。

wait

省略可能です。キーストロークが処理されるまで待ってから、アプリケーションが実行を継続するかどうかを指定する **Boolean** です。既定では **True** です。

例外

例外を引き起こす可能性のある状態を次に示します。

- ユーザーが部分的に信頼されており、必要なアクセス許可を持っていません (SecurityException)。

解説

My.Computer.Keyboard.SendKeys メソッドは、Send メソッドおよび SendWait メソッドと同様の機能を提供します。

wait 引数は、他のアプリケーションが終了しなければ自分のアプリケーションを継続できないようにする場合に便利です。

メモ：
他のアプリケーションをアクティブにするためのマネージ メソッドはありません。このクラスを現在のアプリケーションで使用して、キーを送信するウィンドウを手動で選択するか、**FindWindow** や **SetForegroundWindow** などの Windows API メソッドを使用して、他のアプリケーションにフォーカスを移してください。詳細については、「チュートリアル : Windows API の呼び出し」を参照してください。

keys 引数には単一のキー、またはキーと Alt、Ctrl、Shift との組み合わせ (またはこれらのキーの組み合わせ) を指定できます。各キーは 1 つ以上の文字で表現されます。たとえば、`a` は "a" の文字を表現し、`{ENTER}` であれば Enter キーを表現します。

キーと Shift を組み合わせるには、キー コードの前に `+` (正符号) を付けます。キーと Ctrl を組み合わせるには、キー コードの前に `^` (キャレット) を付けます。キーと Alt を組み合わせるには、キー コードの前に `%` (パーセント記号) を付けます。キーの繰り返しを指定するには、`{`*key number*`}` の形式を使用します。*key* と *number* の間には、空白を入れる必要があります。たとえば、`{LEFT 42}` は "← キーを 42 回押す" という意味になり、`{h 10}` は "'h' を 10 回押す" という意味になります。

キーを押しても表示されない文字 (Eneter や Tab など) を指定する際に使うコードを、次の表に示します。

キー	コード
BackSpace	{BACKSPACE} または {BS}
Break	{BREAK}
CapsLock	{CAPSLOCK}

Clear	{CLEAR}
Delete	{DELETE} または {DEL}
↓	{DOWN}
End	{END}
Enter (テンキー)	{ENTER}
Enter	~
Esc	{ESCAPE} または {ESC}
Help	{HELP}
Home	{HOME}
Ins	{INSERT}
←	{LEFT}
NumLock	{NUMLOCK}
PageDown	{PGDN}
PageUp	{PGUP}
Return	{RETURN}
→	{RIGHT}
ScrollLock	{SCROLLLOCK}
Tab	{TAB}
↑	{UP}
F1 ～ F15	{F1} ～ {F15}

使用例

次の例は **My.Computer.Keyboard.SendKeys** メソッドを使用して、**Shell** 関数を使って起動した外部の電卓アプリケーションにキーストロークを送信します。

VB

```
Dim ProcID As Integer
' Start the Calculator application, and store the process id.
ProcID = Shell("CALC.EXE", AppWinStyle.NormalFocus)
' Activate the Calculator application.
AppActivate(ProcID)
' Send the keystrokes to the Calculator application.
My.Computer.Keyboard.SendKeys("22", True)
My.Computer.Keyboard.SendKeys("*", True)
My.Computer.Keyboard.SendKeys("44", True)
My.Computer.Keyboard.SendKeys("=", True)
' The result is 22 * 44 = 968.
```

要求されたプロセスの識別子を持つアプリケーションが見つからない場合は、ArgumentException 例外が発生します。

Shell 関数の呼び出しは、完全に信頼されている必要があります (**SecurityException** クラス)。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Keyboard

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
UIPermission	ユーザー インターフェイスとクリップボードに関連するアクセス許可を制御します。関連する列挙値 : AllWindows。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

チュートリアル : Windows API の呼び出し

関連項目

My.Computer.Keyboard オブジェクト

Send

SendWait

Microsoft.VisualBasic.Devices.Keyboard.SendKeys(System.String,System.Boolean)

My.Computer.Keyboard.ShiftKeyDown プロパティ

Shift キーが押されているかどうかを示す **Boolean** を取得します。

```
' Usage
Dim value As Boolean = My.Computer.Keyboard.ShiftKeyDown
' Declaration
Public ReadOnly Property ShiftKeyDown As Boolean
```

戻り値

Boolean の値です。Shift キーが押されている場合は **True** で、それ以外の場合は **False** です。

解説

My.Computer.Keyboard.ShiftKeyDown プロパティは、ModifierKeys プロパティと同様の機能を提供します。

使用例

次のコード例は、**My.Computer.Keyboard.ShiftKeyDown** プロパティを使って、このコンピュータの Shift キーの 1 つが押されているかどうかを確認します。

VB

```
If My.Computer.Keyboard.ShiftKeyDown Then
    MsgBox("SHIFT key down")
Else
    MsgBox("SHIFT key up")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Keyboard

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Keyboard オブジェクト

ModifierKeys
Microsoft.VisualBasic.Devices.Keyboard.ShiftKeyDown

My.Computer.Mouse オブジェクト

ローカル コンピュータに接続されているマウスの形式および構成の情報を収集するためのプロパティを公開します。

解説

My.Computer.Mouse オブジェクトは、コンピュータのマウスに関する情報 (マウス ボタンがスワップされているかや、マウス ホイールに関する詳細など) を取得するための手段を提供します。

使用例

この例では、**My.Computer.Mouse.WheelExists** プロパティと **My.Computer.Mouse.WheelScrollLines** プロパティを使用して、マウスにスクロール ホイールがあるかどうかと、ホイール回転時のスクロール量を調べます。

VB

```
If My.Computer.Mouse.WheelExists Then
    Dim lines As Integer = My.Computer.Mouse.WheelScrollLines
    If lines > 0 Then
        MsgBox("Application scrolls " & _
            lines & " line(s) for each wheel turn.")
    Else
        MsgBox("Application scrolls " & _
            (-lines) & " page(s) for each wheel turn.")
    End If
Else
    MsgBox("Mouse has no scroll wheel.")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Mouse

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

参照

関連項目

My.Computer.Mouse オブジェクトのメンバ
My.Computer オブジェクト
Microsoft.VisualBasic.Devices.Mouse

My.Computer.Mouse オブジェクトのメンバ

My.Computer.Mouse オブジェクトは、ローカルのコンピュータにインストールされたマウスの形式や構成に関する情報を取得するためのプロパティを提供します。

プロパティ

プロパティ	説明
ButtonsSwapped	マウスの左ボタンと右ボタンの機能を入れ替えたかどうかを示す **Boolean** を取得します。 このプロパティは、非サーバー アプリケーションだけで使用できます。
WheelExists	マウスにスクロール ホイールが付いているかどうかを示す **Boolean** を取得します。 このプロパティは、非サーバー アプリケーションだけで使用できます。
WheelScrollLines	マウス ホイールを 1 ノッチ回転したときのスクロール量を示す数値を取得します。 このプロパティは、非サーバー アプリケーションだけで使用できます。

参照

関連項目

My.Computer.Mouse オブジェクト

My.Computer.Mouse.ButtonsSwapped プロパティ

マウスの左ボタンと右ボタンの機能を入れ替えたかどうかを示す **Boolean** を取得します。

```
' Usage
Dim value As Boolean = My.Computer.Mouse.ButtonsSwapped
' Declaration
Public ReadOnly Property ButtonsSwapped As Boolean
```

戻り値

Boolean 値。マウスの左ボタンと右ボタンの機能が入れ替えてある場合は **True**、そうでない場合は **False** になります。

例外

例外を引き起こす可能性のある状態を次に示します。

- コンピュータにマウスが接続されていない場合 (InvalidOperationException)。

解説

My.Computer.Mouse.ButtonsSwapped プロパティ

- MouseButtonsSwapped プロパティと類似した機能を提供します。
- コードを実行しているコンピュータに関する情報を提供します。

使用例

次の例は **My.Computer.Mouse.ButtonsSwapped** プロパティを使用して、マウスの左ボタンと右ボタンの機能が入れ替えてあるかどうかを判断します。

VB

```
If My.Computer.Mouse.ButtonsSwapped Then
    MsgBox("Functionality of the mouse buttons is swapped.")
Else
    MsgBox("Default functionality of the mouse buttons.")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Mouse

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**

Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照
関連項目
My.Computer.Mouse オブジェクト
Microsoft.VisualBasic.Devices.Mouse.ButtonsSwapped

My.Computer.Mouse.WheelExists プロパティ

マウスにスクロール ホイールが付いているかどうかを示す **Boolean** を取得します。

```
' Usage
Dim value As Boolean = My.Computer.Mouse.WheelExists
' Declaration
Public ReadOnly Property WheelExists As Boolean
```

戻り値

マウスにスクロール ホイールが付いていれば、Boolean の値は **True** になり、付いていなければ **False** になります。

例外

例外を引き起こす可能性のある状態を次に示します。

- コンピュータにマウスが接続されていない場合 (InvalidOperationException)。

解説

My.Computer.Mouse.WheelExists プロパティは、MouseWheelPresent プロパティと同様の機能を提供します。

My.Computer.Mouse.WheelScrollLines プロパティは、マウスにスクロール ホイールが付いていない場合に例外をスローするため、このプロパティにアクセスする前に **My.Computer.Mouse.WheelExists** プロパティの値を調べる必要があります。

My.Computer.Mouse.WheelExists プロパティは、コードを実行しているコンピュータに関する情報を提供します。

使用例

次の例は、**My.Computer.Mouse.WheelExists** プロパティを使用して、マウスにスクロール ホイールが付いているかどうかと、マウス ホイールを回転させたときのスクロール量を調べます。

VB

```
If My.Computer.Mouse.WheelExists Then
    Dim lines As Integer = My.Computer.Mouse.WheelScrollLines
    If lines > 0 Then
        MsgBox("Application scrolls " & _
            lines & " line(s) for each wheel turn.")
    Else
        MsgBox("Application scrolls " & _
            (-lines) & " page(s) for each wheel turn.")
    End If
Else
    MsgBox("Mouse has no scroll wheel.")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Mouse

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**

Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**不可**
Windows サービス	**可**
Web サイト	**不可**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Mouse オブジェクト

My.Computer.Mouse.WheelScrollLines プロパティ

Microsoft.VisualBasic.Devices.Mouse.WheelExists

My.Computer.Mouse.WheelScrollLines プロパティ

マウス ホイールを 1 ノッチ回転したときのスクロール量を示す数値を取得します。

```
' Usage
Dim value As Integer = My.Computer.Mouse.WheelScrollLines
' Declaration
Public ReadOnly Property WheelScrollLines As Integer
```

戻り値

マウス ホイールを 1 ノッチ回転したときのスクロール量を示す **Integer** です。正の値はその数の行だけスクロールすることを示し、負の値は一度に 1 画面だけスクロールすることを示します。

例外

例外を引き起こす可能性のある状態を次に示します。

- コンピュータにマウスが接続されていない場合 (InvalidOperationException)。
- マウスにスクロール ホイールがない場合 (**InvalidOperationException**)。

解説

このプロパティを使用すると、スクロール バーの付いた複数行のコントロールで、上方向または下方向に何行スクロールするかを判断できます。

My.Computer.Mouse.WheelScrollLines プロパティは、MouseWheelScrollLines プロパティと同様の機能を提供します。

このプロパティはマウスにスクロール ホイールがない場合に例外をスローするため、**My.Computer.Mouse.WheelExists** プロパティを調べて、マウスにスクロール ホイールが付いていることを確認する必要があります。

このプロパティは、コードを実行しているコンピュータに関する情報を提供します。

使用例

次の例では、マウスにスクロール ホイールがあるかどうかを判断し、**My.Computer.Mouse.WheelExists** プロパティを使用してマウス ホイールを回転させたときのスクロール量を調べます。

VB

```
If My.Computer.Mouse.WheelExists Then
    Dim lines As Integer = My.Computer.Mouse.WheelScrollLines
    If lines > 0 Then
        MsgBox("Application scrolls " & _
            lines & " line(s) for each wheel turn.")
    Else
        MsgBox("Application scrolls " & _
            (-lines) & " page(s) for each wheel turn.")
    End If
Else
    MsgBox("Mouse has no scroll wheel.")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Mouse

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**

クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Mouse オブジェクト

My.Computer.Mouse.WheelExists プロパティ

Microsoft.VisualBasic.Devices.Mouse.WheelScrollLines

My.Computer.Network オブジェクト

コンピュータが接続されているネットワークとやり取りするためのプロパティ、イベント、およびメソッドが用意されています。

処理手順

My.Computer.Network オブジェクトに関連するタスクの例を次の表に示します。

タスク	参照項目
ファイルをアップロードします。	方法 : Visual Basic でファイルをアップロードする
ファイルをダウンロードします。	方法 : Visual Basic でファイルをダウンロードする
接続の状態をチェックします。	方法 : Visual Basic で接続ステータスをチェックする
リモート コンピュータが使用可能かどうかを確認します。	方法 : Visual Basic でリモート コンピュータが利用可能かどうかを確認する

使用例

このコード例は、`Order.txt` というファイルを http://www.cohowinery.com/uploads にアップロードします。

VB

```
My.Computer.Network.UploadFile ( _
"C:\My Documents\Order.txt", _
"http://www.cohowinery.com/uploads.aspx")
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Network

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

参照

関連項目

My.Computer.Network オブジェクトのメンバ
My.Computer オブジェクト
My.Computer.Network.NetworkAvailabilityChanged イベント
Microsoft.VisualBasic.Devices.Network

My.Computer.Network オブジェクトのメンバ

ネットワークに関する情報を取得したり、ファイルのアップロードやダウンロードを実行したりするためのプロパティ、イベント、およびメソッドが用意されています。

プロパティ

プロパティ	説明
IsAvailable	ネットワークを使用できるかどうかを示します。

メソッド

メソッド	説明
DownloadFile	ファイルをリモートの場所からダウンロードします。
Ping	リモート コンピュータに ping を実行します。
UploadFile	ファイルをリモートの場所にアップロードします。

イベント

イベント	説明
NetworkAvailabilityChanged イベント	ネットワーク接続の状態が変化したことを示します。

参照

関連項目

My.Computer.Network オブジェクト

その他の技術情報

ネットワーク操作の実行

My.Computer.Network.DownloadFile メソッド

指定のリモート ファイルをダウンロードし、指定の場所に保存します。

```
' Usage
My.Computer.Network.DownloadFile(address ,destinationFileName)
My.Computer.Network.DownloadFile(address ,destinationFileName)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password ,showUI ,
connectionTimeout ,overwrite)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password ,showUI ,
connectionTimeout ,overwrite ,onUserCancel)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password ,showUI ,
connectionTimeout ,overwrite)
My.Computer.Network.DownloadFile(address ,destinationFileName ,userName ,password ,showUI ,
connectionTimeout ,overwrite ,onUserCancel)
My.Computer.Network.DownloadFile(address ,destinationFileName ,networkCredentials ,showUI ,
connectionTimeout ,overwrite)
My.Computer.Network.DownloadFile(address ,destinationFileName ,networkCredentials ,showUI ,
connectionTimeout ,overwrite ,onUserCancel)
' Declaration
Public Sub DownloadFile( _
   ByVal address As String, _
   ByVal destinationFileName As String _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As System.Uri, _
   ByVal destinationFileName As String _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As String, _
   ByVal destinationFileName As String, _
   ByVal userName As String, _
   ByVal password As String _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As System.Uri, _
   ByVal destinationFileName As String, _
   ByVal userName As String, _
   ByVal password As String _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As String, _
   ByVal destinationFileName As String, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal overwrite As Boolean _
)
' -or-
Public Sub DownloadFile( _
   ByVal address As String, _
   ByVal destinationFileName As String, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal overwrite As Boolean, _
```

```
    ByVal onUserCancel As UICancelOption _
)
' -or-
Public Sub DownloadFile( _
    ByVal address As System.Uri, _
    ByVal destinationFileName As String, _
    ByVal userName As String, _
    ByVal password As String, _
    ByVal showUI As Boolean, _
    ByVal connectionTimeout As Integer, _
    ByVal overwrite As Boolean _
)
' -or-
Public Sub DownloadFile( _
    ByVal address As System.Uri, _
    ByVal destinationFileName As String, _
    ByVal userName As String, _
    ByVal password As String, _
    ByVal showUI As Boolean, _
    ByVal connectionTimeout As Integer, _
    ByVal overwrite As Boolean, _
    ByVal onUserCancel As UICancelOption _
)
' -or-
Public Sub DownloadFile( _
    ByVal address As System.Uri, _
    ByVal destinationFileName As String, _
    ByVal networkCredentials As System.Net.ICredentials, _
    ByVal showUI As Boolean, _
    ByVal connectionTimeout As Integer, _
    ByVal overwrite As Boolean _
)
' -or-
Public Sub DownloadFile( _
    ByVal address As System.Uri, _
    ByVal destinationFileName As String, _
    ByVal networkCredentials As System.Net.ICredentials, _
    ByVal showUI As Boolean, _
    ByVal connectionTimeout As Integer, _
    ByVal overwrite As Boolean, _
    ByVal onUserCancel As UICancelOption _
)
```

パラメータ

address

String または Uri。ダウンロードするファイルのパスを、ファイル名とホスト アドレスを含めて指定します。必ず指定します。

destinationFileName

String です。ダウンロードしたファイルの保存時のファイル名とパスを指定します。必ず指定します。

userName

String です。認証するためのユーザー名を指定します。既定値は空の文字列 ("") です。

password

String です。認証するためのパスワードを指定します。既定値は空の文字列 ("") です。

showUI

Boolean です。操作の進行状況を表示するかどうかを指定します。既定値は **False** です。

connectionTimeout

Int32 です。タイムアウト時間をミリ秒を単位として指定します。既定値は 100 秒です。

overwrite

Boolean です。既存のファイルを上書きするかどうかを指定します。既定値は **False** です。

onUserCancel

UICancelOption です。*ShowUI* を **True** に設定した結果として表示されるダイアログ ボックスで、ユーザーが [キャンセル] または [いいえ] をクリックしたときに、どのような動作を実行するかを指定します。既定値は ThrowException です。

networkCredentials

ICredentials です。資格情報を指定します。

例外

例外がスローされる可能性のある状況を次に示します。

- ドライブ名が無効です (ArgumentException)。
- *destinationFileName* がスラッシュで終わっています (**ArgumentException**)。
- *overwrite* が **False** に設定されていますが、コピー先のファイルが既に存在します (IOException)。
- サーバーが、指定された *connectionTimeout* の間に応答しません (TimeoutException)。
- 認証がエラーになります (SecurityException)。
- ユーザーに必要なアクセス許可がありません (**SecurityException**)。
- 要求が Web サイトによって拒否されます (WebException)。

解説

showUI が **True** に設定されている場合、操作の進行状況を示すダイアログ ボックスが表示されます。このダイアログ ボックスにある [キャンセル] ボタンをクリックすると、操作をキャンセルできます。このダイアログ ボックスはモーダルではないため、プログラムの他のウィンドウへの入力を妨げません。

サーバーが指定の *connectionTimeout* の間に応答しない場合、操作はキャンセルされ、例外がスローされます。

DownloadFile は、アプリケーションでネットワークのトレースが有効になっている場合に、トレース情報を出力します。詳細については、「ネットワーク トレースの有効化」を参照してください。

メモ :
DownloadFile メソッドは、オプションの HTTP ヘッダーを送信しません。サーバーによっては、オプションのユーザー エージェント ヘッダーが見つからない場合に 500 (内部サーバー エラー) を返すこともあります。オプションのヘッダーを送信するには、WebClient クラスを使用して要求を作成する必要があります。詳細については、「Visual Basic による .NET Framework でのネットワーク操作」を参照してください。

セキュリティに関するメモ :
FTP プロトコルは、パスワードを含む情報をプレーンテキストで送信するため、重要な情報の送信には使用しないでください。

処理手順

My.Computer.Network.DownloadFile メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
ファイルをダウンロードします。	方法 : Visual Basic でファイルをダウンロードする

使用例

次のコード例は、`WineList.txt` というファイルを `http://www.cohowinery.com/downloads` からダウンロードし、`C:\Documents and Settings\All Users\Documents` に保存します。

VB

```
My.Computer.Network.DownloadFile _
("http://www.cohowinery.com/downloads/WineList.txt", _
"C:\Documents and Settings\All Users\Documents\WineList.txt")
```

次のコード例は、タイアウト時間を 500 ミリ秒に設定して、`WineList.txt` というファイルを `http://www.cohowinery.com/downloads` からダウンロードし、`C:\Documents and Settings\All Users\Documents` に保存します。

VB

```
My.Computer.Network.DownloadFile _
("http://www.cohowinery.com/downloads/", _
"C:\Documents and Settings\All Users\Documents\WineList.txt", _
"", "", False, 500, True)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Network

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用可/不可
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	**あり**
Windows サービス	**あり**
Web サイト	**あり**

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
UIPermission	ユーザー インターフェイスおよびクリップボードに関連するアクセス許可を制御します。関連する列挙値 : AllWindows。
WebPermission	HTTP インターネット リソースへのアクセス許可を制御します。関連する列挙値 : **Unrestricted**。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でファイルをダウンロードする
方法 : Visual Basic でリモート コンピュータが利用可能かどうかを確認する
方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.Network オブジェクト
System.Uri
System.Net.ICredentials
Microsoft.VisualBasic.Devices.Network.DownloadFile

My.Computer.Network.IsAvailable プロパティ

コンピュータがネットワークに接続されているかどうかを示します。

```
' Usage
Dim value As Boolean = My.Computer.Network.IsAvailable
' Declaration
Public ReadOnly Property IsAvailable As Boolean
```

戻り値

Boolean.

解説

IsAvailable は、**Click Once** アプリケーションから呼び出された場合、またはユーザーが NetworkInformationPermission を持たない場合は、必ず **False** を返します。

処理手順

My.Computer.Network.IsAvailable プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
コンピュータがネットワークに接続されているかどうかを調べる	方法 : Visual Basic で接続ステータスをチェックする

使用例

次の例は、プロパティのステータスを調べて報告します。

VB

```
If My.Computer.Network.IsAvailable = True Then
    MsgBox("Computer is connected.")
Else
    MsgBox("Computer is not connected.")
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Network

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
WebPermission	HTTP インターネット リソースにアクセスするための権限を制御します。関連する列挙値 : **Unrestricted**。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でリモート コンピュータが利用可能かどうかを確認する

関連項目

My.Computer.Network オブジェクト

Microsoft.VisualBasic.Devices.Network.IsAvailable

My.Computer.Network.Ping メソッド

指定されたサーバーの ping を実行します。

```
' Usage
Dim value As Boolean = My.Computer.Network.Ping(hostNameOrAddress)
Dim value As Boolean = My.Computer.Network.Ping(address)
Dim value As Boolean = My.Computer.Network.Ping(hostNameOrAddress ,timeout)
Dim value As Boolean = My.Computer.Network.Ping(address ,timeout)
' Declaration
Public Function Ping( _
   ByVal hostNameOrAddress As String _
) As Boolean
' -or-
Public Function Ping( _
   ByVal address As System.Uri _
) As Boolean
' -or-
Public Function Ping( _
   ByVal hostNameOrAddress As String, _
   ByVal timeout As Integer _
) As Boolean
' -or-
Public Function Ping( _
   ByVal address As System.Uri, _
   ByVal timeout As Integer _
) As Boolean
```

パラメータ

hostNameOrAddress

ping を実行するサーバーの URL、コンピュータ名、または IP 番号を指定する **String** です。必ず指定します。

address

ping を実行するサーバーの URI を指定する Uri です。必ず指定します。

timeout

目的のサーバーにコンタクトするまでの時間のしきい値をミリ秒で指定する Int32 です。既定値は 500 です。必ず指定します。

戻り値

処理が成功したかどうかを示す **Boolean** です。

例外

例外がスローされる可能性のある状態を次に示します。

- ネットワーク接続が利用できない場合 (InvalidOperationException)。
- URL が有効でない場合 (PingException)。

解説

Ping メソッドはリモート コンピュータが利用可能かどうかを調べるためのフェール セーフなメソッドではありません。ターゲット コンピュータの ping ポートがオフになっている場合もあれば、ping 要求がファイアウォールやルーターによってブロックされている場合もあります。

Ping メソッドに渡すアドレスは、DNS による解決が可能であることが必要です。また、先頭に "http://" を付けることはできません。

処理手順

My.Computer.Network.Ping メソッドに関連するタスクの例を次の表に示します。

目的	参照項目

リモート コンピュータまたはリモート ホストが利用可能かどうかを調べる	方法 : Visual Basic でリモート コンピュータが利用可能かどうかを確認する

使用例

次の例は、**Ping** メソッドが **True** を返すかどうかを調べることによって、サーバーが ping 可能かどうかを報告します。

VB

```
If My.Computer.Network.Ping("198.01.01.01") Then
   MsgBox("Server pinged successfully.")
Else
   MsgBox("Ping request timed out.")
End If
```

"198.01.01.01" は、ping するサーバーの IP アドレス、URL、またはコンピュータ名に変更してください。

次の例は、**Ping** メソッドが **True** を返すかどうかを調べ、タイムアウト時間に 1000 ミリ秒を指定することによって、サーバーが ping 可能かどうかを報告します。

VB

```
If My.Computer.Network.Ping("www.cohowinery.com",1000) Then
   MsgBox("Server pinged successfully.")
Else
   MsgBox("Ping request timed out.")
End If
```

"www.cohowinery.com" は、ping するサーバーの IP アドレス、URL、またはコンピュータ名に変更してください。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Network

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。

SecurityPermission	コードに適用されたセキュリティ アクセス許可のセットを記述します。関連する列挙値 : ControlPrincipal。
WebPermission	HTTP インターネット リソースにアクセスするための権限を制御します。関連する列挙値 : **Unrestricted**。
SocketPermission	トランスポート アドレスでの接続の確立と承認のための権限を制御します。関連する列挙値 : **Unrestricted**。
PerformanceCounterPermission	Windows NT パフォーマンス カウンタ コンポーネントへのアクセスを制御します。関連する列挙値 : **Unrestricted**。
NetworkInformationPermission	ローカル コンピュータのネットワーク情報およびトラフィックの統計情報へのアクセスを制御します。関連する列挙値 : Ping。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

My.Computer.Network オブジェクト
System.Uri
Microsoft.VisualBasic.Devices.Network.Ping

My.Computer.Network.NetworkAvailabilityChanged イベント

ネットワークの可用性に変化があったときに発生します。

```
' Usage
Public Sub MyComputerNetwork_NetworkAvailabilityChanged( _
   ByVal sender As Object, _
   ByVal e As Devices.NetworkAvailableEventArgs _
)
End Sub
Sub Handle_NetworkAvailabilityChanged()
   AddHandler My.Computer.Network.NetworkAvailabilityChanged, _
      AddressOf MyComputerNetwork_NetworkAvailabilityChanged
End Sub
' Declaration
Public Event NetworkAvailabilityChanged( _
   ByVal sender As Object, _
   ByVal e As Devices.NetworkAvailableEventArgs _
)
```

パラメータ

sender

必ず指定します。イベントの発生元の **Object** を指定します。

e

必ず指定します。ネットワークの可用性に関する情報が格納される NetworkAvailableEventArgs オブジェクトを指定します。

解説

アプリケーションでは、ネットワークの可用性が変化するたびに **NetworkAvailabilityChanged** イベントが生成されます。*e* パラメータの IsNetworkAvailable プロパティを使って、ネットワーク接続の新しい状態を取得できます。ネットワーク接続の現在の状態を取得するには、My.Computer.Network.IsAvailable プロパティ を使用します。

Windows フォーム アプリケーションでは、このイベントはアプリケーションのメイン スレッド上に他のユーザー インターフェイス イベントと共に発生します。そのため、イベント ハンドラはアプリケーションのユーザー インターフェイスに直接アクセスできます。ただし、アプリケーションが他のユーザー インターフェイス イベントを処理している最中にこのイベントが発生した場合は、他のイベント ハンドラが完了するまで、または My.Application.DoEvents メソッドが呼び出されるまでは、このイベントは処理されません。

NetworkAvailabilityChanged イベントの処理には **Handles** ステートメントを使用できません。**AddHandler** ステートメントを使用する必要があります。

Windows フォーム アプリケーションでは、このイベントと同じ機能を持つ **NetworkAvailabilityChanged** イベントが **My.Application** オブジェクトから公開されますが、このイベントは **Handles** ステートメントで処理できます。詳細については、「My.Application.NetworkAvailabilityChanged イベント」を参照してください。

メモ：

多くのネットワーク ハブは、規模の大きな方のネットワークから切断された場合でも、ネットワーク接続を提供します。したがって、回線がつながっている状態では、このイベントはコンピュータとハブを結ぶ接続に変化があったことを意味します。

メモ：

NetworkAvailabilityChanged イベントは、Windows 95 または Windows 98 で実行されるアプリケーションまたは管理者ではないユーザーによって Windows 2000 で実行されるアプリケーションでは発生しません。これらのプラットフォームでアプリケーションを実行する場合は、My.Computer.Network.IsAvailable プロパティ を使ってネットワークの可用性をチェックしてください。

使用例

次のコード例は、**My.Computer.Network.NetworkAvailabilityChanged** イベントを使ってフォームまたはコントロールのユーザー インターフェイスを更新しています。

VB

```
Private Sub DisplayAvailability(ByVal available As Boolean)
    Label1.Text = available.ToString
End Sub

Private Sub MyComputerNetwork_NetworkAvailabilityChanged( _
    ByVal sender As Object, _
    ByVal e As Devices.NetworkAvailableEventArgs)

    DisplayAvailability(e.IsNetworkAvailable)
End Sub

Private Sub Handle_NetworkAvailabilityChanged()
    AddHandler My.Computer.Network.NetworkAvailabilityChanged, _
       AddressOf MyComputerNetwork_NetworkAvailabilityChanged
    DisplayAvailability(My.Computer.Network.IsAvailable)
End Sub
```

このコードでは、フォームまたはコントロールに `Label1` という名前の Label があることを前提としています。`Handle_NetworkAvailabilityChanged` メソッドを呼び出して、ラベルを初期化し、イベント ハンドラをフックする必要があります。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Network

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類別の可用性

プロジェクトの種類	使用可/不可
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	**あり**
Windows サービス	**あり**
Web サイト	**あり**

アクセス許可

アクセス許可は不要です。

参照

関連項目

My.Computer.Network オブジェクト
My.Application.NetworkAvailabilityChanged イベント
Microsoft.VisualBasic.Devices.NetworkAvailableEventArgs

My.Computer.Network.UploadFile メソッド

指定されたファイルを、指定されたホスト アドレスに送信します。

```
' Usage
My.Computer.Network.UploadFile(sourceFileName ,address)
My.Computer.Network.UploadFile(sourceFileName ,address)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password ,showUI ,connect
ionTimeout)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password ,showUI ,connect
ionTimeout ,onUserCancel)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password ,showUI ,connect
ionTimeout)
My.Computer.Network.UploadFile(sourceFileName ,address ,userName ,password ,showUI ,connect
ionTimeout ,onUserCancel)
My.Computer.Network.UploadFile(sourceFileName ,address ,networkCredentials ,showUI ,connect
ionTimeout)
My.Computer.Network.UploadFile(sourceFileName ,address ,networkCredentials ,showUI ,connect
ionTimeout ,onUserCancel)
' Declaration
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As String _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As String, _
   ByVal userName As String, _
   ByVal password As String _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal userName As String, _
   ByVal password As String _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As String, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As String, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal onUserCancel As UICancelOption _
)
```

```
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal userName As String, _
   ByVal password As String, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal onUserCancel As UICancelOption _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal networkCredentials As System.Net.ICredentials, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer _
)
' -or-
Public Sub UploadFile( _
   ByVal sourceFileName As String, _
   ByVal address As System.Uri, _
   ByVal networkCredentials As System.Net.ICredentials, _
   ByVal showUI As Boolean, _
   ByVal connectionTimeout As Integer, _
   ByVal onUserCancel As UICancelOption _
)
```

パラメータ

sourceFileName

アップロードするファイルのパスと名前を指定する **String** です。必ず指定します。

address

String または Uri。送信先サーバーの URL、IP アドレス、または URI です。必ず指定します。

userName

String です。認証するためのユーザー名を指定します。既定値は空の文字列です ("")。

password

String です。認証するためのパスワードを指定します。既定値は空の文字列です ("")。

showUI

処理の進行状況を表示するかどうかを指定する **Boolean** です。既定値は **False** です。

connectionTimeout

タイムアウト時間をミリ秒で指定する Int32 です。既定値は 100 秒です。

onUserCancel

ユーザーが [キャンセル] をクリックしたときに起こすアクションを指定する UICancelOption です。既定値は ThrowException です。

networkCredentials

認証に必要な資格情報を指定する ICredentials です。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- ローカル ファイルのパスが無効です (ArgumentException)。
- *connectionTimeout* が 0 以下です (**ArgumentException**)。
- *address* にファイル名が含まれていません (InvalidOperationException)。
- 認証に失敗しました (SecurityException)。
- ユーザーに必要なアクセス許可がありません (**SecurityException**)。
- 接続タイム アウトです (TimeoutException)。
- Website が要求を拒否しました (WebException)。

解説

showUI に **True** を設定すると、処理の進行状況を示すダイアログ ボックスが表示されます。ここにはユーザーが処理をキャンセルするときにクリックする [キャンセル] ボタンも表示されます。このダイアログ ボックスはモーダルではないため、ユーザーがプログラム内の他のウィンドウに入力できなくなることはありません。

connectionTimeout で指定された時間内にサーバーが応答しない場合は、処理がキャンセルされ、例外がスローされます。

My.Computer.Network.UploadFile は、アプリケーションのネットワーク トレースが可能になっている場合に、トレース情報を出力します。詳細については、「ネットワーク トレースの有効化」を参照してください。

セキュリティに関するメモ：
FTP プロトコルは、情報を (パスワードを含めて) プレーンテキストで送信します。このため、機密情報を転送する際には使用しないでください。

処理手順

My.Computer.Network.UploadFile メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
ファイルのアップロード	方法 : Visual Basic でファイルをアップロードする

使用例

次の例では、ファイル `Order.txt` を `http://www.cohowinery.com/uploads` にアップロードします。

VB

```
My.Computer.Network.UploadFile ( "C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx")
```

次の例は、ファイル `Order.txt` を `http://www.cohowinery.com/uploads` にアップロードします。その際、ユーザー名やパスワードは指定せず、アップロードの進行状況を表示し、タイムアウト時間を 500 ミリ秒に設定します。

VB

```
My.Computer.Network.UploadFile ("C:\My Documents\Order.txt", _
"http://www.cohowinery.com/upload.aspx","","",True,500)
```

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Network

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**あり**

クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	**あり**
Windows サービス	**あり**
Web サイト	**あり**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
UIPermission	ユーザー インターフェイスとクリップボードに関連するアクセス許可を制御します。関連する列挙値 : AllWindows。
WebPermission	HTTP インターネット リソースにアクセスするための権限を制御します。関連する列挙値 : **Unrestricted**。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でファイルをダウンロードする

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.Network オブジェクト

System.Uri

System.Net.ICredentials

Microsoft.VisualBasic.FileIO.UICancelOption

Microsoft.VisualBasic.Devices.Network.UploadFile

My.Computer.Ports オブジェクト

コンピュータのシリアル ポートにアクセスするためのプロパティとメソッドを提供します。

解説

My.Computer.Ports オブジェクトには、.NET Framework のシリアル ポート クラスである SerialPort にアクセスするための、わかりやすいエントリ ポイントが用意されています。

処理手順

My.Computer.Ports オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
シリアル ポートに接続されているモデムをダイヤルする	方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする
シリアル ポートに文字列を送信する	方法 : Visual Basic でシリアル ポートに文字列を送信する
シリアル ポートから文字列を受け取る	方法 : Visual Basic でシリアル ポートから文字列を受信する
利用可能なシリアル ポートを表示する	方法 : Visual Basic で利用可能なシリアル ポートを表示する

使用例

コンピュータの COM1 シリアル ポートに文字列を送る方法を、次の例に示します。

Using ブロックを使用すると、例外が生成された場合でもアプリケーションでシリアル ポートを閉じることができます。シリアル ポートを操作するコードは、このブロックまたは **Try...Catch...Finally** ブロック内で、Close メソッドを使用するための呼び出しと一緒に定義する必要があります。

WriteLine メソッドはデータをシリアル ポートに送信します。

VB

```
Sub SendSerialData(ByVal data As String)
    ' Send strings to a serial port.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        com1.WriteLine(data)
    End Using
End Sub
```

詳細については、「方法 : Visual Basic でシリアル ポートに文字列を送信する」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Ports

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可

Windows サービス	**可**
Web サイト	不可

参照

処理手順

方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする
方法 : Visual Basic でシリアル ポートに文字列を送信する
方法 : Visual Basic でシリアル ポートから文字列を受信する
方法 : Visual Basic で利用可能なシリアル ポートを表示する

関連項目

My.Computer.Ports オブジェクトのメンバ
My.Computer オブジェクト
Microsoft.VisualBasic.Devices.Ports

My.Computer.Ports オブジェクトのメンバ

My.Computer.Ports オブジェクトには、コンピュータのシリアル ポートにアクセスするためのプロパティとメソッドが用意されています。

プロパティ

SerialPortNames	コンピュータ上にあるシリアル ポートの名前のコレクションを取得します。 このプロパティは、Web アプリケーションでのみ使用可能です。

メソッド

OpenSerialPort	SerialPort オブジェクトを作成して開きます。 このメソッドは、Web アプリケーションには使用できません。

参照

関連項目

My.Computer.Ports オブジェクト

My.Computer.Ports.OpenSerialPort メソッド

SerialPort オブジェクトを作成して開きます。

```
' Usage
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName)
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName ,baudRa
te)
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName ,baudRa
te ,parity)
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName ,baudRa
te ,parity ,dataBits)
Dim value As System.IO.Ports.SerialPort = My.Computer.Ports.OpenSerialPort(portName ,baudRa
te ,parity ,dataBits ,stopBits)
' Declaration
Public Function OpenSerialPort( _
   ByVal portName As String _
) As System.IO.Ports.SerialPort
' -or-
Public Function OpenSerialPort( _
   ByVal portName As String, _
   ByVal baudRate As Integer _
) As System.IO.Ports.SerialPort
' -or-
Public Function OpenSerialPort( _
   ByVal portName As String, _
   ByVal baudRate As Integer, _
   ByVal parity As System.IO.Ports.Parity _
) As System.IO.Ports.SerialPort
' -or-
Public Function OpenSerialPort( _
   ByVal portName As String, _
   ByVal baudRate As Integer, _
   ByVal parity As System.IO.Ports.Parity, _
   ByVal dataBits As Integer _
) As System.IO.Ports.SerialPort
' -or-
Public Function OpenSerialPort( _
   ByVal portName As String, _
   ByVal baudRate As Integer, _
   ByVal parity As System.IO.Ports.Parity, _
   ByVal dataBits As Integer, _
   ByVal stopBits As System.IO.Ports.StopBits _
) As System.IO.Ports.SerialPort
```

パラメータ

portName

String です。必ず指定します。開くポートの名前を指定します。

baudRate

Integer です。ポートのボー レートを指定します。

parity

Parity です。ポートのパリティを指定します。

dataBits

Integer です。ポートのデータ ビット設定を指定します。

stopBits

StopBits です。ポートのストップ ビット設定を指定します。

戻り値

指定の引数に基づいて設定された、開かれた **SerialPort** オブジェクト

例外

例外を引き起こす可能性のある状態を次に示します。

- *portName* 引数が **Nothing** または空の文字列である (ArgumentNullException)。
- *baudRate* 引数または *dataBits* 引数が 0 または負数である (ArgumentException)。
- *parity* 引数の型が **Parity** 列挙値のどれでもない (InvalidEnumArgumentException)。
- *stopBits* 引数の型が **StopBits** 列挙値のどれでもない (**InvalidEnumArgumentException**)。

解説

My.Computer.Ports.OpenSerialPort メソッドは、**SerialPort** オブジェクトを作成して開きます。**OpenSerialPort** メソッドに渡した引数により、**SerialPort** オブジェクトの設定が決まります。

SerialPort オブジェクトを使用し終わったら、このオブジェクトを閉じる必要があります。このオブジェクトを明示的に閉じるには Close メソッドを使用し、暗黙的に閉じるには **Using** ステートメントを使用します。詳細については、このページの例を参照してください。

処理手順

My.Computer.Ports.OpenSerialPort メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
シリアル ポートに接続されているモデムをダイヤルする	方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする
シリアル ポートに文字列を送信する	方法 : Visual Basic でシリアル ポートに文字列を送信する
シリアル ポートから文字列を受信する	方法 : Visual Basic でシリアル ポートから文字列を受信する

使用例

この例は、コンピュータの COM1 シリアル ポートに文字列を送信する方法を示します。

Using ブロックを使用すると、例外が生成される場合でもアプリケーションでシリアル ポートを閉じることができます。シリアル ポートを操作するコードは、このブロック内に記述するか、**Close** メソッドの呼び出しを含んだ **Try...Catch...Finally** ブロック内に記述する必要があります。

WriteLine メソッドはシリアル ポートにデータを送信します。

VB

```
Sub SendSerialData(ByVal data As String)
    ' Send strings to a serial port.
    Using com1 As IO.Ports.SerialPort = _
            My.Computer.Ports.OpenSerialPort("COM1")
        com1.WriteLine(data)
    End Using
End Sub
```

詳細については、「方法 : Visual Basic でシリアル ポートに文字列を送信する」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Ports

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**

クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする
方法 : Visual Basic でシリアル ポートに文字列を送信する
方法 : Visual Basic でシリアル ポートから文字列を受信する

関連項目

My.Computer.Ports オブジェクト
My.Computer.Ports.SerialPortNames プロパティ
Using ステートメント (Visual Basic)
System.IO.Ports.Parity
System.IO.Ports.StopBits
System.IO.Ports.SerialPort
Microsoft.VisualBasic.Devices.Ports.OpenSerialPort(System.String)

My.Computer.Ports.SerialPortNames プロパティ

コンピュータのシリアル ポートの名前のコレクションを取得します。

```
' Usage
Dim value As System.Collections.Generic.ReadOnlyCollection`1(Of String) = My.Computer.Ports
.SerialPortNames
' Declaration
Public ReadOnly Property SerialPortNames As System.Collections.Generic.ReadOnlyCollection`1
(Of String)
```

戻り値

コンピュータのシリアル ポートの名前のコレクションです。

解説

My.Computer.Ports.SerialPortNames プロパティは、コンピュータのシリアル ポートの名前のコレクションを取得します。

メモ :
Windows 98 を実行しているコンピュータの場合、**My.Computer.Ports.SerialPortNames** から返されるポートの名前が正しくないことがあります。アプリケーション エラーを防ぐには、ポートの名前を使ってポートを開くときに **Try...Catch...Finally** ステートメントまたは **Using** ステートメントなどの例外処理を使用します。

処理手順

My.Computer.Ports.SerialPortNames プロパティに関連するタスクの例を次の表に示します。

タスク	参照項目
使用可能なシリアル ポートを表示します。	方法 : Visual Basic で利用可能なシリアル ポートを表示する

使用例

この例では、**My.Computer.Ports.SerialPortNames** プロパティから返されるすべての文字列をループします。この文字列は、コンピュータで使用可能なシリアル ポートの名前です。

通常、使用可能なシリアル ポートの一覧からアプリケーションで使用するポートをユーザーが選択します。この例では、シリアル ポートの名前は ListBox コントロールに格納されます。詳細については、「ListBox コントロール (Windows フォーム)」を参照してください。

VB

```
Sub GetSerialPortNames()
    ' Show all available COM ports.
    For Each sp As String In My.Computer.Ports.SerialPortNames
        ListBox1.Items.Add(sp)
    Next
End Sub
```

この例で必要な要素は次のとおりです。

- System 名前空間への参照
- フォーム上の `ListBox1` という名前の **ListBox** コントロール

詳細については、「方法 : Visual Basic で利用可能なシリアル ポートを表示する」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.Devices

クラス : Ports

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	不可
Windows サービス	**可**
Web サイト	不可

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic で利用可能なシリアル ポートを表示する

関連項目

My.Computer.Ports オブジェクト
My.Computer.Ports.OpenSerialPort メソッド
Try...Catch...Finally ステートメント (Visual Basic)
Using ステートメント (Visual Basic)
ReadOnlyCollection
Microsoft.VisualBasic.Devices.Ports.SerialPortNames

My.Computer.Registry オブジェクト

レジストリを操作するプロパティとメソッドを提供します。

解説

これらのプロパティを使用するには、RegistryPermissionAccess 列挙体によって許可された読み取りと書き込みのアクセス権限が必要です。完全に信頼された状態で実行中のコード (既定のセキュリティ ポリシーでは、ユーザーのローカル ハード ドライブにインストールされているすべてのコード) は、レジストリにアクセスするために必要なアクセス許可を持っています。詳細については、「RegistryPermission クラス」を参照してください。

処理手順

My.Computer.Registry オブジェクトに関連するタスクの例を次の表に示します。

タスク	参照項目
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
レジストリ キーの値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
レジストリ キーにデータを書き込む	方法 : Visual Basic でレジストリ キーの値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する
レジストリからデータを読み取ります。	方法 : Visual Basic で、レジストリ キーから値を読み取る

使用例

この例では、`HKEY_CURRENT_USER\Software\MyApp` から値 `Name` を読み取り、メッセージ ボックスにこの値を表示します。

VB

```
Dim readValue As Object
        readValue = My.Computer.Registry.GetValue _
        ("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothing)
        MsgBox("The value is " & CStr(readValue))
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス RegistryProxy (Registry へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

チュートリアル : レジストリ キーの作成と値の変更

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクトのメンバ

My.Computer.Registry.ClassesRoot プロパティ

My.Computer.Registry.CurrentConfig プロパティ

My.Computer.Registry.CurrentUser プロパティ

My.Computer.Registry.DynData プロパティ

My.Computer.Registry.LocalMachine プロパティ

My.Computer.Registry.PerformanceData プロパティ

My.Computer.Registry.Users プロパティ

My.Computer.Registry.GetValue メソッド

My.Computer.Registry.SetValue メソッド

My.Computer オブジェクト

Microsoft.Win32.Registry

概念

一般的なレジストリタスク

My.Computer.Registry オブジェクトのメンバ

My.Computer.Registry オブジェクトには、レジストリ キーを操作するためのプロパティおよびメソッドが用意されています。

プロパティ

ClassesRoot	HKEY_CLASSES_ROOT と同じです。
CurrentConfig	HKEY_CURRENT_CONFIG と同じです。
CurrentUser	HKEY_CURRENT_USER と同じです。
DynData	HKEY_DYNAMIC_DATA と同じです。
LocalMachine	HKEY_LOCAL_MACHINE と同じです。
PerformanceData	HKEY_PERFORMANCE_DATA と同じです。
Users	HKEY_USERS と同じです。

メソッド

GetValue	レジストリ キーから値を取得します。
SetValue	レジストリ キーの値を設定します。

参照

処理手順

チュートリアル : レジストリ キーの作成と値の変更

関連項目

My.Computer.Registry オブジェクト

概念

一般的なレジストリ タスク

My.Computer.Registry.ClassesRoot プロパティ

HKEY_CLASSES_ROOT へのアクセスを提供する RegistryKey 型を返します。

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.ClassesRoot
' Declaration
Public ReadOnly Property ClassesRoot As Microsoft.Win32.RegistryKey
```

戻り値

RegistryKey

解説

My.Computer.Registry オブジェクトには、レジストリ キーを操作するためのメソッドおよびプロパティが用意されています。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

HKEY_CLASSES_ROOT は、主に Windows のファイルの関連付けに関する情報を格納するために使用されます。

System.Security.Permissions 名前空間にある RegistryPermission クラスは、レジストリ内の値へのアクセスを制御します。レジストリ変数は、**RegistryPermission** を持たないコードがアクセスできるメモリ位置に格納することはできません。同様に、アクセス許可が付与されるとき、作業を行うために必要な権限だけが付与されます。

レジストリ アクセス許可のアクセス値は、RegistryPermissionAccess 列挙型で定義します。各メンバを次の表に示します。

値	説明
AllAccess	レジストリ変数へのアクセスの作成、読み取り、書き込みを行います。
[作成]	レジストリ変数へのアクセスを作成します。
NoAccess	レジストリ変数へのアクセスはありません。
読み取り	レジストリ変数への読み込みアクセス。
Write	レジストリ変数への書き込みアクセス。

処理手順

My.Computer.Registry オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する
特定のレジストリ キーに値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
レジストリ キーから値を読み込む	方法 : Visual Basic で、レジストリ キーから値を読み取る
レジストリ キーの値を設定する	方法 : Visual Basic でレジストリ キーの値を設定する

使用例

この例では、`ClassesRoot` レジストリ キーのサブキーの名前を取得し、これを `ListBox1` に追加します。

VB

```
Dim keyList As System.Collections.IEnumerable
keyList = My.Computer.Registry.ClassesRoot.GetSubKeyNames()
For Each keyName As String In keyList
```

```
    ListBox1.Items.Add(keyName)
Next
```

この例では、`ListBox1` という名前の **ListBox** がプロジェクト内にあることが必要です。

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス RegistryProxy (Registry へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

アクセス許可は不要です。

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.ClassesRoot

概念

一般的なレジストリ タスク

セキュリティとレジストリ

My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Computer.Registry.CurrentConfig プロパティ

HKEY_CURRENT_CONFIG へのアクセスを提供する RegistryKey 型を返します。

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.CurrentConfig
' Declaration
Public ReadOnly Property CurrentConfig As Microsoft.Win32.RegistryKey
```

戻り値

RegistryKey

解説

My.Computer.Registry オブジェクトには、レジストリ キーを操作するためのメソッドおよびプロパティが用意されています。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

HKEY_CURRENT_CONFIG は、主にコンピュータ上のデバイスの設定を格納するために使用されます。

System.Security.Permissions 名前空間にある RegistryPermission クラスは、レジストリ内の値へのアクセスを制御します。レジストリ変数は、**RegistryPermission** を持たないコードがアクセスできるメモリ位置に格納することはできません。同様に、アクセス許可が付与されるとき、作業を行うために必要な権限だけが付与されます。

レジストリ アクセス許可のアクセス値は、RegistryPermissionAccess 列挙型で定義します。各メンバを次の表に示します。

値	説明
AllAccess	レジストリ変数へのアクセスの作成、読み取り、書き込みを行います。
Create	レジストリ変数へのアクセスを作成します。
NoAccess	レジストリ変数へのアクセスはありません。
Read	レジストリ変数への読み込みアクセス。
Write	レジストリ変数への書き込みアクセス。

処理手順

My.Computer.Registry オブジェクトに関連するタスクの例を次の表に示します。

作業	参照項目
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する
特定のレジストリ キーに値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
レジストリ キーから値を読み込む	方法 : Visual Basic で、レジストリ キーから値を読み取る
レジストリ キーの値を設定する	方法 : Visual Basic でレジストリ キーの値を設定する

使用例

この例では、サブキー `MyDeviceSettings` を作成します。

VB

```
My.Computer.Registry.CurrentConfig.CreateSubKey("MyDeviceSettings")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス RegistryProxy (Registry へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

アクセス許可は不要です。

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.CurrentConfig

概念

一般的なレジストリ タスク

セキュリティとレジストリ

My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Computer.Registry.CurrentUser プロパティ

HKEY_CURRENT_USER へのアクセスを可能にする RegistryKey 型を返します。

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.CurrentUser
' Declaration
Public ReadOnly Property CurrentUser As Microsoft.Win32.RegistryKey
```

戻り値

RegistryKey

解説

My.Computer.Registry オブジェクトには、レジストリ キーを操作するためのメソッドおよびプロパティが用意されています。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

HKEY_CURRENT_USER は主に Windows での色やフォントなど、ユーザーごとの設定を格納するために使用されます。キーはユーザーごとに異なります。

System.Security.Permissions 名前空間にある RegistryPermission クラスは、レジストリ変数へのアクセスを制御します。レジストリ変数は、**RegistryPermission** を持たないコードがアクセスできるメモリ位置に格納することはできません。同様に、アクセス許可を付与するときには、作業を行うために必要な最小限の権限を付与してください。

レジストリ アクセス許可のアクセス値は、RegistryPermissionAccess 列挙型で定義します。各メンバを次の表に示します。

値	説明
AllAccess	レジストリ変数への作成、読み取り、書き込みアクセス。
Create	レジストリ変数への作成アクセス。
NoAccess	レジストリ変数へのアクセスはありません。
Read	レジストリ変数への読み込みアクセス。
Write	レジストリ変数への書き込みアクセス。

処理手順

My.Computer.Registry オブジェクトに関連する操作の例を次の表に示します。

操作	参照項目
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する
レジストリ キーに値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
レジストリ キーから値を読み込む	方法 : Visual Basic で、レジストリ キーから値を読み取る
レジストリ キーの値を設定する	方法 : Visual Basic でレジストリ キーの値を設定する

使用例

次の例では、サブキー `Software\MyCompany\Preferences\UserData` を削除します。

VB

```
My.Computer.Registry.CurrentUser.DeleteSubKey _
```

```
("Software\MyCompany\Preferences\UserData")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : RegistryProxy (Registry へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

アクセス許可は必要ありません。

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.CurrentUser

概念

一般的なレジストリ タスク

セキュリティとレジストリ

My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Computer.Registry.DynData プロパティ

HKEY_DYNDATA へのアクセスを可能にする RegistryKey 型を返します。

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.DynData
' Declaration
Public ReadOnly Property DynData As Microsoft.Win32.RegistryKey
```

戻り値

RegistryKey

解説

My.Computer.Registry オブジェクトには、レジストリ キーを操作するためのメソッドおよびプロパティが用意されています。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

HKEY_DYNDATA は、動的なレジストリ データを格納するために使用します。動的なレジストリ データには、仮想デバイス ドライバで使用され、Windows 95/98 システムでのみサポートされるものなどがあります。Windows 95/98 以外のシステムでは、**Registry.DynData** から返されるキーのどのメソッド (**CreateSubKey**、**DeleteSubKey**、**GetSubKeyCount**、**GetValueCount** など) を呼び出しても IOException がスローされます。

System.Security.Permissions 名前空間にある RegistryPermission クラスは、レジストリ変数へのアクセスを制御します。レジストリ変数は、**RegistryPermission** を持たないコードがアクセスできるメモリ位置に格納することはできません。同様に、アクセス許可を付与するときには、作業を行うために必要な最小限の権限を付与してください。

レジストリ アクセス許可のアクセス値は、RegistryPermissionAccess 列挙型で定義します。各メンバを次の表に示します。

値	説明
AllAccess	レジストリ変数への作成、読み取り、書き込みアクセス。
Create	レジストリ変数への作成アクセス。
NoAccess	レジストリ変数へのアクセスはありません。
Read	レジストリ変数への読み込みアクセス。
Write	レジストリ変数への書き込みアクセス。

処理手順

My.Computer.Registry オブジェクトに関連する操作の例を次の表に示します。

目的	参照項目
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する
レジストリ キーに値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
レジストリ キーから値を読み込む	方法 : Visual Basic で、レジストリ キーから値を読み取る
レジストリ キーの値を設定する	方法 : Visual Basic でレジストリ キーの値を設定する

使用例

次の例は、サブキー ツリー `Software\MyCompany\Preferences` を削除します。

VB

```
My.Computer.Registry.DynData.DeleteSubKeyTree("Software\MyCompany\Preferences")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : RegistryProxy (Registry へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	**あり**
Windows サービス	**あり**
Web サイト	**あり**

アクセス許可

アクセス許可は不要です。

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト
Microsoft.Win32.RegistryKey
Microsoft.Win32.Registry.DynData

概念

一般的なレジストリ タスク
セキュリティとレジストリ
My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Computer.Registry.GetValue メソッド

レジストリ キーから値を取得します。

```
' Usage
Dim value As Object = My.Computer.Registry.GetValue(keyName ,valueName ,defaultValue)
' Declaration
Public Function GetValue( _
   ByVal keyName As String, _
   ByVal valueName As String, _
   ByVal defaultValue As Object _
) As Object
```

パラメータ

keyName

String です。値を取得するキーです。必ず指定します。

valueName

String です。取得する値です。必ず指定します。

defaultValue

Object です。値が存在しない場合の既定値です。必ず指定します。

解説

valueName では、大文字と小文字は区別されません。

レジストリ キーには、名前に関連付けられていない値が 1 つ含まれることがあります。名前が付いていないこの値がレジストリ エディタに表示された場合、名前の代わりに文字列 "(Default)" が表示されます。この名前のない値を取得するには、null または空白の文字列 ("") を *valueName* に指定します。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- キーの名前が **Nothing** (ArgumentNullException) である場合。
- ユーザーにレジストリ キーを読み取る権限が与えられていない場合 (SecurityException)
- キー名が 255 文字の制限を超えている場合 (ArgumentException)

処理手順

GetValue メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
レジストリ キーから値を読み込む	方法 : Visual Basic で、レジストリ キーから値を読み取る

使用例

この例では、`HKEY_CURRENT_USER\Software\MyApp` から値 `Name` を読み取り、**MessageBox** にこの値を表示します。

VB

```
Dim readValue As Object
readValue = My.Computer.Registry.GetValue("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothi
ng)
MsgBox("The value is " & CStr(readValue))
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス RegistryProxy (Registry へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

アクセス許可は不要です。

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

GetValue

概念

一般的なレジストリ タスク

セキュリティとレジストリ

My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Computer.Registry.LocalMachine プロパティ

HKEY_LOCAL_MACHINE へのアクセスを可能にする RegistryKey 型を返します。

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.LocalMachine
' Declaration
Public ReadOnly Property LocalMachine As Microsoft.Win32.RegistryKey
```

戻り値

RegistryKey

解説

My.Computer.Registry オブジェクトには、レジストリ キーを操作するためのメソッドおよびプロパティが用意されています。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

HKEY_LOCAL_MACHINE は通常、Windows のユーザーが変わっても変わらない設定を格納するために使用します。このキーは、コンピュータのすべてのユーザーによって共有されます。5 つのサブキー (**Hardware**、**SAM**、**Security**、**Software**、および **System**) には、ローカル マシンの構成データが格納されます。

System.Security.Permissions 名前空間にある RegistryPermission クラスは、レジストリ変数へのアクセスを制御します。レジストリ変数は、**RegistryPermission** を持たないコードがアクセスできるメモリ位置に格納することはできません。同様に、アクセス許可を付与するときには、作業を行うために必要な最小限の権限を付与してください。

レジストリ アクセス許可のアクセス値は、RegistryPermissionAccess 列挙型で定義します。各メンバを次の表に示します。

値	説明
AllAccess	レジストリ変数への作成、読み取り、書き込みアクセス。
Create	レジストリ変数への作成アクセス。
NoAccess	レジストリ変数へのアクセスはありません。
Read	レジストリ変数への読み込みアクセス。
Write	レジストリ変数への書き込みアクセス。

処理手順

My.Computer.Registry オブジェクトに関連する操作の例を次の表に示します。

目的	参照項目
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する
レジストリ キーに値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
レジストリ キーから値を読み込む	方法 : Visual Basic で、レジストリ キーから値を読み取る
レジストリ キーの値を設定する	方法 : Visual Basic でレジストリ キーの値を設定する

使用例

次の例は、サブキー `Software\MyCompany\Preferences` を開き、`FontColor` の値に `"red"` を設定します。

VB

```
My.Computer.Registry.LocalMachine.OpenSubKey _
("Software\MyCompany\Preferences", True)
My.Computer.Registry.LocalMachine.SetValue("FontColor", "red")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : RegistryProxy (Registry へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

アクセス許可は不要です。

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.LocalMachine

概念

一般的なレジストリ タスク

セキュリティとレジストリ

My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Computer.Registry.PerformanceData プロパティ

HKEY_PERFORMANCE_DATA へのアクセスを提供する RegistryKey 型を返します。

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.PerformanceData
' Declaration
Public ReadOnly Property PerformanceData As Microsoft.Win32.RegistryKey
```

戻り値

RegistryKey

解説

HKEY_PERFORMANCE_DATA は、ソフトウェア コンポーネントのパフォーマンス情報へのアクセスに使用されます。各ソフトウェア コンポーネントは、インストール時にオブジェクトとカウンタにキーを作成し、実行中にカウンタ データを書き込みます。このデータにはレジストリを使ってアクセスしますが、レジストリ内には格納されません。その代わりに、**HKEY_PERFORMANCE_DATA** 経由でレジストリ関数を呼び出すと、システムが適切なシステム オブジェクト マネージャからデータを収集します。

このキーは Windows 98 システムには存在しません。**Name** メソッドを呼び出す場合以外でこれを使おうとすると、IOException がスローされます。

通常、このキーに書き込みを行うことはできません。このキーにサブキーを作成しようとすると、**IOException** がスローされます。

System.Security.Permissions 名前空間にある RegistryPermission クラスは、レジストリ変数へのアクセスを制御します。レジストリ変数は、**RegistryPermission** を持たないコードがアクセスできるメモリ位置に格納することはできません。同様に、アクセス許可が付与されるとき、作業を行うために必要な最小限の権限が付与されます。

レジストリ アクセス許可のアクセス値は、RegistryPermissionAccess 列挙型で定義します。各メンバを次の表に示します。

値	説明
AllAccess	レジストリ変数への作成、読み取り、書き込みアクセス。
Create	レジストリ変数への作成アクセス。
NoAccess	レジストリ変数へのアクセスはありません。
Read	レジストリ変数への読み込みアクセス。
Write	レジストリ変数への書き込みアクセス。

処理手順

My.Computer.Registry オブジェクトに関連するタスクの例を次の表に示します。

タスク	参照項目
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する
特定のレジストリ キーに値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
レジストリ キーから値を読み込む	方法 : Visual Basic で、レジストリ キーから値を読み取る
レジストリ キーの値を設定する	方法 : Visual Basic でレジストリ キーの値を設定する

使用例

この例では、`MyCompany` サブキーから `ThisSoftware` を取得します。

VB

```
My.Computer.Registry.PerformanceData.GetValue("MyCompany\ThisSoftware")
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス RegistryProxy (Registry へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**あり**
クラス ライブラリ	**あり**
コンソール アプリケーション	**あり**
Windows コントロール ライブラリ	**あり**
Web コントロール ライブラリ	**あり**
Windows サービス	**あり**
Web サイト	**あり**

アクセス許可

アクセス許可は不要です。

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.PerformanceData

概念

一般的なレジストリ タスク

セキュリティとレジストリ

My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Computer.Registry.SetValue メソッド

レジストリ キーに値を書き込みます。

```
' Usage
My.Computer.Registry.SetValue(keyName ,valueName ,value)
My.Computer.Registry.SetValue(keyName ,valueName ,value ,valueKind)
' Declaration
Public Sub SetValue( _
   ByVal keyName As String, _
   ByVal valueName As String, _
   ByVal value As Object _
)
' -or-
Public Sub SetValue( _
   ByVal keyName As String, _
   ByVal valueName As String, _
   ByVal value As Object, _
   ByVal valueKind As Microsoft.Win32.RegistryValueKind _
)
```

パラメータ

keyName

String です。書き込むキーの名前です。必ず指定します。

valueName

String です。書き込む値の名前です。必ず指定します。

value

Object です。書き込む値です。必ず指定します。

valueKind

RegistryValueKind です。必ず指定します。

解説

指定されたキーまたは値が存在しないときは、新たに作成されます。

例外

次の条件を満たす場合は、例外が発生する可能性があります。

- キーの名前が **Nothing** (ArgumentNullException) である場合。
- キー名が 255 文字の制限を超えている場合 (ArgumentException)
- このハイブは無効 (**ArgumentException**) です。
- キーが閉じている場合 (IOException)
- パスが無効 (**IOException**) です。
- レジストリ キーが読み取り専用の場合 (UnauthorizedAccessException)

処理手順

My.Computer.Registry.SetValue メソッドを使用するタスクの例を次に示します。

タスク	参照項目
レジストリ キーの値を設定する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する

使用例

この例では、キー **HKEY_CURRENT_USER\Software\MyApp** の "Author's Name" に、値 `Name` を設定します。

VB

```
Dim readValue As Object
readValue = My.Computer.Registry.GetValue _
("HKEY_CURRENT_USER\Software\MyApp", "Name", Nothing)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス RegistryProxy (Registry へのアクセスを提供します)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

アクセス許可は不要です。

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト
Microsoft.Win32.RegistryValueKind
Microsoft.Win32.Registry.SetValue

概念

一般的なレジストリ タスク
セキュリティとレジストリ
My を使用したレジストリからの読み取りとレジストリへの書き込み
レジストリ アクセス (Visual Basic 6.0 ユーザー向け)

My.Computer.Registry.Users プロパティ

HKEY_USERS へのアクセスを可能にする RegistryKey 型を返します。

```
' Usage
Dim value As Microsoft.Win32.RegistryKey = My.Computer.Registry.Users
' Declaration
Public ReadOnly Property Users As Microsoft.Win32.RegistryKey
```

戻り値

RegistryKey

解説

My.Computer.Registry オブジェクトには、レジストリ キーを操作するためのメソッドおよびプロパティが用意されています。詳細については、「My.Computer.Registry オブジェクト」を参照してください。

HKEY_USERS は主に、初めて使用するユーザーのための、既定の設定を格納するために使用します。

System.Security.Permissions 名前空間にある RegistryPermission クラスは、レジストリ変数へのアクセスを制御します。レジストリ変数は、**RegistryPermission** を持たないコードがアクセスできるメモリ位置に格納することはできません。同様に、アクセス許可を付与するときには、作業を行うために必要な最小限の権限を付与してください。

レジストリ アクセス許可のアクセス値は、RegistryPermissionAccess 列挙型で定義します。各メンバを次の表に示します。

値	レジストリ変数へのアクセス
AllAccess	作成、読み取り、書き込み
Create	作成
NoAccess	アクセスなし
Read	読み取り
Write	書き込み

処理手順

My.Computer.Registry オブジェクトに関連する操作の例を次の表に示します。

目的	参照項目
レジストリ キーを作成する	方法 : Visual Basic でレジストリ キーを作成し、値を設定する
レジストリ キーを削除する	方法 : Visual Basic で、レジストリ キーを削除する
レジストリ キーに値が存在するかどうかを判別する	方法 : Visual Basic で、レジストリ キーに値が存在するかどうかを確認する
レジストリ キーから値を読み込む	方法 : Visual Basic で、レジストリ キーから値を読み取る
レジストリ キーの値を設定する	方法 : Visual Basic でレジストリ キーの値を設定する

使用例

次の例は、レジストリ キーに格納された値の数をカウントして表示します。

VB

```
Dim keyCount as Integer
keyCount = My.Computer.Registry.Users.ValueCount
MsgBox(keyCount)
```

必要条件

名前空間 : Microsoft.VisualBasic.MyServices

クラス : RegistryProxy (Registry へのアクセスを可能にします)

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

アクセス許可は不要です。

参照

処理手順

トラブルシューティング : レジストリの操作

関連項目

My.Computer.Registry オブジェクト

Microsoft.Win32.RegistryKey

Microsoft.Win32.Registry.Users

概念

一般的なレジストリ タスク

セキュリティとレジストリ

My を使用したレジストリからの読み取りとレジストリへの書き込み

My.Forms オブジェクト

現在のプロジェクト内で宣言されている各 Windows フォームのインスタンスにアクセスするためのプロパティを提供します。

解説

My.Forms オブジェクトは、現在のプロジェクト内の各フォームのインスタンスを提供します。プロパティの名前は、そのプロパティのアクセス先のフォームと同じ名前になります。フォームをプロジェクトに追加する方法については、「方法 : プロジェクトに Windows フォームを追加する」を参照してください。

My.Forms オブジェクトによって提供されるフォームにアクセスするには、そのフォームの名前を修飾子なしで使用します。このプロパティ名はフォームの型名と同じになるので、まるで既定のインスタンスがあるかのようにしてフォームにアクセスできます。たとえば、`My.Forms.Form1.Show` は `Form1.Show` と同じ意味です。

My.Forms オブジェクトは、現在のプロジェクトに関連付けられているフォームのみを公開します。参照された DLL 内で宣言されているフォームにはアクセスできません。DLL 内で参照されているフォームにアクセスするには、*DllName.FormName* という形式の修飾名を使用する必要があります。詳細については、「方法 : フォームにアクセスする」を参照してください。

My.Application.OpenForms プロパティを使用すると、アプリケーションの現在開かれているすべてのフォームのコレクションを取得できます。

このオブジェクトとプロパティは、Windows アプリケーションでのみ使用できます。

プロパティ

My.Forms オブジェクトのプロパティを使用すると、現在のプロジェク内のフォームのインスタンスにアクセスできます。プロパティの名前はそのプロパティがアクセスするフォームの名前と同じになり、プロパティの型はフォームの型と同じになります。

メモ :
名前の衝突が存在する場合、フォームにアクセスするためのプロパティ名は *RootNamespace_Namespace_FormName* になります。たとえば、`Form1.` という名前のフォームが 2 つあるとします。一方のフォームがルート名前空間 `WindowsApplication1` 内の名前空間 `Namespace1` にある場合、このフォームにアクセスするには `My.Forms.WindowsApplication1_Namespace1_Form1` を使用します。

My.Forms オブジェクトは、アプリケーションの起動時に作成されたメイン フォームのインスタンスへのアクセスを提供します。他のフォームについては、フォームへのアクセスが発生したときに **My.Forms** オブジェクトがそのフォームの新しいインスタンスを作成して格納します。以降は、そのプロパティにアクセスするとフォームのインスタンスが返されます。

フォームを破棄するには、そのフォームを表すプロパティに **Nothing** を代入します。このプロパティ Set アクセス操作子が対応するフォームの Close メソッドを呼び出し、格納されている値に **Nothing** を代入します。このプロパティに **Nothing** 以外の値を代入した場合は、setter が ArgumentException 例外をスローします。

My.Forms オブジェクトのプロパティにフォームのインスタンスが格納されているかどうかを調べるには、**Is** または **IsNot** 演算子を使用します。これらの演算子を使用して、プロパティの値が **Nothing** かどうかを確認します。

メモ :
通常は、**Is** または **IsNot** 演算子が比較を実行するためには、プロパティの値を読み取る必要があります。しかし、プロパティの現在の値が **Nothing** である場合は、値を読み取ろうとするとフォームの新しいインスタンスが作成され、そのインスタンスが返されてしまいます。そこで、Visual Basic コンパイラは **My.Forms** オブジェクトのプロパティを特別扱いとし、**Is** または **IsNot** 演算子がプロパティの値を変更せずにプロパティのステータスを確認できるようにしています。

処理手順

My.Forms オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
あるフォームから別のフォームにアクセスする	方法 : フォームにアクセスする
あるフォームから別のフォームを制御する	方法 : アプリケーション内のフォーム間でやり取りする

使用例

この例では、既定の `SidebarMenu` フォームのタイトルを変更します。

VB

```
Sub ShowSidebarMenu(ByVal newTitle As String)
    If My.Forms.SidebarMenu IsNot Nothing Then
        My.Forms.SidebarMenu.Text = newTitle
    End If
End Sub
```

この例を実行するには、SidebarMenu という名前のフォームがプロジェクト内に含まれている必要があります。詳細については、「方法 : プロジェクトに Windows フォームを追加する」を参照してください。

このコードは Windows アプリケーション プロジェクトでのみ有効です。

必要条件

プロジェクトの種類別の使用可/不可

プロジェクトの種類	使用可能
Windows アプリケーション	○
クラス ライブラリ	X
コンソール アプリケーション	X
Windows コントロール ライブラリ	X
Web コントロール ライブラリ	X
Windows サービス	X
Web サイト	X

参照

処理手順

方法 : プロジェクトに Windows フォームを追加する
方法 : フォームにアクセスする

関連項目

My.Application.OpenForms プロパティ
My.Application.OpenForms プロパティ
Is 演算子 (Visual Basic)
IsNot 演算子
Form
Close

概念

アプリケーション フォームへのアクセス

My.Log オブジェクト

イベントと例外の情報を作成するためのプロパティとメソッドを、アプリケーション ログのリスナに提供します。

解説

My.Log オブジェクトには、.NET Framework のログ サービスにアクセスするための、わかりやすいエントリ ポイントが用意されています。**WriteEntry** メソッドと **WriteException** メソッドは、アプリケーション ログのリスナにメッセージを書き込みます。リスナはアプリケーションの構成ファイルで構成できます。詳細については、「チュートリアル : My.Application.Log による情報の書き込み先の変更」および「Visual Basic でのアプリケーション ログの使用」を参照してください。

My.Log オブジェクトは、ASP.NET アプリケーションにのみ使用できます。クライアント アプリケーションの場合は、**My.Application.Log** を使用します。詳細については、「My.Application.Log オブジェクト」を参照してください。

必要条件

名前空間 : Microsoft.VisualBasic.Logging

クラス : Log

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

方法 : ログ メッセージを書き込む
方法 : Visual Basic で例外をログに記録する
チュートリアル : My.Application.Log による情報の書き込み先の確認

関連項目

My.Log オブジェクトのメンバ
My.Application オブジェクト
My.Application.Log オブジェクト
Microsoft.VisualBasic.Logging.Log

My.Log オブジェクトのメンバ

My.Log オブジェクトは、イベントと例外の情報を作成するためのプロパティとメソッドを、アプリケーション ログのリスナに提供します。

プロパティ

プロパティ	説明
TraceSource	**My.Log** オブジェクトの基礎にある TraceSource オブジェクトを取得します。 これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

メソッド

メソッド	説明
WriteEntry	アプリケーション ログのリスナにメッセージを書き込みます。
WriteException	アプリケーション ログのリスナに例外の情報を書き込みます。

参照

関連項目

My.Log オブジェクト

My.Request オブジェクト

要求されたページの HttpRequest オブジェクトを取得します。

解説

My.Request オブジェクトには現在の HTTP 要求に関する情報が含まれます。

My.Request オブジェクトは、ASP.NET アプリケーションだけで使用できます。

使用例

次のコード例は、**My.Request** オブジェクトからヘッダー コレクションを取得し、**My.Response** オブジェクトを使ってそれを ASP.NET ページに書き込みます。

VB

```
<script runat="server">
    Public Sub ShowHeaders()
        ' Load the header collection from the Request object.
        Dim coll As System.Collections.Specialized.NameValueCollection
        coll = My.Request.Headers

        ' Put the names of all keys into a string array.
        For Each key As String In coll.AllKeys
            My.Response.Write("Key: " & key & "<br>")

            ' Get all values under this key.
            For Each value As String In coll.GetValues(key)
                My.Response.Write("Value: " & _
                    Server.HtmlEncode(value) & "<br>")
            Next
        Next
    End Sub
</script>
```

参照

関連項目

My.Response オブジェクト

HttpRequest

My.Response オブジェクト

Page に関連付けられている HttpResponse オブジェクトを取得します。このオブジェクトを使用すると、HTTP 応答データをクライアントに送信できます。また、このオブジェクトにはその応答に関する情報が格納されます。

解説

My.Response オブジェクトには、ページに関連する現在の **HttpResponse** オブジェクトが含まれます。

My.Response オブジェクトは、ASP.NET アプリケーションにのみ使用できます。

使用例

次の例は、**My.Request** オブジェクトからヘッダーのコレクションを取得し、**My.Response** オブジェクトを使用して ASP.NET ページにコレクションを書き込みます。

VB

```
<script runat="server">
    Public Sub ShowHeaders()
        ' Load the header collection from the Request object.
        Dim coll As System.Collections.Specialized.NameValueCollection
        coll = My.Request.Headers

        ' Put the names of all keys into a string array.
        For Each key As String In coll.AllKeys
            My.Response.Write("Key: " & key & "<br>")

            ' Get all values under this key.
            For Each value As String In coll.GetValues(key)
                My.Response.Write("Value: " & _
                    Server.HtmlEncode(value) & "<br>")
            Next
        Next
    End Sub
</script>
```

参照

関連項目

My.Request オブジェクト

HttpResponse

My.Resources オブジェクト

アプリケーションのリソースにアクセスするための、プロパティとクラスを提供します。

解説

My.Resources オブジェクトを使用すると、アプリケーションのリソースへのアクセスが可能になり、アプリケーションに必要なリソースを動的に取得できるようになります。詳細については、「アプリケーション リソースの管理」を参照してください。

My.Resources オブジェクトはグローバルなリソースだけを公開します。フォームに関連付けられたリソース ファイルへのアクセスはできません。フォームのリソースには、フォームからアクセスする必要があります。詳細については、「チュートリアル : Windows フォームのローカリゼーション」を参照してください。

アプリケーションのカルチャ固有のリソース ファイルに、**My.Resources** オブジェクトからアクセスできます。既定では、**My.Resources** オブジェクトは、My.Application.UICulture プロパティ のカルチャに一致するリソース ファイルからリソースを検索します。ただし、この動作をオーバーライドして、リソースに使用するカルチャを指定することもできます。詳細については、「アプリケーションのリソース」を参照してください。

プロパティ

My.Resources オブジェクトのプロパティでは、アプリケーションのリソースに読み取り専用でしかアクセスできません。リソースを追加または削除するには、プロジェクト デザイナ を使用します。詳細については、「方法 : リソースを追加または削除する」を参照してください。プロジェクト デザイナで追加されたリソースには、**My.Resources.***resourceName* を使用してアクセスします。

また、ソリューション エクスプローラ からプロジェクトを選択し、[プロジェクト] メニューの [新しい項目の追加] または [既存項目の追加] を選択することによっても、リソース ファイルを追加または削除できます。この方法で追加したリソースへは、**My.Resources.***resourceFileName.resourceName* を使用してアクセスできます。

各リソースには名前、カテゴリ、値が定義され、これらのリソース設定によってプロパティが **My.Resources** オブジェクトに含まれるリソースにアクセスする方法が決まります。プロジェクト デザイナ で追加されたリソースの場合は、次のようになります。

- 名前はプロパティの名前
- リソース データはプロパティの値
- カテゴリはプロパティの種類

カテゴリ	プロパティのデータ型
文字列	文字列
イメージ	Bitmap
アイコン	Icon
Audio	UnmanagedMemoryStream **UnmanagedMemoryStream** クラスは Stream から派生しているため、ストリームを受け取るメソッド (My.Computer.Audio.Play メソッド など) で使用できます。
ファイル	• テキスト ファイルの場合は文字列。 • イメージ ファイルの場合は **Bitmap**。 • アイコン ファイルの場合は **Icon**。 • サウンド ファイルの場合は **UnmanagedMemoryStream**。
その他	デザイナの [型] 列の情報によって決まります。

クラス

My.Resources オブジェクトは各リソース ファイルを、共有プロパティを持つクラスとして公開します。クラス名はリソース ファイルの名前と同じです。先のセクションにあるとおり、リソース ファイルのリソースはクラスのプロパティとして公開されます。

処理手順

My.Resources オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
文字列リソースを取得する	方法 : Visual Basic で文字列リソースを取得する
イメージ リソースを取得する	方法 : Visual Basic でイメージ リソースを取得する
アイコン リソースを取得する	方法 : Visual Basic でアイコン リソースを取得する
オーディオ リソースを取得する	方法 : Visual Basic でオーディオ リソースを取得する
ローカライズされたリソースを取得する	方法 : Visual Basic で、ローカライズされたリソースを取得する

使用例

次の例は、アプリケーションのリソース ファイルに格納された `Form1Icon` という名前のアイコンを、フォームのアイコンに設定します。

VB

```
Sub SetFormIcon()
    Me.Icon = My.Resources.Form1Icon
End Sub
```

この例を実行するには、アプリケーションのリソース ファイルに `Form1Icon` という名前のアイコンが格納されている必要があります。詳細については、「方法 : リソースを追加または削除する」を参照してください。

参照

処理手順

方法 : Visual Basic で文字列リソースを取得する
方法 : Visual Basic でイメージ リソースを取得する
方法 : Visual Basic でアイコン リソースを取得する
方法 : Visual Basic でオーディオ リソースを取得する
方法 : Visual Basic で、ローカライズされたリソースを取得する
方法 : リソースを追加または削除する
チュートリアル : Windows フォームのローカリゼーション

関連項目

My.Application.UICulture プロパティ

概念

アプリケーションのリソース

その他の技術情報

アプリケーション リソースの管理

My.Settings オブジェクト

アプリケーションの設定値にアクセスするためのプロパティとメソッドを提供します。

解説

My.Settings はアプリケーションの設定値へのアクセスを提供し、これによって、プロパティの設定値およびアプリケーションに関するその他の情報を動的に格納および取得できます。詳細については、「アプリケーションの設定の管理」を参照してください。

プロパティ

My.Settings オブジェクトのプロパティを使用すると、アプリケーションの設定値にアクセスできます。設定を追加または削除するには、設定デザイナ を使います。詳細については、「方法 : アプリケーション設定を追加または削除する」を参照してください。

各設定には [名前]、[型]、[スコープ]、および [値] があり、これらの設定値によって、プロパティが **My.Settings** オブジェクトに含まれる各設定に、どのようにアクセスするかが決まります。

- [名前] はプロパティの名前です。
- [型] はプロパティの種類です。
- [スコープ] はプロパティが読み取り専用かどうかを示します。プロパティが読み取り専用なら、この値は [アプリケーション] で、読み書き可能なら [ユーザー] になります。
- このプロパティの既定値は [値] です。

メソッド

メソッド	説明
Reload	ユーザー設定を、前回保存した値から再読み込みします。
Save	現在のユーザー設定を保存します。

My.Settings オブジェクトには、ApplicationSettingsBase クラスから継承した高度なプロパティとメソッドも用意されています。

処理手順

My.Settings オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
アプリケーションの設定値の読み込み	方法 : Visual Basic でアプリケーション設定を読み取る
ユーザー設定の変更	方法 : Visual Basic でユーザー設定を変更する
ユーザー設定の永続化	方法 : Visual Basic でユーザー設定を永続化する
ユーザー設定のプロパティ グリッドの作成	方法 : Visual Basic でユーザー設定のためのプロパティ グリッドを作成する

使用例

次の例は、`Nickname` の設定値を表示します。

VB

```
Sub ShowNickname()
    MsgBox("Nickname is " & My.Settings.Nickname)
End Sub
```

この例を実行するには、アプリケーションに **String** 型の `Nickname` が設定されていることが必要です。詳細については、「方法 : アプリケーション設定を追加または削除する」を参照してください。

参照

処理手順

方法 : Visual Basic でアプリケーション設定を読み取る

方法 : Visual Basic でユーザー設定を変更する
方法 : Visual Basic でユーザー設定を永続化する
方法 : Visual Basic でユーザー設定のためのプロパティ グリッドを作成する
方法 : アプリケーション設定を追加または削除する

関連項目

ApplicationSettingsBase

その他の技術情報

アプリケーションの設定の管理

My.User オブジェクト

現在のユーザーに関する情報へのアクセスを提供します。

解説

My.User オブジェクトのプロパティとメソッドを使用すると、現在のユーザーに関する情報にアクセスできます。"現在のユーザー" という言葉の意味は、Windows アプリケーションと Web アプリケーションでは微妙に異なります。Windows アプリケーションでは、現在のユーザーとはそのアプリケーションを実行しているユーザーのことです。Web アプリケーションでは、現在のユーザーとはそのアプリケーションにアクセスしているユーザーのことです。

My.User プロパティでは、現在のユーザーに関する IPrincipal にもアクセスできます。プリンシパル オブジェクトはユーザーのセキュリティ コンテキストを表し、ユーザーの識別情報と、ユーザーが所属しているロールを含んでいます。

Windows アプリケーションでは、このプロパティは CurrentPrincipal プロパティと同じ働きをします。Web アプリケーションでは、このプロパティは、Current プロパティから返されるオブジェクトの User プロパティと同じ働きをします。

メモ：

Windows アプリケーションでは、[Windows アプリケーション] テンプレートから作成したプロジェクトに限り、**My.User** オブジェクトが既定で初期化されます。その他の種類の Windows プロジェクトの場合は、My.User.InitializeWithWindowsUser メソッド を明示的に呼び出すか、**CurrentPrincipal** に値を割り当てて、**My.User** オブジェクトを初期化する必要があります。

メモ：

Windows 95 および Windows 98 はログオン中のユーザーという概念をサポートしていないので、これらのオペレーティング システムで実行している場合には、**My.User** オブジェクトは現在の Windows ユーザーに関する情報を報告できません。これらのオペレーティング システム上で **My.User** オブジェクトを使用するためには、カスタム認証を実装する必要があります。詳細については、「チュートリアル : カスタムの認証および承認の実装」を参照してください。

処理手順

My.User オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
ユーザーのログイン名を取得する	方法 : ユーザーのログイン名を確認する
ユーザーのドメイン名を取得する (アプリケーションが Windows 認証を使用している場合)	方法 : ユーザーのドメインを確認する
ユーザーのロールを確認する	方法 : ユーザーがグループに属しているかどうかを確認する
カスタム認証を実装する	チュートリアル : カスタムの認証および承認の実装

使用例

この例では、アプリケーションが Windows 認証とカスタム認証のどちらを使用しているかを調べ、その情報に基づいて **My.User.Name** プロパティを解析します。

VB

```
Function GetUserName() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' The application is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
        Dim parts() As String = Split(My.User.Name, "\")
        Dim username As String = parts(1)
        Return username
    Else
        ' The application is using custom authentication.
        Return My.User.Name
    End If
End Function
```

必要条件

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照

関連項目

My.User オブジェクトのメンバ
IPrincipal
CurrentPrincipal
User
Current

My.User オブジェクトのメンバ

My.User オブジェクトを使用すると、現在のユーザーに関する情報にアクセスできます。

プロパティ

プロパティ	説明
CurrentPrincipal	現在のプリンシパルを取得または設定します (ロール ベース セキュリティ の場合)。 これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。
IsAuthenticated	ユーザーが認証されているかどうかを示す値を取得します。
Name	現在のユーザーの名前を取得します。

メソッド

メソッド	説明
InitializeWithWindowsUser	スレッドの現在のプリンシパルを、アプリケーションを起動した Windows ユーザーに設定します。 これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。
IsInRole	指定されたロールに現在のユーザーが属するかどうかを判断します。

参照

関連項目

My.User オブジェクト

My.User.CurrentPrincipal プロパティ

ロール ベースのセキュリティのために現在のプリンシパルを取得または設定します。

```
' Usage
Dim value As System.Security.Principal.IPrincipal = My.User.CurrentPrincipal
' Declaration
Public Property CurrentPrincipal As System.Security.Principal.IPrincipal
```

戻り値

セキュリティ コンテキストを表す IPrincipal 値

例外

例外を引き起こす可能性のある状態を次に示します。

- 呼び出し元がこのプリンシパルを設定するのに必要なアクセス許可を持っていない (SecurityException)。

解説

My.User.CurrentPrincipal プロパティに **IPrincipal** インターフェイスを実装するオブジェクトを設定すると、カスタム認証を有効にできます。

ほとんどの種類のプロジェクトでは、このプロパティはスレッドの現在のプリンシパルを取得および設定します。ASP.NET アプリケーションでは、このプロパティは現在の HTTP 要求のユーザー識別情報に関するセキュリティ情報を取得および設定します。

これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

処理手順

目的	参照項目
ユーザーのログイン名を取得する	方法 : ユーザーのログイン名を確認する
ユーザーのドメイン名を取得する (アプリケーションが Windows 認証を使用している場合)	方法 : ユーザーのドメインを確認する
カスタム認証を実装する	チュートリアル : カスタムの認証および承認の実装

使用例

この例では、アプリケーションが Windows 認証とカスタム認証のどちらを使用しているかを調べ、その情報に基づいて **My.User.Name** プロパティを解析します。

VB

```
Function GetUserName() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' The application is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
        Dim parts() As String = Split(My.User.Name, "\")
        Dim username As String = parts(1)
        Return username
    Else
        ' The application is using custom authentication.
        Return My.User.Name
    End If
End Function
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : User、WebUser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
SecurityPermission	コードに適用される一連のセキュリティ アクセス許可を表します。関連する列挙値 : ControlPrincipal。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : ユーザーのログイン名を確認する
方法 : ユーザーのドメインを確認する
チュートリアル : カスタムの認証および承認の実装

関連項目

My.User オブジェクト
System.Security.Principal.IPrincipal
Microsoft.VisualBasic.ApplicationServices.User.CurrentPrincipal

My.User.InitializeWithWindowsUser メソッド

スレッドの現在のプリンシパルを、このアプリケーションを開始した Windows ユーザーに設定します。

```
' Usage
My.User.InitializeWithWindowsUser()
' Declaration
Public Sub InitializeWithWindowsUser()
```

解説

My.User.InitializeWithWindowsUser メソッドを使用すると、スレッドの現在のプリンシパルを、このアプリケーションを開始した Windows ユーザーに設定できます。Windows アプリケーションの Visual Basic アプリケーション モデルは、このメソッドを起動時に既定で呼び出します。他の種類のプロジェクトでは、このメソッドを明示的に呼び出すか、System.Threading.Thread.CurrentPrincipal に値を割り当てて、スレッドの現在のプリンシパルを設定する必要があります。

Windows プロジェクトでは、**My.User** オブジェクトはスレッドの現在のプリンシパルに基づきます。したがって、このメソッドにより、**My.User** から返される情報が変化することがあります。ASP.NET アプリケーションでは、**My.User** オブジェクトは現在の HTTP 要求のユーザー識別情報に基づくので、このメソッドの影響を受けません。

メモ :
My.User オブジェクトの厳密な動作は、アプリケーションの種類と、そのアプリケーションを実行するオペレーティング システムによって左右されます。詳細については、「My.User オブジェクト」を参照してください。

これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : User、WebUser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
SecurityPermission	コードに適用される一連のセキュリティ アクセス許可を表します。関連する列挙値 : ControlPrincipal。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

チュートリアル : カスタムの認証および承認の実装

関連項目

My.User オブジェクト

Microsoft.VisualBasic.ApplicationServices.User.InitializeWithWindowsUser

My.User.IsAuthenticated プロパティ

ユーザーが認証されているかどうかを示す値を取得します。

```
' Usage
Dim value As Boolean = My.User.IsAuthenticated
' Declaration
Public ReadOnly Property IsAuthenticated As Boolean
```

戻り値

ユーザーが認証されている場合は **True**、それ以外の場合は **False** です。

解説

My.User.IsAuthenticated を使用すると、現在のユーザーが認証されているかどうかをコードから判断できます。

メモ :
My.User の厳密な動作は、アプリケーションの種類およびアプリケーションを実行しているオペレーティング システムによって決まります。詳細については、「My.User オブジェクト」を参照してください。

処理手順

目的	参照項目
カスタムな認証の実装	チュートリアル : カスタムの認証および承認の実装

使用例

次の例は、ユーザーが認証されているかどうかを、リソースにアクセスする前に確認します。

VB

```
If My.User.IsAuthenticated Then
    ' Insert code to access a resource here.
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : User、WebUser

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

使用可能なプロジェクトの種類

プロジェクトの種類	使用可/不可
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**

Web サイト	**可**

アクセス許可

次のアクセス許可が必要になる可能性があります。

アクセス許可	説明
SecurityPermission	コードに適用されたセキュリティ アクセス許可のセットを記述します。関連する列挙値 : ControlPrincipal。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

チュートリアル : カスタムの認証および承認の実装

関連項目

My.User オブジェクト

Microsoft.VisualBasic.ApplicationServices.User.IsAuthenticated

My.User.IsInRole メソッド

現在のユーザーが指定のロールに所属しているかどうかを調べます。

```
' Usage
Dim value As Boolean = My.User.IsInRole(role)
Dim value As Boolean = My.User.IsInRole(role)
' Declaration
Public Function IsInRole( _
   ByVal role As String _
) As Boolean
' -or-
Public Function IsInRole( _
   ByVal role As BuiltInRole _
) As Boolean
```

パラメータ

role

String または BuiltInRole 列挙型。メンバシップを調べるロールを指定します。

戻り値

現在のユーザーが指定のロールのメンバである場合は **True**、それ以外の場合は **False** を返します。

解説

My.User.IsInRole メソッドを使用すると、現在のユーザーが指定のロールのメンバであるかどうかを確認できます。

文字列を受け取る **My.User.IsInRole** メソッドのオーバーロードを使用すると、現在のプリンシパルの IsInRole メソッドに簡単にアクセスできます。

BuiltInRole 列挙体を受け取る **My.User.IsInRole** メソッドのオーバーロードは、現在のプリンシパルに応じて異なる動作をします。現在のプリンシパルが Windows ユーザー プリンシパル (WindowsPrincipal) の場合は、この関数は *role* を等価の WindowsBuiltInRole 列挙体に変換し、IsInRole を呼び出した結果を返します。現在のプリンシパルがそれ以外のプリンシパルである場合は、この関数は、*role* 内の列挙値の名前をそのプリンシパルの **IsInRole** メソッドに渡します。

メモ：

My.User オブジェクトの厳密な動作は、アプリケーションの種類と、そのアプリケーションを実行するオペレーティング システムによって左右されます。詳細については、「My.User オブジェクト」を参照してください。

処理手順

目的	参照項目
ユーザーのロールを確認する	方法 : ユーザーがグループに属しているかどうかを確認する

使用例

この例では、リソースにアクセスする前に、ユーザーが管理者かどうかを調べます。

VB

```
If My.User.IsInRole( _
        ApplicationServices.BuiltInRole.Administrator) Then
    ' Insert code to access a resource here.
End If
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : User、WebUser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

プロジェクトの種類ごとの可用性

プロジェクトの種類	可用性
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

次のアクセス許可が必要です。

アクセス許可	説明
SecurityPermission	コードに適用される一連のセキュリティ アクセス許可を表します。関連する列挙値 : ControlPrincipal。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : ユーザーがグループに属しているかどうかを確認する

チュートリアル : カスタムの認証および承認の実装

関連項目

My.User オブジェクト

BuiltInRole 列挙型

Microsoft.VisualBasic.ApplicationServices.BuiltInRole

Microsoft.VisualBasic.ApplicationServices.User.IsInRole(System.String)

My.User.Name プロパティ

現在のユーザーのユーザー名を取得します。

```
' Usage
Dim value As String = My.User.Name
' Declaration
Public ReadOnly Property Name As String
```

戻り値

String。現在のユーザーのユーザー名です。

解説

My.User オブジェクトを使用して、現在のユーザーに関する情報を取得できます。

ユーザーを認証するプリンシパルも、ユーザー名の形式に適用されます。既定で、アプリケーションでは Windows 認証が使用され、ユーザー名の形式は DOMAIN\USERNAME となります。独自に実装されたプリンシパルでは、これと同じ形式は必ずしも使用されません。

メモ：

My.User.Name プロパティは、ユーザー名を認識しないオペレーティング システムである Windows 95 または Windows 98 で実行された場合に空の文字列を返します。

メモ：

My.User オブジェクトの動作は、アプリケーションの種類やアプリケーションを実行するオペレーティング システムによって多少異なります。詳細については、「My.User オブジェクト」を参照してください。

処理手順

My.User.Name プロパティに関連するタスクの例を次の表に示します。

タスク	参照項目
ユーザーのログイン名を取得します。	方法 : ユーザーのログイン名を確認する
ユーザーのドメイン名を取得します (アプリケーションで Windows 認証を使用する場合)。	方法 : ユーザーのドメインを確認する
カスタム認証を実装します。	チュートリアル : カスタムの認証および承認の実装

使用例

このコード例は、アプリケーションで Windows 認証とカスタム認証のどちらが使用されるかを調べ、その情報を使って **My.User.Name** プロパティを解析します。

VB

```
Function GetUserName() As String
    If TypeOf My.User.CurrentPrincipal Is _
    Security.Principal.WindowsPrincipal Then
        ' The application is using Windows authentication.
        ' The name format is DOMAIN\USERNAME.
        Dim parts() As String = Split(My.User.Name, "\")
        Dim username As String = parts(1)
        Return username
    Else
        ' The application is using custom authentication.
        Return My.User.Name
    End If
End Function
```

必要条件

名前空間 : Microsoft.VisualBasic.ApplicationServices

クラス : User、WebUser

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

プロジェクトの種類別の可用性

プロジェクトの種類	使用
Windows アプリケーション	**可**
クラス ライブラリ	**可**
コンソール アプリケーション	**可**
Windows コントロール ライブラリ	**可**
Web コントロール ライブラリ	**可**
Windows サービス	**可**
Web サイト	**可**

アクセス許可

以下のアクセス許可が必要な場合があります。

アクセス許可	説明
SecurityPermission	コードに適用される一連のセキュリティ アクセス許可を記述します。関連する列挙値 : ControlPrincipal。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : ユーザーのログイン名を確認する
方法 : ユーザーのドメインを確認する
チュートリアル : カスタムの認証および承認の実装

関連項目

My.User オブジェクト
Microsoft.VisualBasic.ApplicationServices.User.Name

My.WebServices オブジェクト

現在のプロジェクトから参照する、各 XML Web サービスのインスタンスを作成したり、使用したりするためのプロパティを提供します。

解説

My.WebServices オブジェクトは、現在のプロジェクトから参照されている各 Web サービスのインスタンスを提供します。それぞれのインスタンスは必要に応じてインスタンス化されます。これらの Web サービスは **My.WebServices** オブジェクトのプロパティを通じてアクセスできます。このときに使用するプロパティの名前は、そのプロパティがアクセスする Web サービスの名前と同じです。SoapHttpClientProtocol を継承するクラスはすべて Web サービスです。Web サービスをプロジェクトに追加する方法については、
「方法 : マネージ コードを使用して XML Web サービスにアクセスする」を参照してください。

My.WebServices オブジェクトは、現在のプロジェクトに関連付けられている Web サービスのみを公開します。参照された DLL 内で宣言されている Web サービスにはアクセスできません。DLL 内で宣言された Web サービスにアクセスするには、Web サービスの修飾名を使用する必要があります。Web サービスの修飾名は *DllName.WebServiceName* という形式になります。詳細については、
「マネージ コードを使用した XML Web サービスへのアクセス」を参照してください。

このオブジェクトとプロパティは、Web アプリケーションでは使用できません。

プロパティ

My.WebServices オブジェクトのプロパティを使用すると、現在のプロジェクトから参照されている Web サービスのインスタンスにアクセスできます。プロパティの名前はそのプロパティがアクセスする Web サービスの名前と同じになり、プロパティの型は Web サービスの型と同じになります。

メモ :

名前の衝突が存在する場合、Web サービスにアクセスするためのプロパティ名は *RootNamespace_Namespace_ServiceName* になります。たとえば、`Service1` という名前の Web サービスが 2 つあるとします。一方の Web サービスがルート名前空間 `WindowsApplication1` 内の名前空間 `Namespace1` にある場合、このサービスにアクセスするには `My.WebServices.WindowsApplication1_Namespace1_Service1` を使用します。

My.WebServices オブジェクトのプロパティに初めてアクセスすると、対応する Web サービスの新しいインスタンスが作成され、プロパティに格納されます。以降は、そのプロパティにアクセスすると Web サービスのインスタンスが返されます。

Web サービスを破棄するには、その Web サービスを表すプロパティに **Nothing** を代入します。これにより、プロパティ Set アクセス操作子が格納値に **Nothing** を代入します。このプロパティに **Nothing** 以外の値を代入した場合は、setter が ArgumentException 例外をスローします。

My.WebServices オブジェクトのプロパティに Web サービスのインスタンスが格納されているかどうかを調べるには、**Is** または **IsNot** 演算子を使用します。これらの演算子を使用して、プロパティの値が **Nothing** かどうかを確認します。

メモ :

通常は、**Is** または **IsNot** 演算子が比較を実行するためには、プロパティの値を読み取る必要があります。しかし、プロパティの現在の値が **Nothing** である場合は、値を読み取ろうとすると Web サービスの新しいインスタンスが作成され、そのインスタンスが返されてしまいます。そこで、Visual Basic コンパイラは **My.WebServices** オブジェクトのプロパティを特別扱いとし、**Is** または **IsNot** 演算子がプロパティの値を変更せずにプロパティのステータスを確認できるようにしています。

処理手順

My.Forms オブジェクトに関連する操作の例を次の表に示します。

目的	参照項目
Web サービスを非同期で呼び出し、完了時にイベントを処理する	方法 : Web サービスを非同期で呼び出す

使用例

この例では、`TemperatureConverter` XML Web サービスの `FahrenheitToCelsius` メソッドを呼び出し、結果を返します。

VB

```
Function ConvertFromFahrenheitToCelsius( _
    ByVal dFahrenheit As Double) _
    As Double
```

```
        Return My.WebServices.TemperatureConverter.FahrenheitToCelsius(dFahrenheit)
    End Function
```

この例を実行するためには、プロジェクトが `Converter` という名前の Web サービスを参照していて、その Web サービスが `ConvertTemperature` メソッドを公開している必要があります。詳細については、「方法 : マネージ コードを使用して XML Web サービスにアクセスする」を参照してください。

このコードは Web アプリケーション プロジェクトでは実行できません。

必要条件

プロジェクトの種類別の使用可/不可

プロジェクトの種類	使用可能
Windows アプリケーション	○
クラス ライブラリ	○
コンソール アプリケーション	○
Windows コントロール ライブラリ	○
Web コントロール ライブラリ	○
Windows サービス	○
Web サイト	X

参照

処理手順

方法 : マネージ コードを使用して XML Web サービスにアクセスする

関連項目

SoapHttpClientProtocol

ArgumentException

概念

アプリケーションの Web サービスへのアクセス

その他の技術情報

マネージ コードを使用した XML Web サービスへのアクセス

TextFieldParser オブジェクト

構造化テキスト ファイルの解析に使用するメソッドとプロパティを提供します。

```
Public Class TextFieldParser
```

例外

例外を引き起こす可能性のある状態を次に示します。

- テキスト フィールドが指定された形式に対応していない。たとえば、固定幅のファイルでは、いずれかのフィールドが指定された幅に収まらない場合です。(MalformedLineException).

解説

TextFieldParser オブジェクトは、構造化テキスト ファイルの解析に使用するメソッドとプロパティを提供します。テキスト ファイルを **TextFieldParser** で解析するのは、テキスト ファイルを反復処理するのと同じです。**ReadFields** メソッドでテキストのフィールドを抽出するのは、文字列を分割するのと同じです。

TextFieldParser では、デリミタで区切られたファイルと固定幅のファイルの 2 種類のファイルを解析できます。**Delimiters** や **HasFieldsEnclosedInQuotes** などの一部のプロパティは、デリミタで区切られたファイルでのみ有効ですが、**FieldWidths** プロパティは固定幅のファイルでのみ有効です。

処理手順

Microsoft.VisualBasic.FileIO.TextFieldParser オブジェクトに関連するタスクの例を次の表に示します。

目的	参照項目
区切り記号で区切られたテキスト ファイルから読み込む	方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
固定幅のテキスト ファイルから読み込む	方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
複数の形式を持つテキスト ファイルから読み込む	方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

使用例

この例では、タブ区切りのテキスト ファイル `Bigfile` を解析します。

VB

```
Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser _
("c:\logs\bigfile")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    MyReader.Delimiters = New String() {vbTab}
    Dim currentRow As String()
    'Loop through all of the fields in the file.
    'If any lines are corrupt, report an error and continue parsing.
    While Not MyReader.EndOfData
        Try
            currentRow = MyReader.ReadFields()
            ' Include code here to handle the row.
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
            " is invalid.  Skipping")
        End Try
    End While
End Using
```

この例は、読み取ったフィールドを処理する関数 `processFields` が存在することを前提としています。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

処理手順

例外のトラブルシューティング : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

関連項目

TextFieldParser オブジェクトのメンバ
TextFieldParser.CommentTokens プロパティ
TextFieldParser.Delimiters プロパティ
TextFieldParser.EndOfData プロパティ
TextFieldParser.ErrorLine プロパティ
TextFieldParser.ErrorLineNumber プロパティ
TextFieldParser.FieldWidths プロパティ
TextFieldParser.HasFieldsEnclosedInQuotes プロパティ
TextFieldParser.LineNumber プロパティ
TextFieldParser.TextFieldType プロパティ
TextFieldParser.TrimWhiteSpace プロパティ
TextFieldParser.Close メソッド
TextFieldParser.PeekChars メソッド
TextFieldParser.ReadFields メソッド
TextFieldParser.ReadLine メソッド
TextFieldParser.ReadToEnd メソッド
TextFieldParser.SetDelimiters メソッド
TextFieldParser.SetFieldWidths メソッド
My.Computer.FileSystem.OpenTextFieldParser メソッド
Microsoft.VisualBasic.FileIO.TextFieldParser

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser オブジェクトのメンバ

TextFieldParser には、次の表に示すような構造化テキスト **files** を解析するためのメソッドやプロパティが用意されています。

プロパティ

CommentTokens	**String** 型です。パーサーは指定されたコメント トークンで始まる行をスキップします。
区切り記号	**String** 型です。テキスト ファイルの区切り記号を指定します。区切り記号入りファイルに対してのみ有効です。
EndOfData	**Boolean** 型です。現在のカーソル位置とファイルの末尾との間に、空白行とコメント行しかない場合に **True** を返します。
ErrorLine	**String** 型です。直前に MalformedLineException を発生させた行を返します。既定値は、"" です。
ErrorLineNumber	**Long** 型です。直前に **MalformedLineException** を発生させた行番号を返します。既定値は、-1 です。
FieldWidths	**Integer** 型です。テキスト ファイルの各列の幅を表します。固定幅のファイルに対してのみ有効です。
HasFieldsEnclosedInQuotes	**Boolean** 型です。フィールドを二重引用符で囲むことができる場合に **True** を返します。区切り記号入りファイルに対してのみ有効です。既定値は **True** です。
LineNumber	**Integer** 型です。現在の行番号を返すか、解析可能な文字がなくなった場合には *-1* を返します。
TextFieldType	**FieldType** 型です。区切り記号入りか固定幅の、ファイルの種類を表します。既定値は **Delimited** です。
TrimWhiteSpace	**Boolean** 型です。フィールド値の前後の空白を切り取るかどうかを指定します。

メソッド

Close	基になるストリームを閉じます。
PeekChars	指定された数の文字を、カーソルを進めずに読み取ります。
ReadFields	現在の行にあるすべてのフィールドを読み込み、それを文字列の配列として返して、カーソルを次の行に進めます。
ReadLine	現在の行を **String** 型で返し、カーソルを次の行に進めます。
ReadToEnd	ストリームの残りを読み取り、それを **String** 型で返します。
SetDelimiters	リーダーの区切り記号に指定された値を設定し、フィールドの種類に **Delimited** を設定します。
SetFieldWidths	リーダーの区切り記号に指定された値を設定し、*field type* に **FixedWidth** を設定します。

参照

関連項目

TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser コンストラクタ

TextFieldParser クラスの新しいインスタンスを初期化します。

```
' Usage
Dim value As New TextFieldParser(path)
Dim value As New TextFieldParser(path, defaultEncoding)
Dim value As New TextFieldParser(path, defaultEncoding, detectEncoding)
Dim value As New TextFieldParser(stream)
Dim value As New TextFieldParser(stream, defaultEncoding)
Dim value As New TextFieldParser(stream, defaultEncoding, detectEncoding)
Dim value As New TextFieldParser(stream, defaultEncoding, detectEncoding, leaveOpen)
Dim value As New TextFieldParser(reader)
' Declaration
Public Sub New( _
   ByVal path As String _
)
' -or-
Public Sub New( _
   ByVal path As String, _
   ByVal defaultEncoding As System.Text.Encoding _
)
' -or-
Public Sub New( _
   ByVal path As String, _
   ByVal defaultEncoding As System.Text.Encoding, _
   ByVal detectEncoding As Boolean _
)
' -or-
Public Sub New( _
   ByVal stream As Stream _
)
' -or-
Public Sub New( _
   ByVal stream As Stream, _
   ByVal defaultEncoding As System.Text.Encoding _
)
' -or-
Public Sub New( _
   ByVal stream As Stream, _
   ByVal defaultEncoding As System.Text.Encoding, _
   ByVal detectEncoding As Boolean _
)
' -or-
Public Sub New( _
   ByVal stream As Stream, _
   ByVal defaultEncoding As System.Text.Encoding, _
   ByVal detectEncoding As Boolean, _
   ByVal leaveOpen As Boolean _
)
' -or-
Public Sub New( _
   ByVal reader As TextReader _
)
```

パラメータ

path

String。解析されるファイルの絶対パス。

stream

Stream。解析されるストリーム。

defaultEncoding

Encoding。エンコーディングをファイルから特定できない場合に使用される文字エンコーディング。既定値は UTF8 です。

detectEncoding

Boolean。ファイルの先頭でバイト順マークを探すかどうかを指定します。既定値は **True** です。

leaveOpen

Boolean。**TextFieldParser** オブジェクトを閉じるときに、*stream* を開いたままにするかどうかを指定します。既定値は **False** です。

reader

TextReader。解析される **TextReader** ストリーム。

解説

path、*stream*、または *reader* の各パラメータによって表されるファイルまたはストリームを解析するために、新しい **TextFieldParser** オブジェクトを作成します。

defaultEncoding パラメータが指定されている場合、このコンストラクタはそれを既定のエンコーディングとして使用します。

detectEncoding パラメータが **True** の場合、このコンストラクタは、ファイルまたはストリームの最初の 3 バイトを調べてエンコーディングを検出しようとします。ファイルが適切なバイト順マークで始まっている場合は、UTF-8、リトル エンディアン Unicode、およびビッグ エンディアン Unicode の各テキストが自動的に認識されます。それ以外の場合は、*defaultEncoding* で指定されたエンコーディングが使用されます。

注意
特定のカルチャ設定で文字セットをコンパイルし、同じ文字を別のカルチャ設定で取得する場合、文字を解釈できずに例外がスローされる可能性があります。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

TextFieldParser オブジェクト
Microsoft.VisualBasic.FileIO.TextFieldParser.#ctor(System.IO.Stream)

TextFieldParser.Close メソッド

現在の **TextFieldParser** オブジェクトを閉じます。

```
' Usage
TextFieldParserObject.Close()
' Declaration
Public Sub Close()
```

解説

このメソッドは、**TextFieldParser** オブジェクトを閉じます。

使用例

次のコード例は、**TextFieldParser** `FileReader` を閉じます。

VB

```
FileReader.Close()
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

関連項目

TextFieldParser オブジェクト

Microsoft.VisualBasic.FileIO.TextFieldParser.Close

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.CommentTokens プロパティ

コメント トークンを定義します。コメント トークンである文字列を行の先頭に置くと、その行がコメントであり、パーサーによって無視されることを示します。

```
' Usage
Dim value As String() = TextFieldParserObject.CommentTokens
' Declaration
Public Property CommentTokens As String()
```

戻り値

String ().

例外

例外を引き起こす可能性のある状態を次に示します。

- コメント トークンに空白が含まれている場合 (ArgumentException)。

解説

これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

長さ 0 のコメント トークンは無視されます。

使用例

次の例では、単一引用符 (') で始まる **TextFieldParser** 行と `FileReader` 行が無視されるように指定します。

VB

```
FileReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
FileReader.CommentTokens = New String() {"'"}
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

TextFieldParser オブジェクト
Microsoft.VisualBasic.FileIO.TextFieldParser.CommentTokens

TextFieldParser.Delimiters プロパティ

テキスト ファイルの区切り記号を定義します。

```
' Usage
Dim value As String() = TextFieldParserObject.Delimiters
' Declaration
Public Property Delimiters As String()
```

戻り値

String ().

例外

例外を引き起こす可能性のある状態を次に示します。

- 改行文字、空の文字列、または **Nothing** を区切り記号として使用しています (ArgumentException)。

解説

このプロパティは、**TextFieldType** = **FieldType.Delimited** の場合にだけ意味を持ちます。

テキスト ファイルの区切り記号の定義は、**SetDelimiters** メソッドでも行うことができます。詳細については、「TextFieldParser.SetDelimiters メソッド」を参照してください。

処理手順

Delimiters プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
区切り記号で区切られたテキスト ファイルから読み込む	方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る

使用例

次の例は、**TextFieldParser** オブジェクトである `FileReader` の区切り記号に、コンマ (,) を指定します。

VB

```
FileReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
FileReader.Delimiters = New String() {","}
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

関連項目

TextFieldParser オブジェクト
TextFieldParser.SetDelimiters メソッド
Microsoft.VisualBasic.FileIO.TextFieldParser.Delimiters

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.EndOfData プロパティ

現在のカーソル位置とファイルの末尾との間に、空白行とコメント行しかない場合に **True** を返します。

```
' Usage
Dim value As Boolean = TextFieldParserObject.EndOfData
' Declaration
Public ReadOnly Property EndOfData As Boolean
```

戻り値

Boolean.

解説

このプロパティはファイルの読み取りの際に、データの最後が読み取られたかどうかを判断するために使用します。

処理手順

EndOfData プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
区切り記号入りファイルからの読み取り	方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
固定幅のファイルからの読み取り	方法 : Visual Basic で固定幅のテキスト ファイルを読み取る

使用例

次の例は **EndofData** プロパティを使用して、ファイル内のすべてのフィールドを **TextFieldReader** と `FileReader` を使ってループ処理します。

VB

```
Dim StdFormat As Integer() = {5, 10, 11, -1}
Dim ErrorFormat As Integer() = {5, 5, -1}
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\testfile.txt")
    FileReader.TextFieldType = FileIO.FieldType.FixedWidth
    FileReader.FieldWidths = StdFormat
    Dim CurrentRow As String()
    While Not FileReader.EndOfData
        Try
            Dim RowType As String = FileReader.PeekChars(3)
            If String.Compare(RowType, "Err") = 0 Then
                ' If this line describes an error, the format of the row will be different.
                FileReader.SetFieldWidths(ErrorFormat)
                CurrentRow = FileReader.ReadFields
                FileReader.SetFieldWidths(StdFormat)
            Else
                ' Otherwise parse the fields normally
                CurrentRow = FileReader.ReadFields
                For Each newString As String In CurrentRow
                    My.Computer.FileSystem.WriteAllText("newFile.txt", newString, True)
                Next
            End If
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
        End Try
    End While
End Using
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
SecurityPermission	コードに適用されたセキュリティ アクセス許可のセットを記述します。関連する列挙値 : ControlEvidence。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

TextFieldParser オブジェクト

Microsoft.VisualBasic.FileIO.TextFieldParser.EndOfData

TextFieldParser.ErrorLine プロパティ

直前に MalformedLineException 例外を発生させた行を返します。

```
' Usage
Dim value As String = TextFieldParserObject.ErrorLine
' Declaration
Public ReadOnly Property ErrorLine As String
```

戻り値

String.

解説

MalformedLineException 例外が一度もスローされていなければ、空の文字列が返されます。

TextFieldParser.ErrorLineNumber プロパティ を使用すると、例外を発生させた行の番号を表示できます。

処理手順

ErrorLine プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
区切り記号で区切られたファイルからの読み取り	方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
固定幅のファイルからの読み込み	方法 : Visual Basic で固定幅のテキスト ファイルを読み取る

使用例

次の例は、**ErrorLine** プロパティを使って、現在の **MalformedLineException** 例外を発生させた行を表示します。

VB

```
Dim FileReader As Microsoft.VisualBasic.FileIO.TextFieldParser
FileReader = My.Computer.FileSystem.OpenTextFieldParser("C:\test.txt")
Dim currentRow As String()
While Not FileReader.EndOfData
    Try
        currentRow = FileReader.ReadFields
        For Each currentField As String In currentRow
            My.Computer.FileSystem.WriteAllText _
            ("C://testfile.txt", currentField, True)
        Next
    Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
        MsgBox("Line " & FileReader.ErrorLine & " is not valid.")
    End Try
End While
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

方法 : Visual Basic で固定幅のテキスト ファイルを読み取る

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する

関連項目

TextFieldParser オブジェクト
Microsoft.VisualBasic.FileIO.TextFieldParser.ErrorLine

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.ErrorLineNumber プロパティ

直前に MalformedLineException 例外を発生させた行の番号を返します。

```
' Usage
Dim value As Long = TextFieldParserObject.ErrorLineNumber
' Declaration
Public ReadOnly Property ErrorLineNumber As Long
```

戻り値

Long.

例外

このプロパティは例外をスローしません。

解説

MalformedLineException 例外が一度もスローされていなければ、-1 が返されます。

TextFieldParser.ErrorLine プロパティ を使用すると、例外を発生させた行の番号を表示できます。行番号を調べる際、空白行とコメントは無視されます。

処理手順

ErrorLineNumber プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
区切り記号で区切られたファイルからの読み取り	方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
固定幅のファイルからの読み込み	方法 : Visual Basic で固定幅のテキスト ファイルを読み取る

使用例

次の例は、**ErrorLineNumber** プロパティを使って、現在の **MalformedLineException** 例外を発生させた行の位置を表示します。

VB

```
Dim FileReader As Microsoft.VisualBasic.FileIO.TextFieldParser
FileReader = My.Computer.FileSystem.OpenTextFieldParser("C:\test.txt")
Dim currentRow As String()
While Not FileReader.EndOfData
    Try
        currentRow = FileReader.ReadFields
        For Each currentField As String In currentRow
            My.Computer.FileSystem.WriteAllText _
            ("C://testfile.txt", currentField, True)
        Next
    Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
        MsgBox("Line " & FileReader.ErrorLineNumber & " is not valid.")
    End Try
End While
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る
方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する

関連項目

TextFieldParser オブジェクト
Microsoft.VisualBasic.FileIO.TextFieldParser.ErrorLineNumber

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.FieldWidths プロパティ

解析対象のテキスト ファイルに含まれる各列の幅を示します。

```
' Usage
Dim value As Integer() = TextFieldParserObject.FieldWidths
' Declaration
Public Property FieldWidths As Integer()
```

戻り値

Integer ().

例外

例外を引き起こす可能性のある状態を次に示します。

- 配列の最後のエントリ以外の場所で、幅の値が 0 以下になっている (ArgumentException)。

解説

このプロパティは、TextFieldParser.TextFieldType プロパティ = **FieldType.FixedWidth** の場合にのみ意味を持ちます。配列の最後のエントリが 0 以下の場合は、そのフィールドは可変長と見なされます。

フィールドの幅を設定するには、**SetFieldWidths** メソッドも使用できます。詳細については、「TextFieldParser.SetFieldWidths メソッド」を参照してください。

処理手順

FieldWidths プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
固定幅のテキスト ファイルから読み込む	方法 : Visual Basic で固定幅のテキスト ファイルを読み取る

使用例

この例では、列の幅を指定して `ParserText.txt` ファイルを読み取ります。列の幅は、1 列目は 5 文字、2 列目は 10 文字、3 列目は 11 文字、4 列目は可変長とします。

VB

```
Using MyReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.FixedWidth
    MyReader.FieldWidths = New Integer() {5, 10, 11, -1}
    Dim currentRow As String()
    While Not MyReader.EndOfData
        Try
            currentRow = MyReader.ReadFields()
            Dim currentField As String
            For Each currentField In currentRow
                MsgBox(currentField)
            Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
            "is not valid and will be skipped.")
        End Try
    End While
End Using
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る
例外のトラブルシューティング : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

関連項目

TextFieldParser オブジェクト
FieldType 列挙型
Microsoft.VisualBasic.FileIO.TextFieldParser.FieldWidths

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.HasFieldsEnclosedInQuotes プロパティ

区切り記号入りファイルを解析するときに、フィールドが引用符で囲まれているかどうかを示します。

```
' Usage
Dim value As Boolean = TextFieldParserObject.HasFieldsEnclosedInQuotes
' Declaration
Public Property HasFieldsEnclosedInQuotes As Boolean
```

戻り値

Boolean.

解説

これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

このプロパティが **True** の場合、パーサーはフィールドが引用符 (" ") で囲まれていて、改行コードを含んでいる可能性もあると想定します。

たとえば `abc, "field2a,field2b", field3` のようにフィールドが引用符で囲まれていて、このプロパティが **True** に設定されている場合は、引用符で囲まれているテキスト全体がそのまま返されます。この例では `abc|field2a,field2b|field3` が返されます。このプロパティを **False** に設定すると、この例では `abc|"field2a|field2b"|field3` が返されます。

使用例

この例では、`myReader` の **HasFieldsEnclosedInQuotes** プロパティを **True** に設定します。

VB

```
FileReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
FileReader.Delimiters = New String() {","}
FileReader.CommentTokens = New String() {""}
FileReader.HasFieldsEnclosedInQuotes = True
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

関連項目

TextFieldParser オブジェクト

Microsoft.VisualBasic.FileIO.TextFieldParser.HasFieldsEnclosedInQuotes

TextFieldParser.LineNumber プロパティ

現在の行番号を返すか、ストリームに解析可能な文字がなくなった場合は -1 を返します。

```
' Usage
Dim value As Long = TextFieldParserObject.LineNumber
' Declaration
Public ReadOnly Property LineNumber As Long
```

戻り値

Integer.

解説

これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

行番号を調べる際、空白行とコメントは無視されません。

使用例

次の例は、テキスト ファイルから "Jones" という名前を検索し、検出された行を報告します。

VB

```
Using FileReader As New Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    FileReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    FileReader.Delimiters = New String() {","}
    Dim currentRow As String()
    While Not FileReader.EndOfData
        Try
            currentRow = FileReader.ReadFields()
            Dim currentField As String
            For Each currentField In currentRow
                If currentField = "Jones" Then
                    MsgBox("The name Jones occurs on line " & _
                    FileReader.LineNumber)
                End If
            Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & _
           "is not valid and will be skipped.")
        End Try
    End While
End Using
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Microsoft Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

関連項目

TextFieldParser オブジェクト

Microsoft.VisualBasic.FileIO.TextFieldParser.LineNumber

TextFieldParser.PeekChars メソッド

指定された数の文字を、カーソルを進めずに読み取ります。

```
' Usage
Dim value As String = TextFieldParserObject.PeekChars(numberOfChars)
' Declaration
Public Function PeekChars( _
   ByVal numberOfChars As Integer _
) As String
```

パラメータ

numberOfChars

読み取る文字数を指定する **Int32** です。必ず指定します。

戻り値

String.

例外

例外がスローされる可能性のある状態を次に示します。

- *numberOfChars* が 0 未満です (ArgumentException)。

解説

numberOfChars の値は、その行の文字数の合計よりも小さいことが必要です。そうでない場合、**PeekChars** から返される文字列は、その行の長さに切り取られます。

空白行は無視されます。

行末文字は返されません。

PeekChars メソッドでは解析は行われません。区切り記号入りフィールド内の行末文字は、行の末尾として解釈されます。

処理手順

PeekChars メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
フィールドを解析する前に、フィールドの形式を調べる	方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

使用例

次の例は **PeekChars** を使用してデータの末尾を探し、その地点でファイルの解析を停止します。

VB

```
Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    MyReader.Delimiters = New String() {","}
    MyReader.CommentTokens = New String() {"'"}
    Dim currentRow As String()
    While (MyReader.PeekChars(1) IsNot "")
        Try
            currentRow = MyReader.ReadFields()
            For Each currentField As String In currentRow
                My.Computer.FileSystem.WriteAllText _
               ("C://testfile.txt", currentField, True)
            Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
        End Try
    End While
End Using
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
FileIOPermission	ファイルとフォルダへのアクセス許可を制御します。関連する列挙値 : Unrestricted。
SecurityPermission	コードに適用されたセキュリティ アクセス許可のセットを記述します。関連する列挙値 : ControlEvidence。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

関連項目

TextFieldParser オブジェクト

PeekChars

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

その他の技術情報

Visual Basic でのファイルの読み取り

TextFieldParser.ReadFields メソッド

現在の行にあるすべてのフィールドを読み込み、それを文字列の配列として返して、カーソルを次のデータを含む行に進めます。

```
' Usage
Dim value As String() = TextFieldParserObject.ReadFields()
' Declaration
Public Function ReadFields() As String()
```

戻り値

String ().

例外

例外がスローされる可能性のある状態を次に示します。

- 指定された形式を使ってフィールドを解析できません (MalformedLineException)。

解説

ReadFields メソッドを使って複数の形式のテキスト ファイルを解析できるように、メソッドが呼び出されるたびに **TextFieldType**、**Delimiters**、および **FieldWidths** の値が (これらが指定されている場合に) 調べられます。**TextFieldType** プロパティと **FieldWidths** または **Delimiters** のプロパティは、必要に応じて正しく設定する必要があります。**TextFieldType** に **Delimited** が設定され、**Delimiters** が設定されていない場合、または **TextFieldType** に **FixedWidth** が設定され、**FieldWidths** が設定されていない場合は、エラーがスローされます。

ReadFields が空白行を検出した場合は、空白行がスキップされて次の空白でない行が返されます。

処理手順

ReadFields メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
区切り記号で区切られたファイルからの読み取り	方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
固定幅のファイルからの読み込み	方法 : Visual Basic で固定幅のテキスト ファイルを読み取る

使用例

次の例は、**ReadFields** メソッドを使ってファイル `ParserText.txt` からデータを読み取り、フィールドを `Testfile.txt` に書き込みます。

VB

```
Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    MyReader.Delimiters = New String() {","}
    Dim currentRow As String()
    While Not MyReader.EndOfData
        Try
            currentRow = MyReader.ReadFields()
            For Each currentField As String In currentRow
                My.Computer.FileSystem.WriteAllText _
                ("C://testfile.txt", currentField, True)
            Next
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
        End Try
    End While
End Using
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

必要なアクセス許可を次に示します。

アクセス許可	説明
FileIOPermission	関連する列挙値 : Unrestricted。
SecurityPermission	関連する列挙値 : ControlEvidence。

詳細については、「コード アクセス セキュリティ」および「アクセス許可の要求」を参照してください。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

TextFieldParser オブジェクト
TextFieldParser.TextFieldType プロパティ
TextFieldParser.FieldWidths プロパティ
TextFieldParser.Delimiters プロパティ
TextFieldParser.ReadLine メソッド
Microsoft.VisualBasic.FileIO.TextFieldParser.ReadFields

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.ReadLine メソッド

現在の行を文字列として返し、カーソルを次の行に進めます。

```
' Usage
Dim value As String = TextFieldParserObject.ReadLine()
' Declaration
Public Function ReadLine() As String
```

戻り値

String

例外

例外がスローされる可能性のある状態を次に示します。

- 対象のファイルが存在しません (FileNotFoundException)。

解説

ReadLine メソッドは、解析を実行しません。区切られたフィールドの中に行末文字があれば、そこで行が終了すると解釈します。

ファイルの最後に到達した場合、**Nothing** を返します。

使用例

次のコード例は、`ParserText.txt` というファイルを読み取って、内容を `Testfile.txt` に書き込みます。

VB

```
Using MyReader As New Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    MyReader.TextFieldType = Microsoft.VisualBasic.FileIO.FieldType.Delimited
    MyReader.Delimiters = New String() {","}
    Dim currentRow As String
    While Not MyReader.EndOfData
        Try
            currentRow = MyReader.ReadLine()
            My.Computer.FileSystem.WriteAllText _
           ("C://testfile.txt", currentRow, True)
        Catch ex As Microsoft.VisualBasic.FileIO.MalformedLineException
            MsgBox("Line " & ex.Message & " is invalid.  Skipping")
        End Try
    End While
End Using
```

`Testfile.txt` が存在しない場合、このファイルを **WriteAllText** メソッドを使って作成します。

この例では、フィールドを単一の文字列として書き込みます。各行が書き込み先のファイル内で単独の行として表示されるようにするため、**VbCrLf** 文字を各行の最後に追加します。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

関連項目

TextFieldParser オブジェクト

TextFieldParser.ReadFields メソッド
TextFieldParser.ReadToEnd メソッド
Microsoft.VisualBasic.FileIO.TextFieldParser.ReadLine
概念
TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.ReadToEnd メソッド

テキスト ファイルの残りを読み取り、それを文字列で返します。

```
' Usage
Dim value As String = TextFieldParserObject.ReadToEnd()
' Declaration
Public Function ReadToEnd() As String
```

戻り値

String.

解説

これは詳細メンバで、[すべての候補] タブをクリックしないと IntelliSense に表示されません。

ファイルの末尾に到達し、読み込むデータがなくなった場合は、**Nothing** が返されます。

ReadToEnd メソッドは、空白行とコメントを無視しません。

使用例

次の例は、**ReadToEnd** メソッドを使って `ParserText.txt` ファイルの全データを読み取り、`Testfile.txt` ファイルに書き込みます。

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    Dim allText As String = FileReader.ReadToEnd
    My.Computer.FileSystem.WriteAllText("C://testfile.txt", allText, True)
End Using
```

`Testfile.txt` が存在しない場合は、**WriteAllText** メソッドによって作成されます。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

関連項目

TextFieldParser オブジェクト
TextFieldParser.ReadFields メソッド
TextFieldParser.ReadLine メソッド
Microsoft.VisualBasic.FileIO.TextFieldParser.ReadToEnd

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.SetDelimiters メソッド

リーダーの区切り記号に指定された値を設定し、フィールドの種類に **Delimited** を設定します。

```
' Usage
TextFieldParserObject.SetDelimiters(delimiters)
' Declaration
Public Sub SetDelimiters( _
   ByVal delimiters As String() _
)
```

パラメータ

delimiters

String 型の配列です。

例外

例外がスローされる可能性のある状態を次に示します。

- 区切り記号の長さが 0 です (ArgumentException)。

解説

Delimiters プロパティの既存の内容は、このメソッドが設定されたときに消去されます。

このメソッドを使用すると、配列を作成せずに区切り記号を設定できます。

処理手順

SetDelimiters メソッドに関連するタスクの例を次の表に示します。

目的	参照項目
テキスト ファイルを解析する	TextFieldParser オブジェクトによるテキスト ファイルの解析

使用例

次の例は、テキスト フィールドのパーサーを開き、区切り記号に `vbTab` を定義します。

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\logs\test.log")
    FileReader.SetDelimiters(vbTab)
End Using
```

`C:\logs\test.log` のパスを、解析するファイルのパスと名前に書き換えてください。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る

関連項目

TextFieldParser オブジェクト

TextFieldParser.SetFieldWidths メソッド

SetDelimiters
概念
TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.SetFieldWidths メソッド

リーダー用のデリミタを指定の値に設定し、フィールドの種類を **FixedWidth** に設定します。

```
' Usage
TextFieldParserObject.SetFieldWidths(fieldWidths)
' Declaration
Public Sub SetFieldWidths( _
   ByVal fieldWidths As Integer() _
)
```

パラメータ

fieldWidths

Integer の配列。

例外

例外がスローされる可能性のある状態を次に示します。

- デリミタが整数 (ArgumentException) ではありません。

解説

Delimiters プロパティの既存の内容は、設定時にクリアされます。

このメソッドを使うと、配列を作成せずにデリミタを設定できます。

処理手順

SetFieldWidths メソッドに関連するタスクの例を次の表に示します。

タスク	参照項目
テキスト ファイルを解析します。	TextFieldParser オブジェクトによるテキスト ファイルの解析

使用例

このコード例は、テキスト フィールド パーサーを開き、フィールドの幅を `5` に定義します。

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\logs\test.log")
    FileReader.SetFieldWidths(5)
End Using
```

パス `C:\logs\test.log` を、解析するファイルのパスと名前に置き換えてください。

このコード例は、テキスト フィールド パーサーを開き、フィールドの幅を `5`、`10`、および変数に定義します。

VB

```
Using MyReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\logs\test.log")
    MyReader.SetFieldWidths(5, 10, -1)
End Using
```

パス `C:\logs\test.log` を、解析するファイルのパスと名前に置き換えてください。

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照
関連項目
TextFieldParser オブジェクト
TextFieldParser.SetDelimiters メソッド
SetFieldWidths
概念
TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.TextFieldType プロパティ

解析対象のファイルが区切り記号で区切られているか固定幅かを表します。

```
' Usage
Dim value As FieldType = TextFieldParserObject.TextFieldType
' Declaration
Public Property TextFieldType As FieldType
```

戻り値

TextFieldType.

解説

このプロパティの既定値は、区切り記号入りです。

処理手順

TextFieldType プロパティに関連するタスクの例を次の表に示します。

目的	参照項目
区切り記号で区切られたテキスト ファイルから読み込む	方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
固定幅のテキスト ファイルから読み込む	方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
複数の形式を持つテキスト ファイルから読み込む	方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

使用例

この例では、**TextFieldParser** である `FileReader` を作成し、区切り記号入りとして指定しています。

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    FileReader.TextFieldType = _
    Microsoft.VisualBasic.FileIO.FieldType.Delimited
    FileReader.SetDelimiters(",")
End Using
```

この例では、**TextFieldParser** である `FileReader` を作成し、固定幅として指定しています。

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    FileReader.TextFieldType = _
    Microsoft.VisualBasic.FileIO.FieldType.FixedWidth
    FileReader.FieldWidths = New Integer() {5, 10, 11, -1}
End Using
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る
例外のトラブルシューティング : Microsoft.VisualBasic.FileIO.TextFieldParser.MalformedLineException

関連項目

TextFieldParser オブジェクト
FieldType 列挙型
FieldType 列挙型
TextFieldParser

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser.TrimWhiteSpace プロパティ

フィールド値の前後の空白を切り取るかどうかを指定します。

```
' Usage
Dim value As Boolean = TextFieldParserObject.TrimWhiteSpace
' Declaration
Public Property TrimWhiteSpace As Boolean
```

戻り値

Boolean.

解説

このプロパティの既定値は **True** です。

使用例

次の例は **TextFieldParser** と `FileReader` を作成し、**TrimWhiteSpace** プロパティに **True** を設定します。

VB

```
Using FileReader As New _
Microsoft.VisualBasic.FileIO.TextFieldParser("C:\ParserText.txt")
    FileReader.TextFieldType = _
    Microsoft.VisualBasic.FileIO.FieldType.Delimited
    FileReader.SetDelimiters(",")
    FileReader.TrimWhiteSpace = True
End Using
```

必要条件

名前空間 : Microsoft.VisualBasic.FileIO

クラス : TextFieldParser

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

アクセス許可

アクセス許可は不要です。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る
方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

TextFieldParser オブジェクト
My.Computer.FileSystem.OpenTextFieldParser メソッド
Microsoft.VisualBasic.FileIO.TextFieldParser.TrimWhiteSpace

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

演算子 (Visual Basic)

このセクションの内容

関連するセクション

Visual Basic リファレンス

Visual Basic

Visual Basic における演算子の優先順位

1 つの式の中で複数の演算が実行されるとき、式の各部分の演算子は一定の順序に従って評価され、演算が実行されます。この順序を演算子の優先順位と呼びます。

優先順位の規則

異なる種類の演算子が 1 つの式の中で使用されているとき、各演算子は次の規則に従って評価されます。

- 算術演算子と文字列連結演算子の優先順位については後で説明しますが、どちらも比較演算子、論理演算子、およびビット処理演算子よりも優先順位は高くなっています。
- すべての比較演算子は優先順位が等しく、論理演算子およびビット処理演算子よりもすべて優先順位が高くなっていますが、算術演算子および文字列連結演算子よりは優先順位が低くなっています。
- 論理演算子とビット処理演算子の優先順位については後で説明しますが、どちらも算術演算子、文字列連結演算子、および比較演算子よりも優先順位は低くなっています。
- 優先順位の等しい演算子は、式の中の位置に基づいて左から右の順序で評価されます。

優先順位

演算子は、次の優先順位で評価されます。

算術演算子および文字列連結演算子

指数演算 (**^**)

単項恒等演算および否定演算 (**+**, **–**)

乗算および浮動小数点除算 (*, **/**)

整数除算 (****)

剰余演算 (**Mod**)

加算と減算 (**+**、**–**)、文字列連結 (**+**)

文字列連結 (**&**)

算術ビット シフト (**<<**、**>>**)

比較演算子

すべての比較演算子 (**=**、**<>**、**<**、**<=**、**>**、**>=**、**Is**、**IsNot**、**Like**、**TypeOf**...**Is**)。

論理演算子およびビット処理演算子

否定 (**Not**)

積 (**And**、**AndAlso**)

包括的論理和 (**Or**、**OrElse**)

排他的論理和 (**Xor**)

= 演算子は、比較演算子 (等号) としてのみ機能し、代入演算子としては使用しません。

文字列連結演算子 (**&**) は算術演算子ではありませんが、優先順位は算術演算子と同じです。

Is および **IsNot** 演算子は、参照されているオブジェクトを比較する演算子です。この演算子は、2 つのオブジェクトの値を比較しません。2 つのオブジェクト変数が同じオブジェクトを参照しているかどうかを調べるだけです。

結合規則

乗算と除算など、1 つの式の中で同じ優先順位の演算子が同時に使用されるときは、左から右の順で各演算子が評価されます。次に例を示します。

```
Dim n1 As Integer = 96 / 8 / 4
Dim n2 As Integer = (96 / 8) / 4
Dim n3 As Integer = 96 / (8 / 4)
```

最初の式は、*96 / 8* (結果は 12) という除算を評価し、次に *12 / 4* (結果は 3) を評価します。コンパイラは `n1` の演算を左から右へ評価するため、評価は、この順序を `n2` に明示的に指定した場合とまったく同じになります。`n1` と `n2` の両方で、結果は 3 になります。一方、`n3` ではかっこによって *8 / 4* が最初に評価されるので、結果は 48 です。

このような動作から、演算子は Visual Basic では左結合であると言われます。

優先順位と結合規則をオーバーライドする

式の一部を他の部分よりも先に評価するには、かっこを使用します。これによって、優先順位と左結合がオーバーライドされることがあります。Visual Basic は、かっこ内の演算は必ずかっこの外よりも先に実行します。ただし、かっこ内では、さらにかっこを使用している場合を除き、通常の優先順位と結合が適用されます。次に例を示します。

```
Dim a, b, c, d, e, f, g As Double
a = 8.0
b = 3.0
c = 4.0
d = 2.0
e = 1.0
f = a - b + c / d * e
' The preceding line sets f to 7.0. Because of natural operator
' precedence and associativity, it is exactly equivalent to the
' following line.
f = (a - b) + ((c / d) * e)
' The following line overrides the natural operator precedence
' and left associativity.
g = (a - (b + c)) / (d * e)
' The preceding line sets g to 0.5.
```

参照

関連項目

= 演算子 (Visual Basic)
Is 演算子 (Visual Basic)
IsNot 演算子
Like 演算子
TypeOf 演算子 (Visual Basic)
機能別の演算子一覧

概念

Visual Basic の演算子および式

機能別の演算子一覧

以下のカテゴリのいずれかを参照するか、ヘルプの目次のこの部分を開いて Visual Basic の演算子のアルファベット順のリストを表示します。

演算子のカテゴリ

演算子	説明
算術演算子	算術演算を行います。
代入演算子	代入演算を行います。
比較演算子	比較演算を行います。
連結演算子	文字列連結演算を行います。
論理/ビット演算子	論理演算を行います。
ビット シフト演算子	ビット パターンに対する算術シフトを行います。
その他の演算子	その他の演算を行います。

参照

関連項目

Visual Basic における演算子の優先順位

概念

Visual Basic の演算子および式

& 演算子 (Visual Basic)

2 つの式に対して文字列連結を行います。

```
result = expression1 & expression2
```

指定項目

result

必ず指定します。文字列 (**String**) 変数またはオブジェクト (**Object**) 変数を指定します。

expression1

必ず指定します。文字列型 (**String**) に拡大変換されるデータ型の式です。

expression2

必ず指定します。文字列型 (**String**) に拡大変換されるデータ型の式です。

解説

expression1 または *expression2* のデータ型が文字列型 (**String**) でなくても文字列型 (**String**) に拡大変換される場合は、文字列型 (**String**) に変換されます。いずれかのデータ型が文字列型 (**String**) に拡大変換されない場合は、コンパイラでエラーが発生します。

通常、演算結果 *result* のデータ型は文字列型 (**String**) です。どちらか、または両方の式の評価が Nothing の場合、または、System.DBNull.Value の値を持つ場合、"" という値を持つ文字列として扱われます。

メモ：

& 演算子は オーバーロード できます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。このようなクラスまたは構造体でこの演算子を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

& 演算子を使って文字列の連結を行う例を次に示します。結果は、2 つの文字列オペランドの連結を表す文字列値になります。

VB

```
Dim sampleStr As String
sampleStr = "Hello" & " World"
' The preceding statement sets sampleStr to "Hello World".
```

参照

関連項目

&= 演算子 (Visual Basic)
連結演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

Visual Basic の連結演算子

&= 演算子 (Visual Basic)

文字列 (**String**) 式を文字列型 (**String**) の変数またはプロパティに連結し、その結果を変数またはプロパティに代入します。

```
variableorproperty &= expression
```

指定項目

variableorproperty

必ず指定します。文字列 (**String**) の変数またはプロパティを指定します。

expression

必ず指定します。任意のブール型 (**String**) の式を指定します。

解説

&= 演算子の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティを ReadOnly (Visual Basic) にすることはできません。**&=** 演算子は、右辺の値を左辺の変数またはプロパティに代入します。

オーバーロード

& 演算子 (Visual Basic) はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。**&** 演算子のオーバーロードは、**&=** 演算子の動作に影響を与えます。コード内で、**&** をオーバーロードするクラスや構造体で **&=** が使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次の例では、**&=** 演算子を使って、2 つの文字列型 (**String**) の変数を連結し、結果を最初の変数に代入します。

VB

```
Dim var1 As String = "Hello "
Dim var2 As String = "World!"
var1 &= var2
' The value of var1 is now "Hello World!".
```

参照

関連項目

& 演算子 (Visual Basic)
+= 演算子 (Visual Basic)
代入演算子
連結演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

代入ステートメント

* 演算子 (Visual Basic)

2 つの数値の積を返します。

```
number1 * number2
```

指定項目

number1

必ず指定します。任意の数式を指定します。

number2

必ず指定します。任意の数式を指定します。

結果

結果は、引数 *number1* と引数 *number2* の積です。

サポートされている型

unsigned 型と浮動小数点型を含むすべての数値型、および 10 進型 (**Decimal**)。

解説

結果のデータ型は、オペランドの型によって決まります。オペランドの型と結果のデータ型の関係を次の表に示します。

オペランドのデータ型	結果のデータ型
両方の式が整数型 (SByte、Byte、Short、UShort、Integer、UInteger、Long、ULong)	*number1* と *number2* のデータ型に対して適切な数値データ型。「演算子の結果のデータ型」に示す「整数演算」の表を参照してください。
両方の式が 10 進型 (Decimal)	**Decimal**
両方の式が単精度浮動小数点型 (Single)	**Single**
どちらか一方の式が浮動小数点型 (**Single** または Double) だが、両方が **Single** ではない場合 (**Decimal** は浮動小数点型ではありません)	**Double**

Nothing に評価される式は、0 として扱われます。

オーバーロード

* 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。この演算子がコード内でこのようなクラスや構造体に対して使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

* 演算子を使って 2 つの数値を乗算する例を次に示します。結果は、2 つのオペランドの積です。

VB

```
Dim testValue As Double
testValue = 2 * 2
' The preceding statement sets testValue to 4.
testValue = 459.35 * 334.9
' The preceding statement sets testValue to 153836.315.
```

参照

関連項目

*= 演算子 (Visual Basic)
算術演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧
概念
Visual Basic における算術演算子

*= 演算子 (Visual Basic)

変数またはプロパティの値を式で指定された値で乗算し、その結果を変数またはプロパティに代入します。

```
variableorproperty *= expression
```

指定項目

variableorproperty

必ず指定します。任意の数値変数またはプロパティです。

expression

必ず指定します。任意の数式を指定します。

解説

=** 演算子の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティは ReadOnly (Visual Basic) にすることはできません。=** 演算子は、右側にある値を、左側にある変数またはプロパティに代入します。

オーバーロード

* 演算子 (Visual Basic) は オーバーロード できます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。***** 演算子をオーバーロードすると、***=** 演算子の動作に影響します。***** をオーバーロードしているクラスまたは構造体で ***=** を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次の例では、***=** 演算子を使って、最初の整数型 (**Integer**) の変数を 2 番目の整数型 (**Integer**) の変数で乗算し、結果を最初の変数に代入します。

VB

```
Dim var1 As Integer = 10
Dim var2 As Integer = 3
var1 *= var2
' The value of var1 is now 30.
```

参照

関連項目

* 演算子 (Visual Basic)
代入演算子
算術演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

代入ステートメント

+ 演算子 (Visual Basic)

2 つの数を加算するか、数値式の正の値を返します。2 つの文字列式を連結するためにも使用できます。

```
expression1 + expression2
- or -
+ expression1
```

指定項目

expression1

必ず指定します。任意の数式または文字列式を指定します。

expression2

+ 演算子が負の値を計算する場合を除き、必ず必要です。任意の数式または文字列式を指定します。

結果

expression1 と *expression2* がどちらも数値の場合、結果は 2 つの合計を計算した値になります。

expression2 が指定されなければ、**+** 演算子は単項恒等演算子となり、式の値は変わりません。つまり、演算において *expression1* の符号が保持されるため、*expression1* が負であれば結果は負になります。

expression1 と *expression2* がどちらも文字列であれば、結果は 2 つの値を連結した文字列になります。

expression1 と *expression2* の型が異なる場合は、2 つの型、内容、および Option Strict ステートメント の設定に応じて異なる処理が行われます。詳細については、「解説」の表を参照してください。

サポートされている型

unsigned 型と浮動小数点型を含むすべての数値型、および 10 進型 (**Decimal**)、文字列型 (**String**)。

解説

一般に、**+** は可能であれば加算を行い、両方の式が文字列の場合にだけ連結を行います。

どちらの式も **Object** でない場合、Visual Basic は次の処理を行います。

式のデータ型	コンパイラによる処理
両方の式が数値型 (**SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、**ULong**、**Decimal**、**Single**、または **Double**) の場合	加算。結果のデータ型は *expression1* と *expression2* のデータ型に対して適切な数値型になります。「演算子の結果のデータ型」に示す「整数演算」の表を参照してください。
両方の式が文字列型 (**String**) の場合	文字列連結。
一方が数値データ型で他方が文字列の場合	**Option Strict** が **On** であれば、コンパイル エラーが発生します。 **Option Strict** が **Off** であれば、**String** が倍精度浮動小数点数型 (**Double**) に暗黙的に変換されて加算されます。 **String** を倍精度浮動小数点数型 (**Double**) に変換できない場合は、InvalidCastException 例外がスローされます。
一方が数値で他方が Nothing (Visual Basic) の場合	**Nothing** を値 0 として加算が行われます。
一方が文字列で他方が **Nothing** の場合	**Nothing** を値 "" として連結が行われます。

一方の式が **Object** 式である場合、Visual Basic は次の処理を行います。

式のデータ型	コンパイラによる処理

Object 式に数値が格納され、もう一方が数値型の場合	**Option Strict** が **On** であれば、コンパイル エラーが発生します。 **Option Strict** が **Off** であれば、加算が行われます。
Object 式に数値が格納され、もう一方が文字列型の場合 **String**	**Option Strict** が **On** であれば、コンパイル エラーが発生します。 **Option Strict** が **Off** であれば、**String** が **Double** に暗黙的に変換されて加算されます。 **String** を倍精度浮動小数点数型 (**Double**) に変換できない場合は、**InvalidCastException** 例外がスローされます。
Object 式に文字列が格納され、もう一方が数値型の場合	**Option Strict** が **On** であれば、コンパイル エラーが発生します。 **Option Strict** が **Off** であれば、文字列の **Object** が **Double** に暗黙的に変換されて加算されます。 文字列の (**Object**) を倍精度浮動小数点数型 (**Double**) に変換できない場合は、**InvalidCastException** 例外がスローされます。
Object 式に文字列が格納され、もう一方が文字列型の場合 **String**	**Option Strict** が **On** であれば、コンパイル エラーが発生します。 **Option Strict** が **Off** であれば、**Object** が **String** に暗黙的に変換されて連結されます。

両方の式が **Object** である場合、Visual Basic は次の処理を行います (**Option Strict Off** の場合のみ)。

式のデータ型	**コンパイラによる処理**
両方の **Object** 式に数値が格納されている場合	加算。
両方の **Object** 式が文字列型 (**String**) である場合	文字列連結。
一方の **Object** 式に数値が格納され、他方に文字列が格納されている場合	文字列の **Object** を倍精度浮動小数点数型 (**Double**) に暗黙的に変換して加算します。 文字列の **Object** を数値に変換できない場合は、**InvalidCastException** 例外がスローされます。

いずれかの **Object** 式が Nothing または DBNull に評価される場合、**+** 演算子はそれを値 "" の **String** として扱います。

メモ：

+ 演算子を使用すると、加算と文字列連結のどちらが行われるのか、事前にはわかりにくい場合があります。連結に **&** 演算子を使用することで、あいまいさがなくなり、プログラムの可読性が向上します。

オーバーロード

+ 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。この演算子がコード内でこのようなクラスや構造体に対して使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

+ 演算子を使って数値を加算する例を次に示します。両方のオペランドが数値であれば、Visual Basic は算術演算を行います。算術演算の結果は、2 つのオペランドの合計になります。

VB

```
Dim sumNumber As Integer
sumNumber = 2 + 2
sumNumber = 4257.04 + 98112
' The preceding statements set sumNumber to 4 and 102369.
```

+ 演算子を使用して、文字列を連結することもできます。両方のオペランドが文字列であれば、Visual Basic はこれらを連結します。連結結

果は 2 つのオペランドの内容を前後につなげた単一の文字列になります。

オペランドの型が同じでない場合、結果は Option Strict ステートメント の設定によって異なります。**Option Strict** が **On** であれば、結果は次の例のようになります。

VB

```
Option Strict On
```

VB

```
    Dim var1 As String = "34"
    Dim var2 As Integer = 6
    Dim concatenatedNumber As Integer = var1 + var2
...   ' The preceding statement generates a COMPILER ERROR.
```

Option Strict が **Off** であれば、結果は次の例のようになります。

VB

```
Option Strict Off
```

VB

```
    Dim var1 As String = "34"
    Dim var2 As Integer = 6
    Dim concatenatedNumber As Integer = var1 + var2
...   ' The preceding statement returns 40 after the string in var1 is
    ' converted to a numeric value. This might be an unexpected result.
    ' We do not recommend use of Option Strict Off for these operations.
```

あいまいさをなくすため、連結には + 演算子ではなく **&** 演算子を使うようにしてください。

参照

関連項目

& 演算子 (Visual Basic)
連結演算子 (Visual Basic)
算術演算子 (Visual Basic)
機能別の演算子一覧
Visual Basic における演算子の優先順位
Option Strict ステートメント

概念

Visual Basic における算術演算子

+= 演算子 (Visual Basic)

数式の値を、数値型の変数またはプロパティの値に加算し、その結果を変数またはプロパティに代入します。これを利用して、**String** 式を **String** 変数またはプロパティに連結し、その結果を変数またはプロパティに代入することもできます。

```
variableorproperty += expression
```

指定項目
variableorproperty

必ず指定します。任意の数値または **String** の変数またはプロパティです。

expression

必ず指定します。任意の数式または文字列 (**String**) 式を指定します。

解説

+= 演算子の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティは ReadOnly (Visual Basic) にすることはできません。**+=** 演算子は、右側にある値を、左側にある変数またはプロパティに代入します。

コンパイル環境に厳密な型指定規則が設定されている場合、この代入演算子は、縮小変換ではなく、暗黙的に拡大変換を実行します。これらの変換の詳細については、「拡大変換と縮小変換」を参照してください。厳密な型指定規則と寛容な型指定規則の詳細については、「Option Strict ステートメント」を参照してください。

寛容な型指定規則が適用される場合、**+=** 演算子は、**+** 演算子と同様に、文字列と数値の暗黙の変換を実行します。各変換の詳細については、「+ 演算子 (Visual Basic)」を参照してください。

メモ：

+= 演算子を使用すると、加算と文字列連結のどちらが行われるのか、事前にはわかりにくい場合があります。連結に **&=** 演算子を使用することで、あいまいさがなくなり、プログラムの可読性が向上します。

オーバーロード

+ 演算子は オーバーロード できます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。**+** 演算子をオーバーロードすると、**+=** 演算子の動作に影響します。**+** をオーバーロードしているクラスまたは構造体で **+=** を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次の例では、**+=** 演算子を使って 2 つの変数の値を結合します。前半では、**+=** を使って最初の数値変数の値を 2 番目の数値変数の値に加算します。後半では、**+=** を使って最初の **String** 変数の値を 2 番目の **String** 変数の値に連結します。どちらの場合も、結果は最初の変数に代入されます。

VB

```
' This part uses numeric variables.
Dim num1 As Integer = 10
Dim num2 As Integer = 3
num1 += num2
```

VB

```
' This part uses string variables.
Dim str1 As String = "10"
Dim str2 As String = "3"
str1 += str2
```

`num1` の値は 13 になり、`str1` の値は "103" になります。

参照
関連項目

+ 演算子 (Visual Basic)
代入演算子
算術演算子 (Visual Basic)
連結演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧
概念
代入ステートメント

= 演算子 (Visual Basic)

値を変数またはプロパティに代入します。

```
variableorproperty = value
```

指定項目
variableorproperty

任意の書き込み可能な変数または任意のプロパティを指定します。

value

任意のリテラル、定数、数式のいずれかを指定します。

解説

等号 (=) の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティを ReadOnly (Visual Basic) にすることはできません。= 演算子は、右側の値を左側の変数またはプロパティに代入します。

メモ：

= 演算子は、比較演算子としても使用されます。詳細については、「比較演算子 (Visual Basic)」を参照してください。

オーバーロード

= 演算子は比較演算子としてのみオーバーロードできます。代入演算子としてはオーバーロードできません。詳細については、「演算子プロシージャ」を参照してください。

使用例

代入演算子の例を次に示します。右側の値が左側の変数に代入されます。

VB

```
Dim testInt As Integer
Dim testString As String
Dim testButton As System.Windows.Forms.Button
Dim testObject As Object
testInt = 42
testString = "This is an example of a string literal."
testButton = New System.Windows.Forms.Button()
testObject = testInt
testObject = testString
testObject = testButton
```

参照
関連項目
&= 演算子 (Visual Basic)
*= 演算子 (Visual Basic)
+= 演算子 (Visual Basic)
-= 演算子 (Visual Basic)
/= 演算子 (Visual Basic)
\= 演算子
^= 演算子 (Visual Basic)
比較演算子 (Visual Basic)
ReadOnly (Visual Basic)
概念
代入ステートメント

- 演算子 (Visual Basic)

2 つの数式の差、または、数式の負の値を返します。

```
expression1 - expression2
- or -
- expression1
```

指定項目

expression1

必ず指定します。任意の数式を指定します。

expression2

– 演算子が負の値を計算している場合を除き、必ず指定します。任意の数式を指定します。

結果

結果は、*expression1* と *expression2* の差、または *expression1* を否定した値です。

結果のデータ型は、*expression1* と *expression2* のデータ型に適した数値型です。「演算子の結果のデータ型」の「整数演算」の表を参照してください。

サポートされている型

符号なし、浮動小数点、**Decimal** を含むすべての数値型。

解説

この構文の最初の使用例では、– 演算子は、2 つの数式の差を求める二項の減算演算子として使用されています。

2 つ目の使用例では、– 演算子は、式の負の値を求める単項の否定演算子として使用されています。この場合、否定は *expression1* の符号を反転させるので、*expression1* が負の場合、結果は正になります。

いずれかの式が Nothing と評価される場合、– 演算子はこれを 0 と見なします。

メモ :

– 演算子は オーバーロード できます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。このようなクラスまたは構造体でこの演算子を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次の例では、– 演算子を使用して 2 つの数値の差を求め、結果を返し、その後、数値を否定します。

VB

```
Dim binaryResult As Double = 459.35 - 334.9
Dim unaryResult As Double = -334.9
```

これらのステートメントを実行した後では、`binaryResult` には 124.45、`unaryResult` には –334.90 が含まれます。

参照

関連項目

-= 演算子 (Visual Basic)
算術演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

Visual Basic における算術演算子

-= 演算子 (Visual Basic)

変数またはプロパティの値から式で指定された値を減算し、その結果を変数またはプロパティに代入します。

```
variableorproperty -= expression
```

指定項目

variableorproperty

必ず指定します。変数またはプロパティを数値で指定します。

expression

必ず指定します。任意の数式を指定します。

解説

-= 演算子の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティを ReadOnly (Visual Basic) にすることはできません。**-=** 演算子は、右側の値を左側の変数またはプロパティに代入します。

オーバーロード

- 演算子 (Visual Basic) はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。- 演算子のオーバーロードは、**-=** 演算子の動作に影響を与えます。コード内で、- をオーバーロードするクラスや構造体で **-=** が使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次の例では、**-=** 演算子を使って、最初の整数型 (**Integer**) の変数から 2 番目の整数型 (**Integer**) の変数を減算し、結果を 1 番目の変数に代入します。

VB

```
Dim var1 As Integer = 10
Dim var2 As Integer = 3
var1 -= var2
' The value of var1 is now 7.
```

参照

関連項目

- 演算子 (Visual Basic)
代入演算子
算術演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

代入ステートメント

<< 演算子 (Visual Basic)

ビット パターンに対して算術左シフトを実行します。

```
result = pattern << amount
```

指定項目

result

必ず指定します。整数値を指定します。ビット パターンをシフトした結果が格納されます。データ型は、*pattern* の型と同じになります。

pattern

必ず指定します。整数の式を指定します。シフトされるビット パターンです。データ型は整数型 (**SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、**ULong**) である必要があります。

amount

必ず指定します。数式を指定します。ビット パターンは、このビット数だけシフトされます。データ型は、整数型 (**Integer**) であるか、整数型 (**Integer**) に拡大変換する必要があります。

解説

数値のシフトは、循環的には行われません。つまり、一方の端からはみ出したビットが、もう一方の端に補われることはありません。左シフトの算術演算では、結果のデータ型の範囲を超えてシフトされるビットは破棄され、右側に空いたビット位置はゼロに設定されます。

結果に保持できるビット数を超えるシフトを回避するために、Visual Basic は *pattern* のデータ型に対応するサイズ マスクを使用して、*amount* の値をマスクします。シフト量には、これらの値のバイナリの AND が使用されます。サイズ マスクは次のとおりです。

***pattern* のデータ型**	**サイズ マスク (10 進数)**	**サイズ マスク (16 進数)**
SByte, **Byte**	7	&H00000007
Short, **UShort**	15	&H0000000F
Integer, **UInteger**	31	&H0000001F
Long, **ULong**	63	&H0000003F

amount が 0 の場合、*result* の値は *pattern* の値と同じです。*amount* が負の値である場合は、符号なしの値と解釈され、適切なサイズ マスクでマスクされます。

数値のシフトではオーバーフロー例外は発生しません。

メモ :

<< 演算子は オーバーロード できます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。このようなクラスまたは構造体でこの演算子を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

<< 演算子を使用して整数値の算術左シフトを実行する例を次に示します。結果のデータ型は、シフトされる前の式と常に同じ型になります。

VB

```
Dim pattern As Short = 192
' The bit pattern is 0000 0000 1100 0000.
Dim result1, result2, result3, result4, result5 As Short
result1 = pattern << 0
result2 = pattern << 4
result3 = pattern << 9
result4 = pattern << 17
result5 = pattern << -1
```

この例の結果は、次のようになります。

- `result1` は 192 (0000 0000 1100 0000) です。
- `result2` は 3072 (0000 1100 0000 0000) です。
- `result3` は -32768 (1000 0000 0000 0000) です。
- `result4` は 384 (0000 0001 1000 0000) です。
- `result5` は 0 (15 桁左にシフト) です。

`result4` のシフト量は 17 AND 15 として計算され、1 になります。

参照

関連項目

ビット シフト演算子
代入演算子
<<= 演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

Visual Basic における算術演算子

<<= 演算子 (Visual Basic)

変数またはプロパティの値に左シフトの算術演算を実行し、その結果を元の変数またはプロパティに代入します。

```
variableorproperty <<= amount
```

指定項目

variableorproperty

必ず指定します。整数型 (**SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long** または **ULong**) の変数またはプロパティです。

amount

必ず指定します。整数型 (**Integer**) に拡大変換されるデータ型の数値表現です。

解説

<<= 演算子の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティは ReadOnly (Visual Basic) にすることはできません。**<<=** 演算子は、右側にある値を、左側にある変数またはプロパティに代入します。

数値のシフトは、循環的には行われません。つまり、一方の端からはみ出したビットが、もう一方の端に補われることはありません。左シフトの算術演算では、結果のデータ型の範囲を超えてシフトされるビットは破棄され、右側に空いたビット位置はゼロに設定されます。

オーバーロード

<< 演算子 (Visual Basic) は オーバーロード できます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。**<<** 演算子をオーバーロードすると、**<<=** 演算子の動作に影響します。**<<** をオーバーロードしているクラスまたは構造体で **<<=** を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

<<= 演算子を使用して、整数型 (**Integer**) 変数のビット パターンを、指定されたビット数だけ左にシフトし、結果を元の変数に代入する例を、次に示します。

VB

```
Dim var As Integer = 10
Dim shift As Integer = 3
var <<= shift
' The value of var is now 80.
```

参照

関連項目

<< 演算子 (Visual Basic)
代入演算子
ビット シフト演算子
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

代入ステートメント

>> 演算子 (Visual Basic)

ビット パターンに対して算術右シフトを実行します。

```
result = pattern >> amount
```

指定項目

result

必ず指定します。整数値を指定します。ビット パターンをシフトした結果が格納されます。データ型は、*pattern* の型と同じになります。

pattern

必ず指定します。整数の式を指定します。シフトされるビット パターンです。データ型は整数型 (**SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、または **ULong**) であることが必要です。

amount

必ず指定します。数式を指定します。ビット パターンは、このビット数だけシフトされます。データ型は、整数型 (**Integer**) であるか、整数型 (**Integer**) に拡大変換する必要があります。

解説

数値のシフトは、循環的には行われません。つまり、一方の端からはみ出したビットが、もう一方の端に補われることはありません。右シフトの算術演算では、右端のビット位置を超えてシフトされるビットは破棄され、左端 (符号) のビットは左側に空いたビット位置に移されます。これは、*pattern* が負の値である場合、空いた位置に 1 が設定されることを示します。それ以外の場合は、ゼロが設定されます。

Byte、**UShort**、**UInteger**、**ULong** のデータ型には符号がないため、シフトされる符号ビットがないことに注意してください。*pattern* が符号なしの型である場合、空いた位置には常にゼロが設定されます。

結果に保持できるビット数を超えるシフトを回避するために、Visual Basic は *pattern* のデータ型に対応するサイズ マスクを使用して、*amount* の値をマスクします。シフト量には、これらの値のバイナリの AND が使用されます。サイズ マスクは次のとおりです。

***pattern* のデータ型になります。**	**サイズ マスク (10 進数)**	**サイズ マスク (16 進数)**
SByte, **Byte**	7	&H00000007
Short, **UShort**	15	&H0000000F
Integer, **UInteger**	31	&H0000001F
Long, **ULong**	63	&H0000003F

amount がゼロの場合、*result* の値は *pattern* の値と同じになります。*amount* が負の値である場合は、符号なしの値と解釈され、適切なサイズ マスクでマスクされます。

数値のシフトではオーバーフロー例外は発生しません。

オーバーロード

>> 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。この演算子がコード内でこのようなクラスや構造体に対して使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

>> 演算子を使用して整数値の算術右シフトを実行する例を次に示します。結果のデータ型は、シフトされる前の式と常に同じ型になります。

VB

```
Dim pattern As Short = 2560
' The bit pattern is 0000 1010 0000 0000.
Dim result1, result2, result3, result4, result5 As Short
result1 = pattern >> 0
result2 = pattern >> 4
```

```
result3 = pattern >> 10
result4 = pattern >> 18
result5 = pattern >> -1
```

この例の結果は次のようになります。

- `result1` は 2560 (0000 1010 0000 0000)。
- `result2` は 160 (0000 0000 1010 0000)。
- `result3` は 2 (0000 0000 0000 0010)。
- `result4` は 640 (0000 0010 1000 0000)。
- `result5` は 0 (右へ 15 回シフト)。

`result4` のシフト量は 18 AND 15 として計算され、2 になります。

負の値の算術シフトを実行する例を次に示します。

VB

```
Dim negPattern As Short = -8192
' The bit pattern is 1110 0000 0000 0000.
Dim negResult1, negResult2 As Short
negResult1 = negPattern >> 4
negResult2 = negPattern >> 13
```

この例の結果は次のようになります。

- `negresult1` は -512 (1111 1110 0000 0000)。
- `negresult2` は -1 (符号ビットがシフトされている)。

参照

関連項目

ビット シフト演算子
代入演算子
>>= 演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

Visual Basic における算術演算子

>>= 演算子 (Visual Basic)

変数またはプロパティの値に右シフトの算術演算を実行し、その結果を元の変数またはプロパティに代入します。

```
variableorproperty >>= amount
```

指定項目

variableorproperty

必ず指定します。変数またはプロパティは、整数型 (**SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、または **ULong**) であることが必要です。

amount

必ず指定します。整数型 (**Integer**) に拡大変換されるデータ型の数値表現です。

解説

>>= 演算子の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティを ReadOnly (Visual Basic) にすることはできません。**>>=** 演算子は、右側の値を左側の変数またはプロパティに代入します。

数値のシフトは、循環的には行われません。つまり、一方の端からはみ出したビットが、もう一方の端に補われることはありません。右シフトの算術演算では、右端のビット位置を超えてシフトされるビットは破棄され、左端のビットは左側に空いたビット位置に移されます。これは、*variableorproperty* が負の値である場合、空いた位置に 1 が設定されることを示します。*variableorproperty* が正の値である場合、またはデータ型が unsigned 型である場合、空いた位置にはゼロが設定されます。

オーバーロード

>> 演算子 (Visual Basic) はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。**>>** 演算子のオーバーロードは、**>>=** 演算子の動作に影響を与えます。コード内で、**>>** をオーバーロードするクラスや構造体で **>>=** が使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

>>= 演算子を使って、整数型 (**Integer**) 変数のビット パターンを、指定されたビット数だけ右にシフトし、結果を元の変数に代入する例を次に示します。

VB

```
Dim var As Integer = 10
Dim shift As Integer = 2
var >>= shift
' The value of var is now 2 (one bit was lost off the right end).
```

参照

関連項目

>> 演算子 (Visual Basic)
代入演算子
ビット シフト演算子
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

代入ステートメント

/ 演算子 (Visual Basic)

2 つの数値の商を計算し、結果を浮動小数点で返します。

```
expression1 / expression2
```

指定項目

expression1

必ず指定します。任意の数式を指定します。

expression2

必ず指定します。任意の数式を指定します。

サポートされている型

unsigned 型と浮動小数点型を含むすべての数値型、および 10 進型 (**Decimal**)。

結果

結果は *expression1* を *expression2* で割った、剰余を含む完全な商です。

The \ 演算子 は整数の商を返します。剰余は破棄します。

解説

結果のデータ型は、オペランドの型によって決まります。オペランドの型と結果のデータ型の関係を次の表に示します。

オペランドのデータ型	結果のデータ型
両方の式が整数型 (SByte、Byte、Short、UShort、Integer、UInteger、Long、ULong) の場合	**Double**
両方の式が Decimal データ型の場合	**Decimal**
両方の式が 単精度浮動小数点型 (Single) (Visual Basic) の場合	**Single**
どちらかの式が浮動小数点型 (単精度浮動小数点型 (Single) (Visual Basic) または 倍精度浮動小数点数型 (Double) (Visual Basic)) の場合	**Double**

除算を実行する前に、整数の数式は必ず **Double** 型に拡大変換されます。結果を整数型に代入する場合は、結果が **Double** からその型に変換されます。このとき、結果がその型に合わなければ、例外がスローされます。詳細は、このヘルプ ページの「0 による除算」を参照してください。

expression1 または *expression2* が Nothing に評価される場合は、式が 0 として扱われます。

0 による除算

expression2 が 0 に評価される場合、**/** 演算子はオペランドのデータ型の違いに応じて異なる動作をします。次の表に、それぞれの動作を示します。

オペランドのデータ型	***expression2* が 0 の場合の動作**
浮動小数点型 (**Single** または **Double**)	無限大 (PositiveInfinity または NegativeInfinity) を返すか、*expression1* も 0 であれば NaN (非数値) を返します
Decimal	DivideByZeroException をスローします
整数 (符号付きまたは符号なし)	整数型に変換して戻そうとすると、整数型が **PositiveInfinity**、**NegativeInfinity**、または **NaN** を処理できないため OverflowException がスローされます。

メモ :

/ 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。この演算子がコード内でこのようなクラスや構造体に対して使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

/ 演算子を使って浮動小数点の除算を実行する例を次に示します。結果は、2 つのオペランドの商です。

VB

```
Dim resultValue As Double
resultValue = 10 / 4
resultValue = 10 / 3
```

この例の式は、2.5 と 3.333333 の値を返します。たとえ両方のオペランドが整数定数であっても、結果が必ず浮動小数点 (**Double**) になっていることに注意してください。

参照

関連項目

/= 演算子 (Visual Basic)
\ 演算子
演算子の結果のデータ型
算術演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

Visual Basic における算術演算子

/= 演算子 (Visual Basic)

変数またはプロパティの値を式で指定された値で除算し、その浮動小数点の結果を変数またはプロパティに代入します。

```
variableorproperty /= expression
```

指定項目

variableorproperty

必ず指定します。任意の数値変数またはプロパティです。

expression

必ず指定します。任意の数式を指定します。

解説

/= 演算子の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティは ReadOnly (Visual Basic) にすることはできません。**/=** 演算子は、右側にある値を、左側にある変数またはプロパティに代入します。

このステートメントは、倍精度浮動小数点数型 (**Double**) の値を左側の変数またはプロパティに代入します。**Option Strict** が **On** の場合、*variableorproperty* は **Double** である必要があります。**Option Strict** が **Off** の場合、暗黙の型変換が行われ、その結果の値を、実行時にエラーがあればそのエラーと共に *variableorproperty* に割り当てます。詳細については、「拡大変換と縮小変換」および「Option Strict ステートメント」を参照してください。

オーバーロード

/ 演算子 (Visual Basic) は オーバーロード できます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。**/** 演算子をオーバーロードすると、**/=** 演算子の動作に影響します。**/** をオーバーロードしているクラスまたは構造体で **/=** を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次の例では、**/=** 演算子を使って、最初の整数型 (**Integer**) の変数を 2 番目の整数型 (**Integer**) の変数で除算し、商を最初の変数に代入します。

VB

```
Dim var1 As Integer = 12
Dim var2 As Integer = 3
var1 /= var2
' The value of var1 is now 4.
```

参照

関連項目

/ 演算子 (Visual Basic)
\= 演算子
代入演算子
算術演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

代入ステートメント

\ 演算子

2 つの数値の商を計算し、結果を整数で返します。

```
expression1 \ expression2
```

指定項目

expression1

必ず指定します。任意の数式を指定します。

expression2

必ず指定します。任意の数式を指定します。

サポートされている型

unsigned 型と浮動小数点型を含むすべての数値型、および 10 進型 (**Decimal**)。

結果

結果は *expression1* を *expression2* で割った整数の商です。余りはすべて破棄し、整数部分だけが保持されます。これを切り捨てと呼びます。

結果のデータ型は *expression1* と *expression2* のデータ型に対して適切な数値型になります。「演算子の結果のデータ型」に示す「整数演算」の表を参照してください。

/ 演算子 (Visual Basic) は、余りを小数部分に保持する完全な商を返します。

解説

Visual Basic では、除算を実行する前に、すべての浮動小数点型の数式が長整数型 (**Long**) の式に変換されます。**Option Strict** が **On** であれば、コンパイル エラーが発生します。**Option Strict** が **Off** の場合でも、値が Long データ型 (Visual Basic) の範囲を超えていれば OverflowException が発生します。長整数型 (**Long**) への変換でも、銀行型丸めが行われます。詳細については、「データ型変換関数」の「小数部分」を参照してください。

expression1 または *expression2* が Nothing に評価される場合は、式が 0 として扱われます。

0 による除算

expression2 が 0 に評価される場合、**** 演算子は DivideByZeroException 例外をスローします。これは、オペランドのデータ型がどの数値型であっても同じです。

メモ：

**** 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。この演算子がコード内でこのようなクラスや構造体に対して使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

**** 演算子を使って整数の除算を実行する例を次に示します。結果は、2 つのオペランドの商の整数部分を表す整数です。余りは破棄されます。

VB

```
Dim resultValue As Integer
resultValue = 11 \ 4
resultValue = 9 \ 3
resultValue = 100 \ 3
resultValue = 67 \ -3
```

この例の式は、2、3、33、および -22 の値をそれぞれ返します。

参照

関連項目

\= 演算子
/ 演算子 (Visual Basic)
Option Strict ステートメント
算術演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧
概念
Visual Basic における算術演算子

\= 演算子

変数またはプロパティの値を式の値で除算し、その結果の整数値を変数またはプロパティに代入します。

```
variableorproperty \= expression
```

指定項目

variableorproperty

必ず指定します。変数またはプロパティを数値で指定します。

expression

必ず指定します。任意の数式を指定します。

解説

\= 演算子の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティを ReadOnly (Visual Basic) にすることはできません。**\=** 演算子は、右側の値を左側の変数またはプロパティに代入します。

整数除算の詳細については、「\ 演算子」を参照してください。

オーバーロード

**** 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。**** 演算子のオーバーロードは、**\=** 演算子の動作に影響を与えます。コード内で、**** をオーバーロードするクラスや構造体で **\=** が使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次の例では、**\=** 演算子を使って、最初の整数型 (**Integer**) の変数を 2 番目の整数型の変数で除算し、整数の結果を最初の変数に代入します。

VB

```
Dim var1 As Integer = 10
Dim var2 As Integer = 3
var1 \= var2
' The value of var1 is now 3.
```

参照

関連項目

\ 演算子
/= 演算子 (Visual Basic)
代入演算子
算術演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

代入ステートメント

^ 演算子 (Visual Basic)

数値のべき乗を求めます。

```
number ^ exponent
```

指定項目

number

必ず指定します。任意の数式を指定します。

exponent

必ず指定します。任意の数式を指定します。

結果

結果は、*number* を *exponent* で累乗したもので、常に **Double** 値です。

サポートされている型

Double です。他の型のオペランドはすべて **Double** に変換されます。

解説

Visual Basic は、すべての指数演算を 倍精度浮動小数点数型 (Double) (Visual Basic) で行います。

exponent の値は、小数、負の値、またはその両方です。

1 つの式の中で複数の指数演算が行われるとき、^ 演算子は左から右の順に評価されます。

メモ：

^ 演算子は オーバーロード できます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。このようなクラスまたは構造体でこの演算子を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

^ 演算子を使って数値のべき乗を求める例を次に示します。結果は、最初のオペランドを第 2 のオペランドで累乗した数値です。

VB

```
Dim exp1, exp2, exp3, exp4, exp5, exp6 As Double
exp1 = 2 ^ 2
exp2 = 3 ^ 3 ^ 3
exp3 = (-5) ^ 3
exp4 = (-5) ^ 4
exp5 = 8 ^ (1.0 / 3.0)
exp6 = 8 ^ (-1.0 / 3.0)
```

この例の結果は、次のようになります。

`exp1` が 4 (2 の 2 乗) に設定されます。

`exp2` が 19683 (3 の 3 乗を 3 乗) に設定されます。

`exp3` が -125 (-5 の 2 乗) に設定されます。

`exp4` が 625 (-5 の 4 乗) に設定されます。

`exp5` が 2 (8 の平方根) に設定されます。

`exp6` が 0.5 (1.0 を 8 の平方根で除算した商) に設定されます。

この例ではかっこが重要な役割を果たしていることに注目してください。演算子の優先順位のために、Visual Basic では通常 ^ 演算子を、単項 – 演算子よりも先に (つまり最初に) 実行します。`exp4` および `exp6` をかっこなしで計算すると、次のような結果になります。

`exp4 = -5 ^ 4` は –(5 の 4 乗) となり、–625 になります。

`exp6 = 8 ^ -1.0 / 3.0` は (8 の –1 乗、0.125) を 3.0 で除算した商となり、0.041666666666666666666666666666667 になります。

参照

関連項目

^= 演算子 (Visual Basic)

算術演算子 (Visual Basic)

Visual Basic における演算子の優先順位

機能別の演算子一覧

概念

Visual Basic における算術演算子

^= 演算子 (Visual Basic)

変数またはプロパティに対する式のべき乗を求め、その結果を変数またはプロパティに戻します。

```
variableorproperty ^= expression
```

指定項目

variableorproperty

必ず指定します。変数またはプロパティを数値で指定します。

expression

必ず指定します。任意の数式を指定します。

解説

^= 演算子の左側には、スカラ変数、プロパティ、配列の要素なども指定できます。変数またはプロパティを ReadOnly (Visual Basic) にすることはできません。**^=** 演算子は、右側の値を左側の変数またはプロパティに代入します。

Visual Basic では、指数演算が必ず倍精度浮動小数点型 (倍精度浮動小数点数型 (Double) (Visual Basic)) で実行されます。これ以外の型のオペランドはすべて **Double** に変換され、結果は必ず **Double** 型になります。

expression の値には、小数や負の数 (またはその両方) を指定できます。

オーバーロード

^ 演算子 (Visual Basic) はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。**^** 演算子のオーバーロードは、**^=** 演算子の動作に影響を与えます。コード内で、**^** をオーバーロードするクラスや構造体で **^=** が使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次の例では、**^=** 演算子を使って、最初の整数型 (**Integer**) の変数の n 乗 (n = 2 番目の変数の値) を求め、結果を最初の変数に代入します。

VB

```
Dim var1 As Integer = 10
Dim var2 As Integer = 3
var1 ^= var2
' The value of var1 is now 1000.
```

参照

関連項目

^ 演算子 (Visual Basic)
代入演算子
算術演算子 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

代入ステートメント

AddressOf 演算子

特定のプロシージャを参照するプロシージャ デリゲート インスタンスを作成します。

```
AddressOf procedurename
```

指定項目
procedurename

必ず指定します。新しく作成されたプロシージャ デリゲートによって参照されるプロシージャを指定します。

解説

AddressOf 演算子により、*procedurename* で指定された関数を参照する関数デリゲートが作成されます。指定されたプロシージャがインスタンス メソッドである場合、関数デリゲートはインスタンスとメソッドの両方を参照します。その後、関数デリゲートが呼び出されると、指定されたインスタンスの指定されたメソッドが呼び出されます。

AddressOf 演算子は、デリゲート コンストラクタのオペランドとして使用できます。また、デリゲートの種類をコンパイラで決定できる状況で使用できます。

使用例

AddressOf 演算子を使って、ボタンの **Click** イベントを処理するデリゲートを指定する例を次に示します。

VB

```
' Add the following line to Sub Form1_Load().
AddHandler Button1.Click, AddressOf Button1_Click
```

AddressOf 演算子を使用して、スレッドの起動関数を指定する例を次に示します。

VB

```
Public Sub CountSheep()
    Dim i As Integer = 1 ' Sheep do not count from 0.
    Do While (True) ' Endless loop.
        Console.WriteLine("Sheep " & i & " Baah")
        i = i + 1
        System.Threading.Thread.Sleep(1000) 'Wait 1 second.
    Loop
End Sub

Sub UseThread()
    Dim t As New System.Threading.Thread(AddressOf CountSheep)
    t.Start()
End Sub
```

参照
関連項目
Declare ステートメント
Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)
概念
デリゲートと AddressOf 演算子

And 演算子 (Visual Basic)

2 つのブール (**Boolean**) 式の論理積を求めます。または、2 つの数式のビットごとの積を求めます。

```
result = expression1 And expression2
```

指定項目

result

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。ブール式の比較の場合、*result* は 2 つの **Boolean** 値の論理積になります。ビットごとの演算の場合、*result* は 2 つの数式のビットパターンの、ビットごとの論理積を表す数値になります。

expression1

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。

expression2

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。

解説

ブール値の比較の場合、*result* が **True** になるのは、*expression1* と *expression2* がどちらも **True** に評価される場合だけです。次の表は、2 つの式と *result* の値の関係を示しています。

***expression1* の値**	***expression2* の値**	***result* の値**
True	**True**	**True**
True	**False**	**False**
False	**True**	**False**
False	**False**	**False**

メモ :

ブール式の比較の場合、**And** 演算子は必ず両方の式を評価します。これは式がプロシージャ呼び出しを実行する場合も同じです。AndAlso 演算子 はショートサーキットを実行します。つまり、*expression1* が **False** であれば、*expression2* は評価されません。

数値に対して使用された場合、**And** 演算子は、2 つの数式内で同じ位置にあるビットごとに比較を行います。次の表に従って、*result* に対応するビットがセットされます。

***expression1* のビット**	***expression2* のビット**	***result* のビット**
1	1	1
1	0	0
0	1	0
0	0	0

メモ :

論理演算子やビット処理演算子は、他の算術演算子や関係演算子より優先順位が低いため、正しい結果を得るにはビットごとの演算をかっこで囲む必要があります。

データ型

オペランドの一方が **Boolean** 式で、もう一方が数式の場合、Visual Basic は **Boolean** 式を数値 (**True** は -1、**False** は 0) に変換してか

ら、ビットごとの演算を実行します。

ブール式どうしの比較の場合、結果のデータ型は **Boolean** になります。ビットごとの比較の場合、結果のデータ型は *expression1* と *expression2* のデータ型に対して適切な数値型になります。「演算子の結果のデータ型」の「リレーショナルおよびビットごとの比較」の表を参照してください。

メモ：
And 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。この演算子がコード内でこのようなクラスや構造体に対して使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

And 演算子を使って 2 つの式の論理積を求める例を次に示します。結果は、2 つの式の両方が **True** かどうかを表す **Boolean** 値です。

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck As Boolean
firstCheck = a > b And b > c
secondCheck = b > a And b > c
```

先の例では、**True**、**False** という結果がそれぞれ生成されます。

And 演算子を使って、2 つの数式のビットごとの論理積を求める例を次に示します。2 つのオペランドの対応するビットが両方とも 1 にセットされていれば、結果パターンにビットがセットされます。

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstPattern, secondPattern, thirdPattern As Integer
firstPattern = (a And b)
secondPattern = (a And c)
thirdPattern = (b And c)
```

先の例では 8、2、0 という結果がそれぞれ生成されます。

参照

関連項目

論理/ビット演算子
Visual Basic における演算子の優先順位
機能別の演算子一覧
AndAlso 演算子

概念

Visual Basic の論理演算子とビット処理演算子

AndAlso 演算子

2 つの式の論理積を簡略的に求めます。

```
result = expression1 AndAlso expression2
```

指定項目

result

必ず指定します。任意のブール型 (**Boolean**) の式を指定します。結果は、2 つの式の比較結果を表すブール値になります。

expression1

必ず指定します。任意のブール型 (**Boolean**) の式を指定します。

expression2

必ず指定します。任意のブール型 (**Boolean**) の式を指定します。

解説

コンパイルされたコードで、1 つの式の結果によってはもう 1 つの式の評価を省略できる場合、そこで使用される論理演算子をショートサーキットと呼びます。最初に評価される式の結果によって演算の最終結果が決まる場合は、もう 1 つの式によって最終結果が変わることはないため、その式を評価する必要はありません。省略される側の式が複雑な場合や、プロシージャの呼び出しを含む場合は、ショートサーキットによってパフォーマンスを向上できます。

両方の式の評価が **True** の場合、*result* は **True** になります。次の表は、2 つの式の値と演算結果 *result* の値の対応を示しています。

***expression1* の値**	***expression2* の値**	***result* の結果**
True	**True**	**True**
True	**False**	**False**
False	(評価しない)	**False**

AndAlso 演算子の利用法の 1 つは、メンバにアクセスする前に、そのオブジェクト インスタンスが存在するかどうかをテストすることです。次のコードは、これを示したものです。

```
If newObject AndAlso newObject.initFinished Then
```

このコードで `initFinished` プロパティにアクセスしようとしたとき、`newObject` 変数にオブジェクト インスタンスが割り当てられていないと、NullReferenceException 例外がスローされます。ただし、**Nothing** の評価は **False** なので、**AndAlso** 演算子によって、コンパイラは、`newObject` が **Nothing** の場合に `initFinished` へのアクセスをスキップします。

データ型

AndAlso 演算子は、ブール型 (Boolean) (Visual Basic) のみに対して定義されます。Visual Basic は、各オペランドを必要に応じて **Boolean** に変換し、完全な **Boolean** に対して演算を実行します。結果を数値型に割り当てる場合、Visual Basic は結果を **Boolean** からこのデータ型に変換します。これにより、予期しない動作が起きることがあります。たとえば、`5 AndAlso 12` の結果は、**Integer** に変換すると `-1` になります。

オーバーロード

And 演算子 (Visual Basic) と IsFalse 演算子 はオーバーロードできます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。**And** および **IsFalse** 演算子をオーバーロードすると、**AndAlso** 演算子の動作に影響します。**And** および **IsFalse** をオーバーロードしているクラスまたは構造体で **AndAlso** を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

AndAlso 演算子を使って 2 つの式の論理積を求める例を次に示します。結果は、結合された式全体が真かどうかを表すブール値です。最初の式が **False** の場合、2 番目の式は評価されません。

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck, thirdCheck As Boolean
firstCheck = a > b AndAlso b > c
secondCheck = b > a AndAlso b > c
thirdCheck = a > b AndAlso c > b
```

前の例では、**True**、**False**、**False** の結果を順に生成します。`secondCheck` の計算では、最初の式が **False** であるため、2 番目の式は評価されません。しかし、`thirdCheck` の計算では、2 番目の式が評価されます。

次の例では、配列の要素の中から指定された値を検索する **Function** プロシージャを示します。配列が空の場合、また、配列の長さが限度を超えている場合、**While** ステートメントは配列の要素を検索値に対してテストしません。

VB

```
Public Function findValue(ByVal arr() As Double, _
    ByVal searchValue As Double) As Double
    Dim i As Integer = 0
    While i <= UBound(arr) AndAlso arr(i) <> searchValue
        ' If i is greater than UBound(arr), searchValue is not checked.
        i += 1
    End While
    If i > UBound(arr) Then i = -1
    Return i
End Function
```

参照

関連項目

論理/ビット演算子
Visual Basic における演算子の優先順位
機能別の演算子一覧
And 演算子 (Visual Basic)
IsFalse 演算子

概念

Visual Basic の論理演算子とビット処理演算子

GetType 演算子

指定された型の Type オブジェクトを返します。**Type** オブジェクトにはプロパティ、メソッド、イベントなど、その型に関する情報が含まれています。

```
GetType(typename)
```

パラメータ

typename

情報を取得する型の名前を指定します。

解説

GetType 演算子は、指定された *typename* の **Type** オブジェクトを返します。*typename* には、定義済みの任意の型の名前を渡すことができます。たとえば、次のような構造です。

- **Boolean** や **Date** などの Visual Basic のデータ型
- System.ArgumentException や System.Double などの .NET Framework のクラス、構造体、モジュール、インターフェイス。
- アプリケーションで定義したクラス、構造体、モジュール、またはインターフェイス。
- アプリケーションで定義した配列。
- アプリケーションで定義したデリゲート。
- Visual Basic、.NET Framework、またはアプリケーションで定義した列挙体。

オブジェクト変数の型オブジェクトを取得する場合は、System.Type.GetType メソッドを使用します。

GetType 演算子は次のような場合に使用できます。

- 実行時に、型のメタデータにアクセスする必要がある場合。**Type** オブジェクトには、型のメンバや配置情報などのメタデータが含まれています。たとえば、アセンブリに対してリフレクションする場合などにこれが必要になります。詳細については、「System.Reflection」を参照してください。
- 2 つのオブジェクト参照を比較して、それらが同じ型のインスタンスを参照しているかを確認することがあります。同じ型のインスタンスを参照している場合、**GetType** は同じ **Type** オブジェクトへの参照を返します。

使用例

GetType 演算子の使用例を次に示します。

VB

```
' The following statement returns the Type object for Integer.
MsgBox(GetType(Integer).ToString())
' The following statement returns the Type object for one-dimensional string arrays.
MsgBox(GetType(String()).ToString())
```

参照

関連項目

Visual Basic における演算子の優先順位

機能別の演算子一覧

概念

Visual Basic の演算子および式

Is 演算子 (Visual Basic)

2 つのオブジェクト変数を比較します。

```
result = object1 Is object2
```

指定項目
result

必ず指定します。任意のブール型 (**Boolean**) の値を指定します。

object1

必ず指定します。任意のオブジェクト名を指定します。

object2

必ず指定します。任意のオブジェクト名を指定します。

解説

Is演算子は、2 つのオブジェクト参照が同じオブジェクトを参照しているかどうかを判定します。ただし、値の比較は行われません。*object1* と *object2* の両方がまったく同じオブジェクト インスタンスを参照している場合、*result* は **True** になります。それ以外の場合は、*result* は **False** です。

Is を **TypeOf** キーワードと共に使用して **TypeOf**...**Is** 式を作成し、オブジェクト変数がデータ型と互換性があるかどうかをテストできます。

使用例

次の例では、**Is** 演算子を使用して、1 組のオブジェクト参照を比較します。結果は、2 つのオブジェクトが同じかどうかを示す **Boolean** 値に割り当てられます。

VB

```
Dim myObject As New Object
Dim otherObject As New Object
Dim yourObject, thisObject, thatObject As Object
Dim myCheck As Boolean
yourObject = myObject
thisObject = myObject
thatObject = otherObject
' The following statement sets myCheck to True.
myCheck = yourObject Is thisObject
' The following statement sets myCheck to False.
myCheck = thatObject Is thisObject
' The following statement sets myCheck to False.
myCheck = myObject Is thatObject
thatObject = myObject
' The following statement sets myCheck to True.
myCheck = thisObject Is thatObject
```

この例で示すように、**Is** 演算子は、事前バインディングされたオブジェクトと遅延バインディングされたオブジェクトの両方をテストするのに使用できます。

参照
関連項目
TypeOf 演算子 (Visual Basic)
IsNot 演算子
Visual Basic における演算子の優先順位
機能別の演算子一覧
概念
Visual Basic における比較演算子
Visual Basic の演算子および式

IsFalse 演算子

式が **False** かどうかを調べます。

IsFalse をコードの中で明示的に呼び出すことはできませんが、Visual Basic コンパイラはこれを使用して **AndAlso** 句からコードを生成します。クラスまたは構造体を定義して、その型の変数を **AndAlso** 句で使用する場合は、そのクラスまたは構造体に対して **IsFalse** を定義する必要があります。

コンパイラは **IsFalse** 演算子と **IsTrue** 演算子を一致したペアと見なします。つまり、どちらか一方を定義した場合は、もう一方も定義することが必要です。

メモ：

IsFalse 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。この演算子がコード内でこのようなクラスや構造体に対して使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

IsFalse 演算子と **IsTrue** 演算子を含む構造体のアウトラインを定義する例を次に示します。

VB

```
Public Structure p
    Dim a As Double
    Public Shared Operator IsFalse(ByVal w As p) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsFalse of w.
        Return b
    End Operator
    Public Shared Operator IsTrue(ByVal w As p) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsTrue of w.
        Return b
    End Operator
End Structure
```

参照

処理手順

方法 : 演算子を定義する

関連項目

IsTrue 演算子

AndAlso 演算子

IsNot 演算子

2 つのオブジェクト変数を比較します。

```
result = object1 IsNot object2
```

指定項目
result

必ず指定します。**Boolean** 値。

object1

必ず指定します。任意の **Object** 変数または式です。

object2

必ず指定します。任意の **Object** 変数または式です。

解説

IsNot演算子は、2 つのオブジェクト参照が別のオブジェクトを参照しているかどうかを判定します。ただし、値の比較は行われません。*object1* と *object2* の両方がまったく同じオブジェクト インスタンスを参照している場合、*result* は **False** になります。それ以外の場合は、*result* は **True** です。

IsNot は **Is** 演算子の逆の働きをします。**IsNot** の利点は、**Not** と **Is** を組み合わせて記述し、コードが読みづらくなるのを防げることです。

Is と **IsNot** 演算子を使用して、事前バインディング オブジェクトと遅延バインディング オブジェクトの両方をテストできます。

使用例

次のコード例では、**Is** 演算子と **IsNot** 演算子の両方を使用して、同じ比較を実行します。

VB

```
Dim o1, o2 As New Object
If Not o1 Is o2 Then MsgBox("o1 and o2 do not refer to the same instance.")
If o1 IsNot o2 Then MsgBox("o1 and o2 do not refer to the same instance.")
```

参照
処理手順
方法 : 2 つのオブジェクトが等しいかどうかをテストする
関連項目
Is 演算子 (Visual Basic)
Visual Basic における演算子の優先順位

IsTrue 演算子

式が **True** かどうかを調べます。

IsTrue をコードから明示的に呼び出すことはできません。Visual Basic コンパイラが、**OrElse** 句からコードを生成するのに使用します。クラスまたは構造体を定義し、**OrElse** 句でその型の変数を使用する場合、そのクラスまたは構造体で **IsTrue** を定義する必要があります。

コンパイラは **IsTrue** および **IsFalse** 演算子を 一致したペアと見なします。つまり、一方を定義したら、もう一方も定義する必要があります。

IsTrue のコンパイラによる使用

クラスまたは構造体を定義した場合、その型の変数を **For**、**If**、**Else If**、または **While** ステートメントまたは **When** 句で使用できます。この場合、コンパイラは条件をテストするために、この型を **Boolean** 値に変換する演算子が必要です。適切な演算子は次の順序で検索されます。

1. クラスまたは構造体から **Boolean** への拡大変換演算子
2. クラスまたは構造体上の **IsTrue** 演算子
3. クラスまたは構造体から **Boolean** への縮小変換演算子

Boolean への変換、または **IsTrue** 演算子を定義していない場合、コンパイラはエラーを返します。

メモ：

IsTrue は オーバーロード できます。つまり、オペランドがクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。このようなクラスまたは構造体でこの演算子を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次のコード例では、**IsFalse** および **IsTrue** 演算子の定義を含む構造体の骨組みを定義します。

VB

```
Public Structure p
    Dim a As Double
    Public Shared Operator IsFalse(ByVal w As p) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsFalse of w.
        Return b
    End Operator
    Public Shared Operator IsTrue(ByVal w As p) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsTrue of w.
        Return b
    End Operator
End Structure
```

参照

処理手順

方法 : 演算子を定義する

関連項目

IsFalse 演算子

OrElse 演算子

Like 演算子

文字列をパターンと比較します。

```
result = string Like pattern
```

指定項目

result

必ず指定します。任意のブール型 (**Boolean**) の変数を指定します。結果は、引数 *string* が引数 *pattern* に一致するかどうかを示す **Boolean** 値です。

string

必ず指定します。任意のブール型 (**String**) の式を指定します。

pattern

必ず指定します。パターン一致規則に適合させる任意の文字列 (**String**) 式を指定します。規則については「解説」で説明します。

解説

string の値が、*pattern* に含まれるパターンに一致する場合、*result* は **True** です。文字列がパターンに一致しない場合、*result* は **False** です。*string* と *pattern* の両方が空白文字列の場合、結果は **True** になります。

比較メソッド

Like 演算子の動作は、Option Compare ステートメント によって決まります。各ソース ファイルの既定の文字列比較メソッドは **Option Compare Binary** です。

パターン オプション

組み込みのパターン一致は、さまざまな文字列比較に利用できます。パターン一致機能では、*string* 内の各文字を特定の文字、ワイルドカード文字、文字のリスト、文字の範囲と比較できます。*pattern* に使用できる文字と、それに一致する文字を次の表に示します。

***pattern* 内の文字**	***string* の一致**
?	任意の 1 文字
*	0 個以上の文字
#	任意の 1 桁 (0–9)
[*charlist* **]**	*charlist* に含まれる任意の 1 文字
[! *charlist* **]**	*charlist* に含まれない任意の 1 文字

文字のリスト

1 文字以上のグループ (*charlist*) を角かっこ (**[ ]**) で囲むことで、*string* に含まれる任意の 1 文字に一致する文字を検索できます。このグループには、数字など、任意の文字コードを含めることができます。

charlist の先頭に感嘆符 (**!**) を付けると、*charlist* に含まれる文字以外の任意の文字が *string* 内にあるかどうかを検索することになります。角かっこの外側に感嘆符がある場合は、感嘆符自体を検索します。

特殊文字

特殊文字の左角かっこ (**[**)、疑問符 (**?**)、シャープ記号 (**#**)、およびアスタリスク (*) に一致させるには、これらの文字を角かっこで囲む必要があります。右角かっこ (**]**) をグループ内に使用して、その文字自体を比較することはできませんが、グループの外で独立した文字として使用することはできます。

[ ] という文字列は、長さ 0 の文字列 (**""**) と見なされます。しかし、角かっこで囲まれた文字リストに含めることはできません。*string* 内の位置に、いずれかの文字グループが含まれているか、それとも文字が含まれていないかをチェックするには、**Like** を 2 回使用します。カスタマイズ例については、「方法 : 文字列がパターンに一致するかを調べる」を参照してください。

文字の範囲

ハイフン (–) を使って範囲の上限と下限を設定することで、*charlist* に文字の範囲を指定できます。たとえば、`[A-Z]` と指定すると、*string* 内の文字の位置に `A` から `Z` までの文字のいずれかが含まれている場合に一致となります。`[!H-L]` と指定すると、対応する文字の位置に `H` から `L` までの範囲に含まれない文字がある場合に一致となります。

文字の範囲を指定する場合、文字は昇順で指定する必要があります。つまり、`[A-Z]` は有効なパターンですが、`[Z-A]` は無効なパターンです

複数の文字範囲

同じ文字位置に対して複数の範囲を指定するには、同じ角かっこの中に、デリミタなしで範囲を指定します。たとえば、`[A-CX-Z]` と指定すると、*string* 内の対応する文字位置に `A` から `C` まで、および `X` から `Z` までの文字が含まれているときに一致となります。

ハイフンの使用

ハイフン (– が *charlist* の先頭 (感嘆符がある場合はその直後) または末尾にある場合は、ハイフン自体を検索します。他の場所にハイフンがある場合は、ハイフンの両側の文字を上限および下限とする、文字の範囲を表します。

照合シーケンス

指定した範囲の意味は、実行時に有効な文字順序により異なります。つまり、**Option Compare** と、コードが実行されるシステムのロケール設定で決まります。**Option Compare Binary** では、`[A-E]` という範囲は `A`、`B`、`C`、`D`、および `E` に一致します。**Option Compare Text** では、`[A-E]` は `A`、`a`、`À`、`à`、`B`、`b`、`C`、`c`、`D`、`d`、`E`、および `e` に一致します。アクセント付き文字の並べ替え順序はアクセントなしの文字よりも後なので、この範囲は `Ê` および `ê` には一致しません。

Digraph 文字

言語によっては、独立した 2 文字を表す文字がアルファベットに含まれています。たとえば、いくつかの言語では `æ` という文字が使用されます。これは、`a` と `e` が連続する場合に、その 2 文字を表す文字です。**Like** 演算子では、この digraph 文字 1 文字と独立した 2 文字が同じものとして認識されます。

digraph 文字を使用する言語がシステム ロケール設定に指定されている場合、*pattern* または *string* のどちらかにその digraph 文字 1 文字が含まれると、他方の文字列では、同じ意味を持つ連続した 2 文字を検索します。同様に、角かっこ内の *pattern* に digraph 文字 1 文字が含まれると、*string* 内では同じ意味を持つ連続した 2 文字を検索します。

オーバーロード

Like 演算子は オーバーロード できます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。このようなクラスまたは構造体でこの演算子を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

Like 演算子を使って、文字列をさまざまなパターンと比較する例を次に示します。結果は、各文字列がパターンに一致したかどうかを示す **Boolean** 変数で表されます。

VB

```
Dim testCheck As Boolean
' The following statement returns True (does "F" satisfy "F"?)
testCheck = "F" Like "F"
' The following statement returns False for Option Compare Binary
'    and True for Option Compare Text (does "F" satisfy "f"?)
testCheck = "F" Like "f"
' The following statement returns False (does "F" satisfy "FFF"?)
testCheck = "F" Like "FFF"
' The following statement returns True (does "aBBBa" have an "a" at the
'    beginning, an "a" at the end, and any number of characters in
'    between?)
testCheck = "aBBBa" Like "a*a"
' The following statement returns True (does "F" occur in the set of
'    characters from "A" through "Z"?)
testCheck = "F" Like "[A-Z]"
' The following statement returns False (does "F" NOT occur in the
'    set of characters from "A" through "Z"?)
testCheck = "F" Like "[!A-Z]"
' The following statement returns True (does "a2a" begin and end with
'    an "a" and have any single-digit number in between?)
testCheck = "a2a" Like "a#a"
' The following statement returns True (does "aM5b" begin with an "a",
'    followed by any character from the set "L" through "P", followed
'    by any single-digit number, and end with any character NOT in
'    the character set "c" through "e"?)
```

```
testCheck = "aM5b" Like "a[L-P]#[!c-e]"
' The following statement returns True (does "BAT123khg" begin with a
'    "B", followed by any single character, followed by a "T", and end
'    with zero or more characters of any type?)
testCheck = "BAT123khg" Like "B?T*"
' The following statement returns False (does "CAT123khg"?) begin with
'    a "B", followed by any single character, followed by a "T", and
'    end with zero or more characters of any type?)
testCheck = "CAT123khg" Like "B?T*"
```

参照

処理手順

方法 : 文字列がパターンに一致するかを調べる

関連項目

比較演算子 (Visual Basic)

InStr 関数 (Visual Basic)

Visual Basic における演算子の優先順位

機能別の演算子一覧

Option Compare ステートメント

StrComp 関数 (Visual Basic)

概念

Visual Basic の演算子および式

Mod 演算子 (Visual Basic)

2 つの数値の除算を行い、その剰余を返します。

```
number1 Mod number2
```

指定項目
number1

必ず指定します。任意の数式を指定します。

number2

必ず指定します。任意の数式を指定します。

サポートされている型

unsigned 型と浮動小数点型を含むすべての数値型、および 10 進型 (**Decimal**)。

結果

結果は *number1* を *number2* で割った後の剰余になります。たとえば、"`14 Mod 4`" という式は 2 に評価されます。

解説

number1 または *number2* が浮動小数点値であった場合、浮動小数点の剰余が返されます。結果のデータ型は、*number1* と *number2* のデータ型で除算を行った結果として想定される、あらゆる値の格納が可能なデータ型の中で、最も小さい型になります。

number1 または *number2* が Nothing に評価される場合は、式が 0 として扱われます。

関連する演算子としては、次のものが挙げられます。

- \ 演算子 は、除算の結果として得られる整数の商を返します。たとえば、"`14 \ 4`" という式は 3 に評価されます。
- / 演算子 (Visual Basic) は、剰余を含む完全な商を、浮動小数点数として返します。たとえば、"`14 / 4`" という式は 3.5 に評価されます。

0 による除算

number2 が 0 である場合、**Mod** 演算子の動作はオペランドのデータ型に応じて異なります。整数を 0 で除算すると DivideByZeroException 例外がスローされます。浮動小数点数を 0 で除算すると、NaN が返されます。

等価な数式

次の数式は、いずれも "`a Mod b`" と同じ意味になります。

```
a - (b * (a \ b))
```

```
a - (b * Fix(a / b))
```

浮動小数点の誤差

浮動小数点数を操作する場合は、メモリ内に必ずしも正確に表現されるわけではないので注意してください。このため、値の比較や **Mod** 演算子など、特定の処理で予期しない結果が返される可能性があります。詳細については、「データ型のトラブルシューティング」を参照してください。

オーバーロード

Mod 演算子の動作は、クラスまたは構造体で再定義できます。これを演算子のオーバーロードといいます。このようなオーバーロードを含むクラスまたは構造体のインスタンスに **Mod** を適用する場合は、再定義された動作を十分に理解しておく必要があります。詳細については、「演算子プロシージャ」を参照してください。

使用例

Mod 演算子を使って、2 つの数値の除算を行い、その剰余を取得する例を次に示します。どちらかの数値が浮動小数点数の場合、結果は剰余を表す浮動小数点数になります。

VB

```
Dim testResult As Double
```

```
testResult = 10 Mod 5
testResult = 10 Mod 3
testResult = 12 Mod 4.3
testResult = 12.6 Mod 5
testResult = 47.9 Mod 9.35
```

この例の式は、0、1、3.4、2.6、および 1.15 の値をそれぞれ返します。

浮動小数点数のオペランドによって誤差が生じる例を次に示します。最初のステートメントのオペランドは、どちらも倍精度浮動小数点型 (**Double**) であり、無限に繰り返されるバイナリ分数である 0.2 が、値 0.20000000000000001 として格納されています。2 つ目のステートメントでは、リテラルの型文字 `D` が指定されているため、2 つのオペランドが 10 進型 (**Decimal**) になり、0.2 が正確に表現されています。

VB

```
firstResult = 2.0 Mod 0.2
' Double operation returns 0.2, not 0.
secondResult = 2D Mod 0.2D
' Decimal operation returns 0.
```

参照

処理手順

データ型のトラブルシューティング

関連項目

算術演算子 (Visual Basic)

Visual Basic における演算子の優先順位

機能別の演算子一覧

Int 関数、Fix 関数 (Visual Basic)

\ 演算子

概念

Visual Basic における算術演算子

Not 演算子 (Visual Basic)

2 つのブール式の論理否定を求めます。また、2 つの数式のビットごとの否定を求めます。

```
result = Not expression
```

指定項目

result

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。

expression

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。

解説

Boolean 式の場合、次の表は、2 つの式の値と演算結果 *result* の値の対応を示しています。

***expression* の値**	***result* の型**
True	**False**
False	**True**

数値の場合、**Not** 演算子は、数式に対してビット単位の反転も行います。次に示す真理値表に従って、演算結果 *result* の対応するビットがセットされます。

***expression* のビット**	***result* 内のビット**
1	0
0	1

メモ :

論理/ビット処理演算子は他の算術演算子や関係演算子より優先順位が低いため、ビットごとの演算は、正確に実行されるようにかっこで囲む必要があります。

データ型

ブール値の否定では、結果のデータ型は **Boolean** です。ビットごとの否定では、結果のデータ型は *expression* と同じです。ただし、式が **Decimal** の場合、結果は **Long** になります。

オーバーロード

Not は オーバーロード できます。つまり、オペランドがクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。このようなクラスまたは構造体でこの演算子を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

使用例

次の例では、**Not** 演算子を使って 2 つの **Boolean** 式の論理否定を求めます。結果は、式の値を反転させた **Boolean** 値です。

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck As Boolean
firstCheck = Not (a > b)
secondCheck = Not (b > a)
```

