Microsoft®

Visual Basic 6.0 ユーザーのための Microsoft® Visual Basic® .net™ 移行ガイド

～やっぱりVBが好き～

msdn®
Microsoft® Developer Network

Visual Basic 6.0ユーザーのための Visual Basic .NET移行ガイド

第1章

Visual Basic 6.0から Visual Basic .NETへ

Microsoft® Visual Basic® .net™

Visual Basic .NETを使う

・Visual Studio .NETをインストールして使う

Visual Basic .NETを使うと、はじめはVisual Basic 6.0との違いに驚くはずです。フォームにコントロールを貼り付けるとそれによってコードが生成されたり、フォームにClassというキーワードが付いていたり、そもそも、VBフォームではなくWindowsフォームになっていたり。

そして、さらに使っていくと、実はそんなに変わってはいないことにも気づくでしょう。

たとえば、「ボタンをクリックすると、メッセージが表示される」アプリケーションを作成するためのプロセスは、Visual Basic 6.0もVisual Basic .NETもまったく同じです。

・プロジェクトを作成する(図1)
・フォームにコントロールを貼り付ける(図2)
・コントロールをダブルクリックし、イベントプロシージャに「メッセージを表示する」コードを追加する(図3)
・[開始]ボタンでテスト実行する(図4)

図1：プロジェクトの作成

図2：コントロールを貼り付ける

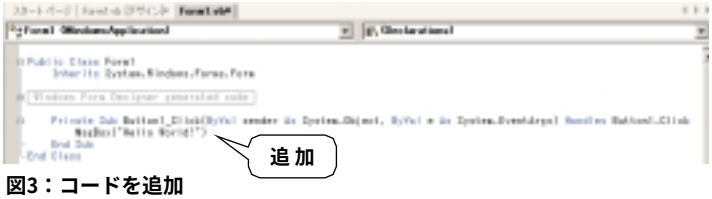


図3：コードを追加

このとき、メッセージを表示するためのMsgBox関数もVisual Basic 6.0と同様に利用することができます（Visual Basic .NETでは、戻り値を利用するかしないかにかかわらず、引数を囲む“(”と“)”が必要です)。

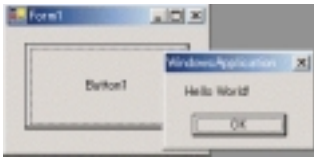


図4：テスト実行結果

```
MsgBox("Hello World!")
```

さらに、Visual Basic .NETでは、.NET Frameworkクラスライブラリを利用することもできます(PlusOne参照)。

```
MessageBox.Show("Hello World!")
```

とにかくVisual Basic .NETを使ってみてください

幸い、Visual Basic .NETはVisual Basic 6.0と共存可能です。今、Visual Basic 6.0の開発環境を利用していても、同じマシンでVisual Basic .NETを利用することができます。

Visual Basic .NETを利用するためには、Visual Studio .NETをインストールします。.NET Framework SDKだけをインストールしてもコンパイルや実行はできますが、ウィザードやデザイナ、デバッガなどのVisualな開発環境を利用したければ、Visual Studio .NETをインストールします。

セットアップを開始すると、まずWindowsのアップグレードが行なわれ、続いてVisual

Studio .NETがインストールされます。

たしかに、Visual Basic 6.0からVisual Basic .NETへの変更は小さなものではありません。しかし、それをVisual Studio .NETやそれが提供するツールがカバーし、移行をサポートしてくれます。Visual Basic .NETの新機能を正しく理解し、準備することで、Visual Basic 6.0ユーザーにとってVisual Basic .NETによる開発は思ったほど大きなハードルではなくなるでしょう。