前の例では、**False** と **True** の結果を順に生成します。

Not 演算子を使って、数式のビットごとの論理否定を求める例を次に示します。結果パターンのビットは、符号ビットを含めて、オペランド パターンで対応するビットを反転した値にセットされます。

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstPattern, secondPattern, thirdPattern As Integer
firstPattern = (Not a)
secondPattern = (Not b)
thirdPattern = (Not c)
```

前の例では、–11、–9、–7 の結果を順に生成します。

参照

関連項目

論理/ビット演算子

Visual Basic における演算子の優先順位

機能別の演算子一覧

概念

Visual Basic の論理演算子とビット処理演算子

Or 演算子 (Visual Basic)

2 つのブール (**Boolean**) 式の論理和を求めます。または、2 つの数式のビットごとの論理和を求めます。

```
result = expression1 Or expression2
```

指定項目

result

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。**Boolean** 式の比較の場合、*result* は 2 つの **Boolean** 値の包括的論理和になります。ビットごとの演算の場合、*result* は 2 つの数式のビットパターンの、ビットごとの包括的論理和を表す数値になります。

expression1

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。

expression2

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。

解説

Boolean 値の比較の場合、*result* が **False** になるのは、*expression1* と *expression2* がどちらも **False** に評価される場合だけです。次の表は、2 つの式と *result* の値の関係を示しています。

***expression1* の値**	***expression2* の値**	***result* の値**
True	**True**	**True**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

メモ :

Boolean 式の比較において、**Or** 演算子は必ず両方の式を評価します。これは式がプロシージャ呼び出しを実行する場合も同じです。OrElse 演算子 はショートサーキットを実行します。つまり、*expression1* が **True** であれば、*expression2* は評価されません。

ビットごとの演算の場合、**Or** 演算子は、2 つの数式内の対応するビットどうしの、ビット単位の比較を行います。次の表に従って、*result* 内の対応するビットがセットされます。

***expression1* のビット**	***expression2* のビット**	***result* のビット**
1	1	1
1	0	1
0	1	1
0	0	0

メモ :

論理演算子やビット処理演算子は、他の算術演算子や関係演算子より優先順位が低いため、ビットごとの演算は、正確に実行されるようにかっこで囲む必要があります。

データ型

オペランドの一方が **Boolean** 式で、もう一方が数式の場合、Visual Basic は **Boolean** 式を数値 (**True** は -1、**False** は 0) に変換してから、ビットごとの演算を実行します。

Boolean 式の比較の場合、結果のデータ型は **Boolean** になります。ビットごとの比較の場合、結果のデータ型は *expression1* と *expression2* のデータ型に対して適切な数値型になります。「演算子の結果のデータ型」の「リレーショナルおよびビットごとの比較」の表を参照してください。

オーバーロード

Or 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。この演算子がコード内でこのようなクラスや構造体に対して使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

Or 演算子を使って 2 つの式の包括的論理和を求める例を次に示します。結果は、2 つの式のどちらかが **True** かどうかを表す **Boolean** 値です。

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck, thirdCheck As Boolean
firstCheck = a > b Or b > c
secondCheck = b > a Or b > c
thirdCheck = b > a Or c > b
```

先の例では、**True**、**True**、**False** という結果がそれぞれ生成されます。

Or 演算子を使って、2 つの数式のビットごとの包括的論理和を求める例を次に示します。オペランド内の、対応するビットのいずれかが 1 にセットされている場合は、結果パターンにビットがセットされます。

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstPattern, secondPattern, thirdPattern As Integer
firstPattern = (a Or b)
secondPattern = (a Or c)
thirdPattern = (b Or c)
```

先の例では、結果はそれぞれ 10、14、14 になります。

参照

関連項目

論理/ビット演算子
Visual Basic における演算子の優先順位
機能別の演算子一覧
OrElse 演算子

概念

Visual Basic の論理演算子とビット処理演算子

OrElse 演算子

2 つの式の包括的論理和を簡略的に求めます。

```
result = expression1 OrElse expression2
```

指定項目

result

必ず指定します。任意のブール型 (**Boolean**) の式を指定します。

expression1

必ず指定します。任意のブール型 (**Boolean**) の式を指定します。

expression2

必ず指定します。任意のブール型 (**Boolean**) の式を指定します。

解説

コンパイルされたコードで、1 つの式の結果によってはもう 1 つの式の評価を省略できる場合、その論理演算をショートサーキットと呼びます。最初に評価される式の結果によって演算の最終結果が決まる場合は、もう 1 つの式によって最終結果が変わることはないため、その式を評価する必要はありません。省略される側の式が複雑な場合や、プロシージャの呼び出しを含む場合は、ショートサーキットによってパフォーマンスを向上できます。

一方の式、または両方の式が **True** に評価される場合、*result* は **True** になります。次の表は、2 つの式と *result* の値の関係を示しています。

***expression1* の値**	***expression2* の値**	***result* の値**
True	(評価しない)	**True**
False	**True**	**True**
False	**False**	**False**

データ型

OrElse 演算子は、ブール型 (Boolean) (Visual Basic) に対してのみ定義されます。Visual Basic は各オペランドを必要に応じてブール型 (**Boolean**) に変換し、演算をすべて **Boolean** で実行します。結果を数値型に代入する場合は、結果が **Boolean** から数値型に変換されます。これによって、予期しない結果が返される可能性があります。たとえば、`5 OrElse 12` の結果を整数 (**Integer**) に変換すると、`-1` になります。

オーバーロード

Or 演算子 (Visual Basic) と IsTrue 演算子 はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。**Or** 演算子と **IsTrue** 演算子のオーバーロードは、**OrElse** 演算子の動作に影響を与えます。コードが、**Or** および **IsTrue** をオーバーロードするクラスや構造体で **OrElse** を使用している場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

OrElse 演算子を使って 2 つの式の論理和を求める例を次に示します。結果は、2 つの式のどちらかが真かどうかを表すブール値です。最初の式が真 (**True**) の場合、2 番目の式は評価されません。

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck, thirdCheck As Boolean
firstCheck = a > b OrElse b > c
secondCheck = b > a OrElse b > c
thirdCheck = b > a OrElse c > b
```

先の例では、**True**、**True**、**False** という結果がそれぞれ生成されます。`firstCheck` の計算では、最初の式が **True** であるため、2 番目の式は評価されません。一方、`secondCheck` の計算では、2 番目の式も評価されます。

2 つのプロシージャ呼び出しを含む **If**...**Then** ステートメントの例を次に示します。最初の呼び出しが **True** を返した場合、2 番目のプロシージャは呼び出されません。このため、2 番目のプロシージャがコードのこの部分で必ず処理されるべき重要なタスクを実行する場合に、予期しない結果が返される可能性があります。

VB

```
If testFunction(5) = True OrElse otherFunction(4) = True Then
    ' If testFunction(5) is True, otherFunction(4) is not called.
    ' Insert code to be executed.
End If
```

参照

関連項目

論理/ビット演算子
Visual Basic における演算子の優先順位
機能別の演算子一覧
Or 演算子 (Visual Basic)
IsTrue 演算子

概念

Visual Basic の論理演算子とビット処理演算子

TypeOf 演算子 (Visual Basic)

オブジェクト参照変数をデータ型と比較します。

```
result = TypeOf objectexpression Is typename
```

指定項目

result

戻り値です。**Boolean** 値。

objectexpression

必ず指定します。参照型に評価される式を指定します。

typename

必ず指定します。データ型の名前です。

解説

TypeOf 演算子は、*objectexpression* のランタイム型に *typename* との互換性があるかどうかを判断します。互換性があるかどうかは *typename* の型のカテゴリによって決まります。次の表に、互換性の条件を示します。

***typename* の型のカテゴリ**	**互換性の条件**
クラス	*objectexpression* の型が *typename*、または *typename* から継承した型
構造体	*objectexpression* の型が *typename*
インターフェイス	*objectexpression* が *typename* を実装しているか、*typename* を実装しているクラスから継承している

objectexpression のランタイム型が互換性の条件を満たしていれば、*result* は **True** になります。それ以外の場合、*result* は **False** になります。

TypeOf には必ずキーワード **Is** を指定して、**TypeOf**...**Is** 式の形で使用します。

使用例

TypeOf...**Is** 式を使用して、2 つのオブジェクト参照変数とさまざまなデータ型の間で、型の互換性をテストする例を次に示します。

VB

```
Dim refInteger As Object = 2
MsgBox("TypeOf Object[Integer] Is Integer? " & TypeOf refInteger Is Integer)
MsgBox("TypeOf Object[Integer] Is Double? " & TypeOf refInteger Is Double)
Dim refForm As Object = New System.Windows.Forms.Form
MsgBox("TypeOf Object[Form] Is Form? " & TypeOf refForm Is System.Windows.Forms.Form)
MsgBox("TypeOf Object[Form] Is Label? " & TypeOf refForm Is System.Windows.Forms.Label)
MsgBox("TypeOf Object[Form] Is Control? " & TypeOf refForm Is System.Windows.Forms.Control)
MsgBox("TypeOf Object[Form] Is IComponent? " & TypeOf refForm Is System.ComponentModel.ICom
ponent)
```

変数 `refInteger` のランタイム型は **Integer** です。**Integer** とは互換性がありますが、**Double** とは互換性がありません。変数 `refForm` のランタイム型は Form です。この変数と互換性があるのは、**Form** (同じ型だから)、Control (**Form** は **Control** を継承しているから)、および IComponent (**Form** は **IComponent** を継承し、Component を実装しているから) です。しかし、`refForm` は Label と互換性がありません。

参照

関連項目

Is 演算子 (Visual Basic)
IsNot 演算子
Visual Basic における演算子の優先順位

Xor 演算子 (Visual Basic)

2 つのブール (**Boolean**) 式の排他的論理和を求めます。または、2 つの数式のビットごとの排他を求めます。

```
result = expression1 Xor expression2
```

指定項目

result

必ず指定します。任意のブール変数または数値変数を指定します。ブール式の比較の場合、*result* は 2 つの **Boolean** 値の排他的論理和です。ビットごとの演算の場合、*result* は 2 つの数式のビットパターンの、ビットごとの排他的論理和を表す数値です。

expression1

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。

expression2

必ず指定します。任意のブール型 (**Boolean**) または数式を指定します。

解説

ブール式の比較の場合、*expression1* と *expression2* のどちらか一方だけが真 (**True**) の場合、つまり、*expression1* と *expression2* の **Boolean** 値が同じでない場合に、*result* が **True** になります。次の表は、2 つの式と *result* の値の関係を示しています。

***expression1* の値**	***expression2* の値**	***result* の値**
True	**True**	**False**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

メモ :

ブール式の比較の場合、**Xor** 演算子は必ず両方の式を評価します。これはプロシージャ呼び出しを作成する場合も同じです。**Xor** の結果は必ず 2 つのオペランドによって決まるため、この演算子に相当するショートサーキットはありません。ショートサーキット論理演算子については、「AndAlso 演算子」と「OrElse 演算子」を参照してください。

ビットごとの演算の場合、**Xor** 演算子は、2 つの数式内の対応するビットどうしの、ビット単位の比較を行います。次の表に従って、*result* 内の対応するビットがセットされます。

***expression1* のビット**	***expression2* のビット**	***result* のビット**
1	1	0
1	0	1
0	1	1
0	0	0

メモ :

論理演算子やビット処理演算子は、他の算術演算子や関係演算子より優先順位が低いため、ビットごとの演算は、正確に実行されるようにかっこで囲む必要があります。

たとえば、5 **Xor** 3 の結果は 6 になります。その理由は、5 と 3 を対応する 2 進数 (それぞれ 101 と 011) に変換してみるとわかります。前出

の表によると、101 Xor 011 の結果は 110 です。これは、10 進数の 6 を 2 進数で表した値です。

データ型

オペランドの一方が **Boolean** 式で、もう一方が数式の場合、Visual Basic は **Boolean** 式を数値 (**True** は -1、**False** は 0) に変換してから、ビットごとの演算を実行します。

Boolean 式の比較の場合、結果のデータ型は **Boolean** になります。ビットごとの比較の場合、結果のデータ型は *expression1* と *expression2* のデータ型に対して適切な数値型になります。「演算子の結果のデータ型」の「リレーショナルおよびビットごとの比較」の表を参照してください。

オーバーロード

Xor 演算子はオーバーロードできます。つまり、オペランドがクラスや構造体を型として持つ場合に、演算子の動作をそのクラスや構造体で再定義できるという意味です。この演算子がコード内でこのようなクラスや構造体に対して使用されている場合は、再定義された後の動作を必ず理解するようにしてください。詳細については、「演算子プロシージャ」を参照してください。

使用例

Xor 演算子を使って 2 つの式の排他的論理和を求める例を次に示します。結果は、2 つの式のどちらか一方が **True** かどうかを表す **Boolean** 値です。

VB

```
Dim a As Integer = 10
Dim b As Integer = 8
Dim c As Integer = 6
Dim firstCheck, secondCheck, thirdCheck As Boolean
firstCheck = a > b Xor b > c
secondCheck = b > a Xor b > c
thirdCheck = b > a Xor c > b
```

先の例では、**False**、**True**、**False** という結果がそれぞれ生成されます。

Xor 演算子を使って 2 つの数式のビットごとの排他的論理和を求める例を次に示します。オペランド内の、対応するビットのどちらか一方だけが 1 にセットされている場合は、結果パターンにビットがセットされます。

VB

```
Dim a As Integer = 10 ' 1010 in binary
Dim b As Integer = 8  ' 1000 in binary
Dim c As Integer = 6  ' 0110 in binary
Dim firstPattern, secondPattern, thirdPattern As Integer
firstPattern = (a Xor b)  '  2, 0010 in binary
secondPattern = (a Xor c) ' 12, 1100 in binary
thirdPattern = (b Xor c)  ' 14, 1110 in binary
```

先の例では 2、12、14 という結果がそれぞれ生成されます。

参照

関連項目

論理/ビット演算子
Visual Basic における演算子の優先順位
機能別の演算子一覧

概念

Visual Basic の論理演算子とビット処理演算子

演算子の結果のデータ型

Visual Basic は、オペランドのデータ型を基に、演算結果のデータ型を確認します。これは、いずれかのオペランドよりも広い範囲を持つデータ型である可能性もあります。

データ型の範囲

次に、対象となるデータ型の範囲を小さいものから順に示します。

- Boolean : 2種類の値
- SByte, Byte : 256 種類の整数値
- Short, UShort : 65,536 (6.5...E+4) 種類の整数値
- Integer, UInteger : 4,294,967,296 (4.2...E+9) 種類の整数値
- Long, ULong : 18,446,744,073,709,551,615 (1.8...E+19) 種類の整数値
- Decimal : 1.5...E+29 種類の整数値、最大範囲 7.9...E+28 (絶対値)
- Single : 最大範囲 3.4...E+38 (絶対値)
- Double : 最大範囲 1.7...E+308 (絶対値)

Visual Basic のデータ型の詳細については、「データ型の概要 (Visual Basic)」を参照してください。

オペランドが Nothing と評価される場合、Visual Basic の算術演算子はこれを 0 と見なします。

10進数演算

Decimal データ型は、浮動小数点でも整数でもないという点に注意してください。

+、**–**、*****、**/**、または **Mod** 演算の一方のオペランドが **Decimal** であり、もう一方が **Single** または **Double** でない場合、Visual Basic は後者のオペランドを **Decimal** に拡大変換します。演算は **Decimal** で実行され、結果のデータ型は **Decimal** です。

浮動小数点数演算

Visual Basic では、ほとんどの浮動小数点演算が、最も効率的なデータ型である Double で実行されます。しかし、一方のオペランドが Single で、他方が **Double** でない場合、Visual Basic は演算を **Single** で実行します。演算に先立って、各オペランドは必要に応じて適切なデータ型に拡大変換され、演算結果はそのデータ型になります。

/ および ^ 演算子

/ 演算子は、Decimal、Single、および Double データ型に対してのみ定義されています。Visual Basic は、演算に先立って、各オペランドを必要に応じて適切なデータ型に拡大変換し、演算結果はそのデータ型になります。

次の表に、**/** 演算子の結果のデータ型を示します。この表が、オペランドのデータ型の組み合わせで左右対称である点に注目してください。結果のデータ型は、オペランドの順序にかかわらず同じです。

	Decimal	**Single**	**Double**	任意の整数型
Decimal	10 進数型	単精度浮動小数点型	倍精度浮動小数点数型	倍精度浮動小数点数型
Single	単精度浮動小数点型	単精度浮動小数点型	倍精度浮動小数点数型	倍精度浮動小数点数型
Double	倍精度浮動小数点数型	倍精度浮動小数点数型	倍精度浮動小数点数型	倍精度浮動小数点数型
任意の整数型	倍精度浮動小数点数型	倍精度浮動小数点数型	倍精度浮動小数点数型	倍精度浮動小数点数型

^ 演算子は、**Double** データ型向けに定義されています。Visual Basic は、演算に先立って、各オペランドを必要に応じて **Double** に変換し、結果のデータ型は必ず **Double** です。

整数演算

整数演算の結果のデータ型は、オペランドのデータ型に依存します。一般的に、Visual Basic は次のポリシーを使用して結果のデータ型を決定します。

- 二項演算子の両方のオペランドが同じデータ型である場合、結果はそのデータ型になります。**Boolean** は例外で、強制的に **Short** になります。
- 符号なしオペランドと符号付きオペランドの演算を行うと、結果は、最低でもいずれかのオペランドの範囲を持つ符号付きのデータ型になります。
- それ以外の場合は、結果は 2 つのオペランドのうち大きいほうのデータ型になります。

結果のデータ型が、いずれのオペランドのデータ型と同じにならない場合もあります。

メモ：
演算結果として返される値が大きすぎる場合、結果のデータ型に格納できないこともあります。結果のデータ型よりも値が大きい場合、OverflowException 例外が発生します。

単項プラスおよび単項マイナス演算子

次の表に、2 つの単項演算子 + および – の結果のデータ型を示します。

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**
単項 +	短整数型 (Short)	SByte 型	バイト型	短整数型 (Short)	ushort 型	整数型	UInteger型	長整数型 (Long)	ulong型
単項 –	短整数型 (Short)	SByte 型	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	10 進数型

<< および >> 演算子

2つの ビット シフト演算子 << および >> の結果のデータ型を次の表に示します。Visual Basic では、各ビット シフト演算子は左側のオペランド (シフトされるビット パターン) の単項演算子として扱われます。

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**
<<, >>	短整数型 (Short)	SByte型	バイト型	短整数型 (Short)	ushort型	整数型	UInteger型	長整数型 (Long)	ulong型

左側のオペランドが **Decimal**、**Single**、**Double**、または **String** の場合、Visual Basic では演算の前にこれが **Long** に変換され、結果のデータ型は **Long** になります。右側のオペランド (シフトするビット数) は **Integer** または **Integer** に拡大変換できるデータ型である必要があります。

二項 +、–、*、Mod 演算子

二項 + と – 演算子、および * と **Mod** 演算子の結果のデータ型を次の表に示します。この表が、オペランドのデータ型の組み合わせで左右対称である点に注目してください。結果のデータ型は、オペランドの順序にかかわらず同じです。

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**
Boolean	短整数型 (Short)	SByte型	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	10 進数型
SByte	SByte型	SByte型	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	10 進数型
Byte	短整数型 (Short)	短整数型 (Short)	バイト型	短整数型 (Short)	ushort型	整数型	UInteger型	長整数型 (Long)	ulong型
Short	短整数型 (Short)	短整数型 (Short)	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	10 進数型
UShort	整数型	整数型	ushort型	整数型	ushort型	整数型	UInteger型	長整数型 (Long)	ulong型

Integer	整数型	整数型	整数型	整数型	整数型	整数型	長整数型 (Long)	長整数型 (Long)	10 進数型
UInteger	長整数型 (Long)	長整数型 (Long)	UInteger型	長整数型 (Long)	UInteger型	長整数型 (Long)	UInteger型	長整数型 (Long)	ulong型
Long	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	10 進数型
ULong	10 進数型	10 進数型	ulong型	10 進数型	ulong型	10 進数型	ulong型	10 進数型	ulong型

\ 演算子

次の表に、**** 演算子の結果のデータ型を示します。この表が、オペランドのデータ型の組み合わせで左右対称である点に注目してください。結果のデータ型は、オペランドの順序にかかわらず同じです。

	Boolean	**SByte**	**Byte**	**Short**	**UShort**	**Integer**	**UInteger**	**Long**	**ULong**
Boolean	短整数型 (Short)	SByte型	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
SByte	SByte型	SByte型	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
Byte	短整数型 (Short)	短整数型 (Short)	バイト型	短整数型 (Short)	ushort型	整数型	UInteger型	長整数型 (Long)	ulong型
Short	短整数型 (Short)	短整数型 (Short)	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
UShort	整数型	整数型	ushort型	整数型	ushort型	整数型	UInteger型	長整数型 (Long)	ulong型
Integer	整数型	整数型	整数型	整数型	整数型	整数型	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
UInteger	長整数型 (Long)	長整数型 (Long)	UInteger型	長整数型 (Long)	UInteger型	長整数型 (Long)	UInteger型	長整数型 (Long)	ulong型
Long	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
ULong	長整数型 (Long)	長整数型 (Long)	ulong型	長整数型 (Long)	ulong型	長整数型 (Long)	ulong型	長整数型 (Long)	ulong型

**** 演算子のいずれかのオペランドが Decimal、Single、または Double の場合、Visual Basic では演算に先立ってこれを Long に変換し、結果のデータ型は **Long** になります。

リレーショナルおよびビットごとの比較

リレーショナル演算 (**=**、**<>**、**<**、**>**、**<=**、**>=**) の結果のデータ型は常に **Boolean**ブール型 (Boolean) (Visual Basic) です。これは、**Boolean** オペランドの論理演算 (**And**、**AndAlso**、**Not**、**Or**、**OrElse**、**Xor**) でも同様です。

ビットごとの論理演算の結果のデータ型は、オペランドのデータ型に依存します。**AndAlso** および **OrElse** は、**Boolean** 用にのみ定義されており、Visual Basic では、演算が実行される前に、必要に応じて各オペランドが **Boolean** に変換されます。

=、<>、<、>、<=、>= 演算子

両方のオペランドが **Boolean** の場合、Visual Basic では **True** の値を **False** よりも小さいものとして見なします。数値型を **String** と比較する場合、Visual Basic では演算に先立って **String** が **Double** に変換されます。**Char** または **Date** のオペランドは、同じデータ型のオペランドとのみ比較可能です。結果のデータ型は常に **Boolean** です。

ビットごとの Not 演算子

次の表に、ビットごとの **Not** 演算子の結果のデータ型を示します。

	Boolean	SByte	Byte	Short	UShort	Integer	UInteger	Long	ULong
Not	ブール値	SByte型	バイト型	短整数型 (Short)	ushort型	整数型	UInteger型	長整数型 (Long)	ulong型

オペランドが **Decimal**、**Single**、**Double**、または **String** の場合、Visual Basic では演算の前にこれが **Long** に変換され、結果のデータ型は **Long** になります。

ビットごとの And、Or、Xor 演算子

次の表に、ビットごとの **And**、**Or**、および **Xor** 演算子の結果のデータ型を示します。この表が、オペランドのデータ型の組み合わせで左右対称である点に注目してください。結果のデータ型は、オペランドの順序にかかわらず同じです。

	Boolean	SByte	Byte	Short	UShort	Integer	UInteger	Long	ULong
Boolean	ブール型	SByte型	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
SByte	SByte型	SByte型	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
Byte	短整数型 (Short)	短整数型 (Short)	バイト型	短整数型 (Short)	ushort型	整数型	UInteger型	長整数型 (Long)	ulong型
Short	短整数型 (Short)	短整数型 (Short)	短整数型 (Short)	短整数型 (Short)	整数型	整数型	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
UShort	整数型	整数型	ushort型	整数型	ushort型	整数型	UInteger型	長整数型 (Long)	ulong型
Integer	整数型	整数型	整数型	整数型	整数型	整数型	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
UInteger	長整数型 (Long)	長整数型 (Long)	UInteger型	長整数型 (Long)	UInteger型	長整数型 (Long)	UInteger型	長整数型 (Long)	ulong型
Long	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)	長整数型 (Long)
ULong	長整数型 (Long)	長整数型 (Long)	ulong型	長整数型 (Long)	ulong型	長整数型 (Long)	ulong型	長整数型 (Long)	ulong型

オペランドが **Decimal**、**Single**、**Double**、または **String** の場合、Visual Basic では演算の前にこれが **Long** に変換され、結果のデータ型は **Long** になります。

その他の演算子

& 演算子は、**String** オペランドの連結のために定義されています。Visual Basic では、演算に先立って、各オペランドが必要に応じて **String** に変換され、結果のデータ型は必ず **String** になります。**Option Strict** が **On** の場合でも、**&** 演算子の目的のために、**String** への変換はすべて拡大変換の対象となります。

Is と **IsNot** 演算子では、両方のオペランドが参照型である必要があります。**TypeOf...Is** 式では、最初のオペランドが参照型、2 番目のオペランドがデータ型の名前である必要があります。いずれの場合も、結果のデータ型は **Boolean** です。

Like 演算子は、**String** オペランドのパターン一致用にのみ定義されています。Visual Basic では、演算に先立って、必要に応じて各オペラン

ドを **String** に変換します。結果のデータ型は常に **Boolean** です。

参照

関連項目

データ型の概要 (Visual Basic)
Visual Basic における演算子の優先順位
機能別の演算子一覧
算術演算子 (Visual Basic)
比較演算子 (Visual Basic)
Option Strict ステートメント

概念

Visual Basic の演算子および式
Visual Basic における算術演算子
Visual Basic における比較演算子

その他の技術情報

演算子 (Visual Basic)

算術演算子 (Visual Basic)

Visual Basic では以下の算術演算子が定義されています。

^ 演算子

* 演算子

/ 演算子

\ 演算子

Mod 演算子

+ 演算子 (単項および二項)

- 演算子 (単項および二項)

参照

関連項目

Visual Basic における演算子の優先順位

概念

Visual Basic における算術演算子

代入演算子

Visual Basic では以下の代入演算子が定義されています。

= 演算子

^= 演算子

*= 演算子

/= 演算子

\= 演算子

+= 演算子

-= 演算子

<<= 演算子

>>= 演算子

&= 演算子

参照

関連項目

Visual Basic における演算子の優先順位

機能別の演算子一覧

概念

代入ステートメント

ビット シフト演算子

Visual Basic では以下のビット シフト演算子が定義されています。

<< 演算子

>> 演算子

参照

関連項目

機能別の演算子一覧

比較演算子 (Visual Basic)

Visual Basic では以下の比較演算子が定義されています。

< 演算子

<= 演算子

> 演算子

>= 演算子

= 演算子

<> 演算子

Is 演算子 (Visual Basic)

IsNot 演算子

Like 演算子

これらの演算子は、2 つの式を比較して、この 2 つが等しいかどうか、等しくない場合はどのように異なるかを判断します。**Is**、**IsNot**、**Like** は、別のヘルプ トピックで解説します。関係比較演算子は、このページで解説します。

```
result = expression1 comparisonoperator expression2
result = object1 [Is | IsNot] object2
result = string Like pattern
```

指定項目

result

必ず指定します。比較の結果を表す **Boolean** 値です。

expression

必ず指定します。任意の式を指定します。

comparisonoperator

必ず指定します。任意の関係比較演算子を指定します。

object1, *object2*

必ず指定します。任意の参照オブジェクト名を指定します。

string

必ず指定します。任意のブール型 (**String**) の式を指定します。

pattern

必ず指定します。任意の文字列 (**String**) 式、または文字の範囲を指定します。

解説

次の表は、比較演算子と条件の一覧です。この組み合わせにより、演算結果 *result* が **True**、**False** のどちらになるかが決まります。

演算子	True if	False if
< (より小さい)	*expression1* < *expression2*	*expression1* >= *expression2*
<= (以下)	*expression1* <= *expression2*	*expression1* > *expression2*
> (より大きい)	*expression1* > *expression2*	*expression1* <= *expression2*
>= (以上)	*expression1* >= *expression2*	*expression1* < *expression2*

= (等しい)	*expression1* = *expression2*	*expression1* <> *expression2*
<>(等しくない)	*expression1* <> *expression2*	*expression1* = *expression2*

メモ：

= 演算子 (Visual Basic) は、代入演算子としても使用されます。

Is 演算子、**IsNot** 演算子、**Like** 演算子は、この表の演算子とは異なる特定の比較機能を持ちます。

数値を比較する

単精度浮動小数点数型 (**Single**) の式を倍精度浮動小数点数型 (**Double**) の式と比較する場合は、単精度浮動小数点数型 (**Single**) 式が倍精度浮動小数点数型 (**Double**) 式に変換されます。この動作は、Visual Basic 6 の場合と逆になっています。

同様に、10 進型 (**Decimal**) の式を単精度浮動小数点数型 (**Single**) または倍精度浮動小数点数型 (**Double**) の式と比較する場合は、10 進型 (**Decimal**) 式が単精度浮動小数点数型 (**Single**) または倍精度浮動小数点数型 (**Double**) 式に変換されます。10 進型 (**Decimal**) 式では、*1E-28* よりも小さい小数値は失われます。1E-28 より小さい小数値が失われると、本来は等しくない 2 つの数値を比較すると、等しいと判断される可能性があります。このため、等号 (=) を使用して 2 つの浮動小数点変数を比較する場合は注意が必要です。2 つの数値の差の絶対値が許容差よりも小さいかどうかを確認することが推奨されます。

浮動小数点の不正確性

浮動小数点数を扱う場合、これらの値がメモリ内で常に正確に表現されているわけではないことに注意してください。これにより、値の比較や Mod 演算子 (Visual Basic) などの特定の演算が予期しない結果になることがあります。詳細については、「データ型のトラブルシューティング」を参照してください。

文字列の比較

文字列を比較するとき、文字列式は **Option Compare** の設定によるアルファベット順の並べ替え順序を基に評価されます。

Option Compare Binary では、文字の内部バイナリ表現による並べ替え順序に基づいて文字列が比較されます。並べ替え順序は、コード ページによって決まります。次の例で、一般的なバイナリの並べ替え順序を示します。

```
A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø
```

Option Compare Text では、文字列比較は、大文字と小文字を区別しない、アプリケーションのロケールで決められたテキストの並べ替え順序に基づいて行われます。**Option Compare Text** を設定して、上の例の文字を並べ替えると、次のテキストの並べ替え順序が適用されます。

```
(A=a) < (À= à) < (B=b) < (E=e) < (Ê= ê) < (Ø = ø) < (Z=z)
```

ロケールの依存関係

Option Compare Text を設定すると、文字列比較の結果が、アプリケーションが実行されているロケールに依存します。あるロケールでは同一であると判断される文字も、別のロケールでは異なる文字として扱われることがあります。文字列比較を使用して重要な判断 (ログオンを許可するかどうかなど) をする場合は、ロケールの区別に十分注意してください。**Option Compare Binary** を設定するか、ロケールを考慮する StrComp 関数 (Visual Basic) を呼び出すかを検討してください。

関係比較演算子を使った型宣言を省略したプログラミング

Option Strict On では、**Object** 式には関係比較演算子を使用できません。**Option Strict** が **Off** の場合、および、*expression1* または *expression2* が **Object** 式の場合、実行時の型によって比較方法が決まります。次の表は、式を比較する方法および比較した結果をオペランドのランタイム型に応じて示したものです。

オペランドの種類	比較の種類
両方が **String** の場合	文字列の並べ替え特性に基づいて並べ替え比較が行われます。
両方が数値の場合	オブジェクトが **Double** に変換され、数値比較
一方が数値、他方が **String** の場合	文字列は倍精度浮動小数点数型 (**Double**) に変換され、数値比較が行われます。文字列型 (**String**) を倍精度浮動小数点数型 (**Double**) に変換できない場合は、InvalidCastException がスローされます。
一方または両方が文字列以外の参照型の場合	**InvalidCastException** がスローされます。

数値の比較では、**Nothing** を 0 として扱います。文字列比較では、**Nothing** は **""** (空白の文字列) として扱います。

オーバーロード

関係比較演算子 (**<**、**<=**、**>**、**>=**、**=**、**<>**) はオーバーロードできます。つまり、オペランドがそのクラスまたは構造体の型であれば、クラスまたは構造体がこの動作を再定義できます。このようなクラスまたは構造体でこれらの演算子を使用している場合、再定義された動作を確認してください。詳細については、「演算子プロシージャ」を参照してください。

= 演算子 (Visual Basic) は関係比較演算子としてのみオーバーロードでき、代入演算子としてはオーバーロードできないことに注意してください。

使用例

式の比較に用いる関係比較演算子のさまざまな使用例を次に示します。関係比較演算子は、指定した式が真 (**True**) かどうかを表す **Boolean** 型の結果を返します。**>** 演算子と **<** 演算子を文字列に適用する場合は、通常のアルファベット順で文字列の比較が行われます。この順序は、ロケールの設定に依存する可能性があります。並べ替えが大文字と小文字を区別するかどうかは、Option Compare ステートメントの設定値によって決まります。

VB

```
Dim testResult As Boolean
testResult = 45 < 35
testResult = 45 = 45
testResult = 4 <> 3
testResult = "5" > "4444"
```

上の例では、最初の比較が **False** を返し、その他の比較が **True** を返します。

参照

処理手順

データ型のトラブルシューティング

関連項目

= 演算子 (Visual Basic)

Visual Basic における演算子の優先順位

機能別の演算子一覧

InvalidCastException

概念

Visual Basic における比較演算子

連結演算子 (Visual Basic)

Visual Basic では以下の算術演算子が定義されています。

& 演算子

+ 演算子

参照

関連項目

Visual Basic における演算子の優先順位
System.Text
StringBuilder

概念

Visual Basic の連結演算子

論理/ビット演算子

Visual Basic では以下の論理演算子またはビット処理演算子が定義されています。

And 演算子 (Visual Basic)

Not 演算子 (Visual Basic)

Or 演算子 (Visual Basic)

Xor 演算子 (Visual Basic)

AndAlso 演算子

OrElse 演算子

IsFalse 演算子

IsTrue 演算子

参照

関連項目

Visual Basic における演算子の優先順位

概念

Visual Basic の論理演算子とビット処理演算子

その他の演算子

Visual Basic に定義されているその他の演算子は次のとおりです。

AddressOf 演算子

GetType 演算子

TypeOf 演算子 (Visual Basic)

参照

関連項目

機能別の演算子一覧

プロパティ (Visual Basic)

このページには、Visual Basic モジュールのメンバであるプロパティが一覧表示されます。特定の Visual Basic オブジェクトのメンバである他のプロパティは、ヘルプの目次内の他の場所に一覧表示されます。これらのプロパティを表示するには、オブジェクト (Visual Basic) ページを開き、目次に同期させます。目次の「オブジェクト」ノードを展開すると、Visual Basic オブジェクトの一覧が表示されます。プロパティは、対応するオブジェクトの下に一覧表示されます。

このセクションの内容

関連するセクション

DateString プロパティ

システムに従った現在の日付を表す文字列型 (**String**) の値を取得または設定します。

```
Public Property DateString As String
```

例外

例外の種類	エラー番号	条件
InvalidCastException	5	**DateString** の値が、無効な形式を使用して設定されています。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

DateString は、システム日付を常に "MM-dd-yyyy" として返します。この形式では、省略形の月名が使用されます。日付の設定で受け入れられる形式は、"M-d-yyyy"、"M-d-y"、"M/d/yyyy"、および "M/d/y" です。これらの形式は、カルチャによって変化しません。つまり、[コントロール パネル] の [地域のオプション] を変更しても変わらないという意味です。

現在のシステム時刻を文字列型 (**String**) として取得または設定するには、TimeString プロパティ を使用します。

現在のシステム日付またはシステム時刻を、自分のロケールの書式、またはカスタムの書式で取得するには、Now プロパティ に Format 関数 を定義し、定義済みの日付/時刻書式 (Format 関数) または ユーザー定義の日付/時刻書式 (Format 関数) を指定します。次にコード例を示します。

```
MsgBox("The formatted date is " & Format(Now, "dddd, d MMM yyyy"))
```

日付型 (**Date**) としての現在のシステム日付にアクセスするには、Today プロパティ を使用します。

セキュリティに関するメモ :

システム日付やシステム時刻を設定するには、アンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

使用例

次の例は、**DateString** プロパティを使って、現在のシステムの日付を表示します。

VB

```
MsgBox("The current date is " & DateString)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

DateString はクラスではなくモジュールのメンバであるため、**DateString** にアクセスするためのオブジェクトを作成する必要はありません。

スマート デバイス開発者のためのメモ

DateString プロパティを使用してシステム日付を取得することはできますが、設定することはできません。

参照

関連項目

Now プロパティ
TimeString プロパティ

Today プロパティ
日付型 (Date) (Visual Basic)
DateTime

Now プロパティ

システムに従った現在の日付と時刻を含む日付型 (**Date**) の値を返します。

```
ReadOnly Public Property Now() As DateTime
```

解説

システム日付を設定するには、Today プロパティ を使用します。システム時刻を設定するには、TimeOfDay プロパティ を使用します。

使用例

次の例は、**Now** プロパティを使って、現在のシステムの日付と時刻を取得します。

VB

```
Dim ThisMoment As Date
' The following statement calls the Get procedure of the Visual Basic Now property.
ThisMoment = Now
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

Now はクラスではなくモジュールのメンバであるため、**Now** にアクセスするためのオブジェクトを作成する必要はありません。

参照

関連項目

Day 関数 (Visual Basic)
Hour 関数 (Visual Basic)
Minute 関数
Month 関数 (Visual Basic)
Second 関数 (Visual Basic)
Weekday 関数 (Visual Basic)
Year 関数 (Visual Basic)
日付型 (Date) (Visual Basic)
DateTime

ScriptEngine プロパティ

使用中のランタイムを表す文字列 (**String**) を返します。

```
ReadOnly Public Property ScriptEngine As String
```

解説

ScriptEngine プロパティは文字列 "VB" を返します。

アプリケーションがスタンドアロン プログラムとして実行されているときは、スクリプトやホスト アプリケーションで **ScriptEngine** を使用できます。

使用例

次の例では、**ScriptEngine** プロパティおよびその他の関連するプロパティを使用して、現在のランタイム情報を示す文字列を返します。

VB

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

スマート デバイス開発者のためのメモ

このプロパティはサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Globals**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

ScriptEngine はクラスではなくモジュールのメンバであるため、**ScriptEngine** にアクセスするためのオブジェクトを作成する必要はありません。

参照

関連項目

ScriptEngineBuildVersion プロパティ
ScriptEngineMajorVersion プロパティ
ScriptEngineMinorVersion プロパティ

ScriptEngineBuildVersion プロパティ

使用中のランタイムのビルド バージョン番号を含む **Integer** を返します。

```
ReadOnly Public Property ScriptEngineBuildVersion As Integer
```

解説

戻り値は、現在のランタイムの DLL に含まれているバージョン情報と一致します。

アプリケーションがスタンドアロン プログラムとして実行されているときは、スクリプトやホスト アプリケーションで **ScriptEngineBuildVersion** を使用できます。

使用例

次の例では、**ScriptEngineBuildVersion** プロパティおよびその他の関連するプロパティを使用して、現在のランタイム情報を示す文字列を返します。

VB

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

スマート デバイス開発者のためのメモ

このプロパティはサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Globals**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

ScriptEngineBuildVersion はクラスではなくモジュールのメンバであるため、**ScriptEngineBuildVersion** にアクセスするためのオブジェクトを作成する必要はありません。

参照

関連項目

ScriptEngine プロパティ
ScriptEngineMajorVersion プロパティ
ScriptEngineMinorVersion プロパティ

ScriptEngineMajorVersion プロパティ

使用中のランタイムのメジャー バージョン番号を整数 (**Integer**) で返します。

```
ReadOnly Public Property ScriptEngineMajorVersion As Integer
```

解説

戻り値は、現在のランタイムの DLL に含まれているバージョン情報と一致します。

アプリケーションがスタンドアロン プログラムとして実行されているときは、スクリプトやホスト アプリケーションで **ScriptEngineMajorVersion** を使用できます。

使用例

次の例では、**ScriptEngineMajorVersion** プロパティおよびその他の関連するプロパティを使用して、現在のランタイム情報を示す文字列を返します。

VB

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

スマート デバイス開発者のためのメモ

このプロパティはサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Globals**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

ScriptEngineMajorVersion はクラスではなくモジュールのメンバであるため、**ScriptEngineMajorVersion** にアクセスするためのオブジェクトを作成する必要はありません。

参照

関連項目

ScriptEngine プロパティ
ScriptEngineBuildVersion プロパティ
ScriptEngineMinorVersion プロパティ

ScriptEngineMinorVersion プロパティ

使用中のランタイムのマイナ バージョン番号を含む **Integer** を返します。

```
ReadOnly Public Property ScriptEngineMinorVersion As Integer
```

解説

戻り値は、現在のランタイムの DLL に含まれているバージョン情報と一致します。

アプリケーションがスタンドアロン プログラムとして実行されているときは、スクリプトやホスト アプリケーションで **ScriptEngineMinorVersion** を使用できます。

使用例

次の例では、**ScriptEngineMinorVersion** プロパティおよびその他の関連するプロパティを使用して、現在のランタイム情報を示す文字列を返します。

VB

```
Function getRuntimeInfo() As String
    Dim runtime As String = ScriptEngine & " Version "
    runtime &= CStr(ScriptEngineMajorVersion) & "."
    runtime &= CStr(ScriptEngineMinorVersion) & "."
    runtime &= CStr(ScriptEngineBuildVersion)
    ' Return the current runtime information.
    Return runtime
End Function
```

スマート デバイス開発者のためのメモ

このプロパティはサポートされていません。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Globals**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

ScriptEngineMinorVersion はクラスではなくモジュールのメンバであるため、**ScriptEngineMinorVersion** にアクセスするためのオブジェクトを作成する必要はありません。

参照

関連項目

ScriptEngine プロパティ
ScriptEngineBuildVersion プロパティ
ScriptEngineMajorVersion プロパティ

TimeOfDay プロパティ

システムに従った現在の時刻を含む日付型 (**Date**) の値を取得または設定します。

```
Public Property TimeOfDay() As DateTime
```

解説

日付型 (**Date**) には日付コンポーネントが含まれます。システム時刻を返す場合、**TimeOfDay** はこれらのコンポーネントをすべて 1 に設定するため、返される値は 1 年の第 1 日を表します。システム時刻を設定する場合、**TimeOfDay** は日付コンポーネントを無視します。

文字列型 (**String**) としての現在のシステム時刻にアクセスするには、TimeString プロパティ を使用します。

現在のシステム日付を取得または設定するには、Today プロパティ を使用します。

セキュリティに関するメモ：

システム日付やシステム時刻を設定するには、アンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

使用例

次の例は、**TimeOfDay** プロパティを使って、現在のシステムの時刻を求めます。

VB

```
Dim currentTime As Date
currentTime = TimeOfDay
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

TimeOfDay はクラスではなくモジュールのメンバであるため、**TimeOfDay** にアクセスするためのオブジェクトを作成する必要はありません。

スマート デバイス開発者のためのメモ

TimeOfDay プロパティを使用して現在の時刻を取得することはできますが、設定することはできません。

参照

関連項目

Timer プロパティ
日付型 (Date) (Visual Basic)
DateTime

Timer プロパティ

午前 0 時から経過した秒数を表す倍精度浮動小数点数型 (**Double**) の値を返します。

```
ReadOnly Public Property Timer() As Double
```

解説

Timer プロパティは、前回の午前 0 時から経過した秒数とミリ秒数の両方を返します。秒は戻り値の整数部分で、ミリ秒は小数部分です。

使用例

Timer プロパティを使用してプログラムの実行を一時中断する例を、次に示します。中断している間に、他の処理を実行できます。

VB

```
Public Sub waitFiveSeconds()
    If TimeOfDay >= #11:59:55 PM# Then
        MsgBox("The current time is within 5 seconds of midnight" & _
            vbCrLf & "The timer returns to 0.0 at midnight")
        Return
    End If
    Dim start, finish, totalTime As Double
    If (MsgBox("Press Yes to pause for 5 seconds", MsgBoxStyle.YesNo)) _
        = MsgBoxResult.Yes Then
        start = Microsoft.VisualBasic.DateAndTime.Timer
        ' Set end time for 5-second duration.
        finish = start + 5.0
        Do While Microsoft.VisualBasic.DateAndTime.Timer < finish
        ' Do other processing while waiting for 5 seconds to elapse.
        Loop
        totalTime = Microsoft.VisualBasic.DateAndTime.Timer - start
        MsgBox("Paused for " & totalTime & " seconds")
    End If
End Sub
```

Timer プロパティを Microsoft.VisualBasic 名前空間で修飾する必要があることに注意してください。なぜなら、**Timer** は、System.Threading、System.Timers、および System.Windows.Forms の名前空間のクラスにも定義されているからです。

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

Timer はクラスではなくモジュールのメンバであるため、**Timer** にアクセスするためのオブジェクトを作成する必要はありません。

参照

関連項目

Randomize 関数 (Visual Basic)
TimeOfDay プロパティ
Today プロパティ
System
System.Windows.Forms
DateTime
ArgumentException
ArgumentOutOfRangeException

TimeString プロパティ

システムに従った現在の時刻を表す文字列型 (**String**) の値を取得または設定します。

```
Public Property TimeString As String
```

例外

例外の種類	エラー番号	条件
InvalidCastException	5	**TimeString** の値を設定する形式が無効です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、「エラー番号」列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と照らし合わせます)。しかし、可能な限り、このエラー処理は Visual Basic の構造化例外処理の概要 で置き換えてください。

解説

TimeString は、システム時刻を常に "HH:mm:ss" として返します。これは 24 時間形式です。この形式はカルチャが異なっても同じです。つまり、コントロール パネルの [地域のオプション] を変更しても変わりません。

現在のシステム日付を文字列型 (**String**) として取得または設定するには、DateString プロパティ を使用します。

現在のシステムの日付または時刻を、使用しているロケールの形式またはカスタムの形式で取得するには、Format 関数 で Now プロパティ を使用して、定義済みの日付/時刻書式 (Format 関数) または ユーザー定義の日付/時刻書式 (Format 関数) を指定します。次にコード例を示します。

```
MsgBox("The formatted time is " & Format(Now, "hh.mm.ss.fff tt"))
```

文字列型 (**Date**) としての現在のシステム時刻にアクセスするには、TimeOfDay プロパティ を使用します。

セキュリティに関するメモ :

システムの日付または時刻を設定するにはアンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

使用例

次の例は、**TimeString** プロパティを使って、現在のシステムの時刻を表示します。

VB

```
MsgBox("The current time is " & TimeString)
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

TimeString はクラスではなくモジュールのメンバであるため、**TimeString** にアクセスするためのオブジェクトを作成する必要はありません。

スマート デバイス開発者のためのメモ

TimeString プロパティでは、システム時刻を取得できますが設定はできません。

参照

関連項目

Now プロパティ
DateString プロパティ
TimeOfDay プロパティ

日付型 (Date) (Visual Basic)
DateTime

Today プロパティ

システムに従った現在の日付を含む日付型 (**Date**) の値を取得または設定します。

```
Public Property Today() As DateTime
```

解説

日付型 (**Date**) には時刻コンポーネントが含まれます。システム日付を返す場合、**Today** はこれらのコンポーネントをすべて 0 に設定するため、返される値は午前 0 時 (00:00:00) を表します。システム日付を設定する場合、**Today** は時刻コンポーネントを無視します。

文字列型 (**String**) としての現在のシステム日付にアクセスするには、DateString プロパティを使用します。

現在のシステム時刻を取得または設定するには、TimeOfDay プロパティを使用します。

セキュリティに関するメモ：

システム日付やシステム時刻を設定するには、アンマネージ コード アクセス許可が必要です。ただし、部分的に信頼されている状況でこの許可を使用すると、プログラムの実行に影響を及ぼす場合があります。詳細については、「SecurityPermission」および「コード アクセス許可」を参照してください。

使用例

次の例は、**Today** プロパティを使って、現在のシステムの日付を求めます。

VB

```
Dim thisDate As Date
thisDate = Today
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **DateAndTime**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

Today はクラスではなくモジュールのメンバであるため、**Today** にアクセスするためのオブジェクトを作成する必要はありません。

スマート デバイス開発者のためのメモ

Today プロパティを使用して現在のシステム日付を取得することはできますが、設定することはできません。

参照

関連項目

Format 関数
Now プロパティ
日付型 (Date) (Visual Basic)
DateTime

ステートメント (Visual Basic)

ここで紹介するトピックには、Visual Basic の宣言ステートメントと実行ステートメント、および多くのステートメントに適用される重要なリストの表が含まれます。

このセクションの内容

関連するセクション

Visual Basic リファレンス

Visual Basic

ステートメント A ～ E

次の表は、Visual Basic 言語のステートメントの一覧です。

AddHandler	Call	Class	Const
Continue	Declare	Delegate	Dim
Do...Loop	End	Enum	Erase
Error	Event	Exit	

参照

関連項目

ステートメント F ～ P

ステートメント Q ～ Z

その他の技術情報

Visual Basic リファレンス

AddHandler ステートメント

実行時にイベントをイベント ハンドラに関連付けます。

```
AddHandler event, AddressOf eventhandler
```

指定項目

event

処理するイベントの名前。

eventhandler

イベントを処理するプロシージャの名前。

解説

AddHandler ステートメントと **RemoveHandler** ステートメントを使うと、プログラムの実行中にいつでもイベント処理を開始および停止できます。

eventhandler プロシージャのシグネチャは、*event* イベントのシグネチャと一致する必要があります。

Handles キーワードと **AddHandler** ステートメントは、どちらも特定のイベントを特定のプロシージャで処理するよう指定する方法ですが、いくつか相違点があります。**AddHandler** ステートメントは、実行時にイベントをプロシージャに関連付けます。特定のイベントを処理するプロシージャを定義するときには、**Handles** キーワードを使用します。詳細については、「Handles」を参照してください。

カスタム イベントの場合は、**AddHandler** ステートメントはイベントの **AddHandler** アクセサを呼び出します。カスタム イベントの詳細については、「Event ステートメント」を参照してください。

使用例

VB

```
Sub TestEvents()
    Dim Obj As New Class1
    ' Associate an event handler with an event.
    AddHandler Obj.Ev_Event, AddressOf EventHandler
    ' Call the method to raise the event.
    Obj.CauseSomeEvent()
    ' Stop handling events.
    RemoveHandler Obj.Ev_Event, AddressOf EventHandler
    ' This event will not be handled.
    Obj.CauseSomeEvent()
End Sub

Sub EventHandler()
    ' Handle the event.
    MsgBox("EventHandler caught event.")
End Sub

Public Class Class1
    ' Declare an event.
    Public Event Ev_Event()
    Sub CauseSomeEvent()
        ' Raise an event.
        RaiseEvent Ev_Event()
    End Sub
End Class
```

参照

関連項目

RemoveHandler ステートメント

Handles

Event ステートメント

概念

イベントとイベント ハンドラ
AddHandler と RemoveHandler

Call ステートメント (Visual Basic)

Function 、**Sub**、またはダイナミック リンク ライブラリ (DLL) プロシージャに制御を渡すフロー制御ステートメントです。

```
[ Call ] procedureName [ (argumentList) ]
```

指定項目

procedureName

必ず指定します。呼び出すプロシージャの名前を指定します。

argumentList

省略可能です。プロシージャを呼び出すときにプロシージャに渡す引数を表す、変数または式のリストを指定します。複数の引数を指定するときは、コンマ (,) で区切ります。*argumentList* を指定する場合は、かっこで囲む必要があります。

解説

通常は、値を返さないプロシージャを呼び出すときは **Call** ステートメントを使用します。プロシージャが値を返す場合は、**Call** ステートメントを使用すると戻り値が破棄されます。

プロシージャを呼び出すときに **Call** ステートメントを使用する必要はありません。ただし、そうするとコードの読みやすさが向上します。

使用例

Call ステートメントが **Sub** プロシージャ、組み込み関数、およびダイナミック リンク ライブラリ (DLL) プロシージャに対して、どのように制御を渡すかを説明するコード例は、次のとおりです。

VB

```
        ' (1) Call a Sub procedure.
        Call printToDebugWindow("Hello World")
...
    ' The above statement passes control to the following Sub procedure.
    Sub printToDebugWindow(ByVal anyString As String)
        Debug.WriteLine(anyString)
    End Sub
```

VB

```
' (2) Call a Visual Basic run-time function (Shell), discard the return value.
Call Shell("C:\WINNT\system32\calc.exe", AppWinStyle.NormalFocus)
' The preceding path is for Windows 2000;
' The Windows XP path is C:\Windows\system32\calc.exe.
```

VB

```
' (3) Call a Microsoft Windows DLL procedure. The Declare statement
' must be Private in a class, not in a module.
Private Declare Sub MessageBeep Lib "User32" (ByVal N As Integer)
Sub callBeepDll()
    Call MessageBeep(-1)
End Sub
```

参照

関連項目

Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)
Declare ステートメント

Class ステートメント (Visual Basic)

クラスの名前を宣言し、そのクラスを構成する変数、プロパティ、イベント、およびプロシージャの定義を示します。

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ MustInherit | NotInheritable ] [ Parti
al ] _
Class name [ ( Of typelist ) ]
    [ Inherits classname ]
    [ Implements interfacenames ]
    [ statements ]
End Class
```

指定項目

attributelist

省略可能です。「属性リスト」を参照してください。

accessmodifier

省略可能です。次のいずれかになります。

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Visual Basic でのアクセス レベル を参照してください。

Shadows

省略可能です。「Shadows」を参照してください。

MustInherit

省略可能です。MustInherit を参照してください。

NotInheritable

省略可能です。NotInheritable を参照してください。

Partial

省略可能です。クラスの部分定義であることを示します。Partial (Visual Basic) を参照してください。

name

必ず指定します。このクラスの名前です。宣言された要素の名前 を参照してください。

Of

省略可能です。これがジェネリック クラスであることを指定します。

typelist

Of キーワードを使用する場合は必ず指定します。このクラスの型パラメータの一覧を指定します。「型リスト」を参照してください。

Inherits

省略可能です。このクラスが他のクラスのメンバを継承することを示します。Inherits ステートメント を参照してください。

classname

Inherits ステートメントを使用する場合は必ず指定します。このクラスの派生元となるクラスの名前です。

Implements

省略可能です。このクラスが、1 つ以上のインターフェイスのメンバを実装していることを示します。Implements ステートメント を参照してください。

interfacenames

Implements ステートメントを使用する場合は必ず指定します。このクラスが実装するインターフェイスの名前を指定します。

statements

省略可能です。このクラスのメンバを定義するステートメントを指定します。

End Class

必ず指定します。**Class** 定義を終了します。

解説

Class ステートメントは新しいデータ型を定義します。クラスはオブジェクト指向プログラミング (OOP: Object-Oriented Programming) の基本的なビルド ブロックです。詳細については、「Visual Basic におけるオブジェクト指向プログラミング」および「クラス : オブジェクトの設計図」を参照してください。

Class は、名前空間またはモジュール レベルでのみ使用できます。つまり、クラスの宣言コンテキストは、ソース ファイル、名前空間、クラス、構造体、モジュール、またはインターフェイスのいずれかである必要があり、プロシージャまたはブロックでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

クラスの各インスタンスは、他のインスタンスからは独立した有効期間を持ちます。インスタンスの有効期間は、New (Visual Basic) 句または CreateObject 関数 (Visual Basic) などの関数で作成されたときに開始します。インスタンスを参照している変数がすべて Nothing (Visual Basic) に設定されるか、他のクラスのインスタンスに設定されると、インスタンスの有効期間は終了します。

既定では、クラスのアクセス レベルは Friend (Visual Basic) です。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。詳細については、「Visual Basic でのアクセス レベル」を参照してください。

規則

- **入れ子** クラス内に別のクラスを定義できます。外側のクラスは包含クラス、内側のクラスは入れ子クラスと呼ばれます。
- **継承** クラスが Inherits ステートメントを使用している場合は、1 つの基本クラスまたはインターフェイスのみを指定できます。複数の要素を継承することはできません。

  自分よりもアクセス レベルの厳しいクラスは継承できません。たとえば、**Public** クラスは **Friend** クラスを継承できません。

  中に入れ子になったクラスは継承できません。
- **実装** クラスが Implements ステートメントを使用している場合は、*interfacenames* に指定したすべてのインターフェイスで定義されているメンバをすべて実装する必要があります。ただし、基本クラスのメンバを再実装する必要はありません。詳細については、「Implements (Visual Basic)」の「再実装」を参照してください。
- **既定のプロパティ。** クラス内の最大 1 つのプロパティを既定のプロパティとして指定できます。詳細については、「既定のプロパティ」を参照してください。

動作

- **アクセス レベル** クラス内では、それぞれのメンバにアクセス レベルを宣言できます。変数と定数を除くクラス メンバは既定で Public (Visual Basic) アクセスになります。変数と定数は既定で Private (Visual Basic) アクセスになります。クラスのアクセス レベルの方がメンバのアクセス レベルよりも厳しいときは、クラスのアクセス レベルが優先されます。
- **スコープ** クラスのスコープは、そのクラスを含んでいる名前空間、クラス、構造体、またはモジュールになります。

  すべてのクラス メンバのスコープは、クラス全体になります。

  有効期間 Visual Basic は静的クラスをサポートしません。静的クラスと同等の機能はモジュールで実現されます。詳細については、「Module ステートメント」を参照してください。

  クラス メンバの有効期間は、宣言された方法と場所によって左右されます。詳細については、「Visual Basic における有効期間」を参照してください。
- **修飾** クラス外部のコードがメンバにアクセスするときには、メンバ名をクラス名で修飾する必要があります。

  入れ子クラスの内部のコードがプログラミング要素に対する非修飾参照を行った場合、Visual Basic は目的の要素をまず入れ子クラス内で検索し、次にそのコンテナ クラス内で検索し、同様にして一番外側のコンテナ要素まで検索します。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」を参照してください。

クラスとモジュール

これらの要素はよく似ていますが、重要な相違点がいくつかあります。

- **用語** 以前のバージョンの Visual Basic には、クラス モジュール (.cls ファイル) と標準モジュール (.bas ファイル) という 2 種類のモジュールがありました。現在のバージョンでは、これらをそれぞれクラスとモジュールと呼びます。
- **共有メンバ** クラスのメンバを共有メンバにするかインスタンス メンバにするかを制御できます。
- **オブジェクト指向** クラスはオブジェクト指向ですが、モジュールはオブジェクト指向ではありません。クラスについては 1 つ以上のインスタンスを作成できます。詳細については、「クラスとモジュール」を参照してください。

使用例

次の例では、**Class** ステートメントを使用してクラスといくつかのメンバを定義しています。

VB

```
Class bankAccount
    Shared interestRate As Decimal
    Private accountNumber As String
    Private accountBalance As Decimal
    Public holdOnAccount As Boolean = False

    Public ReadOnly Property balance() As Decimal
        Get
            Return accountBalance
        End Get
    End Property

    Public Sub postInterest()
        accountBalance = accountBalance * (1 + interestRate)
    End Sub

    Public Sub postDeposit(ByVal amountIn As Decimal)
        accountBalance = accountBalance + amountIn
    End Sub

    Public Sub postWithdrawal(ByVal amountOut As Decimal)
        accountBalance = accountBalance - amountOut
    End Sub
End Class
```

参照

処理手順

方法 : ジェネリック クラスを使用する

関連項目

Interface ステートメント (Visual Basic)

Module ステートメント

Property ステートメント

概念

構造体とクラス

オブジェクトの有効期間 : オブジェクトの作成と破棄

Visual Basic におけるジェネリック型

その他の技術情報

クラスについて

Const ステートメント (Visual Basic)

1 つ以上の定数を宣言および定義します。

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ]
Const constantlist
```

指定項目

attributelist

省略可能です。このステートメントで宣言されるすべての定数に適用される属性のリストです。属性リストが山かっこ ("<" および ">") で囲まれている点に注意してください。

accessmodifier

省略可能です。これらの定数にアクセスできるコードを指定するために使います。Public (Visual Basic)、Protected (Visual Basic)、Friend (Visual Basic)、**Protected Friend**、または Private (Visual Basic) を指定できます。

Shadows

省略可能です。基本クラス内のプログラミング要素を宣言し直して隠ぺいする場合に使用します。「Shadows」を参照してください。

constantlist

必ず指定します。このステートメントで宣言する定数のリストです。

constant [, *constant* ...]

constant の構文と指定項目は次のとおりです。

constantname [As *datatype*] = *initializer*

指定項目	説明
constantname	必ず指定します。定数の名前を指定します。宣言された要素の名前 を参照してください。
datatype	**Option Strict** が **On** の場合は、必ず指定します。定数のデータ型を指定します。
initializer	必ず指定します。コンパイル時に評価して、定数に代入される式です。

解説

決して変わらない値がアプリケーションにある場合は、名前付き定数を定義してリテラル値の代わりに使用できます。名前は値よりも覚えるのが簡単です。定数を一度定義すると、コード内の多くの場所でそれを使用できます。その後のバージョンで値を定義し直すことが必要になった場合も、**Const** ステートメントを変更するだけで済みます。

Const は、モジュール レベルまたはプロシージャ レベルでのみ使用できます。つまり、変数の宣言コンテキストは、クラス、構造体、モジュール、プロシージャ、またはブロックであることが必要で、ソース ファイル、名前空間、インターフェイスでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

ローカル定数 (プロシージャ内) のアクセス レベルは、既定で public になります。このような定数にはアクセス修飾子を指定できません。クラスおよびモジュールのメンバ定数 (プロシージャ外) のアクセス レベルは、既定で private になります。また、構造体のメンバ定数の既定のアクセスレベルは public です。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。

規則

- **宣言コンテキスト**モジュール レベルで宣言された、プロシージャの外側にある定数をメンバ定数といいます。つまり、それを宣言しているクラス、構造体、またはモジュールのメンバです。

  プロシージャ レベルで宣言された定数をローカル定数といいます。つまり、それを宣言しているプロシージャまはたブロックに対してローカルという意味です。

- **属性**属性を適用できるのはメンバ定数だけで、ローカル定数には適用できません。属性はアセンブリのメタデータに情報を提供しますが、これはローカル定数などの一時的な格納領域には意味がありません。

- **修飾子** 既定では、すべての定数が **Shared**、**Static**、そして **ReadOnly** になります。定数を宣言するとき、これらのどのキーワードも指定できません。

  プロシージャ レベルでは、ローカル定数の宣言に **Shadows** またはアクセス修飾子を指定できません。

- **複数の定数** 同じ宣言ステートメントに複数の定数を宣言するには、各定数ごとに *constantname* の指定項目を指定します。複数の定数を指定するときは、コンマ (,) で区切ります。

データ型のルール

- **データ型** **Const** ステートメントでは、変数のデータ型を宣言できます。任意のデータ型、または列挙型の名前を指定できます。
- **既定の型** *datatype* を指定しない場合、定数のデータ型は *initializer* になります。*datatype* と *initializer* の両方を指定する場合は、*initializer* のデータ型を *datatype* と互換性のあるものにする必要があります。*datatype* と *initializer* のどちらも指定しない場合、既定のデータ型は **Object** になります。
- **異なるデータ型。** 複数の定数に異なるデータ型を指定するには、宣言する各変数に対して **As** 句を別々に指定します。ただし、共通の **As** 句を使って複数の定数を同じ型にすることはできません。
- **初期化** すべての定数の値を、*constantlist* で初期化する必要があります。定数に代入するための式を指定するには、*initializer* を使用します。式にはリテラル、他の定義済みの定数、および定義済みの列挙体のメンバを自由に組み合わせることができます。算術演算子と論理演算子を使って、各要素を組み合わせることもできます。

  initializer では、変数または関数は使用できません。ただし、**CByte** や **CShort** などの変換キーワードは使用できます。また、**AscW** も使用できます。この場合には、定数の **String** 引数、または **Char** 引数を指定して、コンパイル時に計算できるようにします。

動作

- **スコープ** ローカル定数には、そのプロシージャまたはブロックの内部からのみアクセスできます。メンバ定数にはそのクラス、構造体、またはモジュール内の任意の場所からアクセスできます。
- **修飾** クラス、構造体、またはモジュールの外部のコードで使用する場合は、その名前でメンバ定数の名前を修飾する必要があります。プロシージャやブロックの外部のコードは、その内部にあるローカル定数を参照できません。

使用例

Const ステートメントを使って、リテラル値の代わりに使用する定数を宣言するコード例は、次のとおりです。

VB

```
' The following statements declare constants.
Const maximum As Long = 459
Public Const helpString As String = "HELP"
Private Const startValue As Integer = 5
```

データ型が **Object** である定数を定義すると、Visual Basic コンパイラはそれを **Object** 型ではなく、*initializer* 型にします。次の例では、定数 `naturalLogBase` のランタイム型は **Decimal** です。

VB

```
Const naturalLogBase As Object = CDec(2.7182818284)
MsgBox("Run-time type of constant naturalLogBase is " & _
    naturalLogBase.GetType.ToString())
```

この例では、**CStr** を使用して Type を **String** に変換できないため、GetType 演算子 から返された **Type** オブジェクトの ToString メソッドが使用されます。

参照

関連項目

Enum ステートメント (Visual Basic)
#Const ディレクティブ
Dim ステートメント (Visual Basic)
ReDim ステートメント (Visual Basic)
データ型変換関数
Asc 関数、AscW 関数

概念

暗黙の型変換と明示的な型変換
組み込み定数と組み込み列挙型
その他の技術情報
Visual Basic の定数と列挙体

Continue ステートメント (Visual Basic)

制御をループの次の反復処理に直ちに移します。

```
Continue { Do | For | While }
```

解説

Do、**For**、または **While** ループの内側から、そのループの次の反復処理に制御を移すことができます。制御は直ちにループ条件のテストに移動します。つまり、**For** ステートメントか **While** ステートメント、または **Until** 句か **While** 句を含む **Do** ステートメントか **Loop** ステートメントに移動します。

Continue は制御の移動を許可するループの任意の場所に定義できます。制御の移動を許可する規則は、GoTo ステートメントの場合と同じです。

たとえば、ループ全体が **Try** ブロック、**Catch** ブロック、または **Finally** ブロックの内部にある場合は、**Continue** を使用してループの外部に制御を移動できます。一方、ループ内に **Try**...**End Try** 構造がある場合は、**Continue** を使用して制御を **Finally** ブロックの外部に移動できず、**Try**...**End Try** 構造の外部に制御を完全に移動する場合に限り、**Try** ブロックまたは **Catch** ブロックの外部に移動できます。

Do ループの内部に別の **Do** ループがあるなど、同じ種類のループが入れ子になっている場合、**Continue Do** ステートメントは自分が定義されている内側の **Do** ループの次の反復処理にスキップします。**Continue** を使用して、同じ種類の外側のループの次の反復処理にスキップすることはできません。

For ループの内部に **Do** ループがあるなど、種類の異なるループが入れ子になっている場合、**Continue Do** または **Continue For** を使用することによって、どちらのループの次の反復処理にもスキップできます。

使用例

次のコード例は **Continue While** ステートメントを使用して、除数が 0 の場合に配列の次の列にスキップします。**Continue While** は **For** ループの内部にあります。制御は **For** ループを含む内側の **While** ループの次の反復処理である `While col < lastcol` ステートメントに移ります。

VB

```
Dim row, col As Integer
Dim lastrow As Integer = 6
Dim lastcol As Integer = 10
Dim a(,) As Double = New Double(lastrow, lastcol) {}
Dim b(7) As Double
row = -1
While row < lastrow
    row += 1
    col = -1
    While col < lastcol
        col += 1
        a(row, col) = 0
        For i As Integer = 0 To b.GetUpperBound(0)
            If b(i) = col Then
                Continue While
            Else
                a(row, col) += (row + b(i)) / (col - b(i))
            End If
        Next i
    End While
End While
```

参照

処理手順

方法 : ループの次の反復処理にスキップする

関連項目

Do...Loop ステートメント (Visual Basic)

For...Next ステートメント (Visual Basic)

While...End While ステートメント (Visual Basic)

Try...Catch...Finally ステートメント (Visual Basic)

Declare ステートメント

外部ファイル内で実装されているプロシージャへの参照を宣言します。

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ Overloads ] _
Declare [ charsetmodifier ] [ Sub ] name Lib "libname" _
[ Alias "aliasname" ] [ ([ parameterlist ]) ]
' -or-
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ Overloads ] _
Declare [ charsetmodifier ] [ Function ] name Lib "libname" _
[ Alias "aliasname" ] [ ([ parameterlist ]) ] [ As returntype ]
```

指定項目

attributelist

省略可能です。「属性リスト」を参照してください。

accessmodifier

省略可能です。次のいずれかになります。

- Public
- Protected
- Friend
- Private
- **Protected Friend**

「Visual Basic でのアクセス レベル」を参照してください。

Shadows

省略可能です。「Shadows」を参照してください。

charsetmodifier

省略可能です。文字セットとファイル検索情報を指定します。次のいずれかになります。

- Ansi (既定値)
- Unicode (Visual Basic)
- Auto

Sub

省略できます。ただし、**Sub** か **Function** のどちらかを指定する必要があります。外部プロシージャが戻り値を返さないことを表します。

Function

省略できます。ただし、**Sub** か **Function** のどちらかを指定する必要があります。外部プロシージャが戻り値を返すことを表します。

name

必ず指定します。この外部参照の名前です。詳細については、「宣言された要素の名前」を参照してください。

Lib

必ず指定します。**Lib** 句を使用して、宣言するプロシージャが含まれているファイルを指定します。

libname

必ず指定します。宣言するプロシージャが含まれているファイルの名前を指定します。

Alias

省略可能です。*name* で指定した名前では、宣言するプロシージャをそのファイル内で特定できないことを表します。識別子は *aliasname*

で指定します。

aliasname

Alias キーワードを使用する場合は必ず指定します。プロシージャを識別する文字列を、次のどちらかの方法で指定します。

ファイル内でのプロシージャのエントリ ポイント名を引用符 ("") で囲んで指定します。

または

シャープ記号 (#) の後ろに、ファイル内でのプロシージャのエントリ ポイントの序数を表す整数を指定します。

parameterlist

プロシージャがパラメータを受け取る場合は、必ず指定します。「パラメータの一覧」を参照してください。

returntype

Function が指定されていて、**Option Strict** が **On** の場合は、必ず指定します。プロシージャによって返される値のデータ型を指定します。

解説

場合によっては、プロジェクト外部のファイル (DLL やコード リソースなど) 内で定義されたプロシージャを呼び出すことがあります。このとき、Visual Basic コンパイラは、目的のプロシージャを正しく呼び出すために必要な情報 (プロシージャの定義場所、識別方法、呼び出しシーケンスや戻り値の型、使用される文字セットなど) を知りません。**Declare** ステートメントは、外部プロシージャへの参照を作成し、この必須情報を提供します。

Declare は、モジュール レベルでのみ使用できます。つまり、外部参照の宣言コンテキストは、クラス、構造体、またはモジュールであることが必要で、ソース ファイル、名前空間、インターフェイス、プロシージャ、ブロックでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

外部参照は、既定で Public (Visual Basic) アクセスになります。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。

規則

- **属性** 外部参照には属性を適用できます。適用した属性はプロジェクト内でのみ有効であり、外部ファイル内では無効です。
- **修飾子** 外部プロシージャは暗黙的に Shared (Visual Basic) になります。外部参照を宣言するときに **Shared** キーワードを使用することはできません。また、この共有ステータスは変更できません。

  外部プロシージャは、オーバーライドに参加したり、インターフェイス メンバを実装したり、イベントを処理したりすることはできません。そのため、**Declare** ステートメント内では **Overrides**、**Overridable**、**NotOverridable**、**MustOverride**、**Implements**、または **Handles** キーワードを使用できません。

- **外部プロシージャ名** この外部参照の名前 (*name*) を、外部ファイル内でのプロシージャのエントリ ポイント名 (*aliasname*) と同じにする必要はありません。エントリ ポイント名を指定するには **Alias** 句を使用します。この方法は、目的の外部プロシージャが Visual Basic の予約済みの修飾子や同じスコープ内の変数、プロシージャ、その他のプログラミング要素と同じ名前を持っている場合に役立ちます。

メモ：
ほとんどの DLL 内のエントリ ポイント名は、大文字と小文字を区別します。

- **外部プロシージャ番号** 別の方法として、**Alias** 句を使用して、外部ファイルのエクスポート テーブル内でのエントリ ポイントの序数を指定することもできます。この方法を使用する場合は、*aliasname* の先頭をシャープ記号 (#) にします。この方法は、外部プロシージャ名のいずれかの文字が Visual Basic 内で許可されていない場合や、外部ファイルがプロシージャを名前なしでエクスポートする場合に役立ちます。

データ型のルール

- **パラメータのデータ型** **Option Strict** が **On** の場合は、*parameterlist* 内の各パラメータのデータ型を指定する必要があります。任意のデータ型、または列挙体、構造体、クラス、インターフェイスの名前を指定できます。各パラメータに渡される引数のデータ型を指定するには、*parameterlist* 内で **As** 句を使用します。

メモ：

外部プロシージャが .NET Framework 用に記述されていない場合は、データ型の対応に注意する必要があります。たとえば、**Integer** 型 (Visual Basic 6.0 では 16 ビット) のパラメータを持つ Visual Basic 6.0 プロシージャへの外部参照を宣言する場合は、その **Declare** ステートメント内で、対応する引数を **Short** 型として指定する必要があります (これが Visual Basic の 16 ビットの整数型に相当します)。同様に、**Long** は Visual Basic 6.0 では異なるデータ幅を持ち、**Date** は実装方法が異なります。

- **戻り値の型** 外部プロシージャが **Function** で、**Option Strict** が **On** の場合は、呼び出し元コードに返される値のデータ型を指定する必要があります。任意のデータ型、または列挙体、構造体、クラス、インターフェイスの名前を指定できます。

メモ：

Visual Basic コンパイラは、指定したデータ型が外部プロシージャのものと互換性を持つかどうかを検証しません。一致しない場合は、実行時に共通言語ランタイムが MarshalDirectiveException 例外を生成します。

- **既定のデータ型** **Option Strict** が **Off** で、*parameterlist* 内でパラメータのデータ型を指定していない場合は、Visual Basic コンパイラは対応する引数を オブジェクト型 (Object)に変換します。同様に、*returntype* を指定していない場合は、コンパイラは戻り値のデータ型を **Object** にします。

メモ：

使用しようとする外部プロシージャが別のプラットフォーム上で記述されている可能性もあるので、データ型を憶測で扱ったり、データ型を既定のままにしたりするのは危険です。すべてのパラメータと戻り値 (ある場合) に対してデータ型を指定した方が安全です。こうすると、コードも読みやすくなります。

動作

- **スコープ** 外部参照のスコープは、そのクラス、構造体、またはモジュール全体になります。
- **有効期間** 外部参照の有効期間は、それが宣言されているクラス、構造体、またはモジュールと同じになります。
- **外部プロシージャの呼び出し** 外部プロシージャの呼び出し方は、**Function** または **Sub** プロシージャを呼び出すときと同じです。つまり、そのプロシージャが値を返す場合は式の中で使用し、値を返さない場合は Call ステートメント (Visual Basic) 内に指定します。

  外部プロシージャに引数を渡すときは、**Declare** ステートメントの *parameterlist* で指定したとおりに指定します。外部ファイルの中でパラメータがどのように宣言されているかを気にする必要はありません。同様に、戻り値がある場合は、**Declare** ステートメントの *returntype* で指定したとおりの方法で使用します。

- **文字セット** *charsetmodifier* では、外部プロシージャを呼び出すときに Visual Basic が文字列をどのようにマーシャリングするかを指定できます。**Ansi** 修飾子は、すべての文字列を ANSI 値にマーシャリングすることを表します。**Unicode** 修飾子は、すべての文字列を Unicode 値にマーシャリングすることを表します。**Auto** 修飾子は、文字列を外部参照の *name* または *aliasname* (指定した場合) に基づく .NET Framework の規則に従ってマーシャリングすることを表します。既定値は、**Ansi** です。

  さらに *charsetmodifier* は、Visual Basic が外部ファイル内で外部プロシージャを検索する方法も表します。**Ansi** および **Unicode** を指定した場合は、Visual Basic は検索中に名前を修飾しません。**Auto** を指定した場合は、Visual Basic は実行時プラットフォームの基本文字セットを調べて、外部プロシージャの名前を次のように修飾します。

  - Windows 95、Windows 98、Windows Millennium Edition などの ANSI プラットフォームでは、まず名前修飾なしで外部プロシージャを検索します。見つからなかった場合は、外部プロシージャ名の末尾に "A" を付けて再度検索します。
  - Windows NT、Windows 2000、Windows XP などの Unicode プラットフォームでは、まず名前修飾なしで外部プロシージャを検索します。見つからなかった場合は、外部プロシージャ名の末尾に "W" を付けて再度検索します。

- **しくみ** Visual Basic は、.NET Framework のプラットフォーム呼び出し (PInvoke) 機構を使用して外部プロシージャの解決とアクセスを行います。**Declare** ステートメントと DllImportAttribute クラスは両方ともこの機構を自動的に使用するので、PInvoke についての知識は必要ありません。詳細については、「チュートリアル : Windows API の呼び出し」を参照してください。

セキュリティに関するメモ：

共通言語ランタイム (CLR) の外部で実行される外部プロシージャはアンマネージ コードと呼ばれます。たとえば Win32 API 関数や COM メソッドなどが該当しますが、このようなプロシージャを呼び出すとセキュリティ上のリスクが考えられます。詳細については、「アンマネージ コード」を参照してください。

使用例

次の例では、現在のユーザーの名前を返す **Function** プロシージャへの外部参照を宣言します。その後、外部プロシージャ `GetUserNameA` を `getUser` プロシージャの一部として呼び出します。

VB

```
Declare Function getUserName Lib "advapi32.dll" Alias "GetUserNameA" _
    (ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
Sub getUser()
    Dim buffer As String = New String(CChar(" "), 25)
    Dim retVal As Integer = getUserName(buffer, 25)
    Dim userName As String = Strings.Left(buffer, InStr(buffer, Chr(0)) - 1)
    MsgBox(userName)
End Sub
```

DllImportAttribute を使うと、アンマネージ コード内の関数を別の方法で使うことができます。インポートした関数を **Declare** ステートメントを使わずに宣言する例を次に示します。

VB

```
' Add an Imports statement at the top of the class, structure, or
' module that uses the DllImport attribute.
Imports System.Runtime.InteropServices
```

VB

```
<DllImportAttribute("kernel32.dll", EntryPoint:="MoveFileW", _
    SetLastError:=True, CharSet:=CharSet.Unicode, _
    ExactSpelling:=True, _
    CallingConvention:=CallingConvention.StdCall)> _
Public Shared Function moveFile(ByVal src As String, _
  ByVal dst As String) As Boolean
    ' This function copies a file from the path src to the path dst.
    ' Leave this function empty. The DLLImport attribute forces calls
    ' to moveFile to be forwarded to MoveFileW in KERNEL32.DLL.
End Function
```

参照

処理手順

チュートリアル : Windows API の呼び出し

関連項目

Imports ステートメント
AddressOf 演算子
Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)
パラメータの一覧
Call ステートメント (Visual Basic)
LastDllError プロパティ (Err オブジェクト)

概念

データ型の変更点 (Visual Basic 6.0 ユーザー向け)
整数型 (Integer) (Visual Basic 6.0 ユーザー向け)

Delegate ステートメント

デリゲートを宣言します。デリゲートとは、型の **Shared** メソッド、またはオブジェクトのインスタンス メソッドを参照する参照型です。パラメータの型と戻り値の型が一致するプロシージャを使って、このデリゲート クラスのインスタンスを作成できます。デリゲート インスタンスの作成後は、そのインスタンスを経由してプロシージャを起動できます。

```
[ <attrlist> ] [ accessmodifier ] _
[ Shadows ] Delegate [ Sub | Function ] name [( Of typeparamlist )] [([ parameterlist ])] [
As type ]
```

指定項目

attrlist

省略可能です。このデリゲートに適用される属性の一覧を指定します。複数の属性を指定するときは、コンマ (,) で区切ります。属性リストは、山かっこ (< および >) で囲む必要があります。

accessmodifier

省略可能です。どのようなコードからデリゲートにアクセスできるのかを指定します。次のいずれかの値を指定します。

- Publicデリゲートを宣言した要素にアクセスできるすべてのコードがアクセスできます。
- Protectedデリゲートのクラス内のコード、または派生クラスのみがアクセスできます。
- Friendデリゲートと同じアセンブリ内のコードだけがアクセスできます。
- Privateデリゲートを宣言した要素内のコードだけがアクセスできます。

Protected Friend と指定すると、デリゲートのクラス、その派生クラス、または同じアセンブリ内のコードからのみアクセスできます。

Shadows

省略可能です。このデリゲートが、基本クラスにある、同じ名前を持つプログラミング要素、またはオーバーロードされる要素を宣言し直すことを示します。宣言された要素は、他の任意の種類の要素でシャドウできます。

シャドウされた要素は、その要素をシャドウする派生クラスからは使用できません。ただし、シャドウする要素がアクセスできない要素の場合は例外です。たとえば、**Private** 要素が基本クラスの要素をシャドウすると、**Private** 要素へのアクセス許可を持たないコードは、基本クラスにアクセスします。

Sub

省略できます。ただし、**Sub** か **Function** のどちらかを指定する必要があります。指定するプロシージャが、値を返さないデリゲートの **Sub** プロシージャとして宣言されることを示します。

Function

省略できます。ただし、**Sub** か **Function** のどちらかを指定する必要があります。指定するプロシージャが、値を返すデリゲートの **Function** プロシージャとして宣言されることを示します。

name

必ず指定します。デリゲート型の名前を指定します。デリゲート型の標準的な名前付け規則に従って名前を付けます。

typeparamlist

省略可能です。このデリゲートの型パラメータを一覧表示します。複数の型パラメータがある場合は、コンマ (,) で区切られます。型リストはかっこで囲み、冒頭に **Of** キーワードを付けます。

parameterlist

省略可能です。プロシージャが呼び出されたときに渡されるパラメータのリストです。パラメータの一覧 はかっこで囲む必要があります。

type

Function プロシージャを指定する場合は、必ず指定します。戻り値のデータ型を指定します。

解説

Delegate ステートメントでは、デリゲート クラスのパラメータの型と戻り値の型を定義します。パラメータの型と戻り値の型が一致するプロシー

ジャを使って、このデリゲート クラスのインスタンスを作成できます。デリゲート インスタンスの作成後は、デリゲートの **Invoke** メソッドを呼び出すことで、そのインスタンスを経由してプロシージャを起動できます。

デリゲートは、名前空間、モジュール、クラス、構造体レベルで宣言できますが、プロシージャ内では宣言できません。

各デリゲート クラスでは、オブジェクト メソッドの仕様を渡すコンストラクタを定義します。デリゲート コンストラクタに渡す引数は、次の形式で表現する必要があります。

AddressOf [*expression*.]*methodname*

コンパイル時の *expression* の型は、シグネチャがデリゲート クラスのシグネチャと同じで、指定された名前のメソッドを持つクラスまたはインターフェイスである必要があります。*methodname* は、共有メソッドまたはインスタンス メソッドのいずれかにできます。クラスの既定メソッドに対してデリゲートを作成する場合も、*methodname* は省略できません。

使用例

次の例では、**Delegate** ステートメントを使って、2 つの数字を操作して数字を返すデリゲートを宣言します。`DelegateTest` メソッドはこのデリゲートの型のインスタンスを受け取り、デリゲートを使って 1 組の数字を操作します。

VB

```
Delegate Function MathOperator( _
    ByVal x As Double, _
    ByVal y As Double _
) As Double

Function AddNumbers( _
    ByVal x As Double, _
    ByVal y As Double _
) As Double
    Return x + y
End Function

Function SubtractNumbers( _
    ByVal x As Double, _
    ByVal y As Double _
) As Double
    Return x - y
End Function

Sub DelegateTest( _
    ByVal x As Double, _
    ByVal op As MathOperator, _
    ByVal y As Double _
)
    Dim ret As Double
    ret = op.Invoke(x, y) ' Call the method.
    MsgBox(ret)
End Sub

Protected Sub Test()
    DelegateTest(5, AddressOf AddNumbers, 3)
    DelegateTest(9, AddressOf SubtractNumbers, 3)
End Sub
```

参照

処理手順

方法 : ジェネリック クラスを使用する

関連項目

AddressOf 演算子

Of

概念

Visual Basic におけるジェネリック型

その他の技術情報

Visual Basic でのデリゲート

Dim ステートメント (Visual Basic)

1 つまたは複数の変数を宣言し、記憶域を割り当てます。

```
[ <attributelist> ] [ accessmodifier ] [[ Shared ] [ Shadows ] | [ Static ]] [ ReadOnly ]
Dim [ WithEvents ] variablelist
```

指定項目

attributelist

省略可能です。「属性リスト」を参照してください。

accessmodifier

省略可能です。次のいずれかになります。

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Visual Basic でのアクセス レベル を参照してください。

Shared

省略可能です。「Shared (Visual Basic)」を参照してください。

Shadows

省略可能です。「Shadows」を参照してください。

Static

省略可能です。「Static (Visual Basic)」を参照してください。

ReadOnly

省略可能です。「ReadOnly (Visual Basic)」を参照してください。

WithEvents

省略可能です。この変数が、イベントを発生させるクラスのインスタンスを参照しているオブジェクト変数であることを指定します。「WithEvents」を参照してください。

variablelist

必ず指定します。このステートメントで宣言する変数のリストです。

```
variable [ , variable ... ]
```

variable の構文と指定項目は次のとおりです。

```
variablename[ ( [ boundslist ] ) ] [ As [ New ] datatype ] [ = initializer ]
```

指定項目	説明
variablename	必ず指定します。変数の名前を指定します。宣言された要素の名前 を参照してください。
boundslist	省略可能です。配列変数の各次元の範囲を示すリストを指定します。
New	省略可能です。この **Dim** ステートメントが実行されるときに、クラスの新しいインスタンスを作成します。
datatype	**Option Strict** が **On** の場合は、必ず指定します。変数のデータ型を指定します。

initializer	**New** が指定されていない場合は省略可能です。変数の作成時に評価され、変数に割り当てられる式を指定します。

解説

プログラム内で変数を使用するときには、変数のデータ型や、その変数にアクセスできるコードなどの情報を Visual Basic コンパイラに伝えるために、個々の変数を宣言する必要があります。次の例では、**Integer** 値を保持する変数を宣言しています。

```
Dim numberOfStudents As Integer
```

Dim は、モジュール レベルまたはプロシージャ レベルでのみ使用できます。つまり、変数の宣言コンテキストは、クラス、構造体、モジュール、プロシージャ、またはブロックであることが必要で、ソース ファイル、名前空間、インターフェイスでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

Option Explicit が **On** の場合 (既定) は、使用するすべての変数を宣言する必要があります。**Option Explicit Off** の場合は、開発者の意図に関係なく、宣言されていないすべての変数が既定で オブジェクト型 (Object)になります。

Dim ステートメントでは各変数のデータ型を指定できます。初期値を指定することもできます。初期値を指定しなかった場合は、既定の設定が使用されます。詳細については、このページの「詳細情報」の下にある「データ型のルール」と「既定値」を参照してください。次の例では、**String** 型の変数を宣言して初期化します。

```
Dim summary As String = "Summary of results"
```

Dim ステートメントで *accessmodifier* を指定すると、変数にアクセスできるコードを指定できます。詳細については、「修飾子」、およびこのページの「詳細情報」の下にある「アクセス レベル」を参照してください。

複数の値を格納する配列を保持する変数も宣言できます。詳細については、このページの「詳細情報」の下にある「配列のルール」を参照してください。配列の詳細については、「Visual Basic における配列」を参照してください。次の例では、**Integer** 型の配列変数を宣言します。

```
Dim days() As Integer
```

通常は、変数を使用するコード範囲の先頭部分に、その変数に関する **Dim** ステートメントを記述します。詳細については、このページの「詳細情報」の下にある「トラブルシューティング」を参照してください。

詳細情報

ここでは、次のトピックについての詳細を説明します。

- 宣言
- データ型
- 配列
- 動作
- トラブルシューティング

宣言のルール

- **宣言コンテキスト** モジュール レベルで宣言した変数、つまりプロシージャの外部で宣言した変数のことをメンバ変数またはフィールドといいます。これは、それを宣言したクラス、構造体、モジュールのメンバです。

  プロシージャ レベルで宣言した変数はローカル変数といいます。これは、それを宣言したプロシージャまたはブロックに対してローカルになります。

- **属性** 属性はメンバ変数にのみ適用できます。ローカル変数には適用できません。属性はアセンブリのメタデータに情報を提供するものであり、ローカル変数などの一時的な記憶領域に対しては意味を持ちません。

- **暗黙的な Dim の使用 Public、Protected、Friend、Protected Friend、Private、Shared、Shadows、Static、ReadOnly、**または **WithEvents** という修飾子を指定した場合は、**Dim** キーワードを省略できます。

  ```
  Public maximumAllowed As Double
  Protected Friend currentUserName As String
  ```

```
Private salary As Decimal
Static runningTotal As Integer
```

モジュール レベルでは、**Static** 修飾子を使用してメンバ変数を宣言できません。プロシージャ レベルでは、**Shared**、**Shadows**、**ReadOnly**、**WithEvents**、または任意のアクセス修飾子を使用してローカル変数を宣言できません。

- **WithEvents 変数** **WithEvents** はメンバ変数に対してのみ指定できます。プロシージャ内のローカル変数には指定できません。

  WithEvents を指定した場合は、その変数のデータ型を **Object** 以外の具体的なクラス型にする必要があります。配列を **WithEvents** で宣言することはできません。

  イベントの詳細については、「イベントとイベント ハンドラ」を参照してください。

- **複数の変数** 1 つの宣言ステートメント内で複数の変数を宣言することができます。その場合は、それぞれの変数について *variablename* を指定し、配列名の後ろにはかっこを指定します。複数の変数を指定するときは、コンマ (,) で区切ります。

```
Dim lastTime, nextTime, allTimes() As Date
```

データ型のルール

- **データ型 Dim** ステートメントでは、変数のデータ型を宣言できます。任意のデータ型、または列挙体、構造体、クラス、インターフェイスの名前を指定できます。

```
Dim finished As Boolean
Dim monitorBox As System.Windows.Forms.Form
```

- **既定の型** *datatype* を指定しない場合、変数は *initializer* のデータ型をとります。*datatype* と *initializer* のどちらも指定しない場合、既定のデータ型は オブジェクト型 (Object)になります。*datatype* と *initializer* の両方を指定する場合は、*initializer* のデータ型を *datatype* と互換性のあるものにする必要があります。

- **異なる型** 宣言する変数ごとに別々の **As** 句を指定すると、それぞれの変数に異なるデータ型を指定できます。共通の **As** 句を使用すると、複数の変数を同じ型として宣言できます。それぞれの変数は、*variablename* の後に来る最初の **As** 句で指定されたデータ型をとります。

```
Dim a, b, c As Single, x, y As Double, i As Integer
' a, b, and c are all Single; x and y are both Double
```

- **初期化 Dim** ステートメントでは、特定の変数の内容を *variablelist* で初期化できます。値型の場合は、*initializer* を使用して、変数に代入する式を指定します。この式は、コンパイル時に計算可能な定数として評価されるものにしてください。

```
Dim quantity As Integer = 10
Dim message As String = "Just started"
```

参照型の場合は、**New** キーワードを使用して、*datatype* で指定したクラスまたは構造体の新しいインスタンスを作成します。**New** を使用する場合は、*initializer* 式を使用しないでください。その代わりに、必要に応じて、変数の作成元になるクラスのコンストラクタに引数を指定します。

```
Dim bottomLabel As New System.Windows.Forms.Label
```

1 つの **As** 句で複数の変数を宣言する場合は、その変数グループに対して *initializer* を指定することはできません。

配列のルール

- **配列変数** *variablelist* の変数が配列であることを指定するには、*variablename* の直後にかっこを指定します。配列に複数の次元がある場合は、次元の数を示すために、かっこの中にコンマを含める必要があります。

```
Dim oneDimension(), twoDimensions(,), threeDimensions(,,) As Byte
```

配列には 1 ～ 32 の次元を指定できます。

詳細については、「Visual Basic における配列」を参照してください。

- **配列の範囲** 各次元の上限と下限を指定できます。そのためには、かっこ内に *boundslist* を指定します。それぞれの次元について、*boundslist* で上限を指定します。下限は省略可能です。下限は、指定してもしなくても常に 0 になります。各次元のインデックスは 0 から上限値までの範囲で変化します。

  次の 2 つのステートメントは等価です。どちらのステートメントも、21 個の **Integer** 要素から成る配列を宣言しています。この配列にアクセスするときには、インデックスは 0 ～ 20 の範囲で変化します。

```
Dim totals(20) As Integer
Dim totals(0 To 20) As Integer
```

  次のステートメントは、**Double** 型の 2 次元配列を宣言しています。この配列は 4 行 (3 + 1) × 6 列 (5 + 1) になります。

```
Dim matrix2(3, 5) As Double
```

  上限の値は、次元のサイズを表しているのではなく、インデックスが取りうる最大の値を表していることに注意してください。次元のサイズは上限プラス 1 になります。

- **空の配列範囲** 配列宣言の範囲の指定をすべて空白にすることもできます。その場合、配列は指定の次元数を持ちますが、初期化されていないので、少なくともいくつかの要素を初期化するまでは、値は **Nothing** になります。**Dim** ステートメントでは、すべての次元について範囲を指定するか、指定しないかのどちらかにする必要があります。

```
Dim messages() As String
```

- **空の配列** -1 を使って配列の次元の上限を宣言できます。これは、配列が **Nothing** ではなく空であることを意味し、共通言語ランタイムの一部の関数では区別が必要です。ただし、Visual Basic のコードではこのような配列に正しくアクセスできません。アクセスしようとすると、実行時に IndexOutOfRangeException エラーが発生します。詳細については、「方法 : 要素を持たない配列を作成する」を参照してください。

- **配列の初期化** 配列の値を初期化するには、初期値を中かっこ (**{}**) で囲んで指定します。

```
Dim longArray() As Long = {0, 1, 2, 3}
```

  多次元配列の場合は、外側の次元の中に、各次元の初期値を中かっこで囲んで指定します。要素は、行優先で指定されます。

```
Dim twoDimensions(,) As Integer = {{0, 1, 2}, {10, 11, 12}}
```

動作

- **既定値** 変数の *initializer* を指定しない場合、Visual Basic は変数をそのデータ型の既定値に初期化します。既定の初期値は、次の表のとおりです。

データ型	既定値
すべての数値型 (**Byte** および **SByte** を含む)	0
Char	バイナリの 0
すべての参照型 (**Object**、**String**、およびすべての配列を含む)	**Nothing**
Boolean	**False**
Date	西暦 1 年 1 月 1 日の午前 12:00 (01/01/0001 12:00:00 AM)

  構造体の各要素は、独立した変数として初期化されます。配列の長さを宣言しただけで要素を初期化していない場合は、各要素は独立した変数として初期化されます。

- **アクセス レベル** プロシージャ内のローカル変数のアクセス レベルは、既定で public になります。このような変数にはアクセス修飾子を指

定できません。

プロシージャ外のクラス変数およびモジュール メンバ変数のアクセス レベルは、既定で private になります。構造体メンバ変数のアクセス レベルは、既定で public になります。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。

- **スコープ** ローカル変数のスコープは、そのプロシージャまたはブロックの内部に限られます。メンバ変数のスコープは、そのクラス、構造体、またはモジュールの全体に及びます。
- **修飾** クラス、構造体、またはモジュールの外部のコードでメンバ変数を使用する場合は、変数名をそのクラス、構造体、またはモジュールの名前で修飾する必要があります。プロシージャまたはブロックの外部のコードから、そのプロシージャまたはブロック内のローカル変数を参照することはできません。
- **有効期間 Static** ローカル変数の有効期間は、それを宣言しているプロシージャの有効期間よりも長くなります。変数の有効期間の境界は、プロシージャが宣言される場所、およびプロシージャが共有 (**Shared**) かどうかによって異なります。

プロシージャの宣言	変数が初期化されるタイミング	変数が無効になるタイミング
モジュール内で宣言	プロシージャが最初に呼び出されたとき	プログラムが実行を終了するとき
クラスまたは構造体内で **Shared** として宣言	特定のインスタンス、または宣言したクラスまたは構造体のいずれかで、プロシージャが最初に呼び出されたとき	プログラムが実行を終了するとき
クラスまたは構造体内で非 **Shared** として宣言	特定のインスタンスで、プロシージャが最初に呼び出されたとき	ガベージ コレクション (GC) に対して、インスタンスが解放されたとき

トラブルシューティング

- **実行順序 Dim** ステートメントは、それ自身は実行可能なステートメントではありません。ただし、そこで変数を初期化する場合は、その初期化処理が代入ステートメントとして扱われます。つまり、変数の値は次の段階を経ることになります。

  1. 変数を宣言しているコード要素に初めて到達すると、Visual Basic は変数をそのデータ型の既定値に初期化します。
  2. 変数の **Dim** ステートメントに到達すると、Visual Basic は変数をその **Dim** ステートメントで指定されている値に初期化します。
  3. 再び変数の **Dim** ステートメントに戻ったときは、Visual Basic は変数をその **Dim** ステートメントで指定されている値に再度設定します。

  このことから、**Dim** ステートメントを使用して変数を初期化するときには次のことが言えます。

  - **Dim** ステートメントを実行する前に変数を使用した場合、その変数は、**Dim** ステートメントで指定された値ではなく、そのデータ型の既定値に設定される。
  - 実行が **Dim** ステートメントに到達しなかった場合、その変数が **Dim** ステートメントで指定された値に初期化されることはない。
  - 変数の値を変更した後に **Dim** ステートメントに戻った場合は、変更後の値が **Dim** ステートメントで指定された値に置き換えられる。

  すべての **Dim** ステートメントを、その変数を使用するコード範囲 (たとえばモジュールやプロシージャ) の先頭部分に記述していれば、意図したとおりに変数を初期化できます。

使用例

次の例では、**Dim** ステートメントのさまざまなオプションを使用して変数を宣言しています。

VB

```
' The following statement declares and initializes a Long variable.
Dim startingAmount As Long = 500
```

VB

```
' The following statement declares a variable that refers to a Button
' object, creates a new Button object, and assigns it to the variable.
Dim switchButton As New System.Windows.Forms.Button
```

VB

```
' The following statement declares a variable that can only be
' accessed by code in the same class, structure, or module.
Private homeTelephone As String = ""
```

VB

```
' The following statement declares a local variable that always retains
' its value, even after its procedure returns to the calling code.
Static totalSales As Double
```

VB

```
' The following statement declares a variable that refers to an array.
Dim highTemperature(31) As Integer
```

VB

```
' The following statement declares and initializes an array variable
' that holds 4 Boolean check values.
Dim checkValues() As Boolean = {False, False, True, False}
```

参照

処理手順

方法 : 配列変数を宣言する

方法 : 配列変数を宣言する

関連項目

Const ステートメント (Visual Basic)

ReDim ステートメント (Visual Basic)

Option Explicit ステートメント (Visual Basic)

概念

Visual Basic での変数宣言

Do...Loop ステートメント (Visual Basic)

指定されたブール型 (**Boolean**) の条件が真 (**True**) である間、または条件が真 (**True**) になるまで、一連のステートメントを繰り返します。

```
Do { While | Until } condition
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop
-or-
Do
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop { While | Until } condition
```

指定項目

While

Until を使用しない場合は、必ず指定します。*condition* が偽 (**False**) になるまでループを繰り返します。

Until

While を使用しない場合は、必ず指定します。*condition* が真 (**True**) になるまでループを繰り返します。

condition

省略できます。ブール式 (**Boolean**) を指定します。*condition* が **Nothing** の場合、Visual Basic はこれを **False** として扱います。

statements

省略可能です。*condition* が真 (**True**) である間、または真になるまで繰り返し実行される、任意の行数のステートメントを記述します。

Exit Do

省略可能です。制御を **Do** ループの外に移します。

Loop

必ず指定します。**Do** ループの定義を終了します。

解説

Do...Loop は一連のステートメントを、条件が満たされるまで何度でも繰り返し実行する場合に使用します。設定した回数だけステートメントを繰り返す場合は、For...Next Statement を選択した方が通常は便利です。

Do...Loop を使用すると、*condition* が真 (**True**) でなくなったときにループを終了するのか、それとも最初に **True** になったときに終了するのかを選択できるので、While...End While ステートメント (Visual Basic) を使用するよりも柔軟性が増します。また、*condition* をループの最初にテストすることも、ループの終わりにテストすることもできます。

規則

- **条件の指定**条件は、通常、2 つの値の比較によって判断されますが、ブール型 (Boolean) (Visual Basic) の値 (**True** または **False**) に評価される式であれば何でも指定できます。指定できる値には、数値型などのその他のデータ型を **Boolean** に変換した値も含まれます。
- **条件のテスト***condition* はループの最初または終わりに 1 回だけテストできます。**While** と **Until** のいずれかを使用して *condition* を指定しますが、両方は使用できません。
- **反復の回数***condition* をループの最初に (**Do** ステートメント内で) テストする場合、ループが一度も実行されない場合があります。ループの終わりに (**Loop** ステートメント内で) テストする場合、ループは必ず少なくとも 1 回は実行されます。
- **ループの入れ子**Do ループは入れ子構造にできます。つまり、ループの内部に別のループを入れることができます。また、種類の異なる制御構造を入れ子にすることもできます。詳細については、「入れ子になった制御構造」を参照してください。
- **ループの外への制御の転送。**Exit ステートメント (Visual Basic) は、**Loop** ステートメントの次のステートメントに制御を直ちに移します。ループの継続が不要になったり不可能になったりする条件 (エラー値や終了要求など) を検出した場合にループを終了できま

す。**Exit Do** ステートメントは **Do** ループの任意の場所に何度でも使用できます。**Exit Do** は **If...Then...Else** の中など、何らかの条件を評価した後によく使用されます。

無限ループ

Exit Do は、無限ループを引き起こす可能性がある条件をテストする場合などに使用されます。無限ループは、実行回数が極端に多くなる (または無限に繰り返される) ループです。このような条件が検出された場合に、**Exit Do** を使用してループから抜けることができます。そうしない限り、ループは実行し続けます。

次の例では、ループを 2^31 回よりも多く実行させる可能性のある値が `number` に代入されています。**If** ステートメントは、この変数の値をチェックし、条件が True の場合にループを終了して、無限ループになるのを防ぎます。

VB

```
Sub exitDoExample()
    Dim counter As Integer = 0
    Dim number As Integer = 8
    Do Until number = 10
        If number <= 0 Then Exit Do
        number -= 1
        counter += 1
    Loop
    MsgBox("The loop ran " & counter & " times.")
End Sub
```

メモ：

無限ループを停止するには、Esc キーを押すか、または Ctrl キーを押しながら Break キーを押します。

使用例

次の例に、入れ子になった **Do**...**Loop**、**While** と **Until** の使用方法、およびループの最初 (**Do** ステートメント) と終わり (**Loop** ステートメント) に条件をテストする方法を示します。

VB

```
    Dim check As Boolean = True
    Dim counter As Integer = 0
    Do
        Do While counter < 20
            counter += 1
            If counter = 10 Then
                check = False
                Exit Do
            End If
        Loop
    Loop Until check = False
End Sub
```

この例では、内側の **Do...Loop** でループを 10 回繰り返すと、フラグの値を偽 (**False**) に設定し、**Exit Do** ステートメントを使ってループから抜けます。外側のループは、フラグの値を調べてループから抜けます。

参照

処理手順

方法 : ループのパフォーマンスを改善する

関連項目

For...Next ステートメント (Visual Basic)

ブール型 (Boolean) (Visual Basic)

Exit ステートメント (Visual Basic)

While...End While ステートメント (Visual Basic)

概念

ループ構造

入れ子になった制御構造

End ステートメント

プログラムの実行を終了させます。

```
End
```

解説

End ステートメントをプロシージャの任意の場所に記述すると、アプリケーション全体の実行を強制終了できます。**End** は **Open** ステートメントで開いたすべてのファイルを閉じ、アプリケーションのすべての変数を削除します。オブジェクトへの参照を保持している他のプログラムおよび実行中のコードがなくなると、アプリケーションはすぐに終了します。

メモ：

End ステートメントは、**Dispose** メソッドや **Finalize** メソッド、またはその他の Visual Basic コードを呼び出すことなく、コードの実行を直ちに停止します。他のプログラムが保持しているオブジェクトへの参照は、無効になります。**End** ステートメントが **Try** ブロックや **Catch** ブロックの内部で実行された場合、対応する **Finally** ブロックに制御が渡されることはありません。

Stop ステートメントはプログラムの実行を中断しますが、**End** ステートメントと異なり、コンパイル済みの実行可能ファイル (.EXE) の内部でない限り、ファイルを閉じたり、変数をクリアすることはありません。

End は開いているリソースの後処理を何も行わずにアプリケーションを終了させるため、使用する前にリソースを正しく閉じてみる必要があります。たとえば、アプリケーションのフォームが 1 つでも開いている場合は、制御が **End** ステートメントに到達する前にフォームを閉じてください。

End の使用はできる限り控えて、すぐに停止する必要がある場合にのみ使用してください。通常の方法 (Return ステートメント (Visual Basic) および Exit ステートメント (Visual Basic)) を使ってプロシージャを終了させれば、プロシージャを正しく閉じることができるだけでなく、正しく閉じるための処理を呼び出し元のコードに追加することもできます。たとえば、コンソール アプリケーションの場合なら、**Main** プロシージャから **Return** を実行するだけです。

セキュリティに関するメモ：

End ステートメントは、System 名前空間にある Environment クラスの Exit メソッドを呼び出します。**Exit** を使用するためには、UnmanagedCode のアクセス許可が必要です。アクセス許可がない場合は SecurityException エラーが発生します。

End (Visual Basic) に続けてキーワードを記述すると、該当するプロシージャまたはブロックの定義を終了するという意味になります。たとえば、**End Function** であれば **Function** プロシージャの定義が終了します。

使用例

ユーザーが要求した場合に **End** ステートメントを使用してコードの実行を終了させるコード例は、次のとおりです。

VB

```
Sub Form_Load()
  Dim answer As MsgBoxResult
  answer = MsgBox("Do you want to quit now?", MsgBoxStyle.YesNo)
  If answer = MsgBoxResult.Yes Then
      MsgBox("Terminating program")
      End
  End If
End Sub
```

スマート デバイス開発者のためのメモ

このステートメントはサポートされていません。

参照

処理手順

方法 : アンマネージ コードへのアクセス許可を要求する

関連項目

Stop ステートメント (Visual Basic)

End (Visual Basic)
SecurityPermissionFlag

Enum ステートメント (Visual Basic)

列挙体を宣言して、そのメンバの値を定義します。

```
[ <attribute list> ] [ access modifier ]  [ Shadows ]
Enum enumeration name [ As data type ]
      member list
End Enum
```

指定項目

attribute list

省略可能です。この列挙体に適用される属性の一覧です。属性リストは山かっこ ("<" および ">") で囲む必要があります。

access modifier

省略可能です。どのようなコードからこの列挙体にアクセスできるのかを指定します。次のいずれかになります。

- Public
- Protected
- Friend
- Private

Protected Friend と指定すると、列挙体のクラス、その派生クラス、または同じアセンブリ内のコードからのみアクセスできます。

Shadows

省略可能です。この列挙体が、基本クラスにある、同じ名前を持つプログラミング要素、またはオーバーロードされる要素を宣言し直すことを示します。Shadows は列挙体自体のみに指定できます。メンバには指定できません。

enumeration name

必ず指定します。列挙体の名前を指定します。有効な名前の詳細については、「宣言された要素の名前」を参照してください。

data type

Option Strict が **On** の場合は、必ず指定します。列挙体およびそのすべてのメンバのデータ型を指定します。

member list

必ず指定します。このステートメントで宣言する定数のメンバ リストです。メンバが複数の場合には、ソース コード行も複数になります。

各 *member* の構文と指定項目は次のとおりです。`[<attribute list>] member name [ = initializer ]`

指定項目	説明
member name	必ず指定します。このメンバの名前です。
initializer	省略可能です。コンパイル時に計算して、このメンバに代入される式です。

End Enum

Enum ブロックを終了します。

解説

論理的に互いに関連があり、変更することのない複数の値が存在する場合に、これらの値を列挙体でまとめて定義できます。このような使い方をすると、列挙体および列挙体のメンバに意味のある名前を付けることができ、値を使用するよりも覚えやすくなります。列挙体を定義したら、列挙体のメンバをコードの随所で使用できます。これによって、相互に関連する値がすべて同じ列挙体名で表現されるため、コードも読みやすくなります。

Enum は、名前空間またはモジュール レベルでのみ使用できます。つまり、列挙体の宣言コンテキストは、ソース ファイル、名前空間、クラス、構造体、モジュール、またはインターフェイスのいずれかである必要があり、プロシージャでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

Enum ステートメントでは、列挙体のデータ型を宣言できます。各メンバのデータ型は、列挙体のデータ型になります。**Byte**、**Integer**、**Long**、**SByte**、**Short**、**UInteger**、**ULong**、または **UShort** のいずれかを指定できます。

メンバに *initializer* を指定しないと、そのメンバが *member list* 内の最初の *member* である場合、Visual Basic で、そのメンバがゼロに初期化されます。最初のメンバでない場合には、直前の *member* に 1 を加えた値に初期化されます。

クラス、構造体、モジュール、およびインターフェイスのメンバ列挙体は、既定でパブリック アクセスです。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。名前空間のメンバの列挙体は、既定でフレンド アクセスです。このアクセス レベルをパブリックに変更することはできますが、プライベートやプロテクトには変更できません。詳細については、「Visual Basic でのアクセス レベル」を参照してください。

- **宣言コンテキスト** モジュール レベルで宣言された、プロシージャの外側にある列挙体をメンバ列挙体といい、これを宣言しているクラス、構造体、モジュール、またはインターフェイスのメンバです。

  名前空間レベルで宣言され、クラス、構造体、モジュール、またはインターフェイスの外側にある列挙体は、この列挙体を宣言している名前空間のみのメンバです。

- **属性** 列挙体全体に対して属性を適用できます。個々のメンバに適用することはできません。属性を指定すると、アセンブリのメタデータに情報が付加されます。

- **修飾子** 既定では、すべての列挙体は型であり、列挙体のフィールドは定数です。したがって、列挙体やそのメンバを宣言するとき、**Shared**、**Static**、および **ReadOnly** のキーワードは使用できません。

データ型のルール

- **既定の型** 列挙体の *data type* を指定しなかった場合、各メンバのデータ型は *initializer* のデータ型になります。*data type* と *initializer* の両方を指定する場合は、*initializer* のデータ型を *data type* と互換性のあるものにする必要があります。*data type* と *initializer* のどちらも指定しない場合、既定のデータ型は **Integer** になります。

- **初期化** **Enum** ステートメントは、*member list* の選択されたメンバの内容を初期化できます。メンバに代入するための式を指定するには、*initializer* を使用します。

  各 *initializer* に指定する式は、リテラル、定義済みの他の定数、定義済みの他の列挙体のメンバを適宜組み合わせます。この列挙体内の定義済みのメンバも式に利用できます。算術演算子と論理演算子を使って、各要素を組み合わせることもできます。

  initializer では、変数または関数は使用できません。ただし、**CByte** や **CShort** などの変換キーワードは使用できます。また、**AscW** も使用できます。この場合には、定数の **String** 引数、または **Char** 引数を指定して、コンパイル時に計算できるようにします。

動作

- **アクセス レベル** 列挙体のメンバはすべて、パブリック アクセスです。他のアクセス修飾子をメンバに指定することはできません。ただし、列挙体本体に制限のより強いアクセス レベルが指定してあると、列挙体に指定されたアクセス レベルが優先されます。

- **スコープ** メンバ列挙体は、列挙体を含んでいるクラス、構造体、モジュール、またはインターフェイスのどこからでもアクセスできます。名前空間のメンバ列挙体は、その名前空間内のどのコードからでもアクセスできます。

- **修飾** クラス、構造体、またはモジュールの外部のコードで使用する場合は、そのクラス、構造体、またはモジュールの名前でメンバ列挙体の名前を修飾する必要があります。

- **無効な値** メンバの値が基になっているデータ型に対する許容範囲を超えている場合、またはいずれかのメンバが基になっているデータ型で許容される最大値に初期化された場合は、コンパイラでエラーが報告されます。

列挙型変数は、**Enum** 型として宣言された変数です。この方法で変数を宣言すると、変数に割り当てる値の管理に役立ちます。ただし、列挙体のデータ型に変換できるデータ型を使用すれば、列挙体のメンバとしては存在しない値を列挙型変数に代入できます。これは、列挙体がフラグ フィールドであり、フラグを組み合わせた値を列挙型変数に代入する場合に役立ちます。列挙型変数に、複数のフラグを代入する例を次に示します。

VB

```
Enum filePermissions
    create = 1
    read = 2
    write = 4
    delete = 8
End Enum
```

VB

```
Dim file1Perm As filePermissions
file1Perm = filePermissions.create Or filePermissions.read
```

列挙体メンバに対する参照はすべて、列挙体変数の名前または列挙体の名前自体で修飾する必要があります。たとえば、前記の例では、最初のメンバを `filePermissions.create` と参照することはできますが、`create` と参照することはできません。

使用例

次の例では、**Enum** ステートメントを使用して、関連する名前付き定数値のセットを定義しています。これらの値は、データベースのデータ入力フォームをデザインするときに選択できる色を表します。

VB

```
Public Enum InterfaceColors
    MistyRose = &HE1E4FF&
    SlateGray = &H908070&
    DodgerBlue = &HFF901E&
    DeepSkyBlue = &HFFBF00&
    SpringGreen = &H7FFF00&
    ForestGreen = &H228B22&
    Goldenrod = &H20A5DA&
    Firebrick = &H2222B2&
End Enum
```

値に正数と負数の両方を含む例を次に示します。

VB

```
Enum SecurityLevel
    IllegalEntry = -1
    MinimumSecurity = 0
    MaximumSecurity = 1
End Enum
```

参照

関連項目

Const ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)
データ型変換関数
Asc 関数、AscW 関数

概念

暗黙の型変換と明示的な型変換
組み込み定数と組み込み列挙型

その他の技術情報

Visual Basic の定数と列挙体

Erase ステートメント (Visual Basic)

配列変数と、その配列変数の要素で使用されていたメモリを解放します。

```
Erase arraylist
```

指定項目
arraylist

必ず指定します。消去する配列変数のリストを指定します。複数の変数を指定するときは、コンマ (,) で区切ります。

解説

Erase ステートメントを使用できるのは、プロシージャ レベルだけです。つまり、プロシージャの中では配列を解放できますが、クラス レベルやモジュール レベルでは解放できません。

Erase ステートメントは、各配列変数に **Nothing** 値を割り当てるのと同じ結果になります。

使用例

次に示す例では、**Erase** ステートメントを使って 2 つの配列を消去し、使用していたメモリを解放しています (それぞれ、記憶要素が 1,000 と 100)。その後、**ReDim** ステートメントで 3 次元配列に新しい配列インスタンスを割り当てています。

VB

```
Dim threeDimArray(9, 9, 9), twoDimArray(9, 9) As Integer
Erase threeDimArray, twoDimArray
ReDim threeDimArray(4, 4, 9)
```

参照
関連項目
Nothing (Visual Basic)
ReDim ステートメント (Visual Basic)

Error ステートメント

エラーの発生をシミュレートします。

```
Error errornumber
```

指定項目
errornumber

必ず指定します。任意の有効なエラー番号を指定できます。

解説

Error ステートメントは下位互換性用にサポートされています。新しいコードでは、**Err** オブジェクトの **Raise** メソッドを使用してランタイム エラーを生成します。特に、オブジェクトを作成する場合はこのメソッドを使用します。

errornumber を定義すると、**Error** ステートメントは、**Err** オブジェクトのプロパティに次の表の既定値を割り当ててからエラー ハンドラを呼び出します。

プロパティ	値
Number	**Error** ステートメントの引数として指定される値。任意の有効なエラー番号を指定できます。
Source	現在の Visual Basic プロジェクトの名前。
Description	この文字列が存在する場合は、指定の **Number** に対する **Error** 関数の戻り値に対応する文字列式。文字列が存在しない場合、**Description** には長さ 0 の文字列 ("") が含まれます。
HelpFile	適切な Visual Basic ヘルプ ファイルへの絶対的なドライブ、パス、およびファイル名。
HelpContext	適切な Visual Basic ヘルプ ファイルの、**Number** プロパティに対応するエラーのコンテキスト ID。
LastDLLError	ゼロ。

エラー ハンドラがないか、または有効でない場合は、**Err** オブジェクトのプロパティによってエラー メッセージが作成および表示されます。

メモ :

一部の Visual Basic ホスト アプリケーションではオブジェクトを作成できません。クラスやオブジェクトを作成できるかどうかを確認するには、ホスト アプリケーションのドキュメントを参照してください。

使用例

Error ステートメントを使ってエラー番号 11 を生成する例を次に示します。

```
On Error Resume Next   ' Defer error handling.
Error 11   ' Simulate the "Division by zero" error.
```

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Microsoft Visual Basic ランタイム (Microsoft.VisualBasic.dll 内)

参照
処理手順
方法 : Visual Basic ランタイム エラーに関する情報を取得する
関連項目
Clear メソッド (Err オブジェクト)

Err オブジェクト (Visual Basic)
On Error ステートメント (Visual Basic)
Raise メソッド (Err オブジェクト)
Resume ステートメント

Event ステートメント

ユーザー定義のイベントを宣言します。

```
[ <attrlist> ] [ accessmodifier ] _
[ Shared ] [ Shadows ] Event eventname[(parameterlist)] _
[ Implements implementslist ]
' -or-
[ <attrlist> ] [ accessmodifier ] _
[ Shared ] [ Shadows ] Event eventname As delegatename _
[ Implements implementslist ]
' -or-
 [ <attrlist> ] [ accessmodifier ] _
[ Shared ] [ Shadows ] Custom Event eventname As delegatename _
[ Implements implementslist ]
   [ <attrlist> ] AddHandler(ByVal value As delegatename)
      [ statements ]
   End AddHandler
   [ <attrlist> ] RemoveHandler(ByVal value As delegatename)
      [ statements ]
   End RemoveHandler
   [ <attrlist> ] RaiseEvent(delegatesignature)
      [ statements ]
   End RaiseEvent
End Event
```

指定項目

指定項目	説明
attrlist	省略可能です。このイベントに適用される属性の一覧を指定します。複数の属性を指定するときは、コンマ (,) で区切ります。属性リストは山かっこ ("<" および ">") で囲む必要があります。
accessmodifier	省略可能です。どのようなコードからイベントにアクセスできるのかを指定します。次のいずれかになります。 • Public - この列挙体を宣言している要素にアクセス可能な、すべてのコードからアクセスできます。 • Protected - そのクラスまたは派生クラス内のコードからのみアクセスできます。 • Friend - 同じアセンブリ内のコードからのみアクセスできます。 • Private - この列挙体を宣言している要素内のコードからのみアクセスできます。 **Protected Friend** と指定すると、イベントのクラス、その派生クラス、または同じアセンブリ内のコードからのみアクセスできます。
Shared	省略可能です。このイベントがクラスや構造体の特定のインスタンスに関連付けられていないことを指定します。
Shadows	省略可能です。このイベントが、基本クラスにある、同じ名前を持つプログラミング要素、またはオーバーロードされる要素を宣言し直して隠ぺいすることを示します。宣言された要素は、他の任意の種類の要素でシャドウできます。 シャドウされた要素は、その要素をシャドウする派生クラスからは使用できません。ただし、シャドウする要素がアクセスできない要素の場合は例外です。たとえば、**Private** 要素が基本クラスの要素をシャドウすると、**Private** 要素へのアクセス許可を持たないコードは、基本クラスにアクセスします。
eventname	必ず指定します。イベントの名前を指定します。イベントの標準的な名前付け規則に従って名前を付けます。
parameterlist	省略可能です。このイベントのパラメータを表すローカル変数のリストです。パラメータの一覧 はかっこで囲む必要があります。

<table>
<tr><td>Implements</td><td>省略可能です。このイベントがインターフェイスのイベントを実装することを示します。</td></tr>
<tr><td>implementslist</td><td>Implements が指定されている場合は、必ず指定します。実装される Sub プロシージャのリストです。複数のプロシージャを指定するときは、コンマ (,) で区切ります。

implementedprocedure [, implementedprocedure ...]

implementedprocedure の構文と指定項目は次のとおりです。

interface.definedname

<table>
<tr><th>指定項目</th><th>説明</th></tr>
<tr><td>interface</td><td>必ず指定します。このプロシージャの包含クラスまたは包含構造体が実装しているインターフェイスの名前です。</td></tr>
<tr><td>definedname</td><td>必ず指定します。interface の中で、プロシージャを定義するために使われている名前を指定します。これは name、つまり定義されているプロシージャを実装するために、このプロシージャが使用している名前と同じである必要はありません。</td></tr>
</table>
</td></tr>
<tr><td>Custom</td><td>必ず指定します。Custom で宣言されたイベントには、カスタムな AddHandler、RemoveHandler、および RaiseEvent の各アクセサを定義する必要があります。</td></tr>
<tr><td>delegatename</td><td>省略可能です。イベント ハンドラのシグネチャを指定するデリゲートの名前です。</td></tr>
<tr><td>AddHandler</td><td>必ず指定します。AddHandler アクセサを宣言します。ここにはイベントハンドラが追加されたとき実行するステートメントを、AddHandler ステートメントを使って明示的に指定するか、Handles 句を使って暗黙的に指定します。</td></tr>
<tr><td>End AddHandler</td><td>必ず指定します。AddHandler ブロックを終了します。</td></tr>
<tr><td>value</td><td>必ず指定します。パラメータ名です。</td></tr>
<tr><td>RemoveHandler</td><td>必ず指定します。RemoveHandler アクセサを宣言します。ここではイベント ハンドラが削除されたときに実行するステートメントを、RemoveHandler ステートメントを使って指定します。</td></tr>
<tr><td>End RemoveHandler</td><td>必ず指定します。RemoveHandler ブロックを終了します。</td></tr>
<tr><td>RaiseEvent</td><td>必ず指定します。RaiseEvent アクセサを宣言します。ここではイベントが発生したときに実行するステートメントを、RaiseEvent ステートメントを使って指定します。通常は、AddHandler アクセサと RemoveHandler アクセサが保持するリストに含まれるデリゲートが起動されます。</td></tr>
<tr><td>End RaiseEvent</td><td>必ず指定します。RaiseEvent ブロックを終了します。</td></tr>
<tr><td>delegatesignature</td><td>必ず指定します。delegatename デリゲートに必要なパラメータと一致するパラメータのリストです。パラメータの一覧 はかっこで囲む必要があります。</td></tr>
<tr><td>statements</td><td>省略可能です。RemoveHandler、RemoveHandler、および RaiseEvent の各メソッドの本体に含まれるステートメントです。</td></tr>
<tr><td>End Event</td><td>必ず指定します。Event ブロックを終了します。</td></tr>
</table>

解説

宣言したイベントは、**RaiseEvent** ステートメントを使って発生させます。通常、イベントの宣言と発生は、次のように行われます。

VB

```
Public Class EventSource
    ' Declare an event.
    Public Event LogonCompleted(ByVal UserName As String)
    Sub CauseEvent()
        ' Raise an event on successful logon.
        RaiseEvent LogonCompleted("AustinSteele")
    End Sub
End Class
```

メモ：

イベントの引数は、プロシージャの引数と同様に宣言できます。ただし、イベントに対して名前付き引数、**ParamArray** 引数、または **Optional** 引数を指定することはできません。イベントは値を返しません。

イベントを処理するためには、**Handles** ステートメントまたは **AddHandler** ステートメントを使用して、イベント ハンドラをイベントに関連付ける必要があります。サブルーチンとイベントのシグネチャは一致していることが必要です。共有イベントを処理するには、**AddHandler** ステートメントを使う必要があります。

Event は、モジュール レベルでのみ使用できます。つまり、イベントの宣言コンテキストは、クラス、構造体、モジュール、またはインターフェイスであることが必要で、ソース ファイル、名前空間、プロシージャ、ブロックでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

ほとんどの状況で、このトピックの「構文」のセクションにある最初の構文を使ってイベントを宣言できますが、場合によっては、イベントの動作をより詳細に制御することが必要になります。このトピックの「構文」セクションの最後に **Custom** キーワードを使用した構文がありますが、これを使用すると、カスタム イベントを定義してイベントを詳細に制御できます。カスタム イベントでは、イベント ハンドラをイベントに追加したり、イベントから削除したり、またはイベントが発生したときにどの処理を実行するかを、細かく指定できます。使用例については、「方法 : メモリの使用量を節約するイベントを宣言する」および「方法 : ブロックを回避するイベントを宣言する」を参照してください。

使用例

次の例はイベントを使用して、10 秒から 0 秒までカウント ダウンします。このコードには、**RaiseEvent** ステートメントなど、イベントに関連するメソッド、プロパティ、およびステートメントの使い方を示す例がいくつも含まれています。

イベントを発生させるクラスをイベント ソース、イベントを処理するメソッドをイベント ハンドラと呼びます。イベント ソースには、そこで生成される複数のイベント ハンドラを設定できます。クラスでイベントが発生すると、そのイベントは、オブジェクトのインスタンスに対するイベントを処理するために選択されたすべてのクラスで発生します。

また、このコード例では、ボタン (`Button1`) および テキスト ボックス (`TextBox1`) のあるフォーム (`Form1`) を使用します。ボタンをクリックすると、10 秒から 0 秒までカウント ダウンする最初のテキスト ボックスが表示されます。カウント時間 (10 秒) が経過すると、最初のテキスト ボックスに "Done" と表示されます。

`Form1` のコードでは、フォームの初期状態と終了状態を指定します。イベント発生時に実行されるコードも含まれます。

この例を使用するには、新しい Windows Forms プロジェクトを開きます。次に、`Button1` という名前のボタンと `TextBox1` という名前のテキスト ボックスを、`Form1` という名前のメイン フォームに追加します。続いてフォームを右クリックし、[コードの表示] をクリックして、コード エディタを開きます。

`Form1` クラスの宣言セクションに、**WithEvents** 変数を追加します。

VB

```
Private WithEvents mText As TimerState
```

`Form1` のコードに以下のコードを追加します。`Form_Load` や `Button_Click` など、重複して存在する可能性のあるプロシージャを置き換えます。

VB

```
Private Sub Form1_Load(ByVal sender As Object, _
                       ByVal e As System.EventArgs) _
```

```
                        Handles MyBase.Load
    Button1.Text = "Start"
    mText = New TimerState
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
                          Handles Button1.Click
    mText.StartCountdown(10.0, 0.1)
End Sub

Private Sub mText_ChangeText() Handles mText.Finished
    TextBox1.Text = "Done"
End Sub

Private Sub mText_UpdateTime(ByVal Countdown As Double) _
                             Handles mText.UpdateTime
    TextBox1.Text = Format(Countdown, "##0.0")
    ' Use DoEvents to allow the display to refresh.
    My.Application.DoEvents()
End Sub

Class TimerState
    Public Event UpdateTime(ByVal Countdown As Double)
    Public Event Finished()
    Public Sub StartCountdown(ByVal Duration As Double, _
                              ByVal Increment As Double)
        Dim Start As Double = DateAndTime.Timer
        Dim ElapsedTime As Double = 0

        Dim SoFar As Double = 0
        Do While ElapsedTime < Duration
            If ElapsedTime > SoFar + Increment Then
                SoFar += Increment
                RaiseEvent UpdateTime(Duration - SoFar)
            End If
            ElapsedTime = DateAndTime.Timer - Start
        Loop
        RaiseEvent Finished()
    End Sub
End Class
```

F5 キーを押してコード例を実行し、[Start] というラベルのボタンをクリックします。最初のテキスト ボックスが、秒のカウント ダウンを開始します。カウント時間 (10 秒) が経過すると、最初のテキスト ボックスに "Done" と表示されます。

メモ :

My.Application.DoEvents メソッドがイベントを処理する方法は、フォームとすべて同じではありません。To allow the form to handle the events directly, you can use multithreading.詳細については、「Visual Basic におけるマルチスレッド」を参照してください。

参照

処理手順

方法 : イベントをクラスに追加する
方法 : メモリの使用量を節約するイベントを宣言する
方法 : ブロックを回避するイベントを宣言する

関連項目

RaiseEvent ステートメント
Implements ステートメント
AddHandler ステートメント
RemoveHandler ステートメント
Handles
Delegate ステートメント
Shared (Visual Basic)
Shadows

その他の技術情報

Visual Basic におけるイベント

Exit ステートメント (Visual Basic)

プロシージャまたはブロックを終了し、そのプロシージャ呼び出しまたはブロック定義の次のステートメントに即座に制御を移します。

```
Exit { Do | For | Function | Property | Select | Sub | Try | While }
```

指定項目

Do

このステートメントのある **Do** ループを直ちに抜けます。**Loop** ステートメントの後ろのステートメントから実行が継続されます。**Exit Do** が使用できるのは **Do** ループの中だけです。入れ子構造になっている **Do** ループの中で使用すると、**Exit Do** は内側のループを抜け、入れ子構造の 1 つ外側のレベルに制御を移します。

For

このステートメントのある **For** ループを直ちに抜けます。**Next** ステートメントの後ろのステートメントから実行が継続されます。**Exit For** が使用できるのは **For**...**Next** ループまたは **For Each**...**Next** ループの中だけです。入れ子構造になっている **For** ループの中で使用すると、**Exit For** は内側のループを抜け、入れ子構造の 1 つ外側のレベルに制御を移します。

Function

このステートメントのある **Function** プロシージャを直ちに抜けます。**Function** プロシージャを呼び出したステートメントの後ろのステートメントから実行が継続されます。**Exit Function** が使用できるのは **Function** プロシージャの中だけです。

Property

このステートメントのある **Property** プロシージャを直ちに抜けます。**Property** プロシージャを呼び出したステートメント、つまりプロパティの値を要求または設定しているステートメントから実行が継続されます。**Exit Property** が使用できるのは、プロパティの **Get** プロシージャまたは **Set** プロシージャの中だけです。

Select

このステートメントのある **Select Case** を直ちに抜けます。**End Select** ステートメントの後ろのステートメントから実行が継続されます。**Exit Select** が使用できるのは **Select Case** ステートメントの中だけです。

Sub

このステートメントのある **Sub** プロシージャを直ちに抜けます。**Sub** プロシージャを呼び出したステートメントの後ろのステートメントから実行が継続されます。**Exit Sub** が使用できるのは **Sub** プロシージャの中だけです。

Try

このステートメントのある **Try** ブロックまたは **Catch** ブロックを直ちに抜けます。**Finally** ブロックが 1 つ存在する場合はそこから、そうでない場合は **End Try** ステートメントの後ろのステートメントから実行を継続します。**Exit Try** が使用できるのは **Try** ブロックまたは **Catch** ブロックの中だけです。**Finally** ブロックの中では使用できません。

While

このステートメントのある **While** ループを直ちに抜けます。**End While** ステートメントの後ろのステートメントから実行が継続されます。**Exit While** が使用できるのは **While** ループの中だけです。**While** ループが入れ子構造になっている場合、**Exit While** のあるループの 1 つ外側のループに制御が移ります。

解説

Exit ステートメントと **End** ステートメントを混同しないでください。**Exit** はステートメントの終了を定義するステートメントではありません。

使用例

Exit ステートメントを使用して、**For**...**Next** ループ、**Do** ループおよび **Sub** プロシージャを抜けるコード例は、次のとおりです。

VB

```
Sub exitStatementDemo()
    Dim demoNum As Single
    ' Set up an infinite loop.
    Do
        For i As Integer = 1 To 10000000
            demoNum = Int(Rnd() * 100)
```

```
            Select Case demoNum
                Case 7 : Exit For
                Case 29 : Exit Do
                Case 54 : Exit Sub
            End Select
        Next i
    Loop
End Sub
```

参照

関連項目

Do...Loop ステートメント (Visual Basic)
End ステートメント
For Each...Next ステートメント (Visual Basic)
For...Next ステートメント (Visual Basic)
Function ステートメント (Visual Basic)
Stop ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)
Try...Catch...Finally ステートメント (Visual Basic)
For Each...Next ステートメント (Visual Basic)

ステートメント F ～ P

次の表は、Visual Basic 言語のステートメントの一覧です。

For Each...Next	For...Next	Function	Get
GoTo	If...Then...Else	Implements	Imports
Inherits	Interface	Mid	Module
Namespace	On Error	Operator	Option Compare
Option Explicit	Option Strict	Property	

参照

関連項目

ステートメント A ～ E

ステートメント Q ～ Z

その他の技術情報

Visual Basic リファレンス

For Each...Next ステートメント (Visual Basic)

コレクションの各要素に対して、一連のステートメントを繰り返し実行する一連のステートメントです。

```
For Each element [ As datatype ] In group
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ element ]
```

指定項目

element

For Each ステートメントには必ず指定します。**Next** ステートメントでは省略可能です。コレクションの各要素を反復処理するために使用する変数を指定します。

datatype

element が宣言されていない場合は、必ず指定します。*element* のデータ型を指定します。

group

必ず指定します。オブジェクト変数です。*statements* で反復処理するコレクションを表します。

statements

省略可能です。**For Each** から **Next** までの間に記述した 1 つ以上のステートメントは、*group* 内の各項目に対して実行されます。

Exit For

省略可能です。プログラムの制御を **For Each** ループの外に移します。

Next

必ず指定します。**For Each** ループの定義を終了します。

解説

コレクションまたは配列の各要素に対して一連のステートメントを繰り返し実行するときには、**For Each**...**Next** ループを使用します。

For...Next ステートメント (Visual Basic) は、ループの毎回の反復を制御変数に関連付けることができ、制御変数の初期値と最終値を決定できるときには役立ちます。しかし、コレクションを扱うときには初期値と最終値の概念は意味を持ちませんし、コレクション内の要素の個数が必ずしも判明しているわけではありません。このような場合には、**For Each**...**Next** ループの方が適しています。

規則

- **データ型** *element* のデータ型は、*group* の要素のデータ型から変換可能なものにする必要があります。

  group のデータ型は、コレクションまたは配列を参照する参照型にする必要があります。つまり、*group* は System.Collections 名前空間の IEnumerable インターフェイスまたは System.Collections.Generic 名前空間の IEnumerable インターフェイスを実装するオブジェクトを参照する必要があります。**IEnumerable** は、コレクションの列挙子オブジェクトを返す GetEnumerator メソッドを定義します。列挙子オブジェクトは **System.Collections** 名前空間の IEnumerator インターフェイスを実装し、Current プロパティと Reset メソッドおよび MoveNext メソッドを公開します。このプロパティとメソッドは、コレクションの反復処理に使用されます。

  group の要素は、通常は **Object** 型にしますが、任意のランタイム データ型を指定できます。

- **宣言** このループの外側で *element* を宣言していない場合は、**For Each** ステートメント内で宣言する必要があります。その場合、*element* のスコープはループの本体になります。ただし、ループの外側と内側の両方で *element* を宣言することはできません。

- **繰り返しの回数** Visual Basic は、ループの開始前にコレクションを 1 回だけ評価します。ステートメント ブロックによって *element* または *group* が変更されても、ループの繰り返しには影響しません。

- **ループの入れ子** **For Each** ループは入れ子構造にできます。つまり、ループの中に別のループを入れることができます。ただし、それぞれのループに一意の *element* 変数を指定する必要があります。

  また、別の種類の制御構造を入れ子にすることもできます。詳細については、「入れ子になった制御構造」を参照してください。

  メモ：

内側の入れ子レベルの **Next** ステートメントの前に外側の入れ子レベルの **Next** ステートメントが来た場合は、コンパイラからエラーが報告されます。ただし、コンパイラがこのエラーを検出できるのは、すべての **Next** ステートメントに *element* を指定した場合に限られます。

- **制御変数の識別** オプションとして、**Next** ステートメントに *element* を指定することもできます。こうするとプログラムの読みやすさが向上し、特に **For Each** ループを入れ子にしている場合は効果があります。その場合には、**For Each** ステートメントで指定したのと同じ変数を指定する必要があります。

- **ループの中断** Exit ステートメント (Visual Basic) は、**Next** ステートメントの次のステートメントにすぐに制御を移します。これを使用すると、ループの継続を不要または不可能にする条件 (エラー値や終了要求など) を検出した場合にループを終了できます。また、**Try**...**Catch**...**Finally** で例外をキャッチした場合にも、**Finally** ブロックの最後で **Exit For** を使用できます。

  For Each ループ内では **Exit For** ステートメントを好きな場所に何回でも記述できます。通常は、何らかの条件を評価した後、たとえば **If**...**Then**...**Else** 構造の後に **Exit For** を記述します。

- **無限ループ Exit For** の 1 つの用途は、無限ループ (実行回数が極端に多いループ、または無限に繰り返されるループ) を引き起こす可能性がある条件をテストすることです。このような条件を検出した場合は、**Exit For** を使用してループを抜けることができます。詳細については、「Do...Loop ステートメント (Visual Basic)」を参照してください。

動作

- **ループの開始 For Each**...**Next** ループの実行を開始するときには、*group* が有効なコレクション オブジェクトを参照しているかどうかが検証されます。有効なコレクション オブジェクトでない場合は、例外がスローされます。有効なコレクション オブジェクトである場合は、列挙子オブジェクトの **MoveNext** メソッドと **Current** プロパティが呼び出され、1 つ目の要素が返されます。**MoveNext** メソッドが次の要素がないことを示した場合、つまりコレクションが空の場合は、**For Each** ループは終了し、**Next** ステートメントの次のステートメントに制御が渡されます。それ以外の場合は、*element* が 1 つ目の要素に設定され、ステートメント ブロックが実行されます。

- **ループの反復 Next** ステートメントに達すると、そのたびに **For Each** ステートメントに制御が戻されます。次の要素を返すために再び **MoveNext** と **Current** が呼び出され、その結果に応じて、ステートメント ブロックが再度実行されるか、ループが終了します。**MoveNext** メソッドが次の要素がないことを示すまで、または **Exit For** ステートメントに達するまでは、このプロセスが繰り返されます。

- **ループの終了** コレクションのすべての要素が *element* に代入されると、**For Each** ループは終了し、**Next** ステートメントの次のステートメントに制御が渡されます。

- **反復値の変更** ループ内で *element* の値を変更すると、コードの読みやすさが低下してデバッグが難しくなります。*group* の値を変更しても、コレクションやその要素には影響が現れません。これらは最初にループに入る時点で確定されます。

- **反復処理の順序 For Each**...**Next** ループを実行すると、コレクションの反復処理の順序は **GetEnumerator** メソッドが返す列挙子オブジェクトの制御下に置かれます。反復処理の順序は、Visual Basic ではなく、列挙子オブジェクトの **MoveNext** メソッドによって決まります。したがって、コレクションのどの要素が最初に *element* に返されるかや、特定の要素の後にどの要素が返されるかを予測することはできません。

  特定の順序でコレクションを反復処理する必要がある場合は、そのコレクションによって公開される列挙子オブジェクトの特性を把握していない限り、**For Each**...**Next** ループが適切な選択肢とは言えません。**For**...**Next** ループや **Do**...**Loop** ループなど、別のループ構造を使用した方が、信頼できる結果が得られます。

- **コレクションの変更 GetEnumerator** から返される列挙子オブジェクトでは、通常、要素の追加、削除、置換、または並べ替えを行ってコレクションを変更することはできません。**For Each**...**Next** ループの開始後にコレクションを変更すると列挙子オブジェクトが無効になり、次に要素にアクセスしようとしたときに InvalidOperationException 例外が発生します。

  ただし、この変更のブロッキングは、Visual Basic ではなく **IEnumerable** インターフェイスの実装によって決定されます。反復処理中の変更を許可するように **IEnumerable** を実装できます。このような動的な変更を行う場合には、使用するコレクションでの **IEnumerable** 実装の特徴を理解しておくことが必要です。

- **コレクション要素の変更** 列挙子オブジェクトの **Current** プロパティは ReadOnly (Visual Basic) であり、各コレクション要素のローカル コピーを返します。これは、**For Each**...**Next** ループ内では要素そのものを変更できないということを意味します。変更は **Current** から返されたローカル コピーにのみ適用され、基のコレクションに反映されることはありません。ただし、要素が参照型である場合、その要素が指すインスタンスのメンバを変更することはできます。次に例を示します。

```
Sub lightBlueBackground(ByVal thisForm As System.Windows.Forms.Form)
    For Each thisControl As System.Windows.Forms.Control In thisForm.Controls
        thisControl.BackColor = System.Drawing.Color.LightBlue
```

```
    Next thisControl
End Sub
```

この例では各 `thisControl` 要素の `BackColor` メンバを変更していますが、`thisControl` そのものは変更できません。

- **配列の反復処理** Array クラスは **IEnumerable** インターフェイスを実装するため、すべての配列は GetEnumerator メソッドを公開します。これは、**For Each**...**Next** ループによって配列を反復処理できることを意味します。ただし、配列要素を読み取るだけで、変更はできません。概要については、「方法 : コレクションまたは配列で、各要素の複数のステートメントを実行する」を参照してください。

使用例

次の例では、**For Each**...**Next** ステートメントを使用して、コレクション内のすべての要素で文字列 "Hello" を検索します。この例は、`thisCollection` コレクションが既に作成されていて、その要素の型が **String** であることを前提にしています。

VB

```
Dim found As Boolean = False
Dim thisCollection As New Collection
For Each thisObject As String In thisCollection
    If thisObject = "Hello" Then
        found = True
        Exit For
    End If
Next thisObject
```

参照

処理手順

方法 : コレクションまたは配列で、各要素の複数のステートメントを実行する

方法 : ループのパフォーマンスを改善する

関連項目

While...End While ステートメント (Visual Basic)

Do...Loop ステートメント (Visual Basic)

概念

ループ構造

Visual Basic におけるコレクション

For...Next ステートメント (Visual Basic)

指定された回数だけ、一連のステートメントを繰り返すフロー制御ステートメントです。

```
For counter [ As datatype ] = start To end [ Step step ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next [ counter ]
```

指定項目

counter

For ステートメントには必ず指定します。数値変数を指定します。このループの制御変数になります。

datatype

counter がまだ宣言されていない場合は、必ず指定します。*counter* のデータ型を指定します。

start

必ず指定します。数式を指定します。*counter* の初期値になります。

end

必ず指定します。数式を指定します。*counter* の最終値になります。

step

省略可能です。数式を指定します。ループを 1 回実行するごとに引数 *counter* を増やす量です。

statements

省略可能です。**For** と **Next** の間に記述したステートメントは、指定した回数だけ実行されます。

Exit For

省略可能です。プログラムの制御を **For** ループの外に移します。

Next

必ず指定します。この **For** ループの定義を終了します。

解説

一連のステートメントを決まった回数だけ繰り返し実行するときには **For**...**Next** 構造を使用します。

ループ内のステートメントを実行する回数が事前にわからない場合には、While...End While ステートメント (Visual Basic) または Do...Loop ステートメント (Visual Basic) を使用します。一方、指定の回数だけループを実行する場合は、**For**...**Next** ループが最適です。このループでは、最初にループに入るときに、繰り返しの回数を決定します。

step には正の数または負の数を指定できます。どちらを指定するかによって、ループの処理方法が次のように決まります。

ステップの値	実行条件
正の数または 0	*counter* <= *end*
負の数	*counter* >= *end*

step を省略すると、ループを繰り返すごとに counter に 1 が加算されます。

規則

- **データ型** *counter* のデータ型は、通常は **Integer** にします。しかし、以上 (>=)、以下 (<=)、加算 (+)、減算 (-) の各演算子をサポートするものならば任意のデータ型を指定できます。これらの演算子をサポートしていれば、ユーザー定義型を指定してもかまいません。

  start、*end*、および *step* には、通常は **Integer** 型として評価される式を指定しますが、*counter* と同じ型に拡張可能な任意のデータ型の式を指定することもできます。*counter* にユーザー定義型を使用する場合は、*start*、*end*、または *step* の型を *counter* と同じ型に

変換するために、**CType** 変換演算子を定義しなければならないことがあります。

- **宣言** このループの外側で *counter* を宣言していない場合は、**For** ステートメント内で宣言する必要があります。その場合、*counter* のスコープはループの本体になります。ただし、ループの外側と内側の両方で *counter* を宣言することはできません。
- **繰り返しの回数** Visual Basic は、*start*、*end*、*step* という反復値をループの開始前に 1 回だけ評価します。ステートメント ブロックによって *end* または *step* が変更されても、ループの繰り返しには影響しません。
- **ループの入れ子** **For** ループは入れ子構造にできます。つまり、ループの中に別のループを入れることができます。ただし、それぞれのループに一意の *counter* 変数を指定する必要があります。有効なステートメントの例を次に示します。

```
                For i As Integer = 1 To 10
    For j As Integer = 1 To 10
        For k As Integer = 1 To 10
            ' Insert statements to operate with current values of i, j, and k.
        Next k
    Next j
Next i
```

また、別の種類の制御構造を入れ子にすることもできます。詳細については、「入れ子になった制御構造」を参照してください。

メモ：

内側の入れ子レベルの **Next** ステートメントの前に外側の入れ子レベルの **Next** ステートメントが来た場合は、コンパイラからエラーが報告されます。ただし、コンパイラがこのエラーを検出できるのは、すべての **Next** ステートメントに *counter* を指定した場合に限られます。

- **制御変数の識別** オプションとして、**Next** ステートメントに *counter* を指定することもできます。こうするとプログラムの読みやすさが向上し、特に **For** ループを入れ子にしている場合は効果があります。その場合には、**For** ステートメントで指定したのと同じ変数を指定する必要があります。
- **ループの中断** Exit ステートメント (Visual Basic) は、**Next** ステートメントの次のステートメントにすぐに制御を移します。これを使用すると、ループの継続を不要または不可能にする条件 (エラー値や終了要求など) を検出した場合にループを終了できます。また、**Try**...**Catch**...**Finally** で例外をキャッチした場合にも、**Finally** ブロックの最後で **Exit For** を使用できます。

  For ループ内では **Exit For** ステートメントを好きな場所に何回でも記述できます。通常は、何らかの条件を評価した後、たとえば **If**...**Then**...**Else** 構造の後に **Exit For** を記述します。
- **無限ループ** **Exit For** の 1 つの用途は、無限ループ (実行回数が極端に多いループ、または無限に繰り返されるループ) を引き起こす可能性がある条件をテストすることです。このような条件を検出した場合は、**Exit For** を使用してループを抜けることができます。詳細については、「Do...Loop ステートメント (Visual Basic)」を参照してください。

動作

- **ループの開始** **For**...**Next** ループの実行が始まるときに、Visual Basic は *start*、*end*、および *step* を 1 回だけ評価します。次に、*start* が *counter* に代入されます。ステートメント ブロックが実行される前に、*counter* と *end* が比較されます。*counter* の値が既に end 値を超えている場合は、**For** ループが終了し、**Next** ステートメントに続くステートメントに制御が渡されます。それ以外の場合は、ステートメント ブロックが実行されます。
- **ループの反復** **Next** ステートメントが実行されるたびに、*step* の値が *counter* に加算され、**For** ステートメントに戻ります。その後再び *counter* と *end* が比較され、その結果に応じて、ステートメント ブロックが再度実行されるかループが終了します。このプロセスは、*counter* が *end* を超えるか、**Exit For** ステートメントに到達するまで継続されます。
- **ループの終了** ループは *counter* が *end* を超えるまで継続されます。*counter* と *end* が等しい場合にはループは継続されます。ブロックの実行を行うかどうかは、*step* が正の場合は *counter* <= *end* の比較によって決定され、*step* が負の場合は *counter* >= *end* の比較によって決定されます。
- **反復値の変更** ループ内で *counter* の値を変更すると、コードの読みやすさが低下してデバッグが難しくなります。*start*、*end*、または *step* の値を変更しても、最初のループ開始時に決定された反復値には影響が及びません。

使用例

次の例は、それぞれ異なる step 値を持つ **For**...**Next** 構造を入れ子にしたようすを示しています。

VB

```
Dim words, digit As Integer
Dim thisString As String = ""
For words = 10 To 1 Step -1
    For digit = 0 To 9
        thisString &= CStr(digit)
    Next digit
    thisString &= " "
Next words
```

この例では、0 から 9 までの数字列 ("0123456789") を 10 個格納する文字列を作成します。それぞれの数字列の間は半角スペースで区切られます。外側のループは、外側のループを 1 回実行するごとにループ カウンタ変数を 1 ずつ減少します。

参照

処理手順

方法 : ループのパフォーマンスを改善する

関連項目

While...End While ステートメント (Visual Basic)

Do...Loop ステートメント (Visual Basic)

Exit ステートメント (Visual Basic)

For Each...Next ステートメント (Visual Basic)

概念

ループ構造

入れ子になった制御構造

Function ステートメント (Visual Basic)

Function プロシージャを定義する名前、パラメータ、およびコードを宣言します。

```
[ <attributelist> ] [ accessmodifier ] [ proceduremodifiers ] [ Shared ] [ Shadows ]
Function name [ (Of typeparamlist) ] [ (parameterlist) ] [ As returntype ] [ Implements imp
lementslist | Handles eventlist ]
    [ statements ]
    [ Exit Function ]
    [ statements ]
End Function
```

指定項目

attributelist

省略可能です。「属性リスト」を参照してください。

accessmodifier

省略可能です。次のいずれかになります。

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Visual Basic でのアクセス レベル を参照してください。

proceduremodifiers

省略可能です。次のいずれかになります。

- Overloads
- Overrides
- Overridable
- NotOverridable
- MustOverride
- **MustOverride Overrides**
- **NotOverridable Overrides**

Shared

省略可能です。「Shared (Visual Basic)」を参照してください。

Shadows

省略可能です。「Shadows」を参照してください。

name

必ず指定します。プロシージャの名前を指定します。宣言された要素の名前 を参照してください。

typeparamlist

省略可能です。ジェネリック プロシージャの型パラメータ リストを指定します。「型リスト」を参照してください。

parameterlist

省略可能です。このプロシージャのパラメータを表すローカル変数名のリストを指定します。「パラメータの一覧」を参照してください。

returntype

Option Strict が **On** の場合は、必ず指定します。このプロシージャによって返される値のデータ型を指定します。

Implements

省略可能です。このプロシージャが 1 つ以上の **Function** プロシージャを実装し、このプロシージャの包含クラスまたは包含構造体によって実装されたインターフェイスに各 **Function** プロシージャが定義されていることを示します。Implements ステートメント を参照してください。

implementslist

Implements が指定されている場合は、必ず指定します。実装する **Function** プロシージャのリストを指定します。

```
implementedprocedure [ , implementedprocedure ... ]
```

implementedprocedure の構文と指定項目は次のとおりです。

```
interface.definedname
```

指定項目	説明
interface	必ず指定します。このプロシージャの包含クラスまたは包含構造体によって実装されるインターフェイスの名前を指定します
definedname	必ず指定します。*interface* の中でプロシージャを定義するために使われる名前を指定します。

Handles

省略可能です。このプロシージャが 1 つ以上の特定のイベントを処理できることを示します。Handles を参照してください。

eventlist

Handles が指定されている場合は、必ず指定します。このプロシージャが処理するイベントのリストを指定します。

```
eventspecifier [ , eventspecifier ... ]
```

eventspecifier の構文と指定項目は次のとおりです。

```
eventvariable.event
```

指定項目	説明
eventvariable	必ず指定します。イベントを生成するクラスまたは構造体のデータ型で宣言したオブジェクト変数を指定します。
event	必ず指定します。このプロシージャが処理するイベントの名前を指定します。

statements

省略可能です。このプロシージャ内で実行されるステートメントのブロックを指定します。

End Function

このプロシージャの定義を終了します。

解説

すべての実行可能コードをプロシージャに含める必要があります。各プロシージャを順番にクラス、構造体、またはモジュールの中に宣言します。これらは、包含クラス、包含構造体、または包含モジュールと呼ばれます。

呼び出し元のコードに値を返す必要がある場合は、**Function** プロシージャを使用します。値を返す必要がない場合は、**Sub** プロシージャを使用します。

Function は、モジュール レベルでのみ使用できます。つまり、関数の宣言コンテキストは、クラス、構造体、モジュール、またはインターフェイスであることが必要で、ソース ファイル、名前空間、プロシージャ、またはブロックでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

Function プロシージャのアクセス レベルは、既定でパブリックです。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。

関数が返す値を使用する場合は、式の右辺に **Function** プロシージャを記述できます。**Function** プロシージャは、**Sqrt**、**Cos**、または **ChrW** などのライブラリ関数と同じ方法で使用します。

Function プロシージャを式の中から呼び出すには、プロシージャ名の後に引数リストをかっこで囲んで指定します。指定する引数がない場合に限って、かっこを省略できます。ただし、常にかっこで囲むとコードは読みやすくなります。

また、**Call** ステートメントで関数を呼び出すこともできます。この場合、戻り値は無視されます。

規則

- **戻り値の型。** **Function** ステートメントは、戻り値のデータ型を宣言できます。任意のデータ型、または列挙、構造体、クラス、またはインターフェイスの名前を指定できます。

  returntype を指定しなかった場合は、プロシージャは **Object** を返します。

- **実装。** このプロシージャが **Implements** キーワードを使用する場合、包含クラスまたは包含構造体も **Implements** ステートメントを **Class** ステートメントまたは **Structure** ステートメントの直後に記述する必要があります。**Implements** ステートメントには、*implementslist* に指定されたすべてのインターフェイスが含まれる必要があります。ただし、**Function** を定義するためにインターフェイスで使用される名前 (*definedname* 内) は、このプロシージャの名前 (*name* 内) と同じである必要はありません。

動作

- **プロシージャからの戻り** **Function** プロシージャが呼び出し元のコードに戻ると、プロシージャを呼び出したステートメントの次のステートメントから実行が継続します。

  Exit Function ステートメントおよび **Return** ステートメントは、**Function** プロシージャを直ちに終了します。**Exit Function** ステートメントと **Return** ステートメントは、プロシージャ内の任意の位置で何回でも指定できます。また **Exit Function** ステートメントと **Return** ステートメントを混在して使用できます。

- **戻り値。** 関数から値を返すには、関数名に値を割り当てるか、または **Return** ステートメントに値を設定します。戻り値を `myFunction` という名前の関数に割り当てて、**Exit Function** ステートメントを使って返すコード例を次に示します。

  VB

  ```
  Function myFunction(ByVal j As Integer) As Double
      myFunction = 3.87 * j
      Exit Function
  End Function
  ```

  name に値を割り当てずに **Exit Function** を使用すると、プロシージャは *returntype* に指定されたデータ型の既定値を返します。*returntype* を指定しないと、プロシージャは **Object** の既定値である **Nothing** を返します。

  Return ステートメントは、戻り値を割り当て、同時に関数を終了します。次に例を示します。

  VB

  ```
  Function myFunction(ByVal j As Integer) As Double
      Return 3.87 * j
  End Function
  ```

トラブルシューティング

- **実行順序。** Visual Basic では、演算効率を高めるために数式が自動的に並べ替えられることがあります。このため、変数の値を変えてしまうような **Function** プロシージャを数式の中で実行しないようにしてください。

使用例

Function ステートメントを使って、**Function** プロシージャの名前、パラメータ、および本体を構成するコードを宣言するコード例は、次のとおりです。**ParamArray** 修飾子を使用すると、関数で異なる数の引数を処理できます。

VB

```
Public Function calcSum(ByVal ParamArray args() As Double) As Double
    calcSum = 0
    If args.Length <= 0 Then Exit Function
    For i As Integer = 0 To UBound(args, 1)
        calcSum += args(i)
    Next i
End Function
```

次のコード例は、上記の例で宣言した関数を呼び出します。

VB

```
Dim returnedValue As Double = calcSum(4, 3, 2, 1)
' The function's local variables are assigned the following values:
' args(0) = 4, args(1) = 3, and so on.
```

参照

処理手順

方法 : ジェネリック クラスを使用する

プロシージャのトラブルシューティング

関連項目

Sub ステートメント (Visual Basic)

パラメータの一覧

Dim ステートメント (Visual Basic)

Call ステートメント (Visual Basic)

Of

概念

パラメータ配列

Get ステートメント

プロパティの値を取得するための **Get** プロパティ プロシージャを宣言します。

```
[ <attributelist> ] [ accessmodifier ] Get()
    [ statements ]
End Get
```

指定項目

attributelist

省略可能です。「属性リスト」を参照してください。

accessmodifier

このプロパティでは、**Get** ステートメントと **Set** ステートメントのうちの 1 つについては、指定を省略できます。次のいずれかを指定できます。

- Protected
- Friend
- Private
- **Protected Friend**

Visual Basic でのアクセス レベル を参照してください。

statements

省略可能です。**Get** プロパティ プロシージャが呼び出されたときに実行する、1 つ以上のステートメントを指定します。

End Get

必ず指定します。この **Get** プロパティ プロシージャの定義を終了します。

解説

WriteOnly でマーク付けされていないすべてのプロパティに、**Get** プロパティ プロシージャが必要です。**Get** プロシージャは、プロパティの現在の値を取得するために使用します。

式でプロパティの値が必要になると、Visual Basic が自動的にプロパティの **Get** プロシージャを呼び出します。

プロパティ宣言の本体には、プロパティの **Get** プロシージャと **Set** プロシージャだけを、Property ステートメントと **End Property** ステートメントの間に定義できます。このプロシージャ以外は何も格納できません。特に、プロパティの現在の値を格納できないことに注意してください。この値は、プロパティの外側に格納する必要があります。その理由は、どちらか一方のプロパティ プロシージャの内部にこの値を格納すると、もう一方のプロパティ プロシージャからアクセスできなくなるためです。この値は、プロパティと同じレベルで宣言された Private (Visual Basic) 変数に格納するという方法が一般的です。**Get** プロシージャは、それが適用されるプロパティの内側に定義する必要があります。

Get プロシージャの既定のアクセス レベルは、*accessmodifier* を **Get** ステートメントで使用しない限り、その包含プロパティのアクセス レベルになります。

規則

- **アクセス レベルの混在** 読み書き可能なプロパティを定義する場合、必要であれば **Get** プロシージャと **Set** プロシージャのどちらかに限り、プロパティとは異なるアクセス レベルを指定できます。これを指定する場合は、プロシージャにプロパティよりも制限の高いアクセス レベルを指定する必要があります。たとえば、プロパティを **Friend** で宣言する場合、**Get** プロシージャを **Private** で宣言できますが、**Public** では宣言できません。

  ReadOnly プロパティを定義する場合、プロパティは **Get** プロシージャだけで機能します。**Get** に違うアクセス レベルを宣言すると、プロパティに 2 つのアクセス レベルを設定することになるため宣言できません。

- **戻り値の型** Property ステートメントには、プロパティが返す値のデータ型を宣言できます。**Get** プロシージャは自動的にそのデータ型を返します。任意のデータ型の他に、列挙値、構造体、クラス、またはインターフェイスの名前を指定できます。

  Property ステートメントに *returntype* が指定されなければ、プロシージャは **Object** を返します。

動作

- **プロシージャからの戻り**`Get` プロシージャから呼び出しコードに制御が戻るとき、プロパティの値を要求したステートメントの内部から実行が継続します。

  `Get` プロパティ プロシージャは Return ステートメント (Visual Basic)を使って値を返すか、またはプロパティ名に戻り値を代入して値を返すことができます。詳細については、「Function ステートメント (Visual Basic)」の "戻り値" を参照してください。

  `Exit Property` ステートメントと `Return` ステートメントは、プロパティ プロシージャを直ちに終了します。プロシージャの任意の場所に、`Exit Property` ステートメントと `Return` ステートメントを何度でも定義できます。また、`Exit Property` ステートメントと `Return` ステートメントを混在して使用できます。

- **戻り値**`Get` プロシージャから値を返すには、プロパティ名に値を代入するか、または Return ステートメント (Visual Basic) に値を指定します。`Return` ステートメントは `Get` プロシージャの戻り値を代入すると同時に、プロシージャを終了します。

  プロパティ名に値を代入せずに `Exit Property` を使用すると、`Get` プロシージャは、そのプロパティのデータ型の既定値を返します。詳細については、「Function ステートメント (Visual Basic)」の "戻り値" を参照してください。

  次の例は、読み取り専用の `quoteForTheDay` プロパティからプライベート変数 `quoteValue` に格納された値を返すための、2 つの方法を示しています。

  VB

```
Private quoteValue As String = "No quote assigned yet."
```

  VB

```
ReadOnly Property quoteForTheDay() As String
    Get
        quoteForTheDay = quoteValue
        Exit Property
    End Get
End Property
```

  VB

```
ReadOnly Property quoteForTheDay() As String
    Get
        Return quoteValue
    End Get
End Property
```

使用例

次の例は、`Get` ステートメントを使って、プロパティの値を返します。

VB

```
Class propClass
    ' Define a private local variable to store the property value.
    Private currentTime As String
    ' Define the read-only property.
    Public ReadOnly Property dateAndTime() As String
        Get
            ' The Get procedure is called automatically when the
            ' value of the property is retrieved.
            currentTime = CStr(Now)
            ' Return the date and time As a string.
            Return currentTime
        End Get
    End Property
End Class
```

参照

処理手順

方法 : フィールドおよびプロパティをクラスに追加する

関連項目

Set ステートメント (Visual Basic)

Property ステートメント

Exit ステートメント (Visual Basic)

GoTo ステートメント

プロシージャ内の指定した行に無条件に分岐します。

```
GoTo line
```

指定項目
line

必ず指定します。任意の行ラベルを指定します。

解説

GoTo ステートメントの分岐先は、同じプロシージャ内だけです。他のプロシージャには分岐できません。分岐先の行には、**GoTo** が参照するための行ラベルが必要です。詳細については、「方法 : ステートメントにラベル付けする」を参照してください。

メモ :

GoTo ステートメントを使用すると、プログラムが読みにくくなり、保守が困難になります。可能な限り、制御構造を使うようにしてください。詳細については、「Visual Basic における制御フロー」を参照してください。

GoTo ステートメントを、**For**...**Next**、**For Each**...**Next**、**SyncLock**...**End SyncLock**、**Try**...**Catch**...**Finally**、**With**...**End With**、または **Using**...**End Using** の外部から内部のラベルへの分岐に使うことはできません。

分岐と Try 構築

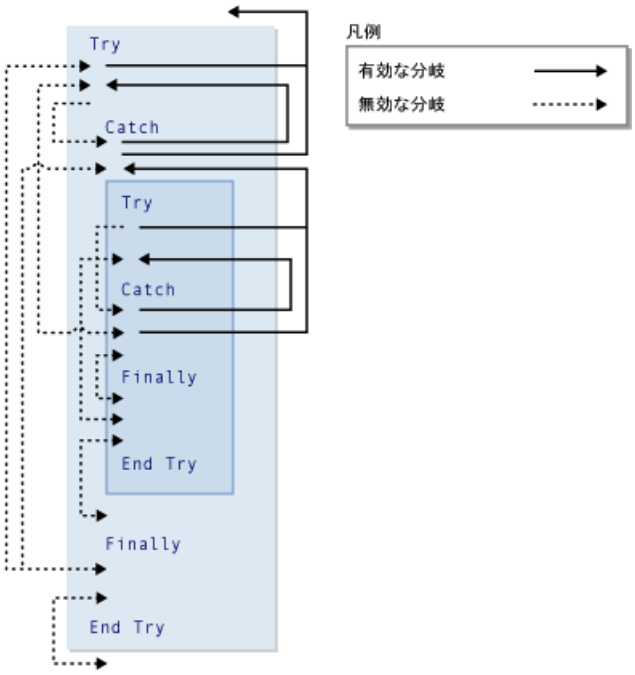
Try...**Catch**...**Finally** 構築の内部では、**GoTo** ステートメントによる分岐に以下の規則が適用されます。

ブロックまたは領域	外部から内部への分岐	内部から外部への分岐
Try ブロック	同じ構築の **Catch** ブロックからのみ分岐可能 [1]	構築全体の外部へのみ分岐可能
Catch ブロック	不可	構築全体の外部、または同じ構築の **Try** ブロックへの分岐のみ可能 [1]
Finally ブロック	不可	不可

[1] ある **Try**...**Catch**...**Finally** 構築が別の構築の中に入れ子になっている場合、**Catch** ブロックから自分と同じレベルの **Try** ブロックへの分岐は可能ですが、それ以外の **Try** ブロックへは分岐できません。入れ子にされた **Try**...**Catch**...**Finally** 構築は、完全に外側の構築の **Try** ブロックまたは **Catch** ブロック内に含まれていることが必要です。

次の例では、**Try** 構築が別の構築にネストされています。2 つの構築のブロックどうしのさまざまな分岐について、有効または無効を示してあります。

Try 構築内の有効な分岐と無効な分岐



使用例

GoTo ステートメントを使って、プロシージャ内の行ラベルに分岐するコード例は、次のとおりです。

VB

```
    Sub gotoStatementDemo()
        Dim number As Integer = 1
        Dim sampleString As String
        ' Evaluate number and branch to appropriate label.
        If number = 1 Then GoTo Line1 Else GoTo Line2
Line1:
        sampleString = "Number equals 1"
        GoTo LastLine
Line2:
        ' The following statement never gets executed because number = 1.
        sampleString = "Number equals 2"
LastLine:
        ' Write "Number equals 1" in the Debug window.
        Debug.WriteLine(sampleString)
    End Sub
```

参照

関連項目

Do...Loop ステートメント (Visual Basic)
For...Next ステートメント (Visual Basic)
For Each...Next ステートメント (Visual Basic)
If...Then...Else ステートメント (Visual Basic)
Select...Case ステートメント (Visual Basic)
Try...Catch...Finally ステートメント (Visual Basic)
While...End While ステートメント (Visual Basic)
With...End With ステートメント (Visual Basic)

If...Then...Else ステートメント (Visual Basic)

条件式の値を評価し、条件に応じて適切なステートメントを実行します。

```
If condition [ Then ]
    [ statements ]
[ ElseIf elseifcondition [ Then ]
    [ elseifstatements ] ]
[ Else
    [ elsestatements ] ]
End If
-or-
If condition Then [ statements ] [ Else [ elsestatements ] ]
```

指定項目

condition

必ず指定します。**True** または **False** に評価されるか、または暗黙的に **Boolean** に変換可能なデータ型に評価される式を指定する必要があります。

Then

単一行形式の場合は必ず指定します。複数行形式の場合は省略可能です。

statements

省略可能です。*condition* が真 (**True**) の場合に実行されるステートメントを、**If**...**Then** の後ろに続けて指定します。

elseifcondition

ElseIf が定義されている場合は必ず指定します。**True** または **False** に評価されるか、または暗黙的に **Boolean** に変換可能なデータ型に評価される式を指定する必要があります。

elseifstatements

省略可能です。*elseifcondition* が真 (**True**) の場合に実行されるステートメントを、**ElseIf**...**Then** の後ろに続けて指定します。

elsestatements

省略可能です。それ以前の *condition* や *elseifcondition* の式がどれも真 (**True**) でない場合に実行される一連のステートメントを指定します。

End If

If...**Then**...**Else** ブロックを終了します。

解説

単一行形式は、短く簡単な条件判断を行うときに使用します。複数行形式は、単一行形式に比べ、より構造化された柔軟な記述ができます。また、コードの読みやすさや保守性が向上し、デバッグもしやすくなります。

複数行形式の **If**...**Then**...**Else** が検出されると、*condition* がテストされます。*condition* が真 (**True**) の場合、**Then** の次のステートメントが実行されます。*condition* が偽 (**False**) の場合は、**ElseIf** ステートメントが順に評価されます。真 (**True**) に評価される *elseifcondition* が見つかると、その **Then** の直後に指定されたステートメントが実行されます。どの *elseifcondition* も **True** にならないか、**ElseIf** ステートメントが 1 つも定義されていない場合は、**Else** の次のステートメントが実行されます。**Then**、**ElseIf**、または **Else** に続くステートメントの実行が終わると、**End If** の次のステートメントからプログラムの実行が続けられます。

ヒント
評価する 1 つの式が、さまざまな値になる場合は、Select...Case ステートメント (Visual Basic) の方が適しています。

単一行形式の場合は、**If**...**Then** で判断した結果として、複数のステートメントを実行できます。すべてのステートメントを、コロンで区切って同じ行に記述する必要があります。次にコード例を示します。

```
If A > 10 Then A = A + 1 : B = B + A : C = C + B
```

複数行形式の場合は、**If** ステートメントのみを最初の行に定義する必要があります。**ElseIf**、**Else**、および **End If** の各ステートメントの前に記述できるのは、行ラベルだけです。複数行形式の **If**...**Then**...**Else** ブロックの最後には、必ず **End If** ステートメントを記述してください。

If ステートメントが複数行形式かどうかを判断するには、キーワード **Then** の後に何が続くかを調べます。ステートメント内の **Then** キーワードの後ろにコメント以外の記述があると、単一行の **If** ステートメントと判断されます。**Then** がない場合は、必ず複数行形式の **If**...**Then**...**Else** の先頭です。

ElseIf 句と **Else** 句はどちらも必要に応じて定義します。また、複数行形式の **If**...**Then**...**Else** では、**ElseIf** 句はいくつ指定してもかまいません。ただし、**Else** 句の後ろには何も指定できません。複数行形式は入れ子構造にできます。

使用例

If...**Then**...**Else** ステートメントの複数行形式と単一行形式を示すコード例は、次のとおりです。

VB

```
Dim number, digits As Integer
Dim myString As String
number = 53
If number < 10 Then
    digits = 1
ElseIf number < 100 Then
    digits = 2
Else
    digits = 3
End If
If digits = 1 Then myString = "One" Else myString = "More than one"
```

この例では、**ElseIf** 条件が **True** になり、`digits` に 2 の値が代入されます。その後、最後のステートメントで `myString` に "More than one" の値が代入されます。

参照

関連項目

#If...Then...#Else ディレクティブ
Choose 関数
Select...Case ステートメント (Visual Basic)
Switch 関数

Implements ステートメント

クラスまたは構造体の定義に実装する必要のある、1 つ以上のインターフェイス (インターフェイス メンバ) を指定します。

```
Implements interfacename [, ...]
-or-
Implements interfacename.interfacemember [, ...]
```

指定項目

interfacename

必ず指定します。インターフェイスを指定します。このインターフェイスのプロパティ、プロシージャ、およびイベントが、クラスまたは構造体の対応するメンバによって実装されます。

interfacemember

必ず指定します。実装されるインターフェイスのメンバを指定します。

解説

インターフェイスは、インターフェイスによってカプセル化されるメンバ (プロパティ、プロシージャ、およびイベント) を表すプロトタイプの集合体です。インターフェイスには、メンバの宣言だけが含まれます。これらのメンバを実装するのは、クラスおよび構造体です。

Implements ステートメントは、**Class** ステートメントまたは **Structure** ステートメントの直後に指定する必要があります。

インターフェイスを実装するときは、インターフェイスで宣言されたメンバをすべて実装する必要があります。メンバのいずれかを省略すると、構文エラーと見なされます。個々のメンバを実装するには、クラスまたは構造体にメンバを宣言するときに、Implements (Visual Basic) キーワードを (**Implements** ステートメントとは別に) 指定します。詳細については、「Implements キーワードおよび Implements ステートメント」を参照してください。

クラスは、プロパティおよびプロシージャを Private (Visual Basic) で実装できますが、そうするとクラスのインスタンスをインターフェイスの型で宣言した変数にキャストしない限り、これらのメンバにアクセスできなくなります。

使用例

Implements ステートメントを使って、インターフェイスのメンバを実装するコード例を次に示します。イベント、プロパティ、およびプロシージャを含むインターフェイスが、`ICustomerInfo` という名前で定義されています。`customerInfo` クラスは、インターフェイスに定義されたすべてのメンバを実装します。

VB

```
Public Interface ICustomerInfo
    Event updateComplete()
    Property customerName() As String
    Sub updateCustomerStatus()
End Interface

Public Class customerInfo
    Implements ICustomerInfo
    ' Storage for the property value.
    Private customerNameValue As String
    Public Event updateComplete() Implements ICustomerInfo.updateComplete
    Public Property CustomerName() As String Implements _
        ICustomerInfo.customerName
        Get
            Return customerNameValue
        End Get
        Set(ByVal value As String)
            ' The value parameter is passed to the Set procedure
            ' when the contents of this property are modified.
            customerNameValue = value
        End Set
    End Property

    Public Sub updateCustomerStatus() Implements _
        ICustomerInfo.updateCustomerStatus
```

```
        ' Add code here to update the status of this account.
        ' Raise an event to indicate that this procedure is done.
        RaiseEvent updateComplete()
    End Sub
End Class
```

customerInfo クラスでは 1 行のソース コードに **Implements** ステートメントを定義し、そのクラスが ICustomerInfo インターフェイスのすべてのメンバを実装することを示しています。その後、クラスの各メンバにおいて、メンバ宣言に **Implements** キーワードを指定し、そのインターフェイスのメンバが実装されることを示しています。

実装されたインターフェイスを使用する 2 つのプロシージャを次に示します。インターフェイスの実装をテストするには、これらのプロシージャをプロジェクトに追加して、testImplements プロシージャを呼び出します。

VB

```
Public Sub testImplements()
    ' This procedure tests the interface implementation by
    ' creating an instance of the class that implements ICustomerInfo.
    Dim cust As ICustomerInfo = New customerInfo()
    ' Associate an event handler with the event that is raised by
    ' the cust object.
    AddHandler cust.updateComplete, AddressOf handleUpdateComplete
    ' Set the customerName Property
    cust.customerName = "Fred"
    ' Retrieve and display the customerName property.
    MsgBox("Customer name is: " & cust.customerName)
    ' Call the updateCustomerStatus procedure, which raises the
    ' updateComplete event.
    cust.updateCustomerStatus()
End Sub

Sub handleUpdateComplete()
    ' This is the event handler for the updateComplete event.
    MsgBox("Update is complete.")
End Sub
```

参照

関連項目

Implements (Visual Basic)

Interface ステートメント (Visual Basic)

概念

Visual Basic のポリモーフィズムのしくみ

その他の技術情報

Visual Basic におけるインターフェイス

ポリモーフィズム

Imports ステートメント

参照されているプロジェクトとアセンブリに定義された名前空間またはプログラミング要素をインポートします。また、同じプロジェクト内に定義された名前空間や要素もインポートします。

```
Imports [ aliasname = ] namespace
-or-
Imports [ aliasname = ] namespace.element
```

指定項目

aliasname

省略可能です。コードが *namespace* を参照するために、完全修飾文字列の代わりに使用するインポート エイリアスまたは名前です。宣言された要素の名前 を参照してください。

namespace

必ず指定します。インポートされる名前空間の完全修飾名を指定します。入れ子構造のどのレベルにある名前空間の文字列でも指定できます。

element

省略可能です。名前空間に宣言されているプログラミング要素の名前を指定します。任意のコンテナ 要素を指定できます。

解説

1 つのファイルで使用できる **Imports** ステートメントの数に制限はありません。このステートメントの前には、**Option Strict** ステートメントなどの任意のオプションを宣言する必要があり、後には **Module** ステートメントや **Class** ステートメントなどの任意のプログラミング要素を宣言する必要があります。

Imports を使用できるのはファイル レベルだけです。つまり、インポートの宣言コンテキストは、ソース ファイルである必要があり、名前空間、クラス、構造体、モジュール、インターフェイス、プロシージャ、またはブロックでは宣言できません。

1 つ以上の名前空間で宣言されているのと同名の項目を使用する必要がある場合は、インポート エイリアスが便利です。詳細と使用例については、「同じ名前を持つ複数の変数がある場合に参照を解決する」の「同じ名前のクラス」を参照してください。

Imports ステートメントを指定しても、他のプロジェクトとアセンブリの要素を自分のプロジェクトで使用できるようになるわけではありません。インポートは参照を設定する代わりには使用できません。もともとプロジェクトで利用可能な名前を修飾する必要がなくなるだけです。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」の「コンテナ要素のインポート」を参照してください。

規則

- **エイリアス名** *aliasname* と同じ名前のモジュールのレベルでメンバを宣言することはできません。宣言した場合、Visual Basic コンパイラは宣言されたメンバにのみ *aliasname* を使用し、これをインポート エイリアスとして認識しなくなります。
- **名前空間名** 入れ子になった名前空間の名前空間名または文字列を 1 つで指定できます。より高いレベルの名前空間から入れ子にされた名前空間を区切るには、ピリオド (.) を使います。次に例を示します。

  ```
  Imports System.Collections.Generic
  ```

- **要素の型** *element* を指定する場合、その要素はコンテナ要素であることが必要です。つまり、他の要素を格納できるプログラミング要素を指定する必要があります。コンテナ要素にはクラス、構造体、モジュール、インターフェイス、および列挙体が含まれます。

動作

- **スコープ** **Imports** ステートメントによって利用可能になる要素のスコープは、*element* を指定するかどうかによって変わります。*namespace* だけを指定した場合、その名前空間の一意な名前を持つすべてのメンバ、およびその名前空間に含まれるコンテナ要素のメンバを、修飾子なしで使用できます。*namespace* と *element* を指定した場合は、その要素のメンバだけを修飾子なしで使用できます。
- **修飾** 名前空間やコンテナ要素の外部にあるコードでは、通常はその名前空間のコンテナ要素の名前でメンバ名を修飾します。**Imports** ステートメントを使用すると、プロジェクトから同名の別のメンバにアクセスできる場合を除いて、このような修飾が不要になります。このような場合は、各 **Imports** ステートメントに *aliasname* を指定します。そうすれば、同名のメンバをインポート エイリアスで修飾するだけで済みます。

使用例

Microsoft.VisualBasic.Strings クラスをインポートして、`str` というエイリアスを設定し、このエイリアスを使って Left メソッドにアクセスするコード例を次に示します。

VB

```
' Place Imports statements at the top of your program.
Imports str = Microsoft.VisualBasic.Strings
```

VB

```
Class testClass1
    Sub showHello()
        ' Display only the word "Hello"
        MsgBox(str.Left("Hello World", 5))
    End Sub
End Class
```

この例では、`Microsoft` 内の `VisualBasic` に入れ子になった名前空間、`Strings` がインポートされています。MsgBox 関数 (Visual Basic) が **Left** メソッドにアクセスするとき、`str` というエイリアスを全修飾文字列 `Microsoft.VisualBasic.Strings` の代わりに使用できます。

参照

関連項目

Namespace ステートメント

概念

Visual Studio の .NET Framework クラス ライブラリの概要

Visual Basic における名前空間

その他の技術情報

Visual Basic と .NET Framework

Inherits ステートメント

現在のクラスまたはインターフェイスの属性、変数、プロパティ、プロシージャ、およびイベントを別のクラスまたは一連のインターフェイスから継承します。

```
Inherits basetypenames
```

指定項目
basetypenames

必ず指定します。このクラスの派生元のクラスの名前です。

または

このインターフェイスの派生元のインターフェイスの名前です。複数の名前を区切るには、コンマを使います。

解説

Inherits ステートメントを使用する場合は、クラス定義内またはインターフェイス定義内の空行やコメントではない 1 行目に配置する必要があります。このステートメントは、**Class** ステートメントまたは **Interface** ステートメントの直後に追加します。

Inherits は、クラスまたはインターフェイスの中でのみ使用できます。つまり、ソース ファイル、名前空間、構造体、モジュール、プロシージャ、またはブロックをコンテキストとして継承を宣言することはできません。

規則

- **クラスの継承。** クラスで **Inherits** ステートメントを使用する場合は、1 つの基本クラスだけを指定できます。

  クラスは、それ自身の入れ子のクラスを継承できません。

- **インターフェイスの継承。** インターフェイスで **Inherits** ステートメントを使用する場合は、1 つ以上の基本インターフェイスを指定できます。2 つのインターフェイスに同じ名前でメンバが定義されている場合でも、それらのインターフェイスを継承できます。その場合、実装コードでは、実装するメンバを指すために名前の修飾を使う必要があります。

  インターフェイスには、それ自身より厳しいアクセス レベルを持つ別のインターフェイスを継承することはできません。たとえば、**Public** インターフェイスは **Friend** インターフェイスを継承できません。

  インターフェイスは、それ自身の入れ子のインターフェイスを継承できません。

.NET Framework で使用されるクラス継承の例として、ArgumentException クラスが挙げられます。このクラスは、SystemException クラスを継承します。この継承によって、**ArgumentException** には、システム例外で必要とされる、Message プロパティや ToString メソッドなどのすべての定義済みのプロパティおよびプロシージャが提供されます。

.NET Framework で使用されるインターフェイス継承の例として、ICollection インターフェイスが挙げられます。このインターフェイスは、IEnumerable インターフェイスを継承します。これにより、**ICollection** はコレクションを走査するために必要な列挙子の定義を継承します。

使用例

次の例は、**Inherits** ステートメントを使用して、`thisClass` というクラスに `anotherClass` という基本クラスのすべてのメンバを継承する方法を示しています。

VB

```
Public Class thisClass
    Inherits anotherClass
    ' Add code to override, overload, or extend members
    ' inherited from the base class.
    ' Add new variable, property, procedure, and event declarations.
End Class
```

次の例は、複数のインターフェイスを継承する方法を示しています。

VB

```
Public Interface thisInterface
    Inherits IComparable, IDisposable, IFormattable
    ' Add new property, procedure, and event definitions.
End Interface
```

これにより、`thisInterface` というインターフェイスには、IComparable、IDisposable、および IFormattable の各インターフェイスにあるすべての定義が含まれるようになります。これらの継承されたメンバは、2 つのオブジェクトを特定の型として比較する機能、割り当てられたリソースを解放する機能、およびオブジェクトの値を **String** で表現する機能を備えています。`thisInterface` を実装するクラスは、すべての基本インターフェイスのすべてのメンバを実装する必要があります。

参照

関連項目

MustInherit

NotInheritable

MyBase

MyClass

概念

Visual Basic でのインターフェイス実装例

その他の技術情報

Visual Basic の継承

Interface ステートメント (Visual Basic)

インターフェイスの名前を宣言し、インターフェイスに含まれるメンバの定義を開始します。

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] _
Interface name [ ( Of typelist ) ]
    [ Inherits interfacenames ]
    [ [ modifiers ] Property membername ]
    [ [ modifiers ] Function memberame ]
    [ [ modifiers ] Sub memberame ]
    [ [ modifiers ] Event memberame ]
    [ [ modifiers ] Interface memberame ]
    [ [ modifiers ] Class memberame ]
    [ [ modifiers ] Structure memberame ]
End Interface
```

指定項目

attributelist

省略可能です。「属性リスト」を参照してください。

accessmodifier

省略可能です。次のいずれかを指定できます。

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Visual Basic でのアクセス レベル を参照してください。

Shadows

省略可能です。「Shadows」を参照してください。

name

必ず指定します。このインターフェイスの名前です。宣言された要素の名前 を参照してください。

Of

省略可能です。ジェネリック インターフェイスであることを指定します。

typelist

Of キーワードを使用する場合は必ず指定します。このインターフェイスの型パラメータのリストを指定します。「型リスト」を参照してください。

Inherits

省略可能です。このインターフェイスが別のインターフェイスの属性およびメンバを継承することを指定します。Inherits ステートメント を参照してください。

interfacenames

Inherits ステートメントを使用する場合は必ず指定します。このインターフェイスの派生元のインターフェイスの名前を指定します。

modifiers

省略可能です。定義するインターフェイス メンバに適した修飾子を指定します。

Property

省略可能です。インターフェイスのメンバであるプロパティを定義します。

Function

省略可能です。インターフェイスのメンバである **Function** プロシージャを定義します。

Sub

省略可能です。インターフェイスのメンバである **Sub** プロシージャを定義します。

Event

省略可能です。インターフェイスのメンバであるイベントを定義します。

Interface

省略可能です。このインターフェイスの内部に入れ子になったインターフェイスを定義します。入れ子になったインターフェイスの定義は、**End Interface** ステートメントで終了する必要があります。

Class

省略可能です。インターフェイスのメンバであるクラスを定義します。このクラスの定義は、**End Class** ステートメントで終了する必要があります。

Structure

省略可能です。インターフェイスのメンバである構造体を定義します。このメンバの構造体の定義は、**End Structure** ステートメントで終了する必要があります。

membername

インターフェイスのメンバとして定義したプロパティ、プロシージャ、イベント、インターフェイス、クラスまたは構造体のそれぞれに必ず指定します。メンバの名前です。

End Interface

Interface の定義を終了します。

解説

インターフェイスにはプロパティやプロシージャなど、クラスや構造体で実装可能なメンバをまとめて定義します。インターフェイスにはメンバのシグネチャだけを定義し、内部の処理は定義しません。

クラスや構造体でインターフェイスを実装する場合は、そのインターフェイスに定義されたすべてのメンバのコードを定義します。最後に、アプリケーションがそのクラスまたは構造体からインスタンスを作成した時点でオブジェクトが存在し、メモリ内で実行されます。詳細については、「Visual Basic におけるオブジェクト指向プログラミング」および「Visual Basic におけるインターフェイス」を参照してください。

インターフェイスとクラスの比較については、「インターフェイスの概要」を参照してください。

Interface は、名前空間またはモジュール レベルでのみ使用できます。つまり、インターフェイスの宣言コンテキストはソース ファイル、名前空間、クラス、構造体、モジュール、またはインターフェイスのいずれかである必要があり、プロシージャまたはブロックでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

インターフェイスには既定で Friend (Visual Basic) のアクセス レベルが設定されます。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。詳細については、「Visual Basic でのアクセス レベル」を参照してください。

規則

- **インターフェイスの入れ子** インターフェイスを別のインターフェイスの内部に定義できます。外側のインターフェイスを包含インターフェイスと呼び、内側のインターフェイスを入れ子になったインターフェイスと呼びます。
- **メンバの宣言** プロパティまたはプロシージャをインターフェイスのメンバとして宣言する場合は、そのシグネチャだけを定義します。シグネチャには要素の種類 (プロパティまたはプロシージャ)、パラメータとその型、および戻り値の型が含まれます。このため、メンバの定義にはコードを 1 行だけ使用し、**End Function** や **End Property** などの終了ステートメントをインターフェイスに定義することはできません。

  これに対し、列挙体や構造体、または入れ子になったクラスやインターフェイスを定義するときには、そのデータ メンバを含める必要があります。
- **メンバ修飾子** モジュールのメンバを定義するときには、アクセス修飾子を使用できません。また、Overloads を除くどのプロシージャ修飾子 (Shared (Visual Basic)など) も指定できません。どのメンバでも、Shadows を使って宣言できます。また、プロパティを定義するときには、ReadOnly (Visual Basic) や WriteOnly の他に Default (Visual Basic) を使用できます。
- **継承** インターフェイスに Inherits ステートメント を使用して、1 つ以上の基本インターフェイスを指定できます。2 つのインターフェイスに同じ名前のメンバが定義されている場合でも、その 2 つを継承できます。その場合は、実装先のコードで名前を修飾して、どちらのメンバを実装しているのか指定する必要があります。

インターフェイスはより制限の高いアクセス レベルを持つ別のインターフェイスは継承できません。たとえば、**Public** インターフェイスは **Friend** インターフェイスを継承できません。

インターフェイスは、自分の中に入れ子になっているインターフェイスを継承できません。

- **実装**クラスに Implements (Visual Basic) ステートメントを使ってインターフェイスを実装するときには、そのインターフェイスに定義されているすべてのメンバを実装する必要があります。さらに、実装先のコードでは、各シグネチャがインターフェイスに定義された対応するシグネチャと完全に一致していることが必要です。ただし、実装先のコードでのメンバの名前は、インターフェイスに定義されたメンバ名と同じでなくてもかまいません。

  クラスにプロシージャを実装するとき、そのプロシージャに **Shared** を指定できません。

- **既定のプロパティ。**インターフェイスには、1 つまでのプロパティを既定のプロパティとして指定できます。既定のプロパティは、プロパティ名を使用せずに参照できます。既定のプロパティを指定するには、Default (Visual Basic) 修飾子を使って宣言します。詳細については、「既定のプロパティ」を参照してください。

  つまり、インターフェイスに既定のプロパティを定義できるのは、インターフェイスが何も継承していない場合だけです。

動作

- **アクセス レベル**すべてのインターフェイス メンバは、暗黙的に Public (Visual Basic) アクセスになります。メンバを定義するとき、アクセス修飾子は使用できません。ただし、インターフェイスを実装するクラスで、実装された各メンバに対してアクセス レベルを宣言できます。

  クラスのインスタンスを変数に割り当てる場合、そのメンバのアクセス レベルは、変数のデータ型が基のインターフェイスかそれとも実装先のクラスかによって変わります。次に例を示します。

  VB

```
Public Interface IDemo
    Sub doSomething()
End Interface
Public Class implementIDemo
    Implements IDemo
    Private Sub doSomething() Implements IDemo.doSomething
    End Sub
End Class
Dim varAsInterface As IDemo = New implementIDemo()
Dim varAsClass As implementIDemo = New implementIDemo()
```

  `varAsInterface` を指定してクラス メンバにアクセスする場合は、すべてのメンバがパブリック アクセスになります。ただし、`varAsClass` を指定してメンバにアクセスする場合、**Sub** プロシージャである `doSomething` はプライベート アクセスになります。

- **スコープ**インターフェイスのスコープは名前空間、クラス、構造体、またはモジュール全体です。

  インターフェイスのすべてのメンバのスコープは、インターフェイス全体になります。

- **有効期間**インターフェイスおよびそのメンバには、有効期間がありません。クラスがインターフェイスを実装し、そのクラスのインスタンスとしてオブジェクトが作成されると、そのオブジェクトが有効期間 (オブジェクトを実行しているアプリケーションが終了するまで) を持ちます。詳細については、「Class ステートメント (Visual Basic)」の「有効期間」を参照してください。

使用例

次の例は **Interface** ステートメントを使用して、`thisInterface` という名前のインターフェイスを定義します。これを実装するには、**Property** ステートメントと **Function** ステートメントを使用する必要があります。

VB

```
Public Interface thisInterface
    Property thisProp(ByVal thisStr As String) As Char
    Function thisFunc(ByVal thisInt As Integer) As Integer
End Interface
```

インターフェイス内で **Property** ステートメントと **Function** ステートメントで開始されたブロックが、**End Property** と **End Function** で終了していない点に注意してください。インターフェイスには、そのメンバのシグネチャのみを定義します。**Property** ブロックと **Function** ブロックの完全

な定義は、`thisInterface` を実装するクラスに記述されています。

参照

関連項目

Class ステートメント (Visual Basic)
Module ステートメント
Structure ステートメント
Property ステートメント
Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)

概念

Visual Basic におけるジェネリック型

その他の技術情報

Visual Basic におけるインターフェイス

Mid ステートメント

文字列型 (**String**) の変数内の指定した文字数分を別の文字列に置き換えます。

```
Mid( _
   ByRef Target As String, _
   ByVal Start As Integer, _
   Optional ByVal Length As Integer _
) = StringExpression
```

指定項目

Target

必ず指定します。変更する文字列型 (**String**) の変数の名前です。

Start

必ず指定します。整数 (**Integer**) を指定します。*Target* 内で、テキストの置換を開始する文字位置です。*Start* のインデックスは 1 から始まります。

Length

省略できます。整数 (**Integer**) を指定します。置き換える文字数を指定します。省略すると、*String* 全体が置き換えられます。

StringExpression

必ず指定します。*Target* の該当部分を置き換える文字列型 (**String**) の式です。

例外

例外の種類	エラー番号	条件
ArgumentException	5	*Start* <= 0 または *Length* < 0 です。

非構造化エラー処理を使用する Visual Basic 6.0 アプリケーションをアップグレードする場合は、"エラー番号" の列を参照してください(エラー番号を Number プロパティ (Err オブジェクト) と比較することもできます)。ただし、可能であれば、このようなエラー制御は Visual Basic の構造化例外処理の概要 に置き換えることを検討してください。

解説

置換される文字数は、必ず *Target* の文字数以下になります。

Visual Basic には Mid 関数 (Visual Basic) と **Mid** ステートメントがあります。この 2 つの要素はどちらも文字列内の指定された数の文字を置換しますが、**Mid** 関数が文字列のコピーを処理して返すのに対し、**Mid** ステートメントは文字列を置換します。詳細については、「Mid 関数 (Visual Basic)」を参照してください。

メモ：

以前のバージョンの Visual Basic では、**MidB** ステートメントは文字数ではなくバイト数に基づいて部分文字列を置換していました。これは主に、2 バイト文字セット (DBCS) アプリケーションで文字列を変換するために使用します。Visual Basic のすべての文字列は Unicode で、**MidB** は現在サポートされていません。

使用例

Mid ステートメントを使って、文字列変数内の指定された文字数をある文字列の文字に置き換える例を次に示します。

VB

```
Dim TestString As String
' Initializes string.
TestString = "The dog jumps"
' Returns "The fox jumps".
Mid(TestString, 5, 3) = "fox"
' Returns "The cow jumps".
Mid(TestString, 5) = "cow"
' Returns "The cow jumpe".
Mid(TestString, 5) = "cow jumped over"
```

```
' Returns "The duc jumpe".
Mid(TestString, 5, 3) = "duck"
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Strings**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll)

参照

関連項目

Mid 関数 (Visual Basic)

その他の技術情報

Visual Basic における文字列

Visual Basic の文字列の概要

Module ステートメント

モジュールの名前を宣言し、モジュールを構成する変数、プロパティ、イベント、およびプロシージャの定義を提供します。

```
[ <attributelist> ] [ accessmodifier ]  Module name
    [ statements ]
End Module
```

指定項目

attributelist

省略可能です。属性リスト を参照してください。

accessmodifier

省略可能です。次のいずれかになります。

- Public
- Friend

Visual Basic でのアクセス レベル を参照してください。

name

必ず指定します。このモジュールの名前です。宣言された要素の名前 を参照してください。

statements

省略可能です。モジュールの変数、プロパティ、イベント、プロシージャ、および入れ子にされた型を定義するステートメントを指定します。

End Module

Module 定義を終了します。

解説

Module ステートメントは、名前空間全体で使用可能な参照型を定義します。モジュール (標準モジュールとも呼ばれる) は、クラスと似ていますが、重要な違いがいくつかあります。モジュールのインスタンスは厳密に 1 つだけであり、作成する必要や変数に代入する必要はありません。モジュールは、継承をサポートすることもインターフェイスを実装することもありません。モジュールは、クラスや構造体が型であるという意味での型ではないことに注意してください。モジュールのデータ型を持つプログラミング要素を宣言することはできません。

Module は、名前空間レベルでのみ使用できます。つまり、モジュールの宣言コンテキストは、ソース ファイルまたは名前空間のどちらかである必要があり、クラス、構造体、モジュール、インターフェイス、プロシージャ、およびブロックでは宣言できません。モジュールは、別のモジュールまたは型に入れ子にできません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

モジュールの有効期間は、プログラムと同じです。すべてのメンバは **Shared** であるため、メンバの有効期間もプログラムと同じです。

既定で、モジュールのアクセス レベルは Friend (Visual Basic) です。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。詳細については、「Visual Basic でのアクセス レベル」を参照してください。

モジュールのすべてのメンバは、暗黙的に **Shared** になります。

クラスとモジュール

両者には共通する点が多くありますが、重要な相違点もあります。

- **用語。** 以前のバージョンの Visual Basic では、モジュールにはクラス モジュール (.cls files) と標準モジュール (.bas files) という 2 つの種類がありました。現在のバージョンでは、呼び名がそれぞれクラスとモジュールに変わっています。
- **共有メンバ。** クラスのメンバは、共有メンバまたはインスタンス メンバのどちらかにすることができます。
- **オブジェクト指向。** クラスはオブジェクト指向ですが、モジュールはオブジェクト指向ではありません。したがって、クラスだけがオブジェクトとしてインスタンス化できます。詳細については、「クラスとモジュール」を参照してください。

規則

- **修飾子** すべてのモジュール メンバは、暗黙的に Shared (Visual Basic) になります。メンバを宣言するときに **Shared** キーワードは使用

できません。また、メンバの共有の状態は変更できません。

- **継承。**モジュールは、Object 以外の型を継承できません。すべてのモジュールが、この型を継承します。特に、モジュールは別のモジュールを継承できません。

  たとえ **Object** を指定するためであっても、Inherits ステートメント をモジュールの定義で使用することはできません。

- **既定のプロパティ。**既定のプロパティ をモジュール内に定義することはできません。

動作

- **アクセス レベル**モジュールの内部では、各メンバを独自のアクセス レベルで宣言できます。モジュール メンバのアクセス レベルは、既定で Public (Visual Basic) になりますが、変数と定数だけは Private (Visual Basic) が既定のアクセス レベルです。モジュールのアクセス レベルがメンバのレベルより厳しい場合、指定したモジュール アクセス レベルが優先されます。

- **スコープ**モジュールのスコープは、名前空間全体です。

  モジュール メンバのスコープは、モジュール全体です。すべてのメンバに型の上位変換が行われ、メンバのスコープがモジュールのある名前空間に上位変換されることに注意してください。詳細については、「型の上位変換」を参照してください。

- **修飾**1 つのプロジェクトが複数のモジュールを持つことができます。また、名前の同じメンバを複数のモジュールに宣言できます。ただし、モジュールの外部にあるメンバを参照する場合は、適切なモジュール名を使ってメンバへの参照を修飾する必要があります。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」を参照してください。

使用例
VB

```
Public Module thisModule
    Sub Main()
        Dim userName As String = InputBox("What is your name?")
        MsgBox("User name is" & userName)
    End Sub
    ' Insert variable, property, procedure, and event declarations.
End Module
```

参照
関連項目
Class ステートメント (Visual Basic)
Namespace ステートメント
Structure ステートメント
Interface ステートメント (Visual Basic)
Property ステートメント
概念
型の上位変換

Namespace ステートメント

名前空間の名前を宣言し、後続のソース コードがその名前空間内にコンパイルされるようにします。

```
Namespace { name | name.name }
    [ componenttypes ]
End Namespace
```

指定項目

name

必ず指定します。名前空間を識別する一意の前を指定します。有効な Visual Basic 識別子を指定する必要があります。詳細については、「宣言された要素の名前」を参照してください。

componenttypes

省略可能です。名前空間を構成する要素を指定します。列挙型、構造体、インターフェイス、クラス、モジュール、デリゲート、他の名前空間などがあります。

End Namespace

Namespace ブロックを終了します。

解説

名前空間は、コードを組織化する枠組みとして使用されます。他のプログラムやアプリケーションに公開されるプログラミング要素を分類および表現する手段となります。名前空間は、クラスや構造体が型であるという意味での型ではないことに注意してください。名前空間のデータ型を持つプログラミング要素を宣言することはできません。

Namespace ステートメントは、ファイル レベルまたは名前空間レベルでのみ使用できます。つまり、名前空間の宣言コンテキストは、ソース ファイルまたは別の名前空間のどちらかである必要があり、クラス、構造体、モジュール、インターフェイス、またはプロシージャでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

名前空間は、**Public** アクセス レベルであるかのように扱われます。名前空間には、同じプロジェクト内のコード、プロジェクトを参照する他のプロジェクト、およびプロジェクトからビルドされたアセンブリからアクセスできます。

規則

- **属性** 名前空間には属性を適用できません。属性は、アセンブリのメタデータに情報を提供します。この情報は、名前空間のようなソース分類子にとって意味を持ちません。
- **修飾子** 名前空間には、アクセス修飾子またはプロシージャ修飾子、またはその他の修飾子を適用できません。型ではないので、このような修飾子は意味を持ちません。
- **入れ子。** 名前空間は、別の名前空間の内部に宣言できます。宣言できる入れ子のレベルに厳密な制限はありませんが、最深部の名前空間に宣言した要素に他のコードからアクセスする場合、入れ子の階層にあるすべての名前空間を含む修飾文字列を使用する必要があることに注意してください。

動作

- **アクセス レベル** 名前空間レベルで宣言したプログラミング要素、つまり他の要素の内部ではなく名前空間の内部で宣言したプログラミング要素は、アクセス レベルが **Public** または **Friend** になります。特に指定しない場合、このような要素のアクセス レベルは、既定で **Friend** です。名前空間レベルで宣言できる要素は、クラス、構造体、モジュール、インターフェイス、列挙、およびデリゲートです。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。
- **コンパイル。** **Namespace** ステートメントの後に宣言したすべてのプログラミング要素は、その名前空間に属します。最後に宣言された名前空間の中に要素をコンパイルする処理は、**End Namespace** ステートメントまたは別の **Namespace** ステートメントが現れるまで続けられます。
- **名前空間への追加。** 名前空間が既に定義されている場合、プロジェクトの外部にある名前空間でも、プログラミング要素をそこに追加できます。**Namespace** ステートメントを使って、後続の要素をその名前空間内にコンパイルするように Visual Basic に指示します。

  次のコード例は、新しいジェネリックなリスト クラスのスケルトンを定義し、それを System.Collections.Generic 名前空間に追加します。

  ```
  Namespace System.Collections.Generic
  ```

```
    Class specialSortedList(Of T)
        Inherits List(Of T)
        ' Insert code to define the special generic list class.
    End Class
End Namespace
```

- **ルート名前空間。** プロジェクト内のすべての名前空間の名前は、ルート名前空間に基づきます。Visual Studio では、プロジェクト名がプロジェクト内のすべてのコードの既定のルート名前空間として割り当てられます。たとえば、プロジェクト名が `Payroll` である場合、そのプログラミング要素は `Payroll` 名前空間に属します。`Namespace funding` と宣言した場合、この名前空間の完全名は `Payroll.funding` になります。

  前記のジェネリックなリスト クラスの例のように、既存の名前空間を **Namespace** ステートメントで指定する場合は、ルート名前空間を null 値に設定できます。この設定を行うには、[プロジェクト] メニューの [プロジェクトのプロパティ] をクリックし、[ルート名前空間] ボックスをクリアして空にします。この設定を前記のジェネリックなリスト クラスの例で行わなかった場合、Visual Basic コンパイラは `System.Collections.Generic` を `Payroll` プロジェクト内の `Payroll.System.Collections.Generic` という完全名を持つ新しい名前空間と見なします。

  Global キーワードを使って、プロジェクトの外部で定義された名前空間の要素を参照することもできます。この方法を使うと、プロジェクト名をルート名前空間として使い続けることができます。それにより、意に反してプログラミング要素が既存の名前空間の要素とマージされる可能性は少なくなります。

トラブルシューティング

- **未定義のデータ型。** ルート名前空間が、予期しない名前空間の連結を招く場合があります。プロジェクトの外部で定義された名前空間への参照を作成すると、Visual Basic コンパイラでは、その名前空間をルート名前空間内の入れ子の名前空間として解釈します。その場合、コンパイラでは、外部の名前空間で定義されたデータ型を認識しません。この状況を回避するには、「ルート名前空間」で説明した手順でルート名前空間を null 値に設定するか、**Global** キーワードを使って外部名前空間の要素にアクセスします。

使用例

次のコード例は、入れ子の関係になるように 2 つの名前空間を宣言します。

VB

```
Namespace n1
    Namespace n2
        Class a
            ' Insert class definition.
        End Class
    End Namespace
End Namespace
```

入れ子になった複数の名前空間を 1 行で宣言するコード例を次に示します。これは前のコード例と同じ意味を持ちます。

VB

```
Namespace n1.n2
    Class a
        ' Insert class definition.
    End Class
End Namespace
```

次のコード例は、前の例で定義したクラスにアクセスします。

VB

```
Dim instance As New n1.n2.a
```

参照

関連項目

Imports ステートメント

Global

概念

宣言された要素の名前

Visual Basic における名前空間

On Error ステートメント (Visual Basic)

エラー処理ルーチンを有効にして、プロシージャ内でルーチンの場所を指定します。また、エラー処理ルーチンを無効にする場合にも使用できます。

On Error ステートメントを使用しないと、発生するすべてのランタイム エラーが致命的なエラーになります。つまり、エラー メッセージが表示され、実行が停止します。

可能な場合には、構造化されていない例外処理と **On Error** ステートメントに頼るよりも、構造化例外処理を使用することをお勧めします。詳細については、「Visual Basic での構造化例外処理」を参照してください。

```
On Error { GoTo [ line | 0 | -1 ] | Resume Next }
```

指定項目

GoTo *line*

必須の *line* 引数で指定した行から始まるエラー処理ルーチンを有効にします。引数 *line* には、任意の行ラベルまたは行番号を指定します。ランタイム エラーが発生すると、指定した行に制御が分岐され、エラー ハンドラがアクティブになります。**On Error** ステートメントと同じプロシージャ内の行を指定する必要があります。それ以外の行を指定すると、コンパイル エラーが発生します。

GoTo 0

現在のプロシージャで有効になっていたエラー ハンドラを無効にし、**Nothing** にリセットされます。

GoTo -1

現在のプロシージャで有効になっていた例外を無効にし、**Nothing** にリセットされます。

Resume Next

ランタイム エラーが発生すると、エラーが発生したステートメントの直後のステートメントに制御が移り、そのステートメントから実行が継続されます。オブジェクトにアクセスするときは、**On Error GoTo** ではなくこの形式を使用します。

解説

"有効な" エラー ハンドラとは、**On Error** ステートメントによって有効化されているものを指します。"アクティブな" エラー ハンドラとは、エラー処理を行っている最中の有効なハンドラのことを指します。

エラー ハンドラがアクティブな間 (エラーの発生から、**Resume**、**Exit Sub**、**Exit Function**、または **Exit Property** の各ステートメントまで) にエラーが発生した場合は、現在のプロシージャのエラー ハンドラはエラーを処理できません。制御は呼び出しプロシージャに戻ります。

呼び出しプロシージャに有効なエラー ハンドラがある場合は、そのハンドラがアクティブになり、エラーを処理します。呼び出しプロシージャのエラー ハンドラもアクティブである場合は、有効であってもアクティブではないエラー ハンドラが見つかるまで、元の呼び出しプロシージャに制御が順次戻されます。有効であってもアクティブではないエラー ハンドラが見つからない場合は、実際にエラーが起こった位置で致命的なエラーが発生します。

エラー ハンドラが呼び出しプロシージャに制御を戻すたびに、そのプロシージャが現在のプロシージャになります。いずれかのプロシージャのエラー ハンドラによってエラーが処理された後は、**Resume** ステートメントで指定された位置から現在のプロシージャの実行が再開されます。

メモ :

エラー処理ルーチンは、**Sub** プロシージャまたは **Function** プロシージャではありません。行ラベルまたは行番号でマークされたコードのセクションです。

Number プロパティ

エラー処理ルーチンは、**Err** オブジェクトの **Number** プロパティの値に基づいて、エラーの原因を判断します。このルーチンでは、他のエラーが発生する前、またはエラーを引き起こす可能性のあるプロシージャが呼び出される前に、**Err** オブジェクトの関連するプロパティ値をテストまたは保存する必要があります。**Err** オブジェクトのプロパティ値には、直前に発生したエラーが反映されます。**Err.Number** に関連付けられているエラー メッセージは **Err.Description** に格納されます。

Throw ステートメント

Err.Raise メソッドによってエラーを発生させたときは、**Exception** プロパティが Exception クラスの新しいインスタンスに設定されます。派生した例外型の例外を発生させるには、**Throw** ステートメントを使用します。**Throw** ステートメントは、スローする例外インスタンスを指定する 1 つのパラメータを持っています。これらの機能を既存の例外処理サポートで使用する方法を次に示します。

VB

```
    On Error GoTo Handler
    Throw New DivideByZeroException()
Handler:
    If (TypeOf Err.GetException() Is DivideByZeroException) Then
    ' Code for handling the error is entered here.
    End If
```

On Error GoTo ステートメントは、例外クラスに関係なく、すべてのエラーをトラップするという点に注意してください。

On Error Resume Next

On Error Resume Next ステートメントは、ランタイム エラーを発生させたステートメントの直後にあるステートメント、または、**On Error Resume Next** ステートメントが入っているプロシージャから最後に呼び出しを行ったステートメントの直後のステートメントを使用して、実行を継続させます。これにより、ランタイム エラーが発生しても処理を続けることができます。プロシージャ内の別の場所に制御を移さなくても、エラー処理ルーチンをエラーが発生する可能性がある場所に配置できます。**On Error Resume Next** ステートメントは、別のプロシージャが呼び出されると非アクティブになるため、実行時にインライン エラー処理が必要な場合は、呼び出しルーチンごとに **On Error Resume Next** ステートメントを実行する必要があります。

メモ：

他のオブジェクトへのアクセス中に生成されたエラーを処理する場合は、**On Error GoTo** よりも **On Error Resume Next** 構成要素の方が適しています。オブジェクトと対話した後に毎回 **Err** を調べれば、そのコードがどのオブジェクトにアクセスしたかが明らかになります。どのオブジェクトが **Err.Number** にエラー コードを設定したかと、どのオブジェクトが最初にエラーを生成したかについても確認できます (このオブジェクトは **Err.Source** で指定されます)。

On Error GoTo 0

On Error GoTo 0 は、現在のプロシージャ内のエラー処理を無効にします。このステートメントは、プロシージャに番号 0 の行があったとしても、その行をエラー処理コードの開始位置にするわけではありません。**On Error GoTo 0** ステートメントを指定しなくても、エラー ハンドラはプロシージャが終了すると自動的に無効になります。

On Error GoTo -1

On Error GoTo -1 は、現在のプロシージャ内の例外を無効にします。このステートメントは、プロシージャに番号 -1 の行があったとしても、その行をエラー処理コードの開始位置にするわけではありません。**On Error GoTo -1** ステートメントを指定しなくても、例外はプロシージャが終了すると自動的に無効になります。

エラーが発生しないときにエラー処理コードが実行されないようにするには、次のコード例のように、エラー処理ルーチンの直前に **Exit Sub**、**Exit Function**、または **Exit Property** ステートメントを指定します。

VB

```
Public Sub InitializeMatrix(ByVal Var1 As Object, ByVal Var2 As Object)
   On Error GoTo ErrorHandler
   ' Insert code that might generate an error here
   Exit Sub
ErrorHandler:
   ' Insert code to handle the error here
   Resume Next
End Sub
```

この例では、エラー処理コードが **Exit Sub** ステートメントに続く形で **End Sub** ステートメントよりも前に置かれ、プロシージャのフローからは分離されています。エラー処理コードは、プロシージャ内の任意の場所に指定できます。

トラップされないエラー

オブジェクトを実行可能ファイルとして実行している場合、オブジェクト内のトラップされないエラーはコントローラ アプリケーションに戻されます。開発環境では、トラップされないエラーがコントローラ アプリケーションに戻されるのは、適切なオプションが設定されている場合だけです。デバッグ時に設定するオプションの説明、そのオプションの設定方法、およびホストがクラスを作成できるかどうかについては、ホスト アプリケーションのドキュメントを参照してください。

他のオブジェクトにアクセスするオブジェクトを作成する場合は、アクセス先のオブジェクトが戻す未処理のエラーを処理する必要があります。このオブジェクト内でエラーを処理できない場合は、**Err.Number** 内のエラー コードをこのオブジェクトの独自のエラーのいずれかに割り当てて、もう 1 レベル上の呼び出し元に渡します。独自のエラーを指定するには、独自のエラー コードを **VbObjectError** 定数に追加する必要がありま

す。たとえば、独自のエラー コードが 1052 の場合は、次のように割り当てます。

VB

```
Err.Number = vbObjectError + 1052
```

注意

Windows ダイナミック リンク ライブラリ (DLL: Dynamic Link Library) の呼び出し中にシステム エラーが起こった場合は例外が発生しないため、Visual Basic エラー トラップではトラップされません。DLL 関数を呼び出すときは、API の仕様に従ってそれぞれの戻り値を調べ、処理が正しく実行されたかどうかを確認する必要があります。エラーの場合には、**Err** オブジェクトの **LastDLLError** プロパティの値を調べます。

使用例

この例では、**On Error GoTo** ステートメントを使用して、プロシージャ内のエラー処理ルーチンの場所を指定します。ゼロによる除算が行われるとエラー番号 6 が生成されます。このエラーがエラー処理ルーチンで処理された後、エラーが発生したステートメントに制御が戻ります。**On Error GoTo 0** ステートメントはエラー トラップを無効にします。その後で、**On Error Resume Next** ステートメントを使用してエラー トラップを遅延させ、次のステートメントによって生成されたエラーのコンテキストが確認できるようにします。**Err.Clear** は、エラーを処理した後に **Err** オブジェクトのプロパティをクリアするために使用されます。

VB

```
Public Sub OnErrorDemo()
   On Error GoTo ErrorHandler   ' Enable error-handling routine.
   Dim x As Integer = 32
   Dim y As Integer = 0
   Dim z As Integer
   z = x / y   ' Creates a divide by zero error
   On Error GoTo 0   ' Turn off error trapping.
   On Error Resume Next   ' Defer error trapping.
   z = x / y   ' Creates a divide by zero error again
   If Err.Number = 6 Then
      ' Tell user what happened. Then clear the Err object.
      Dim Msg As String
      Msg = "There was an error attempting to divide by zero!"
      MsgBox(Msg, , "Divide by zero error")
      Err.Clear() ' Clear Err object fields.
   End If
Exit Sub      ' Exit to avoid handler.
ErrorHandler:  ' Error-handling routine.
   Select Case Err.Number   ' Evaluate error number.
      Case 6   ' Divide by zero error
         MsgBox("You attempted to divide by zero!")
         ' Insert code to handle this error
      Case Else
         ' Insert code to handle other situations here...
   End Select
   Resume Next  ' Resume execution at same line
                ' that caused the error.
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照

処理手順

方法 : Visual Basic ランタイム エラーに関する情報を取得する

関連項目

End ステートメント

Err オブジェクト (Visual Basic)

Exit ステートメント (Visual Basic)

LastDllError プロパティ (Err オブジェクト)

Resume ステートメント

Operator ステートメント

クラスまたは構造体に演算子プロシージャを定義する演算子記号、オペランド、およびコードを宣言します。

```
[ <attrlist> ] Public [ Overloads ] Shared [ Shadows ] [ Widening | Narrowing ]
Operator operatorsymbol ( operand1 [, operand2 ]) [ As [ <attrlist> ] type ]
    [ statements ]
    [ statements ]
    Return returnvalue
    [ statements ]
End Operator
```

指定項目

attrlist

省略可能です。「属性リスト」を参照してください。

Public

必ず指定します。この演算子プロシージャのアクセス レベルが Public (Visual Basic) であることを示します。

Overloads

省略可能です。Overloads を参照してください。

Shared

必ず指定します。この演算子プロシージャが Shared (Visual Basic) プロシージャであることを示します。

Shadows

省略可能です。Shadows を参照してください。

Widening

Narrowing を指定した場合を除き、変換演算子には必ず指定します。この演算子プロシージャが Widening 変換を定義することを示します。このヘルプ ページの「拡大変換と縮小変換」を参照してください。

Narrowing

Widening を指定した場合を除き、変換演算子には必ず指定します。この演算子プロシージャが Narrowing 変換を定義することを示します。このヘルプ ページの「拡大変換と縮小変換」を参照してください。

operatorsymbol

必ず指定します。この演算子プロシージャが定義する演算子の記号または識別子を指定します。

operand1

必ず指定します。単項演算子 (変換演算子を含む) の単一オペランドまたは二項演算子の左オペランドの名前および型を指定します。

operand2

二項演算子では必ず指定します。二項演算子の右オペランドの名前と型を指定します。

operand1 および *operand2* の構文と指定項目は、以下のとおりです。

```
[ ByVal ] operandname [ As operandtype ]
```

指定項目	説明
ByVal	省略可能ですが、指定項目は常に ByVal で渡します。
operandname	必ず指定します。このオペランドを表す変数の名前を指定します。宣言された要素の名前 を参照してください。
operandtype	**Option Strict** が **On** である場合を除き、省略可能です。このオペランドのデータ型を指定します。

type

Option Strict が **On** である場合を除き、省略可能です。この演算子プロシージャによって返される値のデータ型を指定します。

statements

省略可能です。演算子プロシージャが実行するステートメントのブロックを指定します。

returnvalue

必ず指定します。演算子プロシージャから呼び出し元のコードに返す値を指定します。

End Operator

必ず指定します。この演算子プロシージャの定義を終了します。

解説

Operator は、クラスまたは構造体の中でのみ使用できます。つまり、演算子の宣言コンテキストをソース ファイル、名前空間、モジュール、インターフェイス、プロシージャ、またはブロックにすることはできません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

すべての演算子は **Public Shared** にする必要があります。どちらのオペランドにも **ByRef**、**Optional**、または **ParamArray** を指定することはできません。

演算子の記号または識別子を使って戻り値を返すことはできません。**Return** ステートメントを使用し、値を指定する必要があります。**Return** ステートメントは、プロシージャ内の任意の位置で何回でも指定できます。

Overloads キーワードを使うかどうかに関係なく、この方法で演算子を定義することは演算子オーバーロードと呼ばれます。次の表は、定義できる演算子の一覧です。

種類	演算子
単項演算	**+**, **-**, **IsFalse**, **IsTrue**, **Not**
二項演算	**+**, **-**, *****, **/**, ****, **&**, **^**, **>>**, **<<**, **=**, **<>**, **>**, **>=**, **<**, **<=**, **And**, **Like**, **Mod**, **Or**, **Xor**
変換 (単項)	**CType**

二項演算の一覧に示した = 演算子は、代入演算子ではなく比較演算子です。

CType を定義するときは、**Widening** または **Narrowing** を指定する必要があります。

一致したペア

特定の演算子を一致したペアとして定義する必要があります。ペアの一方の演算子を定義する場合には、もう一方の演算子も定義する必要があります。一致するペアを以下に示します。

- **=** および **<>**
- **>** および **<**
- **>=** および **<=**
- **IsTrue** および **IsFalse**

データ型の制限

定義する演算子には、定義先のクラスまたは構造体が伴っている必要があります。つまり、そのクラスまたは構造体が以下の要素のデータ型として指定される必要があります。

- 単項演算子のオペランド。
- 二項演算子の少なくとも 1 つのオペランド。
- 変換演算子のオペランドまたは戻り値の型。

一部の演算子には、次のようなデータ型の制限もあります。

- **IsTrue** 演算子および **IsFalse** 演算子を定義する場合は、両方が **Boolean** 型を返す必要があります。
- **<<** 演算子および **>>** 演算子を定義する場合は、両方が *operand2* の *operandtype* に**Integer** 型を指定する必要があります。

戻り値の型は、どちらかのオペランドの型と同じである必要はありません。たとえば、**=** または **<>** などの比較演算子は、オペランドが両方とも

Boolean ではない場合でも、**Boolean** を返すことができます。

論理演算子およびビット処理演算子

And、**Or**、**Not**、および **Xor** の各演算子は、Visual Basic では論理演算またはビット単位の演算を実行できます。ただし、このような演算子の 1 つをクラスまたは構造体に定義する場合は、ビット単位の演算だけを定義できます。

AndAlso 演算子は、**Operator** ステートメントには直接定義できません。ただし、以下の条件が満たされる場合は、**AndAlso** を使用できます。

- **AndAlso** に使用するオペランド型と同じ型で **And** を定義した。
- **And** からの戻り値として、定義先のクラスまたは構造体と同じ型を指定した。
- **And** を定義したクラスまたは構造体に **IsFalse** 演算子を定義した。

同様に、**Or** を同じオペランドで定義し、クラスまたは構造体の戻り値の型を使用し、クラスまたは構造体に **IsTrue** を定義した場合に、**OrElse** を使用できます。

拡大変換と縮小変換

拡大変換は実行時に常に成功しますが、縮小変換は実行時に失敗することがあります。詳細については、「拡大変換と縮小変換」を参照してください。

変換プロシージャを **Widening** で宣言する場合、プロシージャ コードではエラーを生成しないでください。これは、次のことを意味します。

- *type* 型の有効な値を常に返す必要がある。
- 生成される可能性のあるすべての例外およびその他のエラー条件を処理する必要がある。
- 呼び出したプロシージャから返されたエラーはコードで処理する必要がある。

変換プロシージャでエラーが起こる可能性がある場合、つまり変換プロシージャで未処理の例外が起こる可能性がある場合は、そのプロシージャを **Narrowing** で宣言する必要があります。

使用例

次のコード例は、**Operator** ステートメントを使用して、**And**、**Or**、**IsFalse**、および **IsTrue** の各演算子の演算子プロシージャを持つ構造体の外枠を定義します。**And** および **Or** は、`abc` 型の 2 つのオペランドと `abc` 型の戻り値を持ちます。**IsFalse** および **IsTrue** は、`abc` 型の単一オペランドを持ち、**Boolean** 型を返します。このように定義すると、呼び出し元のコードは、**And**、**AndAlso**、**Or**、および **OrElse** を `abc` 型のオペランドと一緒に使用できます。

VB

```
Public Structure abc
    Dim d As Date
    Public Shared Operator And(ByVal x As abc, ByVal y As abc) As abc
        Dim r As New abc
        ' Insert code to calculate And of x and y.
        Return r
    End Operator
    Public Shared Operator Or(ByVal x As abc, ByVal y As abc) As abc
        Dim r As New abc
        ' Insert code to calculate Or of x and y.
        Return r
    End Operator
    Public Shared Operator IsFalse(ByVal z As abc) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsFalse of z.
        Return b
    End Operator
    Public Shared Operator IsTrue(ByVal z As abc) As Boolean
        Dim b As Boolean
        ' Insert code to calculate IsTrue of z.
        Return b
    End Operator
End Structure
```

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する
方法 : 演算子プロシージャを呼び出す
方法 : 演算子を定義するクラスを使用する

関連項目

IsFalse 演算子
IsTrue 演算子
Widening
Narrowing

概念

拡大変換と縮小変換
演算子プロシージャ

Option Compare ステートメント

文字列のデータを比較するときに使用する、既定の比較方法を宣言します。

```
Option Compare { Binary | Text }
```

指定項目

Binary

省略可能です。文字の内部バイナリ表現による並べ替え順序に基づいて文字列が比較されます。

Text

省略可能です。文字列比較は、大文字と小文字を区別しない、システムのロケールで決められたテキストの並べ替え順序に基づいて行われます。

解説

Option Compare ステートメントを使用する場合は、ファイル内で他のどのソース コード ステートメントよりも先に定義する必要があります。

Option Compare ステートメントは、文字列を比較する方法 (**Binary** または **Text**) を、クラス、モジュール、または構造体に指定します。**Option Compare** ステートメントが定義されない場合、既定のテキスト比較方法は **Binary** になります。

Microsoft Windows の並べ替え順序は、コード ページによって決まります。詳細については、「コード ページ」を参照してください。

次の例では、英語/ヨーロッパ語のコード ページ (ANSI 1252) の文字が、一般的なバイナリ ソート順を生成する **Option Compare Binary** を使ってソートされます。

```
A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø
```

同じコード ページの同じ文字を **Option Compare Text** を使ってソートすると、テキストのソート順は次のようになります。

```
(A=a) < (À = à) < (B=b) < (E=e) < (Ê = ê) < (Z=z) < (Ø = ø)
```

Option Compare は、Visual Studio の統合開発環境 (IDE) またはコマンド ラインからでも設定できます。

統合開発環境 (IDE) で Option Compare を設定するには

1. [ツール] メニューの [オプション] をクリックします。
2. [プロジェクトおよびソリューション] ノードを開きます。
3. [Visual Basic の既定値] を選択します。
4. [Option Compare] の設定を変更します。

コマンド ラインで Option Compare を設定するには

- **vbc** コマンドに /optioncompare コンパイラ オプションを指定します。

使用例

次の例は、**Option Compare** ステートメントを使用して、既定の文字列比較方法にバイナリ モードを設定します。

VB

```
' Set the string comparison method to Binary ("AAA" < "aaa").
Option Compare Binary
```

次の例は、**Option Compare** ステートメントを使用して、既定の文字列比較方法に大文字と小文字を区別しないテキスト モードのソート順を設定します。

VB

```
' Set the string comparison method to Text ("AAA" = "aaa").
Option Compare Text
```

参照

関連項目

比較演算子 (Visual Basic)
InStr 関数 (Visual Basic)
InStrRev 関数 (Visual Basic)
Replace 関数 (Visual Basic)
Split 関数 (Visual Basic)
StrComp 関数 (Visual Basic)
/optioncompare
Option Explicit ステートメント (Visual Basic)
Option Strict ステートメント
/optionexplicit
/optionstrict
[Visual Basic の既定値] ([オプション] ダイアログ ボックス - [プロジェクト])

Option Explicit ステートメント (Visual Basic)

ファイル内のすべての変数を明示的に宣言するよう強制します。

```
Option Explicit { On | Off }
```

指定項目

On

省略可能です。**Option Explicit** チェックを有効にします。**On** または **Off** を指定しない場合、既定は **On** になります。

Off

省略可能です。**Option Explicit** チェックを無効にします。

解説

Option Explicit ステートメントを使用する場合は、ファイル内の他のどのソース コード ステートメントよりも前に記述する必要があります。

ファイル内に **Option Explicit** を記述したときは、すべての変数を **Dim** ステートメントまたは **ReDim** ステートメントで明示的に宣言する必要があります。宣言されていない変数名を使用すると、コンパイル時にエラーが発生します。

Option Explicit ステートメントを使用すると、既存の変数名を誤って入力したり、変数のスコープが明確でないコード内で混乱が起きたりするのを避けることができます。**Option Explicit** ステートメントを使用しない場合、宣言されていない変数はすべて **Object** 型になります。

メモ：
コード内で **Option Explicit** を指定しなかった場合、コンパイラは既定で **Option Explicit On** を使用します。

Visual Studio 統合開発環境 (IDE) またはコマンド ラインで **Option Explicit** を設定することもできます。

統合開発環境 (IDE) で Option Explicit を設定するには

1. [ツール] メニューの [オプション] をクリックします。
2. [プロジェクトおよびソリューション] ノードを展開します。
3. [Visual Basic の既定値] をクリックします。
4. [Option Explicit] 設定を変更します。

コマンド ラインで Option Explicit を設定するには

- **vbc** コマンドに /optionexplicit コンパイラ オプションを含めます。

使用例

次の例では、**Option Explicit** ステートメントを使用して、すべての変数を明示的に宣言するよう強制します。宣言されていない変数を使用すると、コンパイル時にエラーが発生します。

VB

```
' Force explicit variable declaration.
Option Explicit On
```

VB

```
Dim thisVar As Integer
thisVar = 10
' The following assignment produces a COMPILER ERROR because
' the variable is not declared and Option Explicit is On.
thisInt = 10 ' causes ERROR
```

参照

関連項目

Dim ステートメント (Visual Basic)
ReDim ステートメント (Visual Basic)
Option Compare ステートメント
Option Strict ステートメント
/optioncompare
/optionexplicit
/optionstrict
[Visual Basic の既定値] ([オプション] ダイアログ ボックス - [プロジェクト])

Option Strict ステートメント

暗黙的なデータ型変換を拡大変換だけに制限します。

```
Option Strict { On | Off }
```

指定項目

On

省略可能です。**Option Strict** のチェックを有効にします。

Off

省略可能です。**Option Strict** のチェックを無効にします。**On** または **Off** を指定しない場合は、既定で **Off** になります。

解説

Option Strict ステートメントを使用する場合は、ファイル内で他のどのソース コード ステートメントよりも先に定義する必要があります。

Visual Basic では、さまざまなデータ型を他のデータ型に変換できます。変換元のデータ型より変換先のデータ型の方が精度が低い場合、または容量が小さい場合は、データが失われる可能性があります。このような縮小変換が失敗した場合は、ランタイム エラーが発生します。**Option Strict** を使用すると、このような縮小変換に対してコンパイル時に通知が出されるので、ランタイム エラーを回避できます。

Option Strict は暗黙の縮小変換を許可しないだけでなく、遅延バインディングのエラーも生成します。**Object** 型として宣言された変数に代入されるオブジェクトは、遅延バインディングされます。

Option Strict On によって 厳密な型指定が可能になり、データの損失を伴う意図しない型変換を回避でき、遅延バインディングが禁止されて、パフォーマンスが改善されるので、必ず使用することをお勧めします。

メモ：
コードに **Option Strict** を指定しなければ、コンパイラは既定で **Option Strict Off** になります。

Option Strict は、Visual Studio の統合開発環境 (IDE) またはコマンド ラインからでも設定できます。

統合開発環境 (IDE) で Option Strict を設定するには

1. [ツール] メニューの [オプション] をクリックします。
2. [プロジェクトおよびソリューション] ノードを開きます。
3. [Visual Basic の既定値] を選択します。
4. [Option Strict] の設定を変更します。

コマンド ラインで Option Strict を設定するには

- **vbc** コマンドに /optionstrict コンパイラ オプションを指定します。

使用例

次の例は、**Option Strict** ステートメントによって、遅延バインディングおよびデータが損失する変換を回避する方法を示します。

VB

```
Option Strict On
```

VB

```
Dim thisVar As Integer
Dim thisObj As Object = New widget
thisVar = 1000   ' Declared variable does not generate error.
' Attempting to convert Double to Integer generates a COMPILER ERROR.
thisVar = 1234567890.9876542 ' causes ERROR
```

```
  ' Late-bound call generates a COMPILER ERROR.
  Call thisObj.Method1() ' causes ERROR
```

参照

関連項目

Option Compare ステートメント
Option Explicit ステートメント (Visual Basic)
/optioncompare
/optionexplicit
/optionstrict
[Visual Basic の既定値] ([オプション] ダイアログ ボックス - [プロジェクト])

概念

拡大変換と縮小変換

Property ステートメント

プロパティの値を設定および取得するためのプロパティ プロシージャ、およびプロパティの名前を宣言します。

```
[ <attributelist> ] [ Default ] [ accessmodifier ]
[ propertymodifiers ] [ Shared ] [ Shadows ] [ ReadOnly | WriteOnly ]
Property name ( [ parameterlist ] ) [ As returntype ] [ Implements implementslist ]
    [ <attributelist> ] [ accessmodifier ] Get
        [ statements ]
    End Get
    [ <attributelist> ] [ accessmodifier ] Set ( ByVal value As returntype [, parameterlist
 ] )
        [ statements ]
    End Set
End Property
```

指定項目

attributelist

省略可能です。このプロパティ、または **Get** プロシージャや **Set** プロシージャに適用される属性の一覧を指定します。「属性リスト」を参照してください。

Default

省略可能です。このプロパティが、クラスまたは構造体の既定のプロパティであることを指定します。既定のプロパティはパラメータをとる必要があり、プロパティ名を指定しなくても取得や設定を行うことができます。プロパティを **Default** として宣言すると、このプロパティや 2 つのプロパティ プロシージャのいずれかに **Private** を指定できなくなります。

accessmodifier

省略可能です。**Property** ステートメント、および **Get** ステートメントと **Set** ステートメントのうちの最大 1 つに指定します。次のいずれかを指定できます。

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Visual Basic でのアクセス レベル を参照してください。

propertymodifiers

省略可能です。次のいずれかを指定できます。

- Overloads
- Overrides
- Overridable
- NotOverridable
- MustOverride
- **MustOverride Overrides**
- **NotOverridable Overrides**

Shared

省略可能です。Shared (Visual Basic) を参照してください。

Shadows

省略可能です。Shadows を参照してください。

ReadOnly

省略可能です。ReadOnly (Visual Basic) を参照してください。

WriteOnly

省略可能です。WriteOnly を参照してください。

name

必ず指定します。プロパティ名。宣言された要素の名前 を参照してください。

parameterlist

省略可能です。このプロパティのパラメータ、および **Set** プロシージャのパラメータ (あれば) を表すローカル変数名の一覧です。パラメータの一覧 を参照してください。

returntype

Option Strict が **On** の場合は、必ず指定します。このプロパティによって返される値のデータ型を指定します。

Implements

省略可能です。このプロパティが実装する 1 つ以上の各プロパティが、このプロパティの包含クラスまたは包含構造体によって実装されるインターフェイスに定義されていることを示します。Implements ステートメント を参照してください。

implementslist

Implements が指定されている場合は、必ず指定します。実装されるプロパティの一覧。

```
implementedproperty [ , implementedproperty ... ]
```

implementedproperty の構文と指定項目は次のとおりです。

```
interface.definedname
```

指定項目	説明
interface	必ず指定します。このプロパティの包含クラスまたは包含構造体によって実装されるインターフェイスの名前。
definedname	必ず指定します。*interface* の中で、プロパティを定義するために使われている名前を指定します。

Get

プロパティが **WriteOnly** でマーク付けされている場合を除いて必ず指定します。プロパティの値を返すための **Get** プロパティ プロシージャを開始します。

statements

省略可能です。**Get** プロシージャまたは **Set** プロシージャの内部で実行するステートメントのブロック。

End Get

Get プロパティ プロシージャを終了します。

Set

プロパティが **ReadOnly** でマーク付けされている場合を除いて必ず指定します。プロパティの値を格納するための **Set** プロパティ プロシージャを開始します。

End Set

Set プロパティ プロシージャを終了します。

End Property

このプロパティの定義を終了します。

解説

Property ステートメントは、プロパティを宣言するときに指定します。プロパティには **Get** プロシージャ (読み取り専用)、**Set** プロシージャ (書き込み専用)、またはその両方 (読み取り/書き込み) を定義できます。

Property は、モジュール レベルでのみ使用できます。つまり、プロパティの宣言コンテキストは、クラス、構造体、モジュール、またはインターフェイ

スであることが必要で、ソース ファイル、名前空間、プロシージャ、ブロックでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

プロパティの既定のアクセス レベルは、パブリック アクセスです。プロパティのアクセスレベルは、**Property** ステートメントにアクセス修飾子を指定することによって変更できます。また、必要に応じて、一方のプロパティ プロシージャを、より制限の高いアクセス レベルに変更することも可能です。

Visual Basic は、プロパティの割り当ての際に、パラメータを **Set** プロシージャに渡します。**Set** のパラメータを指定しない場合、統合開発環境 (IDE) は `value` と呼ばれる暗黙のパラメータを使用します。このパラメータには、プロパティに割り当てる値が格納されています。通常、この値はプライベートなローカル変数に格納しますが、**Get** プロシージャを呼び出せばいつでも値を取得できます。

規則

- **アクセス レベルの混在。** 読み書き可能なプロパティを定義する場合、必要であれば **Get** プロシージャと **Set** プロシージャのどちらかに限り、プロパティとは異なるアクセス レベルを指定できます。これを指定する場合は、プロシージャにプロパティよりも制限の高いアクセス レベルを指定する必要があります。たとえば、プロパティを **Friend** で宣言する場合、**Set** プロシージャを **Private** で宣言できますが、**Public** では宣言できません。

  ReadOnly プロパティまたは **WriteOnly** プロパティを定義する場合は、1 つのプロパティ プロシージャ (前者なら **Get**、後者なら **Set**) がプロパティ全体を表すことになります。このようなプロシージャに異なるアクセス レベルを宣言すると、プロパティにアクセスレベルを 2 つ設定することになるため、宣言できません。

- **戻り値の型。** **Property** ステートメントには、プロパティが返す値のデータ型を宣言できます。任意のデータ型の他に、列挙値、構造体、クラス、またはインターフェイスの名前を指定できます。

  returntype を指定しない場合、プロパティは **Object** を返します。

- **実装。** プロパティに **Implements** キーワードが指定されている場合は、このプロパティの包含クラスの **Class** ステートメント、または包含構造体の **Structure** ステートメントのすぐ後に、**Implements** ステートメントが指定されている必要があります。**Implements** ステートメントには、*implementslist* に指定された各インターフェイスが定義されていることが必要です。ただし、インターフェイスが **Property** の定義に使用した名前 (*definedname* に指定) は、このプロパティの名前 (*name* に指定) と同じでなくてもかまいません。

動作

- **プロパティ プロシージャからの戻り** **Get** プロシージャまたは **Set** プロシージャから呼び出しコードに制御が戻るとき、プロシージャを呼び出したステートメントの次のステートメントから実行が継続します。

  Exit Property ステートメントと **Return** ステートメントは、プロパティ プロシージャを直ちに終了します。プロシージャの任意の場所に、**Exit Property** ステートメントと **Return** ステートメントを何度でも定義できます。また、**Exit Property** ステートメントと **Return** ステートメントの混在も可能です。

- **戻り値。** **Get** プロシージャから値を返すには、プロパティ名に値を割り当てるか、または **Return** ステートメントに値を設定します。戻り値を `quoteForTheDay` という名前のプロパティに割り当てた後、**Exit Property** ステートメントを使って制御を返すコード例を次に示します。

  VB

```
Private quoteValue As String = "No quote assigned yet."
```

  VB

```
ReadOnly Property quoteForTheDay() As String
    Get
        quoteForTheDay = quoteValue
        Exit Property
    End Get
End Property
```

  name に値を割り当てないで **Exit Property** を使用すると、**Get** プロシージャは、そのプロパティのデータ型の既定値を返します。

  Return ステートメントは **Get** プロシージャの戻り値を代入すると同時に、プロシージャを終了します。次に例を示します。

  VB

```
Private quoteValue As String = "No quote assigned yet."
```

VB

```
ReadOnly Property quoteForTheDay() As String
    Get
        Return quoteValue
    End Get
End Property
```

使用例

クラス内にプロパティを宣言するコード例を次に示します。

VB

```
Class Class1
    ' Define a local variable to store the property value.
    Private propertyValue As String
    ' Define the property.
    Public Property prop1() As String
        Get
            ' The Get property procedure is called when the value
            ' of a property is retrieved.
            Return propertyValue
        End Get
        Set(ByVal value As String)
            ' The Set property procedure is called when the value
            ' of a property is modified.  The value to be assigned
            ' is passed in the argument to Set.
            propertyValue = value
        End Set
    End Property
End Class
```

参照

処理手順

方法 : フィールドおよびプロパティをクラスに追加する

関連項目

Get ステートメント

Set ステートメント (Visual Basic)

パラメータの一覧

概念

既定のプロパティ

ステートメント Q ～ Z

次の表は、Visual Basic 言語のステートメントの一覧です。

RaiseEvent	ReDim	REM	RemoveHandler
Resume	Return	Select...Case	Set
Stop	Structure	Sub	SyncLock
Throw	Try...Catch...Finally	Using	While...End While
With...End With			

参照

関連項目

ステートメント A ～ E

ステートメント F ～ P

その他の技術情報

Visual Basic リファレンス

RaiseEvent ステートメント

モジュール レベルで宣言されたイベントをクラス、フォーム、またはドキュメントで発生させます。

```
RaiseEvent eventname[( argumentlist )]
```

指定項目

eventname

必ず指定します。発生させるイベントの名前を指定します。

argumentlist

省略可能です。変数、配列、または式をコンマで区切った一覧です。*argumentlist* 引数は、かっこで囲む必要があります。引数がない場合、かっこは省略できます。

解説

eventname は必ず指定します。モジュール内で宣言されたイベント名です。Visual Basic 変数の名前付け規則に従います。

イベントが発生したモジュールと宣言された場所が異なる場合は、エラーが発生します。次のコードは、イベントの宣言と、そのイベントが発生するプロシージャの例を示しています。

VB

```
' Declare an event at module level.
Event LogonCompleted(ByVal UserName As String)

Sub Logon(ByVal UserName As String)
    ' Raise the event.
    RaiseEvent LogonCompleted(UserName)
End Sub
```

モジュール内で明示的に宣言されていないイベントを **RaiseEvent** ステートメントで発生させることはできません。たとえば、すべてのフォームは System.Windows.Forms.Form から Click イベントを継承していますが、このイベントを派生フォーム内で **RaiseEvent** を使用して発生させることはできません。フォーム モジュールで `Click` イベントを宣言する場合、このイベントは、フォーム独自の **Click** イベントをシャドウします。OnClick メソッドを呼び出して、フォームの **Click** イベントを呼び出すことができます。

既定では、Visual Basic 内に定義されているイベントは、それぞれのイベント ハンドラを接続が確立された順序で発生させます。イベントには **ByRef** パラメータを設定できるので、後から接続したプロセスは、それ以前のイベント ハンドラで変更されたパラメータを受け取ることがあります。イベント ハンドラの実行が終わると、そのイベントを発生させたサブルーチンに制御が戻ります。

メモ：

非共有イベントを宣言したクラスのコンストラクタ内では、その非共有イベントを発生させないでください。そのようなイベントがランタイム エラーを引き起こさない場合でも、それらのイベントが対応するイベント ハンドラにキャッチされない場合があります。コンストラクタからイベントを発生させる必要がある場合は、**Shared** 修飾子を使用して共有イベントを作成してください。

カスタム イベントを定義すると、イベントの既定の動作を変更できます。カスタム イベントの場合は、**RaiseEvent** ステートメントはイベントの **RaiseEvent** アクセサを呼び出します。カスタム イベントの詳細については、「Event ステートメント」を参照してください。

使用例

次の例では、イベントを使用して 10 秒から 0 秒までをカウントダウンします。このコードは、イベント関連のいくつかのメソッド、プロパティ、ステートメントの例を示しています。**RaiseEvent** ステートメントの使用例も含まれています。

イベントを発生させるクラスをイベント ソース、イベントを処理するメソッドをイベント ハンドラと呼びます。イベント ソースには、そこで生成される複数のイベント ハンドラを設定できます。クラスでイベントが発生すると、そのイベントは、オブジェクトのインスタンスに対するイベントを処理するために選択されたすべてのクラスで発生します。

また、この例では、ボタン (`Button1`) とテキスト ボックス (`TextBox1`) を含んだフォーム (`Form1`) を使用します。ボタンをクリックすると、1 つ目のテキスト ボックスに 10 秒から 0 秒までのカウントダウンが表示されます。カウントダウンが終わると (10 秒が経過すると)、1 つ目のテキスト ボックスに "Done" と表示されます。

Form1 のコードでは、フォームの初期状態と終了状態を指定します。イベント発生時に実行されるコードも含まれます。

この例を使用するには、新しい Windows アプリケーション プロジェクトを開き、メイン フォームに Form1 という名前を付け、このフォームに Button1 という名前のボタンと TextBox1 という名前のテキスト ボックスを追加します。続いてフォームを右クリックし、[コードの表示] をクリックして、コード エディタを開きます。

Form1 クラスの宣言セクションに、**WithEvents** 変数を追加します。

VB

```
Private WithEvents mText As TimerState
```

Form1 のコードに以下のコードを追加します。Form_Load や Button_Click など、重複して存在する可能性のあるプロシージャを置き換えます。

VB

```
Private Sub Form1_Load(ByVal sender As Object, _
                       ByVal e As System.EventArgs) _
                       Handles MyBase.Load
    Button1.Text = "Start"
    mText = New TimerState
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
                          Handles Button1.Click
    mText.StartCountdown(10.0, 0.1)
End Sub

Private Sub mText_ChangeText() Handles mText.Finished
    TextBox1.Text = "Done"
End Sub

Private Sub mText_UpdateTime(ByVal Countdown As Double) _
                             Handles mText.UpdateTime
    TextBox1.Text = Format(Countdown, "##0.0")
    ' Use DoEvents to allow the display to refresh.
    My.Application.DoEvents()
End Sub

Class TimerState
    Public Event UpdateTime(ByVal Countdown As Double)
    Public Event Finished()
    Public Sub StartCountdown(ByVal Duration As Double, _
                              ByVal Increment As Double)
        Dim Start As Double = DateAndTime.Timer
        Dim ElapsedTime As Double = 0

        Dim SoFar As Double = 0
        Do While ElapsedTime < Duration
            If ElapsedTime > SoFar + Increment Then
                SoFar += Increment
                RaiseEvent UpdateTime(Duration - SoFar)
            End If
            ElapsedTime = DateAndTime.Timer - Start
        Loop
        RaiseEvent Finished()
    End Sub
End Class
```

F5 キーを押して上記のコード例を実行し、[Start] というラベルのボタンをクリックしてください。1 つ目のテキスト ボックスが秒のカウントダウンを開始します。カウントダウンが終わると (10 秒が経過すると)、1 つ目のテキスト ボックスに "Done" と表示されます。

メモ：

My.Application.DoEvents メソッドは、フォームとまったく同じ方法でイベントを処理するわけではありません。フォームがイベントを直接処理するようにするには、マルチスレッドを使用します。詳細については、「Visual Basic におけるマルチスレッド」を参照してください。

参照

処理手順

方法 : イベントをクラスに追加する

関連項目

Event ステートメント

AddHandler ステートメント

RemoveHandler ステートメント

Handles

その他の技術情報

Visual Basic におけるイベント

ReDim ステートメント (Visual Basic)

配列変数のストレージ領域を再割り当てします。

```
ReDim [ Preserve ] name(boundlist) [ , name(boundlist) [, ... ] ]
```

指定項目

Preserve

省略可能です。最後の次元の大きさを変更するときに、配列に既に格納されている値をそのまま保持しておく必要がある場合に指定する修飾子です。

name

必ず指定します。配列変数の名前を指定します。宣言された要素の名前 を参照してください。

boundlist

必ず指定します。再定義する配列の各次元の境界を示すリストを指定します。

解説

ReDim ステートメントを使用すると、既に宣言された配列の 1 つ以上の次元のサイズを変更できます。使用している大きな配列の一部の要素が不要になった場合、**ReDim** を使って配列のサイズを減らし、メモリを解放できます。逆に、配列の要素を増やす必要がある場合は、**ReDim** を使って要素を追加できます。

ReDim は、配列だけに使用するためのステートメントです。スカラ (1 つの値のみを格納する変数) 、コレクション、または構造体に使用することはできません。変数を **Array** 型で宣言する場合、新しい配列を作成するための十分な型情報は **ReDim** ステートメントにありません。

ReDim キーワードを使用できるのはプロシージャ レベルだけです。つまり、変数の宣言コンテキストはプロシージャであることが必要で、ソース ファイル、名前空間、インターフェイス、クラス、構造体、モジュール、またはプロシージャでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

規則

- **修飾子Preserve** 修飾子だけを指定できます。この修飾子を指定する場合は、**ReDim** キーワードを省略できません。
- **複数の変数。**同じ宣言ステートメントで複数の配列変数のサイズを変更できます。配列変数ごとに *name* および *boundlist* の各項目を指定します。複数の変数を指定するときは、コンマ (,) で区切ります。
- **配列の境界。***boundlist* の各エントリには、次元の下限と上限を指定できます。下限は、指定してもしなくても、常に 0 です。上限は、次元のサイズではなく添え字に指定できる最大の値です (次元のサイズは上限に 1 を加えた値です)。添え字には、0 から上限値までの値を使用できます。

  boundlist 内の次元数は、元の配列のランクと一致している必要があります。
- **空の配列。**-1 を使って配列の次元の上限を宣言できます。これは、配列は空であるが、Nothing (Visual Basic) ではないことを意味します。詳細については、「方法 : 要素を持たない配列を作成する」を参照してください。ただし、Visual Basic のコードではこのような配列に正しくアクセスできません。アクセスしようとすると、実行時に IndexOutOfRangeException エラーが発生します。
- **データ型ReDim** ステートメントは、配列変数または配列要素のデータ型を変更できません。
- **初期化ReDim** ステートメントには、配列要素を初期化する新しい値を指定できます。
- **ランク。ReDim** ステートメントで配列のランク (次元数) を変更することはできません。
- **Preserve を使ったサイズ変更。Preserve** キーワードを使用すると、配列の最後の次元だけをサイズ変更できます。他の各次元については、既存の配列と同じ境界を指定する必要があります。

  たとえば、次元が 1 つしかない配列の場合、その次元はただ 1 つの次元なので、その次元のサイズを変更しても配列の内容を保持できます。しかし、次元が 2 つ以上ある配列の場合に **Preserve** キーワードを指定すると、最後の次元のサイズだけを変更できます。
- **プロパティ。ReDim** ステートメントは、値の配列を保持するプロパティに対して使用できます。

動作

- **配列の置換。ReDim** ステートメントは既存の配列を解放し、同じランクを持つ配列を新規作成します。配列変数の中で、新しい配

列は、解放された配列に置き換わります。

- **Preserve を使わない初期化。Preserve** を指定しない場合、**ReDim** は、新しい配列の要素をそのデータ型の既定値で初期化します。
- **Preserve を使った初期化。Preserve** 修飾子を指定した場合、Visual Basic では、既存の配列から新しい配列に要素がコピーされます。

使用例

まず配列内の既存のデータを保持しながら動的配列の最後の次元のサイズを増加し、次に一部のデータを喪失しながらサイズを小さくするコード例は、次のとおりです。最終的に、このコードではサイズを元の値にまで縮小し、すべての配列要素を再初期化します。

VB

```
Dim intArray(10, 10, 10) As Integer
ReDim Preserve intArray(10, 10, 20)
ReDim Preserve intArray(10, 10, 15)
ReDim intArray(10, 10, 10)
```

最初の **ReDim** は、新しい配列を作成し、変数 `intArray` 内の既存の配列に置き換えます。**ReDim** は、既存の配列からすべての要素を新しい配列にコピーします。また、各次元においてすべての行の最後に 10 列を追加し、それらの列の要素を 0 (この配列の要素の型である **Integer** の既定値) に初期化します。

2 番目の **ReDim** ステートメントでは、別の配列を新規作成し、そこに収まるすべての要素をコピーします。ただし、各次元の各行の最後から 5 列分が失われます。これらの列が未使用の場合は問題ありません。大規模な配列のサイズを縮小することで、不要になったメモリを解放できます。

3 番目の **ReDim** は、別の新しい配列を作成し、各次元においてすべての行の最後からさらに 5 列分を削除します。今回は、既存の要素をコピーしません。これで、配列は元のサイズに戻り、すべての要素は元の既定値に戻ります。

参照

処理手順

方法 : 要素を持たない配列を作成する

関連項目

Const ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)
Erase ステートメント (Visual Basic)
Nothing (Visual Basic)
IndexOutOfRangeException

REM ステートメント (Visual Basic)

プログラムのソース コード内にコメントを記述するときに指定します。

```
REM comment
' comment
```

指定項目
comment

(省略可能。) 記述するコメント本文を指定します。キーワード **REM** または一重引用符 (') と *comment* との間には空白が必要です。

解説

REM ステートメントを 1 行に単独で記述する方法と、別のステートメントの後に記述する方法があります。**REM** ステートメントを記述した行では、**REM** ステートメントの後に別のステートメントを記述しないでください。別のステートメントの後に **REM** ステートメントを記述する場合は、前のステートメントとの間にスペースを挿入する必要があります。

REM の代わりに一重引用符 (') を使用できます。これは、コメントを別のステートメントの後に記述する場合にも、1 行に単独で記述する場合にも当てはまります。

メモ :

行連結シーケンス (_) を使用して **REM** ステートメントを続けて記述しないでください。コンパイラは、コメントの開始点以降、文字に特別な意味があるかどうかをチェックしません。複数行にコメントを記述する場合は、各行に **REM** ステートメントまたはコメント記号 (') を記述してください。

使用例

REM ステートメントを使ってプログラムにコメントを記述するコード例は、次のとおりです。次の例では、**REM** の代わりに一重引用符 (') を使用する方法も示します。

VB

```
Dim demoStr1, demoStr2 As String
demoStr1 = "Hello" REM Comment after a statement using REM.
demoStr2 = "Goodbye" ' Comment after a statement using the ' character.
REM This entire line is a comment.
' This entire line is also a comment.
```

参照
処理手順
方法 : コード内でステートメントを分割および連結する
概念
コード内のコメント

RemoveHandler ステートメント

イベントとイベント ハンドラの関連付けを解除します。

```
RemoveHandler event, AddressOf eventhandler
```

指定項目

event

処理するイベントの名前。

eventhandler

現在イベントを処理しているプロシージャの名前。

解説

AddHandler ステートメントと **RemoveHandler** ステートメントを使うと、プログラムの実行中にいつでも特定のイベントの処理を開始および停止できます。

カスタム イベントの場合は、**RemoveHandler** ステートメントがイベントの **RemoveHandler** アクセサを呼び出します。カスタム イベントの詳細については、「Event ステートメント」を参照してください。

使用例

VB

```
Sub TestEvents()
    Dim Obj As New Class1
    ' Associate an event handler with an event.
    AddHandler Obj.Ev_Event, AddressOf EventHandler
    ' Call the method to raise the event.
    Obj.CauseSomeEvent()
    ' Stop handling events.
    RemoveHandler Obj.Ev_Event, AddressOf EventHandler
    ' This event will not be handled.
    Obj.CauseSomeEvent()
End Sub

Sub EventHandler()
    ' Handle the event.
    MsgBox("EventHandler caught event.")
End Sub

Public Class Class1
    ' Declare an event.
    Public Event Ev_Event()
    Sub CauseSomeEvent()
        ' Raise an event.
        RaiseEvent Ev_Event()
    End Sub
End Class
```

参照

関連項目

AddHandler ステートメント

Handles

Event ステートメント

概念

イベントとイベント ハンドラ

AddHandler と RemoveHandler

Resume ステートメント

エラー処理ルーチンが終了した後で実行を再開します。

```
Resume [ Next | line ]
```

指定項目

Resume

必ず指定します。エラー ハンドラと同じプロシージャでエラーが発生した場合は、エラーの原因となったステートメントから実行が再開されます。呼び出されたプロシージャでエラーが発生した場合は、エラー処理ルーチンを含むプロシージャから最後に呼び出されたステートメントから実行が再開されます。

Next

省略可能です。エラー ハンドラと同じプロシージャでエラーが発生した場合は、エラーの原因となったステートメントの直後のステートメントから実行が再開されます。呼び出されたプロシージャでエラーが発生した場合は、エラー処理ルーチン (または **On Error Resume Next** ステートメント) を含むプロシージャから最後に呼び出されたステートメントの直後のステートメントから実行が再開されます。

line

省略可能です。必須の *line* 引数で指定した行から実行が再開されます。引数 *line* は行ラベルまたは行番号で、エラー ハンドラと同じプロシージャに含まれる必要があります。

解説

エラー処理ルーチン以外で **Resume** ステートメントを使用すると、エラーが発生します。

Try...Catch...Finally ステートメントを含むプロシージャ内では、**Resume** ステートメントは使用できません。

使用例

この例では、**Resume** ステートメントを使用して、プロシージャ内のエラー処理を終了させ、エラーが発生したステートメントの実行を再開します。**Resume** ステートメントの用途を示すために、エラー番号 55 を生成します。

VB

```
Sub ResumeStatementDemo()
  On Error GoTo ErrorHandler   ' Enable error-handling routine.
  Dim x As Integer = 32
  Dim y As Integer = 0
  Dim z As Integer
  z = x / y   ' Creates a divide by zero error
  Exit Sub   ' Exit Sub to avoid error handler.
ErrorHandler:     ' Error-handling routine.
  Select Case Err.Number   ' Evaluate error number.
      Case 6   ' "Divide by zero" error.
        y = 1 ' Sets the value of y to 1 and tries the calculation again.
      Case Else
        ' Handle other situations here....
  End Select
  Resume   ' Resume execution at same line
  ' that caused the error.
End Sub
```

必要条件

名前空間 : Microsoft.VisualBasic

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)
Error ステートメント

On Error ステートメント (Visual Basic)

Return ステートメント (Visual Basic)

Function、**Sub**、**Get**、**Set**、または **Operator** の各プロシージャを呼び出したコードに制御を戻します。

```
Return
-or-
Return expression
```

指定項目
expression

Function、**Get**、または **Operator** の各プロシージャでは、必ず指定します。呼び出し元のコードに返す値を表す式を指定します。

解説

Sub プロシージャまたは **Set** プロシージャでは、**Exit Sub** ステートメントまたは **Exit Property** ステートメントが **Return** ステートメントに相当します。*expression* を指定することはできません。

Function、**Get**、または **Operator** の各プロシージャでは、**Return** ステートメント内に *expression* を指定する必要があり、*expression* はプロシージャの戻り値の型に変換できるデータ型に評価される必要があります。**Function** プロシージャまたは **Get** プロシージャでは、式を割り当てる代わりにプロシージャ名を戻り値として使うこともできます。その場合、**Exit Function** ステートメントまたは **Exit Property** ステートメントを実行します。**Operator** プロシージャでは、**Return** *expression* を使用する必要があります。

適切な数の **Return** ステートメントを同じプロシージャ内に指定できます。

メモ：

Finally ブロックのコードは、**Try** または **Catch** ブロックの **Return** ステートメントが見つかり、その **Return** ステートメントが実行される前に実行されます。この場合、**Finally** ブロックに **Return** ステートメントがあると、Try または Catch ブロックの **Return** ステートメントよりも先に実行され、戻り値が異なる場合があります。このような混乱を防ぐため、**Return** ステートメントを **Finally** ブロックで使用することは避けてください。

使用例

次のコード例では、**Return** ステートメントを何回も使って、プロシージャで何も行う必要がなくなった時点で呼び出し元のコードに制御を戻しています。

VB

```
Public Function getAgePhrase(ByVal age As Integer) As String
    If age > 60 Then Return "Senior"
    If age > 40 Then Return "Middle-aged"
    If age > 20 Then Return "Adult"
    If age > 12 Then Return "Teen-aged"
    If age > 4 Then Return "School-aged"
    If age > 1 Then Return "Toddler"
    Return "Infant"
End Function
```

参照
関連項目
Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)
Get ステートメント
Set ステートメント (Visual Basic)
Operator ステートメント
Property ステートメント
Exit ステートメント (Visual Basic)
Try...Catch...Finally ステートメント (Visual Basic)

Select...Case ステートメント (Visual Basic)

条件式の値に従って、複数のステートメント ブロックのいずれかを実行させるフロー制御ステートメントです。

```
Select [ Case ] testexpression
    [ Case expressionlist
        [ statements ] ]
    [ Case Else
        [ elsestatements ] ]
End Select
```

指定項目

testexpression

必ず指定します。式を指定します。式の結果は、**Boolean**、**Byte**、**Char**、**Date**、**Double**、**Decimal**、**Integer**、**Long**、**Object**、**SByte**、**Short**、**Single**、**String**、**UInteger**、**ULong**、**UShort** など、基本データ型のいずれかである必要があります。

expressionlist

Case ステートメント内に必ず指定します。*testexpression* と照合する値を表す式の句のリストを指定します。複数の式の句を指定する場合は、コンマ (,) で区切ります。それぞれの句は、次のいずれかの形式をとることができます。

- *expression1* **To** *expression2*
- [**Is**] *comparisonoperator expression*
- *expression*

testexpression に一致する値の範囲の境界を指定するには、**To** キーワードを使用します。*expression1* の値は、*expression2* の値以下である必要があります。

Is キーワードを比較演算子 (=、<>、<、<=、>、>=) と一緒に使用すると、*testexpression* と照合する値に制限を指定できます。キーワード **Is** は、指定しなくても *comparisonoperator* の前に自動的に挿入されます。

expression だけを指定する形式は、*comparisonoperator* を等号 (=) とする、特殊な **Is** 形式として処理されます。この形式は *testexpression* = *expression* として評価されます。

expressionlist 内の式は任意のデータ型をとることができます。ただし、そのデータ型は、引数 *testexpression* のデータ型に暗黙的に変換可能であり、適用される *comparisonoperator* がその 2 つの型で有効である必要があります。

statements

省略可能です。*testexpression* が *expressionlist* 内のいずれかの句に一致する場合に実行される 1 つ以上のステートメントを **Case** の後に指定します。

elsestatements

省略可能です。*testexpression* がどの **Case** ステートメントの *expressionlist* 内にある、どの句とも一致しない場合に実行される 1 つ以上のステートメントを **Case Else** の後に指定します。

End Select

Select...**Case** 構造の定義を終了します。

解説

testexpression がいずれかの **Case** *expressionlist* 句と一致する場合は、その **Case** ステートメントの後から、次の **Case**、**Case Else**、または **End Select** ステートメントまでのステートメントが実行されます。ブロックの実行が終わると、制御は **End Select** ステートメントの次のステートメントに移ります。*testexpression* が 2 つ以上の **Case** 句の *expressionlist* 句に一致する場合は、最初に一致した句の後のステートメントだけが実行されます。

Case Else は、他のどの **Case** ステートメントの *testexpression* 句と *expressionlist* 句の間にも、一致する句が見つからない場合に実行する *elsestatements* を定義する際に使用します。必須ではありませんが、予測できない *testexpression* 値を処理するために、**Select Case** 構造に **Case Else** ステートメントを記述することをお勧めします。*testexpression* に一致する **Case** *expressionlist* 句が 1 つもなく、**Case Else** ステートメントもない場合、制御は **End Select** の次のステートメントに渡されます。

Case 句には複数の式や範囲を指定できます。たとえば、次の行は有効なステートメントです。

```
Case 1 To 4, 7 To 9, 11, 13, Is > maxNumber
```

メモ：

Case ステートメントと **Case Else** ステートメントに使用する **Is** キーワードは、オブジェクト参照を比較するために使う Is 演算子 (Visual Basic) と同じではありません。

文字列の範囲や複数の文字列式を指定することもできます。次の例では、**Case** は "apples" とまったく等しい文字列、アルファベット順で "nuts" と "soup" の間の値を持つ文字列、または現在の `testItem` の値とまったく同じ値を格納する文字列と一致します。

```
Case "apples", "nuts" To "soup", testItem
```

文字列の比較は、**Option Compare** の設定の影響を受けることがあります。**Option Compare Text** が設定されている場合は、"Apples" と "apples" の比較は等しくなりますが、**Option Compare Binary** が設定されている場合は等しくなりません。

メモ：
複数の句が定義された **Case** ステートメントは、ショートサーキットと呼ばれる動作を起こすことがあります。Visual Basic は句を左から右の順に評価し、いずれか 1 つが *testexpression* と一致した場合、残りの句は評価されません。ショートサーキットはパフォーマンスを向上させますが、*expressionlist* 内のすべての式が評価されるという前提で定義すると、予期しない結果になる可能性があります。ショートサーキットの詳細については、「Boolean 式」を参照してください。

Case または **Case Else** のステートメント ブロック内のコードが、そのブロック内のステートメントを実行する必要がなくなった場合は、**Exit Select** ステートメントを使用してブロックを終了できます。制御は、直ちに **End Select** の次のステートメントに移ります。

Select Case は入れ子構造にできます。入れ子にされた各 **Select Case** には、対応する **End Select** ステートメントが必要です。また、外側の **Select Case** の単一の **Case** または **Case Else** のステートメント ブロックの内側に、完全に含まれている必要があります。

使用例

Select Case 構造を使用して、変数 `number` の値に対応する行を書き込む例を次に示します。2 番目の **Case** ステートメントに `number` の現在の値に一致する値が含まれるため、"Between 6 and 8, inclusive" を書き込むステートメントが実行されます。

VB

```
Dim number As Integer = 8
Select Case number
    Case 1 To 5
        Debug.WriteLine("Between 1 and 5, inclusive")
        ' The following is the only Case clause that evaluates to True.
    Case 6, 7, 8
        Debug.WriteLine("Between 6 and 8, inclusive")
    Case 9 To 10
        Debug.WriteLine("Equal to 9 or 10")
    Case Else
        Debug.WriteLine("Not between 1 and 10, inclusive")
End Select
```

参照

関連項目

Choose 関数
End ステートメント
If...Then...Else ステートメント (Visual Basic)
Option Compare ステートメント
Exit ステートメント (Visual Basic)

Set ステートメント (Visual Basic)

値をプロパティに代入するための **Set** プロパティ プロシージャを宣言します。

```
[ <attributelist> ] [ accessmodifier ] Set (ByVal value [ As datatype ])
    [ statements ]
End Set
```

指定項目

attributelist

省略可能です。「属性リスト」を参照してください。

accessmodifier

このプロパティでは、**Get** ステートメントと **Set** ステートメントのうちの 1 つについては、指定を省略できます。次のいずれかを指定できます。

- Protected
- Friend
- Private
- **Protected Friend**

Visual Basic でのアクセス レベル を参照してください。

value

必ず指定します。このパラメータには、プロパティの新しい値が格納されます。

datatype

Option Strict が **On** の場合は、必ず指定します。*value* パラメータのデータ型を指定します。指定するデータ型は、この **Set** ステートメントを宣言するプロパティのデータ型と同じにする必要があります。

statements

省略可能です。**Set** プロパティ プロシージャの呼び出し時に実行される 1 つ以上のステートメントを指定します。

End Set

必ず指定します。**Set** プロパティ プロシージャの定義を終了します。

解説

ReadOnly のマークが付けられたプロパティを除き、すべてのプロパティは **Set** プロパティ プロシージャを持つ必要があります。**Set** プロシージャは、プロパティの値を設定するために使用されます。

プロパティに保存する値が代入ステートメントで指定された場合、Visual Basic は自動的にプロパティの **Set** プロシージャを呼び出します。

Visual Basic は、プロパティに値を代入するときにパラメータを **Set** プロシージャに渡します。**Set** のパラメータを指定しないと、統合開発環境 (IDE: Integrated Development Environment) では `value` というパラメータが暗黙的に使用されます。このパラメータには、プロパティに代入する値が含まれています。通常、この値はプライベートなローカル変数に格納しますが、**Get** プロシージャを呼び出せばいつでも値を取得できます。

プロパティ宣言の本体には、プロパティの **Get** プロシージャと **Set** プロシージャのみを Property ステートメント ステートメントと **End Property** ステートメントの間に記述できます。それ以外のプロシージャを含めることはできません。特に、プロパティの現在の値を含めることはできません。現在の値をどちらかのプロパティ プロシージャの内部に含めると他のプロパティ プロシージャから値にアクセスできなくなるため、この値はプロパティの外部に格納する必要があります。通常は、プロパティと同じレベルで Private (Visual Basic) 変数を宣言し、この中に現在の値を格納します。**Set** プロシージャが適用されるプロパティには、このプロシージャを内部に定義する必要があります。

Set ステートメント内で *accessmodifier* を使ってアクセス レベルを設定しない限り、既定で **Set** プロシージャのアクセス レベルは、それが含まれるプロパティと同じになります。

規則

- **アクセス レベルの混在。** 読み書き可能なプロパティを定義する場合、必要であれば **Get** プロシージャと **Set** プロシージャのどちらかに限

り、プロパティとは異なるアクセス レベルを指定できます。これを指定する場合は、プロシージャにプロパティよりも制限の高いアクセス レベルを指定する必要があります。たとえば、プロパティを **Friend** で宣言した場合、**Set** プロシージャは **Private** で宣言できますが、**Public** では宣言できません。

WriteOnly プロパティを宣言している場合は、**Set** プロシージャはプロパティ全体を表します。別のアクセス レベルを **Set** に宣言するとプロパティに 2 つのアクセス レベルを設定することになるので、このような宣言はできません。

動作

- **プロパティ プロシージャからの制御の戻り。Set** プロシージャから呼び出し元のコードに戻ると、プロパティに格納する値を渡したステートメントの直後から実行が続行されます。

  Set プロパティ プロシージャは、Return ステートメント (Visual Basic) または Exit ステートメント (Visual Basic) を使って呼び出し元に戻ることができます。

  Exit Property ステートメントおよび **Return** ステートメントは、プロパティ プロシージャを直ちに終了します。**Exit Property** ステートメントと **Return** ステートメントは、プロシージャ内の任意の位置で何回でも指定でき、**Exit Property** ステートメントと **Return** ステートメントを同じプロパティ内で混在して使用できます。

使用例

Set ステートメントを使って、プロパティの値を設定するコード例を次に示します。

VB

```
Class propClass
    Private propVal As Integer
    Property prop1() As Integer
        Get
            Return propVal
        End Get
        Set(ByVal value As Integer)
            propVal = value
        End Set
    End Property
End Class
```

参照

処理手順

方法 : フィールドおよびプロパティをクラスに追加する

関連項目

Get ステートメント

Property ステートメント

Sub ステートメント (Visual Basic)

Stop ステートメント (Visual Basic)

実行を中断するステートメントです。

```
Stop
```

解説

Stop ステートメントは、プロシージャ内の任意の場所に置いて、実行を中断できます。**Stop** ステートメントは、プログラム コードのブレークポイントと同じ働きをします。

Stop ステートメントはプログラムの実行を中断しますが、**End** ステートメントと異なり、コンパイル済みの実行可能ファイル (.EXE) の内部でない限り、ファイルを閉じたり、変数をクリアすることはありません。

メモ :

統合開発環境 (IDE: Integrated Development Environment) 外で実行されるコード内に **Stop** ステートメントが含まれている場合は、そこまで進むとデバッガが呼び出されます。そのコードがデバッグ モードとリテール モードのどちらでコンパイルされたかは関係ありません。

使用例

Stop ステートメントを使って、**For...Next** ループを繰り返すごとに中断するコード例は、次のとおりです。

VB

```
Dim i As Integer
For i = 1 To 10
    Debug.WriteLine(i)
    ' Stop during each iteration and wait for user to resume.
    Stop
Next i
```

参照

関連項目

End ステートメント

Structure ステートメント

構造体の名前を宣言し、構造体を構成する変数、プロパティ、イベント、およびプロシージャの定義を提供します。

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ Partial ] _
Structure name [ ( Of typelist ) ]
    [ Implements interfacenames ]
        datamemberdeclarations
    [ methodmemberdeclarations ]
End Structure
```

指定項目

attributelist

省略可能です。「属性リスト」を参照してください。

accessmodifier

省略可能です。次のいずれかになります。

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Visual Basic でのアクセス レベル を参照してください。

Shadows

省略可能です。「Shadows」を参照してください。

Partial

省略可能です。構造体の部分定義を示します。Partial (Visual Basic) を参照してください。

name

必ず指定します。この構造体の名前です。宣言された要素の名前 を参照してください。

Of

省略可能です。これがジェネリックな構造体であることを指定します。

typelist

Of キーワードを使用する場合は必ず指定します。この構造体の型パラメータ リストを指定します。「型リスト」を参照してください。

Implements

省略可能です。この構造体が、複数のインターフェイスのメンバを実装していることを示します。Implements ステートメント を参照してください。

interfacenames

Implements ステートメントを使用する場合は必ず指定します。この構造体が実装するインターフェイスの名前を指定します。

datamemberdeclarations

必ず指定します。構造体のデータ メンバを宣言する、1 つ以上の **Const**、**Dim**、**Enum**、または **Event** ステートメントを指定します。

methodmemberdeclarations

省略可能です。構造体のメソッド メンバとして機能する、**Function**、**Operator**、**Property**、または **Sub** の各プロシージャの宣言を 0 またはそれ以上指定します。

End Structure

必ず指定します。**Structure** の定義を終了します。

解説

Structure ステートメントは、カスタマイズできる複合値型を定義します。構造体は、以前のバージョンの Visual Basic にあったユーザー定義型 (UDT: User-Defined Type) を一般化したものです。詳細については、「構造体 : 独自のデータ型」を参照してください。

構造体は、クラスと同じ機能の多くをサポートします。たとえば、構造体は、プロパティやプロシージャを持つことができ、インターフェイスを実装でき、パラメータ化されたコンストラクタを持つことができます。ただし、継承、宣言、および使用方法に関しては、構造体とクラスの間には大きな違いがあります。また、クラスは参照型ですが、構造体は値型です。詳細については、「構造体とクラス」を参照してください。

Structure は、名前空間またはモジュール レベルでのみ使用できます。つまり、構造体の宣言コンテキストは、ソース ファイル、名前空間、クラス、構造体、モジュール、またはインターフェイスのいずれかである必要があり、プロシージャまたはブロックでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

既定で、構造体のアクセス レベルは Friend (Visual Basic) です。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。詳細については、「Visual Basic でのアクセス レベル」を参照してください。

規則

- **入れ子。** 構造体の内部に別の構造体を定義できます。外側の構造体は包含構造体と呼ばれ、内側の構造体は入れ子構造体と呼ばれます。ただし、包含構造体をとおして入れ子構造体のメンバにアクセスすることはできません。入れ子構造体のメンバにアクセスするには、入れ子構造体のデータ型の変数を宣言する必要があります。
- **メンバの宣言。** 構造体のすべてのメンバを宣言する必要があります。構造体からは何も継承できないため、構造体のメンバを Protected または **Protected Friend** にすることはできません。ただし、構造体そのものを**Protected** または **Protected Friend** にすることはできます。

  最低でも 1 つの非共有変数または非共有の非カスタム イベントを構造体に宣言する必要があります。非共有ではあっても、定数、プロパティ、およびプロシージャだけを宣言することはできません。
- **初期化** 構造体の非共有データ メンバの値を宣言の一部として初期化することはできません。構造体のパラメータ化されたコンストラクタを使ってそのようなデータ メンバを初期化するか、または構造体のインスタンスを作成した後にメンバに値を割り当てる必要があります。
- **継承。** 構造体は、ValueType 以外の型を継承できません。すべての構造体が、この型を継承します。特に、構造体は別の構造体を継承できません。

  たとえ **ValueType** を指定するためであっても、Inherits ステートメント を構造体の定義で使用することはできません。
- **実装。** 構造体で Implements ステートメント を使用する場合、*interfacenames* に指定するすべてのインターフェイスに定義されたすべてのメンバを実装する必要があります。
- **既定のプロパティ。** 構造体には、Default (Visual Basic) 修飾子を使って、最低でも 1 つのプロパティを既定のプロパティとして指定できます。詳細については、「既定のプロパティ」を参照してください。

動作

- **アクセス レベル** 構造体の内部では、各メンバを独自のアクセス レベルで宣言できます。既定で、すべての構造体メンバのアクセス レベルは Public (Visual Basic) です。構造体そのものにこれより厳しいアクセス レベルを指定した場合は、たとえアクセス修飾子を使ってメンバのアクセス レベルを調整していても、メンバへのアクセスが自動的に制限されることに注意してください。
- **スコープ** 構造体は、そこに含まれる名前空間、クラス、構造体、またはモジュールをスコープとします。

  すべての構造体メンバのスコープは、構造体全体になります。
- **有効期間。** 構造体に有効期間はありません。ただし、構造体の各インスタンスには、他のインスタンスに依存しない独自の有効期間があります。

  インスタンスの有効期間は、New (Visual Basic) 句で作成された時点で開始されます。インスタンスに含まれる変数の有効期間が終わった時点で、そのインスタンスの有効期間は終わります。

  構造体インスタンスの有効期間を延長することはできません。静的構造体に相当する機能は、モジュールに用意されています。詳細については、「Module ステートメント」を参照してください。

  構造体メンバの有効期間は、それを宣言する方法と場所で決まります。詳細については、「Class ステートメント (Visual Basic)」の「有効期間」を参照してください。
- **修飾** 構造体の外部にあるコードでは、メンバの名前をその構造体の名前で修飾する必要があります。

入れ子構造体の内部のコードでプログラミング要素を修飾なしで参照した場合、Visual Basic はその要素をまず入れ子構造体の内部で探し、その次にコンテナ構造体の内部で探します。この手順が、最も外側のコンテナ要素にまで繰り返されます。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」を参照してください。

- **メモリの使用量。** 他のすべての複合データ型と同様に、構造体の総メモリ使用量を計算する場合、各メンバのストレージ割り当ての公称サイズを単に合計しただけでは安全ではありません。さらに、メモリ内に格納される順序が宣言の順序と同じであると仮定するのも安全ではありません。構造体のストレージ レイアウトを制御する必要がある場合は、StructLayoutAttribute 属性を **Structure** ステートメントに適用します。

使用例

次の例では、**Structure** ステートメントを使って、従業員に関連のある複数のデータを定義しています。データ項目の機密性に応じて **Public**、**Friend**、**Private** の各メンバを使用する方法が示されています。プロシージャ、プロパティ、およびイベント メンバも示されています。

VB

```
Public Structure employee
    ' Public members, accessible from throughout declaration region.
    Public firstName As String
    Public middleName As String
    Public lastName As String
    ' Friend members, accessible from anywhere within the same assembly.
    Friend employeeNumber As Integer
    Friend workPhone As Long
    ' Private members, accessible only from within the structure itself.
    Private homePhone As Long
    Private level As Integer
    Private salary As Double
    Private bonus As Double
    ' Procedure member, which can access structure's private members.
    Friend Sub calculateBonus(ByVal rate As Single)
        bonus = salary * CDbl(rate)
    End Sub
    ' Property member to return employee's eligibility.
    Friend ReadOnly Property eligible() As Boolean
        Get
            Return level >= 25
        End Get
    End Property
    ' Event member, raised when business phone number has changed.
    Public Event changedWorkPhone(ByVal newPhone As Long)
End Structure
```

参照

関連項目

Class ステートメント (Visual Basic)
Interface ステートメント (Visual Basic)
Module ステートメント
Dim ステートメント (Visual Basic)
Const ステートメント (Visual Basic)
Enum ステートメント (Visual Basic)
Event ステートメント
Operator ステートメント
Property ステートメント

概念

構造体とクラス

Sub ステートメント (Visual Basic)

Sub プロシージャを定義する名前、パラメータ、およびコードを宣言します。

```
[ <attributelist> ] [ accessmodifier ] [ proceduremodifiers ] [ Shared ] [ Shadows ]
Sub name [ (Of typeparamlist) ] [ (parameterlist) ] [ Implements implementslist | Handles e
ventlist ]
    [ statements ]
    [ Exit Sub ]
    [ statements ]
End Sub
```

指定項目

attributelist

省略可能です。「属性リスト」を参照してください。

accessmodifier

省略可能です。次のいずれかになります。

- Public
- Protected
- Friend
- Private
- **Protected Friend**

Visual Basic でのアクセス レベル を参照してください。

proceduremodifiers

省略可能です。次のいずれかになります。

- Overloads
- Overrides
- Overridable
- NotOverridable
- MustOverride
- **MustOverride Overrides**
- **NotOverridable Overrides**

Shared

省略可能です。「Shared (Visual Basic)」を参照してください。

Shadows

省略可能です。「Shadows」を参照してください。

name

必ず指定します。プロシージャの名前を指定します。宣言された要素の名前 を参照してください。

typeparamlist

省略可能です。ジェネリック プロシージャの型パラメータのリストを指定します。「型リスト」を参照してください。

parameterlist

省略可能です。このプロシージャのパラメータを表すローカル変数名のリストを指定します。パラメータの一覧 を参照してください。

Implements

省略可能です。このプロシージャが 1 つ以上の **Sub** プロシージャを実装しており、それぞれが、このプロシージャの包含クラスまたは構造体によって実装されているインターフェイスの中で定義されていることを示します。Implements ステートメント を参照してください。

implementslist

Implements が指定されている場合は、必ず指定します。実装される **Sub** プロシージャのリストを指定します。

```
implementedprocedure [ , implementedprocedure ... ]
```

implementedprocedure の構文と指定項目は次のとおりです。

```
interface.definedname
```

指定項目	説明
interface	必ず指定します。このプロシージャの包含クラスまたは構造体で実装されているインターフェイスの名前を指定します。
definedname	必ず指定します。*interface* の中でプロシージャを定義するために使われている名前を指定します。

Handles

省略可能です。このプロシージャが 1 つまたは複数の特定イベントを処理できることを示します。Handles を参照してください。

eventlist

Handles が指定されている場合は、必ず指定します。このプロシージャが処理するイベントのリストを指定します。

```
eventspecifier [ , eventspecifier ... ]
```

eventspecifier の構文と指定項目は次のとおりです。

```
eventvariable.event
```

指定項目	説明
eventvariable	必ず指定します。イベントを発生させるクラスまたは構造体のデータ型で宣言されたオブジェクト変数を指定します。
event	必ず指定します。このプロシージャが処理するイベントの名前を指定します。

statements

省略可能です。このプロシージャ内で実行するステートメントのブロックを指定します。

End Sub

このプロシージャの定義を終了します。

解説

すべての実行可能コードはプロシージャ内に記述する必要があります。呼び出し元のコードに値を返す必要がない場合は、**Sub** プロシージャを使用します。値を返す必要がある場合は、**Function** プロシージャを使用します。

Sub は、モジュール レベルでのみ使用できます。つまり、Sub プロシージャの宣言コンテキストは、クラス、構造体、モジュール、またはインターフェイスであることが必要で、ソース ファイル、名前空間、プロシージャ、またはブロックでは宣言できません。詳細については、「宣言コンテキストと既定のアクセス レベル」を参照してください。

Sub プロシージャは、既定ではパブリック アクセスになります。アクセス修飾子を使用してこれらのアクセス レベルを調整できます。

規則

- **実装** このプロシージャが **Implements** キーワードを使用する場合、プロシージャの包含クラスまたは構造体では、**Class** または **Structure** ステートメントの直後に **Implements** ステートメントが記述されている必要があります。この **Implements** ステートメントには、*implementslist* で指定した個々のインターフェイスが含まれている必要があります。ただし、インターフェイスが **Sub** プロシージャを定義するために使用する名前 (*definedname* で指定) と、このプロシージャの名前 (*name* で指定) を一致させる必要はありません。

動作

- **プロシージャから戻るときの動作** **Sub** プロシージャが呼び出し元コードに戻ると、そのプロシージャを呼び出したステートメントの次のステートメントから実行が継続します。

  Exit Sub ステートメントと **Return** ステートメントは、**Sub** プロシージャを直ちに終了します。**Exit Sub** ステートメントと **Return** ステート

メントは、プロシージャ内の任意の位置で何回でも指定でき、**Exit Sub** ステートメントと **Return** ステートメントを同じプロパティ内で混在して使用できます。

次の例は、**Sub** プロシージャから制御を戻す方法を示しています。

```
                Sub mySub(ByVal q As String)
    Return
End Sub
```

- **プロシージャの呼び出し。Sub** プロシージャは、**Function** プロシージャと同様に、パラメータを受け取って一連のステートメントを実行する独立したプロシージャです。しかし、**Sub** プロシージャは、値を返す **Function** プロシージャとは異なり、式の中に記述することはできません。

  Sub プロシージャをステートメントの中から呼び出すには、プロシージャ名の後ろに引数リストをかっこで囲んで指定します。指定する引数がない場合に限って、かっこを省略できます。ただし、かっこを指定した方がコードの読みやすさは向上します。

  Call ステートメントを使って **Sub** プロシージャを呼び出すこともできます。この方法を使うと、コードの読みやすさが向上します。

トラブルシューティング

実行の順序 Visual Basic では、演算効率を高めるために数式が自動的に並べ替えられることがあります。そのため、他のプロシージャを呼び出す式が引数リストに複数含まれている場合は、それらが特定の順序で呼び出されるとは限りません。

使用例

次の例では、**Sub** ステートメントを使って、**Sub** プロシージャの名前、パラメータ、および本体を構成するコードを定義しています。

VB

```
Sub computeArea(ByVal length As Double, ByVal width As Double)
    ' Declare local variable.
    Dim area As Double
    If length = 0 Or width = 0 Then
        ' If either argument = 0 then exit Sub immediately.
        Exit Sub
    End If
    ' Calculate area of rectangle.
    area = length * width
    ' Print area to Immediate window.
    Debug.WriteLine(area)
End Sub
```

参照

処理手順

方法 : ジェネリック クラスを使用する

プロシージャのトラブルシューティング

関連項目

Implements ステートメント

Function ステートメント (Visual Basic)

パラメータの一覧

Dim ステートメント (Visual Basic)

Call ステートメント (Visual Basic)

Of

概念

パラメータ配列

SyncLock ステートメント

ステートメント ブロックを実行する前に、そのブロックに対する排他ロックを取得します。

```
SyncLock lockobject
    [ block ]
End SyncLock
```

指定項目
lockobject

必ず指定します。オブジェクト参照に評価される式です。

block

省略可能です。ロックを取得したとき実行されるステートメントのブロックです。

End SyncLock

SyncLock ブロックを終了します。

解説

SyncLock ステートメントを使うと、複数のスレッドがステートメント ブロックを同時に実行するのを回避できます。**SyncLock** は、ブロックを実行しているスレッドがなくなるまで、各スレッドがそのブロックに入らないようにします。

ほとんどの場合、**SyncLock** は、複数のスレッドによってデータが同時に更新されないよう保護するために使用されます。データを操作するステートメントを、途中で割り込まれることなく最後まで実行する必要がある場合は、ステートメントを **SyncLock** ブロックの内部に記述してください。

排他ロックに保護されるステートメント ブロックは、クリティカル セクションと呼ばれることもあります。

ルール

- 分岐。**SyncLock** ブロックの外部からブロックに分岐することはできません。
- ロック オブジェクトの値。*lockobject* の値を **Nothing** にすることはできません。ロック オブジェクトは、**SyncLock** ステートメントで使用する前に作成する必要があります。

  SyncLock ブロックの実行中に、*lockobject* の値を変更しないでください。排他ロックでは、ロック オブジェクトは変更されないでいることが必要です。

動作

- しくみ。スレッドは **SyncLock** ステートメントに到達すると、*lockobject* 式を評価して、式によって返されたオブジェクトの排他ロックを取得するまで実行を中断します。別のスレッドが **SyncLock** ステートメントに到達しても、そのスレッドは最初のスレッドが **End SyncLock** ステートメントを実行するまでロックを取得しません。
- データの保護。*lockobject* が **Shared** であれば、排他ロックは他のスレッドが **SyncLock** ブロックを実行している間は、クラスのどのインスタンス内のスレッドにも **SyncLock** ブロックを実行させません。これにより、すべてのインスタンスで共有されているデータを保護できます。

  lockobject が (**Shared** 変数ではなく) インスタンス変数であれば、排他ロックは現在のインスタンスで動作しているスレッドが、同じインスタンス内の別のスレッドと同時に **SyncLock** ブロックを実行するのを防ぎます。これにより、個々のインスタンスで保持されるデータを保護できます。
- 取得と解放。**SyncLock** ブロックは **Try...Finally** 構造に動作が似ており、**Try** ブロックで *lockobject* に対する排他ロックを取得し、**Finally** ブロックでロックを解放するように動作します。このため、**SyncLock** ブロックは、ブロックがどのように終了された場合でも、ロックを必ず解放します。これは、未処理の例外の場合にも該当します。
- Framework の呼び出し。**SyncLock** ブロックは、System.Threading 名前空間にある **Monitor** クラスの **Enter** メソッドと **Exit** メソッドを呼び出して、排他ロックを取得および解放します。

プログラミング手法

lockobject 式は、必ずクラスに排他的に属するオブジェクトに評価される必要があります。現在のインスタンスに属するデータを保護する場合は **Private** オブジェクト変数を宣言し、すべてのインスタンスに共通のデータを保護するには、**Private Shared** オブジェクト変数を宣言します。

キーワード **Me** を使用して、ロック オブジェクトをインスタンス データから利用できるようにしないでください。自分のクラスの外部にあるコードが、自分のクラスのインスタンスを参照する場合、**SyncLock** ブロックのロック オブジェクトが自分のロック オブジェクトとはまったく異なり、別々のデータを保護しているにもかかわらず、外部のコードがその参照を使用する可能性があります。このとき、自分のクラスと外部のクラスは互いにブロックし合って、関係のない **SyncLock** ブロックを実行させないようにします。同様に、文字列をロックする場合も問題が起きる可能性があります。プロセス内で同じ文字列を使用している他のすべてのコードが、同じロックを共有することになるからです。

また、**Me.GetType** メソッドを使用して、ロック オブジェクトを共有データから利用できるようにしないでください。なぜなら、**GetType** は特定のクラス名に対して、必ず同じ **Type** オブジェクトを返すためです。外部のコードが自分のクラスで **GetType** を呼び出し、自分が使用しているのと同じロック オブジェクトを取得する可能性があります。この結果、2 つのクラスが互いにブロックし合って、**SyncLock** ブロックを実行させなくなります。

使用例

メッセージの簡単なリストを保持するクラスは、次の例のようになります。このクラスはメッセージを配列に格納し、その配列で最後に使用された要素を変数に格納します。`addAnotherMessage` プロシージャは最後の要素をインクリメントし、新しいメッセージを格納します。これら 2 つの処理は **SyncLock** ステートメントと **End SyncLock** ステートメントで保護します。なぜなら、最後の要素がインクリメントされた後、他のスレッドが最後の要素をもう一度インクリメントする前に、新しいメッセージを格納する必要があるからです。

VB

```
Class simpleMessageList
    Public messagesList() As String = New String(50) {}
    Public messagesLast As Integer = -1
    Private messagesLock As New Object
    Public Sub addAnotherMessage(ByVal newMessage As String)
        SyncLock messagesLock
            messagesLast += 1
            messagesList(messagesLast) = newMessage
        End SyncLock
    End Sub
End Class
```

`simpleMessageList` クラスが 1 つのメッセージ リストをすべてのインスタンスで共有するのであれば、`messagesList` 変数と `messagesLast` 変数は **Shared** で宣言します。この場合、`messagesLock` も **Shared** で宣言して、すべてのインスタンスが単一のロック オブジェクトを使用するようにしてください。

参照

関連項目

System.Threading

Monitor

概念

スレッドの同期

その他の技術情報

Visual Basic におけるマルチスレッド

Throw ステートメント (Visual Basic)

プロシージャ内で例外をスローします。

```
Throw [ expression ]
```

指定項目
expression

スローする例外に関する情報を指定します。**Catch** ステートメント内では省略可能です。それ以外の場合は必須です。

解説

Throw ステートメントは、構造化例外処理コード (**Try**...**Catch**...**Finally**) または構造化しない例外処理コード (**On Error GoTo**) で処理できる例外をスローします。Visual Basic では、適切な例外処理コードが見つかるまで呼び出し履歴をさかのぼるため、**Throw** ステートメントを使ってコード内のエラーをトラップできます。

式を持たない **Throw** ステートメントを使用できるのは、**Catch** ステートメント内だけです。この場合、このステートメントは、**Catch** ステートメントによって現在処理されている例外を再スローします。

Throw ステートメントでは、*expression* 例外の呼び出し履歴がリセットされます。*expression* が指定されていない場合、呼び出し履歴は変更されません。例外の呼び出し履歴には、StackTrace プロパティを介してアクセスできます。

使用例

Throw ステートメントを使用して例外をスローするコードを次に示します。

VB

```
' Throws a new exception.
Throw New System.Exception("An exception has occurred.")
```

必要条件

名前空間 : Microsoft.VisualBasic

モジュール : **Interaction**

アセンブリ : Visual Basic ランタイム ライブラリ (Microsoft.VisualBasic.dll 内)

参照
処理手順
方法 : 例外の内部例外をチェックする
関連項目
Try...Catch...Finally ステートメント (Visual Basic)
On Error ステートメント (Visual Basic)
概念
Visual Basic の構造化例外処理の概要
非構造化例外処理の概要

Try...Catch...Finally ステートメント (Visual Basic)

コードの実行中に、コードのブロックで発生する可能性のあるエラーの一部またはすべてを処理する方法を提供します。

```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

指定項目

tryStatements

省略可能です。エラーが発生する可能性のあるステートメント。複合ステートメントを指定することもできます。

Catch

省略可能です。複数の **Catch** ブロックを指定できます。**Try** ブロックの処理中に例外が発生した場合は、それぞれの **Catch** ステートメントがテキストの順序で評価され、その例外を処理できるかどうかが調べられます。このとき、スローされた例外は *exception* で表されます。

exception

省略可能です。任意の変数名を指定します。*exception* の初期値は、スローされたエラーの値です。**Catch** と共に使用して、キャッチされたエラーを指定します。省略した場合、**Catch** ステートメントはどの例外でもキャッチします。

type

省略可能です。クラス フィルタの種類を指定します。*exception* の値が *type* で指定した型、または派生した型の値と一致する場合は、その識別子が例外オブジェクトにバインドされます。

When

省略可能です。**Catch** ステートメントに **When** 句が記述されている場合は、*expression* の評価が **True** になるときにのみ例外がキャッチされます。**When** 句は、例外の型をチェックした後にだけ適用されます。*expression* は、例外を表す識別子を参照することがあります。

expression

省略可能です。暗黙的にブール型 (**Boolean**) に変換できる必要があります。汎用フィルタを記述する任意の式。通常、エラー番号によるフィルタ処理に使用されます。**When** キーワードと共に使用して、エラーがキャッチされる状況を指定します。

catchStatements

省略可能です。関連付けられた **Try** ブロックで発生したエラーを処理するステートメントです。複合ステートメントを指定することもできます。

Exit Try

省略可能です。**Try...Catch...Finally** 構造から抜けるためのキーワードです。**End Try** ステートメントのすぐ下にあるコードから実行が再開されます。その場合でも、**Finally** ステートメントは実行されます。**Finally** ブロック内には記述できません。

Finally

省略可能です。**Try** ステートメントから抜けるときには、必ず **Finally** ブロックが実行されます。

finallyStatements

省略可能です。他のエラー処理がすべて行われた後に実行されるステートメント。

End Try

Try...Catch...Finally 構造の終わりを表します。

解説

Try ブロックと **Catch** ブロックは別々のブロックであるため、Try ブロック内のローカル変数を Catch ブロック内で使用することはできません。変数

を複数のブロックで使用するには、その変数を **Try...Catch...Finally** 構造の外で宣言します。

Try ブロックにはエラーが発生する可能性のあるコードを記述し、**Catch** ブロックには発生したエラーを処理するコードを記述します。**Try** ブロックでエラーが発生すると、そのエラーの処理に適した **Catch** ステートメントにプログラムの制御が渡されます。*exception* 引数は、**Exception** クラスのインスタンス、つまり **Exception** クラスから派生したクラスです。**Exception** クラス インスタンスは、**Try** ブロックで発生したエラーに対応します。インスタンスには、エラー番号やメッセージなどのエラー情報が含まれます。

Catch ステートメントは、*exception* 引数を指定しない場合、どのシステムまたはアプリケーションの例外でもキャッチします。これは、常に、**Try...Catch...Finally** 構造内で、期待する特定の例外すべてをキャッチした後の最後の **Catch** ブロックとして使用する必要があります。*exception* 引数がない場合、制御フローは、**Catch** の後の **Catch** ブロックに到達できません。

ネットワーク共有でホストされているアプリケーションなど、部分的に信頼されている状況では、**Try...Catch...Finally** はその呼び出しを含むメソッドが呼び出される前に発生したセキュリティ例外をキャッチしません。次のコード例をサーバー共有に配置し、そこから実行すると、"System.Security.SecurityException: 要求が失敗しました。" のエラーが発生します。セキュリティ例外の詳細については、SecurityException クラスの説明を参照してください。

VB

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
    Try
        Process.Start("http://www.microsoft.com")
    Catch ex As Exception
        MsgBox("Can't load Web page" & vbCrLf & ex.Message)
    End Try
End Sub
```

このような部分的に信頼されている状況では、`Process.Start` ステートメントを別の **Sub** に入れる必要があります。この **Sub** の最初の呼び出しは失敗し、**Try...Catch** は、`Process.Start` を含んでいる **Sub** が開始されてセキュリティ例外が生成される前に、その失敗をキャッチできます。

メモ：

Try ステートメント内に **Catch** ブロックが 1 つもない場合は、**Finally** ブロックを含める必要があります。

使用例

次の簡単な例は、**Try...Catch...Finally** ステートメントの構造を示しています。

VB

```
Public Sub TryExample()
    Dim x As Integer = 5    ' Declare variables.
    Dim y As Integer = 0
    Try                     ' Set up structured error handling.
        x = x \ y           ' Cause a "Divide by Zero" error.
    Catch ex As Exception When y = 0        ' Catch the error.
        Beep()
        MsgBox("You tried to divide by 0.") ' Show an explanatory message.
    Finally
        Beep()              ' This line is executed no matter what.
    End Try
End Sub
```

参照

関連項目

End ステートメント
Err オブジェクト (Visual Basic)
Exit ステートメント (Visual Basic)
On Error ステートメント (Visual Basic)
GoTo ステートメント
Exception

Using ステートメント (Visual Basic)

Using ブロックの開始を宣言し、オプションでこのブロックが制御するシステム リソースを取得します。

```
Using { resourcelist | resourceexpression }
    [ statements ]
End Using
```

指定項目

resourcelist

resourceexpression を指定しない場合は必ず指定します。この **Using** ブロックが制御する、1 つ以上のシステム リソースのリストです。

resourceexpression

resourcelist を指定しない場合は必ず指定します。この **Using** ブロックによって制御されるシステム リソースを参照する参照変数または式です。

statements

省略可能です。**Using** ブロックが実行するステートメントのブロックです。

End Using

必ず指定します。**Using** ブロックの定義を終了し、制御しているすべてのリソースを破棄します。

resourcelist 部分のリソースは、次の構文と指定項目で指定します。

```
resourcename As New resourcetype [ ( [ arglist ] ) ]
```

または

```
resourcename As resourcetype = resourceexpression
```

resourcelist の指定項目

resourcename

必ず指定します。**Using** ブロックが制御するシステム リソースを参照する参照変数です。

New

Using ステートメントがリソースを取得する場合は、必ず指定します。リソースを既に取得している場合は、2 つ目の構文を使用してください。

resourcetype

必ず指定します。リソースのクラス。このクラスは、IDisposable インターフェイスを実装する必要があります。

arglist

省略可能です。*resourcetype* のインスタンスを作成するために、コンストラクタに渡す引数のリストです。パラメータの一覧 を参照してください。

resourceexpression

必ず指定します。*resourcetype* の要件を満たすシステム リソースを参照する変数または式です。2 番目の構文を使用する場合は、制御を **Using** ステートメントに渡す前にリソースを取得する必要があります。

解説

ファイル ハンドル、COM ラッパー、SQL 接続などのアンマネージ リソースを取得するコードを作成する場合があります。そのようなコードに **Using** ブロックを使用すると、リソースを使い終わったとき、その 1 つ以上のリソースを確実に破棄できます。これにより、他のコードからそれらのリソースを使うことが可能になります。

マネージ リソースであれば、何もコードを追加しなくても、.NET Framework のガベージ コレクタ (GC) が破棄してくれます。マネージ リソースに **Using** ブロックは必要ありません。

Using ブロックは、取得、使用、破棄という 3 つの部分で構成されます。

- 取得システム リソースを参照するための変数を作成、および初期化します。**Using** ステートメントでは 1 つ以上のリソースを取得すること

も、ブロックに入る前にリソースを 1 つ取得して、それを **Using** ステートメントに指定することもできます。*resourceexpression* を指定する場合は、制御を **Using** ステートメントに渡す前にリソースを取得する必要があります。

- 使用リソースにアクセスし、それを使って処理を実行します。リソースを使用するステートメントは、**Using** と **End Using** の間に記述します。
- 破棄*resourcename* に指定されたオブジェクトから **Dispose** メソッドを呼び出します。これによって、オブジェクトはアンマネージ リソースを適切に終了させることができます。**End Using** ステートメントはリソースを **Using** ブロックの制御に基づいて破棄します。

動作

Using ブロックは **Try**...**Finally** 構造 (**Try** でリソースを使用し、**Finally** ブロックでリソースを破棄する) と同様に動作します。このため、**Using** ブロックを使えば、ブロックがどのように終了した場合でも、リソースは必ず破棄されます。これは、StackOverflowException を除く未処理の例外の場合にも該当します。

Using ステートメントによって取得されるリソース変数のスコープは、必ず **Using** ブロックに制限されます。

Using ステートメントで 1 つ以上のシステム リソースを指定すると、**Using** ブロックを入れ子にするのと同じ意味になります。

Using ブロック内の構造化例外処理

Using ブロック内で起きる可能性のある例外を処理することが必要な場合は、ブロック内に **Try**...**Finally** 構造全体を追加してください。**Using** ステートメントがリソースの取得に失敗するケースに対応する必要がある場合は、*resourcename* が **Nothing** であるかどうかをテストします。

Using ブロック以外での構造化例外処理

リソースの取得を細かく制御する必要がある場合、または **Finally** ブロックにコードを追加する必要がある場合は、**Using** ブロックを **Try**...**Finally** 構造に書き換えてください。次に示すスケルトンは、どちらも `resource` の取得と破棄の処理を実行する **Try** 構造と **Using** 構造の例です。

```
            Using resource As New resourceType
    ' Insert code to work with resource.
End Using
' THE FOLLOWING TRY CONSTRUCTION IS EQUIVALENT TO THE USING BLOCK
Dim resource As New resourceType
Try
    ' Insert code to work with resource.
Catch ex As Exception
    ' Insert code to process exception.
Finally
    ' Insert code to do additional processing before disposing of resource.
    resource.Dispose()
End Try
```

メモ :

Using ブロック内のコードで、*resourcename* に含まれるオブジェクトを別の変数に割り当てることはできません。**Using** ブロックの終了時にリソースが破棄され、他の変数は自分がポイントしているリソースにアクセスできなくなります。

使用例

次の例は **Using** ブロックを使用して新しいフォントを取得します。ブロックが終了したとき、必ずシステムによって **Dispose** メソッドがフォント上で呼び出されます。

VB

```
Public Sub setbigbold(ByVal c As Control)
    Using nf As New System.Drawing.Font("Arial", 12.0F, _
        System.Drawing.FontStyle.Bold)

        c.Font = nf
        c.Text = "This is 12-point Arial bold"
    End Using
End Sub
```

参照

処理手順

方法 : システム リソースを破棄する

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

IDisposable

System.Drawing

Font

While...End While ステートメント (Visual Basic)

指定された条件が真 (**True**) である間、一連のステートメントを繰り返し実行します。

```
While condition
    [ statements ]
    [ Exit While ]
    [ statements ]
End While
```

指定項目

condition

必ず指定します。**Boolean** を指定します。*condition* が **Nothing** の場合、Visual Basic では偽 (**False**) として扱われます。

statements

省略可能です。1 つ以上のステートメントを **While** の後に指定します。この部分は、*condition* が **True** であるたびに実行されます。

Exit While

省略可能です。制御を **While** ブロックの外へ移動します。

End While

必ず指定します。**While** ブロックの定義を終了します。

解説

条件が **True** の間、一連のステートメントを不特定の回数繰り返すには、**While...End While** 構造を使用します。条件をテストする場所またはテスト結果の判定を柔軟に指定する必要がある場合は、Do...Loop ステートメント (Visual Basic) の方が適しています。ステートメントを特定の回数繰り返す場合は、For...Next ステートメント (Visual Basic) を使用することをお勧めします。

condition が **True** である場合、**End While** が現れるまですべての *statements* が実行されます。その後、制御は再び **While** ステートメントに戻り、*condition* が再び評価されます。*condition* が真 (**True**) の間、この処理が繰り返されます。**False** である場合は、制御は **End While** ステートメントの後のステートメントに渡ります。

規則

- **条件の名前。** 条件は、通常、2 つの値の比較によって判断されますが、ブール型 (Boolean) (Visual Basic) 値 (**True** または **False**) として評価できる式の場合どのような式でも指定できます。指定できる値には、数値型などのその他のデータ型を **Boolean** に変換した値も含まれます。
- **条件のテスト。** **While** ステートメントは、ループを開始する前に必ず条件を確認します。条件が **True** である限りループが継続されます。
- **反復回数。** ループに最初に入った時点で *condition* が **False** の場合、ループは一度も実行されません。
- **ループの入れ子。** **While** ループは入れ子構造にできます。つまり、**While** ループの内部に別の **While** ループを入れることができます。また、種類の異なる制御構造を入れ子にすることもできます。詳細については、「入れ子になった制御構造」を参照してください。
- **ループの終了。** Exit ステートメント (Visual Basic) は、**End While** ステートメントの次のステートメントにすぐに制御を移します。ループの継続が不要になったり不可能になったりする条件 (エラー値や終了要求など) を検出した場合にループを終了できます。任意の数の **Exit While** ステートメントを **While** ループ内の任意の場所に配置できます。多くの場合に、**Exit While** は、条件評価の直後に使用されます。たとえば、**If...Then...Else** 構造では、このような使い方が一般的です。
- **無限ループ。** **Exit While** は、無限ループを引き起こす可能性がある条件をテストする場合によく使用されます。無限ループとは、実行回数が極端に多い (または無限に繰り返される) ループです。このような条件を検出した場合は、**Exit While** を使ってループを終了できます。詳細については、「Do...Loop ステートメント (Visual Basic)」を参照してください。

使用例

次のコード例は、**While...End While** ステートメントを使って、カウンタ変数をインクリメントします。ループ内に記述されたステートメントは、条件の評価が **True** の間、実行されます。

VB

```
Dim counter As Integer = 0
While counter < 20
    counter += 1
    ' Insert code to use current value of counter.
End While
MsgBox("While loop ran " & CStr(counter) & " times")
```

参照

関連項目

Do...Loop ステートメント (Visual Basic)
For...Next ステートメント (Visual Basic)
ブール型 (Boolean) (Visual Basic)
Exit ステートメント (Visual Basic)

概念

ループ構造
入れ子になった制御構造

With...End With ステートメント (Visual Basic)

オブジェクトや構造体への参照を繰り返し指定する、一連のステートメントを実行します。

```
With object
    [ statements ]
End With
```

指定項目

object

必ず指定します。変数または式です。基本データ型だけでなく、どのデータ型として評価される式でも指定できます。

statements

省略可能です。*object* に対して実行される 1 つ以上のステートメントを、**With** と **End With** の間に定義します。

End With

必ず指定します。**With** ブロックの定義を終了します。

解説

With...End With ステートメントを使用すると、特定のオブジェクトの名前を再定義することなく、そのオブジェクトに対して一連のステートメントを実行できます。オブジェクトの修飾パスが長い場合は、**With...End With** を使用してパフォーマンスを高めることができます。また、**With** ブロックを使うと修飾パスを何度も入力する必要がないため、入力を誤る可能性が低くなります。

たとえば、単一のオブジェクトに対して複数の異なるプロパティを変更する場合、プロパティを割り当てるステートメントを **With...End With** 内に指定すると、プロパティを割り当てるたびにオブジェクトを参照するのではなく、一度参照するだけで済みます。

規則

- **データ型***object* のデータ型には、任意のクラス型や構造体の型、または Visual Basic の基本型 (**Integer** など) も使用できます。.NET Framework ではすべての基本型をクラスまたは構造体に使用でき、そのメンバには **With** ブロックの内部からアクセスできます。
- **宣言***object* は、**With** ブロックの前に宣言する必要があります。**With** ステートメントの中には宣言できません。
- **反復の回数**With ブロックは反復処理を行うブロックではありません。ブロックの内部にループがなければ、ステートメントは一度だけ実行されます。
- **入れ子構造**With...End With は入れ子構造にできます。つまり、**With...End With** の内部に別の **With...End With** を入れることができます。カスタマイズ例については、「方法 : オブジェクトに対して複数のアクションを実行する」を参照してください。

  ただし、外側のステートメントのメンバは内側のステートメントに対してはマスクされているので、外側の **With** ブロックに含まれているオブジェクトのメンバを参照するときは、内側の **With** でオブジェクト参照を完全に修飾する必要があります。

  また、さまざまな種類の制御構造を入れ子にすることもできます。詳細については、「入れ子になった制御構造」を参照してください。

- **制御の転送。**Visual Basic では、Exit ステートメント (Visual Basic) を使用して制御を **With** ブロックの外部に転送できません。すべてのステートメントを実行し終える前に終了するには、**End With** ステートメントにラベルを設定し、GoTo ステートメントを使用してそのラベルに分岐します。詳細については、「方法 : ステートメントにラベル付けする」を参照してください。

  制御を **With** ブロックの外部から内部に転送したり、内部から外部に転送したりすることはできません。ブロックの内部からプロシージャを呼び出すことは可能ですが、制御はその次のステートメントに戻ります。

- **他のオブジェクトへのアクセス。With** ブロックに入ったら、**End With** ステートメントを通過するまで *object* を割り当て直すことはできません。このため、修飾せずにアクセスできるのは、指定したオブジェクトのメソッドやプロパティだけです。他のオブジェクトのメソッドやプロパティを使用するには、オブジェクト名で修飾する必要があります。

使用例

次の例は、**With** ブロックを使用して、単一のオブジェクトに対する一連のステートメントを実行します。この例では `testObject` オブジェクトが既に作成済みで、参照されるプロパティを公開している必要があります。

VB

```
With testObject
```

```
        .Height = 100
        .Text = "Hello, World"
        .ForeColor = System.Drawing.Color.Green
        .Font = New System.Drawing.Font(.Font, _
            System.Drawing.FontStyle.Bold)
    End With
```

参照

処理手順

方法 : オブジェクトに対して複数のアクションを実行する

方法 : ステートメントにラベル付けする

関連項目

Exit ステートメント (Visual Basic)

GoTo ステートメント

概念

入れ子になった制御構造

宣言コンテキストと既定のアクセス レベル

このトピックでは、Visual Basic のどのデータ型がどのデータ型の中で宣言できるかについて説明します。また、指定がない場合の各データ型の既定のアクセス レベルを示します。

宣言コンテキストのレベル

宣言コンテキストとは、プログラミング要素が宣言されるコードの領域のことです。別のプログラミング要素の中であることも多く、この場合このプログラミング要素は "コンテナ要素" と呼ばれます。

宣言コンテキストのレベルは次のとおりです。

- 名前空間レベル : 同じソース ファイルまたは名前空間内にあるが、クラス、構造体、モジュール、インターフェイスは異なる
- モジュール レベル : 同じクラス、構造体、モジュール、インターフェイス内にあるが、プロシージャやブロックは異なる
- プロシージャ レベル : 同じプロシージャまたはブロック (**If** や **For** など) 内にある

次の表に、宣言されたさまざまなプログラミング要素の既定のアクセス レベルを、宣言コンテキスト別に示します。

宣言された要素	名前空間レベル	モジュール レベル	プロシージャ レベル
変数 (Dim ステートメント (Visual Basic))	不可	**Private** (**Structure** では **Public**、**Interface** では不可)	**Public**
定数 (Const ステートメント (Visual Basic))	不可	**Private** (**Structure** では **Public**、**Interface** では不可)	**Public**
列挙型 (Enum ステートメント (Visual Basic))	**Friend**	**Public**	不可
クラス (Class ステートメント (Visual Basic))	**Friend**	**Public**	不可
構造体 (Structure ステートメント)	**Friend**	**Public**	不可
モジュール (Module ステートメント)	**Friend**	不可	不可
インターフェイス (Interface ステートメント (Visual Basic))	**Friend**	**Public**	不可
プロシージャ (Function ステートメント (Visual Basic)、Sub ステートメント (Visual Basic))	不可	**Public**	不可
外部参照 (Declare ステートメント)	不可	**Public** (**Interface** では不可)	不可
演算子 (Operator ステートメント)	不可	**Public** (**Interface** または **Module** では不可)	不可
プロパティ (Property ステートメント)	不可	**Public**	不可
既定のプロパティ (Default (Visual Basic))	不可	**Public** (**Module** では不可)	不可
イベント (Event ステートメント)	不可	**Public**	不可
デリゲート (Delegate ステートメント)	**Friend**	**Public**	不可

詳細については、「Visual Basic でのアクセス レベル」を参照してください。

参照

関連項目

Friend (Visual Basic)
Private (Visual Basic)
Public (Visual Basic)

属性リスト

宣言されたプログラミング要素に適用する属性を指定します。複数の属性を指定するときは、コンマ (,) で区切ります。属性を 1 つ定義する場合の構文は次のとおりです。

```
[ attributemodifier ] attributename [ ( attributearguments | attributeinitializer ) ]
```

指定項目

attributemodifier

ソース ファイルの先頭に定義する属性に、必ず指定します。Assembly または モジュール を指定できます。

attributename

必ず指定します。属性の名前です。

attributearguments

省略可能です。この属性の位置の引数のリストです。複数の引数を指定するときは、コンマ (,) で区切ります。

attributeinitializer

省略可能です。この属性の変数初期化子またはプロパティ初期化子のリストです。複数の初期化子を指定するときは、コンマ (,) で区切ります。

解説

ほとんどすべてのプログラミング要素 (型、プロシージャ、プロパティなど) に、1 つ以上の属性を適用できます。属性はアセンブリのメタデータに格納され、コードに注釈を付けたり、特定のプログラミング要素の使い方を指定したりするのに役立ちます。Visual Basic や .NET Framework で定義された属性を適用することも、独自の属性を定義することもできます。

属性を使用する状況の詳細については、「Visual Basic における属性」および「属性の一般的な使用方法」を参照してください。属性名の詳細については、「宣言された要素の名前」を参照してください。

ルール

- **定義する場所。** 属性は、宣言されたほとんどのプログラミング要素に適用できます。1 つ以上の属性を適用するには、要素の宣言の先頭に属性ブロックを記述します。属性リストの各エントリには、適用する属性の他に、その属性を呼び出す際に使用する修飾子や引数を指定します。
- **山かっこ。** 属性リストを指定するには、属性を山かっこ ("<" と ">") で囲む必要があります。
- **宣言に含める。** 属性は独立したステートメントとして宣言するのではなく、要素の宣言に含める必要があります。行連結シーケンス ("_") を使用すると、宣言ステートメントを複数行に続くソース コードに記述できます。
- **修飾子** ソース ファイルの先頭にあるプログラミング要素に適用する属性には、必ず属性修飾子 (**Assembly** または **Module**) を指定する必要があります。それ以外のプログラミング要素に適用される属性に対して、属性修飾子を指定することはできません。
- **引数。** 属性に対するすべての位置指定引数を、すべての変数初期化子またはプロパティ初期化子の前に置く必要があります。

使用例

DllImportAttribute 属性を **Function** プロシージャのスケルトン定義に適用する例は次のようになります。

VB

```
<DllImportAttribute("kernel32.dll", EntryPoint:="MoveFileW", _
    SetLastError:=True, CharSet:=CharSet.Unicode, _
    ExactSpelling:=True, _
    CallingConvention:=CallingConvention.StdCall)> _
Public Shared Function moveFile(ByVal src As String, _
  ByVal dst As String) As Boolean
    ' This function copies a file from the path src to the path dst.
    ' Leave this function empty. The DLLImport attribute forces calls
    ' to moveFile to be forwarded to MoveFileW in KERNEL32.DLL.
End Function
```

DllImportAttribute は、属性が適用されたプロシージャが、アンマネージ ダイナミック リンク ライブラリ (DLL: Dynamic-Link Library) のエントリ ポイントであることを示します。この属性は、DLL 名を位置指定引数として指定し、その他の情報を変数初期化子として指定します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

Assembly

Module (Visual Basic)

概念

属性の一般的な使用方法

属性の適用

その他の技術情報

Visual Basic における属性

パラメータの一覧

呼び出し時に、プロシージャが受け取ることのできるパラメータを指定します。複数のパラメータを指定するときは、コンマ (,) で区切ります。パラメータを 1 つ定義する場合の構文は次のとおりです。

```
[ <attributelist> ] [ Optional ] [{ ByVal | ByRef }] [ ParamArray ]
parametername[( )] [ As parametertype ] [ = defaultvalue ]
```

指定項目

attributelist

省略可能です。このパラメータに適用する属性の一覧を指定します。属性リストは山かっこ ("<" および ">") で囲む必要があります。

Optional

省略可能です。プロシージャを呼び出すときに、このパラメータを省略できることを示します。

ByVal

省略可能です。呼び出し元のコードにある、対応する引数の基になる可変要素を、プロシージャが置き換えたり再割り当てしたりできないことを示します。

ByRef

省略可能です。呼び出し元のコードにある基の可変要素を、呼び出し元のコード自体と同じようにプロシージャが変更できることを示します。

ParamArray

省略可能です。パラメータ リストの最後のパラメータが、指定されたデータ型の省略可能な要素の配列であることを示します。これを指定すると、呼び出し元のコードから任意の数の引数をプロシージャに渡すことができます。

parametername

必ず指定します。パラメータを表すローカル変数の名前を指定します。

parametertype

Option Strict が **On** の場合は、必ず指定します。パラメータを表すローカル変数のデータ型を指定します。

defaultvalue

Optional パラメータの場合は必ず指定します。任意の定数、または結果がパラメータのデータ型になる定数式を指定します。型が **Object**、つまりクラス、インターフェイス、配列、または構造体の場合、既定値に指定できるのは **Nothing** だけです。

解説

パラメータはかっこで囲み、コンマで区切って指定します。パラメータはどのデータ型でも宣言できます。*parametertype* を指定しなかった場合は、既定値の **Object** に設定されます。

呼び出し元のコードはプロシージャを呼び出すとき、各必須パラメータに引数を渡します。詳細については、「パラメータと引数の違い」を参照してください。

呼び出し元のコードが各パラメータに渡す引数は、呼び出し元のコードにある基の要素へのポインタです。この要素が不変 (定数、リテラル、列挙値、または式) である場合、どのコードからも要素を変更できません。要素が可変 (宣言された変数、フィールド、プロパティ、配列要素、構造体要素) である場合は、呼び出し元のコードから要素を変更できます。詳細については、「変更できる引数と変更できない引数の違い」を参照してください。

可変要素が **ByRef** で渡された場合は、プロシージャからも要素を変更できます。詳細については、「引数の値渡しと参照渡しの違い」を参照してください。

ルール

- **かっこ。** パラメータ リストを指定する場合は、かっこで囲む必要があります。パラメータを指定しない場合でも、空のリストをかっこで囲むことができます。こうすると、プロシージャであると明確に示すことができるので、コードが読みやすくなります。
- **省略可能なパラメータ。** パラメータに **Optional** 修飾子を指定すると、リスト内の後続のすべてのパラメータにも **Optional** 修飾子を使用し、オプションで宣言することが必要になります。

省略可能なパラメータの宣言には、それぞれ *defaultvalue* 句を指定する必要があります。

- **パラメータ配列。ParamArray** パラメータには、**ByVal** を指定する必要があります。

  同じパラメータ リストに **Optional** と **ParamArray** の両方を指定することはできません。

- **値渡し。**既定では、すべての引数が **ByVal** で渡されます。つまり、プロシージャは基になる可変要素を変更できません。ただし、要素が参照型であれば、プロシージャは基になるオブジェクトの内容やメンバを更できます (オブジェクト自体を置き換えたり再割り当てしたりすることはできません)。

- **パラメータ名。**パラメータのデータ型が配列である場合は、*parametername* のすぐ後にかっこを記述します。パラメータ名の詳細については、「宣言された要素の名前」を参照してください。

使用例

パラメータが 2 つ定義された **Function** プロシージャの例は、次のようになります。

VB

```
Public Function howMany(ByVal ch As Char, ByVal st As String) As Integer
End Function
Dim howManyA As Integer = howMany("a"c, "How many a's in this string?")
```

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)
Declare ステートメント
Structure ステートメント
Option Strict ステートメント
DllImportAttribute

概念

属性の適用

その他の技術情報

Visual Basic における属性

型リスト

ジェネリックなプログラミング要素に型パラメータを指定します。複数のパラメータを指定するときは、コンマ (,) で区切ります。型パラメータを 1 つ定義する場合の構文は次のとおりです。

```
typename [ As constraintlist ]
```

指定項目

typename

必ず指定します。型パラメータの名前です。これはプレースホルダです。対応する型引数で指定される定義済みの型に置き換えられます。

constraintlist

省略可能です。*typename* に指定可能なデータ型を制約する要件のリストです。制約を複数指定する場合は、中かっこ ({ }) で囲み、コンマで区切って記述します。制約リストには As キーワードを付ける必要があります。**As** はリストの先頭に一度だけ記述します。

解説

すべてのジェネリックなプログラミング要素は、型パラメータを少なくとも 1 つ受け取る必要があります。型パラメータは特定の型 (構成される要素) のプレースホルダであり、クライアント コードでジェネリック型のインスタンスを作成するときに指定されます。クラス、構造体、インターフェイス、プロシージャ、またはデリゲートをジェネリックで定義できます。

ジェネリック型を定義する状況については、「Visual Basic におけるジェネリック型」を参照してください。型パラメータの名前については、「宣言された要素の名前」を参照してください。

ルール

- **かっこ。** 型パラメータのリストを指定する場合は、リストをかっこで囲み、Of キーワードを使って特定する必要があります。**Of** はリストの先頭に一度だけ記述します。
- **制約。** 型パラメータに対する制約のリストには、次の項目を任意の組み合わせで定義できます。
  - 任意の数のインターフェイス。指定する型は、このリストにあるすべてのインターフェイスを実装している必要があります。
  - 1 つ以下のクラス。指定する型は、そのクラスから継承している必要があります。
  - New (Visual Basic) キーワード。指定する型は、ジェネリック型からアクセス可能な、パラメータを持たないコンストラクタを公開している必要があります。これは、型パラメータを 1 つ以上のインターフェイスで制約している場合に使用します。インターフェイスを実装する型が、必ずしもコンストラクタを公開しているとは限りません。また、コンストラクタのアクセス レベルによっては、ジェネリック型の内部のコードからアクセスできない可能性もあります。
  - Class (Visual Basic) キーワードまたは Structure (Visual Basic) キーワード。**Class** を定義した場合、指定する型は参照型であることが必要です。**Structure** を定義した場合、指定する型は値型であることが必要です。同じ *constraintlist* で **Class** と **Structure** の両方を指定することはできません。

  指定する型は、*constraintlist* に定義されたすべての要件を満たす必要があります。

  型パラメータの制約は、それぞれ他の型パラメータの制約と関連しません。

動作

- **コンパイル時の代入。** ジェネリックなプログラミング要素から構築型を作成する場合は、各型パラメータに対して定義済みの型を指定します。Visual Basic コンパイラは、ジェネリックな要素の内部に出現するすべての *typename* に、指定された型を代入します。
- **制約の省略。** 型パラメータに制約を指定しなければ、コードはその型パラメータにおいて オブジェクト型 (Object) でサポートされた操作およびメンバに制限されます。

使用例

ジェネリックなディクショナリ クラスのスケルトン定義の例を次に示します。スケルトン関数がディクショナリに新しいエントリを追加しています。

VB

```
Public Class dictionary(Of entryType, keyType As {IComparable, IFormattable, New})
    Public Sub add(ByVal et As entryType, ByVal kt As keyType)
```

```
            Dim dk As keyType
            If kt.CompareTo(dk) = 0 Then
            End If
        End Sub
    End Class
```

`dictionary` がジェネリックなので、コードはそこからさまざまなオブジェクトを作成できます。各オブジェクトは同じ関数を含みますが、さまざまなデータ型に対して処理を実行します。文字列 (**String**) のエントリと整数 (**Integer**) のキーを使って `dictionary` オブジェクトを作成するコード行の例を次に示します。

VB

```
Dim dictInt As New dictionary(Of String, Integer)
```

先の例と同等のスケルトン定義の例は、次のとおりです。

VB

```
Public Class dictionary
    Public Sub add(ByVal et As String, ByVal kt As Integer)
        Dim dk As Integer
        If kt.CompareTo(dk) = 0 Then
        End If
    End Sub
End Class
```

参照

処理手順

方法 : ジェネリック クラスを使用する

関連項目

Of

New (Visual Basic)

Class (Visual Basic)

Structure (Visual Basic)

オブジェクト型 (Object)

Function ステートメント (Visual Basic)

Structure ステートメント

Sub ステートメント (Visual Basic)

概念

Visual Basic でのアクセス レベル

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

Visual Basic コンパイラは、コード中のドキュメント コメントを処理して XML ファイルを生成できます。専用のツールを使用すると、この XML ファイルからドキュメントを生成できます。

XML コメントは、型や型メンバなどのコードの構成体に対して使用できます。partial 型の場合は、型の 1 部分にのみ XML コメントを使用できます。ただし、部分型のメンバのコメント作成に対する制限は何もありません。

メモ :
ドキュメント コメントは名前空間には使用できません。その理由は、1 つの名前空間に複数のアセンブリが属し、すべてのアセンブリが同時にロードされるとは限らないからです。

コンパイラは、有効な XML のタグをすべて処理します。ユーザー ドキュメントで一般的に使用される機能を提供するタグを次の表に示します。

<c>	<code>	<example>
<exception> [1]	<include> [1]	<list>
<para>	<param> [1]	<paramref>
<permission> [1]	<remarks>	<returns>
<see> [1]	<seealso> [1]	<summary>
<typeparam> [1]	<value>	

([1] コンパイラが構文を検証します。)

メモ :
ドキュメント コメントのテキストに山かっこを出力するには、`<` と `>` を使用します。たとえば、文字列 `"<text in angle brackets >"` は、`<text in angle brackets>` と出力されます。

参照

処理手順

方法 : Visual Basic で XML ドキュメントを作成する

関連項目

/doc

概念

XML の使用によるコードのドキュメントの作成 (Visual Basic)

<c> (Visual Basic)

説明の内部にあるテキストがコードであることを示します。

```
<c>text</c>
```

パラメータ
text

コードとして指定するテキスト。

解説

<c> タグを使用すると、説明内のテキストをコードとして指定できます。コードとして複数行を指定する場合は、<code> (Visual Basic) タグを使用します。

コンパイル時に /doc を指定すると、ドキュメント コメントをファイルに出力できます。

使用例

summary セクション内で <c> タグを使用して、Counter がコードであることを示す例は次のとおりです。

VB

```
''' <summary>
''' Resets the value the <c>Counter</c> field.
''' </summary>
Public Sub ResetCounter()
    counterValue = 0
End Sub
Private counterValue As Integer = 0
''' <summary>
''' Returns the number of times Counter was called.
''' </summary>
''' <value>Number of times Counter was called.</value>
Public ReadOnly Property Counter() As Integer
    Get
        counterValue += 1
        Return counterValue
    End Get
End Property
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<code> (Visual Basic)

テキストが複数行のコードであることを示します。

```
<code>content</code>
```

パラメータ
content

コードとしてマークするテキストです。

解説

複数の行をコードとして示すには <code> タグを使用します。説明内のテキストをコードとして指定する場合は、<c> (Visual Basic) タグを使用します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、<code> タグを使用して、ID フィールドの使用例を含めます。

VB

```
Public Class Employee
    ''' <remarks>
    ''' <example> This sample shows how to set the <c>ID</c> field.
    ''' <code>
    ''' Dim alice As New Employee
    ''' alice.ID = 1234
    ''' </code>
    ''' </example>
    ''' </remarks>
    Public ID As Integer
End Class
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<example> (Visual Basic)

メンバの例を指定します。

```
<example>description</example>
```

パラメータ
description

サンプル コードの説明。

解説

`<example>` タグでは、メソッドやその他のライブラリ メンバの使用例を指定します。<code> (Visual Basic) タグと組み合わせて使用するのが一般的です。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、`<example>` タグを使用して、`ID` フィールドの使用例を含めます。

VB

```
Public Class Employee
    ''' <remarks>
    ''' <example> This sample shows how to set the <c>ID</c> field.
    ''' <code>
    ''' Dim alice As New Employee
    ''' alice.ID = 1234
    ''' </code>
    ''' </example>
    ''' </remarks>
    Public ID As Integer
End Class
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<exception> (Visual Basic)

スローできる例外を指定します。

```
<exception cref="member">description</exception>
```

パラメータ

member

現在のコンパイル環境から利用可能な例外への参照。コンパイラは、指定された例外が存在することを確認してから、*member* を出力先 XML 内で標準要素名に変換します。*member* は二重引用符 (" ") で囲みます。

description

説明。

解説

スローできる例外を指定するには `<exception>` タグを使用します。このタグは、メソッド定義に適用されます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、`<exception>` タグを使用して、`IntDivide` 関数がスローできる例外を指定します。

VB

```
''' <exception cref="System.OverflowException">
''' Thrown when <paramref name="denominator"/><c> = 0</c>.
''' </exception>
Public Function IntDivide( _
        ByVal numerator As Integer, _
        ByVal denominator As Integer _
) As Integer
    Return numerator \ denominator
End Function
```

参照

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<include> (Visual Basic)

ソース コード内の型やメンバを記述する別のファイルを参照します。

```
<include file="filename" path="tagpath[@name='id']" />
```

パラメータ

filename

必ず指定します。ドキュメントを含むファイルの名前。ファイル名にパスを指定することもできます。*filename* は二重引用符 (" ") で囲みます。

tagpath

必ず指定します。*filename* のタグのパス。その後ろにタグの *name* を指定します。パスは二重引用符 (" ") で囲みます。

name

必ず指定します。タグの名前指定子。その後ろにコメントを指定します。*Name* には *id* を指定します。

id

必ず指定します。タグの ID。その後ろにコメントを指定します。ID は、単一引用符 (' ') で囲みます。

解説

<include> タグを使用すると、ソース コード内の型およびメンバの説明として、別のファイル内のコメントを参照できます。これはソース コードのファイルにドキュメント コメントを直接記述しない方法です。

<include> タグは、W3C の XML Path Language (XPath) Version 1.0 Recommendation を使用します。<include> のカスタマイズ方法の詳細については、http://www.w3.org/TR/xpath を参照してください。

使用例

この例では、<include> タグを使用してメンバもドキュメントのコメントを commentFile.xml ファイルからインポートします。

VB

```
''' <include file="commentFile.xml"
''' path="Docs/Members[@name='Open']/*" />
Public Sub Open(ByVal filename As String)
    ' Code goes here.
End Sub
''' <include file="commentFile.xml"
''' path="Docs/Members[@name='Close']/*" />
Public Sub Close(ByVal filename As String)
    ' Code goes here.
End Sub
```

commentFile.xml の形式は次のとおりです。

```
<Docs>
<Members name="Open">
<summary>Opens a file.</summary>
<param name="filename">File name to open.</param>
</Members>
<Members name="Close">
<summary>Closes a file.</summary>
<param name="filename">File name to close.</param>
</Members>
</Docs>
```

参照

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<list> (Visual Basic)

リストまたは表を定義します。

```
<list type="type">
   <listheader>
      <term>term</term>
      <description>description</description>
   </listheader>
   <item>
      <term>term</term>
      <description>description</description>
   </item>
</list>
```

パラメータ

type

リストの型を指定します。箇条書きリストの場合は "bullet"、番号付きリストの場合は "number"、2 列の表の場合は "table" です。

term

type が "table" の場合のみ使用されます。説明タグで定義された、定義する用語です。

description

type が "bullet" または "number" の場合、*description* はリスト内の項目です。*type* が "table" の場合、*description* は *term* の定義です。

解説

`<listheader>` ブロックは、表または定義リストの見出しの定義に使用します。表を定義する場合は、見出しには *term* のエントリだけを指定します。

リストの各項目は、`<item>` ブロックで指定します。定義リストを作成するときは、*term* と *description* の両方を指定します。ただし、表、箇条書きリスト、または番号付きリストに指定する必要があるのは、*description* のエントリだけです。

リストや表には、必要な数だけ `<item>` ブロックを指定できます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、`<list>` タグを使って、解説セクションに箇条書きリストを定義します。

VB

```
''' <remarks>Before calling the <c>Reset</c> method, be sure to:
''' <list type="bullet">
''' <item><description>Close all connections.</description></item>
''' <item><description>Save the object state.</description></item>
''' </list>
''' </remarks>
Public Sub Reset()
    ' Code goes here.
End Sub
```

参照

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<para> (Visual Basic)

コンテンツが段落として書式設定されていることを示します。

```
<para>content</para>
```

パラメータ
content

段落のテキスト。

解説

<para> タグは、<summary> (Visual Basic)、<remarks> (Visual Basic)、<returns> (Visual Basic) などのタグの中で使用され、テキストに構造体を追加します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、<para> タグを使用して、UpdateRecord メソッドの解説セクションを 2 つの段落に分割します。

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<param> (Visual Basic)

パラメータの名前と説明を定義します。

```
<param name="name">description</param>
```

パラメータ

name

メソッド パラメータの名前。名前は、二重引用符 (" ") で囲みます。

description

パラメータの説明。

解説

<param> タグは、メソッド宣言のコメント内で使用して、メソッドのパラメータの 1 つを説明します。

<param> タグのテキストは、IntelliSense の機能 および オブジェクト ブラウザ に表示されます。

コンパイル時に /doc を指定すると、ドキュメント コメントをファイルに出力できます。

使用例

<param> タグを使って id パラメータの説明を記述する例を次に示します。

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

参照

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<paramref> (Visual Basic)

WORD をパラメータとしてフォーマットします。

```
<paramref name="name"/>
```

パラメータ
name

参照するパラメータの名前。名前は、二重引用符 (" ") で囲みます。

解説

<paramref> タグを使用すると、WORD をパラメータとして指定できます。XML ファイルを処理するときに、このパラメータに対して独立した書式を設定できます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、<paramref> タグを使用して id パラメータを参照します。

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<permission> (Visual Basic)

メンバに要求されるアクセス許可について指定します。

```
<permission cref="member">description</permission>
```

パラメータ

member

現在のコンパイル環境からの呼び出しに利用できる、メンバまたはフィールドへの参照。コンパイラは、指定されたコード要素が存在するかどうかを確認し、*member* を出力先 XML 内で標準要素名に変換します。*member* は二重引用符 (" ") で囲みます。

description

メンバへのアクセスの説明。

解説

<permission> タグを使用すると、メンバのアクセスについて文書化できます。PermissionSet クラスは、アクセスをメンバに指定するために使用します。

コンパイル時に /doc を指定すると、ドキュメント コメントをファイルに出力できます。

使用例

次の例は、<permission> タグを使用して、ReadFile メソッドには FileIOPermission が必要であるという説明を記述しています。

VB

```
''' <permission cref="System.Security.Permissions.FileIOPermission">
''' Needs full access to the specified file.
''' </permission>
Public Sub ReadFile(ByVal filename As String)
    ' Code goes here.
End Sub
```

参照

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<remarks> (Visual Basic)

メンバの開設セクションを指定します。

```
<remarks>description</remarks>
```

パラメータ
description

メンバの説明。

解説

`<remarks>` タグを使用して、型の情報を追加し、<summary> (Visual Basic) で指定された情報を補足します。この情報はオブジェクト ブラウザ に表示されます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、`<remarks>` タグを使用して `UpdateRecord` メソッドの機能について説明します。

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<returns> (Visual Basic)

プロパティまたは関数の戻り値を指定します。

```
<returns>description</returns>
```

パラメータ
description

戻り値の説明。

解説

`<returns>` タグは、メソッド宣言のコメント内で使用され、戻り値について説明します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、`<returns>` タグを使用して `DoesRecordExist` 関数が返す値について説明します。

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<see> (Visual Basic)

他のメンバへのリンクを指定します。

```
<see cref="member"/>
```

パラメータ
member

現在のコンパイル環境からの呼び出しに利用できる、メンバまたはフィールドへの参照。コンパイラは、指定されたコード要素が存在することを確認してから、*member* を出力先 XML 内の要素名に渡します。*member* は、二重引用符 (" ") で囲みます。

解説

テキスト内でリンクを指定するには、<see> タグを使用します。"参照" のセクションに示すテキストには、<seealso> (Visual Basic) タグを使用します。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、<see> タグを UpdateRecord の解説セクションで使用して、DoesRecordExist メソッドを参照します。

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<seealso> (Visual Basic)

「参照」のセクションに示すリンクを指定します。

```
<seealso cref="member"/>
```

パラメータ
member

現在のコンパイル環境からの呼び出しに利用できる、メンバまたはフィールドへの参照。コンパイラは、指定されたコード要素が存在することを確認してから、*member* を出力先 XML 内の要素名に渡します。*member* は、二重引用符 (" ") で囲みます。

解説

`<seealso>` タグを使用すると、「参照」セクションに示すテキストを指定できます。テキスト内でリンクを指定するには、<see> (Visual Basic) タグを使用します。

コンパイル時に /doc を指定すると、ドキュメント コメントをファイルに出力できます。

使用例

次の例は、`DoesRecordExist` の remarks セクションで `<seealso>` タグを使用して、`UpdateRecord` メソッドを参照します。

VB

```
''' <param name="id">The ID of the record to update.</param>
''' <remarks>Updates the record <paramref name="id"/>.
''' <para>Use <see cref="DoesRecordExist"/> to verify that
''' the record exists before calling this method.</para>
''' </remarks>
Public Sub UpdateRecord(ByVal id As Integer)
    ' Code goes here.
End Sub
''' <param name="id">The ID of the record to check.</param>
''' <returns><c>True</c> if <paramref name="id"/> exists,
''' <c>False</c> otherwise.</returns>
''' <remarks><seealso cref="UpdateRecord"/></remarks>
Public Function DoesRecordExist(ByVal id As Integer) As Boolean
    ' Code goes here.
End Function
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<summary> (Visual Basic)

メンバの概要を指定します。

```
<summary>description</summary>
```

パラメータ
description

オブジェクトの要約。

解説

<summary> タグは、型または型のメンバの説明に使用します。型の説明に補足情報を追加するには、<remarks> (Visual Basic) タグを使用します。

<summary> タグのテキストは、IntelliSense では型についての唯一の情報源であり、オブジェクト ブラウザ にも表示されます。

コンパイル時に /doc を指定してドキュメント コメントをファイルに出力します。

使用例

この例では、<summary> タグを使用して ResetCounter メソッドと Counter プロパティを記述します。

VB

```
''' <summary>
''' Resets the value the <c>Counter</c> field.
''' </summary>
Public Sub ResetCounter()
    counterValue = 0
End Sub
Private counterValue As Integer = 0
''' <summary>
''' Returns the number of times Counter was called.
''' </summary>
''' <value>Number of times Counter was called.</value>
Public ReadOnly Property Counter() As Integer
    Get
        counterValue += 1
        Return counterValue
    End Get
End Property
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<typeparam> (Visual Basic)

型パラメータの名前と説明を定義します。

```
<typeparam name="name">description</typeparam>
```

パラメータ

name

型パラメータの名前です。名前は、二重引用符 (" ") で囲みます。

description

型パラメータの説明です。

解説

ジェネリック型またはジェネリック メンバ宣言のコメント内で <typeparam> タグを使用すると、型パラメータの 1 つについての説明を記述できます。

コンパイル時に /doc を指定すると、ドキュメント コメントをファイルに出力できます。

使用例

<typeparam> タグを使って id パラメータの説明を記述する例を次に示します。

VB

```
''' <typeparam name="T">
''' The base item type. Must implement IComparable.
''' </typeparam>
Public Class itemManager(Of T As IComparable)
    ' Insert code that defines class members.
End Class
```

参照

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

<value> (Visual Basic)

プロパティの説明を指定します。

```
<value>property-description</value>
```

パラメータ
property-description

プロパティの説明。

解説

プロパティの説明を記述するには、`<value>` タグを使用します。Visual Studio 開発環境のコード ウィザードを使って追加したプロパティには、<summary> (Visual Basic) タグが追加されます。そのプロパティが表す値の説明を記述するには、`<value>` タグを手動で追加する必要があります。

コンパイル時に /doc を指定すると、ドキュメント コメントをファイルに出力できます。

使用例

`<value>` タグを使用して、`Counter` プロパティに格納される値の説明を記述する例は次のようになります。

VB

```
''' <summary>
''' Resets the value the <c>Counter</c> field.
''' </summary>
Public Sub ResetCounter()
    counterValue = 0
End Sub
Private counterValue As Integer = 0
''' <summary>
''' Returns the number of times Counter was called.
''' </summary>
''' <value>Number of times Counter was called.</value>
Public ReadOnly Property Counter() As Integer
    Get
        counterValue += 1
        Return counterValue
    End Get
End Property
```

参照
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

エラー メッセージ (Visual Basic)

アプリケーションが Visual Basic 環境内で動作しているとき、またはスタンドアロンの実行可能コードとして動作しているときに、エラー メッセージが生成される場合があります。一部のエラー メッセージは、デザイン時またはコンパイル時に発生する場合もあります。**On Error** ステートメントと **Err** オブジェクトの **Number** プロパティを使用すると、トラップできるエラーをテストしたりこれに応答したりすることができます。1 ～ 1000 の範囲で使用されていないエラー番号は、Visual Basic で将来使用するために予約されています。

メモ :

エラーが **Err** オブジェクトの **Raise** メソッドまたは **Error** ステートメントを使用して生成される場合は、"アプリケーション定義またはオブジェクト定義のエラー" と記述されています。この場合のエラー番号は、Visual Basic 言語で定義されるエラーには対応していません。このようなエラーは、ホスト アプリケーション (Microsoft Excel や Visual Basic など) によって定義できますが、コードから生成する場合は、**Raise** メソッドを使用し、必要なすべての引数を指定する必要があります。

Visual Basic で開発しているときには、F1 キーを押すことにより、メッセージに関するヘルプを表示できます。同様に、タスク一覧ウィンドウに表示されるエラー メッセージに対しても F1 キーを使用できます。

このセクションの内容

方法 : Visual Basic コンパイル エラーに関する情報を取得する

コンパイラ エラーの概念と、特定のエラー メッセージに関するヘルプの表示方法について説明します。

方法 : Visual Basic ランタイム エラーに関する情報を取得する

ランタイム エラーの概念と、特定のエラー メッセージに関するヘルプの表示方法について説明します。

Visual Basic コンパイラ メッセージ

コンパイル時に発生する Visual Basic のエラー メッセージが記載されています。

Visual Basic のランタイム メッセージ

実行時に発生する Visual Basic のエラー メッセージが記載されています。

関連するセクション

Visual Basic での例外およびエラー処理

Visual Basic の例外処理に関するトピックへのリンクの一覧を示します。

チュートリアル : 構造化例外処理

Try...Catch...Finally ステートメントを使用して例外を処理する方法について説明します。

方法：Visual Basic コンパイル エラーに関する情報を取得する

Visual Basic のコンパイラ エラーは、コンパイラがコード内に問題を検出したときに発生します。コード エディタでは、エラーの原因となったコードの下に波線が表示され、そのコードの上でマウスを停止させると、エラー メッセージが表示されます。メッセージは、[タスク一覧] にも表示されます。

ヘルプの表示

[スマート コンパイル自動修正] 機能は、一部のエラーの修正方法を提示します。この方法を利用してエラーを修正できます。エラーが起きたとき、波線の右側の下に四角形の記号が表示される場合は、マウスを波線の上に置くか、[タスク一覧] に表示されたエラー メッセージをダブルクリックすると、記号がスマート タグ パネルに変わります。スマート タグ パネルをクリックするか、マウス ポインタを置くと [エラー修正のオプション] ヘルパー ウィンドウが開き、エラーに関する説明とエラーの対処方法のセットが表示されます。詳細については、「方法 : 自動修正でコンパイラ エラーを修正する」を参照してください。

Visual Basic コンパイラにはランタイム エラーを起こす可能性のあるコードに関する警告のセットが含まれています。この情報を使用して、バグの少ない適切かつ高速なコードを作成できます。たとえば、ユーザーが、割り当てが行われていないオブジェクト変数のメンバを呼び出そうとしたり、戻り値を設定せずに関数から戻ろうとしたり、または例外をキャッチするロジックにエラーがある **Try** ブロックを実行しようとしたりすると、コンパイラは警告を表示します。警告の表示と非表示など、警告の詳細については、「Visual Basic での警告の構成」を参照してください。

特定のエラー メッセージに関するヘルプを表示するには

1. [タスク一覧] でエラー メッセージを強調表示し、F1 キーを押します。

   または

2. [ヘルプ] メニューの [検索] をクリックします。

3. [検索] ウィンドウの [検索対象] ボックスに、エラー メッセージを入力します。

メモ：
表示されるエラー メッセージには、コードからの識別子 (変数名やプロジェクト名など) がその要素の型名と共に含まれる場合があります。これらの識別子は、基になるエラー メッセージの一部ではありません。検索には、これらの語句は含めないでください。

参照

処理手順

方法 : Visual Basic ランタイム エラーに関する情報を取得する
方法 : エディタでコードをデバッグする
方法 : 自動修正でコンパイラ エラーを修正する

概念

Visual Basic での警告の構成

方法：Visual Basic ランタイム エラーに関する情報を取得する

Visual Basic のランタイム エラーは、アプリケーションがシステムで処理できない動作を実行しようとした場合に発生します。

Visual Basic がスローするエラーは **Exception** オブジェクトです。Visual Basic では **Throw** ステートメントを使用して、**Exception** オブジェクトを含む任意のデータ型のカスタム エラーを生成できます。キャッチされた例外の番号とメッセージをプログラムで表示して、エラーを識別できます。エラーがキャッチされない場合、プログラムは終了します。

ランタイム エラーは、コードでトラップして調べることができます。エラーを生成するコードを **Try** ブロックで囲むことによって、スローされたエラーを対応する **Catch** ブロックでキャッチできます。

ヘルプの表示

特定のエラー メッセージに関するヘルプを表示するには

1. [ヘルプ] メニューの [検索] をクリックします。
2. [検索] ウィンドウの [検索対象] ボックスに、エラー メッセージを入力します。

メモ：
一部のエラー メッセージでは、コードにある識別子 (プロジェクト名や変数名など) を参照する、エラー メッセージの一部ではない文字列が、単一引用符で囲まれています。検索には、引用符で囲まれた語句は含めないでください。

参照

処理手順

方法 : Visual Basic コンパイル エラーに関する情報を取得する

方法 : 例外処理アシスタント を使用してランタイム エラーを修正する

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

Visual Basic でトラップできるエラー

アプリケーション開発においては、Visual Basic によって提供される構造化例外処理モデルが便利です。一方、特にレガシ アプリケーションをアップグレードする場合は、構造化されていないエラー処理の使用を選択できます。On Error ステートメント (Visual Basic) は、特定のエラーのトラップおよび応答に使用できます。

トラップできるエラー

トラップできるエラーはアプリケーションの実行中に発生します。開発中またはコンパイル時にトラップできるエラーが発生することもあります。**On Error** ステートメントと Err オブジェクト (Visual Basic) を使用すると、トラップできるエラーをテストしたりこれに応答したりできます。

次の表に、エラー コードとそれに関連付けられているメッセージの一覧を示します。

コード	メッセージ
0	エラーなし
3	Return に対応する **GoSub** がありません。(使用されなくなりましたが、互換性のために残されています)
5	プロシージャ呼び出しまたは引数が有効ではありません。(Visual Basic)
6	オーバーフローしました。(Visual Basic エラー) オーバーフローしました。(Visual Basic ランタイム エラー)
7	メモリが不足しています。(Visual Basic コンパイラ エラー) メモリが不足しています。(Visual Basic ランタイム エラー)
9	インデックスが有効範囲にありません。(Visual Basic)
10	この配列は固定か、または一時的にロックされています。(Visual Basic)
11	0 で除算しました。(Visual Basic ランタイム エラー)
13	型が一致しません。(Visual Basic)
14	文字列スペースが不足しています。(Visual Basic)
16	式が複雑すぎます。
17	要求された操作を実行できません。(Visual Basic)
18	ユーザーによる割り込みが発生しました。
20	エラーが発生していないときに Resume を実行することはできません。
28	スタック領域が不足しています。(Visual Basic)
35	Sub または Function が定義されていません。(Visual Basic)
47	DLL のクライアント アプリケーションの数が多すぎます。
48	DLL 読み込み時のエラーです。(Visual Basic)
49	DLL を正しく呼び出せません。

51	内部エラーです。(Visual Basic)
52	ファイル名または番号が正しくありません。
53	ファイルが見つかりません。(Visual Basic ランタイム エラー)
54	ファイル モードが正しくありません。
55	ファイルは既に開かれています。
57	デバイス I/O エラーです。
58	ファイルは既に存在します。
59	レコード長が正しくありません。
61	ディスクがいっぱいです。(Visual Basic)
62	ファイルにこれ以上データがありません。
63	レコード番号が正しくありません。
67	ファイルが多すぎます。
68	デバイスが準備されていません。
70	アクセス許可は拒否されました。(Visual Basic)
71	ディスクが準備されていません。
74	ディスク名は変更できません。
75	パス/ファイル アクセス エラー
76	パスが見つかりません。
91	オブジェクト変数または With ブロック変数が設定されていません。
92	For ループが初期化されていません。(Visual Basic)
93	無効なパターン文字列です。
94	無効な **Null** の使用。(使用されなくなりましたが、互換性のために残されています)
95	アプリケーション定義またはオブジェクト定義のエラーです。
96	サポートされているイベント受信最大数のイベントが既に発生しているので、オブジェクトのイベントを受け取ることができません。
97	オブジェクトが定義クラスのインスタンスではない場合、このオブジェクトに関するフレンド関数は呼び出せません。
98	プロパティまたはメソッドの呼び出しには、引数または戻り値としてプライベート オブジェクトへの参照を含めることはできません。
100	クラス '<classname>' は System.Collections.ICollection インターフェイスを実装しません。

298	システム リソースまたは DLL を読み込むことがきませんでした。
320	指定されたファイル名に含まれる、キャラクタ デバイス名を使用できません。
321	ファイル形式が有効ではありません。
322	必要な一時ファイルを作成できません。
325	形式がリソース ファイルで有効ではありません。
327	指定されたデータ値が見つかりません。
328	無効なパラメータです。配列を書き込むことができません。
335	システム レジストリにアクセスできませんでした。
336	ActiveX コンポーネントが正しく登録されていません。
337	ActiveX コンポーネントが見つかりません。
338	ActiveX コンポーネントが正しく実行されませんでした。
360	オブジェクトは既に読み込まれています。
361	このオブジェクトをロードまたはアンロードできません。
363	指定された ActiveX コントロールが見つかりません。
364	オブジェクトがアンロードされました。
365	このコンテキスト内ではアンロードできません。
368	指定されたファイルは期限を過ぎています。
371	指定されたオブジェクトは、**Show** のオーナー フォームとして使用できません。
380	プロパティの値が無効です。
381	プロパティの配列インデックスが有効ではありません。
382	Set は実行時にはサポートされません。
383	Set はサポートされません。読み取り専用のプロパティです。
385	プロパティ配列のインデックスが必要です。
387	Set は使用できません。
393	Get は実行時にはサポートされません。
394	Get はサポートされていません。書き込み専用のプロパティです。
400	フォームは既に表示されています。モーダルで表示できません。

402	一番上のモーダル フォームを最初に閉じるように、コードを作成する必要があります。
422	プロパティが見つかりません。
423	プロパティまたはメソッドが見つかりません。
424	オブジェクトが必要です。(Visual Basic)
429	ActiveX コンポーネントを作成できません。
430	クラスがオートメーションをサポートしていないか、必要なインターフェイスをサポートしていません。
432	オートメーションの操作中にファイル名またはクラス名を見つけられませんでした。(Visual Basic)
438	オブジェクトでサポートされていないプロパティまたはメソッドです。(Visual Basic)
440	オートメーション エラーです。
442	リモート プロセス用のタイプ ライブラリまたはオブジェクト ライブラリへの参照は失われました。
443	オートメーション オブジェクトには既定値がありません。
445	このオブジェクトではサポートされていない操作です。(Visual Basic)
446	オブジェクトは名前付き引数をサポートしていません。
447	オブジェクトは現在のロケールの設定をサポートしていません。(Visual Basic)
448	名前付き引数が見つかりませんでした (Visual Basic)
449	引数は省略できません。(Visual Basic)
450	引数の数が一致していないか、またはプロパティの割り当てが有効ではありません。
451	Property Let プロシージャが定義されておらず、Property Get プロシージャからオブジェクトが返されませんでした。
452	序数が有効ではありません。
453	関数は指定された DLL には定義されていません。
454	コード リソースが見つかりません。
455	コード リソースのロック エラーです。
457	このキーは既にこのコレクションの要素に割り当てられています。
458	Visual Basic でサポートされていないオートメーションが変数で使用されています。
459	オブジェクトまたはクラスがこのイベント セットをサポートしていません。
460	クリップボードの形式が有効ではありません。
461	メソッドまたはデータ メンバが見つかりません。

462	リモート サーバー コンピュータが存在しないか、利用できません。(Visual Basic)
463	クラスがローカル コンピュータに登録されていません。
481	ピクチャが有効ではありません。
482	プリンタ エラー
735	ファイルを TEMP に保存できません。
744	検索文字列が見つかりませんでした。
746	置換後の文字列が長すぎます。
999	Stop ステートメントで中断しました。
32768	機能がまだ実装されていません。

メッセージ "アプリケーション定義またはオブジェクト定義のエラーです。" は、Visual Basic が認識しないエラー コードに適用されます。

参照

処理手順

例外処理のトラブルシューティング

関連項目

On Error ステートメント (Visual Basic)

Err オブジェクト (Visual Basic)

概念

例外処理の概要

非構造化例外処理の概要

構造化例外処理と非構造化例外処理に適した状況

その他の技術情報

Visual Basic での構造化例外処理

この Visual Basic エラーにヘルプはありません。

ほとんどのエラー メッセージには、ヘルプ ページが関連付けられています。しかし、独自のプロジェクトまたはコードでエラーが生成されたために、それに対応するヘルプ ページが存在しません。

このエラーを解決するには

1. オンライン ヘルプを有効にしていない場合は、それを有効にして、このエラーに関連するドキュメントがないか確認します。ヘルプ ソース : ローカル ヘルプおよびオンライン ヘルプ を参照してください。
2. このエラーに関連するドキュメントを、Microsoft Support Knowledge Base (KB) (http://support.microsoft.com/search/) で検索します。
3. このエラーに関連するドキュメントがないか、MSDN オンライン ライブラリへようこそ (http://msdn.microsoft.com/library/ja) を調べます。
4. ドキュメントが見つからない場合は、状況に関する情報を集め、マイクロソフト製品サポート サービスまでご連絡ください。

   有用な情報の 1 つがエラー ID です。これは、BC30648 のように文字 BC とそれに続く 5 桁のコードで構成されます。エラー ID は [出力] ウィンドウに表示されます。[出力] ウィンドウを表示するには、[表示] メニューの [その他のウィンドウ] を選択し、[出力] をクリックします。代わりに、Ctrl キーと Alt キーを押しながら O キーを押すこともできます。次の点に注意してください。

   - [出力] ウィンドウでエラーを生成するには、コンパイラを実行する必要があります。このためには、F5 キーを押すか、[デバッグ] メニューの [デバッグ開始] をクリックします。
   - [出力] ウィンドウ内の "出力元の表示: " で [ビルド] を選択します。

参照

処理手順

リリース ノートの格納場所

関連項目

[オンライン] ([オプション] ダイアログ ボックス - [環境] - [ヘルプ])

概念

ヘルプ ソース : ローカル ヘルプおよびオンライン ヘルプ

テクニカル サポート オプション (Visual Studio)

その他の技術情報

Visual Studio のヘルプの使い方

製品のサポートとユーザー補助

Visual Basic コンパイラ メッセージ

このセクションでは、コンパイル時に表示される Visual Basic エラー メッセージを示します。

参照

処理手順

方法 : Visual Basic コンパイル エラーに関する情報を取得する

その他の技術情報

エラー メッセージ (Visual Basic)

'>' が必要です。

指定したコードの構文が正しくありません。このエラーは多くの場合、属性の右山かっこが抜けていることを意味します。

Error ID: BC30636

このエラーを解決するには

- 使用しているコードの構文を確認します。

参照

概念

Visual Basic の宣言ステートメント

その他の技術情報

Visual Basic における属性

'=' が必要です (Let または Set 代入ステートメント)

Let 代入ステートメントまたは **Set** 代入ステートメントに等号 (=) がありません。

Let 代入ステートメントおよび **Set** 代入ステートメントは Visual Basic 2005 ではサポートされていません。代入するデータの型は、値を受け取るプログラミング要素のデータ型からコンパイラによって決定できます。**Let** ステートメントは値型の代入に、**Set** ステートメントは参照型の代入に置換されます。

新規の Set ステートメント (Visual Basic) は、プロパティを定義するために使用されます。**Set** 代入ステートメントとは無関係です。

Error ID: BC32020

このエラーを解決するには

- 等号 (=) と、変数やプロパティに代入する値とを含む、標準の代入ステートメントを使用します。

参照

概念

プログラミング要素のサポートに関する変更の概要
値型と参照型
代入ステートメント

'=' が必要です (宣言)

Mid ステートメントや **#Const** コンパイル ディレクティブで値が代入されていません。

Error ID: BC30249

このエラーを解決するには

- 等号 (=) に加えて、文字列やコンパイル定数に代入する値を追加します。

参照

関連項目

Mid ステートメント
#Const ディレクティブ

概念

Visual Basic の宣言ステートメント
代入ステートメント

'<variablename>' はローカル変数またはパラメータでないため、'Catch' 変数として使うことはできません。

Try...Catch...Finally ステートメント内の変数は、ローカルで宣言してあるか、現在のプロシージャのパラメータである必要があります。

Error ID: BC31082

このエラーを解決するには

- **Try...Catch...Finally** ステートメント用のローカル変数またはパラメータを宣言します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

'<typename1>' を '<typename2>'に変換できません。

2 つの互換性のない型どうしで型変換を行おうとしました。

Error ID: BC30741

このエラーを解決するには

- キャストされている型を確認します。

参照

関連項目

CType 関数

'<typename>' 値を 'Char' に変換できません。

エラー メッセージ
'<typename>' 値を Char に変換できません。Microsoft.VisualBasic.ChrW を使用して数値を Unicode 文字として解釈するか、最初に数値を String に変換してから数字を生成します。

String および **Object** 以外のデータ型を **Char** に変換しようとしてる式があります。

Error ID: BC32007

このエラーを解決するには

- **ChrW** 関数を使用して数値を Unicode 文字に変換するか、最初に値を **String** に変換してから **Char** に変換します。

参照

関連項目
Chr 関数、ChrW 関数
文字型 (Char) (Visual Basic)
概念
暗黙の型変換と明示的な型変換

'<typename>' には、'MustOverride' と宣言されているメソッドが含まれているため、'MustInherit' と宣言しなければなりません。

MustOverride メンバを含む型は、**MustInherit** として宣言する必要があります。

Error ID: BC31411

このエラーを解決するには

- 型に **MustInherit** 修飾子を追加します。

参照

関連項目

MustInherit
MustOverride

'<typename>' は、サポートされていない型です。

指定した型は Visual Studio で表現できません。

Error ID: BC30649

このエラーを解決するには

- サポートされている型を使用します。
- 新しい型を作成します。

参照

関連項目

データ型の概要 (Visual Basic)

'<typename>' はインターフェイスであるため、式として使用できません。

式が必要な場所に型名があります。式は、変数、定数、リテラル、プロパティ、および **Function** プロシージャ呼び出しを組み合わせて構成する必要があります。

Error ID: BC30111

このエラーを解決するには

- 型名を削除し、有効な要素を使用して式を作成します。

参照

概念

Visual Basic の演算子および式

'<typename>' は列挙型であるため、式として使用することはできません。

式が必要な場所に型名があります。式は、変数、定数、リテラル、プロパティ、および **Function** プロシージャ呼び出しを組み合わせて構成する必要があります。

Error ID: BC30107

このエラーを解決するには

- 型名を削除し、有効な要素を使用して式を作成します。

参照

概念

Visual Basic の演算子および式

'<typename>' は '<containername>' では型です。式として使用することはできません。

使用中の名前が型の名前として既に宣言されているため、式では使用できません。

Error ID: BC30691

このエラーを解決するには

- メンバに一意の名前を付けます。
- メンバの名前を角かっこで囲みます。

参照

概念

コード内の要素名としてのキーワード
Visual Basic の名前付け規則

'<typename>' は型であるため、式として使用することはできません。

式が必要な場所に型名があります。式は、変数、定数、リテラル、プロパティ、および **Function** プロシージャ呼び出しを適宜組み合わせて構成する必要があります。

Error ID: BC30108

このエラーを解決するには

- 型名を削除し、有効な要素を使用して式を作成します。

参照

概念

Visual Basic の演算子および式

'<typename>' は構造体型であるため、式として使用することはできません。

式が必要な場所に型名があります。式は、変数、定数、リテラル、プロパティ、および **Function** プロシージャ呼び出しを適宜組み合わせて構成する必要があります。

Error ID: BC30110

このエラーを解決するには

- 型名を削除し、有効な要素を使用して式を作成します。

参照

概念

Visual Basic の演算子および式

''<typename>' は名前空間であり、式として使用することはできません。

式が必要な場所に型名があります。式は、変数、定数、リテラル、プロパティ、および **Function** プロシージャ呼び出しで構成する必要があります。

Error ID: BC30112

このエラーを解決するには

- 型名を削除し、有効な要素を使用して式を作成します。

参照

概念

Visual Basic の演算子および式

'<typename>' はデリゲート型です。

エラー メッセージ
'<typename>' はデリゲート型です。デリゲート構造では、引数リストとして単一の AddressOf 式しか使用できません。AddressOf 式は、通常、デリゲート構築の代わりに使用できます。

デリゲート クラスのインスタンスを作成する **New** 句で、デリゲート コンストラクタへの無効な引数リストが指定されています。

デリゲート インスタンスを新規作成するときは、単一の **AddressOf** 式しか指定できません。

このエラーは、デリゲート コンストラクタへ引数を 1 つも渡していない場合、複数の引数を渡している場合、または有効な **AddressOf** 式ではない 1 つの引数を渡している場合に発生する可能性があります。

Error ID: BC32008

このエラーを解決するには

- **New** 句で、デリゲート クラスの引数リストに単一の **AddressOf** 式を使用します。

参照

処理手順
方法 : デリゲート メソッドを呼び出す
関連項目
New (Visual Basic)
AddressOf 演算子
概念
デリゲートと AddressOf 演算子

'<typename>' はクラス型であるため、式として使用できません。

式が必要な場所に型名があります。式は、変数、定数、リテラル、プロパティ、および **Function** プロシージャ呼び出しを組み合わせて構成する必要があります。

Error ID: BC30109

このエラーを解決するには

- 型名を削除し、有効な要素を使用して式を作成します。

参照

概念

Visual Basic の演算子および式

'<typename>' は 'My' グループ内で公開されている別の型と同じ名前です。

エラー メッセージ
'<typename>' は 'My' グループ内で公開されている別の型と同じ名前です。フォームかまたはそれを囲む名前空間の名前を変更してください。

クラスまたは構造体が、いずれかの **My** オブジェクト内のクラスまたは構造体と同じ名前で宣言されています。

My.Forms など、**My** オブジェクトを介してアクセスできる 2 つのクラス間では、名前の衝突を回避できない場合があります。

My オブジェクト内のクラス間で名前が衝突する可能性がある場合、コンパイラによって、型のプロパティ名が *ClassName* から *RootNamespace_Namespace_ClassName* へと変更されます。たとえば、`Form1` という名前の 2 つのフォームがあるとします。一方のフォームがルート名前空間 `WindowsApplication1` の `Namespace1` という名前空間に存在する場合、このフォームには `My.Forms.WindowsApplication1_Namespace1_Form1` でアクセスできます。

このエラーは、2 つのクラスが同じ名前を持ち、それぞれの属する名前空間がアンダースコアで入れ子にされている場合に発生します。これらのクラスのプロパティ名がコンパイラによって新たに構築されますが、それでも名前の衝突を完全に防ぐことはできません。

Error ID: BC36015

このエラーを解決するには

1. 新規フォームの名前を変更します。
2. 名前空間の名前を変更します。

   クラスまたは構造体に既存のものと同じ名前を付けることは避けてください。

参照

関連項目
My.Forms オブジェクト
Form
MyGroupCollectionAttribute
概念
同じ名前を持つ複数の変数がある場合に参照を解決する

<type> '<typename>' のプロパティ '<propertyname>' に対して暗黙的に宣言された 'MustOverride' メソッドを、'<typename>' で Shadows にすることはできません。

指定したメソッド名が、基本クラス内のプロパティによって暗黙的に生成された **MustOverride** メソッドと競合しています。たとえば、`Prop1` という名前のプロパティを宣言した場合は、コンパイラによって暗黙のプロシージャ `get_Prop1` と `set_Prop1` が生成されます。

Error ID: BC31416

このエラーを解決するには

1. メソッドに、一意の名前を付けます。
2. 基本クラス内のプロパティから **MustOverride** 修飾子を削除します。

参照

関連項目

MustOverride
Shadows

概念

Property プロシージャ

'<typename>' はクラスでないため、属性として使用できません。

クラスではない型を属性として使おうとしました。

Error ID: BC31503

このエラーを解決するには

- **System.Attribute** から派生するクラスとしてカスタム属性を定義します。

参照

その他の技術情報

Visual Basic における属性

'<typename>' は 'MustInherit' と宣言されているため、属性として使用できません。

カスタム属性クラスは **MustInherit** として宣言できません。

Error ID: BC31506

このエラーを解決するには

- カスタム属性から **MustInherit** 修飾子を削除します。

参照

関連項目

AttributeUsageAttribute Class

その他の技術情報

Visual Basic におけるカスタム属性

'<typename>' にオーバーライドされていない 'MustOverride' メソッドが含まれているため、属性として使用することはできません。

MustOverride メソッドのあるクラスは、属性として使用できません。

属性クラスの **MustOverride** メンバを使用できるのは、それらのメンバをオーバーライドする派生クラスからだけです。

Error ID: BC31507

このエラーを解決するには

1. 属性クラスのメンバから **MustOverride** 修飾子を削除します。
2. 派生クラスで **MustOverride** メンバを実装し、そのクラスを属性として使用します。

参照

関連項目

AttributeUsageAttribute Class

その他の技術情報

Visual Basic におけるカスタム属性

'<typename>' は 'System.Attribute' から継承していないため、属性として使用できません。

System.Attribute から派生していないクラスを使おうとしました。

Error ID: BC31504

このエラーを解決するには

- クラス定義コードの先頭行に **Imports** ステートメントを追加することによって、**System.Attribute** から派生するクラスとしてカスタム属性を定義します。

参照

関連項目

AttributeUsageAttribute Class

その他の技術情報

Visual Basic におけるカスタム属性

'<typename>' には 'System.AttributeUsageAttribute' 属性がないため、属性として使用することはできません。

属性の使用法を定義する **System.AttributeUsageAttribute** なしで宣言された属性を使おうとしました。

Error ID: BC31505

このエラーを解決するには

- カスタム属性は、**AttributeUsageAttribute** 属性を適用した **System.Attribute** から派生している必要があります。

参照

関連項目

AttributeUsageAttribute Class

その他の技術情報

Visual Basic におけるカスタム属性

同じ名前を持つ別のメンバが 'Shadows' として宣言されているため、<typename> '<membername>' は、'Shadows' として宣言しなければなりません。

このプロパティまたはプロシージャの名前が、**Shadows** 修飾子が適用されている別のプロパティまたはプロシージャの名前と同じです。

Error ID: BC30695

このエラーを解決するには

- **Shadows** 修飾子を使用します。

参照

関連項目

Shadows

<type1>'<typename>' は、インターフェイス '<interfacename>' に対して '<methodname>' を実装しなければなりません。

クラスまたは構造体はインターフェイスを実装しますが、インターフェイスによって定義されたプロシージャは実装しません。インターフェイスのすべてのメンバを実装する必要があります。

Error ID: BC30149

このエラーを解決するには

1. インターフェイスで定義された名前とシグネチャを使用して、プロシージャを宣言します。少なくとも **End Function** ステートメントまたは **End Sub** ステートメントを記述する必要があります。
2. **Implements** 句を **Function** ステートメントまたは **Sub** ステートメントの末尾に追加します。たとえば、次のようにします。

```
Public Sub DoSomething() Implements IBaseInterface.DoSomething
```

参照

関連項目

Implements ステートメント

概念

Implements キーワードおよび Implements ステートメント

<type1> '<typename>' は、インターフェイス '<interfacename>' に対して '<membername>' を実装しなければなりません。

エラー メッセージ
'<typename>' は、インターフェイス '<interfacename>' に対して '<membername>' を実装しなければなりません。プロパティの実装では、'ReadOnly' 指定子または 'WriteOnly' 指定子が一致している必要があります。

クラスまたは構造体はインターフェイスを実装しますが、インターフェイスによって定義されたプロシージャ、プロパティ、またはイベントは実装しません。インターフェイスのすべてのメンバを実装する必要があります。

Error ID: BC30154

このエラーを解決するには

1. インターフェイスで定義された名前とシグネチャを使用して、メンバを宣言します。少なくとも **End Function** ステートメント、**End Sub** ステートメント、または **End Property** ステートメントを記述する必要があります。
2. **Implements** 句を **Function** ステートメント、**Sub** ステートメント、**Property** ステートメント、または **Event** ステートメントの末尾に追加します。たとえば、次のとおりです。

   ```
   Public Event ItHappened() Implements IBaseInterface.ItHappened
   ```

3. プロパティを実装するときは、**ReadOnly** または **WriteOnly** をインターフェイス定義の場合と同様に使用します。
4. プロパティを実装するときは、必要に応じて **Get** プロシージャおよび **Set** プロシージャを宣言します。

参照

関連項目
Implements ステートメント
概念
Implements キーワードおよび Implements ステートメント

<type2> は 'Overridable' と宣言されていないため '<type1>' でオーバーライドできません。

派生クラス内のメンバが、**Overridable** 修飾子が適用されていない基本クラスのメンバをオーバーライドしています。

Error ID: BC31086

このエラーを解決するには

1. 基本クラス内のオーバーライドされるメンバに **Overridable** 修飾子を追加します。
2. **Shadows** キーワードを使用して、基本クラス内の項目をシャドウします。

参照

関連項目

Overridable
Overrides
Shadows

<type1> '<typename1>' は、<type2> '<typename2>' のイベント '<eventname>' に対して暗黙的に宣言されたメンバと競合しています。

型メンバの名前が、イベントについて暗黙的に作成されたメンバの名前と競合しています。イベントは、複数の暗黙的な変数を暗黙的に作成します。たとえば、宣言 `Event X` は、`XEventHandler`、`XEvent`、`add_X`、および `remove_X` という名前を暗黙的に宣言します。

Error ID: BC31061

このエラーを解決するには

- 明示的に宣言したメンバの名前を変更して、名前の競合を解決します。

参照

概念

Visual Basic の宣言ステートメント

その他の技術情報

Visual Basic におけるイベント

<type1> '<typename>' はベース <type2> の '<type1>' をオーバーライドしないため、'Overrides' として宣言することはできません。

Sub、**Function**、または **Property** ステートメントに、同じ名前の型が基本クラス内にないときに備えて **Overrides** を指定してあります。

Error ID: BC30284

このエラーを解決するには

1. 型名のスペルが正しいかどうかを確認します。
2. 不要な **Overrides** キーワードを削除します。

参照

概念

プロパティとメソッドのオーバーライド

'<name>' で宣言された <type1> '<typename>' および <type2> '<typename>' は、<namespace> '<namespacename>' で競合しています。

クラスやモジュールなどの 2 つの宣言された要素が、同じ名前空間内で同じ名前を持っています。

Error ID: BC30175

このエラーを解決するには

- いずれかの要素の宣言を別の名前空間に移動します。
- いずれかの要素の名前を変更します。

参照

概念

同じ名前を持つ複数の変数がある場合に参照を解決する

<type1> '<typename>' および <type2> '<typename>' は <namespace> '<namespacename>' で競合しています。

クラスやモジュールなどの 2 つの宣言された要素が、同じ名前空間内で同じ名前を持っています。

Error ID: BC30179

このエラーを解決するには

- いずれかの要素の宣言を別の名前空間に移動します。
- いずれかの要素の名前を変更します。

参照

概念

同じ名前を持つ複数の変数がある場合に参照を解決する

<type1> '<propertyname>' は、基本クラスである <type2> '<classname>' のイベント '<eventname>' に対して暗黙的に宣言されたメンバと競合しています。

基本クラス内のイベントから形成される暗黙のメンバと同じ名前のプロパティが宣言されています。たとえば、基本クラスで **Event1** という名前のイベントを宣言した場合は、コンパイラによって暗黙のプロシージャ **add_Event1** と **remove_Event1** が生成されます。これと同じ名前のプロパティがクラスにある場合、このプロパティは基本クラスのメンバをシャドウする必要があります。

このメッセージは警告です。**Shadows** が既定で使用されます。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC40014

このエラーを解決するには

1. 基本クラスのメンバを隠す場合は、**Shadows** キーワードをプロパティ宣言に追加します。
2. 基本クラスのメンバを隠さない場合は、プロパティ名を変更します。

参照

関連項目

Property ステートメント
Event ステートメント
Shadows

概念

Visual Basic におけるシャドウ

<type1> '<membername>' は、ベース <type2> '<classname>' で宣言されたオーバーロード可能なメンバに 'Shadow' を実行します。

エラー メッセージ
<type1> '<membername>' は基本クラスの <type2> '<classname>' で宣言されているオーバーロード可能なメンバをシャドウします。基本メソッドをオーバーロードする場合は、このメソッドを 'Overloads' として宣言する必要があります。

基本クラスで定義されているプロシージャやプロパティと同じ名前の **Function** プロシージャや **Sub** プロシージャ、または **Property** が派生クラスに定義されています。プロシージャやプロパティはオーバーロード可能なメンバであるため、派生クラスは、基本クラスのメンバをオーバーロードすることもシャドウすることもできます。ただし、派生クラスの宣言コードには、Overloads も Shadows も指定されていません。いずれのキーワードも指定されていない場合、コンパイラは **Shadows** が指定されていると見なします。

Overloads か **Shadows** のいずれかを指定して、プログラミングを行うことをお勧めします。これによって、コードが読みやすくなり、簡単に理解できるようになります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、
「Visual Basic での警告の構成」を参照してください。

Error ID: BC40003

このエラーを解決するには

- 基本クラスのメソッドやプロパティをオーバーロードする場合は、宣言に **Overloads** キーワードを指定します。
- 基本クラスのメソッドやプロパティをシャドウする場合は、**Overloads** の代わりに **Shadows** キーワードを指定します。
- 基本クラスのメンバをオーバーロードもシャドウもしない場合は、派生クラスのメンバの名前を変更します。

参照

関連項目
Overloads
Shadows
Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)
Property ステートメント

概念
プロシージャのオーバーロード
Visual Basic におけるシャドウ

<type1> '<membername>' は、ベース <type3> '<classname>' の <type2> '<membername>' と競合しているため 'Shadows' と宣言することはできません。

基本クラスで定義されている要素と同じ名前でプログラミング要素が宣言されています。この場合、このクラスの要素は基本クラスの要素をシャドウする必要があります。

このメッセージは警告です。**Shadows** が既定で使用されます。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40004

このエラーを解決するには

- **Shadows** キーワードを宣言に追加するか、宣言されている要素の名前を変更します。

参照

関連項目

Shadows

概念

Visual Basic におけるシャドウ

<型 > パラメータを 'ParamArray' として宣言することはできません。

デリゲート、イベント、または演算子の定義で ParamArray パラメータを宣言しています。

ParamArray パラメータは、**Declare**、**Function**、**Property**、および **Sub** のパラメータでのみ使用できます。

Error ID: BC33009

このエラーを解決するには

- パラメータ リストから **ParamArray** キーワードを削除します。
- 演算子を定義している場合は、**ParamArray** の機能を一連のオーバーロードで実現できることがあります。
- デリゲートまたはイベントを定義している場合は、アプリケーションのこの部分のロジック全体を記述し直す必要があります。デリゲートまたはイベントのパラメータには、Optional (Visual Basic) パラメータ、**ParamArray** パラメータ、またはオーバーロードされたパラメータを使用できません。

参照

関連項目

Overloads
Operator ステートメント

概念

演算子プロシージャ

<型> を 'Optional' として宣言することはできません。

デリゲート、イベント、または演算子の定義で Optional (Visual Basic) パラメータを宣言しています。

Optional パラメータは、**Declare**、**Function**、**Property**、および **Sub** のパラメータでのみ使用できます。

Error ID: BC33010

このエラーを解決するには

- パラメータ リストから **Optional** キーワードを削除します。
- 演算子を定義している場合は、**Optional** の機能を一連のオーバーロードで実現できることがあります。
- デリゲートまたはイベントを定義する場合は、アプリケーションのこの部分のロジック全体を記述し直す必要があります。デリゲートまたはイベントのパラメータには、**Optional** パラメータ、ParamArray パラメータ、またはオーバーロードされたパラメータを使用できません。

参照

関連項目

Overloads
Operator ステートメント

概念

演算子プロシージャ

'<type>' を 2 回以上継承することはできません。

Inherits ステートメントが重複しています。

Error ID: BC30584

このエラーを解決するには

- 余分な **Inherits** ステートメントを削除します。

参照

関連項目

Inherits ステートメント

<type> '<typename>' は、基本クラスのオーバーライド可能なメソッドをシャドウします。

エラー メッセージ
<type> '<typename>' は、基本クラスのオーバーライド可能なメソッドをシャドウします。基本メソッドをオーバーライドする場合は、このメソッドを Overrides として宣言する必要があります。

基本クラスで定義されているオーバーライド可能なプロシージャまたはプロパティと同じ名前でプログラミング要素が宣言されています。この場合、このクラスの要素は基本クラスの要素をシャドウする必要があります。

既定では、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC40005

このエラーを解決するには

- 基本プロシージャをオーバーライドする場合は、**Overrides** キーワードを宣言に追加します。
- 基本プロシージャをシャドウする場合は、**Shadows** キーワードを宣言に追加します。
- オーバーライドまたはシャドウを行わない場合は、宣言されている要素の名前を変更します。

参照

関連項目
Overrides
Shadows
概念
プロパティとメソッドのオーバーライド
Visual Basic におけるシャドウ

<type> '<methodname>' は、継承階層間で、同じ名前のほかのメソッドと競合しているため、'Shadows' と宣言する必要があります。

2 つ以上のインターフェイスから継承しているインターフェイスで、複数の基本インターフェイスで既に定義されているプロシージャと同じ名前のプロシージャが定義されています。このインターフェイスのプロシージャは、基本インターフェイスのプロシージャのいずれかをシャドウする必要があります。

このメッセージは警告です。**Shadows** が既定で使用されます。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42000

このエラーを解決するには

- 基本インターフェイスのプロシージャを隠す場合は、新規プロシージャの宣言に **Shadows** キーワードを追加します。
- 基本インターフェイスのプロシージャを隠さない場合は、新規プロシージャの名前を変更します。

参照

関連項目

Shadows

概念

Visual Basic におけるシャドウ

'<statementname>' ステートメントには配列が必要です。

Erase ステートメントと **ReDim** ステートメントには、配列データ型が必要です。指定した型は配列ではありません。

Error ID: BC30049

このエラーを解決するには

- データ型を配列型に変更します。

参照

関連項目

Erase ステートメント (Visual Basic)

ReDim ステートメント (Visual Basic)

その他の技術情報

Visual Basic における配列

'<specifier1>' と '<specifier2>' を組み合わせることはできません。

Shadows と **Overrides** などの互換性のない 2 つの指定子が共に使用されています。

Error ID: BC31408

このエラーを解決するには

- いずれかの指定子を削除します。

参照

関連項目

Shadows
Overrides

<specifier1> <type> を <specifier2> <type> から継承できません。ベース <type> のアクセスを展開しています。

Private または **Friend** に指定された基本クラスからパブリック クラスを継承しようとしました。

Error ID: BC30509

このエラーを解決するには

- 基本クラスを **Public** として宣言します。または、継承するクラスを **Private** または **Friend** として宣言します。

参照

その他の技術情報

Visual Basic の継承

'<specifier>' は、インターフェイス プロパティ宣言では有効ではありません。

インターフェイス内の **Property** ステートメントに、**Implements** などの無効なキーワードがあります。インターフェイスはメンバを定義するだけで、実装することはできません。

Error ID: BC30273

このエラーを解決するには

- 宣言ステートメントから無効なキーワードを削除します。
- インターフェイス メンバの実装を、インターフェイスを実装するクラスに移動します。

参照

関連項目

Interface ステートメント (Visual Basic)
Implements ステートメント

'<specifier>' は、インターフェイス メソッド宣言では有効ではありません。

インターフェイス内の **Function** ステートメントまたは **Sub** ステートメントに、**Implements** などの無効なキーワードがあります。インターフェイスはメンバを定義するだけで、実装することはできません。

Error ID: BC30270

このエラーを解決するには

1. 宣言ステートメントから無効なキーワードを削除します。
2. インターフェイス メンバの実装を、インターフェイスを実装するクラスに移動します。

参照

関連項目

Interface ステートメント (Visual Basic)
Implements ステートメント

'<specifier>' は、インターフェイス イベント宣言では有効ではありません。

インターフェイス内の **Event** ステートメントに、**Implements** などの無効なキーワードがあります。インターフェイスはメンバを定義するだけで、実装することはできません。

Error ID: BC30275

このエラーを解決するには

1. 宣言ステートメントから対象のキーワードを削除します。
2. インターフェイス メンバの実装を、インターフェイスを実装するクラスに移動します。

参照

関連項目

Interface ステートメント (Visual Basic)
Implements ステートメント

'<specifier>' は、Enum 宣言では有効ではありません。

Enum 宣言では無効な指定子を使用しました。

Error ID: BC30396

このエラーを解決するには

- 指定子を削除します。

参照

概念

列挙型の概要

'<specifier>' は、WithEvents 宣言では有効ではありません。

Dim ステートメントに **ReadOnly** などのキーワードがありますが、**WithEvents** キーワードとの組み合わせは無効です。

Error ID: BC30234

このエラーを解決するには

- **Dim** ステートメントから無効なキーワードを削除します。

参照

関連項目

Dim ステートメント (Visual Basic)
WithEvents

'<specifier>' は、Structure 宣言では有効ではありません。

Structure 宣言で、**Dim** や **Override** などの指定子を使用しました。この指定は無効です。

Error ID: BC30395

このエラーを解決するには

- 指定子を削除します。

参照

概念

構造体とクラス

'<specifier>' は、メンバ変数宣言では有効ではありません。

Dim ステートメントに無効なキーワードがあります。**Dim** ステートメントで使用できるキーワードは、**Friend**、**Private**、**Protected**、**Public**、**ReadOnly**、**Shadows**、**Shared** および **Static** だけです。

このメッセージは、プロシージャの外部で **Static** 変数を宣言した場合にも表示されます。**Static** キーワードを使用できるのはプロシージャ レベルだけです。

Dim ステートメントに有効なキーワードを含める場合は、**Dim** キーワードを省略することもできます。

Error ID: BC30235

このエラーを解決するには

1. **Dim** ステートメントから無効なキーワードを削除します。
2. プロシージャの外部で **Static** 変数を宣言している場合は、宣言をプロシージャの内部へ移動させるか、**Static** キーワードを削除します。

参照

関連項目

Dim ステートメント (Visual Basic)

'<specifier>' は、定数宣言では有効ではありません。

Const ステートメントに無効なキーワードがあります。**Const** ステートメントで使用できるキーワードは、**Friend**、**Private**、**Protected**、**Public**、および **Shadows** だけです。

Error ID: BC30233

このエラーを解決するには

1. キーワードのスペルが正しいかどうかを確認します。
2. **Const** ステートメントから無効なキーワードを削除します。

参照

関連項目

Const ステートメント (Visual Basic)

'<rsetstmt>' は宣言されていません。

エラー メッセージ
'<rsetstmt>' が宣言されていません。RSet ステートメントはサポートされていません。代わりに Microsoft.VisualBasic.RSet を使用してください。

以前のバージョンの Visual Basic に組み込まれていた関数のいくつかが Microsoft.VisualBasic 名前空間に移動してあります。これは、それらの機能をより幅広くすべてのプログラム言語から利用できるようにするためです。

Error ID: BC30821

このエラーを解決するには

- **Microsoft.VisualBasic** 名前空間にある **RSet** 関数を使用します。

参照

関連項目
RSet 関数
概念
プログラミング要素のサポートに関する変更の概要

'<qualifiedmethod>' は '<modifier>' であるため、このコンテキストではアクセスできません。

このコンテキストではアクセスできないメソッドにアクセスしようとしました。メソッドに修飾子が適用されています。

Error ID: BC30390

このエラーを解決するには

- 修飾子を削除します。

参照

関連項目

Private (Visual Basic)
Protected (Visual Basic)
Friend (Visual Basic)

概念

メソッドに対するアクセス レベルの選択に関する考慮事項

'<propertyname1>' と '<propertyname2>' では、'ReadOnly' であるか、または 'WriteOnly' であるかのみが異なるため、'<propertyname1>' で '<propertyname2>' をオーバーライドできません。

オーバーロードしようとしたプロパティは、元のプロパティと **ReadOnly** または **WriteOnly** の部分しか異なっていません。

Error ID: BC30362

このエラーを解決するには

- **ReadOnly** および **WriteOnly** 以外の部分も異なるプロパティでオーバーロードします。

参照

関連項目

Overrides

概念

プロパティとメソッドのオーバーライド

'<propertyname>' は、<type> '<typename>' にある同じ名前のメンバと競合する '<membername>' を暗黙的に定義します。

型メンバの名前が、プロパティについて暗黙的に作成された別のメンバの名前と競合しています。プロパティは、定義済みのメンバと競合する可能性のある複数の暗黙的な変数を、暗黙的に作成します。

Error ID: BC31060

このエラーを解決するには

- 明示的に宣言したメンバの名前を変更して、名前の競合を解決します。

参照

処理手順

方法 : フィールドおよびプロパティをクラスに追加する

概念

Property プロシージャ

プロパティとプロパティ プロシージャ

'<propertyname>' を、プロパティ 'Let' として COM に公開することはできません

エラー メッセージ
'<propertyname>' を、プロパティ 'Let' として COM に公開することはできません。'Let' ステートメントを使用すると、Visual Basic 6.0 から、オブジェクト以外の値 (数値や文字列など) をこのプロパティに代入できなくなります。

COMClassAttribute 属性ブロックを使用しているクラスで、**Object** データ型の **Public** プロパティが宣言されています。Visual Basic 6.0 のプログラムでは、このプロパティに **Variant** としてアクセスできますが、**Set** ステートメントによってプロパティに代入できるのはオブジェクト参照だけです。**Let** ステートメントを使用して値型を代入することはできません。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC42102

この警告に対処するには

- このクラスを使用する Visual Basic 6.0 ユーザーに、このプロパティで **Let** ステートメントは使用できないことを通知することを検討します。

参照

関連項目
Property ステートメント
Public (Visual Basic)
オブジェクト型 (Object)
ComClassAttribute クラス

概念
既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)
Visual Basic で使用される属性
属性の適用

'<property1>' と '<property2>' では、一方のみが 'Default' に宣言されているため、お互いをオーバーロードすることはできません。

プロパティに **Default** を指定する場合は、その名前でオーバーロードされるすべてのプロパティにも **Default** を指定する必要があります。

Error ID: BC30361

このエラーを解決するには

- オーバーロードされるすべてのプロパティが **Default** として宣言されていることを確認します。

参照

関連項目

Default (Visual Basic)

概念

プロシージャのオーバーロードに関する注意事項

<Property> パラメータは、'ByRef' として宣言できません。

プロパティのパラメータは、**ByVal** 修飾子を使って宣言する必要があります。

Error ID: BC30651

このエラーを解決するには

- プロパティのパラメータを **ByVal** として定義します。

参照

関連項目

ByVal
ByRef

<proceduresignature1> 配列パラメータ型の配列または配列パラメータ型のランクのみが異なる <proceduresignature2> をオーバーロードするため、CLS に準拠していません。

別のプロシージャまたはプロパティをオーバーライドしているプロシージャまたはプロパティが `<CLSCompliant(True)>` でマーク付けされていますが、両者のパラメータ リストには、ジャグ配列の入れ子レベルまたは配列のランクの違いしかありません。

次の 3 つの宣言のうち、2 番目と 3 番目の宣言ではエラーが発生します。

```
Overloads Sub processArray(ByVal arrayParam() As Integer)

Overloads Sub processArray(ByVal arrayParam()() As Integer)

Overloads Sub processArray(ByVal arrayParam(,) As Integer)
```

2 番目の宣言では、最初の 1 次元のパラメータ `arrayParam` が配列の配列に変更されています。3 番目の宣言では、`arrayParam` が 2 次元配列 (ランク 2) に変更されています。Visual Basic ではこれらの変更のいずれか 1 つだけの違いでオーバーロードすることが可能ですが、このようなオーバーロードは 共通言語仕様 (CLS) に準拠しません。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に設定しなかった場合は、非準拠であると見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40035

このエラーを解決するには

- CLS に準拠させる必要がある場合は、ここに示した変更点だけでなく、相互により多くの違いを持つようにオーバーロードを定義します。
- ここで示した変更点だけでオーバーロードを定義する必要がある場合は、**CLSCompliantAttribute** を定義から削除するか、`<CLSCompliant(False)>` でマーク付けします。

参照

関連項目

Overloads

概念

CLS 準拠コードの記述

プロシージャのオーバーロード

'ParamArray' で宣言されたパラメータが異なるため、<procedurename1> が <procedurename2> をオーバーライドできません

派生クラスのプロシージャが、基本クラスにあり名前が同じでパラメータ リストが異なるプロシージャをオーバーライドしています。

継承クラスでプロシージャをオーバーライドするには、オーバーライドしているプロシージャのパラメータ リスト、アクセス レベル、戻り値の型 (あれば) が一致していることが必要です。具体的には、Optional (Visual Basic) や ParamArray の宣言がすべて一致する必要があります。

Error ID: BC30906

このエラーを解決するには

- プロシージャをオーバーライドする場合は、パラメータリストを基本クラスのプロシージャのパラメータ リストとまったく同じにしてください。基本クラスのプロシージャで、最後のパラメータが **ParamArray** で宣言されている場合は、オーバーライドしているプロシージャでも **ParamArray** で宣言する必要があります。
- 基本クラスのプロシージャからパラメータ リストを変更する必要があれば、オーバーライドすることはできません。オーバーロードすることを検討してください。詳細については、「プロシージャのオーバーロード」を参照してください。

参照

関連項目

Overrides

概念

プロパティとメソッドのオーバーライド

'<procedurename2>' は、このコンテキストではアクセスできないため、'<procedurename1>' でオーバーライドすることはできません。

プロシージャまたはプロパティが、オーバーライドによるアクセスが許可されないアクセス レベルを持つプロシージャまたはプロパティをオーバーライドしています。

たとえば、プロシージャがアセンブリ内で **Friend** で宣言されている場合、そのプロシージャにはアセンブリの外部からアクセスできません。同じプロジェクトの別のアセンブリにあるプロシージャが、その **Friend** プロシージャをオーバーロードしようとしても、アクセスしてオーバーライドできません。

Error ID: BC31417

このエラーを解決するには

- オーバーライドする方のプロシージャまたはプロパティを、オーバーライドされる方のプロシージャまたはプロパティと同じアセンブリに移動させます。

  または

- **Overrides** キーワードを削除します。

参照

関連項目

Overrides

概念

Visual Basic でのアクセス レベル

プロパティとメソッドのオーバーライド

CLS に準拠していない型 '<typename>' を含んでいるため、イベント '<eventname>' の '<procedurename>' メソッドを CLS 準拠として設定できません。

カスタム イベントに **AddHandler** プロシージャまたは **RemoveHandler** プロシージャが宣言され、それが `<CLSCompliant(True)>` でマーク付けされていますが、このイベントが定義された型が `<CLSCompliant(False)>` でマーク付けされているか、またはマークが付けられていません。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に設定しなかった場合は、非準拠であると見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40053

このエラーを解決するには

- CLS 準拠にする必要がある場合は、CLS 準拠の型の内部にイベントを定義します。
- イベントを包含型の内部に置いておく必要がある場合は、イベントの定義から **CLSCompliantAttribute** を削除するか、`<CLSCompliant(False)>` でマーク付けします。

参照

処理手順

方法 : ブロックを回避するイベントを宣言する
方法 : メモリの使用量を節約するイベントを宣言する

概念

AddHandler と RemoveHandler
CLS 準拠コードの記述

'<procedurename>' には引数がないため、戻り値の型をインデックス化できません。

Function プロシージャまたは **Sub** プロシージャの呼び出しで複数の引数が指定されていますが、プロシージャは引数を取らず、戻り値の型 (**Function** の場合) は配列型ではありません。

Error ID: BC32016

このエラーを解決するには

- プロシージャ呼び出しから引数を削除します。

参照

関連項目

Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)

概念

Sub プロシージャ
Function プロシージャ

'<procedurename>' には、サポートされていない戻り値か、またはサポートされていないパラメータ型が指定されています。

Visual Studio では表現できない型をパラメータまたは戻り値に指定してプロシージャが宣言されています。

Error ID: BC30657

このエラーを解決するには

- サポートされている型として戻り値の型またはパラメータを定義します。
- プロシージャによって使用されるデータを記述する新しい型を作成します。

参照

関連項目

データ型の概要 (Visual Basic)

型パラメータ制約が異なるため、'<procedurename>' で'<baseprocedurename>' をオーバーライドすることはできません。

ジェネリック プロシージャが、ジェネリックな基本クラスのプロシージャをオーバーライドしようとしましたが、型パラメータに指定された制約リストに違いがありました。

基本クラスのプロシージャをオーバーライドするには、オーバーライドするプロシージャのシグネチャが基本クラスのプロシージャのシグネチャと完全に一致する必要があるだけでなく、プロシージャのアクセス レベルと各パラメータの引き渡し機構も一致する必要があります。

ジェネリックな基本クラスのプロシージャをオーバーライドするには、オーバーライドするプロシージャの型パラメータの数と各パラメータの制約リストも一致する必要があります。

オーバーライドの要件の詳細については、「Overrides」を参照してください。

Error ID: BC32077

このエラーを解決するには

- 基本クラスのプロシージャをオーバーライドする場合は、型パラメータの制約を見直して、基本クラスのプロシージャと完全に一致するようにします。
- 型パラメータの制約を変更できない事情がある場合は、基本クラスのプロシージャはオーバーライドできません。宣言から **Overrides** キーワードを削除します。

参照

概念

Visual Basic におけるジェネリック型

'<procedure1>' と '<procedure2>' では、パラメータが 'ByRef' と 'ByVal' のどちらに設定されているかのみが異なるため、お互いにオーバーロードすることはできません。

エラー メッセージ
'<procedure1>' と '<procedure2>' では、パラメータが 'ByRef' と 'ByVal' のどちらで宣言されているかのみが異なるため、互いをオーバーロードできません。シャドウとみなされます。

2 つのプロシージャ宣言で同じ名前と引数リストが指定されており、唯一の相違点は 1 つ以上の引数で **ByRef** または **ByVal** が指定されていることです。オーバーロードされたバージョンのプロシージャは、引数の数、順序、またはデータ型が互いに異なっている必要があります。

このメッセージは警告です。**Shadows** が既定で使用されます。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42003

このエラーを解決するには

- オーバーロードされたバージョンのプロシージャ セットを作成する場合は、各バージョンの引数の数、順序、またはデータ型を異なるものにします。また、**Overloads** キーワードを各宣言に追加します。
- プロシージャをオーバーロードしない場合は、いずれかの宣言のプロシージャ名を変更します。

参照

概念
プロシージャのオーバーロード

'<parametername>' は、このメソッドの型パラメータとして既に宣言されています。

ジェネリック プロシージャに型パラメータと同じ名前を持つ通常のパラメータ、またはローカル変数が定義されています。

プロシージャのすべてのパラメータは、ジェネリック プロシージャのすべての型パラメータを含めて、他のどのパラメータとも同じではない名前を持つ必要があります。プロシージャ パラメータはローカル変数として使用されるため、プロシージャ内で宣言されたすべてのローカル変数も、どのパラメータおよび型パラメータとも同じでない名前を持つ必要があります。

Error ID: BC32089

このエラーを解決するには

- 通常のパラメータまたはローカル変数の名前を変更します。

参照

関連項目

パラメータの一覧

概念

Visual Basic におけるジェネリック プロシージャ

'<nullconstant>' は宣言されていません。

エラー メッセージ
'<nullconstant>' は宣言されていません。Null 定数はサポートされていません。代わりに System.DBNull を使用してください。

Null キーワードを使用しているステートメントがありますが、このキーワードは Visual Basic 2005 でサポートされていません。

Error ID: BC30822

このエラーを解決するには

1. **Null** の代わりに DBNull を使用します。次にコード例を示します。

```
Sub TestDBNull()
    Dim t As DataTable
    ' Assume the DataGrid is bound to a DataTable.
    t = CType(DataGrid1.DataSource, DataTable)
    Dim r As DataRow
    r = t.Rows(datagrid1.CurrentCell.RowNumber)
    r.BeginEdit
    r(1) = System.DBNull.Value ' Assign DBNull to the record.
    r.EndEdit
    r.AcceptChanges
    If r.IsNull(1) Then
        MsgBox("")
    End If
End Sub
```

2. オブジェクト変数を使用するときには、代入と比較に Nothing (Visual Basic) キーワードを使用します。次にコード例を示します。

```
Sub TestNothing()
    Dim cls As Object
    ' cls is Nothing if it has not been assigned using the New keyword.
    If (cls Is Nothing) Then
        MsgBox("cls is Nothing")
    End If
    cls = Nothing ' Assign Nothing to the class variable cls.
End Sub
```

参照

関連項目
Nothing (Visual Basic)
DBNull
概念
プログラミング要素のサポートに関する変更の概要

'<namespacename>' が不適切です。

あいまいな名前を指定したため、ほかの名前と競合しています。Visual Basic のコンパイラには競合を解決するための規則がないため、あいまいでない名前に変更する必要があります。

Error ID: BC30554

このエラーを解決するには

- 完全限定名を使います。

参照

関連項目

Namespace ステートメント

概念

Visual Basic における名前空間

'<namespace>' は有効な名前ではありません。ルートの名前空間名として使用できません。

名前空間の修飾名は、有効な要素名ではありません。名前は、アルファベット文字、10 進数値、およびアンダースコア (_) だけで構成され、アルファベット文字またはアンダースコアで始まる必要があります。

Error ID: BC30113

このエラーを解決するには

- この名前空間が、入れ子になった名前空間セットの一部である場合は、入れ子の外側の名前に無効な文字が含まれていないかどうかを確認します。

参照

概念

Visual Basic における名前空間
宣言された要素の名前

'<name2>' 名前空間または型からインポートされた '<name1>' があいまいです。

あいまいな名前を指定したため、ほかの名前と競合しています。Visual Basic のコンパイラには競合を解決するための規則がないため、あいまいでない名前に変更する必要があります。

Error ID: BC30561

このエラーを解決するには

1. 名前空間インポートを削除して名前のあいまいさを解決します。
2. 完全限定名を使います。

参照

関連項目

Imports ステートメント
Namespace ステートメント

概念

Visual Basic における名前空間

インポート '<name2>' の '<name1>' は、Namespace、Class、Structure、Enum または Module を参照していません。

Imports ステートメントの対象が **Namespace**、**Class**、**Structure**、**Enum**、**Module** のいずれでもありません。**Imports** ステートメントは、参照されるプロジェクトおよびアセンブリから名前空間の名前をインポートするか、またはステートメントが記述されているモジュールと同じプロジェクトで定義されている名前空間の名前をインポートします。

Error ID: BC30467

このエラーを解決するには

- インポートしようとしている要素を調べて、**Imports** ステートメントを使用できるものであることを確認します。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する

関連項目

Imports ステートメント

概念

参照と Imports ステートメント

'<name1>' は、'<name2>' で定義された同じ名前によって <type> と競合しています。

型の名前と競合する名前が検出されました。

Error ID: BC31073

このエラーを解決するには

1. 一意の名前を宣言してください。
2. 競合しているメンバ名を角かっこで囲みます。

参照

その他の技術情報

プログラム構造とコード規則

このコンストラクトをコンパイルするために必要な '<name>' が見つかりません。

コンパイルに必要な要素が見つかりません。

Error ID: BC30930

このエラーを解決するには

1. 必要なヘルパー要素を提供します。
2. コードが有効なマネージ コードであることを確認します。

参照

概念

マネージ実行プロセス

'<name>' は '<classname>' のメンバではありません。

指定したメンバはこのクラスのメンバではありません。

Error ID: BC30456

このエラーを解決するには

1. メンバの名前を調べて正しいことを確認します。
2. クラスの実際のメンバを使用します。

参照

概念

クラス メソッド

'<name>' はプロジェクト '<projectname1>' で宣言されていますが、プロジェクト '<projectname2>' によって参照されていません。

現在のプロジェクトが参照を持たないプロジェクト内で、プログラミング要素が定義されています。

Error ID: BC32004

このエラーを解決するには

- 現在のプロジェクト内で、指定された要素を含むプロジェクトへの参照を追加します。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

関連項目

/reference (Visual Basic)

概念

同じ名前を持つ複数の変数がある場合に参照を解決する

'<name>' は、名前空間 '<namespacename>' では不適切です。

あいまいな名前を指定したため、ほかの名前と競合しています。Visual Basic のコンパイラには競合を解決するための規則がないため、あいまいでない名前に変更する必要があります。

Error ID: BC30560

このエラーを解決するには

- 完全限定名を使います。

参照

関連項目

Namespace ステートメント

概念

Visual Basic における名前空間

'<name>' は、アプリケーション オブジェクト '<list>' では不適切です。

あいまいな名前を指定したため、ほかの名前と競合しています。Visual Basic のコンパイラには競合を解決するための規則がないため、あいまいでない名前に変更する必要があります。エラー メッセージに付属する一覧では、あいまいな名前のあるアプリケーション オブジェクトが詳細に示されています。

Error ID: BC30563

このエラーを解決するには

- 完全限定名を使います。

参照

関連項目

Namespace ステートメント

'<name>' は、モジュール '<modulename1>' および '<modulename2>' 内の宣言間においてあいまいです。

あいまいな名前を指定したため、ほかの名前と競合しています。Visual Basic のコンパイラには競合を解決するための規則がないため、あいまいでない名前に変更する必要があります。

Error ID: BC30562

このエラーを解決するには

- 完全限定名を使います。

参照

関連項目

Namespace ステートメント

概念

Visual Basic における名前空間

'<name>' は、このメソッドへのパラメータとして既に宣言されています。

指定したパラメータ名は、このメソッドについて既に宣言されています。

Error ID: BC30734

このエラーを解決するには

- パラメータの名前が一意になるように変更します。

参照

概念

プロシージャのパラメータと引数

'<name>' は、この <declarationspace> で '<declaration>' として既に宣言されています。

2 つのプログラミング要素が、同じ宣言空間 (クラス、モジュール、インターフェイス、または構造体) で同じ名前で宣言されています。

Error ID: BC30260

このエラーを解決するには

- いずれかの宣言を別の宣言空間に移動するか、または宣言された要素のいずれかの名前を変更します。

参照

概念

同じ名前を持つ複数の変数がある場合に参照を解決する

'<name>' は、イベントの基になるデリゲート型 '<delegatetype>' で使用される型 '<type1>' を公開できません。<specifier> '<type2>' 経由でプロジェクトの外側に実装しています。

プライベート型がパブリック クラスの外側に公開されています。

Error ID: BC30924

このエラーを解決するには

- この型を **Public** 型と宣言するか、別の型を使用します。

参照

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

'<name>' は、<name> 経由で <specifier> <type2> '<typename>' にある <type1> 型を公開できません。

パブリック クラス **MyClass** の外部でプライベート型を公開しようとしました。

Error ID: BC30508

このエラーを解決するには

- 型を **Public** として宣言します。

参照

関連項目

MyClass

'<name>' は、プロジェクト外部で <specifier1> '<type1>' から <specifier2> '<type2>' までに実装してあるイベントの基底デリゲート型 '<delegatetype>' で使用されている '<type>'型を公開できません。

このコードは、パブリック クラスの外部でプライベート型を公開しています。

Error ID: BC30923

このエラーを解決するには

- この型を **Public** 型と宣言するか、別の型を使用します。

参照

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

'<name>' は、<specifier1> '<type1>' から <specifier2> '<type2>' までに実装してあるイベントの基底デリゲート型 '<delegatetype>' を公開できません。

このコードは、パブリック クラスの外部でプライベート型を公開しています。

Error ID: BC30914

このエラーを解決するには

- **Public** 型で宣言するか、別の型を使用します。

参照

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

'<name>' は、<specifier> '<type>' 経由でプロジェクト外部で実装してあるイベントの基底デリゲート型 '<delegatetype>' を公開できません。

このコードは、パブリック クラスの外部でプライベート型を公開しています。

Error ID: BC30915

このエラーを解決するには

- 型を **Public** として宣言します。

  または

- 他の型を使用します。

参照

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

'<name>' は、フィールドまたはプロパティでないため、属性指定子でパラメータとして名前を指定することはできません。

属性ブロックで、属性の変数以外のメンバの値が設定されています。値を代入できるのは、フィールドやプロパティなどの変数のメンバだけです。

Error ID: BC32010

このエラーを解決するには

1. 属性名とメンバ名のスペルが正しいことを確認します。
2. メンバが変数以外の場合は、変数ブロックから引数を削除します。

参照

概念

属性の適用

'<modulename>' はモジュールであるため、アセンブリとして参照できません。

モジュールをアセンブリとして使おうとしました。

Error ID: BC31076

このエラーを解決するには

- アセンブリへの参照を変更します。

参照

関連項目

Module ステートメント
Assembly

'<modifier>' は、Interface 宣言では有効ではありません。

Interface 宣言では無効な修飾子を指定しました。**Interface** 宣言で宣言される **Sub** ステートメント、**Function** ステートメント、および **Property** ステートメントの有効な修飾子は、**Overloads** キーワードと **Default** キーワードだけです。**Public**、**Private**、**Friend**、**Protected**、**Shared**、**Static**、**Overrides**、**MustOverride**、**Overridable** など、その他の修飾子は使用できません。

Error ID: BC30397

このエラーを解決するには

- 修飾子を削除します。

参照

概念

インターフェイス定義

'<modifier>' は、イベント宣言では有効ではありません。

Event ステートメントに **ReadOnly** などの無効なキーワードがあります。

Error ID: BC30243

このエラーを解決するには

- **Event** ステートメントから無効なキーワードを削除します。

参照

関連項目

Event ステートメント

'<modifier>' は、メソッド宣言では有効ではありません。

Function ステートメントまたは **Sub** ステートメントに無効なキーワードがあります。

Error ID: BC30242

このエラーを解決するには

- **Function** ステートメントまたは **Sub** ステートメントから無効なキーワードを削除します。

参照

関連項目

Function ステートメント (Visual Basic)
Sub ステートメント (Visual Basic)

'<modifier>' は、ローカル変数宣言では有効ではありません。

プロシージャ内の **Dim** ステートメントに **Overloads** などの無効なキーワードがあります。

Error ID: BC30247

このエラーを解決するには

- **Dim** ステートメントから無効なキーワードを削除します。

参照

関連項目

Dim ステートメント (Visual Basic)

'<modifier>' は、ローカル定数宣言では有効ではありません。

プロシージャ内の **Const** ステートメントに **Overloads** などの無効なキーワードがあります。

Error ID: BC30246

このエラーを解決するには

- **Const** ステートメントから無効なキーワードを削除します。

参照

関連項目

Const ステートメント (Visual Basic)

'<modifier>' は、デリゲート宣言では有効ではありません。

デリゲートの宣言に対して **Shared** などの修飾子を使おうとしました。デリゲートは、ほかのオブジェクトのメソッドを呼び出すために使用するオブジェクトであり、オブジェクト メソッドの仕様を渡されるコンストラクタを定義します。デリゲートの宣言に対しては修飾子を指定できません。

Error ID: BC30385

このエラーを解決するには

- 修飾子を削除します。

参照

処理手順

方法 : デリゲート メソッドを呼び出す

関連項目

Delegate ステートメント

'<modifier>' は、Declare では有効ではありません。

Declare ステートメントに **ReadOnly** などの無効なキーワードがあります。

Error ID: BC30244

このエラーを解決するには

- **Declare** ステートメントから無効なキーワードを削除します。

参照

関連項目

Declare ステートメント

メソッド '<methodname>' には、同じシグネチャを持つ複数の定義が含まれています。

Function プロシージャ宣言または **Sub** プロシージャ宣言で、前の宣言と同じプロシージャ名と引数リストを使用しています。元のプロシージャをオーバーロードしようとしたことが原因の 1 つとして考えられます。オーバーロードされたプロシージャは、異なる引数リストを持つ必要があります。

Error ID: BC30269

このエラーを解決するには

- プロシージャ名か引数リストを変更するか、または重複している宣言を削除します。

参照

概念

同じ名前を持つ複数の変数がある場合に参照を解決する
プロシージャのオーバーロードに関する注意事項

'<methodname>' で、'MustOverride' と宣言されたメソッドに Shadow を指定することはできません。

MustOverride 修飾子が適用されている同じ名前を持つプロパティまたはメソッドが、派生元のクラスで宣言されています。

Error ID: BC31404

このエラーを解決するには

1. 派生クラス内のオーバーライドするプロパティまたはメソッドに、**Overrides** 修飾子を追加します。
2. 基本クラス内のプロパティまたはメソッドから **MustOverride** 修飾子を削除します。

参照

関連項目

MustOverride

'<methodname>' で、シャドウされたメソッドをオーバーライドできません。

別のメンバをシャドウするメソッドを別のメソッドがオーバーライドしようとしました。

Error ID: BC31406

このエラーを解決するには

- **Shadows** 修飾子を使用して、シャドウされるメンバをオーバーライドします。

参照

関連項目

Shadows
Overrides

'<methodname>' を 2 回以上実行することはできません。

既に実装されているメソッドを実装しようとしました。

Error ID: BC30583

このエラーを解決するには

1. メソッド名の綴りが間違っていないことを確認します。
2. メソッドが重複している場合は、余分な宣言を削除します。

参照

処理手順

方法 : メソッドを使用してアクションを実行する

'<methodname>' と '<methodname>' では、'ReadOnly' であるか、または 'WriteOnly' であるかのみが異なるため、お互いをオーバーロードすることはできません。

ReadOnly と **WriteOnly** の宣言だけが異なる 2 つのメソッドをオーバーロードしようとしました。引数リスト以外の要素を使ってバージョンを区別することはできません。

Error ID: BC30366

このエラーを解決するには

- **ReadOnly** および **WriteOnly** 以外の部分も異なるメソッドでオーバーロードします。

参照

概念

プロシージャのオーバーロードに関する注意事項

'<method1>' と '<method2>' では、戻り値が異なるため、'<method1>' で '<method2>' をオーバーライドすることはできません。

戻り値の型が異なる別のメソッドで、メソッドをオーバーライドしようとしました。継承されたオーバーライド可能なメソッドは、同じ名前とシグネチャを持つメソッドを宣言することでオーバーライドできます。このとき、宣言には **Overrides** 修飾子を使用します。2 つのメソッドのシグネチャが一致している必要があります。

Error ID: BC30437

このエラーを解決するには

- 2 つのメソッドの戻り値の型を確認し、必要な場合は変更して一致させます。

参照

概念

プロパティとメソッドのオーバーライド

'<method1>' と '<method2>' では、省略可能な引数の既定値のみが異なるため、'<method1>' で '<method2>' をオーバーライドすることはできません。

オーバーロードしようとしたメソッドは、省略可能なパラメータの既定値が元のメソッドと異なっています。つまり、シグネチャが異なっています。継承されたオーバーライド可能なメソッドは、同じ名前とシグネチャを持つメソッドを宣言することでオーバーライドできます。このとき、宣言には **Overrides** 修飾子を使用します。

Error ID: BC30307

このエラーを解決するには

- 2 つのメソッドのシグネチャを同じにします。

参照

概念

プロパティとメソッドのオーバーライド
オーバーライド修飾子

'<method1>' と '<method2>' では、省略可能な引数のみが異なるため、'<method1>' で '<method2>' をオーバーライドすることはできません。

オーバーロードしようとしたメソッドは、省略可能なパラメータの値が元のメソッドと異なっています。つまり、シグネチャが異なっています。継承されたオーバーライド可能なメソッドは、同じ名前とシグネチャを持つメソッドを宣言することでオーバーライドできます。このとき、宣言には **Overrides** 修飾子を使用します。

Error ID: BC30308

このエラーを解決するには

- 2 つのメソッドのシグネチャを同じにします。

参照

関連項目

Overrides

概念

プロパティとメソッドのオーバーライド

設定されている引数が 'ByRef' に対し 'ByVal' という点で異なるため、'<method1>' で '<method2>' を上書きすることはできません。

オーバーロードしようとしたメソッドのパラメータには、**ByVal** ではなく **ByRef** が指定されています。オーバーライドしたメンバは、基本クラスから継承したメンバと同じ引数を持っている必要があります。

Error ID: BC30398

このエラーを解決するには

- パラメータを両方とも **ByRef** または **ByVal** にします。

参照

概念

プロパティとメソッドのオーバーライド

引数の値渡しおよび参照渡し

'<method2>' は 'Declare' ステートメントであるため、'<method1>' で '<method2>' をオーバーライドすることはできません。

Declare ステートメント付きで宣言されている基本クラス名のデリゲートをオーバーライドしようとしました。

Error ID: BC30474

このエラーを解決するには

1. オーバーライドされるメンバの **Declare** ステートメントを削除します。
2. このメソッドをオーバーライドしないようにします。

参照

関連項目

Declare ステートメント

概念

プロパティとメソッドのオーバーライド

'<method1>' はベース メソッドのアクセスを展開するので、'<method2>' をオーバーライドできません。

プロシージャで **Overrides** が指定されていますが、プロシージャで宣言されているアクセシビリティによる制限が、オーバーライドされるメソッドのアクセシビリティの制限よりも緩く設定されています。アクセシビリティを拡張することはできません。つまり、オーバーライドするメソッドを、オーバーライドされるメソッドよりもアクセスしやすくすることはできません。たとえば、基本クラスのメソッドが **Protected** である場合は、この基本クラスのメソッドを **Public** のメソッドでオーバーライドできません。

Error ID: BC32203

このエラーを解決するには

- **Overrides** キーワードを削除します。または、少なくても基本クラスのメソッドと同じ制限にはなるように、アクセシビリティを変更します。

参照

概念

プロパティとメソッドのオーバーライド
Visual Basic でのアクセス レベル
Visual Basic におけるシャドウ

'<method1>' と '<method2>' では、省略可能な引数の既定値のみが異なるため、お互いをオーバーロードすることはできません。

オーバーロードしようとしたメソッドは、省略可能なパラメータしか異なっていません。省略可能なパラメータを持つメソッドは、省略可能なパラメータを持つバージョンと持たないバージョンの 2 つのオーバーロードされたメソッドと同じです。このため、いずれかのバージョンの引数リストと一致する引数リストを使ってメソッドをオーバーロードすることはできません。

Error ID: BC30305

このエラーを解決するには

- 省略可能なパラメータ以外の部分が異なっているメソッドでオーバーロードします。

参照

概念

プロシージャのオーバーロード
プロシージャのオーバーロードに関する注意事項

'<method1>' と '<method2>' では、戻り値の型のみが異なるため、お互いをオーバーロードすることはできません。

オーバーロードしようとしたメソッドは、戻り値の型しか異なっていません。オーバーロードでは、メソッドの 2 つのバージョンを引数リストの違いで区別する必要があります。引数リスト以外の部分でメソッドを区別することはできません。

Error ID: BC30301

このエラーを解決するには

- 引数リストが異なるメソッドでオーバーロードします。

参照

概念

プロシージャのオーバーロード
プロシージャのオーバーロードに関する注意事項

'<method1>' と '<method2>''|2' では、'ParamArray' として宣言されたパラメータのみが異なるため、お互いをオーバーロードすることはできません。

ParamArray パラメータ 1 つ以上だけが異なる 2 つのメソッドをオーバーロードしようとしました。**ParamArray** パラメータを持つプロシージャは、コンパイラによって、パラメータの配列に渡される内容で区別される無限の数のオーバーロードを持つものと見なされます。

Error ID: BC30368

このエラーを解決するには

- **ParamArray** パラメータ以外の部分も異なるメソッドでオーバーロードします。

参照

概念

プロシージャのオーバーロードに関する注意事項
パラメータ配列

'<method1>' と '<method2>' では、パラメータが 'ByRef' と 'ByVal' のどちらに設定されているかのみが異なるため、お互いにオーバーロードすることはできません。

オーバーロードしようとしたメソッドは、**ByRef** または **ByVal** として宣言されているパラメータしか異なっていません。

Error ID: BC30345

このエラーを解決するには

- **ByRef** または **ByVal** のパラメータ名以外の部分も異なるメソッドでオーバーロードします。

参照

概念

プロシージャのオーバーロード

プロシージャのオーバーロードに関する注意事項

'<method1>' と '<method2>' では、省略可能な引数のみが異なるため、お互いをオーバーロードすることはできません。

オーバーロードしようとしたメソッドは、省略可能なパラメータしか異なっていません。省略可能なパラメータを持つメソッドは、省略可能なパラメータを持つバージョンと持たないバージョンの 2 つのオーバーロードされたメソッドと同じです。このため、いずれかのバージョンの引数リストと一致する引数リストを使ってメソッドをオーバーロードすることはできません。

Error ID: BC30300

このエラーを解決するには

- 省略可能なパラメータ以外の部分が異なっているメソッドでオーバーロードします。

参照

概念

プロシージャのオーバーロード
プロシージャのオーバーロードに関する注意事項

'<method>' は '<modifier>' であるため、このコンテキストではアクセスできません。

Private として宣言されているメソッドであるため、このコンテキストではアクセスできないメソッドにアクセスしようとしました。Visual Basic コンパイラでは、クラスのメンバすべてがインポートされ、大文字と小文字が区別されないため、大文字と小文字だけが異なる名前が衝突したために、このエラーが発生した可能性があります。

Error ID: BC30389

このエラーを解決するには

- メソッドを **Public** として宣言することを検討します。
- 名前の衝突がエラーの原因である場合は、衝突している名前を大文字小文字以外で区別してください。

参照

関連項目

Private (Visual Basic)

別の '<methodname2>' が 'Overloads' と宣言されているため、<method> '<methodname1>' は 'Overloads' と宣言しなければなりません。

同じ名前を持つ別のプロパティまたはメソッドが、**Overloads** 修飾子で宣言されています。

Error ID: BC31409

このエラーを解決するには

1. プロパティまたはメソッドに **Overloads** 修飾子を追加します。
2. 同じ名前を持つすべてのプロパティまたはメソッドから **Overloads** 修飾子を削除します。

参照

関連項目

Overloads

<メッセージ> このエラーは、プロジェクト '<projectname1>' の '<filename1>' へのファイル参照とプロジェクト '<projectname2>' の '<filename2>' へのファイル参照との混合によって生じた可能性があります。

エラー メッセージ
<メッセージ> このエラーは、プロジェクト '<projectname1>' の '<filepathname1>' へのファイル参照とプロジェクト '<projectname2>' の '<filepathname2>' へのファイル参照との混合によって生じた可能性があります。両方のアセンブリが同一である場合は、同じ場所から参照するようにこれらの参照を置き換えてください。

プロジェクト内のコードが別のプロジェクトのメンバにアクセスしていますが、このプロジェクトのソリューションは、Visual Basic コンパイラに参照の解決を許可するよう構成されていません。

別のアセンブリで定義されている型にアクセスするには、Visual Basic コンパイラが、そのアセンブリへの参照を保持する必要があります。これは、プロジェクト間の循環参照にならない、単一であいまいさのない参照である必要があります。

Error ID: BC30970

このエラーを解決するには

1. プロジェクトから参照するのに最適なアセンブリを作成しているプロジェクトがどれかを特定します。この判断には、ファイル アクセスの容易さや更新の頻度などの基準を使用できます。
2. プロジェクトのプロパティに、使用する型が定義されているアセンブリを含むファイルへの参照を追加します。

参照
処理手順
方法 : Visual Studio のリファレンスを追加または削除する
方法 : プロジェクト プロパティおよび構成設定を変更する
壊れた参照のトラブルシューティング
概念
プロジェクト参照
同じ名前を持つ複数の変数がある場合に参照を解決する
その他の技術情報
名前空間およびコンポーネントの参照

<メッセージ> このエラーは、ファイル参照と '<assemblyname>' へのプロジェクト参照との混合によって生じた可能性があります。

エラー メッセージ
<メッセージ> このエラーは、ファイル参照と '<assemblyname>' へのプロジェクト参照との混合によって生じた可能性があります。この場合、プロジェクト '<projectname1>' の '<assemblyfilename>' へのファイル参照を '<projectname2>' へのプロジェクト参照で置き換えてください。

プロジェクト内のコードが別のプロジェクトのメンバにアクセスしていますが、このプロジェクトのソリューションは、Visual Basic コンパイラに参照の解決を許可するよう構成されていません。

別のアセンブリで定義されている型にアクセスするには、Visual Basic コンパイラが、そのアセンブリへの参照を保持する必要があります。これは、プロジェクト間の循環参照にならない、単一であいまいさのない参照である必要があります。

Error ID: BC30971

このエラーを解決するには

1. プロジェクトから参照するのに最適なアセンブリを作成しているプロジェクトがどれかを特定します。この判断には、ファイル アクセスの容易さや更新の頻度などの基準を使用できます。
2. プロジェクトのプロパティに、使用する型が定義されているアセンブリを含むプロジェクトへの参照を追加します。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
方法 : プロジェクト プロパティおよび構成設定を変更する
壊れた参照のトラブルシューティング
概念
プロジェクト参照
同じ名前を持つ複数の変数がある場合に参照を解決する
その他の技術情報
名前空間およびコンポーネントの参照

省略可能な引数の型が異なるため、'<membername1>' で '<membername2>' をオーバーライドできません。

オーバーロードされたメンバは、必須パラメータのデータ型が 1 つ以上異なっている必要があります。

Error ID: BC30697

このエラーを解決するには

- 少なくとも 1 つの必須パラメータを別のデータ型として定義します。

参照

関連項目

Overloads

'<membername1>' と '<membername2>' は、省略可能な引数の型においてのみ異なるため、お互いをオーバーロードすることはできません。

オーバーロードされたメンバは、必須パラメータのデータ型が 1 つ以上異なっている必要があります。

Error ID: BC30696

このエラーを解決するには

- 少なくとも 1 つの必須パラメータを別のデータ型として定義します。

参照

関連項目

Overloads

'<membername>' は古い形式です: '<errormessage>'

ステートメントが、ObsoleteAttribute 属性が適用されているクラスまたは構造体のメンバにアクセスしています。また、ディレクティブがそれをエラーとして扱うように設定されています。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

Error ID: BC30668

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. ソース コード参照でメンバ名のスペルが正しいことを確認します。

参照

概念

Visual Basic で使用される属性
属性の適用

'<membername>' は宣言されていないか、またはそれを含むモジュールがデバッグ セッションで読み込まれていません。

指定したメンバが見つかりませんでした。

Error ID: BC30699

このエラーを解決するには

1. メンバが正しく定義されていること、およびスペルが正しいことを確認します。
2. モジュール内で宣言されているメンバのいずれかにアクセスします。場合によっては、メンバが宣言されているモジュールが読み込まれていないために、デバッグ環境でメンバを特定できないことがあります。

参照

その他の技術情報

Visual Studio でのデバッグ

'<membername>' は宣言されていません。

'<*membername*>' は宣言されていません。**Debug** オブジェクト機能は、**System** アセンブリの **System.Diagnostics.Debug** で利用できます。

指定したメンバ名が見つかりませんでした。

Error ID: BC30816

このエラーを解決するには

1. メンバのスペルを確認します。
2. **Imports** ステートメントを使用するか、その他の名前空間で定義されたメンバを完全限定します。たとえば、**WriteLine** 関数は、**System.Diagnostics.Debug** 名前空間で宣言されています。

参照

その他の技術情報

Visual Studio でのデバッグ

'<membername>' は、継承インターフェイス '<interfacename1>' および '<interfacename2>' 間ではあいまいです。

インターフェイスが、同じ名前の 2 つ以上のメンバを複数のインターフェイスから継承しています。

Error ID: BC30685

このエラーを解決するには

- 使用する基本インターフェイスに値をキャストします。次に例を示します。