本書では、

- **なぜVisual Basic .NETへの移行が必要なのか**
- **どのような点が変わるのか**
- **その準備のために今何ができるのか**

について解説します。ぜひ、Visual Basic .NETを実際に使いながら読み進めていくことをお勧めします。

トレーナーをやっている関係で、よく「Visual Basicってかなり変わっちゃうんですよねー」とか「やっぱり、これからはC#ですか？」といった声を耳にします。いつも次のようにお答えします。「思ったほどは変わっていませんよ。それより、すごい進化しているんでぜひVisual Basic .NETを触ってみてください。多分、気に入ってもらえると思いますよ」と。

Plus One　MsgBoxとMessageBox.Showを比較する

MsgBox関数を利用しても、MessageBox.Showメソッドを利用してもメッセージボックスを表示することができます。図5のふたつのコードの実行結果を比較してみてください。

```
MsgBox("HelloWorld!",MsgBoxStyle.OKCancel, "Message")
```


```
MessageBox.Show("HelloWorld!","Message",MessageBoxButtons.OKCancel)
```

図5：同じメッセージボックスが表示される

同じメッセージボックスが表示されます。実際にMsgBox関数は、内部でMessageBox.Showメソッドを呼び出しています。

また、次のコードの実行結果を比較してみて下さい。

```
MsgBox("Hello World!")
MessageBox.Show("Hello World!")
```

タイトルバーに注目してください。MsgBox関数では、第3引数を省略したときプロジェクト名が表示されます（図6の左）。これはVisual Basic 6.0のMsgBox関数と同じです。MsgBox関数では、引数が省略された場合にプロジェクト名を参照するコードが実行されます。これは便利ともいえますが、その分冗長なコードが動いているともいえます。

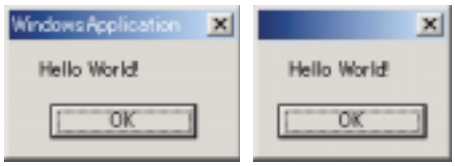


図6：MsgBox("Hello World!")の実行結果（左）とMessageBox.Show("Hello World!")の実行結果（右）

Visual Basic 6.0からVisual Basic .NETへ

- **よりパワフルに再設計された開発ツール**
- **ターゲットは大規模エンタープライズシステム**
- **オブジェクト指向のサポート**
- **Frameworkの採用**
- **ただし、「Visual Basic 6.0との100%の互換性」はない**

Visual Basic 6.0でも、n階層の分散型システムを開発することができました。しかし、Visual Basic .NETは、より大規模で高性能なエンタープライズ向けのシステムやXML Webサービスを、簡単に作成できることを目的に開発されました。

そのため、Visual Basic 6.0に対して単なる機能追加が行なわれただけでなく、製品の再設計が行なわれました。利用できるアーキテクチャ、文法、オブジェクトなど多くの点が変更されています。

まず、Visual Basic .NETではオブジェクト指向が完全にサポートされるようになります。たとえば、

- **コードの継承**
- **オーバーロード**
- **オーバーライド**
- **コンストラクタ／デストラクタ**

が、利用可能になりました。Visual Basic 6.0でも、COMで必須の「インターフェイスの継承」は可能です。しかし、他のオブジェクト指向言語にあるような「コードの継承」がVisual Basicにはない！ といわれていましたが、Visual Basic .NETからはこの機能を利用できます。

そして、なんといってもVisual Basic 6.0からVisual Basic .NETでの最大の変更点は.NET Frameworkの採用です（第2章の「.NET Framework」参照）。

これにより、Windowsフォーム、ASP.NET、ADO.NETなどが利用できます。また、レガシーな機能が排除されたことや、共通言語ランタイム、共通の型システムを使用することにより、C#などの他の.NET対応言語との相互利用性が高まりました。これで、Visual Basicから各種APIを利用するときに発生していた、Visual BasicとVisual C++間の型の違いによる苦労なども解決されます。

このようなさまざまな恩恵の代償として、「Visual Basic 6.0との100%の互換性」はなくなりました。

従来のコードは、そのままではビルドエラーになったり、ランタイムエラーになったりします。Visual Basic 6.0からVisual Basic .NETへの変更点を考慮し、修正が必要です。