```
Interface Left
    Sub MySub()
End Interface

Interface Right
    Sub MySub()
End Interface

Interface LeftRight
    Inherits Left, Right
End Interface

Module test
    Sub Main()
        Dim x As LeftRight
        ' x.MySub()  'x is ambiguous.
        CType(x, Left).MySub() ' Cast to base type.
        CType(x, Right).MySub() ' Call the other base type.
    End Sub
End Module
```

参照

概念

インターフェイスの概要

'<membername>' は、この <containername> に対して生成された '<procedurename>' によって既に宣言されています。

共通言語ランタイムによって宣言されているメンバと同じ名前のメンバが宣言されました。

Error ID: BC30733

このエラーを解決するには

- メンバの名前を変更して競合を解決します。

参照

その他の技術情報

Visual Studio でのデバッグ

'<membername>' は、複数の基本インターフェイスに存在します。

エラー メッセージ
'<membername>' は、複数の基本インターフェイスに存在します。派生インターフェイスの名前の代わりに、Implements 句で <'membername'> を宣言しているインターフェイスの名前を使用してください。

このインターフェイスは同じ名前のメンバを複数のインターフェイスから継承するため、あいまいさが発生します。

Error ID: BC31040

このエラーを解決するには

- 派生インターフェイスの名前の代わりに、**Implements** 句内の定義元インターフェイスの名前を使用します。

参照

関連項目
Implements (Visual Basic)
その他の技術情報
Visual Basic におけるインターフェイス

型パラメータ制約が異なるため、'<membername>' は '<interfacename>.<interfacemembername>' を実装することができません。

ジェネリックなイベント、プロパティ、またはプロシージャに、インターフェイスに定義されたメンバと類似するメンバを実装しようとしていますが、型パラメータの制約リストが違っています。

インターフェイスのメンバを実装する場合、実装しているメンバはシグニチャだけでなく、各パラメータの値渡しの方法もそのインターフェイスのメンバと完全に同じであることが必要です。

それに加えて、ジェネリック インターフェイスのメンバを実装する場合は、実装しているメンバの型パラメータの数、および各型パラメータの制約リストも一致している必要があります。

インターフェイスの実装の詳細については、「Implements キーワードおよび Implements ステートメント」を参照してください。

Error ID: BC32078

このエラーを解決するには

- インターフェイスのメンバを実装する必要がある場合は、型パラメータの制約をインターフェイスのメンバのものと完全に一致するように修正します。
- 型パラメータの制約を現状のままにしておく必要がある場合は、この宣言にインターフェイスのメンバを実装できません。Implements (Visual Basic) キーワードを宣言から削除します。

参照

概念

Visual Basic におけるジェネリック型
Visual Basic でのインターフェイス実装例

'<membername>' は、クラス、構造体、またはインターフェイス外部では 'Shadows' としてのみ宣言できます。

Shadows キーワードが、名前空間、モジュール、またはファイル レベルでの宣言に記述されています。シャドウは、基本要素から継承できるプログラミング要素内だけで意味を持ちます。

Error ID: BC32200

このエラーを解決するには

- **Shadows** キーワードを削除するか、クラス、構造体、またはインターフェイス内に宣言を移動します。

参照

関連項目

Shadows

概念

Visual Basic におけるシャドウ

'<member>' は '<eventname>' に対して暗黙的に宣言されているため、ベース <class> '<classname>' の 'MustOverride' メソッドを Shadow にすることはできません。

指定したイベントが、**MustOverride** 修飾子で宣言されたメソッドと同じ名前のメンバを暗黙的に宣言しています。

Error ID: BC31413

このエラーを解決するには

- 基本クラス内のメソッドから **MustOverride** を削除するか、一意の名前をメソッドまたはプロパティに付けます。

参照

関連項目

MustOverride

その他の技術情報

Visual Basic におけるイベント

'<メンバ>' はすでにこの構造体で宣言されています

列挙体のメンバがこの構造体内で 1 回以上宣言されています。

Error ID: BC31421

このエラーを解決するには

- 余分な宣言を削除します。

参照

その他の技術情報

デバッガのロードマップ

'<member>' は、すべての Enum で暗黙的に宣言されている、同じ名前の予約メンバと競合しています。

型メンバの名前が、すべての列挙体で暗黙的に宣言された別のメンバの名前と競合しています。

Visual Basic コンパイラは、宣言されている特定のプログラミング要素に対応する暗黙のメンバを作成します。列挙体は、メンバ `value__member` を暗黙で宣言します。

Error ID: BC31420

このエラーを解決するには

- メンバの名前を変更します。

参照

概念

Visual Basic の宣言ステートメント
列挙型の概要
宣言された要素の名前

'<mathfunction1>' は宣言されていません。

エラー メッセージ
'<mathfunction1>' は宣言されていません。この関数は System.Math クラスに移動され、現在は '<mathfunction2>' という名前になっています。

以前のバージョンの Visual Basic に組み込まれていた関数のいくつかが System.Math 名前空間に移動しています。これは、それらの機能をより幅広くすべてのプログラミング言語から利用できるようにするためです。

Error ID: BC30819

このエラーを解決するには

- **System.Math** で宣言されているメソッドを使用します。

参照

関連項目
Math
概念
プログラミング要素のサポートに関する変更の概要

'<lsetstmt> が宣言されていません。

エラー メッセージ
'<lsetstmt>' が宣言されていません。LSet ステートメントはサポートされていません。代わりに Microsoft.VisualBasic.LSet を使用してください。

以前のバージョンの Visual Basic に組み込まれていた関数のいくつかが Microsoft.VisualBasic 名前空間に移動してあります。これは、それらの機能をより幅広くすべてのプログラム言語から利用できるようにするためです。

Error ID: BC30820

このエラーを解決するには

- **Microsoft.VisualBasic** 名前空間にある **LSet** 関数を使用します。

参照

関連項目
LSet 関数
概念
プログラミング要素のサポートに関する変更の概要

'<keyword>' は、インスタンス メソッド内でのみ有効です。

Me、**MyClass**、**MyBase** の各キーワードは、特定のクラス インスタンスを表します。これらのキーワードは、共有の **Function** プロシージャや **Sub** プロシージャの中では使用できません。

Error ID: BC30043

このエラーを解決するには

- プロシージャからキーワードを削除するか、またはプロシージャ宣言から **Shared** キーワードを削除します。

参照

関連項目

Me
MyClass
MyBase

概念

継承の基本
オブジェクト変数の代入

'<keyword>' は、クラス内でのみ有効です。

クラスに関連する **Me** や **MyClass** などのキーワードが、クラス定義の外部で使用されています。

Error ID: BC32002

このエラーを解決するには

- キーワードを使用しているコードがクラス インスタンスに関連する場合は、コードをクラスの実装に移動します。
- キーワードを使用しているコードがクラスに適用されない場合は、無効なキーワードを削除します。

参照

関連項目

Me
MyClass
Class ステートメント (Visual Basic)

'<keyword>' は、構造体内では有効ではありません。

構造体は値型であり、参照型ではありません。構造体はクラスから作成されたインスタンスではないため、**Me**、**MyClass**、**MyBase** の各キーワードは構造体の中では使用できません。

Error ID: BC30044

このエラーを解決するには

- 構造体をクラスに変更するか、またはプロシージャからキーワードを削除します。

参照

関連項目

Me
MyClass
MyBase

概念

継承の基本

'<keyword>' は、モジュール内で有効ではありません。

クラス インスタンスに関連する **Me** や **MyBase** などのキーワードが、モジュールの中で使用されています。モジュールは継承やインスタンスを持ちません。

Error ID: BC32001

このエラーを解決するには

- キーワードを使用しているコードがクラス インスタンスに関連する場合は、コードをクラスの実装に移動します。
- キーワードを使用しているコードがモジュールに適用される場合は、無効なキーワードを削除します。

参照

関連項目

Me
MyBase

'<propertyname>' の '<keyword>' アクセサは古い形式です: '<errormessage>' (Visual Basic 警告)

ステートメントがプロパティの読み込みまたは書き込みを行っていますが、対応するプロシージャには、それを警告として扱うように ObsoleteAttribute 属性とディレクティブでマーク付けされています。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

既定では、**ObsoleteAttribute** の **IsError** プロパティが **False** であるため、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40019

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. ソース コード参照でプロパティ名のスペルが正しいことを確認します。
3. このメッセージが表示されるような方法でプロパティにアクセス (読み取りまたは書き込み) するのは避けてください。

参照

概念

Visual Basic で使用される属性
属性の適用
Property プロシージャ

'<propertyname>' の '<keyword>' アクセサは古い形式です: '<errormessage>' (Visual Basic エラー)

ステートメントが、プロパティを読み取るか書き込もうとしましたが、そのプロパティに対応するプロシージャには ObsoleteAttribute 属性とそれをエラーとして扱うディレクティブがマークされています。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

Error ID: BC30911

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. ソース コード参照でプロパティ名のスペルが正しいことを確認します。
3. このメッセージが生成される操作方法 (読み取りまたは書き込み) でプロパティにアクセスしないようにします。

参照

概念

Visual Basic で使用される属性
属性の適用
Property プロシージャ

'<propertyname>' の '<keyword>' アクセサは古い形式です (Visual Basic 警告)

プロパティを警告として扱うために、その対応するプロシージャが ObsoleteAttribute の属性およびディレクティブでマークされているプロパティに対して、ステートメントが読み取りまたは書き込みを試みています。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

既定では、**ObsoleteAttribute** の **IsError** プロパティが **False** であるため、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40020

このエラーを解決するには

1. ソース コード参照でプロパティ名のスペルが正しいことを確認します。
2. このメッセージが表示されるような方法でプロパティにアクセス (読み取りまたは書き込み) するのは避けてください。

参照

概念

Visual Basic で使用される属性
属性の適用
Property プロシージャ

'<propertyname>' の '<keyword>' アクセサは古い形式です (Visual Basic エラー)

ステートメントがプロパティを読み取りまたは書き込みしようとしましたが、対応するプロシージャが、ObsoleteAttribute 属性として、またディレクティブによりエラーとして処理するようマークされています。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定します。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

Error ID: BC30912

このエラーを解決するには

1. ソース コード参照でプロパティ名のスペルが正しいことを確認します。
2. このメッセージが生成される原因になった方法 (読み取りまたは書き込み) でプロパティにアクセスすることを避けます。

参照

概念

Visual Basic で使用される属性
属性の適用
Property プロシージャ

'<interfacename1>' で '<methodname>' を実装できません。インターフェイス '<interfacename2>' に、対応する <method> がありません。

1 番目のインターフェイスの派生元のインターフェイスに、実装しようとしたメソッドに対応するメソッドがありません。

Error ID: BC30401

このエラーを解決するには

- メソッドの名前のスペルが正しいことを確認するか、不足しているメソッドを用意します。

参照

関連項目

コンストラクタとデストラクタの使用方法

'<interfacename>.<membername>' は、基本クラス '<baseclassname>' によって既に実装されているため、さらに実装することはできません。<type> の再実装と見なされます。

派生クラスのプロパティ、プロシージャ、またはイベントに、インターフェイス メンバを指定する **Implements** 句が使用されていますが、そのメンバは基本クラスで既に実装されています。

基本クラスで実装されたインターフェイス メンバを、派生クラスで再実装することは可能です。これは基本クラスの実装をオーバーライドすることと同じではありません。詳細については、「Implements (Visual Basic)」を参照してください。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42015

このエラーを解決するには

- インターフェイス メンバを再実装する場合は、特別なアクションは必要ありません。MyBase キーワードを使って基本クラスの実装にアクセスする場合を除き、派生クラスのコードは再実装されたメンバにアクセスします。
- インターフェイス メンバを再実装するつもりでない場合は、プロパティ、プロシージャ、またはイベントの宣言から **Implements** 句を削除します。

参照

概念

Implements キーワードおよび Implements ステートメント

その他の技術情報

Visual Basic におけるインターフェイス

'<typename>' は予約された名前であるため、'<implementsclause>' で '<typename>' を実装できません。

Implements ステートメントで予約名が指定されました。

Error ID: BC31415

このエラーを解決するには

- 実装できなかったインターフェイス メンバの名前を変更します。

参照

関連項目

Implements ステートメント

'<functionname>' は宣言されていません (Visual Basic Error)

エラー メッセージ
'<functionname>' は宣言されていません。Microsoft.VisualBasic 名前空間ではファイル I/O 機能を使用できます。

Error ID: BC30815

このエラーを解決するには

- **My.Computer.FileSystem** で定義されている I/O 関数、**System.IO** 名前空間、Visual Basic ランタイム関数、または **Microsoft.VisualBasic.FileSystem** オブジェクトを使用します。

参照

関連項目
My.Computer.FileSystem オブジェクト
その他の技術情報
Visual Basic におけるファイル アクセス

'<functionname>' は宣言されていません (スマート デバイスおよび Visual Basic コンパイラ エラー)

<functionname> が宣言されていません。通常、ファイル I/O 機能は **Microsoft.VisualBasic** 名前空間で使用できますが、.NET Compact Framework のターゲット バージョンではサポートされていません。

Error ID: BC30766

このエラーを解決するには

- **System.IO** 名前空間で定義された関数を使用してファイル操作を実行します。

参照

関連項目

System.IO

その他の技術情報

Visual Basic におけるファイル アクセス

'<function>' は宣言されていません。

エラー メッセージ
'<function>' は宣言されていません。この関数は Microsoft.VisualBasic 名前空間に移動されました。

Error ID: BC30818

このエラーを解決するには

- **Microsoft.VisualBasic** 名前空間にある適切な関数を使用します。

参照

関連項目

Visual Basic ランタイム ライブラリのメンバ

'<filename>' はアセンブリでないため、参照できません。

.NET アセンブリではないファイルが参照されました。

Error ID: BC31519

このエラーを解決するには

1. アセンブリの名前と場所を確認します。
2. アセンブリが破損していないことを確認します。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する

概念

プロジェクト参照

'<式>' を型制約として使用することはできません。

型パラメータに対する有効な制約を表さない式が、制約リストに含まれています。

制約リストには、型パラメータに渡される型引数に対する要件を設定します。次の要件を任意に組み合わせて指定できます。

- 型引数は、1 つまたは複数のインターフェイスを実装する必要があります。
- 型引数は、最大で 1 つのクラスを継承する必要があります。
- 型引数は、作成側のコードがアクセスできるパラメータなしのコンストラクタを公開する必要があります (**New** 制約を含む)。

特定のクラスまたはインターフェイスを制約リストに指定しない場合は、次のいずれかを指定することにより、一般的な要件を暗黙に示すことができます。

- 型引数は、値型である必要があります (**Structure** 制約を含む)。
- 型引数は、参照型である必要があります (**Class** 制約を含む)。

同じ型パラメータに対して **Structure** と **Class** の両方を指定することはできません。また、これらを複数回指定することもできません。

Error ID: BC32061

このエラーを解決するには

- 式と各要素のスペルが正しいかどうかを確認します。
- 式が上記の要件を満たしていない場合は、制約リストから式を削除します。
- 式がインターフェイスまたはクラスを参照する場合は、コンパイラがそのインターフェイスまたはクラスにアクセスできることを確認します。場合によっては、有効な名前かどうかを確認することや、プロジェクトに参照を追加することが必要です。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」の「プロジェクトへの参照」を参照してください。

参照

処理手順

方法 : 宣言された要素名を修飾する
方法 : Visual Studio のリファレンスを追加または削除する

概念

Visual Basic におけるジェネリック型
値型と参照型

'<eventname>' は '<containername>' のイベントではありません。

指定したイベントは、このオブジェクトで宣言されていません。

Error ID: BC30676

このエラーを解決するには

1. イベント名のスペルを確認します。
2. 正しいオブジェクトにアクセスしていることを確認します。必要に応じて、オブジェクトへの完全限定参照または **Imports** ステートメントを使用して、適切な名前空間をインポートします。

参照

関連項目

Imports ステートメント

その他の技術情報

Visual Basic におけるイベント

'<eventname>' はイベントであるため、直接呼び出すことはできません。

'*<eventname>*' はイベントであるため、直接呼び出すことはできません。**RaiseEvent** ステートメントを使用して、イベントを発生させます。

プロシージャの呼び出しで、プロシージャ名についてイベントが指定されています。イベント ハンドラはプロシージャですが、イベント自体はシグナル デバイスであり、発生および処理する必要があります。

Error ID: BC32022

このエラーを解決するには

- **RaiseEvent** ステートメントを使用してイベントを通知し、それを処理するプロシージャを呼び出します。

参照

処理手順

方法 : イベントをクラスに追加する

関連項目

RaiseEvent ステートメント

'<eventname>' は、暗黙に宣言されている <type> '<typename>'と競合する '<membername>' を暗黙に定義します。

型メンバの名前が、イベントについて暗黙的に作成されたメンバの名前と競合しています。イベントは、複数の変数を暗黙に作成します。たとえば、宣言 `Event X` は、`XEventHandler`、`XEvent`、`add_X`、および `remove_X` という名前を暗黙的に宣言します。

Error ID: BC31059

このエラーを解決するには

- 明示的に宣言したメンバの名前を変更して、名前の競合を解決します。

参照

概念

Visual Basic の宣言ステートメント

その他の技術情報

Visual Basic におけるイベント

<error>: '<structurename1>' に '<structurename2>'が含まれています。

構造体のメンバに、その構造体そのものを指定すると、構造体の定義が循環します。

Error ID: BC30293

このエラーを解決するには

- 構造体メンバの名前を変更します。

参照

その他の技術情報

構造体 : 独自のデータ型

<error>: '<constructorname1>' が '<constructorname2>' を呼び出しています。

コンストラクタの呼び出しが循環しています。コンストラクタでは **Me.New()** または **MyClass.New()** が呼び出されます。異なる引数リストを持つ、オーバーロードされたコンストラクタを呼び出そうとしたことが原因の 1 つとして考えられます。

Error ID: BC30297

このエラーを解決するには

- 異なる引数リストを使用して、オーバーロードされたコンストラクタを呼び出します。
- アクセスできるオーバーロードされたコンストラクタがない場合は、**Me.New()** および **MyClass.New()** の呼び出しを削除します。

参照

関連項目

コンストラクタとデストラクタの使用方法

<error>: '<classname1>' は '<classname2>' から継承されます。

循環する継承階層が検出されました。クラスがそれ自身を継承するように指定されているか、またはそのクラスを直接または最終的に継承する別のクラスを継承するように指定されています。

循環する継承パスをトレースするために、このメッセージが複数回表示されることもあります。

Error ID: BC30256

このエラーを解決するには

- 循環する継承パス内の少なくとも 1 つの **Inherits** ステートメントを削除して、循環を切断します。

参照

概念

継承の基本

'<emptyconstant>' は宣言されていません。

エラー メッセージ
'<emptyconstant>' は宣言されていません。Empty 定数はサポートされていません。代わりに Nothing を使用してください。

宣言または代入ステートメントが、変数、定数、列挙型のメンバ、プロパティ、または関数の戻り値に、**Empty** 値を代入しようとしています。

以前のバージョンの Visual Basic では、初期化されていないデータ ストレージを表すために **Empty** キーワードが使用されていました。Visual Basic 2005 では **Empty** はサポートされていません。初期化されていない変数には、そのデータ型の既定値が設定されます。既定値の詳細については、「Dim ステートメント (Visual Basic)」の「既定値」を参照してください。

Nothing (Visual Basic) キーワードは任意のデータ型の既定値を表します。**Empty** の代わりにこのキーワードを使用できます。

Error ID: BC30823

このエラーを解決するには

- **Empty** の代わりに **Nothing** を使用します。

  または

- プログラミング要素のデータ型に適した既定値を使用します。

  または

- 変数宣言の場合には、初期値を代入しないようにします。こうすると、変数は、その変数の既定値で初期化されます。

参照

関連項目
Nothing (Visual Basic)
概念
プログラミング要素のサポートに関する変更の概要

'<elementname>' はプロジェクト '<projectname>' の型 '<typename>' を参照していますが、型 '<typename>' がプロジェクト '<projectname>' に見つかりません。

式は、別のプロジェクトで参照されるクラス、構造体、モジュール、またはインターフェイスにアクセスしますが、そのプロジェクトには指定された型が含まれていません。

このエラーは、プロジェクトが同じソリューション内の別のプロジェクトへの間接参照を作成するときに発生します。通常、プロジェクトは、他のプロジェクトへの参照を作成するアセンブリへの参照を作成します。アセンブリが他のプロジェクトの指定された型にアクセスする場合、その型への間接参照が確立されます。しかし、他のプロジェクトに指定された型が存在しない場合は、このエラーが生成されます。

Error ID: BC30960

このエラーを解決するには

- 問題の型が既にどこにも定義されていない場合は、その型にアクセスしようとするステートメントを削除するか置き換えます。問題の型への間接参照を提供するアセンブリでも、同じ変更が必要になる場合があります。
- 問題の型がどこかに定義されている場合は、それを定義するプロジェクトまたはアセンブリへの直接参照を作成します。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する

概念

プロジェクト参照

その他の技術情報

名前空間およびコンポーネントの参照

参照の管理

'<elementname>' は古い形式です: '<errormessage>'

アクセスしようとしているプログラミング要素が、ObsoleteAttribute 属性としてマークされています。また、ディレクティブによる警告対象とされています。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

既定では、**ObsoleteAttribute** の **IsError** プロパティが **False** であるため、これは警告メッセージになります。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC40000

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. ソース コード参照の要素名のスペルが正しいことを確認します。

参照

概念

Visual Basic で使用される属性
属性の適用

'<elementname>' は古い形式です (Visual Basic 警告)

ステートメントが、ObsoleteAttribute 属性でマーク付けされているプログラミング要素にアクセスしています。また、ディレクティブがそれを警告として扱うように設定されています。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

既定では、**ObsoleteAttribute** の **IsError** プロパティが **False** であるため、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC40008

このエラーを解決するには

- ソース コード参照で要素名のスペルが正しいことを確認します。

参照

概念

Visual Basic で使用される属性
属性の適用

'<elementname>' は古い形式です (Visual Basic エラー)

ステートメントが、ObsoleteAttribute 属性でマーク付けされているプログラミング要素にアクセスしています。また、ディレクティブがそれをエラーとして扱うように設定されています。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

Error ID: BC31075

このエラーを解決するには

- ソース コード参照で要素名のスペルが正しいことを確認します。

参照

概念

Visual Basic で使用される属性
属性の適用

'<型> '<typename>' に同じ名前のメンバが多種類存在するため、'<elementname>' があいまいです。

式は、同じ名前の複数のメンバを含むクラス、構造体、モジュール、またはインターフェイスに定義されているプログラミング要素にアクセスします。

このエラーの最も可能性の高い原因は、大文字と小文字の区別です。Visual Basic では、名前の大文字と小文字が区別されません。つまり、コード内のさまざまな場所にある名前で大文字小文字を統一する必要はありません。たとえば、`XYZ` という名前で変数を定義し、後でその変数に `xyz` としてアクセスする場合、コンパイラは 2 つの名前を同等と見なします。

しかし、Visual C# や Visual C++ など、他の言語では大文字 と小文字が区別されます。このような言語では、`XYZ` と `xyz` は同じ名前と見なされません。したがって、これらの言語で記述されたクラスでは、`XYZ` という名前の変数と、`xyz` という名前のプロパティを定義できます。共通言語ランタイム (CLR: common language runtime) は、アセンブリ内で大文字小文字の区別を維持しています。しかし、Visual Basic アプリケーションでアセンブリに `XYZ` および `xyz` の名前でアクセスする場合は、同じ名前として表示されます。

Error ID: BC31429

このエラーを解決するには

1. 定義している型のソース コードを制御できる場合は、メンバの名前を変更して、大文字と小文字以外でも区別できるようにすることを検討します。定義している型が既に公開され、他のアプリケーションで使用されている場合はこの方法では解決できない可能性があります。
2. 定義している型のメンバの名前を変更できない場合は、問題のプログラミング要素をコードから削除します。Visual Basic で、複数の定義を持っているとして表示される要素にはアクセスできません。

参照

処理手順

Visual Basic における変数のトラブルシューティング

概念

宣言された要素の名前

その他の技術情報

共通言語ランタイム

'<qualifiedelementname>' へのエイリアスのインポート '<elementname>' は Namespace、Class、Structure、Interface、Enum または Module を参照していません。

Imports ステートメント は、インポートできないプログラミング要素を指定しています。

Imports ステートメントは、要素名の前に付ける必要がある修飾文字列を短縮するか不要にするために使用します。**Imports** ステートメント内で要素を修飾すると、一意な要素宣言へのパスを明確化できます。それ以降は、その要素を指す参照を修飾する必要はありません。

Imports は名前空間によく使用されますが、クラス、モジュール、構造体、インターフェイス、または列挙をインポートすると、それらの要素を指す参照に長い修飾文字列を付ける必要はなくなります。

詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」の「コンテナ要素のインポート」を参照してください。

Error ID: BC30798

このエラーを解決するには

1. **Imports** ステートメント内の修飾文字列に含まれる要素のスペルをチェックします。特に、修飾の対象である最後の要素についてはスペルをよく確認してください。
2. 修飾している要素が有効な型 (名前空間、クラス、モジュール、構造体、インターフェイス、または列挙) かどうかを確認します。有効な型ではない場合は、**Imports** ステートメントを削除します。

参照

概念

参照と Imports ステートメント

型 '<internaltypename>' のアクセスをアセンブリの外側に展開しているため、'<derivedtypename>' は <型> '<constructedbasetypename>' から継承できません。

派生クラスまたはインターフェイスが、制限された型を基本クラスまたはインターフェイスへの型引数として使うことにより、制限された型のアクセス レベルを拡張しようとしています。

このエラーは次のようなコードで発生します。

```
Public Class baseClass(Of t)
End Class
Public Class derivedClass
    Inherits baseClass(Of restrictedStructure)
End Class
Friend Structure restrictedStructure
    Dim firstMember As Integer
End Structure
```

アセンブリの外側にあるコードには、`restrictedStructure` へのアクセスは許可されません。しかし、`derivedClass` へのアクセスは、これを参照できるコードであれば可能です。したがって、`derivedClass` は、`restrictedStructure` を型引数として使用している場合は、`baseClass` を継承できません。これを許可すると、どのアセンブリ内のコードにも `restrictedStructure` を公開することが可能になるからです。

Error ID: BC30922

このエラーを解決するには

- クラスまたはインターフェイスのアクセス レベルを修正して、制限された型のアクセス レベルが派生型によって拡張されないようにします。

  または

- アクセス レベルを修正できない場合は、基本クラスまたはインターフェイスを作成するとき、制限された型を型引数として使用しないでください。

参照

関連項目

Inherits ステートメント
型リスト

概念

継承の基本
Visual Basic でのアクセス レベル
Visual Basic におけるジェネリック型

型 '<internaltypename>' のアクセスを <領域> '<regionname>' へ展開しているため、'<derivedtypename>' は <型> '<constructedbasetypename>' から継承できません。

派生クラスまたは派生インターフェイスが、内部型を基本クラスまたは基本インターフェイスへの型引数として使用することにより、内部型のアクセス レベルを拡張しようとしています。

このエラーは次のようなコードで発生します。

```
Public Class containingClass
    Public Class baseClass(Of t)
    End Class
    Friend Class derivedClass
        Inherits baseClass(Of internalStructure)
    End Class
    Private Structure internalStructure
        Dim firstMember As Integer
    End Structure
End Class
```

`containingClass` の外部にあるコードは、`internalStructure` にアクセスできません。ただし、`derivedClass` には、同じアセンブリ内のコードからアクセスできます。したがって、`derivedClass` は、`internalStructure` を型引数として使用する場合、`baseClass` を継承できません。これを認めると、コードを定義する領域全体に `internalStructure` が公開されることになるからです。

Error ID: BC30921

このエラーを解決するには

- クラスまたはインターフェイスのアクセス レベルを調整して、派生型が内部型のアクセス レベルを拡張しないようにします。

  または

- アクセス レベルを調整できない場合は、基本クラスまたは基本インターフェイスを作成するときに内部型を型引数として使用しないでください。

参照

関連項目

Inherits ステートメント
型リスト

概念

継承の基本
Visual Basic でのアクセス レベル
Visual Basic におけるジェネリック型

'<declaration1>' と '<declaration2>' には、異なるアクセル レベルが指定されているため、'<declaration1>' で '<declaration2>' をオーバーライドすることはできません。

プロシージャ宣言またはプロパティ宣言で、同じ名前を持つ継承された要素をオーバーライドしようとしていますが、継承された要素とは異なるアクセシビリティが指定されています。**Public**、**Private** など、継承された要素のアクセシビリティは、オーバーライドで保持される必要があります。

Error ID: BC30266

このエラーを解決するには

- オーバーライドする側の要素のアクセシビリティを変更して、継承された要素のアクセシビリティと同じにします。

参照

概念

プロパティとメソッドのオーバーライド
Visual Basic でのアクセス レベル

'<declaration2>' は 'Shared' として宣言されているため、'<declaration1>' で '<declaration2>' をオーバーライドすることはできません。

プロシージャ宣言またはプロパティ宣言で、同じ名前を持つ継承された要素をオーバーライドしようとしていますが、継承された要素は **Shared** として指定されています。共有される要素はクラスのどのインスタンスにも関連付けられていないため、オーバーライドできません。

Error ID: BC30268

このエラーを解決するには

- 継承された要素から **Shared** キーワードを削除するか、またはオーバーライドの宣言を削除します。

参照

概念

プロパティとメソッドのオーバーライド

'<declaration2>' は、'NotOverridable' として宣言されているため、'<declaration1>' で '<declaration2>' をオーバーライドすることはできません。

プロシージャ宣言またはプロパティ宣言で、同じ名前を持つ継承された要素をオーバーライドしようとしていますが、継承された要素は **NotOverridable** として指定されています。

Error ID: BC30267

このエラーを解決するには

- 継承された要素の宣言から **NotOverridable** キーワードを削除するか、またはオーバーライドの宣言を削除します。

参照

概念

プロパティとメソッドのオーバーライド

'<classname1>' を <type> '<classname2>' から継承できません。'<classname2>' は 'NotInheritable' として宣言されています。

クラスが別のクラスを継承しようとしていますが、目的の基本クラスが **NotInheritable** として指定されています。**NotInheritable** クラスは主に、意図しない派生を避けるために使用されます。

Error ID: BC30299

このエラーを解決するには

- 希望する基本クラスの定義から **NotInheritable** キーワードを削除するか、または **Inherits** ステートメントを削除します。

参照

関連項目

NotInheritable
Inherits ステートメント

概念

継承の基本

'<classname>' は継承元のインターフェイス '<interfacename>' が CLS に準拠していないため、CLS に準拠していません。

クラスまたはインターフェイスが `<CLSCompliant(True)>` でマーク付けされていますが、この継承元の型または実装している型が `<CLSCompliant(False)>` でマーク付けされているか、マークが付いていません。

クラスまたはインターフェイスを 共通言語仕様 (CLS) 準拠にするには、その継承階層全体を準拠させる必要があります。つまり、その継承元の型も (直接的、間接的継承を含めて) すべて CLS 準拠であることが必要です。同様に、クラスが 1 つ以上のインターフェイスを実装する場合、継承階層全体ですべて準拠している必要があります。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に設定しなかった場合は、非準拠であると見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40029

このエラーを解決するには

- CLS 準拠にする必要がある場合は、この型を別の継承階層または別の実装スキームに定義します。
- この型を現在の継承階層または実装スキームに残しておく必要がある場合は、**CLSCompliantAttribute** を定義から削除するか、`<CLSCompliant(False)>` でマーク付けします。

参照

概念

CLS 準拠コードの記述

'<classname>' は、CLS に準拠していない '<baseclassname>' から派生しているため、CLS に準拠していません。

クラスまたはインターフェイスが `<CLSCompliant(True)>` としてマークされていますが、それが継承または実装している型が `<CLSCompliant(False)>` としてマークされています (またはマークされていません)。

クラスまたはインターフェイスを共通言語仕様 (CLS) 準拠にするためには、継承階層全体を CLS に準拠させる必要があります。つまり、直接的または間接的に継承するすべての型を CLS に準拠させる必要があります。同様に、クラスが 1 つ以上のインターフェイスを実装する場合は、そのすべてのインターフェイスの継承階層全体を CLS 準拠にする必要があります。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40026

このエラーを解決するには

- CLS 準拠にする必要がある場合は、この型を別の継承階層または実装スキームの中で定義します。
- この型を現在の継承階層または実装スキームに残しておく必要がある場合は、**CLSCompliantAttribute** を定義から削除するか、`<CLSCompliant(False)>` でマーク付けします。

参照

概念

CLS 準拠コードの記述

'implements <derivedinterfacename>' からの '<baseinterfacename>.<membername>' は基本クラス '<baseclassname>' によって既に実装されているため、さらに実装することはできません。<type> の再実装と見なされます。

派生クラス内のプロパティ、プロシージャ、またはイベントが、基本クラス内の基本インターフェイスに既に実装されている派生インターフェイス メンバを **Implements** 句に指定しています。

実装しようとしているメンバは、基本インターフェイスに定義され、派生インターフェイスに継承されています。基本クラスは、基本インターフェイスを直接実装します。派生クラスは派生インターフェイスを実装しますが、基本クラスに既にメンバが実装されていることは見落としやすい事実です。

派生クラスは、基本クラスに実装されているインターフェイス メンバを再実装できます。これは、基本クラスの実装をオーバーライドするということではありません。詳細については、「Implements (Visual Basic)」を参照してください。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42014

このエラーを解決するには

- インターフェイス メンバを再実装する場合は、特別なアクションは必要ありません。MyBase キーワードを使って基本クラスの実装にアクセスしない限り、派生クラスのコードでは再実装したメンバがアクセスされます。
- インターフェイス メンバを再実装しない場合は、**Implements** 句をプロパティ、プロシージャ、またはイベントの宣言から削除します。

参照

概念

Implements キーワードおよび Implements ステートメント

その他の技術情報

Visual Basic におけるインターフェイス

'<attributename>' をアセンブリに 2 回以上適用することはできません。

指定した属性は、属性に対して 1 回しか適用できません。

Error ID: BC31521

このエラーを解決するには

1. この属性の余分な適用を削除します。
2. 独自に開発したカスタム属性を使用している場合は、**AttributeUsageAttribute** を変更し、**AllowMultiple** プロパティに **True** を設定します。

参照

関連項目

AttributeUsageAttribute Class
AttributeUsageAttribute.AllowMultiple Property

GUID '<number>' の形式が正しくないため、'<attribute>' を適用できません。

COMClassAttribute 属性ブロックで、グローバル一意識別子 (GUID: globally unique identifier) の形式として有効でない GUID が指定されています。**COMClassAttribute** は GUID を使ってクラス、インターフェイス、および作成イベントを一意に識別します。

GUID は 16 バイトで構成され、前の 8 バイトは数値、後の 8 バイトはバイナリです。GUID は uuidgen.exe などの Microsoft ユーティリティで生成され、空間および時間内で一意であることが保証されています。

Error ID: BC32500

このエラーを解決するには

1. COM オブジェクトを識別するために必要な正しい GUID を決定します。
2. **COMClassAttribute** 属性ブロックに示される GUID 文字列が正しくコピーされていることを確認します。

参照

関連項目

ComClassAttribute クラス
Guid Structure

概念

Visual Basic で使用される属性
属性の適用

'<assemblyname>' はアセンブリであるため、モジュールとして参照できません。

アセンブリをモジュールとして使おうとしました。

Error ID: BC31077

このエラーを解決するには

- モジュールへの参照を変更します。

参照

関連項目

Module ステートメント
Assembly

'<argumentname>' は、'<methoddeclaration>' のパラメータではありません。

プロシージャ呼び出しで、プロシージャによって定義されていない追加の引数が指定されています。

Error ID: BC30272

このエラーを解決するには

- 引数リストから余分な引数を削除します。

参照

概念

プロシージャのパラメータと引数
同じ名前を持つ複数の変数がある場合に参照を解決する

'}' が必要です。

配列初期化子または制約リストが、正しい方法で終了されていません。

配列を初期化する要素値は、中かっこ (**{}**) で囲む必要があります。

```
Dim demoArray() As Integer = New Integer() {1, 2, 3}
```

同様に、ジェネリック型パラメータの制約リストに含める制約も中かっこで囲む必要があります。

```
Public Class dictionaryMaker(Of t As {IComparable, IDisposable, New})
```

Error ID: BC30370

このエラーを解決するには

- "}" を使って配列初期化子または制約リストを終了します。

参照

処理手順

方法 : 配列変数を初期化する

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

その他の技術情報

Visual Basic における配列

'<typename>' はジェネリック型であり、型引数を必要とします。

変数、プロシージャのパラメータ、または関数の戻り値が、ジェネリックなクラスまたは構造体の型で宣言されていますが、この宣言には型引数が指定されていません。

すべてのジェネリックなクラスおよび構造体は、その特質上、少なくとも 1 つの型パラメータを使って定義されます。構成されるクラスまたは構造体を、ジェネリック型を使って宣言する場合は、ジェネリック型で定義されるすべての型パラメータに対して型引数を指定する必要があります。

Error ID: BC32076

このエラーを解決するには

- 型リストをかっこで囲んで、先頭に **Of** キーワードを付けて、宣言に追加します。

参照

関連項目

Of
型リスト

概念

Visual Basic におけるジェネリック型

デザイナで生成された型 '<型>' の '<コンストラクタ>' は InitializeComponent メソッドを呼び出さなければなりません。

デザイナで生成された型のコンストラクタが、その型の `InitializeComponent` メソッドを呼び出していません。

デザイナで生成された型の各コンストラクタは、その型の `InitializeComponent` メソッドを呼び出す必要があります。

Error ID: BC40054

このエラーを解決するには

- コンストラクタに `InitializeComponent` メソッドの呼び出しを追加します。

参照

関連項目

コンストラクタとデストラクタの使用方法

DesignerGeneratedAttribute

アセンブリ '<assemblyname>' の型 '<typename>' はそれ自体に転送され、サポートされていない型もそれ自体に転送されました。

アセンブリが自身の 1 つの型を別のアセンブリに転送するために TypeForwardedToAttribute を使用していますが、同じアセンブリ内の同じ型を指定しています。

型の転送とは、クラス、構造体、インターフェイス、デリゲート、または列挙体の定義を、もともと定義されているアセンブリとは別のアセンブリに再割り当てすることです。型の転送は、多くの場合コードのリファクタリングと併せて使用されます。これによって、アセンブリを複数のアセンブリに分割したり、コードを別のアセンブリに移動したりします。

型を自分自身に転送すると循環転送になります。この転送された型に別のアセンブリがアクセスしようとした場合は、転送されていない型に到達できないまま、永遠に転送が繰り返されることになります。

Error ID: BC31425

このエラーを解決するには

- 目的の型を別のアセンブリ内の型に転送するか、転送を行いません。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する

関連項目

Type Forwarding

TypeForwardedToAttribute

概念

プロジェクト参照

'<membername> には型パラメータとして同じ名前が指定されています。

ジェネリックなクラスまたは構造体のメンバに、そのクラスまたは構造体の型パラメータと同じ名前が付いています。

すべての変数、定数、列挙値、プロシージャ、プロパティ、イベントおよびデリゲートは、すべての型パラメータと名前が違っている必要があります。

Error ID: BC32054

このエラーを解決するには

- クラスまたは構造体のメンバの名前を変更して、どの型パラメータ名とも同じにならないようにします。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

'<名前>' は、既定のインスタンスからそれ自体を参照できません。'Me' を使用してください。

フォームの内部から、そのフォームの既定のインスタンスを参照しようとしています。これにより、フォームがそれ自体を再帰的に呼び出す可能性があります。

ほとんどの場合、フォームの現在のインスタンスを参照するときには、既定のインスタンスではなく、**Me** を使用する必要があります。

Error ID: BC31139

このエラーを解決するには

- **Me** を使用してオブジェクトを参照します。

参照

その他の技術情報

デバッガのロードマップ

ベース <型> のアクセスをアセンブリの外側に展開しているため、'<typename>' は <型> '<basetypename>' から継承できません。

基本クラスまたはインターフェイスを継承したクラスまたはインターフェイスの、アクセス レベルの制限が低くなっています。

たとえば、**Public** インターフェイスが **Friend** インターフェイスを継承する、または **Protected** クラスが **Private** クラスを継承するなどの例が挙げられます。この場合、基本クラスまたはインターフェイスへのアクセスが、指定したレベルで制限されません。

Error ID: BC30910

このエラーを解決するには

- 派生クラスまたはインターフェイスのアクセス レベルを、基本クラスまたはインターフェイスと少なくとも同じレベルで制限するように変更します。

  または

- 制限の低いアクセス レベルを使用する必要がある場合は、**Inherits** ステートメントを削除します。より制限の高い基本クラスまたはインターフェイスは継承できません。

参照

関連項目

Class ステートメント (Visual Basic)
Interface ステートメント (Visual Basic)
Inherits ステートメント

概念

Visual Basic でのアクセス レベル

'<membername>' は、型 '<typename>' を <containertype> '<containertypename>' 経由でプロジェクトの外側に公開できません。

変数、プロシージャ パラメータ、または関数の戻り値がコンテナ外部に公開されていますが、これらの要素はコンテナ外部に公開してはならない型として宣言されています。

次のスケルトン コードは、このエラーが発生する状況の例を示しています。

```
Private Class privateClass
End Class
Public Class mainClass
    Public exposedVar As New privateClass
End Class
```

型を **Protected**、**Friend**、**Protected Friend**、または **Private** として宣言したときは、その宣言コンテキストの外側でのアクセスを制限することを意図しています。このような型を、それよりもアクセス レベルの緩い変数のデータ型として使用することは、この目的に反します。上記のスケルトン コードでは、`exposedVar` を **Public** として宣言しており、`privateClass` をコンテキスト外のコードに公開しようとしています。

Error ID: BC30909

このエラーを解決するには

- 変数、プロシージャ パラメータ、または関数の戻り値のアクセス レベルを変更して、少なくとも、そのデータ型のアクセス レベルと同じ厳しさにします。

参照

概念

Visual Basic でのアクセス レベル

戻り値の型が指定された '<eventsignature>' を宣言しているため、型 '<typename>' は、インターフェイス '<interfacename>' を実装できません。

クラスまたは構造体がインターフェイスの実装を試みていますが、そのインターフェイスには値を返すイベントが宣言されています。

Visual Basic は現在、値を返すイベントの宣言をサポートしていません。

Error ID: BC30945

このエラーを解決するには

- **Implements** ステートメントをクラスまたは構造体の定義から削除するか、別のインターフェイスを実装します。

参照

関連項目

Implements ステートメント

概念

イベントとイベント ハンドラ

Implements キーワードおよび Implements ステートメント

'.' が必要です。

感嘆符 (**!**) を含んでいる **Handles** 句がコードに存在します。

Error ID: BC30287

このエラーを解決するには

- **Handles** 句がオブジェクト内のイベントを参照している場合、オブジェクトとイベントを区切るためにピリオド (.) を使用します。

  この例は、`Button1` オブジェクトの **Click** イベントを処理します。

  VB

```
Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
```

参照

関連項目

Handles

')' が必要です。

プロシージャ定義内やプロシージャ呼び出し内の引数リストなどのリストが、右かっこ (**)**) で閉じられていません。

Error ID: BC30198

このエラーを解決するには

- リストの最後に右かっこを追加します。

参照

概念

プロシージャのパラメータと引数

'(' は無効です

エラー メッセージ
'(' は無効です。インスタンス生成されないジェネリック型の配列は許可されていません。

Visual Basic では、データ型が指定されていない配列はコンパイルできません。ジェネリック型のデータ型パラメータを配列のデータ型として使うことはできません。

Error ID: BC32095

このエラーを解決するには

- 配列を使用する場合は、特定のデータ型で宣言する必要があります。データ型パラメータは使用できません。
- データ型パラメータに渡すことができるデータ型である要素のグループが必要な場合は、配列ではなくコレクションを使う必要があります。

参照

概念

Visual Basic におけるジェネリック型
Visual Basic におけるコレクション
オブジェクトのグループの管理

'(' が必要です。

プロシージャ呼び出し内の引数リストが、左かっこ (() で始まっていません。

Error ID: BC30199

このエラーを解決するには

- リストの先頭に左かっこを追加します。

参照

概念

プロシージャのパラメータと引数

'#Region' ステートメントの終わりには、対応する '#End Region' を指定しなければなりません。

#Region ディレクティブは、Visual Studio コード エディタのアウトライン機能を使用して展開したり折りたたんだりできる、コードのブロックを指定するために使用します。**#Region** ステートメントは、最初から最後まで同じコード ブロックにある必要があります。

Error ID: BC30681

このエラーを解決するには

- コード内の **#Region** ステートメントより後の適切な位置に **#End Region** を挿入します。

参照

関連項目

#Region ディレクティブ

'#Region' および '#End Region' ステートメントは、メソッド本体の内部では有効ではありません。

#Region ブロックは、クラス、モジュール、または名前空間のレベルで宣言する必要があります。折りたたみ可能なコード ブロックは、複数のプロシージャを含めることができますが、プロシージャの内部で開始または終了することはできません。

Error ID: BC32025

このエラーを解決するには

1. 直前のプロシージャが、**End Function** ステートメントまたは **End Sub** ステートメントによって正しく終了していることを確認します。
2. **#Region** ディレクティブおよび **#End Region** ディレクティブが同じコード ブロック内にあることを確認します。

参照

関連項目

#Region ディレクティブ

'#If' ブロックの最後には、対応する '#EndIf' が必要です。

#If は条件付きコンパイル ディレクティブです。**#If** ブロックが **#End If** ディレクティブで終了されていません。

Error ID: BC30012

このエラーを解決するには

- 条件付きコンパイル ブロックの最後に **#End If** ディレクティブを追加します。

参照

関連項目

#If...Then...#Else ディレクティブ

'#ExternalSource' ステートメントの終わりには、対応する '#End ExternalSource' を指定しなければなりません。

#ExternalSource ディレクティブは、外部のコードを参照しており、そのコード内でいつ例外が発生したのかをコンパイラが正確に報告できるようにします。**#ExternalSource** ブロックは、**#ExternalSource** で開始し、**#End ExternalSource** で終了する必要があります。

Error ID: BC30579

このエラーを解決するには

1. コード内の適切な位置に **#End ExternalSource** を追加します。
2. 必要ない場合は、ブロックを開始する **#ExternalSource** を削除します。

参照

関連項目

#ExternalSource ディレクティブ

その他の技術情報

条件付きコンパイル (Visual Basic)

'#ExternalSource' ディレクティブは入れ子にできません。

#ExternalSource ブロックの中で別の **#ExternalSource** ディレクティブを指定しようとしました。**#ExternalSource** ディレクティブは、外部のコードを参照しており、そのコード内でいつ例外が発生したのかをコンパイラが正確に報告できるようにします。

#ExternalSource ブロックをほかの **#ExternalSource** ブロック内に入れ子にすることはできません。

Error ID: BC30580

このエラーを解決するには

- 内側の **#ExternalSource** ディレクティブを外側の **#ExternalSource** ブロックの外に移動します。

参照

関連項目

#ExternalSource ディレクティブ

その他の技術情報

条件付きコンパイル (Visual Basic)

'#End Region' の前には、対応する '#Region' を指定しなければなりません。

#Region を使用すると、Visual Studio コード エディタのアウトライン機能を使用して展開したり折りたたんだりできる、コードのブロックを指定できます。**#Region** ステートメントは、最初から最後まで同じコード ブロックにある必要があります。

Error ID: BC30680

このエラーを解決するには

- **#Region** を、対応する **#End Region** ステートメントより前の適切な位置に挿入します。

参照

関連項目

#Region ディレクティブ

'#End ExternalSource' の前には、対応する '#ExternalSource' を指定しなければなりません。

#ExternalSource ディレクティブは、外部のコードを参照しており、そのコード内でいつ例外が発生したのかをコンパイラが正確に報告できるようにします。**#ExternalSource** ブロックは、**#ExternalSource** で開始し、**#End ExternalSource** で終了する必要があります。

Error ID: BC30578

このエラーを解決するには

1. コード内の適切な位置に **#ExternalSource** を追加します。
2. 不要な場合は、**#End ExternalSource** を削除します。

参照

関連項目

#ExternalSource ディレクティブ

その他の技術情報

条件付きコンパイル (Visual Basic)

'#ElseIf'、'#Else' または '#EndIf' の前には、対応する '#If' 節が必要です。

#ElseIf、**#Else**、および **#End If** は条件付きコンパイル ディレクティブです。**#ElseIf**、**#Else**、または **#End If** の前に、対応する **#If** ディレクティブが存在しません。

Error ID: BC30013

このエラーを解決するには

1. 対象の **#If** と、この句とが、間に条件付きコンパイル ブロックを挿入することによって分離されていないこと、または **#End If** 句を間違った場所に記述したために分離されていないことを確認します。

メモ :
1 つの **#If** ブロックに指定できる **#Else** は 1 つだけです。したがって、**#Else** ディレクティブを続けて指定すると、このエラーが発生します。

2. 前にある **#If** ディレクティブで、先頭の # が抜けていないことを確認します。
3. すべて問題ない場合は、条件付きコンパイル ブロックの先頭に **#If** ディレクティブを追加します。

参照

関連項目

#If...Then...#Else ディレクティブ

'#Else' の前には、対応する '#If' または '#ElseIf' が必要です。

#ElseIf は条件付きコンパイル ディレクティブです。**#ElseIf** 句の前には、対応する **#If** 句か **#ElseIf** 句が必要です。

Error ID: BC30014

このエラーを解決するには

1. 先行する **#If** または **#ElseIf** と、この **#ElseIf** とが、間に条件付きコンパイル ブロックを挿入することによって分離されていないこと、または **#End If** 句を間違った場所に記述したために分離されていないことを確認します。
2. **#ElseIf** の前に **#Else** ディレクティブがある場合は、その **#Else** を削除するか、**#ElseIf** に変更します。
3. すべて問題ない場合は、条件付きコンパイル ブロックの先頭に **#If** ディレクティブを追加します。

参照

関連項目

#If...Then...#Else ディレクティブ

'#If' ブロックの一部として '#ElseIf' を '#Else' の後に使用することはできません。

#ElseIf 条件付きコンパイル ディレクティブが **#Else** ディレクティブより後ろにあります。**#Else** は、**#End If** ディレクティブで終わる条件ブロックの最後のディレクティブである必要があります。

Error ID: BC32030

このエラーを解決するには

1. 前にある **#Else** を **#ElseIf** にすることができるかどうかを確認します。
2. 前の **#If** ブロックが正しく終了されていること、および新しい **#If** ブロックが開始されていることを確認します。
3. 他に間違いがなければ、この **#ElseIf** ディレクティブおよび対応するステートメント ブロックを **#Else** ブロックより前に移動します。

参照

関連項目

#If...Then...#Else ディレクティブ

'#Else' の前には、対応する '#If' または '#ElseIf' が必要です。

#Else は条件付きコンパイル ディレクティブです。**#Else** ディレクティブの前に、対応する **#If** ディレクティブ、または **#ElseIf** ディレクティブが存在しません。

Error ID: BC30028

このエラーを解決するには

1. 先行する **#If** または **#ElseIf** と、この **#Else** とが、間に条件付きコンパイル ブロックを挿入することによって分離されていないこと、または **#End If** 句を間違った場所に記述したために分離されていないことを確認します。
2. **#Else** の前に別の **#Else** ディレクティブが存在することを確認します。存在する場合は、その **#Else** を削除するか、**#ElseIf** に変更します。
3. すべて問題ない場合は、条件付きコンパイル ブロックの先頭に **#If** ディレクティブを追加します。

参照

関連項目

#If...Then...#Else ディレクティブ

'!' 左側のオペランドには、タイプ パラメータ、クラス、またはインターフェイス型のいずれかが必要ですが、オペランドの型は '<type>' です。

ピリオド (.) の代わりに感嘆符 (!) を使用するディクショナリ メンバ アクセスは、クラスまたはインターフェイスの場合にのみ有効です。

Error ID: BC30103

このエラーを解決するには

- 感嘆符の左辺の式を、定義したクラス型またはインターフェイス型に評価される式に置き換えます。

参照

概念

コード内の特殊文字

'-' が必要です。

この行の構文が正しくありません。

Error ID: BC30601

このエラーを解決するには

- このエラーの発生元となったコードの構文を確認します。

参照

その他の技術情報

プログラム構造とコード規則

インスタンスを経由する共有メンバへのアクセスです。正規の式は評価されません。

クラスまたは構造体のインスタンス変数を使用して、そのクラスまたは構造体の中で定義されている **Shared** 変数、プロパティ、プロシージャ、またはイベントにアクセスしようとしています。この警告は、インスタンス変数を使用して、クラスまたは構造体の暗黙的な共有メンバ (定数や列挙体など) や入れ子になったクラスまたは構造体にアクセスしようとする場合にも発生します。

メンバを共有する目的は、メンバを 1 つだけ作成して、そのメンバを、それを宣言したクラスまたは構造体のすべてのインスタンスから利用できるようにすることです。したがって、**Shared** メンバを使用するときには、クラスまたは構造体の個々のインスタンスを表す変数を通じてアクセスするよりも、クラスまたは構造体の名前を通じてアクセスした方が、本来の目的にかなっています。

インスタンス変数を通じて **Shared** メンバにアクセスすると、そのメンバが **Shared** であることが分かりにくくなって、コードが理解しにくくなります。さらに、このようなアクセス方法を、他の処理を実行する式 (たとえば共有メンバのインスタンスを返す **Function** プロシージャなど) の一部として使用した場合は、Visual Basic はその式と本来実行されたはずの他の処理を省略します。

使用例および詳細については、「Shared (Visual Basic)」を参照してください。

既定では、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC42025

このエラーを解決するには

- **Shared** メンバにアクセスするときには、それを定義したクラスまたは構造体の名前を使用します。
- 2 つのプログラミング要素が同じ名前であるときには、スコープの影響に注意してください。次の例では `testClass` という変数を宣言していますが、この変数名は、型となるクラスの名前と同じです。

```
Public Class testClass
    Public Shared Sub sayHello()
        MsgBox("Hello")
    End Sub
End Class
Module testModule
    Public Sub Main()
        Dim testClass As testClass
        testClass.sayHello()
    End Sub
End Module
```

この場合、`testClass` 変数は同名のクラスよりも狭いスコープを持つので、コンパイラは `sayHello()` の呼び出しを変数を通じたものと解釈します。

参照

関連項目

Shared (Visual Basic)

概念

Visual Basic におけるスコープ

アクセス修飾子は、'Get' または 'Set' のいずれか 1 つにのみ適用できますが、両方には適用できません。

プロパティ宣言では、Property ステートメント、Get ステートメント、および Set ステートメント (Visual Basic) でアクセス レベルを指定します。

プロパティにはアクセス レベルを指定できます。さらに、プロパティ プロシージャ (**Get** または **Set**) のうち最大 1 つに別のアクセス レベルを指定できます (ただし、そのアクセス レベルがプロパティのアクセス レベルよりも制限が厳しい場合)。両方のプロパティ プロシージャにアクセス レベルを指定することはできません。

Error ID: BC31101

このエラーを解決するには

- **Get** ステートメントまたは **Set** ステートメントからアクセス修飾子を削除します。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

概念

Property プロシージャ

アクセス修飾子 '<accessmodifier>' は有効ではありません。

Get ステートメント または Set ステートメント (Visual Basic) に、それらを含んでいるプロパティに指定されたアクセス レベルよりも制限の低いアクセス レベルが指定されています。

プロパティには常にアクセス レベルを指定できます。また、その最大 1 つのプロパティ プロシージャ (**Get** または **Set**) に、別のアクセス レベルを指定できます。ただし、プロパティよりも制限の高いアクセス レベルであることが必要です。たとえば、プロパティが **Friend** の場合、プロパティ プロシージャに **Private** は指定できますが、**Public** は指定できません。両方のプロパティ プロシージャにアクセス レベルを指定することはできません。

Error ID: BC31100

このエラーを解決するには

- プロパティ プロシージャのアクセス レベルを、プロパティよりも制限の高いものにするか、アクセス修飾子自体を削除します。
- Property ステートメント に制限の低いアクセス レベルを宣言し、プロパティ プロシージャの 1 つに、それよりも制限の高いアクセス レベルを宣言します。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

概念

Property プロシージャ

エイリアス '<namespacename>' は、既に宣言されています。

Imports を使って、既に宣言されている名前空間をインポートしようとしました。

Error ID: BC30572

このエラーを解決するには

- インポートしようとしている名前空間を確認します。

参照

関連項目

Imports ステートメント
Alias

'AddressOf' オペランドはメソッドの名前でなければなりません。かっこは不要です。

AddressOf 演算子は、特定のプロシージャを参照するプロシージャ デリゲート インスタンスを作成します。構文は次のとおりです。

AddressOf *procedurename*

AddressOf の後の引数をかっこで囲んでありますが、このかっこは必要ありません。

Error ID: BC30577

このエラーを解決するには

1. **AddressOf** の後の引数を囲んでいるかっこを削除します。
2. 引数がメソッド名であることを確認します。

参照

関連項目

AddressOf 演算子

概念

デリゲートと AddressOf 演算子

'AddressOf' 式は、デバッグ ウィンドウでは有効ではありません。

AddressOf ステートメントはソース コードでのみ使用できます。

Error ID: BC30731

このエラーを解決するには

- デバッグ コードから **AddressOf** ステートメントを削除します。

参照

関連項目

AddressOf 演算子

その他の技術情報

Visual Studio でのデバッグ

'<typename>' は、'MustInherit' として宣言されていて作成することができないので、'AddressOf' 式を '<typename>' に変換することはできません。

ステートメントが、**AddressOf** 式を、基本クラスとしてのみ使用でき、インスタンスの作成には使用できない型に変換しようとしています。

AddressOf 演算子は、特定のプロシージャを参照するプロシージャ デリゲート インスタンスを作成します。

Error ID: BC30939

このエラーを解決するには

- **AddressOf** 式を、特定のデリゲート型に代入します。

参照

関連項目

AddressOf 演算子

概念

デリゲートと AddressOf 演算子

その他の技術情報

Visual Basic でのデリゲート

'<typename>' は、デリゲート型でないため、'AddressOf' 式を '<typename>' に変換できません。

ステートメントが、**AddressOf** 式をデリゲート型ではない型に変換しようとしました。

AddressOf 演算子は、特定のプロシージャを参照するプロシージャ デリゲート インスタンスを作成します。**AddressOf** は、デリゲート コンストラクタのオペランドとして使用できます。また、デリゲートの種類をコンパイラで決定できる状況で使用できます。

Error ID: BC30581

このエラーを解決するには

- 変換先の型をデリゲート型に変更します。

参照

関連項目

AddressOf 演算子

概念

デリゲートと AddressOf 演算子

'AddHandler' メソッド、'RemoveHandler' メソッド、および 'RaiseEvent' メソッド パラメータは、<modifier>' と宣言できません。

AddHandler メソッド、**RemoveHandler** メソッド、および **RaiseEvent** メソッドのパラメータは、**Optional** 修飾子や **ParamArray** 修飾子を付けて宣言できません。

Optional 修飾子または **ParamArray** 修飾子は、**Declare** 宣言、**Function** 宣言、**Property** 宣言、および **Sub** 宣言のパラメータでのみ使用できます。

Error ID: BC31138

このエラーを解決するには

- パラメータ リストから **Optional** キーワードまたは **ParamArray** キーワードを削除します。

参照

関連項目

Event ステートメント
AddHandler
RemoveHandler
RaiseEvent
Optional (Visual Basic)
ParamArray

その他の技術情報

Visual Basic におけるイベント

'AddHandler' または 'RemoveHandler' ステートメントのイベント オペランドは、ドットで限定された式または簡易名でなければなりません。

AddHandler または **RemoveHandler** のイベント引数に指定した項目がイベントとして認識されません。

Error ID: BC30677

このエラーを解決するには

- 次の例のように、イベントを発生させるオブジェクトの名前に続けて、ドット (.)、およびイベント名を指定します。

  VB

```
' Assume that the class Wobject has an event named ThisEvent.
Dim wObj As New Wobject
' Assume that this class has as method named ThisEventHandler.
AddHandler wObj.ThisEvent, AddressOf Me.ThisEventHandler
```

参照

関連項目

AddHandler ステートメント

RemoveHandler ステートメント

概念

AddHandler と RemoveHandler

その他の技術情報

Visual Basic におけるイベント

'AddHandler' は既に宣言されています。

カスタム イベントの宣言に **AddHandler** 宣言が複数あります。**AddHandler** 宣言は、イベント ハンドラを追加するためのプロシージャを宣言します。

Error ID: BC31127

このエラーを解決するには

- 余分な **AddHandler** ステートメントを削除します。

参照

関連項目

AddHandler ステートメント
Event ステートメント

イベント '<eventname>' に対して 'AddHandler' 定義がありません。

イベントを **Custom** として宣言している場合は、イベント ハンドラ追加用のプロシージャを記述する必要があります。

Error ID: BC31130

このエラーを解決するには

1. **Custom Event** ステートメントと **End Event** ステートメントの間で、**AddHandler** を宣言します。
2. このイベント宣言内のその他のプロシージャが正しく終了していることを確認します。

参照

関連項目

AddHandler ステートメント
Event ステートメント

'AddHandler' 宣言の終わりには、対応する 'End AddHandler' を指定してください。

AddHandler 宣言の終わりには、**End AddHandler** ステートメントが必要です。

Error ID: BC31115

このエラーを解決するには

- **AddHandler** 宣言が **End AddHandler** ステートメントで終わっていることを確認します。

参照

関連項目

AddHandler ステートメント
Event ステートメント

'AddHandler' および 'RemoveHandler' メソッドには、パラメータを 1 つだけ指定しなければなりません。

カスタム イベントの宣言には、カスタム イベントの **As** 句で指定したデリゲート型のパラメータを 1 つ持つ、**AddHandler** 宣言か **RemoveHandler** 宣言を含める必要があります。

Error ID: BC31133

このエラーを解決するには

- 余分なパラメータをパラメータ リストから削除し、パラメータの型を、カスタム イベントの **As** 句で指定したデリゲート型に変更します。

使用例

この例のカスタム イベントでは、**AddHandler** 宣言と **RemoveHandler** 宣言に対して正しいパラメータ型を指定しています。

VB

```
Custom Event Test As System.EventHandler
    AddHandler(ByVal value As System.EventHandler)
        ' Code for adding an event handler goes here.
    End AddHandler

    RemoveHandler(ByVal value As System.EventHandler)
        ' Code for removing an event handler goes here.
    End RemoveHandler

    RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
        ' Code for raising an event goes here.
    End RaiseEvent
End Event
```

参照

関連項目

Event ステートメント

AddHandler

RemoveHandler

その他の技術情報

Visual Basic におけるイベント

'AddHandler' および 'RemoveHandler' メソッド パラメータには、含んでいるイベントと同じデリゲート型を指定しなければなりません。

Custom Event 宣言には、カスタム イベントの **As** 句によって指定されたデリゲート型のパラメータを 1 つ持つ、**AddHandler** 宣言か **RemoveHandler** 宣言を含める必要があります。

Error ID: BC31136

このエラーを解決するには

- パラメータの型を、カスタム イベントの **As** 句で指定されたデリゲート型に変更します。

使用例

この例のカスタム イベントでは、**AddHandler** 宣言と **RemoveHandler** 宣言に対して正しいパラメータ型を指定しています。

VB

```
Custom Event Test As System.EventHandler
    AddHandler(ByVal value As System.EventHandler)
        ' Code for adding an event handler goes here.
    End AddHandler

    RemoveHandler(ByVal value As System.EventHandler)
        ' Code for removing an event handler goes here.
    End RemoveHandler

    RaiseEvent(ByVal sender As Object, ByVal e As EventArgs)
        ' Code for raising an event goes here.
    End RaiseEvent
End Event
```

参照

関連項目

Event ステートメント

AddHandler

RemoveHandler

その他の技術情報

Visual Basic におけるイベント

'AddHandler' メソッド、および 'RemoveHandler' メソッドのパラメータを、'ByRef' で宣言できません。

AddHandler メソッド、および **RemoveHandler** メソッドのパラメータは、**ByRef** 修飾子を付けて宣言できません。

Error ID: BC31134

このエラーを解決するには

- パラメータから **ByRef** キーワードを削除します。

参照

関連項目

Event ステートメント
AddHandler
RemoveHandler
ByRef

その他の技術情報

Visual Basic におけるイベント

明示的に型指定されているパラメータが存在する場合、すべてのパラメータが明示的に型指定されていなければなりません。

関数の中に異なる種類の型宣言が混在しています。

Error ID: BC30529

このエラーを解決するには

- すべてのパラメータの型を明示的に指定します。

参照

関連項目

Type

Visual Basic ソリューションのすべてのプロジェクトは、同じプラットフォームを対象にしなければなりませんが、追加しようとしているプロジェクトは、ソリューションで指定したプラットフォームとは異なるプラットフォームを対象にしています。

Visual Basic ソリューションのすべてのプロジェクトは同じプラットフォームを対象とする必要がありますが、追加しようとしているプロジェクトは、ソリューションで指定されているプラットフォーム以外を対象としています。

Error ID: BC30765

このエラーを解決するには

- ソリューションで指定されているプラットフォームを対象とするプロジェクトだけを追加します。

参照

処理手順

方法 : Visual Basic または Visual C# を使用してデバイス アプリケーションを作成する

パラメータ '<parametername>' と一致する引数は '<type1>' から '<type2>' へ下位変換します。

オーバーロードされたメソッドを呼び出そうとしましたが、コンパイラは、縮小変換を行わずに呼び出すことのできるメソッドを発見できません。縮小変換では、変換後のデータ型が変換前の値の一部を正確に格納できない可能性があります。

Error ID: BC30520

このエラーを解決するには

- **Option Strict Off** を指定します。

参照

関連項目

Option Strict ステートメント

概念

オーバーロードされたプロパティとメソッド

拡大変換と縮小変換

'<methodname>' のパラメータ '<parametername>' に対して引数が指定されていません。

必須のパラメータに対して引数が指定されていません。

Error ID: BC30455

このエラーを解決するには

- 指定されているパラメータに引数を指定します。

参照

処理手順

プロシージャのトラブルシューティング

概念

クラス メソッド

配列宣言で下限を指定することはできません。

配列は常に 0 の下限を持ちます。下限に 0 を指定すると、コードを読みやすくすることができます。ただし、下限に他の値を指定することはできません。

Error ID: BC30805

このエラーを解決するには

- 要素の合計数より 1 つ少ない数を配列の次元に設定します。たとえば、`Dim y(6)` は `Dim x(3 To 9)` と同じサイズ (7 要素) です。`Dim y(0 To 6)` のように指定することもできます。
- オフセットを使用して 0 以外の下限をシミュレートします。たとえば、次のコードは 3 ～ 9 に次元設定された配列をシミュレートしています。

```
Const offset As Integer = 3
Dim arrayIndex As Integer
' arrayIndex can vary between 3 and 9.
Dim y(0 To 6)
' The preceding statement allocates the same number of elements
' as Dim y(3 To 9).
y(arrayIndex - offset) = value
' The preceding statement converts arrayIndex to the
' corresponding index of y.
```

参照

処理手順

方法 : 配列の下限に 0 を指定する

概念

Visual Basic における配列の次元

その他の技術情報

Visual Basic における配列

ループ コントロール変数として宣言された配列を初期サイズで宣言することはできません

For Each ループが配列をその *element* 反復変数として使用していますが、その配列が初期化されています。

このエラーが発生する様子を次のステートメントに示します。

```
Dim arrayList As New List(Of Integer())
For Each listElement() As Integer In arrayList
For Each listElement(1) As Integer In arrayList
```

1 つ目の **For Each** ステートメントは、`arrayList` の要素に正しい方法でアクセスしています。2 つ目の **For Each** ステートメントはエラーを発生させます。

Error ID: BC32039

このエラーを解決するには

- *element* 反復変数の宣言から初期化の処理を削除します。

参照

関連項目

For...Next ステートメント (Visual Basic)

概念

Visual Basic の配列の概要

Visual Basic におけるコレクション

配列の上下限を、型指定子に記述できません。

配列のサイズをデータ型指定子の中で宣言することはできません。

Error ID: BC30638

このエラーを解決するには

- 型の後ではなく、変数名の直後で配列のサイズを指定します。次に例を示します。

```
Dim Array(8) As Integer
```

- 配列を定義し、必要な要素の数で初期化します。次に例を示します。

```
Dim Array2() As Integer = New Integer(8) {}
```

参照

処理手順

方法 : 配列変数を宣言する

概念

配列サイズの宣言 (Visual Basic 6.0 ユーザー向け)

その他の技術情報

Visual Basic における配列

型パラメータに使用される 'New' に引数を渡すことはできません。

宣言ステートメントまたは代入ステートメントは、ジェネリック型を呼び出し、New (Visual Basic) 制約がある型パラメータにコンストラクタ引数を渡します。

型パラメータの制約リストには、その型パラメータに渡す型引数が、作成元のコードがアクセスできるパラメータなしのコンストラクタを常に公開しなければならないという制約を指定できます。型パラメータでは、パラメータを受け取るコンストラクタの作成を強制することはできません。**New** 制約のある型パラメータは、そのような種類のコンストラクタを受け付けません。

Error ID: BC32085

このエラーを解決するには

- ジェネリック型を呼び出すステートメントから、型引数の後にある引数リストを削除します。コンストラクタ引数をそれらに対応する型パラメータに渡すことはできません。

参照

概念

Visual Basic におけるジェネリック型
値型と参照型

配列の次元を、負の値のサイズに設定することはできません。

配列の境界に負の数値が指定されています。負の添字を指定できるのは、上限に -1 を使用して空の配列を宣言するときだけです。空の配列に要素はありません。ただし、**Nothing** ではありません。

Error ID: BC30611

このエラーを解決するには

- 配列の境界の負の数値を正の数値に置き換えます。

参照

その他の技術情報

Visual Basic における配列

配列が、制限の 32 次元を超えています。

配列の次元数は 32 次元以内に制限されています。

Error ID: BC30052

このエラーを解決するには

- 配列宣言の中の次元数を減らします。

参照

その他の技術情報

Visual Basic における配列

配列初期化子は非定数の次元に指定することはできません。空の初期化子 '{}' を使用してください。

コンパイル時に不確定である次元が配列によって初期化されます。

このエラーが発生するコード例を次に示します。

```
Dim j As Integer
Dim intArray As Integer = New Integer(1, j) {{0, 100}, {1,101}}
```

このエラーを回避するコード例を次に示します。

```
Dim intArray As Integer = New Integer(1, j) {}
For i As Integer = 0 To j
    intArray(0, i) = i
    intArray(1, i) = 100 + i
Next i
```

Error ID: BC30949

このエラーを解決するには

- 可能な場合は、配列宣言に定数の次元を指定します。
- 定数の次元を指定できない場合は、定数ではない次元が確定される時点でループを使って配列を初期化する必要があります。

参照

処理手順

方法 : 配列変数を初期化する

方法 : 多次元配列を初期化する

関連項目

For Each...Next ステートメント (Visual Basic)

概念

Visual Basic の配列の概要

初期化子には <number> 個の要素が余分です。

配列初期化子に含まれる要素の数が多すぎます。

Error ID: BC30568

このエラーを解決するには

- **ReDim** を使って配列のサイズを変更します。

参照

概念

Visual Basic の配列の概要

配列初期化子の次元が少なすぎます。

初期化子の中で配列に対して指定されている次元の数が少なすぎます。

Error ID: BC30565

このエラーを解決するには

- 配列初期化子を調べて、必要な次元の数を確認します。
- 不足している引数を指定します。

参照

概念

Visual Basic の配列の概要
Visual Basic における多次元配列

配列初期化子の次元が多すぎます。

初期化子の中で配列に対して指定されている次元の数が多すぎます。

Error ID: BC30566

このエラーを解決するには

- 配列初期化子を調べて、必要な次元の数を確認します。
- 余分な引数を削除します。

参照

概念

Visual Basic の配列の概要
Visual Basic における多次元配列
Visual Basic におけるジャグ配列

配列初期化子の要素が <number> 個足りません。

初期化子の中で配列に対して指定されている要素の数が少なすぎます。

Error ID: BC30567

このエラーを解決するには

- 不足している引数を指定します。
- **ReDim** を使って配列のサイズを変更します。

参照

概念

Visual Basic の配列の概要
Visual Basic におけるジャグ配列

配列初期化子は配列に対してのみ有効ですが、'<variablename>' の型は '<typename>' です。

非配列変数を値のリストで初期化しようとしました。

Error ID: BC30679

このエラーを解決するには

- 変数を配列として宣言および初期化します。次に例を示します。

```
Dim intarray As Integer() = {1, 5, 9}
```

- 変数を単一の値として初期化します。次に例を示します。

```
Dim intvalue As Integer = 1
```

参照

関連項目

Dim ステートメント (Visual Basic)

概念

Visual Basic での変数宣言

その他の技術情報

Visual Basic における配列

配列の下限に指定できるのは '0' のみです。

宣言ステートメントまたは **New** 句に、下限が 0 ではない配列が指定されています。

配列変数を作成または初期化するとき、各次元の上限をオプションで指定できます。この場合、下限は常に 0 なので、その次元の長さは上限に 1 を足した数になります。下限もオプションで指定できますが、必ず 0 を指定する必要があります。これにはコードが読みやすくなるという利点があります。

Error ID: BC32059

このエラーを解決するには

- 下限の指定を 0 に変更するか、指定を削除します。

参照

処理手順

方法 : 配列の下限に 0 を指定する

その他の技術情報

Visual Basic における配列

変数とその型の両方で、配列修飾子を指定することはできません。

変数とその型の両方に対して配列修飾子があります。配列の配列を示しています。

Error ID: BC31087

このエラーを解決するには

- 型指定子から配列修飾子を削除します。

参照

その他の技術情報

Visual Basic における配列

配列インデックス式が見つかりません。

配列の初期化において、配列の範囲を定義する添字が 1 つ以上見つかりません。たとえば、ステートメントに `myArray (5,5,,10)` のような式が含まれているとします。この場合、3 番目の添字が抜けています。

Error ID: BC30306

このエラーを解決するには

- 不足しているインデックスを指定します。

参照

概念

Visual Basic の配列の概要

配列を 'New' で宣言することはできません。

New キーワードを指定できるのは、配列宣言の初期化部分だけです。つまり、**New** は等号 (=) の右側に記述する必要があります。これは、配列変数に代入する新しい配列型を作成できるようにするためです。

クラスの初期化に使用する方法は、配列には利用できません。次の 2 つのコード行は、いずれもクラスを元にしてオブジェクトを初期化するので、どちらも有効であり、かつ同じです。

```
Dim formA as Form = New Form
Dim formA as New Form
```

ただし、配列の初期化では、クラスの初期化と同じ方法を使用できません。

配列用の **New** 句では、かっこ **()** および中かっこ **{}** の両方を使用する必要があります。かっこで New の対象の型が配列であることを示し、中かっこで初期化用の値を指定します。中かっこが空の場合、つまり配列の値を初期化しない場合でも、コンパイラは中かっこを必要とします。

Error ID: BC30053

このエラーを解決するには

- `Dim myDates() As New Date` などのステートメントを `Dim myDates() As Date = New Date() {}` などのステートメントに置き換えます。

参照

処理手順

方法 : 配列変数を初期化する

その他の技術情報

Visual Basic における配列

構造体メンバとして宣言された配列を初期サイズで宣言することはできません。

構造体内の配列が初期サイズを指定して宣言されました。構造体要素を初期化できません。配列サイズの宣言は、初期化処理の一種です。

Error ID: BC31043

このエラーを解決するには

1. 構造体内の配列を、サイズの初期化が行われない動的配列として定義します。
2. 特定のサイズの配列が必要な場合は、コードの実行時に ReDim ステートメント (Visual Basic) を使用して、動的配列を再定義できます。次に例を示します。

```
Structure demoStruct
    Public demoArray() As Integer
End Structure
Sub useStruct()
    Dim struct As demoStruct
    ReDim struct.demoArray(9)
    Struct.demoArray(2) = 777
End Sub
```

参照

処理手順

方法 : 構造体を宣言する

その他の技術情報

Visual Basic における配列

型 'System.Void' の配列はこの式では使用できません。

代入ステートメントまたは宣言の式に、Void 型の配列が指定されています。

Void 構造体は、.NET Framework (特に Visual C# および Visual C++) で内部的に使用される特別な型です。これは値を返さないメソッドの、戻り値の型を表します。Visual Basic では、値を返さない場合は **Sub** プロシージャを使用し、値を返す場合は **Function** プロシージャを使います。

配列に **Void** 型を指定しても意味がありません。また、これはどのようなコンテキストでも許可されていません。

Error ID: BC31428

このエラーを解決するには

1. 配列を指定するかっこを削除します。
2. ランタイム型と **Void** を比較する特別な理由がない場合は、この型への参照自体を削除します。

参照

関連項目

Void

属性の引数として使用される配列は、すべての要素の値を明示的に指定する必要があります。

属性引数として使用する配列に、値が指定されていない要素があります。

Error ID: BC31110

このエラーを解決するには

- すべての要素に値を指定します。

参照

処理手順

配列のトラブルシューティング

その他の技術情報

Visual Basic における配列

'As Any' は、'Declare' ステートメントではサポートされていません。

Any データ型は、Visual Basic 6.0 とそれ以前のバージョンの **Declare** ステートメントで、任意のデータ型を受け取ることができる引数を宣言するため使用されていました。Visual Basic ではオーバーロードがサポートされているので、**Any** データ型を使用することはありません。

Error ID: BC30828

このエラーを解決するには

1. 使用する特定の型のパラメータを宣言します。次に例を示します。

   VB

   ```
   Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" ( _
       ByVal lpBuffer As String, _
       ByRef nSize As Integer) _
       As Integer
   ```

2. 呼び出しているプロシージャで **Void*** が必要とされている場合は、MarshalAsAttribute 属性を使用して **As Any** を指定します。

   VB

   ```
   Declare Sub SetData Lib "..\LIB\UnmgdLib.dll" ( _
       ByVal x As Short, _
       <System.Runtime.InteropServices.MarshalAsAttribute( _
           System.Runtime.InteropServices.UnmanagedType.AsAny)> _
           ByVal o As Object)
   ```

参照

処理手順

チュートリアル : Windows API の呼び出し

関連項目

Declare ステートメント

MarshalAsAttribute

概念

マネージ コードでのプロトタイプの作成

'As' が必要です。

宣言している 1 つ以上の要素に **As** 句が指定されていない宣言があります。

Error ID: BC30197

このエラーを解決するには

- 必要に応じて **As** 句を指定するか、**Option Strict Off** に変更します。

参照

関連項目

As (Visual Basic)
Option Strict ステートメント

'As'、コンマ、または ')' が必要です。

指定された型パラメータに **As**、コンマ (,)、または閉じかっこ ( )) がありません。

Error ID: BC32100

このエラーを解決するには

- **As**、コンマ、または閉じかっこを付けてください。

参照

処理手順

データ型のトラブルシューティング

概念

エラーの種類

パスの長さが 259 文字を超えるため、アセンブリ '<assemblyname>' を作成できません。

このアセンブリのパスの長さが、最大長である 259 文字を超えています。

Error ID: BC31518

このエラーを解決するには

- 短い名前を使用するか、浅いファイル階層を選択して、短いパスを作成します。

参照

概念

アセンブリ

アセンブリ '<filepath1>' は、'<filepath2>' (プロジェクト '<projectname1>' によって参照される) と '<filepath3>' (プロジェクト '<projectname2>' によって参照される) との間で不適切な、アセンブリ '<assemblyidentity>' を参照しています。'

エラー メッセージ
アセンブリ '<filepath1>' は、'<filepath2>' (プロジェクト '<projectname1>' によって参照される) と '<filepath3>' (プロジェクト '<projectname2>' によって参照される) との間で不適切な、アセンブリ '<assemblyidentity>' を参照しています。''<filepath2>' が使用されます。両方のアセンブリが同一である場合は、参照を同じ場所に変更してください。

アセンブリが別のアセンブリ内の型にアクセスしていますが、アクセス先のアセンブリに対するファイル参照が複数あります。

コンパイラは、別の場所にあるファイルに同じアセンブリの同じバージョンが含まれていることを保証できません。したがって、これらのファイル参照はあいまいですが、コンパイラはそこから 1 つを選択する必要があります。

アセンブリ ID には、アセンブリの名前、バージョン、公開キー (公開キーがある場合)、およびカルチャが含まれます。この情報によって、アセンブリは一意に識別されます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42204

このエラーを解決するには

1. 目的のアセンブリを最も適切に表現しているファイルを判別します。判断基準としては、バージョンの新しさ、ファイルのアクセシビリティ、更新の可能性などがあります。
2. このアセンブリに対するすべてのファイル参照を変更し、選択したファイルへのまったく同じファイル パスを使うようにします。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
概念
アセンブリ
アセンブリの概要
アセンブリの利点
プロジェクト参照
その他の技術情報
参照の管理

アセンブリ '<filepath1>' は、'<filepath2>' と '<filepath3>' との間で不適切なアセンブリ '<assemblyidentity>' を参照しています。

エラー メッセージ
アセンブリ '<filepath1>' は、'<filepath2>' と '<filepath3>' との間で不適切なアセンブリ '<assemblyidentity>' を参照しています。'<filepath2>' が使用されます。

アセンブリが別のアセンブリ内の型にアクセスしていますが、アクセス先のアセンブリに対するファイル参照が複数あります。

コンパイラは、それぞれの場所のファイルに同じアセンブリの同じバージョンが含まれていることを保証できません。したがって、これらのファイル参照はあいまいと見なされ、コンパイラはいずれか 1 つを選択する必要があります。

アセンブリ ID には、アセンブリの名前、バージョン、公開キー (公開キーがある場合)、およびカルチャが含まれます。この情報により、アセンブリを一意に識別します。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42205

このエラーを解決するには

1. 目的のアセンブリを最も適切に表現しているファイルを判別します。判断基準としては、バージョンの新しさ、ファイルのアクセシビリティ、更新の可能性などがあります。
2. このアセンブリに対するすべてのファイル参照を変更し、選択したファイルへのまったく同じファイル パスを使うようにします。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
概念
アセンブリ
アセンブリの概要
アセンブリの利点
プロジェクト参照
その他の技術情報
参照の管理

アセンブリ属性 '<attributename>' は、有効ではありません : <error>

共通言語ランタイムは、アセンブリの属性を受け入れません。

Error ID: BC30129

このエラーを解決するには

1. 属性を宣言から削除するか、またはコンテキストに対して有効な属性に置き換えます。
2. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。

参照

処理手順

方法 : 独自の属性を定義する

Assembly または Module 属性ステートメントは、ファイルの最初の宣言でなければなりません。

グローバル属性は、ソース ファイルの先頭で宣言する必要があります。つまり、**Option** ステートメントと **Imports** ステートメントを宣言したら、他のステートメントを宣言する前にグローバル属性を宣言してください。

Error ID: BC30637

このエラーを解決するには

- `<Module:>` や `<Assembly:>` などのグローバル属性をソース ファイルの先頭に配置します。

参照

概念

Visual Basic におけるグローバル属性

その他の技術情報

Visual Basic における属性

'Assembly' または 'Module' が必要です。

グローバル属性がアセンブリ全体に適用されるのか、現在のモジュールだけに適用されるのか示さずに指定されています。グローバル属性では **Assembly** または **Module** のいずれかを指定する必要があります。

グローバル属性とは、特定のプログラミング要素の宣言に適用されるのではなく、それ自体でソース行に記述される属性です。

Error ID: BC32015

このエラーを解決するには

1. 属性をグローバルにする場合は、属性ブロックの先頭に **Assembly** キーワードまたは **Module** キーワードを追加し、後ろにコロン (:) を記述します。
2. 属性をグローバルにしない場合は、属性ブロックを削除するか、プログラミング要素の宣言に移動します。

参照

関連項目

Assembly
Module (Visual Basic)

概念

属性の適用
Visual Basic におけるグローバル属性

このバイナリ演算子のパラメータのうち、少なくとも 1 つが型 '<typename>' を含んでいなければなりません。

二項演算子の定義で、その演算子が定義されているクラスまたは構造体の型とは異なる型が、両方のパラメータに指定されています。

演算子をクラスまたは構造体に定義するときには、少なくとも 1 つのパラメータはそのクラスまたは構造体の型にする必要があります。

Error ID: BC33021

このエラーを解決するには

- 1 つまたは両方のパラメータの型を、演算子が定義されているクラスまたは構造体の型に変更します。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

属性 '<attributename>' を複数回適用することはできません。

この属性は 1 回しか適用できません。属性を複数回適用できるかどうかは、**AttributeUsage** 属性によって決まります。

Error ID: BC30663

このエラーを解決するには

1. 属性が適用されているのが 1 回だけであることを確認します。
2. 独自に作成したカスタム属性を使用している場合は、複数の属性を使用できるよう、次に示す例のように **AttributeUsage** 属性を変更することを検討します。

```
<AttributeUsage(AllowMultiple := True)>
```

参照

関連項目

AttributeUsageAttribute Class

その他の技術情報

Visual Basic における属性

Visual Basic におけるカスタム属性

'<attributename>' を '<membername>' に適用できません。この属性はこの種類の宣言では有効ではありません。

使用している属性が項目と合っていません。

Error ID: BC30662

このエラーを解決するには

1. 使用している項目に合った属性を選択します。たとえば、メソッドを使用している場合は、メソッド用の属性を選択します。
2. 独自に作成したカスタム属性を使用している場合は、使用している項目の種類に合わせて **AttributeUsage** 属性を変更します。

参照

関連項目

AttributeUsageAttribute Class

その他の技術情報

Visual Basic における属性

Visual Basic におけるカスタム属性

属性 '<attributename>' を省略可能パラメータを含むメソッドに適用できません。

この属性を使用できるのは、必須引数を使用するメソッドまたは引数を使用しないメソッドだけです。

Error ID: BC30645

このエラーを解決するには

1. 省略可能なパラメータを持たないメソッドを定義します。
2. 省略可能なパラメータを持つメソッドで使用できる属性を使用します。
3. このコンテキストで使用できるカスタム属性を定義します。

参照

関連項目

AttributeUsageAttribute Class

その他の技術情報

Visual Basic における属性

属性 '<attributename>' をモジュールに適用できません。

AttributeUsageAttribute に **AttributeTargets.Module** が指定されていないモジュールに対して属性を適用しようとしました。この属性の宣言時に、モジュールに対して適用するよう定義されませんでした。

Error ID: BC30549

このエラーを解決するには

- 属性の宣言を確認して、**AttributeTargets.Module** または **AttributeTargets.All** を指定します。

参照

関連項目

AttributeUsageAttribute Members
AttributeTargets Enumeration

属性 '<attributename>' をアセンブリに適用できません。

AttributeUsageAttribute に **AttributeTargets.Assembly** が指定されていないアセンブリに対して属性を適用しようとしました。この属性の宣言時に、アセンブリに対して適用するように定義されませんでした。

Error ID: BC30548

このエラーを解決するには

- 属性の定義を確認して、**AttributeTargets.Assembly** または **AttributeTargets.All** を指定します。

参照

関連項目

AttributeUsageAttribute Members
AttributeTargets Enumeration

このプロジェクトでは、同じパラメータ値を使用する場合でも、属性 '<attributename>' を 2 回以上指定することはできません。

同じプログラミング要素に対して 1 つのカスタム属性が複数回指定されていますが、このカスタム属性に AttributeUsageAttribute が適用されており、さらに、この属性の AllowMultiple プロパティに **False** が設定されています。**AllowMultiple** は、その属性を複数回指定できるかどうかを制御します。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成 」を参照してください。

Error ID: BC41000

このエラーを解決するには

- 余分に指定されているカスタム属性を削除するか、カスタム属性に適用されている **AttributeUsageAttribute** で **AllowMultiple** を **True** に設定します。

参照

関連項目

AttributeUsageAttribute
AllowMultiple

概念

属性の適用

属性 '<attributename>' は、有効ではありません : <error>

共通言語ランタイムは、現在のコンテキスト内の属性を受け入れません。

Error ID: BC30127

このエラーを解決するには

1. 属性を宣言から削除するか、またはコンテキストに対して有効な属性に置き換えます。
2. エラー メッセージを確認し、適切なアクションを実行します。

参照

処理手順

方法 : 独自の属性を定義する

属性に Public コンストラクタがないため、この属性を使用できません。

使用している属性のコンストラクタが **Private** であるため、使用できません。

Error ID: BC30758

このエラーを解決するには

- 独自に開発したカスタム属性でこのエラーが表示される場合は、**SubNew** コンストラクタのアクセス修飾子を **Public** に変更します。

参照

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄

その他の技術情報

Visual Basic における属性

属性定数 '<constantname>' を代入式のターゲットにすることはできません。

属性内で宣言された定数に値を代入しようとしました。

Error ID: BC31510

このエラーを解決するには

- 間違っている代入を削除します。
- 独自に開発したカスタム属性を使用する場合は、属性内のメンバをフィールドとして再定義します。

参照

その他の技術情報

Visual Basic における属性
定数と列挙型 (Visual Basic)

属性コンストラクタには、型 '<typename>' の 'ByRef' パラメータが含まれます。属性を適用するには、byref パラメータを含むコンストラクタを使用できません。

属性がプログラミング要素に適用されていますが、**ByRef** パラメータを受け取る属性コンストラクタが使用されています。

属性はコンパイル時に適用されるため、コンパイラは属性のコンストラクタに渡すための具体的な値を必要とします。**ByRef** パラメータは値へのポインタを受け取りますが、これはコンパイル時には評価できません。

ByRef パラメータを受け取る属性コンストラクタを定義して、それを継承などの目的に使用できますが、属性を適用する場合は、**ByRef** パラメータを受け取らないコンストラクタを使用する必要があります。

Error ID: BC36006

このエラーを解決するには

- **ByRef** パラメータを受け取らないコンストラクタを使用して属性を適用するか、属性の適用自体を取りやめます。

参照

関連項目

ByRef

概念

Visual Basic における属性
属性の適用
引数の値渡しおよび参照渡し

Attribute コンストラクタに、'<type>' 型のパラメータが指定されていますが、Integral、Floating-point、または Enum 型のいずれでもないか、あるいは Char、String、Boolean、System.Type のいずれか、またはこれらの型の 1 次元配列ではありません。

カスタム属性定義に、パラメータについて無効なデータ型を指定するコンストラクタが含まれています。属性は、パラメータとして特定のデータ型しか使用できません。これは、それらのデータ型だけをアセンブリのメタデータ内でシリアル化できるためです。

Error ID: BC30045

このエラーを解決するには

- パラメータのデータ型を **Byte**、**Short**、**Integer**、**Long**、**Single**、**Double**、**Char**、**String**、**Boolean**、**System.Type** のいずれかまたは列挙型に変更します。

参照

その他の技術情報

Visual Basic におけるカスタム属性

属性メンバ '<membername>' は 'Public' と宣言されていないため、代入式のターゲットにすることはできません。

属性のプライベート メンバに値を代入しようとしました。

Error ID: BC31511

このエラーを解決するには

1. 代入を削除します。
2. 独自に作成したカスタム属性を使用する場合は、属性メンバのアクセス修飾子を **Public** に変更します。

参照

関連項目

Public (Visual Basic)

その他の技術情報

Visual Basic における属性

属性指定子は、完全なステートメントではありません。

エラー メッセージ
属性指定子は完全なステートメントではありません。属性を後続のステートメントに適用する場合は、行連結文字を使用します。

属性ブロックがソース コード行に単独で記述されています。属性は宣言ステートメントの先頭に記述する必要があります。また、ステートメントの一部である必要があります。

Error ID: BC32035

このエラーを解決するには

- 宣言ステートメントが後続の行にある場合は、属性の後ろにスペースとアンダースコア (_) をそれぞれ 1 つ追加して、ソース コード行を連結します。
- 属性ブロックに関連付けられている宣言ステートメントがない場合は、宣言ステートメントを指定するか、属性ブロックを削除します。
- 属性をアセンブリ全体または現在のアセンリ モジュールに適用する場合は、属性ブロックをソース コードの単独の行のままにします。属性名を山かっこ (< >) で囲んで、その前に **Assembly:** または **Module:** と記述します。属性ブロックの後ろにスペースやアンダースコアを追加しないでください。また、この属性ブロックはソース ファイルの先頭に記述するようにします。

参照

処理手順
方法 : コード内でステートメントを分割および連結する
概念
属性の適用

属性 'System.Runtime.InteropServices.DefaultCharSetAttribute' はこのバージョンではサポートされていません。

System.Runtime.InteropServices.DefaultCharSetAttribute 属性では、マーシャリングされた文字列で使用する文字セットを指定できます。この値には、System.Runtime.InteropServices.CharSet 列挙体のメンバを指定します。

現在のバージョンの Visual Basic では、この属性がサポートされていません。将来のバージョンではサポートされる可能性があります。

Error ID: BC32510

このエラーを解決するには

- Declare ステートメントを使用して、宣言する外部プロシージャに対して文字セットを指定します。次に例を示します。

```
Ansi Declare Function GetUserName Lib "advapi32.dll" _
    (ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
Unicode Declare Sub externalProc Lib "projectlib.dll" _
    (ByVal arg As Double)
```

  Declare ステートメント内で文字セットを指定しなかった場合は、既定で ANSI になります。

参照

関連項目

Declare ステートメント
DefaultCharSetAttribute
CharSet

その他の技術情報

Visual Basic における属性

ローカル変数に属性を適用することはできません。

ローカル変数には属性を適用できません。

Error ID: BC30660

このエラーを解決するには

- 変数を 1 つ上のレベルのスコープ (クラス レベルやモジュール レベルなど) で宣言します。

参照

関連項目

AttributeUsageAttribute Class

その他の技術情報

Visual Basic における属性

属性はジェネリックまたはジェネリック内に入れ子にされた型にすることができません。

属性が、ジェネリック型として宣言されているか、ジェネリック型の内部に宣言されています。

Visual Basic および .NET Framework は、現時点では属性とジェネリック型の組み合わせをサポートしていません。したがって、次のような制限があります。

- 属性をジェネリック型にすることや、ジェネリック型の内部で宣言することはできません。
- 属性がジェネリック クラスを継承することも、ジェネリック クラスが属性を継承することもできません。
- 属性を適用するときには、次のような引数は指定できません。
    - ジェネリック型
    - ジェネリック型から作成された型
    - 包含型の型パラメータ
    - 包含型の型パラメータから作成された型

Error ID: BC32067

このエラーを解決するには

- 属性の宣言に **Of** キーワードと型パラメータ リストが含まれる場合は、これらを削除します。
- 属性の宣言がジェネリック型の内部にある場合は、ジェネリック型の内部ではない場所に移動します。

参照

関連項目

Attribute

概念

Visual Basic における属性

Visual Basic におけるジェネリック型

不適切なチェックサム値です。16 進数ではないか、または奇数の 16 進数です。

チェックサムの値に無効な 16 進の数字が含まれるか、数字の数が奇数です。

ASP.NET は Visual Basic のソース ファイル (拡張子 .vb) を生成するとき、チェックサムを計算して `#externalchecksum` で指定された隠しソース ファイルに格納します。ユーザーが同じ目的で .vb ファイルを生成することもできますが、このプロセスは内部の機能に任せておくことをお勧めします。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42033

このエラーを解決するには

1. ASP.NET がこの Visual Basic ソース ファイルを生成している場合は、プロジェクトのビルドを再起動してください。
2. 再起動してもこの警告が消えない場合は、ASP.NET をインストールし直してもう一度ビルドしてください。
3. それでも警告が消えないか、または ASP.NET を使用していない場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト製品サポート サービスにご連絡ください。

参照

概念

ASP.NET の概要

その他の技術情報

製品のサポートとユーザー補助

GUID の不正な形式

指定された GUID の形式が正しくありません。

Error ID: BC42035

このエラーを解決するには

- GUID の形式をチェックします。

参照

概念

エラーの種類

クラス '<partialclassname>' に指定された基本クラス '<baseclassname1>' は、その他の partial 型の 1 つである基本クラス '<baseclassname2>' と異なることはできません

クラスが複数の部分宣言で定義されていて、その中に、複数の基本クラスを指定している複数の Inherits ステートメントが含まれています。

クラスを複数の部分宣言に分割して定義した場合、コンパイラはそのクラスをすべての部分宣言の結合体として扱います。これはメンバだけでなく、実装、継承、アクセス レベルに対しても当てはまります。

クラスは複数のインターフェイスを実装できますが、複数の基本クラスから継承することはできません。そのため、すべての **Inherits** ステートメントは同じ基本クラスを指定する必要があります。

Error ID: BC30928

このエラーを解決するには

- どのクラスを部分クラスの基本クラスにするかを決定し、それ以外の基本クラスを指定しているすべての **Inherits** ステートメントを部分宣言から削除します。

参照

関連項目

Partial (Visual Basic)
Inherits ステートメント

概念

継承の基本

配列の配列を初期化する場合、トップレベル配列に対してのみ境界を指定できます。

ジャグ配列 (配列の配列) の配列初期化で、いずれかの下位レベルの初期の長さが設定されています。配列宣言ステートメントでは、長さを指定できるのはトップレベルの配列だけです。

Error ID: BC32014

このエラーを解決するには

1. トップレベルの配列以外のすべての配列から、長さの指定を削除します。
2. **New** キーワードを使用して、後続の代入ステートメントで下位レベルの配列の初期の長さを設定します。

参照

処理手順

方法 : 配列変数を宣言する

関連項目

New (Visual Basic)

概念

Visual Basic におけるジャグ配列

かっこ付きの識別子の終わりに ']' がありません。

エスケープされた名前内の角かっこ ([]) は、対になっている必要があります。

Error ID: BC30034

このエラーを解決するには

- エスケープされた名前の最後に右角かっこを配置します。

参照

概念

宣言された要素の名前

'Finally' からの分岐は有効ではありません。

Finally ブロック内の **GoTo** ステートメントがブロックの外側に分岐しています。**Catch** ブロックまたは **Finally** ブロックの内外への分岐は無効です。

Error ID: BC30101

このエラーを解決するには

- **GoTo** ステートメントを削除し、条件判断制御構造またはループ制御構造を使用してプログラムの論理を実装することを検討してください。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)
GoTo ステートメント

その他の技術情報

Visual Basic における制御フロー

'ByVal' および 'ByRef' を組み合わせて使用することはできません。

ByVal キーワードと **ByRef** キーワードを一緒に使うことはできません。

Error ID: BC30641

このエラーを解決するには

- **ByVal** または **ByRef** のいずれか一方の修飾子だけを指定します。

参照

関連項目

ByVal
ByRef

'<type>' に変換できません。

無効な変換を試みました。

Error ID: BC30748

このエラーを解決するには

- 互換性のある型変換を行います。

参照

その他の技術情報

Visual Basic における型変換

型 '<typename1>' は型 '<typename2>' に変換できないため、'ByRef' パラメータ '<parametername>' の値を、一致する引数に戻してコピーすることはできません。

呼び出し元の引数の型に変換して戻すことができない型のパラメータが、プロシージャに宣言されています。

クラスまたは構造体を定義するとき、その型を他の型に変換するための 1 つ以上の変換演算子を定義できます。また、他の型から元のクラス型または構造体の型に戻すための、逆変換演算子も定義できます。このようなクラス型または構造体の型をプロシージャ呼び出しに使用すると、Visual Basic はこれらの変換演算子を使用して、引数の型を対応するパラメータの型に変換します。

引数を ByRef で渡した場合に、Visual Basic は参照を渡すのではなく、引数の値をプロシージャのローカル変数にコピーする場合があります。このようなケースでは、プロシージャから制御が戻ったとき、Visual Basic はローカル変数の値を呼び出し元のコードの引数にコピーして戻す必要があります。

ByRef の引数の値がプロシージャにコピーされるとき、引数とパラメータの型が同じであれば、変換は必要ありません。しかし、型が同じでない場合、Visual Basic は双方向に型を変換することが必要になります。どちらか一方が自分で定義したクラス型または構造体の型である場合、Visual Basic はそれを他の型と双方向に変換する必要があります。つまり、双方向の変換演算子を定義することが必要です。

Error ID: BC33037

このエラーを解決するには

- 可能であれば、呼び出し元の引数にプロシージャのパラメータと同じ型を使用します。そうすれば、Visual Basic は変換を行う必要がありません。
- パラメータの型と異なる型の引数を指定してプロシージャを呼び出す必要があるが、値を呼び出し元の引数に戻す必要がない場合は、パラメータを **ByRef** ではなく ByVal で定義します。
- 値を呼び出し元の引数に返す必要がある場合は、Visual Basic が呼び出し元の引数の型に戻すことができるように、逆変換演算子を定義します。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
演算子プロシージャ

モジュール '<modulename>' のインスタンスを作成できません。

モジュールは単一の共有インスタンスとしてだけ存在し、追加のインスタンスは作成できません。

Error ID: BC30166

このエラーを解決するには

- モジュールをクラスに変更するか、**New** 句でクラス名に置き換えます。

参照

関連項目

Module ステートメント

概念

Implements キーワードおよび Implements ステートメント

.NET Framework ディレクトリが見つかりません : <error>

.NET Framework アセンブリの格納先ディレクトリが見つかりませんでした。

Error ID: BC31508

このエラーを解決するには

1. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
2. エラーが再発する場合は、作業内容を保存し、Visual Studio を再起動します。
3. エラーが再発する場合は、Visual Basic を再インストールします。
4. 再インストール後もエラーが発生する場合は、マイクロソフト プロダクト サポート サービスまでご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

'<interfacename1>.<membername>' を実装できません。実装すると、型引数に実装された '<interfacename2>.<membername>' の実装と競合する可能性があります。

クラスが 2 つ以上のジェネリック インターフェイスを実装し、そのうちの 1 つが別のインターフェイスから継承されるので、同じインターフェイス メンバの 2 つの実装が、型引数の特定の値において衝突している可能性があります。

このエラーは次のようなコードで発生します。

```
Public Interface iFace1(Of t)
    Sub testSub()
End Interface
Public Interface iFace2(Of u)
    Inherits iFace1(Of u)
End Interface
Public Class testClass(Of y, z)
    Implements iFace1(Of y), iFace2(Of z)
    Public Sub testSuby() Implements iFace1(Of y).testSub
    End Sub
    Public Sub testSubz() Implements iFace1(Of z).testSub
    End Sub
End Class
```

`iFace2` は `iFace1` からその独自の型パラメータ (`u`) を使用して継承するので、`testClass` は、同じ型引数が `y` および `z` に渡された場合に、同一のシグネチャを持つ 2 つのバージョンの `iFace1.testSub` を実装します。この場合、どちらの形式にアクセスすればよいかがはっきりしなくなります。

Error ID: BC32125

このエラーを解決するには

- インターフェイスの継承構造を変更して、`iFace1` を 2 つの異なる型引数で実装できないようにします。

  または

- **Implements** ステートメントから、実装の衝突を起こしているインターフェイスの一方を削除します。

参照

関連項目

Class ステートメント (Visual Basic)
Interface ステートメント (Visual Basic)
Implements ステートメント

概念

Implements キーワードおよび Implements ステートメント
Visual Basic におけるジェネリック型

インターフェイス '<interfacename1>' を実装できません。実装すると、型引数に実装された別のインターフェイス '<interfacename2>' の実装と競合する可能性があります。

複数のインターフェイスを指定する **Implements** ステートメントがクラス宣言に含まれていますが、少なくとも 1 つのインターフェイスがジェネリックであり、実装の 2 つで型引数の特定の値が競合する可能性があります。

このエラーは次のようなコードで発生します。

```
Public Interface iFace1
    Sub testSub(ByVal arg As String)
End Interface
Public Interface iFace2(Of t)
    Sub testSub(ByVal arg As t)
End Interface
Public Class testClass
    Implements iFace1, iFace2(Of String)
End Class
```

`iFace2` が **String** を使って作成されるため、`testClass` はシグネチャの同じ 2 つのバージョンの `testSub` を実装する必要があります。これは、アクセスするバージョンをあいまいにします。

Error ID: BC32072

このエラーを解決するには

- ジェネリック インターフェイスに渡す型引数を変更して、競合が起こらないようにします。

  または

- **Implements** ステートメントからインターフェイスの 1 つを削除して、実装の競合が起こらないようにします。

参照

関連項目

Class ステートメント (Visual Basic)
Interface ステートメント (Visual Basic)
Implements ステートメント

概念

Implements キーワードおよび Implements ステートメント
Visual Basic におけるジェネリック型

いくつかの型引数のインターフェイス '<interfacename2>' と同一である可能性があるため、インターフェイス '<interfacename1>' を継承できません。

あるジェネリック インターフェイスは、別のジェネリック インターフェイスから 2 回以上継承しており、そのうちの 2 つの継承が型引数の特定の値において衝突している可能性があります。

このエラーは次のようなコードで発生します。

```
Public Interface interfaceA(Of u)

End Interface

Public Interface derivedInterface(Of t1, t2)

Inherits interfaceA(Of t1), interfaceA(Of t2)

End Interface
```

`derivedInterface` が、`t1` と `t2` の両方に同じ型を指定して作成または実装された場合、2 つの形式の `interfaceA` を同じ型の引数で継承せざるを得なくなります。そうした場合、どちらの形式にアクセスすればよいかがはっきりしなくなります。

Error ID: BC32120

このエラーを解決するには

- 派生したインターフェイスに指定する型引数の一方を変更して、衝突が起こらないようにします。

  または

- **Inherits** ステートメントから、継承または実装の衝突を起こしていると思われるインターフェイスの一方を削除します。

参照

関連項目

Interface ステートメント (Visual Basic)
Inherits ステートメント

概念

インターフェイスの概要
継承の基本
Visual Basic におけるジェネリック型

継承元のインターフェイス '<interfacename2>' がインターフェイス '<interfacename3>' と、いくつかの型引数において同一である可能性があるため、インターフェイス '<interfacename1>' を継承できません

ジェネリック インターフェイスが 2 つ以上のジェネリック インターフェイスを継承しており、そのうちの 2 つが型引数の特定の値において衝突している可能性があります。

このエラーは次のようなステートメントで発生します。

```
Public Interface interfaceA(Of u)
End Interface
Public Interface interfaceX(Of v)
    Inherits interfaceA(Of v)
End Interface
Public Interface derivedInterface(Of t1, t2)
    Inherits interfaceA(Of t1), interfaceX(Of t2)
End Interface
```

`derivedInterface` が、`t1` と `t2` の両方に同じ型を指定して作成または実装された場合、2 つの形式の `interfaceA` を同じ型の引数で継承する必要があります。そうした場合、どちらの形式にアクセスすればよいかがはっきりしなくなります。

Error ID: BC32123

このエラーを解決するには

- 派生したインターフェイスに指定する型引数の一方を変更して、衝突が起こらないようにします。

  または

- **Inherits** ステートメントから、継承または実装の衝突を発生させていると思われるインターフェイスの一方を削除します。

参照

関連項目

Interface ステートメント (Visual Basic)
Inherits ステートメント

概念

インターフェイスの概要
継承の基本
Visual Basic におけるジェネリック型

継承元のインターフェイス '<interfacename2>' がインターフェイス '<interfacename3>' と、いくつかの型引数において同一である可能性があるため、インターフェイス '<interfacename1>' を継承できません。

ジェネリック インターフェイスが 2 つ以上のジェネリック インターフェイスを継承しており、そのうちの 2 つが型引数の特定の値において衝突している可能性があります。

このエラーは次のようなステートメントで発生します。

```
Public Interface interfaceA(Of u)
    Inherits interfaceX(Of u)
End Interface
Public Interface interfaceX(Of v)
End Interface
Public Interface derivedInterface(Of t1, t2)
    Inherits interfaceA(Of t1), interfaceX(Of t2)
End Interface
```

`derivedInterface` が、`t1` と `t2` の両方に同じ型を指定して作成または実装された場合、2 つの形式の `interfaceX` を同じ型の引数で継承せざるを得なくなります。そうした場合、どちらの形式にアクセスすればよいかがはっきりしなくなります。

Error ID: BC32121

このエラーを解決するには

- 派生したインターフェイスに指定する型引数の一方を変更して、衝突が起こらないようにします。

  または

- **Inherits** ステートメントから、継承または実装の衝突を起こしていると思われるインターフェイスの一方を削除します。

参照

関連項目

Interface ステートメント (Visual Basic)
Inherits ステートメント

概念

インターフェイスの概要
継承の基本
Visual Basic におけるジェネリック型

継承元のインターフェイス '<interfacename2>' が、インターフェイス '<interfacename4>' の継承元であるインターフェイス '<interfacename3>' と、いくつかの型引数において同一である可能性があるため、インターフェイス '<interfacename1>' を継承できません。

ジェネリック インターフェイスが 2 つ以上のジェネリック インターフェイスを継承しており、そのうちの 2 つが型引数の特定の値において衝突している可能性があります。

このエラーは次のようなステートメントで発生します。

```
Public Interface interfaceA(Of u)
End Interface
Public Interface interfaceX(Of v)
    Inherits interfaceA(Of v)
End Interface
Public Interface interfaceY(Of w)
    Inherits interfaceA(Of w)
End Interface
Public Interface derivedInterface(Of t1, t2)
    Inherits interfaceX(Of t1), interfaceY(Of t2)
End Interface
```

`derivedInterface` が、`t1` と `t2` の両方に同じ型を指定して作成または実装された場合、2 つの形式の `interfaceA` を同じ型の引数で継承せざるを得なくなります。そうした場合、どちらの形式にアクセスすればよいかがはっきりしなくなります。

Error ID: BC32122

このエラーを解決するには

- 派生したインターフェイスに指定する型引数の一方を変更して、衝突が起こらないようにします。

  または

- **Inherits** ステートメントから、継承または実装の衝突を起こしていると思われるインターフェイスの一方を削除します。

参照

関連項目

Interface ステートメント (Visual Basic)
Inherits ステートメント

概念

インターフェイスの概要
継承の基本
Visual Basic におけるジェネリック型

'<name>' は、'System.MarshalByRefObject' を基本クラスとして持つクラス '<classname>' の値付きフィールド '<name>' のメンバであるため、参照できません。

System.MarshalByRefObject クラスを使用すると、リモート処理をサポートするアプリケーションは、アプリケーション ドメインの境界を越えてオブジェクトにアクセスできます。型がアプリケーション ドメインの境界を越えて使用される場合は、**MarshalByRejectObject** クラスから型を継承する必要があります。オブジェクトの状態はコピーしないでください。オブジェクトのメンバは、それらが作成されたアプリケーション ドメインの外部では使用できません。

Error ID: BC30310

このエラーを解決するには

1. 参照を確認して、参照されているメンバが有効であることを確認します。
2. **Me** キーワードを使用して、メンバを明示的に限定します。

参照

関連項目

Dim ステートメント (Visual Basic)
MarshalByRefObject

クラスの明示的なインスタンスを指定しないで、共有メソッドまたは共有メンバ初期化子内からクラスのインスタンス メンバへ参照することはできません。

共有プロシージャの内部から、クラスの非共有メンバを参照しようとしています。次のコードは、この状態を示す例です。

```
Class sample
    Public x as Integer
    Public Shared Sub setX()
        x = 10
    End Sub
End Class
```

この例では、`x = 10` の代入ステートメントにより、このエラー メッセージが表示されます。これは、共有プロシージャがインスタンス変数にアクセスしようとしているからです。

変数 `x` は、Shared (Visual Basic) として宣言されていないためインスタンス メンバです。`sample` クラスの各インスタンスに、独自の変数 `x` が個別に含まれています。1 つのインスタンスで `x` に値を設定したり、この値を変更したりしても、他の各インスタンスにある `x` の値は変わりません。

一方、`setX` プロシージャは、`sample` クラスの全インスタンスで **Shared** されています。これは、クラスの特定のインスタンスに関連付けられているわけではなく、操作がインスタンスごとに独立して行われることを意味します。特定のインスタンスとの関係がないため、`setX` はインスタンス変数にアクセスできません。**Shared** 変数のみを操作できます。`setX` で共有変数に値を設定するか、この値を変更した場合、このクラスのすべてのインスタンスで変更後の値が利用できます。

Error ID: BC30369

このエラーを解決するには

1. このメンバを、クラスのすべてのインスタンスで共有するのか、インスタンスごとに保持するのかを決定します。
2. メンバの実体を 1 つだけ作成して、すべてのインスタンスで共有する場合は、メンバ宣言に **Shared** キーワードを追加します。プロシージャ宣言に **Shared** キーワードを含めます。
3. インスタンスごとにメンバの実体を独自に作成する場合は、メンバ宣言に **Shared** を指定しないでください。プロシージャ宣言から **Shared** キーワードを削除します。

参照

関連項目

Shared (Visual Basic)

/win32icon および /win32resource を同時に指定することはできません。

/win32resource 処理と **/win32icon** 処理は、どちらも出力ファイルにアイコンの情報を挿入するため、この 2 つは同時に指定できません。

Error ID: BC2023

このエラーを解決するには

- 出力ファイルに .ico ファイルを挿入するには、**/win32icon** オプションのみを使用します。Windows エクスプローラでは、この .ico ファイルが出力ファイルを表します。

  または

- 出力ファイルに Win32 リソース ファイルを挿入するには、**/win32resource** オプションのみを使用します。Win32 リソースには、Windows エクスプローラでアプリケーションを識別するときに役立つ、バージョンやビットマップ (アイコン) の情報が含まれます。

参照

関連項目

/win32icon
/win32resource

スタックの上部にないメソッドのローカル変数の値を設定することはできません。

呼び出し履歴の最上位にある変数しか変更できません。たとえば、`procedure1` が `procedure2` を呼び出し、現在 `procedure1` にいる場合は、`procedure2` 内の変数を変更できません。

Error ID: BC30711

このエラーを解決するには

- 呼び出し履歴の最上位にある変数を変更します。

参照

その他の技術情報

Visual Studio でのデバッグ

'<filename>' を書き込むために開くことができません。

指定されたファイルを書き込むために開くことができません。おそらくファイルが既に開いているためと思われます。

Error ID: BC2012

このエラーを解決するには

1. ファイルを一度閉じてから開いてください。
2. ファイルのアクセス許可を確認してください。

参照

関連項目

My.Computer.FileSystem.WriteAllText メソッド
My.Computer.FileSystem.WriteAllBytes メソッド

その他の技術情報

Visual Basic でのファイルへの書き込み

'Case' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Case ステートメントはソース コードでしか使用できません。

Error ID: BC30719

このエラーを解決するには

- [イミディエイト] ウィンドウから **Case** ステートメントを削除します。

参照

その他の技術情報

Visual Studio でのデバッグ

'Case Else' は、'Select Case' ステートメント内でのみ使用できます。

Case Else ステートメントが **Select** ブロックの外側にあります。**Case Else** ステートメントは、**Select** ステートメント、または **Select Case** ステートメントと、対応する **End Select** ステートメントの間でのみ使用できます。

Error ID: BC30071

このエラーを解決するには

- **Case Else** ステートメントを削除するか、**Select** ブロック内に移動します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

'Case' を、同一の 'Select' ステートメント内で 'Case Else' の後に置くことはできません。

Case Else ステートメントでは、最初の **Case** で一致するものが見つからない場合に実行するステートメントを指定します。**Case** ステートメントが、同じ **Select** ブロック内で **Case Else** の後に見つかりました。

Error ID: BC30321

このエラーを解決するには

- **Case Else** ステートメントを、コード内で **Case** ステートメントの後ろの適切な位置に移動します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

'Case' は、'Select Case' ステートメント内でのみ使用できます。

Case ステートメントが **Select** ブロックの外側にあります。**Case** ステートメントは、**Select** ステートメント、または **Select Case** ステートメントと、対応する **End Select** ステートメントの間でのみ使用できます。

Error ID: BC30072

このエラーを解決するには

- **Case** ステートメントを削除するか、**Select** ブロック内に移動します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

ファイル '<filepath>' で、名前空間の名前 '<namespacename1>' の大文字と小文字の指定が、名前空間の名前 '<namespacename2>' の大文字と小文字の指定と一致しません。

同じ名前空間がプロジェクト内に複数回出現していますが、大文字と小文字の表記が異なっています。

大文字と小文字の区別は、プログラミング要素の名前に大文字と小文字が使用されることを指します。Visual Basic では大文字と小文字は区別されませんが、共通言語ランタイム (CLR: Common Language Runtime) では区別されます。詳細については、「宣言された要素の名前」の「名前の大文字と小文字の区別」を参照してください。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40055

このエラーを解決するには

- 念のため、名前空間を参照するときには大文字と小文字を正確に区別してください。共通言語ランタイムによる変換の誤りを防ぐことができます。

参照

関連項目

Namespace ステートメント

概念

Visual Basic における名前空間
宣言された要素の名前
Visual Basic の名前付け規則

'<name1>' は '<name2>' から継承するため、'Catch' ブロックは到達しませんでした。

コード内の **Catch**ブロックに到達することはありません。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42029

このエラーを解決するには

- 余分な **Catch** ブロックを削除します。

参照

処理手順

方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

'Catch' ブロック に到達しませんでした。<例外> 同じ 'Try' ステートメントで処理しました。

コードの **Catch** ブロックは、その前にある **Try** ブロックで処理されるため、到達されません。

既定では、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC42031

このエラーを解決するには

- 余分なステートメントを削除します。

参照

処理手順

方法 : Visual Basic で例外をキャッチする
方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する
方法 : Visual Basic で Catch ブロックを使用してエラーをフィルタ処理する
チュートリアル : 構造化例外処理

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

'Catch' を、'Try' ステートメント内にある 'Finally' の後に置くことはできません。

Finally によって **Try** ステートメント ブロックが終了された後に、**Catch** が記述されています。**Catch** は、**Try...Catch...Finally** ステートメント ブロック内で記述してください。

Error ID: BC30379

このエラーを解決するには

- **Catch** ステートメントをコード内のより適切な位置に移動します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

'Catch' を、'Try' ステートメントの外に置くことはできません。

Catch は **Try...Catch...Finally** ステートメント ブロック内でのみ使用する必要があります。**Try** ブロック内に不要な **Catch** ステートメントが存在します。または、対応する **Try** ブロックの境界の範囲外に **Catch** ステートメントが記述されています。

Error ID: BC30380

このエラーを解決するには

- 不要であれば **Catch** ステートメントを削除し、必要であれば **Try...Catch...Finally** ステートメント ブロック内に記述します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

型 '<typename>' は、'System.Exception'、または 'System.Exception' を継承するクラスではないため、'Catch' でキャッチできません。

Catch は例外だけを受け取ることができます。例外から派生していないものを受け取ろうとしました。

Error ID: BC30392

このエラーを解決するには

- **Catch** ステートメントを削除してください。または、**Catch** の対象を実際の例外に変更します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

'Catch' の終わりには、一致する 'End Try' を指定しなければなりません。

Catch ステートメントに対応する **End Try** ステートメントがありません。**Catch** ステートメントには、対応する **End Try** ステートメントが必要です。

Error ID: BC30441

このエラーを解決するには

1. **Catch** ステートメントを削除します。
2. **End Try** ステートメントを追加してブロックを終了させます。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

'Catch' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Catch ステートメントはソース コードでのみ使用できます。

Error ID: BC30715

このエラーを解決するには

- デバッグ コードから **Catch** ステートメントを削除します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

その他の技術情報

Visual Studio でのデバッグ

'Char' 値を '<typename>' に変換することはできません。

エラー メッセージ
'Char' 値を '<typename>' に変換できません。Microsoft.VisualBasic.AscW を使用して文字を Unicode 値として解釈するか、Microsoft.VisualBasic.Val を使用して文字を数字として解釈します。

式で、**Char** の値を **String** または **Object** 以外のデータ型に変換しようとしています。

Error ID: BC32006

このエラーを解決するには

- **AscW** 関数を使用して **Char** の値を Unicode 文字コードとして解釈するか、**Val** 関数を使用して数字として解釈します。

参照

関連項目
Asc 関数、AscW 関数
Val 関数
文字型 (Char) (Visual Basic)

文字定数には 1 文字のみを指定しなければなりません。

複数の文字を含む文字定数や null 文字 (すべてのビットがゼロ) の文字定数は無効です。

Error ID: BC30004

このエラーを解決するには

- 文字定数を 1 つの文字に変更するか、null 文字を有効な文字に変更します。

参照

関連項目

文字型 (Char) (Visual Basic)

文字が有効ではありません。

現在のコンテキストでは、この文字を使用できません。たとえば、宣言された要素名ではティルダ (~) や縦棒 (|) を使用できません。

Error ID: BC30037

このエラーを解決するには

1. プログラミング要素に余分な文字が誤って挿入されていないかどうかを確認します。
2. プログラミング要素が、その種類の要素について有効な文字だけを使用して記述または作成されているかどうかを確認します。

参照

概念

宣言された要素の名前

クラス '<classname>' にはアクセス可能な 'Sub New' が含まれていません。このクラスを継承できません。

クラスが Inherits ステートメントを使って基本クラスを指定していますが、その基本クラスのどのコンストラクタにもアクセスできません。

これは、目的の基本クラスにコンストラクタがない場合や、別のクラスからのアクセスを禁止するアクセス レベルがコンストラクタに指定されている場合などに起こります。

クラスを継承するとき、コンストラクタでは MyBase を使って基本クラスのコンストラクタを呼び出す必要があります。この呼び出しを作成しない場合、また明示的なコンストラクタも作成しない場合は、Visual Basic によって `MyBase.New()` を呼び出す暗黙のコンストラクタが自動的に作成されます。

Error ID: BC31399

このエラーを解決するには

1. 基本クラスのソース コードに手を加えることができる場合は、その少なくとも 1 つのコンストラクタのアクセス レベルを変更して、別のクラスからコンストラクタにアクセスできるようにします。

2. 基本クラスのコンストラクタのアクセス レベルを変更できない場合は、別のクラスから継承するか、または継承しないようにします。

参照

関連項目

Inherits ステートメント
MyBase
New (Visual Basic)

概念

継承の基本
Visual Basic でのアクセス レベル

クラス '<classname>' が見つかりません。

エラー メッセージ
'<classname>' クラスが見つかりません。このエラーは通常、'Microsoft.VisualBasic.dll' が適切でない場合に発生します。

定義済みのメンバが見つかりませんでした。

Error ID: BC31098

このエラーを解決するには

1. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
2. エラーが再発する場合は、作業内容を保存し、Visual Studio を再起動します。
3. エラーが再発する場合は、Visual Basic を再インストールします。
4. 再インストール後もエラーが発生する場合は、マイクロソフト プロダクト サポート サービスまでご連絡ください。

参照

その他の技術情報
製品のサポートとユーザー補助

クラス '<classname>' には既定のプロパティがないため、インデックス処理を実行できません。

クラスに対する既定のプロパティの指定に失敗しました。

Error ID: BC30367

このエラーを解決するには

- 宣言の最初に **Default** キーワードを指定することで、プロパティを既定のプロパティとして指定します。

参照

関連項目

Default (Visual Basic)

概念

既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)

クラス '<classname>' はそれ自体から継承することはできません: <メッセージ>

クラス定義の Inherits ステートメントが、それ自体のクラスを指定しています。

クラスを別のクラスから継承すると、継承したクラスのすべてのメンバが利用可能になるため、それらのメンバをもう一度定義する必要がありません。このようなクラスを派生クラスと呼び、継承されたクラスを基本クラスと呼びます。

自分のクラスのメンバは継承しなくてもすべて利用可能なので、それ自体を継承しても意味がありません。

Error ID: BC30257

このエラーを解決するには

1. **Inherits** ステートメントのクラス名のスペルを確認します。
2. 別のクラスを継承するつもりがないなら、**Inherits** ステートメント全体を削除してください。
3. 表示されるメッセージを調べて対処法を考えてください。

参照

その他の技術情報

Visual Basic の継承
クラスについて

クラス '<classname>' を作成できませんでした：<error>

クラスの初期化に失敗しました。.NET Framework のインストールが破損していることが原因である可能性があります。

Error ID: BC32400

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. .NET Framework を再インストールし、プログラムを再コンパイルします。
3. エラーが再発する場合は、Visual Basic コンパイラを再インストールします。
4. Visual Basic コンパイラを再インストールした後でもエラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

'クラス' 制約は、同じ型パラメータに対して複数回指定できません。

制約リストに複数の Class (Visual Basic) 制約が含まれています。

型パラメータに対する制約リストでは、型パラメータに渡される型引数を値型にするか (Structure (Visual Basic) 制約)、参照型にするか (**Class** 制約) を指定できます。同じ型パラメータに対して両方の制約を指定することはできません。また、これらを複数回指定することもできません。

Error ID: BC32101

このエラーを解決するには

- 冗長な **Class** キーワードを削除します。制約リスト内に 1 つだけになるようにします。

参照

概念

Visual Basic におけるジェネリック型
値型と参照型

'クラス' 制約と '構造体' 制約は、組み合わせて使用できません。

制約リストに Class (Visual Basic) 制約と Structure (Visual Basic) 制約の両方が含まれています。

型パラメータに対する制約リストでは、型パラメータに渡される型引数を値型にするか (**Structure** 制約)、参照型にするか (**Class** 制約) を指定できます。同じ型パラメータに対して両方の制約を指定することはできません。また、これらを複数回指定することもできません。

Error ID: BC32104

このエラーを解決するには

- 型引数を値型または参照型にする必要があるかどうかを判断します。
    - 型引数を値型にする場合は、制約リストから **Class** キーワードを削除します。
    - 型引数を参照型にする場合は、制約リストから **Structure** キーワードを削除します。

参照

概念

Visual Basic におけるジェネリック型
値型と参照型

'クラス' 制約と特定のクラス型は、組み合わせて使用できません。

制約リストに Class (Visual Basic) 制約と、定義済みのクラスの名前の両方が含まれています。

制約リストには、型パラメータに渡される型引数に対する要件を設定します。次の要件を任意に組み合わせて指定できます。

- 型引数は、1 つまたは複数のインターフェイスを実装する必要がある
- 型引数は、最大で 1 つのクラスを継承する必要がある
- 型引数は、作成しているコードからアクセス可能なパラメータなしのコンストラクタを公開する必要がある (**New** 制約を指定する)

制約リストに特定のクラスやインターフェイスを何も含めない場合は、次のいずれかを指定して汎用的な要件を設定できます。

- 型引数は値型であることが必要 (**Structure** 制約を指定する)
- 型引数は参照型であることが必要 (**Class** 制約を指定する)

同じ型パラメータに対して **Structure** と **Class** の両方を指定することはできません。また、これらを複数回指定することもできません。

Error ID: BC32107

このエラーを解決するには

- 型引数が任意の参照型であることを許可する場合は、クラス名を制約リストから削除します。
- 型引数が指定されたクラスを継承するように要件を指定する場合は、制約リストから **Class** キーワードを削除します。

参照

概念

Visual Basic におけるジェネリック型
値型と参照型

クラス '<classname1>' の基本クラス '<classname2>' には、引数なしで呼び出される、アクセス可能な 'Sub New' が 2 つ以上指定されているため、このクラスで 'Sub New' を宣言してください。

派生クラスでコンストラクタが宣言されていません。Visual Basic では、どの基本クラス コンストラクタを呼び出すのかがわからないため、コンストラクタを生成できません。

派生クラスでコンストラクタが宣言されない場合、Visual Basic は、**MyBase.New()** を呼び出す暗黙的なパラメータなしコンストラクタを生成しようとします。引数なしで呼び出せるアクセス可能なコンストラクタが基本クラスに存在しない場合、または複数存在する場合、Visual Basic では、暗黙的なコンストラクタを生成できません。

このような状況は、ある基本クラス コンストラクタが単一の **Optional** 引数を持ち、別の基本クラス コンストラクタが単一の **ParamArray** 引数を持っている場合などに発生します。これらのコンストラクタは引数なしで呼び出すことができます。

Error ID: BC32036

このエラーを解決するには

1. 派生クラス内に最低 1 つの **Sub New** コンストラクタを宣言および実装します。
2. すべての **Sub New** の 1 行目に、基本クラス コンストラクタ **MyBase.New()** の呼び出しを追加します。

参照

関連項目

コンストラクタとデストラクタの使用方法
Optional (Visual Basic)
ParamArray

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄
省略可能なパラメータ
パラメータ配列

クラス '<classname>' は、基本クラス '<baseclassname>' にある '<constructorname>' が古い形式に設定されているため、'Sub New' を宣言しなければなりません: '<errormessage>'

クラス宣言にコンストラクタが含まれていません。また、基本クラスのコンストラクタには、ObsoleteAttribute 属性とこれを警告として扱うディレクティブがマークされています。

派生クラスでコンストラクタが宣言されない場合、Visual Basic は、**MyBase.New()** を呼び出す暗黙的な、パラメータなしのコンストラクタを生成しようとします。基本クラス内に、引数を指定しないで呼び出すことができるアクセス可能なコンストラクタが存在しない場合、Visual Basic は暗黙のコンストラクタを生成できません。このケースでは、そのようなコンストラクタが **ObsoleteAttribute** 属性でマーク付けされているため、Visual Basic はこれを呼び出すことができません。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

既定では、**ObsoleteAttribute** の **IsError** プロパティが **False** であるため、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC41002

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. **Sub New** を使用して、コンストラクタを派生クラスに宣言します。

参照

概念

Visual Basic で使用される属性
属性の適用

クラス '<classname>' は、基本クラス '<baseclassname>' にある '<constructorname>' が古い形式に設定されているため、'Sub New' を宣言しなければなりません。

クラス宣言にコンストラクタが含まれていません。また、基本クラスのコンストラクタには、ObsoleteAttribute 属性とこれを警告として扱うディレクティブがマークされています。

派生クラスでコンストラクタが宣言されない場合、Visual Basic は、**MyBase.New()** を呼び出す暗黙のパラメータなしコンストラクタを生成しようとします。引数なしで呼び出せるアクセス可能なコンストラクタが基本クラスに存在しない場合、Visual Basic では、暗黙のコンストラクタを生成できません。このケースでは、そのようなコンストラクタが **ObsoleteAttribute** 属性でマーク付けされているため、Visual Basic はこれを呼び出すことができません。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

既定では、**ObsoleteAttribute** の **IsError** プロパティが **False** であるため、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC41001

このエラーを解決するには

- **Sub New** を使って、派生クラスにコンストラクタを宣言します。

参照

概念

Visual Basic で使用される属性
属性の適用

'<classname>' クラスは、'MustInherit' として宣言するか、次の継承 'MustOverride' メンバをオーバーライドしなければなりません : '<membername(s)>'

MustOverride メンバを持つ基本クラスから派生したクラスは、そのメンバをオーバーライドするか、または **MustInherit** 修飾子を使う必要があります。

Error ID: BC30610

このエラーを解決するには

- クラス定義に **MustInherit** 修飾子を追加します。
- **Overrides** キーワードを使ってオーバーライドを宣言します。

参照

関連項目

Overrides
MustInherit

その他の技術情報

Visual Basic の継承

クラス '<classname>' は、基本クラス '<baseclassname>' にある '<constructorname>' が古い形式に設定されているため、'Sub New' を宣言しなければなりません: '<errormessage>'

クラスにコンストラクタが宣言されておらず、それをエラーとして扱うように基本クラス コンストラクタが ObsoleteAttribute 属性とディレクティブでマーク付けされています。

派生クラスでコンストラクタが宣言されない場合、Visual Basic は、**MyBase.New()** を呼び出す暗黙的な、パラメータなしのコンストラクタを生成しようとします。基本クラス内に、引数を指定しないで呼び出すことができるアクセス可能なコンストラクタが存在しない場合、Visual Basic は暗黙のコンストラクタを生成できません。このケースでは、そのようなコンストラクタが **ObsoleteAttribute** 属性でマーク付けされているため、Visual Basic はこれを呼び出すことができません。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

Error ID: BC30918

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. **Sub New** を使用して、コンストラクタを派生クラスに宣言します。

参照

概念

Visual Basic で使用される属性
属性の適用

クラス '<classname>' は、基本クラス '<baseclassname>' にある '<constructorname>' が古い形式に設定されているため、'Sub New' を宣言しなければなりません。

クラスにコンストラクタが宣言されておらず、それをエラーとして扱うように基本クラス コンストラクタが ObsoleteAttribute 属性とディレクティブでマーク付けされています。

派生クラスでコンストラクタが宣言されない場合、Visual Basic は、**MyBase.New()** を呼び出す暗黙的な、パラメータなしのコンストラクタを生成しようとします。基本クラス内に、引数を指定しないで呼び出すことができるアクセス可能なコンストラクタが存在しない場合、Visual Basic は暗黙のコンストラクタを生成できません。このケースでは、そのようなコンストラクタが **ObsoleteAttribute** 属性でマーク付けされているため、Visual Basic はこれを呼び出すことができません。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

Error ID: BC30917

このエラーを解決するには

- **Sub New** を使用して、コンストラクタを派生クラスに宣言します。

参照

概念

Visual Basic で使用される属性
属性の適用

基本クラス '<classname2>' に、引数なしで呼び出せるアクセス可能な 'Sub New' が指定されていないため、クラス '<classname>' は、'Sub New' を宣言しなければなりません。

派生クラスがコンストラクタを宣言しておらず、呼び出し可能な基本クラス コンストラクタも存在しないため、Visual Basic がコンストラクタを生成できません。

派生クラスでコンストラクタが宣言されていない場合、Visual Basic は、**MyBase.New()** を呼び出す暗黙的なパラメータなしコンストラクタを生成しようとします。その基本クラスの中に、引数なしで呼び出されるアクセス可能なコンストラクタがない場合、または複数ある場合は、Visual Basic が暗黙的なコンストラクタを生成できません。

Error ID: BC30387

このエラーを解決するには

1. 派生クラス内で最低 1 つの **Sub New** コンストラクタを宣言および実装します。
2. すべての **Sub New** の 1 行目に、基本クラス コンストラクタの呼び出し (**MyBase.New()**) を追加します。

参照

関連項目

コンストラクタとデストラクタの使用方法
Optional (Visual Basic)
ParamArray

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄
省略可能なパラメータ
パラメータ配列

ジェネリック、またはジェネリック型に含まれるクラスは、属性クラスから継承できません

ジェネリック クラスまたはジェネリック型の中に入れ子になったクラスに、属性クラスを継承したという指定がなされています。

Visual Basic および .NET Framework は、現時点では属性とジェネリック型の組み合わせをサポートしていません。そのため、次のような制限があります。

- 属性をジェネリック型にしたり、ジェネリック型の中で宣言したりすることはできません。
- 属性がジェネリック クラスを継承したり、ジェネリック クラスが属性を継承したりすることはできません。
- 属性を適用するときには、次のような引数は指定できません。
    - ジェネリック型
    - ジェネリック型から作成された型
    - 包含型の型パラメータ
    - 包含型の型パラメータから作成された型

Error ID: BC32074

このエラーを解決するには

- 基本クラスを属性クラス以外の何かに変更するか、**Inherits** ステートメント自体を削除します。

参照

関連項目

Inherits ステートメント
Attribute

概念

Visual Basic における属性
Visual Basic におけるジェネリック型
継承の基本

クラスを '<specifier>' として宣言することはできません。

Class または **Module** の宣言で使われている指定子が無効です。

Error ID: BC30461

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Class ステートメント (Visual Basic)
Module ステートメント

クラスは、ほかのクラスからのみ継承できます。

Inherits ステートメントで、クラス以外の型が基本型として指定されています。

Error ID: BC30258

このエラーを解決するには

- 基本型を定義済みのクラスに変更します。

参照

概念

継承の基本

'Class_Terminate' イベントは現在サポートされていません。

エラー メッセージ
'Class_Terminate' イベントは現在サポートされていません。'Sub Finalize' を使用してリソースを解放してください。

以前のバージョンの Visual Basic の **Class_Terminate** イベントは、クラス デストラクタに置き換えられました。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42002

このエラーを解決するには

- **Sub** プロシージャを **Finalize** という名前で宣言してクラスを終了します。**Sub Finalize** は、インスタンスへのアクティブな参照が存在しなくなったことをガベージ コレクタが検出したときに呼び出されます。

参照

処理手順
方法 : Dispose Finalize パターンを実装する (Visual Basic)
関連項目
コンストラクタとデストラクタの使用方法
概念
クラス (Visual Basic 6.0 ユーザー向け)

'Class_Initialize' イベントはサポートされていません。

エラー メッセージ
'Class_Initialize' イベントはサポートされていません。Sub New メソッドを使用してクラスを初期化します。

以前のバージョンの Visual Basic の **Class_Initialize** イベントは、クラス コンストラクタに置き換えられました。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、
「Visual Basic での警告の構成 」を参照してください。

Error ID: BC42001

このエラーを解決するには

- **New** という名前の付いた **Sub** プロシージャを 1 つ以上定義してクラスを初期化します。**Sub New** は、クラス インスタンスが新規作成されたときに呼び出されます。

参照

関連項目
コンストラクタとデストラクタの使用方法
概念
Class_Initialize の変更点 (Visual Basic 6.0 ユーザー向け)
クラス (Visual Basic 6.0 ユーザー向け)

'Class' ステートメントの終わりには、対応する 'End Class' を指定しなければなりません。

Class は **Class** ブロックを開始するために使用します。したがって、ブロックの先頭にだけ記述でき、対応する **End Class** ステートメントでブロックの終了を示す必要があります。余分な **Class** ステートメントがあります。または、**Class** ブロックが **End Class** で終了していません。

Error ID: BC30481

このエラーを解決するには

- 必要のない **Class** ステートメントを探し、削除します。
- **Class** ブロックを対応する **End Class** で終了します。

参照

関連項目

End ステートメント

インターフェイス内のクラスを、'<specifier>' と宣言することはできません。

インターフェイス内で宣言したクラスに無効な修飾子が指定されました。

Error ID: BC31070

このエラーを解決するには

- **Public**、**Private**、**Shared**、**Friend**、**Protected**、**Protected Friend**、**Overridable** などの修飾子を削除します。

参照

関連項目

Class ステートメント (Visual Basic)

その他の技術情報

Visual Basic におけるインターフェイス

コード ページ '<name>' は無効か、インストールされていません。

指定されたコード ページが無効であるか、見つかりません。

Error ID: BC2016

このエラーを解決するには

- コードをチェックし、解決方法を個別のケースに応じて決定してください。

参照

その他の技術情報

デバッガのロードマップ

<type> '<name3>' の '<name2>' に対して暗黙的に宣言された '<name1>' と競合しています。

型メンバの名前が、暗黙的に作成された別のメンバの名前と競合しています。

Error ID: BC31058

このエラーを解決するには

- 明示的に宣言したメンバの名前を変更して、名前の競合を解決します。

参照

概念

Visual Basic の宣言ステートメント

既定プロパティと '|1' 上で定義された 'DefaultMemberAttribute' の間の競合が発生しました。

クラス、構造体、またはインターフェイスの既定のプロパティが Default (Visual Basic) キーワードを指定して宣言されていますが、別のメンバを既定のメンバとして指定する DefaultMemberAttribute も同時に適用されています。

型には既定のメンバを 1 つまで設定できます。既定のプロパティを宣言すると、Visual Basic は そのプロパティを宣言しているクラス、構造体、またはインターフェイスに **DefaultMemberAttribute** を自動的に適用します。

Error ID: BC32304

このエラーを解決するには

1. どのメンバをクラス、構造体、またはインターフェイスの既定のメンバにするかを決定します。
2. 衝突している宣言 (**Default** キーワードまたは **DefaultMemberAttribute**) を削除します。

参照

処理手順

方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)

関連項目

Default (Visual Basic)

DefaultMemberAttribute

概念

既定のプロパティ

コンパイルに失敗しました：<message>

エラー メッセージが示す理由によりコンパイルに失敗しました

Error ID: BC2010

このエラーを解決するには

- エラー メッセージを確認して、エラーの詳しい原因を調べてください。

参照

その他の技術情報

Compiling an Application

'Compare' が必要です。

Option ステートメントで **Text** か **Binary** が指定されていますが、**Compare** が指定されていません。

Error ID: BC30208

このエラーを解決するには

- **Compare** キーワードを **Option** ステートメントに追加します。たとえば、「`Option Compare Binary`」のように記述します。

参照

関連項目

Option Compare ステートメント

カンマ、')'、または有効な式の継続文字が必要です。

プロシージャ定義またはプロシージャ呼び出しの引数リストなどのリストで、そのメンバが正しく区切られていないか、正しく終了していません。考えられる原因の 1 つとして、2 つのリスト メンバを空白だけで区切られています。

Error ID: BC32017

このエラーを解決するには

- リスト メンバを区切るにはカンマ (,) を、リストを終了するには右かっこ ()) を使用します。

参照

概念

Sub プロシージャ
Function プロシージャ

コンマ、または ')' が必要です。(型引数リスト)

予期しない要素が、型引数リストに含まれています。このようなリストの内部ではコンマ (,) を使って型引数を区切り、閉じかっこ (**)**) を使ってリストを終了してください。

Error ID: BC32094

このエラーを解決するには

- コンマを追加して型引数リストを続けるか、または閉じかっこを追加してリストを完成させます。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

コンマ、または ')' が必要です。(型パラメータ リスト)

予期しない要素が、型パラメータ リストに含まれています。このようなリストの内部ではコンマ (,) を使って型パラメータを区切り、閉じかっこ (**)**) を使ってリストを終了してください。

Error ID: BC32099

このエラーを解決するには

- コンマを追加して型パラメータ リストを続けるか、または閉じかっこを追加してリストを完成させます。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

コンマ、または ')' が必要です (プロシージャ パラメータ リスト)。

予期しない要素が、パラメータ リストに含まれています。このようなリストの内部ではコンマ (,) を使ってパラメータを区切り、閉じかっこ (()) を使ってリストを終了してください。

Error ID: BC30213

このエラーを解決するには

- コンマを追加して型パラメータ リストを続けるか、または閉じかっこを追加してリストを完成させます。

参照

処理手順

方法 : プロシージャにパラメータを定義する

関連項目

パラメータの一覧

カンマが必要です。

引数リストで、2 つの引数の間にカンマ (,) がありません。

Error ID: BC30196

このエラーを解決するには

- すべての引数がカンマ (,) で区切られているかどうかを確認します。

参照

概念

プロシージャのパラメータと引数

Const 宣言に、配列初期化子を指定することはできません。

Const 宣言の中で配列初期化子を使おうとしました。

Error ID: BC30445

このエラーを解決するには

- 初期化子の前後の中かっこ ({}) を削除し、有効な式に変更します。

参照

関連項目

Const ステートメント (Visual Basic)

制約 '<constantname>' が、それ自体の値に依存することはできません。

コードに循環する依存関係があり、定数がそれ自体の値に依存しています。たとえば、`Const a = Const b; Const b = Const a` のようになっています。

Error ID: BC30500

このエラーを解決するには

- コード内で定数が評価される部分を確認し、適切に変更します。

参照

概念

定数の概要

定数を代入式のターゲットにすることはできません。

値を代入するコンテキストに定数があります。実行時に値を代入できるのは、書き込み可能な変数、プロパティ、および配列要素だけです。

Error ID: BC30074

このエラーを解決するには

- 書き込み可能な単独の変数、プロパティ、または配列要素に定数を置き換えます。

参照

概念

代入ステートメント

その他の技術情報

Visual Basic の定数と列挙体

定数式が必要です。

Const ステートメントで定数を適切に初期化していないか、または配列宣言で変数を使用して要素の数が指定されています。

Error ID: BC30059

このエラーを解決するには

1. 宣言が **Const** ステートメントである場合は、リテラル、前に宣言された定数、列挙型メンバ、または演算子で結合されたリテラル、定数、および列挙型メンバの組み合わせによって、定数が初期化されているかどうかを確認します。
2. 宣言で配列を指定している場合は、変数を使用して要素の数を指定しているかどうかを確認します。変数を使用して指定している場合は、変数を定数式に置き換えます。

参照

処理手順

方法 : 配列変数を宣言する

関連項目

Const ステートメント (Visual Basic)

定数式は、型 '<typename>' では表現できません。

対象の型に収まらない定数を評価しようとしました。通常は、型の範囲をオーバーフローしていることが原因です。

Error ID: BC30439

このエラーを解決するには

- 対象の型を評価する定数を扱うことのできるいずれかの型に変更します。

参照

概念

定数の概要

その他の技術情報

定数と列挙型 (Visual Basic)

定数は、class、structure、または array 型ではなく、組み込み型または列挙型でなければなりません。

定数を、クラス、構造体、配列型として、またはジェネリック型を用いて定義される型パラメータとして宣言しようとしました。

定数は、組み込み型
(**Boolean**、**Byte**、**Date**、**Decimal**、**Double**、**Integer**、**Long**、**Object**、**SByte**、**Short**、**Single**、**String**、**UInteger**、**ULong**、または **UShort**) か整数型の 1 つを基にした **Enum** 型である必要があります。

Error ID: BC30424

このエラーを解決するには

1. 定数を組み込み型または **Enum** 型として宣言します。
2. **True**、**False**、**Nothing** など特定の値を定数にすることもできます。コンパイラは、これら定義済みの値を、適切な組み込み型として解釈します。

参照

関連項目
データ型の概要 (Visual Basic)
概念
定数の概要
Visual Basic におけるデータ型
その他の技術情報
定数と列挙型 (Visual Basic)

定数には、値を指定しなければなりません。

値を代入せずに定数を宣言しました。

Error ID: BC30438

このエラーを解決するには

- 定数に値を代入します。

参照

概念

定数の概要

その他の技術情報

定数と列挙型 (Visual Basic)

制約 '<constraint1>' が、型パラメータ '<typeparametername>' に対して既に指定されている制約 '<constraint2>' と競合しています。

ジェネリック型が宣言されましたが、型パラメータの制約と矛盾が生じています。

このエラーは次のようなステートメントで発生することがあります。

```
Public Class testClass(Of t As {Structure, Class })
```

制約である `Structure` と `Class` によって、型パラメータ `t` に矛盾が生じています。その理由は、**Structure** 制約は対応する型引数が値型であることを要求していますが、**Class** は参照型であることを要求しているからです。

Error ID: BC32119

このエラーを解決するには

- 矛盾が生じないように、型パラメータの制約を変更します。

参照

関連項目

型リスト
Structure (Visual Basic)
Class (Visual Basic)

概念

Visual Basic におけるジェネリック型
値型と参照型

制約 '<constraint1>' は、型パラメータ制約 '<typeparameter1>' から取得された間接的な制約 '<constraint2>' と競合しています。

直接的な制約と間接的な制約の組み合わせにより、競合する制約を備えたジェネリック型が宣言されています。

このエラーは次のようなステートメントで発生することがあります。

```
Public Class testClass(Of t1 As {Structure, t2}, t2 As Class)
```

直接的な制約 `Structure` と間接的な制約 `Class` により、型パラメータ `t1` について競合が発生しています。**Structure** 制約は対応する型引数に値型を要求し、**Class** 制約は参照型を要求するからです。

Error ID: BC32110

このエラーを解決するには

- 制約の競合が生じないように、型パラメータの制約を変更します。

参照

関連項目

型リスト
Structure (Visual Basic)
Class (Visual Basic)

概念

Visual Basic におけるジェネリック型
値型と参照型

制約型 '<typename>' は、既にこの型パラメータに指定しています。

制約リストに特定のクラス制約またはインターフェイス制約が 2 回以上指定されています。

制約リストには、型パラメータに渡される型引数に対する要件を設定します。次の要件を任意に組み合わせて指定できます。

- 型引数は、1 つまたは複数のインターフェイスを実装する必要がある
- 型引数は、最大で 1 つのクラスを継承する必要がある

型に同じ型を 2 回以上実装または継承させることはできません。また、同じ制約リストに特定の型を 2 回以上指定できません。

Error ID: BC32071

このエラーを解決するには

- 同じクラスまたはインターフェイスが指定された冗長な部分を削除します。制約リストに 2 回以上指定することはできません。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

この型パラメータの制約は、'|1' のその他の partial 型の 1 つで宣言された、対応する型パラメータ上の制約と一致しません。

クラスまたは構造体を複数の宣言に分割して定義した場合、コンパイラはこのクラスまたは構造体を、それぞれの部分宣言のすべての結合体として扱います。そのため、それぞれの部分宣言では、競合する修飾子や型パラメータ リストを定義できません。

Error ID: BC30932

このエラーを解決するには

1. どちらの型パラメータ リストが、クラスまたは構造体に適した型パラメータ リストであるのかを判断します。各パラメータ、パラメータの順序、およびパラメータの制約リストも考慮します。
2. すべての部分宣言が同一の型パラメータ リストを使用していることを確認します。

参照

関連項目

Partial (Visual Basic)

概念

Visual Basic におけるジェネリック型

コンストラクタは、'<typename>' を含むアセンブリ '<assemblyname>' を間接的に参照しています。

エラー メッセージ
構成要素により、<typename> を含むアセンブリ '<assemblyname>' への間接的な参照が生成されます。<filename> へのファイル参照をプロジェクトに追加してください。

式にクラス、構造体、インターフェイス、列挙値、またはデリゲートなどの型が使用されていますが、アセンブリにその型を定義しているアセンブリへのプロジェクト参照がありません。

Error ID: BC31528

このエラーを解決するには

- プロジェクトのプロパティに、使用する型が定義されているアセンブリを含むファイルへの参照を追加します。

参照

処理手順
方法 : プロジェクト プロパティおよび構成設定を変更する
方法 : Visual Studio のリファレンスを追加または削除する
概念
同じ名前を持つ複数の変数がある場合に参照を解決する

コンストラクタは、'<typename>' を含むプロジェクト '<projectname>' を間接的に参照しています。

エラー メッセージ
コンストラクタは、'<typename>' を含むプロジェクト '<projectname>' を間接的に参照しています。プロジェクトに '<projectname>' へのプロジェクト参照を追加してください。

式またはステートメントが、別のプロジェクトで定義されている型にアクセスしますが、アクセス元のプロジェクトには、型を定義しているプロジェクトへの直接参照がありません。

型はクラス、構造体、インターフェイス、モジュール、または列挙体の可能性があります。

問題の型を定義するプロジェクトは、型を含むアセンブリを作成します。プロジェクトが、型を定義しているプロジェクトを直接参照しない場合、コンパイラは、その型を含むアセンブリがソリューション内にあり、アクセス可能であることを保証できません。

Error ID: BC31533

このエラーを解決するには

- 問題の型を定義しているプロジェクトを特定し、それに対するプロジェクト参照を追加します。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
概念
プロジェクト参照
その他の技術情報
名前空間およびコンポーネントの参照
参照の管理

コンストラクタ '<名前>' はそれ自体を呼び出すことはできません。

クラスまたは構造体の **Sub New** プロシージャがそれ自体を呼び出しています。

コンストラクタの目的は、クラスまたは構造体を最初に作成するとき、それを初期化することです。クラスまたは構造体には、複数のコンストラクタを作成できます。ただし、すべてのコンストラクタでパラメータ リストが同じでないことが必要です。コンストラクタから別のコンストラクタを呼び出して、それ自体の機能に加えて別の機能も実行することが可能です。しかし、コンストラクタが自分を呼び出すことには意味がなく、実際これが許可されると、無限再帰が起きてしまいます。

Error ID: BC30298

このエラーを解決するには

1. 呼び出されるコンストラクタのパラメータ リストを調べます。このパラメータ リストは、呼び出しを行っているコンストラクタのものと異なっている必要があります。
2. 別のコンストラクタを呼び出すつもりでない場合は、**Sub New** 呼び出し全体を削除してください。

参照

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄

コンストラクタの呼び出しは、インスタンス コンストラクタ内の最初のステートメントとしてのみ有効です。

New() の呼び出しが、コンストラクタの最初のステートメントの後で行われています。コンストラクタから別のコンストラクタを明示的に呼び出す場合は、**Sub New()** ステートメントに続く最初のステートメントで呼び出す必要があります。

Error ID: BC30282

このエラーを解決するには

- **New()** の呼び出しを削除するか、コンストラクタの先頭に移動します。

参照

関連項目

コンストラクタとデストラクタの使用方法

コンストラクタは、Function ではなく、Sub として宣言しなければなりません。

Function New を宣言しようとしました。コンストラクタは **Sub New** として宣言する必要があります。

Error ID: BC30493

このエラーを解決するには

- **Function** の代わりに **Sub** を使用します。

参照

関連項目

Sub ステートメント (Visual Basic)

コンストラクタは、インターフェイス メソッドを実装できません。

New キーワードを使用してインターフェイスのメソッドを実装しようとしました。この指定は無効です。

Error ID: BC30550

このエラーを解決するには

- **Implements** キーワードを使ってインターフェイスを実装します。

参照

関連項目

Implements ステートメント

概念

インターフェイスの概要

'Continue Do' は、'Do' ステートメント内でのみ使用できます。

Continue Do ステートメントは **Do...Loop** ループ内でのみ使用できます。

Error ID: BC30782

このエラーを解決するには

1. **Continue Do** ステートメントが **For...Next** ループ内にある場合は、このステートメントを **Continue For** に変更します。
2. **Continue Do** ステートメントが **While...End While** ループ内にある場合は、このステートメントを **Continue While** に変更します。
3. 上記以外の場合は、**Continue Do** ステートメントを削除します。

参照

関連項目

Continue ステートメント (Visual Basic)
Do...Loop ステートメント (Visual Basic)

'Continue For' は、'For' ステートメント内でのみ使用できます。

Continue For ステートメントは **For...Next** ループ内でのみ使用できます。

Error ID: BC30783

このエラーを解決するには

1. **Continue For** ステートメントが **Do...Loop** 内にある場合は、このステートメントを **Continue Do** に変更します。

   または

   Continue For ステートメントが **While...End While** ループ内にある場合は、このステートメントを **Continue While** に変更します。

2. 上記以外の場合は、**Continue For** ステートメントを削除します。

参照

関連項目

Continue ステートメント (Visual Basic)
For...Next ステートメント (Visual Basic)

'Continue' の後には、'Do'、'For'、または 'While' が必要です。

Continue ステートメントの後には、**Do**、**For**、**While** のいずれかを指定する必要があります。どれを指定するかは、**Continue** ステートメントが **Do...Loop** ループ、**For...Next** ループ、または **While...End While** ループのいずれで使用されているかによって決まります。

Error ID: BC30781

このエラーを解決するには

1. **Continue** ステートメントが **Do...Loop** ループ内にある場合は、このステートメントを **Continue Do** に変更します。
2. **Continue** ステートメントが **For...Next** ループ内にある場合は、このステートメントを **Continue For** に変更します。
3. **Continue** ステートメントが **While...End While** ループ内にある場合は、このステートメントを **Continue While** に変更します。
4. 上記以外の場合は、**Continue** ステートメントを削除します。

参照

関連項目

Continue ステートメント (Visual Basic)

'Continue' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Continue ステートメントはソース コードでのみ使用できます。

Error ID: BC30780

このエラーを解決するには

- デバッグ コードから **Continue** ステートメントを削除します。

参照

関連項目

Continue ステートメント (Visual Basic)

[イミディエイト ウィンドウ]

その他の技術情報

Visual Studio でのデバッグ

'Continue While' は、'While' ステートメント内でのみ使用できます。

Continue While ステートメントは **For...Next** ループ内でのみ使用できます。

Error ID: BC30784

このエラーを解決するには

1. **Continue While** ステートメントが **Do...Loop** ループ内にある場合は、このステートメントを **Continue Do** に変更します。
2. **Continue While** ステートメントが **For...Next** ループ内にある場合は、このステートメントを **Continue For** に変更します。
3. 上記以外の場合は、**Continue While** ステートメントを削除します。

参照

関連項目

Continue ステートメント (Visual Basic)
While...End While ステートメント (Visual Basic)

'<type1>' から '<type2>' への変換は定数式では発生しません。

Const ステートメント内の初期化式が、定数について宣言した型に変換できないデータ型に評価されます。

Error ID: BC30060

このエラーを解決するには

- 定数について宣言した型に変換できるデータ型の式を使用して、定数を初期化します。

参照

関連項目

Const ステートメント (Visual Basic)

その他の技術情報

Visual Basic における型変換

属性に対する引数として使用される '<type1>' から '<type2>' への変換は定数式では発生しません。

属性の引数に使用される式が、対応する属性のパラメータとは異なるデータ型に評価され、Visual Basic では属性の引数に必要な型変換を行うことができません。

属性はそれ自体に適用される要素のメタデータを提供するので、コンパイラはコンパイル時にすべてのメタデータを構成できる必要があります。このため、すべての属性はコンパイル時に不変の値を使用する必要があります。したがって、すべての属性の引数は、コンパイル時定数の値に評価される必要があります。

特定の型変換では、コンパイル時に不変の値を生成できません。たとえば、**String** から **Double** または **Date** (実行時のロケール設定に応じて変わる) への変換などがそうです。その他の変換でも、派生型の配列から **Object** の配列への変換などでは、さまざまな問題が伴うため、コンパイラは属性の引数に対するこれらの変換を処理できません。

Error ID: BC30934

このエラーを解決するには

- 対応するパラメータと同じデータ型に (属性で定義されたとおりに) 評価される式を使用します。

参照

関連項目

Const ステートメント (Visual Basic)

概念

属性の適用

その他の技術情報

Visual Basic における属性

Visual Basic における型変換

'Date' から 'Double' への変換には、'Date.ToOADate' メソッドの呼び出しが必要です。

Date 型の値を **Double** 型の値にキャストしようとしました。このキャストには、System.DateTime.ToOADate メソッドを使う必要があります。

Error ID: BC30532

このエラーを解決するには

- **System.DateTime.ToOADate** メソッドを使用して値を変換します。

参照

その他の技術情報

Visual Basic における型変換

'Double' から 'Date' への変換には、'Date.FromOADate' の呼び出しが必要です。

Date 型の値を **Double** 型の値にキャストしようとしました。このキャストには、System.DateTime.FromOADate(System.Double) メソッドを使う必要があります。

Error ID: BC30533

このエラーを解決するには

- **FromOADate** メソッドを使用して値を変換します。

参照

その他の技術情報

Visual Basic における型変換

変換演算子によって基本型から変換することはできません

パラメータ型を使って変換演算子が宣言されていますが、戻り値の型はこのパラメータ型から派生します。

Visual Basic は、コンパイルを実行するときに、参照型から継承階層内のすべての型 (つまり派生元の型または派生先の型) への変換が既に定義されていると見なします。このような変換は実行時に失敗する可能性がありますが、コンパイラは実行結果を予測できないので、どのような変換のコンパイルも許可されます。

コンパイラはこの変換が既に定義されているものと判断するため、変換を定義し直すことはできません。

Error ID: BC33030

このエラーを解決するには

- この演算子定義を完全に削除します。これは既に定義されています。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

変換演算子によって派生型から変換することはできません。

変換演算子が、戻り値の型を継承したパラメータ型で宣言されています。

Visual Basic はコンパイル時に、参照型からその継承階層にあるすべての型 (つまり、その型が派生した型と、その型から派生した型) への変換が、既に定義済みで存在すると判断します。このような変換は実行時に失敗する可能性がありますが、コンパイラは実行結果を予測できないので、これらの変換のコンパイルはすべて許可されます。

コンパイラはこの変換が既に定義されているものと判断するため、変換を定義し直すことはできません。

Error ID: BC33031

このエラーを解決するには

- この演算子定義を完全に削除します。これは既に定義されています。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

換演算子によって、ある型からその基本型に変換することはできません。

変換演算子の戻り値の型に、パラメータ型の派生元の型が宣言されています。

Visual Basic はコンパイル時に、参照型からその継承階層にあるすべての型 (つまり、その型が派生した型と、その型から派生した型) への変換が、既に定義済みで存在すると判断します。このような変換は実行時に失敗する可能性がありますが、コンパイラは実行結果を予測できないので、これらの変換のコンパイルはすべて許可されます。

コンパイラはこの変換が既に定義されているものと判断するため、変換を定義し直すことはできません。

Error ID: BC33026

このエラーを解決するには

- この演算子定義を完全に削除します。これは既に定義されています。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

変換演算子によって、ある型からその派生型に変換することはできません。

変換演算子の戻り値の型に、パラメータ型を派生した型が宣言されています。

Visual Basic はコンパイル時に、参照型からその継承階層にあるすべての型 (つまり、その型が派生した型と、その型から派生した型) への変換が、既に定義済みで存在すると判断します。このような変換は実行時に失敗する可能性がありますが、コンパイラは実行結果を予測できないので、これらの変換のコンパイルはすべて許可されます。

コンパイラはこの変換が既に定義されているものと判断するため、変換を定義し直すことはできません。

Error ID: BC33027

このエラーを解決するには

- この演算子定義を完全に削除します。これは既に定義されています。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

変換演算子によって、ある型から同じ型に変換することはできません。

変換演算子がパラメータと戻り値の両方に同じ型を指定して宣言されています。

同じ型に変換しても意味がありません。

Error ID: BC33024

このエラーを解決するには

- パラメータの型または戻り値の型を変更します。一方の型は、この演算子を定義したクラスまたは構造体の型であることが必要です。もう一方は、別の型である必要があります。
- パラメータの内容に対して変換を実行する必要がある場合は、Function ステートメント (Visual Basic) を使用して、演算子ではなく **Function** プロシージャを定義します。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

変換演算子によってインターフェイス型から変換することはできません。

変換演算子が、パラメータにインターフェイス型を指定して宣言されています。

Visual Basic はコンパイル時に、インターフェイスから参照型へのすべての変換が、既に定義済みで存在すると判断します。このような変換は実行時に失敗する可能性がありますが、コンパイラは実行結果を予測できないので、これらの変換のコンパイルはすべて許可されます。

コンパイラはこの変換が既に定義されているものと判断するため、変換を定義し直すことはできません。

Error ID: BC33029

このエラーを解決するには

- この演算子定義を完全に削除します。これは既に定義されています。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

変換演算子によって Object から変換することはできません。

変換演算子が、パラメータに オブジェクト型 (Object)を指定して宣言されています。

Visual Basic はコンパイル時に、参照型からその継承階層にあるすべての型 (つまり、その型が派生した型と、その型から派生した型) への変換が、既に定義済みで存在すると判断します。**Object** は .NET Framework の汎用データ型であるため、すべての型が **Object** から派生しています。

コンパイラはこの変換が既に定義されているものと判断するため、変換を定義し直すことはできません。

Error ID: BC33032

このエラーを解決するには

- この演算子定義を完全に削除します。これは既に定義されています。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

汎用データ型としてのオブジェクト型 (Object)

変換演算子によってインターフェイス型に変換することはできません。

変換演算子が、戻り値の型にインターフェイス型を指定して宣言されています。

Visual Basic はコンパイル時に、参照型からインターフェイスへのすべての変換が、既に定義済みで存在すると判断します。このような変換は実行時に失敗する可能性がありますが、コンパイラは実行結果を予測できないので、これらの変換のコンパイルはすべて許可されます。

コンパイラはこの変換が既に定義されているものと判断するため、変換を定義し直すことはできません。

Error ID: BC33025

このエラーを解決するには

- この演算子定義を完全に削除します。これは既に定義されています。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

変換演算子によって Object に変換することはできません。

変換演算子が、戻り値の型に オブジェクト型 (Object)を指定して宣言されています。

Visual Basic はコンパイル時に、参照型からその継承階層にあるすべての型 (つまり、その型が派生した型と、その型から派生した型) への変換が、既に定義済みで存在すると判断します。**Object** は .NET Framework の汎用データ型であるため、すべての型が **Object** から派生しています。

コンパイラはこの変換が既に定義されているものと判断するため、変換を定義し直すことはできません。

Error ID: BC33028

このエラーを解決するには

- この演算子定義を完全に削除します。これは既に定義されています。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

汎用データ型としてのオブジェクト型 (Object)

変換演算子は、'Widening' または 'Narrowing' のいずれかとして宣言されなければなりません。

Operator ステートメントで Widening または Narrowing が指定されていません。

変換演算子を定義する場合は、**Widening** または **Narrowing** のいずれかで宣言する必要があります。この 2 つは一度に 1 つしか選択できないため、両方を指定することはできません。

Error ID: BC33017

このエラーを解決するには

- 変換演算子を **Widening** と **Narrowing** のどちらにするかを決定し、**Operator** ステートメントに正しいキーワードを指定します。どちらか一方を指定する必要があります。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

拡大変換と縮小変換

演算子プロシージャ

'<typename1>' から '<typename2>' への変換は、明示的に行う必要があります。

この変換を暗黙的に行うことはできません。

Error ID: BC30664

このエラーを解決するには

- **CType** または特定の変換関数 (**CInt**、**CDec** など) を使います。

参照

関連項目

データ型変換関数

'ByRef' パラメータ '<parametername>' の値を一致する引数へ戻してコピーすると、型 '<typename1>' から型 '<typename2>' に下位変換します

対応するパラメータ型を拡張する引数を指定してプロシージャが呼び出されましたが、パラメータから引数への変換が縮小変換です。

クラスまたは構造体を定義するとき、その型を他の型に変換するための 1 つ以上の変換演算子を定義できます。また、他の型から元のクラス型または構造体の型に戻すための、逆変換演算子も定義できます。このようなクラス型または構造体の型をプロシージャ呼び出しに使用すると、Visual Basic はこれらの変換演算子を使用して、引数の型を対応するパラメータの型に変換します。

引数を ByRef で渡した場合に、Visual Basic は参照を渡すのではなく、引数の値をプロシージャのローカル変数にコピーする場合があります。このようなケースでは、プロシージャから制御が戻ったとき、Visual Basic はローカル変数の値を呼び出し元のコードの引数にコピーして戻す必要があります。

ByRef の引数の値がプロシージャにコピーされるとき、引数とパラメータの型が同じであれば、変換は必要ありません。しかし、型が同じでない場合、Visual Basic は双方向に型を変換することが必要になります。どちらか一方が自分で定義したクラス型または構造体の型である場合、Visual Basic はそれを他の型と双方向に変換する必要があります。一方が拡張変換であれば、その逆の変換は縮小変換になる可能性があります。

Error ID: BC32053

このエラーを解決するには

- 可能であれば、呼び出し元の引数にプロシージャのパラメータと同じ型を使用します。そうすれば、Visual Basic は変換を行う必要がありません。
- パラメータの型と異なる型の引数を指定してプロシージャを呼び出す必要があるが、値を呼び出し元の引数に戻す必要がない場合は、パラメータを **ByRef** ではなく ByVal で定義します。
- 呼び出し元の引数に値を戻す必要がある場合は、可能であれば逆変換演算子をWidening で定義します。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
演算子プロシージャ
拡大変換と縮小変換

その他の技術情報

Visual Basic における型変換

標準ライブラリ '<filename>' が見つかりませんでした。

Visual Basic は、コンパイルおよびリンクに必要な標準 DLL ライブラリの 1 つを見つけることができなかったか、開くことができませんでした。

多くの場合、使用できないライブラリは、mscorlib.dll または microsoft.visualbasic.dll です。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40049

このエラーを解決するには

1. エラー メッセージに示されたファイルが、Visual Basic を実行しているハード ディスク内に存在するかどうかを確認します。通常、標準ライブラリは、\WINNT\Microsoft.NET\Framework または \WINDOWS\Microsoft.NET\Framework のサブディレクトリにあります。
2. Visual Basic からの読み取りアクセスを禁止する設定または属性が、ファイルまたはディレクトリに適用されていないかどうかを確認します。
3. ファイル名と拡張子のスペルが正しいかどうかを確認します。大文字と小文字の違いは影響ありません。
4. ファイルの場所が正しく、アクセスできる状態にある場合、ディスク上でファイルが壊れている可能性があります。可能な場合は、Visual Basic を再インストールします。
5. 正確なファイル名と拡張子を記録し、マイクロソフト製品サポート サービスにお問い合わせください。

参照

処理手順

方法 : コマンド ライン コンパイラを起動する

概念

テクニカル サポート オプション (Visual Studio)

その他の技術情報

コマンド ラインからのビルド (Visual Basic)

ライブラリ '<libraryname>' が見つかりませんでした。

指定したライブラリが見つかりませんでした。

この問題は、多数のアセンブリを参照する応答ファイル (.rsp) を使用してコマンド プロンプトからコンパイルを行う場合に発生します。

Error ID: BC2017

このエラーを解決するには

- ライブラリ名が正しく指定されていることを確認します。

参照

その他の技術情報

デバッガのロードマップ

'D' は指数を表すためには使用できません。'E' を使用してください。

D 文字を使用して指数演算を示すことはできません。

Error ID: BC30827

このエラーを解決するには

- ^ 演算子または **E+** 文字を使用して、指数部があることを示します。たとえば、次のようにします。

```
Const Mole = 6.02E+23 ' Same as 6.02D23
Const Mole2 = 6.02 * 10 ^ 23 ' Same as 6.02D23
```

参照

関連項目

^ 演算子 (Visual Basic)

概念

数値データ型

'Custom' 修飾子は、明示的なデリゲート型なしで宣言されたイベントでは無効です。

非カスタム イベントとは異なり、**Custom Event**の宣言では、イベントのデリゲート型を明示的に指定する **As** 句をイベント名の後に指定する必要があります。

非カスタム イベントであれば、**As** 句と明示的なデリゲート型を指定するか、またはイベント名のすぐ後にパラメータ リストを指定することで定義できます。

Error ID: BC31122

このエラーを解決するには

1. カスタム イベントと同じパラメータ リストを指定してデリゲートを定義します。

   たとえば、**Custom Event**が `Custom Event Test(ByVal sender As Object, ByVal i As Integer)` で定義されている場合、対応するデリゲートの定義は次のようになります。

   VB

   ```
   Delegate Sub TestDelegate(ByVal sender As Object, ByVal i As Integer)
   ```

2. カスタム イベントのパラメータ リストを、デリゲート型を指定する **As** 句で書き換えます。

   先の例を使用すると、**Custom Event**の宣言は次のように書き換えられます。

   VB

   ```
   Custom Event Test As TestDelegate
   ```

使用例

次の例は、**Custom Event** を宣言して、必須である **As** 句をデリゲート型と一緒に指定します。

VB

```
Delegate Sub TestDelegate(ByVal sender As Object, ByVal i As Integer)
Custom Event Test As TestDelegate
    AddHandler(ByVal value As TestDelegate)
        ' Code for adding an event handler goes here.
    End AddHandler

    RemoveHandler(ByVal value As TestDelegate)
        ' Code for removing an event handler goes here.
    End RemoveHandler

    RaiseEvent(ByVal sender As Object, ByVal i As Integer)
        ' Code for raising an event goes here.
    End RaiseEvent
End Event
```

参照

関連項目

Custom
Event ステートメント
Delegate ステートメント

その他の技術情報

Visual Basic におけるイベント

'Custom' 修飾子は、インターフェイスで宣言されたイベントでは無効です。

カスタム イベントは、**AddHandler** メソッド、**RemoverHandler** メソッド、および **RaiseEvent** メソッドの実装を用意する必要があるため、インターフェイス内では宣言できません。

Custom キーワードは、カスタム イベントを実装する派生クラスで使用できます。

Error ID: BC31121

このエラーを解決するには

- **Custom** キーワードをインターフェイス内のイベント宣言から削除します。

使用例

次のコード例は、インターフェイス内に宣言されたイベントをカスタム イベントとして実装する方法を示しています。

VB

```
Interface TestInterface
    Delegate Sub TestDelegate(ByVal sender As Object, ByVal i As Integer)

    Event Test As TestDelegate
End Interface

Class TestClass
    Implements TestInterface

    Public Custom Event Test As TestInterface.TestDelegate _
    Implements TestInterface.Test
        AddHandler(ByVal value As TestInterface.TestDelegate)
            ' Code for adding an event handler goes here.
        End AddHandler

        RemoveHandler(ByVal value As TestInterface.TestDelegate)
            ' Code for removing an event handler goes here.
        End RemoveHandler

        RaiseEvent(ByVal sender As Object, ByVal i As Integer)
            ' Code for raising an event goes here.
        End RaiseEvent
    End Event
End Class
```

参照

関連項目

Custom
Event ステートメント
Delegate ステートメント
Class ステートメント (Visual Basic)
Interface ステートメント (Visual Basic)

その他の技術情報

Visual Basic におけるイベント

'Custom' 修飾子は 'Event' 宣言の直前にのみ使用することができます。

Custom キーワードは、カスタム イベント宣言の **Event** キーワードの前にのみ置かれます。

Error ID: BC31140

このエラーを解決するには

- **Custom** キーワードを **Event** キーワードの前に移動します。

  または

- **Custom** キーワードを削除します。

参照

関連項目

Custom

Event ステートメント

その他の技術情報

Visual Basic におけるイベント

'Custom Event' の終わりには、対応する 'End Event' を指定しなければなりません。

Custom Event 宣言の終わりには、対応する **End Event** ステートメントを指定してください。

Error ID: BC31114

このエラーを解決するには

- **Custom Event** 宣言が **End Event** ステートメントで終わっていることを確認します。

参照

関連項目

Event ステートメント

'Currency' は現在サポートされていません。'Decimal' 型を使用してください。

Currency データ型は **Decimal** データ型で置換されています。

Error ID: BC30803

このエラーを解決するには

- **Currency** データ型の代わりに **Decimal** データ型を使用します。

参照

関連項目

10 進型 (Decimal) (Visual Basic)

概念

データ型の変更点 (Visual Basic 6.0 ユーザー向け)

派生クラスで基本クラスのイベントを発生させることはできません。

イベントを発生させることができるのは、そのイベントが宣言されている宣言空間からだけです。したがって、クラスは、派生元のクラスを含め、ほかのクラスからイベントを発生させることはできません。

Error ID: BC30029

このエラーを解決するには

- **Event** ステートメントまたは **RaiseEvent** ステートメントを移動して、両方のステートメントが同じクラス内に存在するようにします。

参照

処理手順

方法 : 派生クラスを作成する

関連項目

Event ステートメント

RaiseEvent ステートメント

デリゲートでインターフェイス メソッドを実装することはできません。

デリゲートは、共有プロシージャまたはオブジェクトのインスタンス プロシージャを指す参照型です。参照先のプロシージャが代入によって変更される可能性があるため、**Delegate** ステートメントは **Handles** 句や **Implements** 句をサポートしていません。

Error ID: BC30018

このエラーを解決するには

- **Delegate** ステートメントから **Implements** 句を削除します。

参照

関連項目

Delegate ステートメント
Handles
Implements ステートメント

概念

デリゲートと AddressOf 演算子

デリゲートでイベントを処理することはできません。

デリゲートは、共有プロシージャまたはオブジェクトのインスタンス プロシージャを指す参照型です。参照先のプロシージャが代入によって変更される可能性があるため、**Delegate** ステートメントは **Handles** 句や **Implements** 句をサポートしていません。

Error ID: BC30019

このエラーを解決するには

- **Delegate** ステートメントから **Handles** 句を削除します。

参照

関連項目

Delegate ステートメント
Handles
Implements ステートメント

概念

デリゲートと AddressOf 演算子

イベント '<eventname>' のデリゲート型 '<delegatename>' は CLS に準拠していません。

Event ステートメント は、シグネチャを指定するためにデリゲートを使いますが、Delegate ステートメント には `<CLSCompliant(False)>` のマークが付けられているか、何もマークが付けられていません。

CLSCompliantAttribute 属性をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40050

このエラーを解決するには

- CLS に準拠する必要があり、デリゲートの定義を変更できる場合は、**CLSCompliantAttribute** をデリゲートの宣言に適用して `<CLSCompliant(True)>` のマークを付けます。
- デリゲートの定義を変更できない場合や、準拠のマークをデリゲートに付けられない場合は、**CLSCompliantAttribute** を **Event** ステートメントから削除するか、このステートメントに `<CLSCompliant(False)>` のマークを付けます。

参照

概念

CLS 準拠コードの記述

インターフェイス内のデリゲートを '<specifier>' と宣言することはできません。

インターフェイス内で宣言したデリゲートに無効な修飾子が指定されました。

Error ID: BC31068

このエラーを解決するには

- **Public**、**Private**、**Shared**、**Friend**、**Protected**、**Protected Friend**、**Overridable** などの修飾子を削除します。

参照

概念

インターフェイスの概要

その他の技術情報

Visual Basic におけるインターフェイス

デリゲートの実装に必要な、型 '<typename>' の Delegate コンストラクタが見つかりません。

型を含むアセンブリが破損しているか、無効です。

Error ID: BC31074

このエラーを解決するには

1. この型を含むアセンブリが有効であることを確認します。
2. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
3. エラーが再発する場合は、作業内容を保存し、Visual Studio を再起動します。

参照

その他の技術情報

製品のサポートとユーザー補助

Delegate クラス '<classname>' には Invoke メソッドが含まれていないため、この型の式をメソッド呼び出しのターゲットに設定することはできません。

Invoke がデリゲート クラスで実装されていないため、デリゲートを通じた **Invoke** の呼び出しに失敗しました。

Error ID: BC30220

このエラーを解決するには

1. デリゲート クラスのインスタンスが **Dim** ステートメントによって作成されており、プロシージャが **AddressOf** 演算子によってデリゲート インスタンスに代入されているかどうかを確認します。
2. デリゲート クラスを実装するコードを探し、そこで **Invoke** プロシージャが実装されているかどうかを確認します。

参照

関連項目

Delegate ステートメント
AddressOf 演算子
Dim ステートメント (Visual Basic)

その他の技術情報

Visual Basic でのデリゲート

'Optional' と宣言されていないパラメータに対して既定値を指定することはできません。

プロシージャを呼び出すときは、必要な引数を指定する必要があります。また、引数は宣言内の既定値を取りません。

Error ID: BC32024

このエラーを解決するには

- 引数を **Optional** として宣言するか、既定値を削除します。

参照

関連項目

Optional (Visual Basic)

概念

省略可能なパラメータ

Default プロパティ アクセスは、インターフェイス '<interfacename1>' の継承インターフェイス メンバ '<defaultpropertyname>' とインターフェイス '<interfacename2>' の '<defaultpropertyname>' との間で不適切です。

インターフェイスが、2 つのインターフェイスを継承しており、継承元の各インターフェイスで既定のプロパティが同じ名前で宣言されています。修飾子を付けないと、この既定のプロパティに対するアクセスをコンパイラが解決できません。次に例を示します。

```
Public Interface Iface1
    Default Property prop(ByVal arg As Integer) As Integer
End Interface
Public Interface Iface2
    Default Property prop(ByVal arg As Integer) As Integer
End Interface
Public Interface Iface3
    Inherits Iface1, Iface2
End Interface
Public Class testClass
    Public Sub accessDefaultProperty()
        Dim testObj As Iface3
        Dim testInt As Integer = testObj(1)
    End Sub
End Class
```

`testObj(1)` と指定した場合、コンパイラは既定のプロパティに解決しようとします。しかし、継承しているインターフェイスが原因で、既定のプロパティとして 2 つの可能性が生じます。その結果、コンパイラからこのエラーが示されます。

Error ID: BC30686

このエラーを解決するには

- 同じ名前のメンバを継承しないようにします。上の例で `testObj` が、たとえば `Iface2` のメンバである必要がない場合、次のように宣言します。

  ```
          Dim testObj As Iface1
  ```

  または

- 継承しているインターフェイスをクラス内で実装します。こうすると、継承したプロパティを別々の名前で実装できます。ただし、実装したクラスで既定のプロパティにすることができるのは、そのうちの 1 つだけです。次に例を示します。

  ```
  Public Class useIface3
      Implements Iface3
      Default Public Property prop1(ByVal arg As Integer) As Integer Implements Iface1.p
  rop
          ' Insert code to define Get and Set procedures for prop1.
      End Property
      Public Property prop2(ByVal arg As Integer) As Integer Implements Iface2.prop
          ' Insert code to define Get and Set procedures for prop2.
      End Property
  End Class
  ```

参照

概念

インターフェイスの概要

既定のプロパティ

既定プロパティ '<propertyname1>' は、'<classname>' の既定プロパティ '<propertyname2>' と競合しているため、'Shadows' と宣言できません。

基本クラスで定義されているプロパティと同じ名前のプロパティが宣言されています。この場合、このクラスのプロパティは基本クラスのプロパティをシャドウする必要があります。

このメッセージは警告です。**Shadows** が既定で使用されます。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC40007

このエラーを解決するには

- **Shadows** キーワードを宣言に追加するか、宣言されているプロパティの名前を変更します。

参照

関連項目

Shadows

概念

Visual Basic におけるシャドウ

既定のメンバ '<membername>' はプロパティではありません。

Default メンバを宣言しようとしました。**Default** キーワードは、既定のプロパティを指定するためのキーワードです。

Error ID: BC30555

このエラーを解決するには

- **Default** を使用してプロパティを指定しようとしていることを確認します。または、**Default** キーワードを削除します。

参照

関連項目

Default (Visual Basic)

'Default' を '<specifier>' と組み合わせて使用することはできません。

Default キーワードと **Shared**、**Private** などの指定子とを組み合わせようとしましたが、このコンテキストでは無効です。

Error ID: BC30490

このエラーを解決するには

- **Default** を削除します。

  または

- 矛盾する指定子を削除します。

参照

関連項目

Default (Visual Basic)

'Default' は、<type> 内で 1 つのプロパティにのみ適用できます。

クラス、インターフェイス、または構造体の複数のプロパティ名に **Default** を適用しようとしました。

Error ID: BC30359

このエラーを解決するには

- **Default** 宣言を 1 つだけにします。

参照

関連項目

Default (Visual Basic)

構造内の 'Declare' ステートメントを '<specifier>' として宣言できません。

Declare 宣言に、**Structure** 宣言では有効ではない指定子があります。

Error ID: BC30791

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Delegate ステートメント
Structure ステートメント

モジュール内の Declare ステートメントを '<specifier>' として宣言できません。

Declare 宣言に、**Module** 宣言では有効ではない指定子があります。モジュールはインスタンス化できず、継承をサポートしません。また、インターフェイスを実装できません。

Error ID: BC30786

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Delegate ステートメント
Module ステートメント

'Declare' ステートメントは、ジェネリック型またはジェネリック型に含まれる型では使用できません。

Declare ステートメントが、ジェネリック クラスまたは構造体の一部に含まれているか、ジェネリック クラスまたは構造体の中で宣言されているクラスまたは構造体の一部に含まれています。

Visual Basic および .NET Framework は、現時点では外部参照とジェネリック型の組み合わせをサポートしていません。コンパイラに外部プロシージャを正しく呼び出させるためには、すべてのパラメータと戻り値の型を指定する必要があります。

Error ID: BC32075

このエラーを解決するには

- **Declare** ステートメントをジェネリック型のスコープ外に移動させるか、完全に削除します。

参照

関連項目

Declare ステートメント

概念

Visual Basic におけるジェネリック型

宣言が必要です。

代入ステートメントやループ ステートメントなどの非宣言ステートメントが、プロシージャの外側にあります。プロシージャの外側に記述できるのは宣言だけです。

または、プログラミング要素が、**Dim** や **Const** などの宣言キーワードを使用せずに宣言されています。

Error ID: BC30188

このエラーを解決するには

- 非宣言ステートメントをプロシージャの本体に移動します。
- 適切な宣言キーワードを使用して宣言を開始します。
- 宣言キーワードのスペルが正しいかどうかを確認します。

参照

関連項目

Dim ステートメント (Visual Basic)

概念

Visual Basic におけるプロシージャ

日付定数が有効ではありません。

日付定数が無効な日付で宣言されています。

Error ID: BC31085

このエラーを解決するには

- 定数で使用した日付を訂正します。

参照

関連項目

日付型 (Date) (Visual Basic)

その他の技術情報

定数と列挙型 (Visual Basic)

'Do' の終わりには、対応する 'Loop' を指定しなければなりません。

Do ステートメントに対応する **Loop** ステートメントがありません。**Loop**ステートメントは、**Do** ループの最後に使用する必要があります。

Error ID: BC30083

このエラーを解決するには

- この **Do** ループが、入れ子になったループの一部である場合は、すべてのループが正しく終了していることを確認します。
- **Do** ループの最後に **Loop** ステートメントを追加します。

参照

関連項目

Do...Loop ステートメント (Visual Basic)

この式の評価中に 0 で除算しました。

この式の計算において、数値をゼロで除算しようとしました。

Error ID: BC30542

このエラーを解決するには

- 定数と変数の値を調べて、式の分母にゼロが発生した原因を確認します。

参照

関連項目

代入演算子

'ElseIf' の前には、対応する 'If' または 'ElseIf' を指定しなければなりません。

ElseIf ステートメントに対応する **If** ステートメントがありません。**ElseIf** ステートメントの前には、対応する **If** ステートメントまたは別の **ElseIf** ステートメントを指定してください。

Error ID: BC36005

このエラーを解決するには

1. この **If** ブロックが、入れ子になった制御構造の一部である場合は、すべての制御構造が正しく終了していることを確認します。
2. **If** ブロックに入れ子になっている他の制御構造がそれぞれ正しく終了していることを確認します。
3. **If** ブロックの書式が正しいことを確認します。

参照

関連項目

If...Then...Else ステートメント (Visual Basic)

概念

条件判断構造

'Else' の前には、対応する 'If' または 'ElseIf' を指定しなければなりません。

Else ステートメントに対応する **If** ステートメントがありません。**Else** ステートメントの前には、対応する **If** ステートメントを指定してください。

Error ID: BC30086

このエラーを解決するには

1. この **If** ブロックが、入れ子になった **If** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
2. **If** ブロック内の他の制御構造がそれぞれ正しく終了していることを確認します。
3. **If** ブロックの書式が正しいことを確認します。

参照

関連項目

If...Then...Else ステートメント (Visual Basic)

この変換演算子のパラメータ型または戻り値の型は、含んでいる型でなければなりません。

変換演算子が、パラメータの型と戻り値の型の両方に、その演算子が定義されたクラスまたは構造体以外の型を指定して定義されています。

クラスまたは構造体に変換演算子を定義する場合は、そのクラスまたは構造体の型に変換するか、またはその型から変換する必要があります。

Error ID: BC33022

このエラーを解決するには

- パラメータ型または戻り値の型を、演算子が定義されているクラスまたは構造体の型に変換します。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

Win32 リソースの作成中にエラーが発生しました：<error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、インメモリ リソースの作成エラーを報告しています。環境に問題があるか、またはコンピュータのメモリが不足している可能性があります。

Error ID: BC30136

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べてください。詳細とアドバイスについては、Al.exe ツールのエラーと警告を参照してください。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト製品サポート サービスにご連絡ください。

参照

関連項目

アセンブリ リンカ (Al.exe)
Al.exe ツールのエラーと警告

その他の技術情報

製品のサポートとユーザー補助

アセンブリ マニフェストを作成中にエラーが発生しました：<error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、アセンブリの生成前の段階でのエラーを報告しています。

指定したキー ファイルまたはキー コンテナに原因がある場合があります。アセンブリに完全に署名を行うには、公開キーと秘密キーに関する情報を含む有効なキー ファイルを指定する必要があります。アセンブリに遅延署名を行うには、[遅延署名のみ] チェック ボックスをオンにして、公開キーの情報を含む有効なキー ファイルを指定する必要があります。アセンブリに遅延署名を行う場合は、秘密キーは必要ありません。詳細については、「方法 : アセンブリに署名する (Visual Studio)」を参照してください。

Error ID: BC30140

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べてください。詳細とアドバイスについては、「Al.exe ツールのエラーと警告」のエラー AL1019 に関する説明を参照してください。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト製品サポート サービスにご連絡ください。

参照

処理手順

方法 : アセンブリに署名する (Visual Studio)

関連項目

[署名] ページ (プロジェクト デザイナ)
アセンブリ リンカ (Al.exe)
Al.exe ツールのエラーと警告

その他の技術情報

製品のサポートとユーザー補助

Enums は整数型として宣言されなければなりません。

列挙型で使用できる型は **SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、および **ULong** のみです。他のデータ型は使用できません。

Error ID: BC30650

このエラーを解決するには

- **SByte**、**Byte**、**Short**、**UShort**、**Integer**、**UInteger**、**Long**、または **ULong** データ型を指定します。

参照

関連項目

データ型の概要 (Visual Basic)
Enum ステートメント (Visual Basic)

'Enum' の終わりには、対応する 'End Enum' を指定しなければなりません。

Enum ステートメントに対応する **End Enum** ステートメントがありません。列挙の最後には **End Enum** ステートメントが必要です。

Error ID: BC30185

このエラーを解決するには

1. **Enum** メンバの書式が正しいかどうかを確認します。
2. 列挙の最後に **End Enum** ステートメントを追加します。

参照

関連項目

Enum ステートメント (Visual Basic)

インターフェイス内の Enum を、'<specifier>' と宣言することはできません。

インターフェイス内で宣言した列挙型に無効な修飾子が指定されました。

Error ID: BC31069

このエラーを解決するには

- **Public**、**Private**、**Shared**、**Friend**、**Protected**、**Protected Friend**、**Overridable** などの修飾子を削除します。

参照

関連項目

Enum ステートメント (Visual Basic)

その他の技術情報

Visual Basic におけるインターフェイス

'Enum <enumname>' は少なくとも 1 つのメンバを含んでいる必要があります。

空の列挙体があります。**Enum** ステートメントと **End Enum** ステートメントの間で少なくとも 1 つのメンバを宣言する必要があります。

Error ID: BC30280

このエラーを解決するには

- 列挙体にメンバ宣言を追加します。

参照

関連項目

Enum ステートメント (Visual Basic)

'EndIf' ステートメントはサポートされていません。代わりに 'End If を使用してください。

EndIf ステートメントはサポートされなくなりました。

Error ID: BC30826

このエラーを解決するには

- **If** ステートメントを終了するには、**End If** を使用します。

参照

関連項目

If...Then...Else ステートメント (Visual Basic)

'End With' の前には、対応する 'With' を指定しなければなりません。

End With ステートメントに対応する **With** ステートメントがありません。**End With** ステートメントの前には、対応する **With** ステートメントを指定してください。

Error ID: BC30093

このエラーを解決するには

1. この **With** ブロックが、入れ子になった **With** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
2. **With** ブロック内の他の制御構造がそれぞれ正しく終了していることを確認します。
3. **With** ブロックの書式が正しいことを確認します。

参照

関連項目

With...End With ステートメント (Visual Basic)

'End While' の前には、対応する 'While' を指定しなければなりません。

End While ステートメントに対応する **While** ステートメントがありません。**End While** ステートメントの前には、対応する **While** ステートメントを指定してください。

Error ID: BC30090

このエラーを解決するには

1. この **While** ブロックが、入れ子になった **While** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
2. **While** ブロック内の他の制御構造がそれぞれ正しく終了していることを確認します。
3. **While** ブロックの書式が正しいことを確認します。

参照

関連項目

While...End While ステートメント (Visual Basic)

'End Using' の前には、対応する 'Using' を指定しなければなりません。

End Using ステートメントより前に、対応する **Using** の宣言がありません。

Error ID: BC36007

このエラーを解決するには

- **End Using** ステートメントが余分な場合は削除します。
- Using ステートメント (Visual Basic) がない場合は指定します。
- **End Using** ステートメントをコード内の適切な位置に移動します。

参照

関連項目

End (Visual Basic)

'End Try' の前には、対応する 'Try' を指定しなければなりません。

End Try は **Try** ブロックを終了するために使用します。したがって、ブロックの最後でだけ使用できます。**End Try** が重複しています。または、対応する **Try** ブロックの境界の範囲外に **EndTry** ステートメントが記述されています 。

Error ID: BC30383

このエラーを解決するには

1. 必要のない **End Try** ステートメントを探し、削除します。
2. **End Try** ステートメントをコード内の適切な位置に移動します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

'End SyncLock' の前には、対応する 'SyncLock' が必要です。

SyncLock ブロックは、**SyncLock** キーワードで始まり、**End SyncLock** 構成体で終わります。

Error ID: BC30674

このエラーを解決するには

- **SyncLock** ブロックが **SyncLock** ステートメントで始まっていることを確認します。

参照

関連項目

SyncLock ステートメント

'End Sub' は、行の最初のステートメントでなければなりません。

End Sub ステートメントが、コロン (:) ステートメント区切り記号に続いています。**End Sub** は、ソース行の唯一のステートメントでなければなりません。

Error ID: BC32031

このエラーを解決するには

- 複数のステートメントをそれぞれの行に分割します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

Sub ステートメント (Visual Basic)

'End Sub' の前には、一致する 'Sub' を指定しなければなりません。

End Sub ステートメントより前に、対応する **Sub** プロシージャの定義がありません。

Error ID: BC30429

このエラーを解決するには

- **End Sub** ステートメントが余分な場合は削除します。
- **Sub** プロシージャがない場合は指定します。
- **End Sub** ステートメントをコード内の適切な位置に移動します。

参照

関連項目

End (Visual Basic)

概念

Sub プロシージャ

'End Sub' が必要です。

End ステートメントは、その **End** ステートメントが終了させるブロックと対応している必要があります。**Sub** プロシージャ定義が正しく終了していません。

Error ID: BC30026

このエラーを解決するには

- プロシージャの最後に **End Sub** ステートメントを追加します。

参照

関連項目

End ステートメント

概念

Sub プロシージャ

'End Structure' の前には、対応する 'Structure' を指定しなければなりません。

End Structure ステートメントは、**Structure** ステートメントで始まる構造体ブロックの最後でのみ使用できます。

Error ID: BC30621

このエラーを解決するには

- 現在の構造体ブロックが **Structure** ステートメントで始まっていることを確認します。

参照

概念

構造体とクラス

その他の技術情報

構造体 : 独自のデータ型

'End' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Stop ステートメントと **End** ステートメントは実行を中断するため、デバッグ コンテキストでは使用できません。

Error ID: BC30123

このエラーを解決するには

- [イミディエイト] ウィンドウで **End** ステートメントまたは **Stop** ステートメントを実行しないようにします。

参照

関連項目

[イミディエイト ウィンドウ]

'End' ステートメントが有効ではありません。

End キーワードの使い方が間違っているか、現在のコンテキストに合っていません。

Error ID: BC30678

このエラーを解決するには

1. **End** を単独で使用してプロシージャがスコープを失うようにします。
2. **End** をその他のキーワードと一緒に使用して、構造体やプロパティなどのコード ブロックを終了させます。

参照

関連項目

End ステートメント
End (Visual Basic)

クラス ライブラリ プロジェクトで 'End' ステートメントを使用することはできません。

DLL の作成に使ったクラス ライブラリ プロジェクトでは、**End** キーワードを使ってプロシージャのコードの実行を終了できません。

Error ID: BC30615

このエラーを解決するには

- **While**、**For** などの制御構造を使ってプログラム実行のフローを制御します。

参照

その他の技術情報

Visual Basic における制御フロー

'End Set' は、行の最初のステートメントでなければなりません。

End Set ステートメントの前にコロン (:) ステートメント区切り記号があります。**End Set** はソース行の唯一のステートメントでなければなりません。

Error ID: BC32034

このエラーを解決するには

- 複数のステートメントをそれぞれの行に分割します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

Set ステートメント (Visual Basic)

'End Set' の前には、対応する 'Set' を指定しなければなりません。

End Set は、**Set** プロパティ プロシージャを終了するために使います。**End Set** 構成体が **Set** プロパティ プロシージャの外にあります。

Error ID: BC30632

このエラーを解決するには

1. **Set** プロパティ プロシージャが、**Property** キーワードと **End Property** 構成体の間で宣言されていることを確認します。
2. **Set** プロパティ プロシージャが、**Set** キーワードで始まり、**End Set** 構成体で終わっていることを確認します。

参照

関連項目

Property ステートメント

概念

プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

'End Select' の前には、対応する 'Select Case' を指定しなければなりません。

End Select ステートメントに対応する **Select** ステートメントまたは **Select Case** ステートメントがありません。**End Select** ステートメントの前には、対応する **Select** ステートメントまたは **Select Case** ステートメントを指定してください。

Error ID: BC30088

このエラーを解決するには

1. この **Select** ブロックが、入れ子になった **Select** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
2. **Select** ブロック内の他の制御構造がそれぞれ正しく終了していることを確認します。
3. **Select** ブロックの書式が正しいことを確認します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

'End RemoveHandler' は、行の最初のステートメントでなければなりません。

End RemoveHandler ステートメントの前にコロン (:) ステートメント区切り記号があります。**End RemoveHandler** はソース行の唯一のステートメントでなければなりません。

Error ID: BC31119

このエラーを解決するには

- 複数のステートメントをそれぞれの行に分割します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

RemoveHandler ステートメント

Event ステートメント

'End RemoveHandler' の前には、対応する 'RemoveHandler' 宣言を指定しなければなりません。

End RemoveHandler ステートメントに対応する **RemoveHandler** ステートメントが存在しません。**End RemoveHandler** ステートメントの前に、対応する **RemoveHandler** ステートメントを指定する必要があります。

Error ID: BC31125

このエラーを解決するには

- 前にある **RemoveHandler** ステートメントが有効であり、スペルが正しいことを確認します。

参照

関連項目

RemoveHandler ステートメント
Event ステートメント

'End RaiseEvent' は、行の最初のステートメントでなければなりません。

End RaiseEvent ステートメントの前にコロン (:) ステートメント区切り記号があります。**End RaiseEvent** はソース行の唯一のステートメントでなければなりません。

Error ID: BC31120

このエラーを解決するには

- 複数のステートメントをそれぞれの行に分割します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

RaiseEvent ステートメント

Event ステートメント

'End RaiseEvent' の前には、一致する 'RaiseEvent' を指定しなければなりません。

End RaiseEvent ステートメントに対応する **RaiseEvent** ステートメントがありません。**End RaiseEvent** ステートメントの前に、対応する **RaiseEvent** ステートメントを指定してください。

Error ID: BC31126

このエラーを解決するには

- 前にある **RaiseEvent** ステートメントが有効であり、スペルが正しいことを確認します。

参照

関連項目

RaiseEvent ステートメント
Event ステートメント

'End Property' の前には、一致する 'Property' を指定しなければなりません。

End Property ステートメントより前に、対応する **Property** の定義がありません。

Error ID: BC30431

このエラーを解決するには

- **End Property** ステートメントが余分な場合は削除します。
- **Property** プロシージャがない場合は指定します。
- **End Property** をコード内の適切な位置に移動します。

参照

関連項目

End (Visual Basic)

概念

プロパティとプロパティ プロシージャ

'End Operator' は、行の最初のステートメントでなければなりません。

End Operator ステートメントの前にコロン **:** ステートメント区切り記号があります。**End Operator** はソース行の唯一のステートメントであることが必要です。

Error ID: BC33006

このエラーを解決するには

- 複数のステートメントをそれぞれの行に分割します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する
方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

Operator ステートメント

'End Operator' の前には、対応する '演算子' を指定しなければなりません。

End Operator ステートメントより前に、対応する **Operator** の宣言がありません。

Error ID: BC33007

このエラーを解決するには

- **End Operator** ステートメントが余分な場合は削除します。
- **Operator** プロシージャがない場合は指定します。
- **End Operator** ステートメントをコード内の適切な位置に移動します。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

End (Visual Basic)
Operator ステートメント

概念

演算子プロシージャ

'End Operator' が必要です。

End ステートメントは、その **End** ステートメントが終了させるブロックと対応している必要があります。**Operator** プロシージャ定義が正しく終了していません。

Error ID: BC33005

このエラーを解決するには

- プロシージャの最後に **End Operator** ステートメントを追加します。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

End ステートメント

概念

演算子プロシージャ

ステートメントの終わりを指定してください。

ステートメントは構文的に完了していますが、ステートメントを終了させる要素の後ろにプログラミング要素があります。すべてのステートメントの最後には、行の終端記号が必要です。

Error ID: BC30205

このエラーを解決するには

1. 2 つの異なるステートメントが誤って同じ行に記述されていないかどうかを確認します。
2. ステートメントを終了させる要素の後ろに行の終端記号を挿入します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

概念

ステートメントの概要

パラメータ リストを終了させる必要があります。

プロシージャ宣言で、**ParamArray** の指定の後にその他の引数が指定されています。

Error ID: BC30192

このエラーを解決するには

- **ParamArray** を引数リストの最後に移動します。

参照

概念

パラメータ配列

式の終わりが必要です。

完成している式の後に余分な文字が付いています。

Error ID: BC30710

このエラーを解決するには

- オブジェクトやステートメントの構文が正しいことを確認します。

参照

その他の技術情報

Visual Studio でのデバッグ

'End Namespace' の前には、対応する 'Namespace' を指定しなければなりません。

名前空間は、**Namespace** ステートメントで始まり、**End Namespace** 構成要素で終わります。

Error ID: BC30623

このエラーを解決するには

- 名前空間ブロックが **Namespace** ステートメントで始まっていることを確認します。

参照

関連項目

Namespace ステートメント

概念

Visual Basic における名前空間

'End Module' の前には、対応する 'Module' を指定しなければなりません。

モジュール ブロックは、**Module** ステートメントで始まり、**End Module** 構成要素で終わります。

Error ID: BC30622

このエラーを解決するには

- モジュール ブロックが **Module** ステートメントで始まっていることを確認します。

参照

関連項目

Module ステートメント

'End Interface' の前には、対応する 'Interface' を指定しなければなりません。

End Interface ステートメントに対応する **Interface** ステートメントがありません。**End Interface** ステートメントの前には、対応する **Interface** ステートメントを指定しなければなりません。

Error ID: BC30252

このエラーを解決するには

1. 先行する **Interface** ステートメントのスペルが間違っていないことや、無効でないことを確認します。
2. **Interface** メンバの書式が正しいことを確認します。

参照

関連項目

Interface ステートメント (Visual Basic)

'End If' の前には、対応する 'If' を指定しなければなりません。

End If ステートメントに対応する **If** ステートメントがありません。**End If** ステートメントの前には、対応する **If** ステートメントを指定してください。

Error ID: BC30087

このエラーを解決するには

1. この **If** ブロックが、入れ子になった **If** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
2. **If** ブロック内の他の制御構造がそれぞれ正しく終了していることを確認します。
3. **If** ブロックの書式が正しいことを確認します。

参照

関連項目

If...Then...Else ステートメント (Visual Basic)

'End Get' は、行の最初のステートメントでなければなりません。

End Get ステートメントの前にコロン (:) ステートメント区切り記号があります。**End Get** はソース行の唯一のステートメントでなければなりません。

Error ID: BC32033

このエラーを解決するには

- 複数のステートメントをそれぞれの行に分割します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

Get ステートメント

'End Get' の前には、対応する 'Get' を指定しなければなりません。

End Get は、**Get** プロパティ プロシージャを終了するために使います。**End Get** 構成要素が **Get** プロパティ プロシージャの外にあります。

Error ID: BC30630

このエラーを解決するには

1. **Get** プロパティ プロシージャが、**Property** キーワードと **End Property** 構成要素の間で宣言されていることを確認します。
2. **Get** プロパティ プロシージャが、**Get** キーワードで始まり、**End Get** 構成要素で終わっていることを確認します。

参照

関連項目

Property ステートメント

概念

プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

'End Function' は、行の最初のステートメントでなければなりません。

End Function ステートメントの前にコロン (:) ステートメント区切り記号があります。**End Function** はソース行の唯一のステートメントでなければなりません。

Error ID: BC32032

このエラーを解決するには

- 複数のステートメントをそれぞれの行に分割します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

Function ステートメント (Visual Basic)

'End Function' の前には、一致する 'Function' を指定しなければなりません。

End Function ステートメントより前に、対応する **Function** プロシージャの定義がありません。

Error ID: BC30430

このエラーを解決するには

1. **End Function** ステートメントが余分な場合は削除します。
2. **Function** プロシージャがない場合は指定します。
3. **End Function** ステートメントをコード内の適切な位置に移動します。

参照

関連項目

End (Visual Basic)

概念

Function プロシージャ

'End Function' が必要です。

End ステートメントは、その **End** ステートメントが終了させるブロックと対応している必要があります。**Function** プロシージャ定義が正しく終了していません。

Error ID: BC30027

このエラーを解決するには

- プロシージャの最後に **End Function** ステートメントを追加します。

参照

関連項目

End ステートメント

概念

Function プロシージャ

'End Event' の前には、一致する 'Custom Event' を指定しなければなりません。

End Event ステートメントに対応する **Custom Event** ステートメントがありません。**End Event** ステートメントの前には、対応する **Custom Event** ステートメントを指定してください。

Error ID: BC31123

このエラーを解決するには

- 前にある **Custom Event** ステートメントが有効であり、スペルが正しいことを確認します。

参照

関連項目

Event ステートメント

'End Enum' の前には、対応する 'Enum' を指定しなければなりません。

End Enum ステートメントに対応する **Enum** ステートメントがありません。**End Enum** ステートメントの前には、対応する **Enum** ステートメントを指定してください。

Error ID: BC30184

このエラーを解決するには

1. 先行する **Enum** ステートメントのスペルが間違っていないこと、およびステートメント自体が無効でないことを確認します。
2. **Enum** メンバの書式が正しいことを確認します。

参照

関連項目

Enum ステートメント (Visual Basic)

'End Class' の前には、一致する 'Class' を指定しなければなりません。

End Class は **Class** ブロックを終了するために使用します。したがって、ブロックの最後でだけ使用できます。**End Class** が重複しています。または、対応する **Class** ブロックの境界の範囲外に **End Class** ステートメントが記述されています。

Error ID: BC30460

このエラーを解決するには

- 重複する **End Class** ステートメントを探し、削除します。
- **End Class** ステートメントをコード内の適切な位置に移動します。

参照

関連項目

End (Visual Basic)

'End AddHandler' は、行の最初のステートメントでなければなりません。

End AddHandler ステートメントの前にコロン (:) ステートメント区切り記号があります。**End AddHandler** はソース行の唯一のステートメントでなければなりません。

Error ID: BC31118

このエラーを解決するには

- 複数のステートメントをそれぞれの行に分割します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

AddHandler ステートメント

Event ステートメント

'End AddHandler' の前には、対応する 'AddHandler' を指定しなければなりません。

End AddHandler ステートメントに対応する **AddHandler** ステートメントがありません。**End AddHandler** ステートメントの前に、対応する **AddHandler** ステートメントを指定してください。

Error ID: BC31124

このエラーを解決するには

- 前にある **AddHandler** ステートメントが有効であり、スペルが正しいことを確認します。

参照

関連項目

AddHandler ステートメント
Event ステートメント

<qualifiedcontainername>' でのプロジェクト レベル インポート '<qualifiedelementname>' でエラーが発生しました: <errormessage>

ステートメントが別のアセンブリに定義されたプログラミング要素にアクセスしていますが、そのアセンブリへのプロジェクト参照がありません。

たとえば、コードから `desiredEnumeration` という名前の列挙体に、`otherNamespace.otherClass.desiredEnumeration` という修飾文字列を使ってアクセスするとします。コンパイラが自分のプロジェクトの参照の中に `otherNamespace.otherClass` を見つけることができない場合、このエラーが生成されます。

Error ID: BC30797

このエラーを解決するには

1. プログラミング要素の修飾文字列にあるすべての要素のスペルが正しいことを確認します。
2. 目的のプログラミング要素が定義されたアセンブリへの参照が、自分のプロジェクトにあることを確認します。
3. エラー メッセージを調べて、適切なアクションを実行します。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する
方法 : Visual Studio のリファレンスを追加または削除する

概念

同じ名前を持つ複数の変数がある場合に参照を解決する

一時 Win32 リソース ファイル '<filename>' を保存中にエラーが発生しました : <error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、インメモリ リソースの書き込み先となるファイルの名前の取得エラーを報告しました。

Error ID: BC30137

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べます。詳細とアドバイスについては、Al.exe ツールのエラーと警告を参照してください。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

関連項目

アセンブリ リンカ (Al.exe)

Al.exe ツールのエラーと警告

その他の技術情報

製品のサポートとユーザー補助

アセンブリ マニフェストのオプションを設定中にエラーが発生しました : <error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、アセンブリに署名するために使用するキー ファイルまたはキー名の検索エラーを報告しました。

Error ID: BC30139

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べます。詳細とアドバイスについては、Al.exe ツールのエラーと警告 を参照してください。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

関連項目

アセンブリ リンカ (Al.exe)

Al.exe ツールのエラーと警告

その他の技術情報

製品のサポートとユーザー補助

'Error' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Error ステートメントはソース コードでのみ使用できます。

Error ID: BC30724

このエラーを解決するには

- デバッグ コードから **Error** ステートメントを削除します。

参照

関連項目

Error ステートメント

その他の技術情報

Visual Studio でのデバッグ

式またはステートメントの評価が中止されました。

Stop Evaluation メニュー項目が選択され、評価が中断されました。

Error ID: BC30721

このエラーを解決するには

- Stop Evaluation. を選択せずに実行を継続します。

参照

その他の技術情報

Visual Studio でのデバッグ

式またはステートメントの評価がタイムアウトしました。

式の評価が時間内に完了しませんでした。

Error ID: BC30722

このエラーを解決するには

1. 入力したコードが正しいことを確認してください。
2. 実行時間を短くするために、式を単純化します。

参照

その他の技術情報

Visual Studio でのデバッグ

'<classname>' イベントが見つかりません。

コンパイラが認識できないイベントを処理しようとしました。

Error ID: BC30590

このエラーを解決するには

- イベント名のスペルが正しいことを確認してください。

参照

概念

イベントとイベント ハンドラ

イベント '<eventname>' を '<name>' からアクセスできないため、処理できません。

アクセスできないイベントを処理しようとしました。たとえば、**Handles** 変数が共有の場合は、イベントを処理するメソッドも共有にする必要があります。

Error ID: BC30585

このエラーを解決するには

- イベントがアクセス可能であることを確認します。

参照

概念

イベントとイベント ハンドラ

'DefaultEvent' 属性で指定されたイベント '<eventname>' は、このクラスに対して公開されているアクセス可能なイベントではありません。

DefaultEventAttribute 属性には、クラス内の公開されているアクセス可能なイベントの名前を指定する必要があります。このイベントに属性が適用されます。

Error ID: BC30770

このエラーを解決するには

1. 公開されているアクセス可能なイベントが、このクラスで定義されていることを確認します。
2. クラス内のイベントの名前が、**DefaultEventAttribute** 属性によって指定された名前と一致していることを確認します。

参照

関連項目

Event ステートメント
Class ステートメント (Visual Basic)
Public (Visual Basic)
DefaultEventAttribute

イベント '<eventname>' は、ベース <type> '<classname>' のメンバと競合する '<membername>' を暗黙的に宣言するため、このイベントは 'Shadows' と宣言しなければなりません。

イベントの宣言で使用した名前を使用して暗黙のメンバの名前が形成されますが、基本クラスにこれと同じ名前のメンバがあります。たとえば、**Event1** という名前のイベントを宣言した場合は、コンパイラによって暗黙のプロシージャ**add_Event1** と **remove_Event1** が生成されます。これと同じ名前のメンバが基本クラスにある場合、このクラスのイベントは基本クラスのメンバをシャドウする必要があります。

このメッセージは警告です。**Shadows** が既定で使用されます。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC40012

このエラーを解決するには

1. 基本クラスのメンバを隠す場合は、**Shadows** キーワードをプロパティ宣言に追加します。
2. 基本クラスのメンバを隠さない場合は、イベントの名前を変更します。

参照

関連項目

Event ステートメント

Shadows

概念

Visual Basic におけるシャドウ

デリゲート型が、'<eventname1>' によって実装されている別のイベントのデリゲート型と一致しないため、イベント '<eventname1>' でイベント '<eventname2>' を実装することはできません。

Visual Basic はイベントを実装できません。そのイベントのデリゲート型が別のイベントのデリゲート型と一致しないためです。このエラーは、インターフェイス内で複数のイベントを定義し、同じイベントを使用してそれらのイベントを実装しようとすると発生する場合があります。1 つのイベントが 2 つ以上のイベントを実装できるのは、実装されるすべてのイベントが **As** 構文を使用して宣言され、同じデリゲート型を指定している場合だけです。

Error ID: BC31407

このエラーを解決するには

- イベントを個別に実装します。

参照

その他の技術情報

Visual Basic におけるイベント

デリゲート型 '<delegate1>' および '<delegate2>' が一致しないため、イベント '<eventname1>' はインターフェイス '<interface>' 上のイベント '<eventname2>' を実装できません。

Visual Basic はイベントを実装できません。そのイベントのデリゲート型がインターフェイスのイベントのデリゲート型と一致しないためです。このエラーは、インターフェイス内で複数のイベントを定義し、同じイベントを使用してそれらのイベントを実装しようとすると発生する場合があります。1 つのイベントが 2 つ以上のイベントを実装できるのは、実装されるすべてのイベントが **As** 構文を使用して宣言され、同じデリゲート型を指定している場合だけです。

Error ID: BC31423

このエラーを解決するには

- イベントを個別に実装します。

  または

- インターフェイス内のイベントを **As** 構文を使用して定義し、同じデリゲート型を指定します。

参照

関連項目

Event ステートメント

Delegate ステートメント

その他の技術情報

Visual Basic におけるイベント

イベント名の長さは 1011 文字を超過できません。

Event ステートメントを指定して宣言する各イベントの名前、つまり識別子は 1011 文字に制限されています。

Error ID: BC32204

このエラーを解決するには

- 識別子の文字数を減らします。

参照

関連項目

Event ステートメント

概念

宣言された要素の名前

イベントを、戻り値の型を持つデリゲート型で宣言することはできません。

関数プロシージャについてデリゲートが指定されました。

Error ID: BC31084

このエラーを解決するには

- **Sub** 用のデリゲートを指定します。

参照

その他の技術情報

Visual Basic におけるイベント

イベントに戻り値を指定することはできません。

イベントは引数を受け取ることはできますが、値を直接返すことはできません。

Error ID: BC30032

このエラーを解決するには

- **Event** ステートメントから戻り値の型を削除します。

参照

関連項目

Event ステートメント

'As' 句を伴って宣言されたイベントは、デリゲート型を指定しなければなりません。

デリゲートでない型が指定されました。

Error ID: BC31044

このエラーを解決するには

1. **As** 句を削除します。
2. **As** 句のあるデリゲートを指定します。

参照

関連項目

Event ステートメント
Delegate ステートメント

その他の技術情報

Visual Basic におけるイベント
Visual Basic でのデリゲート

モジュールのイベントを '<specifier>' として宣言できません

Module ステートメント内のイベントでは無効な指定子をイベントに使用しました。モジュールはインスタンス化できず、継承をサポートしません。また、インターフェイスを実装できません。

Error ID: BC30434

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Module ステートメント

インターフェイス内のイベントは、'<implements>' として宣言することはできません。

インターフェイス内で宣言されたイベントは、その他のインターフェイスのイベントを実装できません。

Error ID: BC30688

このエラーを解決するには

1. **Implements** ステートメントを削除します。
2. クラスまたは構造体の中でイベントを実装します。

参照

関連項目

Interface ステートメント (Visual Basic)

共有 WithEvents 変数のイベントを、非共有メソッドで処理できません。

Shared 修飾子を使って宣言されている変数は共有変数です。共有変数は、ただ 1 つのストレージ位置を表します。**WithEvents** 修飾子付きで宣言された変数は、変数が属している型が変数によって生成されるイベントのセットを処理することを表しています。変数に値を代入すると、**WithEvents** 宣言によって作成されたプロパティが既存の各イベント ハンドラをアンフックし、**Add** メソッドを介して新しいイベント ハンドラをフックします。

Error ID: BC30594

このエラーを解決するには

- イベント ハンドラを **Shared** で宣言します。

参照

関連項目

Shared (Visual Basic)
WithEvents

概念

イベントとイベント ハンドラ

'Exit AddHandler'、'Exit RemoveHandler'、および 'Exit RaiseEvent' は有効ではありません。

エラー メッセージ
'Exit AddHandler'、'Exit RemoveHandler'、および 'Exit RaiseEvent' は有効ではありません。イベント メンバを終了する場合は 'Return' を使用します。

Custom Event 宣言内の **AddHandler** メソッド、**RemoveHandler** メソッド、**RaiseEvent** メソッドを終了するために **Exit** ステートメントを使用することはできません。代わりに、戻り値の式を指定しないで **Return** ステートメントを使用して、メソッドを終了してください。

Error ID: BC31111

このエラーを解決するには

- **Exit** ステートネントを **Return** ステートメントに置き換えます。

  Return ステートメントに戻り値の式を指定しないようにしてください。

参照

関連項目
Event ステートメント
AddHandler
RemoveHandler
RaiseEvent
Return ステートメント (Visual Basic)
その他の技術情報
Visual Basic におけるイベント

'Exit Do' は、'Do' ステートメント内でのみ使用できます。

Exit Do ステートメントが **Do** ループの外側にあります。**Exit Do** ステートメントは、**Do** ステートメントと、これに対応する **Loop** ステートメントの間でのみ有効です。

Error ID: BC30089

このエラーを解決するには

1. **Exit Do** より前に有効な **Do** ステートメントがあり、その後に有効な **Loop** ステートメントがあることを確認します。
2. **Do** ループ内の他の制御構造がそれぞれ正しく終了していることを確認します。

参照

関連項目

Do...Loop ステートメント (Visual Basic)

'Exit For' は、'For' ステートメント内でのみ使用できます。

Exit For ステートメントが **For** ループの外側にあります。**Exit For** ステートメントは、**For** ステートメントか **For Each** ステートメントと、これに対応する **Next** ステートメントの間でのみ有効です。

Error ID: BC30096

このエラーを解決するには

1. **Exit For** より前に有効な **For** ステートメントか **For Each** ステートメントがあり、その後に有効な **Next** ステートメントがあることを確認します。
2. **For** ループ内の他の制御構造がそれぞれ正しく終了していることを確認します。

参照

関連項目

For...Next ステートメント (Visual Basic)
For Each...Next ステートメント (Visual Basic)

'Exit Function' は、Sub または Property 内では有効ではありません。

Sub プロシージャまたは **Property** プロシージャに **Exit Function** があります。**Exit** ステートメントは、その **Exit** ステートメントが記述されているブロックと対応している必要があります。

Error ID: BC30067

このエラーを解決するには

- **Exit Function** を **Exit Sub** ステートメントまたは **Exit Property** ステートメントに置き換えます。

参照

概念

Sub プロシージャ
Function プロシージャ
Property プロシージャ

'Exit' の後には、'Sub'、'Function'、'Property'、'Do'、'For'、'While'、'Select'、または 'Try' を指定しなければなりません。

Exit ステートメントに無効なキーワードがあります。**Exit** では、後続のステートメントに制御を移す元のブロックを指定する必要があります。たとえば、`End While` のように記述します。

Error ID: BC30240

このエラーを解決するには

- **Exit** の後ろに有効なキーワードを追加するか、または **Exit** ステートメントを削除します。

参照

関連項目

Exit ステートメント (Visual Basic)

'Exit Operator' は無効です。演算を終了するには、'Return' を使用してください。

Exit Operator ステートメントが、**Operator** プロシージャ内にあります。

Operator プロシージャから戻るには、Return ステートメント (Visual Basic) を使用する必要があります。Exit ステートメント (Visual Basic) は、**Operator** キーワードを受け入れません。また、**End Operator** ステートメントは、呼び出し元のコードに制御を返しません。

Error ID: BC33008

このエラーを解決するには

- **Exit Operator** ステートネントを **Return** ステートメントに置き換えます。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

'Exit Property' は、Function または Sub では有効ではありません。

Function プロシージャまたは **Sub** プロシージャに **Exit Property** があります。**Exit** ステートメントは、その **Exit** ステートメントが記述されているブロックと対応している必要があります。

Error ID: BC30066

このエラーを解決するには

- **Exit Property** を **Exit Function** ステートメントまたは **Exit Sub** ステートメントに置き換えます。

参照

概念

Sub プロシージャ
Function プロシージャ
Property プロシージャ

'Exit Select' は、'Select' ステートメント内でのみ使用できます。

Exit Select ステートメントが **Select** ブロックの外側にあります。**Exit Select** ステートメントは、**Select** または **Select Case** ステートメントと、これに対応する **End Select** ステートメントの間でのみ有効です。

Error ID: BC30099

このエラーを解決するには

1. **Exit Select** より前に有効な **Select** ステートメントまたは **Select Case** ステートメントがあり、その後に有効な **End Select** ステートメントがあることを確認します。
2. **Select** ブロック内の他の制御構造がそれぞれ正しく終了していることを確認します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

'Exit' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Exit ステートメントはソース コードでのみ使用できます。

Error ID: BC30713

このエラーを解決するには

- デバッグ コードから **Exit** ステートメントを削除します。

参照

関連項目

Exit ステートメント (Visual Basic)

End ステートメント

その他の技術情報

Visual Studio でのデバッグ

'Exit Sub' は、Function または Property では有効ではありません。

Function プロシージャまたは **Property** プロシージャに **Exit Sub** があります。**Exit** ステートメントは、その **Exit** ステートメントが記述されているブロックと対応している必要があります。

Error ID: BC30065

このエラーを解決するには

- **Exit Sub** を状況に応じて **Exit Function** ステートメントか **Exit Property** ステートメントに置き換えます。

参照

概念

Sub プロシージャ
Function プロシージャ
Property プロシージャ

'Exit Try' は、'Try' ステートメント内部にのみ記述できます。

Exit Try は **Try** ステートメント ブロック内でのみ使用できます。**Exit Try** ステートメントが重複しています。または、対応する **Try** ブロックの境界の範囲外に **Exit Try** ステートメントが記述されています。

Error ID: BC30393

このエラーを解決するには

1. 必要のない **Exit Try** ステートメントを探し、削除します。
2. **Exit Try** ステートメントをコード内の適切な位置に移動します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

'Exit While' は、'While' ステートメント内でのみ使用できます。

Exit While ステートメントが **While** ブロックの外側にあります。**Exit While** ステートメントは、**While** ステートメントと、これに対応する **End While** ステートメントの間でのみ有効です。

Error ID: BC30097

このエラーを解決するには

1. **Exit While** より前に有効な **While** ステートメントがあり、その後に有効な **End While** ステートメントがあることを確認します。
2. **While** ブロック内の他の制御構造がそれぞれ正しく終了していることを確認します。

参照

関連項目

While...End While ステートメント (Visual Basic)

'Dim'、'Const'、'Public'、'Private'、'Protected'、'Friend'、'Shadows'、'ReadOnly'、'Shared' のいずれかを指定してください。

宣言ステートメントに宣言キーワードがありません。属性の宣言でメソッドが呼び出されていることが原因の 1 つとして考えられます。

Error ID: BC30195

このエラーを解決するには

1. 属性の宣言内でメソッドが宣言されているかどうかを確認します。
2. 適切な宣言キーワードを使用してステートメントを開始します。

参照

関連項目

配列を 'New' で宣言することはできません。

その他の技術情報

Visual Basic における宣言された要素

明示的な初期化は、明示的な境界を含む配列に対しては許可されていません。

サイズを指定して宣言されている配列は初期化できません。

Error ID: BC30672

このエラーを解決するには

- 配列を宣言し、別に初期化します。
- 動的配列として宣言および初期化し、必要に応じて **ReDim** を使用します。次に例を示します。

```
Dim A() As Integer = {0, 1, 2, 3}
ReDim Preserve A(3)
```

参照

その他の技術情報

Visual Basic における配列

単一の型指定子を持つ複数の宣言子で、明示的な初期化を行うことはできません。

複数の変数が同一行で宣言された場合は、初期化できません。

Error ID: BC30671

このエラーを解決するには

1. 各項目を個別に宣言および初期化します。
2. 複数の項目を同時に宣言し、次に各項目を初期化します。次に例を示します。

```
Dim x, b, i As Integer
x = 9 : b = 9 : i = 9 ' ":" is the same as a new line.
```

参照

関連項目

Dim ステートメント (Visual Basic)

概念

Visual Basic での変数宣言

指数が有効ではありません。

浮動小数点リテラルで有効ではない指数を使おうとしました。このエラーは、通常、浮動小数点リテラルの型と、浮動小数点リテラルを保持するために必要なサイズが対応しない場合に発生します。

Error ID: BC30495

このエラーを解決するには

- 浮動小数点リテラルの型を確認し、有効な型に変更します。

参照

処理手順

方法 : 数値を計算する

式は、含んでいるプロパティ '<propertyname>' を再帰的に呼び出します。

プロパティ定義の **Set** プロシージャ内のステートメントが、そのプロパティの名前に値を格納しています。

プロパティの値を格納するには、プロパティのコンテナに **Private** 変数を定義し、それを **Get** プロシージャおよび **Set** プロシージャ内で使用する方法をお勧めします。この場合は **Set** プロシージャが、この **Private** 変数に受け取った値を格納します。

Get プロシージャは **Function** プロシージャと同様の動作をするため、プロパティ名に値を代入したり、**End Get** ステートメントに達したときに制御を戻したりすることが可能です。しかし、推奨されるのは、Return ステートメント (Visual Basic) にこの値のための **Private** 変数を定義するという方法です。

Set プロシージャは **Sub** プロシージャと同様の動作をし、値を返しません。したがって、プロシージャ名またはプロパティ名は、**Set** プロシージャ内では特別な意味を持たず、この名前に値を格納することはできません。

次の例は、エラーを引き起こす可能性のある方法を示しています。その後に続けて、推奨される方法を示します。

```
Public Class illustrateProperties
' The code in the following property causes this error.
    Public Property badProp() As Char
        Get
            Dim charValue As Char
            ' Insert code to update charValue.
            badProp = charValue
        End Get
        Set(ByVal Value As Char)
            ' The following statement causes this error.
            badProp = Value
            ' The value stored in the local variable badProp
            ' is not used by the Get procedure in this property.
        End Set
    End Property
' The following code uses the recommended approach.
    Private propValue As Char
    Public Property goodProp() As Char
        Get
            ' Insert code to update propValue.
            Return propValue
        End Get
        Set(ByVal Value As Char)
            propValue = Value
        End Set
    End Property
End Class
```

既定では、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42026

このエラーを解決するには

- 上記の例で示した推奨される方法を使用するようにプロパティ定義を変更します。

参照

関連項目

Property ステートメント
Set ステートメント (Visual Basic)

概念

Property プロシージャ

式は、含んでいる演算子 '<operatorsymbol>' を再帰的に呼び出します。

演算子プロシージャ内の式で、定義の対象となっている演算子が使用されています。そのため、使用するデータ型が原因で演算子プロシージャはそれ自身を呼び出すことになります。

定義中の演算子プロシージャがそれ自身を呼び出すことになるのは、同じ演算子を以下のいずれかと一緒に使用した場合です。

- 演算子を定義しているのと同じオペランド
- 演算子を定義しているのと同じデータ型のオペランド
- 演算子を定義しているのと同じデータ型に拡大されるデータ型のオペランド

再帰呼び出しは、プロシージャがそれ自身を呼び出すことを指します。再起呼び出しは、無限ループに陥る可能性があります。無限ループは、制御が同じ一組のステートメントを繰り返し通過し、アプリケーションが外部から終了されるまで停止しない状態を指します。再起を終了するために使用できる 1 つ以上のテストがコードに含まれていない場合、無限ループが発生する危険があります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42004

このエラーを解決するには

- 演算子プロシージャからそれ自身を呼び出すことが必要なロジックである場合は、確実に通過する場所で少なくとも 1 つの条件をテストし、その結果に基づいて再起呼び出しを終了できるようにします。
- 演算子プロシージャからそれ自身を呼び出す必要がないロジックであれば、再起呼び出しを削除するか、自身のプロシージャを呼び出さないステートメントに置き換えます。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

式またはステートメントは、デバッグ ウィンドウでは有効ではありません。

Try...Catch...Finally ステートメントはデバッグ コンテキストでは使用できません。

Error ID: BC30706

このエラーを解決するには

- デバッグ コードから **Try...Catch...Finally** ステートメントを削除します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

その他の技術情報

Visual Studio でのデバッグ

型 ''ValueType' の式は、'Object' または 'ValueType' に変換できません。

式の評価結果が、共通言語ランタイム (CLR) でボックス化できない型になります。

ボックス化とは、型を **Object** (場合によっては ValueType) に変換するために不可欠な処理です。共通言語ランタイムは、ArgIterator や TypedReference など一部の型をボックス化できません。

この式を含むステートメント内で **CType** または **CObj** を使用していない場合、Visual Basic は暗黙的な変換を実行しようとして、このエラーが起こります。

Error ID: BC31394

このエラーを解決するには

1. 問題の型に評価される式を探します。
2. ステートメントのどの部分が問題の型をボックス化しようとしたかを調べます。
3. ステートメントを書き直して、ボックス化変換が行われないようにします。

参照

概念

暗黙の型変換と明示的な型変換

型 '<typename1>' の式を型 '<typename2>' にすることはできません。

TypeOf...**Is** 式は、保持できないデータ型へのオブジェクト参照変数をテストします。

いくつかの場合において、コンパイラは、**TypeOf**...**Is** テストが必ず失敗すると判断できます。たとえば、2 つのクラス間に継承関係がないなどの場合です。

このエラーは次のようなコードで発生します。

```
Dim refVar as System.Windows.Forms.Form

If TypeOf refVar Is System.Array

End If
```

これは、Form および Array は、まったく関係のない型なので、コンパイラは、`refVar` がどんな値であっても **TypeOf**...**Is** 式が **False** を返すと判断できるからです。

Error ID: BC31430

このエラーを解決するには

- 実際に使用できるデータ型の変数をテストするか、**TypeOf**...**Is** テストをすべて削除します。

参照

処理手順

方法 : オブジェクト変数で参照している型を確認する

関連項目

TypeOf 演算子 (Visual Basic)

式の型は '<typename>' で、コレクション型ではありません。

For Each ステートメントで指定されているグループ変数は、コレクション オブジェクトでも配列でもありません。また、変数の型が IEnumerable インターフェイスを実装していません。変数の型は、Visual Basic コレクション デザイン パターンをサポートしているか、**IEnumerable** を実装している必要があります。

Error ID: BC32023

このエラーを解決するには

- Visual Basic コレクション デザインをサポートするクラス型か、**IEnumerable** を実装するクラス型としてグループ変数を宣言します。

参照

関連項目

For Each...Next ステートメント (Visual Basic)
IEnumerable

概念

Visual Basic のコレクション クラス

式は、配列またはメソッドではありません。引数一覧を指定することはできません。

配列またはメソッドではない式に引数リストを渡しました。引数リストを受け取ることができるのは配列またはメソッドだけです。

Error ID: BC30471

このエラーを解決するには

- 引数リストを削除します。

参照

概念

プロシージャのパラメータと引数

式はメソッドではありません。

メソッド呼び出しの対象がメソッドではありません。

Error ID: BC30454

このエラーを解決するには

- メソッド名の入力に誤りがないことを確認します。

参照

概念

プロパティとメソッド

クラス メソッド

Expression は値であるため、代入式のターゲットにすることはできません。

式に値を代入しようとしているステートメントがあります。実行時に値を代入できるのは、書き込み可能な変数、プロパティ、または配列に対してのみです。次の例は、どのような場合にこのエラーが発生するのかを示します。

```
Dim yesterday As Integer
ReadOnly maximum As Integer = 45
yesterday + 1 = DatePart(DateInterval.Day, Now)
' The preceding line is an ERROR because of an expression on the left.
maximum = 50
' The preceding line is an ERROR because maximum is declared ReadOnly.
```

上記の例は、プロパティおよび配列の要素についても当てはまります。

間接アクセス 値の型による間接アクセスでも、このエラーが発生します。次の例では、Location を通して間接的にアクセスすることによって Point に値を設定しようとしています。

```
' Assume this code runs inside Form1.
Dim exitButton As New System.Windows.Forms.Button()
exitButton.Text = "Exit this form"
exitButton.Location.X = 140
' The preceding line is an ERROR because of no storage for Location.
```

上記の例で、最後のステートメントは、**Location** プロパティから返される **Point** 構造体を一時的に割り当てているだけであるため、失敗します。構造体は値型であり、一時的な構造体は、このステートメントの実行後には保持されません。**Location** 用の変数を宣言して使用すると、より永続的に割り当てられる **Point** 構造体が作成されるため、問題を解決できます。上記例の最後のステートメントを置き換えるためのコードを次の例に示します。

```
Dim exitLocation as New System.Drawing.Point(140, exitButton.Location.Y)
exitButton.Location = exitLocation
```

Error ID: BC30068

このエラーを解決するには

- ステートメントが式に値を代入する場合は、式を、書き込みできる単一の変数、プロパティ、または配列要素に置き換えます。
- ステートメントが構造体など値型を通じて間接的にアクセスする場合は、その値型を保持するための変数を作成します。
- 適切な構造体 (またはその他の値型) を変数に代入します。
- 変数を使用して、プロパティにアクセスし、値を代入します。

参照

処理手順

プロシージャのトラブルシューティング

概念

Visual Basic の演算子および式

代入ステートメント

式は制限がある型 '<typename>' を含んでいるため、'Object' または 'ValueType' から継承されたメンバにアクセスするのに使用できません。

式の評価結果が、共通言語ランタイム (CLR) でボックス化できない型になりますが、式はボックス化を要求するメンバにアクセスします。

ボックス化とは、型を **Object** (場合によっては ValueType) に変換するために不可欠な処理です。共通言語ランタイムは、ArgIterator、RuntimeArgumentHandle、および TypedReference など一部の構造体型をボックス化できません。

この式は、制限された型を使用して、GetHashCode や ToString などの、Object または **ValueType** から継承されたメソッドを呼び出そうとします。このメソッドにアクセスするために、Visual Basic が暗黙のボックス化変換を実行しようとすることにより、このエラーが発生します。

Error ID: BC31393

このエラーを解決するには

1. 問題の型に評価される式を探します。
2. ステートメントのどの部分が、**Object** または **ValueType** から継承されるメソッドを呼び出そうとするかを調べます。
3. ステートメントを書き直して、メソッド呼び出しが行われないようにします。

参照

概念

暗黙の型変換と明示的な型変換

式が必要です。

+ や = などの演算子には右辺オペランドの式が必要ですが、認識できる式がありません。

Error ID: BC30201

このエラーを解決するには

- 演算子の右辺オペランドに有効な式を指定します。

参照

概念

Visual Basic の演算子および式
代入ステートメント

式は値を生成しません。

値の生成が必要なコンテキストにおいて、値を生成しない式を使おうとしました。たとえば、**Function** を使用する必要のあるコンテキストで **Sub** を呼び出しました。

Error ID: BC30491

このエラーを解決するには

- 値を生成する式に変更します。

参照

概念

エラーの種類

現時点では、式を評価できません。

指定した式は現在のコンテキストで評価できませんでした。

Error ID: BC30736

このエラーを解決するには

- 式の構文を変更します。

参照

概念

プロシージャのパラメータと引数

フィールド '<fieldname>' の型はサポートされていません。

Visual Studio で表現できない型でフィールドが宣言されています。

Error ID: BC30656

このエラーを解決するには

1. サポートされている型としてフィールドを定義します。
2. フィールド内のデータを記述する新しい型を作成します。

参照

関連項目

データ型の概要 (Visual Basic)

フィールドまたはプロパティ '<membername>' が見つかりません。

属性について未定義のフィールドまたはプロパティが参照されています。

Error ID: BC30661

このエラーを解決するには

1. フィールドまたはプロパティの名前のスペルを確認します。
2. プロジェクトから正しい属性クラスにアクセスできることを確認し、必要に応じて、その属性クラスをプロジェクト参照に追加します。

参照

関連項目

Imports ステートメント

ファイル <filename> が見つかりませんでした。

指定したファイルが見つかりませんでした。指定したファイル パスが間違っている可能性もあります。

Error ID: BC2001

このエラーを解決するには

- ファイルの名前と場所が正しいことを確認します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

ファイル名は、異なる GUID およびチェックサム値で既に宣言しました。

ファイルの宣言が行われましたが、そのファイルは別の GUID とチェックサムで既に存在しています。

Error ID: BC42034

このエラーを解決するには

- ファイル名とパスが正しく指定されていることを確認します。
- 不必要なファイルは削除します。

参照

概念

エラーの種類

'Finally' は、'Try' ステートメント内で 1 回のみ記述できます。

Finally は **Try...Catch...Finally** ブロックを終了するために使用します。したがって、ブロックの最後で 1 度だけ使用できます。

Error ID: BC30381

このエラーを解決するには

- 必要のない **Finally** ステートメントを探し、削除します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

'Finally' を、'Try' ステートメントの外に置くことはできません。

Finally は **Try...Catch...Finally** ブロックを終了するために使用します。したがって、ブロックの最後で 1 度だけ使用できます。不要な **Finally** ステートメントが存在します。または、対応する **Try** ブロックの境界の範囲外に **Finally** ステートメントが記述されています。

Error ID: BC30382

このエラーを解決するには

1. 必要のない **Finally** ステートメントを探して、削除します。
2. **Finally** ステートメントをコード内の適切な位置に移動します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

'Finally' の終わりには、一致する 'End Try' を指定しなければなりません。

Finally ステートメントに対応する **End Try** ステートメントがありません。**Finally** ステートメントには、対応する **End Try** ステートメントが必要です。

Error ID: BC30442

このエラーを解決するには

1. **Finally** ステートメントを削除します。
2. **End Try** ステートメントを追加してブロックを終了させます。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

'Finally' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Finally ステートメントはソース コードでしか使用できません。

Error ID: BC30716

このエラーを解決するには

- デバッグ コードから **Finally** ステートメントを削除します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

その他の技術情報

Visual Studio でのデバッグ

メソッド本体の First ステートメントをメソッド宣言と同じ行に記述することはできません。

Function、**Sub**、**Get**、**Set**、**Property** の各ステートメントは、ソース コード行でほかの行と分けて記述する必要があります。

Error ID: BC30040

このエラーを解決するには

1. プロシージャ宣言に先行する行ラベルを削除します。
2. プロシージャ宣言に先行するステートメントを前のソース コード行に移動します。
3. プロシージャ宣言に続くステートメントを後続のソース コード行に移動します。

参照

処理手順

方法 : ステートメントにラベル付けする

概念

Visual Basic におけるプロシージャ

この 'Sub New' の最初のステートメントは、'MyBase.New' または 'MyClass.New' への呼び出しでなければなりません (パラメータのないアクセス可能なコンストラクタが複数あります)。

エラー メッセージ
この Sub New の最初のステートメントは、MyBase.New または MyClass.New の呼び出しである必要があります。これは、'<derived>' の基本クラス '<base>' に、引数なしで呼び出すことのできるアクセス可能な 'Sub New' が複数存在するためです。

クラス コンストラクタで、基本クラス コンストラクタの呼び出しが指定されていません。また、Visual Basic では暗黙の呼び出しを指定できません。呼び出しの対象となる基本クラス コンストラクタを判断できないためです。

Error ID: BC32038

このエラーを解決するには

- このコンストラクタの最初の行として、基本クラス コンストラクタ **MyBase.New()** の呼び出しを追加するか、**MyClass.New()** または **Me.New()** を使用して、このクラスにある別のコンストラクタの呼び出しを追加します。

参照
関連項目
コンストラクタとデストラクタの使用方法
MyBase
概念
オブジェクトの有効期間 : オブジェクトの作成と破棄

この 'Sub New' の最初のステートメントは、'MyBase.New' または 'MyClass.New' への呼び出しでなければなりません (パラメータのないアクセス可能なコンストラクタがありません)。

エラー メッセージ
この 'Sub New' の最初のステートメントは、'MyBase.New' または 'MyClass.New' の呼び出しである必要があります。これは、'<derivedname>' の基本クラス '<basename>' が、引数なしで呼び出すことのできるアクセス可能な 'Sub New' を持たないためです。

派生クラスの各コンストラクタは、基本クラスのコンストラクタ (**MyBase.New**) を呼び出す必要があります。派生クラスがアクセスできるパラメータなしのコンストラクタが基本クラスにある場合は、**MyBase.New** を自動的に呼び出すことができます。それ以外の場合は、パラメータを使用して基本クラスのコンストラクタを呼び出す必要があります。この呼び出しは、自動的に行うことはできません。その場合、派生クラスのあらゆるコンストラクタの最初のステートメントでは、基本クラスに対してパラメータ化されたコンストラクタを呼び出すか、基本クラスのコンストラクタを呼び出す、派生クラス内の別のコンストラクタを呼び出す必要があります。

Error ID: BC30148

このエラーを解決するには

- 必要なパラメータを指定して **MyBase.New** を呼び出すか、またはそのような呼び出しを行うピア コンストラクタを呼び出します。

参照

関連項目

コンストラクタとデストラクタの使用方法

'<derivedclassname>' の基本クラス '<baseclassname>' にある '<constructorname>' が古い形式に設定されているため、この 'Sub New' の最初のステートメントは、明示的な 'MyBase.New' または 'MyClass.New' への呼び出しでなければなりません。

クラスのコンストラクタが基本クラス コンストラクタを明示的に呼び出していません。しかも基本クラスの暗黙のコンストラクタが ObsoleteAttribute 属性でマーク付けされ、ディレクティブでそれをエラーとして扱うように設定されています。

派生クラス コンストラクタが基本クラス コンストラクタを呼び出していないと、Visual Basic はパラメータを持たない基本クラス コンストラクタへの暗黙の呼び出しを生成しようとします。基本クラス内に、引数を指定しないで呼び出すことができるアクセス可能なコンストラクタが存在しない場合、Visual Basic は暗黙の呼び出しを生成できません。このケースでは、そのようなコンストラクタが **ObsoleteAttribute** 属性でマーク付けされているため、Visual Basic はこれを呼び出すことができません。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

Error ID: BC30919

このエラーを解決するには

- **MyBase.New()** の呼び出し、または **MyClass.New()** の呼び出しを、派生クラスの **Sub New** の最初のステートメントとして定義します。

参照

概念

Visual Basic で使用される属性
属性の適用

'<derivedclassname>' の基本クラス '<baseclassname>' にある '<constructorname>' が古い形式に設定されているため、この 'Sub New' の最初のステートメントは、明示的な 'MyBase.New' または 'MyClass.New' への呼び出しでなければなりません: '<errormessage>'

クラス コンストラクタが基本クラスのコンストラクタを明示的に呼び出しておらず、暗黙の基本クラスのコンストラクタが、ObsoleteAttribute 属性として、またディレクティブによりエラーとして処理するようマークされています。

派生クラスのコンストラクタが基本クラスのコンストラクタを呼び出していないときは、Visual Basic は、パラメータなしの基本クラス コンストラクタに対する暗黙的な呼び出しを生成しようとします。その基本クラスの中に、引数なしで呼び出されるアクセス可能なコンストラクタがない場合は、Visual Basic が暗黙的な呼び出しを生成できません。この場合、必要なコンストラクタが **ObsoleteAttribute** 属性でマークされているため、Visual Basic から呼び出すことができません。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

Error ID: BC30920

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. **MyBase.New()** または **MyClass.New()** の呼び出しを派生クラスの **Sub New** の最初のステートメントとして記述します。

参照

概念

Visual Basic で使用される属性
属性の適用

'<derivedclassname>' の基本クラス '<baseclassname>' にある '<constructorname>' が古い形式に設定されているため、この 'Sub New' の最初のステートメントは、明示的な 'MyBase.New' または 'MyClass.New' への呼び出しでなければなりません。

クラスのコンストラクタが基本クラス コンストラクタを明示的に呼び出しておらず、それを警告として扱うように、基本クラスの暗黙のコンストラクタが ObsoleteAttribute 属性とディレクティブでマーク付けされています。

派生クラス コンストラクタが基本クラス コンストラクタを呼び出していないと、Visual Basic はパラメータを持たない基本クラス コンストラクタへの暗黙の呼び出しを生成しようとします。基本クラス内に、引数を指定しないで呼び出すことができるアクセス可能なコンストラクタが存在しない場合、Visual Basic は暗黙の呼び出しを生成できません。このケースでは、そのようなコンストラクタが **ObsoleteAttribute** 属性でマーク付けされているため、Visual Basic はこれを呼び出すことができません。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

既定では、**ObsoleteAttribute** の **IsError** プロパティが **False** であるため、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC41003

このエラーを解決するには

- **MyBase.New()** の呼び出し、または **MyClass.New()** の呼び出しを、派生クラスの **Sub New** の最初のステートメントとして定義します。

参照

概念

Visual Basic で使用される属性
属性の適用

'<derivedclassname>' の基本クラス '<baseclassname>' にある '<constructorname>' が古い形式に設定されているため、この 'Sub New' の最初のステートメントは、明示的な 'MyBase.New' または 'MyClass.New' への呼び出しでなければなりません: '<errormessage>'

クラスのコンストラクタが基本クラス コンストラクタを明示的に呼び出しておらず、それを警告として扱うように、基本クラスの暗黙のコンストラクタが ObsoleteAttribute 属性とディレクティブでマーク付けされています。

派生クラス コンストラクタが基本クラス コンストラクタを呼び出していないと、Visual Basic はパラメータを持たない基本クラス コンストラクタへの暗黙の呼び出しを生成しようとします。基本クラス内に、引数を指定しないで呼び出すことができるアクセス可能なコンストラクタが存在しない場合、Visual Basic は暗黙の呼び出しを生成できません。このケースでは、そのようなコンストラクタが **ObsoleteAttribute** 属性でマーク付けされているため、Visual Basic はこれを呼び出すことができません。

ObsoleteAttribute を適用することで、任意のプログラミング要素を使用しない要素としてマークできます。これを行う場合は、属性の IsError プロパティを **True** または **False** に設定できます。**True** に設定した場合、コンパイラは要素を使用する試みをエラーとして扱います。**False** に設定した場合、または既定値の **False** を使用した場合、コンパイラは要素を使用する試みに対して警告を発行します。

既定では、**ObsoleteAttribute** の **IsError** プロパティが **False** であるため、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC41004

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. **MyBase.New()** の呼び出し、または **MyClass.New()** の呼び出しを、派生クラスの **Sub New** の最初のステートメントとして定義します。

参照

概念

Visual Basic で使用される属性
属性の適用

型が、'System.Collections.Generic.IEnumerable(Of T)' の複数のインスタンス生成を実装するため、型 '<typename>' の 'For Each' は不適切です。

For Each ステートメントが複数の GetEnumerator メソッドを持つ反復子変数を指定しています。

反復子変数の型は、.NET Framework のいずれかの **Collections** 名前空間の System.Collections.IEnumerable または System.Collections.Generic.IEnumerable インターフェイスを実装する型にする必要があります。それぞれの構築について異なる型引数を使用すれば、1 つのクラスで複数の構築されたジェネリック インターフェイスを実装できます。このようなクラスを反復子変数に使用した場合、この変数は複数の **GetEnumerator** メソッドを持つことになります。その場合、Visual Basic はどのメソッドを呼び出すかを判断できません。

Error ID: BC32096

このエラーを解決するには

- DirectCast または TryCast を使用して、反復子変数の型を、目的の **GetEnumerator** メソッドを定義しているインターフェイスへとキャストします。

参照

関連項目

For Each...Next ステートメント (Visual Basic)

概念

Visual Basic でのインターフェイス実装例

For Loop に指定する変数 '<variablename>' は、終わりの For loop によって既に使用されています。

入れ子になった **For** ループはすべて、一意のコントロール変数を使用する必要があります。

Error ID: BC30069

このエラーを解決するには

- いずれかのコントロール変数の名前を変更します。

参照

関連項目

For...Next ステートメント (Visual Basic)

'For' ループに指定する変数に '<type>' 型は使えません。

使おうとしたループ制御変数のデータ型が無効です。**For** ループの先頭で、開始点、終了点、およびステップ値が、記述されている順に評価されます。これら 3 つの式はすべて、変数の型に暗黙的に変換できる必要があります。**For** ループ変数が **Object** 型である場合、実行時に少なくとも 1 つの式は数値型である必要があります。また、3 つの式はすべて、その中で最も範囲の広い数値型に強制的に変換できる必要があります。

Error ID: BC30337

このエラーを解決するには

- ループ制御変数のデータ型を調べて、有効な型に変更します。

参照

関連項目

For (Visual Basic)
Do...Loop ステートメント (Visual Basic)
For...Next ステートメント (Visual Basic)

'For' の終わりには、対応する 'Next' を指定しなければなりません。

For ステートメントに、対応する **Next** ステートメントがありません。**Next**ステートメントは、**For** ループの最後に使用する必要があります。

Error ID: BC30084

このエラーを解決するには

- この **For** ループが、入れ子になったループの一部である場合は、すべてのループが正しく終了していることを確認します。
- **For** ループの最後に **Next** ステートメントを追加します。

参照

関連項目

For...Next ステートメント (Visual Basic)

'Get' は既に宣言されています。

コード ブロックに複数の **Get** ステートメントが記述されています。**Get** ステートメントは、プロパティに値を取得するために使用する **Get** プロパティ プロシージャを宣言します。

Error ID: BC30443

このエラーを解決するには

- 余分な **Get** ステートメントを削除します。

参照

関連項目

Get ステートメント

プロパティ '<propertyname>' の 'Get' アクセサにアクセスできません。

ステートメントがプロパティの値を取得しようとしましたが、プロパティの **Get** プロシージャへのアクセス許可がありません。

Get ステートメントが Property ステートメントよりも制限の高いアクセス レベルでマーク付けされている場合、プロパティ値の読み取りは次のケースでエラーになります。

- **Get** ステートメントが Private (Visual Basic) でマーク付けされており、呼び出し元のコードがプロパティが定義されたクラスまたは構造体の外側にある場合。
- **Get** ステートメントが Protected (Visual Basic) でマーク付けされており、呼び出し元のコードがプロパティが定義されたクラスまたは構造体の内部にも、派生クラスの内部にもない場合。
- **Get** ステートメントが Friend (Visual Basic) でマーク付けされており、呼び出し元のコードがプロパティが定義されたのと同じアセンブリにない場合。

Error ID: BC31103

このエラーを解決するには

- プロパティが定義されたソース コードを変更できる場合は、**Get** プロシージャをプロパティ自体と同じアクセス レベルで宣言することを検討してください。
- プロパティが定義されたソース コードを変更できない場合、または **Get** プロシージャをプロパティ自体よりも厳しいアクセス レベルで制限する必要がある場合は、プロパティ値を読み取るステートメントをプロパティへのアクセスが可能なコード領域に移動することを検討してください。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

概念

Property プロシージャ

ジェネリック型 '<genericTypeName>' は、1 度しかインポートできません。

Imports ステートメントに、既に別の型パラメータ化でインポートされているジェネリック型が指定されています。

ジェネリック型から構築された型は複数宣言できます。構築された型を宣言しても、ジェネリック型を再定義することにはならないからです。しかし、ジェネリック型を 2 回以上インポートした場合は、複数の定義をしたのと同じことになります。

Error ID: BC32086

このエラーを解決するには

1. 問題の **Imports** ステートメントが含まれているソース ファイル内に、同じジェネリック型を指定している別の **Imports** ステートメントがある場合は、どちらかのステートメントを削除します。
2. 同じジェネリック型を別の型パラメータ化でインポートする必要がある場合は、複数のソース ファイルを使用します。

参照

概念

Visual Basic におけるジェネリック型

ジェネリック パラメータの制約型 <typename> は CLS に準拠していません。

ジェネリック型が `<CLSCompliant(True)>` としてマークされていますが、1 つの型パラメータに対する制約で指定している型が、`<CLSCompliant(False)>` としてマークされているか、マークされていないか、非準拠の型であるため修飾されていません。

型を 共通言語仕様 (CLS) 準拠にするためには、CLS 準拠の型のみを使用する必要があります。このことは、ジェネリック型の型パラメータに対する制約についても当てはまります。

次の Visual Basic データ型は CLS に準拠していません。

- SByte 型 (Visual Basic)
- UInteger データ型
- ULong データ型 (Visual Basic)
- UShort 型 (Visual Basic)

CLSCompliantAttribute 属性をプログラミング要素に適用するときは、その属性の *isCompliant* パラメータを **True** または **False** に設定することで準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40040

このエラーを解決するには

- ジェネリック型が、この特定の型で制限された型パラメータを使用する必要がある場合は、**CLSCompliantAttribute** を削除します。この型は CLS 準拠になりません。
- ジェネリック型を CLS 準拠にする必要がある場合は、この制約の型を、最も近い CLS 準拠型に変更します。たとえば、2,147,483,647 を超える値の範囲が必要でない場合は、**UInteger** の代わりに **Integer** を使用できます。拡張範囲が必要な場合は、**UInteger** の代わりに **Long** を使用できます。

参照

概念

Visual Basic におけるジェネリック型
CLS 準拠コードの記述

省略可能なパラメータ型として使用されるジェネリック パラメータは、クラスの制約がある型でなければなりません。

型パラメータを使用する省略可能なパラメータを定義してプロシージャが宣言されましたが、この型パラメータが参照型に制限されていません。

各省略可能なパラメータには、必ず既定値を指定する必要があります。パラメータが参照型の場合、オプションの既定値は必ず **Nothing** になります。この値はすべての参照型において有効です。しかし、パラメータが値型である場合、その型は Visual Basic で定義済みの基本データ型であることが必要です。この理由は、ユーザー定義の構造体などの複合値型に有効な既定値がないためです。

省略可能なパラメータに型パラメータを使用する場合は、有効な既定値を持たない値型が使用されるのを防ぐために、型パラメータを必ず参照型にする必要があります。つまり、型パラメータを **Class** キーワードまたは特定のクラスの名前を使って制限する必要があります。

Error ID: BC32124

このエラーを解決するには

- 参照型だけを受け取るように型パラメータを制限するか、省略可能なパラメータに型パラメータを使わないようにします。

参照

関連項目

型リスト
Class (Visual Basic)
Nothing (Visual Basic)

概念

Visual Basic におけるジェネリック型
省略可能なパラメータ
値型と参照型

その他の技術情報

構造体 : 独自のデータ型

ジェネリック メソッドは 'Handles' 句を使用することはできません。

ジェネリックな **Sub** プロシージャの宣言に、Handles 句が指定されています。

Handles 句には、**Sub** プロシージャが処理するイベントのリストを指定します。**Sub** プロシージャにイベント ハンドラを定義するためには、プロシージャのシグネチャを、処理を行う各イベントと同じにする必要があります。ジェネリック プロシージャが 2 回以上作成される場合もありますが、Visual Basic はそのシグネチャをコンパイル時に予測できません。このため、Visual Basic はシグネチャを **Handles** 句に指定されたイベントのシグネチャと確実に一致させることができません。

Error ID: BC32080

このエラーを解決するには

- **Sub** プロシージャをジェネリックにする必要がある場合は、**Handles** 句を宣言から削除します。AddHandler ステートメント を使用して、このイベント ハンドラをイベントに関連付けます。
- **Sub** プロシージャが、**Handles** 句を使用してイベントを関連付ける必要がある場合は、Of 句を宣言から削除します。**Handles** は、ジェネリックではないプロシージャに指定する必要があります。

参照

概念

Visual Basic におけるジェネリック型
イベントとイベント ハンドラ

ジェネリック メソッドを COM に公開できません。

1 つ以上のジェネリック プロシージャを含む .NET Framework コンポーネントが COM コンポーネントにエクスポートされています。

コンポーネント オブジェクト モデル (COM: Component Object Model) はジェネリック型をサポートしておらず、ジェネリック型と相互運用することもできません。

Error ID: BC30943

このエラーを解決するには

- .NET Framework コンポーネントからジェネリック プロシージャを削除するか、COM 相互運用にジェネリック プロシージャを使うことを避けます。

参照

概念

Visual Basic におけるジェネリック型

その他の技術情報

COM 相互運用

'As' 句のない関数です。Object の戻り値の型と見なされます。

Function プロシージャで **As** 句が指定されていません。

As 句は、プログラミング要素に関連付けるデータ型を指定します。Function ステートメント (Visual Basic) では、この句を使って、**Function** プロシージャから呼び出し元のコードに返す値のデータ型を指定します。**As** 句を **Function** ステートメントに含めない場合、戻り値のデータ型は既定で **Object** になります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42021

このエラーを解決するには

- **As** 句を **Function** ステートメントに含めて、戻り値のデータ型を指定します。

参照

概念

Function プロシージャ

前の関数の評価がタイムアウトしたため、関数の評価は無効になりました。

エラー メッセージ
前の関数の評価がタイムアウトしたため、関数の評価は無効になりました。関数の評価をもう一度有効にするには、もう一度ステップを実行するか、またはデバッグを再起動してください。

Visual Studio デバッガで、プロシージャ呼び出しが式に指定されていますが、別の評価がタイムアウトしています。

プロシージャ呼び出しがタイム アウトした原因には、無限ループなどが考えられます。詳細については、「For...Next ステートメント (Visual Basic)」を参照してください。

無限ループの特殊なケースが再帰です。詳細については、「再帰プロシージャ」を参照してください。

Error ID: BC30957

このエラーを解決するには

1. 可能であれば、前回の関数の評価がどれかを判断し、タイムアウトした理由を調べます。判断できない場合、このエラーが再度表示される可能性があります。
2. デバッガのステップをもう一度実行するか、デバッガを終了させて再起動します。

参照

処理手順
方法 : 実行を開始する
関連項目
Visual Basic の式
概念
方法 : デバッグの停止と実行の停止
コードのステップ実行の概要
その他の技術情報
Visual Studio でのデバッグ

関数 '<procedurename>' すべてのコード パス上では値を返しません。

エラー メッセージ
関数 '<procedurename>' すべてのコード パス上では値を返しません。結果が使用されるときに、Null 参照の例外が実行時に発生する可能性があります。

Function プロシージャには、値を返さない可能性があるコードへのパスが少なくとも 1 つあります。

Function プロシージャからは、以下の方法のいずれかで値を返すことができます。

- 値を **Function** プロシージャ名に代入し、**Exit Function** ステートメントを実行します。
- 値を **Function** プロシージャ名に代入し、**End Function** ステートメントを実行します。
- 値を Return ステートメント (Visual Basic) に含めます。

値をプロシージャ名に代入しないで、制御を **Exit Function** または **End Function** に渡した場合、プロシージャは戻り値のデータ型の既定値を返します。詳細については、「Function ステートメント (Visual Basic)」の「動作」を参照してください。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42105

このエラーを解決するには

- 制御フローのロジックを調べて、ステートメントが値を返す前に値が割り当てられていることをすべて確認してください。

  常に **Return** ステートメントを使うと、値を返すプロシージャが確実に値を返すようにするのが簡単になります。この場合は、**End Function** の直前のステートメントを **Return** ステートメントにする必要があります。

参照

関連項目
Function ステートメント (Visual Basic)
概念
Function プロシージャ

名前空間の型の完全修飾名に、<number> 文字を超える名前を指定することはできません。

型の完全修飾名は、指定した文字数を超えることができません。

Error ID: BC30031

このエラーを解決するには

- 型の名前を短くしてください。

参照

概念

Visual Basic における型チェック

'Get' ステートメントの終わりには、対応する 'End Get' を指定しなければなりません。

Get プロパティ プロシージャは、必ず **End Get** ステートメントで終了してください。

Error ID: BC30631

このエラーを解決するには

- **Get** プロパティ プロシージャの最後に **End Get** 構成体があることを確認します。

参照

関連項目

Property ステートメント

概念

プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

'Get' ステートメントは現在サポートされていません。

Get ステートメントはサポートされていません。ファイル I/O 機能は **Microsoft.VisualBasic** 名前空間で使用できます。

ファイル操作に **Get** を使うことはできません。プロパティ プロシージャ構文でのみ使用できます。

Error ID: BC30829

このエラーを解決するには

- ファイルを操作するには、**System.IO** のメンバ、**FileSystemObject**、および Visual Basic ランタイム関数を使います。

参照

関連項目

Get ステートメント

その他の技術情報

ドライブ、ディレクトリ、およびファイルの処理

Visual Basic におけるファイル アクセス

'Get' ステートメントは現在サポートされていません (Visual Basic)

Get ステートメントはサポートされていません。通常、ファイル I/O 機能は **Microsoft.VisualBasic** 名前空間で使用できますが、.NET Compact Framework のターゲット バージョンではサポートされていません。

Error ID: BC30767

このエラーを解決するには

- System.IO 名前空間で定義された関数を使用してファイル操作を実行します。

参照

関連項目

Get ステートメント

System.IO

その他の技術情報

Visual Basic におけるファイル アクセス

'Global' の後には、'.' および識別子が必要です。

Global キーワードが名前空間のアクセス以外のコンテキストに指定されています。

Global の目的は、ルート レベルの名前空間へのアクセスが防止された名前空間の構造の内部から、コードがルート レベルの名前空間にアクセスできるようにすることです。

Error ID: BC36000

このエラーを解決するには

- ルート レベルの名前空間にアクセスするには、**Global** キーワードの後にピリオド (.) を記述し、その後に名前空間を指定します。

```
Dim keyInfo As Global.System.ConsoleKeyInfo
```

- ルート レベルの名前空間にアクセスする必要がない場合は、**Global** キーワードを削除します。

参照

関連項目

Global

'Global' はハンドルで許可されていません。ローカル名を指定してください。

Handles 句では、ローカル イベントを参照する必要があります。**Global** キーワードを指定すると、グローバルなプログラミング要素にアクセスできます。

Error ID: BC36002

このエラーを解決するには

- イベントのグローバル インスタンスではなく、ローカル インスタンスを参照するように **Handles** 句を変更します。

参照

関連項目

Global

Handles

その他の技術情報

Visual Basic におけるイベント

'Global' はこのコンテキストで許可されていません。識別子を指定してください。

許可されていないステートメントで **Global** キーワードが使用されています。

Global キーワードを使用すると、コードをコンパイルする名前空間階層の外部に定義されている名前空間にアクセスできるようになります。**Global** キーワードは、修飾パスを .NET Framework クラス ライブラリの最も外側の名前空間レベルでスタートさせます。詳細については、「Global」を参照してください。

Imports や **Namespace** など一部のステートメントは、コードをコンパイルする名前空間から独立しています。これらのステートメントは、ルート レベルの名前空間 (System、Microsoft.VisualBasic など) から始まる完全修飾パスを必要とします。このようなステートメントでは、**Global** キーワードは不必要であり、許可されていません。

Error ID: BC36001

このエラーを解決するには

- ステートメントから **Global** キーワードを削除します。このキーワードは不要です。

参照

関連項目

Global
Imports ステートメント
Namespace ステートメント

概念

参照と Imports ステートメント

'Gosub' ステートメントは現在サポートされていません。

GoSub を使ってプロシージャを呼び出すことはできません。

Error ID: BC30814

このエラーを解決するには

- **GoSub** を使わずに直接プロシージャを呼び出します。次に例を示します。

```
CalculateInterest(Amount, Rate, Time)
```

参照

概念

Visual Basic におけるプロシージャ

'<labelname>' は、このステートメントを含まない 'For' または 'For Each' ステートメントの内側にあるため、'Go to <labelname>' は有効ではありません。

GoTo ステートメントのジャンプ先は現在のコード ブロック内に制限されています。

Error ID: BC30757

このエラーを解決するには

- コードを再構成して **GoTo** ステートメントとラベルが、いずれも **For** ブロック内に入るようにします。

参照

関連項目

GoTo ステートメント
For (Visual Basic)

'<labelname>' は、このステートメントを含まない 'SyncLock' ステートメントの内側にあるため、'Go to <labelname>' は有効ではありません。

SyncLock ブロックに分岐できません。

Error ID: BC30755

このエラーを解決するには

- ラベルを **SyncLock** ブロックの前に移動するようにコードを再構成します。

参照

関連項目

SyncLock ステートメント

'<labelname>' は、このステートメントを含まない 'Try'、'Catch' または 'Finally' ステートメントの内側にあるため、'Go to <labelname>' は有効ではありません。

Try...Catch...Finally ブロックに分岐できません。

Error ID: BC30754

このエラーを解決するには

- ラベルを **Try...Catch...Finally** ブロックの前に移動するようにコードを再構成します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

'<labelname>' は、このステートメントを含まない 'With' ステートメントの内側にあるため、'Go to <labelname>' は有効ではありません。

GoTo ステートメントのジャンプ先は現在のコード ブロック内に制限されています。

Error ID: BC30756

このエラーを解決するには

- コードを再構成して **GoTo** ステートメントとラベルが、いずれも **With** ブロック内に入るようにします。

参照

関連項目

GoTo ステートメント

<linelabel>' は、このステートメントを含まない 'Using' ステートメントの内側にあるため、'GoTo <linelabel>' は有効ではありません。

Using 構造の外部にある **GoTo** ステートメントがこの構造内の行ラベルに分岐しようとしました。

Using...**End Using** 構造の外部からこの構造の内部に分岐することはできません。

Error ID: BC36009

このエラーを解決するには

- **GoTo** ステートメント内の行ラベルを、**For**...**Next**、**For Each**...**Next**、**SyncLock**...**End SyncLock**、**Try**...**Catch**...**Finally**、**With**...**End With**、または **Using**...**End Using** 構造の内部に含まれていないラベルに変更します。

  または

- **GoTo** ステートメントを完全に削除します。**Using**...**End Using** 構造の内部に入るためには、プログラムの制御を **Using** ステートメントに進ませるしか方法はありません。

参照

関連項目

GoTo ステートメント
Using ステートメント (Visual Basic)

'Go to' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

GoTo ステートメントは分岐を行うため、デバッグ コンテキストでは使用できません。

Error ID: BC30133

このエラーを解決するには

- [イミディエイト] ウィンドウで **GoTo** ステートメントを実行しないようにします。

参照

関連項目

[イミディエイト ウィンドウ]

Handles 句には、含んでいる型またはその基本型の 1 つで定義した WithEvents 変数が必要です。

Handles 句に **WithEvents** 変数が指定されませんでした。プロシージャ宣言の末尾に **Handles** キーワードを指定すると、そのプロシージャは、**WithEvents** キーワードを使って宣言されたオブジェクト変数が発生させるイベントを処理します。

Error ID: BC30506

このエラーを解決するには

- 必要な **WithEvents** 変数を指定します。

参照

関連項目

Handles

概念

WithEvents と Handles 句

クラスの 'Handles' は、単一の識別子で限定された 'WithEvents' 変数、'MyBase'、'MyClass' または 'Me' を指定しなければなりません。

イベント ハンドラを指定するには、**Handles** ステートメントで、**WithEvents** キーワードで宣言されたオブジェクト変数または **MyBase** キーワードで修飾されたメンバのいずれかを指定する必要があります。

Error ID: BC31412

このエラーを解決するには

1. **WithEvents** 修飾子を使用して、**Handles** ステートメントで使用する変数を宣言します。
2. 基本クラス内の現在のクラスにイベントの名前を指定します。

参照

関連項目

Handles

WithEvents

その他の技術情報

Visual Basic におけるイベント

モジュール内の 'Handles' には、'WithEvents' 変数を単一の識別子で修飾して指定する必要があります。

イベント ハンドラを指定する場合は、**WithEvents** キーワードを付けて宣言したオブジェクト変数を **Handles** ステートメントに指定してください。

Error ID: BC31418

このエラーを解決するには

- **WithEvents** 修飾子を使用して、**Handles** ステートメントで使用される変数を宣言します。

参照

関連項目

Handles

WithEvents

その他の技術情報

Visual Basic におけるイベント

'Handles' は演算子の宣言で有効ではありません。

Operator ステートメントに Handles キーワードが指定されています。

イベントを処理できるのは **Sub** プロシージャだけです。**Operator** プロシージャでは処理できません。イベント ハンドラの詳細については、「方法 : Visual Basic でイベント ハンドラを呼び出す」を参照してください。

Operator プロシージャには **Public** キーワードと **Shared** キーワードが 2 つとも必要で、変換演算子には **Widening** キーワードか **Narrowing** キーワードのいずれかが必要です。詳細については、「演算子プロシージャ」を参照してください。

Error ID: BC33003

このエラーを解決するには

- このプロシージャでイベントを処理する場合は、**Sub** プロシージャに変更してください。
- このプロシージャに演算子を定義する必要がある場合は、**Handles** キーワードを宣言から削除してください。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

Operator ステートメント

識別子が必要です。

宣言された要素名として認識できないプログラミング要素が、コンテキストで要素名が必要とされる場所にあります。ステートメントの先頭以外の場所で属性が指定されていることが原因の 1 つとして考えられます。

Error ID: BC30203

このエラーを解決するには

- ステートメント内のすべての属性が先頭に配置されているかどうかを確認します。
- ステートメント内のすべての要素名のスペルが正しいかどうかを確認します。

参照

概念

宣言された要素の名前

その他の技術情報

Visual Basic における属性

識別子が長すぎます。

各プログラミング要素の名前、つまり識別子は 1023 文字に制限されています。また、完全限定名は 1023 文字を超過できません。つまり、識別子の文字列全体 (`<namespace>.<...>.<namespace>.<class>.<element>`) の長さは 1023 文字を超過できません。文字列には、メンバ アクセス演算子 (.) も含みます。

Error ID: BC30033

このエラーを解決するには

- 識別子の文字数を減らします。

参照

概念

宣言された要素の名前

'If' の終わりには、対応する 'End If' を指定しなければなりません。

If ステートメントに対応する **End If** ステートメントがありません。**If** ブロックの最後には **End If** ステートメントが必要です。

Error ID: BC30081

このエラーを解決するには

1. この **If** ブロックが、入れ子になった **If** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
2. **If** ブロックの最後に **End If** ステートメントを追加します。

参照

関連項目

If...Then...Else ステートメント (Visual Basic)

'If'、'ElseIf'、'Else'、'End If'、または 'Const' が必要です。

ソース行が # 文字で始まっていますが、有効な条件付きコンパイル ディレクティブが # のすぐ後ろに続いていません。有効なディレクティブは次のとおりです。**#Const**、**#ExternalSource**、**#If**、**#Else**、**#ElseIf**、**#End If**、**#Region**。

Error ID: BC30248

このエラーを解決するには

1. 条件付きコンパイル ディレクティブのスペルが正しいかどうかを確認します。
2. # 文字とディレクティブの間に空白がないことを確認します。
3. # 文字を削除するか、すぐ後ろに有効なディレクティブを追加します。

参照

その他の技術情報

ディレクティブ (Visual Basic)

応答ファイルで指定されているため、/noconfig オプションを無視します。

/noconfig オプションを設定すると、コンパイル時に Vbc.rsp ファイルが使用されなくなります。しかし、コンパイラは他のどの応答ファイルよりも先に Vbc.rsp ファイルを処理するため、応答ファイルに指定された **/noconfig** オプションに従うことができません。

Error ID: BC2025

このエラーを解決するには

1. **/noconfig** オプションを応答ファイルから削除します。
2. コマンド ライン コンパイラを呼び出すときに **/noconfig** オプションを指定します。

参照

関連項目

/noconfig
@ (応答ファイルの指定) (Visual Basic)

呼び出し式またはインデックス式が無効です。

プロシージャ、プロパティ、または配列でない式の後ろに、かっこで囲んだ値を指定しています。

このエラーは次のようなコードで発生します。

```
Dim testVariable As Object = Nothing(1)
```

Nothing は引数やインデックスを取らないので、かっこで囲んだ値を指定できません。

Error ID: BC32303

このエラーを解決するには

- かっことその中の値を削除します。

参照

処理手順

方法 : 値を返すプロシージャを呼び出す
方法 : 値を返さないプロシージャを呼び出す
方法 : 配列に値を格納する
方法 : 配列から値を取得する
方法 : プロパティに値を格納する
方法 : プロパティから値を取得する

実装された型は、インターフェイスでなければなりません。

Implements ステートメントでインターフェイスが指定されていません。クラスが実装できるのはインターフェイスだけです。

Error ID: BC30232

このエラーを解決するには

1. インターフェイス名のスペルが正しいかどうかを確認します。
2. ステートメントでクラスを指定する場合は、**Inherits** ステートメントを使用することを検討してください。

参照

関連項目

Implements ステートメント
Inherits ステートメント

その他の技術情報

Visual Basic の継承

インターフェイス '<interfacename>' に対して実装するクラス '<underlyingclassname>' は '<accesslevel>' であるため、このコンテキストではアクセスできません。

基になるクラスを CoClassAttribute で指定してインターフェイスが宣言されていますが、そのクラスのアクセス レベルが妨げになりコードからインターフェイスにアクセスできません。

CoClassAttribute をインターフェイスに適用すると、基になるクラスがインターフェイスに関連付けられます。コードでは **New** 句を使って、インターフェイスからオブジェクトを直接作成できます。

New 句を使用して基のクラスにアクセスすることはできません。たとえば、クラスが **Private** である場合、コンパイラはこのエラーを生成します。

Error ID: BC31109

このエラーを解決するには

1. 基になるクラスのソース コードを管理する立場にある場合は、クラスのアクセス レベルをコードからアクセスできるレベルに調整します。
2. なんらかの理由で基になるクラスのアクセス レベルを変更できない場合は、**New** 句を削除します。このインターフェイスからオブジェクトを直接作成することはできません。

参照

関連項目

New (Visual Basic)
CoClassAttribute

概念

Visual Basic でのアクセス レベル

インターフェイス <interfacename> に実装するクラス '<classname>' が見つかりません。

相互運用機能アセンブリ内の実装元のクラスが使用できません。

Error ID: BC31094

このエラーを解決するには

1. COM オブジェクトのタイプ ライブラリが有効であることを確認します。
2. タイプ ライブラリ インポータ (Tlbimp.exe) を使用して相互運用機能アセンブリを手動で作成し、その相互運用機能アセンブリへの参照をプロジェクトで追加します。

参照

関連項目

Implements ステートメント
タイプ ライブラリ インポータ (Tlbimp.exe)

その他の技術情報

COM 相互運用

'Implements' は演算子の宣言で有効ではありません。

Operator ステートメントに Implements (Visual Basic) キーワードが指定されています。

インターフェイスのメンバを実装できるのは、**Function** プロシージャか **Sub** プロシージャ、プロパティ、またはイベントだけです。実装の詳細については、「Visual Basic でのインターフェイス実装例」を参照してください。

Operator プロシージャには **Public** キーワードと **Shared** キーワードが 2 つとも必要で、変換演算子には **Widening** キーワードか **Narrowing** キーワードのいずれかが必要です。詳細については、「演算子プロシージャ」を参照してください。

Error ID: BC33004

このエラーを解決するには

- このプロシージャにインターフェイスのメンバを実装する必要がある場合は、**Function** プロシージャか **Sub** プロシージャ、プロパティ、またはイベントに書き換えてください。
- このプロシージャに演算子を定義する必要がある場合は、**Implements** キーワードを宣言から削除してください。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

'Implements' は、モジュールでは有効ではありません。

Implements ステートメントまたは **Implements** 句がモジュールの中にあります。モジュールはインターフェイスを実装できません。

Error ID: BC30231

このエラーを解決するには

- **Implements** ステートメントまたは **Implements** 句を削除するか、またはモジュールをクラスとして再入力します。

参照

関連項目

Implements ステートメント
Module ステートメント

'Implements' ステートメントは、'Inherits' ステートメントの後およびクラス内のすべての宣言の前に記述しなければなりません。

Implements ステートメントが無効な場所で検出されました。

Error ID: BC31053

このエラーを解決するには

- **Inherits** ステートメントの直後、および任意の宣言の前に **Implements** ステートメントを配置します。たとえば、次のようにします。

```
Class Derived
   Inherits Base
   Implements One
   Sub SubOne() Implements One.Method1
      ' Add code to implement the method.
   End Sub
End Class
```

参照

関連項目

Implements (Visual Basic)

その他の技術情報

Visual Basic におけるインターフェイス

'<typename1>' から '<typename2>' への暗黙的な変換です。

式または代入ステートメントが、あるデータ型の値を受け取って、それを別の型に変換しています。変換キーワードが使用されていないため、これは暗黙の変換です。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42016

このエラーを解決するには

- 可能なら、同じデータ型の値を使用して、Visual Basic が型を変換する必要がないようにします。
- **CType** または他のいずれかの変換キーワードを使用して、明示的な変換にします。

参照

関連項目

データ型変換関数

概念

暗黙の型変換と明示的な型変換

'ByRef' パラメータ '<parametername>' の値を、一致する引数に戻してコピーする際の、'<typename1>' から '<typename2>' への暗黙的な変換です。

プロシージャの呼び出しに指定された ByRef 引数の型が、それに対応するパラメータの型と異なります。

引数を **ByRef** で渡す場合、Visual Basic は、参照を渡す代わりに、引数の値をプロシージャ内のローカル変数にコピーする場合があります。このようなケースでは、プロシージャから制御が戻ったとき、Visual Basic はローカル変数の値を呼び出し元のコードの引数にコピーして戻す必要があります。

ByRef の引数の値がプロシージャにコピーされるとき、引数とパラメータの型が同じであれば、変換は必要ありません。しかし、型が異なる場合、Visual Basic は、双方向で変換を実行する必要があります。**CType** またはその他の変換キーワードをプロシージャの引数またはパラメータで使うことはできないため、このような変換は常に暗黙の変換となります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC41999

このエラーを解決するには

- 可能であれば、呼び出し元の引数にプロシージャのパラメータと同じ型を使用します。そうすれば、Visual Basic は変換を行う必要がありません。
- パラメータの型と異なる型の引数を指定してプロシージャを呼び出す必要があるが、値を呼び出し元の引数に戻す必要がない場合は、パラメータを **ByRef** ではなく ByVal で定義します。

参照

概念

Visual Basic におけるプロシージャ
プロシージャのパラメータと引数
引数の値渡しおよび参照渡し
暗黙の型変換と明示的な型変換

構造体メンバの初期化子は、'Shared' メンバおよび定数に対してのみ有効です。

構造体の宣言内で構造体メンバ変数が初期化されました。

Error ID: BC31049

このエラーを解決するには

- 変数の代わりに定数を使用します。
- パラメータ化されたコンストラクタを構造体に追加します。たとえば、次のようにします。

```
Structure TestStruct
   Public t As Integer
   Sub New(ByVal Tval As Integer)
      t = Tval
   End Sub
End Structure
```

参照

処理手順

方法 : 構造体を宣言する

その他の技術情報

Visual Basic の定数と列挙体

初期化エラーです。

プログラムの初期化中にエラーが発生しました。インストールを途中でキャンセルしたことが原因である可能性があります。

Error ID: BC2000

このエラーを解決するには

- インストールし直します。

参照

概念

エラーの種類

インターフェイス内では、'Inherits' ステートメントはすべての宣言の前に記述しなければなりません。

Inherits ステートメントは、クラス定義内の空行やコメントではない 1 行目に記述する必要があります。

Error ID: BC30357

このエラーを解決するには

- **Inherits** ステートメントをすべての宣言の前に移動します。

参照

関連項目

Inherits ステートメント

Inherits ステートメントはクラス内のすべての宣言の前に記述する必要があります。

Inherits ステートメントは、クラス定義の先頭に配置する必要があります。

Error ID: BC30683

このエラーを解決するには

- **Inherits** ステートメントをクラスの先頭に移動します。

参照

関連項目

Inherits ステートメント

その他の技術情報

Visual Basic の継承

'Inherits' は、モジュールでは有効ではありません。

Inherits ステートメントがモジュールの中にあります。モジュールはクラスを継承できません。

Error ID: BC30230

このエラーを解決するには

- **Inherits** ステートメントを削除するか、またはモジュールをクラスとして再入力します。

参照

関連項目

Inherits ステートメント
Module ステートメント

'Inherits' は、'Class' ステートメント内で 1 回のみ使用でき、1 つのクラスのみを指定できます。

同じクラス内に複数の **Inherits** ステートメントが表示されるか、または **Inherits** ステートメントで複数のクラスが指定されます。クラスが継承できるのは 1 つの基本クラスだけです。

Error ID: BC30121

このエラーを解決するには

- 余分な **Inherits** ステートメントを削除し、残りの **Inherits** ステートメントが基本クラスを 1 つだけ指定しているかどうかを確認します。

参照

関連項目

Inherits ステートメント

概念

継承の基本

'System.<classname>' からの継承は、有効ではありません。

クラスは、**System.Array**、**System.Delegate**、**System.Enum**、または **System.ValueType** を継承できません。

Error ID: BC30015

このエラーを解決するには

- **Inherits** ステートメントを削除するか、または有効な基本クラスから継承するように変更します。

参照

概念

継承の基本

オブジェクト変数の宣言

'<typename>' の型に関する情報が、ランタイムに読み込まれていません。

ランタイムによって読み込まれていない型が参照されました。

Error ID: BC30750

このエラーを解決するには

1. 現在のスコープで型が定義されるようにコードを再構成します。
2. メンバが定義されていることと、メンバ名のスペルが正しいことを確認します。
3. モジュール内で宣言されているいずれかのメンバにアクセスしてみます。場合によっては、メンバが宣言されているモジュールが読み込まれていないために、デバッグ環境でメンバを特定できないことがあります。

参照

その他の技術情報

Visual Studio でのデバッグ

型パラメータ制約 '<typeparameter1>' から取得された間接的な制約 '<constraint1>' は、型パラメータ制約 '<typeparameter2>' から取得された間接的な制約 '<constraint2>' と競合しています。

ジェネリック型の宣言に定義された制約に、間接的な制約と組み合わせたことによる矛盾があります。

このエラーは次のようなステートメントで発生します。

```
Public Class testClass(Of t1 As {t2, t3}, t2 As Structure, t3 As Class)
```

間接的な制約である `Structure` と `Class` によって、型パラメータ `t1` に矛盾が生じています。その理由は、**Structure** 制約は対応する型引数が値型であることを要求していますが、**Class** は参照型であることを要求しているからです。

Error ID: BC32109

このエラーを解決するには

- 型パラメータの制約を変更して、制約が矛盾しないようにします。

参照

関連項目

型リスト
Structure (Visual Basic)
Class (Visual Basic)

概念

Visual Basic におけるジェネリック型
値型と参照型

型パラメータ制約 '<typeparameter1>' から取得された間接的な制約 '<constraint1>' は、制約 '<constraint2>' と競合しています。

直接的な制約と間接的な制約が組み合わされているため、競合する制約がジェネリック型に宣言されています。

このエラーは次のようなステートメントで発生します。

```
Public Class testClass(Of t1 As {t2, Class}, t2 As Structure)
```

間接的な制約である `Structure` と直接的な制約である `Class` は、型パラメータ `t1` に関して競合を起こします。**Structure** 制約は型引数が値型であることを要求しますが、**Class** は参照型であることを要求し、これが競合の原因となります。

Error ID: BC32111

このエラーを解決するには

- 型パラメータの制約を変更して、制約が競合しないようにします。

参照

関連項目

型リスト
Structure (Visual Basic)
Class (Visual Basic)

概念

Visual Basic におけるジェネリック型
値型と参照型

<typename>' を含むアセンブリ <assemblyname> バージョン <laterversionnumber> へ間接的に参照しようとしています。

エラー メッセージ
<typename>' を含むアセンブリ <assemblyname> バージョン <laterversionnumber> へ間接的に参照しようとしています。このプロジェクトは、<assemblyname> バージョン <earlierversionnumber> の前のバージョンを参照します。<typename>' を使用するには、<assemblyname> への参照をバージョン <laterversionnumber> 以降に置き換えなければなりません。

式が別のプロジェクトへの間接参照を作成していますが、このプロジェクトは同じアセンブリの以前のバージョンを参照しています。

通常は、アセンブリの最新バージョンだけを使用する必要があります。

Error ID: BC32207

このエラーを解決するには

1. エラー メッセージに示された型名を手掛かりに、同じアセンブリを参照するプロジェクトを特定します。
2. 他のプロジェクトがアセンブリのどのバージョンを参照するのかを確認し、同じバージョンを参照するようにプロジェクトを変更します。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
壊れた参照のトラブルシューティング
概念
プロジェクト参照

<dimensionnumber> 次元のインデックス '<indexnumber>' が範囲外です。

定義されている範囲を超えているインデックスがあります。

Error ID: BC30702

このエラーを解決するには

- 配列の境界を超えないようにプロシージャのロジックを変更します。
- 配列を定義し直して値の範囲を広げます。

参照

その他の技術情報

Visual Basic における配列

モジュールでの '<keyword>' キーワードの使い方が不適切です。

モジュールは、インスタンスを持たず、継承をサポートせず、インターフェイスを実装しません。そのため、次のキーワードはモジュールの宣言に適用されません。

- MustInherit
- NotInheritable
- Overloads
- Private (Visual Basic)
- Protected (Visual Basic)
- Shadows
- Shared (Visual Basic)
- Static (Visual Basic)

Module ステートメント でサポートされるキーワードは、Public (Visual Basic) と Friend (Visual Basic) だけです。

また、Implements (Visual Basic) ステートメントまたは Inherits ステートメント をモジュールのステートメント ブロックで使うことはできません。

既定では、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC42028

このエラーを解決するには

- このプログラミング要素をモジュールにする必要がある場合は、**Public** キーワードまたは **Friend** キーワードだけを宣言内で使用します。特に指定しない場合、モジュールのアクセス レベルは **Friend** になります。
- このプログラミング要素のインスタンスを作成する必要がある場合は、クラスとして宣言します。そうすると、クラス宣言に適用されるキーワードを使用できます。

参照

関連項目

Class ステートメント (Visual Basic)

'Imports' ステートメントは、宣言の前に記述しなければなりません。

ソース ファイル内で、**Imports** ステートメントが宣言ステートメントの後に定義されています。

Imports ステートメントは参照されているプロジェクトとアセンブリから名前空間の名前をインポートします。また、同じプロジェクト内に定義された名前空間の名前もインポートします。**Imports** ステートメントはソース ファイル内で、どの識別子の参照よりも先に定義する必要があります。

Error ID: BC30465

このエラーを解決するには

- **Imports** ステートメントをソースファイルの先頭、つまりすべての宣言ステートメントの前に移動します。

参照

関連項目

Imports ステートメント

Imports エイリアス '<name>' は、ルート名前空間で宣言された '<name2>' と競合しています。

Imports ステートメントで使用されるエイリアス名が、ルート名前空間で宣言された別の名前と競合しています。

Error ID: BC31403

このエラーを解決するには

- **Imports** ステートメントで使用されるエイリアス名に一意の名前を選択します。

参照

関連項目

Imports ステートメント

型 '<typename>' を、アセンブリまたはモジュール '<name>' からインポートできませんでした。

参照されるすべてのライブラリが、互いに互換性があることを確認してください。

Error ID: BC31091

このエラーを解決するには

- 参照される型を含むファイルが破損していないことを確認します。

参照

関連項目

Imports ステートメント

概念

参照と Imports ステートメント

暗黙的な変数 '<variablename>' は '<メッセージ>' であるため、無効です。

デバッグの最中に、無効な暗黙の変数が生成されました。

Error ID: BC36016

このエラーを解決するには

1. 表示されているメッセージから、原因と対処法を調べます。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト製品サポート サービスにご連絡ください。

参照

概念

テクニカル サポート オプション (Visual Studio)

その他の技術情報

製品のサポートとユーザー補助

内部コンパイル エラー : コード ジェネレータが不適切な入力を取得しました。

Visual Basic コンパイラが内部で記述したファイルの 1 つを正しく解釈できませんでした。

Error ID: BC30011

このエラーを解決するには

1. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
2. エラーが再発する場合は、Visual Basic コンパイラを再インストールします。
3. Visual Basic コンパイラを再インストールした後でもエラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

内部コンパイル エラー

式の評価中に内部エラーが発生しました。

Error ID: BC30747

このエラーを解決するには

- Visual Studio を再起動します。

参照

その他の技術情報

製品のサポート

'<typename>' の 'Microsoft.VisualBasic.ComClassAttribute' の 'InterfaceId' および 'EventsId' パラメータに同じ値を指定することはできません。

COMClassAttribute 属性ブロックで、作成イベントと同じグローバル一意識別子 (GUID: globally unique identifier) がインターフェイスに指定されています。これらの識別子を両方指定する場合は、それぞれ別の識別子である必要があります。また、クラス識別子とも異なっている必要があります。

GUID は 16 バイトで構成され、前の 8 バイトは数値、後の 8 バイトはバイナリです。uuidgen.exe などの Microsoft ユーティリティで生成され、一意であることが保証されています。

Error ID: BC32507

このエラーを解決するには

1. COM オブジェクトのインターフェイスおよび作成イベントを識別するために必要な正しい GUID を確認します。
2. **COMClassAttribute** 属性ブロックに示される GUID 文字列が正しくコピーされていることを確認します。

参照

関連項目

ComClassAttribute クラス

概念

Visual Basic で使用される属性

属性の適用

'Interface' の終わりには、対応する 'End Interface' を指定しなければなりません。

Interface ステートメントに対応する **End Interface** ステートメントがありません。インターフェイス定義を終了させるには **End Interface** ステートメントを使用する必要があります。

Error ID: BC30253

このエラーを解決するには

1. **Interface** の各メンバの書式が正しいことを確認します。
2. インターフェイス定義の最後に **End Interface** ステートメントを追加します。

参照

関連項目

Interface ステートメント (Visual Basic)

インターフェイス メンバは、メソッド、プロパティ、イベント、または型定義でなければなりません。

インターフェイスのメンバになるのは、入れ子になった型、メソッド、プロパティ、およびイベントだけです。

Error ID: BC30602

このエラーを解決するには

- 項目がメソッド、プロパティ、イベント、または入れ子になった型 (列挙型など) であることを確認します。

参照

概念

インターフェイス定義

インターフェイスはほかのインターフェイスからのみ継承できます。

インターフェイス以外のものから継承しようとしました。

Error ID: BC30354

このエラーを解決するには

- 基本インターフェイスまたは継承しようとしているインターフェイスが実際にインターフェイスであることを確認します。

参照

関連項目

Interface ステートメント (Visual Basic)

概念

継承の基本

インターフェイス '<interfacename>' は、このクラスによって実装されていません。

このクラスまたは構造体に実装されていないインターフェイスのメンバを、このクラスまたは構造体のメンバが実装しようとしています。

Error ID: BC31035

このエラーを解決するには

- クラスまたは構造体の先頭に **Implements** ステートメントを追加して、適切なインターフェイスを実装します。
- このエラーの発生元となったメンバから **Implements** キーワードを削除します。

参照

関連項目

Implements (Visual Basic)
Implements ステートメント

その他の技術情報

Visual Basic におけるインターフェイス

インターフェイス '<interfacename>' は、それ自体からは継承できません: <メッセージ>

インターフェイス定義内の Inherits ステートメント にそれ自身が指定されています。

インターフェイスは別のインターフェイスを継承できます。継承するインターフェイスのすべてのメンバが提供されるため、それらのメンバを再度定義する必要はありません。このようなインターフェイスは、派生インターフェイスと呼ばれます。また、継承されるインターフェイスは、基本インターフェイスと呼ばれます。

インターフェイスが自身を継承しても、既にすべてのメンバを持っているので、無意味です。

Error ID: BC30296

このエラーを解決するには

1. **Inherits** ステートメントに指定したインターフェイス名のスペルを確認します。
2. 別のインターフェイスから継承するつもりがない場合は、**Inherits** ステートメントを完全に削除します。
3. 表示されたメッセージを読んでヒントを探します。

参照

その他の技術情報

Visual Basic の継承
Visual Basic におけるインターフェイス

インターフェイス '<interfacename>' には既定のプロパティがないため、インデックス処理を実行できません。

インデックス式の結果は、配列型の値、オーバーロードされた既定プロパティのセットを持つ型の値、またはオーバーロードされたプロパティのセットになる必要があります。インターフェイスには既定のメソッドまたはプロパティを 1 つだけ設定できます。これは、インターフェイスが既定のメソッドまたはプロパティを含むインターフェイスを継承できること、またはインターフェイスの定義ブロックが、既定として設定されたメソッドまたはプロパティを含むことができることを意味します。

Error ID: BC30547

このエラーを解決するには

- インターフェイスで既定のプロパティを指定します。

参照

関連項目

Interface ステートメント (Visual Basic)

インターフェイス '<interfacename>' は、この型によって 1 回のみ実装できます。

指定したインターフェイスはこの型によって既に実装されており、再実装できません。

Error ID: BC31033

このエラーを解決するには

- この型を 1 回だけ使用して、このインターフェイスを実装します。

参照

関連項目

Implements (Visual Basic)

その他の技術情報

Visual Basic におけるインターフェイス

整数の定数が必要です。

2 番目の引数が整数リテラルでない **#ExternalSource** ディレクティブがあります。このコンテキストでは、整数リテラルだけが有効です。名前付き定数または列挙型のメンバは無効です。

Error ID: BC30204

このエラーを解決するには

1. リテラルの代わりに、名前付き定数か列挙体のメンバを使用します。
2. **#ExternalSource** ディレクティブの 2 番目の引数として整数リテラルを指定します。

参照

関連項目

#ExternalSource ディレクティブ

'IsNot' には参照型を持つオペランドが必要ですが、このオペランドの値型は '<typename>' です。

式の中で、IsNot 演算子と少なくとも 1 つの値型オペランドが使用されています。

IsNot 演算子は、2 つのオブジェクト参照が別のオブジェクトを参照しているかどうかを判定します。この演算子では 2 つの参照型のポインタ値を比較するため、値型は使用できません。

Error ID: BC31419

このエラーを解決するには

- 2 つの値型の値を比較する場合は、= または <> 比較演算子を使用します。
- 2 つの参照型のポインタを比較する場合は、両方のオペランドでオブジェクト参照を使用していることを確認してください。参照変数や、参照変数のような動作をする要素 (Me キーワードなど) を使用することができます。

参照

処理手順

方法 : 2 つのオブジェクトが等しいかどうかをテストする

概念

Visual Basic における比較演算子

'<typeparametername>' は、クラス制約のない型パラメータであるため、型 '<typeparametername>' の 'IsNot' オペランドは 'Nothing' とのみ比較できます。

型パラメータが IsNot 演算子 のオペランドとして使用されていますが、この型パラメータの定義には Class (Visual Basic) キーワードもなければ制約リストにクラス名も指定されていません。

IsNot は 2 つの参照型を比較して、メモリ内で異なるオブジェクト インスタンスをポイントしているかどうかを判断します。もう一方のオペランドが Nothing (Visual Basic) でない限り、参照型でないオペランドを受け取ることはできません。

Error ID: BC32097

このエラーを解決するには

- この型パラメータに指定される型引数が常に参照型であることを要求できる場合は、**Class** キーワードを追加するか、その型パラメータの制約リストにクラス名を指定します。
- この型パラメータに指定される型引数が常に参照型であることを要求できない場合は、それを **IsNot** 式から削除してください。**IsNot** 演算子を使って、他の参照型との比較を行うことはできません。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型
値型と参照型
Visual Basic における比較演算子

'Is' には参照型を持つオペランドが必要ですが、このオペランドの値型は '<typename>' です。

Is 比較演算子は、2 つのオブジェクト変数が同じインスタンスを参照しているかどうかを判定します。値型に対してこの比較を行うことはできません。

Error ID: BC30020

このエラーを解決するには

- 2 つの値型を比較するには、適切な算術比較演算子または **Like** 演算子を使います。

参照

関連項目

Is 演算子 (Visual Basic)
Like 演算子
比較演算子 (Visual Basic)

'<typeparametername>' は、クラス制約のない型パラメータであるため、型 '<typeparametername>' の 'Is' オペランドは 'Nothing' とのみ比較できます。

型パラメータが Is 演算子 (Visual Basic) のオペランドとして使用されていますが、この型パラメータの定義には Class (Visual Basic) キーワードもなければ制約リストにクラス名も指定されていません。

Is は 2 つの参照型を比較して、メモリ内で同じオブジェクト インスタンスをポイントしているかどうかを判断します。一方のオペランドが Nothing (Visual Basic) でない限り、参照型でないオペランドを受け取ることはできません。

Error ID: BC32052

このエラーを解決するには

- この型パラメータに指定される型引数が常に参照型であることを要求できる場合は、**Class** キーワードを追加するか、その型パラメータの制約リストにクラス名を指定します。
- この型パラメータに指定される型引数が常に参照型であることを要求できない場合は、それを **Is** 式から削除してください。**Is** 演算子を使って、他の参照型との比較を行うことはできません。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型
値型と参照型
Visual Basic における比較演算子

'Is' が必要です。

TypeOf 演算子に **Is** 句がありません。

Error ID: BC30224

このエラーを解決するには

- **TypeOf** 演算子の後ろに **Is** 句を追加します。たとえば、`TypeOf MyControl Is ComboBox` のように記述します。

参照

概念

Visual Basic における比較演算子

遅延バインドの解決です。ランタイム エラーが発生する可能性があります。

オブジェクト型 (Object)に宣言された変数に、オブジェクトが割り当てられています。

変数を **Object** 型で宣言した場合、コンパイラは遅延バインディングを実行する必要がありますが、これによって実行時に余分な処理が発生します。また、アプリケーションがランタイム エラーを起こす可能性もあります。たとえば、Form を **Object** 変数に割り当ててから System.Xml.XmlDocument.NameTable プロパティにアクセスしようとすると、**Form** クラスが `NameTable` プロパティを公開していないので、ランタイムによって MemberAccessException がスローされます。

変数を特定の型に宣言すると、コンパイラはコンパイル時に事前バインディングを実行できます。これによって、パフォーマンスが向上し、特定の型のメンバへのアクセスの制御が可能になります。また、コードも読みやすくなります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42017

このエラーを解決するには

- 可能であれば、変数を特定の型に宣言してください。

参照

概念

事前バインディングと遅延バインディング
オブジェクト変数の宣言

数字を指定するラベルの後には、コロンが必要です。

行番号にはその他のラベルと同じ規則が適用され、コロンを記述する必要があります。

Error ID: BC30801

このエラーを解決するには

- コード行の先頭に番号を記述し、その後ろにコロンを記述します。次に例を示します。

```
400:    X += 1
```

参照

関連項目

GoTo ステートメント

ラベルは、メソッドの外では有効ではありません。

ラベルを追加できるのは、**Sub**、**Function**、Property **Get**、Property **Set** の各プロシージャ内のステートメントだけです。ラベルを追加できるのは実行可能なステートメントだけであり、実行可能なステートメントはすべてプロシージャ内にある必要があります。

Error ID: BC30016

このエラーを解決するには

- ステートメントからラベルを削除するか、またはステートメントをプロシージャ内に移動します。

参照

処理手順

方法 : ステートメントにラベル付けする

関連項目

Sub ステートメント (Visual Basic)
Function ステートメント (Visual Basic)
Get ステートメント
Set ステートメント (Visual Basic)

ラベルは、イミディエイト ウィンドウでは有効ではありません。

行ラベルは分岐を可能にするため、デバッグ コンテキストでは使用できません。

Error ID: BC30134

このエラーを解決するには

- [イミディエイト] ウィンドウで行ラベルを入力しないようにします。

参照

処理手順

方法 : ステートメントにラベル付けする

関連項目

[イミディエイト ウィンドウ]

ラベル '<labelname>' が定義されていません。

参照先の行ラベルまたは行番号は、参照のスコープの中で定義されていません。ラベルは、参照を含むプロシージャの中にある必要があります。

Error ID: BC30132

このエラーを解決するには

- 行ラベルと参照が同じプロシージャ内に存在するように、コードを再構成します。

参照

処理手順

方法 : ステートメントにラベル付けする

関連項目

Property ステートメント

概念

Visual Basic におけるプロシージャ

ラベル '<labelname>' は、現在のメソッドで既に定義されています。

プロシージャ内でラベルの名前が重複しています。プロシージャ内のすべての名前は一意である必要があります。

Error ID: BC30094

このエラーを解決するには

- 重複しているラベルの名前を変更します。

参照

処理手順

方法 : ステートメントにラベル付けする

概念

宣言された要素の名前

キーワードは、識別子として有効ではありません。

要素名が必要な場所にキーワードがあります。このコンテキストでは、変数やプロシージャなどの要素が必要です。

Error ID: BC30183

このエラーを解決するには

- キーワードを有効な要素名に置き換えます。

参照

概念

Visual Basic の演算子および式

キーワードは型の名前を指定しません。

型名が必要な場所でキーワードが使用されています。このコンテキストでは、基本データ型、または型を表す宣言済みの要素名が必要です。

Error ID: BC30180

このエラーを解決するには

- キーワードを有効な型名に置き換えます。

参照

関連項目

Class ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)
Function ステートメント (Visual Basic)
Property ステートメント
Structure ステートメント

先頭の '.' または '!' は、'With' ステートメント内でのみ使用できます。

With ブロックの外部にピリオド (.) または感嘆符 (!) がありますが、その左に式が記述されていません。メンバ アクセス (.) およびディクショナリ メンバ アクセス (**!**) では、そのメンバを含む要素を指定する式が必要です。この式は、アクセサのすぐ左に記述するか、メンバ アクセスを含む **With** ブロックのターゲットとして記述する必要があります。

Error ID: BC30157

このエラーを解決するには

1. **With** ブロックの書式が正しいことを確認します。
2. **With** ブロックがない場合は、アクセサの左側に、メンバを含む定義済みの要素に評価される式を追加します。

参照

関連項目

With...End With ステートメント (Visual Basic)

概念

コード内の特殊文字

アクセスするインスタンスがインターフェイス型であるため、遅延バインドされたオーバーロードの解決は '<procedurename>' に適用されません。

コンパイラがオーバーロードされたプロパティまたはプロシージャへの参照の解決を試みましたが、引数が **Object** 型で、参照元のオブジェクトのデータ型がインターフェイス型であるため、参照が失敗しました。引数が **Object** 型である場合、コンパイラは遅延バインディングで参照を解決します。

このような状況では、コンパイラは基のインターフェイスを調べてではなく、実装しているクラスを調べてオーバーロードを解決します。そのクラスがオーバーロードされたいずれかのクラスの名前を変更していると、コンパイラは名前が違うためにそれをオーバーロードされたクラスと見なしません。このため、コンパイラは名前が変更されたクラスを無視します。よって、このクラスは参照の解決先として正しい場合でも選択されません。

Error ID: BC30933

このエラーを解決するには

- **CType** を使用して、**Object** 型の引数を、呼び出そうとしているオーバーロードのシグネチャに指定された型にキャストします。

  参照元のオブジェクトを基のインターフェイスにキャストしても意味がありません。このエラーを防ぐには、引数をキャストする必要があります。

使用例

オーバーロードされた **Sub** プロシージャを、コンパイル時にこのエラーが発生する方法で呼び出す例を次に示します。

```
Module m1
    Interface i1
        Sub s1(ByVal p1 As Integer)
        Sub s1(ByVal p1 As Double)
    End Interface
    Class c1
        Implements i1
        Public Overloads Sub s1(ByVal p1 As Integer) Implements i1.s1
        End Sub
        Public Overloads Sub s2(ByVal p1 As Double) Implements i1.s1
        End Sub
    End Class
    Sub Main()
        Dim refer As i1 = New c1
        Dim o1 As Object = 3.1415
        ' The following reference is INVALID and causes a compiler error.
        refer.s1(o1)
    End Sub
End Module
```

この例では、コンパイラが定義どおりに `s1` の呼び出しを許可したとすると、`i1` インターフェイスではなく `c1` クラスを介して解決が行われます。この場合、コンパイラは `s2` の名前が `c1` のときの名前と違うため、`i1` の定義によれば `s2` を選択するのが正しいにもかかわらずこれを無視します。

このエラーを修正するには、呼び出しの部分を次のいずれかのコード行に変更します。

```
refer.s1(CType(o1, Integer))
refer.s1(CType(o1, Double))
```

これらのコード行はどちらも、**Object** 型の変数である `o1` を、オーバーロードの際に定義されたいずれかのパラメータ型に明示的にキャストします。

参照

関連項目

CType 関数

概念

プロシージャのオーバーロード

オーバーロードの解決法

'Lib' が必要です。

Declare ステートメントで、**Lib** キーワードを使用せずにライブラリ名が指定されています。

Error ID: BC30218

このエラーを解決するには

- ライブラリ名の前に **Lib** キーワードを追加します。

参照

関連項目

Declare ステートメント

'Let' および 'Set' 代入ステートメントは、現在サポートされていません。

Let および **Set** は代入で使用できません。

Error ID: BC30807

このエラーを解決するには

- 代入ステートメントから **Let** および **Set** を削除します。次に例を示します。

```
X = 2
```

参照

関連項目

Property ステートメント

概念

プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

構造体のメソッド内にあるローカル変数を 'Static' と宣言することはできません。

構造体内のローカル変数には **Static** 修飾子を適用できません。

Error ID: BC31400

このエラーを解決するには

1. ローカル変数から **Static** 修飾子を削除します。
2. より広いスコープを持つ静的変数として変数を宣言します。

参照

関連項目

Static (Visual Basic)

ジェネリック メソッド内のローカル変数には 'Static' を宣言できません。

ジェネリック プロシージャ内のローカル変数の宣言で **Static** を指定しています。

Visual Basic および .NET Framework は、現時点では静的変数とジェネリック プロシージャの組み合わせをサポートしていません。

Error ID: BC32068

このエラーを解決するには

- 変数宣言から **Static** キーワードを削除します。

参照

処理手順

方法 : 変数の有効期間を延長する

関連項目

Static (Visual Basic)

概念

Visual Basic におけるジェネリック型

Visual Basic におけるジェネリック プロシージャ

ローカル変数に、それを含む関数と同じ名前を指定することはできません。

Dim ステートメントで、**Function** プロシージャと同じ名前の変数を指定しています。

Error ID: BC30290

このエラーを解決するには

- 変数宣言を削除するか、または変数の名前を変更します。

参照

概念

同じ名前を持つ複数の変数がある場合に参照を解決する

Function プロシージャ

ローカル変数 '<variablename>' は現在のブロックで既に宣言されています。

同じ名前を持つ 2 つの変数が同じブロック (プロシージャまたは **If...Then** などのコントロール ブロック) 内で宣言されています。

Error ID: BC30288

このエラーを解決するには

- いずれかの宣言を別のブロックに移動するか、またはいずれかの変数の名前を変更します。

参照

概念

同じ名前を持つ複数の変数がある場合に参照を解決する

Visual Basic におけるシャドウ

ローカル変数 '<variablename>' は宣言されているため、参照できません。

プロシージャ内の変数が、宣言される前に使用されています。

Error ID: BC32000

このエラーを解決するには

- 変数が最初に参照される前に宣言を移動します。

参照

概念

Visual Basic での変数宣言

'Line' ステートメントはサポートされていません (Visual Basic コンパイラ エラー)

Line ステートメントはサポートされていません。ファイル I/O 機能を利用するには **Microsoft.VisualBasic.FileSystem.LineInput** を使用します。グラフィックス機能を利用するには **System.Drawing.Graphics.DrawLine** を使用します。

Error ID: BC30830

このエラーを解決するには

1. ファイルにアクセスしている場合は、**Microsoft.VisualBasic.FileSystem.LineInput** を使用します。
2. グラフィックス処理を行っている場合は、**System.Drawing.Graphics.Drawline** を使用します。

参照

関連項目

System.IO Namespace
System.Drawing Namespace

その他の技術情報

Visual Basic におけるファイル アクセス

'Line' ステートメントはサポートされていません (Smart Device/Visual Basic コンパイラ エラー)

Line ステートメントはサポートされなくなりました。通常、ファイル I/O 機能は Microsoft.VisualBasic.FileSystem.LineInput(System.Int32) として使用できますが、.NET Compact Framework のターゲット バージョンではサポートされていません。

Error ID: BC30768

このエラーを解決するには

- ファイル アクセスを実行するには、System.IO 名前空間で定義された関数を使用します。
- グラフィックス処理を行っている場合は、System.Drawing.Graphics.DrawLine を使用します。

参照

関連項目

System.IO
System.Drawing

その他の技術情報

Visual Basic におけるファイル アクセス

行が長すぎます。

ソース テキストの 1 行の文字数は、65,535 文字以内に制限されています。

Error ID: BC30494

このエラーを解決するには

- 行の長さを 65,535 文字以内にします。

参照

概念

エラーの種類

Loop コントロール変数を、プロパティまたは遅延バインディングされたインデックス付き配列に指定することはできません。

For ループを反復処理するために使用する変数は、数値データ型である必要があります。

Error ID: BC30039

このエラーを解決するには

- ループ制御変数の宣言を変更して **Integer**、**Short**、**Long**、**Byte**、**Single**、**Double**、または **Decimal** を指定します。

参照

関連項目

For...Next ステートメント (Visual Basic)

対応する 'Do' に条件がある場合、'Loop' に条件を指定することはできません。

Loop ステートメントに **While** 句または **Until** 句があり、対応する **Do** ステートメントにも同様な句があります。条件は、同じループの **Do** ステートメントと **Loop** ステートメントのいずれか 1 方のみに指定できます。

Error ID: BC30238

このエラーを解決するには

- **Do** ステートメントまたは **Loop** ステートメントから **While** 句および **Until** 句を削除します。

参照

関連項目

Do...Loop ステートメント (Visual Basic)

Loop ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Loop ステートメントは [イミディエイト] ウィンドウで使用できません。

Error ID: BC30708

このエラーを解決するには

- デバッグ コードから **Loop** ステートメントを削除します。

参照

関連項目

Loop

その他の技術情報

Visual Studio でのデバッグ

'Loop' の前には、対応する 'Do' を指定しなければなりません。

Loop ステートメントに対応する **Do** ステートメントがありません。**Loop** の前には、対応する **Do** ステートメントを指定しなければなりません。

Error ID: BC30091

このエラーを解決するには

1. この **Do** ループが、入れ子になった **Do** ループ セットの一部である場合は、すべてのループが正しく終了していることを確認します。
2. **Do** ループ内の他の制御構造がそれぞれ正しく終了していることを確認します。
3. この **Do** ループの書式が正しいことを確認します。

参照

関連項目

Do...Loop ステートメント (Visual Basic)

構造体のメンバを '<specifier>' として宣言できません。

Structure ステートメント内のメンバに対して無効な指定子を使用しました。

Error ID: BC30435

このエラーを解決するには

- 指定子を削除します。

参照

概念

構造体とクラス

その他の技術情報

構造体 : 独自のデータ型

モジュールのメンバを '<specifier>' として宣言できません。

Module ステートメント内のメンバに対して無効な指定子を使用しました。モジュールはインスタンス化できず、継承をサポートしません。また、インターフェイスを実装できません。

Error ID: BC30436

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Module ステートメント

メンバ '<membername1>' は、ベースクラス '<baseclassname>' のメンバと競合する '<implicitmembername>' を暗黙的に宣言しています。

エラー メッセージ
メンバ '<membername1>' は、ベースクラス '<baseclassname>' のメンバと競合する '<implicitmembername>' を暗黙的に宣言しています。メンバ'を 'Overloads' として宣言することはできません。

派生クラスのプロパティが、基本クラスのメンバと同じ名前で暗黙のメンバを生成し、Overloads キーワードを指定しています。

オーバーロードは、同じクラスにあるプロパティまたはプロシージャに複数のバージョンを定義するために使用されます。基本クラスのメンバが既に **Overloads** を指定していない限り、基本クラスのメンバにさらにバージョンを定義することはできません。競合している基本クラス メンバは **Overloads** を指定しないので、コンパイラでは、このプロパティが暗黙の基本クラス メンバの Shadows であると見なします。

Visual Basic コンパイラは、宣言されている特定のプログラミング要素に対応する暗黙のメンバを作成します。これらの暗黙、つまり合成のメンバの説明を次の表に示します。

宣言された要素	暗黙的に作成されるメンバ
列挙値	`value__` メンバ
イベント	`add_<eventname>` プロシージャ `remove_<eventname>` プロシージャ `<eventname>Event` フィールド `<eventname>EventHandler` デリゲート
プロパティ	`get_<propertyname>` プロシージャ `set_<propertyname>` プロシージャ
My.Form メンバ、**My.WebService** メンバ、または、MyGroupCollectionAttribute 属性でマークされたクラスのメンバ	`m_<variablename>` **Static** 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ
WithEvents 変数	`_<variablename>` 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ

名前の競合が発生する可能性があるため、これらの暗黙のメンバと同じフォームを使用している宣言されたプログラミング要素に名前を付けることは避ける必要があります。たとえば、`get_` または `set_` で始まる要素名の使用は避けてください。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40022

このエラーを解決するには

- 基本クラス メンバを隠す、つまりシャドウする必要がある場合は、プロパティ宣言内の Overloads キーワードを Shadows キーワードに置き換えます。
- 基本クラス メンバをシャドウする必要がない場合は、プロパティ名を変更して上記の表に示した名前の競合を回避します。

参照

概念

宣言された要素の名前

メンバ '<membername1>' は、ベースクラス '<baseclassname>' でメンバ '<membername2>' に対して暗黙的に宣言されたメンバと競合する '<implicitmembername>' を暗黙的に宣言します。

エラー メッセージ
メンバ '<membername1>' は、ベースクラス '<baseclassname>' でメンバ '<membername2>' に対して暗黙的に宣言されたメンバと競合する '<implicitmembername>' を暗黙的に宣言します。そのため、このメンバを 'Shadows' で宣言する必要があります。

派生クラスのメンバが、基本クラスの暗黙のメンバと同じ名前で暗黙のメンバを生成しています。暗黙のメンバに Overloads が指定されていないため、コンパイラはこのメンバが基本クラスの暗黙のメンバを Shadows していると見なします。このメンバに **Shadows** キーワードを明示的に指定すると、コードが読みやすくなり、エラーを起こしにくくなります。

Visual Basic コンパイラは、宣言されている特定のプログラミング要素に対応する暗黙のメンバを作成します。これらの暗黙のメンバ、つまり合成メンバを次の表にまとめます。

宣言された要素	暗黙的に作成されるメンバ
列挙値	`value__` メンバ
イベント	`add_<eventname>` プロシージャ `remove_<eventname>` プロシージャ `<eventname>Event` フィールド `<eventname>EventHandler` デリゲート
プロパティ	`get_<propertyname>` プロシージャ `set_<propertyname>` プロシージャ
My.Form メンバ、**My.WebService** メンバ、または、MyGroupCollectionAttribute 属性でマークされたクラスのメンバ	`m_<variablename>` **Static** 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ
WithEvents 変数	`_<variablename>` 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ

名前の競合が発生する可能性があるため、これらの暗黙のメンバと同じフォームを使用している宣言されたプログラミング要素に名前を付けることは避ける必要があります。たとえば、`get_` または `set_` で始まる要素名の使用は避けてください。

このメッセージは既定では警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40018

このエラーを解決するには

- 基本クラスの暗黙のメンバを隠ぺい、つまりシャドウする必要がある場合は、派生クラスのメンバの宣言に Shadows キーワードを指定します。
- 基本クラスの暗黙のメンバをシャドウする必要がない場合は、上記の表にある名前との衝突を避けるため、派生クラスのメンバの名前を変更します。

参照

概念

宣言された要素の名前

メンバ '<membername1>' は、ベース型 '<basetypename>' でメンバ '<membername2>' に対して暗黙的に宣言されたメンバと競合しています。'Overloads' として宣言することはできません

派生クラスのプロパティまたはプロシージャが、基本クラスの暗黙のメンバと同じ名前を使用し、Overloads キーワードを指定しています。

オーバーロードはプロパティまたはプロシージャを、同じクラスの中で複数の形式で定義するために使います。基本クラスのメンバに、**Overloads** が指定されていなければ、この基本クラスのメンバの別の形式を定義できません。暗黙のメンバに **Overloads** が指定されていないため、コンパイラはこのプロパティまたはプロシージャが基本クラスの暗黙のメンバを Shadows すると見なします。

Visual Basic コンパイラは、宣言されている特定のプログラミング要素に対応する暗黙のメンバを作成します。これらの暗黙のメンバ、つまり合成メンバを次の表にまとめます。

宣言された要素	暗黙的に作成されるメンバ
列挙値	`value__` メンバ
イベント	`add_<eventname>` プロシージャ `remove_<eventname>` プロシージャ `<eventname>Event` フィールド `<eventname>EventHandler` デリゲート
プロパティ	`get_<propertyname>` プロシージャ `set_<propertyname>` プロシージャ
My.Form メンバ、**My.WebService** メンバ、または、MyGroupCollectionAttribute 属性でマークされたクラスのメンバ	`m_<variablename>` **Static** 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ
WithEvents 変数	`_<variablename>` 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ

名前の競合が発生する可能性があるため、これらの暗黙のメンバと同じフォームを使用している宣言されたプログラミング要素に名前を付けることは避ける必要があります。たとえば、`get_` または `set_` で始まる要素名の使用は避けてください。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC40023

このエラーを解決するには

- プロパティまたはプロシージャの名前を、前述の表に示した名前と競合しないように変更します。

参照

概念

宣言された要素の名前

メンバ '<membername>' は、型パラメータと同じ名前のメンバ '<implicitmembername>' を暗黙的に定義します。

ジェネリック クラスのメンバが、クラスの型パラメータと同じ名前を持つ暗黙のメンバを生成しています。

Visual Basic コンパイラは、宣言されている特定のプログラミング要素に対応する暗黙のメンバを作成します。これらの暗黙または合成のメンバの説明を次の表に示します。

宣言された要素	暗黙的に作成されるメンバ
列挙値	`value__` メンバ
イベント	`add_<eventname>` プロシージャ `remove_<eventname>` プロシージャ `<eventname>Event` フィールド `<eventname>EventHandler` デリゲート
プロパティ	`get_<propertyname>` プロシージャ `set_<propertyname>` プロシージャ
My. コレクション変数	`m_<variablename>` **Static** 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ
WithEvents 変数	`_<variablename>` 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ

名前の競合が発生する可能性があるため、これらの暗黙のメンバと同じフォームを使用している宣言されたプログラミング要素に名前を付けることは避ける必要があります。たとえば、`get_` または `set_` で始まる要素名の使用は避けてください。

Error ID: BC32070

このエラーを解決するには

- 型パラメータの名前を変更できる場合は、前述の表に示した名前との衝突を避けるために、型パラメータの名前を変更します。
- 型パラメータの名前を変更できない場合は、前述の表に示した名前との衝突を避けるために、クラス メンバの名前を変更します。

参照

関連項目

型リスト

概念

宣言された要素の名前

Visual Basic におけるジェネリック型

メンバ '<membername>' は、ベース型 '<basetypename>' のメンバ '<membername>' と競合しているため 'Overloads' と宣言してはなりません。

プロパティまたはプロシージャに Overloads キーワードを使用して、既存のプロパティまたはプロシージャを同じ名前で再度宣言していますが、既存のプロパティまたはプロシージャが基本クラスにあります。

オーバーロードはプロパティまたはプロシージャを、同じクラスの中で複数の形式で定義するために使います。基本クラスのメンバに、基本クラス内で Overloads が指定されていなければ、このメンバの別の形式を定義できません。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40021

このエラーを解決するには

- 基本クラスのメンバに対して別の形式を定義する必要があり、基本クラスのソース コードへのアクセスが許可されている場合は、基本クラスの定義に Overloads キーワードを追加します。
- 基本クラスのソース コードへのアクセスが許可されていない場合は、派生クラス内でメンバをオーバーロードできません。**Overloads** キーワードを削除します。
- 基本クラスのメンバの別の形式を定義するのではなく、このメンバを置き換える場合は、**Overloads** キーワードではなく Overrides キーワードを使います。
- 基本クラスのメンバを、派生クラスの新しいメンバで隠ぺいする場合は、**Overloads** キーワードではなく、Shadows キーワードを使います。

参照

概念

プロシージャのオーバーロード
継承の基本

メンバ '<membername>' がクラス '<classname>' に見つかりません。

エラー メッセージ
'<membername>' メンバが '<classname>' クラスに見つかりません。このエラーは通常、'Microsoft.VisualBasic.dll' のバージョンが一致していない場合に発生します。

定義済みのメンバが見つかりませんでした。

Error ID: BC31097

このエラーを解決するには

1. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
2. エラーが再発する場合は、作業内容を保存し、Visual Studio を再起動します。
3. エラーが再発する場合は、Visual Basic を再インストールします。
4. 再インストール後もエラーが発生する場合は、マイクロソフト プロダクト サポート サービスまでご連絡ください。

参照

その他の技術情報
製品のサポートとユーザー補助

クラス '<classname>' は同じ名前とシグネチャが指定された複数のメンバを含むため、このシグネチャに一致するメンバ '<classname>.<procedurename>' をオーバーライドすることはできません: <signaturelist>

プロシージャまたはプロパティが、継承されたプロシージャまたはプロパティをオーバーライドしようとしましたが、コンパイラによって、同じ名前とシグネチャを持つ複数の形式の基本プロシージャまたは基本プロパティが検出されました。

次のスケルトン宣言に示すように、このエラーは、ジェネリック型が構築された状況で発生する可能性があります。

```
Public Class baseClass(Of t)
    Public Overridable Sub doSomething(ByVal inputValue As String)
    End Sub
    Public Overridable Sub doSomething(ByVal inputValue As t)
    End Sub
End Class
Public Class derivedClass
    Inherits baseClass(Of String)
    Overrides Sub doSomething(ByVal inputValue As String)
    End Sub
End Class
```

`derivedClass` は 型パラメータである `t` に **String** を指定する `baseClass` を継承するため、`derivedClass` に関する限り、`baseClass` にある `doSomething` の 2 つのバージョンは同じシグネチャを受け取ることになります。その結果、コンパイラはどちらのバージョンをオーバーライドするべきかが判断できません。

Error ID: BC30935

このエラーを解決するには

- 基本クラスに指定する型引数を変更して、メンバのプロシージャまたはプロパティの 1 つ以上のシグネチャが同じにならないようにします。

  または

- 使用中の型引数のセットを持つ基本クラスを継承する必要がある場合は、このエラー メッセージに表示されているプロシージャまたはプロパティをオーバーライドしないでください。

参照

関連項目

Overridable

Overrides

概念

プロパティとメソッドのオーバーライド

クラス '<interfacename>' は同じ名前とシグネチャが指定された複数のメンバを含むため、このシグネチャに一致するインターフェイス '<interfacename>.<procedurename>' を実装することはできません: <signaturelist>

プロシージャまたはプロパティが、実装されたインターフェイスの中で定義されているプロシージャまたはプロパティを実装しようとしましたが、同じ名前とシグネチャを持つインターフェイス プロシージャまたはプロパティが複数見つかりました。

このエラーは、次のスケルトン宣言に示すように、構築ジェネリック型を使用している状況で発生する可能性があります。

```
Public Interface baseInterface(Of t)
    Sub doSomething(ByVal inputValue As String)
    Sub doSomething(ByVal inputValue As t)
End Class
Public Class implementingClass
    Implements baseInterface(Of String)
    Sub doSomething(ByVal inputValue As String) _
        Implements baseInterface(Of String).doSomething
    End Sub
End Class
```

`implementingClass` は、`baseInterface` を実装するときに型パラメータ `t` に **String** を渡しているので、`implementingClass` に関する限りは、`baseInterface` 内の 2 つのバージョンの `doSomething` がまったく同じシグネチャを持つことになります。そのため、コンパイラはどちらのバージョンを実装するべきか判断できません。

Error ID: BC30937

このエラーを解決するには

- 基本クラスに指定する型引数を変更して、メンバのプロシージャまたはプロパティの 1 つ以上のシグネチャが同じにならないようにします。

  または

- この基本クラスを実装しないようにします。個々のメンバを実装する必要があるため、現在使用している一連の型引数でこれを実装することはできません。

参照

関連項目

Implements (Visual Basic)
Implements ステートメント

概念

Implements キーワードおよび Implements ステートメント

'Me' を代入式のターゲットにすることはできません。

Me キーワードは常に、コードが実行されているクラスの特定のインスタンスを表します。この値は変更できません。

Error ID: BC30062

このエラーを解決するには

- **Me** に値を代入するステートメントを削除します。

参照

関連項目

Me

概念

オブジェクト変数の代入

エラーの最大数が制限を超えました。

コンパイル エラーの最大数は 100 です。コンパイラは、この制限値を設定することによって、エラー レポートのサイズを制御しています。

Error ID: BC30041

このエラーを解決するには

- コンパイルによって生成されたその他のエラーを見直して、できるだけ多くのエラーを解決します。ソース コードに 100 個よりも多いエラーが含まれている場合、今表示されていないエラーが次回コンパイル時に表示される場合があります。

参照

概念

Visual Basic での警告の構成

その他の技術情報

Visual Basic コンパイラ

警告の最大数が制限を超えました。

コンパイル警告の最大数は 100 です。コンパイラは、この制限値を設定することによって、エラー レポートおよび警告レポートのサイズを制御しています。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42206

このエラーを解決するには

- コンパイルによって生成されたその他の警告を見直して、できるだけ多くの警告を解決します。ソース コードに 100 個よりも多い警告が含まれている場合、今表示されていない警告が次回コンパイル時に表示される場合があります。

参照

概念

Visual Basic での警告の構成

その他の技術情報

Visual Basic コンパイラ

'<operatorsymbol2>' には対応する '<operatorsymbol1>' 演算子が必要です。

演算子が定義されていますが、それと対になる必須の演算子が定義されていません。

次の演算子は一対として定義する必要があります。

- = と <>
- > と <
- >= と <=
- **IsTrue** と **IsFalse**

クラスまたは構造体の中で上記の演算子のいずれかを定義した場合は、対になる演算子を同じクラスまたは構造体の中で定義する必要があります。

対になる演算子をコード内で明示的に使用しない場合でも、Visual Basic の処理ではその演算子が必要になります。たとえば、For...Next ステートメント (Visual Basic) の中でカウンタ変数として使用するクラスまたは構造体を定義した場合、Visual Basic は、+ 演算子に加えて >= 演算子と <= 演算子の両方を必要とします。

Error ID: BC33033

このエラーを解決するには

- 同じクラスまたは構造体の中で対になる演算子を定義します。Visual Basic はこの演算子を予想外の状況で使用する可能性があるので、適切な処理を行うよう定義してください。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ
Visual Basic の演算子および式

'<methodname>' メソッドにはリンク デマンドが指定されていますが、このメソッドは、リンク デマンドが指定されていない以下のメソッドをオーバーライドまたは実装します。セキュリティに問題が発生する可能性があります。

セキュリティのリンク確認要求アクションがメソッドに追加されています。ただし、このメソッドはリンク確認要求を持たないメソッドをオーバーライドまたは実装しています。そのため、オーバーライドされたメソッドまたは実装されたメソッドを、不十分なアクセス許可で呼び出すことができ、セキュリティ上の問題を引き起こす可能性があります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42200

このエラーを解決するには

- オーバーライドまたは実装されているメソッドにリンク確認要求を追加します。

参照

概念

リンク確認要求
Demand と LinkDemand
セキュリティの最適化

メソッド '<methodname>' に、デリゲート '<delegatename>' と同じシグネチがありません。

メソッドのシグネチャと使用しようとしているデリゲートのシグネチャが一致していません。**Delegate** ステートメントでは、デリゲート クラスのパラメータの型と戻り値の型を定義します。パラメータ、パラメータの型、および戻り値の型が一致するプロシージャを使って、このデリゲート型のインスタンスを作成できます。各デリゲート クラスでは、オブジェクト メソッドの仕様を渡すコンストラクタを定義します。

Error ID: BC30408

このエラーを解決するには

- シグネチャを調べて一致しない個所を見つけ、修正します。

参照

関連項目

Delegate ステートメント

メソッド '<methodname>' でイベント '<eventname>' を処理できません。指定されているシグネチャが異なります。

指定したイベント ハンドラのパラメータまたはパラメータの型が、指定したイベントについて定義されたパラメータまたはパラメータの型と一致しません。

Error ID: BC31029

このエラーを解決するには

- パラメータ リストが一致するように、イベントまたはイベント ハンドラを再定義します。

参照

その他の技術情報

Visual Basic におけるイベント

構造体のメソッドを、'Protected' または 'Protected Friend' に宣言することはできません。

構造体のメソッドを **Protected** または **Protected Friend** として宣言しました。

Error ID: BC31067

このエラーを解決するには

- 構造体内のメソッドを **Public** または **Private** として宣言します。

参照

関連項目

Structure ステートメント
Protected (Visual Basic)
Friend (Visual Basic)

メソッド宣言ステートメントは論理行の最初のステートメントでなければなりません。

Function ステートメントまたは **Sub** ステートメントの前にコロン (:) ステートメント区切り記号があります。**Function** および **Sub** はソース行の唯一のステートメントでなければなりません。

Error ID: BC32009

このエラーを解決するには

- 複数のステートメントをそれぞれの行に分割します。

参照

処理手順

方法 : コード内でステートメントを分割および連結する

関連項目

Sub ステートメント (Visual Basic)

Function ステートメント (Visual Basic)

メソッドに ParamArray と Optional パラメータの両方を指定することはできません。

ParamArray 引数は自動的に省略可能であり、プロシージャ宣言内で唯一の省略可能な引数である必要があります。先行する引数はすべて必須である必要があります。

Error ID: BC30046

このエラーを解決するには

- 宣言内の省略可能な引数を削除します。

参照

関連項目

ParamArray

概念

パラメータ配列

メソッドに 'Try' ステートメントと 'On Error' または 'Resume' ステートメントの両方を含むことはできません。

Try ステートメントと **On Error** または **Resume** が組み合わせて使用されています。

Error ID: BC30544

このエラーを解決するには

- **On Error** または **Resume** を削除します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

メソッドの呼び出しが値を返しませんでした。

メソッドの呼び出しで予期した値が返されませんでした。

Error ID: BC30723

このエラーを解決するには

- 呼び出したメソッドが、値を返す関数であることを確認します。

参照

その他の技術情報

Visual Studio でのデバッグ

メソッドの引数は、かっこで囲む必要があります。

Visual Basic の最近のバージョンでは、プロシージャ呼び出しを制御する規則が単純になっています。すべての引数はかっこで囲む必要があります。

Error ID: BC30800

このエラーを解決するには

- 引数リストをかっこで囲みます。次に例を示します。

```
MySub(ArrayIndex, NewValue)
```

参照

概念

プロシージャ呼び出しシーケンス (Visual Basic 6.0 ユーザー向け)
プロシージャのパラメータと引数

メソッド '<methodname>' は、インターフェイス '<interfacename>' で既に宣言されています。

インターフェイス メソッドを 2 回宣言したため宣言が重複しています。

Error ID: BC30377

このエラーを解決するには

- 重複している宣言を削除します。

参照

概念

Visual Basic でのインターフェイス実装例

'MustOverride' メソッド '<methodname>' を、'MyClass' で呼び出すことはできません。

MyClass は **Me** と同等ですが、MyClass で呼び出されるメソッドは、すべて **NotOverridable** として扱われます。

Error ID: BC30614

このエラーを解決するには

- **MustOverride** 修飾子を削除するか、または基本クラスでメソッドを宣言し、そのクラスを継承してオーバーライドします。

参照

関連項目

MustOverride
MustInherit
NotOverridable

別の部分宣言で 'NotInheritable' と宣言された部分型であるため、'MustOverride' を '<procedurename>' で指定することはできません。

プロシージャまたはプロパティが、複数の部分宣言に定義されたクラスの内部に **MustOverride** を指定して宣言されていますが、そのクラスに対する **NotInheritable** が部分定義の 1 つに指定されています。

クラスを複数の部分宣言に分割して定義した場合、コンパイラはそのクラスを、各部分宣言すべての結合体として扱います。これはメンバだけでなく、実装、継承、およびアクセス レベルに対しても同じです。

プロシージャまたはプロパティをオーバーライドするには、クラスが基本クラスを継承していることが必要です。したがって、基本クラスのプロシージャやプロパティに **MustOverride** を指定するには、そのクラスに **MustInherit** を指定する必要があります。**MustInherit** と **NotInheritable** は互いに矛盾しているので、同じクラスにこの 2 つを指定することはできません。

Error ID: BC30927

このエラーを解決するには

- プロパティまたはプロシージャをオーバーライドする必要がある場合は、**NotInheritable** キーワードを部分宣言から削除します。
- クラスを **NotInheritable** にする必要がある場合は、プロシージャまたはプロパティから **MustOverride** キーワードを削除します。継承できないクラスをオーバーライドすることはできません。

参照

関連項目

Partial (Visual Basic)
MustOverride
MustInherit
NotInheritable

概念

継承の基本

'MustInherit' は、その他の partial 型の 1 つに指定された 'NotInheritable' と組み合わせて使用できないので、partial 型 '<partialtypename>' に指定できません。

クラスが複数の部分宣言に定義されていますが、その 1 つに **MustInherit** が指定され、別の 1 つに **NotInheritable** が指定されています。

クラスを複数の部分宣言に分割して定義した場合、コンパイラはそのクラスを、各部分宣言すべての結合体として扱います。これはメンバだけでなく、実装、継承、およびアクセス レベルに対しても同じです。

抽象であると同時にシールであるクラスを作成することはできません。つまり、継承を要求しながら禁止することはできません。したがって、同じクラスに **MustInherit** と **NotInheritable** の両方は指定できません。

Error ID: BC30926

このエラーを解決するには

- クラスが継承を要求するのか、継承を禁止するのか、またはそのどちらでもないかを決定し、それに応じて不適切なキーワードを削除します。

参照

関連項目

Partial (Visual Basic)
MustInherit
NotInheritable

概念

継承の基本

モジュールを汎用にはできません。

Module ステートメントに **Of** キーワードを使用して、ジェネリック型パラメータが定義されています。

ジェネリックなクラス、構造体、インターフェイス、プロシージャ、およびデリゲートは定義して使用できますが、ジェネリックなモジュールは定義できません。

Error ID: BC32073

このエラーを解決するには

1. **Module** ステートメントから **Of** キーワードを削除します。
2. ジェネリック モジュールの機能が必要な場合は、ジェネリック クラスであればそれに最も近い機能を使用できます。**Module** ステートメントではなく、Class ステートメント (Visual Basic) を使用してください。

参照

処理手順

方法 : 複数のデータ型に同一の機能を提供できるクラスを定義する

関連項目

Module ステートメント

Of

概念

Visual Basic におけるジェネリック型

モジュールを '<specifier>' として宣言することはできません。

無効なアクセス修飾子でモジュールが宣言されました。

Error ID: BC31052

このエラーを解決するには

- アクセス修飾子を **Public**、**Friend** などの有効なアクセス修飾子に置き換えます。

参照

関連項目

Module ステートメント

'Module' ステートメントは、ファイルまたは名前空間レベルでのみ発生します。

Module ステートメントは、ソース ファイルの先頭に記述する必要があります。つまり、**Option** ステートメント、**Imports** ステートメント、グローバル属性、および名前空間宣言の直後に、その他のすべての宣言より先に記述する必要があります。

Error ID: BC30617

このエラーを解決するには

- **Module** ステートメントを名前空間宣言またはソース ファイルの先頭に移動します。

参照

関連項目

Module ステートメント

'Module' ステートメントの終わりには、対応する 'End Module' を指定しなければなりません。

モジュール ブロックの最後には、**EndModule** ステートメントが必要です。

Error ID: BC30625

このエラーを解決するには

- **Module** ブロックの最後に **End Module** ステートメントを追加します。

参照

関連項目

End (Visual Basic)

モジュール属性 '<attributename>' は、有効ではありません : <error>

共通言語ランタイムは、モジュールの属性を受け入れません。

Error ID: BC30130

このエラーを解決するには

1. 属性を宣言から削除するか、またはコンテキストに対して有効な属性に置き換えます。
2. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。

参照

処理手順

方法 : 独自の属性を定義する

モジュール '<modulename>' を型として使用することはできません。

モジュールを型として使おうとしました。**Get** 型として以外にモジュールを使うことはできません。

Error ID: BC30371

このエラーを解決するには

- 有効な型として使用できるものにモジュールを置き換えます。

参照

関連項目

Module ステートメント

クラス '<classname>' の 'Microsoft.VisualBasic.ComClassAttribute' は、<type> '<typename>' にある同じ名前のメンバと競合する <type> '<membername>' を暗黙的に宣言します。

COMClassAttribute属性ブロックを使用しているクラスで、基本クラスのメンバと同じ名前のインターフェイスが暗黙的に定義されています。この場合、インターフェイスの名前は基本クラスのメンバをシャドウする必要があります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC42101

このエラーを解決するには

1. 基本クラスのメンバを隠す場合は、**ComClassAttribute** 属性ブロックで `InterfaceShadows:=True` を設定します。
2. 基本クラスのメンバを隠さない場合は、クラスの名前を変更します。

参照

関連項目

ComClassAttribute クラス
ComClassAttribute.InterfaceShadows プロパティ

概念

Visual Basic で使用される属性
属性の適用

'Microsoft.VisualBasic.ComClassAttribute' がクラス '<classname>' に指定されましたが、'<classname>' には COM に公開できるパブリック メンバが含まれていないため、COM インターフェイスは生成されません。

COMClassAttribute 属性ブロックを使用しているクラスで、**Public** のプロパティやメソッドが定義されていません。クラスを COM オブジェクトとして公開する場合は、そのプロパティとメソッドを **Public** アクセスで宣言する必要があります。

このメッセージは既定では警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40011

このエラーを解決するには

- クラス内の複数のプロパティまたはメソッドに **Public** キーワードを追加するか、**COMClassAttribute** 属性ブロックを削除します。

参照

関連項目

Public (Visual Basic)
ComClassAttribute クラス

概念

Visual Basic で使用される属性
属性の適用

'Microsoft.VisualBasic.ComClassAttribute' ジェネリックまたはジェネリック型に含まれるクラスには適用できません。

クラスが ComClassAttribute で宣言されていますが、このクラスはジェネリックであるか、ジェネリック クラスまたはジェネリック構造体に含まれています。

COM 相互運用で使用できるようにするためには、.NET Framework クラスは次の要件を満たす必要があります。

- **Public** であり、すべてのコンテナが **Public** であり、かつ、少なくとも 1 つの **Public** メンバを公開している。
- 抽象クラスではない (つまり **MustInherit** で宣言されていない)。
- ジェネリックではない。またはジェネリック コンテナ型の中で宣言されていない。

Error ID: BC31527

このエラーを解決するには

- クラスの宣言を変更して、ジェネリックにならないようにします。さらに、コンテナ要素がジェネリックでないようにします。

  または

- クラスまたはそのコンテナ要素をジェネリックにする必要がある場合は、クラス宣言から **ComClassAttribute** を削除します。このクラスは COM に公開できません。

参照

関連項目

ComClassAttribute

概念

Visual Basic におけるジェネリック型

その他の技術情報

COM 相互運用

'Microsoft.VisualBasic.ComClassAttribute' を、'MustInherit' に宣言されているクラスに適用することはできません。

クラスが ComClassAttribute で宣言されていますが、その宣言で **MustInherit** を指定しています。

COM 相互運用で使用できるようにするためには、.NET Framework クラスは次の要件を満たす必要があります。

- **Public** であり、すべてのコンテナが **Public** であり、かつ、少なくとも 1 つの **Public** メンバを公開している。
- 抽象クラスではない (つまり **MustInherit** で宣言されていない)。
- ジェネリックではない。またはジェネリック コンテナ型の中で宣言されていない。

Error ID: BC32508

このエラーを解決するには

- クラス宣言から **MustInherit** キーワードを削除します。

  または

- クラスまたはそのコンテナ要素をジェネリックにする必要がある場合は、クラス宣言から **ComClassAttribute** を削除します。このクラスは COM に公開できません。

参照

関連項目

MustInherit
ComClassAttribute

その他の技術情報

COM 相互運用

'Microsoft.VisualBasic.ComClassAttribute' は、そのコンテナ '<classname2>' が 'Public' と宣言されていないため、'<classname1>' に適用できません。

COMClassAttribute 属性ブロックを使用しているクラスが、**Public** ではないクラスの内部で宣言されています。クラスを COM オブジェクトとして公開する場合は、そのコンテインメント階層全体を **Public** アクセスで宣言する必要があります。

Error ID: BC32504

このエラーを解決するには

- すべてのクラスを **Public** として宣言するか、**COMClassAttribute** 属性ブロックを削除します。

参照

関連項目

ComClassAttribute クラス
Public (Visual Basic)

概念

Visual Basic で使用される属性
属性の適用

'Microsoft.VisualBasic.ComClassAttribute' は、'Public' と宣言されていないため、'<classname>' に適用できません。

ComClassAttribute を使ってクラスが宣言されていますが、**Public** が宣言に含まれていません。

COM 相互運用に準拠するため、.NET Framework クラスは以下の要件を満たす必要があります。

- クラスが **Public** であり、そのすべてのコンテナが **Public** であり、少なくとも 1 つの **Public** メンバを公開する必要があります。
- クラスを抽象にはできません。つまり、**MustInherit** を使って宣言できません。
- クラスをジェネリックにすることや、ジェネリックなコンテナ型の内部で宣言することはできません。

Error ID: BC32509

このエラーを解決するには

- クラス宣言に **Public** キーワードを追加します。

  または

- クラスまたはそのコンテナ要素を **Public** にできない場合は、クラス宣言から **ComClassAttribute** を削除します。クラスを COM に公開することはできません。

参照

関連項目

Public (Visual Basic)
ComClassAttribute

その他の技術情報

COM 相互運用

'Microsoft.VisualBasic.ComClassAttribute' と '<attribute>' の両方を同じクラスに適用することはできません。

COMClassAttribute 属性ブロックが、COM オブジェクトに適用されない属性と組み合わせて使用されています。.NET Framework 属性と COM 属性が混在していることが原因の 1 つとして考えられます。

Error ID: BC32501

このエラーを解決するには

- **COMClassAttribute** 属性ブロックまたは COM に適用されない属性のいずれかを削除します。

参照

関連項目

ComClassAttribute クラス

概念

Visual Basic で使用される属性

属性の適用

インターフェイス メソッドを実装するメソッドまたはイベントを、'Shared' として宣言することはできません。

インターフェイスのメンバを実装するメソッドまたはイベントを **Shared** として宣言しようとしました。継承可能ではないクラスを除いて、クラスに実装するメソッドやイベントに **Shared** や **Private** を指定できません。

Error ID: BC30505

このエラーを解決するには

- **Shared** キーワードを削除します。

参照

関連項目

Shared (Visual Basic)

モジュール内のメソッドで、インターフェイス メンバを実装することはできません。

インターフェイス メンバは、クラスまたは構造体によって実装される必要があります。

Error ID: BC31083

このエラーを解決するには

- クラスまたは構造体でインターフェイス メンバを実装します。

参照

関連項目

Class ステートメント (Visual Basic)
Structure ステートメント
Implements ステートメント
Interface ステートメント (Visual Basic)

モジュールのメソッドを '<specifier>' として宣言できません。

Module ステートメント内のメンバでは無効な指定子を使用しました。モジュールはインスタンス化できず、継承をサポートしません。また、インターフェイスを実装できません。

Error ID: BC30433

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Module ステートメント

'Overrides' として宣言されているメソッドを 'Overridable' として宣言することはできません。これらのメソッドは暗黙的にオーバーライド可能です。

Overrides 修飾子が適用されているメソッドは、他の多くのメソッドとは異なり、既定でオーバーライドできます。

Error ID: BC30730

このエラーを解決するには

- **Overrides** 修飾子が適用されているメソッドから **Overridable** 修飾子を削除します。

参照

関連項目

Overrides
Overridable

構造体内で宣言するメソッドに 'Handles' 句を指定できません。

構造体のメソッドでは、**Handles** キーワードを使ってイベントを処理できません。

Error ID: BC30728

このエラーを解決するには

- 構造体をクラスに変更することを検討します。

  構造体がインターフェイス内で定義されたイベント ハンドラを実装している場合に限り、**AddHandler** を使用してイベントを構造体内のイベント ハンドラに関連付けることができます。

参照

概念

構造体とクラス

クラス : オブジェクトの設計図

AddHandler と RemoveHandler

その他の技術情報

Visual Basic におけるイベント

メソッドを 'Static' として宣言することはできません。

プロシージャ全体に **Static** 修飾子を適用することはできません。

Error ID: BC30810

このエラーを解決するには

- **Static** 修飾子を使用して、プロシージャ内の各項目を静的として宣言します。

参照

関連項目

Static (Visual Basic)

名前 '<name>' は宣言されていません。

1 つのステートメントでプログラミング要素を参照していますが、指定された名前に完全に一致する要素をコンパイラが見つけることができません。

Error ID: BC30451

このエラーを解決するには

1. 参照元のステートメントで名前のスペルを確認します。Visual Basic は、大文字と小文字を区別しませんが、その他の違いがあった場合にはまったく異なる名前であると見なします。アンダースコア (_) も名前の一部であり、スペルに含まれます。
2. オブジェクトとメンバの間にメンバ アクセス演算子 (.) を指定していることを確認します。たとえば、TextBox の `TextBox1` という名前のコントロールがある場合、このコントロールの Text プロパティにアクセスするには、「`TextBox1.Text`」と入力する必要があります。代わりに「`TextBox1Text`」と入力した場合、別の名前と見なされます。
3. スペルが正しく、オブジェクト メンバ アクセスの構文が正しい場合は、その要素が宣言されていることを確認します。詳細については、「Visual Basic における宣言された要素」を参照してください。
4. プログラミング要素が宣言されている場合は、その要素がスコープ内にあることを確認します。プログラミング要素が宣言されている領域の外側にあるステートメントからこの要素を参照している場合は、要素名を修飾しなければならない可能性があります。詳細については、「Visual Basic におけるスコープ」を参照してください。

参照

関連項目

宣言と定数の概要

概念

Visual Basic の名前付け規則

宣言された要素の名前

その他の技術情報

宣言された要素の参照

名前 <membername> は CLS に準拠していません。

アセンブリが `<CLSCompliant(True)>` としてマークされていますが、このアセンブリが公開するメンバの名前がアンダースコア (_) で始まっています。

プログラミング要素には 1 つ以上のアンダースコアを含めることができますが、共通言語仕様 (CLS) に準拠するためには、先頭をアンダースコアにしないでください。宣言された要素の名前 を参照してください。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40031

このエラーを解決するには

- ソース コードを制御できる場合は、メンバ名を変更して、アンダースコアで始まらないようにします。
- メンバの名前を変更できない場合は、定義から **CLSCompliantAttribute** を削除するか、`<CLSCompliant(False)>` としてマークします。アセンブリは引き続き `<CLSCompliant(True)>` としてマークできます。

参照

概念

宣言された要素の名前
Visual Basic の名前付け規則
CLS 準拠コードの記述

ルート名前空間 <fullnamespacename> にある名前 <namespacename> は CLS に準拠していません。

アセンブリが `<CLSCompliant(True)>` としてマークされているのに、ルート名前空間の名前の要素がアンダースコア (_) で始まっています。

プログラミング要素には 1 つ以上のアンダースコアを含めることができますが、共通言語仕様 (CLS) に準拠するためには、先頭をアンダースコアにしないでください。宣言された要素の名前 を参照してください。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40039

このエラーを解決するには

- CLS 準拠にする必要がある場合は、ルート名前空間の名前を変更し、要素がアンダースコアで始まらないようにします。
- ルート名前空間の名前を変更できない場合は、アセンブリから **CLSCompliantAttribute** を削除するか、アセンブリを `<CLSCompliant(False)>` としてマークします。

参照

処理手順

方法 : アプリケーションの名前空間を変更する

関連項目

Namespace ステートメント

/rootnamespace

概念

Visual Basic における名前空間

宣言された要素の名前

Visual Basic の名前付け規則

CLS 準拠コードの記述

'MyClass' の後には '.' および識別子を指定しなければなりません。

MyClass は真のオブジェクト変数ではないため、単独で指定できません。基本クラスの **NotOverridable** のように、現在のインスタンスのメンバにアクセスするときだけ使用します。

Error ID: BC32028

このエラーを解決するには

1. メンバにアクセスする場合は、**MyClass** の後ろに、メンバ アクセス演算子 (.) および現在のインスタンスのメンバを指定します。
2. クラス メンバにアクセスしない場合は、**Me** キーワードを使用します。

参照

関連項目

MyClass
Me
NotOverridable

概念

継承の基本

'MyClass' をクラスの外で使用することはできません。

クラス構造体の外側で **MyClass** キーワードを使用しました。

Error ID: BC30470

このエラーを解決するには

- **MyClass** の宣言をクラス構造体の中に移動します。

参照

関連項目

MyClass

概念

プロパティとメソッドのオーバーライド

'MyBase' の後には '.' および識別子を指定しなければなりません。

MyBase は真のオブジェクト変数ではないため、単独で指定できません。現在のインスタンスの基本クラスのメンバにアクセスするときだけ使用します。

Error ID: BC32027

このエラーを解決するには

- メンバにアクセスする場合は、**MyBase** の後ろに、メンバ アクセス演算子 (.) および基本クラスのメンバを指定します。
- メンバにアクセスしない場合は、基本クラスのインスタンスを宣言および初期化するか、**MyBase** への参照を削除します。

参照

関連項目

MyBase

概念

継承の基本

メソッド '<methodname>' は 'MustOverride' として宣言されているため、'MyBase' と共に使用することはできません。

MyBase と共に **MustOverride** として宣言されているメソッドを使おうとしました。

Error ID: BC30399

このエラーを解決するには

- **MustOverride** 宣言を削除します。

参照

関連項目

MyBase
MustOverride

名前付き引数が必要です。

プロシージャ呼び出し内の引数リストでは、最初に名前によって引数を渡し、次に位置によって引数を渡しています。プロシージャ呼び出しでは、位置を指定する引数はすべて、名前で渡す引数の前に記述する必要があります。

Error ID: BC30241

このエラーを解決するには

- 位置によって渡される引数が、名前によって渡される引数の前に来るように、引数リストを書き直します。

参照

概念

位置と名前による引数渡し

名前付き引数を ParamArray パラメータに一致させることはできません。

引数の宣言名、コロンと等号 (:=)、引数の値の順で記述した名前付き引数を指定しました。パラメータ配列を名前で渡すことはできません。プロシージャを呼び出すときは、パラメータ配列に対してコンマで区切った不特定数の引数を渡しますが、コンパイラは複数の引数を 1 つの名前に関連付けることができません。

Error ID: BC30587

このエラーを解決するには

- 名前ではなく位置で引数を渡します。

参照

関連項目

ParamArray

概念

位置と名前による引数渡し

名前付き引数は、配列インデックスとしては有効ではありません。

`Array(Index := 10)` などのように、名前による引数渡しの構文を使用して配列インデックスが指定されています。この構文は、配列の添字には無効です。

Error ID: BC30075

このエラーを解決するには

- `Array(10)` などのように、通常の変数または定数式を配列のインデックスとして指定します。

参照

概念

Visual Basic の配列の概要
位置と名前による引数渡し

名前空間または型 '<name>' は既にインポートされています。

名前空間を複数回インポートしようとしました。

Error ID: BC31051

このエラーを解決するには

- 余分な **Imports** ステートメントを削除します。

参照

関連項目

Imports ステートメント

インポート '<qualifiedelementname>' で指定された名前空間または型が、パブリック メンバを含んでいないか、または見つかりません。

エラー メッセージ
インポート '<qualifiedelementname>' で指定された名前空間または型が、パブリック メンバを含んでいないか、または見つかりません。名前空間または型が定義されていて、少なくとも 1 つのパブリック メンバを含んでいることを確認してください。エイリアスが他のエイリアスを含まないようにしてください。

Imports ステートメントが、コンテナ要素を指定していますが、それが見つからないか、**Public** メンバを定義していません。

コンテナ要素は、名前空間、クラス、構造体、モジュール、インターフェイス、または列挙体です。コンテナ要素には、変数、プロシージャ、その他のコンテナ要素などのメンバが含まれます。

インポートの目的は、コードで名前空間または型のメンバに修飾なしでアクセスできるようにすることです。場合によっては、プロジェクトでも、名前空間または型への参照を追加することが必要になります。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」の「コンテナ要素のインポート」を参照してください。

指定されたコンテナ要素が見つからない場合、コンパイラはそれを使用する参照を解決できません。要素が見つかっても、それが **Public** メンバを一切公開しない場合、参照は成功しません。どちらの場合も、要素をインポートしても無意味です。

コンテナ要素をインポートしてそれにインポート エイリアスを割り当てる場合は、そのインポート エイリアスを別の要素のインポートには使用できないことに注意してください。次のコードはコンパイル エラーになります。

```
Imports winfrm = System.Windows.Forms

' The following statement is INVALID because it reuses an import alias.

Imports behav = winfrm .Design.Behavior
```

Error ID: BC40056

このエラーを解決するには

1. プロジェクトからコンテナ要素にアクセスできることを確認します。
2. コンテナ要素の指定に、別のインポートのインポート エイリアスが含まれていないことを確認します。
3. コンテナ要素が少なくとも 1 つの **Public** メンバを公開することを確認します。

参照

関連項目
Imports ステートメント
Namespace ステートメント
Public (Visual Basic)
概念
Visual Basic における名前空間
同じ名前を持つ複数の変数がある場合に参照を解決する

プロジェクト レベルのインポート '<qualifiedelementname>' で指定された名前空間または型が、パブリック メンバを含んでいないか、または見つかりません。

エラー メッセージ
プロジェクト レベルのインポート '<qualifiedelementname>' で指定された名前空間または型が、パブリック メンバを含んでいないか、または見つかりません。名前空間または型が定義されていて、少なくとも 1 つのパブリック メンバを含んでいることを確認してください。エイリアスが他のエイリアスを含まないようにしてください。

プロジェクトのインポート プロパティが コンテナ要素を指定していますが、それが見つからないか、**Public** メンバが定義されていません。

コンテナ要素は、名前空間、クラス、構造体、モジュール、インターフェイス、または列挙体です。コンテナ要素には、変数、プロシージャ、その他のコンテナ要素などのメンバが含まれます。

インポートの目的は、コードで名前空間または型のメンバに修飾なしでアクセスできるようにすることです。場合によっては、プロジェクトでも、名前空間または型への参照を追加することが必要になります。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」の「コンテナ要素のインポート」を参照してください。

指定されたコンテナ要素が見つからない場合、コンパイラはそれを使用する参照を解決できません。要素が見つかっても、それが **Public** メンバを一切公開しない場合、参照は成功しません。どちらの場合も、要素をインポートしても無意味です。

プロジェクト デザイナを使用して、インポートする要素を指定します。[参照設定] ページの [インポートされた名前空間] セクションを使用します。ソリューション エクスプローラの [My Project] アイコンをダブルクリックして、プロジェクト デザイナに移動します。

Error ID: BC40057

このエラーを解決するには

1. プロジェクト デザイナを開き、[参照] ページに切り替えます。
2. [インポートされた名前空間] セクションで、プロジェクトからコンテナ要素にアクセスできることを確認します。
3. コンテナ要素が少なくとも 1 つの **Public** メンバを公開することを確認します。

参照

処理手順
方法 : プロジェクト プロパティおよび構成設定を変更する
関連項目
Public (Visual Basic)
概念
プロジェクトのプロパティ
Visual Basic における名前空間
同じ名前を持つ複数の変数がある場合に参照を解決する

'Namespace' ステートメントの終わりには、対応する 'End Namespace' を指定しなければなりません。

Namespace ブロックの終わりには、**End Namespace** ステートメントを指定してください。

Error ID: BC30626

このエラーを解決するには

- **Namespace** ブロックの最後に **End Namespace** ステートメントを追加します。

参照

関連項目

Namespace ステートメント

概念

Visual Basic における名前空間

'Namespace' ステートメントは、名前空間レベルでのみ発生します。

Namespace ステートメントは、ソース ファイルの **Option** ステートメント、**Imports** ステートメント、およびグローバル属性の後かつ、その他のすべての宣言より前に記述する必要があります。

Error ID: BC30618

このエラーを解決するには

- **Namespace** ステートメントを名前空間宣言またはソース ファイルの先頭に移動します。

参照

関連項目

Namespace ステートメント

概念

Visual Basic における名前空間

デリゲート '<delegatename>':<suberrorlist> のシグネチャと一致する、アクセス可能なメソッド '<procedurename>' がありません。

代入ステートメントは、プロシージャのアドレスをデリゲート変数に代入しますが、コンパイラは、シグネチャの一致するバージョンのプロシージャを見つけることができません。

コードでプロシージャのアドレスが使用されると、コンパイラは、デリゲートとパラメータ リストが一致するバージョンのプロシージャを見つけようとします。プロシージャに複数のオーバーロード バージョンが定義されている場合、コンパイラは、シグネチャが一致するバージョンを見つけようとします。詳細については、「オーバーロードの解決法」を参照してください。

コンパイラは、シグネチャの一致するバージョンのプロシージャを見つけられない場合に、このエラーを生成します。たとえば、プロシージャまたはデリゲートがジェネリックである場合や、渡された型引数が示すシグネチャが他のシグネチャに一致しない場合に、この状態が起こります。

Error ID: BC30950

このエラーを解決するには

1. パラメータ リストが一致するようにプロシージャまたはデリゲートを再定義します。

   または

   プロシージャと一致するパラメータ リストを持つ新しいデリゲートを定義するか、デリゲートと一致するパラメータ リストを持つ新しいプロシージャを定義します。

2. プロシージャまたはデリゲートがジェネリックである場合は、シグネチャが他のシグネチャと一致するような型引数をプロシージャまたはデリゲートに渡します。

参照

関連項目

AddressOf 演算子
Delegate ステートメント

概念

デリゲートと AddressOf 演算子
オーバーロードの解決法
Visual Basic におけるジェネリック型

正しいシグネチャを持つ、アクセス可能な 'Main' メソッドは、'<name>' に見つかりませんでした。

コマンド ライン アプリケーションには、**Sub Main** を定義する必要があります。**Main** は、クラス内で定義される場合には **Public Shared** として、モジュール内で定義される場合には **Public** として宣言される必要があります。

Main の詳細については、「Visual Basic バージョンの Hello World!」を参照してください。

Error ID: BC30737

このエラーを解決するには

- プロジェクトに **Public Sub Main** プロシージャを定義します。クラス内で定義する場合にだけ **Shared** として宣言します。

参照

関連項目

Visual Basic プログラムの構造

概念

Visual Basic バージョンの Hello World!

Visual Basic におけるプロシージャ

最も固有な、アクセス可能な '<procedurename>' がありません: <signaturelist>

代入ステートメントによって、オーバーロードされたプロシージャのアドレスがデリゲート変数に代入されましたが、コンパイラがオーバーロードされたバージョンの中から解決できません。

複数のオーバーロードされたバージョンに定義されたプロシージャのアドレスをコードで使用する場合、コンパイラはどのオーバーロードを使うかを判断する必要があります。デリゲートとパラメータ リストが一致する単一のバージョンが、コンパイラによって検索されます。詳細については、「オーバーロードの解決法」を参照してください。

コンパイラはシグネチャが一致するプロシージャを 2 つ以上検出した場合に、このエラーを生成します。これは、たとえば、あるジェネリック オーバーロードに型引数を渡すときに、別のオーバーロードと同じシグネチャがこのオーバーロードに与えられている場合などに起きる可能性があります。

Error ID: BC30794

このエラーを解決するには

- この衝突が、他のオーバーロードと同じシグネチャを持つジェネリック オーバーロードによって発生している場合は、そのジェネリック オーバーロードに渡される型引数を変更します。

参照

関連項目

AddressOf 演算子
Delegate ステートメント

概念

デリゲートと AddressOf 演算子
オーバーロードの解決法
Visual Basic におけるジェネリック型

アクセス可能な非ジェネリック '<procedurename>' が見つかりません。

ステートメントは、オーバーロードされたバージョンが複数あるプロシージャを呼び出していますが、すべてのオーバーロードされたバージョンがジェネリックであり、呼び出しで型引数を渡していません。

ジェネリック バージョンが 1 つだけあり、型引数を指定しないでそのバージョンを呼び出す場合は、コンパイラでは型の推論を試すことができます。詳細については、「Visual Basic におけるジェネリック プロシージャ」の「型の推定」を参照してください。しかし、ジェネリック バージョンが複数ある場合、型引数を指定しない限り、コンパイラでは特定のバージョンを選び出すことができません。1 つの型引数を渡す場合、オーバーロードされたバージョンの 1 つに定義されたすべての型パラメータに型引数を渡す必要があります。

Error ID: BC32117

このエラーを解決するには

- 型引数リストを指定してプロシージャを呼び出します。

参照

関連項目

Overloads

概念

プロシージャのオーバーロード
Visual Basic におけるジェネリック型
Visual Basic におけるジェネリック プロシージャ

この型引数の数を受け付けるアクセス可能な '<genericprocedurename>' がありません。

ステートメントが複数のオーバーロードされたバージョンを持つジェネリック プロシージャを呼び出していますが、呼び出しに指定された型引数と同じ数の型パラメータを定義した、オーバーロードされたバージョンが 1 つもありません。

ジェネリック バージョンが 1 つしかない場合は、型引数を指定せずにそれを呼び出すことができ、コンパイラは型の推論を試行できます。詳細については、「Visual Basic におけるジェネリック プロシージャ」の「型の推定」を参照してください。しかし、複数のジェネリック バージョンがある場合は、型引数を指定しない限り、コンパイラはその中の 1 つを選ぶことができません。型引数を 1 つ指定する場合は、オーバーロードされたバージョンの 1 つに定義されているすべての型パラメータに対して、型引数を指定する必要があります。

Error ID: BC32118

このエラーを解決するには

- どのオーバーロードされたバージョンを呼び出すか決めてから、正しい数の型引数を指定します。

参照

関連項目

Overloads

概念

プロシージャのオーバーロード
Visual Basic におけるジェネリック型
Visual Basic におけるジェネリック プロシージャ

'Next' ステートメントは、対応する 'For' ステートメントよりも多い変数を指定しています。

入れ子になったループが、入れ子にあるループの数より多いループ変数を指定している単一の **Next** ステートメントで終了しています。次に例を示します。

```
For I = 1 To 10
   For J = 1 To 5
      ' ...
Next J, I, K   ' Next J, I is valid, but there is no loop on K.
```

Error ID: BC32037

このエラーを解決するには

1. **Next** ステートメントで、入れ子になったすべてのループ変数がループの開始とは逆の順序で指定されていることを確認します。
2. **Next** ステートメントから余分な変数を削除します。

参照

関連項目

For...Next ステートメント (Visual Basic)

'Next' の前には、対応する 'For' を指定しなければなりません。

Next ステートメントに対応する **For** ステートメントまたは **For Each** ステートメントがありません。**Next** ステートメントの前には、対応する **For** ステートメントまたは **For Each** ステートメントを指定してください。

Error ID: BC30092

このエラーを解決するには

1. この **For** ループが、入れ子になった **For** ループの一部である場合は、すべてのループが正しく終了していることを確認します。
2. **For** ブロック内の他の制御構造がそれぞれ正しく終了していることを確認します。
3. この **For** ループの書式が正しいことを確認します。

参照

関連項目

For...Next ステートメント (Visual Basic)
For Each...Next ステートメント (Visual Basic)

'Next' が必要です。

On Error Resume 構造に対応する **Next** がありません。

Error ID: BC30003

このエラーを解決するには

- **On Error** ステートメントの最後に **Next** キーワードを追加します。

参照

関連項目

On Error ステートメント (Visual Basic)
Error ステートメント
Resume ステートメント

概念

非構造化例外処理の概要

Next に指定する変数は、For loop に指定する変数 '<variablename>' と一致しません。

Next ループの **For...Next** ステートメント内の制御変数は、対応する **For** ステートメント内の変数と一致している必要があります。

Error ID: BC30070

このエラーを解決するには

1. **Next** ステートメント内の変数のスペルと、対応する **For** ステートメント内の変数のスペルが一致していることを確認します。
2. 外側のループの一部を誤って削除していないかどうかを確認します。
3. このループが、入れ子になったループ セットの一部である場合は、すべてのループが正しく終了しているかどうかを確認します。

参照

関連項目

For...Next ステートメント (Visual Basic)

'New' はこのコンテキストでは有効ではありません。

型名が必要な場所で **New** キーワードが使用されています。このコンテキストでは、**New** と組み合わせることができない値型が必要です。

Error ID: BC30200

このエラーを解決するには

- **New** キーワードを削除します。

参照

関連項目

New (Visual Basic)
Enum ステートメント (Visual Basic)

'New' 制約は、同じ型パラメータに対して複数回指定できません。

複数の New (Visual Basic) 制約が制約リストに含まれています。

型パラメータの制約リストには、その型パラメータに渡す型引数が、作成元のコードがアクセスできるパラメータなしのコンストラクタを公開する必要があることを指定できます。型が複数のパラメータなしのコンストラクタを持つことはできないため、この要件を複数回指定することはできません。

Error ID: BC32081

このエラーを解決するには

- 余分な **New** キーワードを削除します。制約リストには 1 つだけを残してください。

参照

概念

Visual Basic におけるジェネリック型

'New' 制約と 'Structure' 制約は、組み合わせて使用できません。

制約リストに New (Visual Basic) 制約と Structure (Visual Basic) 制約の両方が含まれています。

型パラメータの制約リストには、その型パラメータに渡される型引数が値型でなければならない (**Structure** 制約を使用)、または参照型でなければならない (Class (Visual Basic) 制約を使用) ことを指定できます。同じ型パラメータに対して、この両方の制約を指定することはできません。また、このいずれかを複数回指定することもできません。

New 制約は、作成しているコードからアクセス可能なパラメータなしのコンストラクタを、型引数が公開する必要があることを指定します。しかし、構造体は非共有のパラメータなしコンストラクタを持つことができません。このため、**New** 制約と **Structure** 制約を同時に使うと衝突します。

Error ID: BC32103

このエラーを解決するには

1. 型引数に値型と参照型のどちらを使用するかを判断します。
2. 型引数を値型にする場合は、制約リストから **New** キーワードを削除します。
3. 型引数を参照型にする場合は、制約リストから **Structure** キーワードを削除します。

参照

概念

Visual Basic におけるジェネリック型
値型と参照型

クラス '<classname>' には、オーバーライドされていない 'MustOverride' が含まれているため、'New' は使用できません。

オーバーライドされていない **MustOverride** メンバを含むクラスに対して **New** を使おうとしました。

Error ID: BC30376

このエラーを解決するには

- **New** ステートメントを削除します。

参照

関連項目

MustOverride

'New' をインターフェイスで使用することはできません。

変数をインターフェイス型で宣言する際に Dim ステートメント (Visual Basic) で New (Visual Basic) 句が使用されています。

インターフェイスは参照型ですが、インターフェイスのインスタンスを作成することはできません。クラスまたは構造体のインスタンスを作成する場合にだけ **New** を使用できます。

Error ID: BC30375

このエラーを解決するには

1. 変数がインターフェイス型の場合は、**New** キーワードを削除します。
2. 変数がインスタンスへの参照である場合は、変数をクラス型か構造体型で宣言します。インスタンスを作成するために **New** キーワードを残します。

参照

関連項目

Interface ステートメント (Visual Basic)
Class ステートメント (Visual Basic)
Structure ステートメント

'New' は、'New' 制約がない型パラメータで使用できません。

型パラメータの宣言ステートメントで New (Visual Basic) 句を使用して、作成する型が指定されていますが、この型パラメータには **New** 制約が指定されていません。

型パラメータに制約を指定すると、ジェネリック型の作成時にその型パラメータに渡す、すべての型引数に対する要件を定義できます。**New** 制約は、作成しているコードからアクセス可能なパラメータなしのコンストラクタを、型引数が公開する必要があることを指定します。この制約が指定されている場合は、宣言ステートメントの **New** 句によって、その型のインスタンスを作成できます。

Error ID: BC32046

このエラーを解決するには

- 型引数に、アクセス可能なパラメータなしのコンストラクタを公開するよう要求できる場合は、型パラメータの宣言に **New** 制約を追加します。
- 型引数に、アクセス可能なパラメータなしのコンストラクタを公開するよう要求できない場合は、宣言ステートメントから **New** 句を削除します。その型パラメータに渡されるすべての型引数に対し、常にインスタンスの作成が可能になるようにすることはできません。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

'MustInherit' で宣言されたクラスでは、'New' を使用することはできません。

MustInherit クラスは直接インスタンス化できないため、**MustInherit** クラスに対して **New** 演算子を使うことはできません。コンパイル時の型が **MustInherit** である変数や値を持つことはできますが、これらの変数や値は、null 値であるか、**MustInherit** 型から派生した通常クラスのインスタンスへの参照を含む必要があります。

Error ID: BC30569

このエラーを解決するには

- **New** 演算子を削除します。

参照

関連項目

MustInherit
New (Visual Basic)

入力ソースが指定されていません。

コンパイル時に入力ソースが指定されていません。

Error ID: BC2008

このエラーを解決するには

- ソース コード ファイルが含まれているかどうか確認します。

参照

処理手順

方法 : Visual Basic でプロジェクトをコンパイルして実行する

インデックスの数がインデックス付き配列の次元より少ない値です。

配列要素へのアクセスに使用するインデックスの数は、配列のランク、つまり配列について宣言した次元の数と同じである必要があります。

Error ID: BC30105

このエラーを解決するには

- 添字の合計数が配列のランクと同じになるまで、配列参照に添字を追加します。

参照

その他の技術情報

Visual Basic における配列

インデックス番号がインデックス付き配列の次元を超えています。

配列要素へのアクセスに使用するインデックスの数は、配列のランク、つまり配列について宣言した次元の数と同じである必要があります。

Error ID: BC30106

このエラーを解決するには

- 添字の合計数が配列のランクと同じになるまで、配列参照から添字を削除します。

参照

その他の技術情報

Visual Basic における配列

'NotOverridable' を、別のメソッドをオーバーライドしないメソッドに対して指定することはできません。

プロパティとメソッドは、既定で **NotOverridable** です。**NotOverridable** 修飾子を使用できるのは、別のプロパティやメソッドをオーバーライドするメソッドに対してだけです。

Error ID: BC31088

このエラーを解決するには

- 別のメンバをオーバーライドしないプロパティやメソッドの **NotOverridable** 修飾子を削除します。

参照

関連項目

Overrides
Overloads
NotOverridable
Overridable

'NotInheritable' クラスに、'<specifiername>' として宣言されたメンバを指定することはできません。

オーバーライド修飾子は **NotInheritable** クラスと一緒に使用できません。このクラスのメンバにはオーバーライドできないためです。

Error ID: BC30607

このエラーを解決するには

- **Overridable**、**NotOverridable**、**MustOverride** などのオーバーライド修飾子をクラス定義から削除します。

参照

関連項目

Overridable
NotOverridable
MustOverride

概念

プロパティとメソッドのオーバーライド

'Nothing' は、評価できません。

デバッグ ウィンドウで **Nothing** キーワードが使用されました。

Error ID: BC30701

このエラーを解決するには

- デバッグ時は **Nothing** を使用しないでください。

参照

関連項目

Nothing (Visual Basic)

構造体内の共有されていないメンバは 'New' として宣言できません。

非共有の変数が、構造体内で **New** 句を使って宣言されています。

以下に示すコードのように、共有の参照変数を構造体内で初期化したり、非共有の参照変数を初期化なしで使用したりできます。

```
Shared structVar1 As New System.ApplicationException

Dim structVar2 As System.ApplicationException
```

しかし、非共有の参照変数を構造体の中で初期化することはできません。次のコードは無効です。

```
Dim structVar3 As New System.ApplicationException ' INVALID IN A STRUCTURE
```

Error ID: BC30795

このエラーを解決するには

- 参照変数の宣言から **Shared** 修飾子または **New** キーワードを削除します。

参照

関連項目

Structure ステートメント
Shared (Visual Basic)
New (Visual Basic)

含んでいるクラスが 'Serializable' として公開されていないため、'NonSerialized'属性はこのメンバに影響を与えません。

既定では、クラスとそのメンバはシリアル化できません。NonSerializedAttribute 属性は、シリアル化可能なクラスの特定のメンバをシリアル化しない場合に限り必要です。

Error ID: BC30772

このエラーを解決するには

- クラスに SerializableAttribute 属性を追加します。

  または

- メンバから **NonSerializedAttribute** 属性を削除します。

参照

関連項目

NonSerializedAttribute
SerializableAttribute

その他の技術情報

Visual Basic における属性

非組み込み型名は条件付きコンパイル式で許可されていません。

条件付きコンパイル式から非組み込み型が参照されています。組み込み型のみを使用できます。

Visual Basic の組み込み型の詳細については、「データ型の概要 」を参照してください。

Error ID: BC31426

このエラーを解決するには

- 非組み込み型を、組み込み型に置き換えます。

参照

関連項目

データ型の概要

その他の技術情報

条件付きコンパイル (Visual Basic)

'<typename>' で見つかった正しいシグネチャを持つ、アクセス可能な 'Main' メソッドは、スタートアップ メソッドにはなりません。それらのメソッドはすべてジェネリック、またはジェネリック型の中に入れ子にされているためです。

クラス、モジュール、または構造体に、プロジェクトの起動プロシージャとして修飾された **Main** プロシージャがありません。

Visual Basic では、プロジェクトの起動プロシージャでの型引数の使用を禁止する必要があります。そのため、ジェネリックではなく、またどのジェネリック型にも含まれない、少なくとも 1 つの **Main** プロシージャにアクセスできる必要があります。

Error ID: BC30796

このエラーを解決するには

- ジェネリックではなく、またジェネリック型に含まれない **Main** プロシージャを、少なくとも 1 つ定義します。

  または

- プロジェクトの [プロパティ] ページの [スタートアップ フォーム] または [スタートアップ オブジェクト] に、別のフォームまたはモジュールを指定します。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

概念

Visual Basic におけるジェネリック型

Visual Basic バージョンの Hello World!

CLS に準拠していない 'Mustoverride' メンバは、CLS に準拠している '<classname>' に使用できません。

クラスが `<CLSCompliant(True)>` でマーク付けされていますが、このクラスに `<CLSCompliant(False)>` でマーク付けされるか、またはこのマークが付いていない **MustOverride** プロパティまたはプロシージャが含まれています。

クラスが 共通言語仕様 (CLS) に準拠している場合、そのクラスを使用するアプリケーションは、同じく `<CLSCompliant(True)>` でマーク付けされたメンバにのみアクセスし、そうでないメンバは無視します。しかし、**MustOverride** プロパティまたはプロシージャがあると、アプリケーションはこれにアクセスしてオーバーライドする必要があるため無視できません。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に設定しなかった場合は、非準拠であると見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40034

このエラーを解決するには

- CLS 準拠にする必要があり、このソース コードを変更できる場合は、メンバを `<CLSCompliant(True)>` でマーク付けします。
- CLS 準拠にする必要があり、クラスのソース コードを変更できない場合、または CLS 準拠であるかどうかが明確でない場合は、このメンバを別のクラスに定義します。
- このメンバを非準拠にしておく必要がある場合は、**MustOverride** キーワードを定義から削除するか、**CLSCompliantAttribute** をクラス定義から削除するか、またはクラスを `<CLSCompliant(False)>` でマーク付けします。

参照

関連項目

MustOverride

概念

CLS 準拠コードの記述

CLS に準拠していない <membername> は、CLS に準拠しているインターフェイスに使用できません。

インターフェイス内のプロパティ、プロシージャ、またはイベントが `<CLSCompliant(True)>` でマーク付けされていますが、そのインターフェイス自体が `<CLSCompliant(False)>` でマーク付けされているか、またはマークが付いていません。

インターフェイスが 共通言語仕様 (CLS) に準拠している場合、そのすべてのメンバも準拠している必要があります。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40033

このエラーを解決するには

- CLS 準拠にする必要があり、インターフェイスのソース コードを変更できる場合、かつすべてのメンバが準拠している場合は、インターフェイスを `<CLSCompliant(True)>` でマーク付けします。
- CLS 準拠にする必要があり、インターフェイスのソース コードを変更できない場合、または CLS 準拠であるかどうかが明確でない場合は、このメンバを別のインターフェイスに定義します。
- このメンバを現在のインターフェイス内に残しておく必要がある場合は、インターフェイスの定義から **CLSCompliantAttribute** を削除するか、`<CLSCompliant(False)>` でマーク付けします。

参照

関連項目

Interface ステートメント (Visual Basic)

概念

CLS 準拠コードの記述

エディット コンティニュによって作成されたオブジェクトの評価はまだ利用可能ではありません。

Visual Studio デバッガでは、新しいオブジェクトが作成されていますが、まだそれにアクセスできません。

オブジェクトが使用できない理由として、メモリ割り当てエラーやオブジェクトが同期のとれていない状態にあることなどが挙げられます。これらは、一時的な内部状況であり、通常はデバッガへの入力の結果ではありません。

Error ID: BC30958

このエラーを解決するには

- いったん終了してから、デバッガを再起動します。

参照

処理手順

方法 : 実行を開始する

関連項目

Visual Basic の式

概念

方法 : デバッグの停止と実行の停止

コードのステップ実行の概要

その他の技術情報

Visual Studio でのデバッグ

コンパイラ エラーまたは削除により、オブジェクトは現在存在しません。

Visual Basic コンパイラ内でコード要素へのアクセス エラーが発生しました。

Error ID: BC32300

このエラーを解決するには

1. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
2. エラーが再発する場合は、Visual Basic コンパイラを再インストールします。
3. Visual Basic コンパイラを再インストールした後でもエラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

'On Go to' および 'On GoSub' ステートメントは、現在サポートされていません。

GoTo ステートメントや **GoSub** ステートメントで変数や式の値を使用してプログラム フローを制御することはできません。

Error ID: BC30817

このエラーを解決するには

- **If...Then...Else** ステートメントまたは **Case** ステートメントを使用するようにアプリケーションを再構成します。

参照

関連項目

On Error ステートメント (Visual Basic)
If...Then...Else ステートメント (Visual Basic)
Case (Visual Basic)

'On Error' ステートメントは、'Using' ステートメント内では有効ではありません。

On Error ステートメントが **Using** ステートメント内に定義されていますが、そのコンテキストには有効ではありません。

Error ID: BC36013

このエラーを解決するには

- **On Error** ステートメントの代わりに、**Try...Catch** ブロックなどの構造化エラー処理を使用します。

参照

処理手順

方法 : Visual Basic で Try...Catch ブロックを使用してコードを検査する

関連項目

On Error ステートメント (Visual Basic)

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

構造化例外処理と非構造化例外処理に適した状況

'On Error' ステートメントは、'SyncLock' ステートメント内では有効ではありません。

On Error ステートメントは **SyncLock** ブロック内で使用できません。スレッドの同期が中断されるためです。

Error ID: BC30752

このエラーを解決するには

- **On Error** ステートメントを **SyncLock** ブロックの外に配置します。

参照

関連項目

On Error ステートメント (Visual Basic)

'On Error' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

On Error ステートメントはソース コードでのみ使用できます。

Error ID: BC30720

このエラーを解決するには

- デバッグ コードから **On Error** ステートメントを削除します。

参照

その他の技術情報

Visual Studio でのデバッグ

省略された引数に ParamArray パラメータを一致させることはできません。

ParamArray パラメータと一致するために必要な引数が省略されました。任意の数の引数をプロシージャに渡すことができる **ParamArray** は、引数リストの最後の引数でだけ使用できます。

Error ID: BC30588

このエラーを解決するには

- 省略されている引数を指定します。

参照

関連項目

ParamArray

ジェネリック型またはメソッドの型引数を指定するときには 'Of' が必要です。

ステートメント内で、Of 句を使用しないでジェネリック型から型を作成しようとしたか、ジェネリック メソッドを呼び出そうとしました。

Visual Basic のジェネリック型の構文では、型パラメータおよび型引数の呼び出しは、**Of** キーワードで始める必要があります。また、型パラメータリストまたは型引数リストをかっこで囲む必要もあります。

Error ID: BC32093

このエラーを解決するには

- 型引数リストの先頭に **Of** キーワードを追加し、リスト全体をかっこで囲みます。

参照

処理手順

方法 : ジェネリック クラスを使用する

概念

Visual Basic におけるジェネリック型

1 つの言語要素に対して使用可能な XML コメント ブロックは 1 つのみ

言語要素に複数の XML コメント ブロックが適用されています。

Error ID: BC42301

このエラーを解決するには

- 不要な XML コメント ブロックを削除します。

参照

処理手順

方法 : Visual Basic で XML ドキュメントを作成する

概念

XML の使用によるコードのドキュメントの作成 (Visual Basic)

'Public'、'Private'、'Protected'、'Friend'、または 'Protected Friend' のいずれかのみを指定できます。

要素の宣言でアクセシビリティが競合しています。指定できるアクセシビリティは 1 つだけです。

Error ID: BC30176

このエラーを解決するには

- アクセシビリティを 1 つだけ選択し、その他のキーワードを宣言から削除します。

参照

概念

Visual Basic でのアクセス レベル

'NotOverridable'、'MustOverride'、または 'Overridable' のいずれかのみを指定できます。

要素の宣言でオーバーライドの指定が競合しています。オーバーライド キーワードは、相互に排他的です。

Error ID: BC30177

このエラーを解決するには

- オーバーライド指定子を 1 つだけ選択し、その他のキーワードを宣言から削除します。

参照

概念

プロパティとメソッドのオーバーライド

変換演算子のみが '<キーワード>' として宣言できます。

演算子が変換演算子ではない場合に、Operator ステートメント は、Widening または Narrowing を指定します。

すべての演算子は、Public (Visual Basic) と Shared (Visual Basic) の両方で宣言する必要があります。ただし、変換演算子だけは、Widening と Narrowing のどちらかで宣言できますが、両方で宣言することはできません。

演算子の定義には、Overloads キーワードまたは Shadows キーワードをオプションとして含めることができます。それ以外のキーワードは、Operator ステートメント に使用できません。

Error ID: BC33019

このエラーを解決するには

- 演算子の定義から **Widening** キーワードまたは **Narrowing** キーワードを削除します。型変換が実行されないので、これらは適用されません。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

その他の技術情報

Visual Basic における型変換

演算子はオーバーロードできません。

オーバーロードできるのは一部の演算子だけです。定義可能な演算子を次の表にまとめます。

タイプ	演算子
単項演算	**+**, **-**, **IsFalse**, **IsTrue**, **Not**
二項演算	**+**, **-**, *****, **/**, ****, **&**, **^**, **>>**, **<<**, **=**, **<>**, **>**, **>=**, **<**, **<=**, **And**, **Like**, **Mod**, **Or**, **Xor**
変換 (単項)	**CType**

二項演算の一覧に示した = 演算子は比較演算子であり、代入演算子ではありません。

Error ID: BC33002

このエラーを解決するには

1. 演算子をオーバーロード可能なものの中から選択します。
2. 直接オーバーロードできない演算子をオーバーロードした機能が必要な場合は、適切なパラメータを受け取って適切な値を返す **Function** プロシージャを作成します。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

Operator ステートメント
Function ステートメント (Visual Basic)

概念

演算子プロシージャ

演算子宣言は次のいずれかでなければなりません: +、-、*、\\、/、^、&、Like、Mod、And、Or、Xor、Not、<<、>>、=、<>、<、<=、>、 >=、CType、IsTrue、または IsFalse

宣言できる演算子は、オーバーロード可能な演算子のみです。宣言できる演算子を次の表に示します。

タイプ	演算子
単項演算	**+**, **-**, **IsFalse**, **IsTrue**, **Not**
二項演算	**+**, **-**, *****, **/**, ****, **&**, **^**, **>>**, **<<**, **=**, **<>**, **>**, **>=**, **<**, **<=**, **And**, **Like**, **Mod**, **Or**, **Xor**
変換 (単項)	**CType**

二項演算の一覧に示した = 演算子は比較演算子であり、代入演算子ではありません。

Error ID: BC33000

このエラーを解決するには

1. 演算子をオーバーロード可能なものの中から選択します。
2. 直接オーバーロードできない演算子をオーバーロードする機能が必要な場合は、適切なパラメータを受け取って適切な値を返す **Function** プロシージャを作成します。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

Function ステートメント (Visual Basic)

概念

演算子プロシージャ

演算子 '<operatorname>' は、型 '<typename1>' および '<typename2>' に対して定義されていません。

エラー メッセージ
'<operatorname>' 演算子は、'<typename1>' 型および '<typename2>' 型で定義されていません。2 つの参照型を比較する場合は、'Is' 演算子を使用します。

指定した型について不適切な方法で演算子を使おうとしました。2 つのオブジェクトを比較するときに、**Is** 演算子の代わりに "=" 演算子を使用すると、このエラーが発生する場合があります。

Error ID: BC31080

このエラーを解決するには

1. **Is** 演算子を使用して、2 つの参照型を比較します。
2. 不等式を表すには、**Not** 演算子を **Is** 演算子と組み合わせて使用します。たとえば、次のようにします。

VB

```
If Not A Is B Then
```

参照

関連項目

Is 演算子 (Visual Basic)
= 演算子 (Visual Basic)
Not 演算子 (Visual Basic)

演算子 '<operatorname>' は、型 '<type1>' および '<type2>' に対して定義されていません。

二項演算子によって実行される演算が、演算対象の値を返す 2 つのコード要素に対して無効です。

Error ID: BC30452

このエラーを解決するには

- 2 つの要素を確認し、互換性のあるものにします。

参照

関連項目

Visual Basic における演算子の優先順位

概念

Visual Basic の演算子および式

演算子の効率のよい組み合わせ

演算子 '<operatorname>' は、型 '<typename>' に対して定義されていません。

String などの型に対し、単項演算子 (**+**、**-**、または **Not**) を使おうとしました。この使用方法は定義されていません。

Error ID: BC30487

このエラーを解決するには

- 単項演算子を式に対して適切な演算子に変更します。

参照

関連項目

Visual Basic における演算子の優先順位

概念

Visual Basic の演算子および式

演算子の効率のよい組み合わせ

演算子 '<演算子>' には、2 個のパラメータを指定しなければなりません

バイナリ演算子が、2 つより少ないかまたは 2 つより多いパラメータで定義されています。

バイナリ演算子には、必ず 2 つのパラメータを指定する必要があります。

Error ID: BC33015

このエラーを解決するには

- 2 つのパラメータを指定するように定義を修正します。
- パラメータを 1 つだけ指定する必要がある場合は、単項演算子を定義してください。
- パラメータなしか、3 つ以上のパラメータを指定する必要がある場合は、演算子を定義するのではなく、Function ステートメント (Visual Basic) を使用して **Function** プロシージャを定義する必要があります。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

演算子 '<演算子>' には、1 個のパラメータを指定しなければなりません

単項演算子がパラメータなしか、2 つ以上のパラメータを使って定義されています。

単項演算子には、必ず 1 つのパラメータを指定する必要があります。

Error ID: BC33014

このエラーを解決するには

- 1 つのパラメータを指定するように定義を修正します。
- パラメータを 2 つ指定する必要がある場合は、バイナリ演算子を定義してください。
- パラメータなしか、3 つ以上のパラメータを指定する必要がある場合は、演算子を定義するのではなく、Function ステートメント (Visual Basic) を使用して **Function** プロシージャを定義する必要があります。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

演算子 '<演算子>' には、1 個または 2 個のパラメータを指定しなければなりません。

単項演算子またはバイナリ演算子がパラメータなしで定義されているか、3 つ以上のパラメータを指定して定義されています。

単項演算子またはバイナリ演算子には、1 つまたは 2 つのパラメータを定義する必要があります。

Error ID: BC33016

このエラーを解決するには

- 1 つまたは 2 つのパラメータを指定するように定義を修正します。
- パラメータなしか、3 つ以上のパラメータを指定する必要がある場合は、演算子を定義するのではなく、Function ステートメント (Visual Basic) を使用して **Function** プロシージャを定義する必要があります。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

演算子 '<演算子>' には、型 'Integer' の第 2 パラメータを指定しなければなりません。

ビット シフト演算子が、2 番目のパラメータに **Integer** 以外の型を使って宣言されています。

式の中で右シフト演算子 (>>) または左シフト演算子 (<<) を使用する場合は、2 つ目のオペランドにシフトする量を指定します。Visual Basic では、このオペランドに **Integer** を拡大した任意のデータ型を指定できます。しかし、この定義では、2 番目のオペランドを必ず **Integer** 型にします。ビット シフト演算子を含むクラスまたは構造体を定義する場合は、その定義の 2 番目のオペランドに **Integer** を指定する必要があります。

Error ID: BC33041

このエラーを解決するには

- ビット シフト演算子の定義を変更して、**Integer** 値を返すようにします。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

ビット シフト演算子

概念

演算子プロシージャ

演算子 '<演算子>' には、ブール型の戻り値の型を指定しなければなりません。

比較演算子または論理演算子が、ブール型 (Boolean) (Visual Basic) 以外の戻り値の型で宣言されています。

比較演算子 (**=**、**<>**、**<**、**<=**、**>**、**>=**、**Is**、**IsNot**、**IsFalse**、**IsTrue**、**Like**、**TypeOf**) の結果は **True** と **False** 以外にはなり得ません。詳細については、「Visual Basic における比較演算子」を参照してください。

論理演算子 (**And**、**AndAlso**、**Not**、**Or**、**OrElse**、**Xor**) の処理は、完全にブール値のドメイン内で行われます。詳細については、「Visual Basic の論理演算子とビット処理演算子」を参照してください。

Error ID: BC33023

このエラーを解決するには

- この比較演算子または論理演算子の戻り値の型を **Boolean** に変更します。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

演算子 '<operatorsymbol>' は、すべてのコードのパスでは値を返しません。

エラー メッセージ
演算子 '<operatorsymbol>' は、すべてのコードのパスでは値を返しません。結果が使用されるときに、null 参照の例外が実行時に発生する可能性があります。

演算子 プロシージャに、値を返さないコードへのパスが 1 つ以上含まれている可能性があります。

演算子プロシージャから値を返すことができるのは、それを Return ステートメント (Visual Basic) に含めた場合だけです。

制御が **End Operator** ステートメントに渡された場合、演算子プロシージャはプロパティのデータ型の既定値を返します。詳細については、「Function ステートメント (Visual Basic)」の "動作" を参照してください。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42106

このエラーを解決するには

- 制御フローのロジックを調べて、可能なパスがすべて **Return** ステートメントで終了していることを確認してください。特に、**End Operator** の直前のステートメントは必ず **Return** ステートメントにしてください。

参照
関連項目
Operator ステートメント
概念
演算子プロシージャ

'Select'、'Case' ステートメントの式で使用された型オブジェクトのオペランドです。ランタイム エラーが発生する可能性があります。

Select...**Case** 構造が オブジェクト型 (Object)の式を 1 つ以上使用しています。

変数または式が **Object** 型に評価される場合、コンパイラは遅延バインディングを実行する必要がありますが、これによって実行時に余分な処理が発生します。また、アプリケーションがランタイム エラーを起こす可能性もあります。たとえば、Form を **Object** 変数に代入し、これを数値と比較しようとすると、Visual Basic は **Form** オブジェクトを数値に変換できないので、ランタイムから InvalidCastException がスローされます。

Select...**Case** 構造内の式は、すべて同じデータ型にするか、相互に変換できる密接な関係のあるデータ型にする必要があります。これは、各 **Case** ステートメントが、**Select**...**Case** 構造の基になっているテスト式に対して少なくとも 1 つの値を比較するからです。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42036

このエラーを解決するには

- 可能であれば、すべての式を比較演算子で正しく比較できるデータ型になるよう変更します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

概念

Visual Basic における算術演算子

Visual Basic における比較演算子

演算子 '<operatorsymbol>' に使用される Object 型のオペランドです。ランタイム エラーが発生する可能性があります。

演算子を使用している式の一方または両方のオペランドが オブジェクト型 (Object)です。

変数または式が **Object** として評価されるときは、コンパイラが遅延バインディングを行います。これにより、実行時に追加の処理が必要になります。また、アプリケーションがランタイム エラーを起こす可能性もあります。たとえば、Form を **Object** 変数に代入し、それを / 演算子 (Visual Basic) で使おうとしたとします。その場合、Visual Basic は **Form** オブジェクトを数値に変換できないので、ランタイムから InvalidCastException がスローされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42019

このエラーを解決するには

- 可能であれば、その演算子で正しく処理できるデータ型になるようオペランドを変更します。

参照

概念

Visual Basic における算術演算子

演算子 '<operatorsymbol>' に対して使用される Object 型のオペランドです。オブジェクト ID をテストするには、'IsNot' 演算子を使用してください

<> 演算子を使用している式の一方または両方のオペランドが オブジェクト型 (Object)です。

2 つのオブジェクト参照が同じオブジェクト インスタンスを参照しているかどうかを判断するときは、**Is** または **IsNot** 演算子を使用する必要があります。詳細については、「Visual Basic における比較演算子」の「オブジェクトの比較」を参照してください。

変数または式が **Object** 型に評価される場合、コンパイラは遅延バインディングを実行する必要がありますが、これによって実行時に余分な処理が発生します。また、アプリケーションがランタイム エラーを起こす可能性もあります。たとえば、Form を **Object** 変数に代入し、これを <> 演算子と組み合わせて使用しようとすると、Visual Basic は **Form** オブジェクトを値比較に適したデータ型に変換できないので、ランタイムから InvalidCastException がスローされます。両方のオペランドが **Form** 型として評価される場合でも、この演算は失敗します。<> 演算子は **Form** 型のオペランド用には定義されていないからです。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42032

このエラーを解決するには

- 2 つのオブジェクト参照が同じオブジェクト インスタンスを参照しているかどうかを調べる場合は、**Is** 演算子または **IsNot** 演算子を使用します。

参照

処理手順

方法 : 2 つのオブジェクトが関連しているかどうかを決める

方法 : 2 つのオブジェクトが同一であるかどうか判別する

関連項目

IsNot 演算子

概念

Visual Basic における比較演算子

演算子 '<operatorsymbol>' に対して使用される Object 型のオペランドです。オブジェクト ID をテストするには、'Is' 演算子を使用してください。

式に = が使用されていますが、、一方または両方のオペランドが オブジェクト型 (Object)です。

2 つのオブジェクト参照が同じオブジェクト インスタンスを参照しているかどうかを調べる場合は、**Is** 演算子または **IsNot** 演算子を使用する必要があります。詳細については、「Visual Basic における比較演算子」の「オブジェクトの比較」を参照してください。

変数または式が **Object** 型に評価される場合、コンパイラは遅延バインディングを実行する必要がありますが、これによって実行時に余分な処理が発生します。また、アプリケーションがランタイム エラーを起こす可能性もあります。たとえば、Form を **Object** 型の変数に代入し、それに = 演算子を付けて使用すると、Visual Basic では **Form** オブジェクトを値の比較に適したデータ型に変換できないため、ランタイムが InvalidCastException をスローします。両方のオペランドが **Form** 型に評価される場合でも、= は **Form** であるオペランド用に定義されていないため、処理がエラーになります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42018

このエラーを解決するには

- 2 つのオブジェクト参照が同じオブジェクト インスタンスを参照しているかどうかを調べる場合は、**Is** 演算子または **IsNot** 演算子を使用します。

参照

処理手順

方法 : 2 つのオブジェクトが関連しているかどうかを決める
方法 : 2 つのオブジェクトが同一であるかどうか判別する

関連項目

Is 演算子 (Visual Basic)

概念

Visual Basic における比較演算子

省略可能な引数には、既定値を指定する必要があります。

呼び出し元のプロシージャによってパラメータが指定されない場合、省略可能なパラメータでは既定値を指定する必要があります。

Error ID: BC30812

このエラーを解決するには

- 省略可能なパラメータの既定値を指定します。次に例を示します。