ただ、この変更を自動で行なってくれるツールとして、アップグレードウィザード（後述）があります。このツールは、変更された多くの部分を自動的にアップグレードしてくれます。このツールでアップグレードした後に、自動的に変更されなかった部分にのみ修正を加えることで、移行に関する作業を大幅に軽減することができます。

Visual Studio .NET 言語の選択

- **Visual Basic 6.0/ASP（VBScript）/VBAユーザーはVisual Basic .NET**
- **Visual C++ 6.0ユーザーはC#**

.NET Frameworkは、20種類以上もの言語に対応しています。Microsoftだけでも Visual Basic .NET、C#、C++、Java言語、JScript .NETと5つの言語に対応しています。これだけの言語の中で開発者はいったいどの言語を選択すればいいのでしょうか。MicrosoftはC#を推奨しているのでしょうか？
「いいえ、そうではありません！」（Microsoft Visual Studioプロダクトマネージャ談）。

実際に、.NET Frameworkのパワーは、どの言語でも同様に活用することができます。Visual Studio 6.0世代では、Visual Basic 6.0の持つ生産性とVisual C++ 6.0の持つパフォーマンスを組み合わせることにより、最強のアプリケーションを構築することができました。

図7に示した各言語の位置付けからもわかるように、Visual Basic .NETとC#は、Visual Basic 6.0にまさる生産性と、Visual C++ 6.0と肩を並べるパフォーマンスを共に実現しています。もはや、性能論議で各言語を比較する時代は終わったのです。

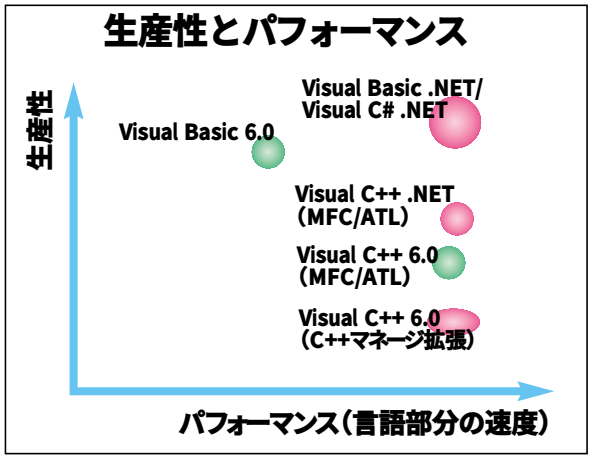


図7：各言語の生産性とパフォーマンス

開発者は、生産性や、パフォーマンスではなく、好みの言語、使い慣れた言語での開発をすることができます。Visual Basic 6.0、ASP（VBScript）、VBAユーザーにとって、Visual Basic .NETとC#、どちらが早く習熟でき、効率よく開発できそうでしょうか？

ここが選択のポイントになります。

実際にVisual Basic .NETとC#をコンパイルすることによって得られる実行コード（MSIL）を比較してみます。
まずは、それぞれの言語で同じことを実行するコードを作成します（リスト1・2）。

```
Sub Proc1()
    Dim x, y, z As Integer
    x = 10
    y = 20
    z = x + y
    Console.WriteLine(z)
End Sub
```

リスト1：Visual Basic .NETサンプルコード

```
void Proc1()
{
    int x, y, z;
    x = 10;
    y = 20;
    z = x + y;
    Console.WriteLine(z);
}
```

リスト2：C#サンプルコード

その後、コンパイル（ビルド）し[注1]、その中のコードをILDASM.EXEで覗いてみます。

```
.maxstack  2
.locals init ([0] int32 x,
              [1] int32 y,
              [2] int32 z)
IL_0000:  ldc.i4.s   10
IL_0002:  stloc.0
IL_0003:  ldc.i4.s   20
IL_0005:  stloc.1
IL_0006:  ldloc.0
IL_0007:  ldloc.1
IL_0008:  add
IL_0009:  stloc.2
IL_000a:  ldloc.2
IL_000b:  call void [mscorlib]System.Console::WriteLine(int32)
IL_0010:  ret
```