```
Sub Proc1(ByVal X As Integer, _
      Optional ByVal Y As String = "Default Value")
   MsgBox("Default argument is: " & Y)
End Sub
```

参照

関連項目

Optional (Visual Basic)

概念

プロシージャ宣言 (Visual Basic 6.0 ユーザー向け)

省略可能なパラメータに Structure 型を指定することはできません。

省略可能なパラメータは、構造体型にできません。

Error ID: BC31405

このエラーを解決するには

1. 必要に応じて、構造体型のパラメータを宣言します。
2. 構造体に代わる静的なクラス メンバを宣言します。

参照

関連項目

Static (Visual Basic)

その他の技術情報

構造体 : 独自のデータ型

省略可能な引数を、型 '<type>' として宣言することはできません。

省略可能なパラメータを構造体データ型にすることはできません。

Error ID: BC30423

このエラーを解決するには

1. 省略可能なパラメータに構造体を渡すには、パラメータを **Object** として宣言します。
2. **CType** を使用して、プロシージャ内でパラメータを対象の構造体型にキャストします。

参照

関連項目

CType 関数

概念

構造体とクラス

'Optional' が必要です。

プロシージャ宣言内の省略可能な引数に続いて必須引数が指定されています。省略可能な引数の後に続く引数も、すべて省略可能であることが必要です。

Error ID: BC30202

このエラーを解決するには

1. 引数を必須にする場合は、引数リスト内で最初の省略可能な引数の前に移動します。
2. 引数を省略可能にする場合は、**Optional** キーワードを使用します。

参照

概念

省略可能なパラメータ

'Optional' と 'ParamArray' を組み合わせて使用することはできません。

Optional 修飾子と **ParamArray** 修飾子を一緒に使うことはできません。

Error ID: BC30642

このエラーを解決するには

- **ParamArray** または **Optional** のいずれか一方の修飾子だけを指定します。

参照

関連項目

ParamArray
Optional (Visual Basic)

Option Strict On では、すべての変数宣言に 'As' 句が必要です。

宣言内に、**As** 句なしで宣言された変数があります。**Option Strict** が **On** の場合は、すべての変数、プロパティ、プロシージャ引数、および関数の戻り値を **As** 句で宣言して、そのデータ型を指定する必要があります。たとえば、`Dim MyNum As Short` のように記述します。

Error ID: BC30209

このエラーを解決するには

1. **As** キーワードのスペルが正しいかどうかを確認します。
2. 変数宣言に **As** 句を追加するか **Option Strict Off** に変更します。

参照

関連項目

Option Strict ステートメント

概念

Visual Basic での変数宣言

Option Strict On では、すべての関数、プロパティ宣言および演算子宣言に 'As' 句を指定する必要があります。

宣言内に、**As** 句なしで宣言されたプロパティまたは関数の戻り値があります。**Option Strict** が **On** の場合は、すべての変数、プロパティ、プロシージャ引数、および関数の戻り値を **As** 句で宣言して、データ型を指定する必要があります。たとえば、「`Dim MyNum As Short`」のように記述します。

Error ID: BC30210

このエラーを解決するには

1. **As** キーワードのスペルが正しいかどうかを確認します。
2. 宣言されたプロパティや関数の戻り値に **As** 句を付けるか、**Option Strict Off** に変更します。

参照

関連項目

Option Strict ステートメント

概念

Property プロシージャ

Function プロシージャ

Option Strict On では、演算子 '<operatorname>' に対して Object 型のオペランドを使用することはできません。

オブジェクト変数用に定義されている演算子は、**Is** および **TypeOf...Is** だけです。**Option Strict** が **On** の場合、すべてのオペランドは、指定した演算子で定義されているデータ型である必要があります。

Error ID: BC30038

このエラーを解決するには

- **CInt** や **CStr** などの適切な型変換関数を使用して、演算子について定義されたデータ型にオペランドを変換します。

参照

関連項目

Is 演算子 (Visual Basic)
データ型変換関数

概念

Visual Basic における比較演算子

Option Strict On では、演算子 '<operatorname>' に対して Object 型のオペランドを使うことはできません。

エラー メッセージ
Option Strict On の場合は、'<operatorname>' 演算子で Object 型のオペランドを使用できません。Is 演算子を使用して、オブジェクト識別子をテストしてください。

= などの算術比較演算子が 1 つ以上のオブジェクト変数と共に使用されていますが、**Option Strict** が **On** になっています。

Error ID: BC32013

このエラーを解決するには

1. オブジェクト変数に数値が含まれていて、算術比較を行う場合は、**Option Strict Off** に変更します。
2. オブジェクト識別子を比較する場合は、**Is** 演算子を使用します。

参照

関連項目

比較演算子 (Visual Basic)
Is 演算子 (Visual Basic)
Option Strict ステートメント

Option Strict On では、'ByRef' パラメータ '<parametername>' の値を一致する引数に戻してコピーする際に、型 '<typename1>' から型 '<typename2>' に下位変換することはできません。

引数で宣言されている型に一致させるためには、拡張変換する必要のあるデータ型が、プロシージャ呼び出しで **ByRef** 引数として指定されていますが、**Option Strict** が **On** になっています。引数がプロシージャに渡されるとき拡大変換は許可されますが、プロシージャが呼び出しコード内の可変個引数の内容を変更するとき、変換は縮小になります。**Option Strict On** では、縮小変換を行うことができません。

Error ID: BC32029

このエラーを解決するには

- プロシージャ呼び出しで、宣言された型と同じデータ型を使用して各 **ByRef** 引数を指定するか、**Option Strict Off** にします。

参照

関連項目

Option Strict ステートメント

概念

引数の値渡しおよび参照渡し
拡大変換と縮小変換
暗黙の型変換と明示的な型変換

Option Strict On では、遅延バインディングを使用できません。

Visual Basic では、任意のデータ型間で暗黙の型変換を行うことができます。ただし、あるデータ型の値を、精度が低いデータ型や容量の小さなデータ型に変換した場合には、データが失われる可能性があります。**Option Strict On** を指定すると、コンパイル時にこのような種類のデータ変換が通知され、こうした変換を回避することができます。**Option Strict On** と遅延バインディングを一緒に使用することはできません。

Error ID: BC30574

このエラーを解決するには

- **Option Strict** を **Off** に変更します。このためには **On** を削除するか **Off** を明示的に指定します。

参照

関連項目

Option Strict ステートメント

概念

拡大変換と縮小変換

Option Strict On を使用すると、'<type1>' から '<type2>' への暗黙の型変換が行われません。Visual Basic 6.0 のコレクション型は .NET Framework のコレクション型と互換性がありません。

Option Strict On を指定した場合、'`<type1>`' から '`<type2>`' への暗黙の型変換が行われません。Visual Basic 6.0 のコレクション型は .NET Framework のコレクション型と互換性がありません。

Visual Basic 6.0 で使用されるコレクション オブジェクトは、Visual Studio 2005 で使用されるコレクション オブジェクトとは異なります。

Error ID: BC30753

このエラーを解決するには

- 型変換キーワードのいずれかを使用して、コレクション オブジェクトを明示的に変換します。CType 関数 キーワードおよび DirectCast キーワードでは、変換に失敗した場合、実行時例外がスローされます。TryCast キーワードでは、変換に失敗した場合、Nothing (Visual Basic) が返されます。

参照

関連項目

CType 関数
DirectCast
TryCast
Nothing (Visual Basic)

概念

Visual Basic におけるコレクション

Option Strict On で '<type1>' から '<type2>' への暗黙的な変換はできません。

Long から **Integer** への変換などのように、変換後の値を格納できない可能性のある型変換を行おうとしましたが、型チェック スイッチ (Option Strict ステートメント) が **On** に設定されています。

この種の変換を縮小変換と呼び、実行時の失敗の原因になる場合があります。このため、**Option Strict On** では暗黙的な縮小変換が許可されていません。

Error ID: BC30512

このエラーを解決するには

1. <type1> から <type2> へのような種類の変換が存在するかどうかを確認します。共に Visual Basic の要素型であるか、共にクラス インスタンスである場合、一般的に「拡大変換と縮小変換」の表を参照して判断できます。
2. <type1> から <type2> への縮小変換のみが存在する場合は、明示的なキャストを使用する必要があります。CType 関数 キーワードおよび DirectCast キーワードでは、変換に失敗した場合、実行時例外がスローされます。TryCast キーワードは、参照型のみに適用でき、変換が失敗した場合は Nothing (Visual Basic) を返します。
3. 縮小変換が存在し、実行時に失敗してもプログラムで対応可能な場合、または、実行時に失敗する可能性がないとわかっている場合は、ソース コードの先頭に **Option Strict Off** を指定できます。ただし、予期しない結果やプログラムの途中終了を回避するために、変換処理を Try...Catch...Finally ステートメント (Visual Basic) ブロックで囲む必要があります。
4. <type1> から <type2> への変換が存在しない場合は、プログラム ロジックを再度確認する必要があります。予測される <type1> の値に対応して <type2> に値を代入するコードを記述することができます。
5. <type1> から <type2> への変換が存在せず、型の 1 つが自分が定義したクラスまたは構造体である場合は、この型から別の型へ、または別の型からこの型へ変換する変換演算子を定義することができます。詳細については、「方法 : 変換演算子を定義する」を参照してください。
6. すべての場合における一般的なガイドラインとしていうと、**Catch** ブロック内で失敗をトラップでき、失敗に有効に対処できるのでなければ、縮小変換の使用は避けてください。

参照

処理手順

方法 : 変換演算子を定義する

関連項目

Option Strict ステートメント
CType 関数
DirectCast
TryCast
Nothing (Visual Basic)
Try...Catch...Finally ステートメント (Visual Basic)

概念

拡大変換と縮小変換

Option Strict Custom はコマンドライン コンパイラ へのオプションとしてのみ使用することができます。 (vbc.exe)

Option Strict ステートメントは **On** および **Off** のみを引数として受け取ります。`Option Strict Custom` は使用できません。

厳密な言語セマンティクスが守られていないときに警告を出すには、**/optionstrict:custom** コンパイラ オプションを使用します。

Error ID: BC31141

このエラーを解決するには

1. `Option Strict Custom` をソース コードから削除します。
2. **/optionstrict:custom** オプションを指定します。詳細については、「/optionstrict」を参照してください。

参照

関連項目

Option (Visual Basic)
Option Strict ステートメント
/optionstrict

'Option Strict' の後には、'On' または 'Off' のみ指定できます。

Option Strict ステートメントには、**On** または **Off** を指定する必要があります。その他すべての値が無効です。

Error ID: BC30620

このエラーを解決するには

- **On** または **Off** を **Option Strict** ステートメントに指定します。

参照

関連項目

Option (Visual Basic)
Option Compare ステートメント
Option Explicit ステートメント (Visual Basic)
Option Strict ステートメント

'Option' ステートメントは、宣言または 'Imports' ステートメントの前に記述しなければなりません。

Option ステートメントは、ソース コードにおいて、最初の非コメント ステートメントである必要があります。

Error ID: BC30627

このエラーを解決するには

- **Option** ステートメントをソース コードの先頭、**Imports** ステートメントや **Namespace** ステートメントの直前に移動します。

参照

関連項目

Option (Visual Basic)
Option Compare ステートメント
Option Explicit ステートメント (Visual Basic)
Option Strict ステートメント

'Option' の後には、'Compare'、'Explicit'、または 'Strict' を指定しなければなりません。

Option ステートメントで設定できるオプションは、**Compare**、**Explicit**、および **Strict** だけです。

Error ID: BC30206

このエラーを解決するには

1. オプション キーワードのスペルが正しいかどうかを確認します。
2. **Option** ステートメントが不完全な形で終了していないかどうかを確認します。

参照

関連項目

Option (Visual Basic)
Option Compare ステートメント
Option Explicit ステートメント (Visual Basic)
Option Strict ステートメント

'Option Explicit' の後には、'On' または 'Off' のみを指定できます。

Option Explicit ステートメントには、**On** か **Off** を指定する必要があります。その他すべての値が無効です。

Error ID: BC30640

このエラーを解決するには

- **Option Explicit On** か **Option Explicit Off** を指定します。

参照

関連項目

Option Explicit ステートメント (Visual Basic)

'Option Compare' の後には、'Text' または 'Binary' を指定しなければなりません。

Option Compare ステートメントに不正な設定があるか、設定がありません。**Option Compare** では **Text** または **Binary** のみを設定できます。

Error ID: BC30207

このエラーを解決するには

1. 設定指定子のスペルが正しいかどうかを確認します。
2. **Text** または **Binary** を **Option Compare** ステートメントに追加します。たとえば、`Option Compare Text` のように記述します。

参照

関連項目

Option Compare ステートメント

'Option <specifier>' ステートメントは、1 つのファイルに付き 1 回のみ使用できます。

Option Compare ステートメント、**Option Explicit** ステートメント、および **Option Strict** ステートメントは、同じソース ファイル内でそれぞれ 1 度だけ使用できます。

Error ID: BC30225

このエラーを解決するには

- 重複している **Option** ステートメントを削除します。

参照

関連項目

Option (Visual Basic)
Option Compare ステートメント
Option Explicit ステートメント (Visual Basic)
Option Strict ステートメント

<optionname> オプションには、引数 <argument> が必要です。

必須引数が省略されています。

Error ID: BC2006

このエラーを解決するには

- 不足している引数を指定します。

参照

概念

エラーの種類

オプション <optionname> の後には '+' または '-' のみ指定できます。

オプションの後のテキストは、加算記号 (+) またはマイナス記号 (-) ではありません。

Error ID: BC2009

このエラーを解決するには

- オプションを有効にするには、オプションの後のテキストを加算記号 (+) で置き換えます。

  または

- オプションを無効にするには、オプションの後のテキストをマイナス記号 (+) で置き換えます。

参照

関連項目

Visual Basic コンパイラ オプション一覧 (アルファベット順)

概念

エラーの種類

演算子は 'Shared' として宣言されなければなりません。

Operator ステートメントに Shared (Visual Basic) キーワードが指定されていません。

Operator プロシージャには Public (Visual Basic) キーワードと **Shared** キーワードが必要です。また、変換演算子の場合は Widening キーワードか Narrowing キーワードのいずれかも必要です。

Error ID: BC33012

このエラーを解決するには

- **Operator** ステートメントに **Shared** キーワードを追加します。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

演算子は 'Public' として宣言されなければなりません。

Operator ステートメントに Public (Visual Basic) キーワードが指定されていません。

Operator プロシージャには **Public** キーワードと Shared (Visual Basic) キーワードが必要です。また、変換演算子の場合は Widening キーワードか Narrowing キーワードのいずれかも必要です。

Error ID: BC33011

このエラーを解決するには

- **Operator** ステートメントに **Public** キーワードを追加します。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

演算子をモジュール内で宣言できません。

Operator ステートメントがモジュール定義に含まれています。

演算子は、定義しているクラスや構造体の中に定義できますが、そのクラスまたは構造体を、演算子の少なくとも一方のオペランドとして受け取る必要があります。

演算子はプログラミング要素のインスタンスを、そのオペランドの 1 つとして使用する必要があり、インスタンスを持つのはクラスと構造体だけです。したがって、これ以外のプログラミング要素の一部として、演算子を定義することはできません。

Error ID: BC33018

このエラーを解決するには

- モジュールで演算が必要な場合は、Function ステートメント (Visual Basic) を使用して、その演算を実行する **Function** プロシージャを定義します。
- または、モジュールの内部にクラスまたは構造体を定義して、そのクラスまたは構造体に演算子を定義することもできます。ただし、演算子はそのクラスまたは構造体のインスタンスを、演算子の少なくとも一方のオペランドとして受け取る必要があります。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

概念

演算子プロシージャ

演算子は '<keyword>' として宣言できません。

演算子の定義には無効なキーワードを使って、演算子が宣言されています。

すべての演算子は、Public (Visual Basic) と Shared (Visual Basic) の両方で宣言する必要があります。また、変換演算子は、Widening と Narrowing のどちらかで宣言する必要がありますが、両方で宣言することはできません。演算子の定義には、Overloads キーワードまたは Shadows キーワードをオプションとして含めることができます。それ以外のキーワードは、Operator ステートメント に使用できません。

Error ID: BC33013

このエラーを解決するには

- 演算子の定義から無効なキーワードを削除します。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

概念

演算子プロシージャ

'As' 句のない演算子です。Object の型と見なされます。

演算子プロシージャに **As** 句が指定されていません。

As 句はプログラミング要素に関連付けるデータ型に指定します。Operator ステートメントでは、演算子プロシージャが呼び出し元のコードに返す値のデータ型を指定する際に使用します。**Operator** ステートメントに **As** 句を指定しない場合、戻り値のデータ型は既定で **Object** 型になります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC41005

このエラーを解決するには

- **Operator** ステートメントに **As** 句を定義して、戻り値のデータ型を指定します。

参照

関連項目

Operator ステートメント

概念

演算子プロシージャ

メモリが不足しています。(Visual Basic コンパイラ エラー)

メモリが不足しています。

Error ID: BC2004

このエラーを解決するには

- 不要なアプリケーション、ドキュメント、およびソース ファイルを閉じます。
- 不要なコントロールおよびフォームを削除して、同時に読み込まれる数を減らします。
- **Public** 変数の数を減らします。
- ディスクの空き容量を調べてください。
- 追加のメモリを取り付けるかメモリの再割り当てを実行して、利用可能な RAM の容量を増やします。
- 必要なくなった場合にメモリが解放されていることを確認します。

参照

概念

エラーの種類

オーバーフローしました。(Visual Basic エラー)

リテラルの値が、代入先のデータ型の制限を超えています。

Error ID: BC30036

このエラーを解決するには

- 代入先のデータ型の値の範囲を調べ、その範囲内に収まるようにリテラルを書き直します。

参照

関連項目

データ型の概要 (Visual Basic)

'<method>' にアクセスできないため、オーバーロードの解決に失敗しました。

オーバーロードされたメソッドを呼び出そうとしましたが、コンパイラは、パラメータの数が一致するパラメータ リストを持つアクセス可能なメソッドを発見できません。

Error ID: BC30517

このエラーを解決するには

1. 指定した引数リストを確認します。
2. 呼び出そうとしたメソッドがアクセス可能かどうかを調べます。

参照

概念

オーバーロードされたプロパティとメソッド

この型引数の数を受け付ける '<genericprocedurename>' がないため、オーバーロードの解決に失敗しました

コンパイラが、適切な数の型パラメータを持つオーバーロードされたジェネリック プロシージャにアクセスできないため、このプロシージャへの呼び出しが解決されません。

ジェネリック プロシージャを呼び出すには、すべての型パラメータに 1 つずつ型引数を渡す必要があります。または、型引数をまったく渡さないで、コンパイラに型の推論を任せることもできます。詳細については、「Visual Basic におけるジェネリック プロシージャ」の「型の推定」を参照してください。

Error ID: BC32087

このエラーを解決するには

1. 呼び出そうとしているバージョンを、呼び出し元コードからアクセスできるようにします。Visual Basic でのアクセス レベル を参照してください。
2. 呼び出し元コードの型引数を追加または削除して、型引数リストが、呼び出そうとしているバージョンの型パラメータ リストと一致するようにします。

   または

   呼び出し元コードからすべての型引数を削除し、コンパイラに型の推論を任せます。競合やあいまいさがあると、型の推論が失敗する可能性があります。

参照

関連項目

型リスト

概念

オーバーロードされたプロパティとメソッド
オーバーロードの解決法
Visual Basic におけるジェネリック型

この引数の数を受け付ける '<method>' がないため、オーバーロードの解決に失敗しました。

オーバーロードされたメソッドを呼び出そうとしましたが、コンパイラは、対応する引数リストを持つメソッドを発見できません。

Error ID: BC30516

このエラーを解決するには

- 指定した引数リストを確認します。

参照

概念

オーバーロードされたプロパティとメソッド

これらの引数で呼び出される、アクセス可能な '<method>' がないため、オーバーロードの解決に失敗しました：<error>

オーバーロードされたメソッドを呼び出そうとしましたが、コンパイラは、対応する引数リストを持つメソッドを発見できません。

Error ID: BC30518

このエラーを解決するには

- 指定した引数リストを確認します。

参照

概念

オーバーロードされたプロパティとメソッド

下位変換しないで呼び出される、アクセス可能な '<method>' がないため、オーバーロードの解決に失敗しました : <error>

オーバーロードされたメソッドを呼び出そうとしましたが、コンパイラは、縮小変換を行わずに呼び出すことのできるメソッドを発見できません。縮小変換では、変換後のデータ型が変換前の値の一部を正確に格納できない可能性があります。

Error ID: BC30519

このエラーを解決するには

- **Option Strict Off** を指定します。

参照

関連項目

Option Strict ステートメント

概念

オーバーロードされたプロパティとメソッド

拡大変換と縮小変換

これらの引数に最も固有な、アクセス可能な '<method>' がないため、オーバーロードの解決に失敗しました：<error>

オーバーロードされたメソッドを呼び出そうとしましたが、指定した引数リストを変換した場合に一致するパラメータ リストを持つオーバーロードが複数存在し、コンパイラがオーバーロードを選択できません。

コンパイラは、呼び出しの引数リストのデータ型とオーバーロードのパラメータ リストのデータ型とを、できるだけ密接に一致させようとします。この場合、各引数を対応するパラメータに拡大変換する必要があります。これは、型チェック スイッチ (Option Strict ステートメント) が **On** であるのか **Off** であるのかを問いません。

拡大変換が必要なオーバーロードが複数検出された場合、コンパイラは、引数のデータ型に最もよく一致するオーバーロードを検索します。つまり、拡大変換の程度が一番少なくて済むオーバーロードを検索します。オーバーロードの 1 つが、1 つの引数のデータ型によく当てはまり、別のオーバーロードが別の引数のデータ型によく当てはまる場合に、コンパイラにより、このエラー メッセージが生成されます。使用例および詳細については、「オーバーロードの解決法」を参照してください。

Error ID: BC30521

このエラーを解決するには

1. このメソッドに対応するオーバーロードをすべて確認して、呼び出すオーバーロードを判断します。
2. 呼び出しステートメントで、引数のデータ型を変更し、目的とするオーバーロードで定義されているパラメータのデータ型と一致させます。データ型を、定義されているデータ型に変換するためには、CType 関数 を使用する必要がある場合があります。

参照

関連項目

Overloads
Option Strict ステートメント
CType 関数

概念

プロシージャのオーバーロード
プロシージャのオーバーロードに関する注意事項
オーバーロードの解決法
オーバーロードされたプロパティとメソッド

複数の基本インターフェイスで宣言されているメソッドをオーバーロードできません。

複数の継承インターフェイスが同じメソッドを暗黙にオーバーロードしています。

Error ID: BC31410

このエラーを解決するには

- **Overloads** 修飾子の代わりに **Shadows** 修飾子を使用します。

参照

関連項目

Overloads
Shadows

ParamArray パラメータは 1 次元配列でなければなりません。

ParamArray 引数は 1 次元の配列型として宣言する必要があります。この宣言では、**ParamArray** 引数に複数の次元を指定しています。

Error ID: BC30051

このエラーを解決するには

- **ParamArray** 引数名に続くかっこ内のカンマをすべて削除します。

参照

関連項目

ParamArray

概念

パラメータ配列

その他の技術情報

Visual Basic における配列

ParamArray パラメータは配列でなければなりません。

ParamArray 引数は 1 次元の配列型として宣言する必要があります。この宣言では、**ParamArray** 引数名の後ろにかっこが記述されていません。

Error ID: BC30050

このエラーを解決するには

- **ParamArray** 引数名の後ろに 1 対のかっこを追加します。

参照

関連項目

ParamArray

概念

パラメータ配列

その他の技術情報

Visual Basic における配列

ParamArray 引数は、'ByVal' として宣言しなければなりません。

ParamArray パラメータでは、**ByRef** 修飾子を使用できません。

Error ID: BC30667

このエラーを解決するには

- **ByVal** 修飾子を使って **ParamArray** パラメータを宣言します。

参照

関連項目

ParamArray
ByRef
ByVal

ParamArray パラメータには、配列型を指定しなければなりません。

MSCorlib アセンブリにアクセスするときに問題が発生しました。その結果、**ParamArray** パラメータに無効な型があります。

Error ID: BC31092

このエラーを解決するには

1. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
2. エラーが再発する場合は、作業内容を保存し、Visual Studio .NET を再起動します。
3. エラーが再発する場合は、Visual Basic を再インストールします。
4. 再インストール後もエラーが発生する場合は、マイクロソフト製品サポート サービスまでご連絡ください。

参照

関連項目

ParamArray

その他の技術情報

製品のサポートとユーザー補助

'<methodname>' のパラメータ '<parametername>' には、一致する省略された引数が既にあります。

プロシージャ呼び出しで、引数の位置による指定を省略した後で、同じ引数が名前で指定されています。次に例を示します。

```
Public Sub ABC(ByVal X As Byte, Optional ByVal Y As Byte = 0, _
                                Optional ByVal Z As Byte = 0)
' ...
Call ABC(6, , Y:=3)   ' Argument Y omitted by position, supplied by name.
```

Error ID: BC32021

このエラーを解決するには

- 引数を位置によって指定するか、位置による指定を省略するカンマを削除します。

参照

概念

位置と名前による引数渡し

'<methodname>' のパラメータ '<parametername>' には、一致する引数が既に存在します。

同じプロシージャ呼び出し内で、引数が複数回、名前によって渡されています。

Error ID: BC30274

このエラーを解決するには

- すべての引数名のスペルが正しいかどうかを確認します。
- 引数リストから重複している引数を削除します。

参照

概念

位置と名前による引数渡し

パラメータは、既に '<argumentname>' という名前で宣言されています。

プロシージャ宣言で 2 つの引数が同じ名前で定義されています。プロシージャ宣言内のすべての引数名は一意である必要があります。

Error ID: BC30237

このエラーを解決するには

- いずれかの引数の名前を変更します。

参照

概念

プロシージャのパラメータと引数

同じ名前を持つ複数の変数がある場合に参照を解決する

パラメータを、定義する関数と同じ名前にすることはできません。

パラメータとそれを定義している関数の両方で同じ名前が使われています。

Error ID: BC30530

このエラーを解決するには

- 関数内のパラメータ名を確認し、衝突しているパラメータの名前を変更します。

参照

関連項目

パラメータ コレクション エディタ

パラメータの指定子が重複しています

1 つのパラメータ指定子が、このパラメータに複数回適用されています。

Error ID: BC30785

このエラーを解決するには

- 重複している指定子を削除します。

参照

関連項目

パラメータの一覧

単項演算子のパラメータの型は、それを含む型 '<typename>' でなければなりません。

単項演算子の定義で、その演算子が定義されているクラスまたは構造体の型とは異なる型が、パラメータに指定されています。

演算子をクラスまたは構造体に定義するときには、少なくとも 1 つのパラメータはそのクラスまたは構造体の型にする必要があります。単項演算子の場合、唯一のオペランドはそのクラスまたは構造体の型である必要があります。

Error ID: BC33020

このエラーを解決するには

- パラメータの型を、演算子が定義されているクラスまたは構造体の型に変更します。
- パラメータとして渡すデータ型とは別のデータ型を演算の結果として返す必要がある場合は、変換演算子を代わりに定義します。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

型 '<operator>' を 'For' ステートメントで使用するには、'<typename>' にしなければなりません。

>= 演算子または <= 演算子が定義されていない型のカウンタ変数が、それ自身の型のパラメータを使って **For** ループに指定されています。

カウンタ変数は、包含型を比較する以上演算子 (>=) および以下演算子 (<=) 演算子をサポートする型である必要があります。つまり、両方のオペランドは、カウンタ変数の型であることが必要です。

数値データ型をカウンタ変数に使う場合、>= 演算子と <= 演算子は包含型でサポートされます。ユーザー定義のクラスまたは構造体を使う場合は、両方の演算子をそのクラスまたは構造体の型のオペランドで定義する必要があります。

Error ID: BC33040

このエラーを解決するには

1. カウンタ変数のデータ型が正しいスペルであることを確認します。
2. ユーザー定義のクラスまたは構造体をカウンタ変数に使う場合は、クラスまたは構造体を比較する >= 演算子と <= 演算子を定義します。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

For...Next ステートメント (Visual Basic)
Operator ステートメント

概念

演算子プロシージャ

アセンブリをビルド中に問題が発生した可能性があります：<error>

Visual Basic コンパイラによって呼び出された ALink ツールが、アセンブリのビルド エラーを報告しています。署名されたアセンブリが、署名されていないアセンブリを参照していることが原因の 1 つとして考えられます。

このメッセージは警告です。コンパイラはアセンブリの生成を続けます。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40009

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
3. エラーが再発する場合は、Visual Basic コンパイラを再インストールします。
4. Visual Basic コンパイラを再インストールした後でもエラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

アセンブリ '<assemblyname>' をビルド中に問題が発生した可能性があります：<error>

Visual Basic コンパイラによって呼び出された ALink ツールが、アセンブリのビルド エラーを報告しています。以下の原因が考えられます。

- 署名されたアセンブリが、署名されていないアセンブリを参照している。この場合は、参照アセンブリがセキュリティ基準を満たしているかどうかを考慮する必要があります。
- 64 ビット アプリケーションを 32 ビット プラットフォーム上でビルドしている。この場合、すべての参照アセンブリの 64 ビット バージョンが、対象となるプラットフォームにインストールされていることを確認する必要があります。共通言語ランタイム (CLR: Common Language Runtime) アセンブリの場合、この処理は自動的に行われますが、このエラー メッセージは依然として出力されます。

このメッセージは警告です。コンパイラはアセンブリの生成を続けます。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC40010

このエラーを解決するには

1. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。
2. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
3. エラーが再発する場合は、Visual Basic コンパイラを再インストールします。
4. Visual Basic コンパイラを再インストールした後でもエラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト製品サポート サービスにご連絡ください。

参照

概念

共通言語ランタイムの概要

その他の技術情報

製品のサポートとユーザー補助

プロパティ '<propertyname>' は、既定値でないベース <type> '<typename>' のプロパティをオーバーライドするため、'Default' として宣言できません。

Default プロパティを宣言しようとしましたが、基本クラスで **Default** として宣言されていないプロパティをオーバーライドするので、有効な宣言ではありません。

Error ID: BC30504

このエラーを解決するには

- **Default** 宣言を削除します。

参照

関連項目

Default (Visual Basic)

プロパティ '<propertyname>' は、すべてのコードのパスでは値を返しません。

エラー メッセージ
プロパティ '<propertyname>' は、すべてのコードのパスでは値を返しません。結果が使用されるときに、null 参照の例外が実行時に発生する可能性があります。

プロパティの **Get** プロシージャに、値を返さないコードへのパスが 1 つ以上含まれている可能性があります。

プロパティの **Get** プロシージャから値を返すには、次のいずれかの方法を使います。

- プロパティ名に値を割り当ててから、**Exit Property** ステートメントを実行します。
- プロパティ名に値を割り当ててから、**End Get** ステートメントを実行します。
- Return ステートメント (Visual Basic) に値を含めます。

制御が **Exit Property** または **End Get** に渡されたとき、プロパティ名に値が割り当てられていない場合、**Get** プロシージャはプロパティのデータ型の既定値を返します。詳細については、「Function ステートメント (Visual Basic)」の "動作" を参照してください。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42107

このエラーを解決するには

- 制御フローのロジックを調べて、ステートメントが値を返す前に値が割り当てられていることをすべて確認してください。

  常に **Return** ステートメントを使うと、値を返すプロシージャが確実に値を返すようにするのが簡単になります。この場合は、**End Get** の直前のステートメントを **Return** ステートメントにする必要があります。

参照

関連項目
Property ステートメント
Get ステートメント
概念
Property プロシージャ

必要なパラメータを持たないプロパティを、'Default' として宣言することはできません。

必須パラメータではなく **Default** 修飾子を使用してパラメータが指定されました。

Error ID: BC31048

このエラーを解決するには

- 既定のプロパティの必須パラメータを宣言します。
- プロパティ定義から **Default** 修飾子を削除します。

参照

概念

既定のプロパティ

モジュールのプロパティを '<specifier>' として宣言することはできません。

MustOverride、**NotOverridable**、**Overridable** などの指定子をモジュール内のプロパティに対して使おうとしました。

Error ID: BC30503

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Module ステートメント
Module (Visual Basic)

'WriteOnly' として宣言されているプロパティに 'Get' を使用することはできません。

Get プロシージャは、プロパティの値を読み取ります。**WriteOnly** プロパティを読み取ることはできません。

Error ID: BC30023

このエラーを解決するには

- **Property** ステートメントから **WriteOnly** キーワードを削除するか、プロパティ本体から **Get** プロシージャを削除します。

参照

関連項目

Property ステートメント
Get ステートメント
WriteOnly

'ReadOnly' として宣言されているプロパティに 'Set' を使用することはできません。

Set プロシージャは、プロパティに値を書き込みます。**ReadOnly** プロパティに書き込むことはできません。

Error ID: BC30022

このエラーを解決するには

- **Property** ステートメントから **ReadOnly** キーワードを削除するか、プロパティから **Set** プロシージャを削除します。

参照

関連項目

Property ステートメント
Set ステートメント (Visual Basic)
ReadOnly (Visual Basic)

プロパティを '<modifiername>' として宣言することはできません。

Property 宣言の構文にエラーがあります。このエラーは通常、**WithEvents** などの指定したキーワードを **Property** 宣言で使用できないことを意味します。

Error ID: BC30639

このエラーを解決するには

- **Property** 宣言の構文を確認します。

参照

概念

Property プロシージャ

プロジェクト レベルの条件付きコンパイル定数の形式が有効ではありません : <error>

プロジェクトについて指定した条件付きコンパイル オプションが無効です。

Error ID: BC31031

このエラーを解決するには

- このプロジェクトで使用される条件付きコンパイル定数を確認します。

参照

その他の技術情報

条件付きコンパイル (Visual Basic)

プロジェクト レベルの条件付きコンパイル定数 '<constantname>' の形式が有効ではありません : <error>

プロジェクトについて指定した条件付きコンパイル オプションが無効です。

Error ID: BC31030

このエラーを解決するには

- このプロジェクトで使用される条件付きコンパイル定数を確認します。

参照

その他の技術情報

条件付きコンパイル (Visual Basic)

プロジェクトは閉じられました。

Visual Basic コンパイラ内でコード モデルへのアクセス エラーが発生しました。

Error ID: BC32301

このエラーを解決するには

1. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
2. エラーが再発する場合は、Visual Basic コンパイラを再インストールします。
3. Visual Basic コンパイラを再インストールした後でもエラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

プロジェクトには既にアセンブリ <assemblyidentity> への参照が指定されています。

エラー メッセージ
プロジェクトには既にアセンブリ <assemblyidentity> への参照が指定されています。<filepath>' への 2 番目の参照を追加することはできません。

プロジェクトが、同じアセンブリへの参照を複数作成しています。

アセンブリは、アセンブリの名前、バージョン、公開キー (使用する場合)、およびカルチャによって識別されます。

このエラーが発生する原因の 1 つは、元のアセンブリ参照とは異なるファイル パスによって別のアセンブリが参照されていることです。

Error ID: BC32208

このエラーを解決するには

- 2 番目の参照を削除します。同じアセンブリを参照するため、この参照は不要です。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
壊れた参照のトラブルシューティング
概念
プロジェクト参照

プロジェクト '<projectname1>' は、'<typename>' を含むプロジェクト '<projectname2>' への間接的な参照を生じさせます。

エラー メッセージ
プロジェクト '<projectname1>' は、'<typename>' を含むプロジェクト '<projectname2>' への間接的な参照を生じさせます。プロジェクトに '<projectname2>' へのプロジェクト参照を追加してください。

プロジェクト内のコードが別のプロジェクトで定義されている型にアクセスしますが、アクセス元のプロジェクトには、型を定義しているプロジェクトへの直接参照がありません。

型はクラス、構造体、インターフェイス、モジュール、または列挙体の可能性があります。

問題の型を定義するプロジェクトは、型を含むアセンブリを作成します。プロジェクトが、型を定義しているプロジェクトを直接参照しない場合、コンパイラは、その型を含むアセンブリがソリューション内にあり、アクセス可能であることを保証できません。

Error ID: BC31532

このエラーを解決するには

- 問題の型を定義しているプロジェクトを特定し、それに対するプロジェクト参照を追加します。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
概念
プロジェクト参照
その他の技術情報
名前空間およびコンポーネントの参照
参照の管理

プロジェクト '<projectname>' は、'<classname>' を含むアセンブリ '<assemblyname>' への間接的な参照を生じさせます。

エラー メッセージ
'<projectname>' プロジェクトは、'<classname>' を含む '<assemblyname>' アセンブリを間接参照しています。'<assemblyname>' への参照をプロジェクトに追加してください。

このプロジェクトには、プロジェクト参照に含まれていないアセンブリに依存するメンバがあります。

Error ID: BC31515

このエラーを解決するには

- [ソリューション エクスプローラ] の [参照] を右クリックし、[参照の追加] をクリックして、アセンブリへの参照を追加します。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
概念
プロジェクト参照

プロジェクト '<projectname>' ではアセンブリ '<assemblyname>' のバージョン '<versionnumber1>' への参照が必要ですが、アセンブリ '<assemblyname>' のバージョン '<versionnumber2>' が参照されています (Visual Basic 警告)。

エラー メッセージ
プロジェクト '<projectname>' ではアセンブリ '<assemblyname>' のバージョン '<versionnumber1>' への参照が必要ですが、アセンブリ '<assemblyname>' のバージョン '<versionnumber2>' が参照されています。バージョン '<versionnumber1>' への参照が作成されました。

プロジェクトは、どこかで他の場所で定義されているアセンブリへの間接参照を作成しますが、そのプロジェクトには、同じアセンブリの旧バージョンへの直接参照も含まれています。

旧バージョンではなく新しいバージョンで定義されている型およびプログラミング要素にアクセスできるように、コンパイラは、アクセスを解決するときに新しいバージョンへの間接参照を使用します。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42203

このエラーを解決するには

- 旧バージョンのアセンブリへの直接参照を削除するか、参照先を新しいバージョンに変更します。

参照
処理手順
方法 : Visual Studio のリファレンスを追加または削除する
概念
アセンブリの概要
プロジェクト参照
その他の技術情報
名前空間およびコンポーネントの参照
参照の管理

プロジェクト '<projectname>' ではアセンブリ '<assemblyname>' のバージョン '<versionnumber1>' への参照が必要ですが、アセンブリ '<assemblyname>' のバージョン '<versionnumber2>' が参照されています (Visual Basic エラー)。

プロジェクトは、どこかで他の場所で定義されているアセンブリへの間接参照を作成しますが、そのプロジェクトには、同じアセンブリの新しいバージョンへの直接参照も含まれています。

コンパイラが、コードでの間接参照の使用を許可している場合、後のバージョンでは定義されているが前のバージョンにはない型およびプログラミング要素にアクセスできない可能性があります。

Error ID: BC32209

このエラーを解決するには

- 旧バージョンのアセンブリへの間接参照を削除して、新しいバージョンへの直接参照を使用します。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する

概念

アセンブリの概要

プロジェクト参照

その他の技術情報

名前空間およびコンポーネントの参照

参照の管理

'<projectname>' は、直接的または間接的に '<projectname>' を参照しているため、プロジェクト '<projectname>' でプロジェクト '<projectname>' を参照できません。

プロジェクトが 2 番目に示されているプロジェクトへの参照を含んでいます。さらに、2 番目のプロジェクトが、自身を参照しているプロジェクトへの参照を含んでいます。

Error ID: BC30761

このエラーを解決するには

- どちらかの参照を削除します。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する

プロジェクト '<projectname>' でファイル '<filename>' への参照を生成できません。

プロジェクト '*<projectname>*' でファイル '*<filename>*' への参照を生成できません。この参照を削除し、再度追加してから、ソリューションの完全再ビルドを行ってください。

ファイルへの参照を作成するときにエラーが発生しました。

Error ID: BC31514

このエラーを解決するには

1. [ソリューション エクスプローラ] の [参照設定] で項目を右クリックし、[削除] をクリックして、参照を削除します。
2. [ソリューション エクスプローラ] の [参照] を右クリックし、[参照の追加] をクリックして、参照を復元します。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する

概念

プロジェクト参照

プロパティ アクセサを '<keyword>' として宣言できません。

Get または **Set** プロシージャの宣言に、プロパティ プロシージャでは無効なキーワードが含まれています。

Get ステートメントと Set ステートメント (Visual Basic) ではアクセス修飾子 (**Public**、**Protected**、**Friend**、**Protected Friend**、**Private**) のみを指定できます。

Error ID: BC31099

このエラーを解決するには

- **Get** ステートメントまたは **Set** ステートメントから無効なキーワードを削除します。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

概念

Property プロシージャ

プロパティ アクセサを 'NotOverridable' プロパティ内で '<accessmodifier>' として宣言できません。

NotOverridable プロパティの Get ステートメントまたは Set ステートメント (Visual Basic) に、**Private** キーワードが含まれています。

Property ステートメントに **NotOverridable** と **Private** を組み合わせて指定できない理由を以下に説明します。

1. 基本クラスのプロパティまたはプロシージャをオーバーライドしていないプロパティまたはプロシージャには、既定で NotOverridable が設定されます。
2. しかし、基本クラスのプロパティまたはプロシージャをオーバーライドした派生クラスのプロパティまたはプロシージャには、既定で Overridable が設定されます。オーバーライドの階層構造を終了するには、オーバーライドを **NotOverridable** で宣言します。**NotOverridable** はこれ以外の目的には使用できません。つまり、**NotOverridable** は Overrides と組み合わせた場合しか使用できません。
3. 基本クラスのプロパティまたはプロシージャが Private (Visual Basic) で宣言されている場合、派生クラスからアクセス不可能であるため、そのプロパティまたはプロシージャをオーバーライドできません。このため、**Private** と **Overridable** を組み合わせて使うことはできません。
4. プロパティまたはプロシージャをオーバーライドするには、オーバーライドしているプロパティまたはプロシージャとシグネチャが同じであるだけでなく、アクセス レベルも同じであることが必要です。つまり、オーバーライドしているプロパティまたはプロシージャに **Private** を指定できません。オーバーライド可能なプロパティまたはプロシージャには、**Private** を指定できないからです。
5. **NotOverridable** を指定できるのは、オーバーライドしているプロパティまたはプロシージャだけなので、これを **Private** と組み合わせて指定することはできません。

同じ理由で、オーバーライドしているプロパティの個々のプロパティ プロシージャ (**Get** および **Set**) を **Private** にすることはできません。

Error ID: BC31106

このエラーを解決するには

- **Get** ステートメントまたは **Set** ステートメントから **Private** キーワードを削除するか、**Property** ステートメントから **Overrides** キーワードと **NotOverridable** キーワードを削除します。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

概念

Property プロシージャ

シャドウとオーバーライドの違い

プロパティ アクセサを 'Default' プロパティ内で '<accessmodifier>' として宣言できません。

既定のプロパティの Get ステートメントまたは Set ステートメント (Visual Basic) に、**Private** キーワードが含まれています。

既定のプロパティを **Private** にすることはできません。その個々のプロパティ プロシージャ (**Get** または **Set**) も同様です。

Error ID: BC31107

このエラーを解決するには

- **Get** ステートメントまたは **Set** ステートメントから **Private** キーワードを削除するか、Property ステートメントから **Default** を削除します。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

方法 : 既定のプロパティを宣言する/呼び出す (Visual Basic)

概念

Property プロシージャ

プロパティ アクセスはプロパティに割り当てるか、またはその値を使わなければなりません。

代入または値の使用を伴わないでプロパティにアクセスしようとしました。

Error ID: BC30545

このエラーを解決するには

- **Property** プロシージャを確認し、不足している代入を指定します。

参照

関連項目

代入演算子

概念

Property プロシージャ

パラメータなしのプロパティ '<propertyname>' が見つかりません。

使用している属性のプロパティにはパラメータが必要です。

Error ID: BC30658

このエラーを解決するには

- プロパティのパラメータを指定します。

参照

その他の技術情報

プロパティ (Visual Basic)

Property '<propertyname>' は、ベース '<name>' の既定プロパティをオーバーライドするため、'Default' として宣言しなければなりません。

コード内の **Property** ステートメントは基本クラス、構造体、またはインターフェイスの **Default** プロパティをオーバーロードするため、**Default** として宣言する必要があります。

Error ID: BC30360

このエラーを解決するには

- プロパティを **Default** として宣言します。

参照

関連項目

Default (Visual Basic)

プロパティ '<propertyname>' は 'WriteOnly' です。

WriteOnly と宣言されたプロパティから値を取得しようとしました。

Error ID: BC30524

このエラーを解決するには

- プロパティの宣言から **WriteOnly** 指定子を削除します。

参照

関連項目

WriteOnly

プロパティ '<propertyname>' は 'ReadOnly' です。

ReadOnly と宣言されたプロパティに値を代入しようとしました。

Error ID: BC30526

このエラーを解決するには

- プロパティの宣言から **ReadOnly** 指定子を削除します。

参照

関連項目

ReadOnly (Visual Basic)

プロパティ '<propertyname>' の型はサポートされていません。

参照先アセンブリ内のプロパティに、表現できない型があります。

Error ID: BC30643

このエラーを解決するには

- アプリケーションで使用できる型を指定します。

参照

概念

Visual Basic におけるデータ型

プロパティ '<propertyname>' は、ベースクラス '<baseclassname>' でメンバ '<membername>' に対して暗黙的に宣言されたメンバと競合する '<implicitmembername>' を暗黙的に宣言します。

エラー メッセージ
メンバ '<membername1>' は、ベースクラス '<baseclassname>' でメンバ '<membername2>' に対して暗黙的に宣言されたメンバと競合する '<implicitmembername>' を暗黙的に宣言します。このため、'メンバ' は 'Overloads' と宣言してはなりません。

派生クラス内のプロパティが、基本クラスの暗黙的なメンバと同じ名前を持つ暗黙的なメンバを生成しており、かつ Overloads キーワードを指定しています。

同じクラスの中に複数のバージョンのプロパティまたはプロシージャを定義するには、オーバーロードを行います。基本クラスのメンバが既に **Overloads** を指定している場合を除いては、基本クラスのメンバの別のバージョンを定義することはできません。暗黙的なメンバは **Overloads** を指定していないので、コンパイラは、このプロパティが暗黙的な基本クラス メンバをShadowsしていると見なします。

Visual Basic コンパイラは、宣言されている特定のプログラミング要素に対応する暗黙のメンバを作成します。これらの暗黙または合成のメンバの説明を次の表に示します。

宣言された要素	暗黙的に作成されるメンバ
列挙値	`value__` メンバ
イベント	`add_<eventname>` プロシージャ `remove_<eventname>` プロシージャ `<eventname>Event` フィールド `<eventname>EventHandler` デリゲート
プロパティ	`get_<propertyname>` プロシージャ `set_<propertyname>` プロシージャ
My.Form メンバ、**My.WebService** メンバ、または、MyGroupCollectionAttribute 属性でマークされたクラスのメンバ	`m_<variablename>` **Static** 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ
WithEvents 変数	`_<variablename>` 変数 `<variablename>` プロパティ `get_<variablename>` プロシージャ `set_<variablename>` プロシージャ

名前の競合が発生する可能性があるため、これらの暗黙のメンバと同じフォームを使用している宣言されたプログラミング要素に名前を付けることは避ける必要があります。たとえば、`get_` または `set_` で始まる要素名の使用は避けてください。

このメッセージは既定では警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40024

このエラーを解決するには

- 暗黙的な基本クラス メンバを隠ぺいまたはシャドウする場合は、プロパティ宣言内の Overloads キーワードを Shadows キーワードに置き換えます。
- 暗黙的な基本クラス メンバをシャドウする必要がない場合は、プロパティの名前を変更して、上記の表に示した名前と衝突しないようにします。

参照

概念

宣言された要素の名前

'Private' アクセサを含むため、プロパティを '<propertymodifier>' と宣言できません。

Private プロパティ プロシージャ (**Get** または **Set**) を持つプロパティが、Overridable でマーク付けされています。

基本クラスのプロパティまたはプロシージャが Private (Visual Basic) で宣言されている場合、派生クラスはそのプロパティまたはプロシージャにアクセスできないため、オーバーライドできません。このため、**Private** と **Overridable** を組み合わせて使うことはできません。これはプロパティそのものに対してだけでなく、個々のプロパティ プロシージャに対しても同じです。

Error ID: BC31108

このエラーを解決するには

- Property ステートメントから **Overridable** キーワードを削除するか、Get ステートメントまたは Set ステートメント (Visual Basic) から **Private** キーワードを削除します。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

概念

Property プロシージャ

Property Get、Property Let、および Property Set はサポートされていません。新しい Property 宣言構文を使用してください。

Property Get/Let/Set はサポートされていません。新しい **Property** 宣言構文を使用してください。

プロパティを宣言するための構文が変更されました。現在は、ブロック内でプロパティを定義します。

Error ID: BC30808

このエラーを解決するには

1. **Property** キーワードで始まるコード ブロックでプロパティを定義します。プロパティの最後には、**End Property** 構成体を配置します。
2. **Get** キーワードを使って、プロパティ ブロック内で **Get** プロパティ プロシージャを定義します。**Get** プロパティ プロシージャの最後には **End Get** 構成体を配置します。
3. **Set** キーワードを使って、プロパティ ブロックの中で **Set** プロパティ プロシージャを定義します。**Set** プロパティ プロシージャの最後には **End Set** 構成体を配置します。
4. プロパティに値を代入する場合は、常に **Set** プロパティ プロシージャを使用します。**Let** プロパティ プロシージャは不要になりましたのでサポートされていません。

参照

関連項目

Property ステートメント

その他の技術情報

言語の変更点 (Visual Basic 6.0 ユーザー向け)

プロパティに 'End Property' がありません。

End ステートメントは、その **End** ステートメントが終了させるブロックと対応している必要があります。**Property** 定義が正しく終了していません。

Error ID: BC30025

このエラーを解決するには

- プロパティ定義の最後に **End Property** ステートメントを追加します。

参照

関連項目

End ステートメント

概念

Property プロシージャ

プロパティまたはフィールド '<name>' に、有効な属性型が指定されていません。

有効な属性の型を受け入れない属性コンストラクタを使用している可能性があります。属性にその他の適切なコンストラクタがある場合でも、エラーが発生します。

Error ID: BC30659

このエラーを解決するには

- この属性をどのように使用しているかを確認します。

参照

関連項目

AttributeUsageAttribute Class

その他の技術情報

Visual Basic における属性

Property パラメータには 'Value' という名前が必要です。

Property ステートメント内のパラメータに **Value** という名前が付いています。この名前は、**Set** プロシージャでプロパティに代入される新しい値のためにプロパティ定義内で予約されています。

Error ID: BC32201

このエラーを解決するには

- **Property** ステートメント内のパラメータの名前を変更します。

参照

関連項目

Set ステートメント (Visual Basic)

概念

Property プロシージャ

Protected 型はクラスの内部でのみ宣言できます。

モジュール内の型が **Protected** として宣言されました。

Error ID: BC31047

このエラーを解決するには

1. クラス内で型を宣言します。
2. **Protected** 修飾子を削除します。

参照

関連項目

Class ステートメント (Visual Basic)
Protected (Visual Basic)

'As' 句のないプロパティです。Object の型と見なされます。

プロパティ宣言に **As** 句が指定されていません。

As 句はプログラミング要素に関連付けるデータ型を指定します。Property ステートメントでは、プロパティの **Get** プロシージャが呼び出し元のコードに返す値のデータ型を指定する際に使用します。**Property** ステートメントに **As** 句を指定しない場合、プロパティのデータ型は既定で **Object** 型になります。

既定では、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC42022

このエラーを解決するには

- **Property** ステートメントに **As** 句を定義して、プロパティのデータ型を指定します。

参照

関連項目

Property ステートメント
Get ステートメント

概念

Property プロシージャ

'ReadOnly' または 'WriteOnly' 指定子を持たないプロパティには、'Get' および 'Set' の両方を指定する必要があります。

プロパティを **ReadOnly** または **WriteOnly** として宣言していない場合は、その値の読み取りおよび書き込みを行うためのプロシージャを記述する必要があります。

Error ID: BC30124

このエラーを解決するには

1. **Property** ステートメントと **End Property** ステートメントの間に、**Get** プロシージャおよび **Set** プロシージャの両方を記述します。
2. **Property** 宣言内のその他のプロシージャが正しく終了していることを確認します。

参照

関連項目

Property ステートメント
Get ステートメント
Set ステートメント (Visual Basic)

'RaiseEvent' 宣言の終わりには、対応する 'End RaiseEvent' を指定しなければなりません。

RaiseEvent 宣言の終わりには、対応する **End RaiseEvent** ステートメントを指定してください。

Error ID: BC31117

このエラーを解決するには

- **RaiseEvent** 宣言が **End RaiseEvent** ステートメントで終わっていることを確認します。

参照

関連項目

RaiseEvent ステートメント
Event ステートメント

'<eventname>' イベントに 'RaiseEvent' が定義されていません。

イベントを **Custom** として宣言している場合は、そのイベントを発生させるためのプロシージャを記述する必要があります。

Error ID: BC31132

このエラーを解決するには

1. **Custom Event** ステートメントと **End Event** ステートメントの間で、**RaiseEvent** を宣言します。
2. このイベント宣言内のその他のプロシージャが正しく終了してあることを確認します。

参照

関連項目

RaiseEvent ステートメント
Event ステートメント

'RaiseEvent' は既に宣言されています。

カスタム イベントの宣言に **RaiseEvent** 宣言が複数あります。**RaiseEvent** 宣言は、そのイベントを発生させるためのプロシージャを宣言します。

Error ID: BC31129

このエラーを解決するには

- 余分な **RaiseEvent** ステートメントを削除します。

参照

関連項目

RaiseEvent ステートメント
Event ステートメント

'RaiseEvent' メソッドには、含んでいるイベントのデリゲート型 '<signature>' と同じシグネチャを指定しなければなりません。

Custom Event 宣言では、カスタム イベントの **As** 句で指定されているデリゲート型と同じシグネチャを含む **RaiseEvent** を宣言する必要があります。

シグネチャを一致させるためには、**RaiseEvent** 宣言とデリゲートとで、パラメータ数および各パラメータの型を一致させる必要があります。

Error ID: BC31137

このエラーを解決するには

- デリゲート型のパラメータと一致するように、**RaiseEvent** 宣言のパラメータを変更します。

使用例
この例は、**RaiseEvent** 宣言で正しいパラメータ型が指定されているカスタム イベントを示します。

VB

```
Delegate Sub TestDelegate(ByVal sender As Object, ByVal i As Integer)
Custom Event Test As TestDelegate
    AddHandler(ByVal value As TestDelegate)
        ' Code for adding an event handler goes here.
    End AddHandler

    RemoveHandler(ByVal value As TestDelegate)
        ' Code for removing an event handler goes here.
    End RemoveHandler

    RaiseEvent(ByVal sender As Object, ByVal i As Integer)
        ' Code for raising an event goes here.
    End RaiseEvent
End Event
```

参照
関連項目
Event ステートメント
RaiseEvent
Delegate ステートメント
その他の技術情報
Visual Basic におけるイベント

'RaiseEvent' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

RaiseEvent ステートメントは [イミディエイト] ウィンドウで使用できません。

Error ID: BC30739

このエラーを解決するには

- コード エディタ内のコードで **RaiseEvent** ステートメントをテストします。

参照

関連項目

RaiseEvent ステートメント

その他の技術情報

Visual Studio でのデバッグ

'Case' ステートメントに指定された範囲が有効ではありません。

不正な範囲が **Case** ステートメントに指定されています。

同じ式を複数の値と比較する場合は、**If...Then...Else** ステートメントの代わりに **Select...Case** ステートメントを使用できます。**If** ステートメントと **ElseIf** ステートメントは、各ステートメントで異なる式を評価できますが、**Select** ステートメントは、1 つの式を 1 回だけ評価し、その式をすべての比較に使用します。各 **Case** ステートメントには、1 つ以上の複数の値、値の範囲、または値と比較演算子の組み合わせを指定できます。

Error ID: BC40052

このエラーを解決するには

- すべての値が含まれるように範囲を変更するか、**Case Else** ステートメントを使って未定義の値を受け取るようにします。

参照

関連項目

Select...Case ステートメント (Visual Basic)

概念

条件判断構造

拡大変換と縮小変換

'ReadOnly' と 'WriteOnly' を組み合わせて使用することはできません。

ReadOnly 修飾子と **WriteOnly** 修飾子を一緒に使うことはできません。

Error ID: BC30635

このエラーを解決するには

- **ReadOnly** または **WriteOnly** のいずれか一方の修飾子だけを指定します。

参照

関連項目

ReadOnly (Visual Basic)
WriteOnly

'ReadOnly' 属性プロパティ '<propertyfield>' を代入式のターゲットにすることはできません。

属性の **ReadOnly** プロパティに値を代入しようとしました。

Error ID: BC31501

このエラーを解決するには

1. プロパティに値を代入するステートメントを削除します。
2. 独自に作成したプロパティを使用している場合は、属性プロパティから **ReadOnly** 修飾子または **Shared** 修飾子を削除します。

参照

関連項目

Shared (Visual Basic)

その他の技術情報

Visual Basic における属性

'ReadOnly' プロパティでは、'Get' でアクセス修飾子を指定することはできません。

ReadOnly プロパティの宣言で、Property ステートメントと Get ステートメントの両方にアクセス レベルが指定されています。

プロパティには常にアクセス レベルを指定できます。また、その最大 1 つのプロパティ プロシージャ (**Get** または **Set**) に、別のアクセス レベルを指定できます。ただし、プロパティよりも制限の高いアクセス レベルであることが必要です。両方のプロパティ プロシージャにアクセス レベルを指定することはできません。

Error ID: BC31105

このエラーを解決するには

- **Get** ステートメントからアクセス修飾子を削除します。これによって、プロパティ全体が **ReadOnly** になります。このプロパティには 2 つのアクセス レベルを定義できません。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

概念

Property プロシージャ

'ReadOnly' プロパティ '<propertyname>' を代入式のターゲットにすることはできません。

値を代入するコンテキストに **ReadOnly** プロパティがあります。実行時に値を代入できるのは、書き込み可能な変数、プロパティ、および配列要素だけです。

Error ID: BC30098

このエラーを解決するには

- 変数を宣言している **Property** ステートメントから **ReadOnly** キーワードを削除するか、そのプロパティに値を代入しているステートメントを削除します。

参照

関連項目

ReadOnly (Visual Basic)
Property ステートメント

'ReadOnly' プロパティには 'Get' を指定する必要があります。

プロパティを **ReadOnly** として宣言している場合は、その値を読み取るためのプロシージャを記述する必要があります。

Error ID: BC30126

このエラーを解決するには

1. **Property** ステートメントと **End Property** ステートメントの間に、**Get** プロシージャを記述します。
2. **Property** 宣言内のその他のプロシージャが正しく終了していることを確認します。

参照

関連項目

Property ステートメント
Get ステートメント

'ReadOnly' 変数を代入式のターゲットにすることはできません。

値を代入するコンテキストで **ReadOnly** プロパティが見つかりました。実行時に値を代入できるのは、書き込み可能な変数、プロパティ、および配列要素だけです。

Error ID: BC30064

このエラーを解決するには

- 変数を宣言している **Dim** ステートメントから **ReadOnly** キーワードを削除するか、または値を代入するステートメントを削除します。

参照

関連項目

ReadOnly (Visual Basic)
Dim ステートメント (Visual Basic)

'ReDim' によって、配列の次元数を変更することはできません。

配列のランク (次元の数) を **ReDim** で変更しようとしました。**ReDim** ステートメントを使用すると、既に宣言されている配列の次元のサイズの変更ができます。しかし、配列のランクは変更できません。

Error ID: BC30415

このエラーを解決するには

- 配列のサイズではなくランクを変更する必要があることを確認してください。可能な場合、**Dim** を使って必要なランクの配列を新しく宣言します。

参照

関連項目

ReDim ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)

概念

Visual Basic の配列の概要

参照されたオブジェクトには値 'Nothing' が指定されています。

使用しているオブジェクトの値が **Nothing** になっていますが、使用できる値が必要です。**Nothing** は、オブジェクトに値が何も代入されていないか、値として **Nothing** が代入されたために、値のないオブジェクトであることを示す値です。

Error ID: BC30760

このエラーを解決するには

1. エラーが発生したプロシージャのスコープ内でオブジェクトが宣言されていることを確認します。
2. オブジェクトの値が設定されていることを確認します。

メモ :
Nothing の値は 0 または空の文字列とは異なります。**IsNothing** を使用して、オブジェクトに **Nothing** の値が含まれているかどうかを確認できます。

参照

関連項目

Nothing (Visual Basic)
IsNothing 関数

参照オブジェクト '<objectname>' の値は 'Nothing' です。

参照されているオブジェクトが設定されていないか、**Nothing** の値に設定されています。

Error ID: BC30705

このエラーを解決するには

- オブジェクト名のスペルが正しいこと、およびオブジェクトが設定されていることを確認します。

参照

関連項目

Nothing (Visual Basic)

その他の技術情報

Visual Studio でのデバッグ

参照された配列要素に 'Nothing' の値が含まれています。

参照されている配列要素が設定されていないか、以前に **Nothing** に設定されました。

Error ID: BC30744

このエラーを解決するには

- 配列内の値が正しく設定されていることを確認します。

参照

関連項目

Nothing (Visual Basic)

参照された '<membername>' には値 'Nothing' が指定されています。

参照中の項目には値が代入されていないか、**Nothing** の値が代入されました。

Error ID: BC30759

このエラーを解決するには

- 項目名のスペルが正しいこと、および正しく設定されていることを確認します。

参照

関連項目

Nothing (Visual Basic)

別のコンストラクタを呼び出している間は、作成中のオブジェクトへの参照は、有効ではありません。

オブジェクトのコンストラクタがオブジェクトを作成する前に、そのオブジェクトのメンバが参照されました。

Error ID: BC31095

このエラーを解決するには

- コンストラクタを別のコンストラクタから呼び出すときは、**MyBase**、**MyClass**、または **Me** は使用しないでください。

参照

関連項目

コンストラクタとデストラクタの使用方法

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄

非共有メンバを参照するには、オブジェクト参照が必要です。

非共有メンバの参照に対して、オブジェクト参照が指定されていません。共有ではないメンバの修飾には、クラス名 自体は使用できません。最初にインスタンスをオブジェクト変数として宣言してから、変数名でインスタンスを参照します。

Error ID: BC30469

このエラーを解決するには

1. インスタンスをオブジェクト変数として宣言します。
2. 変数名でインスタンスを参照します。

参照

概念

同じ名前を持つ複数の変数がある場合に参照を解決する

参照には型 '<typename>' を含むアセンブリ '<assemblyidentity>' が必要ですが、プロジェクト '<projectname1>' と '<projectname2>' があいまいであるため、適切な参照が見つかりませんでした。

プロジェクト外で定義されているクラス、構造体、インターフェイス、列挙体、またはデリゲートなどの型が、式で使用されています。しかし、その型を定義する複数のアセンブリへのプロジェクト参照があります。

問題のプロジェクトは、同じ名前のアセンブリを複数作成します。このため、コンパイラは、アクセスしている型に、どちらのアセンブリを使用すればよいか判断できません。

別のアセンブリで定義されている型にアクセスするには、Visual Basic コンパイラが、そのアセンブリへの参照を保持する必要があります。これは、プロジェクト間の循環参照にならない、単一であいまいさのない参照である必要があります。

Error ID: BC30969

このエラーを解決するには

1. プロジェクトから参照するのに最適なアセンブリを作成しているプロジェクトがどれかを特定します。この判断には、ファイル アクセスの容易さや更新の頻度などの基準を使用できます。
2. プロジェクトのプロパティに、使用する型が定義されているアセンブリを含むファイルへの参照を追加します。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する
方法 : プロジェクト プロパティおよび構成設定を変更する
壊れた参照のトラブルシューティング

概念

プロジェクト参照
同じ名前を持つ複数の変数がある場合に参照を解決する

その他の技術情報

名前空間およびコンポーネントの参照

参照には型 <typename>' を含むアセンブリ '<assemblyidentity>' が必要ですが、循環参照により適切な参照が見つからなかった可能性があります: <referencedependencylist>

プロジェクト外で定義されているクラス、構造体、インターフェイス、列挙、またはデリゲートなどの型が、式で使用されています。しかし、そのアセンブリへのプロジェクト参照は、循環参照セットの一部になっています。

複数のプロジェクトが互いに参照しあっている場合、その参照が循環する可能性があります。たとえば、2 つのプロジェクトは互いへの参照を保持できます。もっと一般的な例としては、あるプロジェクトから次のプロジェクトへの参照のチェーンが、最終的に最初のプロジェクトに戻ることがあります。このような場合、参照を解決するための最終的なプロジェクトがチェーンの最後に存在しません。

別のアセンブリで定義されている型にアクセスするには、Visual Basic コンパイラが、そのアセンブリへの参照を保持する必要があります。これは、プロジェクト間の循環参照にならない、単一であいまいさのない参照である必要があります。

Error ID: BC30962

このエラーを解決するには

- プロジェクトのプロパティに、使用している型を定義するアセンブリを作成するプロジェクトへの直接参照を追加します。

参照

処理手順

方法 : Visual Studio のリファレンスを追加または削除する
方法 : プロジェクト プロパティおよび構成設定を変更する
壊れた参照のトラブルシューティング

概念

プロジェクト参照

その他の技術情報

名前空間およびコンポーネントの参照

型 '<membername>' を含むモジュール '<modulename>' への参照が必要です。

エラー メッセージ
'<membername>' 型を含む '<modulename>' モジュールへの参照が必要です。プロジェクトに参照を追加してください。

アセンブリ内の別の型に依存する型がプロジェクトにありますが、そのアセンブリがプロジェクト参照には含まれていません。

Error ID: BC30653

このエラーを解決するには

- プロジェクト内の依存する型についてプロジェクト参照を追加します。

参照
概念
プロジェクト参照

実装されたインターフェイス '<interfacename>' を含むモジュール '<modulename>' への参照が必要です。参照をプロジェクトに追加してください。

エラー メッセージ
実装された '<interfacename>' インターフェイスを含む '<modulename>' モジュールへの参照が必要です。プロジェクトに参照を追加してください。

プロジェクト内で直接参照されないモジュールでインターフェイスが定義されています。Visual Basic コンパイラでは、インターフェイスが複数のモジュールで定義されている場合に備えて、あいまいさを避けるために参照が必要になります。

Error ID: BC30010

このエラーを解決するには

- 参照されないモジュールの名前をプロジェクト参照に含めます。

参照

処理手順
壊れた参照のトラブルシューティング
その他の技術情報
名前空間およびコンポーネントの参照

イベント '<eventname>' の定義を含むモジュール '<modulename>' への参照が必要です。参照をプロジェクトに追加してください。

イベント '*<eventname>*' の定義を含むモジュール '*<modulename>*' への参照が必要です。プロジェクトに参照を追加してください。

プロジェクト内で直接参照されないモジュールでイベントが定義されています。Visual Basic コンパイラでは、イベントが複数のモジュールで定義されている場合に備えて、あいまいさを避けるために参照が必要になります。

Error ID: BC30006

このエラーを解決するには

- 参照されないモジュールの名前をプロジェクト参照に含めます。

参照

処理手順

壊れた参照のトラブルシューティング

その他の技術情報

名前空間およびコンポーネントの参照

基本クラス '<classname>' を含むモジュール '<modulename>' への参照が必要です。参照をプロジェクトに追加してください。

エラー メッセージ
基本クラス '<classname>' を含む '<modulename>' モジュールへの参照が必要です。プロジェクトに参照を追加してください。

プロジェクト内で直接参照されないモジュールでクラスが定義されています。Visual Basic コンパイラでは、クラスが複数のモジュールで定義されている場合に備えて、あいまいさを避けるために参照が必要になります。

Error ID: BC30008

このエラーを解決するには

- 参照されないモジュールの名前をプロジェクト参照に含めます。

参照

処理手順
壊れた参照のトラブルシューティング
その他の技術情報
名前空間およびコンポーネントの参照

型 '<membername>' を含むアセンブリ '<assemblyname>' への参照が必要です。

エラー メッセージ
型 '<membername>' を含むアセンブリ '<assemblyname>' への参照が必要です。プロジェクトに参照を追加してください。

アセンブリ内の別の型に依存する型がプロジェクトにありますが、そのアセンブリがプロジェクト参照には含まれていません。

Error ID: BC30652

このエラーを解決するには

- プロジェクト内の依存する型についてプロジェクト参照を追加します。

参照
概念
プロジェクト参照

実装されたインターフェイス '<interfacename>' を含むアセンブリ '<assemblyname>' への参照が必要です。参照をプロジェクトに追加してください。

エラー メッセージ
実装されたインターフェイス '<interfacename>' を含む '<assemblyname>' アセンブリへの参照が必要です。プロジェクトに参照を追加してください。

プロジェクト内で直接参照されないダイナミック リンク ライブラリ (DLL: Dynamic-link Library) またはアセンブリでインターフェイスが定義されています。Visual Basic コンパイラでは、インターフェイスが複数の DLL またはアセンブリで定義されている場合に備えて、あいまいさを避けるために参照が必要になります。

Error ID: BC30009

このエラーを解決するには

- 参照されない DLL またはアセンブリの名前をプロジェクト参照に含めます。

参照

処理手順
壊れた参照のトラブルシューティング
その他の技術情報
名前空間およびコンポーネントの参照

イベント '<eventname>' の定義を含むアセンブリ '<assemblyname>' への参照が必要です。参照をプロジェクトに追加してください。

イベント '*<eventname>*' の定義を含むアセンブリ '*<assemblyname>*' への参照が必要です。プロジェクトに参照を追加します。

プロジェクト内で直接参照されないダイナミック リンク ライブラリ (DLL: Dynamic-link Library) またはアセンブリでイベントが定義されています。Visual Basic コンパイラでは、イベントが複数の DLL またはアセンブリで定義されている場合に備えて、あいまいさを避けるために参照が必要になります。

Error ID: BC30005

このエラーを解決するには

- 参照されない DLL またはアセンブリの名前をプロジェクト参照に含めます。

参照

処理手順

壊れた参照のトラブルシューティング

その他の技術情報

名前空間およびコンポーネントの参照

基本クラス '<classname>' を含むアセンブリ '<assemblyname>' への参照が必要です。参照をプロジェクトに追加してください。

エラー メッセージ
基本クラス '<classname>' を含むアセンブリ '<assemblyname>' への参照が必要です。プロジェクトに参照を追加してください。

プロジェクト内で直接参照されないダイナミック リンク ライブラリ (DLL: Dynamic-link Library) またはアセンブリでクラスが定義されています。Visual Basic コンパイラでは、クラスが複数の DLL またはアセンブリで定義されている場合に備えて、あいまいさを避けるために参照が必要になります。

Error ID: BC30007

このエラーを解決するには

- 参照されない DLL またはアセンブリの名前をプロジェクト参照に含めます。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
壊れた参照のトラブルシューティング
概念
プロジェクト参照
その他の技術情報
名前空間およびコンポーネントの参照

'ReDim' ステートメントには、配列の各次元の新しい境界の一覧をかっこで囲んだものが必要です。

配列の新しいサイズを指定するには、**ReDim** ステートメントの中で指定する必要があります。

Error ID: BC30670

このエラーを解決するには

- 配列の各ランクのサイズを指定します。次に例を示します。

```
ReDim arr(5, 6)
```

参照

関連項目

ReDim ステートメント (Visual Basic)

配列変数を宣言するために 'ReDim' ステートメントを使用することはできません。

ReDim は、既存の配列のサイズを変更するときにしか使用できません。

Error ID: BC30811

このエラーを解決するには

- 配列を宣言するときにサイズを指定します。次に例を示します。

```
Dim X(20) As Integer
```

参照

関連項目

配列の概要

ReDim ステートメント (Visual Basic)

概念

ReDim ステートメント (Visual Basic 6.0 ユーザー向け)

'RemoveHandler' は既に宣言されています。

カスタム イベントの宣言に **RemoveHandler** 宣言が複数あります。**RemoveHandler** 宣言は、イベント ハンドラを削除するためのプロシージャを宣言します。

Error ID: BC31128

このエラーを解決するには

- 余分な **RemoveHandler** ステートメントを削除します。

参照

関連項目

RemoveHandler ステートメント
Event ステートメント

イベント '<eventname>' に対して 'RemoveHandler' 定義が指定されていません。

イベントを **Custom** として宣言している場合は、イベント ハンドラ削除用のプロシージャを記述する必要があります。

Error ID: BC31131

このエラーを解決するには

1. **Custom Event** ステートメントと **End Event** ステートメントの間で、**RemoveHandler** を宣言します。
2. このイベント宣言内のその他のプロシージャが正しく終了していることを確認します。

参照

関連項目

RemoveHandler ステートメント
Event ステートメント

'RemoveHandler' 宣言の終わりには、対応する 'End RemoveHandler' を指定してください。

RemoveHandler 宣言の終わりには、対応する **End RemoveHandler** ステートメントを指定してください。

Error ID: BC31116

このエラーを解決するには

- **RemoveHandler** 宣言が **End RemoveHandler** ステートメントで終わっていることを確認します。

参照

関連項目

RemoveHandler ステートメント
Event ステートメント

関係演算子が必要です。

Case ステートメントに **Is** 句がありますが、=、> などの比較演算子がありません。**Case** ステートメントに演算子が含まれていない場合は = を使用します。**Is** は使用しません。

Error ID: BC30239

このエラーを解決するには

- **Is** キーワードを削除するか、後ろに比較演算子を記述します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

比較演算子 (Visual Basic)

概念

Visual Basic における比較演算子

'Resume' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Resume ステートメントを使用できるのはソース コード内だけです。

Error ID: BC30714

このエラーを解決するには

- デバッグ コードから **Resume** ステートメントを削除します。

参照

関連項目

Resume

その他の技術情報

Visual Studio でのデバッグ

'Resume' または 'GoTo' が必要です。

On Error ステートメントに **Resume** 句も **GoTo** 句も含まれていません。いずれかの句を使用する必要があります。

Error ID: BC32019

このエラーを解決するには

- **On Error** ステートメントに **Resume** 句と **GoTo** 句のいずれかを追加します。

参照

関連項目

On Error ステートメント (Visual Basic)
Resume ステートメント
GoTo ステートメント

応答ファイル <filename> が複数回含まれました。

応答ファイルが、複数回指定されています。応答ファイル内には、コンパイラ オプションやコンパイル対象のソース コード ファイルが記述されています。

Error ID: BC2003

このエラーを解決するには

- 余分な応答ファイル指定子を削除します。

参照

関連項目

Visual Basic コンパイラ オプション一覧 (カテゴリ別)

リソース名 '<resourcename>' を 2 回以上使用することはできません。

ダイナミック リンク ライブラリ (DLL: dynamic link library) または実行可能ファイルで指定したリソースの名前が、複数回使用されました。

Error ID: BC31502

このエラーを解決するには

- リソースに一意の名前を付けます。

参照

関連項目

/linkresource (Visual Basic)
/resource (Visual Basic)

関数 '<procedurename>' の戻り値の型は CLS に準拠していません。

Function プロシージャが `<CLSCompliant(True)>` でマーク付けされていますが、戻り値の型が `<CLSCompliant(False)>` でマーク付けされているか、マークが付けられていないか、または非準拠の型であるため不適切です。

プロシージャを 共通言語仕様 (CLS) に準拠させるためには、CLS 準拠の型のみを使う必要があります。これはパラメータの型、戻り値の型、およびすべてのローカル変数の型に対して言えることです。

次の Visual Basic データ型は CLS に準拠していません。

- SByte 型 (Visual Basic)
- UInteger データ型
- ULong データ型 (Visual Basic)
- UShort 型 (Visual Basic)

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に設定しなかった場合は、非準拠であると見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40027

このエラーを解決するには

- **Function** プロシージャがこの特定の型を返す必要がある場合は、**CLSCompliantAttribute** を削除します。このプロシージャを CLS に準拠させることはできません。
- **Function** プロシージャを CLS に準拠させる必要がある場合は、戻り値の型を CLS に準拠した最も近い型に変更します。たとえば、2,147,483,647 を超える値の範囲が必要でない場合は、**UInteger** の代わりに **Integer** を使用できます。範囲を拡張する必要がある場合は、**UInteger** を **Long** で置き換えてください。
- オートメーションまたは COM オブジェクトとやり取りする場合は、一部の型に .NET Framework とはデータ幅が異なるものがあることに注意してください。たとえば、**int** は他の環境では 16 ビットです。そのようなコンポーネントに 16 ビットの整数を返す場合は、Visual Basic のマネージ コードで、**Integer** 型ではなく **Short** 型で宣言してください。

参照

概念

CLS 準拠コードの記述

'Return' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Return ステートメントは分岐を行うため、デバッグ コンテキストでは使用できません。

Error ID: BC30147

このエラーを解決するには

- [イミディエイト] ウィンドウで **Return** ステートメントを実行しないようにします。

参照

関連項目

[イミディエイト ウィンドウ]

'AddHandler'、'RemoveHandler'、または 'RaiseEvent' にある 'Return' ステートメントは値を返すことができません。

Custom Event 宣言内の **AddHandler** メソッド、**RemoveHandler** メソッド、および **RaiseEvent** メソッドには、こうしたメソッドを終了する **Return** ステートメントを含むことができます。ただし、メソッドは値を返すことができないので、この **Return** ステートメントには戻り値を指定できません。

Error ID: BC30940

このエラーを解決するには

- **Return** ステートメントの後に続く式を削除します。

参照

関連項目

Event ステートメント
AddHandler
RemoveHandler
RaiseEvent
Return ステートメント (Visual Basic)

その他の技術情報

Visual Basic におけるイベント

Sub または Set の 'Return' ステートメントは、値を返すことができません。

Sub プロシージャと **Set** プロシージャは、値を返すことができません。

Error ID: BC30647

このエラーを解決するには

- 現在のプロシージャを関数に変更します。現在のプロシージャが、プロパティの一部になっている場合は **Get** プロパティ プロシージャに変更します。
- **ByRef** キーワードを使って参照渡しで渡されるパラメータの値を変更すると、**Sub** プロシージャから値を返すことができます。

参照

関連項目

Return ステートメント (Visual Basic)

概念

Sub プロシージャ
Function プロシージャ
Property プロシージャ

Function、Get、および演算子内の 'Return' ステートメントは、値を返さなければなりません。

Return ステートメントは、呼び出し元のプロシージャに値を返すときに使います。プログラム フローを制御するために単独で **Return** ステートメントを使うことはできません。

Error ID: BC30654

このエラーを解決するには

1. 関数またはプロシージャが返す値を指定します。
2. 現在のプロシージャを終了するには、**End** ステートメントを使います。

参照

関連項目

Return ステートメント (Visual Basic)
End (Visual Basic)

'<演算子>' の戻り値型およびパラメータ型は、'For' ステートメントで使用するためには '<typename>' でなければなりません。

For ループに指定されているカウンタ変数の型に、その変数の型のパラメータと戻り値を使った + 演算子または - 演算子が定義されていません。

カウンタ変数の型は、その変数の型で完全に機能する加算 (+) 演算子と減算 (-) 演算子をサポートする必要があります。つまり、両方のオペランドと戻り値の型が、カウンタ変数の型と同じであることが必要です。

カウンタ変数に数値データ型を使用する場合は、その変数の型で + 演算子と - 演算子を使用することが可能です。ユーザー定義のクラスまたは構造体を使用する場合は、そのクラスまたは構造体の型のオペランドと戻り値を使って 2 つの演算子を定義する必要があります。

Error ID: BC33039

このエラーを解決するには

1. カウンタ変数のデータ型のスペルが正しいことを確認します。
2. カウンタ変数にユーザー定義のクラスまたは構造体を使用する場合は、そのクラスまたは構造体で完全に機能する + 演算子と - 演算子を定義します。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

For...Next ステートメント (Visual Basic)
Operator ステートメント

概念

演算子プロシージャ

'<logicaloperator>' の戻り値およびパラメータの型を '<shortcircuitoperator>' 式で使用するには、'<typename>' にしなければなりません。

And 演算子または **Or** 演算子が、不適切なパラメータで宣言されているか、AndAlso 演算子 または OrElse 演算子 用の戻り値の型で宣言されています。

ショートサーキット演算子 (**AndAlso** または **OrElse**) を直接定義していないため、それらに対応する論理決定演算子を定義する必要があります。必須の演算子は、次の表に示すとおりです。

ショートサーキット演算子	論理演算子	決定演算子
AndAlso	And 演算子 (Visual Basic)	IsFalse 演算子
OrElse	Or 演算子 (Visual Basic)	IsTrue 演算子

Visual Basic では、これらの論理決定演算子を使って、**AndAlso** または **OrElse** のショートサーキット ロジックを作成します。これを適切に機能させるには、**And** 定義または **Or** 定義の両方のオペランドと戻り値の型は、包含型 (**And** または **Or** を定義するクラスまたは構造体の型) である必要があります。

Error ID: BC33034

このエラーを解決するには

- 両方のオペランドの型および戻り値の型を、演算子が定義されているクラスまたは構造体の型に変更します。

  または

- ショートサーキット演算子 (**AndAlso** または **OrElse**) には、それらに対応する **And** 演算子または **Or** 演算子が定義されているクラスまたは構造体の型であるオペランドを使用しないでください。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

Visual Basic の論理演算子とビット処理演算子

実行時例外がスローされました : <exception> - <exception>

デバッグ ウィンドウ内の式でランタイム エラーが発生しました。

Error ID: BC30703

このエラーを解決するには

- 状況に応じてコードを確認します。

参照

その他の技術情報

Visual Studio でのデバッグ

実行時例外がスローされました。

デバッグ ウィンドウ内の式でランタイム エラーが発生しました。

Error ID: BC30704

このエラーを解決するには

- コードを確認してください。

参照

その他の技術情報

Visual Studio でのデバッグ

ルート名前空間 '<namespacename>' は CLS に準拠していません。

アセンブリが `<CLSCompliant(True)>` としてマークされているのに、ルート名前空間の名前がアンダースコア (_) で始まっています。

プログラミング要素には 1 つ以上のアンダースコアを含めることができますが、共通言語仕様 (CLS) に準拠するためには、先頭をアンダースコアにしないでください。宣言された要素の名前 を参照してください。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40038

このエラーを解決するには

- CLS 準拠にする必要がある場合は、ルート名前空間の名前を変更し、アンダースコアで始まらないようにします。
- ルート名前空間の名前を変更できない場合は、アセンブリから **CLSCompliantAttribute** を削除するか、アセンブリを `<CLSCompliant(False)>` としてマークします。

参照

処理手順

方法 : アプリケーションの名前空間を変更する

関連項目

Namespace ステートメント

/rootnamespace

概念

Visual Basic における名前空間

宣言された要素の名前

Visual Basic の名前付け規則

CLS 準拠コードの記述

'Set' メソッドに 2 つ以上のパラメータを指定することはできません。

Set プロパティ プロシージャについて複数のパラメータが定義されています。

Error ID: BC31063

このエラーを解決するには

- **Set** プロパティ プロシージャから余分なパラメータを削除します。

参照

処理手順

方法 : フィールドおよびプロパティをクラスに追加する

概念

Property プロシージャ

プロパティとプロパティ プロシージャ

'Set' は既に宣言されています。

コード ブロックに複数の **Set** ステートメントが記述されています。**Set** ステートメントは、プロパティに値を代入するために使用する **Set** プロパティ プロシージャを宣言します。

Error ID: BC30444

このエラーを解決するには

- 余分な **Set** ステートメントを削除します。

参照

関連項目

Set ステートメント (Visual Basic)

プロパティ '<propertyname>' の 'Set' アクセサにアクセスできません。

ステートメントがプロパティの値を格納しようとしましたが、プロパティの **Set** プロシージャへのアクセス許可がありません。

Set ステートメント (Visual Basic) が Property ステートメントよりも制限の高いアクセス レベルでマーク付けされている場合にプロパティ値の設定を試みると、次のケースでエラーになります。

- **Set** ステートメントが Private (Visual Basic) でマーク付けされており、呼び出し元のコードがプロパティが定義されたクラスまたは構造体の外側にある場合。
- **Set** ステートメントが Protected (Visual Basic) でマーク付けされており、呼び出し元のコードがプロパティが定義されたクラスまたは構造体の内部にも、派生クラスの内部にもない場合。
- **Set** ステートメントが Friend (Visual Basic) でマーク付けされており、呼び出し元のコードがプロパティが定義されたのと同じアセンブリにない場合。

Error ID: BC31102

このエラーを解決するには

- プロパティが定義されたソース コードを変更できる場合は、**Set** プロシージャをプロパティ自体と同じアクセス レベルで宣言できないか検討してください。
- プロパティが定義されたソース コードを変更できない場合、または **Set** プロシージャをプロパティ自体よりも高いアクセス レベルで制限する必要がある場合は、プロパティ値を設定するステートメントをプロパティへのアクセスが可能なコード領域に移動することを検討します。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

概念

Property プロシージャ

'Select' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Select ステートメントはソース コードでのみ使用できます。

Error ID: BC30718

このエラーを解決するには

- デバッグ コードから **Select** ステートメントを削除します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

'Select Case' の終わりには、対応する 'End Select' を指定しなければなりません。

Select または **Select Case** ステートメントに対応する **End Select** ステートメントがありません。**Select** ブロックの最後には **End Select** ステートメントが必要です。

Error ID: BC30095

このエラーを解決するには

1. この **Select** ブロックが、入れ子になった **Select** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
2. **Select** ブロックの最後に **End Select** ステートメントを追加します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

セキュリティ属性 '<attributename>' が有効ではありません : <error>

共通言語ランタイムは、セキュリティ コンテキスト内の属性を受け入れません。

Error ID: BC30128

このエラーを解決するには

1. 属性を宣言から削除するか、またはコンテキストに対して有効な属性に置き換えます。
2. 二重引用符で囲まれたエラー メッセージを確認し、適切なアクションを実行します。

参照

処理手順

方法 : 独自の属性を定義する

セキュリティ属性 '<attributename>' をモジュールに適用することはできません。

共通言語ランタイムは、モジュールのセキュリティ属性を受け入れません。

Error ID: BC30131

このエラーを解決するには

- 宣言から属性を削除します。

参照

処理手順

方法 : 独自の属性を定義する

このコンテキストでは、式を評価中の値の変更は有効ではありません。

オブジェクトの状態を変更するステートメントは、このコンテキストで使用できません。

Error ID: BC30700

このエラーを解決するには

- その他の項目の状態を変更せずにステートメントが評価されるようにコードを変更します。

参照

関連項目

Nothing (Visual Basic)

共有 'Sub New' にパラメータを指定することはできません。

共有 **Sub New** ステートメントの中でパラメータが指定されていますが、この指定は無効です。

Error ID: BC30479

このエラーを解決するには

- パラメータを削除します。

参照

関連項目

Shared (Visual Basic)

共有 'Sub New' を '<specifier>' として宣言することはできません。

Shared Sub New 宣言内で指定子を使おうとしました。この使用方法は無効です。

Error ID: BC30480

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Shared (Visual Basic)
Sub ステートメント (Visual Basic)

プロパティ宣言で、'Shared' を '<specifier>' と組み合わせて使用することはできません。

Shared 修飾子と組み合わせようとした指定子は、プロパティの宣言でのこのような組み合わせにおいては無効です。

Error ID: BC30502

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Shared (Visual Basic)

メソッド宣言で、'Shared' を '<specifier>' と組み合わせて使用することはできません。

Shared 修飾子と、**Overridable**、**NotOverridable**、**MustOverride** などの指定子を組み合わせて使おうとしました。メソッドの宣言では、このような組み合わせは無効です。

Error ID: BC30501

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Shared (Visual Basic)
Overridable
NotOverridable
MustOverride

'Shared' 属性プロパティ '<propertyfield>' を、代入式のターゲットにすることはできません。

属性の **ReadOnly** プロパティまたは **Shared** プロパティに値を代入しようとしました。

Error ID: BC31500

このエラーを解決するには

1. プロパティに値を代入するステートメントを削除します。
2. 独自に作成したプロパティを使用している場合は、属性プロパティから **ReadOnly** 修飾子または **Shared** 修飾子を削除します。

参照

関連項目

Shared (Visual Basic)

その他の技術情報

Visual Basic における属性

'Set' ステートメントの終わりには、対応する 'End Set' を指定しなければなりません。

Set プロパティ プロシージャは、必ず **End Set** ステートメントで終了してください。

Error ID: BC30633

このエラーを解決するには

- **Set** プロパティ プロシージャの最後に **End Set** 構成体があることを確認します。

参照

関連項目

Property ステートメント

概念

プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

'Set' パラメータには、それを含むプロパティと同じ型を指定しなければなりません。

Set プロパティ プロシージャのパラメータが、それ自身が属するプロパティとは異なる型を持っています。

Error ID: BC31064

このエラーを解決するには

- プロパティのデータ型と一致するようにパラメータのデータ型を **Set** に変更します。たとえば、次のようにします。

```
Class Class1
   ' Declare a local variable to hold the property value.
   Private Fval As Integer

   Property F() As Integer
      Get
         Return Fval
      End Get
      Set(ByVal Value As Integer)
         Fval = Value
      End Set
   End Property
End Class
```

参照

処理手順

方法 : フィールドおよびプロパティをクラスに追加する

概念

Property プロシージャ

プロパティとプロパティ プロシージャ

'Set' パラメータを '<specifier>' に宣言することはできません。

Set プロパティ プロシージャ内のパラメータに無効な修飾子が指定されました。

Error ID: BC31065

このエラーを解決するには

- **ByRef**、**ParamArray**、**Optional** などの無効な修飾子を削除します。

参照

処理手順

方法 : フィールドおよびプロパティをクラスに追加する

概念

Property プロシージャ

プロパティとプロパティ プロシージャ

ソース ファイル <filename> が複数回指定されました。

1 つのファイルが複数回指定されています。

Error ID: BC2002

このエラーを解決するには

- 余分なファイル指定子を削除します。

参照

関連項目

My.Computer.FileSystem オブジェクト

<partialtypename>' に指定されたアクセス '<accesslevel1>' は、その他の partial 型の 1 つで指定されたアクセス '<accesslevel2>' と一致しません。

クラスまたは構造体が複数の部分宣言に分かれて定義され、アクセス レベルの指定に矛盾があります。

クラスまたは構造体を複数の宣言に分割して定義した場合、コンパイラはその型をすべての部分宣言の結合体として扱います。これは、メンバだけでなく、実装、継承、およびアクセス レベルについても同様です。

クラスまたは構造体の定義で、異なるアクセス レベルを混在させることはできません。**Protected Friend** というキーワードの組み合わせであっても、同じ宣言ステートメント内に隣接して指定する場合にのみ認められます。

Error ID: BC30925

このエラーを解決するには

- クラスの適切なアクセス レベルを決定し、それと矛盾するアクセス レベルの指定を削除します。

参照

関連項目

Partial (Visual Basic)
Class ステートメント (Visual Basic)
Structure ステートメント

概念

Visual Basic でのアクセス レベル
クラス : オブジェクトの設計図

その他の技術情報

構造体 : 独自のデータ型

指定子が重複しています。

1 つのキーワードが同じ宣言内で複数回使用されています。

Error ID: BC30178

このエラーを解決するには

- 重複して使用されているキーワードを宣言から削除します。

参照

その他の技術情報

Visual Basic における宣言された要素

指定子および属性は、'Namespace' ステートメントでは有効ではありません。

名前空間は、プログラム ブロックおよびプログラミング要素の階層構造用に使用されます。修飾子や属性は取りません。

Error ID: BC32026

このエラーを解決するには

- **Namespace** ステートメントからすべてのキーワードおよび属性ブロックを削除します。

参照

関連項目

Namespace ステートメント

概念

Visual Basic における名前空間
Visual Basic における属性
属性の適用

指定子および属性は、このステートメントでは有効ではありません。

Inherits などの変更できない宣言に、属性、または **Public** や **ReadOnly** などのキーワードが記述されています。このようなキーワードや属性を記述できるのは、変更できる宣言だけです。

Error ID: BC30193

このエラーを解決するには

- 宣言から属性やキーワードを削除します。

参照

その他の技術情報

キーワード (Visual Basic)
Visual Basic における属性

指定子は、'AddHandler'、'RemoveHandler' および 'RaiseEvent' メソッドでは有効ではありません。

AddHandler メソッド、**RemoveHandler** メソッド、および **RaiseEvent** メソッドの宣言を、**Public**、**ReadOnly** などのキーワードで変更できません。このようなキーワードを記述できるのは、変更できる宣言だけです。

Error ID: BC31135

このエラーを解決するには

- メソッドの宣言からキーワードを削除します。

参照

関連項目

Event ステートメント
AddHandler
RemoveHandler
RaiseEvent

その他の技術情報

キーワード (Visual Basic)
Visual Basic におけるイベント

指定子は、宣言の先頭でのみ有効です。

アクセシビリティとその他の事項を指定するキーワードは、**Dim** や **Function** などの宣言キーワードの前に記述する必要があります。

Error ID: BC30181

このエラーを解決するには

- 属性リスト (存在する場合) と宣言キーワードの間にあるすべての指定子を移動します。

参照

関連項目

Class ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)
Function ステートメント (Visual Basic)
Property ステートメント
Structure ステートメント

ステートメントを、メソッド本体の外側に表示することはできません。

実行可能なステートメントは、プロシージャ内で使用する必要があります。

Error ID: BC30689

このエラーを解決するには

- ステートメントをプロシージャまたはサブルーチンの中に移動します。

参照

概念

Visual Basic におけるプロシージャ

ステートメントをメソッド本体の内部に表示することはできません。

プロシージャ内に、別のプロシージャ宣言などの無効なステートメントがあります。プロシージャ宣言を入れ子にすることはできません。

Error ID: BC30289

このエラーを解決するには

- プロシージャ宣言を削除するか、または外側のプロシージャの外に移動します。

参照

概念

Visual Basic におけるプロシージャ

ステートメントを、プロパティ本体内部に記述することはできません。

エラー メッセージ
ステートメントはプロパティ本体内で宣言できません。プロパティの終わりと認識されます。

プロパティの中に予想外のステートメントがあります。

Error ID: BC30634

このエラーを解決するには

- プロパティの構文が正しいこと、および最後に **End Property** 構成体があることを確認します。

参照

関連項目
Property ステートメント
概念
プロパティ プロシージャの変更点 (Visual Basic 6.0 ユーザー向け)

ステートメントを列挙型の本体内に記述することはできません。

エラー メッセージ
ステートメントは列挙型の本体内に記述できません。列挙型の終わりと認識されます。

予想外の言語構成要素が見つかりました。**End Enum** 構成体が欠けていると認識され、これ以降のソース コードは列挙型の外部と見なされます。

Error ID: BC30619

このエラーを解決するには

1. 列挙型内のコードの構文を確認します。
2. インターフェイス定義の最後に **End Enum** 構成体があることを確認します。

参照

関連項目

Enum ステートメント (Visual Basic)

ステートメントをイベント本体内部に記述することはできません。

エラー メッセージ
ステートメントをイベント本体内部に記述することはできません。イベントの終わりと見なされます。

イベント本体内では無効なステートメントがイベント本体内に記述されています。

Error ID: BC31112

このエラーを解決するには

- 問題のステートメントの前に **End Event** を追加します。

参照

処理手順
アプリケーション イベントのサンプル
関連項目
Event ステートメント

ステートメントをインターフェイス本体内部に記述することはできません。

インターフェイス メンバの宣言に、メンバを終了するステートメントが **End** *membername* の形式で定義されています。

インターフェイスには、そのメンバのシグネチャのみを定義します。したがって、インターフェイスに定義されるプロシージャやプロパティには、名前とシグネチャを指定する開始行しかありません。インターフェイスの内部には、内部宣言 (つまり **End Function**、**End Property**、または **End Sub** ステートメント) のどのコードも定義しません。

Error ID: BC30603

このエラーを解決するには

- インターフェイス定義から **End** *membername* ステートメントを削除します。

参照

関連項目

Interface ステートメント (Visual Basic)
End (Visual Basic)

ステートメントをインターフェイス本体内部に記述できません (Visual Basic Error)

予想外の言語構成要素が見つかりました。**End Interface** 構成体が欠けていると認識され、これ以降のソース コードはインターフェイスの外部と見なされます。

Error ID: BC30604

このエラーを解決するには

1. インターフェイス定義内のコードの構文を確認します。
2. インターフェイス定義の最後に **End Interface** 構成体があることを確認します。

参照

関連項目

Interface ステートメント (Visual Basic)

その他の技術情報

Visual Basic におけるインターフェイス

If ステートメント行の外側でステートメント ブロックを終了することはできません。

単一行の **If** ステートメントに、コロン (:) で区切られた複数のステートメントが含まれています。そのうちの 1 つは、この単一行 **If** ステートメントの外側にあるコントロール ブロックの **End** ステートメントです。単一行の **If** ステートメントでは **End If** ステートメントを使用しません。

Error ID: BC32005

このエラーを解決するには

- **End If** ステートメントを含むコントロール ブロックの外側に、単一行の **If** ステートメントを移動します。

参照

関連項目

If...Then...Else ステートメント (Visual Basic)

ステートメントは、'Get' または 'Set' メソッドを宣言しません。

Property プロシージャに **Get** または **Set** の宣言ステートメントが指定されていません。プロパティは、**Property** ステートメントと **End Property** ステートメントの間のコード ブロックとして定義されます。このブロック内に、宣言ステートメントと終了ステートメントで囲まれた内部ブロックとして、各 **Property** プロシージャを記述します。

Error ID: BC30576

このエラーを解決するには

- **Get** または **Set** の宣言ステートメントを指定します。

参照

概念

Property プロシージャ

ステートメントは 'AddHandler'、'RemoveHandler'、または 'RaiseEvent' メソッドを宣言しません。

ステートメントは、**Custom Event** プロシージャに、**AddHandler**、**RemoveHandler**、または **RaiseEvent** の宣言ステートメントを指定しません。カスタム イベント宣言とは **Custom Event** ステートメントと **End Event** ステートメントとで囲まれたコード ブロックです。このブロック内に、宣言ステートメントと **End** ステートメントで囲んだ内部ブロックとして、各 **Custom Event** プロシージャを記述します。

Error ID: BC31113

このエラーを解決するには

- **AddHandler**、**RemoveHandler**、または **RaiseEvent** の宣言ステートメントを指定します。

参照

関連項目

Event ステートメント
AddHandler
RemoveHandler
RaiseEvent

その他の技術情報

Visual Basic におけるイベント

名前空間のステートメントが無効です。

ステートメントは、名前空間レベルでは宣言できません。名前空間レベルで許可されている宣言は、モジュール、インターフェイス、クラス、デリゲート、列挙体、および構造体の宣言だけです。

Error ID: BC30001

このエラーを解決するには

- モジュール、クラス、インターフェイス、構造体、列挙体、またはデリゲートの定義内にステートメントを移動します。

参照

概念

Visual Basic におけるスコープ
Visual Basic における名前空間

ステートメントは、インターフェイス内では有効ではありません。

インターフェイス内で無効なステートメントが検出されました。

Error ID: BC31041

このエラーを解決するには

- インターフェイス内のステートメントが、要素は定義するものの、実装は行っていないことを確認します。

参照

関連項目

Implements (Visual Basic)

その他の技術情報

Visual Basic におけるインターフェイス

ステートメントはメソッド内では有効ではありません。

このステートメントは、**Sub**、**Function**、プロパティ **Get** プロシージャ、および、プロパティ **Set** プロシージャ内では有効ではありません。一部のステートメントは、モジュール レベルまたはクラス レベルで配置できます。**Option Strict** などのその他のステートメントは、名前空間レベルで指定し、その他すべての宣言に先行する必要があります。

Error ID: BC30024

このエラーを解決するには

- 問題のステートメントをプロシージャから削除します。

参照

関連項目

Sub ステートメント (Visual Basic)
Function ステートメント (Visual Basic)
Get ステートメント
Set ステートメント (Visual Basic)

ステートメントはイベント '<eventname>' の含まれている 'AddHandler' を再帰的に呼び出しています。

イベント定義の **AddHandler** アクセサ内のステートメントからは、イベントを直接参照できません。

イベント ハンドラのリストを、イベントが定義されたクラス、構造体、またはモジュールに、プライベートなフィールドとして格納するという方法をお勧めします。詳細については、「方法 : ブロックを回避するイベントを宣言する」および「方法 : メモリの使用量を節約するイベントを宣言する」を参照してください。

Error ID: BC41998

このエラーを解決するには

- イベント定義を修正して、再帰を回避します。

参照

処理手順

方法 : ブロックを回避するイベントを宣言する
方法 : メモリの使用量を節約するイベントを宣言する

関連項目

AddHandler
Event ステートメント

ステートメントおよびラベルは、'Select Case' と最初の 'Case' の間では有効ではありません。

開始の **Select** ステートメントまたは **Select Case** ステートメントと、最初の **Case** ステートメントとの間に、コメント以外のステートメントがあります。

Error ID: BC30058

このエラーを解決するには

- 間にあるステートメントがコメントである場合は、先頭にコメント デリミタ (' または **REM**) を追加します。それ以外の場合は、ステートメントを移動または削除します。

参照

関連項目

Select...Case ステートメント (Visual Basic)

スタティック ローカル変数 '<variablename>' は、既に宣言されています。

同じ名前を持つ別の静的ローカル変数が既に宣言されています。

Error ID: BC31401

このエラーを解決するには

1. 余分な静的ローカル変数の宣言を削除します。
2. それぞれの静的ローカル変数に一意の名前を付けます。

参照

関連項目

Static (Visual Basic)

停止の要求は保留中です。

Visual Studio デバッガで、プロシージャ呼び出しが式に指定されていますが、スレッド停止の要求があります。

デバッガはアクティブでないスレッド上のプロシージャを呼び出しません。

Error ID: BC30946

このエラーを解決するには

- 可能であれば、スレッド中断の要求がどこから出ているかを調べ、次からは要求されないようにします。
- デバッグを終了させて、再起動します。

参照

処理手順

方法 : 実行を開始する

関連項目

Visual Basic の式

概念

方法 : デバッグの停止と実行の停止

コードのステップ実行の概要

その他の技術情報

Visual Studio でのデバッグ

'Stop' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Stop ステートメントと **End** ステートメントは実行を中断するため、デバッグ コンテキストでは使用できません。

Error ID: BC30122

このエラーを解決するには

- [イミディエイト] ウィンドウで **Stop** ステートメントまたは **End** ステートメントを実行しないようにします。

参照

関連項目

[イミディエイト ウィンドウ]

文字列定数の終わりには、二重引用符を指定しなければなりません。

文字列定数は、引用符で囲む必要があります。

ErrorID: BC30648

このエラーを解決するには

- リテラル文字列が引用符 (") で終わっていることを確認してください。他のテキスト エディタから値を貼り付けた場合は、貼り付けた文字が、左右で向きの違う二重引用符 (“ または ”) や 2 つの単一引用符 ('') などの二重引用符に似た文字ではなく、有効な二重引用符であることを確認します。

参照

その他の技術情報

Visual Basic における文字列

文字列定数を指定してください。

文字列定数が必要とされる場所に、二重引用符 (") 以外の文字があります。文字列が引用符で囲まれていないことが原因の 1 つとして考えられます。

Error ID: BC30217

このエラーを解決するには

1. 文字列が二重引用符で正しく囲まれているかどうかを確認します。
2. 文字列定数を指定します。

参照

概念

定数とリテラルのデータ型

'Delegate' の後には、'Sub' または 'Function' が必要です。

Delegate ステートメントで **Sub** プロシージャまたは **Function** プロシージャが指定されていません。**Delegate** キーワードの直後に、**Sub** キーワードまたは **Function** キーワードが必要です。

Error ID: BC30278

このエラーを解決するには

1. **Delegate** の直後に **Sub** キーワードまたは **Function** キーワードを追加します。
2. 必要に応じて、プロシージャ名、引数リスト、および戻り値の型を指定します。

参照

関連項目

Delegate ステートメント

概念

Visual Basic におけるプロシージャ

'Sub' または 'Function' が必要です。

Declare ステートメントに **Sub** キーワードも **Function** キーワードも指定されていません。**Declare** は、外部プロシージャを宣言します。プロシージャの種類を指定する必要があります。

Error ID: BC30215

このエラーを解決するには

- **Sub** キーワードまたは **Function** キーワードを **Declare** ステートメントに追加します。

参照

関連項目

Declare ステートメント

'Sub New' はインターフェイス メンバを実装できません。

Sub New はコンストラクタであり、メンバは実装できません。

Error ID: BC31042

このエラーを解決するには

- **Sub New** プロシージャから **Implements** ステートメントを削除します。

参照

関連項目

Implements (Visual Basic)

その他の技術情報

Visual Basic におけるインターフェイス

'Sub New' でイベントを処理することはできません。

Sub New と **Handles** を組み合わせて使おうとしましたが、この組み合わせは無効です。プロシージャ宣言の末尾にキーワード **Handles** を指定すると、そのプロシージャは、キーワード **WithEvents** を使って宣言されたオブジェクト変数が発生させるイベントを処理します。

Error ID: BC30497

このエラーを解決するには

- このコンテキストでは **New** を使用しないでください。

参照

関連項目

Handles
Dim ステートメント (Visual Basic)
WithEvents

'Sub New' を 'Overrides' として宣言することはできません。

コンストラクタが、継承されたコンストラクタをオーバーライドするように指定されています。コンストラクタはオーバーライドできません。

Error ID: BC30283

このエラーを解決するには

- **Sub** 宣言から **Overrides** キーワードを削除します。

参照

関連項目

コンストラクタとデストラクタの使用方法

概念

プロパティとメソッドのオーバーライド

'Sub New' をインターフェイスで宣言することはできません。

インターフェイスの中で **Sub New** を宣言しようとしました。**Sub New** はコンストラクタであるため、クラス作成時に 1 度だけ実行できます。同じクラスか派生クラスの別のコンストラクタで、コード行の先頭に記述して呼び出す以外に、明示的に呼び出すことはできません。

Error ID: BC30363

このエラーを解決するには

- **Sub New** を削除するか、またはコード内のより適切な位置に移動します。

参照

関連項目

コンストラクタとデストラクタの使用方法

'Sub New' を '<modifier>' として宣言することはできません。

Overrides などの修飾子を指定して **Sub New** を宣言しようとしました。この指定は無効です。

Error ID: BC30364

このエラーを解決するには

- 修飾子を削除します。

参照

関連項目

コンストラクタとデストラクタの使用方法

'Sub Main' が、'<name>' に見つかりませんでした。

Sub Main がありません。または、**Sub Main** の指定されている位置が正しくありません。

Error ID: BC30420

このエラーを解決するには

1. **Sub Main** ステートメントを指定します。既に **Sub Main** ステートメントを指定している場合は、コード内の適切な位置に移動します。**Sub Main** の詳細については、「Visual Basic の Main プロシージャ」を参照してください。
2. プロジェクトのスタートアップ オブジェクトの場所は、プロジェクト デザイナの [スタートアップ フォーム] ボックスに指定します。

詳細については、「 How to: Set Visual Basic Project Properties」を参照してください。

詳細については、「 方法 : アプリケーションのスタートアップ オブジェクトを変更する および
方法 : アプリケーションのスタートアップ オブジェクトを変更する (Visual Basic) および
方法 : アプリケーションのスタートアップ オブジェクトを変更する (Visual Basic)」を参照してください。

参照

関連項目

Sub ステートメント (Visual Basic)

概念

Visual Basic の Main プロシージャ

'Sub Main()' が '<namespace>' で 2 回以上宣言されています : <message>

Sub Main は名前空間内で 1 回だけ宣言できます。

Error ID: BC30738

このエラーを解決するには

- プロジェクトに **Sub Main** プロシージャが 1 つしかないことを確認します。

参照

概念

Visual Basic におけるプロシージャ

構造体に、'Inherits' ステートメントを指定することはできません。

構造体は、クラスとは違って継承をサポートしていません。

Error ID: BC30628

このエラーを解決するには

- **Inherits** ステートメントを構造体から削除します。
- 構造体の代わりにクラスを使用するようにアプリケーションをデザインし直します。

参照

概念

構造体とクラス

その他の技術情報

構造体 : 独自のデータ型

構造体で、共有されていない、パラメータなしの 'Sub New' を宣言することはできません。

構造体の中で **Sub New** コンストラクタを宣言する場合は、引数を受け取るか、**Shared** 修飾子を使って宣言する必要があります。

Error ID: BC30629

このエラーを解決するには

- 引数を受け取るように **Sub New** コンストラクタを変更します。
- コンストラクタを共有するように **Shared** 修飾子を適用します。

参照

関連項目

Structure ステートメント

その他の技術情報

構造体 : 独自のデータ型

'Structure' ステートメントの終わりには、対応する 'End Structure' を指定しなければなりません。

構造体は、**Structure** ステートメントで始まり、**End Structure** 構成要素で終わります。

Error ID: BC30624

このエラーを解決するには

- 構造体ブロックの最後に **End Structure** ステートメントがあることを確認します。

参照

概念

構造体とクラス

その他の技術情報

構造体 : 独自のデータ型

インターフェイス内の構造体を、'<specifier>' と宣言することはできません。

インターフェイス内で宣言した構造体に無効な修飾子が指定されました。

Error ID: BC31071

このエラーを解決するには

- **Public**、**Private**、**Shared**、**Friend**、**Protected**、**Protected Friend**、**Overridable** などの修飾子を削除します。

参照

関連項目

Structure ステートメント

その他の技術情報

Visual Basic におけるインターフェイス

'Structure' 制約は、同じ型パラメータに対して複数回指定できません。

制約リストに複数の Structure (Visual Basic) 制約が含まれています。

型パラメータに対する制約リストでは、型パラメータに渡される型引数を値型にするか (**Structure** 制約)、参照型にするか (Class (Visual Basic) 制約) を指定できます。同じ型パラメータに対して両方の制約を指定することはできません。また、これらを複数回指定することもできません。

Error ID: BC32102

このエラーを解決するには

- 冗長な **Structure** キーワードを削除します。制約リスト内に 1 つだけになるようにします。

参照

概念

Visual Basic におけるジェネリック型
値型と参照型

'Structure' 制約と特定のクラス型は、組み合わせて使用できません。

制約リストに Structure (Visual Basic) 制約と、定義済みのクラスの名前の両方が含まれています。

制約リストには、型パラメータに渡される型引数に対する要件を設定します。次の要件を任意に組み合わせて指定できます。

- 型引数は、1 つまたは複数のインターフェイスを実装する必要があります。
- 型引数は、最大で 1 つのクラスを継承する必要があります。
- 型引数は、作成側のコードがアクセスできるパラメータなしのコンストラクタを公開する必要があります (**New** 制約を含む)。

特定のクラスまたはインターフェイスを制約リストに含めない場合は、次のいずれかを指定することで、より汎用的な要件を設定できます。

- 型引数は、値型である必要があります (**Structure** 制約を含む)。
- 型引数は、参照型である必要があります (**Class** 制約を含む)。

同じ型パラメータに対して **Structure** と **Class** の両方を指定することはできません。また、これらを複数回指定することもできません。

Error ID: BC32108

このエラーを解決するには

- 型引数を値型にする場合は、制約リストからクラス名を削除します。
- 型引数を特定のクラス名から継承する場合は、制約リストから **Structure** キーワードを削除します。

参照

概念

Visual Basic におけるジェネリック型
値型と参照型

'Structure <structurename>' には、少なくとも 1 つのインスタンス メンバ変数宣言またはイベント宣言が含まれていなければなりません。

機能的に空の構造体があります。**Structure** ステートメントと **End Structure** ステートメントの間で少なくとも 1 つの変数メンバを宣言する必要があります。

Error ID: BC30281

このエラーを解決するには

- 構造体に **Dim** ステートメントまたは **Event** ステートメントを追加します。

参照

関連項目

Structure ステートメント
Dim ステートメント (Visual Basic)
Event ステートメント

構造体 '<structurename>' は少なくとも 1 つのインスタンス メンバ変数、または 'Custom' に設定されていない少なくとも 1 つのインスタンス イベント宣言を含まなければなりません。

構造体定義には、非共有変数および非共有の非カスタム イベントを含めることができません。

各構造体には、全インスタンスで共有 (Shared (Visual Basic)) するのではなく、特定のインスタンスそれぞれに適用される変数またはイベントが必要です。非共有の定数、プロパティ、およびプロシージャはこの要件を満たしません。また、非共有の変数がなく、非共有のイベントが 1 つだけ存在する場合、このイベントを Custom イベントにすることはできません。

Error ID: BC30941

このエラーを解決するには

- **Shared** ではない変数またはイベントを 1 つ以上定義します。イベントを 1 つだけ定義する場合には、そのイベントを非カスタムかつ非共有にする必要があります。

参照

処理手順

方法 : 構造体を宣言する

関連項目

Structure ステートメント

その他の技術情報

構造体 : 独自のデータ型

構造体 '<structurename>' に、それ自体のインスタンスを含めることはできません: <エラー>

構造体に宣言した変数が、それ自体のインスタンスで初期化されています。

構造体に他の構造体のインスタンスを含めることはできますが、それ自体の内部インスタンスを含めることはできません。これを行うと、無限再帰が起こります。

Error ID: BC30294

このエラーを解決するには

1. 宣言ステートメントの初期化式のスペルを確認します。
2. 同じ構造体の別のインスタンスを作成する場合、その構造体の外部に宣言して作成する必要があります。

参照

関連項目

Structure ステートメント

その他の技術情報

構造体 : 独自のデータ型

構造体 '<structurename>' には既定のプロパティがないため、インデックス処理を実行できません。

既定プロパティの構文を、既定プロパティを持たない構造体で使おうとしました。

Error ID: BC30690

このエラーを解決するには

- 標準プロパティの構文を使用して構造体のプロパティにアクセスします。

参照

概念

既定のプロパティの変更点 (Visual Basic 6.0 ユーザー向け)
構造体とその他のプログラミング要素
既定のプロパティ

条件付きコンパイル式に構文エラーがあります。

条件付きコンパイル式で原因を特定できない構文エラーが生成されます。

Error ID: BC31427

このエラーを解決するには

1. 条件付きコンパイルのドキュメントおよび式で使用しているキーワードのドキュメントを確認します。ドキュメントに記載されているプログラム例と、このエラーを生成するソース行を比較します。
2. エラーの原因を特定できない場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

その他の技術情報

条件付きコンパイル (Visual Basic)

キャスト演算子の構文エラーです。コンマで区切られた 2 つの引数が必要です。

CType 変換関数、または **DirectCast** か **TryCast** の変換キーワードが式に使用されていますが、引数が 1 つしか指定されていません。

CType、**DirectCast**、および **TryCast** では、いずれも引数が 2 つ必要です。1 つ目の引数は変換する式で、2 つ目の引数は変換後のデータ型またクラス型です。

Error ID: BC30944

このエラーを解決するには

- 変換に必要な引数を 2 つとも指定します。
- 特定の データ型変換関数の 1 つ (**CString** など) を使用する場合は、**CType** ではなくその関数名を使用する必要があります。その場合は、引数は 1 つしか指定しません。

参照

関連項目

CType 関数
DirectCast
TryCast
データ型変換関数

構文エラーです (Visual Basic エラー)。

ソース コード行で、原因を特定できない構文エラーが生成されます。

Error ID: BC30035

このエラーを解決するには

1. ソース コード行で使用されている各キーワードをドキュメントで確認します。ドキュメントに記載されているプログラム例と、このエラーを生成するソース行を比較します。
2. エラーの原因を特定できない場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

'SyncLock' ステートメントは、イミディエイト ウィンドウでは無効です。

SyncLock ステートメントはスレッドを同期するため、デバッグ コンテキストでは使用できません。

Error ID: BC30135

このエラーを解決するには

- [イミディエイト] ウィンドウで **SyncLock** ステートメントを実行しないようにします。

参照

関連項目

[イミディエイト ウィンドウ]

SyncLock ステートメント

'SyncLock' ステートメントの終わりには、対応する 'End SyncLock' が必要です。

SyncLock ブロックは、**SyncLock** キーワードで始まり、**End SyncLock** 構成要素で終わります。

Error ID: BC30675

このエラーを解決するには

- **SyncLock** ブロックの最後に **End SyncLock** 構成要素があることを確認します。

参照

関連項目

SyncLock ステートメント

'<typename>' は参照型でないため、'SyncLock' オペランドに型 '<typename>' を指定することはできません。

SyncLock ステートメントを使用すると、複数のスレッドによって同じステートメントが同時に実行されるのを防ぐことで、複数のステートメントを 1 つの式に同期させることができます。**SyncLock** ステートメント内の式の型は、クラス、モジュール、インターフェイス、配列、デリゲートなどの参照型である必要があります。

Error ID: BC30582

このエラーを解決するには

- 型を適切な参照型に変更します。

参照

関連項目

SyncLock ステートメント

その他の技術情報

Visual Basic におけるマルチスレッド

保留の要求は保留中です。

Visual Studio デバッガで、プロシージャ呼び出しが式に指定されていますが、スレッド中断の要求があります。

デバッガはアクティブでないスレッド上のプロシージャを呼び出しません。

Error ID: BC30947

このエラーを解決するには

- 可能であれば、スレッド中断の要求がどこから出ているかを調べ、次からは要求されないようにします。
- デバッグを終了させて、再起動します。

参照

処理手順

方法 : 実行を開始する

関連項目

Visual Basic の式

概念

方法 : デバッグの停止と実行の停止

コードのステップ実行の概要

その他の技術情報

Visual Studio でのデバッグ

'System.Void' は、GetType 式でのみ使用できます。

代入ステートメントまたは宣言に含まれる式が、変数、プロシージャのパラメータ、関数の戻り値、または型引数の型として Void を使用しています。

Void 構造体は、.NET Framework (特に Visual C# および Visual C++) で内部的に使用される特別な型です。これは値を返さないメソッドの、戻り値の型を表します。Visual Basic では、値を返さない場合は **Sub** プロシージャを使用し、値を返す場合は **Function** プロシージャを使います。

GetType 演算子 演算子を使用して参照変数をテストすると、そのランタイム型が **Void** かどうかを確認できます。しかし、それ以外のコンテキストで **Void** を使うことはできません。

Error ID: BC31422

このエラーを解決するには

1. 変数のランタイム型を **Void** と比較する場合は、**GetType** 演算子を使います。
2. ランタイム型と **Void** を比較する特別な理由がないのなら、この型への参照自体を削除します。

参照

関連項目

GetType 演算子
Void

'System.STAThreadAttribute' および 'System.MTAThreadAttribute' の両方を同じメソッドに適用することはできません。

System.STAThreadAttribute 属性と **System.MTAThreadAttribute** 属性とは同時に指定できません。

Error ID: BC31512

このエラーを解決するには

- **System.MTAThreadAttribute** または **System.STAThreadAttribute** のいずれか一方のキーワードだけを指定します。

参照

関連項目

STAThreadAttribute Class
MTAThreadAttribute Class

その他の技術情報

Visual Basic における属性

'System.STAThreadAttribute' および 'System.MTAThreadAttribute' の両方に '1' を適用することはできません。

System.STAThreadAttribute 属性と **System.MTAThreadAttribute** 属性とは同時に指定できません。

Error ID: BC31513

このエラーを解決するには

- **System.MTAThreadAttribute** または **System.STAThreadAttribute** のいずれか一方のキーワードだけを指定します。

参照

関連項目

STAThreadAttribute Class
MTAThreadAttribute Class

その他の技術情報

Visual Basic における属性

'System.Runtime.InteropServices.DllImportAttribute' はインターフェイス メソッドに適用できません。

プロシージャがインターフェイスの中で定義されていますが、このプロシージャ定義は DllImportAttribute を適用しています。

共通言語ランタイム (CLR) は、.NET Framework 外のアンマネージ DLL (Dynamic-Link Library) の中で定義されている置換プロシージャを指定しているときに、この属性と EntryPoint プロパティを認識します。**DllImportAttribute** が適用されているプロシージャがコードから呼び出されると、共通言語ランタイムは、そのプロシージャの代わりに指定されたアンマネージ プロシージャを呼び出します。

インターフェイス内のプロシージャ定義は実装を一切含んでいないので、.NET Framework 外部のアンマネージ プラットフォームと相互運用することはできません。

Error ID: BC31530

このエラーを解決するには

- このプロシージャの定義から **DllImportAttribute** を削除します。

参照

関連項目

Interface ステートメント (Visual Basic)
DllImportAttribute

'System.Runtime.InteropServices.DllImportAttribute' はインスタンス メソッドに適用できません。

非共有のプロシージャが DllImportAttribute で宣言されています。

共通言語ランタイム (CLR) は、.NET Framework 外のアンマネージ DLL (Dynamic-Link Library) の中で定義されている置換プロシージャを指定しているときに、この属性と EntryPoint プロパティを認識します。**DllImportAttribute** が適用されているプロシージャがコードから呼び出されると、共通言語ランタイムは、そのプロシージャの代わりに指定されたアンマネージ プロシージャを呼び出します。

.NET Framework 外のアンマネージ プラットフォームは、.NET Framework と同じようには非共有プロシージャをサポートしないので、非共有プロシージャを使用してこれらのプラットフォームと相互運用することはできません。

Error ID: BC31529

このエラーを解決するには

- プロシージャをクラスまたは構造体の各インスタンスに個別に適用する必要がない場合は、そのプロシージャを **Shared** として宣言します。
- プロシージャを **Shared** にできない場合は、そのプロシージャの宣言から **DllImportAttribute** を削除します。

参照

関連項目

Shared (Visual Basic)
DllImportAttribute

'System.Runtime.InteropServices.DllImportAttribute' は、'AddHandler'、'RemoveHandler' または 'RaiseEvent' メソッドには適用できません。

DllImportAttribute 属性が、**AddHandler** メソッドか **RemoveHandler** メソッド、**RaiseEvent** メソッドの宣言に適用されました。この属性を使用できるのは、空の **Function** または **Sub** だけです。

Error ID: BC31531

このエラーを解決するには

- メソッドの宣言から **DllImportAttribute** 属性を削除します。

参照

関連項目

Event ステートメント
AddHandler
RemoveHandler
RaiseEvent
DllImportAttribute

'System.Runtime.InteropServices.DllImportAttribute' を、空でない本体を持つ Sub、Function、または Operator に適用することはできません。

空ではない **Sub**、**Function**、または **Operator** に **DllImportAttribute** 属性が適用されました。

Error ID: BC31522

このエラーを解決するには

- **Sub**、**Function**、または **Operator** からすべてのコードを削除して、この属性を使用します。

参照

関連項目

DllImportAttribute Class
Declare ステートメント

'System.Runtime.InteropServices.DllImportAttribute' は、ジェネリックまたはジェネリック型に含まれるメソッドには適用できません。

プロシージャ宣言に DllImportAttribute が指定されていますが、このプロシージャはジェネリック プロシージャであるか、ジェネリックなクラスまたは構造体に含まれています。

共通言語ランタイム (CLR: Common Language Runtime) は、この属性と属性の EntryPoint プロパティを見つけると、.NET Framework の外部のアンマネージ ダイナミック リンク ライブラリ (DLL) に定義された置換プロシージャが指定されたと認識します。**DllImportAttribute** が適用されたプロシージャが呼び出されると、共通言語ランタイムは指定されたアンマネージ プロシージャを代わりに呼び出します。

.NET Framework 外のアンマネージ プラットフォームはジェネリック型を認識しないため、ジェネリック型を使用して相互運用できません。

Error ID: BC31526

このエラーを解決するには

- プロシージャとそのコンテナのどちらもジェネリックにする必要がない場合は、ジェネリックを指定しないように **Of** 句を削除します。
- プロシージャまたはそのコンテナをジェネリックにする必要がある場合は、このプロシージャの宣言から **DllImportAttribute** を削除します。

参照

関連項目

DllImportAttribute

概念

Visual Basic におけるジェネリック型

'System.Runtime.InteropServices.DllImportAttribute' を 'Get' または 'Set' に適用することはできません。

Get プロパティ プロシージャまたは **Set** プロパティ プロシージャに **DllImportAttribute** 属性が適用されました。

Error ID: BC31524

このエラーを解決するには

- **DllImportAttribute** を、**Get** プロパティ プロシージャおよび **Set** プロパティ プロシージャから削除します。

参照

関連項目

DllImportAttribute Class

概念

Property プロシージャ

'System.Runtime.InteropServices.DllImportAttribute' を 'Declare' に適用することはできません。

Declare 関数に **DllImportAttribute** 属性が適用されました。この属性を使用できるのは、空の **Function** または **Sub** だけです。

Error ID: BC31523

このエラーを解決するには

- **Declare** ステートメントから **DllImportAttribute** 属性を削除します。

参照

関連項目

DllImportAttribute Class
Declare ステートメント

'Microsoft.VisualBasic.ComClassAttribute' は、既定のプロパティに対して 0 を予約するため、'System.Runtime.InteropServices.DispIdAttribute' 値を '<typename>' に適用できません。

DispIdAttribute 属性ブロックでディスパッチ ID (DISPID : dispatch id) の値に 0 が指定されていますが、値 0 は、**COMClassAttribute** の適用先となるクラスの既定のプロパティを表すために予約されています。

DISPID は、COM オブジェクトによって公開されるプロパティやメソッドにアクセスするために、**IDispatch:Invoke** メソッドへの引数として COM 内で使用されます。

Error ID: BC32505

このエラーを解決するには

- **DispIdAttribute** で DISPID に 0 より大きい値を指定します。

参照

関連項目

ComClassAttribute クラス
DispIdAttribute

概念

Visual Basic で使用される属性
属性の適用

'Microsoft.VisualBasic.ComClassAttribute' は 0 より小さい値を予約するため、'System.Runtime.InteropServices.DispIdAttribute' 値を '<typename>' に適用できません。

COMClassAttribute 属性ブロックでディスパッチ ID (DISPID: dispatch identifier) の値が 0 未満に指定されていますが、その値は、DispIdAttribute の適用先となるクラスの専用の関数用に予約されています。

DISPID は、COM オブジェクトが公開するプロパティやメソッドにアクセスするために、**IDispatch:Invoke** メソッドへの引数として COM 内で使用されます。

Error ID: BC32506

このエラーを解決するには

- **DispIdAttribute** の DISPID に 0 より大きい値を指定します。

参照

関連項目

ComClassAttribute クラス
DispIdAttribute

概念

Visual Basic で使用される属性
属性の適用

'System.ObsoleteAttribute' を 'AddHandler'、'RemoveHandler'、または 'RaiseEvent' 定義に適用できません。

エラー メッセージ
'System.ObsoleteAttribute' は、'AddHandler'、'RemoveHandler'、'RaiseEvent' の定義には適用できません。必要であれば、イベントに直接属性を適用します。

カスタム イベントの **AddHandler** プロシージャ、**RemoveHandler** プロシージャ、または **RaiseEvent** プロシージャに ObsoleteAttribute が適用されています。

ObsoleteAttribute はプログラミング要素を、使用されなくなった要素としてマーク付けし、この先のバージョンで削除される予定であることをユーザーに知らせます。

カスタム イベントの特定の部分が使用されているのに、他の部分を使用されなくなったとマーク付けしても意味がありません。

Error ID: BC31142

このエラーを解決するには

- 個々のプロシージャ宣言から **ObsoleteAttribute** を削除し、イベント宣言全体に適用します。

参照

関連項目
Event ステートメント
AddHandler ステートメント
RemoveHandler ステートメント
RaiseEvent ステートメント
ObsoleteAttribute

'System.Nullable' は、型パラメータ '<typeparametername>' の 'Structure' 制約を満たしていません。

Structure 制約のある型パラメータに Nullable の型引数を渡してジェネリック型が呼び出されました。

共通言語ランタイム (CLR: Common Language Runtime) では、**Nullable** 構造体をそれ自体への型引数として使うことは認められていません。これも構造体には違いなく、他の点では **Structure** 制約を満たしますが、再帰的に使用した場合には `Nullable(Of Nullable(Of Nullable))` などの厄介な構造が作成される可能性があります。

Error ID: BC32115

このエラーを解決するには

- **Structure** 制約を型パラメータから削除するか、型引数を **Nullable** 以外の値型に変更します。

参照

関連項目

Structure (Visual Basic)
Nullable

概念

Visual Basic におけるジェネリック型

System.Diagnostics.DebuggerHiddenAttribute は、プロパティ定義に適用するときに 'Get' または 'Set' に適用しません。

エラー メッセージ
System.Diagnostics.DebuggerHiddenAttribute をプロパティ定義に適用しても、'Get' または 'Set' に適用されません。この属性を 'Get' および 'Set' プロシージャに必要に応じて直接適用してください。

DebuggerHiddenAttribute がプロパティ宣言に適用されています。

ソース コードでは、**DebuggerHiddenAttribute** をプロシージャに適用できます。それによって、Visual Studio 2005 デバッガに、プロシージャの内部で停止しないこと、およびプロシージャ内のブレークポイントの設定を許可しないことを通知できます。

プロパティに **DebuggerHiddenAttribute** を適用することはできますが、何の効果もありません。効果が得られるのは、プロパティの **Get** プロシージャまたは **Set** プロシージャに適用した場合だけです。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40051

このエラーを解決するには

- **DebuggerHiddenAttribute** をプロパティ宣言から削除し、プロパティの **Get** プロシージャまたは **Set** プロシージャに、必要に応じて適用します。

参照

関連項目
Property ステートメント
Get ステートメント
Set ステートメント (Visual Basic)
DebuggerHiddenAttribute
概念
Property プロシージャ

System.CLSCompliantAttribute はプロパティ 'Get' または 'Set' に適用できません。

プロパティの定義で、**Get** ステートメントまたは **Set** ステートメントに CLSCompliantAttribute 属性が適用されています。

プロパティを 共通言語仕様 (CLS) に準拠させるには、プロパティ全体が `<CLSCompliant(True)>` でマーク付けされていることが必要です。**CLSCompliantAttribute** は **Get** ステートメントまたは **Set** ステートメントではなく、Property ステートメントに適用する必要があります。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の **isCompliant** パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40043

このエラーを解決するには

- **Get** ステートメントまたは **Set** ステートメントから **CLSCompliantAttribute** を削除します。
- プロパティを CLS 準拠にする場合は、**Property** ステートメントを `<CLSCompliant(True)>` でマーク付けします。

参照

関連項目

Get ステートメント
Set ステートメント (Visual Basic)

概念

CLS 準拠コードの記述

ファイル '<filename>' はテキスト ファイルではありません。

指定したファイルはバイナリ ファイルであるため書き込むことができません。

Error ID: BC2015

このエラーを解決するには

- 正しいファイル名およびパスを指定していることを確認します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.FileSystem.WriteAllText メソッド

My.Computer.FileSystem.WriteAllBytes メソッド

その他の技術情報

Visual Basic でのファイルへの書き込み

既定のアセンブリ参照 '<reference>' は既に自動的に追加されています。無視されます

既定のアセンブリ参照の追加が試行されましたが、そのアセンブリは既に追加されているため、試行が無視されます。

Error ID: BC2024

このエラーを解決するには

- 不要なコードを削除します。

参照

概念

アセンブリ

ConnectionTimeout は 0 より大きくなければなりません。

My.Computer.Network オブジェクトを使用してファイルをアップロードおよびダウンロードする場合、*connectionTimeout* には 0 よりも大きい値を指定する必要があります。

このエラーを解決するには

- *connectionTimeout* に 0 よりも大きい値を指定します。

参照

処理手順

方法 : Visual Basic でファイルをアップロードする

方法 : Visual Basic でファイルをダウンロードする

関連項目

My.Computer.Network.UploadFile メソッド

My.Computer.Network.DownloadFile メソッド

その他の技術情報

Visual Basic による .NET Framework でのネットワーク操作

'<キーワード>' キーワードは、継承されたメンバをオーバーロードするために使用されます。'Sub New' をオーバーロードするときには '<キーワード>' を使用しないでください。

コンストラクタが **Overloads** キーワードを使って宣言されています。

Visual Basic では、コンストラクタの継承またはオーバーロードはサポートされていません。

Error ID: BC32040

このエラーを解決するには

- すべてのコンストラクタ宣言から **Overloads** キーワードを削除します。

参照

関連項目

Overloads
コンストラクタとデストラクタの使用方法

値 '<value>' はオプション '<optionname>' に対して無効です。

無効な値がコマンド ライン オプションに指定されました。

Error ID: BC2014

このエラーを解決するには

- 値を調べて、無効な値であることを確認してください。

参照

関連項目

Switch 関数

型 '<typename>' は、配列の要素の型、戻り値の型、フィールドの型、汎用引数の型、'ByRef' パラメータ型、または 'Object' または 'ValueType' に変換された式の型にすることはできません

制限された型を持つ変数、プロシージャのパラメータ、型パラメータ、関数の戻り値、または配列が式に宣言されています。

共通言語ランタイム (CLR: Common Language Runtime) には、特別な言語でしかサポートしない型があります。これらの型をアプリケーション内でデータ型として使用することはできません。このような型には、ArgIterator、RuntimeArgumentHandle、TypedReference の各構造体などがあります。

Error ID: BC31396

このエラーを解決するには

- 制限された構造体をデータ型の宣言に使わないようにします。

参照

関連項目

ArgIterator
RuntimeArgumentHandle
TypedReference

対象の .NET Compact Framework バージョンは、ANSI、Auto、または Unicode 修飾子の使用をサポートしていません

使用している .NET Compact Framework のバージョンは、**Ansi**、**Auto**、および **Unicode** の各修飾子をサポートしていません。

Error ID: BC30763

このエラーを解決するには

- 修飾子を削除します。

参照

関連項目

Ansi
Auto
Unicode (Visual Basic)

対象の .NET Compact Framework バージョンは、'End' ステートメントをサポートしていません。

使用している .NET Compact Framework のバージョンは、**End** ステートメントをサポートしていません。

Error ID: BC30769

このエラーを解決するには

- ステートメントを削除します。

参照

関連項目

End (Visual Basic)

対象の .NET Compact Framework バージョンは、遅延バインディングされたオーバーロードの解決をサポートしていません。

対象の .NET Compact Framework バージョンは、遅延バインディングのオーバーロードの解決をサポートしていません。

Error ID: BC30764

このエラーを解決するには

- 型オブジェクトのパラメータを適切な型にキャストします。

参照

概念

オーバーロードの解決法

その他の技術情報

Visual Basic におけるオブジェクト

.NET Compact Framework のターゲット バージョンでは、遅延バインディングはサポートされていません。

使用している .NET Compact Framework のバージョンは、遅延バインディングをサポートしていません。

Error ID: BC30762

このエラーを解決するには

1. オブジェクトとして宣言された変数に対する、関数、サブ関数、またはプロパティの呼び出しを回避します。
2. オブジェクト変数の配列としての使用を回避します。
3. オブジェクト変数を使用したディクショナリ メンバ アクセス式の使用を回避します。

参照

概念

コード内の特殊文字

その他の技術情報

Visual Basic におけるオブジェクト

エラーの最大数が制限を超えました。

コンパイラのエラーの最大数である 100 個に到達しました。

Error ID: BC2020

このエラーを解決するには

- 各エラーを順次デバッグするか解決して、生成されるエラーの数を減らします。

参照

概念

エラーの種類

この継承は、<type1> '<typename1> とその入れ子になった <type2> '<typename2>' の間で循環依存の関係が発生する原因になります。

継承構造が原因で、入れ子になったクラスの間で循環依存が起こります。つまり、2 つのクラスが互いに継承し合っています。

このエラー メッセージは次のようなコードで発生します。

```
Public Class c1
    Inherits c3.c4
    Public Class c2
    End Class
End Class
Public Class c3
    Inherits c1.c2
    Public Class c4
    End Class
End Class
```

このコードでは、クラス `c1` がクラス `c4` を継承していますが、`c4` は `c3` の内部に入れ子になっており、その `c3` は `c1` の内部に入れ子になっている `c2` を継承しています。

Error ID: BC30907

このエラーを解決するには

- 循環依存が起きないように、継承構造を変更します。

参照

概念

継承の基本

'<procedurename>'に対する引数が多すぎます。

プロシージャ呼び出しで指定した引数の数が、プロシージャについて定義されている引数の数を超えています。

Error ID: BC30057

このエラーを解決するには

- プロシージャの定義を調べ、余分な引数を呼び出しから削除します。

参照

概念

プロシージャのパラメータと引数

'<generictypename>' の型引数が少なすぎます。

ジェネリック クラス、構造体、インターフェイス、またはデリゲートが、型パラメータよりも少ない型引数を指定して呼び出されています。

ジェネリック型を呼び出すときは、そのジェネリック型で定義されているそれぞれの型パラメータに 1 つの型引数を指定する必要があります。コンパイラはジェネリック プロシージャに対してのみ型推論を行い、ジェネリック型に対しては行いません。

Error ID: BC32042

このエラーを解決するには

- 型引数リストに型引数を追加して、呼び出そうとするジェネリック型のそれぞれの型パラメータに対して型引数が 1 つあるようにします。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

Visual Basic におけるジェネリック プロシージャ

'Throw' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Throw ステートメントはソース コードでのみ使用できます。

Error ID: BC30725

このエラーを解決するには

- デバッグ コードから **Throw** ステートメントを削除します。

参照

関連項目

Throw ステートメント (Visual Basic)

その他の技術情報

Visual Studio でのデバッグ

'Throw' ステートメントでは、'Catch' ステートメントの外側、または 'Finally' ステートメントの内側にあるオペランドを省略できません。

Catch ステートメントの外側の **Throw** ステートメントでは、例外オブジェクトの名前を指定する必要があります。

Error ID: BC30666

このエラーを解決するには

1. **System.Exception** から派生した例外オブジェクトの名前を指定します。
2. コードを再構成して、**Throw** ステートメントを **Catch** ブロックの中に移動します。

参照

関連項目

Throw ステートメント (Visual Basic)

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の例外クラス

その他の技術情報

Visual Basic での例外およびエラー処理

'Throw' オペランドは、'System.Exception' から派生しなければなりません。

Throw に指定する引数は、**System.Exception** のインスタンスであるか、**System.Exception** の派生クラスのインスタンスである必要があります。

Error ID: BC30665

このエラーを解決するには

- 次の例のように、**System.Exception** から派生した引数を指定します。

```
Throw New System.Exception("This is an error.")
```

参照

関連項目

Throw ステートメント (Visual Basic)

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の例外クラス

その他の技術情報

Visual Basic での例外およびエラー処理

モジュール内の型を '<specifier>' に宣言することはできません。

無効な修飾子を適用した型がモジュール内で宣言されました。

Error ID: BC30735

このエラーを解決するには

- モジュール内で宣言した型に対して **Protected** 修飾子や **Protected Friend** 修飾子を使用しないようにします。

参照

関連項目

Module ステートメント
Dim ステートメント (Visual Basic)

型が必要です。

型名として認識できないソース コード文字があります。このコンテキストでは、基本データ型、または型を表す宣言済みの要素名が必要です。

Error ID: BC30182

このエラーを解決するには

- 型が定義されているかどうかを確認します。
- 型名のスペルが正しいかどうかを確認します。

参照

関連項目

Class ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)
Function ステートメント (Visual Basic)
Property ステートメント
Structure ステートメント

型文字は、このコンテキストでは有効ではありません。

このコンテキストにおいて無効な $ などの文字を使おうとしました。

Error ID: BC30468

このエラーを解決するには

- 無効な文字を削除します。

参照

概念

型文字

型の制約を 'NotInheritable' クラスに指定することはできません。

NotInheritable としてマークされたクラスが制約リストに含まれています。

型パラメータの制約リストには、最大 1 つのクラスを指定できます。その型パラメータに渡す型引数は、指定のクラスを継承している必要があります。したがって、シール クラス、つまり **NotInheritable** として宣言されたクラスを型パラメータの制約に指定することはできません。

Error ID: BC32060

このエラーを解決するには

- 型パラメータがクラスを継承できるようにする必要があり、そのクラスの定義を制御できる場合は、クラスの定義から **NotInheritable** 宣言を削除します。
- クラスを **NotInheritable** のままにする必要がある場合は、そのクラスを制約として使用することはできません。そのクラス名を制約リストから削除してください。

参照

概念

Visual Basic におけるジェネリック型

型の制約 '<式>' は、クラスかインターフェイスでなければなりません。

制約リストに、型パラメータに対する有効な制約を表していない式が含まれています。

制約リストには、型パラメータに渡される型引数に対する要件を設定します。次の要件を任意に組み合わせて指定できます。

- 型引数は、1 つまたは複数のインターフェイスを実装する必要がある
- 型引数は、最大で 1 つのクラスを継承する必要がある
- 型引数は、作成しているコードからアクセス可能なパラメータなしのコンストラクタを公開する必要がある
- 型引数は参照型であることが必要か、または値型であることが必要

Error ID: BC32048

このエラーを解決するには

- 式とその要素のスペルが正しいことを確認します。
- 式が上記の一覧にある要件を満たしていない場合は、式を制約リストから削除します。
- 式がインターフェイスまたはクラスを参照している場合は、コンパイラがそのインターフェイスまたはクラスにアクセスできることを確認してください。名前の修飾が必要な場合や、プロジェクトへの参照を追加することが必要な場合があります。詳細については、「同じ名前を持つ複数の変数がある場合に参照を解決する」の「プロジェクトへの参照」を参照してください。

参照

処理手順

方法 : 宣言された要素名を修飾する
方法 : Visual Studio (C#、J#) のリファレンスを追加および削除する

概念

Visual Basic におけるジェネリック型
値型と参照型

型文字は Imports エイリアスでは許可されていません。

Imports ステートメントのインポート エイリアスに、1 つ以上の識別子の型宣言文字が含まれています。

インポート エイリアスは Visual Basic で有効な名前であることが必要です。識別子の型宣言文字 (**%**、**&**、**@**、**!**、**#**、および **$**) は、使用可能な文字には含まれません。宣言された要素の名前 を参照してください。

Error ID: BC31398

このエラーを解決するには

- インポート エイリアスから識別子の型宣言文字を削除します。

参照

関連項目

Imports ステートメント

概念

型文字
宣言された要素の名前
同じ名前を持つ複数の変数がある場合に参照を解決する

型文字は、ラベル識別子では許可されていません。

ステートメント ラベルに、少なくとも 1 つの識別子の型宣言文字が含まれています。

ステートメント ラベルは Visual Basic で有効な名前であることが必要です。識別子の型宣言文字 (**%**、**&**、**@**、**!**、**#**、および **$**) は、指定可能な文字に含まれません。宣言された要素の名前 を参照してください。

Error ID: BC31395

このエラーを解決するには

- ステートメント ラベルから識別子の型宣言文字を削除します。

参照

処理手順

方法 : ステートメントにラベル付けする

概念

型文字

宣言された要素の名前

型文字は型パラメータの宣言で使用できません。

型パラメータの宣言に、識別子の型宣言文字が少なくとも 1 つ含まれています。

ジェネリック型の型パラメータには、有効な Visual Basic の名前を指定する必要があります。識別子の型宣言文字 (**%**、**&**、**@**、**!**、**#**、および **$**) は、使用可能な文字には含まれません。宣言された要素の名前 を参照してください。

Error ID: BC32041

このエラーを解決するには

- 型パラメータの宣言から識別子の型宣言文字を削除します。

参照

関連項目

型リスト

概念

型文字
宣言された要素の名前
Visual Basic におけるジェネリック型

'Sub' は値を返さないため、'Sub' 宣言で型文字を使用することはできません。

Sub プロシージは、**Function** プロシージャと同様に、引数を受け取って一連のステートメントを実行できる、独立したプロシージャです。**Function** プロシージャとは異なり、**Sub** プロシージャは値を返さないため、型宣言を含めることはできません。

Error ID: BC30303

このエラーを解決するには

- **Sub** プロシージャを **Function** プロシージャに変更します。

参照

概念

Sub プロシージャ
Function プロシージャ

型文字 '<charactername>' が宣言されたデータ型 '<type>' と一致しません。

変数を 1 つのデータ型で宣言していますが、互換性のないデータ型を表す型宣言文字で参照されています。たとえば、`K` を **Integer** として宣言している場合に、「`K$ = 10`」のように記述しています。

Error ID: BC30277

このエラーを解決するには

- 宣言した変数のデータ型を変更するか、または参照内の型宣言文字を変更または削除します。

参照

概念

型文字

明示的な型を含む宣言では、型文字 '<character>' を使用できません。

データ型の明示的な指定において、$ などの型文字が使用されています。

Error ID: BC30302

このエラーを解決するには

- 宣言から型文字を削除するか、または `As <Type>` 句を削除します。

参照

概念

型文字

暗黙の宣言と明示的宣言

属性を汎用にできないため、型引数は有効ではありません。

属性が型引数リストを使用して適用されています。

Visual Basic および .NET Framework は、現時点では属性とジェネリック型の組み合わせをサポートしていません。そのため、次のような制限があります。

- 属性をジェネリック型にしたり、ジェネリック型の中で宣言したりすることはできません。
- 属性がジェネリック クラスを継承したり、ジェネリック クラスが属性を継承したりすることはできません。
- 属性を適用するときには、次のような引数は指定できません。
    - ジェネリック型
    - ジェネリック型から作成された型
    - 包含型の型パラメータ
    - 包含型の型パラメータから作成された型

Error ID: BC32066

このエラーを解決するには

- 型引数を通常の引数にする場合は、**Of** キーワードを削除します。これにより、型引数リストが通常の引数リストになります。
- 型引数を型パラメータに提供する場合は、**Of** キーワードとすべての型引数を削除します。属性には型引数を指定できません。

参照

関連項目

型リスト

Attribute

概念

Visual Basic における属性

Visual Basic におけるジェネリック型

予期しない型引数

Implements 句が、実装しようとしているインターフェイス メンバの型引数を渡しています。

Implements 句には、インターフェイスおよびインターフェイスが実装しようとしているメンバのみを指定する必要があります。つまり、ジェネリック プロシージャを宣言する場合は、インターフェイス プロシージャを実装しない場合と同じように、**Of** 句と型引数を宣言の本体に指定する必要があります。

このエラーは次のようなコードで発生します。

```
Public Interface testInterface
    Sub testSub(Of t)()
End Interface
Public Class testClass
    Implements testInterface
    Public Sub testSub() Implements testInterface.testSub(Of t)()
    End Sub
End Class
```

Implements 句の前にある宣言は、アクセス修飾子またはプロシージャ修飾子が追加される可能性があることを除き、インターフェイスの定義とよく似ています。このエラーを回避するコード例を次に示します。

```
    Public Sub testSub(Of t)() Implements testInterface.testSub
```

Error ID: BC32088

このエラーを解決するには

- 型引数リストを **Implements** 句から削除します。
- インターフェイスの汎用メンバを実装する場合は、型引数リストを宣言の本体の **Implements** キーワードの前に配置します。

参照

関連項目

Implements (Visual Basic)
型リスト

概念

Implements キーワードおよび Implements ステートメント
Visual Basic におけるジェネリック型
Visual Basic におけるジェネリック プロシージャ

メソッド '<procedurename>' に対して推論された型引数には次の警告が表示されます: <warninglist>

ジェネリック プロシージャが型引数を指定しないで呼び出され、推測された型引数によって 1 つ以上の警告が生成されました。

通常、ジェネリック型を呼び出すときには、ジェネリック型に定義された各型パラメータに型引数を指定します。型引数を何も指定しなければ、コンパイラは型パラメータに渡される型を推測しようとします。推測された型があいまいな場合、またはそれらが予測不可能な結果をもたらす状況を生み出す場合、コンパイラはこの警告を生成します。

型パラメータの制約によって、それに渡すことができる型引数が制限されます。たとえば、型パラメータが、IComparable インターフェイスを実装するクラスに制限される場合があります。詳細については、「Visual Basic におけるジェネリック プロシージャ」の "制約" を参照してください。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC41006

このエラーを解決するには

- 型引数をジェネリック プロシージャに渡して、コンパイラが型を推測しなくて済むようにします。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

Visual Basic におけるジェネリック プロシージャ

メソッド '<procedurename>' に対して推論された型引数には、次のエラーが表示されます :<errorlist>

ジェネリック プロシージャが型引数を指定せずに呼び出され、型引数の推論の結果が 1 つ以上の制約違反を含んでいます。

通常、ジェネリック型を呼び出すときには、ジェネリック型に定義された各型パラメータに型引数を指定します。型引数を何も指定しなければ、コンパイラは型パラメータに渡される型を推論しようとします。推論された型が、1 つ以上の型パラメータ制約に違反していると、コンパイラはこのエラーを生成します。

型パラメータの制約は、そのパラメータに渡すことができる型引数を制限します。たとえば、IComparable インターフェイスを実装するクラスに制限するなどが可能です。詳細については、「Visual Basic におけるジェネリック プロシージャ」の "制約" を参照してください。

Error ID: BC30954

このエラーを解決するには

- ジェネリック プロシージャに型引数を指定して、コンパイラが型を推論する必要がないようにします。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型
Visual Basic におけるジェネリック プロシージャ

メソッド '<procedurename>' の型引数は、デリゲート '<delegatename>' から推論できませんでした。

代入ステートメントで **AddressOf** を使用して、ジェネリック プロシージャのアドレスをデリゲートに代入していますが、ジェネリック プロシージャに型引数が何も指定されていません。

通常、ジェネリック型を呼び出すときには、ジェネリック型に定義された各型パラメータに型引数を指定します。型引数を何も指定しなければ、コンパイラは型パラメータに渡される型を推測しようとします。コンパイラが型を推測するための十分な情報をコンテキストから得ることができない場合は、エラーが生成されます。

Error ID: BC30952

このエラーを解決するには

- **AddressOf** 式の中で、ジェネリック プロシージャに型引数を指定します。

参照

関連項目

AddressOf 演算子
型リスト

概念

Visual Basic におけるジェネリック型
Visual Basic におけるジェネリック プロシージャ

型引数は式 '<式>' に適用できません

インポート エイリアスが、そのインポート エイリアスに型引数を渡す Of 句によって定義されています。

型引数はジェネリック型で使用されるものであり、ジェネリックになり得るのはクラス、構造体、インターフェイス、プロシージャ、デリゲートのみです。名前空間やインポート エイリアスはジェネリックになりません。

Error ID: BC32058

このエラーを解決するには

- 問題の **Of** 句と型引数をインポート エイリアスから削除します。

参照

関連項目

Imports ステートメント
型リスト

概念

参照と Imports ステートメント
同じ名前を持つ複数の変数がある場合に参照を解決する
Visual Basic におけるジェネリック型

'<genericproceduresignature>' の型パラメータ '<typeparametername>' に対して型引数を推論できませんでした。

エラー メッセージ
'<genericproceduresignature>' の型パラメータ '<typeparametername>' に対して型引数を推論できませんでした。型引数を、パラメータ '<parametername>' に渡される引数から推論することができませんでした。

ジェネリック プロシージャが型引数を渡さずに呼び出されたため、コンパイラではパラメータの 1 つに渡す型を推論できません。

通常、ジェネリック プロシージャを呼び出すには、ジェネリック プロシージャに定義されたすべての型パラメータに 1 つずつ型引数を渡す必要があります。型引数を渡さないと、コンパイラは型パラメータに渡す型を推論しようとします。呼び出しのコンテキストから型パラメータについて矛盾するデータ型情報が提供される場合、型の推論は失敗します。

このエラーは次のようなコードで発生します。

```
Public Sub doSomething(Of t)(ByVal arg1 As t(), ByVal arg2 As t)
End Sub
Call doSomething(6, 42)
```

この例では、コンパイラは `arg2` に 42 という値が渡されることから `t` の型は **Integer** であると推論します。ただし、この推論が成り立つには、`arg1` が **Integer()** 型 (**Integer** の配列) である必要がありますが、`arg1` に渡される 6 という値はこの型に一致しません。

Error ID: BC32051

このエラーを解決するには

- 型引数をジェネリック プロシージャに渡して、コンパイラが型を推論しなくて済むようにします。
- 型引数の型に一致する型で通常の引数を渡します。

参照

関連項目
型リスト
概念
Visual Basic におけるジェネリック型
Visual Basic におけるジェネリック プロシージャ

'<genericproceduresignature>' の型パラメータ '<typeparametername1>' に対して型引数を推論できませんでした。

エラー メッセージ
'<genericproceduresignature>' の型パラメータ '<typeparametername1>' に対して型引数を推論できませんでした。パラメータ '<parametername1>' に渡される引数から推論する型引数が、パラメータ '<parametername2>' に渡される引数から推論する型引数と競合しています。

ジェネリック プロシージャが型引数を指定せずに呼び出されており、型の推論によって型パラメータの間で競合するデータ型が生成されています。

通常、ジェネリック プロシージャを呼び出すときには、ジェネリック プロシージャに定義された各型パラメータに型引数を指定します。型引数を何も指定しなければ、コンパイラは型パラメータに渡される型の推論を試みます。呼び出しのコンテキストから型パラメータと競合するデータ型情報が得られた場合、型の推論はエラーになります。

このエラーは次のようなコードで発生します。

```
Public Sub takeTwoValues(Of t)(ByVal x As t, ByVal y As t)
End Sub
Call takeTwoValues(4, 2.5)
```

コンパイラは最初の引数から、型パラメータ `t` に対して **Integer** を推論しますが、2 番目の引数では同じ型パラメータに対して **Double** を推論するため、`t` に渡すデータ型に関して競合が生じます。

Error ID: BC32116

このエラーを解決するには

- ジェネリック型に型引数を指定して、コンパイラが型を推論する必要がないようにします。

  ```
  Call takeTwoValues(Of Double)(4, 2.5)
  ```

  このケースでは、通常の 2 つの引数がそれぞれ異なるデータ型を持つため、呼び出し元のコードは両方のデータ型に適合する型引数を渡す必要があります。この例の場合で言えば、**Integer** は **Double** に拡大されます。

  または

- ジェネリック プロシージャの定義を変更して、通常のパラメータに対して異なる型パラメータを指定し、型の推論による競合が発生しないようにします。

  ```
  Public Sub takeTwoValues(Of t1, t2)(ByVal x As t1, ByVal y As t2)
  ```

参照

関連項目
型リスト
概念
Visual Basic におけるジェネリック型
Visual Basic におけるジェネリック プロシージャ

型パラメータ '<typeparametername>' の 'New' 制約を満たすには、型引数 '<typeargumentname>' にパブリックのパラメータなしインターフェイス コンストラクタを指定しなければなりません。

型引数に、New (Visual Basic) 制約が指定された型パラメータへの、アクセス可能なパラメータなしコンストラクタを持たない型が指定されています。

制約リストには、型パラメータに渡される型引数に対する要件を設定します。設定可能な要件の 1 つは、作成しているコードからアクセス可能なパラメータなしのコンストラクタを、型引数が公開することです。この要件を指定するには、制約リストに **New** 制約を定義します。

Error ID: BC32083

このエラーを解決するには

1. ジェネリック型の名前と、型引数の型名のスペルが正しいことを確認します。
2. アクセス可能なパラメータなしのコンストラクタを公開する型引数の型を選択します。この型パラメータにこのような型引数が指定できる場合を除き、この特定のジェネリック型を呼び出すことはできません。

参照

処理手順

方法 : ジェネリック クラスを使用する

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

型引数 '<typeargumentname>' は型パラメータ '<typeparametername>' の 'Structure' 制約を満たしていません。

ジェネリック型に指定された型引数が、その一致する型パラメータに対する値型の制約を満たしていません。

制約リストには、型パラメータに渡される型引数に対する要件を設定します。制約リストに特定のクラスやインターフェイスを何も含めない場合は、次のいずれかを指定して汎用の要件を設定できます。

- 型引数は値型であることが必要 (Structure (Visual Basic) 制約を含める)
- 型引数は参照型であることが必要 (Class (Visual Basic) 制約を含める)

同じ型パラメータに対して **Structure** と **Class** の両方を指定することはできません。また、これらを複数回指定することもできません。

Error ID: BC32105

このエラーを解決するには

- 任意の値型の型引数を選択します。

参照

処理手順

方法 : ジェネリック クラスを使用する

概念

Visual Basic におけるジェネリック型

値型と参照型

型引数 '<typeargumentname>' は型パラメータ '<typeparametername>' の 'クラス' 制約を満たしていません。

ジェネリック型に指定された型引数が、対応する型パラメータの参照型制約を満たしていません。

制約リストには、型パラメータに渡される型引数に対する要件を設定します。制約リストに特定のクラスやインターフェイスを何も含めない場合は、次のいずれかを指定して汎用の要件を設定できます。

- 型引数は値型であることが必要 (Structure (Visual Basic) 制約を含める)
- 型引数は参照型であることが必要 (Class (Visual Basic) 制約を含める)

同じ型パラメータに対して **Structure** と **Class** の両方を指定することはできません。また、これらを複数回指定することもできません。

Error ID: BC32106

このエラーを解決するには

- 任意の参照型の型引数を選択します。

参照

処理手順

方法 : ジェネリック クラスを使用する

概念

Visual Basic におけるジェネリック型

値型と参照型

型引数 '<typeargumentname>' は、制約型 '<typeparametername>' から継承したり、この型を実装したりしません。

ジェネリック型に渡される型引数が、それに対応する型パラメータの継承または実装の制約を満たしていません。

制約リストは、型パラメータに渡される型引数に対する要件を設定します。次の要件を指定できます。

- 型引数は、1 つまたは複数のインターフェイスを実装する必要があります。
- 型引数は、最大で 1 つのクラスを継承する必要があります。

上記の複数の要件を単一の型パラメータに組み合わせて指定できます。ジェネリック型に定義されたすべての型パラメータの制約を満たす型引数が渡されない限り、Visual Basic では型を作成できません。

Error ID: BC32044

このエラーを解決するには

- 型パラメータに指定されたすべてのインターフェイスを実装し、クラスが指定されていればそれを継承した型の型引数を選択します。

参照

処理手順

方法 : ジェネリック クラスを使用する

概念

Visual Basic におけるジェネリック型

型引数 '<typeargumentname>' は 'MustInherit' として宣言され、型パラメータ '<typeparametername>' の 'New' 制約を満たしていません。

ジェネリック型が型引数に **MustInherit** クラスを指定して呼び出されていますが、対応する型パラメータが **New** 制約を指定して宣言されています。

New 制約を指定した場合、対応する型引数に渡される型は、オブジェクトの作成をサポートする必要があります。しかし、抽象クラス、つまり **MustInherit** で宣言されるクラスは、そこからオブジェクトを作成できないため、コンストラクタを公開しません。

Error ID: BC32082

このエラーを解決するには

1. 型引数に使用するクラスを抽象クラスにする必要がない場合は、**MustInherit** キーワードを宣言から削除します。
2. 型引数に使用するクラスを抽象クラスにする必要があるが、ジェネリック型の作成に使用する必要がない場合は、型引数に別のクラスを渡します。
3. 対応する型パラメータが、そこに渡される型からオブジェクトを作成する必要がない場合は、宣言から **New** 制約を削除します。

参照

関連項目

New (Visual Basic)
MustInherit

概念

Visual Basic におけるジェネリック型

'型' '<typename1>' は、含んでいる型 '<typename2>' が CLS に準拠していないため、CLS 準拠として設定できません。

クラスまたはインターフェイスが `<CLSCompliant(True)>` としてマークされていますが、上層の入れ子レベルの型が `<CLSCompliant(False)>` としてマークされています (またはマークされていません)。

クラスまたはインターフェイスを共通言語仕様 (CLS) 準拠にするためには、コンテインメント階層全体を CLS に準拠させる必要があります。つまり、上層の入れ子レベルの型をすべて CLS 準拠にする必要があります。

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40030

このエラーを解決するには

- CLS 準拠にする必要がある場合は、この型を別のコンテインメント階層の中で定義します。
- この型を現在のコンテインメント階層の中に残す必要がある場合は、その定義から **CLSCompliantAttribute** を削除するか、その型を `<CLSCompliant(False)>` としてマークします。

参照

概念

CLS 準拠コードの記述

型 '<typename>' を 'For' ステートメントで使用するには、演算子 '<operator>' を定義しなければなりません。

必要な演算子がサポートされていない型のカウンタ変数が、**For** ループに指定されています。

For ループのカウンタ変数は、次のすべての演算子をサポートする任意のデータ型で宣言できます。

- 以上 (>=)
- 以下 (<=)
- 加算 (+)
- 減算 (-)

数値データ型をカウンタ変数に使う場合は、これらのすべての演算子がサポートされます。ユーザー定義のクラスまたは構造体を使う場合は、これらのすべての演算子をクラスまたは構造体に定義する必要があります。

また、**For** ステートメント内の *start* 式、*end* 式、および *step* 式のデータ型をカウンタ変数のデータ型に拡大する必要があることに注意してください。カウンタ変数がユーザー定義のクラスまたは構造体であり、*start* 式、*end* 式、または *step* 式がそれとは別の型である場合は、必要な変換を実行するために **CType** 変換演算子を定義する必要があります。

Error ID: BC33038

このエラーを解決するには

1. カウンタ変数のデータ型が正しいスペルであることを確認します。
2. ユーザー定義のクラスまたは構造体をカウンタ変数に使う場合は、すべての必要な演算子をそのクラスまたは構造体に定義します。
3. *start* 式、*end* 式、および *step* 式のデータ型によっては、その型をカウンタ変数のデータ型に変換するために 1 つ以上の **CType** 変換演算子を定義する必要があります。

参照

処理手順

方法 : 演算子を定義する
方法 : 変換演算子を定義する

関連項目

For...Next ステートメント (Visual Basic)
Operator ステートメント
CType 関数

概念

演算子プロシージャ

型 '<typename>' を '<shortcircuitoperator>' 式で使用するには、演算子 '<determinantoperator>' を定義しなければなりません

AndAlso 演算子または OrElse 演算子がクラス型または構造体型のオペランドを使用していますが、そのクラスまたは構造体に必要な演算子が定義されていません。

ショート サーキット演算子 (**AndAlso** または **OrElse**) を直接定義していないので、対応する論理演算子および決定演算子を定義する必要があります。必要な演算子は、次の表に示すとおりです。

ショートサーキット演算子	論理演算子	決定演算子
AndAlso	And 演算子 (Visual Basic)	IsFalse 演算子
OrElse	Or 演算子 (Visual Basic)	IsTrue 演算子

Visual Basic ではこれらの論理演算子および決定演算子を使って、**AndAlso** または **OrElse** に必要なショート サーキットのロジックを構築します。これを正しく行うために、**And** または **Or** の定義において、オペランドの型と戻り値の型はその包含型であることが必要です。つまり、**And** または **Or** を含んでいるクラスまたは構造体の型である必要があります。

Error ID: BC33035

このエラーを解決するには

- **AndAlso** 演算子または **OrElse** 演算子のオペランドの型に使用するクラスまたは構造体の内部に、**And** 演算子と **IsFalse** 演算子を定義するか、または **Or** 演算子と **IsTrue** 演算子を定義します。**And** または **Or** のオペランドの型には、必ずそれを定義しているクラスまたは構造体の型を使用してください。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

Visual Basic の論理演算子とビット処理演算子

型 '<typename>' が定義されていません。

ステートメントは未定義の型を参照しました。型は、**Enum**、**Structure**、**Class**、**Interface** などの宣言ステートメントで定義できます。

Error ID: BC30002

このエラーを解決するには

1. 型定義とその参照の両方で同じスペルを使用しているかどうかを確認します。
2. 参照から型定義にアクセスできるかどうかを確認します。たとえば、型が別のモジュール内にあって **Private** として宣言されている場合は、型定義を参照元のモジュールに移動するか、型を **Public** として宣言します。
3. 型の名前空間がプロジェクト内で再定義されていないか確認します。再定義されていた場合は、**Global** キーワードを使用して型名を完全修飾します。たとえば、プロジェクトで `System` という名前空間が定義されている場合、System.Object 型にアクセスするには、**Global** キーワードを使って `Global.System.Object` のように完全修飾する必要があります。
4. 型は定義されていても、型が定義されているオブジェクト ライブラリまたはタイプ ライブラリが Visual Basic に登録されていない場合は、[プロジェクト] メニューの [参照の追加] をクリックし、適切なオブジェクト ライブラリまたはタイプ ライブラリを選択します。

参照

関連項目

Enum ステートメント (Visual Basic)
Structure ステートメント
Class ステートメント (Visual Basic)
Interface ステートメント (Visual Basic)
Global
[参照の追加] ダイアログ ボックス

概念

Visual Basic における名前空間

'<typename>' 型が定義されていないか、その型を含むモジュールがデバッグ セッションに読み込まれていません。

使用できない型または定義されていない型が参照されました。

Error ID: BC30743

このエラーを解決するには

1. 型名のスペルが正しいことを確認します。
2. 現在のクラスまたはモジュール内で型を定義します。

参照

その他の技術情報

Visual Studio でのデバッグ

型 <typename> は CLS に準拠していません。

変数、プロパティ、または関数の戻り値が、CLS 準拠ではないデータ型で宣言されています。

アプリケーションを 共通言語仕様 (CLS) に準拠させるためには、CLS 準拠の型のみを使う必要があります。

次の Visual Basic データ型は CLS に準拠していません。

- SByte 型 (Visual Basic)
- UInteger データ型
- ULong データ型 (Visual Basic)
- UShort 型 (Visual Basic)

Error ID: BC40041

このエラーを解決するには

- アプリケーションを CLS に準拠させる必要がある場合は、この要素のデータ型を CLS に準拠した最も近い型に変更します。たとえば、2,147,483,647 を超える値の範囲が必要でない場合は、**UInteger** の代わりに **Integer** を使用できます。範囲を拡張する必要がある場合は、**UInteger** を **Long** で置き換えてください。
- アプリケーションを CLS に準拠させる必要がない場合は、何も変更する必要はありません。ただし、準拠しないことは頭に入れておいてください。

参照

概念

CLS 準拠コードの記述

型 '<typename>' は、異なるバージョンのアセンブリ '<assemblyname>' からインポートされています。

エラー メッセージ
型 '<typename>' は、異なるバージョンのアセンブリ '<assemblyname>' からインポートされています。同じアセンブリの異なるバージョンを、同じプロジェクトで使用することはできません。

同一アセンブリの異なるバージョンから型がインポートされたため、あいまいさが発生します。

Error ID: BC31525

このエラーを解決するには

- 同一アセンブリの複数のバージョンへの参照を削除します。

参照
概念
プロジェクト参照

型 '<typename>' は型パラメータが指定されてないため、型引数を指定することができません。

ジェネリックではない型を呼び出すときに、宣言ステートメントまたは代入ステートメントで Of 句が指定されています。

ジェネリック型は、1 つ以上の型パラメータを使って指定できるデータ型を操作するクラス、構造体、インターフェイス、プロシージャ、またはデリゲートです。ジェネリック型から型を作成するには、型引数を各型パラメータに渡します。型を作成する過程で、各型引数はそれに対応する型パラメータが現れるたびにそれと置き換えられ、コードが生成されます。

型パラメータは **Of** 句を使ってかっこの中に定義し、型引数は **Of** 句を使ってかっこの中に指定します。**Of** 句は、ジェネリック型を扱う場合にだけ使用します。

ジェネリックではない型は型パラメータを受け付けないため、このような型を呼び出すときに型引数は指定できません。

Error ID: BC32045

このエラーを解決するには

1. 宣言ステートメントまたは代入ステートメントで使用する型のスペルを確認します。
2. ジェネリックではない型を呼び出すときに **Of** 句とかっこを指定している場合は、それらを削除します。プロシージャ、デリゲート、またはクラスコンストラクタの通常の引数リストを囲むかっこは削除しないでください。

参照

処理手順
方法 : ジェネリック クラスを使用する
関連項目
型リスト
概念
Visual Basic におけるジェネリック型

型 '<typename>' にはコンストラクタがありません。

型が **Sub New()** の呼び出しをサポートしません。コンパイラまたはバイナリ ファイルが破損していることが原因の 1 つとして考えられます。

Error ID: BC30251

このエラーを解決するには

1. 型が別のプロジェクトまたは参照ファイル内にある場合は、プロジェクトまたはファイルを再インストールします。
2. 型が同じプロジェクト内にある場合は、型を含むアセンブリを再コンパイルします。
3. エラーが再発する場合は、Visual Basic コンパイラを再インストールします。
4. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

関連項目

コンストラクタとデストラクタの使用方法

その他の技術情報

製品のサポートとユーザー補助

型 '<typename>' は、型パラメータから継承することはできません。

ジェネリックな型パラメータを指定する Inherits ステートメント がクラスまたはインターフェイスに含まれています。

まだ定義されていない型を継承することはできません。継承には基本クラスのメンバを再利用する働きがあるので、基本クラスのメンバが既に定義されている必要があります。ジェネリックな型パラメータは、型引数で指定される特定の型に置き換えられるプレースホルダです。したがって、型はプレースホルダを継承できません。

Error ID: BC32055

このエラーを解決するには

- 継承する型が別の型を継承する必要がある場合は、型パラメータの代わりに特定の型を使用します。
- 基本型をジェネリックな型パラメータで表す必要がある場合は、他の型はこの型を継承できません。Inherits ステートメント を削除します。

参照

概念

Visual Basic におけるジェネリック型

その他の技術情報

Visual Basic の継承

型 '<typename>' は、その中に入れ子にされた型から継承することはできません。

クラス定義またはインターフェイス定義に含まれている Inherits ステートメント で、それ自身に入れ子になっている型が指定されています。

継承は、直線的である必要があり、循環することはできません。型は、それ自身を継承する型を継承できません。

この制限と関連して、型はまだ定義されていない型を継承できない、という制限があります。継承には基本クラスのメンバを再利用する働きがあるので、基本クラスのメンバが既に定義されている必要があります。したがって、Visual Basic は次のようなコードをコンパイルできません。

```
Public Class outerClass
    ' The following statement is INVALID because innerClass is not defined.
    Inherits innerClass
    Public Class innerClass
        Public Sub doSomething()
        End Sub
    End Class
End Class
```

Error ID: BC30908

このエラーを解決するには

- 継承する型 (入れ子の外側の型) が内部の型を継承する必要がある場合は、内部の型を外部の型の外に移動します。
- 内部の型を外部の型に入れ子にする必要がある場合は、外部の型は内部の型を継承できません。Inherits ステートメント を削除します。

参照

その他の技術情報

Visual Basic の継承

型 '<typename>' のコンテナ '<containername>' が 'Public' と宣言されていないため、この型は属性では使用できません。

この属性の定義されているクラスまたはモジュールが、**Public** 修飾子で宣言されていません。アクセス修飾子を指定していないクラスおよびモジュールは、既定で **Friend** として宣言されます。

Error ID: BC31517

このエラーを解決するには

- この属性が定義されているクラスまたはモジュールに **Public** 修飾子を追加します。

参照

関連項目

Public (Visual Basic)

型 '<typename>' は 'Public' と宣言されていないため、属性で使用することはできません。

属性はパブリックである必要があります。

Error ID: BC31516

このエラーを解決するには

- 属性クラスに **Public** 修飾子を追加します。

参照

関連項目

Public (Visual Basic)

型 '<typename>' と、'<filename>' で宣言された部分的な型 '<typename>' がコンテナ '<containername>' 競合しますが、そのうち 1 つが部分的な宣言であるためマージされました。

同じコンテナ型の中にクラスまたは構造体の定義が複数あり、その 2 つ以上の定義が **Partial** としてマークされていません。

クラスまたは構造体の複数の定義のうち、少なくとも 1 つには Partial (Visual Basic) キーワードを使用する必要があります。しかし、すべての部分定義でこのキーワードを使用することをお勧めします。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40047

このエラーを解決するには

- クラスまたは構造体のすべての部分定義について Partial (Visual Basic) キーワードを使用します。

型 '<typename>' と部分的な型 '<typename>' がコンテナ '<containername>' で競合しますが、そのうち 1 つが部分的な宣言であるためマージされました。

同じコンテナ型の中にクラスまたは構造体の定義が複数あり、その 2 つ以上の定義が **Partial** としてマークされていません。

クラスまたは構造体の複数の定義のうち、少なくとも 1 つには Partial (Visual Basic) キーワードを使用する必要があります。しかし、すべての部分定義でこのキーワードを使用することをお勧めします。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40046

このエラーを解決するには

- クラスまたは構造体のすべての部分定義について Partial (Visual Basic) キーワードを使用します。

参照

関連項目

Partial (Visual Basic)

型 '<typename>' は、直接的または間接的にそれ自体から継承しているため、サポートされていません。

クラスまたはインターフェイスが自分自身を継承しているか、自分自身から派生した別のクラスまたはインターフェイスを継承しています。

Visual Basic は循環継承をサポートしません。

Error ID: BC30916

このエラーを解決するには

- 継承構造を変更して、他のクラスまたはインターフェイスを継承していない基本クラスまたはインターフェイスを基盤にするようにします。

参照

概念

継承の基本

アセンブリ '<assemblyname1>' の型 '<typename>' は、アセンブリ '<assemblyname2>' に転送されました。

エラー メッセージ
アセンブリ '<assemblyname1>' の型 '<typename>' は、アセンブリ '<assemblyname2>' に転送されました。プロジェクトに '<assemblyname2>' への参照が見つからないか、またはアセンブリ '<assemblyname2>' に型 '<typename>' が見つかりません。

アセンブリのソース コード内の式が、別のアセンブリに転送された型を参照していますが、その型が転送先のアセンブリに見つかりません。

型の転送とは、クラス、構造体、インターフェイス、デリゲート、または列挙値の定義を、最初に定義したアセンブリ以外のアセンブリに再割り当てすることを指します。型の転送は、多くの場合コードのリファクタリングと併せて使用されます。これによって、アセンブリを複数のアセンブリに分割したり、コードを別のアセンブリに移動したりします。

転送された型は、元のアセンブリでも一時的に使用できますが、コードのリファクタリングによって型が元のアセンブリから削除されたときに、おそらくは未定義になります。

Error ID: BC31424

このエラーを解決するには

- 型が転送先のアセンブリに存在することを確認します。
- プロジェクトに転送先のアセンブリへの参照が含まれていることを確認します。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
関連項目
Type Forwarding
TypeForwardedToAttribute
概念
プロジェクト参照

''TryCast' オペランドはクラスの制約がある型パラメータでなければなりませんが、'<typeparametername>' にはクラスの制約がありません。

TryCast 演算子が、参照型であることが保証されない型パラメータ オペランドと共に使用されています。

TryCast は、クラス、インターフェイスなどの参照型だけを操作します。引数として型パラメータを **TryCast** に渡すには、型パラメータが常に参照型であるように制約する必要があります。これを行うには、以下の項目の 1 つ以上を型パラメータの制約リストに指定します。

- 1 つ以上のインターフェイス名 (型引数はこれらのすべてのインターフェイスを実装する必要があります)
- 1 つのクラス名 (型引数はこのクラスを継承する必要があります)
- New (Visual Basic) 制約 (型引数は、作成元のコードがアクセスできるパラメータなしのコンストラクタを公開する必要があります。したがって、型引数はクラスである必要があります)
- Class (Visual Basic) 制約 (型引数は参照型である必要があります)

Error ID: BC30793

このエラーを解決するには

- この型パラメータを **TryCast** に渡す必要がある場合は、前掲の一覧に示した制約の 1 つ以上を使って型パラメータを制約します。
- 参照型しか受け付けない型パラメータにできない場合は、**TryCast** でその型パラメータを使うことはできません。代わりに、CType 関数 を使用できる場合があります。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型
値型と参照型
拡大変換と縮小変換
暗黙の型変換と明示的な型変換

'TryCast' オペランドは参照型でなければなりませんが、'<typename>' は値型です。

TryCast 演算子の少なくとも 1 つの引数に、値型が指定されています。

TryCast は、2 つの引数の間の継承または実装の関係を調べます。そのため、参照型以外を引数に指定できません。詳細については、「値型と参照型」を参照してください。

Error ID: BC30792

このエラーを解決するには

- **DirectCast** または **CType** を使用して変換を実行します。この 2 つには値型を指定できます。

参照

関連項目

TryCast
DirectCast
CType 関数

概念

値型と参照型

'Try' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Try ステートメントはソース コードでのみ使用できます。

Error ID: BC30717

このエラーを解決するには

- デバッグ コードから **Try** ステートメントを削除します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

その他の技術情報

Visual Studio でのデバッグ

Try には少なくとも 1 つの 'Catch' または 1 つの 'Finally' を指定しなければなりません。

Try ブロックには、**End Try** ステートメントの前に、**Finally** ブロックか、少なくとも 1 つの **Catch** ブロックを指定する必要があります。

Error ID: BC30030

このエラーを解決するには

- **Catch** ブロックまたは **Finally** ブロックを **Try** ブロック内に追加します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

'Try' の終わりには、対応する 'End Try' を指定しなければなりません。

Try は **Try** ブロックを開始するために使用します。したがって、ブロックの先頭にだけ記述でき、対応する **End Try** ステートメントでブロックの終了を示す必要があります。余分な **Try** ステートメントがあります。または、**Try** ブロックが **End Try** で終了していません。

Error ID: BC30384

このエラーを解決するには

- 余分な **Try** ステートメントを探し、削除します。または、対応する **End Try** を指定してブロックを終了します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic の構造化例外処理の概要

'<generictypename>' の型引数が多すぎます。

ジェネリック クラス、構造体、インターフェイス、またはデリゲートが、型パラメータよりも多くの型引数を指定して呼び出されています。

ジェネリック型を呼び出すときは、そのジェネリック型で定義されているそれぞれの型パラメータに 1 つの型引数を指定する必要があります。

Error ID: BC32043

このエラーを解決するには

- 型引数リストから型引数を削除して、呼び出そうとするジェネリック型のそれぞれの型パラメータに対して型引数が 1 つになるようにします。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

ファイル '<filename>' への参照を生成できません。COM DLL を参照するには TLBIMP ユーティリティを使ってください : <error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、COM+ DLL ファイルの検索エラーまたは検証エラーを報告しています。

Error ID: BC30142

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べます。詳細とアドバイスについては、Al.exe ツールのエラーと警告を参照してください。
2. 目的の参照先が COM+ DLL ではなく COM DLL の場合は、タイプ ライブラリ インポータ (Tlbimp.exe) を使用して参照を作成します。
3. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

関連項目

アセンブリ リンカ (Al.exe)
Al.exe ツールのエラーと警告
タイプ ライブラリ インポータ (Tlbimp.exe)

その他の技術情報

製品のサポートとユーザー補助

必要なファイル '<filename>' が見つかりません。

Microsoft Visual Studio 2005 で必要とされるファイルがないか、破損しています。

Error ID: BC30655

このエラーを解決するには

- Microsoft Visual Studio 2005 を再インストールします。

参照

その他の技術情報

製品のサポート

DLL ファイル '<filename>' にエントリ ポイント '<name>' が見つかりません : <error>

指定したダイナミック リンク ライブラリ (DLL: dynamic link library) でプロシージャ名が見つかりませんでした。

Error ID: BC31014

このエラーを解決するには

- プロシージャの名前のスペルが正しいことと、DLL 内の関数と大文字と小文字の区別が一致していることを確認します。

参照

処理手順

チュートリアル : Windows API の呼び出し

式を評価できません。

式を評価できませんでした。

Error ID: BC30707

このエラーを解決するには

- 式の構文を確認します。

参照

その他の技術情報

Visual Studio でのデバッグ

アセンブリを作成できません : <error message>

Visual Basic コンパイラは、マニフェストを含むアセンブリを生成するためにアセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出しましたが、アセンブリを生成する出力段階でリンカからエラーが報告されました。

Error ID: BC30145

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べてください。詳細とアドバイスについては、「Al.exe ツールのエラーと警告」を参照してください。
2. アセンブリ リンカ (Al.exe) または 厳密名ツール (Sn.exe) を使用して、手動でアセンブリに署名します。
3. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト製品サポート サービスにご連絡ください。

アセンブリを手動で署名するには

1. 厳密名ツール (Sn.exe) を使用して、公開キーと秘密キーの一対のファイルを作成します。

   このファイルは、.snk の拡張子を持ちます。

2. プロジェクトから、エラーが発生している COM 参照を削除します。
3. Windows の [スタート] ボタンをクリックし、[すべてのプログラム] をポイントします。次に [Microsoft Visual Studio 2005] をポイントし、[Visual Studio Tools] をポイントして、[Visual Studio 2005 コマンド プロンプト] をクリックします。
4. アセンブリ ラッパーを格納するディレクトリに移動します。
5. 次のコードを入力します。

   ```
   tlbimp <path to COM reference file> /out:<output assembly name> /keyfile:<path to .snk
   file>
   ```

   コードの一例として、次のように入力することができます。

   ```
   tlbimp c:\windows\system32\msi.dll /out:Interop.WindowsInstaller.dll /keyfile:"c:\docu
   ments and settings\mykey.snk"
   ```

   パスやファイルに空白が含まれている場合には、二重引用符 (") を使用します。

6. Visual Studio で、作成したファイルに .NET アセンブリへの参照を追加します。

参照

処理手順

方法 : 公開キーと秘密キーのキー ペアを作成する

関連項目

アセンブリ リンカ (Al.exe)

Al.exe ツールのエラーと警告

厳密名ツール (Sn.exe)

その他の技術情報

製品のサポートとユーザー補助

リソース ファイル '<error message>' を埋め込めません : <error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、ネイティブな COM+ リソース ファイルをアセンブリに直接埋め込むときにエラーが発生したことを報告しています。

Error ID: BC30143

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べてください。詳細とアドバイスについては、Al.exe ツールのエラーと警告を参照してください。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト製品サポート サービスにご連絡ください。

参照

関連項目

アセンブリ リンカ (Al.exe)

Al.exe ツールのエラーと警告

その他の技術情報

製品のサポートとユーザー補助

XML ドキュメント ファイル '<名前> を作成できません: <メッセージ>

エラーのため XML ドキュメント ファイルを作成できません。

Error ID: BC42311

このエラーでは、以下のメッセージが表示されることがあります。

- XML コメント ブロックを XML ドキュメント コメントのアプリケーションをサポートする言語要素と関連付けできません。.
- XML コメント タグ 'returns' は、'WriteOnly' プロパティに許可されていません。.
- XML コメントを partial の <type> に対して複数回適用することはできません。.
- XML コメント タグ 'returns' は、'declare sub' 言語要素に許可されていません.
- XML ドキュメント解析エラー : 開始タグ '<tag>' に対応する終了タグがありません.
- XML コメントの型パラメータ <parameter> が対応する <keyword> ステートメント上の型パラメータと一致しません.
- XML コメントの型パラメータには 'name' 属性を指定しなければなりません。.
- XML コメントの例外には 'cref' 属性を指定しなければなりません.

このエラーを解決するには

- 表示されたメッセージをよく読み、推奨される作業を行います。

参照

処理手順

方法 : Visual Basic で XML ドキュメントを作成する

パス '<filename>' に一時ファイル を作成できません : <error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、ファイルの作成エラー、またはインメモリ リソースのファイルへの書き込みエラーを報告しています。設定に問題がある可能性があります。

Error ID: BC30138

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べます。詳細とアドバイスについては、Al.exe ツールのエラーと警告を参照してください。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

関連項目

アセンブリ リンカ (Al.exe)

Al.exe ツールのエラーと警告

その他の技術情報

製品のサポートとユーザー補助

キー ファイル '<filename>' から厳密な名前のアセンブリを作成できません : <error>

指定されたキー ファイルから、厳密な名前付きアセンブリを作成できませんでした。

Error ID: BC31026

このエラーを解決するには

- 正しいキー ファイルが指定されていて、ファイルが別のアプリケーションによってロックされていないことを確認します。

参照

関連項目

厳密名ツール (Sn.exe)

アセンブリ リンカ オブジェクトを作成できません : <error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、作業オブジェクトの作成エラーを報告しました。メタデータ ファイルへの参照の生成に問題が発生したか、またはインポート ファイルの呼び出しに失敗した可能性があります。

Error ID: BC30141

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べてください。詳細とアドバイスについては、Al.exe ツールのエラーと警告を参照してください。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト製品サポート サービスにご連絡ください。

参照

関連項目

アセンブリ リンカ (Al.exe)

Al.exe ツールのエラーと警告

その他の技術情報

製品のサポートとユーザー補助

.NET ランタイム インターフェイスを作成できません : <error>

インターフェイスを作成するときに内部コンパイラ エラーが発生したか、インターフェイスに対するメソッド呼び出しに失敗しました。

Error ID: BC31024

このエラーを解決するには

1. 作業内容を保存し、Visual Studio を再起動します。
2. エラーが再発する場合は、Visual Studio を再インストールします。
3. 再インストール後もエラーが発生する場合は、マイクロソフト プロダクト サポート サービスまでご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

セキュリティ属性を '<name>' に適用できません : <error>

セキュリティ属性を適用するときにエラーが発生しました。セキュリティ属性の使い方が間違っている可能性がありますが、コンパイラに問題がある場合もあります。

Error ID: BC31520

このエラーを解決するには

1. 属性の構文と引数を確認します。
2. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
3. エラーが再発する場合は、作業内容を保存し、Visual Studio を再起動します。
4. エラーが再発する場合は、Visual Basic を再インストールします。
5. 再インストール後もエラーが発生する場合は、マイクロソフト プロダクト サポート サービスまでご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

メンバにアクセスできません。

指定したメンバにアクセスできませんでした。

Error ID: BC30749

このエラーを解決するには

1. メンバが定義されていることと、メンバ名のスペルが正しいことを確認します。
2. モジュール内で宣言されているいずれかのメンバにアクセスしてみます。場合によっては、メンバが宣言されているモジュールが読み込まれていないために、デバッグ環境でメンバを特定できないことがあります。

参照

その他の技術情報

Visual Studio でのデバッグ

'Private' と宣言された型は、別の型の内部になければなりません。

別の型に含まれない型に対して **Private** 修飾子が使用されました。

Error ID: BC31089

このエラーを解決するには

1. **Friend** などの制限の緩いアクセス修飾子を使用します。
2. 別の型の内部で型を宣言します。

参照

関連項目

Private (Visual Basic)

このコンテキストでタイプを使用することはできません。

#If ディレクティブの中で型を参照しようとしました。

Error ID: BC30458

このエラーを解決するには

- 参照を削除します。

参照

関連項目

#If...Then...#Else ディレクティブ

その他の技術情報

条件付きコンパイル (Visual Basic)

'TypeOf ... Is' の左には参照型を持つオペランドが必要ですが、このオペランドの型は '<type>' です。

TypeOf...Is 式は、オブジェクト変数の実行時の型の互換性を確認します。値型に対して互換性を確認することはできません。

Error ID: BC30021

このエラーを解決するには

- **Option Strict** が **Off** の場合、**TypeName** 関数または **VarType** 関数を使用して、変数のデータ型に関する情報を取得します。
- **Option Strict** が **On** の場合、変数のデータ型は変数宣言によって決まります。

参照

関連項目

TypeName 関数 (Visual Basic)
VarType 関数 (Visual Basic)
Option Strict ステートメント

概念

Visual Basic における比較演算子

'Type' ステートメントは現在サポートされていません。'Structure' ステートメントを使用してください。

Type キーワードを使ってユーザー定義型を定義することはできません。

Error ID: BC30802

このエラーを解決するには

- **Structure** ステートメントを使って新しい型を定義します。

参照

関連項目

Structure ステートメント

その他の技術情報

構造体 : 独自のデータ型

型パラメータまたは型パラメータで構築された型は、属性の引数に許可されていません。

属性を適用するために、型パラメータである引数または型パラメータを使って作成された引数が使用されています。

Visual Basic および .NET Framework は、現時点では属性とジェネリック型の組み合わせをサポートしていません。したがって、次のような制限があります。

- 属性をジェネリック型にすることや、ジェネリック型の内部で宣言することはできません。
- 属性がジェネリック クラスを継承することも、ジェネリック クラスが属性を継承することもできません。
- 属性を適用するときには、次のような引数は指定できません。
    - ジェネリック型
    - ジェネリック型から作成された型
    - 包含型の型パラメータ
    - 包含型の型パラメータから作成された型

Error ID: BC32079

このエラーを解決するには

- 属性に渡す引数を再構築して、型パラメータまたは型パラメータから作成した型が引数に含まれないようにします。

参照

関連項目

型リスト

Attribute

概念

Visual Basic における属性

Visual Basic におけるジェネリック型

型パラメータは修飾子として使用できません。

プログラミング要素を修飾する文字列に型パラメータが含まれています。

型パラメータは、ジェネリック型を作成するときに渡す型に関する要件を表します。型パラメータは、特定の定義済みの型を表しません。修飾文字列には、コンパイル時に定義されている要素だけを含める必要があります。

このエラーは次のようなステートメントで発生します。

```
Public Function checkText(Of c As System.Windows.Forms.Control) _
    (ByVal badText As String) As Boolean
    Dim saveText As c.Text
    ' Insert code to look for badText within saveText.
End Function
```

Error ID: BC32098

このエラーを解決するには

1. 型パラメータを修飾文字列から削除するか、定義済みの型に置き換えます。
2. 修飾するプログラミング要素を見つけるために、構築型を使用する必要がある場合は、そのためのプログラム ロジックを追加する必要があります。

参照

処理手順

方法 : 宣言された要素名を修飾する

関連項目

型リスト

概念

同じ名前を持つ複数の変数がある場合に参照を解決する

Visual Basic におけるジェネリック型

型パラメータはこの宣言に指定できません。

プログラミング要素が型パラメータ リストを使って宣言されていますが、そのプログラミング要素をジェネリック型にするのは不適切です。

ジェネリック型にするのが不適切なプログラミング要素として、プロパティ、演算子、イベント、および構造体が挙げられます。このような要素を型パラメータ リストを使って宣言すると、このエラーが発生します。

Error ID: BC32065

このエラーを解決するには

- **Of** キーワードと型パラメータを宣言から削除します。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

'Structure' 制約が指定された型パラメータを制約として使用することはできません。

Structure 制約が指定された型パラメータが、別の型パラメータの制約として使用されています。

Structure 制約では、型パラメータに渡される型引数が値型であることが要求されます。しかし、値型を実装または継承することはできないため、型パラメータを制約として使用するには他の型パラメータにそれを実装するか、それを継承する必要がありますが、意味がありません。

これが意味を持つのは、両方の型引数の型がまったく同じである場合だけです。そのようなケースでは、ジェネリック型に型パラメータは 1 つしか必要ありません。

このエラーは次のようなステートメントで発生します。

```
Class c1(Of t1 As Structure, t2 As t1)
```

`t1` に渡される型は値型であることが必要なので、`t2` に渡された型は、`t1` に渡された型を実装または継承できません。

Error ID: BC32114

このエラーを解決するには

- **Structure** に制限された型パラメータを、他の型パラメータの制約リストから削除します。
- 両方の型パラメータに同じ値型が必要な場合は、一方の型パラメータにだけジェネリック型を定義します。

参照

関連項目

型リスト

Structure (Visual Basic)

概念

Visual Basic におけるジェネリック型

値型と参照型

型パラメータは 'Implements' 句では許可されていません。

ジェネリック型における **Implements** 句が、実装するメンバとして型パラメータを指定しています。

Implements 句には、インターフェイスとメンバを指定する必要があります。この句では、型パラメータをインターフェイスに渡すことはできますが、型パラメータをメンバに渡すことや、型パラメータをメンバの名前として使うことはできません。

このエラーは次のようなステートメントで発生します。

```
Class c1(Of t)
    Implements i1(Of t)
    Public Sub doSomething() Implements t
End Class
```

Error ID: BC32056

このエラーを解決するには

- **Implements** キーワードの後に、インターフェイス名とインターフェイスの実在のメンバを指定します。適切であれば、型パラメータをインターフェイスに渡すことができます。

  ```
      Public Sub doSomething() Implements i1(Of t).doSomething
  ```

参照

関連項目

Implements (Visual Basic)
型リスト

概念

Implements キーワードおよび Implements ステートメント
Visual Basic におけるジェネリック型

パラメータ名 '<typeparametername1>' は、'<partialtypename>' のその他の partial 型の 1 つで宣言された、対応する型パラメータの名前 '<typeparametername2>' と一致しません。

ジェネリック クラスまたはジェネリックな構造体が、複数の部分宣言で矛盾した型パラメータを指定して定義されています。

クラスまたは構造体を複数の部分宣言に分割して定義した場合、コンパイラはその型を、それぞれの部分宣言のすべての結合体として扱います。これはメンバだけでなく、実装、継承、およびアクセス レベルに対しても同じです。

ジェネリック クラスまたはジェネリックな構造体の定義に含まれる型パラメータに、複数の名前を指定することはできません。

Error ID: BC30931

このエラーを解決するには

- 型パラメータの名前を決め、すべての部分宣言で同じ名前を使用します。

参照

関連項目

Partial (Visual Basic)
Class ステートメント (Visual Basic)
Structure ステートメント
型リスト

概念

クラス : オブジェクトの設計図
Visual Basic におけるジェネリック型

その他の技術情報

構造体 : 独自のデータ型

型パラメータを、定義する関数と同じ名前に設定することはできません。

ジェネリック型の型パラメータが、ジェネリック型と同じ名前で宣言されています。

ジェネリック型の型パラメータ リストでは、それぞれの型パラメータを、次に示す名前とは異なる名前にする必要があります。

- 同じ型パラメータ リストに含まれている他のすべての型パラメータ。
- このジェネリック型を包含するジェネリック型の型パラメータ リストに含まれているすべての型パラメータ。
- ジェネリック型そのものの名前。

Error ID: BC32090

このエラーを解決するには

- 型パラメータの名前を、上記のリストに示したどの名前とも異なるものに変更します。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

型パラメータは名前 '<typeparametername>' で既に宣言されています。'

ジェネリック型の型パラメータが、同じジェネリック型の他の型パラメータと同じ名前で宣言されています。

ジェネリック型の型パラメータ リストでは、それぞれの型パラメータを、次に示す名前とは異なる名前にする必要があります。

- 同じ型パラメータ リストに含まれている他のすべての型パラメータ。
- このジェネリック型を包含するジェネリック型の型パラメータ リストに含まれているすべての型パラメータ。
- ジェネリック型そのものの名前。

Error ID: BC32049

このエラーを解決するには

- 型パラメータの名前を、上記のリストに示したどの名前とも異なるものに変更します。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

型パラメータ '<typeparametername1>' は、型パラメータ '<typeparametername2>' の 'New' 制約を満たすために 'New' 制約または 'Structure' 制約が必要です。

New 制約を満たさない型パラメータを渡すジェネリック型が、ステートメントで作成されています。

New 制約は、型パラメータに渡される型引数が、オブジェクトを作成するコードからアクセスできるパラメータなしのコンストラクタを公開する必要があることを意味します。値型には常にパラメータなしのコンストラクタがありますが、参照型にはパラメータなしのコンストラクタがあるとは限りません。したがって、**Structure** 制約は、**New** 制約を満たしますが、**Class** 制約、つまりクラス名またはインターフェイス名は **New** 制約を満たしません。

このエラーは次のようなコードで発生します。

```
Public Class c1(Of t As New)
End Class
Public Class c2(Of u)
    Public testObject As New c1(Of u)
End Class
```

クラス `c2` がオブジェクトを `c1` から作成するときに、型パラメータ `u` は型パラメータ `t` の **New** 制約を満たしません。

Error ID: BC32084

このエラーを解決するには

- ジェネリック型に渡す型パラメータが **Structure** 制約または **New** 制約を満たす場合に、適切な制約が定義に追加されます。

  ```
  Public Class c2(Of u As Structure)
  ```

- 型パラメータが **Structure** 制約または **New** 制約を満たすことができない場合、型パラメータはジェネリック型に渡されません。他のものを型引数として渡す必要があります。

参照

関連項目

New (Visual Basic)
Structure (Visual Basic)
Class (Visual Basic)

概念

Visual Basic におけるジェネリック型
値型と参照型

型パラメータ '<typeparametername>' は、それ自体に制約されることはできません: '<errormessage>'

型パラメータの制約リストにその型パラメータ自身が含まれています。

型パラメータの制約リストには、任意の数のインターフェイスと最高 1 つのクラスを指定できます。型パラメータに渡される型引数は、指定のすべてのインターフェイスを実装し、指定のクラスを継承する必要があります。コンパイラでは、制約リストがコード中に現れた時点で、インターフェイスとクラスが既に定義されている必要があります。型パラメータは、ジェネリック型を作成するコードによって適切な引数が渡されて置き換えられるまで、定義済みの型とは見なされません。

Error ID: BC32113

このエラーを解決するには

1. 型パラメータと制約リスト内の制約のスペルを両方とも確認します。
2. スペルに誤りがない場合は、型パラメータの名前を制約リストから削除します。型パラメータがそれ自身を制約することはできません。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

型パラメータ '<typeparametername>' には、クラスの制約を 1 つだけ指定できます。

制約リストに複数のクラスが含まれています。

型パラメータの制約リストは、任意の数のインターフェイスを受け付けますが、クラスは多くても 1 つだけ受け付けます。型パラメータに渡される型引数は、このクラスを継承する必要があり、Visual Basic では複数の継承はサポートされません。

Error ID: BC32047

このエラーを解決するには

- 1 つのクラスを選び出し、それだけを制約リストに指定します。
- この制約リストに指定できないクラスを扱うために、新しい型パラメータを定義することもできます。

参照

概念

Visual Basic におけるジェネリック型

型パラメータ '<typeparametername>' には、それを囲む型の型パラメータと同じ名前が付けられています。

ジェネリック型の型パラメータが、それを包含するジェネリック型の型パラメータと同じ名前で宣言されています。

ジェネリック型の型パラメータ リストでは、それぞれの型パラメータを、次に示す名前とは異なる名前にする必要があります。

- 同じ型パラメータ リストに含まれている他のすべての型パラメータ。
- このジェネリック型を包含するジェネリック型の型パラメータ リストに含まれているすべての型パラメータ。
- ジェネリック型そのものの名前。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40048

このエラーを解決するには

- 型パラメータの名前を、上記のリストに示したどの名前とも異なるものに変更します。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

<genericprocedurename>' の型パラメータ '<typeparametername>' を推論できません。

型引数リストを指定せずにジェネリック プロシージャを呼び出しており、いずれかの型引数についての型推論が失敗しました。

ジェネリック プロシージャを呼び出すときには、通常は、そのプロシージャで定義されている個々の型パラメータに対して型引数を指定します。しかし、別の方法として、型引数リストをすべて省略することもできます。その場合は、コンパイラが個々の型引数の型を呼び出しのコンテキストに基づいて推論しようとします。詳細については、「Visual Basic におけるジェネリック プロシージャ」の「型の推定」を参照してください。

型推論が失敗する一因として考えられるのは、型パラメータと呼び出し元の型とのランクの不一致です。次に例を示します。

```
Public Sub displayLargest(Of t As IComparable)(ByVal arg() As t)
    ' Insert code to find and display the largest element of arg().
End Sub
Public Sub callGenericSub()
    Dim testValue As Integer
    findLargest(testValue)
    Dim testMatrix(,) As Integer
    findLargest(testMatrix)
End Sub
```

このコードでは、`findLargest` に対する 2 つの呼び出しがどちらもこのエラーを引き起こします。型パラメータ `t` は 1 次元の配列を要求しているのに、コンパイラがこの 2 つの呼び出しから推論する型引数はスカラ (`testValue`) と 2 次元配列 (`testMatrix`) であるからです。

Error ID: BC32050

このエラーを解決するには

- 通常の引数の型を、型推論の結果がジェネリック プロシージャで宣言されている型パラメータと一致するように指定します。

  または

- ジェネリック プロシージャを呼び出すときに完全な型引数リストを指定して、型推論が不要になるようにします。

参照

関連項目

型リスト

概念

Visual Basic におけるジェネリック型

Visual Basic におけるジェネリック プロシージャ

型または 'New' が必要です。

ジェネリック型の宣言内の型パラメータに **As** キーワードを使って制約リストが指定されていますが、このリストには有効な制約が指定されていません。

型パラメータに対する制約は、有効なクラスまたはインターフェイス、つまりキーワード **Class**、**Structure**、または **New** のいずれかである必要があります。無効な制約が指定されていたり、制約が一切指定されていない場合、コンパイラはこのエラーを生成します。

Error ID: BC32092

このエラーを解決するには

1. 型パラメータをどのように制約するかを確認し、適切な制約を制約リストに指定します。
2. クラスまたはインターフェイスによって型パラメータを制約する場合、制約のスペルが正しいことを確認します。
3. 複数の制約を 1 つの型パラメータに指定するには、制約をコンマで区切り、制約リストを中かっこ ({ }) で囲む必要があります。

参照

関連項目

Class (Visual Basic)
Structure (Visual Basic)
New (Visual Basic)
型リスト

概念

Visual Basic におけるジェネリック型

パラメータ '<parametername>' の型は CLS に準拠していません。

プロシージャが `<CLSCompliant(True)>` としてマークされていますが、宣言しているパラメータの型が `<CLSCompliant(False)>` としてマークされているか、マークされていないか、非準拠の型であるため不適切です。

プロシージャを共通言語仕様 (CLS) 準拠にするためには、CLS 準拠の型のみを使用する必要があります。このことは、パラメータの型、戻り値の型、およびすべてのローカル変数の型に当てはまります。

次の Visual Basic データ型は CLS に準拠していません。

- SByte 型 (Visual Basic)
- UInteger データ型
- ULong データ型 (Visual Basic)
- UShort 型 (Visual Basic)

CLSCompliantAttribute をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40028

このエラーを解決するには

- プロシージャがこの特定の型のパラメータを取る必要がある場合は、**CLSCompliantAttribute** を削除します。このプロシージャは CLS 準拠になりません。
- プロシージャを CLS 準拠にする必要がある場合は、このパラメータの型を、最も近い CLS 準拠型に変更します。たとえば、2,147,483,647 を超える値の範囲が必要でない場合は、**UInteger** の代わりに **Integer** を使用できます。範囲を拡張する必要がある場合は、**UInteger** を **Long** で置き換えてください。
- オートメーションまたは COM オブジェクトとやり取りする場合は、一部の型に .NET Framework とはデータ幅が異なるものがあることに注意してください。たとえば **int** は、他の環境では多くの場合 16 ビットです。そのようなコンポーネントから 16 ビットの整数を受け取る場合は、Visual Basic のマネージ コードで、**Integer** 型ではなく **Short** 型として宣言してください。

参照

概念

CLS 準拠コードの記述

省略可能なパラメータ <parametername> に対する省略可能な値の型は、CLS に準拠していません

プロシージャが `<CLSCompliant(True)>` でマーク付けされていますが、Optional (Visual Basic) パラメータが非準拠型の既定値で宣言されています。

プロシージャを 共通言語仕様 (CLS) に準拠させるためには、CLS 準拠の型のみを使う必要があります。これはパラメータの型、戻り値の型、およびすべてのローカル変数の型に対して言えることです。また、省略可能なパラメータの既定値にも適用されます。

次の Visual Basic データ型は CLS に準拠していません。

- SByte 型 (Visual Basic)
- UInteger データ型
- ULong データ型 (Visual Basic)
- UShort 型 (Visual Basic)

CLSCompliantAttribute 属性をプログラミング要素に適用するときは、属性の *isCompliant* パラメータを **True** または **False** に設定して準拠または非準拠を示します。このパラメータの既定値はありません。値を指定する必要があります。

CLSCompliantAttribute を要素に適用しなかった場合は、非準拠と見なされます。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40042

このエラーを解決するには

- 省略可能なパラメータがこの特定の型の既定値を持つ必要がある場合は、**CLSCompliantAttribute** を削除します。このプロシージャを CLS に準拠させることはできません。
- プロシージャを CLS に準拠させる必要がある場合は、この既定値の型を CLS に準拠した最も近い型に変更します。たとえば、2,147,483,647 を超える値の範囲が必要でない場合は、**UInteger** の代わりに **Integer** を使用できます。範囲を拡張する必要がある場合は、**UInteger** を **Long** で置き換えてください。
- オートメーションまたは COM オブジェクトとやり取りする場合は、一部の型に .NET Framework とはデータ幅が異なるものがあることに注意してください。たとえば、**int** は他の環境では 16 ビットです。そのようなコンポーネントから 16 ビットの整数を受け取る場合は、Visual Basic のマネージ コードで、**Integer** 型ではなく **Short** 型で宣言してください。

参照

概念

CLS 準拠コードの記述

メンバ '<membername>' の型は CLS に準拠していません。

このメンバに指定されたデータ型が、共通言語仕様 (CLS) のデータ型ではありません。.NET Framework および Visual Basic ではこのデータ型はサポートされているので、これは使用しているコンポーネントではエラーになりません。ただし、CLS に厳密に準拠するコードで書かれた別のコンポーネントでは、このデータ型がサポートされない可能性があります。そのようなコンポーネントは、使用しているコンポーネントと正常にやり取りできない場合があります。

次の Visual Basic データ型は CLS に準拠していません。

- SByte 型 (Visual Basic)
- UInteger データ型
- ULong データ型 (Visual Basic)
- UShort 型 (Visual Basic)

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40025

このエラーを解決するには

- コンポーネントが他の .NET Framework コンポーネントだけとやり取りする場合、または他のコンポーネントとのやり取りがない場合、コードを変更する必要はありません。
- .NET Framework 向けに書かれていないコンポーネントとやり取りする場合、リフレクションを使用するかドキュメントを参照して、このデータ型がそのコンポーネントでサポートされているかどうかを確認できる可能性があります。サポートされている場合は、コードを変更する必要はありません。
- このデータ型をサポートしないコンポーネントとやり取りする場合は、最も近い CLS 準拠型にデータ型を置き換える必要があります。たとえば、2,147,483,647 を超える値の範囲が必要でない場合は、**UInteger** の代わりに **Integer** を使用できます。拡張範囲が必要な場合は、**UInteger** を **Long** で置き換えることができます。
- オートメーション オブジェクトまたは COM オブジェクトとやり取りする場合は、一部のデータ型のデータ幅が .NET Framework では異なることに注意してください。たとえば、**uint** は他の環境では 16 ビットです。そのようなコンポーネントに 16 ビットの引数を渡す場合は、Visual Basic のマネージ コードで、**UInteger** 型ではなく **UShort** 型で宣言してください。

参照

概念

リフレクションの概要

CLS 準拠コードの記述

その他の技術情報

リフレクション

参照ライブラリ '<filename>' を読み込めません : <error>

このプロジェクトによって参照されるライブラリを読み込むことができませんでした。

Error ID: BC31011

このエラーを解決するには

- 指定した場所にファイルがあることと、ファイルが別のアプリケーションによってロックされていないことを確認します。

参照

概念

プロジェクト参照

クラス '<classname>' の情報を読み込めません。

使用できないクラスが参照されました。

Error ID: BC30712

このエラーを解決するには

1. クラスが定義されていること、およびクラス名のスペルが正しいことを確認します。
2. モジュール内で宣言されているいずれかのメンバにアクセスしてみます。場合によっては、メンバが宣言されているモジュールが読み込まれていないために、デバッグ環境でメンバを特定できないことがあります。

参照

概念

クラス : オブジェクトの設計図

その他の技術情報

Visual Studio でのデバッグ

DLL ファイル '<filename>' を読み込めません : <error>

このプロジェクトによって参照されるダイナミック リンク ライブラリ (DLL: dynamic link library) を読み込むことができませんでした。

Error ID: BC31013

このエラーを解決するには

- 指定した場所にファイルがあることと、ファイルが別のアプリケーションによってロックされていないことを確認します。

参照

概念

Visual Basic、Visual C#、および Visual J# のファイルの種類とファイル名の拡張子

プロジェクト参照

リソース ファイル '<filename>' にリンクできません : <error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、アセンブリからネイティブな COM+ リソース ファイルにリンクするときにエラーが発生したことを報告しています。

Error ID: BC30144

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べてください。詳細とアドバイスについては、Al.exe ツールのエラーと警告を参照してください。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

関連項目

アセンブリ リンカ (Al.exe)
Al.exe ツールのエラーと警告

その他の技術情報

製品のサポートとユーザー補助

'<typename>' の型情報を取得できません。

共通言語ランタイムによって読み込まれていない型が参照されました。

Error ID: BC30751

このエラーを解決するには

- 現在のスコープで型が定義されるようにコードを再構成します。

参照

その他の技術情報

Visual Studio でのデバッグ

XML を解析できません : <error>

使用中の XML ファイルにエラーがあります。

Error ID: BC31023

このエラーを解決するには

- XML が整形式であること、有効であること、およびファイルが破損していないことを確認します。

参照

その他の技術情報

XML ドキュメントと XML データ

応答ファイル '<filename>' を開けません。

アクセスしようとしているファイルが使用できません。

Error ID: BC2011

このエラーを解決するには

- ネットワーク接続がアクティブなままであることを確認します。
- ファイルに対する権限を確認します。

参照

関連項目

FileIOPermission

概念

エラーの種類

リソース ファイル '<filename>' を開けません : <error>

指定したリソース ファイルを開くときにエラーが発生しました。

Error ID: BC31509

このエラーを解決するには

1. 指定したファイルの名前と場所が正しいことを確認します。
2. ファイルが破損していないことを確認します。

参照

関連項目

/linkresource (Visual Basic)
/resource (Visual Basic)

モジュール ファイル '<filename>' を開けません : <error>

このプロジェクトのモジュールを含むファイルを開くことができませんでした。

Error ID: BC31007

このエラーを解決するには

- 指定した場所にファイルがあることと、ファイルが別のアプリケーションによってロックされていないことを確認します。

参照

概念

Visual Basic、Visual C#、および Visual J# のファイルの種類とファイル名の拡張子

キー ファイル '<filename>' を開けません : <error>

このプロジェクトについて指定したキー ファイルを開くことができませんでした。

Error ID: BC31025

このエラーを解決するには

- キー ファイルが存在することと、別のアプリケーションによってロックされていないことを確認します。

参照

関連項目

厳密名ツール (Sn.exe)

概念

Visual Basic におけるグローバル属性

ファイル '<filename>' を開けません : <error>

このプロジェクト内のファイルを開くときにエラーが発生しました。

Error ID: BC31027

このエラーを解決するには

- ファイルが存在すること、破損していないこと、または別のアプリケーションによってロックされていないことを確認します。

参照

概念

Visual Basic、Visual C#、および Visual J# のファイルの種類とファイル名の拡張子

ファイル '<filename>' に署名できません : <error>

指定されたファイルに署名しようとしたときにエラーが発生しました。このエラーが発生する原因は複数あります。

- 十分なアクセス許可がない。
- Authenticode 署名に必要なシステム ファイルが存在しない。
- 存在しない証明書や秘密キー ファイルを参照している。
- ファイル名または URL のスペルが間違っている。

Error ID: BC31028

このエラーを解決するには

1. 正しい証明書および秘密キー ファイル名を入力します。
2. Authenticode 署名を使用している場合は、Signcode.exe ファイルと Javasign.dll ファイルが存在し、それらに読み取り専用属性が設定されていないことを確認します。
3. ファイルに対する **Write** 許可があることを確認します。

参照

関連項目

ファイル署名ツール (Signcode.exe)

概念

配置と Authenticode 署名

アセンブリに署名できません : <error message>

Visual Basic コンパイラは、アセンブリ リンカ (Al.exe、Alink とも呼ばれます) を呼び出して、マニフェストを含むアセンブリを生成します。リンカが、アセンブリ作成後のアセンブリの署名でのエラーを報告しています。キーが無効である可能性があります。

Error ID: BC30146

このエラーを解決するには

1. 引用符で囲まれたエラー メッセージを調べます。詳細とアドバイスについては、Al.exe ツールのエラーと警告を参照してください。
2. エラーが再発する場合は、エラーが発生したときの状況に関する情報を収集し、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

関連項目

アセンブリ リンカ (Al.exe)
Al.exe ツールのエラーと警告

その他の技術情報

製品のサポートとユーザー補助

一時パスが使用できないため、一時ファイルを書き込めません。

Visual Basic で、一時ファイルが格納されているディレクトリのパスを判断できませんでした。

Error ID: BC30698

このエラーを解決するには

1. Visual Studio を再起動します。
2. 問題が再発する場合は、Visual Studio を再インストールします。

参照

その他の技術情報

製品のサポート

出力をメモリに書き込めません。

出力をメモリに書き込むときに問題が発生しました。

Error ID: BC31020

このエラーを解決するには

1. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
2. エラーが再発する場合は、作業内容を保存し、Visual Studio を再起動します。
3. エラーが再発する場合は、Visual Basic を再インストールします。
4. 再インストール後もエラーが発生する場合は、マイクロソフト製品サポート サービスまでご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

使用されないローカル変数 '<localvariablename>' です。

プロシージャにローカル変数が宣言されていますが、一度も使用されません。

プロシージャのローカル変数の中に、スペルが間違っているものが存在する可能性があります。実際には別のステートメントで使用されている変数だが、スペルが間違っているという場合、コンパイラはこれらを 2 つの異なる変数と見なします。

既定では、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、
「Visual Basic での警告の構成」を参照してください。

Error ID: BC42024

このエラーを解決するには

1. プロシージャ内のローカル変数の中で、スペルの間違いをチェックします。大文字と小文字は区別されません。`ABC` と `abc` は同じ変数と見なされます。
2. スペルの間違いが存在しない場合は、この変数の宣言を削除するか、プロシージャ内の別のステートメントでこの変数を使用します。

参照

関連項目

Dim ステートメント (Visual Basic)

概念

宣言された要素の名前

スケジュールされていないファイバです。

デバッガは、物理スレッドにスケジュールされない論理ファイバに式が含まれているため、式を評価できません。SQL サーバーでファイバを使用してプロセスが実行されると、この問題が起こる可能性があります。

ファイバは、スタックとレジスタのコンテキストから構成され、スレッドで実行できます。ファイバは、スレッドからスワップ アウトされた後で別のスレッドで再起動されることがあります。

Error ID: BC30948

このエラーを解決するには

- ファイバが物理スレッドにスケジュールされるようにします。

参照

概念

SQL のデバッグ

その他の技術情報

Visual Studio でのデバッグ

オプション <optionname> は無効です。無視されます。

コマンド ラインから無効なオプションが指定されました。

Error ID: BC2007

このエラーを解決するには

- 指定されたオプションを調べて、無効なオプションであることを確認してください。

参照

関連項目

Switch 関数

enum の基になる型 '<typename>' は CLS に準拠していません。

この列挙値に指定されたデータ型は、共通言語仕様 (CLS) のデータ型ではありません。.NET Framework および Visual Basic はこのデータ型をサポートするため、このコンポーネントの内部ではエラーになりませんが、CLS に厳密に準拠したコードに作成された別のコンポーネントは、このデータ型をサポートしない可能性があります。そのようなコンポーネントは、このコンポーネントと正常にやり取りできない可能性があります。

次の Visual Basic データ型は CLS に準拠していません。

- SByte 型 (Visual Basic)
- UInteger データ型
- ULong データ型 (Visual Basic)
- UShort 型 (Visual Basic)

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC40032

このエラーを解決するには

- このコンポーネントが他の .NET Framework コンポーネントとのみやり取りする場合、または他のどのコンポーネントともやり取りしない場合は、何も変更する必要はありません。
- .NET Framework 用に作成されていないコンポーネントとやり取りする場合は、リフレクションまたはドキュメントを調べて、このデータ型がサポートされているかどうかを確認できる可能性があります。サポートされている場合は、何も変更する必要はありません。
- このデータ型をサポートしないコンポーネントとやり取りする場合は、CLS 準拠の型の中で最も近いデータ型に置き換える必要があります。たとえば、2,147,483,647 を超える値の範囲が必要でない場合は、**UInteger** の代わりに **Integer** を使用できます。範囲を拡張する必要がある場合は、**UInteger** を **Long** で置き換えてください。
- オートメーションまたは COM オブジェクトとやり取りする場合は、一部の型に .NET Framework とはデータ幅が異なるものがあることに注意してください。たとえば、**uint** は他の環境では 16 ビットです。そのようなコンポーネントに 16 ビットの引数を渡す場合は、Visual Basic のマネージ コードで、**UInteger** 型ではなく **UShort** 型で宣言してください。

参照

概念

リフレクションの概要

CLS 準拠コードの記述

その他の技術情報

リフレクション

出力ファイル '<filename>' に書き込めません : <error>

ファイルの作成で問題が発生しました。

出力ファイルを書き込み用に開くことができません。ファイル (または、そのファイルが格納されているフォルダ) は、別のプロセスによって排他的に開かれているか、読み取り専用属性が設定されている可能性があります。

ファイルが排他的に開かれている一般的な原因として、以下の状況が考えられます。

- このアプリケーションが既に実行されていて、ファイルを使用している。この問題を解決するためには、アプリケーションを終了します。
- 別のアプリケーションがファイルを開いている。この問題を解決するためには、他のアプリケーションによるファイルへのアクセスを中断します。どのアプリケーションがファイルにアクセスしているのかわからない場合があります。この場合、アプリケーション終了する最も簡単な方法として、コンピュータの再起動が考えられます。

プロジェクトの出力ファイルのいずれか 1 つだけが読み取り専用である場合にも、この例外がスローされます。

Error ID: BC31019

このエラーを解決するには

1. プログラムをもう一度コンパイルして、エラーが再発するかどうかを調べます。
2. エラーが再発する場合は、作業内容を保存し、Visual Studio を再起動します。
3. エラーが再発する場合は、コンピュータを再起動します。
4. エラーが再発する場合は、Visual Basic を再インストールします。
5. 再インストール後もエラーが発生する場合は、マイクロソフト製品サポート サービスまでご連絡ください。

Windows エクスプローラでファイル属性を確認するには

1. 対象のフォルダを開きます。
2. [表示] メニューの [詳細] を選択します。
3. 列ヘッダーを右クリックして、ドロップダウン リストから [属性] を選択します。

ファイルやフォルダの属性を変更するには

1. Windows エクスプローラで、ファイルかフォルダを右クリックして、[プロパティ] を選択します。
2. [全般] タブの [属性] セクションにある [読み取り専用] チェック ボックスをオフにします。
3. [OK] をクリックします。

参照

その他の技術情報

製品のサポートとユーザー補助

'Using' の終わりには、対応する 'End Using' を指定しなければなりません。

Using ステートメントに対応する **End Using** ステートメントがありません。

Using ブロックの最後には **End Using** ステートメントが必要です。

Error ID: BC36008

このエラーを解決するには

1. この **Using** ブロックが、入れ子になった **Using** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
2. **Using** ブロックの最後に **End Using** ステートメントを追加します。

参照

処理手順

方法 : システム リソースを破棄する

関連項目

Using ステートメント (Visual Basic)

型 '<typename>' の 'Using' オペランドは System.Idisposable を実装しなければなりません。

IDisposable インターフェイスを実装しない型のリソースが **Using** ステートメントに指定されています。

Using ブロックの目的は、ブロックの終了時にシステム リソースが確実に破棄されるようにすることです。この目的を満たすには、リソースは **IDisposable** から実装された Dispose メソッドを公開する必要があります。

Error ID: BC36010

このエラーを解決するには

- リソースを **Using** ステートメントのリソース リストから削除するか、**IDisposable** を実装するリソースに置き換えます。

参照

処理手順

方法 : システム リソースを破棄する

関連項目

Using ステートメント (Visual Basic)

IDisposable

'Using' リソース変数には明示的な初期化が必要です。

Using ステートメントに少なくとも 1 つのリソースが指定されていますが、そのリソースはステートメント内で **New** 句を使って初期化されていません。

制御を **Using** ブロックに渡す前にリソースを取得しておかない場合は、**Using** ステートメントでリソースを取得する必要があります。これを行うには、指定されたクラスからオブジェクトを作成する必要があるため、**New** 句が必要です。

Error ID: BC36011

このエラーを解決するには

- リソースを取得済みである場合は、取得したリソースを評価する **Using** ステートメント内で参照変数または式を使用します。

```
Dim newFont As New System.Drawing.Font

Using newFont
```

- リソースを取得済みでない場合は、**New** 句を **Using** ステートメントに追加します。

```
Using newFont as New System.Drawing.Font
```

参照

処理手順

方法 : システム リソースを破棄する

関連項目

Using ステートメント (Visual Basic)

New (Visual Basic)

'Using' リソース変数の型を配列型にすることはできません。

Using ステートメントが配列変数をリソースに指定しています。

Array クラスは IDisposable インターフェイスを実装していないので、**Using** ブロックは配列リソースに対して Dispose メソッドを呼び出すことができません。

Error ID: BC36012

このエラーを解決するには

- **Using** ブロック内で別の種類のシステム リソースを使用します。

参照

処理手順

方法 : システム リソースを破棄する

関連項目

Using ステートメント (Visual Basic)

概念

Visual Basic の配列の概要

'Using' ステートメントは、イミディエイト ウィンドウでは有効ではありません。

Using ステートメントはソース コードでしか使用できません。

Error ID: BC30186

このエラーを解決するには

- [イミディエイト] ウィンドウから **Using** ステートメントを削除します。

参照

関連項目

Using ステートメント (Visual Basic)

[イミディエイト ウィンドウ]

その他の技術情報

Visual Studio でのデバッグ

値 '<valuename1>' を値 '<valuename2>' に変換できません。

2 つの互換性のない型どうしで型変換を行おうとしました。

Error ID: BC30742

このエラーを解決するには

- キャストされている型を確認します。

参照

関連項目

CType 関数

型 '<typename1>' の値を '<typename2>' に変換できません。

エラー メッセージ
型 '<typename1>' の値を '<typename2>' に変換できません。型の不一致は、ファイル参照とアセンブリ '<assemblyname>' へのプロジェクト参照との混合によって生じた可能性があります。プロジェクト '<projectname1>' の '<filepath>' へのファイル参照を '<projectname2>' へのプロジェクト参照に置き換えてください。

プロジェクトでプロジェクト参照とファイル参照の両方が作成される場合、コンパイラは型が別の型に正しく変換されることを保証できません。

次の擬似コードは、このエラーが発生する可能性がある状況を示しています。

```
' ================ Visual Basic project P1 ================

' P1 makes a PROJECT REFERENCE to project P2

' and a FILE REFERENCE to project P3.