リスト3：ILDASM.EXEで覗いてみる

すると、それぞれ異なる言語からコンパイルされた結果、リスト3のような同じコードが生成されることがわかります。

注1）このとき、ビルドオプションが異なると結果が異なることがあります。なお、Visual Basic .NETとVisual C#のデフォルトのビルドオプションは異なるため、デフォルトのままでは同じコードは生成されません

アップグレードウィザードの利用

- **Visual Basic 6.0プロジェクトをVisual Basic .NETプロジェクトにアップグレードする**
  コマンドラインツール
  アップグレードウィザード

Visual Basic 6.0で作成したプロジェクトは、ツールによりアップグレードできます。

コマンドラインから、Vbupgrade.exeの引数にプロジェクトを指定することでアップグレードできます。このとき、オプションを指定することも可能です。

また、 Visual Basic 6.0のプロジェクトをVisual Studio .NETで開くと、自動的にウィザードが起動し（図8)、プロジェクトをアップグレードできます。このとき、ウィザード形式でオプションを指定することができます（図9)。

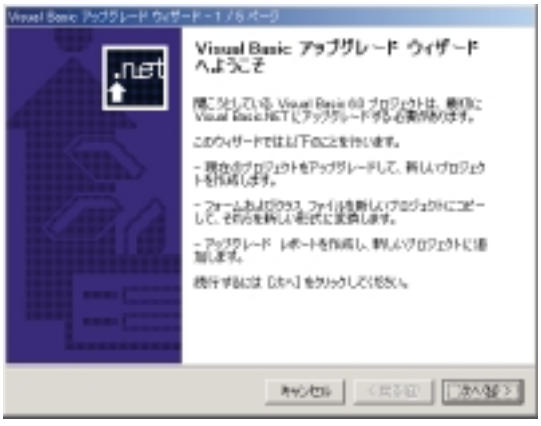


図8：アップグレードウィザード

注意しなくてはいけないことは、アップグレードウィザードがアップグレードしてくれるのは「Visual Basic 6.0のプロジェクトのみ」、つまりVisual Basic 6.0でコンパイル、実行が可能なもののみ、という点です。Visual Basic 5.0以前のプロジェクトをアップグレードする場合は、まず、Visual Basic 6.0のプロジェクトへのアップグレードを行ない、その後、.NETにアップグレードします。具体的には、コントロールはVBXではなくActiveXコントロール（OCX）にアップグレードしておく、Visual Basic 6.0でコンパイル＆実行が可能であることなどがそれにあたります。

アップグレードウィザードを利用すると、次の点が変更されます。

- **Visual Basic 6.0からVisual Basic .NETに言語レベルでアップグレード**
- **フォームやコントロールのアップグレード**
- **アーキテクチャやデータアクセス手法のアップグレード**

一般に、95%が自動アップグレードされ、残りの5%を手動でアップグレードする必要があるといわれていますが、プロジェクトのタイプによって、この数字は異なります。また、あらかじめVisual Basic 6.0でどのようなプロジェクトを作っていたかとい

うことも影響します。この手動によるアップグレードを最小限に抑えるための方法は第3章〜第5章で解説していきます。

図9：ウィザードによるアップグレード

アップグレードウィザードの効果

- コードのアップグレード
- フォームのアップグレード
- その他のアップグレード（リソースファイル、ADOなど）

コードのアップグレード

データ型、構文などは、意味的に同じになるように適切にアップグレードされます。たとえば、VariantはObjectに、整数型はその大きさを保つ違う名前のオブジェクトに変換されます。