Public commonObject As P3.commonClass

commonObject = P2.getCommonClass()

' ================ Visual Basic project P2 ================

' P2 makes a PROJECT REFERENCE to project P3

Public Function getCommonClass() As P3.commonClass

Return New P3.commonClass

End Function

' ================ Visual Basic project P3 ================

Public Class commonClass

End Class
```

プロジェクト `P1` は、プロジェクト `P2` を経由するプロジェクト `P3` への間接プロジェクト参照を作成し、一方でプロジェクト `P3` への直接ファイル参照も作成します。`commonObject` の宣言では、`P3` へのファイル参照を使用しますが、`P2.getCommonClass` の呼び出しでは `P3` へのプロジェクト参照を使用します。

この状況で問題となるのは、ファイル参照では `P3` の出力ファイル (通常は p3.dll) のファイル パスとファイル名を指定するのに対して、プロジェクト参照ではソース プロジェクト (`P3`) をプロジェクト名で識別することです。これが原因で、コンパイラは、2 つの参照のどちらもからも同じソース コードの `P3.commonClass` 型に到達することを保証できません。

通常、この状況は、プロジェクト参照とファイル参照が混在する場合に起こります。前記の例では、`P1` が `P3` を参照するためにファイル参照ではなく直接プロジェクト参照を作成すると、この問題は起こりません。

Error ID: BC30955

このエラーを解決するには

- ファイル参照をプロジェクト参照に変更します。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
概念
プロジェクト参照
その他の技術情報
Visual Basic における型変換
名前空間およびコンポーネントの参照
参照の管理

型 '<typename1>' の値を '<typename2>' に変換することはできません。(複数ファイル参照)

エラー メッセージ
型 '<typename1>' の値を '<typename2>' に変換できません。型の不一致は、プロジェクト '<projectname1>' の '<filepath1>' へのファイル参照とプロジェクト '<projectname2>' の '<filepath2>' へのファイル参照との混合によって生じた可能性があります。両方のアセンブリが同一である場合は、同じ場所から参照するようにこれらの参照を置き換えてください。

プロジェクトで同じアセンブリへの複数のファイル参照が作成される場合、コンパイラは型が別の型に正しく変換されることを保証できません。

それぞれのファイル参照は、プロジェクトの出力ファイルのファイル パスおよび名前を指定しています (通常は DLL ファイル)。コンパイラは、出力ファイルが同じソースから得られることも、それらが同じアセンブリの同じバージョンを表していることも保証できません。したがって、異なる参照内の型が同じ型であることも、ある型を他の型に変換できることも保証できません。

複数の参照アセンブリのアセンブリ ID が同じであることがわかっている場合は、単一のファイル参照を使用できます。アセンブリ ID には、アセンブリの名前、バージョン、公開キー (公開キーがある場合)、およびカルチャが含まれます。この情報により、アセンブリを一意に識別します。

Error ID: BC30961

このエラーを解決するには

- 複数の参照アセンブリのアセンブリ ID が同じ場合は、一方のファイル参照を削除するか置き換えて、単一のファイル参照だけにします。
- 複数の参照アセンブリのアセンブリ ID が同じではない場合は、コードを変更して、一方のアセンブリの型を他方のアセンブリの型に変換しないようにします。

参照

処理手順
方法 : Visual Studio のリファレンスを追加または削除する
概念
プロジェクト参照
その他の技術情報
Visual Basic における型変換
名前空間およびコンポーネントの参照
参照の管理

型 '<type1>' の値を '<type2>' に変換できません。

ステートメントが、定義されていない方法で、あるデータ型を別のデータ型に変換しようとしました。このエラーでは以下の原因が考えられます。

- 変換が存在しない 2 つのデータ型間での変換が指定されています。**Boolean** 値から **Date** 型への変換などが該当します。
- 配列の初期化で、**New** 句の後に中かっこ (**{}**) が指定されていません。この場合、<type2> は、'<type> の 1 次元配列' 形式になります。

Error ID: BC30311

このエラーを解決するには

- 式のデータ型が変換先のデータ型に変換できるものであることを確認します。
- <type2> が配列の場合は、**New** 句で型名の後にかっこと中かっこの両方を指定します。次のコードは、配列の正しい初期化方法を示します。

```
Dim anIntArray As Integer() = New Integer() {}
```

参照

処理手順

方法 : 配列変数を初期化する

その他の技術情報

Visual Basic における型変換

'<type3>' が参照型でないため、'<type1>' 型の値を '<type2>' に変換できません。

ステートメントで、ある配列型を別の配列型に変換しようとしましたが、変換先の配列の要素のデータ型が参照型ではありません。

Error ID: BC30333

このエラーを解決するには

- 両方の配列の要素のデータ型を調べて、エラーの原因を特定します。

参照

概念

配列の変換

その他の技術情報

Visual Basic における型変換

'<type3>' は '<type4>' から派生していないため、型 '<type1>' の値を '<type2>' に変換できません。

あるステートメントで、1 つの配列型を別の配列型に変換しようとしましたが、配列の要素のデータ型がどちらも参照型ではないか、または 2 つの配列の要素型の間では拡大変換も縮小変換もできません。

Error ID: BC30332

このエラーを解決するには

- 両方の配列の要素のデータ型を調べて、エラーの原因を特定します。

参照

概念

配列の変換

その他の技術情報

Visual Basic における配列

配列型の次元数が異なるため、'<type1>' 型の値を '<type2>' に変換できません。

配列を次元数の異なる別の配列に変換しようとしました。

Error ID: BC30414

このエラーを解決するには

- 次元の数が一致するよう、一方または両方の配列を変更します。

参照

その他の技術情報

Visual Basic における配列

変数 '<variablename>' は、値が割り当てられる前に参照によって渡されています。

エラー メッセージ
変数 '<variablename>' は、値が割り当てられる前に参照によって渡されています。Null 参照の例外が実行時に発生する可能性があります。

値が代入されていない変数が、プロシージャ呼び出しによって引数として **ByRef** パラメータに渡されています。

変数に値が代入されない場合、変数にはそのデータ型の既定値が格納されます。参照データ型の既定値は、Nothing (Visual Basic) です。値が **Nothing** の参照変数を読み取ると、状況によっては NullReferenceException が発生します。

ByRef でプロシージャに引数を渡した場合、引数の基になる変数はプロシージャで変更できます。

既定では、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、「Visual Basic での警告の構成」を参照してください。

Error ID: BC42030

このエラーを解決するには

- プロシージャで **ByRef** の引数をとおして変数に値を代入する場合に、変数に既に値が格納されているかどうかが問題にならない場合は、アクションは必要ありません。
- 値を代入する前に引数を読み取るロジックがプロシージャ内にあり、その変数が値型である場合、プロシージャのロジックが、変数に既定値が含まれるかどうかに依存しないことを確認してください。
- 値を代入する前に引数を読み取るロジックがプロシージャ内にあり、その変数が参照型である場合、プロシージャのロジックが **Nothing** の値を扱えることを確認してください。たとえば、Try...Catch...Finally ステートメント (Visual Basic) を使って **NullReferenceException** をキャッチできます。

参照

処理手順
Visual Basic における変数のトラブルシューティング
関連項目
Dim ステートメント (Visual Basic)
ByRef
概念
値型と参照型
引数の値渡しおよび参照渡し
Visual Basic での変数宣言

変数 '<variablename>' は、値が割り当てられる前に参照によって渡されています。(構造体の変数)

エラー メッセージ
変数 '<variablename>' は、値が割り当てられる前に参照によって渡されています。Null 参照の例外が実行時に発生する可能性があります。構造体、またはすべての参照メンバが使用前に初期化されていることを確認してください。

値が代入されていない構造体変数が、プロシージャ呼び出しによって引数として **ByRef** パラメータに渡されています。

構造体変数に値が代入されない場合、構造体の各メンバにはそのデータ型の既定値が格納されます。参照データ型の既定値は、Nothing (Visual Basic) です。値が **Nothing** の参照メンバを読み取ると、状況によっては NullReferenceException が発生します。

ByRef でプロシージャに引数を渡した場合、引数の基になる変数はプロシージャで変更できます。

既定では、このメッセージは警告です。警告を表示しない方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42108

このエラーを解決するには

- プロシージャで **ByRef** の引数をとおして構造体メンバに値を代入する場合に、メンバに既に値が格納されているかどうかが問題にならない場合は、アクションは必要ありません。
- 値を代入する前に構造体メンバを読み取るロジックがプロシージャ内にあり、そのメンバが値型である場合、プロシージャのロジックが、メンバに既定値が含まれるかどうかに依存しないことを確認してください。
- 値を代入する前に構造体メンバを読み取るロジックがプロシージャ内にあり、そのメンバが参照型である場合、プロシージャのロジックが **Nothing** の値を扱えることを確認してください。たとえば、Try...Catch...Finally ステートメント (Visual Basic) を使って **NullReferenceException** をキャッチできます。

参照

処理手順
Visual Basic における変数のトラブルシューティング
関連項目
Dim ステートメント (Visual Basic)
ByRef
Structure ステートメント
概念
値型と参照型
引数の値渡しおよび参照渡し
Visual Basic での変数宣言
その他の技術情報
構造体 : 独自のデータ型

変数 '<variablename>' は、値が割り当てられる前に参照によって使用されています。

エラー メッセージ
変数 '<variablename>' は、値が割り当てられる前に参照によって使用されています。null 参照の例外が実行時に発生する可能性があります。

アプリケーション コードの実行経路の中に、値を割り当てる前に変数を読み込むことになる経路が少なくとも 1 つあります。

変数に値が一度も割り当てられない場合は、そのデータ型の既定値が格納されます。参照データ型の場合は、Nothing (Visual Basic) がその既定値になります。**Nothing** の値を持つ参照変数を読み込むと、状況によっては NullReferenceException が発生する可能性があります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42104

このエラーを解決するには

- 制御フローのロジックを調べて、変数を読み込むステートメントに制御が渡されるよりも前に、有効な値が格納されることを確認します。
- 変数が常に有効な値を持つための確実な方法の 1 つは、変数を宣言時に初期化することです。「Dim ステートメント (Visual Basic)」の「初期化」を参照してください。

参照

処理手順
Visual Basic における変数のトラブルシューティング
関連項目
Dim ステートメント (Visual Basic)
概念
値型と参照型
Visual Basic での変数宣言

変数 '<variablename>' は、囲まれたブロック内の変数を非表示にします。

ブロック内の変数の名前が、別のローカル変数の名前と同じです。

Error ID: BC30616

このエラーを解決するには

- ブロック内の変数の名前を変更して、別のローカル変数の名前と重複しないようにします。たとえば、次のようにします。

```
Dim a, b, x As Integer
If a = b Then
   Dim y As Integer = 20 ' Uniquely named block variable.
End If
```

- このエラーの一般的な原因は、イベント ハンドラ内で `Catch e As Exception` を使用することにあります。その場合は、**Catch** ブロックの変数名を `e` ではなく `ex` とします。
- その他の一般的な原因としては、**Try** ブロック内で宣言されたローカル変数に個別の **Catch** ブロックでアクセスしようとしたことが考えられます。これを修正するには、**Try...Catch...Finally** 構造体の外部で変数を宣言します。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)

概念

Visual Basic での変数宣言

変数 '<variablename>' は、値が割り当てられる前に参照によって使用されています。(Visual Basic エラー)

値が割り当てられる前に変数が使用されているため、NullReferenceException がスローされます。

Error ID: BC42109

このエラーを解決するには

- 変数に値が割り当てられていることを確認します。

参照

関連項目

例外のトラブルシューティング : System.NullReferenceException

概念

Visual Basic での変数宣言

変数宣言ステートメントは、イミディエイト ウィンドウでは有効ではありません。

変数宣言は [イミディエイト] ウィンドウでは使用できません。

Error ID: BC30709

このエラーを解決するには

- [イミディエイト] ウィンドウでは変数を宣言しないでください。

参照

関連項目

[イミディエイト ウィンドウ]

その他の技術情報

Visual Studio でのデバッグ

'As' 句のない変数です。Object の型と見なされます。

変数宣言で **As** 句が指定されていません。

As 句は、プログラミング要素に関連付けるデータ型を指定します。Dim ステートメント (Visual Basic) では、この句は変数のデータ型を指定します。**As** 句を **Dim** ステートメントに含めない場合、変数のデータ型は既定で **Object** になります。

既定では、このメッセージは警告です。警告を非表示にする方法や、警告をエラーとして扱う方法の詳細については、Visual Basic での警告の構成 を参照してください。

Error ID: BC42020

このエラーを解決するには

- **As** 句を **Dim** ステートメントに含めて、変数のデータ型を指定します。

参照

関連項目

Dim ステートメント (Visual Basic)

概念

Visual Basic での変数宣言

Visual Basic でサポートされていないオートメーションが変数で使用されています。

タイプ ライブラリまたはオブジェクト ライブラリで定義されている変数を使おうとしましたが、Visual Basic でサポートされていないデータ型が含まれています。

このエラーを解決するには

- Visual Basic で認識される型の変数を使用します。

  または

- **FileGet** または **FileGetOBject** の使用時にこのエラーが発生する場合は、使おうとしているファイルが **FilePut** または **FilePutObject** によって書き込みが行われたことを確認します。

参照

関連項目

データ型の概要 (Visual Basic)

モジュールの変数を '<specifier>' として宣言することはできません。

Module ステートメント内の変数に対して **MustInherit** などの指定子を使用しました。モジュールはインスタンス化できず、継承をサポートしません。また、インターフェイスを実装できません。

Error ID: BC30593

このエラーを解決するには

- 指定子を削除します。

参照

関連項目

Module ステートメント

'Variant' は現在サポートされていません。'Object' 型を使用してください。

バリアント型 (**Variant**) は、オブジェクト型 (**Object**) に置き換えられました。

Error ID: BC30804

このエラーを解決するには

- アイテムをオブジェクト型 (**Object**) として宣言します。

参照

関連項目

オブジェクト型 (Object)

概念

データ型の変更点 (Visual Basic 6.0 ユーザー向け)

以下のエラーから Visual Basic 2005 コンパイラを回復できません : <error>

Visual Basic コンパイラは次のエラー *<error>* を回復できません。

一部の IDE 機能が無効になっています。

Visual Basic コンパイラで内部エラーが発生しました。

Error ID: BC31021

このエラーを解決するには

1. 作業内容を保存し、Visual Studio を再起動します。
2. エラーが再発する場合は、Visual Studio を再インストールします。
3. 再インストール後もエラーが発生する場合は、マイクロソフト製品サポート サービスまでご連絡ください。

参照

その他の技術情報

製品のサポートとユーザー補助

警告番号 '<number>' (オプション '<optionname>') は構成可能でないか、または有効ではありません。

警告 ID が無効です。ID は 0 より大きい整数値である必要があります。

Error ID: BC2026

このエラーを解決するには

- 使用されている警告 ID が有効であることを確認します。

参照

概念

Visual Basic での警告の構成

警告がエラーとして処理されました <error>

[警告をエラーとして扱う] オプションをオンにしてコンパイルされたアセンブリを実行しているときに、警告が検出されました。

Error ID: BC31072

このエラーを解決するには

1. 警告の原因を解決します。
2. [警告をエラーとして扱う] オプションをオフにしてアセンブリをコンパイルします。

参照

関連項目

/warnaserror (Visual Basic)
[ビルド] ページ ([プロパティ ページ])
配置プロジェクトの [プロパティ ページ] ダイアログ ボックス ([構成プロパティ] - [ビルド])

Web メソッドを、デバッグ ウィンドウで評価することはできません。

XML Web サービス メソッドはデバッグ ウィンドウではテストできません。

Error ID: BC30732

このエラーを解決するには

- コード エディタで XML Web サービス メソッドをテストします。

参照

その他の技術情報

Visual Studio でのデバッグ

含んでいるクラスが Web サービスとして公開されていないため、'WebMethod' 属性はこのメンバに影響を与えません。

WebMethodAttribute 属性は、メソッドをリモートの Web クライアントから呼び出し可能にしますが、メソッドのクラスが WebService から派生していなければ呼び出し可能にはなりません。

Error ID: BC30771

このエラーを解決するには

- クラスが **WebService** から派生するように変更します。

  または

- メソッドから **WebMethodAttribute** 属性を削除します。

参照

処理手順

チュートリアル : Visual Basic または Visual C# を使った XML Web サービスの作成

'Wend' ステートメントはサポートされていません。代わりに 'End While' を使用してください。

Wend キーワードは、**End While** 構成体に置き換えられました。

Error ID: BC30809

このエラーを解決するには

- **End While** を使用して **While** ブロックを終了させてください。

参照

関連項目

While...End While ステートメント (Visual Basic)
End (Visual Basic)

'While' の終わりには、対応する 'End While' を指定しなければなりません。

While ステートメントに対応する **End While** ステートメントがありません。**While** ブロックの最後には **End While** ステートメントが必要です。

Error ID: BC30082

このエラーを解決するには

1. この **While** ブロックが、入れ子になった **While** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
2. **While** ブロックの最後に **End While** ステートメントを追加します。

参照

関連項目

While...End While ステートメント (Visual Basic)

'Widening' と 'Narrowing' を組み合わせて使用することはできません。

Operator ステートメントに Widening と Narrowing が両方とも指定されています。

変換演算子を定義する場合は、**Widening** または **Narrowing** のいずれかで宣言する必要があります。この 2 つは一度に 1 つしか選択できないため、両方を指定することはできません。

Error ID: BC33001

このエラーを解決するには

- **Widening** キーワードまたは **Narrowing** キーワードを **Operator** ステートメントから削除します。どちらか一方を指定する必要があります。

参照

処理手順

方法 : 演算子を定義する

方法 : 変換演算子を定義する

関連項目

Operator ステートメント

概念

演算子プロシージャ

'With' コンテキストおよびステートメントは、デバッグ ウィンドウでは有効ではありません。

With ステートメントはソース コードでのみ使用できます。

Error ID: BC30726

このエラーを解決するには

- デバッグ コードから **With** ステートメントを削除します。

参照

関連項目

With...End With ステートメント (Visual Basic)

その他の技術情報

Visual Studio でのデバッグ

'With' の終わりには、対応する 'End With' を指定しなければなりません。

With ステートメントに対応する **End With** ステートメントがありません。**With** ブロックの最後には **End With** ステートメントが必要です。

Error ID: BC30085

このエラーを解決するには

- この **With** ブロックが、入れ子になった **With** ブロックの一部である場合は、すべてのブロックが正しく終了していることを確認します。
- **With** ブロックの最後に **End With** ステートメントを追加します。

参照

関連項目

With...End With ステートメント (Visual Basic)

'WithEvents' 変数は、イベントを発生させません。

WithEvents キーワードを指定して宣言されているオブジェクト変数が、イベントを発生させることのできるクラス インスタンスを参照していません。

Error ID: BC30591

このエラーを解決するには

- オブジェクト変数がイベントを発生させることのできるクラス インスタンスを参照していることを確認します。

参照

概念

イベントとイベント ハンドラ

イベントとイベント ハンドラ

'WithEvents' 変数は、'<containername>' にアクセス可能なインスタンス イベントを発生させません。

参照先のオブジェクトが、アクセス可能なイベントを発生しません。

Error ID: BC30600

このエラーを解決するには

- イベントのアクセス修飾子を変更するか、または現在のコンテキストでアクセスできるイベントを選択します。

参照

概念

イベントとイベント ハンドラ

WithEvents の変数名の長さは 1019 文字を超過できません。

WithEvents を指定して宣言する各変数の名前、つまり識別子は 1019 文字に制限されています。

Error ID: BC32205

このエラーを解決するには

- 識別子の文字数を減らします。

参照

関連項目

WithEvents

概念

宣言された要素の名前

'WithEvents' 変数は、クラス、インターフェイスまたはクラスの制約がある型パラメータとしてのみ型指定することができます。

構造体として型指定されている変数の宣言で、**WithEvents** を使用しました。これは、**WithEvents** 修飾子の有効な使い方ではありません。

Error ID: BC30413

このエラーを解決するには

- 構造体で定義されているイベントを処理するには、**AddHandler** を使用します。

参照

関連項目

Dim ステートメント (Visual Basic)
AddHandler ステートメント

概念

WithEvents と Handles 句

'WithEvents' 変数を配列として型指定することはできません。

WithEvents を使って配列を宣言しようとしました。個別の変数の場合、いくつでも **WithEvents** を指定して宣言できますが、配列を宣言することはできません。

Error ID: BC30476

このエラーを解決するには

- 変数を個別に宣言します。

参照

関連項目

Dim ステートメント (Visual Basic)

概念

WithEvents と Handles 句

'WithEvents' 変数には 'As' 句を指定しなければなりません。

WithEvents キーワードに **As** 句が指定されていません。イベントを発生できる特定のクラスとして変数を宣言してください。

Error ID: BC30412

このエラーを解決するには

- イベントを発生できるクラスを指定する **As** 句を追加します。

参照

関連項目

Dim ステートメント (Visual Basic)

概念

WithEvents と Handles 句

'Set' で 'WriteOnly' プロパティにアクセス修飾子を指定することはできません。

WriteOnly プロパティの宣言で、Property ステートメントと Set ステートメント (Visual Basic) の両方にアクセス レベルを指定しています。

プロパティにはアクセス レベルを指定できます。さらに、プロパティ プロシージャ (**Get** または **Set**) のうち最大 1 つに別のアクセス レベルを指定できます (ただし、そのアクセス レベルがプロパティのアクセス レベルよりも制限が厳しい場合)。両方のプロパティ プロシージャにアクセス レベルを指定することはできません。

Error ID: BC31104

このエラーを解決するには

- **Set** ステートメントからアクセス修飾子を削除します。これは完全な **WriteOnly** プロパティを表し、このプロパティに 2 つのアクセス レベルを持たせることはできません。

参照

処理手順

方法 : 複数のアクセス レベルを持つプロパティを宣言する

概念

Property プロシージャ

'WriteOnly' プロパティには 'Set' を指定する必要があります。

プロパティを **WriteOnly** として宣言している場合は、その値を書き込むためのプロシージャを記述する必要があります。

Error ID: BC30125

このエラーを解決するには

1. **Property** ステートメントと **End Property** ステートメントの間に、**Set** プロシージャを記述します。
2. **Property** 宣言内のその他のプロシージャが正しく終了していることを確認します。

参照

関連項目

Property ステートメント
Set ステートメント (Visual Basic)

'<genericprocedurename>' メソッド '|1' に渡される型引数の数が正しくありません。

ジェネリック プロシージャが、その型パラメータの数と一致しない数の型引数を指定して呼び出されました。

Error ID: BC30951

このエラーを解決するには

- ジェネリック プロシージャに定義されたすべての型パラメータに対して型引数を指定します。

  または

- 型引数を何も指定せずにジェネリック プロシージャを呼び出し、コンパイラが型引数の推論を試みるようにします。

参照

関連項目

AddressOf 演算子
型リスト

概念

Visual Basic におけるジェネリック型
Visual Basic におけるジェネリック プロシージャ

XML ドキュメント解析エラー : 開始タグ '<tag>' に対応する終了タグがありません

エラー メッセージ
XML ドキュメント解析エラー : 開始タグ '<tag>' に対応する終了タグがありません。XML コメントは無視されます。

XML コメントに開始タグがありますが、対応する終了タグがありません。

Error ID: BC42316

このエラーを解決するには

- 開始タグに対応する終了タグを追加します。

  または

- <seealso> (Visual Basic) などのように、タグの内部にテキストがない場合は、右山形かっこの前にスラッシュを指定します。

参照

関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML ドキュメント解析エラー : <error>

このエラーでは、以下のメッセージが表示されることがあります。詳細については、以下のトピックを参照してください。

- 同一の属性を伴う XML コメント タグ '<tag>' が、同じ XML コメント ブロック内に複数回記述されています。
- XML のコメント タグ '<tag>' は、'<element>' 言語要素では使用できない
- XML コメント パラメータ '<parameter>' が対応する <keyword> ステートメント上のパラメータと一致しません。
- XML コメント パラメータには 'name' 属性を指定しなければなりません。
- XML コメントには、解決できなかった 'cref' 属性 '<attribute>' を伴うタグがあります。
- XML コメント タグ 'include' には '<attribute>' 属性を指定しなければなりません。

Error ID: BC42304

参照

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

概念

XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML コメントの型パラメータには 'name' 属性を指定しなければなりません。

エラー メッセージ
XML コメントの型パラメータには 'name' 属性を指定しなければなりません。XML コメントは無視されます。

XML コメント タグには **name** 属性が必要ですが、これが指定されていないか、または完全ではありません。

Error ID: BC42318

このエラーを解決するには

- 必須の **name** 属性をタグに追加するか、正しく指定します。

参照

関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML コメントの型パラメータ <parameter> が対応する <keyword> ステートメント上の型パラメータと一致しません

エラー メッセージ
XML コメントのパラメータ <parameter> が対応する <keyword> ステートメント上の型パラメータと一致しません。XML コメントは無視されます。

XML コメント タグが、メソッドの一部の (すべてのではない) 型パラメータに対して指定されています。

Error ID: BC42317

このエラーを解決するには

- XML コメント タグを、メソッドのすべての型パラメータに対して指定します。

参照

関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML コメント タグ 'returns' は、'WriteOnly' プロパティに許可されていません。

エラー メッセージ
XML コメント タグ 'returns' は、'WriteOnly' プロパティに許可されていません。XML コメントは無視されます。

書き込み専用のプロパティの宣言に対する XML コメントに、<returns> タグを含めることはできません。

Error ID: BC42313

このエラーを解決するには

- サポートされない <returns> タグを削除します。

参照

関連項目
<returns> (Visual Basic)
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
Property ステートメント
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML コメント タグ 'returns' は、'declare sub' 言語要素に許可されていません

エラー メッセージ
XML コメント タグ 'returns' は、'declare sub' 言語要素に許可されていません。XML コメントは無視されます。

Declare Sub の XML コメントに、<returns> タグを含めることはできません。

Error ID: BC42315

このエラーを解決するには

- サポートされない <returns> タグを削除します。

参照

関連項目

<returns> (Visual Basic)
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
Declare ステートメント

概念

XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML コメント タグ 'include' には '<attribute>' 属性を指定しなければなりません。

エラー メッセージ
XML コメント タグ 'include' には '<attribute>' 属性を指定しなければなりません。XML コメントは無視されます。

<include> タグに渡された **file** または **path** 属性が見つからないか、または不完全です。

Error ID: BC42310

このエラーを解決するには

- <include> タグに必要な属性を追加するか、正しく記述します。

参照

関連項目
<include> (Visual Basic)
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML のコメント タグ '<tag>' は、'<element>' 言語要素では使用できない

エラー メッセージ
XML のコメント タグ '<tag>' は、'<element>' 要素では使用できません。XML コメントは無視されます。

XML のコメント タグは、そこに適用された言語要素でサポートされません。

Error ID: BC42301

このエラーを解決するには

- サポートされない XML コメント タグを削除します。

参照

関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

同一の属性を伴う XML コメント タグ '<tag>' が、同じ XML コメント ブロック内に複数回記述されています。

エラー メッセージ
同一の属性を伴う XML コメント タグ <tag> が、同じ XML コメント ブロック内に複数回記述されています。XML コメントは無視されます。

XML コメントに、同一の属性を伴うタグが複数記述されています。

Error ID: BC42305

このエラーを解決するには

- 重複しているタグを削除します。

参照

関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML コメント パラメータには 'name' 属性を指定しなければなりません。

エラー メッセージ
XML コメント パラメータには 'name' 属性を指定しなければなりません。XML コメントは無視されます。

XML タグに、必須の **name** 属性が指定されていません。

Error ID: BC42308

このエラーを解決するには

- 必須の **name** 属性をタグに追加します。

参照

関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML コメント パラメータ '<parameter>' が対応する <keyword> ステートメント上のパラメータと一致しません。

エラー メッセージ
XML コメント パラメータ <parameter> が対応する <keyword> ステートメント上のパラメータと一致しません。XML コメントは無視されます。

XML コメント タグが、メソッドの一部の (すべてのではない) パラメータに対して指定されています。

Error ID: BC42307

このエラーを解決するには

- XML コメント タグを、メソッドのすべてのパラメータに対して指定します。

参照

関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML コメントは行内で最初のステートメントでなければなりません。

エラー メッセージ
XML コメントは行内で最初のステートメントでなければなりません。XML コメントは無視されます。

XML コメントが他の言語要素の後に置かれており、これは許可されていません。

Error ID: BC42302

このエラーを解決するには

- XML コメントを新しい行に移動します。

参照

関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)
概念
XML の使用によるコードのドキュメントの作成 (Visual Basic)

XML コメントには、解決できなかった 'cref' 属性 '<attribute>' を伴うタグがあります。

エラー メッセージ
XML コメントには、解決できなかった 'cref' 属性 '<attribute>' を伴うタグがあります。XML コメントは無視されます。

タグには、識別子の相対名を指定して他の XML 要素へのリンクを指定する **cref** 属性があります。コンパイル時に、ユーザーが指定した値の修飾 XML 識別子で値が置き換えられます。コンパイラは通常の解決規則を使用して、データ型またはメンバを検索します。

Error ID: BC42309

このエラーを解決するには

- **cref** 属性が有効なコード要素を指しているかどうかを確認します。

参照

処理手順
方法 : Visual Basic で XML ドキュメントを作成する
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

XML コメントの例外には 'cref' 属性を指定しなければなりません

<exception> タグを使用すると、メソッドによってスローされる可能性のある例外をドキュメントに出力できます。必須の **cref** 属性には、メンバの名前を指定します。この名前はドキュメント生成機能によってチェックされます。メンバが存在する場合は、メンバの名前がドキュメント ファイル内で標準の要素名に変換されます。

Error ID: BC42319

このエラーを解決するには

- **cref** 属性を、次のように例外に追加します。

```
'''<exception cref="member">description</exception>
```

参照

処理手順

方法 : Visual Basic で XML ドキュメントを作成する

関連項目

<exception> (Visual Basic)

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

XML コメントを partial の <type> に対して複数回適用することはできません。

エラー メッセージ
XML コメントを、partial の <type> に複数回適用することはできません。この <type> に対する XML コメントは無視されます。

XML コメント ブロックは、部分型の 1 つの部分にのみ定義できます。

XML コメントブロックが部分型の複数の部分に定義されると、この警告がコメント ブロックごとに作成され、最上位の XML コメントが無視されます。

Error ID: BC42314

このエラーを解決するには

- 余分なコメント ブロックを削除します。

参照
処理手順
方法 : Visual Basic で XML ドキュメントを作成する
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

XML コメントは、メソッドまたはプロパティには記述できません。

XML コメントはメソッドまたはプロパティ内には記述できません。XML コメントは無視されます。XML コメント ブロックはメソッドまたはプロパティ内には記述できません。

Error ID: BC42303

このエラーを解決するには

- コメント ブロックを削除するか、適切な場所に移動します。

参照

処理手順

方法 : Visual Basic で XML ドキュメントを作成する

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

XML コメント ブロックは、適用される言語要素の直前に入れなければなりません。

エラー メッセージ
XML コメント ブロックは、適用される言語要素の直前に入れなければなりません。XML コメントは無視されます。

このエラーは、タグが間違った位置にある場合や、タグの形式が不適切である場合にも発生します。

Error ID: BC42300

このエラーを解決するには

1. コメント ブロックを、適用される言語要素の前に移動します。開始タグの前に、余分な文字を誤って入力しないよう注意してください。
2. タグが有効であることを確認します。

参照

処理手順
方法 : Visual Basic で XML ドキュメントを作成する
関連項目
ドキュメント コメントとして推奨される XML タグ (Visual Basic)

XML コメント ブロックを XML ドキュメント コメントのアプリケーションをサポートする言語要素と関連付けできません。

XML コメント ブロックを、有効な言語要素に関連付けできません。XML コメントは無視されます。このエラーは、コメント ブロック内の XML タグに対応する言語要素がない場合にも発生します。

Error ID: BC42312

このエラーを解決するには

- 余分な XML タグを削除するか、このタグが記述する言語要素を追加します。

参照

処理手順

方法 : Visual Basic で XML ドキュメントを作成する

関連項目

ドキュメント コメントとして推奨される XML タグ (Visual Basic)

オプション '<オプション>' の後に '+' または '-' を指定できません。無視されます。

コンパイラ オプションが + 値または - 値で指定されましたが、これらの値はサポートされていません。

コンパイラ オプションとそれがサポートしている値の詳細については、「Visual Basic コンパイラ オプション一覧 (アルファベット順)」を参照してください。

Error ID: BC2028

このエラーを修正してオプションを使用するには

- コンパイラ オプションを値なしで指定します。

このエラーを修正してオプションを使用しないようにするには

- コンパイラ オプションを削除します。

参照

関連項目

Visual Basic コンパイラ オプション一覧 (アルファベット順)

エラー報告を自動的に送信することはできません。

エラー メッセージ
エラー報告を自動的に送信できません。エラー報告の送信設定を構成するには、'http://go.microsoft.com/fwlink/?LinkId=42039' を参照してください。

/errorreport:send コンパイラ オプションを指定しましたが、コンピュータではエラー レポートの自動送信は設定されていません。Visual Basic コンパイラの内部エラーについての情報は一切送信されません。

Error ID: BC2027

このエラーを解決するには

- **/errorreport:send** コンパイラ オプションを削除するか、**/errorreport:queue**、**/errorreport:prompt**、または **/errorreport:none** と置き換えます。

  または

- http://go.microsoft.com/fwlink/?LinkId=42039 の指示に従って、エラーの自動報告を有効にします。

参照

関連項目
/errorreport
その他の技術情報
http://go.microsoft.com/fwlink/?LinkId=42039

ファイル '<commentfile>' の XML フラグメント '<elementpath>' を含めることができません。

XML コメント内の `<include>` タグに存在しない要素の XPath 式が含まれているか、またはファイルに正しくない形式の XML が含まれています。

Error ID: BC42320

このエラーを解決するには

1. ファイル内の XML が有効であることを確認します。
2. XML フラグメントがファイル内に存在することを確認します。

参照

関連項目

<include> (Visual Basic)

XML ファイルの形式が正しくありません。ファイル '<commentfile>' を含めることができません。

いずれかの XML コメントの `<include>` タグが、存在しないファイル、または正しくない形式の XML を含むファイルを参照しています。

Error ID: BC42321

このエラーを解決するには

1. 該当するファイルが存在していることを確認します。
2. ファイル内の XML が有効であることを確認します。

参照

関連項目

<include> (Visual Basic)

属性 'StructLayout' はジェネリック型に適用できません。

このトピックは、Visual Studio 2005 SP1 に合わせて新たに追加されました。

StructLayoutAttribute 属性がジェネリック型に対して適用されています。ジェネリック型の構造体は、さまざまな型引数に基づいて実行時に構築されるため、コンパイルの時点ではそれがどのような構造体かはわかりません。**StructLayoutAttribute** 属性を指定するには対象となる構造体がコンパイルの時点でわかっている必要があります。そのため、この属性をジェネリック型に適用することはできません。

Error ID: BC30972

このエラーを解決するには

- ジェネリック型から **StructLayoutAttribute** 属性を削除します。

参照

概念

Visual Basic におけるジェネリック型
Visual Basic における属性

式で使用されるクラスがデバッグ セッション中に読み込まれていません。

このトピックは、Visual Studio 2005 SP1 に合わせて新たに追加されました。

デバッグ中に、まだ読み込まれていないクラスが参照されました。デバッガに対しクラスを強制的に読み込ませるには、そのクラスの新しいインスタンスを作成し、それを一時変数に代入します。これで、エラーの発生した式を再評価できるようになります。

Error ID: BC30973

このエラーを解決するには

1. **New** キーワードを使ってクラスのインスタンスを作成し、そのインスタンスを一時変数に代入します。
2. エラーの発生した式を評価します。

参照

関連項目

New (Visual Basic)

コメント ステートメントは評価できません。

このトピックは、Visual Studio 2005 SP1 に合わせて新たに追加されました。

デバッグ ウィンドウでコメントが使用されました。

Error ID: BC30974

このエラーを解決するには

- デバッグ ウィンドウではコメントを使わないようにします。

参照

概念

コード内のコメント

その他の技術情報

Visual Studio でのデバッグ

For Loop コントロール変数 '<変数名>' は、それを囲む For loop によって既に使用されています。

このトピックは、Visual Studio 2005 SP1 に合わせて新たに追加されました。

入れ子にされた **For** ループには、それぞれ一意の制御変数 (ループ カウンタ) が使用されている必要があります。

Error ID: BC30975

このエラーを解決するには

- いずれかの制御変数の名前を変更します。

参照

関連項目

For...Next ステートメント (Visual Basic)

Next に指定する変数は、For loop に指定する変数と一致しません。

このトピックは、Visual Studio 2005 SP1 に合わせて新たに追加されました。

Next ループの **For...Next** ステートメント内の制御変数は、対応する **For** ステートメント内の変数と一致している必要があります。

Error ID: BC30976

このエラーを解決するには

1. **Next** ステートメント内の変数のスペルと、対応する **For** ステートメント内の変数のスペルが一致していることを確認します。
2. 外側のループの一部を誤って削除していないかどうかを確認します。
3. このループが、入れ子になったループ セットの一部である場合は、すべてのループが正しく終了しているかどうかを確認します。

参照

関連項目

For...Next ステートメント (Visual Basic)

'System.Nullable(Of T)' のメソッドを 'AddressOf' 演算子のオペランドとして使用することはできません。

このトピックは、Visual Studio 2005 SP1 に合わせて新たに追加されました。

ステートメントに **AddressOf** 演算子が使用され、Nullable 構造体のプロシージャまたはプロパティを表すオペランドが指定されています。

Error ID: BC32126

このエラーを解決するには

- **AddressOf** 句に指定されているプロシージャ名またはプロパティ名を置き換えます。**Nullable** のメンバ以外のオペランドを使用する必要があります。

参照

関連項目

AddressOf 演算子

Nullable

概念

定義された値を持たない可能性のある値型

Visual Basic におけるジェネリック型

Visual Basic のランタイム メッセージ

このセクションでは、実行時に表示される Visual Basic エラー メッセージを示します。

参照

処理手順

方法 : Visual Basic ランタイム エラーに関する情報を取得する

その他の技術情報

エラー メッセージ (Visual Basic)

<アドレス> は無効なリモート ファイル アドレスです。

エラー メッセージ
<address> は無効なリモート ファイル アドレスです。有効なアドレスはプロトコル、パスおよびファイル名を含んでいる必要があります。

アップロード処理、またはダウンロード処理に、無効なアドレスが指定されました。アドレスには `ftp` や `http` などのプロトコル、有効なパス、およびファイル名を含める必要があります。ファイル パスの解析が間違っている可能性があります。

このエラーを解決するには

- 指定したアドレスを調べて問題のある場所を判断し、必要な情報を追加します。

参照

処理手順
方法 : Visual Basic でファイルをアップロードする
方法 : Visual Basic でファイルをダウンロードする
関連項目
My.Computer.Network.UploadFile メソッド
My.Computer.Network.DownloadFile メソッド
その他の技術情報
Visual Basic による .NET Framework でのネットワーク操作

区切り記号は Nothing または空の文字列は使用できません。

Delimiters プロパティに **Nothing** が設定されているか、または 空の **String** ("") であるため、**TextFieldParser** はファイルからデータを読み込むことができません。

このエラーを解決するには

- **Delimiters** に有効な値を指定します。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る

関連項目

TextFieldParser.SetDelimiters メソッド
TextFieldParser.Delimiters プロパティ
TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

二重引用符は、EscapeQuote が True に設定されている区切り符号で分けられたフィールドには有効なコメント トークンではありません。

TextFieldParser の区切り記号として二重引用符が指定されましたが、**EscapeQuotes** に **True** が設定されています。

このエラーを解決するには

- **EscapeQuotes** に **False** を設定します。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る

関連項目

TextFieldParser.SetDelimiters メソッド
TextFieldParser.Delimiters プロパティ
TextFieldParser オブジェクト

ログは既に、この名前でこのコンピュータに作成されています。

既に使用されている名前で、ログの作成をしようとしました。

このエラーを解決するには

- 競合しているログを削除します。
- ログに違う名前を使用します。

参照

処理手順

方法 : ログ メッセージを書き込む
方法 : アプリケーションの起動時または終了時にメッセージをログに記録する
方法 : アプリケーション イベント ログに書き込む
チュートリアル : My.Application.Log による情報の書き込み先の確認
チュートリアル : My.Application.Log による情報の書き込み先の変更

関連項目

My.Application.Log オブジェクト

概念

Visual Basic でのアプリケーション ログの使用

プロパティまたはメソッドの呼び出しには、引数または戻り値としてプライベート オブジェクトへの参照を含めることはできません。

このエラーでは以下の原因が考えられます。

- クライアントが、アウトプロセス コンポーネントのプロパティまたはメソッドを呼び出し、引数の 1 つとしてプライベート オブジェクトへの参照を渡そうとしました。
- アウトプロセス コンポーネントがクライアントに対してコールバック メソッドを呼び出し、プライベート オブジェクトへの参照を渡そうとしました。
- アウトプロセス コンポーネントが、それ自身が発生させているイベントの引数として、プライベート オブジェクトへの参照を渡そうとしました。
- クライアントが、それ自身が処理しているイベントの **ByRef** 引数に、プライベート オブジェクト参照を代入しようとしました。

このエラーを解決するには

- 参照を削除します。

参照

関連項目

Private (Visual Basic)

スタートアップ フォームが指定されていません。

アプリケーションで WindowsFormsApplicationBase クラスが使用されますが、スタートアップ フォームが指定されていません。

これは、プロジェクト デザイナで [アプリケーション フレームワークを有効にする] チェック ボックスがオンに設定されているとき、スタートアップ フォームが指定されていない場合に発生します。詳細については、「[アプリケーション] ページ (プロジェクト デザイナ) (Visual Basic)」を参照してください。

このエラーを解決するには

1. アプリケーションのスタートアップ オブジェクトを指定します。

   詳細については、「方法 : アプリケーションのスタートアップ オブジェクトを変更する」を参照してください。

2. OnCreateMainForm メソッドをオーバーライドして MainForm プロパティをスタートアップ フォームに設定します。

参照

関連項目

WindowsFormsApplicationBase
OnCreateMainForm
MainForm

概念

Visual Basic アプリケーション モデルの概要

'<name>' ディレクトリへのアクセスが拒否されました。

デバイス、ファイル、またはディスクにアクセスできません。コンピュータのアクセス許可が不十分です。

このエラーを解決するには

- 必要なアクセス許可があることを確認します。

参照

概念

アクセス許可

追加に失敗しました。重複するキーの値が指定されました。

キー値が別のキー値と同じであるため、**Add** 演算に失敗しました。キー値は一意である必要があります。

このエラーを解決するには

- キー値が一意であることを確認します。

参照

概念

エラーの種類

最後の要素以外のすべてのフィールド幅は 0 より大きくなければなりません。

エラー メッセージ
最後の要素以外のすべてのフィールド幅は 0 より大きい必要があります。最後の要素内の 0 以下のフィールド幅は最後のフィールドが可変長であることを示しています。

最後の要素以外のフィールド幅が 0 または 0 より小さく設定されているため、処理が失敗しました。最後のフィールドが可変長であることを示すために、0 と等しいか、0 よりも小さいフィールド幅を最後の要素に使用できますが、その他の要素には使用できません。

このエラーを解決するには

- フィールド幅を正しいサイズに設定します。

参照

処理手順
方法 : Visual Basic で固定幅のテキスト ファイルを読み取る
関連項目
TextFieldParser.SetFieldWidths メソッド
TextFieldParser.FieldWidths プロパティ
TextFieldParser オブジェクト

無効なイベント ログ名が指定されました。

無効な名前がイベント ログに指定されました。多くの場合、名前に無効な文字が含まれていること、ファイル名が空であること、またはファイル名が長すぎることが原因です。

このエラーを解決するには

- 指定された名前が 8 文字を超える場合は、他のイベント ログ名と競合しないことを確認します。名前が一意であるかどうかを判断する際に、最初の 8 文字だけがチェックの対象とされます。
- パスを指定している場合は、パスが正しく解析されることを確認します。
- 無効な文字が名前に含まれていないことを確認します。ファイル名に使用できない文字には、<、>、:、"、/、\、および | があります。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する
方法 : Visual Basic でファイルの名前を変更する
方法 : カスタム イベント ログを作成または削除する

単一インスタンスのスタートアップに必要なオペレーティング システムのリソースが取得できないため、予期しないエラーが発生しました。

アプリケーションに必要なオペレーティング システムのリソースが取得できませんでした。この問題には、以下のような原因が考えられます。

- アプリケーションに、指定されたオペレーティング システム オブジェクトを作成するためのアクセス許可がない。
- 共通言語ランタイムに、メモリ マップ ファイルを作成するためのアクセス許可がない。
- アプリケーションからオペレーティング システム オブジェクトにアクセスする必要があるが、別のプロセスがそれを使用している。

このエラーを解決するには

1. アプリケーションに、指定されたオペレーティング システム オブジェクトを作成するための十分なアクセス許可があることを確認します。
2. 共通言語ランタイムに、メモリ マップ ファイルを作成するための十分なアクセス許可があることを確認します。
3. コンピュータを再起動して、元のインスタンス アプリケーションに接続するために必要なリソースを使用している可能性のあるプロセスをすべて削除します。
4. エラーが発生した状況を記録して、マイクロソフト製品サポート サービスにご連絡ください。

参照

処理手順

方法 : アプリケーションのインスタンス化の動作を指定する

その他の技術情報

デバッガのロードマップ

製品のサポートとユーザー補助

別のイベント ログが、既にこの名前のソースを登録しています。

イベント ログにエントリを書き込もうとしましたが、指定のソースは別のイベント ログに登録されています。

ログにエントリを書き込むには、事前に EventLog コンポーネントのインスタンスの Source プロパティを設定しておく必要があります。これにより、システムはコンポーネントの書き込み先のイベント ログに指定のソースが登録されているかどうかを確認し、必要に応じて CreateEventSource を呼び出します。

このエラーを解決するには

1. DeleteEventSource または DeleteEventSource メソッドを使用して、最初のログに対するソースの関連付けを削除します。
2. ソースを新しいログに登録します。

参照

処理手順

方法 : イベント ソースを削除する

方法 : アプリケーションをイベント ログ エントリのソースとして追加する

関連項目

My.Application.Log オブジェクト

アプリケーション定義またはオブジェクト定義のエラーです。

アプリケーション定義エラーまたはユーザー定義エラーが発生しました。エラーを確認してください。

このエラーを解決するには

- エラーの内容を調べ、その定義場所および対処方法を検討します。

参照

概念

エラーの種類

引数 '<argument1>' は、引数 '<argument2>' の長さ以下でなければなりません。

引数の長さが有効な制限値を超えています。この制限値は 2 番目の引数の長さによって決まります。

このエラーを解決するには

1. 引数の値の長さが有効な範囲内にあることを確認します。
2. 引数が計算によって指定される場合は、計算で処理される値を確認します。

参照

概念

引数の値渡しおよび参照渡し
パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 '<argumentname>' は、多次元配列にできません。

配列引数のランクが 1 を超えていますが、この配列には 1 次元しか指定できません。

このエラーを解決するには

- 引数のランクを 1 に変更します。

参照

概念

Visual Basic の配列の概要
Visual Basic における多次元配列

引数 <argumentname> は、空の文字列または Nothing を使用できません。

引数に、空の文字列 ("") または **Nothing** 以外の値を指定する必要があります。値が正しく計算されていない可能性があります。

このエラーを解決するには

- 有効な値を引数に指定します。

参照

関連項目

Nothing (Visual Basic)
例外のトラブルシューティング : System.NullReferenceException

概念

エラーの種類

引数 '<argumentname>' を数値に変換することはできません。

文字列などの変数を数値に変換しようとしました。

このエラーを解決するには

- 有効な変換を行います。

参照

その他の技術情報

Visual Basic における型変換

引数 '<argumentname>' を型 '<typename>' に変換することはできません。

変換によって引数が無効な型に変更されようとしています。

このエラーを解決するには

- 型を有効な型の 1 つに変更します。

参照

関連項目

変換関数 (Visual Basic)

引数 '<argumentname>' を型 'Date' に変換することはできません。

変数の型を **Date** 型に変換しようとしていますが、値が有効な日付として認識されません。

このエラーを解決するには

- 有効な変換を行います。

参照

関連項目

日付型 (Date) (Visual Basic)

その他の技術情報

Visual Basic における型変換

引数 '<argumentname>' は有効な値ではありません。

引数の値が無効です。

このエラーを解決するには

1. 引数の値を確認します。
2. 引数が計算によって指定される場合は、計算で処理される値を確認します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 '<argumentname>' は配列には有効ではありません。

配列に無効な引数が含まれています。

このエラーを解決するには

- 式内の引数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。

参照

概念

引数の値渡しおよび参照渡し

その他の技術情報

Visual Basic における配列

引数 '<argumentname>' は、Nothing です。

式に null の引数があります。

このエラーを解決するには

1. 式内の引数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。
2. 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 '<argumentname>' は Nothing か、または空です。

引数の名前が **Nothing** に初期化されているか空のため、無効です。引数には特定のデータ型と有効な名前が必要です。

このエラーを解決するには

- 有効な引数名を追加します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Application.Info.DirectoryPath プロパティ

Nothing (Visual Basic)

引数 '<argumentname>' は 0 より大きいか、または -1 でなければなりません。

引数の値が 0 未満ですが、-1 ではありません。

このエラーを解決するには

1. 引数の値が有効な範囲内にあることを確認します。
2. 引数が計算によって指定される場合は、計算で処理される値を確認します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 '<argumentname>' は、1 以上でなければなりません。

引数の値が 1 未満です。

このエラーを解決するには

1. 引数の値が有効な範囲内にあることを確認します。
2. 引数が計算によって指定される場合は、計算で処理される値を確認します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 '<argumentname>' は、-1 以上でなければなりません。

引数の値が -1 未満です。

このエラーを解決するには

1. 引数の値が有効な範囲内にあることを確認します。
2. 引数が計算によって指定される場合は、計算で処理される値を確認します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 <argumentname> は 0 以上でなければなりません。

引数の値が 0 未満です。引数の値が 0 以上である必要があります。

このエラーを解決するには

1. 引数の値が 0 以上であることを確認します。
2. 引数が計算によって指定される場合は、計算で処理される値を確認します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 '<argumentname>' は 0 以上でなければなりません。

引数が無効です。0 未満の値が指定されています。

このエラーを解決するには

1. 引数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。
2. 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 '<argumentname>' は 0 より大きくなければなりません。

引数が無効です。0 以下の値が指定されています。

このエラーを解決するには

1. 式内の引数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。
2. 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 '<argumentname>' は、-32768 から 65535 までの範囲でなければなりません。

引数の値が -32,768 ～ 65,535 の範囲内にありません。

このエラーを解決するには

1. 引数の値が有効な範囲内にあることを確認します。
2. 引数が計算によって指定される場合は、計算で処理される値を確認します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数 '<argumentname>' は 0 から 99 までの範囲でなければなりません。

引数の値が 0 ～ 99 の範囲外にあるため、引数は無効です。

このエラーを解決するには

1. 式内の引数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。
2. 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

概念

引数の値渡しおよび参照渡し

引数 '<argumentname>' は 1 から 255 までの範囲でなければなりません。

引数の値が 0 ～ 255 の範囲外にあるため、引数は無効です。

このエラーを解決するには

1. 式内の引数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。
2. 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

概念

引数の値渡しおよび参照渡し

'<argumentname1>' は '<argumentname2>' 以下でなければなりません。

指定された引数の長さが、その置き換えの対象となる引数の長さを超えています。

このエラーを解決するには

- 元の引数の長さを、置き換える引数の長さと同じにするか短くします。

参照

概念

エラーの種類

引数 'Access' が有効ではありません。Append モードの有効な値は、'OpenAccess.Write' および 'OpenAccess.Default' です。

アクセス値が Append モードでは無効です。

このエラーを解決するには

- **OpenAccess.Write** か **OpenAccess.Default** の値を変更します。

参照

関連項目

OpenAccess 列挙型

引数 'Access' が有効ではありません。Input モードの有効な値は、'OpenAccess.Read' および 'OpenAccess.Default' です。

アクセス値が **Input** モードでは無効です。

このエラーを解決するには

- **OpenAccess.Read** か **OpenAccess.Default** の値を変更します。

参照

関連項目

OpenAccess 列挙型

引数 'Access' が有効ではありません。Output モードの有効な値は 'OpenAccess.Write' および 'OpenAccess.Default' です。

アクセス値が Output モードでは無効です。

このエラーを解決するには

- **OpenAccess.Write** か **OpenAccess.Default** の値を変更します。

参照

関連項目

OpenAccess 列挙型

引数 BasePath はフォルダのパスでなければなりません。

引数 *BasePath* は、フォルダへのパスで構成される必要があります。文字列の解析に誤りがあり、有効なパスと認識されない値を渡している可能性があります。

このエラーを解決するには

- *BasePath* に渡す値をチェックして、フォルダへの有効なパスになるようにします。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

BasePath

BasePath

BasePath

引数 'Life' を 0 にすることはできません。

Life の引数が無効です。この引数には、資産の耐用年数を示す **Double** 型の値を指定する必要があります。

このエラーを解決するには

- 式内の引数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。
- 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

関連項目

DDB 関数
SYD 関数
SLN 関数

概念

引数の値渡しおよび参照渡し

引数 'Conversion' が有効ではありません。

VbStrConv 列挙型の値が無効です。0 より小さいことが原因である可能性があります。列挙型の値は 0 以上である必要があります。

このエラーを解決するには

- 値が 0 以上であることを確認します。

参照

関連項目

VbStrConv 列挙型

引数には Nothing を使用できません。

引数に null 値が指定されましたが、値を指定する必要があります。

このエラーを解決するには

- オブジェクトのインスタンスを作成せずにオブジェクトを使おうとした可能性があります。このような場合は、**New** キーワードを使用してインスタンスを作成してください。
- 値が正しく処理されていることを確認します。

参照

関連項目

例外のトラブルシューティング : System.NullReferenceException

引数を 0 より小さくすることはできません。

0 未満の値を指定しましたが、この引数を 0 未満にすることはできません。

このエラーを解決するには

- 問題の値がどのように計算されているかを確認し、値が正しくなるようにします。

参照

その他の技術情報

デバッガのロードマップ

引数を空にすることはできません。

引数の 1 つが、空の文字列 ("") で定義されました。

このエラーを解決するには

- 有効な値を引数に指定します。

参照

概念

エラーの種類

引数は省略できません。(Visual Basic)

引数の数と型が合っていません。引数の数が間違っているか、または省略できない引数が省略されています。ユーザー定義のプロシージャの呼び出しで省略できるのは、プロシージャ定義で **Optional** として宣言されている引数だけです。

このエラーを解決するには

1. 必要な引数をすべて指定します。
2. 省略した引数が省略可能であることを確認します。省略可能でない場合は、その引数を指定するか、またはプロシージャ定義で **Optional** として宣言します。

参照

概念

エラーの種類

引数が有効ではありません。

無効な引数が関数またはサブルーチンに渡されました。

このエラーを解決するには

1. 引数を確認して、無効である理由を判断します。
2. 引数が計算によって指定される場合は、計算結果の値を確認します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

引数値 '<pathname>' に、パス名には有効でない文字が含まれています。

パス名に無効な文字が 1 つ以上含まれています。

このエラーを解決するには

- パス名から無効な文字を削除します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Application.Info.DirectoryPath プロパティ

引数 'Period' は 'Life' 引数以下でなければなりません。

資産の減価償却費計算の対象期間を指定する *Period* 引数の値が *Life* 引数の値を超えています。

このエラーを解決するには

- *Life* 引数と *Period* 引数の両方が同じ単位で指定されていることを確認します。たとえば、*Life* を月単位で指定する場合は、*Period* も月単位で指定します。

参照

関連項目

DDB 関数

SYD 関数

概念

引数の値渡しおよび参照渡し

引数 'Per' が有効ではありません。

Per に指定された引数が無効です。この引数の値は、支払い期間を 1 ～ *NPer* の範囲で指定する **Double** である必要があります。

このエラーを解決するには

- 引数が有効な範囲内にあることを確認します。

参照

関連項目

IPmt 関数

PPmt 関数

概念

引数の値渡しおよび参照渡し

引数 'Path' は Nothing か、または空です。

引数のパス名が **Nothing**に初期化されているか空のため、無効です。パスは特定のデータ型と有効な名前を持っている必要があります。

このエラーを解決するには

- 有効なパス名を追加します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Application.Info.DirectoryPath プロパティ

Nothing (Visual Basic)

引数 'NPer' は 0 より大きくなければなりません。

NPer 関数では、引数に 0 より大きい値を指定する必要があります。この関数は、定額の支払いを定期的に行い、利率が一定であると仮定した場合の、投資に必要な期間を示す **Double** 値を返します。

このエラーを解決するには

- 式内の引数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。
- 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

関連項目

NPer 関数

概念

引数の値渡しおよび参照渡し

配列の次元が、'VBFixedArray' 属性で指定された次元と一致しません。

配列の次元が、**VBFixedArray** 属性で指定された次元と一致しません。

このエラーを解決するには

- 配列の **Dim** ステートメント内の配列次元と、**VBFixedArray** 属性で指定された次元を一致させます。

参照

関連項目

ReDim ステートメント (Visual Basic)

VBFixedArrayAttribute クラス

概念

Visual Basic の配列の概要

オートメーション オブジェクトには既定値がありません。

指定されたオブジェクトの既定のメンバを Visual Basic によって決定できません。

このエラーを解決するには

- オブジェクトのドキュメントを参照して、そのプロパティまたはメソッドを明示的に指定します。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

オートメーション エラーです。

メソッドの実行中またはオブジェクト変数のプロパティの取得中または設定中にエラーが発生しました。このエラーは、オブジェクトを作成したアプリケーションによって報告されました。

このエラーを解決するには

1. **Err** オブジェクトのプロパティを確認し、エラーの原因と特質を調べます。
2. アクセス ステートメントの直前で **On Error Resume Next** ステートメントを使用し、アクセス ステートメントの直後のエラーを確認します。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

DLL を正しく呼び出せません。

ダイナミック リンク ライブラリ (DLL: dynamic-link library) に渡される引数は、ルーチンが受け取る引数と一致している必要があります。呼び出し規約は、引数の数、型、および順序に関係があります。プログラムが、型または数の間違った引数を渡された DLL 内のルーチンを呼び出している可能性があります。

このエラーを解決するには

1. すべての引数の型が、呼び出しているルーチンの宣言で指定されている引数の型と一致していることを確認します。
2. 呼び出しているルーチンの宣言で指定されている引数の数と同じ数の引数を渡していることを確認します。
3. DLL 内のルーチンが引数を値渡しで受け取る場合は、そのルーチンの宣言で引数に **ByVal** が指定されていることを確認します。

参照

関連項目

Call ステートメント (Visual Basic)
Declare ステートメント

概念

エラーの種類

ファイル モードが正しくありません。

ファイルの内容を操作するステートメントは、ファイルを開くときに指定したモードに対応している必要があります。以下の原因が考えられます。

- **FilePutObject** ステートメントまたは **FileGetObject** ステートメントでシーケンシャル ファイルを指定しています。
- **Print** ステートメントで、**Output** または **Append** 以外のアクセス モードで開いたファイルを指定しています。
- **Input** ステートメントで、**Input** 以外のアクセス モードで開いたファイルを指定しています。
- 読み取り専用のファイルに書き込みを行おうとしました。

このエラーを解決するには

- **FilePutObject** や **FileGetObject** で参照されているファイルが、**Random** アクセスまたは **Binary** アクセスで開いたファイルだけであることを確認します。
- **Print** で **Output** アクセス モードまたは **Append** アクセス モードで開いたファイルを指定します。そうでない場合は、別のステートメントを使ってファイルにデータを書き込むか、ファイルを適切なモードで開き直します。
- **Input** に **Input** で開いたファイルを指定します。そうでない場合は、別のステートメントを使ってファイルにデータを書き込むか、ファイルを適切なモードで開き直します。
- 読み取り専用ファイルに書き込んでいる場合は、ファイルの読み取り/書き込み属性を変更するか、そのファイルに書き込まないようにします。
- **My.Computer.FileSystem** オブジェクトで準備されている機能を使用します。

参照

処理手順

トラブルシューティング : テキスト ファイルの読み取りと書き込み

関連項目

My.Computer.FileSystem オブジェクト

ファイル名または番号が正しくありません。

指定されたファイルにアクセスしようとしたときにエラーが発生しました。このエラーでは以下の原因が考えられます。

- **FileOpen** ステートメントで指定されていない名前または番号を持つファイルを参照しているか、**FileOpen**ステートメントで指定されたファイルが後で閉じられました。
- ステートメントが、範囲外の番号を持つファイルを参照しています。
- ステートメントが、無効な名前または番号を持つファイルを参照しています。

このエラーを解決するには

1. ファイル名が **FileOpen** ステートメントで指定されていることを確認します。**FileClose** ステートメントを引数なしで呼び出した場合は、すべての開いているファイルを誤って閉じてしまった可能性があります。
2. コードがファイル番号をアルゴリズムによって生成している場合は、番号が有効であることを確認します。
3. ファイル名がオペレーティング システムの規約に準拠していることを確認します。

参照

関連項目

FileOpen 関数

概念

Visual Basic の名前付け規則

レコード長が正しくありません。

このエラーでは以下の原因が考えられます。

- **FileGet**、**FileGetObject**、**FilePut**、または **FilePutObject** の各ステートメントで指定されたレコード変数の長さと、対応する **FileOpen** ステートメントで指定された長さが異なります。
- **FilePut** ステートメントまたは **FilePutObject** ステートメント内の変数が、可変長文字列であるか、可変長文字列を含んでいます。
- **FilePut** ステートメントまたは **FilePutObject** ステートメント内の変数が、**Variant** 型であるか、この型を含んでいます。

このエラーを解決するには

1. レコード変数の型を定義するユーザー定義型の固定長変数のサイズ合計が、**FileOpen** ステートメントの *Len* 句で指定された値と同じであることを確認します。
2. **FilePut** ステートメントまたは **FilePutObject** ステートメント内の変数が可変長の文字列であるか、可変長文字列を含む場合は、その可変長文字列の長さが、**FileOpen** ステートメントの *Len* 句で指定されたレコード長よりも 2 文字以上短いことを確認します。
3. **FilePut** ステートメントまたは **FilePutObject** ステートメント内の変数が **Variant** 型であるか、この型を含む場合は、その可変長文字列の長さが、**FileOpen** ステートメントの *Len* 句で指定されたレコード長よりも 4 バイト以上短いことを確認します。

参照

関連項目

FileGet 関数
FileGetObject 関数
FilePut 関数
FilePutObject 関数

レコード番号が正しくありません。

a FileGet、**FilePut**、**FileGetObject**、または **FilePutObject** ステートメント内のレコード番号が 0 以下です。

このエラーを解決するには

- レコード番号の生成で使用される計算を確認します。レコード番号を格納する変数またはレコード番号の計算で使用される変数のスペルを確認します。モジュール内に **Option Explicit On** を記述した場合を除き、スペルの間違っている変数は暗黙的に宣言され、0 に初期化されます。

参照

関連項目

Option Explicit ステートメント (Visual Basic)

BaseLogName は、Nothing または空文字列にできません。

BaseFileName プロパティの値を **Nothing** または空の文字列にすることはできません。

BaseFileName プロパティは、ログ ファイルの基本名を指定します。

このエラーを解決するには

- **BaseFileName** プロパティを 1 文字以上の文字列に設定します。

参照

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
BaseFileName

BaudRate は 0 より大きくなければなりません。

My.Computer.Ports.OpenSerialPort メソッドに指定する *BaudRate* 引数は、0 よりも大きいことが必要です。

このエラーを解決するには

- *BaudRate* 引数の値を、正の数に変更します。

参照

関連項目

My.Computer.Ports.OpenSerialPort メソッド

'Before' と 'After' 引数を組み合わせることはできません。

引数リストに *Before* 引数と *After* 引数の両方が含まれています。これらの引数は相互に排他的です。

このエラーを解決するには

- *Before* と *After* のいずれかを削除します。

参照

関連項目

XmlNodeOrder Enumeration

型 '<type1>' の start 値、型 '<type2>' の limit 値、および型 '<type3>' の step 値を、共通の型に変換できません。

For...Next 制御構造で指定された値が異なる型を持っているため無効です。共通の型に変換できません。

このエラーを解決するには

- 1 つ以上の値の型を変更して、すべての値が共通の型に変換できるようにします。

参照

関連項目

For...Next ステートメント (Visual Basic)

概念

デバッグ (Visual Basic 6.0 ユーザー向け)

型 '<type1>' の start 値、および型 '<type2>' の step 値を共通の型に変換できません。

For...Next 制御構造で指定された値が異なる型を持っているため無効です。共通の型に変換できません。

このエラーを解決するには

- 1 つ以上の値の型を変更して、すべての値が共通の型に変換できるようにします。

参照

関連項目

For...Next ステートメント (Visual Basic)

概念

実行制御

デバッグ (Visual Basic 6.0 ユーザー向け)

型 '<type1>' の引数 '<argumentname>' を型 '<type2>' に変換できません。

変換操作で引数を互換性のない型に変換しようとしています。

このエラーを解決するには

- 引数の変換先となる型が有効であることを確認します。

参照

その他の技術情報

Visual Basic における型変換

オブジェクトが定義クラスのインスタンスではない場合、このオブジェクトに関するフレンド関数は呼び出せません。

クラスの **Friend** プロシージャを呼び出そうとしたか、**Friend** プロパティまたはメソッドにプロセス間またはスレッド間のいずれかでアクセスしようとしました。**Friend** プロシージャはクラスの外部にあるモジュールからでも呼び出すことができます。ただし、そのモジュールはクラスが定義されているプロジェクトに含まれている必要があります。

このエラーを解決するには

- クラスが定義されているプロジェクト内のモジュールからプロシージャの呼び出しまたはアクセスを行っていることを確認します。

参照

関連項目

Friend (Visual Basic)

指定された引数を使って割合を計算することはできません。

利率計算に必要な引数の一部が記述されていません。

このエラーを解決するには

1. 式内の引数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。
2. 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

関連項目

財務処理の概要

指定された引数を使って期間を計算することはできません。

NPer 関数の呼び出しで、記述されていない必須の引数があります。

このエラーを解決するには

- *Rate*、*Prnt*、および *PV* の値が関数呼び出しで指定されていることを確認します。

参照

関連項目

NPer 関数

ActiveX コンポーネントを作成できません。

デザイン時に ActiveX コントロールをフォームに配置しようとしたか、ActiveX コントロールが配置されているフォームをプロジェクトに追加しようとしましたが、レジストリ内に関連情報が見つかりませんでした。

このエラーを解決するには

- レジストリ内の情報が削除されているか、破損しています。ActiveX コントロールを再インストールするか、コントロール販売元に連絡してください。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

レジストリ ハイブを削除できません。

レジストリ ハイブの削除が行われました。ハイブとは、**HKEY_CURRENT_USER** や **HKEY_LOCAL_MACHINE** などの最上位のレジストリ キーのことです。ハイブは削除できません。

このエラーを解決するには

- 削除しようとしているレジストリ キーを調べて、パスを正しく指定していることを確認します。

参照

処理手順

方法 : Visual Basic で、レジストリ キーを削除する

関連項目

My.Computer.Registry オブジェクト

概念

一般的なレジストリ タスク

配列型が Nothing であるため、指定できません。

配列の型を判断できません。**Nothing** の値に設定されています。

このエラーを解決するには

- 配列に **Nothing** 以外の値を指定します。

参照

関連項目

Nothing (Visual Basic)

その他の技術情報

Visual Basic における配列

ディスク名は変更できません。

Rename ステートメントではファイル パスにドライブ名を指定できません。

このエラーを解決するには

- ファイル パスからドライブ文字を削除します。

参照

関連項目

Rename 関数

ファイルを TEMP に保存できません。

コンポーネントが TEMP という名前のディレクトリを見つけることができないか、TEMP ディレクトリのあるドライブまたはパーティションに、情報を保存するための十分な空き領域がありません。

このエラーを解決するには

1. "TEMP" という名前のディレクトリを作成し、TEMP 環境変数にそのパスを設定します。
2. 不要なファイルを削除してドライブに空き領域を作成するか、別のパーティションに TEMP ディレクトリを作成して TEMP 環境変数にそのパスを設定します。

参照

概念

エラーの種類

必要な一時ファイルを作成できません。

TEMP 環境変数で指定されているディレクトリのあるドライブがいっぱいであるか、TEMP 環境変数でドライブまたはディレクトリが、無効または読み取り専用になっています。

このエラーを解決するには

1. ドライブがいっぱいである場合は、ファイルを削除します。
2. TEMP 環境変数で別のドライブを指定します。
3. TEMP 環境変数で有効なドライブを指定します。
4. 現在指定されているドライブまたはディレクトリから、読み取り専用の制限を削除します。

参照

概念

エラーの種類

要求された操作を実行できません。(Visual Basic)

操作を実行できません。プロジェクトの現在の状態が無効になります。たとえば、実行中のコードをプログラムで変更しようとした可能性があります。

このエラーを解決するには

- コードの実行を停止し、必要に応じて変更を加え、操作をやり直します。

参照

概念

エラーの種類

文字列 "<string>" から型 '<typename>' へのキャストが有効ではありません。

文字列変数の型の変換が無効です。

このエラーを解決するには

- 変換の対象となっている型を確認して、有効な型に変換します。

参照

その他の技術情報

Visual Basic における型変換

型 '<typename1>' から型 '<typename2>' へのキャストが有効ではありません。

変数の型の変換が無効です。

このエラーを解決するには

- 変換の対象となっている型を確認して、有効な型に変換します。

参照

その他の技術情報

Visual Basic における型変換

クラス '<classname>' は System.Collections.ICollection インターフェイスを実装しません。

このクラスに **System.Collections.ICollection** インターフェイスが実装されていません。このインターフェイスは、すべてのコレクションのサイズ、列挙子、および同期メソッドを定義します。Visual Studio の再インストールが必要な場合があります。

このエラーを解決するには

- Visual Studio を再インストールします。

参照

関連項目

ICollection Interface

クラスがオートメーションをサポートしていないか、必要なインターフェイスをサポートしていません。

GetObject 関数呼び出しまたは **CreateObject** 関数呼び出しで指定したクラスが外部からプログラム可能なインターフェイスを公開していません。あるいは、.dll から .exe へ、または .exe から .dll へプロジェクトを変更しました。

このエラーを解決するには

1. オブジェクトを作成したアプリケーションのドキュメントを参照して、このクラスのオブジェクトでオートメーションを使用する上での制限を確認します。
2. .dll から .exe へ、または .exe から .dll へプロジェクトを変更した場合は、古い .dll または .exe を手動で登録解除する必要があります。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

クラスがローカル コンピュータに登録されていません。

このエラーは通常、ファイルの破損が原因で発生します。Visual Studio の再インストールを必要とする場合があります。

このエラーを解決するには

- Visual Studio を再インストールします。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

コード リソースが見つかりません。

コード リソース内のプロシージャの呼び出しが行われましたが、コード リソースが見つかりませんでした。

このエラーを解決するには

- リソースが使用できる状態にあり、正しく参照されていることを確認します。

参照

概念

エラーの種類

コード リソースのロック エラーです。

コード リソース内のプロシージャが呼び出されました。コード リソースは 見つかりましたが、リソースをロックしようとしたときにエラーが発生しました。コード リソースが使用中である可能性があります。

このエラーを解決するには

- リソースを調べ、エラーの原因を判断します。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

クリップボードの形式が有効ではありません。

指定されたクリップボード形式は、実行中のメソッドと互換性がありません。このエラーでは以下の原因が考えられます。

- クリップボードの **GetText** メソッドまたは **SetText** メソッドを、**vbCFText** または **vbCFLink** 以外のクリップボード形式で使用している場合。
- クリップボードの **GetData** メソッドまたは **SetData** メソッドを、**vbCFBitmap**、**vbCFDIB**、または **vbCFMetafile** 以外のクリップボード形式で使用している場合。
- **DataObject** の **GetData** メソッドまたは **SetData** メソッドを、Microsoft Windows によって登録済み形式用に予約されている範囲 (&HC000-&HFFFF) にあるクリップボード形式と一緒に使用しているが、その形式が Microsoft Windows に登録されていない場合。

このエラーを解決するには

- 無効な形式を削除し、有効な形式を指定します。

参照

概念

Clipboard オブジェクト (Visual Basic 6.0 ユーザー向け)
クリップボード : その他のデータ形式の追加

コレクション インデックスのサイズは、1 以上かつコレクションのサイズ以下の範囲でなければなりません。

指定されたインデックス値が範囲内にありません。有効範囲は、1 ～ コレクションのサイズまでです。

このエラーを解決するには

- 指定した値を有効な値に変更します。

参照

概念

Controls コレクション (Visual Basic 6.0 ユーザー向け)

リモート プロセス用のタイプ ライブラリまたはオブジェクト ライブラリへの参照は失われました。

リモート プロセスのオブジェクト ライブラリまたはタイプ ライブラリへの接続が解除されました。

このエラーを解決するには

1. **Application** オブジェクトを再起動します。
2. このヘルプ トピックを表示したエラー ダイアログ ボックスの [OK] をクリックして、[参照設定] ダイアログ ボックスを表示します。失われた参照が、その左側に "MISSING" という語と共に表示されます。
3. 失われた参照を削除します。
4. [参照設定] ダイアログ ボックスで、この **Application** オブジェクトのチェック ボックスをオンにします。

参照

概念

エラーの種類

ターゲット ディレクトリがソース ディレクトリ内にあるため、操作を完了できませんでした。

循環処理が失敗しました。循環処理とは、循環していて終了できない処理です。たとえば、オブジェクト A がオブジェクト B を継承し、そのオブジェクト B がオブジェクト A を継承しているような処理を指します。

このエラーを解決するには

- 継承を使用するときには、参照が循環していないことを確認する必要があります。

参照

概念

エラーの種類

デバッグの基礎 : ブレークポイント

内部エラーのため、メモリ情報を取得できませんでした。

My.Computer.Info オブジェクトの、いずれかのメモリ情報プロパティの呼び出しに失敗しました。

このエラーを解決するには

- **My.Computer.Info** オブジェクトのメモリ情報プロパティの呼び出しを、**Try...Catch** ブロックで囲みます。

参照

関連項目

My.Computer.Info オブジェクト

Try...Catch...Finally ステートメント (Visual Basic)

その他の技術情報

Visual Basic での例外およびエラー処理

内部エラーのため、完全なオペレーション システム名を取得できませんでした。

エラー メッセージ
内部エラーのため、完全なオペレーション システム名を取得できませんでした。現在のコンピュータ上に WMI が存在していないことが原因と考えられます。

My.Computer.Info.OSFullName プロパティの呼び出しが失敗しました。WMI (Windows Management Instrumentation) が現在のコンピュータにインストールされていないことが原因と考えられます。

このエラーを解決するには

1. **My.Computer.Info.OSFullName** プロパティの呼び出しを囲むように **Try...Catch** ブロックを追加します。
2. WMI とそのインストール方法の詳細については、http://support.microsoft.com にアクセスして "Windows Management Instrumentation Core" を検索してください。

参照

関連項目
My.Computer.Info.OSFullName プロパティ
Try...Catch...Finally ステートメント (Visual Basic)
その他の技術情報
Visual Basic での例外およびエラー処理

DataBits は 0 より大きくなければなりません。

My.Computer.Ports.OpenSerialPort メソッドの *DataBits* 引数は、0 より大きい必要があります。

このエラーを解決するには

- *DataBits* 引数の値を正数に変更します。

参照

関連項目

My.Computer.Ports.OpenSerialPort メソッド

0 で除算しました。(Visual Basic ランタイム エラー)

除数として使用されている式の値が 0 です。

このエラーを解決するには

1. 式内の変数のスペルを確認します。変数名のスペルが間違っていると、数値変数が暗黙的に作成され、0 に初期化される場合があります。
2. 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

概念

引数の値渡しおよび参照渡し
パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

0 で除算しました (Visual Basic エラー)。

除数として使用されている式の値が 0 です。

このエラーを解決するには

1. 式内の変数のスペルを確認します。変数のスペルが間違っているために、0 に初期化した数値変数が暗黙的に作成される場合があります。
2. 式内の変数に対する前の演算を確認します。特に、他のプロシージャからこのプロシージャに引数として渡される変数に注意します。

参照

概念

エラーの種類

ディスクが準備されていません。

指定されたドライブにディスクが入っていないか、ドライブのドアが開いています。

このエラーを解決するには

1. 指定したドライブを確認し、必要に応じてディスクを挿入します。
2. ドライブのドアを閉じます。

参照

概念

エラーの種類

ディスクがいっぱいです。(Visual Basic)

Print 操作や **Write** 操作、**FileClose** 操作を行ったり、要求されたファイルを作成するために充分な空き領域がありません。

このエラーを解決するには

- ファイルを別のディスクに移動するか、ファイルを削除して、ディスクに空き領域を作ります。

参照

概念

エラーの種類

'Dir' 関数は、'Pathname' 引数で最初に呼び出されなければなりません。

Dir 関数の最初の呼び出しに *PathName* 引数が記述されていません。**Dir** の最初の呼び出しには *PathName* を記述する必要がありますが、**Dir** の以降の呼び出しでは次のアイテムを取得するためのパラメータを記述する必要はありません。

このエラーを解決するには

- 関数の呼び出しに *PathName* 引数を指定します。

参照

関連項目

Dir 関数

デバイスが準備されていません。

ネットワーク接続が解除されたか、アクセスしようとしているデバイスが現在オフラインまたは存在しません。

このエラーを解決するには

1. デバイスの電源、およびコンピュータとデバイスをつなくケーブルを確認します。ネットワークを経由してプリンタにアクセスしようとしている場合は、コンピュータとプリンタの間に論理接続があることを確認します。たとえば、LPT1 とネットワーク プリンタ ID を関連付ける接続です。
2. ネットワークに再接続して、やり直します。

参照

概念

エラーの種類

デバイス I/O エラーです。

プログラムでプリンタやディスク ドライブなどのデバイスを使用しているときに、入力エラーまたは出力エラーが発生しました。

このエラーを解決するには

- デバイスが正常に動作していることを確認してから、操作をやり直します。

参照

概念

エラーの種類

ドライブ '<drivename>' が見つかりません。

ドライブが見つかりません。ネットワーク接続が解除された、または、アクセスしようとしているドライブがオフラインになっているか存在しません。

このエラーを解決するには

1. ドライブの電源を確認します。
2. コンピュータとネットワークをつないでいるケーブルが正しく接続されていることを確認します。
3. ドライブにネットワーク経由でアクセスする場合は、コンピュータとドライブの間に論理接続があることを確認します。
4. ネットワークに再接続して、やり直します。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

Encoding を Nothing に設定できません

encoding パラメータに **Nothing** が設定されていますが、有効な値を指定する必要があるため、ファイルの読み取りまたは書き込みが失敗しました。

Encoding は、ファイルへの書き込みにどのエンコーディングを使うかを決定するために使用します。既定値は UTF-8 です。

このエラーを解決するには

- *encoding* パラメータに有効な値を指定します。

参照

関連項目

My.Computer.FileSystem.ReadAllText メソッド
My.Computer.FileSystem.WriteAllText メソッド

概念

ファイル エンコーディング

その他の技術情報

Visual Basic でのファイルの読み取り
Visual Basic でのファイルへの書き込み

DLL 読み込み時のエラーです。(Visual Basic)

ダイナミック リンク ライブラリ (DLL: dynamic-link library) は、**Declare** ステートメントの **Lib** 句で指定されているライブラリです。このエラーでは以下の原因が考えられます。

- ファイルが DLL 実行可能ではありません。
- ファイルが Microsoft Windows DLL ではありません。
- DLL が、存在しない別の DLL を参照しています。
- DLL または参照先の DLL が、パスで指定されたディレクトリにありません。

このエラーを解決するには

- ファイルがソース テキスト ファイルであり、DLL が実行可能でない場合は、ファイルをコンパイルし、DLL 実行可能形式にリンクする必要があります。
- ファイルが Microsoft Windows DLL でない場合は、Microsoft Windows と等価の DLL を入手します。
- DLL が、存在しない別の DLL を参照している場合は、参照される DLL を入手し、使用できる状態にします。
- DLL または参照先の DLL がパスで指定されたディレクトリにない場合は、参照先のディレクトリに DLL を移動します。

参照

関連項目

Declare ステートメント

エラー番号は 0 から 65535 の範囲でなければなりません。

エラー番号が 0 ～ 65535 の間にありません。

このエラーを解決するには

- 番号が有効な範囲内にあることを確認します。

参照

関連項目

Err オブジェクト (Visual Basic)
Number プロパティ (Err オブジェクト)

式 '<name>' はプロシージャではありませんが、プロシージャ呼び出しの対象として発生します。

プロシージャではない式に対してプロシージャ呼び出しを行いました。

このエラーを解決するには

1. プロシージャ名のスペルが正しいことを確認します。
2. <*name*> によって呼び出されるプロシージャを指定します。

参照

概念

Visual Basic におけるプロシージャ

機能がまだ実装されていません。

Visual Basic 2005 で現在実装されていない機能にアクセスしようとしました。

このエラーを解決するには

- 参照を削除します。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

式が複雑すぎます。

浮動小数点式で、入れ子になっている部分式が多すぎます。

このエラーを解決するには

- エラーが発生しない程度に式を分割します。

参照

概念

Visual Basic の演算子および式

型 '<typename>' のフィールド '<fieldname>' は 'ReadOnly' です。

ReadOnly として宣言されているフィールドに書き込もうとしました。

このエラーを解決するには

- フィールドを変更して **ReadOnly** の宣言を無効にします。

参照

関連項目

ReadOnly (Visual Basic)

ファイル '<filename>' は開いているため、削除できません。

開いているファイルを削除しようとしました。

このエラーを解決するには

- ファイルを閉じてから削除します。

参照

関連項目

FileClose 関数
My.Computer.FileSystem.DeleteFile メソッド
My.Computer.FileSystem オブジェクト

ファイル '<filename>' は書込み禁止にされています。

書き込みを行おうとしているファイルは書き込み保護されているため変更できません。

このエラーを解決するには

- ファイルのアクセス許可を読み取り/書き込みに変更するか、別のファイルに書き込みます。

参照

処理手順

トラブルシューティング : テキスト ファイルの読み取りと書き込み

ファイル '<filename>' が見つかりません。

ファイル名が間違っているか、ファイル自体が存在しません。

このエラーを解決するには

- ファイル名が正しく、ファイルが存在することを確認します。

参照

関連項目

My.Computer.FileSystem オブジェクト

ファイルは既に存在します。

このエラーが発生するのは、新しいファイルに既存のファイルの名前を付けたり、現在読み込んであるプロジェクトを Save As コマンドで既に存在する名前で保存したりした場合です。

このエラーを解決するには

- 名前を確認し、競合のない名前に変更するか、不要である場合は既存のファイルまたはプロジェクトを削除します。

参照

概念

エラーの種類

ファイルは既に開かれています。

先にファイルを閉じてから、別の **FileOpen** または他の操作を実行する必要のある場合があります。このエラーでは以下の原因が考えられます。

- 既に開いているファイルに対して、シーケンシャル出力モードの **FileOpen** 操作が実行されました。
- ステートメントが、開いているファイルを参照しています。

このエラーを解決するには

- ファイルを閉じてからステートメントを実行します。

参照

関連項目

FileOpen 関数

ファイル形式が有効ではありません。

ファイルをコンポーネント内に読み込もうとしましたが、ファイル内のデータの形式がコンポーネントと互換性がありませんでした。または、コンポーネント データをファイルに保存しようとしましたが、データ形式がファイル形式と互換性がありませんでした。

このエラーを解決するには

1. コンポーネントのドキュメントを参照し、ディスク ファイル データの正しい形式を確認します。
2. コンポーネントのドキュメントを参照し、形式間の変換をサポートしているかどうかを確認します。

参照

概念

エラーの種類

型 '<typename>' のフィールド '<fieldname>' を含む構造体のファイル入出力 (I/O) は有効ではありません。

ファイルに書き込もうとしている構造体には、サポートされていない型が含まれています。

このエラーを解決するには

- 構造体を調べてサポートされていない型を確認し、有効な型に変更します。

参照

関連項目

My.Computer.FileSystem.WriteAllBytes メソッド

概念

エラーの種類

型 '<typename>' のファイル入出力 (I/O) は有効ではありません。

サポートされていない型をファイルに書き込もうとしています。

このエラーを解決するには

- 構造体を調べてサポートされていない型を確認し、有効な型に変更します。

参照

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.WriteAllBytes メソッド

ファイルが存在しない場合、ファイル情報をクエリすることはできません。

存在しないファイルに関する情報を取得するために、FileInfo オブジェクトの照会が行われました。多くの場合、これはファイル名の指定を間違えたことが原因です。

このエラーを解決するには

- ファイル名を正しく指定していることを確認します。

参照

処理手順

方法 : Visual Basic でファイルについての情報を取得する
方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.FileSystem オブジェクト

書き込みのためにファイルが開いている間は、ファイル情報をクエリすることはできません。

現在、データを書き込むために開いているファイルに関する情報の取得が行われました。

このエラーを解決するには

- ファイルを閉じてから、もう一度処理を行ってください。

参照

処理手順

方法 : Visual Basic でファイルについての情報を取得する

関連項目

FileInfo

FileIOPermission

バイト配列に読み取るにはファイルが大きすぎます。

バイト配列に読み込もうとしているファイルのサイズが 4 GB を超えています。**My.Computer.FileSystem.ReadAllBytes** メソッドではこのサイズよりも大きいファイルを読み込むことができません。

このエラーを解決するには

- StreamReader を使ってファイルを読み込んでください。詳細については、「.NET Framework のファイル I/O とファイル システムの基礎」を参照してください。

参照

処理手順

方法 : StreamReader を使用してファイルからテキストを読み取る (Visual Basic)

関連項目

My.Computer.FileSystem.ReadAllBytes メソッド

StreamReader

その他の技術情報

Visual Basic におけるファイル アクセス

オートメーションの操作中にファイル名またはクラス名を見つけられませんでした。(Visual Basic)

GetObject 関数の呼び出しで、ファイル名またはクラスについて指定された名前が見つかりませんでした。

このエラーを解決するには

- 名前を確認して、やり直します。class パラメータに使用した名前が、システムに登録されている名前と一致していることを確認します。

参照

概念

エラーの種類

ファイルが見つかりません。(Visual Basic ランタイム エラー)

指定された場所にファイルがありませんでした。このエラーでは以下の原因が考えられます。

- ステートメントが、存在しないファイルを参照しています。
- ダイナミック リンク ライブラリ (DLL: dynamic-link library) 内のプロシージャを呼び出していますが、**Declare** ステートメントの **Lib** 句で指定されたライブラリが見つかりません。
- 存在しないプロジェクトを開こうとしたか、存在しないテキスト ファイルを読み込もうとしました。

このエラーを解決するには

- ファイル名のスペルとパスの指定を確認します。

参照

関連項目

Declare ステートメント

FileName で指定されたファイルは、FileEncoding で指定されたエンコーディングを使用しません。

ファイルの読み込みが行われましたが、指定されたエンコーディングはファイルに使用されていません。

このエラーを解決するには

- ファイルのエンコーディングを正しく指定しているか確認します。

参照

処理手順

方法 : Visual Basic でテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem.ReadAllText メソッド

My.Computer.FileSystem オブジェクト

概念

ファイル エンコーディング

FileName で指定されたファイルは有効な XML ファイルではありません。

指定したファイル名が有効な XML ファイルではありません。XML ドキュメントの許容可能な構造と内容を指定するには、文書型定義 (DTD: Document Type Definition)、Microsoft XML-Data Reduced (XDR) スキーマ、または XML スキーマ定義言語 (XSD: XML Schema definition language) スキーマを使用します。XSD スキーマは、.NET Framework 内で XML 文法を指定するための好ましい方法です。

メモ :

以前のバージョンの Visual Studio 2005 では、型指定されたデータセットや XML スキーマを作成するときに XML デザイナを使用していました。現在でも、XML スキーマ ファイルを作成および編集するときには XML デザイナを使用できます。しかし Visual Studio 2005 では、型指定されたデータ セットを作成および編集するときにはデータセット デザイナを使用します。詳細については、「データセット デザイナ」を参照してください。

このエラーを解決するには

- 正しいファイル名を指定していることを確認します。
- 目的の XML ファイルを XML デザイナに読み込んで、このファイルに有効な XML が含まれていることを確認します。[XML] メニューの [XML データの整合性チェック] をクリックします。エラーがタスク一覧に表示されます。
- XML ファイルに XML スキーマが関連付けられている場合は、要素が定義された構造どおりに出現し、各要素の内容がスキーマで指定されている宣言されたデータ型に従っていることを確認します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

System.Xml

ファイルの open モードは有効な値に設定されていませんでした。

ファイルのオープン モードに指定された値が無効です。OpenMode 列挙に指定できる有効な値は、次の表に示すとおりです。

値	モード
1	**OpenMode.Input**
2	**OpenMode.Output**
4	**OpenMode.Random**
8	**OpenMode.Append**
32	**OpenMode.Binary**

このエラーを解決するには

- ファイルのオープン モードに指定した値を確認します。

参照

関連項目

OpenMode 列挙型
My.Computer.FileSystem オブジェクト

その他の技術情報

Visual Basic でのファイルの読み取り
Visual Basic でのファイルへの書き込み

For ループが初期化されていません。(Visual Basic)

For...Next ループ内にジャンプしました。

このエラーを解決するには

- ループ内へのジャンプを削除します。

参照

処理手順

方法 : ステートメントにラベル付けする

関連項目

For Each...Next ステートメント (Visual Basic)

形式がリソース ファイルで有効ではありません。

プロジェクト内のリソース ファイルが Windows リソース ファイルの標準ファイル形式に準拠していないか、リソース ファイルと Visual Basic プロジェクトの間で不一致があります。

このエラーを解決するには

1. リソース ソースとリソース ファイルを作成し直し、Windows Software Development Kit (SDK) で定義されている構文に準拠します。
2. 無効なリソース ファイルをプロジェクトから削除します。

参照

概念

エラーの種類

Get はサポートされていません。書き込み専用のプロパティです。

プロパティは書き込み専用のため読み取ることができません。

このエラーを解決するには

1. プロパティを調べ、どのような条件下で設定できるかを確認します。
2. プロパティへの参照を削除します。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

Get は実行時にはサポートされません。

デザイン時にしかアクセスできないプロパティを実行時に読み取ろうとしました。

このエラーを解決するには

1. プロパティを調べ、どのような条件下で設定できるかを確認します。
2. プロパティへの参照を削除します。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

別のコンストラクタを呼び出している間は、作成中のオブジェクトへの暗黙的な参照は有効ではありません。

オブジェクトのコンストラクタがオブジェクトを作成する前に、そのオブジェクトのメンバが参照されました。

Error ID: BC31096

このエラーを解決するには

- コンストラクタを別のコンストラクタから呼び出すときは、**MyBase**、**MyClass**、または **Me** は使用しないでください。

参照

関連項目

コンストラクタとデストラクタの使用方法

概念

オブジェクトの有効期間 : オブジェクトの作成と破棄

システムの時刻を設定するためのセキュリティ アクセス許可が十分ではありません。

システムの時刻を設定するための十分なアクセス許可がありません。

このエラーを解決するには

- セキュリティ アクセス許可を変更します。

参照

関連項目

System.Security.Permissions Namespace

概念

セキュリティ ポリシー レベル

その他の技術情報

.NET Framework 構成ツール (Mscorcfg.msc) を使用したセキュリティ ポリシーの設定

システムの日付を設定するためのセキュリティ アクセス許可が十分ではありません。

システムの日付を設定するための十分なアクセス許可がありません。

このエラーを解決するには

- システムの日付を設定できるアクセス許可に変更します。

参照

関連項目

System.Security.Permissions

概念

セキュリティ ポリシー レベル

その他の技術情報

.NET Framework 構成ツール (Mscorcfg.msc) を使用したセキュリティ ポリシーの設定

ファイルにこれ以上データがありません。

Input ステートメントで空のファイルまたはすべてのデータが使用されているファイルからデータを読み取っているか、バイナリ アクセスで開いたファイルで **EOF** 関数が使用されました。

このエラーを解決するには

1. **Input** ステートメントの直前に **EOF** 関数を使用して、ファイルの終わりを検出します。
2. ファイルをバイナリ アクセスで開いている場合は、**Seek** および **Loc** を使用します。

参照

関連項目

Input 関数
EOF 関数
Seek 関数
Loc 関数

内部エラーです。(Visual Basic)

Visual Basic で内部誤動作が発生しました。

このエラーを解決するには

- このエラーが **Error** ステートメントまたは **Raise** メソッドによって生成されたものではないことを確認します。**Error** ステートメントまたは **Raise** メソッドによって生成されたエラーでない場合は、マイクロソフト製品サポート サービスに連絡して、このエラー メッセージが表示されたときの状況をお伝えください。

参照

関連項目

On Error ステートメント (Visual Basic)
Raise メソッド (Err オブジェクト)

<場所> で内部エラーが発生しました。

内部エラーが発生しました。エラーが発生した行は、エラー メッセージに示されています。

このエラーを解決するには

- このエラーが **Error** ステートメントまたは **Raise** メソッドによって生成されていないことを確認してください。確認できましたらマイクロソフト製品サポート サービス に連絡して、メッセージが表示された状況をご報告ください。

参照

その他の技術情報

デバッガのロードマップ

Microsoft Visual Basic ランタイムで内部エラーが発生しました。

Microsoft Visual Basic ランタイムで内部エラーが発生しました。

このエラーを解決するには

- エラーが発生した状況を記録して、マイクロソフト製品サポート サービスにご連絡ください。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

無効なパターン文字列です。

検索の **Like** 演算で指定されているパターン文字列が無効です。

このエラーを解決するには

1. リスト式の有効な文字を確認します。
2. パターン範囲の指定で、`[a-z]` のように、範囲の開始文字が終了文字より小さいことを確認します。
3. パターン範囲の指定で、`[a--z]` のように、複数のハイフンが続けて指定されていないことを確認します。
4. パターン範囲を右角かっこで終了します。

参照

関連項目

Like 演算子

サブキーが含まれているため、キーを削除できません

サブキーがあるため、キーを削除できません。

このエラーを解決するには

- 指定したキーのサブキーを確認し削除してから、この処理を実行します。

参照

処理手順

方法 : Visual Basic で、レジストリ キーを削除する

トラブルシューティング : レジストリの操作

関連項目

DeleteSubKey

概念

My を使用したレジストリからの読み取りとレジストリへの書き込み

'<name>' が実行時連結式の結果である場合、'<typename>' 型の値のフィールドへの実行時連結の割り当ては有効ではありません。

遅延バインディングによる代入は無効です。

このエラーを解決するには

- 事前バインディングによる代入を行います。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

引数 '<argumentname>' の長さは 0 より大きくなければなりません。

引数の値が 0 以下です。

このエラーを解決するには

- 引数に 0 より大きい値を指定します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

現在の Delimiter を使用して、行 <数値> を解析できません。

指定された行が、指定された以外の区切り記号を使用しているため、解析できません。

このエラーを解決するには

- 行を正しく解析できるように **Delimiters** を修正するか、その行を処理するための例外処理コードを追加します。

参照

処理手順

方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem.OpenTextFieldParser メソッド
TextFieldParser オブジェクト
TextFieldParser.Delimiters プロパティ
TextFieldParser.SetDelimiters メソッド

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

現在の FieldWidth を使用して、行 <数値> を解析できません。

指定された行は、そのフィールドの幅が指定よりも大きいため、解析できません。

このエラーを解決するには

- 行を正しく解析できるように **FieldWidths** を修正するか、その行を処理するための例外処理コードを追加します。

参照

処理手順

方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem.OpenTextFieldParser メソッド
TextFieldParser オブジェクト
TextFieldParser.FieldWidths プロパティ
TextFieldParser.SetFieldWidths メソッド

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

最大行サイズを超えているため、行 <数値> を解析できません。

最大行サイズを超えているため、行を読み込むことができません。これは、ほとんどの場合は、ファイル内に始まりの引用符があるのに終わりの引用符がないことを意味しています。

このエラーを解決するには

- ファイル内に終わりの引用符を挿入します。

参照

関連項目

TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

ロケール ID '<name>' はこのシステム上でサポートされていません。

VbStrConv を使おうとしていますが、指定したロケールがこのシステムでサポートされていません。

このエラーを解決するには

- このロケールがサポートされるようにコンピュータをアップグレードするか、このロケールをサポートするコンピュータでアプリケーションを実行します。

参照

関連項目

System.Globalization Namespace

概念

.NET Framework ベースの国際対応アプリケーションの概要

型 '<typename>' のループ コントロール変数は、'System.IComparable' インターフェイスを実装しません。

変数の型が、**System.IComparable** インターフェイスを実装しない型になっています。**System.IComparable** は、値型またはクラスで実装する必要があります。

このエラーを解決するには

- 変数が値型であることを確認します。

参照

関連項目

IComparable Interface

COM クラスから派生したマネージ クラスは、遅延バインディングされた呼び出しはできません。

COM クラスから派生するマネージ クラスに対して遅延バインディングによる呼び出しを行いましたが、そのような呼び出しはサポートされていません。

このエラーを解決するには

- 事前バインディングによる呼び出しを行います。

参照

概念

エラーの種類

メソッド '<methodname>' を <number> 引数で呼び出すことはできません。

メソッドの呼び出しで指定された引数の数が不正です。

このエラーを解決するには

1. 入力した引数の数を確認します。
2. 呼び出し対象メソッドのパラメータの数と比較します。
3. 2 つの数字が一致していない場合は、引数の数を調整します。

参照

概念

引数の値渡しおよび参照渡し

メソッド '<methodname>' には '<parametername>' という名前のパラメータはありません。

指定された名前付き引数が、対象となるメソッドの引数と一致しません。

このエラーを解決するには

- 名前付き引数が正しく指定してあることを確認します。

参照

概念

引数の値渡しおよび参照渡し

メソッドまたはデータ メンバが見つかりません。

アクセスしようとしたメソッドまたはデータ メンバが見つかりませんでした。

このエラーを解決するには

- メソッドまたはデータ メンバの名前のスペルが正しいことを確認します。

参照

概念

エラーの種類

My.Application.Log は空きディスク領域を確定できません。

FileLogTraceListener クラスは、ディスクの空き容量の合計サイズを調べることができませんでした。

このエラーを解決するには

- エラーが発生した状況を記録して、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

関連項目

FileLogTraceListener

その他の技術情報

製品のサポート

製品のサポートとユーザー補助

名前付き引数 '<argumentname>' が複数回指定されました。

名前付き引数が複数回指定されました。

このエラーを解決するには

- 1 つの指定を除いてすべての名前付き引数を削除します。

参照

概念

位置と名前による引数渡し

名前付き引数が見つかりませんでした (Visual Basic)

名前付き引数は、プロシージャ定義に記述されていないと、プロシージャの呼び出しで使用できません。名前付き引数を指定して呼び出したプロシージャが、その名前で引数を受け取るように定義されていません。

このエラーを解決するには

- 引数名のスペルを確認します。

参照

概念

エラーの種類

名前付き引数を ParamArray 引数と同じにすることはできません。

パラメータ配列は値渡しで渡す必要があります。

このエラーを解決するには

- 名前付き引数ではなく、**ByVal** を使用して引数を渡します。

参照

概念

パラメータ配列
引数の値渡しおよび参照渡し
位置と名前による引数渡し

プロパティ配列のインデックスが必要です。

このプロパティ値は、単一の値ではなく配列で構成されています。アクセス先のプロパティ配列のインデックスが指定されていません。

このエラーを解決するには

- コンポーネントのドキュメントを参照し、配列のインデックスの範囲を調べます。プロパティ アクセス ステートメントで適切なインデックスを指定します。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

下位変換せずにこれらの引数を伴って呼び出される、オーバーロードされたアクセス可能な '<methodname>' はありません

オーバーロードされたメソッドの呼び出しで、縮小変換をしない限り、入力された引数リストと一致するメソッドがありません。

このエラーを解決するには

1. **Option Strict Off** を指定します。
2. オーバーロードされたメソッドのいずれかのシグネチャに一致するように引数を変更します。

参照

概念

引数の値渡しおよび参照渡し
拡大変換と縮小変換

下位変換せずにこれらの引数を伴って呼び出される、オーバーロードされたアクセス可能な '<methodname>' はありません : <list>

オーバーロードされたメソッドが呼び出されましたが、縮小変換をしない限り、入力された引数リストと一致するメソッドがありません。

このエラーを解決するには

1. **Option Strict Off** を指定します。
2. オーバーロードされたメソッドのシグネチャに一致するように引数を変更します。

参照

概念

引数の値渡しおよび参照渡し
拡大変換と縮小変換

上位変換せずにこれらの引数を伴って呼び出される、オーバーロードされたアクセス可能な '<methodname>' はありません : <list>

オーバーロードされたメソッドを呼び出して、拡大変換をしない限り、入力された引数リストに一致するメソッドがありません。

このエラーを解決するには

- **Option Strict Off** を指定します。
- オーバーロードされたメソッドのいずれかのシグネチャに一致するように引数を変更します。

参照

関連項目

Option Strict ステートメント

概念

拡大変換と縮小変換

これらの引数で呼び出される、オーバーロードされたアクセス可能な '<methodname>' はありません : <list>

オーバーロードされたメソッドを呼び出して、指定された引数リストに一致するメソッドがありません。

このエラーを解決するには

- 引数リストと、呼び出し対象のオーバーロードされたメソッドを比較します。
- オーバーロードされたメソッドとシグネチャが一致するように、引数リストを調整します。

参照

概念

引数の値渡しおよび参照渡し

型 '<typename>' の既定メンバが見つかりません。

遅延バインディングによる、インデックス **Get** 呼び出しまたはインデックス **Set** 呼び出しが試みられました。

このエラーを解決するには

- 実行する呼び出しに応じて、**Get** または **Set** に対応する既定のプロパティがオブジェクトに存在することを確認します。

参照

関連項目

Get ステートメント
Set ステートメント (Visual Basic)

'<filename>' に一致するファイルが見つかりません。

ファイル名が間違っているか、ファイル自体が存在しません。

このエラーを解決するには

- ファイルの名前が正しいことを確認します。

参照

関連項目

My.Computer.FileSystem オブジェクト

パラメータ '<parametername>' の '<typename>' 型の引数をメソッド '<methodname>' で受け入れることはできません。

メソッドの呼び出しで、無効な型の引数が指定されています。

このエラーを解決するには

1. 引数の型を確認します。
2. 引数の型をメソッドのパラメータの型と突き合わせて、両者を対応させます。必要に応じて、型を変更します。

参照

概念

引数の値渡しおよび参照渡し

マウスが存在しません。

My.Computer.Mouse オブジェクトのいずれかのプロパティが呼び出されましたが、コンピュータにマウスが接続されていないか、またはマウス ポートがありません。

このエラーを解決するには

- **My.Computer.Mouse** オブジェクトのプロパティの呼び出しを、**Try...Catch** ブロックで囲みます。

  または

- コンピュータにマウスを取り付けます。

参照

関連項目

My.Computer.Mouse オブジェクト

Try...Catch...Finally ステートメント (Visual Basic)

その他の技術情報

Visual Basic での例外およびエラー処理

マウスのホイールが存在しません。

My.Computer.Mouse.WheelScrollLines プロパティが呼び出されましたが、マウスにスクロール ホイールがありません。

このエラーを解決するには

- **My.Computer.Mouse.WheelScrollLines** プロパティを呼び出す前に **My.Computer.Mouse.WheelExists** プロパティを調べて、マウスにスクロール ホイールがあることを確認します。

  または

- スクロール ホイールのあるマウスをコンピュータにインストールします。

参照

関連項目

My.Computer.Mouse.WheelScrollLines プロパティ
My.Computer.Mouse.WheelExists プロパティ

その他の技術情報

Visual Basic での例外およびエラー処理

NumberOfChars は 0 より大きくなければなりません。

TextFieldParser オブジェクトの **PeekChars** メソッドを使用するとき、*NumberOfChars* には 0 よりも大きい値を指定する必要があります。

このエラーを解決するには

- *NumberOfChars* を 0 よりも大きい値に変更します。

参照

処理手順

方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem.OpenTextFieldParser メソッド

TextFieldParser.PeekChars メソッド

TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

オブジェクトは現在のロケールの設定をサポートしていません。(Visual Basic)

アクセスしようとしたオブジェクトは、現在のプロジェクトのロケール設定をサポートしていません。

このエラーを解決するには

- オブジェクトでサポートされるロケール設定を確認します。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

オブジェクトは名前付き引数をサポートしていません。

アクセスしようとしたオブジェクトのメソッドは、名前付き引数をサポートしていません。

このエラーを解決するには

- このオブジェクトのメソッドにアクセスするときは、引数を位置で指定します。引数の位置と型の詳細については、オブジェクトのドキュメントを参照してください。

参照

概念

エラーの種類

このオブジェクトではサポートされていない操作です。(Visual Basic)

このオブジェクトでサポートされていないメソッドまたはプロパティを参照しました。

このエラーを解決するには

1. このオブジェクトの詳細については、オブジェクトのドキュメントを参照してください。
2. プロパティ名とメソッド名のスペルが正しいことを確認します。

参照

概念

エラーの種類

オブジェクトでサポートされていないプロパティまたはメソッドです。(Visual Basic)

このオートメーション オブジェクトには存在しないメソッドまたはプロパティを指定しました。

このエラーを解決するには

- オブジェクトのドキュメントでオブジェクトの詳細について参照し、プロパティとメソッドのスペルを確認します。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

オブジェクトまたはクラスがこのイベント セットをサポートしていません。

コンポーネントで **WithEvents** 変数を使おうとしましたが、そのコンポーネントは指定されたイベント セットのイベント ソースとして使用できません。たとえば、オブジェクトのイベントをシンクしてから、最初のオブジェクトを **Implements** で実装する別のオブジェクトを作成する場合があります。実装したオブジェクトからこのイベントをシンクできると考えることもできますが、常にシンクできるとは限りません。**Implements** では、メソッドおよびプロパティ用のインターフェイスのみが実装されます。**WithEvents** は、プライベートな **UserControls** をサポートしていません。**ObjectEvent** を発生させるために必要な型情報がランタイム時には利用できないためです。

このエラーを解決するには

- イベントの発生元ではないコンポーネントのイベントはシンクできません。

参照

関連項目

WithEvents
Implements ステートメント

オブジェクトが必要です。(Visual Basic)

プロパティまたはメソッドへの参照では、通常、オブジェクト修飾子を明示的に指定する必要があります。これが通常の場合です。

このエラーを解決するには

1. オブジェクトのプロパティまたはメソッドを参照する際に、有効なオブジェクト修飾子が指定してあることを確認します。オブジェクト修飾子を指定していなかった場合は指定します。
2. オブジェクト修飾子のスペルを確認し、プログラム内の参照位置からオブジェクトを参照できることを確認します。
3. ホスト アプリケーションの File Open コマンドにパスを指定している場合は、コマンドの引数が正しいことを確認します。
4. オブジェクトのドキュメントを参照して、このアクションが有効であることを確認します。

参照

概念

エラーの種類

オブジェクト変数または With ブロック変数が設定されていません。

無効な変数が参照されています。オブジェクト変数を作成するには、オブジェクト変数を宣言し、**Set** ステートメントを使用して有効な参照をオブジェクト変数に代入します。同様に、**With...End With** ブロックは **With** ステートメントのエントリ ポイントを実行して初期化する必要があります。

このエラーを解決するには

1. オブジェクト変数が有効なオブジェクトを参照していることを確認し、問題があればオブジェクトに対する参照を指定し直します。
2. **Nothing** に設定されたオブジェクト変数を使用していないことを確認します。
3. オブジェクトが記述されているオブジェクト ライブラリが [**Add References**] ダイアログ ボックスで選択されていることを確認します。
4. **With** ブロックが **With** ステートメントのエントリ ポイントを実行して初期化されていることを確認します。

参照

関連項目

With...End With ステートメント (Visual Basic)

概念

オブジェクト変数の宣言

ターゲット パスに 1 つ以上のフォルダが存在しません。

ターゲットのパスに指定された 1 つ以上のディレクトリが存在しません。特に相対パスが指定された場合に、パスが正しく解析されないことが原因である可能性があります。

このエラーを解決するには

- 正しいターゲットのパスを指定していることを確認します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

カスタム ログ名の最初の 8 文字が有効です。

イベント ログ名の一意性を調べるときには、最初の 8 文字だけが考慮されます。最初の 8 文字が同じイベント ログが複数ある場合は、衝突する可能性があります。

このエラーを解決するには

- イベント ログに、最初の 8 文字が一意になる名前を付けます。

参照

処理手順

方法 : カスタム イベント ログを作成または削除する

その他の技術情報

イベント ログの管理

演算子が、型 '<typename>' に対して有効ではありません。

演算子の 1 つがこの型に対して有効ではありません。

このエラーを解決するには

- 扱っている型に対して適切な演算子を使用します。

参照

その他の技術情報

演算子 (Visual Basic)

演算子が、'<name1>' および '<name2>' に対して有効ではありません。

演算子の 1 つが演算対象の型に対して有効ではありません。

このエラーを解決するには

- 扱っている型に対して適切な演算子を使用します。

参照

その他の技術情報

演算子 (Visual Basic)

オーバーフローしました。(Visual Basic ランタイム エラー)

代入対象の制限を超える値を代入しようとしたため、オーバーフローが発生しました。

このエラーを解決するには

1. 代入、計算、およびデータ型変換の結果がその型で使用できる変数の範囲を超えていないかどうかを確認し、超えていた場合は、より広い範囲の値を格納できる型の変数に値を代入します。
2. プロパティへの代入がプロパティの許容範囲内に収まっていることを確認します。
3. 強制的に整数型に変換される計算で使用されている数が、整数型の範囲を超える結果を持たないことを確認します。

参照

概念

エラーの種類

文字列スペースが不足しています。(Visual Basic)

Visual Basic 2005 では、非常に大きな文字列を使用できます。ただし、他のプログラムの要求や文字列の操作方法によっては、このエラーが発生する場合があります。

このエラーを解決するには

1. 評価時に文字列を一時的に作成する必要がある式でこのエラーが発生していないかどうかを確認します。
2. 不要なアプリケーションをメモリから削除して領域を拡張します。

参照

関連項目

文字列操作の概要

概念

エラーの種類

スタック領域が不足しています。(Visual Basic)

スタックとはメモリ内にある作業領域のことで、スタックのサイズは、実行しているプログラムの要求に応じて動的に増減します。スタックの制限を超えました。

このエラーを解決するには

1. プロシージャをあまり多くのレベルにわたって入れ子にしないようにします。
2. 再帰的に呼び出し可能なプロシージャが正常に終了していることを確認します。
3. ローカル変数に必要な領域が足りない場合は、一部の変数をモジュール レベルで宣言してみてください。また、**Property**、**Sub**、または **Function** の各キーワードの前に **Static** を付けて、プロシージャのすべての変数を静的変数として宣言することもできます。この他、**Static** ステートメントを使ってプロシージャ内で個別に静的変数を宣言することもできます。
4. 固定長文字列のいくつかを可変長文字列で定義し直します。固定長文字列では、可変長文字列よりもスタック領域が多く消費されます。スタック領域を必要としないモジュール レベルで文字列を定義することもできます。
5. **Calls** ダイアログ ボックスを使ってスタック上でアクティブなプロシージャを表示し、入れ子になった **DoEvents** 関数呼び出しの数を確認します。
6. 既にスタック上にあるイベント プロシージャを呼び出すイベントを発生させることによって、"イベント連鎖" が発生していないことを確認します。イベント連鎖は、終了しない再帰プロシージャ呼び出しに似ていますが、コードで明示的に呼び出されるのではなく Visual Basic によって呼び出されるため、再帰プロシージャ呼び出しに比べてわかりにくくなります。**Calls** ダイアログ ボックスを使って、スタック上でアクティブなプロシージャを確認します。

参照

概念

[メモリ] ウィンドウ

メモリが不足しています。(Visual Basic ランタイム エラー)

メモリが不足しています。

このエラーを解決するには

1. 開いているアプリケーション、ドキュメント、ソース ファイルなどのうち、不要なものを閉じます。
2. 大きなモジュールやプロシージャを分割します。
3. メモリ常駐プログラムを除去します。
4. 不要なデバイス ドライバを除去します。
5. パブリック変数の数を減らします。

参照

概念

エラーの種類

序数が有効ではありません。

ダイナミック リンク ライブラリ (DLL: dynamic-link library) の呼び出しで、**#num** 構文を使用して、プロシージャ名の代わりに数値を使用するように指定されています。このエラーでは以下の原因が考えられます。

- **#num** 式の序数への変換に失敗しました。
- 指定した **#num** は、DLL 内の関数を指定しません。
- タイプ ライブラリに無効な宣言があるため、無効な除数が内部で使用されています。

このエラーを解決するには

1. 式が有効な数値を表していることを確認するか、プロシージャを名前で呼び出します。
2. **#num** が DLL 内の有効な関数を指定していることを確認します。
3. コードをコメント アウトすることによって、問題を起こしているプロシージャ呼び出しを特定します。プロシージャの **Declare** ステートメントを作成し、問題をタイプ ライブラリ販売元に報告します。

参照

関連項目

Declare ステートメント

パス '<pathname>' が見つかりませんでした。

ファイル アクセス操作またはディスク アクセス操作で、指定されたパスが見つかりません。ファイルのパスには、ファイルが格納されているドライブ、ディレクトリ、およびサブディレクトリを含める必要があります。相対パスと絶対パスのどちらでも指定できます。

このエラーを解決するには

- パスに必要な要素が含まれていることを確認します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.FileSystem オブジェクト

パスが見つかりません。

ファイル アクセス操作またはディスク アクセス操作で、指定されているパスが見つかりませんでした。ファイルへのパスには、ファイルがある場所のドライブ、ディレクトリ、およびサブディレクトリを指定します。相対パスと絶対パスのどちらでも指定できます。

このエラーを解決するには

- パスを確認し、指定し直します。

参照

概念

エラーの種類

パス/ファイル アクセス エラー

ファイル アクセス操作またはディスク アクセス操作で、オペレーティング システムがパスとファイル名を関連付けることができませんでした。

このエラーを解決するには

1. ファイル指定の形式が正しいことを確認します。ファイル名には、絶対パスと相対パスのどちらでも指定できます。絶対パスでは、まずドライブ名を指定し (パスが別のドライブにある場合)、続いてルートからファイルまでのパスを明示的に指定します。絶対パスを指定しない場合、現在のドライブおよびディレクトリを基準にした相対パスになります。
2. 保存しようとしているファイルが既存の読み取り専用ファイルを置き換えようとしていないかどうかを確認します。既存の読み取り専用ファイルを置き換えようとしていた場合は、その既存ファイルの読み取り専用属性を変更してファイルを置き換えるか、または別のファイル名で保存します。
3. 読み取り専用ファイルをシーケンシャル アクセスの **Output** モードまたは **Append** モードで開こうとしていないことを確認します。この場合は、ファイルを **Input** モードで開くか、ファイルの読み取り専用属性を変更します。
4. データベースまたはドキュメントの中で Visual Basic プロジェクトを変更しようとしていなかったことを確認します。

参照

概念

エラーの種類

アクセス許可は拒否されました。(Visual Basic)

書き込もうとしているディスクが書き込み保護されているか、またはアクセスしようとしているファイルがロックされています。

このエラーを解決するには

1. 書き込み保護されているファイルを開くには、ファイルの書き込み保護属性を変更します。
2. 他のプロセスによってファイルがロックされていないかどうかを確認し、ロックされていた場合は、解放されるまで待ってからファイルを開きます。
3. レジストリにアクセスするには、自分のユーザー アクセス許可で、この種のレジストリ アクセスが可能であることを確認します。

参照

概念

エラーの種類

ピクチャが有効ではありません。

ビットマップ、アイコン、または Windows メタファイル以外のグラフィックス形式を、フォームまたはコントロールの **Picture** プロパティに割り当てようとしました。

このエラーを解決するには

- **Picture** プロパティに読み込もうとしているファイルが、Visual Basic でサポートされている有効なグラフィックス ファイルであることを確認します。

参照

概念

グラフィックス (Visual Basic 6.0 ユーザー向け)

プリンタ エラー

プリンタでエラーが発生しましたが、ファイルを送信したコンピュータにその他の情報は返されていません。

このエラーを解決するには

- プリンタの物理的な確認を行います。コンピュータとプリンタの接続をすべて確認します。ほとんどのプリンタでは、"オフライン" や "用紙切れ" などのエラー情報が表示されます。

参照

概念

エラーの種類

プロシージャ呼び出しまたは引数が有効ではありません。(Visual Basic)

呼び出しの一部を完了できません。

このエラーを解決するには

- 使用できる値の範囲を超えている引数がないかどうかを確認します。

参照

概念

エラーの種類

処理 '<processname>' が見つかりませんでした。

指定されたプロセスが見つかりません。

このエラーを解決するには

- 正しいプロセス名を指定したことを確認します。

参照

その他の技術情報

プロセスの管理

プロパティ '<propertyname>' は、空の文字列または Nothing に設定できません。

このプロパティには、空の文字列 ("") または **Nothing** 以外の値を指定する必要があります。値が正しく計算されていない可能性があります。

このエラーを解決するには

- 有効な値を指定します。

参照

関連項目

Nothing (Visual Basic)

例外のトラブルシューティング : System.NullReferenceException

概念

エラーの種類

プロパティ '<propertyname>' は Nothing に設定できません。

プロパティには **Nothing** 以外の値を設定する必要があります。値が正しく計算されていない可能性があります。

このエラーを解決するには

- 有効な値を指定します。

参照

関連項目

Nothing (Visual Basic)
例外のトラブルシューティング : System.NullReferenceException

概念

エラーの種類

プロパティの配列インデックスが有効ではありません。

指定した値は、プロパティ配列インデックスとして有効ではありません。

このエラーを解決するには

- コンポーネントのドキュメントを参照し、指定したプロパティの有効な範囲内にインデックスがあることを確認します。

参照

その他の技術情報

Visual Basic における配列

Property Let プロシージャが定義されておらず、Property Get プロシージャからオブジェクトが返されませんでした。

プロパティ、メソッド、および演算の中には、**Collection** にしか適用できないものがあります。コレクションにしか適用できない演算またはプロパティが、コレクション以外のオブジェクトに対して指定されています。

このエラーを解決するには

1. オブジェクト名およびプロパティ名のスペルを確認するし、オブジェクトが **Collection** オブジェクトであることを確認します。
2. コレクションにオブジェクトを追加するのに使用した **Add** メソッドの構文が正しく、各識別子のスペルが正しいことを確認します。

参照

関連項目

Collection オブジェクト (Visual Basic)

プロパティが見つかりません。

このオブジェクトは指定されたプロパティをサポートしていません。

このエラーを解決するには

1. プロパティ名のスペルを確認します。
2. オブジェクトのドキュメントを参照し、間違ったプロパティにアクセスしていないかどうかを確認します。たとえば、オブジェクトでサポートされているプロパティの名前が "caption" なのに "text" プロパティにアクセスしようとしていた、というようなことが考えられます。

参照

概念

エラーの種類

プロパティまたはメソッドが見つかりません。

参照先のオブジェクト メソッドまたはオブジェクト プロパティが見つかりません。

このエラーを解決するには

- オブジェクト名のスペルが間違っている可能性があります。オブジェクトについて定義されているプロパティやメソッドを確認するには、オブジェクト ブラウザを表示します。該当するオブジェクト ライブラリを選択すると、使用できるプロパティとメソッドの一覧が表示されます。

参照

概念

エラーの種類

プロパティの値が無効です。

オブジェクトまたはコントロールのプロパティの 1 つに、そのプロパティの範囲外の値が設定されています。

このエラーを解決するには

- プロパティの値を有効な設定に変更します。

参照

概念

エラーの種類

型 '<typename>' でパブリック メンバ '<membername>' が見つかりません。

アクセスしようとしているメンバが見つかりません。このエラーは、遅延バインディング オブジェクトの非パブリック メンバにアクセスすると発生します。

このエラーを解決するには

- メンバが存在する場合は、**Public** と宣言されていることを確認します。

参照

関連項目

Public (Visual Basic)

概念

事前バインディングと遅延バインディング

'ReDim' によって、配列の次元数を変更することはできません。

ReDim ステートメントを使用して、配列のランク (次元の数) を変更する演算が行われました。**ReDim** は既に宣言された配列の 1 つ以上の次元のサイズを変更するために使用できますが、配列のランクを変更することはできません。

このエラーを解決するには

- 変更の対象が、次元のサイズではなく、配列のランクであることを確認します。可能な場合は、**Dim** を使用して、希望のランクを持つ配列を新規に宣言します。

参照

関連項目

ReDim ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)

概念

Visual Basic の配列の概要
Visual Basic における多次元配列

'ReDim' では最も右にある次元のみ変更できます。

ReDim ステートメントで、**Preserve** キーワードを使用して配列の最後の次元でない次元を変更しようとしました。**Preserve** を使用するときは、配列の最後の次元のみをサイズ変更できます。他のすべての次元については、既存の配列と同じサイズを指定する必要があります。

このエラーを解決するには

- **Preserve** キーワードを削除します。

参照

関連項目

ReDim ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)
Preserve

概念

Visual Basic の配列の概要
Visual Basic における多次元配列

'ReDim' 'Preserve' オペランドを Nothing にすることはできません。

ReDim ステートメントで、**Preserve** キーワードを使用して配列の最後の次元以外の次元を変更しようとしていますが、オペランドに有効な値が指定されていません。

このエラーを解決するには

- **Preserve** オペランドを有効な値に変更します。

参照

関連項目

ReDim ステートメント (Visual Basic)
Dim ステートメント (Visual Basic)
Preserve

概念

Visual Basic の配列の概要
Visual Basic における多次元配列

レジストリ キー '<keyname>' が見つかりませんでした。

レジストリ キーが見つかりません。

このエラーを解決するには

1. キー名のスペルを確認します。
2. キーにアクセスするために充分なセキュリティ アクセス許可があることを確認します。

参照

関連項目

レジストリの概要

概念

レジストリ アクセス (Visual Basic 6.0 ユーザー向け)

レジストリ キー '<keyname>' を作成できませんでした。

指定されたレジストリ キーを作成できませんでした。アクセス許可が不十分である可能性があります。

このエラーを解決するには

- キーへの十分なアクセス許可があることを確認します。

参照

関連項目

レジストリの概要

概念

レジストリ アクセス (Visual Basic 6.0 ユーザー向け)

置換後の文字列が長すぎます。

指定した置換文字列の長さが長すぎます。

このエラーを解決するには

- コンポーネントのドキュメントで長さの制限について確認します。

参照

処理手順

方法 : 対話モードで検索する

エラーが発生していないときに Resume を実行することはできません。

Resume ステートメントがエラー処理コードの外にあるか、またはエラーが発生していないのにエラー ハンドラ内にジャンプしました。

このエラーを解決するには

1. **Resume** ステートメントをエラー ハンドラ内に移動するか、削除します。
2. プロシージャ間にまたがってラベルにジャンプすることはできないため、エラー ハンドラを識別するラベルをプロシージャ内で探してください。**On Error GoTo** ステートメントではなく **GoTo** ステートメントのターゲットとして重複するラベルが指定されている場合は、目的のターゲットに合わせて行ラベルを変更します。

参照

概念

非構造化例外処理の概要

ルート フォルダの名前を変更できません。

ルートディレクトリの名前を変更しようとしました。指定したファイルパスが間違っている可能性があります。

このエラーを解決するには

- ファイルのパスが正しく指定されていることを確認します。

参照

処理手順

方法 : Visual Basic でディレクトリの名前を変更する

方法 : Visual Basic でディレクトリを移動する

関連項目

My.Computer.FileSystem.RenameDirectory メソッド

My.Computer.FileSystem.RenameFile メソッド

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

検索文字列が見つかりませんでした。

指定したテキストが見つかりませんでした。

このエラーを解決するには

- 指定したテキストのスペルが正しいことを確認します。

参照

処理手順

方法 : 対話モードで検索する

Set は実行時にはサポートされません。

デザイン時にだけ値を設定できるプロパティを設定または変更しようとしました。

このエラーを解決するには

1. プロパティへの参照をコードから削除します。
2. プロパティの値が実行時にだけ返されるように参照を変更します。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

Set はサポートされません。読み取り専用のプロパティです。

読み取り専用のプロパティを設定または変更しようとしました。

このエラーを解決するには

1. プロパティへの参照をコードから削除します。
2. プロパティの値が実行時にだけ返されるように参照を変更します。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

Set は使用できません。

実行時に設定できない、または特定の条件下でしか設定できないプロパティ設定を変更しようとしました。たとえば、フォームの **Appearance** プロパティ、**ControlBox** プロパティ、**MinButton** プロパティ、または **MaxButton** プロパティの設定を実行時に変更しようとしたり、親メニューで表示される最後のサブメニューの **Visible** プロパティに **False** を設定しようとしたりした可能性があります。

このエラーを解決するには

- プロパティを調べ、どのような条件下で設定できるかを確認します。

参照

処理手順

方法 : プロジェクト プロパティおよび構成設定を変更する

SimplifiedChinese と VbStrConv.TraditionalChinese 引数を組み合わせることはできません。

アプリケーションで **VbStrConv** 列挙体の **SimplifiedChinese** メンバと **TraditionalChinese** メンバを組み合わせようとしていますが、これらのメンバは相互に排他的です。

このエラーを解決するには

- **VbStrConv.SimplifiedChinese** と **VbStrConv.TraditionalChinese** のいずれかを削除します。

参照

関連項目

VbStrConv 列挙型

System.Globalization Namespace

概念

.NET Framework ベースの国際対応アプリケーションの概要

操作中に、一部のファイルおよびフォルダが例外を発生させました

操作の実行時にユーザーが必要なアクセス許可を持っていないため、ディレクトリおよびファイルが例外を引き起こす可能性があります。

このエラーを解決するには

- すべてのファイルおよびフォルダにアクセスできるアクセス許可を追加します。

参照

処理手順

トラブルシューティング : テキスト ファイルの読み取りと書き込み

関連項目

FileIOPermission

一部のサブキーが削除できません

レジストリ キーの削除が行われましたが、一部に削除できないサブキーがあるため処理が失敗しました。多くの場合、この原因はアクセス許可がないことです。

このエラーを解決するには

- 指定したサブキーを削除するための十分なアクセス許可があることを確認します。

参照

関連項目

My.Computer.Registry オブジェクト
DeleteSubKey
DeleteSubKey
RegistryPermission

ソース フォルダとターゲット フォルダが同じです。

ファイルをコピーまたは移動しようとしましたが、指定したソース ディレクトリとターゲット ディレクトリが同じです。ファイル パスの解析が間違っている可能性があります。

このエラーを解決するには

- ソース ディレクトリとターゲット ディレクトリが正しく指定されていることを確認します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する
方法 : Visual Basic でファイルを移動する
方法 : Visual Basic で特定のパターンを持つファイルをディレクトリにコピーする

関連項目

My.Computer.FileSystem.MoveFile メソッド
My.Computer.FileSystem.CopyFile メソッド

EventLogName ではなく、EventLogSource で指定されているソース名が、ログに登録されます。

EventLog が別のログに登録されたソースを参照しようとしています。イベント ログにエントリを書き込む場合、Source プロパティを指定する必要があります。**Source** プロパティでは、エントリの有効なソースとして、イベント ログにコンポーネントを登録します。1 つのソースは、同時に 1 つのイベント ログにしか (エントリを書き込むために) 関連付けることができません。

既定では、コンポーネントを有効なソースとして登録していない状態でエントリを書き込もうとすると、ソースが自動的にイベント ログに登録され、**Source** プロパティの値がソース文字列として使用されます。

このエラーを解決するには

- ソースが正しいログに登録されていることを確認します。これには、CreateEventSource メソッドまたはそのいずれかのオーバーロードを使用して、コンポーネントを一意に識別する文字列をイベント ログに指定します。

参照

処理手順

方法 : アプリケーションをイベント ログ エントリのソースとして追加する

方法 : イベント ソースを削除する

概念

イベント ログの参照

その他の技術情報

イベント ログの管理

関数は指定された DLL には定義されていません。

ユーザー ライブラリ参照内のダイナミック リンク ライブラリ (DLL: dynamic-link library) は見つかりましたが、指定した DLL 関数がこの DLL 内に見つかりませんでした。

このエラーを解決するには

1. 関数宣言で、有効な序数を指定します。
2. DLL の名前とエイリアスが正しいことを確認します。

参照

概念

エラーの種類

その他の技術情報

製品のサポートとユーザー補助

指定されたイベント ログはこのコンピュータに存在しません。

コンピュータ上に存在しないイベント ログにアクセスしようとしました。

このエラーを解決するには

- 正しいファイル名およびパスを指定していることを確認します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

その他の技術情報

イベント ログの管理

指定されたレジストリ キーは存在しません。

指定されたレジストリ キーが存在しません。

このエラーを解決するには

- 指定したレジストリ キーを調べて、パスを正しく記述していることを確認してください。
- 書き込みを行う前に、そのレジストリ キーを作成します。

参照

処理手順

チュートリアル : レジストリ キーの作成と値の変更

関連項目

My.Computer.Registry オブジェクト

RegistryKey

概念

一般的なレジストリ タスク

指定されたレジストリ キーは、2 つ以上連続するバックスラッシュを含んでいるため、有効ではありません。

レジストリ キーに、2 つ以上の連続する円記号 (\) を含むパスが指定されています。これによって複数の文字列を組み合わせたパスが作成され、余分な円記号を含んでしまう可能性があります。

このエラーを解決するには

- 指定するレジストリ キーを調べて、余分な円記号が挿入される場所と目的を確認してください。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.Registry オブジェクト

指定されたレジストリ パスには、先頭に有効なハイブ名がありません

指定されたレジストリ パスが、有効なハイブ名で始まりません。有効なハイブ名は、以下のとおりです。

- **HKEY_CLASSES_ROOT**
- **HKEY_CURRENT_CONFIG**
- **HKEY_CURRENT_USER**
- **HKEY_DYNDATA**
- **HKEY_LOCAL_MACHINE**
- **HKEY_PERFORMANCE_DATA**
- **HKEY_USERS**

このエラーを解決するには

- 有効なハイブ名を指定していることを確認します。

参照

関連項目

My.Computer.Registry オブジェクト

概念

一般的なレジストリ タスク

その他の技術情報

文字列の操作

Stop ステートメントで中断しました。

実行を中断する **Stop** ステートメントが見つかりました。

このエラーを解決するには

- **Stop** ステートメントを削除します。

参照

関連項目

Stop ステートメント (Visual Basic)
制御フローの概要

概念

Visual Basic の Stop ステートメント

'StrConv.LinguisticCasing' には、'StrConv.LowerCase' または 'StrConv.UpperCase' が必要です。

StrConv.LinguisticCasing を使用しようとしましたが、**StrConv.LowerCase** または **StrConv.UpperCase** と組み合わせた場合に限り有効です。

このエラーを解決するには

- **StrConv.LowerCase** または **StrConv.UpperCase** を、**StrConv.LinguisticCasing** と組み合わせて使用します。

参照

関連項目

StrConv 関数
StrConv 定数 (Visual Basic 6.0 ユーザー向け)

'FileSystem' API の文字列の長さが最大 32767 文字を超えています。

文字列の長さが、最大長である 32,767 文字を超えています。

このエラーを解決するには

- 文字列を短くします。

参照

その他の技術情報

Visual Basic における文字列

Sub または Function が定義されていません。(Visual Basic)

Sub または **Function** を呼び出すには、事前に定義しておく必要があります。このエラーでは以下の原因が考えられます。

- プロシージャ名のスペルが間違っています。
- [参照設定] ダイアログ ボックスで参照を明示的に追加していないプロジェクトのプロシージャを呼び出しています。
- 呼び出し元のプロシージャから参照できないプロシージャを指定しました。
- 指定されたライブラリまたはコード リソースにない Windows ダイナミック リンク ライブラリ (DLL: dynamic-link library) ルーチンまたは Macintosh コード リソース ルーチンを宣言しました。

このエラーを解決するには

1. プロシージャ名のスペルが正しいことを確認します。
2. [参照設定] ダイアログ ボックスで、呼び出そうとしているプロシージャを含むプロジェクトの名前を探します。名前が表示されていない場合は、[参照] をクリックして探します。プロジェクト名の左にあるチェック ボックスをオンにして、[OK] をクリックします。
3. ルーチンの名前を確認します。

参照

関連項目

Sub ステートメント (Visual Basic)
Function ステートメント (Visual Basic)

概念

エラーの種類
プロジェクト参照

インデックスが有効範囲にありません。(Visual Basic)

配列の添字が無効です。有効な範囲内にありません。次元の添字の最小値は常に 0 です。添字の最大値は、その次元に対する **GetUpperBound** メソッドによって返されます。

このエラーを解決するには

- 有効な範囲内に収まるように添字を変更します。

参照

関連項目

System.Array.GetUpperBound(System.Int32)

概念

Visual Basic の配列の概要

システム イベント ログを削除できません。

システム イベント ログの削除をしようとしましたが、このログは削除できません。システム ログには、システムの起動やハードウェアのエラーなどのシステム イベントが記録されます。

このエラーを解決するには

- アプリケーションがシステム イベント ログではなく、アプリケーション ログまたはカスタム ログに書き込むという方法を検討します。
- システム イベント ログを削除しようとしないでください。

参照

処理手順

方法 : カスタム イベント ログを作成または削除する

その他の技術情報

イベント ログの管理

ターゲット フォルダはファイルです。

フォルダ関連の処理を実行するときにフォルダではなくファイルを指定しています。

このエラーを解決するには

- 指定するパスを確認し、ファイルではなくフォルダを確実に指定するようにします。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する
方法 : Visual Basic でファイルの絶対パスを確認する

関連項目

My.Computer.FileSystem オブジェクト

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動
Visual Basic におけるファイル、ディレクトリ、およびドライブのプロパティ

TargetFilePath に既存のフォルダが指定されている

TargetFilePath パラメータに既存のディレクトリが指定されていますが、新しいディレクトリ名を指定する必要があります。

このエラーを解決するには

- 既存のディレクトリではなく新しいディレクトリを指定するように *TargetFilePath* を変更します。

参照

処理手順

方法 : Visual Basic でファイルを移動する
方法 : Visual Basic でディレクトリを移動する
方法 : Visual Basic でファイルの名前を変更する
方法 : Visual Basic でディレクトリの名前を変更する

TextFieldParser はスペース文字を含むコメント トークンをサポートしていません。

空白を含むコメント トークンが指定されました。**TextFieldParser** は空白がトークンの先頭にある場合を除き、空白を含むコメント トークンをサポートしません。トークンの先頭にある空白は無視されます。

このエラーを解決するには

- コメント トークンを正しく指定します。

参照

関連項目

TextFieldParser.CommentTokens プロパティ
TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

TextFieldParser は行末文字を含む区切り記号をサポートしていません

行末文字を含む区切り記号が指定されていますが、**TextFieldParse**r は行末文字を含む区切り記号をサポートしていません。

このエラーを解決するには

- 正しい区切り記号を指定します。

参照

関連項目

TextFieldParser.Delimiters プロパティ
TextFieldParser.SetDelimiters メソッド
TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

最大バッファサイズを超えたため、TextFieldParser は読み取り操作を完了できません。

最大バッファ サイズ (10,000,000 バイト) を超えたため、処理を完了できませんでした。

このエラーを解決するには

- ファイルに不適切なフィールドがないことを確認してください。

参照

処理手順

方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem.OpenTextFieldParser メソッド

TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

UploadFile のアドレスにはファイル名が含まれていなければなりません。

UploadFile メソッドを使用してファイルをアップロードする場合は、アップロード先のパスにファイル名を指定する必要があります。

このエラーを解決するには

- ファイル名を指定します。

参照

処理手順

方法 : Visual Basic でファイルをアップロードする

関連項目

My.Computer.Network オブジェクト

My.Computer.Network.UploadFile メソッド

その他の技術情報

Visual Basic による .NET Framework でのネットワーク操作

ファイルは既に開いています。

既に開いているファイルを開こうとしています。

このエラーを解決するには

- ファイルを閉じてから開くようにします。

  または

- 余分な **FileOpen** ステートメントをコードから削除します。

参照

関連項目

FileOpen 関数
My.Computer.FileSystem オブジェクト

ファイルは現在閉じられています

開いているファイルに対してのみ実行可能な処理が試行されましたが、そのファイルは現在閉じています。

このエラーを解決するには

- ファイルを開いてから処理をもう一度実行してください。

参照

関連項目

FileOpen 関数
FileClose 関数
My.Computer.FileSystem オブジェクト

その他の技術情報

Visual Basic におけるファイル アクセス

読み取りのため、ファイルが現在開いています。

現在読み取り中のために開かれているファイルへのアクセスが行われました。

このエラーを解決するには

- ファイルを閉じてから、もう一度処理を行ってください。

参照

処理手順

方法 : Visual Basic でファイルについての情報を取得する
方法 : Visual Basic でテキスト ファイルを読み取る
方法 : Visual Basic でバイナリ ファイルを読み取る

関連項目

FileIOPermission

書き込みのため、ファイルが現在開いています。

現在書き込み中のため開かれているファイルへのアクセスが行われました。

このエラーを解決するには

- ファイルを閉じてから、もう一度処理を行ってください。

参照

処理手順

方法 : Visual Basic でファイルについての情報を取得する
方法 : Visual Basic でテキストをファイルに書き込む
方法 : Visual Basic でバイナリ ファイルに書き込む

関連項目

FileIOPermission

同じパスのファイルが既に存在するため、フォルダを作成できません。

ディレクトリの作成が行われましたが、同じパスのファイルが既に存在します。これによって、特に相対パスを扱う場合に、パスが間違って解析される可能性があります。

このエラーを解決するには

- パスを正しく指定していることを確認します。
- 使用しなくなった余分なファイルは削除します。

参照

処理手順

方法 : Visual Basic でディレクトリを作成する
方法 : Visual Basic でディレクトリの名前を変更する

関連項目

My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.CreateDirectory メソッド
My.Computer.FileSystem.RenameDirectory メソッド

その他の技術情報

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

入力されたパスはファイル を指定していますが、ファイル パスの終わりにはディレクトリの区切り記号が指定されています。

ファイル名の入力を必要とする処理が行われましたが、指定された名前がディレクトリの区切り記号 (\) で終わっています。ファイル パスの指定が間違っている可能性があります。

このエラーを解決するには

- 指定したファイル パスを調べて、正しいことを確認してください。
- 余分な文字を削除します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

その他の技術情報

Visual Basic でのファイルの読み取り

Visual Basic でのファイルへの書き込み

パスが設定されていません。

移動やコピーなどのファイル操作を、ファイルのパスを指定せずに行っています。パスが間違っている可能性もあります。

このエラーを解決するには

- ファイルのパスが指定されているか、また正しく指定されているかを調べます。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

その他の技術情報

Visual Basic におけるファイル アクセス

リモート サーバー コンピュータが存在しないか、利用できません。(Visual Basic)

現在使用できないか、存在しないリモート サーバーに接続しようとしています。ネットワーク接続が切断されているために、このエラーが発生する場合があります。

このエラーを解決するには

- ネットワークに再接続して、やり直します。

参照

概念

エラーの種類

ソース フォルダが存在しません。

存在しないソース フォルダを参照する処理を実行しようとしました。指定したファイル パスが間違っている可能性があります。

このエラーを解決するには

- 正しいパスを使用していることを確認します。特に、相対パスを使用している場合は注意してください。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

その他の技術情報

Visual Basic でのファイルの読み取り

Visual Basic でのファイルおよびディレクトリの作成、削除、および移動

指定されたパスは存在しません。

指定されたパスが存在しません。文字列の結合が正しくない可能性があります。

このエラーを解決するには

- パスが正しく指定されていることを確認します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

概念

文字列とその他の型との変換

その他の技術情報

Visual Basic における文字列の解析

TextFieldParser に渡されたストリームを読み取れません。

TextFieldParser オブジェクト は、渡されたストリームを読み取ることができません。テキスト ファイルではないファイルを読み取ろうとしている可能性があります。

このエラーを解決するには

- ファイルがテキスト ファイルであることを確認します。

参照

処理手順

方法 : Visual Basic で複数の書式を持つテキスト ファイルを読み取る

関連項目

My.Computer.FileSystem.OpenTextFieldParser メソッド

TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

<argumentname> の値は正数でなければなりません。

ReserveDiskSpace プロパティの値は、0 よりも大きいことが必要です。

ReserveDiskSpace プロパティには、メッセージをログ ファイルに書き込むために必要なディスクの空き容量をバイト数で指定します。

このエラーを解決するには

- **ReserveDiskSpace** プロパティに正の数を設定します。

参照

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
ReserveDiskSpace

<argumentname> の値は 1000 以上でなければなりません。

MaxFileSize プロパティの値は、1000 以上であることが必要です。

MaxFileSize プロパティは、新しいログ ファイルを作成する前に、ログ ファイルに書き込み可能な最大のバイト数を指定します。

このエラーを解決するには

- **MaxFileSize** プロパティに 1000 以上の数を設定します。

参照

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
MaxFileSize

この配列は固定か、または一時的にロックされています。(Visual Basic)

このエラーでは以下の原因が考えられます。

- 固定サイズの配列の要素数を変更するために **ReDim** を使用しています。
- モジュール レベルの動的配列を再定義しようとしていますが、その配列の要素の 1 つがプロシージャに引数として渡されています。要素が引数として渡された場合、その配列はロックされ、プロシージャ内の参照パラメータのメモリが解放されなくなります。
- 配列を含む **Variant** 変数に値を代入しようとしましたが、その **Variant** は現在ロックされています。

このエラーを解決するには

1. **ReDim** を使用して元の配列を固定ではなく動的として宣言するか (配列がプロシージャ内で宣言されている場合)、要素数を指定せずに宣言します (配列がモジュール レベルで宣言されている場合)。
2. その要素を渡す必要があるのかどうかを確認します。これは、要素がモジュール内のすべてのプロシージャから参照できるためです。
3. **Variant** がロックされている原因を判断し、解決します。

参照

その他の技術情報

Visual Basic における配列

このキーは既にこのコレクションの要素に割り当てられています。

既にコレクションの別のメンバを指定しているキーを指定しました。キーは、**Add** メソッドで指定される、コレクションの特定のメンバを一意に識別する文字列です。

このエラーを解決するには

- このメンバに別のキーを指定します。

参照

概念

エラーの種類

この操作は、ファイルが閉じられているときにのみ実行することができます。

ファイルが閉じているときにだけ実行可能な処理が、ファイルが開いているときに実行されました。

このエラーを解決するには

- ファイルを閉じてから、もう一度処理を行ってください。

参照

処理手順

方法 : Visual Basic でファイルについての情報を取得する

関連項目

FileIOPermission

FileClose

この単一インスタンス アプリケーションは元のインスタンスに接続できませんでした。

この単一インスタンス アプリケーションは元のインスタンスに接続できませんでした。この問題の原因として考えられるものを次にいくつか示します。

- 元のインスタンスが応答しなくなっている。
- カーネル オブジェクトを作成するためのアクセス許可がアプリケーションにない。カーネル オブジェクトの詳細については、「ミューテックス」を参照してください。

  カーネル オブジェクトの基本名は、アセンブリの GUID、メジャー バージョン番号、およびマイナ バージョン番号を連結して付けられます。たとえば、基本名には 3639f15d-9547-43da-8145-60da347829915.1 などがあります。

アプリケーションの開発時にこのエラーを修正するには

1. アプリケーションが応答しない状態にならないことを確認します。
2. アプリケーションに、カーネル オブジェクトを作成するための十分なアクセス許可があることを確認します。
3. アプリケーションの元のインスタンスを再起動します。
4. コンピュータを再起動して、元のインスタンス アプリケーションに接続するために必要なリソースを使用している可能性のあるプロセスをすべて削除します。
5. エラーが発生した状況を記録して、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

処理手順

方法 : アプリケーションのインスタンス化の動作を指定する

その他の技術情報

デバッガのロードマップ

製品のサポートとユーザー補助

このシステムに、日本語ロケールのサポートは含まれていません。

VbStrConv.Japanese 列挙型のメンバを使おうとしていますが、使用しているシステムでは日本語ロケールがサポートされていません。

このエラーを解決するには

- 日本語ロケールがサポートされるようにコンピュータをアップグレードするか、このロケールをサポートするコンピュータでアプリケーションを実行します。

参照

関連項目

VbStrConv 列挙型

System.Globalization Namespace

概念

.NET Framework ベースの国際対応アプリケーションの概要

このシステムには、指定されたロケールのサポートは含まれていません。

VbStrConv 列挙型を使おうとしていますが、このシステムでは指定したロケールがサポートされていません。

このエラーを解決するには

- ロケールがサポートされるようにコンピュータをアップグレードするか、ロケールをサポートするコンピュータでアプリケーションを実行します。

参照

関連項目

VbStrConv 列挙型

System.Globalization Namespace

概念

.NET Framework ベースの国際対応アプリケーションの概要

このシステムには、簡体字中国語ロケールのサポートは含まれていません。

VbStrConv.SimplifiedChinese 列挙型のメンバを使おうとしていますが、使用しているシステムでは簡体字中国語ロケールがサポートされていません。

このエラーを解決するには

- 簡体字中国語ロケールがサポートされるようにコンピュータをアップグレードするか、このロケールをサポートするコンピュータでアプリケーションを実行します。

参照

関連項目

VbStrConv 列挙型

System.Globalization Namespace

概念

.NET Framework ベースの国際対応アプリケーションの概要

このシステムには、繁体字中国語ロケールのサポートは含まれていません。

VbStrConv.TraditionalChinese 列挙型のメンバを使おうとしましたが、使用しているシステムでは繁体字中国語ロケールがサポートされていません。

このエラーを解決するには

- 繁体字中国語ロケールがサポートされるようにコンピュータをアップグレードするか、このロケールをサポートするコンピュータでアプリケーションを実行します。

参照

関連項目

VbStrConv 列挙型

System.Globalization Namespace

概念

.NET Framework ベースの国際対応アプリケーションの概要

DLL のクライアント アプリケーションの数が多すぎます。

Visual Basic のダイナミック リンク ライブラリ (DLL: dynamic-link library) が対応できるホスト アプリケーションのアクセスの数には制限があります。Visual Basic ホストである使用中のアプリケーションまたはその他のアプリケーションが、Visual Basic DLL に同時にアクセスしています (一部は使用中のアプリケーションからアクセスできます)。

このエラーを解決するには

- Visual Basic にアクセスしている実行中アプリケーションの数を減らします。

参照

概念

エラーの種類

ファイルが多すぎます。

オペレーティング システムの制限を超える数のファイルがルート ディレクトリに作成されているか、CONFIG.SYS ファイルの files= 設定で指定された数を超えるファイルが開かれています。

このエラーを解決するには

1. プログラムでルート ディレクトリ内のファイルに対して、開く、閉じる、または保存の操作を行っている場合は、サブディレクトリが使用されるようにプログラムを変更します。
2. CONFIG.SYS ファイルの files= 設定で指定するファイルの数を増やし、コンピュータを再起動します。

参照

概念

エラーの種類

型が一致しません。(Visual Basic)

値を別の型に変換しようとしましたが、その変換は無効です。

このエラーを解決するには

1. 代入が有効であることを確認します。
2. 1 つのプロパティまたは値を受け取るプロシージャにオブジェクトを渡していないことを確認します。
3. 式を使用する場所でモジュールまたはプロジェクトを使用していないかどうかを確認します。

参照

概念

エラーの種類

引数 '<argumentname>' の型は '<typename>' で、数値ではありません。

引数が数値である場所に数値以外の引数が指定されています。

このエラーを解決するには

- 引数が数値になっているのを確認します。

参照

概念

引数の値渡しおよび参照渡し

パラメータの引き渡し方法 (Visual Basic 6.0 ユーザー向け)

内部システム エラーにより、シリアル ポート名を取得できません。

My.Computer.Ports.SerialPortNames プロパティが呼び出されたときに、内部エラーが発生しました。

このエラーを解決するには

1. トラブルシューティングの詳細については、「デバッガのロードマップ」を参照してください。
2. エラーが発生した状況を記録して、マイクロソフト プロダクト サポート サービスにご連絡ください。

参照

関連項目

My.Computer.Ports.SerialPortNames プロパティ

その他の技術情報

デバッガのロードマップ

製品のサポートとユーザー補助

ログのストリームを取得できません。

エラー メッセージ
ログのストリームを取得できません。<name> に基づく、可能性のあるファイル名は既に使用中です。

<name> に基づく、可能性のあるすべてのログ ファイル名が既に使用中であるため、FileLogTraceListener クラスは新しいログ ファイルを作成できませんでした。

ログ ファイルの数が多すぎる場合は、アプリケーションに構造上の問題がある可能性があります。詳細については、**FileLogTraceListener** クラスのドキュメントを参照してください。

このエラーを解決するには

1. Daily または Weekly に LogFileCreationSchedule プロパティを設定して、ログ ファイル名に日付スタンプを含めます。
2. 既存のログをアーカイブし、それをコンピュータから削除して、**FileLogTraceListener** オブジェクトが新しいログを作成できるようにします。

参照

関連項目
My.Application.Log オブジェクト
My.Log オブジェクト
FileLogTraceListener
LogFileCreationSchedule

FieldWidth が Nothing または空であるため、固定幅フィールドを読み取れません。

FieldWidths プロパティに **Nothing** が設定されているか、または空であるため、**TextFieldParser** は固定幅のフィールドを読み込むことができません。

このエラーを解決するには

- **FieldWidths** に有効な値を設定します。

参照

処理手順

方法 : Visual Basic で固定幅のテキスト ファイルを読み取る

関連項目

TextFieldParser.SetFieldWidths メソッド
TextFieldParser.FieldWidths プロパティ
TextFieldParser オブジェクト

区切り記号が Nothing または empty であるため、区切り記号で分けられたフィールドを読み取れません。

Delimiters プロパティが **Nothing** に設定されているか、空になっているため、**TextFieldParser** はファイルを読み込むことができません。

このエラーを解決するには

- **Delimiters** に有効な値を設定します。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る

関連項目

TextFieldParser.SetDelimiters メソッド
TextFieldParser.Delimiters プロパティ
TextFieldParser オブジェクト

概念

TextFieldParser オブジェクトによるテキスト ファイルの解析

EscapeQuote が True に設定されている場合、二重引用符は有効な区切り記号でないため、区切り記号で分けられたフィールドを読み取れません。

引用符 (") がデリミタとして使用され、**EscapeQuotes** が **True** に設定されているため、**TextFieldParser** が、ファイルから情報を読み取れません。

このエラーを解決するには

- **EscapeQuotes** を **False** に設定します。

参照

処理手順

方法 : Visual Basic でコンマ区切りのテキスト ファイルを読み取る

関連項目

TextFieldParser.SetDelimiters メソッド
TextFieldParser.Delimiters プロパティ
TextFieldParser オブジェクト

ネットワーク接続が利用できないため、ping を実行できません。

ネットワークに接続していないため、操作を実行できません。

このエラーを解決するには

- ネットワークに接続し、操作をやり直します。

参照

関連項目

My.Computer.Network.Ping メソッド
My.Computer.Network.IsAvailable プロパティ
My.Computer.Network オブジェクト

その他の技術情報

Visual Basic による .NET Framework でのネットワーク操作

サポートされているイベント受信最大数のイベントが既に発生しているので、オブジェクトのイベントを受け取ることができません。

オブジェクトによってサポートされているイベント レシーバの最大数を超えました。

このエラーを解決するには

- イベント レシーバの数を減らします。

参照

処理手順

方法 : COM シンクによって処理されるイベントを発生させる

チュートリアル : イベントの処理

書き込みを行うと MaximumSize 値を超えてしまうため、ログ ファイルに書き込めません。

FileLogTraceListener クラスは、次の理由でログ ファイルへ書き込みできませんでした。

- ログ ファイルのサイズ (バイト数) が MaxFileSize プロパティの値よりも大きい

  および

- DiskSpaceExhaustedBehavior プロパティの値が ThrowException

このエラーを解決するには

1. 既存のログをアーカイブし、それをコンピュータから削除して、**FileLogTraceListener** オブジェクトが新しいログを作成できるようにします。
2. **MaxFileSize** プロパティの値を変更して、ログをより多く書き込むことができるようにします。
3. **DiskSpaceExhaustedBehavior** プロパティに DiscardMessages を設定して、ログのサイズが大きすぎる場合に警告を出さないでメッセージを破棄するようにします。

参照

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
MaxFileSize
DiskSpaceExhaustedBehavior
FileLogTraceListener

書き込みを行うと、空きディスク領域が ReservedSpace 値よりも少なくなるため、ログ ファイルに書き込めません。

FileLogTraceListener クラスは、以下の理由でログ ファイルに書き込むことができません。

- ディスクの空き容量 (バイト単位) が ReserveDiskSpace プロパティの値より小さい。

  および

- DiskSpaceExhaustedBehavior プロパティの値が ThrowException である。

このエラーを解決するには

1. 既存のログをアーカイブし、コンピュータから削除して、**FileLogTraceListener** オブジェクトが新しいログを作成できるようにします。
2. **ReserveDiskSpace** プロパティの値を小さくして、予約済みのディスク領域を減らします。
3. **DiskSpaceExhaustedBehavior** プロパティを DiscardMessages に設定して、十分な空き容量がない場合に警告なしでメッセージが破棄されるようにします。

参照

関連項目

My.Application.Log オブジェクト
My.Log オブジェクト
ReserveDiskSpace
DiskSpaceExhaustedBehavior
FileLogTraceListener

ユーザーによる割り込みが発生しました。

ユーザーが **Ctrl** キーを押しながら **Break** キーを押したか、別の割り込みキーを押しました。

このエラーを解決するには

- 操作をやり直します。

参照

概念

エラーの種類

クラス コンストラクタにクラスの既定のインスタンスを使用すると、無限再帰呼び出しの原因となります。

クラスの既定のインスタンスが、クラスのコンストラクタで使用されています。これは、無限の再帰呼び出し (無限ループとも呼ばれる) になる可能性があります。

このエラーを解決するには

- 既定のインスタンスをクラス コンストラクタから削除します。

参照

関連項目

コンストラクタとデストラクタの使用方法

型 'Object' の引数を使う場合は、'FilePut' ではなく 'FilePutObject' を使用してください。

FilePut メソッドには **Object** 型の引数があります。あいまいさを避けるために **FilePut** の代わりに **FilePutObject** を使用する必要があります。

このエラーを解決するには

- **FilePut** を **FilePutObject** に置き換えます。
- *Object* 引数をより具体的な型にキャストします。
- **My.Computer.FileSystem** オブジェクトで利用できる機能を使用します。

参照

関連項目

FilePutObject 関数
My.Computer.FileSystem オブジェクト
My.Computer.FileSystem.WriteAllBytes メソッド

型 'Object' 引数を使う場合は、'FileGet' ではなく 'FileGetObject' を使用してください。

FileGet メソッドに **Object** 型の引数が指定されています。あいまいさを避けるために **FileGet** ではなく **FileGetObject** を使用する必要があります。

My.Computer.Filesystem の機能を利用すると、**FileGet** や **FileGetObject** よりも非常に簡単で、パフォーマンスもかなり向上します。

このエラーを解決するには

1. **FileGet** を **FileGetObject** に置き換えます。
2. *Object* 引数をより具体的な型にキャストします。

参照

関連項目

FileGetObject 関数
My.Computer.FileSystem オブジェクト

VbStrConv.Wide と VbStrConv.Narrow 引数を組み合わせることはできません。

アプリケーションで **VbStrConv** 列挙体のメンバ **Wide** と **Narrow** を組み合わせようとしていますが、これらのメンバは相互に排他的です。

このエラーを解決するには

- **VbStrConv.Wide** または **VbStrConv.Narrow** のいずれかを削除します。

参照

関連項目

VbStrConv 列挙型

System.Globalization Namespace

概念

.NET Framework ベースの国際対応アプリケーションの概要

VbStrConv.Wide および VbStrConv.Narrow は、指定されたロケールに適用できません。

アプリケーションで **VbStrConv** 列挙型の **Wide** メンバまたは **Narrow** メンバを使おうとしていますが、これらのメンバは指定されたロケールには適用できません。

このエラーを解決するには

- **VbStrConv.Wide** または **VbStrConv.Narrow** のいずれかを削除します。

参照

関連項目

VbStrConv 列挙型

System.Globalization Namespace

概念

.NET Framework ベースの国際対応アプリケーションの概要

引数の数が一致していないか、またはプロパティの割り当てが有効ではありません。

無効な代入が行われました。

このエラーを解決するには

1. 指定した引数の数と、代入先で必要とされる引数の数が一致していることを確認します。
2. プロパティへの値の代入を確認します。

参照

関連項目

代入演算子

ファイル名を指定しなければなりません。

ファイル名を指定する必要がある操作 (ファイルの移動やコピーなど) を行おうとしています。

このエラーを解決するには

- 必要なファイル名を指定します。
- パスの末尾にディレクトリの区切り記号 (\) を含めないように注意してください。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

関連項目

My.Computer.FileSystem オブジェクト

その他の技術情報

Visual Basic におけるファイル アクセス

名前を指定してください。

ファイル名の変更などのファイル操作を実行するには、名前を指定する必要があります。

このエラーを解決するには

- 名前を指定してください。

参照

関連項目

My.Computer.FileSystem オブジェクト

概念

エラーの種類

現在のフォルダの下のパス (そのフォルダのサブフォルダの 1 つ) を指定する必要があります。

指定されたパスは現在のディレクトリにありません。相対パスが指定されたとき、ファイルのパスが間違って解析される可能性があります。

このエラーを解決するには

- パスを正しく指定していることを確認します。

参照

処理手順

方法 : Visual Basic でファイル パスを解析する

Visual Basic のコマンド ライン ツール

ここでは、Visual Basic のコマンド ライン コンパイラとキーワード アップグレード ツールについて説明します。

このセクションの内容

Visual Basic コンパイラ

コマンド ライン コンパイラについて説明します。コマンド ライン コンパイラは、Visual Studio 統合開発環境 (IDE) を使用せずにプログラムをコンパイルするためのツールです。

Visual Basic キーワード アップグレード ツール

キーワード アップグレード コマンド ライン ツールは、Visual Basic .NET 2002 および 2003 で有効な Visual Basic ソース ファイルを、有効な Visual Basic 2005 ファイルに変換します。

関連するセクション

Visual Basic リファレンス

Vbupgrade のコマンド ライン構文

Visual Basic コンパイラ

Visual Basic のコマンド ライン コンパイラを使用すると、Visual Studio の統合開発環境 (IDE) 内からでもプログラムをコンパイルできます。ここでは、Visual Basic のコンパイラ オプションについて説明します。

このセクションの内容

コマンド ラインからのビルド (Visual Basic)

Visual Studio IDE からプログラムをコンパイルするために用意されている、Visual Basic コマンド ライン コンパイラについて説明します。

Visual Basic コンパイラ オプション一覧 (アルファベット順)

コンパイラ オプションがアルファベット順に示されています。

Visual Basic コンパイラ オプション一覧 (カテゴリ別)

コンパイラ オプションが機能グループ別に示されています。

関連するセクション

プロジェクト デザイナの概要

プロジェクト デザイナを使って、プロジェクトにグローバル設定を指定する方法について説明します。

Visual Basic

Visual Basic ヘルプはここから使用します。

コマンド ラインからのビルド (Visual Basic)

Visual Basic プロジェクトは、1 つ以上の独立したソース ファイルで構成されています。コンパイルと呼ばれるプロセスの間に、これらのファイルが 1 つのパッケージにまとめられて、アプリケーションとして実行できる実行可能ファイルになります。

Visual Studio 統合開発環境 (IDE) からプログラムをコンパイルする方法の他に、Visual Basic には、コマンド ライン コンパイラが用意されています。コマンド ライン コンパイラは、IDE の機能のすべてが必要なわけではない場合に使用できます。たとえば、システム メモリや記憶領域が限られているコンピュータを使用している場合や、そうしたコンピュータを対象としてプログラムを作成している場合などに使用できます。

コマンド ラインからコンパイルする場合、**/reference** コンパイラ オプションを使用して Visual Basic ランタイム ライブラリを明示的に参照する必要があります。

Visual Studio IDE でソース ファイルをコンパイルするには、[ビルド] メニューの [ビルド] をクリックします。コマンド ラインでプロジェクト (.vbproj) ファイルをコンパイルするには、「Devenv コマンド ライン スイッチ」を参照してください。

このセクションの内容

方法 : コマンド ライン コンパイラを起動する

MS-DOS プロンプトまたは特定のサブディレクトリからコマンド ライン コンパイラを起動する方法を説明します。

コンパイル コマンド ラインのサンプル

変更を加えて使用できるコマンド ラインのサンプルを紹介します。

関連するセクション

Visual Basic コンパイラ

アルファベット順または用途別に整理されたコンパイラ オプションの一覧です。

条件付きコンパイルの概要

コードの特定のセクションをコンパイルする方法を説明します。

方法 : ビルドの準備および管理

各ビルドに含める要素の整理、プロジェクト プロパティの選択、およびプロジェクトのビルド順序の設定の方法を説明します。

方法：コマンド ライン コンパイラを起動する

コマンド ライン コンパイラを起動するには、コマンド ライン (MS-DOS プロンプト) で実行可能ファイルの名前を入力します。既定の方法で、つまり Windows のコマンド プロンプトからコンパイルする場合は、実行可能ファイルの絶対パスを入力する必要があります。この既定の動作をオーバーライドするには、Visual Studio のコマンド プロンプトを使うか、または PATH 環境変数を変更します。どちらの方法でも、任意のディレクトリからコンパイラ名を入力するだけでコンパイルできます。

このコマンド ラインからコンパイルする場合、**/reference** コンパイラ オプションを使用して Microsoft Visual Basic ランタイム ライブラリを明示的に参照する必要があります。

メモ：

使用している設定またはエディションによっては、表示されるダイアログ ボックスやメニュー コマンドがヘルプに記載されている内容と異なる場合があります。設定を変更するには、[ツール] メニューの [設定のインポートとエクスポート] をクリックします。詳細については、「Visual Studio の設定」を参照してください。

Visual Studio コマンド プロンプトを使ってコンパイラを起動するには

1. Microsoft Visual Studio のプログラム グループ内の、Visual Studio Tools プログラムのフォルダを開きます。
2. Visual Studio 2005 をインストールしている場合は、Visual Studio のコマンド プロンプトを使用して、コンピュータの任意のディレクトリからコンパイラにアクセスできます。
3. Visual Studio のコマンド プロンプトを起動します。
4. コマンド ラインで `vbc.exe exe /imports:Microsoft.VisualBasic,System` *sourceFileName* と入力し、Enter キーを押します。

   たとえば、ソース コードを `SourceFiles` ディレクトリに格納したとすると、コマンド プロンプトを開き、`cd SourceFiles` と入力してそのディレクトリに移動します。ソース ファイルをディレクトリに `Source.vb` という名前で格納していれば、`vbc.exe /imports:Microsoft.VisualBasic,System Source.vb` と入力してコンパイルできます。

Windows のコマンド プロンプト用に、コンパイラへの PATH 環境変数を設定するには

1. Windows の検索機能を使用して、ローカル ディスクで Vbc.exe を検索します。

   コンパイラ が格納されているディレクトリの名前は、Windows ディレクトリの場所、およびインストールされている .NET Compact Framework のバージョンによって決まります。複数のバージョンの .NET Compact Framework がインストールされている場合は、使用するバージョン (通常は最新のバージョン) を決定する必要があります。

2. [スタート] メニューの [マイ コンピュータ] を右クリックし、ショートカット メニューの [プロパティ] をクリックします。
3. [詳細] タブをクリックし、[環境変数] をクリックします。
4. [システム環境変数] の一覧の [Path] を選択し、[編集] をクリックします。
5. [システム変数の編集] ダイアログ ボックスで、[変数値] ボックスの文字列の末尾にカーソルを移動し、セミコロン (;) に続けて手順 1 で検索されたディレクトリの完全パスを入力します。
6. [OK] をクリックして編集を完了し、ダイアログ ボックスを閉じます。

   PATH 環境変数を変更したら、Windows のコマンド プロンプトを使用して、コンピュータの任意のディレクトリから Visual Basic コンパイラを実行できます。

Windows のコマンド プロンプトを使ってコンパイラを起動するには

1. [スタート] メニューの [アクセサリ] フォルダをクリックし、[コマンド プロンプト] を開きます。
2. コマンド ラインで `vbc.exe /imports:Microsoft.VisualBasic,System` *sourceFileName* と入力し、Enter キーを押します。

   たとえば、ソース コードを `SourceFiles` ディレクトリに格納したとすると、コマンド プロンプトを開き、`cd SourceFiles` と入力してそのディレクトリに移動します。ソース ファイルをディレクトリに `Source.vb` という名前で格納していれば、`vbc.exe /imports:Microsoft.VisualBasic,System Source.vb` と入力してコンパイルできます。

参照

関連項目

/reference (Visual Basic)

概念

条件付きコンパイルの概要

その他の技術情報

Visual Basic コンパイラ

コンパイル コマンド ラインのサンプル

Visual Basic プログラムを Visual Studio からコンパイルする代わりに、コマンド ラインでコンパイルして、実行可能 (.exe) ファイルやダイナミック リンク ライブラリ (.dll) ファイルを作成できます。

Visual Basic のコマンド ライン コンパイラは、ファイルの入出力、アセンブリ、およびデバッグを制御する一連のオプションとプリプロセッサ オプションをサポートしています。各オプションには、それぞれ *-option* と */option* という 2 つの形式があり、どちらの形式でも同じように使用できます。ただし、ここでは */option* という形式だけを使用します。

以下のコマンド ラインのサンプルは、変更を加えてコード内で使用できます。

目的	コマンド ライン
File.vb をコンパイルして File.exe を作成する	**vbc /reference:Microsoft.VisualBasic.dll File.vb**
File.vb をコンパイルして File.dll を作成する	**vbc /target:library File.vb**
File.vb をコンパイルして My.exe を作成する	**vbc /out:My.exe File.vb**
最適化を有効にし、**DEBUG** シンボルを定義して、現在のディレクトリのすべての Visual Basic ファイルをコンパイルして File2.exe を作成する	**vbc /define:DEBUG=1 /optimize /out:File2.exe *.vb**
現在のディレクトリのすべての Visual Basic ファイルをコンパイルし、ロゴも警告も表示せずに File2.dll のデバッグ バージョンを作成する	**vbc /target:library /out:File2.dll /nowarn /nologo /debug *.vb**
現在のディレクトリのすべての Visual Basic ファイルを Something.dll にコンパイルする	**vbc /target:library /out:Something.dll *.vb**

コマンド ラインからコンパイルするとき、**/reference** コンパイラ オプションを使用して Microsoft Visual Basic ランタイム ライブラリを明示的に参照する必要があります。

参照

概念

条件付きコンパイルの概要

その他の技術情報

Visual Basic コンパイラ

Visual Basic コンパイラ オプション一覧 (アルファベット順)

Visual Basic のコマンド ライン コンパイラは、Visual Studio 統合開発環境 (IDE) からプログラムをコンパイルする方法の代替として用意されています。Visual Basic コマンド ライン コンパイラ オプションの一覧をアルファベット順で次に示します。

オプション	目的
@ (応答ファイルの指定)	応答ファイルを指定します。
/?	コンパイラ オプションを表示します。このコマンドは、**/help** オプションを指定するのと同じです。コンパイルは行われません。
/addmodule	指定ファイルからのすべての型情報をコンパイル中のプロジェクトで使用できるようにします。
/baseaddress	ダイナミック リンク ライブラリ (DLL: Dynamic Link Library) のベース アドレスを指定します。
/bugreport	バグを簡単に報告するための情報を含むファイルを作成します。
/codepage	コンパイルですべてのソース コード ファイルに使用するコード ページを指定します。
/debug	デバッグ情報を生成します。
/define	条件付きコンパイルの記号を定義します。
/delaysign	アセンブリに完全に署名するか、部分的に署名するかを指定します。
/doc	ドキュメント コメントを XML ファイルに出力します。
/errorreport	Visual Basic コンパイラが内部コンパイラ エラーを出力する方法を指定します。
/filealign	出力ファイルのセクションを配置する場所を指定します。
/help	コンパイラ オプションを表示します。このコマンドは、**/?** オプションを指定するのと同じです。コンパイルは行われません。
/imports	指定のアセンブリから名前空間をインポートします。
/keycontainer	アセンブリに厳密な名前を付けるために、キー ペアのキー コンテナの名前を指定します。
/keyfile	アセンブリに厳密な名前を付けるキーおよびキー ペアを含むファイルを指定します。
/libpath	/reference オプションによって参照されるアセンブリの場所を指定します。
/linkresource	マネージ リソースへのリンクを作成します。
/main	起動時に使用する **Sub Main** プロシージャを含むクラスを指定します。
/netcf	.NET Compact Framework を対象とするようにコンパイラを設定します。
/noconfig	コンパイルに Vbc.rsp を使用しません。
/nologo	コンパイラの著作権情報が表示されないようにします。
/nostdlib	コンパイラが標準のライブラリを参照しないよう指定します。

/nowarn	警告を生成するコンパイラの機能が表示されないようにします。
/optimize	コードの最適化を有効または無効にします。
/optioncompare	文字列比較をバイナリで行うか、ロケール固有のテキストの意味を使用して行うかを指定します。
/optionexplicit	変数の明示的宣言を必須にします。
/optionstrict	厳密な言語の意味を適用します。
/out	出力ファイルを指定します。
/platform	コンパイラが出力ファイルの対象とするプロセッサのプラットフォームを指定します。
/quiet	構文関連のエラーおよび警告に関するコードが表示されないようにします。
/recurse	コンパイルするソース ファイルをサブディレクトリで検索します。
/reference	アセンブリからメタデータをインポートします。
/removeintchecks	整数のオーバーフロー チェックを無効にします。
/resource	マネージ リソースをアセンブリに埋め込みます。
/rootnamespace	すべての型宣言の名前空間を指定します。
/sdkpath	Mscorlib.dll と Microsoft.VisualBasic.dll の場所を指定します。
/target	出力ファイルの形式を指定します。
/utf8output	UTF-8 エンコーディングを使用してコンパイラ出力を表示します。
/verbose	コンパイル時のその他の情報を出力します。
/warnaserror	警告をエラーとして扱います。
/win32icon	.ico ファイルを出力ファイルに挿入します。
/win32resource	Win32 リソースを出力ファイルに挿入します。

参照

関連項目

Visual Basic コンパイラ オプション一覧 (カテゴリ別)

概念

プロジェクト デザイナの概要

@ (応答ファイルの指定) (Visual Basic)

コンパイラ オプションと、コンパイルするソース コード ファイルを含むファイルを指定します。

```
@response_file
```

引数
response_file

必ず指定します。コンパイラ オプションやコンパイルするソース コード ファイルの一覧を含むファイル。ファイル名に空白が含まれている場合は、二重引用符 (" ") で囲みます。

解説

コンパイラでは、応答ファイルで指定したコンパイラ オプションとソース コード ファイルをコマンド ラインに指定した場合と同様に処理します。

コンパイル時に複数の応答ファイルを指定するには、次のように、複数の応答ファイル オプションを指定します。

```
@file1.rsp @file2.rsp
```

応答ファイルでは、複数のコンパイラ オプションとソース コード ファイルを 1 行に記述できます。1 つのコンパイラ オプションは 1 行に指定する必要があり、複数行にわたって指定できません。応答ファイルには、シャープ記号 (#) で始まるコメントを記述できます。

コマンド ラインに指定したオプションと、1 つ以上の応答ファイルで指定したオプションを組み合わせることができます。コンパイラでは、検出順にコマンド オプションを処理します。このため、コマンド ライン引数が、応答ファイルで先に指定したオプションをオーバーライドすることがあります。逆に、応答ファイルのオプションが、コマンド ラインや他の応答ファイルで先に指定したオプションをオーバーライドすることもあります。

Visual Basic では、Vbc.exe ファイルと同じディレクトリに Vbc.rsp ファイルがあります。**/noconfig** オプションが使用されている場合を除き、Vbc.rsp ファイルは既定で含まれています。詳細については、「/noconfig」を参照してください。

メモ :

@ オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

サンプルの応答ファイルの一部を次に示します。

```
# build the first output file
/target:exe
/out:MyExe.exe
source1.vb
source2.vb
```

次の例では、**@** オプションを `File1.rsp` という応答ファイルで使用する方法を示します。

```
vbc @file1.rsp
```

参照
関連項目
/noconfig
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/addmodule

指定ファイルからのすべての型情報をコンパイル中のプロジェクトで使用できるようにします。

```
/addmodule:fileList
```

引数
fileList

必ず指定します。メタデータは含まれるが、アセンブリ マニフェストは含まれないファイルの、コンマ区切りのリストです。空白を含むファイル名は、引用符 (" ") で囲みます。

解説

fileList パラメータで表示されるファイルは、**/target:module** オプション (または他のコンパイラで **/target:module** に相当するオプション) を指定して作成する必要があります。

/addmodule を使用して追加されたすべてのモジュールは、実行時に出力ファイルと同じディレクトリに配置されている必要があります。つまり、コンパイル時にはどのディレクトリのモジュールでも指定できますが、実行時にはモジュールをアプリケーション ディレクトリに置く必要があります。アプリケーション ディレクトリにモジュールがないと、TypeLoadException エラーが発生します。

/target:module 以外の /target (Visual Basic) オプションを **/addmodule** と共に暗黙的または明示的に指定すると、**/addmodule** に渡すファイルはプロジェクトのアセンブリの一部になります。アセンブリは、**/addmodule** で追加されたファイルを 1 つ以上含む出力ファイルを実行する必要があります。

アセンブリを含むファイルからメタデータをインポートするには、/reference (Visual Basic) を使用します。

メモ :

/addmodule オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

モジュールを作成する場合のコード例を次に示します。

VB

```
' t1.vb
' Compile with vbc /target:module t1.vb.
' Outputs t1.netmodule.

Public Class TestClass
    Public i As Integer
End Class
```

モジュールの型をインポートする場合のコード例を次に示します。

VB

```
' t2.vb
' Compile with vbc /addmodule:t1.netmodule t2.vb.
Option Strict Off

Namespace NetmoduleTest
    Module Module1
        Sub Main()
            Dim x As TestClass
            x = New TestClass
            x.i = 802
            System.Console.WriteLine(x.i)
        End Sub
    End Module
End Namespace
```

`t1` を実行すると、`802` が出力されます。

参照

関連項目

/target (Visual Basic)

/reference (Visual Basic)

コンパイル コマンド ラインのサンプル

その他の技術情報

Visual Basic コンパイラ

/baseaddress

ダイナミック リンク ライブラリ (DLL: Dynamic Link Library) を作成するときの既定のベース アドレスを指定します。

```
/baseaddress:address
```

引数
address

必ず指定します。DLL のベース アドレス。このアドレスは 16 進数として指定する必要があります。

解説

DLL の既定のベース アドレスは、.NET Framework によって設定されます。

このアドレスの下位ワードは丸められます。たとえば、0x11110001 と指定すると、丸められて 0x11110000 になります。

DLL の署名プロセスを完了するには、厳密名ツール (Sn.exe) の **–R** オプションを使用します。

ターゲットが DLL でない場合、このオプションは無視されます。

Visual Studio IDE で /baseaddress を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. [次へ] をクリックします。
4. [DLL ベース アドレス] ボックス内の値を変更します。

メモ：
[DLL ベース アドレス] ボックスは、対象が DLL である場合を除き、読み取り専用です。

参照
関連項目
/target (Visual Basic)
コンパイル コマンド ラインのサンプル
厳密名ツール (Sn.exe)
その他の技術情報
Visual Basic コンパイラ

/bugreport

バグ レポートの作成時に使用するファイルを作成します。

```
/bugreport:file
```

引数
file

必ず指定します。バグ レポートを作成するファイルの名前。ファイル名に空白が含まれている場合は、二重引用符 (" ") で囲みます。

解説

file には次の情報が格納されます。

- コンパイル時のすべてのソース コード ファイルのコピー。
- コンパイルで使用されたコンパイラ オプションの一覧。
- コンパイラ、共通言語ランタイム、およびオペレーティング システムのバージョン情報。
- コンパイラの出力 (指定されている場合)。
- 問題の説明。プロンプトが表示されます。
- 問題の修正方法の説明。プロンプトが表示されます。

すべてのソース コード ファイルのコピーが *file* に収められるため、問題があると思われるコードをできるだけ小さなプログラムとして再生成できます。

セキュリティに関するメモ：

/bugreport オプションを設定すると、現在時刻、コンパイラのバージョン、.NET Framework のバージョン、オペレーティング システムのバージョン、ユーザー名、コンパイラを起動したコマンド ライン引数、すべてのソース コード、参照されるアセンブリのバイナリ形式を含む機密性の高い情報を格納するファイルが作成されます。このオプションを設定するには、ASP.NET アプリケーションのサーバー側のコンパイルに使用する web.config ファイル内のコマンド ライン オプションを指定します。このオプションの設定を回避するには、Machine.config ファイルを変更して、ユーザーがサーバーでコンパイルすることを許可しないようにします。

このオプションを **/errorreport:prompt**、**/errorreport:queue**、または **/errorreport:send** と組み合わせて使用している場合に、アプリケーションで内部コンパイラ エラーが発生すると、*file* 内の情報がマイクロソフトに送信されます。マイクロソフトのエンジニアは、この情報を基にエラーの原因を特定し、Visual Basic の次のリリースの改善に役立てます。既定では、マイクロソフトに情報は送信されません。ただし、アプリケーションを、既定で有効になっている **/errorreport:queue** でコンパイルした場合、アプリケーションがエラー レポートを収集します。そして、コンピュータの管理者がログインすると、エラー レポート システムが、ログイン以降に発生したエラー レポートをマイクロソフトに送信できるポップアップ ウィンドウを表示します。

メモ：

/bugreport オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

`T2.vb` をコンパイルし、すべてのバグ レポート情報を `Problem.txt` ファイルに出力する場合のコード例です。

```
vbc /bugreport:problem.txt t2.vb
```

参照
関連項目
/debug (Visual Basic)
/errorreport
コンパイル コマンド ラインのサンプル
securityPolicy の trustLevel 要素 (ASP.NET 設定スキーマ)

その他の技術情報

Visual Basic コンパイラ

/codepage (Visual Basic)

コンパイルですべてのソース コード ファイルに使用するコード ページを指定します。

```
/codepage:id
```

引数
id

必ず指定します。コンパイラは、*id* で指定されたコード ページを使用して、ソース ファイルのエンコーディングを解釈します。

解説

特定のエンコーディングで保存されたソース コードをコンパイルするには、**/codepage** を使用して、使用するコード ページを指定します。**/codepage** オプションは、コンパイル時にすべてのソース コード ファイルに適用されます。詳細については、「コード ページのエンコーディング サポート」を参照してください。

ソース コード ファイルが現在の ANSI コード ページ、Unicode、またはシグネチャ付き UTF-8 を使って保存されている場合は **/codepage** オプションは不要です。Visual Studio は、ユーザーが [エンコード] ダイアログ ボックスで他のエンコーディングを指定しない限り、すべてのソース コード ファイルを、既定で現在の ANSI コード ページを使って保存します。Visual Studio は [エンコード] ダイアログ ボックスを使って、他のコード ページで保存されたソース コード ファイルを開きます。詳細については、「[エンコード] ダイアログ ボックス」を参照してください。

メモ：

/codepage オプションは Visual Studio 開発環境内では利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

参照
関連項目
[エンコード] ダイアログ ボックス
その他の技術情報
Visual Basic コンパイラ

/debug (Visual Basic)

コンパイラでデバッグ情報を生成し、出力ファイルを作成します。

```
/debug[+ | -]
' -or-
/debug:[full | pdbonly]
```

引数
+ | -

省略可能です。+ または **/debug** を指定すると、コンパイラによってデバッグ情報が生成され、.pdb ファイルにその情報が出力されます。- を指定するのは、**/debug** を指定しないのと同じです。

full | **pdbonly**

省略可能です。コンパイラによって生成されるデバッグ情報の種類を指定します。**/debug:pdbonly** を指定しなかった場合、既定値は **full** となり、実行中のプログラムにデバッガをアタッチできます。**pdbonly** 引数を指定すると、プログラムがデバッガで開始されたときにはソースコードをデバッグできますが、実行中のプログラムをデバッガにアタッチしたときはアセンブリ言語のコードしか表示されません。

解説

このオプションを使用してデバッグ ビルドを作成します。**/debug**、**/debug+**、または **/debug:full** を指定しないと、プログラムの出力ファイルをデバッグできません。

既定では、デバッグ情報は生成されません (**/debug-**)。デバッグ情報を生成するには、**/debug** または **/debug+** を指定します。

アプリケーションのデバッグ パフォーマンスを設定する方法については、「イメージのデバッグの簡略化」を参照してください。

Visual Studio 統合開発環境で /debug を設定するには

1. ソリューション エクスプローラでプロジェクトが選択されている状態で、[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. [次へ] をクリックします。
4. [DLL ベース アドレス] ボックス内の値を変更します。

使用例

デバッグ情報をファイル `App.exe` に出力する場合のコード例です。

```
vbc /debug /out:app.exe test.vb
```

参照
関連項目
/bugreport
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/define (Visual Basic)

条件付きコンパイル定数を定義します。

```
/define:["]symbol=value[,symbol=value]["]
' -or-
/d:["]symbol=value[,symbol=value]["]
```

引数
symbol

必ず指定します。定義する記号。

value

必ず指定します。*symbol* に代入する値。*value* が文字列の場合、これは引用符ではなく、バックスラッシュ/引用符のシーケンス (\") で囲む必要があります。

解説

/define オプションは、ソース ファイル内で **#Const** プリプロセッサ ディレクティブを使用するのと同じ効果を持ちます。ただし、**/define** で定義された定数は public で、プロジェクト内のすべてのファイルに適用されます。

このオプションで作成される記号を **#If**...**Then**...**#Else** ディレクティブで使用すると、ソース ファイルを条件付きでコンパイルできます。

/d は **/define** の省略形です。

記号の定義をコンマで区切ると、**/define** を使用して複数の記号を定義できます。

Visual Studio 統合開発環境で /define を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. [次へ] をクリックします。
4. [カスタム定数] ボックス内の値を変更します。

使用例

2 つの条件付きコンパイル定数を定義して使用する場合のコード例を次に示します。

VB

```
    ' Vbc /define:DEBUGMODE=True,TRAPERRORS=False test.vb
    Sub mysub()
#If debugmode Then
        ' Insert debug statements here.
         MsgBox("debug mode")
#Else
        ' Insert default statements here.
#End If
    End Sub
```

参照
関連項目
#If...Then...#Else ディレクティブ
#Const ディレクティブ
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/delaysign

アセンブリに完全に署名するか、部分的に署名するかを指定します。

```
/delaysign[+ | -]
```

引数
+ | -

省略可能です。完全署名されたアセンブリを作成する場合は、**/delaysign-** を使用します。アセンブリ内に公開キーを配置し、署名付きハッシュの領域を確保する場合は、**/delaysign+** を使用します。既定値は **/delaysign-** です。

解説

/delaysign オプションは、/keyfile または /keycontainer と共に使用しなければ無効になります。

完全署名されたアセンブリを要求すると、コンパイラはマニフェスト (アセンブリ メタデータ) を含むファイルをハッシュし、そのハッシュに秘密キーで署名します。結果として得られるデジタル署名は、マニフェストを含むファイルに格納されます。アセンブリを遅延署名に設定すると、コンパイラは署名の計算も格納も行いませんが、後で署名を追加できるようにファイルに領域を確保します。

たとえば **/delaysign+** を使用すると、組織内の開発者が署名のないテスト バージョンのアセンブリを配布し、テスト担当者がそのアセンブリをグローバル アセンブリ キャッシュに登録して使用することができます。このアセンブリに対する作業が終わったら、組織の秘密キーについて責任を持つ人物が、そのアセンブリに完全署名することができます。この分離方式を使用すると、すべての開発者がアセンブリに対して作業を行えるようにしつつ、組織の秘密キーを漏洩から保護できます。

アセンブリに対する署名の詳細については、「厳密な名前付きアセンブリの作成と使用」を参照してください。

Visual Studio 統合開発環境で /delaysign を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [署名] タブをクリックします。
3. [遅延署名のみ] に値を設定します。

参照
関連項目
/keyfile
/keycontainer
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/doc

ドキュメント コメントを XML ファイルに出力します。

```
/doc[+ | -]
' -or-
/doc:file
```

引数

+ | -

省略可能です。+ を指定するか、または **/doc** だけを指定すると、コンパイラがドキュメント情報を生成し、XML ファイルに出します。- を指定すると、**/doc** を指定しないのと同じ意味になり、ドキュメント情報は作成されません。

file

/doc: を使用する場合は、必ず指定します。コンパイルするソース コード ファイルからコメントを書き込む、XML 形式の出力ファイルを指定します。ファイル名に空白が含まれる場合は、二重引用符 (" ") で囲む必要があります。

解説

/doc オプションを使って、コンパイル時にドキュメント コメントを含む XML ファイルを生成するかどうかを制御できます。**/doc:***file* の構文を使用する場合、*file* パラメータには XML ファイルの名前を指定します。**/doc** または **/doc+** を使用する場合、コンパイラは作成している実行可能ファイルまたはライブラリから XML ファイル名を取得します。**/doc-** を使用するか、または **/doc** オプションを指定しなければ、コンパイラは XML ファイルを作成しません。

ソース コード ファイルでは、ドキュメント コメントは次の定義の前に記述できます。

- クラスやインターフェイスなどのユーザー定義型
- フィールド、イベント、プロパティ、関数、またはサブルーチンなどのメンバ

生成した XML ファイルを Visual Studio の IntelliSense 機能で使用するには、XML ファイルの名前をサポートするアセンブリの名前と同じにします。XML ファイルをアセンブリと同じディレクトリに格納して、アセンブリが Visual Studio プロジェクトで参照されるとき、.xml ファイルも同時に見つかるようにしてください。IntelliSense がプロジェクト内のコード、またはプロジェクトによって参照されるプロジェクト内のコードを処理するために、XML ドキュメント ファイルは必要ありません。

/target:module を使ってコンパイルしなければ、XML ファイルにはタグ `<assembly></assembly>` が記述されます。これらのタグは、コンパイルによる出力ファイルのアセンブリ マニフェストを含むファイル名を指定します。

コード内のコメントからドキュメントを生成する方法については、「ドキュメント コメントとして推奨される XML タグ (Visual Basic)」を参照してください。

Visual Studio 統合開発環境で /doc を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. [XML ドキュメント ファイルを生成する] ボックスに値を設定します。

使用例

具体的な例については、「XML の使用によるコードのドキュメントの作成 (Visual Basic)」を参照してください。

参照

概念

XML の使用によるコードのドキュメントの作成 (Visual Basic)

その他の技術情報

Visual Basic コンパイラ

/errorreport

Visual Basic コンパイラが内部コンパイラ エラーを出力する方法を指定します。

```
/errorreport:{ prompt | queue | send | none }
```

解説

このオプションは、Visual Basic 内部コンパイル エラー (ICE) を Microsoft の Visual Basic チームに報告するときの便利な方法として機能します。既定で、コンパイラは情報を Microsoft に送信しません。ただし、内部コンパイル エラーが発生した場合、このオプションを使って Microsoft に報告できます。マイクロソフトのエンジニアは、この情報を基に原因を特定し、Visual Basic の次のリリースの改善に役立てます。

ユーザーが使用できるレポート送信機能は、使用しているコンピュータやユーザー ポリシーのアクセス許可に応じて異なります。

次の表に、**/errorreport** オプションの働きをまとめます。

オプション	動作
prompt	内部コンパイル エラーが発生すると、コンパイラで収集されたデータがそのままの形でダイアログ ボックスに表示されます。エラー レポートに機密情報が含まれているかどうかを確認し、レポートを Microsoft に送信するかどうかを決定できます。送信することを決定し、コンピュータおよびユーザーのポリシー設定でもこの措置が許容される場合は、コンパイラからデータが Microsoft に送信されます。
queue	エラー レポートをキューに配置します。管理者権限を使ってログインするとポップアップ ウィンドウが表示され、前回のログイン以降に発生したエラーを報告できます。エラー レポートを送信するためのダイアログ ボックスは、3 日に 1 度表示されます。**/errorreport** オプションを指定しなかった場合の既定の動作がこれです。
send	内部コンパイル エラーが発生し、コンピュータおよびユーザーのポリシー設定で情報の送信が許容される場合は、コンパイラからデータが Microsoft に送信されます。
none	内部コンパイル エラーが発生した場合、エラーの情報を収集することも、Microsoft に情報を送信することもしません。

コンパイラからは、エラー発生時のスタックを含むデータが送信されます。スタックには、通常、一部のソース コードが含まれます。**/errorreport** を /bugreport オプションと一緒に指定すると、ソース ファイル全体が送信されます。

このオプションを /bugreport オプションと一緒に使用すると Microsoft のエンジニアがエラーを再現しやすくなるので、この組み合わせは最適です。

メモ：

/errorreport オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

次のコード例は、T2.vb をコンパイルしようとします。内部コンパイル エラーが発生すると、コンパイラからはエラー レポートを Microsoft に送信するかどうかを確認するメッセージが表示されます。

```
vbc /errorreport:prompt t2.vb
```

参照

関連項目

コンパイル コマンド ラインのサンプル

/bugreport

その他の技術情報

/filealign

出力ファイルのセクションを配置する場所を指定します。

```
/filealign:number
```

引数
number

必ず指定します。出力ファイル内のセクションの配置を指定する値です。有効な値は 512、1024、2048、4096、および 8192 です (単位はバイト)。

解説

出力ファイル内の各セクションの配置を指定するには、**/filealign** オプションを使用します。セクションは、エラー コードまたはデータが含まれる、ポータブル実行可能 (PE) ファイル内の連続するメモリのブロックです。**/filealign** オプションでは、非標準の配置でアプリケーションをコンパイルできます。このオプションが必要になるケースは多くありません。

各セクションは、**/filealign** の値の倍数となる境界上に配置されます。固定された既定値はありません。**/filealign** が指定されていない場合は、コンパイラによりコンパイル時に既定値が選択されます。

セクションのサイズを指定すると、出力ファイルのサイズが変化します。セクションのサイズの変更は、容量の小さなデバイス上で動作するプログラムに対して有効な場合があります。

メモ :

/filealign オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

参照
その他の技術情報
Visual Basic コンパイラ

/help、/?(Visual Basic)

コンパイラ オプションを表示します。

```
/help
' -or-
/?
```

解説

コンパイルでこのオプションを組み込むと、出力ファイルは作成されず、コンパイルも実行されません。

メモ：

/help オプションは Visual Studio 開発環境内では利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

コマンド ラインからヘルプを表示する場合のコード例です。

```
vbc /help
```

参照

関連項目

コンパイル コマンド ラインのサンプル

その他の技術情報

Visual Basic コンパイラ

/imports (Visual Basic)

指定のアセンブリから名前空間をインポートします。

```
/imports:namespaceList
```

引数
namespaceList

必ず指定します。インポートする名前空間のコンマで区切られたリストです。

解説

/imports オプションを使用すると、現在のソース ファイルのセット内または参照先アセンブリで定義されたすべての名前空間がインポートされます。

/imports で指定する名前空間のメンバは、コンパイルのすべてのソース コード ファイルで使用できます。1 つのソース コード ファイルで名前空間を使用するには、Imports ステートメント を使用します。

Visual Studio 統合開発環境で /imports を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [参照設定] タブをクリックします。
3. [ユーザー インポートの追加] ボタンの横に名前空間の名前を入力します。
4. [ユーザー インポートの追加] ボタンをクリックします。

使用例

/imports:system を指定するとコンパイルされる場合のコード例を次に示します。

VB

```
Module MyModule
    Sub Main()
        Console.WriteLine("test")
        ' Otherwise, would need
        ' System.Console.WriteLine("test")
    End Sub
End Module
```

参照
関連項目
コンパイル コマンド ラインのサンプル
概念
参照と Imports ステートメント
その他の技術情報
Visual Basic コンパイラ

/keycontainer

アセンブリに厳密な名前を付けるために、キー ペアのキー コンテナの名前を指定します。

```
/keycontainer:container
```

引数
container

必ず指定します。キーを含むコンテナ ファイル。ファイル名に空白が含まれる場合は、二重引用符 (" ") で囲む必要があります。

解説

コンパイラでは、アセンブリ マニフェストに公開キーを挿入し、秘密キーを使用して最後のアセンブリに署名することによって、共有可能コンポーネントが作成されます。キー ファイルを生成するには、コマンド ラインで「`sn -k` *file*」と入力します。`-i` オプションを使用すると、キー ペアがコンテナにインストールされます。詳細については、「厳密名ツール (Sn.exe)」を参照してください。

/target:module を使用してコンパイルすると、キー ファイル名はモジュールに保持され、/addmodule でアセンブリをコンパイルするときに作成されるアセンブリに組み込まれます。

このオプションは、任意の Microsoft Intermediate Language (MSIL) モジュールのソース コードで、カスタム属性 (AssemblyKeyNameAttribute) として指定することもできます。

/keyfile を使用して、暗号に関する情報をコンパイラに渡すこともできます。部分署名されたアセンブリを作成する場合は、/delaysign を使用します。

アセンブリに対する署名の詳細については、「厳密な名前付きアセンブリの作成と使用」を参照してください。

メモ :

/keycontainer オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

ソース ファイル `Input.vb` をコンパイルし、キー コンテナを指定する場合のコード例です。

```
vbc /keycontainer:key1 input.vb
```

参照
関連項目
/keyfile
コンパイル コマンド ラインのサンプル
概念
アセンブリ
その他の技術情報
Visual Basic コンパイラ

/keyfile

アセンブリに厳密な名前を付けるキーおよびキー ペアを含むファイルを指定します。

```
/keyfile:file
```

引数
file

必ず指定します。キーを含むファイル。ファイル名に空白が含まれる場合、ファイル名を引用符 (" ") で囲みます。

解説

コンパイラが、アセンブリ マニフェストに公開キーを挿入し、秘密キーを使用して最後のアセンブリに署名します。キー ファイルを生成するには、コマンド ラインで「`sn -k file`」と入力します。詳細については、「厳密名ツール (Sn.exe)」を参照してください。

/target:module を使用してコンパイルすると、キー ファイル名はモジュールに保持され、/addmodule でアセンブリをコンパイルするときに作成されるアセンブリに組み込まれます。

/keycontainer を使用して、暗号に関する情報をコンパイラに渡すこともできます。部分署名されたアセンブリを作成する場合は、/delaysign を使用します。

このオプションは、任意の Microsoft Intermediate Language (MSIL) モジュールのソース コードでカスタム属性 (AssemblyKeyFileAttribute) として指定することもできます。

コマンド ライン オプションまたはカスタム属性によって、コンパイル時に **/keyfile** と /keycontainer の両方が同時に指定されると、コンパイラは先にキー コンテナを処理します。コンテナが検出された場合、アセンブリはキー コンテナの情報で署名されます。キー コンテナを検出できなかった場合、コンパイラは **/keyfile** で指定されたファイルを検索します。キー ファイルが見つかった場合は、そのファイル内の情報を使用してアセンブリに署名され、次のコンパイル時にキー コンテナが有効になるように、キー情報がキー コンテナに組み込まれます (**sn -i** と同様)。

キー ファイルには、公開キーだけが含まれていることがあります。

アセンブリに対する署名の詳細については、「厳密な名前付きアセンブリの作成と使用」を参照してください。

メモ :

/keyfile オプションは Visual Studio 開発環境内では利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

ソース ファイル `Input.vb` をコンパイルし、キー ファイルを指定する場合のコード例です。

```
vbc /keyfile:myfile.sn input.vb
```

参照
関連項目
/reference (Visual Basic)
コンパイル コマンド ラインのサンプル
概念
アセンブリ
その他の技術情報
Visual Basic コンパイラ

/libpath

参照先アセンブリの場所を指定します。

```
/libpath:dirList
```

引数
dirList

必ず指定します。参照先アセンブリが現在の作業ディレクトリ (コンパイラを起動するディレクトリ) または共通言語ランタイムのシステム ディレクトリにない場合に、コンパイラが探すディレクトリのリストです。ディレクトリはセミコロンで区切って指定します。ディレクトリ名に空白が含まれる場合は、二重引用符 (" ") で囲む必要があります。

解説

/libpath オプションでは、/reference オプションによって参照されるアセンブリの場所を指定します。

コンパイラは、完全には修飾されていないアセンブリ参照を次の順序で検索します。

1. 現在の作業ディレクトリ。これは、コンパイラが起動されるディレクトリです。
2. 共通言語ランタイムのシステム ディレクトリ。
3. **/libpath** で指定したディレクトリ。
4. LIB 環境変数で指定したディレクトリ。

/libpath オプションは追加して指定できます。繰り返して指定すると前の値に追加されます。

アセンブリ参照を指定するには、**/reference** を使用します。

Visual Studio 統合開発環境で /libpath を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [参照設定] タブをクリックします。
3. [参照パス...] ボタンをクリックします。
4. [参照パス] ダイアログ ボックスで、[フォルダ:] ボックスにディレクトリ名を入力します。
5. [フォルダの追加] をクリックします。

使用例

`T2.vb` をコンパイルして .exe ファイルを作成する場合のコード例です。コンパイラはアセンブリを参照するために、まず作業ディレクトリ (C: ドライブのルートディレクトリ) を探し、次に C: ドライブの New Assemblies ディレクトリを探します。

```
vbc /libpath:c:\;"c:\New Assemblies" /reference:t2.dll t2.vb
```

参照
関連項目
コンパイル コマンド ラインのサンプル
概念
アセンブリ
その他の技術情報
Visual Basic コンパイラ

/linkresource (Visual Basic)

マネージ リソースへのリンクを作成します。

```
/linkresource:filename[,identifier[,public|private]]
' -or-
/linkres:filename[,identifier[,public|private]]
```

引数
filename

必ず指定します。アセンブリにリンクするリソース ファイルです。ファイル名に空白が含まれる場合、ファイル名を引用符 (" ") で囲みます。

identifier

省略可能です。リソースの論理名です。リソースを読み込むために使用する名前です。既定値はファイル名です。オプションとして、ファイルがアセンブリ マニフェスト内でパブリックかプライベートかを指定できます (例 : `/linkres:filename.res,myname.res,public`)。既定では、*filename* はアセンブリ内でパブリックです。

解説

/linkresource オプションは、リソース ファイルを出力ファイルに埋め込みません。リソース ファイルを出力ファイルに埋め込むには、**/resource** オプションを使用してください。

/linkresource オプションでは、**/target:module** 以外のいずれかの **/target** オプションが必要です。

たとえば、*filename* が リソース ファイル ジェネレータ (Resgen.exe) または開発環境で作成された .NET Framework リソース ファイルである場合は、System.Resources 名前空間のメンバを使用してアクセスできます。詳細については、「ResourceManager」を参照してください。実行時にその他のすべてのリソースにアクセスするには、Assembly クラスで **GetManifestResource** で始まるメソッドを使用します。

ファイル名はどのようなファイル形式でもかまいません。たとえば、ネイティブ DLL をアセンブリの一部として含め、そのネイティブ DLL をグローバル アセンブリ キャッシュにインストールして、アセンブリ内のマネージ コードからアクセスできるようにすることもできます。

/linkres は **/linkresource** の省略形です。

メモ :

/linkresource オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

次に、`In.vb` をコンパイルしてリソース ファイル `Rf.resource` にリンクする場合のコード例を示します。

```
vbc /linkresource:rf.resource in.vb
```

参照
関連項目
/target (Visual Basic)
/resource (Visual Basic)
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/main

Sub Main プロシージャを含むクラスまたはモジュールを指定します。

```
/main:location
```

引数
location

必ず指定します。プログラムの起動時に呼び出される **Sub Main** プロシージャを含むクラスまたはモジュールの絶対パス名です。"**/main:module**" または "**/main:namespace.module**" の形式で指定します。

解説

実行可能ファイルまたは Windows 実行可能プログラムを作成するときは、このオプションを使用します。**/main** オプションを省略すると、コンパイラは、すべてのパブリック クラスおよびモジュールで有効な共有 **Sub Main** を検索します。

Main プロシージャのさまざまな形式に関しては、「Visual Basic の Main プロシージャ」を参照してください。

Form を継承するクラスが *location* に指定され、そのクラスに **Main** プロシージャが存在しなかった場合は、アプリケーションを起動する既定の **Main** プロシージャがコンパイラによって提供されます。これにより、開発環境で作成したコードをコマンド ラインでコンパイルできるようになります。

VB

```
' Compile with /r:System.dll,SYSTEM.WINDOWS.FORMS.DLL /main:MyC
Public Class MyC
    Inherits System.Windows.Forms.Form
End Class
```

Visual Studio 統合開発環境で /main を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。

   詳細については、「プロジェクト デザイナの概要」を参照してください。

2. [アプリケーション] タブをクリックします。

3. [アプリケーション フレームワークを有効にする] チェック ボックスがオンになっていないことを確認してください。

4. [スタートアップ オブジェクト] ボックス内の値を変更します。

使用例

`T2.vb` および `T3.vb` をコンパイルし、**Sub Main** プロシージャが `Test2` クラスにあることを指定する場合のコード例です。

```
vbc t2.vb t3.vb /main:Test2
```

参照
関連項目
/target (Visual Basic)
コンパイル コマンド ラインのサンプル
概念
Visual Basic バージョンの Hello World!
Visual Basic の Main プロシージャ
その他の技術情報
Visual Basic コンパイラ

/netcf

.NET Compact Framework を対象とするようにコンパイラを設定します。

```
/netcf
```

解説

/netcf オプションでは、Visual Basic コンパイラが完全な .NET Framework ではなく .NET Compact Framework を対象にします。.NET Framework だけで使用できる言語機能は無効になります。

/netcf オプションは /sdkpath と組み合わせて使用するよう設計されています。**/netcf** によって無効になる言語機能は、**/sdkpath** の対象となるファイルで使用できる言語機能と同じです。

メモ：

/netcf オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。**/netcf** オプションは Visual Basic デバイス プロジェクトがロードされている場合に設定されます。

/netcf オプションは、次の言語機能を変更します。

- プログラムの実行を終了する End (Visual Basic) キーワードが無効になります。次のプログラムは、**/netcf** なしでコンパイルおよび実行可能ですが、**/netcf** を指定するとコンパイル時に失敗します。

  VB

```
Module Module1
    Sub Main()
        End   ' not valid to terminate execution with /netcf
    End Sub
End Module
```

- すべての形式の遅延バインディング。認識されている遅延バインディングのシナリオと同じ状況になると、コンパイル時エラーが発生します。次のプログラムは、**/netcf** なしでコンパイルおよび実行可能ですが、**/netcf** を指定するとコンパイル時に失敗します。

  VB

```
Class LateBoundClass
    Sub S1()
    End Sub

    Default Property P1(ByVal s As String) As Integer
        Get
        End Get
        Set(ByVal Value As Integer)
        End Set
    End Property
End Class

Module Module1
    Sub Main()
        Dim o1 As Object
        Dim o2 As Object
        Dim o3 As Object
        Dim IntArr(3) As Integer

        o1 = New LateBoundClass
        o2 = 1
```

```
        o3 = IntArr

        ' Late-bound calls
        o1.S1()
        o1.P1("member") = 1

        ' Dictionary member access
        o1!member = 1

        ' Late-bound overload resolution
        LateBoundSub(o2)

        ' Late-bound array
        o3(1) = 1
    End Sub

    Sub LateBoundSub(ByVal n As Integer)
    End Sub

    Sub LateBoundSub(ByVal s As String)
    End Sub
End Module
```

- Auto、Ansi、Unicode (Visual Basic) 修飾子が無効になります。Declare ステートメント ステートメントの構文も `Declare Sub|Function name Lib "library" [Alias "alias"] [([arglist])]` に変更されます。**/netcf** によるコンパイルへの影響を、次のコードで示します。

  VB

```
' compile with: /target:library
Module Module1
    ' valid with or without /netcf
    Declare Sub DllSub Lib "SomeLib.dll" ()

    ' not valid with /netcf
    Declare Auto Sub DllSub1 Lib "SomeLib.dll" ()
    Declare Ansi Sub DllSub2 Lib "SomeLib.dll" ()
    Declare Unicode Sub DllSub3 Lib "SomeLib.dll" ()
End Module
```

- Visual Basic で削除された Visual Basic 6.0 のキーワードを使用すると、**/netcf** を使用した場合に別のエラーが発生します。これにより、次のキーワードに対するエラー メッセージが影響を受けます。
  - **Open**
  - **Close**
  - **Put**
  - **Print**
  - **Write**
  - **Input**
  - **Lock**
  - **Unlock**
  - **Seek**
  - **Width**

- **Name**
- **FreeFile**
- **EOF**
- **Loc**
- **LOF**
- **Line**

使用例

次のコードでは、C ドライブ上の .NET Compact Framework の既定のインストール ディレクトリにある Mscorlib.dll のバージョンと Microsoft.VisualBasic.dll のバージョンを使用して、.NET Compact Framework で `Myfile.vb` をコンパイルします。通常、最近のバージョンの .NET Compact Framework を使用します。

```
vbc /netcf /sdkpath:"c:\Program Files\Microsoft Visual Studio .NET 2003\CompactFrameworkSDK
\v1.0.5000\Windows CE " myfile.vb
```

参照

関連項目

コンパイル コマンド ラインのサンプル

/sdkpath

その他の技術情報

Visual Basic コンパイラ

/noconfig

よく使用される .NET Framework アセンブリを自動的に参照したり、**System** および **Microsoft.VisualBasic** 名前空間をインポートしたりしないようコンパイラに指示します。

```
/noconfig
```

解説

/noconfig オプションは、コンパイルに Vbc.rsp ファイルを使用しないようにコンパイラに指示します。Vbc.rsp ファイルは、Vbc.exe ファイルと同じディレクトリにあります。Vbc.rsp ファイルはよく使用される .NET Framework アセンブリを参照し、**System** 名前空間と **Microsoft.VisualBasic** 名前空間をインポートします。コンパイラは **/nostdlib** オプションが指定されている場合を除き、System.dll アセンブリを暗黙的に参照します。**/nostdlib** オプションを指定すると、コンパイラは Vbc.rsp を使ってコンパイラしなくなるか、または System.dll アセンブリを自動的に参照しなくなります。

メモ：
Mscorlib.dll アセンブリおよび Microsoft.VisualBasic.dll アセンブリは、常に参照されます。

Vbc.rsp ファイルを変更して、すべての Vbc.exe のコンパイルで使用する追加のコンパイラ オプションを指定できます (**/noconfig** オプションが指定されている場合を除きます)。詳細については、「@ (応答ファイルの指定) (Visual Basic)」を参照してください。

コンパイラは、**vbc** コマンドに渡されるオプションを最後に処理します。このため、コマンド ラインに指定したオプションは、Vbc.rsp ファイル内の同じオプションの設定をオーバーライドします。

メモ：
/noconfig オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

参照

関連項目

/nostdlib (Visual Basic)
@ (応答ファイルの指定) (Visual Basic)
/reference (Visual Basic)

その他の技術情報

Visual Basic コンパイラ

/nologo (Visual Basic)

コンパイル時の著作権画面や情報メッセージの表示を中止します。

```
/nologo
```

解説

/nologo を指定すると、コンパイラは著作権画面を表示しません。既定では、**/nologo** は無効です。

メモ：

/nologo オプションは Visual Studio 2005 の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

T2.vb を、著作権画面を表示しないように指定してコンパイルするコード例を次に示します。

```
vbc /nologo t2.vb
```

参照

関連項目

コンパイル コマンド ラインのサンプル

その他の技術情報

Visual Basic コンパイラ

/nostdlib (Visual Basic)

コンパイラが標準のライブラリを自動的に参照しないよう指定します。

```
/nostdlib
```

解説

/nostdlib オプションは、System.dll アセンブリへの自動参照を削除して、コンパイラが Vbc.rsp ファイルを読み取らないようにします。Vbc.exe ファイルと同じディレクトリにある Vbc.rsp ファイルは、一般に使用される .NET Framework アセンブリを参照し、**System** 名前空間および **Microsoft.VisualBasic** 名前空間をインポートします。

メモ：

Mscorlib.dll アセンブリおよび Microsoft.VisualBasic.dll アセンブリは、常に参照されます。

メモ：

/nostdlib オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

次のコードは、標準ライブラリを参照しないで `T2.vb` をコンパイルします。**_MYTYPE** 条件付きコンパイル定数を文字列 "Empty" に設定して、**My** オブジェクトを削除する必要があります。

```
vbc /nostdlib /define:_MYTYPE=\"Empty\" T2.vb
```

参照

関連項目

/noconfig

コンパイル コマンド ラインのサンプル

概念

My で利用可能なオブジェクトのカスタマイズ

その他の技術情報

Visual Basic コンパイラ

/nowarn

警告を生成するコンパイラの機能が表示されないようにします。

```
/nowarn[:numberList]
```

引数
numberList

省略可能です。コンパイラが省略する警告の ID 番号をコンマ区切りで指定したリストです。警告の ID が指定されていない場合、すべての警告が省略されます。

解説

/nowarn オプションを指定すると、コンパイラは警告を生成しません。個別の警告を省略するには、**/nowarn** オプションに、コロンに続けて警告の ID を 指定します。警告番号が複数ある場合は、コンマで区切ります。

必要なのは、警告 ID の数値を指定することだけです。たとえば、使用されていないローカル変数の警告である BC42024 を省略するには、`/nowarn:42024` と指定します。

警告の ID 番号の詳細については、「Visual Basic での警告の構成」を参照してください。

Visual Studio 統合開発環境で /nowarn を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. すべての警告を無効にするには、[すべての警告を表示しない] チェック ボックスをオンにします。

   または

   特定の警告を無効にするには、警告の隣にあるドロップダウン リストの [なし] をクリックします。

使用例

`T2.vb` をコンパイルし、警告を表示しないようにする場合のコード例です。

```
vbc /nowarn t2.vb
```

次のコードでは、`T2.vb` をコンパイルし、使われていないローカル変数の警告 (42024) を表示しません。

```
vbc /nowarn:42024 t2.vb
```

参照
関連項目
コンパイル コマンド ラインのサンプル
概念
Visual Basic での警告の構成
その他の技術情報
Visual Basic コンパイラ

/optimize

コンパイラの最適化を有効または無効にします。

```
/optimize[ + | - ]
```

引数
+ | -

省略可能です。**/optimize-** オプションを指定すると、コンパイラの最適化は無効になります。**/optimize+** オプションを指定すると、最適化は有効になります。既定では、最適化は無効です。

解説

コンパイラを最適化すると、出力ファイルのサイズが小さくなり、動作が速くなり、処理の効率が向上します。ただし、最適化を行うと出力ファイルにおいてコードの配置が変更されるので、**/optimize+** を指定するとデバッグは困難になります。

/target:module でアセンブリに生成されたすべてのモジュールは、アセンブリと同じ **/optimize** の設定を使用する必要があります。詳細については、「/target (Visual Basic)」を参照してください。

/optimize オプションと **/debug** オプションを組み合わせることができます。

Visual Studio 統合開発環境で /optimize を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。

   詳細については、「プロジェクト デザイナの概要」を参照してください。

2. [コンパイル] タブをクリックします。

3. [詳細] をクリックします。

4. [最適化を有効にする] チェック ボックスを変更します。

使用例

T2.vb をコンパイルし、コンパイラの最適化を有効にする場合のコード例です。

```
vbc t2.vb /optimize
```

参照
関連項目
/debug (Visual Basic)
コンパイル コマンド ラインのサンプル
/target (Visual Basic)
その他の技術情報
Visual Basic コンパイラ

/optioncompare

文字列比較の方法を指定します。

```
/optioncompare:{binary | text}
```

解説

/optioncompare の指定には 2 つの形式があります。バイナリ モードの文字列比較を使用するときは **/optioncompare:binary** を指定し、テキスト モードの文字列比較を使用するときは **/optioncompare:text** を指定します。既定では、コンパイラは **/optioncompare:binary** を使用します。

Microsoft Windows では、バイナリの並べ替え順序の判別にコード ページが使用されます。通常、バイナリの並べ替え順序は次のようになります。

```
A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø
```

テキスト ベースの文字列比較は、大文字と小文字を区別しない、システムのロケールで決められたテキストの並べ替え順序に基づいて行われます。通常、テキストの並べ替え順序は次のようになります。

```
(A = a) < (À = à) < (B=b) < (E=e) < (Ê = ê) < (Z=z) < (Ø = ø)
```

Visual Studio 統合開発環境で /optioncompare を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. [Option Compare] ボックスの値を変更します。

プログラムで /optioncompare を設定するには

- Option Compare ステートメント を参照してください。

使用例

ProjFile.vb をコンパイルし、バイナリ モードで文字列比較を行う場合のコード例です。

```
vbc /optioncompare:binary projFile.vb
```

参照

関連項目

/optionexplicit
/optionstrict
コンパイル コマンド ラインのサンプル
Option Compare ステートメント
[Visual Basic の既定値] ([オプション] ダイアログ ボックス - [プロジェクト])

その他の技術情報

Visual Basic コンパイラ

/optionexplicit

宣言されていない変数を使用している場合に、コンパイラによってエラーが報告されます。

```
/optionexplicit[+ | -]
```

引数
+ | -

省略可能です。**/optionexplicit+** を指定すると、変数の明示的な宣言が必要になります。既定では **/optionexplicit+** オプションになります。これは **/optionexplicit** を指定するのと同じです。**/optionexplicit-** オプションを指定すると、変数を暗黙的に宣言できます。

解説

Visual Studio 統合開発環境で /optionexplicit を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. [Option Explicit] ボックスの値を変更します。

プログラムで /optionexplicit を設定するには

- Option Explicit ステートメント (Visual Basic) を参照してください。

使用例

次のコードは、**/optionexplicit-** が指定されている場合にコンパイルされます。

VB

```
Module Module1
    Sub Main()
        i = 99
        System.Console.WriteLine(i)
    End Sub
End Module
```

参照
関連項目
/optioncompare
/optionstrict
コンパイル コマンド ラインのサンプル
Option Explicit ステートメント (Visual Basic)
[Visual Basic の既定値] ([オプション] ダイアログ ボックス - [プロジェクト])
その他の技術情報
Visual Basic コンパイラ

/optionstrict

暗黙の型変換を制限するための、厳密な型の意味を適用します。

```
/optionstrict[+ | -]
/optionstrict[:custom]
```

引数
+ | -

省略可能です。**/optionstrict+** を指定すると、暗黙の型変換が制限されます。このオプションの既定値は **/optionstrict-** です。**/optionstrict+** オプションは、**/optionstrict** と同じ動作になります。寛容な型指定規則では、どちらも使用できます。

custom

必ず指定します。厳密な言語規則が守られていない場合に警告します。

解説

/optionstrict+ を指定すると、拡張型変換だけが暗黙のうちに行われます。10 進型 (**Decimal**) のオブジェクトを整数型 (Integer) のオブジェクトに代入するなど、縮小による暗黙の型変換はエラーとして報告されます。

暗黙の縮小変換で警告を生成するには、**/optionstrict:custom** を使用します。特定の警告を無視するには **/nowarn:numberlist**、特定の警告をエラーとして扱うようにするには **/warnaserror:numberlist** を使用します。

Visual Studio IDE で /optionstrict を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. [Option Strict] ボックス内の値を変更します。

プログラムで /optionstrict を設定するには

- Option Strict ステートメント を参照してください。

使用例

厳密な型指定規則を使用して `Test.vb` をコンパイルする場合のコード例です。

```
vbc /optionstrict+ test.vb
```

参照
関連項目
/optioncompare
/optionexplicit
/nowarn
/warnaserror (Visual Basic)
コンパイル コマンド ラインのサンプル
Option Strict ステートメント
[Visual Basic の既定値] ([オプション] ダイアログ ボックス - [プロジェクト])
その他の技術情報
Visual Basic コンパイラ

/out (Visual Basic)

出力ファイルの名前を指定します。

```
/out:filename
```

引数
filename

必ず指定します。コンパイラで作成される出力ファイルの名前。ファイル名に空白が含まれる場合、ファイル名を引用符 (" ") で囲みます。

解説

コンパイラは、**/out** オプションの後に指定されたファイル名を出力ファイルとして使用します。

作成するファイルのフルネームと拡張子を指定します。ファイル名を指定しないと、.exe ファイルの場合は **Sub Main** プロシージャを含むソース コード ファイルの名前が使用され、.dll ファイルの場合は最初のソース コード ファイルの名前が使用されます。

Visual Studio 統合開発環境で /out を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [アプリケーション] タブをクリックします。
3. [アセンブリ名] ボックス内の値を変更します。

使用例

`T2.vb` をコンパイルして出力ファイル `T2.exe` を作成する場合のコード例です。

```
vbc t2.vb /out:t3.exe
```

参照
関連項目
/target (Visual Basic)
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/platform (Visual Basic)

出力ファイルをどのプラットフォーム用の共通言語ランタイム (CLR) で実行するかを指定します。

```
/platform:{ x86 | x64 | Itanium | anycpu }
```

引数

anycpu

任意のプラットフォームで実行するアセンブリをコンパイルします。**/platform** オプションを指定しなかった場合の既定の動作です。

x86

32 ビットの x86 互換 CLR で実行されるアセンブリをコンパイルします。

x64

AMD64 または EM64T 命令セットをサポートするコンピュータ上の 64 ビット CLR で実行されるアセンブリをコンパイルします。

Itanium

Itanium プロセッサ搭載コンピュータ上の 64 ビット CLR で実行されるアセンブリをコンパイルします。

解説

/platform オプションでは、出力ファイルの実行対象となるプロセッサの種類を指定します。

通常、Visual Basic で記述された .NET Framework アセンブリは、プラットフォームに関係なく同じように実行されます。ただし、プラットフォーム間で動作の異なる場合があります。その一般的な例を次に示します。

- プラットフォームによってメンバのサイズが変わる構造体 (ポインタ型など)
- 定数のサイズを含むポインタ演算
- ハンドルに IntPtr ではなく **Integer** を使用した不適切なプラットフォーム呼び出しまたは COM 宣言
- **IntPtr** から **Integer** へのキャスト
- すべてのプラットフォームに存在するとは限らないコンポーネントを使用したプラットフォーム呼び出しまたは COM 相互運用機能

コードの実行対象となるアーキテクチャをあらかじめ想定している場合は、**/platform** オプションを使用することで、問題が緩和されることもあります。具体的には、以下の警告があります。

- 64 ビット プラットフォームを対象とすることにし、アプリケーションを 32 ビット コンピュータで実行した場合、このスイッチを使わなかった場合よりも早い段階でエラー メッセージが生成され、また、メッセージ自体もプラットフォームに特化した具体的な内容になります。
- このスイッチで 32 ビット オプションを設定し、その後、アプリケーションを 64 ビット コンピュータで実行した場合、ネイティブな実行環境ではなく、WOW サブシステムで実行されます。

64 ビットの Windows オペレーティング システムでは次のようになります。

- **/platform:x86** を指定してコンパイルしたアセンブリは、WOW64 環境の 32 ビット CLR 上で実行されます。
- **/platform:anycpu** を指定してコンパイルした実行可能ファイルは、64 ビット CLR 上で実行されます。
- **/platform:anycpu** を指定してコンパイルした DLL は、ロード先のプロセスと同じ CLR 上で実行されます。

64 ビットの Windows オペレーティング システム上で実行されるアプリケーションの開発の詳細については、「64 ビット アプリケーション」を参照してください。

Visual Studio 統合開発環境で /nowarn を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。

3. [詳細] をクリックします。

4. [ターゲット CPU] ボックスの値を変更します。

使用例

次の例は、**/platform** を使用して、Itanium 搭載の 64 ビット Windows オペレーティング システム上の 64 ビット CLR でのみ実行されるアプリケーションをコンパイルする方法を示しています。

```
vbc /platform:Itanium myItanium.vb
```

参照

関連項目

コンパイル コマンド ラインのサンプル

その他の技術情報

Visual Basic コンパイラ

/quiet

構文関連のエラーおよび警告に関するコードが表示されないようにします。

```
/quiet
```

解説

既定では、**/quiet** は無効です。コンパイラは、構文関連のエラーまたは警告を報告するときに、ソース コード行も出力します。コンパイラの出力を解析するアプリケーションの場合、診断テキストだけがコンパイラから出力される方が便利な場合があります。

次の例の場合、**/quiet** を指定せずにコンパイルを行うと、`Module1` からはソース コードを含むエラーが出力されます。

```
Module Module1
    Sub Main()
        x()
    End Sub
End Module
```

出力

```
E:\test\t2.vb(3) : error BC30451: Name 'x' is not declared.

x

~
```

/quiet を指定してコンパイルを行うと、コンパイラからの出力は次の行だけになります。

```
E:\test\t2.vb(3) : error BC30451: Name 'x' is not declared.
```

メモ：

/quiet オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

次に示すのは、`T2.vb` をコンパイルし、構文関連のコンパイラ診断に対してコードを表示しない場合のコード例です。

```
vbc /quiet t2.vb
```

参照

関連項目

コンパイル コマンド ラインのサンプル

その他の技術情報

Visual Basic コンパイラ

/recurse

指定ディレクトリまたはプロジェクト ディレクトリのすべての子ディレクトリ内のソース コード ファイルをコンパイルします。

```
/recurse:[dir\]file
```

引数

dir

省略可能です。検索を開始するディレクトリ。ディレクトリを指定しない場合は、プロジェクト ディレクトリから検索されます。

file

必ず指定します。検索するファイル。ワイルドカード文字を使用できます。

解説

/recurse を使用しなくても、ファイル名にワイルドカードを使用すると、プロジェクト ディレクトリ内で一致するすべてのファイルをコンパイルできます。出力ファイル名を指定しないと、処理する最初の入力ファイルに基づいて、コンパイラで出力ファイル名が指定されます。通常は、コンパイル対象のファイルの一覧をアルファベット順に表示したときの最初のファイル名が使用されます。このため、**/out** オプションを使用して出力ファイルを指定することをお勧めします。

メモ：

/recurse オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

現在のディレクトリにあるすべての Visual Basic ファイルをコンパイルする場合のコード例です。

```
vbc *.vb
```

`Test\ABC` ディレクトリおよびその下の全ディレクトリ内の Visual Basic ファイルをすべてコンパイルし、`Test.ABC.dll` を生成する場合のコード例を次に示します。

```
vbc /target:library /out:Test.ABC.dll /recurse:Test\ABC\*.vb
```

参照

関連項目

/out (Visual Basic)

コンパイル コマンド ラインのサンプル

その他の技術情報

Visual Basic コンパイラ

/reference (Visual Basic)

指定のアセンブリ内の型情報をコンパイル中のプロジェクトで使用できるようにします。

```
/reference:fileList
' -or-
/r:fileList
```

引数
fileList

必ず指定します。アセンブリのファイル名をコンマで区切ったリストです。ファイル名に空白が含まれる場合は、二重引用符 (" ") で囲む必要があります。

解説

インポートするファイルは、アセンブリ メタデータを含んでいる必要があります。アセンブリの外側ではパブリックな型だけが参照できます。/addmodule は、モジュールからメタデータをインポートします。

別のアセンブリ (Assembly B) を参照するアセンブリ (Assembly A) を参照するときに、アセンブリ B も参照する必要があるのは、次の場合です。

- アセンブリ A の型がアセンブリ B の型を継承しているか、アセンブリ B のインターフェイスを実装している場合。
- アセンブリ B の戻り値の型やパラメータの型を持つフィールド、プロパティ、イベント、またはメソッドを呼び出す場合。

/libpath を使用して、1 つ以上のアセンブリ参照があるディレクトリを指定します。

コンパイラが (モジュールではなく) アセンブリ内の型を認識するためには、その型をコンパイラに解決させる必要があります。これには、たとえばその型のインスタンスを定義するなどの方法があります。コンパイラのためにアセンブリ内の型名を解決する方法は他にもあります。たとえば、アセンブリ内の型を継承すると、コンパイラはその型名を認識できるようになります。

既定では、頻繁に使用される .NET Framework アセンブリを参照する Vbc.rsp 応答ファイルが使用されます。コンパイラで Vbc.rsp を使用しない場合は、**/noconfig** を使用します。

/reference の省略形は **/r** です。

Visual Studio 統合開発環境で /reference を設定するには

- [参照の追加] ダイアログ ボックス を参照してください。

使用例

ソース ファイル `Input.vb` と、`Metad1.dll` および `Metad2.dll` からの参照アセンブリをコンパイルし、`Out.exe` を生成する場合のコード例です。

```
vbc /reference:metad1.dll,metad2.dll /out:out.exe input.vb
```

参照
関連項目
/noconfig
/target (Visual Basic)
Public (Visual Basic)
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/removeintchecks

整数演算のオーバーフロー エラー チェックのオンとオフを切り替えます。

```
/removeintchecks[+ | -]
```

引数
+ | -

省略可能です。**/removeintchecks-** オプションを指定すると、すべての整数の計算について、オーバーフローや 0 による除算などのエラーがコンパイラによってチェックされます。既定のクラスは **/removeintchecks-** です。

/removeintchecks または **/removeintchecks+** を指定すると、エラー チェックは行われず、整数の計算時間が短縮されます。ただし、エラー チェックを行わないと、データ型の容量がオーバーフローしてもエラーが通知されず、誤った結果が格納される場合があります。

Visual Studio 統合開発環境で /removeintchecks を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. [詳細] をクリックします。
4. [整数オーバーフローのチェックを解除] ボックスの値を変更します。

使用例

`Test.vb` をコンパイルし、整数のオーバーフロー エラー チェックをオフにする場合のコード例を次に示します。

```
vbc /removeintchecks+ test.vb
```

参照
関連項目
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/resource (Visual Basic)

マネージ リソースをアセンブリに埋め込みます。

```
/resource:filename[,identifier[,public|private]]
' -or-
/res:filename[,identifier[,public|private]]
```

引数
filename

必ず指定します。出力ファイルに埋め込むリソース ファイルの名前。既定では、*filename* はアセンブリ内でパブリックです。ファイル名に空白が含まれている場合は、二重引用符 (" ") で囲みます。

identifier

省略可能です。リソースの論理名。リソースを読み込むときに使用します。既定値はファイル名です。オプションとして、**/res:***filename.res*、*myname.res*、**public** では、リソースがアセンブリ マニフェスト内でパブリックかプライベートかを指定できます。

解説

リソース ファイルを出力ファイルに組み込まずにリソースをアセンブリにリンクするには、**/linkresource** オプションを使用します。

たとえば、*filename* が リソース ファイル ジェネレータ (Resgen.exe) または開発環境で作成された .NET Framework リソース ファイルである場合は、System.Resources 名前空間のメンバを使用してアクセスできます (詳細については「ResourceManager」を参照してください)。実行時に他のすべてのリソースにアクセスするには、GetManifestResourceInfo、GetManifestResourceNames、またはGetManifestResourceStream のいずれかのメソッドを使用します。

/res は **/resource** の省略形です。

Visual Studio 統合開発環境で /resource を設定するには

- [リソース] ページ (プロジェクト デザイナ) を参照してください。

使用例

`In.vb` をコンパイルし、リソース ファイル `Rf.resource` をアタッチする場合のコード例です。

```
vbc /res:rf.resource in.vb
```

参照
関連項目
/win32resource
/linkresource (Visual Basic)
/target (Visual Basic)
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/rootnamespace

すべての型宣言の名前空間を指定します。

```
/rootnamespace:namespace
```

引数
namespace

現在のプロジェクトのすべての型宣言を含める名前空間の名前。

解説

Visual Studio 実行可能ファイル (Devenv.exe) を使用して Visual Studio 統合開発環境で作成されたプロジェクトをコンパイルする場合、**/rootnamespace** を使用して RootNamespace プロパティの値を指定します。詳細については、「Devenv コマンド ライン スイッチ」を参照してください。

出力ファイルの名前空間の名前を確認するには、共通言語ランタイムの MSIL 逆アセンブラ (`ldasm.exe`) を使用します。

Visual Studio 統合開発環境で /rootnamespace を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [アプリケーション] タブをクリックします。
3. [ルート名前空間] ボックス内の値を変更します。

使用例

`In.vb` をコンパイルし、すべての型宣言を名前空間 `mynamespace` に含める場合のコード例です。

```
vbc /rootnamespace:mynamespace in.vb
```

参照
関連項目
MSIL 逆アセンブラ (Ildasm.exe)
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/sdkpath

Mscorlib.dll と Microsoft.VisualBasic.dll の場所を指定します。

```
/sdkpath:path
```

引数
path

Mscorlib.dll および Microsoft.VisualBasic.dll のコンパイルに使用するバージョンを格納するディレクトリ。このパスは、読み込まれるまで検査されません。ディレクトリ名に空白が含まれている場合は、文字列を二重引用符 (" ") で囲みます。

解説

このオプションを指定すると、Visual Basic のコンパイラは既定の場所以外から Mscorlib.dll および Microsoft.VisualBasic.dll の各ファイルを読み込みます。**/sdkpath** オプションは /netcf と組み合わせて使用するよう設計されています。.NET Compact Framework は、これらのサポート ライブラリの別バージョンを使用して、デバイスで見つからない型や言語機能を使用することを回避します。

メモ：

/sdkpath オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。**/sdkpath** オプションは Visual Basic デバイス プロジェクトがロードされている場合に設定されます。

使用例

次のコードでは、C ドライブ上の .NET Compact Framework の既定のインストール ディレクトリにある Mscorlib.dll のバージョンと Microsoft.VisualBasic.dll のバージョンを使用して、.NET Compact Framework で `Myfile.vb` をコンパイルします。通常、最近のバージョンの .NET Compact Framework を使用します。

```
vbc /netcf /sdkpath:"c:\Program Files\Microsoft Visual Studio .NET 2003\CompactFrameworkSDK
\v1.0.5000\Windows CE " myfile.vb
```

参照
関連項目
コンパイル コマンド ラインのサンプル
/netcf
その他の技術情報
Visual Basic コンパイラ

/target (Visual Basic)

コンパイラの出力形式を指定します。

```
/target:{exe | library | module | winexe}
```

解説

次の表に、**/target** オプションの働きをまとめます。

オプション	動作
/target:exe	実行可能なコンソール アプリケーションをコンパイラで作成します。 **/target** オプションを指定しなかった場合の既定のオプションです。拡張子 .exe を使って実行可能ファイルが作成されます。 **/out** オプションで特に指定しない限り、出力ファイル名は **Sub Main** プロシージャを含む入力ファイルの名前と同じになります。 .exe ファイルを生成するためにコンパイルするソース コード ファイルで必要な **Sub Main** プロシージャは 1 つだけです。**/main** コンパイラ オプションを使用して、**Sub Main** プロシージャを含むクラスを指定します。
/target:library	コンパイラにダイナミック リンク ライブラリ (DLL) を作成させます。 拡張子が .dll であるダイナミック リンク ライブラリ ファイルが作成されます。 **/out** オプションで特に指定しない限り、出力ファイル名は最初の入力ファイルと同じになります。 DLL の作成には、**Sub Main** プロシージャは不要です。
/target:module	アセンブリに追加できるモジュールをコンパイラで生成します。 出力ファイルの拡張子は .netmodule です。 .NET の共通言語ランタイムは、アセンブリのないファイルを読み込むことができません。ただし、アセンブリがないファイルでも、**/reference** を使用してアセンブリのアセンブリ マニフェストに組み込むことができます。 あるモジュールのコードが別のモジュールの内部型を参照する場合は、**/reference** を使用して両方のモジュールをアセンブリ マニフェストに組み込む必要があります。 /addmodule は、モジュールからメタデータをインポートします。
/target:winexe	実行可能な Windows プログラムをコンパイラで作成します。 拡張子 .exe を使って実行可能ファイルが作成されます。Windows プログラムは、.NET Framework クラス ライブラリまたは Win32 API のユーザー インターフェイスを提供するプログラムです。 **/out** オプションで特に指定しない限り、出力ファイル名は **Sub Main** プロシージャを含む入力ファイルの名前と同じになります。 .exe ファイルを生成するためにコンパイルするソース コード ファイルで必要な **Sub Main** プロシージャは 1 つだけです。コード内に **Sub Main** プロシージャを持つクラスが複数ある場合、**/main** コンパイラ オプションを使用して、どのクラスに **Sub Main** プロシージャが含まれているかを指定します。

/target:module を指定しない限り、**/target** を使用すると、.NET Framework のアセンブリ マニフェストが出力ファイルに組み込まれます。

Vbc.exe のインスタンスごとに生成される出力ファイルは、多くても 1 つです。**/out** や **/target** のようなコンパイラ オプションを 2 回以上指定すると、コンパイラが最後に認識したオプションだけが有効になります。コンパイルされたすべてのファイルに関する情報は、マニフェストに収められます。**/target:module** で作成されるファイル以外のすべての出力ファイルでは、マニフェスト内にアセンブリ メタデータが格納されます。出力ファイルのメタデータを表示するには、MSIL 逆アセンブラ (Ildasm.exe) を使用します。

/t は **/target** の省略形です。

Visual Studio 統合開発環境で /target を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、

「プロジェクト デザイナの概要」を参照してください。

2. [アプリケーション] タブをクリックします。
3. [アプリケーションの種類] ボックス内の値を変更します。

メモ：
/target:module オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

`in.vb` をコンパイルして `in.dll` を作成する場合のコード例です。

```
vbc /target:library in.vb
```

参照

関連項目

/main
/out (Visual Basic)
/reference (Visual Basic)
/addmodule
コンパイル コマンド ラインのサンプル

概念

アセンブリ

その他の技術情報

Visual Basic コンパイラ

/utf8output (Visual Basic)

UTF-8 エンコーディングを使用してコンパイラ出力を表示します。

```
/utf8output[+ | -]
```

引数

+ | -

省略可能です。このオプションの既定値は **/utf8output-** で、コンパイラの出力では UTF-8 エンコーディングは使用されません。**/utf8output** の指定は、**/utf8output+** の指定と同じです。

解説

いくつかの国際対応の設定では、コンパイラの出力をコンソールに正しく表示できません。そのような場合は、**/utf8output** を使用してコンパイラの出力をファイルにリダイレクトできます。

メモ：

/utf8output オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

`In.vb` をコンパイルし、コンパイラの出力を UTF-8 エンコードで表示する場合のコード例です。

```
vbc /utf8output in.vb
```

参照

関連項目

コンパイル コマンド ラインのサンプル

その他の技術情報

Visual Basic コンパイラ

/verbose

コンパイラで、詳細なステータスとエラー メッセージを生成します。

```
/verbose[+ | -]
```

引数

+ | -

省略可能です。**/verbose** を指定すると **/verbose+** を指定した場合と同じになり、詳細なメッセージが生成されます。このオプションの既定値は **/verbose-** です。

解説

/verbose オプションを指定すると、コンパイラが生成したエラーの総数についての情報、モジュールが読み込んでいるアセンブリ、および現在コンパイルされているファイルが表示されます。

メモ：

/verbose オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

`In.vb` をコンパイルし、詳細なステータス情報を表示する場合のコード例を次に示します。

```
vbc /verbose in.vb
```

参照

関連項目

コンパイル コマンド ラインのサンプル

その他の技術情報

Visual Basic コンパイラ

/warnaserror (Visual Basic)

コンパイラは、最初に発生した警告をエラーとして扱います。

```
/warnaserror[+ | -][:numberList]
```

引数

+ | -

省略可能です。既定では **/warnaserror-** が有効になっていて、警告が通知されても出力ファイルは生成されます。**/warnaserror** オプションは **/warnaserror+** と同じ機能を持ち、警告をエラーとして扱います。

numberList

省略可能です。**/warnaserror** オプションを適用する警告の ID 番号を、コンマで区切って指定したリストです。警告 ID の指定を省略すると、**/warnaserror** オプションはすべての警告に適用されます。

解説

/warnaserror オプションは、すべての警告をエラーとして扱います。通常は警告として通知されるメッセージが、エラーとして通知されるようになります。同じ警告が 2 回以上通知された場合、2 回目以降は警告として報告されます。

既定では **/warnaserror-** が有効になっており、警告のメッセージが表示されるだけです。**/warnaserror** オプションは **/warnaserror+** と同じ機能を持ち、警告をエラーとして扱います。

エラーと見なす警告の番号をコンマ区切りで指定することにより、一部の特定の警告だけをエラーとして扱うこともできます。

メモ：

/warnaserror オプションを使って、警告の表示方法を制御することはできません。警告を出さないようにするには、/nowarn オプションを使用します。

/warnaserror を設定して、Visual Studio IDE のすべての警告をエラーとして扱うには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [コンパイル] タブをクリックします。
3. [すべての警告を表示しない] チェック ボックスがオフになっていることを確認します。
4. [すべての警告をエラーとして扱う] チェック ボックスをオンにします。

/warnaserror を設定して、Visual Studio IDE の特定の警告をエラーとして扱うには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。
2. [コンパイル] タブをクリックします。
3. [すべての警告を表示しない] チェック ボックスがオフになっていることを確認します。
4. [すべての警告をエラーとして扱う] チェック ボックスがオフになっていることを確認します。
5. [通知] 列の、エラーとして扱う警告に隣接した [エラー] を選択します。

使用例

`In.vb` をコンパイルし、すべての警告の初回発生分をエラーとして表示するコード例は次のようになります。

```
vbc /warnaserror in.vb
```

`T2.vb` をコンパイルし、未使用のローカル変数に対する警告 (42024) だけをエラーとして扱うコードは次のようになります。

```
vbc /warnaserror:42024 t2.vb
```

参照

関連項目

コンパイル コマンド ラインのサンプル

概念

Visual Basic での警告の構成

その他の技術情報

Visual Basic コンパイラ

/win32icon

.ico ファイルを出力ファイルに挿入します。Windows エクスプローラでは、この .ico ファイルが出力ファイルを表します。

```
/win32icon:filename
```

引数
filename

出力ファイルに加える .ico ファイル。ファイル名に空白が含まれている場合は、二重引用符 (" ") で囲みます。

解説

.ico ファイルは、Microsoft Windows リソース コンパイラ (RC) を使って作成できます。リソース コンパイラは Visual C++ プログラムをコンパイルするときに呼び出され、.ico ファイルは .rc ファイルから作成されます。**/win32icon** オプションと **/win32resource** オプションは相互に排他的です。

.NET Framework リソース ファイルを参照するには「/linkresource (Visual Basic)」を、.NET Framework リソース ファイルを結合するには「/resource (Visual Basic)」を参照してください。.res ファイルのインポートについては、「/win32resource」を参照してください。

Visual Studio IDE で /win32icon を設定するには

1. ソリューション エクスプローラでプロジェクトを選択します。[プロジェクト] メニューの [プロパティ] をクリックします。詳細については、「プロジェクト デザイナの概要」を参照してください。
2. [アプリケーション] タブをクリックします。
3. [アイコン] ボックス内の値を変更します。

使用例

`In.vb` をコンパイルし、ico ファイル `Rf.ico` をアタッチする場合のコード例です。

```
vbc /win32icon:rf.ico in.vb
```

参照
関連項目
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

/win32resource

Win32 リソース ファイルを出力ファイルに挿入します。

```
/win32resource:filename
```

引数
filename

出力ファイルに加えるリソース ファイルの名前です。ファイル名に空白が含まれている場合は、二重引用符 (" ") で囲みます。

解説

Win32 リソース ファイルは、Microsoft Windows リソース コンパイラ (RC) を使って作成できます。

Win32 リソースには、Windows エクスプローラでアプリケーションを識別するときに役立つ、バージョンやビットマップ (アイコン) の情報が含まれます。**/win32resource** を指定しない場合、コンパイラはアセンブリのバージョンに基づいてバージョン情報を生成します。**/win32resource** オプションと **/win32icon** オプションは相互に排他的です。

.NET Framework リソース ファイルを参照するには「/linkresource (Visual Basic)」を、.NET Framework リソース ファイルを結合するには「/resource (Visual Basic)」を参照してください。

メモ：

/win32resource オプションは Visual Studio の開発環境内からは利用できません。このオプションを利用できるのは、コマンド ラインからコンパイルするときだけです。

使用例

`In.vb` をコンパイルし、Win32 リソース ファイル `Rf.res` をアタッチする場合のコード例です。

```
vbc /win32resource:rf.res in.vb
```

参照
関連項目
コンパイル コマンド ラインのサンプル
その他の技術情報
Visual Basic コンパイラ

Visual Basic コンパイラ オプション一覧 (カテゴリ別)

Visual Basic のコマンド ライン コンパイラは、Visual Studio 統合開発環境 (IDE) からプログラムをコンパイルする方法の代替として用意されています。次に、Visual Basic コマンド ライン コンパイラ オプションの一覧をその機能ごとに示します。

コンパイラ出力

オプション	目的
/nologo	コンパイラの著作権情報が表示されないようにします。
/utf8output	UTF-8 エンコーディングを使用してコンパイラ出力を表示します。
/verbose	コンパイル時のその他の情報を出力します。

最適化

オプション	目的
/filealign	出力ファイルのセクションを配置する場所を指定します。
/optimize	最適化を有効または無効にします。

出力ファイル

オプション	目的
/doc	ドキュメント コメントを XML ファイルに出力します。
/netcf	.NET Compact Framework を対象とするようにコンパイラを設定します。
/out	出力ファイルを指定します。
/target	出力形式を指定します。

.NET アセンブリ

オプション	目的
/addmodule	指定ファイルからのすべての型情報をコンパイル中のプロジェクトで使用できるようにします。
/delaysign	アセンブリに完全に署名するか、部分的に署名するかを指定します。
/imports	指定のアセンブリから名前空間をインポートします。
/keycontainer	アセンブリに厳密な名前を付けるために、キー ペアのキー コンテナの名前を指定します。
/keyfile	アセンブリに厳密な名前を付けるキーおよびキー ペアを含むファイルを指定します。
/libpath	/reference オプションによって参照されるアセンブリの場所を指定します。
/reference	アセンブリからメタデータをインポートします。

デバッグ/エラーのチェック

オプション	目的
/bugreport	バグを簡単に報告するための情報を含むファイルを作成します。
/debug	デバッグ情報を生成します。
/nowarn	警告を生成するコンパイラの機能が表示されないようにします。

/quiet	構文関連のエラーおよび警告に関するコードが表示されないようにします。
/removeintchecks	整数のオーバーフロー チェックを無効にします。
/warnaserror	警告をエラーとして扱います。

ヘルプ

オプション	目的
/?	コンパイラ オプションを表示します。このコマンドは、**/help** オプションを指定するのと同じです。コンパイルは行われません。
/help	コンパイラ オプションを表示します。このコマンドは、**/?** オプションを指定するのと同じです。コンパイルは行われません。

言語

オプション	目的
/optionexplicit	変数の明示的宣言を必須にします。
/optionstrict	厳密な型の意味を適用します。
/optioncompare	文字列比較をバイナリで行うか、ロケール固有のテキストの意味を使用して行うかを指定します。

プリプロセッサ

オプション	目的
/define	条件付きコンパイルの記号を定義します。

リソース

オプション	目的
/linkresource	マネージ リソースへのリンクを作成します。
/resource	マネージ リソースをアセンブリに埋め込みます。
/win32icon	.ico ファイルを出力ファイルに挿入します。
/win32resource	Win32 リソースを出力ファイルに挿入します。

その他

オプション	目的
@ (応答ファイルの指定)	応答ファイルを指定します。
/baseaddress	ダイナミック リンク ライブラリ (DLL: Dynamic Link Library) のベース アドレスを指定します。
/codepage	コンパイルですべてのソース コード ファイルに使用するコード ページを指定します。
/errorreport	Visual Basic コンパイラが内部コンパイラ エラーを出力する方法を指定します。
/main	起動時に使用する **Sub Main** プロシージャを含むクラスを指定します。
/noconfig	コンパイルに Vbc.rsp を使用しません。
/nostdlib	コンパイラが標準のライブラリを参照しないよう指定します。
/platform	コンパイラが出力ファイルの対象とするプロセッサのプラットフォームを指定します。
/recurse	コンパイルするソース ファイルをサブディレクトリで検索します。

/rootnamespace	すべての型宣言の名前空間を指定します。
/sdkpath	Mscorlib.dll と Microsoft.VisualBasic.dll の場所を指定します。

参照

関連項目

Visual Basic コンパイラ オプション一覧 (アルファベット順)

概念

プロジェクト デザイナの概要

Visual Basic キーワード アップグレード ツール

キーワード アップグレード コマンド ライン ツールにより、Visual Basic .NET ソース ファイルは現在のバージョンの Visual Basic との互換性を持ちます。

```
vb7to8[.exe] [arguments]
```

引数

引数を指定する順序は決まっていません。

引数	説明
filename	*filename* ファイルを処理するようにツールに指示します。ファイル名に空白が含まれている場合は、二重引用符 (" ") で囲みます。 ツールは、複数のファイルを一度に処理できます。
/check	アップグレードの必要なファイルを報告するだけで、変更は実行しません。 アップグレードの必要がない場合は、"アップグレードが必要なファイルはありません" というメッセージが返されます。
/codepage: *number*	コンパイルですべてのソース コード ファイルに使用するコード ページを指定します。ツールは、*number* で指定されたコード ページを使用して、ソース ファイルのエンコーディングを解釈します。 **/codepage** オプションは、ソース コード ファイルを保存するときに現在の ANSI コード ページ、Unicode、または UTF-8 がシグネチャ付きで使用された場合は不要です。
/help または **/?**	ツール オプションを表示します。
/nobackup	ツールは、バックアップ ファイルを作成しません。既存のソース ファイルが上書きされます。 既定では、ファイルが変更されると元のファイルの内容が新しいファイルにコピーされます。新しいファイルの名前は、元のファイル名に ".old" を付けた形式になります。
/nologo	著作権画面を表示しません。
/quiet	構文関連のエラーおよび警告に関するコードを表示しません。
/recurse: *wildcard*	現在のディレクトリおよびサブディレクトリ内で *wildcard* の指定に一致するすべてのファイルを処理します。 *wildcard* の指定には、再帰の開始位置を示すパスを含めることができます。
/utf8output[+|-]	UTF-8 エンコーディングを使用してツール出力を表示します。 このオプションの既定値は **/utf8output-** で、コンパイラの出力では UTF-8 エンコーディングは使用されません。**/utf8output** の指定は、**/utf8output+** の指定と同じです。
/verbose	詳細なエラー メッセージと状態を表示します。

解説

Visual Basic 2005 には、Visual Basic .NET では予約されない新しい予約キーワードがあります。新しいキーワード名を識別子として使うソース ファイルは、Visual Basic 2005 では有効ではありませんが、以前のバージョンの Visual Basic 言語では有効です。

キーワード アップグレード ツールは、現在のバージョンの Visual Basic で新しく予約キーワードになった識別子が使用されているすべての場所を特定し、それらを角かっこ ([ ]) で囲みます。ツールを実行しても書式は維持され、ソースに他の変更は加えられません。

注意

このツールは、Visual Basic 7.0 または 7.1 のソース ファイルにだけ使用してください。他の種類のファイルを処理すると、Visual Basic キーワードの周囲に角かっこが追加され、ファイルの内容が壊れる可能性があります。

ソース ファイルをアップグレードして以前のバージョンとの互換性を保つには、識別子名を角かっこで囲みます。キーワードを識別子名として使う方法の詳細については、「コード内の要素名としてのキーワード」を参照してください。

参照

関連項目

Vbupgrade のコマンド ライン構文

概念

コード内の要素名としてのキーワード

その他の技術情報

Visual Basic のコマンド ライン ツール

Visual Basic のアップグレード リファレンス

このトピックでは、アプリケーションで Visual Basic を以前のバージョンから現在のバージョンへアップグレードするための情報へのリンクを示します。

このセクションの内容

Visual Basic 6.0 のアップグレード リファレンス

Visual Basic 6.0 プロジェクトを新しいバージョンの Visual Basic にアップグレードする際に起こる問題についての情報を掲載します。

関連するセクション

リファレンス (Visual Basic)

Visual Basic プログラミングに関するさまざまなリファレンスへのリンクを提供します。

Visual Basic 6.0 ユーザー向けのヘルプ

Visual Basic 6.0 の知識を活用して Visual Basic 2005 で生産性を高めるための方法を説明します。

言語の変更点 (Visual Basic 6.0 ユーザー向け)

Visual Basic の以前のバージョンからの言語の変更点について、包括的に説明します。

.NET Framework の参照情報

このトピックには、.NET Framework クラス ライブラリの使用方法に関する情報へのリンクがあります。

このセクションの内容

Visual Studio の .NET Framework クラス ライブラリ

.NET Framework のクラスをベースとする機能の開発をサポートする Visual Studio のトピックへのリンクを提供します。

関連するセクション

.NET Framework 全般リファレンス

クラス ライブラリ、ASP.NET の構文を始めとする、.NET Framework の完全な参照トピックが記載されています。

.NET Framework クラス ライブラリ リファレンス

.NET Framework クラス ライブラリのメンバの完全な参照トピックが記載されています。

.NET Framework クラス ライブラリの概要

開発プロセスの迅速化と最適化に役立ち、さらにシステム機能の使用手段を用意するのに役立つ、クラス、インターフェイス、および値型の概要を示します。

.NET Framework の概要

共通言語ランタイムなどの .NET Framework の主要な機能を説明します。

.NET Framework のサンプル

.NET Framework テクノロジを使用するアプリケーションやコンポーネントの作成方法を示す、サンプル プログラムを紹介します。

Visual Basic 言語仕様

Visual Basic 8.0 言語仕様は、Visual Basic の文法と構文に関するあらゆる疑問に答える信頼性のある情報源です。これには、Visual Basic リファレンス ドキュメントで取り上げられていない、言語に関する詳細な情報が全面的に網羅されています。

Visual Basic の仕様は、Microsoft ダウンロード センターから Microsoft Word ファイル形式で入手できます。Microsoft Visual Basic ディベロッパー センターから入手することもできます。

参照

その他の技術情報

Visual Basic リファレンス

Visual Basic のサンプル アプリケーション

Visual Basic ドキュメントのサンプルは、Visual Studio に読み込んで実行できる Visual Basic プロジェクトです。サンプルは、小さなブロックのコードであり、単一のプログラミング タスクに限定された例を示しています。スニペットは、IntelliSense コード スニペット機能を使用してコード エディタに挿入する、コードのブロックです。

以下に示すサンプルに加えて、「Visual Basic Developer Center」(http://www.microsoft.com/japan/msdn/vbasic) のサンプルも参照してください。

このセクションの内容

Visual Basic アプリケーションのサンプル

これらのサンプルでは、プロジェクト、ユーザー、およびアセンブリの各タスクの例を示します。

データのサンプル

これらのサンプルでは、データ アクセスの例を示します。

Visual Basic 言語のサンプル

これらのサンプルでは、Visual Basic 言語の概念を示します。

Visual Basic のセキュリティのサンプル

これらのサンプルでは、セキュリティ タスクの例を示します。

Visual Basic のサーバー コンポーネントのサンプル

これらのサンプルでは、Windows オペレーティング システムのコンポーネントとやり取りするアプリケーションの例を示します。

Visual Basic Windows フォームのサンプル

これらのサンプルでは、Windows フォーム アプリケーションの例を示します。

方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング

サンプルで使用するサンプル データベースのインストール方法を示します。

Visual Basic のコーディング規則

サンプルの作成で使用したコーディング ガイドラインを説明します。

関連するセクション

Visual Basic の IntelliSense コード スニペット

コード エディタに直接挿入できるコードのブロックです。

アプリケーション サンプル

.NET Framework を使用して記述されたアプリケーションです。

技術サンプル

これらのサンプルは、.NET Framework の個別の機能の例を示します。

参照

その他の技術情報

Visual Basic のプログラミング ガイド

方法：サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング

Visual Basic 用のサンプルの中には、Microsoft SQL Server Express で実行中の Northwind データベースを使用するものが多くあります。このページでは、そのインストール手順を説明します。

SQL Server Express をダウンロードしてインストールするには

1. Windows で Internet Explorer を開きます。
2. 次の URL: http://www.microsoft.com/japan/downloads (Microsoft ダウンロード センター) をクリックします。
3. [キーワード] ボックスに「**SQL Server Express**」と入力します。
4. [検索] をクリックします。
5. 結果のページで、SQL Server Express のダウンロード ページまでリンクをたどり、手順に従います。

SQL Server 用の Northwind サンプル データベースと Pubs サンプル データベースをインストールするには

1. Windows で Internet Explorer を開きます。
2. 次の URL: http://www.microsoft.com/downloads/details.aspx?familyid=06616212-0356-46a0-8da2-eebc53a68034&displaylang=en (「Northwind and Pubs Sample Databases」) をクリックします。
3. [ダウンロード] をクリックします。
4. [ファイルのダウンロード] ボックスの [今すぐこのファイルを保存する] を選択して、[OK] をクリックします。
5. ファイルのダウンロード後に、Nwind.exe ファイルをダブルクリックし、データベースをインストールします。

Microsoft Access 用の Northwind サンプル データベースをインストールするには

1. Windows で Internet Explorer を開きます。
2. 次の URL: http://www.microsoft.com/downloads/details.aspx?familyid=c6661372-8dbe-422b-8676-c632d66c529c&displaylang=en (「Access 2000 Tutorial: Northwind Traders Sample Database」) をクリックします。
3. [ダウンロード] をクリックします。
4. [ファイルのダウンロード] ボックスの [今すぐこのファイルを保存する] を選択して、[OK] をクリックします。
5. ファイルのダウンロード後に、Nwind.exe ファイルをダブルクリックし、データベースをインストールします。

接続文字列を確認するには

1. サーバー エクスプローラまたはデータベース エクスプローラを開きます。
2. [データ接続] ノードを展開し、目的の接続を選択します。
3. サーバーのデータベースに接続します。
4. 新しい接続を右クリックし、[プロパティ] をクリックします。
5. [プロパティ] ウィンドウで、接続文字列プロパティを調べます。サンプルではこの接続文字列を使用します。接続文字列は、サンプルの種類に応じて、以下の場所のいずれかで見つかります。
   - フォームのコード。
   - app.config ファイル。
   - プロジェクト設定。ソリューション エクスプローラで、[My Project] をダブルクリックし、[設定] タブで接続文字列の設定を見つけます。
   - データセット ファイル。

トラブルシューティング

- サンプルでは、ローカル コンピュータの VSDOTNET インスタンスにサンプル データベースがインストールされているものと想定しています。データベースが別の場所にインストールされている場合は、サンプル コード内の接続文字列を更新する必要があります。

セキュリティ

追加的なソフトウェアをインストールする場合は、製品の更新がないかどうか、チェックを欠かさないようにします。

参照

処理手順

方法 : サンプル データベースをインストールする

その他の技術情報

サーバー エクスプローラまたはデータベース エクスプローラによるデータへの接続

Visual Basic アプリケーションのサンプル

以下のサンプルは、アーキテクチャ、プロジェクト、ユーザー、およびアセンブリの各タスクの例を示します。

このセクションの内容

アプリケーション イベントのサンプル

プロジェクト デザイナを通じてアクセスするアプリケーション イベントの例を示します。

ClickOnce のサンプル

ClickOnce を使用して、アプリケーションを配置および更新します。

コンソール アプリケーションのサンプル

コンソール ウィンドウを使用した簡単な入出力の例を示します。

Web サービス利用のサンプル

現在の天気、今日の漫画、通貨換算などを提供する、既存のさまざまな .NET Framework Web サービスを利用する方法の例を示します。

ゲームのサンプル

My 機能、オブジェクト指向プログラミング、および描画の例を示します。

ガベージ コレクションのサンプル

共通言語ランタイム (CLR) のガベージ コレクタを制御する方法を示します。

ログ記録のサンプル

My.Application.Log オブジェクトを使用して、ファイルおよびイベント ログに対し情報のログを記録します。

マルチスレッドのサンプル

マルチスレッド タスクの 2 種類の実装方法を示します。

Office オートメーションのサンプル

Microsoft Agent、Microsoft Word、および Microsoft Excel のオートメーションの例を示します。

リフレクションのサンプル

リフレクションを使用して、アセンブリの型情報を調べます。

リソースと設定のサンプル

My.Settings オブジェクトと **My.Resources** オブジェクトの例を示します。

メール送信のサンプル

System.Web.Mail 名前空間のクラスを使用して SMTP で電子メールを送信する方法を示します。

スタック フレームの検査のサンプル

任意のポイントでの呼び出し履歴の情報を、実行中のプログラムのコード内から取得する方法を説明します。

システムおよび環境情報のサンプル

環境情報を提供するいくつかのクラスの使用例を示します。

TCP リモート処理のサンプル

.NET Framework リモート処理アーキテクチャを使用する方法の例を示します。

ユーザー情報のサンプル

My.User オブジェクトとログイン フォーム テンプレートの概要を示します。ここでは、簡単なカスタム認証を実装しています。

Visual Basic 6.0 のアップグレードのサンプル

Visual Basic 6.0 VCR の例を、Visual Basic 6.0 と Visual Basic 2005 の両方のコードで示します。

参照

My の参照

My 機能を使用すると、アプリケーションおよびユーザーのプロパティにアクセスできます。

My.Application オブジェクト

My.Application オブジェクトを使用すると、アプリケーションのプロパティにアクセスできます。

関連するセクション

データのサンプル

これらのサンプルでは、データ操作タスクの例を示します。

Visual Basic Windows フォームのサンプル

これらのサンプルでは、Windows フォーム アプリケーションの例を示します。

Visual Basic 言語のサンプル

これらのサンプルでは、Visual Basic 言語の概念を示します。

Visual Basic のサーバー コンポーネントのサンプル

これらのサンプルでは、Windows オペレーティング システムのコンポーネントとやり取りするアプリケーションの例を示します。

Visual Basic のセキュリティのサンプル

これらのサンプルでは、セキュリティ タスクの例を示します。

Visual Basic のサンプル アプリケーション

このページは、このドキュメントに含まれているすべての Visual Basic サンプルへの出発点です。

アプリケーション イベントのサンプル

Download sample

このサンプルでは、**My.Computer.Network** イベントを使用してネットワーク対応のアプリケーションを作成する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

`MainForm` には StatusStrip コントロールが含まれており、ネットワーク接続が現在接続されているかどうかを示します。ネットワーク接続のステータスが変化するたびに、My.Computer.Network.NetworkAvailabilityChanged イベントが発生します。このイベントのハンドラは My Project\MyEvents.vb ファイルにあります。このファイルにアクセスするには、ソリューション エクスプローラの [My Project] をダブルクリックします。プロジェクト デザイナの [アプリケーション] タブで、[コードの表示] ボタンをクリックします。[Status Panel] を更新するロジックは、`MainForm` の `SetConnectionStatus` メソッドにあります。Visual Basic のネットワーク イベントはメイン アプリケーション スレッドにマーシャリングされます。したがって、これらのイベントのハンドラでは、ユーザー インターフェイスを直接操作できます。

参照

関連項目

My.Computer.Network オブジェクト

ClickOnce のサンプル

Download sample

このサンプルでは、ClickOnce 機能を使用するいくつかのタスクの例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出 ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このソリューションをビルドおよび発行するには

1. ソリューションをビルドします。ソリューション エクスプローラでプロジェクト名を右クリックし、[ビルド] をクリックします。
2. ソリューションを発行します。ソリューション エクスプローラでプロジェクト名を右クリックし、[発行] をクリックします。発行ウィザードの詳細については、「方法 : ClickOnce アプリケーションを発行する」を参照してください。

使用例

ClickOnce 機能を使用すると、インストールおよび保守の次のようなタスクをプログラムで処理できます。

- **更新のチェック**　CheckForUpdate メソッドを使用して、新しい更新をチェックします。
- **アプリケーションの更新**　UpdateAsync メソッドを使用して、更新されたアプリケーションをダウンロードします。
- **オプション ファイルのダウンロード**　DownloadFileGroupAsync メソッドを使用して、ファイルをダウンロードします。
- **配置情報**　**My.Application.Deployment** プロパティを使用して、名前、バージョン、位置、および最後の更新チェックの情報を取得および報告します。

参照

関連項目

My.Application.Deployment プロパティ
My.Application.IsNetworkDeployed プロパティ
CheckForUpdateCompleted
DownloadFileGroupCompleted
UpdateProgressChanged
UpdateCompleted

その他の技術情報

ClickOnce の配置

コンソール アプリケーションのサンプル

Download sample

このサンプルでは、コンソール アプリケーションを作成する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このプロジェクトには、XML ファイルが埋め込みリソースとして含まれています。XML ファイルの中身は Northwind データベースの `Products` テーブルです。アプリケーションは、型指定されていないデータセットを XML ファイルから読み込みます。コンソール アプリケーションの実行時に、ユーザーからの入力として受け入れられるのは、製品 ID または "quit" のいずれかです。"quit" を入力した場合、コンソール アプリケーションは終了します。有効な製品 ID を入力した場合、対応する製品情報が表示されます。無効な製品 ID を入力した場合、例外メッセージが表示されます。

参照

関連項目

Console Class
Visual Basic プログラムの構造
DataSet.ReadXml Method

概念

コンソール アプリケーションの構築

Web サービス利用のサンプル

Download sample

このアプリケーションでは、現在の天気、今日の漫画、通貨換算などを提供する、既存のさまざまな .NET Framework Web サービスを利用する方法の例を示します。

TabControl を使用して、5 種類の Web サービスの利用例を示します。

- 米国の郵便番号に対応する現地の日時を取得します (http://www.ripedev.com/webservices/LocalTime.asmx)。
- ユーロ ドルを、ECC 加盟 12 か国のいずれかの通貨に変換します (http://www.xml-webservices.net/services/conversions/euro_convert/euro_conver.asmx)。
- ISBN 番号を入力して、本の売上ランキングと価格情報を Amazon および Barnes & Noble から検索します (http://www.perfectxml.net/WebServices/SalesRankNPrice/BookService.asmx)。
- 米国の郵便番号に対応する現在の天気の状況を取得します (http://www.learnxmlws.com/services/weatherRetriever.asmx)。
- Daily Dilbert Web サービスから、Dilbert の今日の漫画を表示します。この例では、Web サービスを非同期で呼び出します (http://www.esynaps.com/WebServices/DailyDilbert.asmx)。
- UDDI を使用して Web サービスの URL エラーを処理する方法を示します。これにより、"フォールバック プラン" を実装できます (http://www.vbws.com/services/ServerTime.asmx)。ステータス メッセージと、構造化エラー処理内のエラー メッセージを使用して、データ取得プロセスの状況をユーザーに通知します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

必要条件

このアプリケーションを実行するには、Microsoft UDDI (Universal Description Discovery and Integration) SDK 1.76 が必要です。これは、http://msdn.microsoft.com/library/en-us/dnanchor/html/anch_uddi.asp?frame=true からダウンロードできます。

使用例

これらは、http://www.xmethods.com/ に掲載されていた、実在する Web サービスです。

参照

処理手順

方法 : マネージ コードを使用して XML Web サービスにアクセスする

ゲームのサンプル

Download sample

このサンプル アプリケーションは、GDI+ グラフィックス、タイマ機能、ユーザー構成、およびハイ スコア保存の機能を備えた、Windows フォームの簡単なゲームを実装します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

サンプル ドキュメントを表示するには

1. ソリューション エクスプローラで、[ドキュメント] フォルダをダブルクリックします。
2. Visual Basic Express Edition を使用している場合は、[ドキュメント] フォルダ内の ReadMe.htm を右クリックします。[ブラウザで表示] を選択します。
3. Visual Basic の別のバージョンを使用している場合は、[Documentation] フォルダ内の ReadMe.htm をダブルクリックします。

使用例

このサンプルは、大半のアプリケーションに実装されている次のような機能のデモンストレーションです。

- **ユーザー オプション**　`Options` フォームで、ユーザーがハイ スコアをリセットしたり、サウンドのオンとオフを切り替えたりできます。
- **ヘルプ**　このアプリケーションには、コンパイルされたヘルプ プロジェクトが含まれています。コンパイルされたヘルプには、[ヘルプ] メニュー項目からアクセスします。
- **マウス**　マウスは、ゲームのプレイに使用するのに加え、タイトル バーが非表示になっているときにフォームをドラッグして移動するのにも使用します。
- **キーボード**　M キーを押すと、メニューおよびタイトル バーのオンとオフが切り替わります。"P" キーを押すと、ゲームのアクションが一時中断および再開されます。

Visual Basic および .NET Framework の次のような機能が中核として使用されています。

- **GDI+ グラフィックス**　ゲームのアクションの実装には、System.Drawing 名前空間の描画関数が全面的に使用されています。
- **レジストリ**　ハイ スコアはレジストリに格納されます。

- **オブジェクト指向プログラミング**　ゲームのアクションと描画は、複数のクラスの連係により制御されます。ゲームのメインのクラスは `Grid` クラスと `Block` クラスです。`PointTranslator` クラスは、ブロック描画のためのユーティリティ関数を追加します。`HighScores` クラスは、スコアの取得と設定を処理します。

参照

関連項目

My.Computer.Audio オブジェクト
My.Computer.Registry オブジェクト
System.Drawing
HelpProvider
MouseDown
KeyDown

ガベージ コレクションのサンプル

Download sample

このサンプルでは、共通言語ランタイム (CLR) のガベージ コレクタを制御する方法を示します。作成されたオブジェクトの `Finalize` メソッドをガベージ コレクタが呼び出すタイミングや、終了処理を抑止する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. F5 キーを押します。

2. アプリケーションを閉じます。フォームが閉じられるときに、未終了のオブジェクトがどのように終了するかを確認します。

使用例

フォームのボタンでは、割り当てられたオブジェクト、空きオブジェクト、およびガベージ コレクションされたオブジェクトの数を操作するためのコードを実行できます。

- **オブジェクトの作成**　このボタンでは、`ListNode` 型 (このプロジェクトで定義した型) のオブジェクトの階層構造を作成します。
- **オブジェクトを Nothing に設定**　このボタンでは、`lstCreatedObjects` リスト ボックスで選択されたオブジェクトを **Nothing** に設定することによって解放し、ガベージ コレクションの対象とします。
- **Dispose の呼び出し**　このボタンでは、`lstCreatedObjects` ボックスで選択されたオブジェクトを破棄します。`ListNode` クラスには IDisposable インターフェイスが実装されています。
- **ガベージ コレクタの実行**　このサブルーチンでは、ガベージ コレクタを強制的に実行させます。Collect メソッドは確定的ではありません。つまり、オペレーティング システムは、現在実行中のタスクが完了するまで、ガベージ コレクタの実行を先延ばしする場合があります。実際上は、`Collect` メソッドの呼び出しから 1 秒程度のうちにガベージ コレクタが実行されるものと考えて問題ありません。

サンプルは、SuppressFinalize メソッドを使用すると、特定のオブジェクトにおけるガベージ コレクタのパフォーマンスが向上することも示しています。

参照

関連項目

GC
Collect
SuppressFinalize

その他の技術情報

ガベージ コレクション

ログ記録のサンプル

Download sample

このサンプルでは、**My.Application.Log** オブジェクトを使用および構成する方法の例を示します。

Visual Basic には、ログ記録のしくみが 2 種類用意されています。**My.Application.Log** では、オペレーティング システムのイベント ログにアクセスでき、またテキスト ファイルに対してメッセージを書き込むこともできます。EventLog コンポーネントでは、オペレーティング システムのイベント ログにアクセスできます。**EventLog** コンポーネントでは、ログの追加と削除、イベント ログ ソースの追加と削除、メッセージの書き込み、およびメッセージの削除を行うことができます。サンプル コードについては、「イベント ログのサンプル」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. F5 キーを押します。このサンプルに対して構成されたリスナがメイン フォームに一覧表示されます。
2. フォームの TextBox コントロールにメッセージを書き込みます。
3. [Write Log Message] ボタンをクリックします。

   各リスナにメッセージが書き込まれます。

   a. XmlWriterTraceListener が書き込む XML ファイルは、c:\logsamples\SampleLog.xml にあります。

   b. FileLogTraceListener が書き込むプレーンテキスト ファイルは、"Application Data" ディレクトリにあります。プログラムを Visual Studio で実行している場合は、"c:\Documents And Settings\[user]\Application Data\Microsoft Corporation\Microsoft Visual Studio\[8.0*]\LogSample.log" のようなパスになります。

使用例

このサンプルには以下が含まれています。

- `Form1` **My.Application.Log** に対して現在有効になっているすべてのリスナと、**My.Application.Log** にテキストを書き込むためのコントロールを表示する、単純なフォームです。
- `app.config` **My.Application.Log** の構成を定義します。メッセージを書き込む場所や、フィルタの対象とするメッセージなどです。

このサンプルでは、**My.Application.Log** を使用して、プレーンテキスト ファイル、イベント ログ、および XML ファイルに書き込む方法の例を示します。他の場所 (たとえばデータベースや電子メール メッセージ) に対しても同様に書き込むことができるように、カスタムの TraceListener を実装することも簡単です。

各メッセージには、フォームで選択する "重大度レベル" が割り当てられます。メッセージを受け取ったリスナは、フィルタ レベルに基づいてそのメッ

セージを破棄する場合があります。たとえば、EventLogTraceListener コンポーネントはエラー メッセージのみを書き込むように構成されているため、重大度レベルが "情報" または "警告" として書き込まれたメッセージは、XML およびプレーンテキスト ファイルには書き込まれますが、**EventLog** には書き込まれません。

サンプルの動作を変更するには、プロジェクトの app.config ファイルを編集します。変更の際は、ファイル内のコメントを参考にしてください。

My.Application.Log およびアプリケーション モデルを使用して、起動、シャットダウン、および未処理例外のログを記録できます。このサンプルのコードを参照するには、ソリューション エクスプローラの [My Project] をダブルクリックします。アプリケーションペインの [コードの表示] ボタンをクリックします。

参照

概念

Visual Basic でのアプリケーション ログの使用

その他の技術情報

イベント ログのサンプル

マルチスレッドのサンプル

Download sample

このソリューションでは、複数のスレッドを使用してタスクを同時実行する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、時間がかかるタスクを別個のスレッドで実行します。ボタンをクリックすると処理が起動されます。

- **同じスレッドで実行** メイン フォームと同じスレッドでタスクを実行します。この場合、タスクが完了するまでは、ユーザーはメイン フォームとのやり取りをブロックされます。このタスクでは、マルチスレッド処理のコードは不要です。
- **ワーカー プール スレッドで実行** このタスクでは、コンパイラがすべてのデリゲート用に作成した `BeginInvoke` メソッドを使用してタスクを実行します。
- **バックグラウンド ワーカー** このタスクでは、BackgroundWorker コンポーネントを使用して新しいスレッドを作成します。

このサンプルでは、各スレッドはデータにアクセスしていないので、同期は含まれていません。

参照

関連項目

Delegate ステートメント
DebuggerStepThroughAttribute Class
AppDomain.GetCurrentThreadId Method
Thread Class

概念

非同期プログラミングの概要

その他の技術情報

デリゲートを使用した非同期プログラミング

Office オートメーションのサンプル

Download sample

このサンプルでは、Microsoft Agent、Microsoft Word、および Microsoft Excel のオートメーションの例を示します。さまざまな COM Office アセンブリを参照するときに、Visual Studio は COM ランタイム呼び出し可能ラッパー (RCW) を自動的に作成します。このため、.NET Framework からそれらを使用するプログラムを作成できます。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

6. Office のインストール構成によっては、プロジェクト内の Office 参照が有効でない場合があります。参照を更新するには、次の手順に従います。

   a. ソリューション エクスプローラで、[My Project] をダブルクリックします。

   b. プロジェクト デザイナの [参照設定] タブをクリックします。

   c. [追加] をクリックします。

   d. [参照の追加] ダイアログ ボックスの [COM] タブをクリックします。

   e. [Microsoft Agent Control] を見つけ、それをダブルクリックして、プロジェクトに追加します。

   f. 同じ手順で、Microsoft Word および Microsoft Excel の参照を追加します。

このサンプルを実行するには

- F5 キーを押します。

必要条件

Merlin に話をさせたい場合は、音声認識パックのインストールが必要なことがあります。これは、Microsoft Office XP に含まれていますが、既定ではインストールされません。その方法については、「Install speech recognition」(http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/input_speech_install.mspx) を参照してください。

使用例

3 つの TabPage コントロールが、TabControl に追加されており、以下の 3 つのプログラミング タスクの例を示します。

- **Microsoft Agent**　Office Agent ライブラリを使用すると、Merlin というキャラクタを登場させて動作させることができます。Merlin を登場させる前に、`AgentObjects.Controller` オブジェクトのインスタンスを作成する必要があります。このオブジェクトから、Merlin を表す `IAgentCtlCharacter` オブジェクトにアクセスします。`IAgentCtlCharacter` は `Play` メソッドを実装しています。このメソッドを使用して、さまざまなアニメーションを制御します。Microsoft Agent の詳細については、http://www.microsoft.com/japan/msdn の文書お

よび SDK ドキュメントを参照してください。

- **Microsoft Word**　RichTextBox コントロールに簡単なテキスト ファイルを表示します。Word のインスタンスを開始し、Word アプリケーションを使用して、**RichTextBox** コントロールのテキストのスペルをチェックします。
- **Microsoft Excel**　DataSet を作成し、XML ドキュメントからデータを読み込みます。次に、DataBind メソッドを使用して、これを DataGrid にバインドします。次に、**DataSet** の内容を Excel のスプレッドシートにエクスポートし、列の値の平均を計算する Excel 関数を実行します。

参照

処理手順

方法 : Office のプライマリ相互運用機能アセンブリをインストールする
方法 : プライマリ相互運用機能アセンブリを利用して Office アプリケーションを使用する

概念

Office プロジェクトのアセンブリの概要

リフレクションのサンプル

Download sample

このサンプルでは、リフレクションを使用して、アセンブリの型情報を調べます。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このフォームでは、次の 3 つのレベルでアセンブリを調べることができます。

- **アセンブリ**　GetAssemblies メソッドを使用して、読み込まれているアセンブリの一覧を取得します。一覧は ListBox コントロールに表示します。
- **型**　アセンブリ名が選択されると、GetTypes メソッドを使用して、そのアセンブリのすべての型を別の **ListBox** コントロールに表示します。
- **メンバ**　[Types] ボックスで型が選択されると、GetMembers メソッドを使用して、その型のすべてのメンバを一覧表示します。

参照

関連項目

Assembly
AppDomain
GetAssemblies
GetTypes
GetMembers

概念

リフレクションの概要

リソースと設定のサンプル

Download sample

このサンプルでは、**My.Settings** オブジェクト、**My.Resources** オブジェクト、プロジェクト デザイナ、およびリソース エディタを使用してアプリケーションのリソースと設定を管理する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

  アプリケーションを実行するたびに、異なる背景が表示されます。

使用例

アプリケーションが実行された回数を追跡するために、設定が使用されます。アプリケーションの実行のたびに設定値が調べられ、その値に応じて、利用可能な複数のデスクトップ イメージのうちのいずれか 1 つが選択されます。このため、アプリケーションを実行するたびに新しい背景が表示されます。

コードは、`Form1` の `SetBackground` メソッドと、アプリケーションの起動イベントおよびシャットダウン イベントの中にあり、My Project\MyEvents.vb で処理されています。このファイルを参照するには、ソリューション エクスプローラ の [My Project] をダブルクリックします。プロジェクト デザイナの [アプリケーション] パネルで、[アプリケーション イベントの表示] ボタンをクリックします。

設定の作成および変更は、プロジェクト デザイナ の [設定] タブで行います。設定にはアプリケーション設定とユーザー設定の 2 種類があります。アプリケーション設定とは、アプリケーションのすべてのインスタンスが共通の値を持つ設定のことです。ユーザー設定とは、ユーザーごとに作成および変更する設定のことです。このサンプルでは、アプリケーションが実行された回数を保持する設定はユーザー設定です。

参照

概念

アプリケーション リソースへのアクセス
アプリケーション設定へのアクセス

メール送信のサンプル

Download sample

このサンプルでは、System.Web.Mail 名前空間のクラスを使用して SMTP で電子メールを送信する方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. SmtpServer プロパティの設定をチェックします。詳細については、「必要条件」を参照してください。
2. F5 キーを押します。

必要条件

1. インターネット インフォメーション サービス (IIS: Internet Information Services) が必要です。
2. SMTP サービスをインストールして実行する必要があります。SMTP サービスは IIS と共にインストールされるオプションです。
3. **SmtpServer** プロパティを、SMTP サーバーを実行中の IP アドレスまたはサーバー名に設定する必要があります。多くの場合、これは "localhost" コンピュータです。その場合、**SmtpServer** プロパティ値は、表示名である "localhost" か、または "127.0.0.1" のいずれかに設定できます。このアプリケーションの既定値は後者です。SMTP サーバー名は次のようにして見つけます。
   a. IIS スナップインを開きます ([コントロール パネル] で、[管理ツール] を開き、[インターネット インフォメーション サービス] を開きます)。
   b. [既定の SMTP 仮想サーバー] ノードを展開します。
   c. [ドメイン] をクリックします。右パネルに [ドメイン名] の一覧が表示されます。**SmtpServer** プロパティは、これらのいずれかの名前か、または対応する IP アドレスに設定できます。
4. 既定の SMTP 仮想サーバーも実行している必要があります。注意が必要なのは、SMTP サービスは実行中だが IIS の既定の SMTP サーバーは停止している、ということがあり得る点です。サービスそのものと、そのサービスが実行する実際のサーバーは異なります。既定の SMTP 仮想サーバーが実行中でない場合は、IIS スナップインのノードに赤い×印のアイコンが表示されます。サーバーを開始するには、ノードを右クリックし、[開始] をクリックします。
5. 中継の制限を次のように設定します。
   a. IIS で、[既定の SMTP 仮想サーバー] を右クリックし、[プロパティ] をクリックします。
   b. [プロパティ] ダイアログ ボックスの [アクセス] タブをクリックします。

c. [中継] をクリックします。

d. [中継の制限] ダイアログ ボックスで、[以下のリストに含まれるコンピュータのみ] ボックスを選択します。下部のチェック ボックスを、オンまたはオフにします。

使用例

このアプリケーションは、Microsoft Outlook 電子メール クライアントに似たコントロールを持つ単一のフォームを備えています。フォームを読み込むときには、ユーザーが SMTP サービスをインストールして実行していることが確認されます。

参照

処理手順

Mailer サンプル : SMTP メール クラスの例

方法 : IIS 内で SMTP 仮想サーバーをインストールおよび構成する

スタック フレームの検査のサンプル

Download sample

このサンプルでは、任意のポイントでの呼び出し履歴の情報を、実行中のプログラムのコード内から取得する方法を説明します。呼び出し履歴を使用すると、現在のプロシージャに至るまでにアプリケーションが実行したプロシージャの順番を確認できます。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

.NET Framework では、この機能は StackFrame クラスに用意されています。このサンプルでは、別の 2 つのクラスも使用します。StackTrace クラスと MethodInfo クラスです。**StackTrace** のパラメータなしのコンストラクタは、アプリケーションの起動時からの情報を提供する **StackFrame** オブジェクトの完全なセットを返します。また、**StackTrace** のコンストラクタには Exception のインスタンスを引数に受け取るものもあります。このコンストラクタは、例外を発生させたコードまでさかのぼるスタック トレース情報を提供します。

[Test Procedure Stack] ボタンのイベント ハンドラには、いくつかのメソッドを呼び出すコードが記述されており、呼び出しのスタックが数レベルの深さに及びます。[Test Procedure Stack] ボタンがクリックされると、ProcA を呼び出し、そこからさらに ProcB を呼び出します。ProcB は GetFullStackFrameInfo プロシージャを次のように呼び出し、新しい `StackTrace` オブジェクトを渡します。

```
GetFullStackFrameInfo(New StackTrace())
```

`GetFullStackFrameInfo` プロシージャのコードでは、**StackTrace** オブジェクトの FrameCount プロパティを使用して、現在のプロシージャからさかのぼるスタック フレームの総数を取得します。次に、各スタック フレームについての情報をサンプル上のリスト ボックスに追加します。その際、GetFrame メソッドを呼び出して、インデックス番号に対応するスタック フレーム オブジェクトを取得します。

GetStackFrameInfo プロシージャは、コードで渡された **StackFrame** オブジェクトの **GetFrame** を呼び出し、**MethodInfo** オブジェクトのプロパティを取得して、プロシージャの情報をリスト ボックスに表示します。

[Test Exception Handling] ボタンのイベント ハンドラでは、スタックが複数のレイヤの深さまで至ったところでエラーを生成します。コードは例外を発生させ、その例外を **StackFrame** オブジェクトのコンストラクタに渡します。

参照

関連項目

System.Diagnostics
StackFrame
System.Reflection

MethodInfo
FrameCount

システムおよび環境情報のサンプル

Download sample

このサンプルでは、環境情報を提供するいくつかのクラスの使用例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

メイン フォームには、いくつかのタブ ページを持つ TabControl があります。各タブ ページは、Environment または SystemInformation のメンバを呼び出します。呼び出すグループは、それぞれ次のように異なります。

- **プロパティ** このタブは、**Environment** プロパティを使用する例を示します。
- **特別なフォルダ** このコードは、SpecialFolder の GetNames メソッドを呼び出して、すべての特別なフォルダ名のリストを取得します。このリストを、ListBox コントロールのデータ ソースとして使用します。
- **メソッド** このタブは、**Environment** のメソッド (Exit や ExpandEnvironmentVariables など) を呼び出した結果を示します。GetLogicalDrives メソッドおよび GetCommandLineArgs メソッドが返した文字列配列を、**ListBox** コントロールのデータ ソースとして使用します。
- **環境変数** このコードは GetEnvironmentVariables メソッドを呼び出します。返されたコレクションの `Keys` プロパティを、**ListBox** コントロールのデータ ソースとして使用します。このタブ ページは、GetEnvironmentVariable メソッドを呼び出して 1 つの環境変数を取得する例も示します。
- **システム情報** このタブは、**SystemInformation** が提供するすべての情報を取得します。

参照

関連項目

Environment
SystemInformation

TCP リモート処理のサンプル

Download sample

このサンプルでは、.NET Framework リモート処理アーキテクチャを使用する方法の例を示します。また、利用可能なオブジェクトおよびサーバー位置情報の指定では、バイナリ内にハードコーディングしたり、システム レジストリにデータを記録したりするのではなく、XML 構成ファイルを使用しています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. `RemoteCustomer` プロジェクトをビルドします。
2. `RemoteHost` プロジェクトを開き、F5 キーを使用して、`RemoteHost` アプリケーションを起動します。
3. Visual Studio の別のインスタンスを起動し、`Client` プロジェクトを読み込みます。次に、コンパイル済みの `RemoteCustomer` アプリケーションへの参照 (`RemoteCustomer.dll`) を追加します。
4. F5 キーを押して `Client` アプリケーションを起動します。
5. 2 台の異なるコンピュータを使用してテストするには、クライアントの .config ファイルに変更を加える必要があり、`localhost` をサーバー コンピュータの IP アドレスまたは DNS 名に変更します。加えて、ポート番号を変更する場合は、サーバーのファイルにも変更が必要なことがあります。例については、クライアントとホストのそれぞれの .config ファイルのコメントを参照してください。

使用例

このサンプルにはソリューションが 3 つあり、それぞれにプロジェクトが 1 つずつ含まれています。

- `RemoteCustomer`: サーバー
- `RemoteHost`: ホスト
- `Client`: クライアント

RemoteCustomer

`RemoteCustomer` は、作成可能な 3 つの型を公開するクラス ライブラリです。

- `Customer` (アクティブにされたクライアント)
- `SingleCallCustomer` (WellKnown singlecall)

- `SingletonCustomer` (WellKnown singleton)

各クラスは同じ名前空間 `RemotingSample` に属します。この名前空間は、各クラスのソース ファイルで定義されています。

RemoteHost

`RemoteHost` プロジェクトには、Windows フォーム アプリケーションが含まれています。このアプリケーションは、Host.exe.config という XML 構成ファイルを読み取って、`RemoteCustomer` プロジェクトで定義されているクラスを利用できるようにします。実稼動のアプリケーションの場合は、クラス ライブラリ (RemoteCustomer.dll) を Windows サービスでホストする手もあります。

Client

`Client` プロジェクトは Windows フォーム アプリケーションです。Client.exe.config という構成ファイルに基づいて、リモート オブジェクトを検索する場所を判断します。

参照

関連項目

System.Runtime.Remoting

その他の技術情報

.NET Framework リモート処理の概要

ユーザー情報のサンプル

Download sample

このサンプルでは、**My.User** オブジェクトとログイン フォーム テンプレートの概要を示します。ここでは、簡単なカスタム認証を実装しています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、以下のタスクの例を示します。

- **ユーザー名**　Windows の現在のユーザー名をメイン フォームに表示します。この値は My.User オブジェクト から取得します。
- **データベース アクセス**　開発者は、**My.User** により、独自の手法によるカスタムの認証および承認を組み込むことができます。そのとき、IPrincipal インターフェイスと IIdentity インターフェイスを使用します。これらのインターフェイスは、固有の認証および承認アーキテクチャとは無関係な、現在のユーザーの側面を示します。これらのインターフェイスを実装するときに、ユーザーの認証方法や、ユーザーが現在割り当てられているロールを定義します。このサンプルでは、**IPrincipal** インターフェイスと **IIdentity** インターフェイスのきわめて基本的な実装を使用して、開発者が定義した承認および認証のしくみを **My.User** で使用するように構成する方法の例を示しています。

セキュリティに関するメモ：
このサンプルの実装は、安全なコーディング手法に従っているわけではありません。ユーザーは、パスワードに "password" と入力すると認証されます。

- **認証モード**　このプログラムの動作は、認証モードにより制御されます。ソリューション エクスプローラの [My Project] ノードをダブルクリックして、プロジェクト デザイナを開きます。[アプリケーション] タブで、[認証モード] には [アプリケーション定義] が選択されています。アプリケーションを実行すると、[ログイン] フォームが表示されます。

参照

関連項目

My.User オブジェクト
IPrincipal
IIdentity

Visual Basic 6.0 のアップグレードのサンプル

Download sample

このサンプルでは、Visual Basic 6.0 のアプリケーションを示し、これを Visual Basic 2005 にアップグレードする方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

6. Visual Basic 6.0 で Visual Basic 6.0 プロジェクトを開きます。

このサンプルを実行するには

- F5 キーを押します。

使用例

加えた変更の一部を次に示します。

- **オブジェクト指向の手法**　`Tape`、`Recorder`、`Channel`、および `TapeFrame` の各クラスを実装しました。モジュールはクラスに置き換えました。
- **ByRef と ByVal**　**ByRef** は **ByVal** に適宜置き換えました。
- **XML コメント**　整形されたコメント ブロックは XML コメントに置き換えました。
- **名前付け規則**　ハンガリー記法による名前は削除しました。
- **Boolean 型**　**Integer** 値は **Boolean** 変数に置き換えました。

参照

その他の技術情報

Visual Basic アプリケーションのサンプル

データのサンプル

以下のサンプルは、データのさまざまな使用例を示します。

このセクションの内容

カスタム データ コントロールのサンプル

データ ソース ウィンドウで自作のコントロールを使用する方法を説明します。

ComboBox のデータ バインディングのサンプル

ComboBox コントロールにデータをバインドする方法を示します。

データベース作成のサンプル

データベース、テーブル、ストアド プロシージャ、およびビューを作成する SQL ステートメントをコードで実行します。

データ検索のサンプル

データベースのデータのサブセットを表示する方法の例を示します。

データ検証のサンプル

データソース デザイナの使用例を示し、`PropertyChanged` イベントを使用してユーザー入力を検証します。

ローカル データのサンプル

Visual Studio 2005 のクライアント データ機能 (データ ソース ウィンドウ、型指定された TableAdapter および BindingSource オブジェクトなど) の例を示します。

オブジェクトのバインドのサンプル

データベース コンポーネント以外のオブジェクトにコントロールをバインドする方法の例を示します。

シリアル化のサンプル

オブジェクトをシリアル化および逆シリアル化する方法を説明します。

ストアド プロシージャのサンプル

ADO.NET および Microsoft SQL Server を使用して、プログラムでストアド プロシージャを作成および使用する方法の例を示します。

XMLDocument クラスのサンプル

XmlDocument クラスを使用して XML データを操作する方法を示します。

参照

クライアント データ アプリケーションの作成

Visual Studio に用意されている、データベース アプリケーションの作成およびデータベースの設計と保守を行うためのツールについて説明します。

System.Data

データ アプリケーションで一般的に使用する .NET Framework のオブジェクトについて説明します。

XmlDocument

XML データの操作に使用するクラスについて説明します。

関連するセクション

BindingSource コンポーネントのサンプル

Windows フォーム コントロールへのバインディングの例を示します。

クライアント データ アプリケーションの作成

Visual Studio におけるデータ アクセス プログラミングについて説明します。

XML データの使用

XML、XSLT、およびスキーマと連携する Visual Studio のツールと機能について説明します。

Visual Basic アプリケーションのサンプル

プロジェクト、ユーザー、およびアセンブリの各タスクの例を示します。

Visual Basic Windows フォームのサンプル

Windows フォーム アプリケーションの例を示します。

Visual Basic 言語のサンプル

Visual Basic 言語の概念を説明します。

Visual Basic のサーバー コンポーネントのサンプル

Windows オペレーティング システムのコンポーネントとやり取りするアプリケーションの例を示します。

Visual Basic のセキュリティのサンプル

セキュリティ タスクの例を示します。

Visual Basic のサンプル アプリケーション

Visual Basic のすべてのサンプルの一覧を示します。

カスタム データ コントロールのサンプル

Download sample

このサンプルでは、[データ ソース] ウィンドウで自作のコントロールを使用する方法を説明します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

必要条件

このサンプルには、Northwind サンプル データベースが必要です。詳細については、
「方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング」を参照してください。

使用例

サンプル ソリューションには 2 つのプロジェクトが含まれています。1 つは、`MyCompanyControls` という名前の Windows コントロール ライブラリ プロジェクトです。もう 1 つは、`CustomDataControls` という名前の Windows アプリケーション プロジェクトです。

コントロール ライブラリでは `PhoneBox` コントロールを定義します。このコントロールは、MaskedTextBox コントロールを継承し、Mask プロパティを市外局番付きの電話番号に設定するだけのものです。このライブラリ プロジェクトには、埋め込みリソース PhoneBox.bmp も含まれており、[ツールボックス] および [データ ソース] ウィンドウのアイコンを表します。

`CustomDataControls` プロジェクトでは `PhoneBox` コントロールを利用可能です。プロジェクトのショートカット メニューの [リビルド] を選択すると、[ツールボックス] の [MyCompanyControls] という新しいセクションに `PhoneBox` が表示されます。[ツールボックス] のアイテムを表示するためには、デザイナで `Form1` を開いている必要があります。

String 用のコントロールのリストに `PhoneBox` を追加するには、[データ ソース] ウィンドウを使用します。[データ ソース] ウィンドウで、`Employee` テーブルの `HomePhone` 列を選択します。ドロップダウンを選択し、[カスタマイズ] をクリックします。指定された型用のコントロールのリストをカスタマイズするためのダイアログ ボックスが開きます。このリストには、Visual Studio の [ツールボックス] に現在含まれているコントロールで DefaultBindingPropertyAttribute を実装しているものの一覧が設定されています。`PhoneBox` コントロールは、**DefaultBindingPropertyAttribute** をサポートする **MaskedTextBox** を継承しています。[データ型] の [文字列] を選択し、一覧から `PhoneBox` コントロールを見つけます。`PhoneBox` コントロールの横のチェックは、文字列プロパティを持つ任意のオブジェクトに対して `PhoneBox` コントロールが利用可能であることを、Visual Studio に対して示します。

参照

処理手順

方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング
方法: [データ ソース] ウィンドウにカスタム コントロールを追加する

その他の技術情報

ComboBox のデータ バインディングのサンプル

Download sample

このサンプルでは、ComboBox コントロールおよび DataGridView コントロールにデータをバインドする方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

必要条件

このサンプルには、Northwind サンプル データベースが必要です。詳細については、
「方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング」を参照してください。

使用例

このコードでは、6 種類のデータ ソースを **ComboBox** コントロールにバインドする方法の例を示します。バインドの対象データは次のとおりです。

- 配列
- 文字列の ArrayList
- クラス オブジェクトの **ArrayList**
- DataTable
- DataView
- BindingSource オブジェクト

メイン フォームの読み込み時に、簡単な SQL SELECT ステートメントを使用して、Northwind データベースの `Products` テーブルを DataSet に取得します。このとき、`ProductName` 列を並べ替えたビューを提供する **DataView** も併せて作成します。TableAdapter および **BindingSource** を使用して、`Products` テーブルを別の **DataSet** に読み込みます。

次にユーザーは、コンボ ボックス コントロールをバインドすることにより、データを読み込むことができます。バインドの対象は、他の色の配列、図形の配列リスト、構造体で定義された営業部門が格納された高度な配列リスト、いずれかのデータセットに含まれている製品テーブル、または並べ替えられたデータ ビューです。バインド先がデータセット、データ ビュー、または営業部門の高度な配列リストの場合は、コンボ ボックスでエントリを選択したときに、そのエントリに関連付けられた値も表示されます。バインド先がデータ コネクタの場合は、コンボ ボックスとデータグリッド ビューが同期します。コンボ ボックスの値を変えると、データグリッドビューも同じレコードに移動します。グリッド内でデータセットをスクロールするか、またはナビゲーション ツール バーを使用すると、コンボ ボックスが更新されます。

メイン フォームには、コンボ ボックス コントロール、データを読み込むためのボタン コントロール、およびデータ ソースと選択された値を表示するための 2 つのラベル コントロールとグリッドがあります。**ArrayList**、**DataSet**、および **DataView** の使用により、コンボ ボックス コントロールに表示されている各項目に値を関連付けることができます。たとえばユーザーが、コンボ ボックスにバインドされている製品テーブルから製品エントリ `Chai` を選択すると、選択されたエントリとして Chai が表示されますが、それに関連付けられている `ProductId` も SelectedValue プロパティを通じて取得できます。ValueMember プロパティを使用すると、その関連付けられた値を持つ項目を選択できます。DisplayMember プロパティを使用すると、コンボ ボックス コントロールに表示される項目を選択できます。

このサンプルの作成

このフォームの大部分は、コンポーネントをフォームにドラッグし、スマート タグと [プロパティ] ウィンドウの設定を使用することによって作成されました。次に示すのは、このフォームの **DataGridView** 部分を 1 から作成する方法の簡単な概要です。

1. 新しい Windows アプリケーション プロジェクトを作成します。
2. `Form1` を開いた状態で、[データ ソース] ウィンドウを選択します。[データ] メニューを使用してアクティブにすることもできます。
3. [データ ソース] ウィンドウで、[新しいデータ ソースの追加] をクリックします。
4. データ ソース構成ウィザードで、データ ソースの種類として [データベース] を選択します。
5. データ接続は、Northwind が格納されているサーバーを選択します。
6. 次に示す手順で、厳密に型指定されたアプリケーション設定ファイルに接続文字列を保存します。
    a. [データベース オブジェクトの選択] で、`Products` テーブルを選択します。
    b. [完了] をクリックして、Northwind データベースに応じて型指定されたデータセットを作成します。[データ ソース] ウィンドウに結果が表示されます。
7. [データ ソース] ウィンドウで、`Products` テーブルを `Form1` にドラッグします。
8. この結果、データ バインドされた **DataGridView** コントロールと BindingNavigator コントロールがフォームのデザイナ画面に表示されます。また、`NorthwindDataSet`、`ProductsTableAdapter`、および `ProductsBindingSource` がコンポーネント トレイに追加されます。

フォーム内でのデータの読み込み

この例では、フォームへデータを読み込むときに、ユーザーが指定するパラメータは使用しません。データセット デザイナを使用すると、再利用可能な DataAdapter を使用して `dsProducts2` にデータを読み込むことができます。

[データ ソース] ウィンドウから Employees テーブルをドラッグすると、Visual Studio は、TableAdapter で既定のクエリを呼び出すコードを `Form.Load` イベントに自動的に配置します。このサンプルでは、このコードは `btnDC` の Click メソッドに移動してあります。

```
' Fill the Lookup Tables
Me.ProductsTableAdapter.Fill(Me.NorthwindDataSet.Products)
```

参照

処理手順

方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング
方法 : Windows フォームの ComboBox または ListBox コントロールをデータにバインドする

関連項目

ComboBox
DataSource
ArrayList
DataTable
DataView

概念

Windows フォームがサポートするデータ ソース

データ検索のサンプル

Download sample

このサンプルでは、データベースのデータのサブセットを表示する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出 ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

必要条件

このサンプルには、Northwind サンプル データベースが必要です。詳細については、
「方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング」を参照してください。

使用例

このフォームの大部分は、Visual Basic のデザイナ ツールを使用して作成されています。次に示すのは、このフォームを 1 から作成する方法の概要です。

このサンプルを作成するには

1. 新しい Windows アプリケーション プロジェクトを作成します。

2. データ ソース構成ウィザードを使用して、Northwind データベースへの接続を作成し、`Customers` テーブルを選択して、接続文字列をアプリケーション構成ファイルに保存します。

3. [データ ソース] ウィンドウ を開き、`Customers` テーブルをフォームにドラッグします。DataGridView コントロールと BindingNavigator コントロールがフォームに追加され、`Customers` テーブルに合わせて構成されます。

4. **DataGridView** のスマート タグで、[クエリの追加] をクリックします。ダイアログ ボックスで、[新しいクエリ名] を「**FillByLastName**」に変更します。[クエリ テキスト] で、`Select` ステートメントの末尾に「`WHERE LastName LIKE @lastName + '%'`」を追加します。検索のための ToolStrip コントロールがフォームに追加されます。

5. ソリューション エクスプローラで、[NorthwindDataSet] をダブルクリックします。コード ファイル NorthwindDataSet.xsd が開きます。`FillByLastName` メソッドが CustomersTableAdapter に追加されることに注意してください。

参照

処理手順

方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング

概念

データの新機能

TableAdapter の概要

データ検証のサンプル

Download sample

このサンプルでは、データセット デザイナの使用例を示し、PropertyChanged イベントを使用してユーザー入力を検証します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

このサンプルを実行するには

1. このページの [サンプルのダウンロード] ボタンをクリックします。[ファイルのダウンロード] ダイアログ ボックスで、[開く] をクリックします。
2. 使用中のコンピュータにファイルを保存します。
3. Visual Studio で、[ファイル] メニューの [開く] をポイントし、[プロジェクト/ソリューション] をクリックします。
4. ファイルを保存したフォルダを表示し、ソリューション ファイルを選択します。Visual Studio でソリューションが開かれます。
5. F5 キーを押します。

必要条件

このサンプルには、Northwind サンプル データベースが必要です。詳細については、
「方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング」を参照してください。

使用例

このサンプルでは、Visual Studio のデザイナで使用できるデータ検証機能の例を示します。データの検証には、恣意的な架空の規則を 2 つ使用します。コードはデータセット デザイナを使用して追加します。列名をダブルクリックすると、列変更のイベントが作成されます。フィールド リストをダブルクリックすると、行変更のイベントが作成されます。これらのイベント ハンドラで、提示された値を検証するための次のようなコードが追加されています。

```
Private Sub EmployeeDataTable_BirthDateChanging( _
    ByVal sender As EmployeeDataTable, _
    ByVal e As BirthDateChangeEventArg) Handles Me.BirthDateChanging

    Dim minimumAgeDate As New DateTime(DateTime.Now().Year - 18, _
        DateTime.Now().Month, DateTime.Now().Day)
    If e.ProposedValue > minimumAgeDate Then
        e.Row.SetColumnError(e.BirthDateColumn, _
            "Employees must be at least 18 years of age.")
    Else
        e.Row.SetColumnError(e.BirthDateColumn, "")
    End If
End Sub

Private Sub EmployeesDataTable_EmployeesRowChanging( _
    ByVal sender As System.Object, ByVal e As _
    EmployeesRowChangeEvent) Handles Me.EmployeesRowChanging
    If (e.Row.BirthDate = e.Row.HireDate) Then
        e.Row.RowError = e.Row.RowError & _
            "Birth date and hire date are the same."
    Else
        e.Row.ClearErrors()
    End If
End Sub
```

行エラーおよび列エラーが定義されているときには、DataGridView コントロールに警告アイコンが表示されます。行エラーについては、アイコンは行の左に表示されます。列エラーについては、アイコンはセルに表示されます。

参照

処理手順
方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング
関連項目
DataRow
その他の技術情報
データの検証

データベース作成のサンプル

Download sample

このサンプルは、SQL ステートメントをコードから実行する方法を示しています。

コードで、データベース、テーブル、ストアド プロシージャ、およびビューを作成する SQL ステートメントを実行します。次に、Northwind データベースのデータをテーブルに読み込む SQL ステートメントを実行します。次に、このテーブルをクエリして DataSet にデータを読み込んでから、これを DataGrid にバインドして表示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. F5 キーを押します。

2. データベースの作成後にアプリケーションを再度実行する場合は、デモ データベースが含まれている SQL Server インスタンスに対してアクティブな接続を確立したままになっているアプリケーションを閉じる必要があります。つまり、データベースに対する接続を開くときにサーバー エクスプローラを使用した場合は、Visual Studio をいったん閉じて開き直す必要があります。また、その他のアプリケーション (SQL クエリ アナライザなど) が接続を開いたままにしている場合もあります。別のやり方としては、システム トレイから SQL Server サービス マネージャを使用して、SQL Server インスタンスを停止して再起動する方法もあります。

必要条件

このサンプルには、Northwind データベースが必要です。詳細については、
「方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング」を参照してください。

使用例

このアプリケーションは、データベースの作成とデータの読み込みの手順を示します。

- **データベースの作成**　SqlCommand クラスで、CREATE DATABASE ステートメントを使用して `How to Demo` データベースを作成します。
- **データ テーブルの作成**　**SqlCommand** クラスで、CREATE TABLE ステートメントを使用して `NW_Seafood` という名前のテーブルを作成します。このテーブルには、製品 ID、製品名、単位量、および単価を表すフィールドがあります。
- **ストアド プロシージャの作成**　**SqlCommand** クラスで、INSERT INTO ステートメントおよび SELECT ステートメントを使用して、新しいテーブルにコードを追加し、テーブルからデータを取得します。
- **ビューの作成**　**SqlCommand** クラスで、CREATE VIEW ステートメントを使用して、 `NW_Seafood` テーブルの行を選択します。
- **テーブルへのデータの読み込み**　**SqlCommand** クラスで、EXECUTE ステートメントを使用して、Northwind データベースから行を取

得するプロシージャを実行し、それらの行を `NW_Seafood` テーブルに追加します。

- **テーブルのデータの表示**　**SqlCommand** クラスで、SELECT ステートメントを使用して **DataSet** にデータを読み込み、これを **DataGrid** コントロールの DataSource として使用します。テーブルおよび列のスタイル オブジェクトを使用して、**DataGrid** の外観をカスタマイズします。

参照

処理手順

方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング

関連項目

SqlCommand Class

概念

ADO.NET と ADO の比較

その他の技術情報

ADO.NET の概要

ローカル データのサンプル

Download sample

このサンプルでは、Visual Studio 2005 のローカル データ機能 ([データ ソース] ウィンドウ、TableAdapter、BindingSource コンポーネントなど) の例を示します。このサンプルでは、コードを記述せずにデータ バインド フォームを作成する方法と、プロジェクト内のデータベース ファイルに接続する方法も示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [Open] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

   次に、Northwind データベースをインストールします。

サンプルを実行するには

1. Microsoft ダウンロード センター ページ (http://www.microsoft.com/japan/downloads) を参照します。
2. [ダウンロードの検索] ペインの [製品/テクノロジ] ボックスで、[Access] を選択します。
3. [キーワード] ボックスで、「**Northwind**」と入力します。
4. [検索] をクリックします。
5. 結果ページで、[Access 2000 Tutorial: Northwind Traders Sample Database] のリンクをたどり、ダウンロードおよびインストールの指示に従います。
6. Nwind.mdb ファイルをソリューション エクスプローラ ([Local Data] プロジェクト ノード) にドラッグします。

   データ ソース構成ウィザードが開きます。

7. [キャンセル] をクリックして、ウィザードを終了します。
8. F5 キーを押します。

サンプル ドキュメントを表示するには

1. ソリューション エクスプローラで、[ドキュメント] フォルダをダブルクリックします。
2. Visual Basic Express Edition を使用している場合は、[ドキュメント] フォルダ内の ReadMe.htm を右クリックします。[ブラウザで表示] を選択します。
3. Visual Basic の別のバージョンを使用している場合は、[Documentation] フォルダ内の ReadMe.htm をダブルクリックします。

必要条件

このサンプルでは、Nwind.mdb という名前の Access サンプル データベースが必要です。詳細については、

「方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング」を参照してください。

使用例

Visual Studio 2005 には、[データ ソース] ウィンドウが導入されています。これを使用すると、サーバー ベースのデータへのバインディングに加え、ローカル データ、Web サービス、およびビジネス オブジェクトへのバインディングを設定できます。テーブルをフォームにドラッグすると、データ バインド フォームとなり、以前のバージョンでは手作業で完了する必要のあった手順の大半が処理されます。

こうしたアプリケーションを作成する基本的な手順は次のとおりです。

1. Windows アプリケーション プロジェクトを作成します。
2. 新しいデータ ソースを追加します。
3. テーブル列の表示コントロールを選択するか、または既定の選択項目を受け入れます。
4. テーブルをフォームにドラッグします。
5. レコードを挿入、更新、および削除するためのコードを追加します。ToolStrip の Save アイテムの `Click` イベントのコードをチェックします。

参照

処理手順

方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング

その他の技術情報

データのサンプル

オブジェクトのバインドのサンプル

Download sample

このサンプルでは、データベース コンポーネント以外のオブジェクトにコントロールをバインドする方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

サンプル ソリューションには 2 つのプロジェクトが含まれています。`CustomerLibrary` プロジェクトでは、`Customers`、`Orders`、`CustomerManager` など、いくつかのビジネス クラスを定義しています。これらのクラスのインスタンスは、`CustomerManager` クラスが作成します。それらのインスタンスは、クライアント プロジェクトで表示されます。

フォーム デザイナでは、ビジネス オブジェクトをビジネス クラスにリンクする 2 つの BindingSource コントロールを作成します。フォーム上のコントロールは、**BindingSource** コントロールにバインドされます。詳細については、「方法 : オブジェクトのデータに接続する」を参照してください。

参照

処理手順

方法 : オブジェクトのデータに接続する

シリアル化のサンプル

Download sample

このソリューションでは、オブジェクトをシリアル化および逆シリアル化する方法を説明します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このアプリケーションでは 2 つのクラスを使用し、1 つは標準のシリアル化、もう 1 つはカスタムのシリアル化を実行します。ユーザー インターフェイスには、2 つのクラスをシリアル化および逆シリアル化するためのボタンが用意されています。標準のシリアル化を行うクラスは、SOAP ファイルまたはバイナリ ファイルにシリアル化できます。カスタムのシリアル化を行うクラスは、SOAP ファイルにシリアル化されます。SOAP ファイルの表示には TextBox コントロールを使用します。2 つのクラスはいずれも SerializableAttribute 属性を持ちます。これらのクラスの説明を次の表に示します。

クラスのメンバ	SerializationClass	CustomSerializationClass
`PublicVariable`	パブリック変数は変更なしでシリアル化および逆シリアル化されます。	パブリック変数は変更なしでシリアル化および逆シリアル化されます。
`PublicProperty`	プロパティをサポートするプライベート変数は変更なしでシリアル化および逆シリアル化されます。	プロパティをサポートするプライベート変数は、ユーザー インターフェイスで設定された値にかかわらず、-1 としてシリアル化されます。したがって、-1 として逆シリアル化されます。
`NonSerializedVariable`	この変数は、NonSerializedAttribute が設定されているため、シリアル化されません。パブリック変数はシリアル化されず、逆シリアル化された値は 0 です。	**NonSerializedAttribute** が設定されている変数ですが、カスタムのシリアル化によりこの設定がオーバーライドされているため、シリアル化されます。パブリック変数は変更なしでシリアル化および逆シリアル化されます。

3 種類のシリアル化の例が示されます。

- **SOAP 形式** `Class1` は SoapFormatter クラスを使用してシリアル化されます。
- **バイナリ形式** `Class1` は BinaryFormatter クラスを使用してシリアル化されます。
- **カスタム形式** `Class2` は ISerializable インターフェイスを実装しており、カスタムのシリアル化を実行するための GetObjectData メソッドが含まれています。

参照

関連項目

System.Runtime.Serialization.Formatters.Soap
Serialize
Deserialize
ISerializable
GetObjectData
SerializableAttribute

ストアド プロシージャのサンプル

Download sample

このアプリケーションでは、ADO.NET および Microsoft SQL Server を使用して、プログラムでストアド プロシージャを作成および使用する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

必要条件

このサンプルには、Northwind サンプル データベースが必要です。詳細については、
「方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング」を参照してください。

使用例

以下のタスクの例を示します。

- SQL コマンドとリソース ファイルを使用して新しいデータベースを作成し、実行するスクリプトを制御します。新しいデータベースは、新しいテーブルを使用して作成されます。テーブルには、追加のスクリプトが設定されます。追加のスクリプトを実行すると、このサンプルで使用するストアド プロシージャが作成されます。
- SqlDataReader オブジェクトを使用して、パラメータが不要なストアド プロシージャを実行します。
- 入力パラメータが必要なストアド プロシージャを実行して、SqlDataAdapter の結果をグリッド コントロールに読み込みます。
- 入力パラメータと出力パラメータの両方が必要なストアド プロシージャを実行して、戻り値を利用します。

参照

処理手順

方法 : 単一の値を返す SQL ステートメントを作成および実行する
方法 : 値を返さないストアド プロシージャを実行する
方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング

概念

データ アダプタ コマンドのパラメータ
DataReader によるデータの取得
TableAdapter の概要
データの新機能

XMLDocument クラスのサンプル

Download sample

このサンプルでは、System.Xml 名前空間の一部である、XmlDocument クラスの使用方法を示します。このクラスは、XML DOM (Document Object Model) を公開しており、XML データの操作に使用できます。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

フォームは次の 3 つの部分に分かれています。

- タスクの一覧
- XML の表示領域
- 実行結果の表示領域

3 つの領域はいずれも、動的に読み込みおよび更新されます。一連のボタン コントロールではなく、アイテムのコレクションを使用しているため、必要なオプションをいくつでも追加でき、デザイナでコントロールを再整列する必要がありません。

サンプルは、読み込み時に、プロジェクト フォルダの \bin フォルダに次の 3 つのファイルが存在するかどうかをチェックします。

- Simple.xml
- New.xml
- Bad.xml

ファイルが存在する場合は、CheckedListBox コントロールに読み込まれている一連のコマンドを選択できます。\bin フォルダに対しては、読み取りと書き込みのアクセス許可が必要です。一部のコマンドで変更をディスクに保存する必要があるからです。

参照

処理手順

方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング

関連項目

XmlDocument

System.Xml

Visual Basic Windows フォームのサンプル

これらのサンプルでは、Windows フォーム アプリケーションの一般的なタスクの例を示します。

このセクションの内容

アニメーションのサンプル

テキストの移動、ボールの跳ね返り、およびイメージのアニメーションが含まれています。

クリップボードのサンプル

My.Clipboard オブジェクトを使用した、テキスト、整数、イメージ、およびオブジェクトのコピーと貼り付けの例を示します。

ユーザー コントロールのカスタム描画のサンプル

カスタム描画のユーザー コントロールを作成および使用する方法の例を示します。

ダイアログ コンポーネントのサンプル

ファイル選択、色選択、およびフォルダ選択の例を示します。

ドラッグ アンド ドロップのサンプル

テキスト、イメージ、およびツリー ビュー ノードのドラッグ アンド ドロップ機能の例を示します。

ダイナミック コントロール作成のサンプル

XML 入力ファイルを使用して、実行時にコントロールを追加します。

エクスプローラ スタイルのアプリケーションのサンプル

Windows エクスプローラに似た、分割ペイン ウィンドウから成る 2 つのフォームが含まれています。

GDI+ ブラシのサンプル

ブラシ オプションの効果を参照します。

GDI+ イメージのサンプル

ズームやクリッピングなど、イメージを操作します。

GDI+ ペンのサンプル

ペン オプションの効果を表示します。

GDI+ テキストのサンプル

テキスト オプションの効果を表示します。

ヘルプ ファイルのサンプル

コンパイルされたヘルプ、HTML ヘルプ、およびツールヒントの例を示します。

ListBox および ComboBox のサンプル

リストの一般的なタスクの例をいくつか示します。

メニューのサンプル

プログラムでメニューを作成およびマージします。

NotifyIcon のサンプル

システム トレイからアクセスできるアプリケーションを作成します。

印刷のサンプル

My.Printers オブジェクトを使用します。

正規表現のサンプル

Regex を使用してユーザー入力を検証します。

StatusStrip コントロールのサンプル

テキストおよびユーザー描画の StatusBar パネルの例を示します。

テキスト検証のサンプル

MaskedTextBox を使用してテキストを検証します。

スレッドによるユーザー インターフェイス更新のサンプル

別のスレッドからユーザー インターフェイスを更新する方法の例を示します。

トップレベル フォームのサンプル

タスク バーでのアプリケーションの表示方法を制御します。

ビジュアルな継承のサンプル

既存のフォームを継承して新しいフォームを作成します。

Web ブラウザのサンプル

Web ブラウザを作成します。

参照

.NET Framework における Windows フォーム アプリケーション

Windows フォーム アプリケーションの設計についての情報です。

Windows フォーム コントロール

Windows フォーム コントロールについての情報です。

関連するセクション

データのサンプル

これらのサンプルでは、データ操作タスクの例を示します。

Visual Basic 言語のサンプル

これらのサンプルでは、Visual Basic 言語の概念を示します。

Visual Basic のサーバー コンポーネントのサンプル

これらのサンプルでは、Windows オペレーティング システムのコンポーネントとやり取りするアプリケーションの例を示します。

Visual Basic のセキュリティのサンプル

これらのサンプルでは、セキュリティ タスクの例を示します。

参照

処理手順

FlowLayoutPanel コントロールのサンプル
MaskedTextBox コントロールのサンプル
TableLayoutPanel コントロールのサンプル
BindingNavigator コントロールのサンプル
SplitContainer コントロールのサンプル

その他の技術情報

Windows フォーム コントロールの例

アニメーションのサンプル

Download sample

このサンプルでは、System.Drawing 名前空間のオブジェクトを使用してアニメーションを実装する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、Timer オブジェクトを使用してアニメーションの変化を制御し、RadioButton コントロールを使用してアニメーション オブジェクトを選択します。次の 3 つのアニメーション スタイルを示します。

- **フレーム アニメーション。**ウィンクする目は、一連の静的なイメージ (フレーム) として実装されています。タイマ刻みのたびに、次のフレームを表示します。
- **飛行物体。**弾むボールのサイズと速度は、クライアント領域に対して相対的です。このアニメーションは、フォームの Graphics オブジェクトへの描画により実装されています。
- **テキスト アニメーション。**LinearGradientBrush オブジェクトを使用してテキストを描画しています。タイマ刻みのたびに、**LinearGradientBrush** のパラメータを変更して、異なるブラシを作成します。このパラメータにより、グラデーション塗りつぶしの開始点と終了点が移動します。

参照

関連項目

Graphics
LinearGradientBrush

概念

グラフィックスについて

クリップボードのサンプル

Download sample

このサンプルは、クリップボードのアイテムをさまざまな形式で配置および取得する方法を示しています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

サンプル ドキュメントを表示するには

1. ソリューション エクスプローラで、[ドキュメント] フォルダをダブルクリックします。
2. Visual Basic Express Edition を使用している場合は、[ドキュメント] フォルダ内の ReadMe.htm を右クリックします。[ブラウザで表示] をクリックします。
3. Visual Basic の別のバージョンを使用している場合は、[Documentation] フォルダ内の ReadMe.htm をダブルクリックします。

使用例

メイン フォームのメニュー コマンドで、クリップボードを使用する 2 つの主要なタスク (クリップボードへのコピーとクリップボードからの取得) を制御します。My.Computer.Clipboard オブジェクトを使用して、次の 6 つのデータ型がコピーおよび貼り付けされます。

- イメージ　この機能では、My.Computer.Clipboard.SetImage メソッドおよび My.Computer.Clipboard.GetImage メソッドを使用して、PictureBox コントロール間でピクチャをコピーします。イメージ ファイルは **My.Resources** に格納されています。
- テキスト　この機能では、My.Computer.Clipboard.SetText メソッドおよび My.Computer.Clipboard.GetText メソッドを使用して、TextBox コントロール間でテキストをコピーします。My.Computer.Clipboard.SetText メソッドはオーバーロードされています。このタスクで使用しているオーバーロードはパラメータを持たず、クリップボードに格納されている任意の種類のテキストをコピーします。
- リッチ テキスト　"テキスト" 機能と同様に、My.Computer.Clipboard.SetText メソッドおよび My.Computer.Clipboard.GetText メソッドを使用してテキストをコピーおよび貼り付けます。この機能では、My.Computer.Clipboard.SetText メソッドのオーバーロードを使用しており、これは TextDataFormat 列挙体を受け取ります。渡される値は Rtf なので、リッチ テキストのみがコピーされます。
- HTML　"リッチ テキスト" 機能と同様に、**TextDataFormat** 列挙体の値 (Html) を使用して、クリップボードから HTML のみを取得します。
- クラス インスタンス　`Pixel` クラスがプロジェクトで定義されており、`Serializable` 属性を持ちま

す。My.Computer.Clipboard.SetData メソッドおよび My.Computer.Clipboard.GetData メソッドを使用して、データをコピーおよび貼り付けできます。`LoadPixel` メソッドは、フォームに書き込む文字列を作成します。

- ファイル　Windows エクスプローラからファイルをコピーすると、クリップボードにコピーされます。ファイル名の取得には My.Computer.Clipboard.GetFileDropList メソッドを使用します。

いずれの場合も、左の列のコピー元のコントロールからデータがコピーされ、右の列のコピー先のコントロールに貼り付けられます。

参照

処理手順

方法 : Visual Basic でクリップボードに書き込む
方法 : Visual Basic でクリップボードを消去する
方法 : Visual Basic でクリップボードから読み込む
方法 : Visual Basic でクリップボードからイメージを取得する

関連項目

My.Computer.Clipboard オブジェクト

ユーザー コントロールのカスタム描画のサンプル

Download sample

このサンプルでは、カスタム描画のユーザー コントロールを作成および使用する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、カスタム描画のユーザー コントロールを作成する方法、フォームで使用する方法、イベントに応答させる方法、およびデザイナで使用するプロパティを定義する方法の例を示します。作成するのは、スコア記録のコントロールです。1 つはデジタル スコアボードで、数値をデジタル形式で表示します。もう 1 つはビーズ式のスコアボードで、ビーズを前後にスライドさせてスコアを追跡します。いずれも、さまざまなイベントに応答し、またデザイン時に利用できるプロパティが多数あります。

- **カスタム描画**　このサンプルのカスタム コントロールはいずれも、UserControl クラスを継承して OnPaint をオーバーライドする方法の例を示しています。カスタム描画のユーザー コントロールは、**UserControl** を継承します。カスタム描画は、コントロールの **OnPaint** メソッドをオーバーライドすることにより実行します。**OnPaint** メソッドには PaintEventArgs 引数があり、クリップ四角形 (Rectangle) とグラフィックス オブジェクト (Graphics) をそこから取得します。

- **カスタム コントロール イベント**　カスタム コントロールでは、通常のコントロールが応答するのと同じイベントに応答できます。これは、**UserControl** クラス内でイベント ハンドラ メソッドをオーバーライドすることによって行われます。このサンプルのカスタム コントロールはいずれも、MouseUp や Click などのさまざまなイベントに応答する方法の例を示しています。

- **プロパティ**　通常は、Visual Studio を使用してデザイン時に編集できるコントロールについては、プロパティを作成することが望まれます。このサンプルのカスタム コントロールはいずれも、Visual Studio の [プロパティ] ウィンドウに表示される **Public** プロパティを作成する方法の例を示しています。

参照

処理手順

フォームへのイメージの描画

グラフィックスの表示

その他の技術情報

コントロールのカスタム描画およびレンダリング

ダイアログ コンポーネントのサンプル

Download sample

このサンプルでは、OpenFileDialog、SaveFileDialog、ColorDialog、および FontDialog の各コンポーネントの例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、**OpenFileDialog**、**SaveFileDialog**、**ColorDialog**、および **FontDialog** の各コンポーネントの例を示します。サンプル フォームには、2 つのページを持つ TabControl コントロールがあります。1 つ目のタブ ページでは、4 つのコンポーネントを使用して、ファイルのオープン、ファイルの保存、およびテキストのフォントと色の設定を行います。2 つ目のタブ ページでは、FileNames プロパティと Multiselect プロパティを使用して、複数のファイルを選択できます。

参照

関連項目

FileNames
Multiselect

その他の技術情報

OpenFileDialog コンポーネント (Windows フォーム)
SaveFileDialog コンポーネント (Windows フォーム)
ColorDialog コンポーネント (Windows フォーム)
FontDialog コンポーネント (Windows フォーム)

ドラッグ アンド ドロップのサンプル

Download sample

このサンプルには、Windows アプリケーション内のフォームのドラッグ アンド ドロップ機能に関する、次の 3 つの例が含まれています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルには、ドラッグ アンド ドロップの 3 つの例が含まれています。

- **TextBox**　コントロール上でのドロップを防ぐ方法を示します。1 つの TextBox コントロールから他の 2 つへテキストをドラッグできます。そのうちの 1 つでは、AllowDrop プロパティが **True** に設定されていません。
- **TreeView**　TreeView コントロール間でノードを移動する方法を示します。
- **PictureBox**　イメージをコピーする方法を示します。PictureBox 間でイメージをドラッグできます。AllowDrop プロパティは [プロパティ] ウィンドウに表示されず、コードで **True** に設定されます。

ドラッグするデータは、`String`、`Bitmap`、または `MetaFile` の各クラスのインスタンスであるか、あるいは ISerializable インターフェイスまたは IDataObject インターフェイスを実装したオブジェクトである必要があります。

参照

処理手順

チュートリアル : Windows フォームにおけるドラッグ アンド ドロップ操作の実行

関連項目

Metafile Class
ISerializable Interface
IDataObject Interface
AllowDrop

ダイナミック コントロール作成のサンプル

Download sample

このサンプルでは、実行時に Windows アプリケーションのフォームにコントロールを追加する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

フォームにコントロールを追加するためのいくつかの方法の例を示します。メイン フォームへのコントロールの追加は、コントロールの新しいインスタンスを宣言し、フォームの ControlCollection コレクションに参照を追加することで行います。もっと複雑なシナリオとしては、新しいフォームのインスタンスを作成し、そのフォームにコントロールを追加する例を示します。作成するコントロールは XML ファイルで指定します。

参照

関連項目

System.Windows.Forms
Control

その他の技術情報

Windows フォーム内でのイベント ハンドラの作成
イベントの処理と発生

エクスプローラ スタイルのアプリケーションのサンプル

Download sample

このサンプルには、`DirectoryScanner` および `ExplorerStyleViewer` という、エクスプローラに似た 2 つのフォームが含まれています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルには、エクスプローラ風のインターフェイスを持つフォームが 2 つあります。1 つはディレクトリ スキャナ、もう 1 つはエクスプローラ スタイルのビューアです。2 つのフォームをサポートするファイルは、プロジェクトの別個のフォルダに格納されています。

- **DirectoryScanner** すべての論理ドライブまたはユーザーが選択した開始ディレクトリの配下にある、すべてのディレクトリおよびサブディレクトリをスキャンする、簡単なアプリケーションです。ドライブのリストの取得には、Directory.GetLogicalDrives メソッドを使用します。最新のスキャンを反映したディレクトリ構造はツリー ビュー コントロールに表示します。ディレクトリは、サブディレクトリおよびファイルをすべて含めた合計サイズに基づいて、緑、黄、または赤に色分けされています。ファイルのリストの取得には Directory.GetFiles メソッドを使用し、ファイル サイズの取得には FileInfo クラスを使用します。

- **ExplorerStyleViewer** Windows エクスプローラ アプリケーションの簡易バージョンです。`ExplorerStyleViewer` では、FileSystemInfo.Attributes プロパティを使用して、`DirectoryScanner` よりも詳しいファイル情報を表示します。この中では、TreeView.ImageList プロパティを使用してアイコンをファイルの種類と関連付ける方法の例を示します。ファイルの種類と関連付けられたアプリケーションがある場合、ユーザーは、Windows エクスプローラと同様に、ファイルをダブルクリックすることによってそのアプリケーションを実行できます。アプリケーションの起動には Process.Start メソッドを使用します。

参照

関連項目

Directory Class
FileInfo Class
TreeView.ImageList Property
ListView Class
TreeView Class

その他の技術情報

ListView コントロール (Windows フォーム)
TreeView コントロール (Windows フォーム)

GDI+ ブラシのサンプル

Download sample

このサンプルでは、Brush オブジェクトのいくつかの機能の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

ユーザーは、ブラシの表示を左右するさまざまなパラメータを選択できます。各パラメータを選択した後で、ユーザーが選択した図形がピクチャ ボックスに再描画され、新しいブラシの例が示されます。PathGradientBrush や LinearGradientBrush など、比較的高度なオブジェクトも含まれています。

参照

処理手順

方法 : 描画する Graphics オブジェクトを作成する

その他の技術情報

GDI+ マネージ コードについて

GDI+ イメージのサンプル

Download sample

このサンプル アプリケーションは、GDI+ を使用してイメージを操作する方法を示しています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、次のような例が示されます。

- **ズーム**　ズーム機能では、画面上でのイメージのサイズが変更されます。実行するズームの度合いは、イメージの元のサイズに対する倍率で表すのが一般的です。ズームを実行しても、元のイメージのサイズは変更されません。画面上に表示されるイメージのサイズが変更されるだけです。このサンプルでは、`Zoom` メソッドでこの例が示されます。

- **サイズ変更**　サイズ変更機能では、イメージの実際のサイズが変更されます。サイズ変更の比率に従って、幅と高さの両方が変更されます。このサンプルでは、`ResizeImage` メソッドでこの例が示されます。

- **反転および回転**　反転では、x 軸または y 軸を中心としてイメージが反転移動されます。その結果、鏡映されたイメージか、上下逆になったイメージのどちらかとなります。回転では、中心の周りで、一定の角度だけイメージが回転されます。イメージの場合、90 度の倍数の角度だけ回転することが特によくあります。このサンプルでは、反転および回転の例は `RotateFlip` メソッドで示されます。

- **反転およびグレースケール**　ColorMatrix を使用して、反転またはグレースケールのイメージを作成します。イメージに適用するときには、**ColorMatrix** が色変換の基盤となります。このサンプルでは、`DrawNegativeImage` メソッドおよび `ConverttoGrayScale` メソッドでこの例が示されます。

- **トリミング**　トリミングでは、イメージの一部の領域を別個のイメージとして取り出します。このサンプルでは、`CropButton_Click` メソッドでこの例が示されます。

- **サムネイル**　イメージは、通常のサイズかサムネイルのどちらかで保存できます。サムネイルを作成する方法の 1 つには、イメージの複製を作成し、サイズを変更して、その複製をサムネイルとして保存するという方法があります。このサンプルでは、`SaveThumbnailAs_Click` メソッドでこの例が示されます。

参照

処理手順

方法 : 描画する Graphics オブジェクトを作成する

方法 : イメージをトリミングおよびスケーリングする
方法 : イメージを回転、反転、および傾斜させる
方法 : サムネイル イメージを作成する

その他の技術情報

Windows フォームにおけるグラフィックスと描画

GDI+ ペンのサンプル

Download sample

このサンプルでは、Pen オブジェクトのいくつかの機能の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

デモンストレーション

ユーザーは、いくつかのコントロールを使用して、以下のペンの特性を選択できます。

- Width
- DashStyle
- MiterLimit
- StartCap
- EndCap
- DashCap
- LineJoin
- Alignment

またユーザーは、以下の 3 つの変換も選択できます。

- ScaleTransform
- RotateTransform
- TranslateTransform

ユーザーは、**Pen** と変換を設定したら、描画する図形を選択できます。

- **直線**　線の描画には DrawLine メソッドを使用します。
- **交差する直線**　線の描画には DrawLines メソッドを使用します。

- **円および曲線**　線の描画には DrawEllipse メソッドと DrawArc メソッドを使用します。

参照

関連項目

Pen

その他の技術情報

GDI+ マネージ コードについて

GDI+ テキストのサンプル

Download sample

このサンプルは、GDI+ を使用してテキストを扱うときに利用できる数多くの機能の一部を示しています。影、浮き出し、ブロック テキスト、シャーリング、反転など、いくつかの効果をデモンストレーションします。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

サンプル ドキュメントを表示するには

1. ソリューション エクスプローラで、[ドキュメント] フォルダをダブルクリックします。
2. Visual Basic Express Edition を使用している場合は、[ドキュメント] フォルダ内の ReadMe.htm を右クリックします。[ブラウザで表示] を選択します。
3. Visual Basic の別のバージョンを使用している場合は、[Documentation] フォルダ内の ReadMe.htm をダブルクリックします。

使用例

Brush クラスおよび Pen クラスのプロパティを設定することで実装できる効果もありますが、次に示すように、コードが必要な効果もあります。

- **影**　影を付けるには、テキストを 2 回描画します。1 回目は灰色でオフセット位置に描画します。2 回目は黒で描画します。
- **浮き出し**　浮き出し効果を付けるには、テキストを 2 回描画します。1 回目は黒でオフセット位置に描画します。2 回目は現在の背景色である白で描画します。
- **ブロック テキスト**　この効果を得るには、黒でテキストの描画を繰り返し、描画のたびにテキストを上および右に移動します。次にテキストをメインの色で描画します。
- **シャーリング**　描画におけるシャーリングとは、イメージを傾斜させることを表します。テキストを傾斜させるには、Graphics オブジェクトの Transform プロパティの Shear メソッドを呼び出します。
- **反転**　テキストを反転するには、テキストの高さを測り、テキストをスケーリングして、原点の位置を変更する必要があります。MeasureString メソッドが返す高さには、ディセンダと空白の分の余分なスペーシングが含まれていますが、テキストはベースライン (すべての大文字の下端に位置する線) を中心に反転されます。ベースラインの上の高さを算出するには、GetCellAscent メソッドを使用します。**GetCellAscent** が返す値はデザイン メトリック値なので、ピクセルに変換し、フォント サイズに合わせてスケーリングする必要があります。テキストはスケーリング変換を使用して反転されますが、テキストがフォームの可視領域の外に描画されるのを防ぐために、最初に

新しい原点が設定されます。次に、反転されたテキストをまず描画します。これは GraphicsState クラスのデモンストレーションです。反転されたテキストを最初に描画しているのは、**GraphicsState** オブジェクトの使用をデモンストレーションするというだけの理由からです。最後に縦向きのテキストを描画します。

参照

関連項目

Graphics Class
Graphics.DrawString Method
Pen Class
Graphics.Transform Property
Graphics.Save Method
Brush
Shear
MeasureString
GraphicsState

ヘルプ ファイルのサンプル

Download sample

このサンプルは、ToolTip、HelpProvider、および ErrorProvider の各コントロールを使用して Windows ベースのアプリケーションにヘルプを追加する方法を示しています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

TabControl の各タブ ページには、それぞれ異なる種類のヘルプが表示されます。

- **ツールヒント**　**ToolTip** コンポーネントがフォームに追加されています。これにより、フォーム上の各コントロールに `ToolTip1` プロパティが追加されます。

- **ポップアップ**　`hpAdvancedCHM` という **HelpProvider** コンポーネントがフォームに追加されています。これにより、フォーム上の各コントロールに、ヘルプ文字列を設定する Help String on hpAdvancedCHM プロパティが追加されます。また、各コントロールには ShowHelp on hpAdvancedCHM プロパティも追加されており、**True** に設定されています。フォームの HelpButton プロパティは **True** に設定されており、隅の閉じるボタンの横に疑問符のボタンが表示されます。ヘルプ ボタンを表示するためには、最小化ボタンおよび最大化ボタンはオフにする必要があります。

- **HTML ヘルプ**　`hpAdvancedCHM` という **HelpProvider** コンポーネントがフォームに追加されています。これにより、HelpKeyword on hpAdvancedCHM、HelpNavigator on hpAdvancedCHM、および ShowHelp on hpAdvancedCHM の各プロパティが各コントロールに追加されます。ShowHelp on hpAdvancedCHM プロパティは **True** に設定されています。コンパイル済みヘルプ ファイルでは、指定されたキーワードに対応するヘルプ ページが表示されます。フォームの **HelpButton** プロパティは **True** に設定されており、隅の閉じるボタンの横に疑問符のボタンが表示されます。ヘルプ ボタンを表示するためには、最小化ボタンおよび最大化ボタンはオフにする必要があります。

- **エラー プロバイダ**　適切なデータ型に検証できないコントロールについて、**ErrorProvider** コンポーネントを使用してユーザーに警告を表示する方法を示します。

参照

関連項目

Form.HelpButton Property

その他の技術情報

HelpProvider コンポーネント (Windows フォーム)

ToolTip コンポーネント (Windows フォーム)
ErrorProvider コンポーネント (Windows フォーム)

ListBox および ComboBox のサンプル

Download sample

このサンプルでは、ListBox および ComboBox の基本的なタスク (項目の追加、データベース テーブルへのバインディング、選択された項目へのアクセスなど) の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

必要条件

このサンプルには、Northwind データベースが必要です。詳細については、
「方法 : サンプル用のデータベース コンポーネントのインストールおよびトラブルシューティング」を参照してください。

使用例

タブ コントロールのタブ付きページを使用して、次のようなさまざまな手法の例を示します。

- 項目の追加　このサンプルでは、コンピュータ上で現在実行中のプロセスを表す Process オブジェクトの配列を取得します。Process オブジェクトが、**ListBox** コントロールの Items コレクションに追加されます。**Process** オブジェクトのどのプロパティを **ListBox** コントロールに表示するかは、**ListBox** コントロールの DisplayMember プロパティを使用して指定します。

- DataTable へのバインド　このタブ ページのコードでは、[マイ ドキュメント] フォルダにあるすべてのファイルのリストを DataTable に読み込んでから、リスト ボックスをその **DataTable** にバインドします。

- 配列へのバインド　このタブ ページのコードは、[項目の追加] ページと似ています。このコードでは、**Process** オブジェクトを 1 つずつ **Items** コレクションに追加するのではなく、**Process** オブジェクトの配列全体を **ListBox** コントロールの DataSource として使用します。項目の表示および取得の制御には、**ListBox** コントロールの ValueMember プロパティおよび **DisplayMember** プロパティを使用します。

- ComboBox　このタブ ページのコードは、TableAdapter および型指定されたデータセットという新しい機能を使用して、SQL Server から取得した **DataTable** に **ComboBox** コントロールを バインドします。この例では、ローカル コンピュータの SQL Server に Northwind サンプル データベースが含まれているものと想定しています。このページのコントロールでは、**ComboBox** コントロールの動作を制御するレイアウト プロパティを指定できます。

参照

関連項目

ListControl.DisplayMember Property

ListControl.ValueMember Property
ListBox.SelectedIndexChanged Event
ListBox.SelectedIndices Property
ListBox.SelectedItems Property
ListBox.SelectionMode Property
Process Class
Process Class

概念

データの新機能

その他の技術情報

ListBox コントロール (Windows フォーム)
ComboBox コントロール (Windows フォーム)

メニューのサンプル

Download sample

このサンプルでは、Windows アプリケーション内での MenuStrip コントロールの使用方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、簡単な Windows アプリケーションで **MenuStrip** コントロールをテストできます。このサンプルでは、メニュー ストリップのいくつかの機能の例を示します。具体的には、コントロール、メニュー イベント、メニュー プロパティ、複数のイベントの処理、メニュー イベントへのイベント ハンドラの割り当て、実行時のメニュー ストリップに対するオブジェクトの追加と削除、ショートカットの表示、および StatusStrip のインジケータとしての使用です。いくつかの機能についての詳しい説明を以下に示します。

- **ショートカット** `Form1` の [Options List] メニューなど、一部のメニュー項目にはショートカットが含まれています。ShowShortcutKeys プロパティを使用すると、メニューにこの情報を表示するかどうかを切り替えることができます。ショートカット キーを使用すると、アプリケーションが使いやすくなります。

- **StatusStrip コントロール** [View] メニューの [Status Strip] では、チェック式のメニュー項目を使用して、**StatusStrip** コントロールの表示と非表示を切り替えます。

- **チェック ボックス付きリスト** `MainMenuStrip` の [Checked List] メニューは、一度に 1 つしか選択できないチェック式のメニュー項目のコレクションを作成する方法を示しています。このコレクションに含まれる項目を 2 つ同時にオンにすることはできません。作成方法については、`MenuOption_Click` イベント ハンドラを参照してください。

- **メニュー項目を実行時に追加** `CreateInitialMenus` プロシージャは、ToolStripMenuItem、ToolStripSeparator、ToolStripComboBox、ToolStripButton、ToolStripSplitButton、または ToolStripLabel を実行時に作成および追加する方法を示します。コレクションに対して項目を追加または削除するには、DropDownItems プロパティを使用します。

- **AddHandler** `CreateInitialMenus` プロシージャでは、**AddHandler** を使用して、実行時に作成したメニュー項目の特定のイベントに対してイベント ハンドラを割り当てます。

参照

処理手順

方法 : デザイナを使用して標準アイテムで基本的な Windows フォーム ToolStrip を作成する

関連項目

ToolStrip コントロールの概要 (Windows フォーム)

概念

ToolStrip テクノロジの概要

その他の技術情報

MenuStrip コントロール (Windows フォーム)

ToolStrip コントロール (Windows フォーム)

StatusStrip コントロール

NotifyIcon のサンプル

Download sample

このサンプルでは、タスク バーの状態通知領域にアイコンを表示するプログラムの作成方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このアプリケーションでは、NotifyIcon コントロール、およびそれに関連付けられた ContextMenu を使用して、以下のシステム データにアクセスし、それを表示します。

- ユーザーのコンピュータの最後の再起動から経過した時間の長さ (分単位)
- ユーザーの現在のタイム ゾーン
- 現在の日付
- ユーザーのコンピュータにインストールされている .NET Framework のバージョン
- ユーザーがアプリケーションを実行中のオペレーティング システムの現在のバージョン

参照

関連項目

NotifyIcon

その他の技術情報

NotifyIcon コンポーネント (Windows フォーム)

印刷のサンプル

Download sample

このサンプル アプリケーションは、テキストの印刷とプリンタの設定の方法を示しています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

サンプル ドキュメントを表示するには

1. ソリューション エクスプローラで、[Documentation] フォルダをダブルクリックします。
2. Visual Basic Express Edition を使用している場合は、[Documentation] フォルダ内の ReadMe.htm を右クリックします。[ブラウザで表示] を選択します。
3. Visual Basic の別のバージョンを使用している場合は、[Documentation] フォルダ内の ReadMe.htm をダブルクリックします。

使用例

このサンプルは、数段落のテキストを印刷し、印刷プレビュー、印刷設定、およびページ設定をデモンストレーションします。

参照

その他の技術情報

Visual Basic ガイド ツアー

正規表現のサンプル

Download sample

このサンプル アプリケーションでは、正規表現の使用方法を示します。メイン フォームには、3 つのタブ ページを持つ TabControl があります。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

必要条件
このサンプルの画面スクレーピング機能を使用するには、インターネット接続が必要です。

使用例
このサンプル アプリケーションでは、正規表現の使い方を示します。タブ ページを使用して、次の例を示します。

- **基本的な解析手法**　このタブ ページでは、Regex クラスを使用して、文字列内の数字、有効な郵便番号、電子メール アドレス、および日付を見つけます。
- **画面スクレーピング**　このタブでは、正規表現のもっと高度な使用法を示します。Web アドレスを入力すると、そこに含まれているリンクまたはイメージを表示できます。HTML タグ属性は ListView コントロールに表示されます。イメージをダブルクリックするとそのイメージを参照できます。リンクをダブルクリックすると Internet Explorer が開いてそのページを参照できます。
- **テスト**　このタブは、正規表現を手早く試してみるのに便利です。また、検索および置換の方法や、文字列分割の方法も示します。

参照
関連項目
Regex Class
その他の技術情報
.NET Framework の正規表現

StatusStrip コントロールのサンプル

Download sample

このサンプルでは、さまざまな種類のデータを StatusStrip コントロールにビジュアルに表示する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

以下の種類のデータを表示します。

- **テキスト**　パネルにテキストを表示します。
- **キーボードのステータス**　CapsLock キーと NumLock キーのステータスを表すテキストをパネルに表示します。
- **日付**　現在のカルチャで日付をパネルに表示します。
- **プログレス**　ProgressBar コントロールを **StatusStrip** コントロールに表示します。

参照

関連項目

StatusStrip

その他の技術情報

StatusStrip コントロール

テキスト検証のサンプル

Download sample

このサンプルでは、次の 2 種類のテキスト検証方法を比較します。

- カスタム TextBox コントロールで正規表現を使用する方法
- MaskedTextBox コントロールを使用する方法

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、特定の種類の入力用として設計した特別な用途のテキスト ボックスを作成する方法の例を示します。このサンプルには、電子メール アドレス用、IP アドレス用、および電話番号用のテキスト ボックスが含まれています。

MaskedTextBox コントロールでは、ユーザーが入力を行っている時点で検証が実行されます。したがって、不正なテキストは **MaskedTextBox** に表示されません。

`RegExTextBox` クラスは、このサンプルで **TextBox** から派生したもので、他のいくつかのクラスの基本クラスとなっています。このクラスは、**TextBox** コントロールの内容を所定の正規表現に照らして検証します。ユーザーがメイン フォームの [検証] ボタンをクリックすると、コードはコントロールのコレクションをループし、すべての `RegExTextBox` コントロールを見つけます。そしてそれらを、この基本型にキャストしてから、`IsValid` プロパティを呼び出します。入力が無効の場合、**TextBox** のテキストの色は赤になります。

参照

関連項目

正規表現 (Visual Studio)

スレッドによるユーザー インターフェイス更新のサンプル

Download sample

このサンプルでは、別のスレッドからユーザー インターフェイスを更新する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

ユーザー インターフェイスは、別のスレッドで実行されているプロシージャからは更新できません。このサンプルは、別のスレッドからデータが与えられる場合に、ユーザー インターフェイスを更新する方法の例を示します。

参照

その他の技術情報

Visual Basic におけるマルチスレッド

トップレベル フォームのサンプル

Download sample

このサンプルでは、Microsoft Word のような、複数のトップレベル フォームを持つアプリケーションの例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. F5 キーを押します。

2. [ファイル] メニューの [New] コマンドを使用して複数の文書ウィンドウを作成し、すべての文書がタスク バーに表示されることを確認します。

3. タスク マネージャを使用して、このアプリケーションのプロセスは 1 つのみが実行中であることを確認します。

使用例

文書を扱うアプリケーションの多くは、複数ウィンドウをサポートしています。各文書をそれぞれ別個のウィンドウで開き、それらがタスク バーに表示されます。文書間の切り替えには Alt + Tab キーを使用できますが、各文書をそれぞれ別個のプロセスで読み込む場合のようなオーバーヘッドはかかりません。このサンプルは、その動作を実装します。

アプリケーションのスタートアップ オブジェクトは、`FormsManager` というクラス内の `Sub Main` メソッドです。`Main` メソッドでは、`FormsManager` クラスの `NewForm` メソッドを呼び出して新しいフォーム インスタンスを作成し、その新しいフォームを List オブジェクトに追加します。最初のフォームを作成したら、System.Windows.Forms.Application.Run を呼び出して、メイン アプリケーション スレッドを開始します。このため、最初のフォームを閉じてもプロセスは終了しません。各文書フォームには、[New] メニュー項目が用意されています。このメニュー項目がクリックされると、`NewForm` メソッドを再度呼び出し、新しい文書フォームを開きます。アプリケーションの起動時に `Sub Main` で開いたのと同様です。

各フォームには、[閉じる] および [終了] メニュー項目も用意されています。[閉じる] がクリックされると、クローズ処理を開始します。各フォームには Closing イベントがあります。これにより、文書の内容が変更されているかどうかの確認と、その場合に保存を実行するかどうかユーザーに指定してもらう処理を、フォームのコードで行うことができます。

変更されている文書を閉じるとき (そのフォームを閉じるか、またはアプリケーションを終了するとき) には、フォームの内容を保存するかどうかを問い合わせるダイアログ ボックスが表示されます。[はい] をクリックすると、コードはフォームの `Save` メソッドを呼び出し、フォームを閉じます。[No] をクリックすると、フォームはそのまま閉じます。[キャンセル] をクリックすると、フォームは閉じず、終了処理を中断するようアプリケーションに通知するカスタム イベントが発生します。このアプリケーションでは、ファイルを保存するためのファイル I/O は実際には実行していません。

参照

関連項目

System.Windows.Forms.Form.Closing
System.Windows.Forms.Application.Run

ビジュアルな継承のサンプル

Download sample

このサンプルでは、Windows フォームでの継承の使用方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. [ビルド] メニューの [ビルド] をクリックして、プロジェクトをコンパイルします。これにより、派生されたフォームを Windows フォーム デザイナで参照できます。

2. F5 キーを押します。

使用例

フォームもクラスであり、.NET Framework の他のクラスと同様です。ただしフォームは、ビジュアルなユーザー インターフェイス要素を表示できるという点が異なります。この型を継承して使用するには、System.Windows.Forms.Form の派生クラスをさらに派生したクラスを作成します。

`MainForm`、`ControlSamples`、および `MySamples` という 3 つのフォームは、`BaseForm` から派生します。`BaseForm` クラスは、Label コントロール、Button コントロール、およびフォームの外観に影響する他のプロパティを備えています。派生したフォームでは、中央の空いた領域に、新しい **My** 機能の使用例か、または新しい Windows フォーム コントロールの例を配置しています。このサンプルのスタートアップ オブジェクトは `MainForm` で、`ControlSamples` および `MySamples` を起動します。

参照

処理手順

方法 : Windows フォームにコントロールを固定する

関連項目

Form

その他の技術情報

Windows フォームのビジュアルの継承

Web ブラウザのサンプル

Download sample

このサンプルでは、簡単な Web ブラウザを作成します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルは、WebBrowser コントロールを基盤としています。以下の機能が追加されています。

- **移動。** ToolStrip を使用した、[戻る]、[進む]、[更新]、[ホーム]、および [移動] の各オプションを備えています。
- **共通のサイト。ToolStrip** と ToolStripButton コントロールを使用して、特定のサイトに移動します。2 番目のフォームを使用して、サイトを一覧表示します。サイトの一覧は、XmlDocument クラスを使用して XML ファイルで保持します。

参照

関連項目

WebBrowser
ToolStrip
XmlDocument

Visual Basic 言語のサンプル

これらのサンプルでは、Visual Basic の構文と **My** 機能の例を示します。

このセクションの内容

API 呼び出しのサンプル

Win32 API 関数を呼び出して、プロセスとウィンドウを列挙および制御し、システム設定を照会および変更する例を示します。

配列のサンプル

Integer 値と **Object** 値の配列を作成し、配列の並べ替えと検索を実行します。

コールバックのサンプル

コールバック プロシージャを実装するための 4 つのメソッドの例を示します。

カスタム例外のサンプル

アプリケーション固有のエラー用の例外を作成します。

DateTime および TimeSpan のサンプル

DateTime メソッドと TimeSpan メソッドの動作の例を示します。

イベント アクセサのサンプル

イベント アクセサを作成する方法と使用する方法を示します。イベント アクセサを使用すると、イベントの実装をカスタマイズできます。

例外処理のサンプル

アプリケーションの例外処理のいくつかの方法を示します。

ファイル システムのサンプル

My.Computer.FileSystem オブジェクト の例を示します。

ジェネリックのサンプル

5 種類のジェネリック コレクションを作成します。

オブジェクト指向プログラミングのサンプル

オブジェクト指向プログラミングで利用できる手法の例を示します。

演算子のオーバーロードのサンプル (Visual Basic)

通貨を表すクラスを作成し、そのクラス用の演算子を実装します。

オーバーロードとオーバーライドのキーワードのサンプル

Overloads と Overrides の例を示します。

文字列の書式設定のサンプル

書式設定のオプションをユーザーに指定させ、その結果を表示します。

文字列メソッドのサンプル

いくつかの String メソッドに対するパラメータをユーザーに指定させ、その結果を表示します。

一時ファイルのサンプル

一時ファイルを作成および削除します。

Visual Basic 2005 の言語機能のサンプル

Visual Basic 言語の新機能を取り上げます。

XML コメントのサンプル

Visual Basic の XML コメント機能の例を示します。

参照

Visual Basic でのアプリケーションの開発

プログラムの構造、データ アクセス、および Visual Basic エディタについての情報を提供します。

Visual Basic のプログラミング ガイド

Visual Basic 言語の要素を説明します。

関連するセクション

Visual Basic アプリケーションのサンプル

これらのサンプルでは、プロジェクト、ユーザー、およびアセンブリの各タスクの例を示します。

データのサンプル

これらのサンプルでは、データ アクセスの例を示します。

Visual Basic Windows フォームのサンプル

これらのサンプルでは、Windows アプリケーションの例を示します。

Visual Basic のサーバー コンポーネントのサンプル

これらのサンプルでは、Windows オペレーティング システムのコンポーネントとやり取りするアプリケーションの例を示します。

Visual Basic のセキュリティのサンプル

これらのサンプルでは、セキュリティ タスクの例を示します。

Visual Basic のサンプル アプリケーション

このページは、このドキュメントに含まれているすべての Visual Basic サンプルへの出発点です。

API 呼び出しのサンプル

Download sample

このサンプルでは、Win32 API 関数を呼び出して、プロセスとウィンドウを列挙および制御し、システム設定を照会および変更する例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

`Win32API` クラスは、Win32API.vb という別個のファイルに定義されています。Declare ステートメントを使用した各 API 宣言は、`Win32API` クラスの **Shared** メンバとして利用できます。共有メンバは、クラス名を使用して直接呼び出すことができます。使用のためにクラスのインスタンスを作成する必要はありません。

メイン フォームには、4 つのタブ ページを持つタブ コントロールがあります。各タブ ページでは、Win32 API 呼び出しを使用した以下の処理の例を示します。

- アクティブなプロセス　Windows で実行中の全プロセスの一覧をリスト ビューに読み込みます。デリゲートを利用して Win32 コールバック関数 **EnumWindows** を使用する方法を示すのが主眼です。
- アクティブなウィンドウ　Win32 API を使用して、コールバック関数 **EnumWindows** の結果をフィルタ処理することにより、ListBox コントロールにすべてのアクティブなウィンドウの一覧を読み込みます。この例では、**EnumWindows** の定義には Declare ステートメントではなく DllImportAttribute クラスを使用しています。これにより、2 つのメソッドの使用方法の違いを示します。
- ウィンドウの表示　ユーザーがウィンドウのキャプションまたはクラス名を入力すると、そのウィンドウを前面に持ってくることができます。この例は、1 つの Win32 API 関数に対して複数の種類のパラメータをオーバーロードし、型の異なる変数を関数に渡せるようにすることが主眼です。
- API 呼び出し　**GetDiskFreeSpace**、**GetDiskFreeSpaceEx**、**GetDriveType**、**CreateDirectory**、**SwapMouseButton**、**IsPwrHibernateAllowed**、**SetSuspendState**、および **Beep** の各 Win32 API 呼び出しの使用方法を示します。これらは、マネージ クラスの同等の呼び出しからは取得できない情報を返す呼び出しの例です。

参照

関連項目

Declare ステートメント
DllImportAttribute

配列のサンプル

Download sample

このサンプルは、値型の配列および参照型の配列を扱う方法を示しています。値型には **Integer** を使用します。参照型には、プロジェクト内で定義する `Customer` クラスを使用します。 `Customer` クラスは、 `Id` と `Name` という 2 つのパブリック メンバを公開します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

サンプル ドキュメントを表示するには

1. ソリューション エクスプローラで、[Documentation] フォルダをダブルクリックします。
2. Visual Basic Express Edition を使用している場合は、[Documentation] フォルダ内の ReadMe.htm を右クリックします。[ブラウザで表示] をクリックします。
3. Visual Basic の別のバージョンを使用している場合は、[Documentation] フォルダ内の ReadMe.htm をダブルクリックします。

使用例

`Customer` クラスは、IComparable インターフェイスを実装しています。これは、Sort 機能および BinarySearch 機能で必要です。`Customer` オブジェクトは、`Name` または `Id` プロパティで並べ替えることができます。[配列:] オプションを [文字列] から [オブジェクト] に変更すると、`Customer` オブジェクトの配列を Customer の `Name` フィールドまたは `Id` プロパティで並べ替えるかどうかを決定する [Field to use for Sorts] コントロールが有効になります。並べ替えの実行前に `Customer` クラスの共有メソッド `SetCompareKey` を呼び出すと、使用するフィールドを変更できます。

6 つのボタンにより、以下のタスクを実行します。

- 静的配列の作成　{} 構文を使用して配列を作成し、値の配列を読み込みます。配列を読み込んだら、サポート プロシージャ `DisplayArrayData` が配列を列挙し、値をリストに配置します。配列の宣言と初期化の詳細については、「Dim ステートメント (Visual Basic)」を参照してください。

- 並べ替え　静的配列の作成のプロシージャと同じコードを使用してデータを読み込みます。データを読み込んだら、Array Class クラスの共有メソッド Sort を適用し、データを `lstAfter` コントロールに読み込みます。整数などのプリミティブ型および文字列は、自動で比較可能です。 `Customer` クラスを並べ替え可能にするには、**IComparable** インターフェイスを実装します。

- 反転　このプロシージャは、Reverse メソッドを使用して、配列の要素の順序を反転します(**Reverse** は、要素を逆順で並べ替えるのではなく、配列の要素の順序を反転します)。

- バイナリ サーチ　バイナリ サーチを実行するには、配列の要素が並べ替えられている必要があります。データを配列に読み込み、並べ替えを実行してから、**BinarySearch** メソッドを使用してバイナリ サーチを実行します。[Search For] というラベルの付いたテキスト ボックス コントロールに指定した値が検索条件として使用されます。アイテムが見つかった場合、そのインデックス位置が表示されます。アイテムが見つからなかった場合、**BinarySearch** メソッドは、アイテムがあった場合の位置を表すビット補数を返します。

- 動的配列の作成　テキスト ボックス コントロール `txtLength` に設定されている数値を使用して、配列の **ReDim** を実行します。次に、新しい各要素をループし、アイテムの値を要求する入力ボックスを表示します。すべての値を取得したら、その値を表示します。

- 多次元配列の作成　2 列および 3 行から成る 2 次元配列を作成し、リスト ボックスに表示します。

参照

関連項目

Array Class
Array.Sort Method
Array.BinarySearch Method
Dim ステートメント (Visual Basic)
IEnumerable Interface
Icomparable

コールバックのサンプル

Download sample

このサンプルでは、ユーザーが一連の呼び出しを実行でき、実行中にクライアントへのコールバックが行われます。ここでは、インターフェイスとデリゲートの両方を使用しています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

メイン フォームにはボタンが 4 つあり、それぞれ異なるプログラミング技法を使用してコールバック メソッドを実行します。

- インターフェイス　1 つのメソッド `CallbackMethod` を持つ `ICallback` インターフェイスを定義します。サンプルのメイン フォームである `Form1` では、`CallbackMethod` メソッドを定義することにより、このインターフェイスを実装します。`Class1` には、`ICallback` の 1 つのインスタンスへの参照が含まれています。この参照は、`RegisterInterFaceForCallback` メソッドを通じて割り当てられます。`Class1` オブジェクトの `UseInterfaceCallback` メソッドを呼び出すと、`ICallback` オブジェクトの `Callback` メソッドが実行されます。`ICallback` オブジェクトへの参照は、`UnRegisterInterfaceForCallback` メソッドを呼び出すことで削除できます。

- デリゲート　このプロジェクトには、1 つのデリゲート `Delegate1` の定義が含まれています。`Class1` には、`Delegate1` オブジェクトに対する 1 つの参照 `del1` が含まれています。`RegisterDelegateForCallback` メソッドを使用して、デリゲートのインスタンスを `del1` に割り当てることができます。この例では、デリゲートは `Form1` の `CallbackMethod` メソッドです。`Class1` オブジェクトの `UseDelegateCallback` メソッドを呼び出すと、`Form1` の `CallbackMethod` が実行されます。このメソッドは、`Invoke` メソッドを使用して実行されます。デリゲートへの参照は、`Class1` の `UnRegisterDelegateForCallback` メソッドを使用して削除できます。

- 非同期　`Class1` の `UseAsynchDelegateCallback` メソッドは、`del1` デリゲートへの非同期呼び出しを行います。呼び出しには `Invoke` メソッドではなく `BeginInvoke` メソッドを使用します。

- クライアントへの非同期のコールバック　デリゲート オブジェクトへの割り当てなしでコールバック メソッドが呼び出されます。AsyncCallback デリゲートと `BeginInvoke` メソッドを使用します。

参照

関連項目

Delegate ステートメント
Interface ステートメント (Visual Basic)
Delegate

AsyncCallback

カスタム例外のサンプル

Download sample

このソリューションでは、2 種類の高度な例外管理手法の例を示します。カスタム例外と、グローバル例外処理です。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. ソリューション エクスプローラで、Client プロジェクトを右クリックし、[スタートアップ プロジェクトに設定] をクリックします。
2. Ctrl + F5 キーを押します。グローバル例外ハンドラの効果を確認するためには、プログラムはデバッガの外部で実行するのが最適です。

使用例

このサンプルには、Windows フォーム クライアントとクラス ライブラリの 2 つのプロジェクトが含まれています。

クラス ライブラリ (サーバー)

このクラス ライブラリでは、`Customer` クラスと、アプリケーション定義のエラーを公開するためのカスタム例外クラスのセットを定義します。これらの例外クラスは、継承の階層構造によって関連付けられています。基本クラスは `CRMSystemException` で、このクラスは ApplicationException を継承しています。`CustomerException` および `EmployeeException` は、`CRMSystemException` を継承しています。さらに、`CustomerException` を継承した例外として、`CustomerNotFoundException` と `CustomerNotDeletedException` の 2 つがあります。各クラスは、それぞれ異なるレベルの情報および機能を公開しています。

クライアント コード

クライアント アプリケーションは、クラス ライブラリを参照しています。2 つのコマンド ボタンで、`Customer` クラスに対するメソッドを実行し、カスタム例外をキャッチします。コマンド ボタンはもう 1 つあり、処理されていない例外を発生させます。さらに、[Turn on Global Exception Trap] というラベルのチェック ボックスがあり、グローバル例外トラップを有効にします。これを有効にすると、既定の Windows フォーム ハンドラではなく、OnThreadException メソッドで定義されているコードが実行されます。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)
ApplicationException
ThreadException
OnThreadException

その他の技術情報

Visual Basic での構造化例外処理

DateTime および TimeSpan のサンプル

Download sample

このサンプルは、DateTime 構造体と TimeSpan 構造体を示しています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. F5 キーを押します。

2. 適切なテキストボックス コントロールに値を入力し、ページの [Refresh] ボタンをクリックすると、対応するメソッドが実行され、結果がラベル コントロールに表示されます。

サンプル ドキュメントを表示するには

1. ソリューション エクスプローラで、[ドキュメント] フォルダをダブルクリックします。

2. Visual Basic Express Edition を使用している場合は、[ドキュメント] フォルダ内の ReadMe.htm を右クリックします。[ブラウザで表示] を選択します。

3. Visual Basic の別のバージョンを使用している場合は、[Documentation] フォルダ内の ReadMe.htm をダブルクリックします。

使用例

フォームには、いくつかのタブ ページを持つタブ コントロールがあります。各タブ ページは、**DateTime** メンバまたは **TimeSpan** メンバのグループを示しています。各タブ内では、コントロールを使用して、さまざまなメソッドおよびプロパティの入力パラメータを指定できます。

参照

関連項目

DateTime

TimeSpan

イベント アクセサのサンプル

Download sample

このサンプルでは、イベント アクセサを作成および使用する方法を示します。この機能を使用すると、イベントの実装をカスタマイズし、カスタムの **AddHandler**、**RemoveHandler**、および **RaiseEvent** サポートを作成できます。これは、サーバー コントロールが使用するメモリを最適化したり、動作が不適切な呼び出し元やハンドラがあった場合にコントロールの堅牢さを高めたりする目的で使用されることがよくあります。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、`MemoryOptimizedBaseControl` と `ReliabilityOptimizedControl` の 2 つのコントロールを作成します。それぞれ、5 秒おきに `Click` イベントを発生させます。2 つのクラスの違いは、このイベントの実装方法の違いです。

メモリ効率のよいカスタム コントロール `MemoryOptimizedBaseControl` コントロールでは、イベント アクセサを使用して、サーバー コントロールでのイベント作成で消費されるメモリを最適化します。内部の `EventList` オブジェクトが使用されるのは、呼び出し元のコードが `Click` イベントに対するイベント ハンドラを追加した場合のみです。サーバー イベントの発生は、内部タイマを使用してシミュレートしています。

信頼性の高いカスタム コントロール `ReliabilityOptimizedControl` では、イベント アクセサを使用して、信頼性を高めたサーバー コントロールを作成します。サーバー コントロールは、`BeginInvoke` を使用して、`Click` イベントを常に非同期で発生させます。このため、クライアントのイベント ハンドラが互いにブロックすることはありません。つまり、動作が不適切なイベント ハンドラがあっても、他のハンドルの処理はブロックされません。サーバー イベントの発生は、内部タイマを使用してシミュレートしています。

参照

処理手順

方法 : メモリの使用量を節約するイベントを宣言する

方法 : ブロックを回避するイベントを宣言する

関連項目

Event ステートメント

例外処理のサンプル

Download sample

このソリューションでは、**Try...Catch...Finally** を使用した構造化例外処理の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- ブレークポイントを使用するデバッグ モードでアプリケーションを起動するには、F5 キーを押します。ブレークポイントを無視してプログラムを実行するには、Ctrl + F5 キーを押します。

使用例

このコードでは、5 種類の方法によるファイルのオープンの例を示します。テストには 5 つのコマンド ボタンを使用します。各ボタンは、[Text File To Open] というラベルのテキスト ボックスで指定されたファイルを開きます。[No Error Handling] というラベルのボタン以外は、**Try**、**Catch**、および **Finally** の各ブロックを使用して、さまざまなレベルのエラー処理を実行します。

- エラー処理なし　フォームで指定されたファイルを、FileStream クラスを使用して開きます。ファイルが存在しない場合は、例外がスローされます。リリース モードでは、プログラムの実行が中断されます。デバッグ モードでは、[例外処理アシスタント] が表示されます。
- 基本のエラー処理　ファイルを開くための呼び出しは Try...Catch...Finally ステートメント (Visual Basic) でラップされており、これがすべてのエラーをキャッチします。エラー メッセージが表示され、プログラムの実行は継続されます。
- 詳細なエラー処理　複数の **Catch** 句を使用することにより、エラーに対して詳細に対応します。プログラムは、固有のエラーをキャッチすることにより、ファイルが存在しないのか、フォルダが存在しないのか、またはその他の I/O エラーが発生したのかを判断できます。
- カスタム メッセージ　複数の **Catch** 句およびスタック トレースを使用して、例外についての詳細なエラー メッセージを示します。
- Try、Catch、Finally　ファイルが開かれている場合、**Finally** 句を使用して閉じます。

参照

関連項目

Try...Catch...Finally ステートメント (Visual Basic)
Exception
ApplicationException
StackTrace

ファイル システムのサンプル

Download sample

このサンプルは、My.Computer.FileSystem オブジェクトのメソッドを示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. [ビルド] メニューの [FileSystemSample のビルド] をクリックします。
2. F5 キーを押します。

サンプル ドキュメントを表示するには

1. ソリューション エクスプローラで、[ドキュメント] フォルダをダブルクリックします。
2. Visual Basic Express Edition を使用している場合は、[ドキュメント] フォルダ内の ReadMe.htm を右クリックします。[ブラウザで表示] を選択します。
3. Visual Basic の別のバージョンを使用している場合は、[Documentation] フォルダ内の ReadMe.htm をダブルクリックします。

使用例

このサンプルは、My.Computer.FileSystem オブジェクトのファイル システム メソッドを実行するためのインターフェイスです。一方の側の TreeView コントロールで、実行するメソッドの種類を選択します。メソッドを選択すると、フォームにカスタム コントロールが表示され、そのメソッドのパラメータを入力でき、メソッドを実行できます。したがって、このプロジェクトには、デモンストレーションする各メソッドに対して、それぞれ別個のユーザー コントロールが用意されています。

参照

関連項目

My.Computer.FileSystem オブジェクト

ジェネリックのサンプル

Download sample

このサンプルでは、さまざまなジェネリック コレクションを作成および操作する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは次の例を示します。

- Queue、List、Stack、Dictionary、および SortedList の各コレクションを System.Collections.Generic 名前空間から使用します。
- 各コレクション型について、コレクションの内容を並べ替えるメソッドを使用します。
- データ構造の項目を追加および削除します。

このサンプルにはフォームが 1 つあります。起動時には、2 組の RadioButton コントロール、4 つの Button コントロール、および 2 つの ListBox コントロールがこのフォームに表示されます。処理元の **ListBox** コントロールには、格納されているデータが表示されます。2 組の **RadioButton** グループでは、データの型とコレクションの型を制御します。各ボタンでは、消去、読み込み、並べ替え、および反転の各タスクを実行します。コレクションの処理結果は、ターゲットの **ListBox** コントロールに表示されます。各コレクション型に対応するコードは #Region ディレクティブ 内に整理されています。

参照

関連項目

System.Collections.Generic

オブジェクト指向プログラミングのサンプル

Download sample

このソリューションでは、Visual Basic のオブジェクト指向機能のいくつかの例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

プロジェクトには、いくつかのクラス定義が含まれています。`Customer` は基本クラスで、3 つのプロパティのみで構成されています。`Customer` クラスから、4 つのクラスが派生されています。それぞれ、`Customer` のプロパティを継承していますが、カテゴリの異なるクラス メンバも追加されています。

- `CustomerPropertySyntax` は `Customer` クラスに変更を加えたもので、構築時に `AccountNumber` プロパティを設定します。`AccountNumber` は、構築後の変更はできません。
- `CustomerWithConstructor` は `Customer` に変更を加えたもので、インスタンス化のときに 3 つのプロパティすべてを設定するコンストラクタを含みます。
- `CustomerWithParameterizedProperty` は `Customer` をカスタマイズしたもので、パラメータを受け取るプロパティを追加しています。
- `CustomerWithSharedMembers` は、`Customer` クラスに共有プロパティと共有メソッドを追加しています。

参照

関連項目

Class ステートメント (Visual Basic)
Property ステートメント
ReadOnly (Visual Basic)
Shared (Visual Basic)

その他の技術情報

クラスについて

演算子のオーバーロードのサンプル (Visual Basic)

Download sample

このソリューションでは、米ドルとユーロを題材として、オーバーロードされた演算子の概念と使用法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルは、ユーロを表す `EURO` と、米ドルを表す `USD` という 2 つのクラスで構成されています。`Form1` は、2 つのクラス用のテスト アプリケーションです。ユーザーは、`USD` にさまざまな値を入力し、演算子を実行して、`USD` を `EURO` に変換できます。

以下のプログラミング技法が使用されています。

- **オーバーロードされた演算子。** `USD` クラスでオーバーロードされている演算子は、コンストラクタ、+、-、*、および / です。
- **演算子の縮小と拡大。** `EURO` クラスと `USD` クラスは、**Double** との間で拡大変換と縮小変換を実行します。

参照

概念

拡大変換と縮小変換

オーバーロードとオーバーライドのキーワードのサンプル

Download sample

このサンプルでは、オーバーロードやオーバーライドなどの Visual Basic の言語機能を使用して派生クラスを拡張する方法の例を示します。また、**Public**、**Private**、および **Protected** を使用して、クラスのメンバに対するさまざまなレベルのアクセスを実現する方法も示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このアプリケーションは、輸送手段登録システムをシミュレートしたもので、自動車、ボート、および自転車をサポートしています。

このアプリケーションでは、`Vehicle` という基本クラスを使用し、そこから `Automobile`、`Boat`、および `Bicycle` の各クラスを派生しています。それぞれの派生クラスでは、基本クラスを何らかの方法で拡張しています。たとえば、基本クラスのメソッドをオーバーライドする、独自の新しいメソッドまたはプロパティを実装する、基本クラスのメンバを置換 (シャドウ) するなどです。また、`DepartmentOfMotorVehicles` という **Friend** クラスもあります。このクラスは、データベースに対するデータの読み込みと書き込みをシミュレートします。

このアプリケーションでは、クラスおよびそのメンバでの以下のステートメントおよび修飾子の使用例を示します。

キーワード	使用方法
Inherits ステートメント	`Automobile`、 `Boat`、および `Bicycle` で使用しており、 `Vehicle` が基本クラスであることを示します。
NotInheritable	`DepartmentOfMotorVehicles` クラスで使用しています。このクラスは基本クラスになれません。
MustInherit	`Vehicle` クラスで使用しています。 `Vehicle` を派生したクラスでないとインスタンス化できません。
Overloads	`Boat` クラスの `Register` メソッドはオーバーロードされており、最大乗客数が追加されています。
Overridable	`Vehicle` クラスの `ID` プロパティで使用しています。基本クラスで実装されていますが、派生クラスでも別のバージョンを実装できます。
Overrides	3 つの派生クラスすべてにおいて、`CurrentValue` プロパティに使用されており、基本クラスの既定の実装を置き換えます。
MustOverride	`Vehicle` クラスのメンバ (`ComputeRegistrationFee` および `Salary`) で使用しています。これらのメンバは `Vehicle` では実装されておらず、`MustInherit` と指定されていない派生クラスで実装する必要があります。

Shadows	`Bicycle` クラスの `Register` メソッドで使用しています。
Public (Visual Basic)	クラスおよびそのメンバで使用しており、クラスまたはメンバをクライアント アプリケーションが利用できることを示します。
Protected (Visual Basic)	`Vehicle` クラスのフィールドで使用しています。これらのフィールドは、`Vehicle` クラスおよび派生クラスからはアクセスできますが、クライアント アプリケーションからはアクセスできません。
Friend (Visual Basic)	`DepartmentOfMotorVehicles` クラスで使用しています。このクラスはアセンブリで使用されますが、クライアント アプリケーションではアクセスできません。
Private (Visual Basic)	クラスのフィールドで使用しています。これらのフィールドはクライアント アプリケーションではアクセスできません。
Shared (Visual Basic)	`DepartmentOfMotorVehicles` クラスのメソッドで使用しています。これにより、`DepartmentOfMotorVehicles` クラスを関数ライブラリのように利用できます。

参照

その他の技術情報

Visual Basic におけるオブジェクト指向プログラミング

文字列の書式設定のサンプル

Download sample

このサンプル アプリケーションでは、標準の書式指定コードおよびカルチャ固有のカスタムの書式指定コードをさまざまに使用して、数値、日時、および列挙値を文字列として表示する方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

以下の書式設定オプションの例を示します。

- 標準およびカスタムの書式コードを使用して文字列として書式設定された数値。書式の変更は CultureInfo の選択に基づきます。
- 標準およびカスタムの書式コードを使用して文字列として書式設定された日付/時刻の値。書式の変更は **CultureInfo** の選択に基づきます。
- 標準の書式コードを使用して文字列として書式設定された列挙値。

参照

関連項目

CultureInfo Class
IFormatProvider Interface
IFormattable Interface
Object.ToString Method
Int32.ToString Method
DateTime.ToString Method
Enum.ToString Method

概念

カルチャ別の書式設定

文字列メソッドのサンプル

Download sample

このサンプルは、文字列操作のためのいくつかのメソッドを示しています。このサンプルの目的は、関数の呼び出し方を示すことではなく、関数の動作を示すことです。メイン フォームには、いくつかのタブ ページを持つ TabControl があります。各タブ ページは、それぞれが String プログラミング機能の 1 つに対応しています。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

サンプル ドキュメントを表示するには

1. ソリューション エクスプローラで、[ドキュメント] フォルダをダブルクリックします。
2. Visual Basic Express Edition を使用している場合は、[ドキュメント] フォルダ内の ReadMe.htm を右クリックします。[ブラウザで表示] を選択します。
3. Visual Basic の別のバージョンを使用している場合は、[Documentation] フォルダ内の ReadMe.htm をダブルクリックします。

使用例

メイン フォームには、3 つのタブ ページを持つ **TabControl** があり、**String** のメンバ メソッド、**String** の共有メソッド、および StringWriter のメソッドに対応しています。ユーザーは、各タブ ページで文字列値を入力でき、ボタンをクリックすると **String** のメソッドを実行できます。基になっている設計では、`Method` クラスと `Parameter` クラスが使用されています。`Method` クラスの各インスタンスは、**String** のそれぞれ異なるメソッドを表します。この設計により、フォームで入力された値を **String** の適切なメソッドに渡しやすくなっています。

メソッド	説明
System.String.Insert(System.Int32,System.String) System.String.Remove	これらのメソッドは、新しい **String** オブジェクトを作成して返します。これらのメソッドにはオーバーロードされたものが多くあり、1 つ、2 つ、または 3 つのパラメータを受け取ります。コードは、フォームの入力フィールドのいくつかを無視する場合があります。
System.String.IndexOf System.String.StartsWith System.String.EndsWith	これらのメソッドは、既存の文字列についての情報を返しますが、**String** オブジェクトの作成や変更は行いません。

System.String.Format System.String.Join	これらのメソッドの多くは、タスクの完了のために 2 つの **String** を必要としたり、新しい文字列を作成したりするため、**Shared** メソッドとして実装されています。
System.Text.StringBuilder.ToString	StringBuilder クラスでは、文字列内の文字を操作できます。**ToString** メソッドは、**StringBuilder** オブジェクトに含まれているテキストを取得します。
System.IO.StringWriter.Write System.IO.TextWriter.WriteLine System.IO.StringWriter.ToString	**StringWriter** クラスは、出力文字列にテキストを追加する必要があるときに利用できます。**StringWriter** クラスは、ファイルに書き込むのと同じようにテキストを書き込むことのできる内部バッファです。**Write** メソッドおよび **WriteLine** メソッドで、バッファにテキストを追加します。**ToString** メソッドは、**StringWriter**オブジェクトに含まれているテキストを取得します。

String クラスのメソッドを一覧表示しているボタンは、実際には RadioButton コントロールです。Appearance プロパティを Button に設定することでボタンの外観になります。コントロールの見た目はボタンですが、クリックすると選択されたままになります。

String クラスのメソッドを選択するために使用する各ボタンからは、同一のイベント ハンドラ `HandleCheckedChanged` が呼び出されます。このプロシージャでは、多数の Handles 句を使用しています。プロシージャの内部では、**If...Then** ステートメントと sender パラメータを使用して、どのボタンが選択されたかを判断し、適切な処理を行います。

タブ コントロールの上部にコントロールを常に表示させておく方法はないため、コントロールのグループの単一のインスタンスを各ページに表示します。このサンプルでは、この機能を実現するために、タブ コントロールのページが選択されると、次のようにして、"共通" のコントロールすべてが格納されている Panel コントロールの Parent プロパティを、選択されたページに設定します。

```
pnlDemo.Parent = tabStringDemo.SelectedTab
```

ブレークポイントをトリガして **StringBuilder** と **StringWriter** のコードをたどれるようにするために、このサンプルは System.Diagnostics.Debugger.Break メソッドを使用します。このメソッドは、[Step through code] というラベルの付けられた CheckBox コントロールがオンにされたときに呼び出されます。

参照

関連項目

Handles
String
StringBuilder
StringWriter
System.Diagnostics.Debugger.Break

その他の技術情報

TabControl コントロール (Windows フォーム)

一時ファイルのサンプル

Download sample

このサンプルでは、一時ファイルを作成および使用する方法と、ユーザーのコンピュータの \Temp ディレクトリを見つける方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このチュートリアルでは、以下のタスクの例を示します。

- ユーザーのコンピュータの \Temp ディレクトリを見つけます。
- System.IO.Path.GetTempFileName メソッドを使用して、一時ファイルを作成します。
- 一時ファイルの属性を Temporary に設定して、パフォーマンスを向上させます。
- 一時ファイルを削除します。

参照

関連項目

Path.GetTempFileName Method
Path Class
StreamReader Class
StreamWriter Class
System.IO Namespace

Visual Basic 2005 の言語機能のサンプル

Download sample

このサンプルでは、ジェネリック、演算子のオーバーロード、**Using** キーワードなど、Visual Basic の新しい言語機能の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、以下の言語機能の例を示します。

- **演算子のオーバーロード** `ValidatedString` クラスは、カスタム検証を持つ文字列を表します。`&` 演算子は、2 つの `ValidatedString` オブジェクトの連結を返します。
- **ジェネリック** `Pair` クラスは 2 つの値を保持します。`Pair` クラスのインスタンスを宣言するときには、2 つの値の型を指定します。`Match` メソッドは、ペアの型および値が同じかどうかを調べます。
- **Using** Web クライアントからの要求で開かれるストリーム リソースの参照で Using ステートメント (Visual Basic) を使用します。ストリームのインスタンスは **Using** ブロックの末尾で自動的に破棄されます。
- **TryCast および IsNot** TryCast は、オブジェクトを特定の型にキャストし、変換された値を返します。キャストに失敗した場合は **Nothing** を返します。**IsNot** 演算子を使用して、返った結果が **Nothing** かどうかを調べます。

参照

関連項目

Operator ステートメント

Using ステートメント (Visual Basic)

概念

Visual Basic におけるジェネリック型

その他の技術情報

Visual Basic 言語のサンプル

XML コメントのサンプル

Download sample

このサンプルでは、Visual Basic の XML コメント機能の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このサンプルでは、XML コメントの作成、エディタでの表示、およびプログラムでのアクセスの例を示します。

`Automobile` クラスでは、次の型要素を定義します。これらはそれぞれ、XML コメントの異なる側面の例を示します。

- **型** `Automobile` および `EngineOption`
- **プロパティ** `Model`
- **メソッド** コンストラクタ、`Upgrade`、`CalculateMaxTowing`、`WillItTow`
- **イベント** `NeedsTuneUp`
- **デリゲート** `RotateTires`、`ReplaceSparkPlugs`
- **フィールド** `car`、`modelValue`

[Create Car] ボタンの `Click` イベント ハンドラでは、`Automobile` のいくつかのメンバを呼び出します。コード エディタで IntelliSense を使用するときには、XML コメントがツールヒントとして表示されます。

[List Summaries] ボタンの `Click` イベント ハンドラでは、XmlDocument クラスを使用して、XML コメントが含まれている XML ファイルを照会します。この XML ファイルは、プロジェクトをコンパイルしたときに作成され、プロジェクト アセンブリと共に格納されます。このファイルの位置と名前は、プロジェクト デザイナの [コンパイル] タブで指定できます。

参照

処理手順

方法 : Visual Basic で XML ドキュメントを作成する

Visual Basic のサーバー コンポーネントのサンプル

これらのサンプルでは、コンポーネントを使用した一般的なタスクの例を示します。

このセクションの内容

COM ポートのサンプル

通信ポートを制御する方法の例を示します。

イベント ログのサンプル

イベント ログを作成および削除する方法と、システム イベント ログおよびカスタム イベント ログに対する読み込みおよび書き込みの方法の例を示します。

ファイル通知のサンプル

FileSystemWatcher クラスを使用して、ファイルが作成、削除、変更、または名前変更されたときに応答します。

プロセス管理のサンプル

System.Diagnostics 名前空間のオブジェクトを使用して、実行中のプロセスについての情報を集めます。

メッセージ キューのサンプル

メッセージ キューのメッセージを Windows フォーム アプリケーションから送受信する方法を示します。

パフォーマンス カウンタのサンプル

パフォーマンス カウンタに対して読み込みと書き込みを行う方法を示します。

プロセス制御のサンプル

System.Diagnostics 名前空間の Process クラスを使用して、プロセスを起動および監視する方法の例を示します。

システム イベントへの応答のサンプル

SystemEvents クラスを使用します。このクラスを使用するアプリケーションは、システム アクティビティによって発生したイベントに反応できます。

サービス マネージャのサンプル

Windows サービスを制御する方法を示します。

WMI のサンプル

System.Management 名前空間のオブジェクトを使用して、WMI (Windows Management Instrumentation) にアクセスします。

参照

Process

このクラスは、プロセスの開始および停止に使用します。

FileSystemWatcher

このクラスは、ファイルが変更されたときにイベントを発生させます。

SystemEvents

このクラスは、ディスプレイ設定が変更されたときにイベントを発生させます。

関連するセクション

Visual Basic アプリケーションのサンプル

これらのサンプルでは、プロジェクト、ユーザー、およびアセンブリの各タスクの例を示します。

データのサンプル

これらのサンプルでは、データ アクセスの例を示します。

Visual Basic Windows フォームのサンプル

これらのサンプルでは、Windows フォーム アプリケーションの例を示します。

Visual Basic 言語のサンプル

これらのサンプルでは、Visual Basic 言語の概念を示します。

Visual Basic のセキュリティのサンプル

これらのサンプルでは、セキュリティ タスクの例を示します。

Visual Basic のサンプル アプリケーション

このページは、このドキュメントに含まれているすべての Visual Basic サンプルへの出発点です。

ディレクトリ サービスのサンプル

Active Directory へのアクセスと管理を行うサンプルが含まれており、一般的なディレクトリ操作の実行方法を示します。

.NET Framework アプリケーション用のサーバー ベースのコンポーネント

.NET Framework で用意されているサーバー ベースのコンポーネント (MessageQueue、EventLog など) について説明します。

COM ポートのサンプル

Download sample

このサンプルでは、通信ポートを制御する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

必要条件

このアプリケーションでは、いずれかの COM ポートにモデムが取り付けられている必要があります。

使用例

- 利用できる COM ポートを判断する方法
- COM ポートを使用してモデムと通信する方法
- Win32 API 呼び出しを使用して COM ポートとの通信を制御する方法

参照

処理手順

方法 : Visual Basic で、シリアル ポートに接続されているモデムをダイヤルする

その他の技術情報

コンピュータのポートへのアクセス

イベント ログのサンプル

Download sample

このサンプルでは、イベント ログを作成および削除する方法と、システム イベント ログおよびカスタム イベント ログに対する読み込みおよび書き込みの方法の例を示します。

Visual Basic には、ログ記録のしくみが 2 種類用意されています。

- このサンプルで使用している EventLog コンポーネントでは、オペレーティング システムのイベント ログにアクセスできます。コンポーネントでは、ログの追加と削除、イベント ログ ソースの追加と削除、メッセージの書き込み、およびメッセージの削除を行うことができます。
- **My.Application.Log** オブジェクトでは、いくつかの **EventLog** 機能にアクセスすることもできます。加えて、**My.Application.Log** は、テキスト ファイルにメッセージを書き込むことのできる機能も持ちます。詳細については、「ログ記録のサンプル」および「アプリケーションからの情報のログ記録」を参照してください。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

  イベント ログをサポートしていないオペレーティング システムもあります。詳細については、「**EventLog**」を参照してください。

使用例

メイン フォームである `Form1` には、3 つの Button コントロールがあり、イベント ログの読み込み、書き込み、および作成と削除を実行します。それぞれ、実行する操作に固有の情報を指定するための新しいフォームが開きます。

`WriteForm` フォームは、イベント ログ エントリのテキスト、ID、および種類を指定するためのフォームで、WriteEntry メソッドを使用してアプリケーション イベント ログにエントリを書き込みます。

`ReadForm` フォームは、コンピュータのイベント ログの名前の一覧を ListBox コントロールに設定します。この一覧の取得には GetEventLogs メソッドを使用しており、各ログの LogDisplayName プロパティの値を表示しています。選択したログの最新の 10 エントリを、Entries プロパティを使用して取得し、RichTextBox コントロールに表示します。

`CreateDeleteForm` フォームは CreateEventSource メソッドと Delete メソッドを呼び出します。SourceExists メソッドを使用して、イベント ログとソースが既に存在しないかどうかを作成前に検査します。Exists メソッドを使用して、イベント ログが存在することを削除前に検査します。

参照

関連項目

EventLog
System.Diagnostics

概念

EventLog コンポーネントの概要

アプリケーションからの情報のログ記録

その他の技術情報

ログ記録のサンプル

ファイル通知のサンプル

Download sample

このサンプルでは、FileSystemWatcher クラスを使用して、ファイルが作成、削除、変更、または名前変更されたときに応答します。

サンプル フォームでは、**FileSystemWatcher** クラスの以下のプロパティを設定できます。

- Path　このオブジェクトで "監視" するファイル システム パスです。
- Filter　監視するファイルの種類です。既定値は *.* です。
- NotifyFilter　監視する変更の種類です。このプロパティには、0 または System.IO.NotifyFilters 列挙値を組み合わせて指定できます。既定では、**FileSystemWatcher** は FileName、DirectoryName、および LastWrite の各変更を検出します。
- IncludeSubdirectories　選択したパスの下位ディレクトリも監視対象に含めるかどうかを示します。
- EnableRaisingEvents　**true** の場合、オブジェクトは、ファイル システムの変更を監視し、変更されたときにイベントを発生させます。基本的には、これは "オン/オフ スイッチ" です。このデモでは、このプロパティは、トグル ボタンとして表示されている CheckBox コントロールを使用して制御します。

セキュリティに関するメモ　このコード例は、概念を説明するために作成されました。安全なコーディング方法を説明するためのものではないため、実際のアプリケーションや Web サイトには使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. F5 キーを押します。
2. 右側のコントロールを使用して、監視するファイルを変更します。
3. Windows エクスプローラを使用して、監視しているファイルに変更を加えます。ファイル変更の実行方法は問いません。

使用例

フォーム上のコントロールでは、ファイルを作成、削除、変更、および名前変更して、イベントを生成させることができます。フォームのコンポーネント トレイには、ツールボックスの [コンポーネント] タブにある **FileSystemWatcher** コンポーネントが含まれています。このコンポーネントを使用すると、デザイン時に [プロパティ] ウィンドウを使用してプロパティを簡単に設定できます。

フォームの [作成]、[名前の変更]、[変更]、および [Delete Sample File] の各ボタンを使用して、**FileSystemWatcher** のイベントを生成します。これらのボタンでは、選択したフォルダに一時ファイルが作成されます。処理が済んだ後で、残っているサンプル ファイルをすべて削除するには、[Delete All Sample Files] ボタンを使用します。このサンプルでは、**FileSystemWatcher** のイベントのうち、次の 5 つを処理します。

- Created、Changed、および Deleted　これら 3 つのイベントは、すべて同じイベント ハンドラ `HandleChangedCreatedDeleted` で処理されます。イベントの詳細はフォームに表示されます。
- Renamed　古い名前と新しい名前が表示されます。
- Error　例外メッセージが表示されます。

参照

関連項目

FileSystemWatcher

NotifyFilters

プロセス管理のサンプル

Download sample

このサンプルでは、System.Diagnostics 名前空間のオブジェクトを使用して、実行中のプロセスについての情報を集めます。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. F5 キーを押します。

2. 詳細およびスレッド データを読み込む対象のプロセスをクリックします。メニューを使用して、特定のプロセスが読み込んでいるモジュールの一覧を表示します。プログラムの実行中に F5 キーを押すと、プロセスの一覧が更新されます。

使用例

現在のコンピュータのプロセスは、GetProcesses メソッドを呼び出すことによって ListView コントロールに表示されます。プロセスを選択すると、そのプロセスについての詳細が別の **ListView** コントロールに表示されます。プロセスの詳細の取得には、そのプロセスを表す Process インスタンスが使用されます。Threads プロパティを使用してプロセスのスレッドが列挙され、別の **ListView** コントロールに表示されます。各スレッドは ProcessThread インスタンスにより表されます。各プロセスのモジュールを調べるには、プロセスを選択し、メイン メニューまたはコンテキスト メニューの [モジュール] をクリックします。別のフォームに、プロセスが読み込んでいるモジュールが表示され、コードが読み込まれているファイル名も示されます。

以下の 3 つのプロセスについては、完全な情報は提供されません。

- `_Total` これはコードで追加されます。プロセスを表すものではありません。実行時間の概要とプロセッサ使用率の提示に使用されます。
- `Idle` プロセスとして返されますが、データを返すことのできるプロセスではありません。
- `System` このプロセスはモジュール情報は公開しません。

場合によっては、プロセスに関する情報の一部にアクセスできないことがあります。その場合には、メッセージ ボックスが表示されます。

参照

関連項目

Process
System.Diagnostics
ProcessThread
Threads

メッセージ キューのサンプル

Download sample

このサンプルでは、メッセージ キューのメッセージを Windows フォーム アプリケーションから送受信する方法を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. MSMQListener ソリューション ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. F5 キーを押して MSMQListener プロジェクトをコンパイルおよび実行します。
2. Visual Studio の別のインスタンスを開き、MSMQClient ソリューション ファイルを開きます。
3. F5 キーを押して MSMQClient プロジェクトをコンパイルおよび実行します。

使用例

このアプリケーションは、プライベート メッセージ キューに対してメッセージを送信します。メッセージは 1 つずつ受信され、MSMQListener により処理されます。このサンプルは、次の 2 つの構成要素が連係して動作します。

- メッセージを送信する Windows フォーム クライアント。これは単純なアプリケーションで、ユーザーがプライベート メッセージ キューに対してオーダーをポストできます。
- クライアントが送信したメッセージを処理する Windows フォーム リスナ。リスナはまず、現在キューにあるすべてのオーダーを受信します。次に、別のオーダーが届くのを待機し、届いたときには処理します。リスナは、メッセージを受信したときには ReceiveCompleted イベントに応答します。

参照

処理手順

方法 : MessageQueue コンポーネントのインスタンスを作成する

関連項目

MessageQueue

パフォーマンス カウンタのサンプル

Download sample

このサンプルでは、パフォーマンス カウンタに対して読み込みと書き込みを行う方法を示します。これらのカウンタを使用すると、オペレーティング システムおよび .NET Framework アプリケーションのパフォーマンスを追跡できます。システム カウンタとカスタム カウンタの両方を使用します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

1. 書き込みが可能なカスタム パフォーマンス カウンタを作成します。

   a. サーバー エクスプローラを開きます。

   b. [サーバー] ノードを開き、使用しているコンピュータを表すサーバー ノードを開きます。次に、[パフォーマンス カウンタ] ノードを開きます。

   c. [パフォーマンス カウンタ] を右クリックし、[新しいカテゴリの作成] をクリックします。

   d. [パフォーマンス カウンタ ビルダ] ダイアログ ボックス で、[カテゴリ名] に「**パフォーマンス カウンタのサンプル**」と入力し、説明がある場合は [カテゴリの説明] ボックスに入力します。

   e. [新規作成] ボタンをクリックして、新しいパフォーマンス カウンタを追加します。

   f. カウンタの名前と説明を入力します。[種類] には [NumberOfItems32] を選択します。

2. F5 キーを押してプロジェクトを実行します。

3. 先ほど作成したカスタム パフォーマンス カウンタを選択します。[カテゴリ] と [カウンタ] で選択し、ボタンを使用して値をインクリメントまたはデクリメントします。

使用例

デモンストレーションされるタスクには以下が含まれます。

- **カテゴリの一覧表示**　パフォーマンス カウンタのカテゴリを GetCategories メソッドで取得し、ComboBox コントロールに表示します。
- **カウンタの一覧表示**　パフォーマンス カウンタを GetCounters メソッドで取得し、**ComboBox** コントロールに表示します。取得されるのは、選択したカテゴリのカウンタのみです。
- **データの取得**　NextValue メソッドを使用して、選択したカウンタの現在の値を取得します。
- **カスタム カウンタ**　このアプリケーションでは、カスタム カウンタはインクリメントのみが可能です。カスタム カウンタは、**NextValue** メソッドを

呼び出すことのできないカウンタとして定義されています。

参照

関連項目

PerformanceCounter

System.Diagnostics

概念

パフォーマンスしきい値の監視の概要

パフォーマンスしきい値の監視の概要

プロセス制御のサンプル

Download sample

このサンプルでは、System.Diagnostics 名前空間の Process クラスを使用して、プロセスを起動および監視する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

Process クラスを使用して、以下を行います。

- Start メソッドを使用して、コマンド ライン引数を指定して、およびコマンド ライン引数を指定しないで、アプリケーションを実行します。
- GetProcesses メソッドを使用して、実行中のすべてのアプリケーションを一覧表示します。
- Modules プロパティを使用して、読み込まれているモジュールを一覧表示します。
- GetCurrentProcess メソッドを使用して、実行中のアプリケーションに関する情報を取得します。

このサンプルでは、StringBuilder クラスおよびジェネリック クラスの List を使用したプログラミング技法の例も示します。

参照

関連項目

Process
System.Diagnostics
Start
GetProcesses
Modules
GetCurrentProcess
StringBuilder

システム イベントへの応答のサンプル

Download sample

このサンプルでは、SystemEvents クラスを使用します。このクラスを使用するアプリケーションは、システム アクティビティによって発生したイベントに反応できます。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

Microsoft.Win32.SystemEvents クラスにはいくつかのパブリックなイベントがあり、コードでシンクできます。このサンプルでは、**SystemEvents** クラスで利用可能なイベントのうちの 5 つの例を示します。フォームのチェック ボックスをオンにして、イベントに対応するイベント ハンドラを追加します。イベントの種類とその発生方法を次の表に示します。

変更	イベント	発生方法
Screen	DisplaySettingsChanged	画面の解像度を変更します。
時刻	TimeChanged	現在のシステム時刻を変更します。
電源モード	PowerModeChanged	ラップトップ コンピュータで実行中の場合、電源接続のステータスを変更します。
ユーザー設定	UserPreferenceChanged	Windows の任意の設定を変更します。たとえば、デスクトップを右クリックして [画面のプロパティ] ダイアログ ボックスを表示し、設定を変更します。
使用できるフォント	InstalledFontsChanged	フォントをインストールまたは削除します。

参照

関連項目

SystemEvents
DisplaySettingsChanged
TimeChanged
PowerModeChanged
UserPreferenceChanged
InstalledFontsChanged

サービス マネージャのサンプル

Download sample

このサンプルでは、Windows サービスを制御する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

Windows サービスは、ユーザーとの対話なしで、長時間にわたってタスクを実行し続けるプログラムです。このサンプルでは、System.ServiceProcess 名前空間の ServiceController クラスを使用します。このクラスの共有メソッド GetServices では、実行中のサービスのリストを取得できます。**ServiceController** クラスを使用すると、サービスを開始、停止、一時中断、または再開できます。

ユーザー インターフェイスを使用して、サービスを開始、停止、一時中断、または再開します。アプリケーションは、サービスの現在の状態をチェックして、どのボタンを使用可能にするかを判断します。[Actions] メニューの [Relist All Services] コマンドでは、**GetServices** メソッドを呼び出して、ListView コントロールにプロセスを表示します。[Actions] メニューの [Refresh] コマンドでは、**ListView** コントロールに既に一覧表示されているプロセスのステータスを更新します。

参照

関連項目

ServiceController
System.ServiceProcess

その他の技術情報

Windows サービスの監視

WMI のサンプル

Download sample

このサンプルでは、System.Management 名前空間のオブジェクトを使用して、WMI (Windows Management Instrumentation) にアクセスします。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出 ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

TabControl の 2 つのタブ ページを使用して、次の 2 種類の WMI タスクの例を示します。

- クエリ　ManagementObjectSearcher オブジェクトを使用してシステム情報をクエリする方法を示します。クエリするのは、オペレーティング システムの情報 (名前、バージョン、メーカー、コンピュータ名、および Windows ディレクトリ)、コンピュータ システムの情報 (メーカー、モデル、システムの種類、および合計物理メモリ)、プロセッサ名、BIOS バージョン、およびタイム ゾーンです。これらのクエリの作成には、SelectQuery クラスを使用します。

- オブジェクト　ManagementClass クラスおよび EnumerationOptions クラスを使用して、トップレベルの WMI クラスおよびそれらのすべてのデータ管理オブジェクトを列挙する方法を示します。

参照

関連項目

ManagementObject
ManagementObjectSearcher
SelectQuery
ManagementOperationObserver
ManagementClass
EnumerationOptions

Visual Basic のセキュリティのサンプル

これらのサンプルでは、セキュリティ タスクの例を示します。

このセクションの内容

暗号化ハッシュ アルゴリズムのサンプル

ハッシュ アルゴリズムを使用します。

対称アルゴリズムのサンプル

対称アルゴリズムを暗号化に使用します。

参照

HashAlgorithm

このクラスはハッシュ アルゴリズムの計算に使用します。

SymmetricAlgorithm

このクラスはデータの暗号化に使用します。

関連するセクション

ClickOnce によるオンデマンド配置の技術サンプル

System.Deployment.Application 名前空間のクラスを使用した、ClickOnce によるアプリケーションの配置の例を示します。

ClickOnce バックグラウンド更新技術のサンプル

ClickOnce API を使用して、バックグラウンド タスクとして ClickOnce アプリケーションの更新をダウンロードする例を示します。

公開キー暗号方式の技術サンプル

公開キー暗号化方式を使用してメッセージを交換する方法の例を示します。

暗号サービス

暗号化の主要な概念を紹介し、.NET Framework における実装方法について説明します。

安全なコーディングのガイドライン

.NET Framework を使用して、悪意のあるコードや望まない処理が実行されないように、防御機構を構築する方法について説明します。

Visual Basic アプリケーションのサンプル

これらのサンプルでは、プロジェクト、ユーザー、およびアセンブリの各タスクの例を示します。

データのサンプル

これらのサンプルでは、データ アクセスの例を示します。

Visual Basic Windows フォームのサンプル

これらのサンプルでは、Windows フォーム アプリケーションの例を示します。

Visual Basic 言語のサンプル

これらのサンプルでは、Visual Basic 言語の概念を示します。

Visual Basic のサーバー コンポーネントのサンプル

これらのサンプルでは、サーバーベース コンポーネントの例を示します。

Visual Basic のサンプル アプリケーション

このページは、このドキュメントに含まれているすべての Visual Basic サンプルへの出発点です。

暗号化ハッシュ アルゴリズムのサンプル

Download sample

このサンプルでは、ハッシュ関数を使用してデータ整合性を確認する方法の例を示します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

ソリューション エクスプローラでサンプル ファイルを開くには

1. [サンプルのダウンロード] をクリックします。

   [ファイルのダウンロード] メッセージ ボックスが表示されます。

2. [開く] をクリックし、zip フォルダ ウィンドウの左列で、[ファイルをすべて展開] をクリックします。

   抽出ウィザードが開きます。

3. [次へ] をクリックします。ファイルを抽出するディレクトリを必要に応じて変更し、[次へ] をクリックします。

4. [展開されたファイルを表示する] チェック ボックスがオンになっていることを確認して [完了] をクリックします。

5. サンプルの .sln ファイルをダブルクリックします。

   サンプル ソリューションがソリューション エクスプローラに表示されます。場合によっては、ソリューションの位置が信頼されていないという、セキュリティ上の警告が表示されることがあります。[OK] をクリックして続行します。

このサンプルを実行するには

- F5 キーを押します。

使用例

このアプリケーションでは、2 つのハッシュ値を比較します。1 つはファイルのテキストのハッシュ値、もう 1 つは TextBox コントロールのテキストのハッシュ値です。ハッシュ値とは、ひとまとまりのデータを、一意かつ簡潔な数値で表現したものです。テキスト ファイルは、Northwind データベースの `Products` テーブルの XML バージョンです。アプリケーションを起動すると、**My.Resources** に格納されている XML ファイルが取得され、**TextBox** コントロールに読み込まれます。XML ファイルのハッシュ値が算出されます。[Compare!] ボタンをクリックすると、**TextBox** テキストのハッシュ値が算出されます。**TextBox** コントロールのテキストを変更していない場合、ハッシュ値は一致します。ソース ファイルに 1 文字でも変更を加えると、ハッシュ値は変わります。

算出するハッシュの種類は、MD5、SHA1、SHA384 のいずれかを、RadioButton コントロールで決定します。各ハッシュ アルゴリズムに対しては、HashAlgorithm クラスから派生した、それぞれ別個の .NET Framework クラスが存在します。ハッシュ値の算出にはそれらのクラスが使用されます。

参照

関連項目

MD5CryptoServiceProvider
SHA1CryptoServiceProvider
SHA384Managed
StreamReader
WriteXml
ReadXml

概念

暗号化の概要

対称アルゴリズムのサンプル

Download sample

このサンプルでは、Rijndael クラスと TripleDESCryptoServiceProvider クラスを使用して、「暗号化の概要」で解説されている対称 (共有キー) 暗号を実装します。

セキュリティに関するメモ：
このサンプル コードは概念を示す目的で提供されているものです。必ずしも最も安全なコーディング手法に従っているわけではないので、アプリケーションまたは Web サイトでは使用しないでください。Microsoft は、サンプル コードが意図しない目的で使用された場合に、付随的または間接的な損害について責任を負いません。

このサンプルを実行するには

1. このページの [サンプルのダウンロード] ボタンをクリックします。[ファイルのダウンロード] ダイアログ ボックスで、[開く] をクリックします。
2. 使用中のコンピュータにファイルを保存します。
3. Visual Studio で、[ファイル] メニューの [開く] をポイントし、[プロジェクト/ソリューション] をクリックします。
4. ファイルを保存したフォルダを表示し、ソリューション ファイルを選択します。Visual Studio でソリューションが開かれます。
5. F5 キーを押します。

使用例

`SampleCrypto` クラスは、暗号アルゴリズム、salt と初期化ベクタ (IV)、および 1 つの暗号化ファイルを定義する暗号化ルーチンと復号化ルーチンをカプセル化します。コンストラクタはパラメータを 1 つ受け取り、それにより暗号方式の種類 (Rijndael または TripleDES) を決定します。クラス内でこれに対応するフィールドは `crpSym` で、SymmetricAlgorithm 抽象型です。これは、**Rijndael** クラスまたは **TripleDESCryptoServiceProvider** クラスのインスタンスに設定されます。`CreateSaltIVFile` メソッドは、salt 値と IV 値を .dat ファイルに保存します。`Decrypt` メソッドと `Encrypt` メソッドは、`SourceFileName` プロパティで指定されたソース ファイルを処理します。

フォームの読み込み時に、`SampleCrypto` クラスの 1 つのインスタンスが作成されます。フォーム上のコントロールは、`SampleCrypto` クラスのメソッドとプロパティをデモンストレーションします。[Encrypt with Password] コントロールがオンになっている場合、キーは自動的に生成および設定されるのではなく、"salt 処理" されたパスワードから取り出されます。さらに、salt および初期化ベクタ (IV) は、暗号化なしで .dat ファイルに保持されます。この方法を使うと、他人に文書を安全に送ることができます。つまり、暗号化されたドキュメントは安全性の低い伝送路で送信されるが、.dat ファイルおよびパスワードは、非対称 (公開キー) 暗号を使用して安全に転送できます。非対称暗号は、最も安全な形式の暗号化ですが、処理に要する時間ははるかに長くなります。このため、共有キー、salt/IV ファイル、パスワードなどの小さなアイテムに使用するのが普通です。

参照

関連項目

TripleDESCryptoServiceProvider
Rijndael
SymmetricAlgorithm
Salt
IV

概念

.NET Framework の暗号モデル