また、デフォルトが変更になっている場合なども、Visual Basic 6.0でのデフォルトを明示的に付加することで、元のプログラムの意図を引き継ぎます。たとえば、Visual Basic 6.0では引数になにも指定しないとByRefでしたが、Visual Basic .NETではByValがデフォルトになっています。そこで、Visual Basic 6.0でなにも指定がない場合、Visual Basic .NETにアップグレードするとウィザードがByRefを追加してくれます。

詳細は、第4章を参照してください。

フォームのアップグレード

VBのFormは、Windowsフォームに置き換えられます。VB Formのメソッドやプロパティも、できる限り対応するメソッドやプロパティに置き換えられます。

標準コントロールの多くも、対応する新しいコントロールに置き換えられ、ActiveXコントロールを利用している場合も、必要なラッパークラスや設定が自動的に追加されるため、ほとんどそのまま利用可能です。

詳細は、第5章を参照してください。

その他のアップグレード

リソースファイル（.res）は、国別のXML形式のファイル（.resx）に変更になります（従来のresファイルが使えなくなったというわけではありません）。

Web Classデザイナを利用しているプロジェクトはASP.NETにアップグレードされます（ただし、Visual Studio .NETベータ版ではエラーになり、変更はできません）。

ADOはADO.NETではなく、ADO 2.7にアップグレードされます。ADOのコードやコントロールとバインディングされているデータなども適切にアップグレードされます。さらに、DataEnvironmentで実現していたものも、Visual Basic .NETに変更されます。

詳細は、第3章を参照してください。

Plus One

アップグレードウィザードの本当の活用法、それは、ずばり、

プログラムの移行より、プログラマの移行（!?）

アップグレードウィザードを利用すると、Visual Basic 6.0のプログラムをVisual Basic .NETにアップグレードできます。すでに紹介したように、フォームや、コードをできる限りアップグレードしてくれて、さらに、レポートやコメント（詳細は、「アップグレードレポートとアップグレードコメント」を参照）まで付けて、手で修正すべき部分を指摘してくれます。もちろん、この機能を利用して、プログラムをスムーズにアップグレードすることができます。

アップグレードウィザードのもうひとつの使い方として、これをVisual Basic .NET学習のためのツールとして利用することができます。既存のプログラムをアップグレードウィザードにかけてみると、既存のコードのどの部分が変わらず使えて、どこを変更すべきかが一目瞭然です。また、どのように直せばよいか、あるいは、根本的に変更しなくてはいけないところはどこかを知ることもできます。

そのような観点から、アップグレードウィザードを活用してみてください。さっそく、今お持ちのVisual Basic 6.0プロジェクトをVisual Basic .NETにアップグレードしてみてください。グッとイメージがわき、効果的なVisual Basic .NETの予習ができます。

アップグレードレポートとアップグレードコメント

- **アップグレードレポート**
  - **アップグレードに関する情報の一覧**
  - **重大度レベルとオンラインヘルプへのリンク**
- **アップグレードコメント**
  - **コード内に挿入**
  - **重大度レベルとオンラインヘルプへのリンク**
  - **タスクリストに表示**

Visual Basic 6.0のグラフィック関連の標準コントロール（LineコントロールやShapeコントロール）は、Visual Basic .NETでは使えません。これらのコントロールは、できる限り他のコントロールに置き換えられますが、適切にアップグレードできない場合は、赤色のラベルになって警告されます。

たとえば、Lineコントロールを利用している場合、アップグレードウィザードでアップグレードすると、垂直／水平線は、幅／高さが0のLabelコントロールに置き換わります。また、斜めのLineコントロールは、同じ幅と高さを持つ赤色のラベルに変わります（図10）。

そして、このとき、アップグレードレポート（_UpgradeReport.htm）がプロジェクトに追加されます（図11）。アップグレードレポートには、アップグレードに関する全般的な情報や、アップグレードされたもの、されなかったものとその重大度レベル、さらに、適切にアップグレードするためになにをすればよいのか？　などのアドバイスが提示されます。また、ここに表示されるオンラインヘルプへのリンクをクリックして、より詳細なヘルプを表示することもできます。

これらのサポートにより、スムーズにコードを編集することができます。

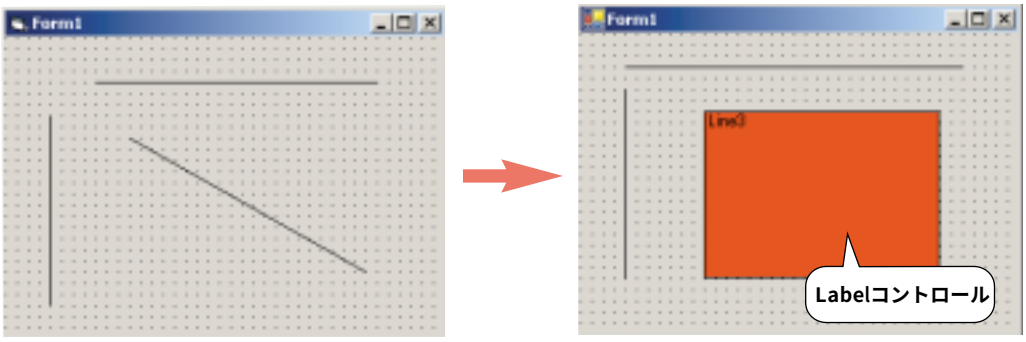


図10：LineコントロールはLabelコントロールに

コード中に、次のようなアップグレードコメントが挿入される場合もあります。コメントには、重大度レベルや、アドバイス、オンラインヘルプへのリンク情報が含まれます。

```
'UPGRADE_ISSUE: LineプロパティLine1.BorderWidthはランタイムでサポートされません。詳細については、ms-help://MS.MSDNVS/vbcon/html/vbup2066.htmをクリックしてください。
```

アップグレードコメントの重大度レベルは4通りあります。以下にそれらコメントを重大度の高い順に示します。「」内は、アップグレードレポート上に表示されるタイトルです。

【UPGRADE_ISSUE】「コンパイルエラー」

・修正が必要であり、このままではコンパイルエラーになるような致命的なコード
（例）LSet x = y

【UPGRADE_TODO】「ToDo」

・仕上げが必要なコード
（例）APIにマーシャリングする属性を追加する必要があるコード

【UPGRADE_WARNING】「実行時の警告」

・必要に応じて、コードを変更するとよいコード
（例）「Dim a(10 to 20) As String」が「Dim a(20) As String」に変更されたとき（このままでは、無駄が生じそうだというとき）

【UPGRADE_NOTE】

・変更が必要かどうかを検討すべきコード（レポートには表示されない）
（例）レガシー機能を利用していたが適切に対応するものにアップグレードできたとき

タスクリストで「コメントを表示する」に設定すると上記のコメントがタスクリストに表示されます。

なお、デザインエラーのように、アップグレードレポートにのみ表示され、コメントとしては追加されないものもあります。

以上のようなさまざまなツールが、Visual Basic 6.0からVisual Basic .NETへのスムーズな移行作業をサポートしてくれます。

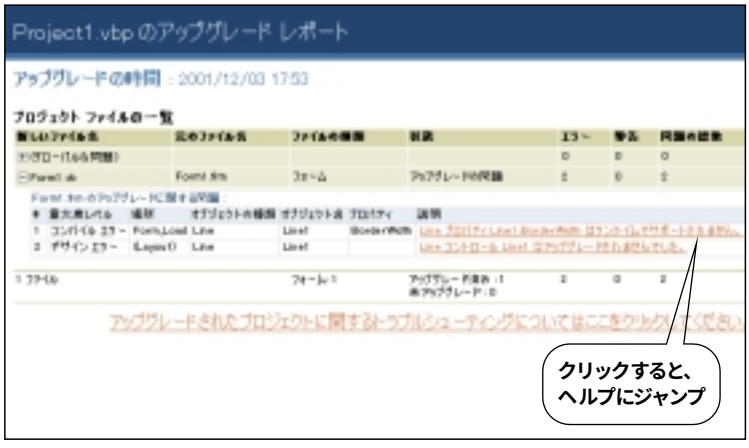


図11：アップグレードレポートがプロジェクトに追加される

第2章

Visual Basic .NET の利点

Microsoft® Visual Basic® .net™

Visual Basic .NETの利点

- エラーの少ない、よりよいコードが書ける
- オブジェクト指向のサポート
- .NET Frameworkへのフルアクセス

Visual Basic .NETの利点を紹介します。

より堅牢なアプリケーションの構築が可能に

エラーが少なく、生産性の高い、よりよいコードが書けるようになります。Visual Basic .NETでは、従来よりも厳密なタイプチェックが行なわれるようになります。これにより、バグの入り込みにくい安定したコードを記述できます。

オブジェクト指向の、より完全なサポート

オブジェクト指向が、より完全にサポートされたことで、オブジェクト指向に基づいて設計されたシステムの実装が容易になります。

.NET Frameworkへのフルアクセス

さらに、Visual Basic .NETは.NET Frameworkへのフルアクセスが可能です。これにより、Windowsフォーム、ASP.NET（Webフォーム、XML Webサービス）、ADO.NETなどの新しい機能を利用することができます。今まではWindows APIを利用しなければ解決できなかったような場面でも、.NET Frameworkが提供する機能を利用できます。これで、Visual Basicユーザーが手を焼いていたWindows APIの呼び出しの時の苦労が軽減されます。

.NET言語による共通技術の利用

設計、開発、実行すべての段階において、Visual Studioで共通のフレームワークを使います。つまり、言語は違っても、C#、 C++などと共通の技術（共通のランタイム＝共通言語ランタイム、クラスライブラリ、ツールなど）が利用できることになります。また、共通のデータ型のセット（共通型システム）を利用します。これにより、言語間の相互利用における型の問題がなくなります。

例外処理

例外処理が利用できるようになります。 これは、従来のエラー処理に比べてより詳細な対処が可能で、かつ、低コストであることが特徴です（第4章 の「エラー処理（例外処理）」を参照）。

その他

そのほかの利点として、レジストリ登録が不要、簡単なファイルコピーによる配置、DLL Hellの排除などをあげることができます。これらに関する詳細は、この章の後半で紹介しています。

また、今まではVisual Basicで作成できなかった、マルチスレッドのアプリケーションやWindowsサービス（常駐プログラム）なども作成できるようになっています。

Visual Basic .NETの開発環境の変更

- **開発環境の改良**
  - **タスクリスト**
  - **ダイナミックヘルプ**

Visual Basic .NETの開発環境の改良も見逃せません。改良された開発環境の一例を紹介します。

タスクリスト

Visual Basic 6.0と同様に、コーディング中に自動構文チェッカーがミスをチェックしています。さらにVisual Basic .NETでは、構文ミスがあると、エディタ中に波線が表示され、それと同時にこのタスクリストにリアルタイムで表示されます。さらに、ビルドエラーや、TODOコメントなども表示されます。「表示」メニューの［タスクの表示］にある［すべて］をチェックをすると、すべての情報が表示されます。またはフィルタをかけて必要な情報のみを表示することもできます。

TODOコメントを利用すると、保留にしている作業を忘れずにすむため便利です。

```
' TODO: 条件判定の式を追加する。
```

とコード中に記述しておくと、図1のように一覧で表示されます。タスクリスト上をダブルクリックすると、その行にジャンプして編集することができます。

図1：タスクリスト

ダイナミックヘルプ

名前の通り動的なヘルプで、たとえば、コーディング中にはテキストカーソルの位置にあるコードに関するヘルプが、また、フォームのデザイン中であれば今アクティブになっているオブジェクトに関するヘルプへのリンクが一覧で表示されます。図2は、エディタで「Sub」とコーディングしたとき（Subのコードの付近にカーソルがある状態の時）の様子です。

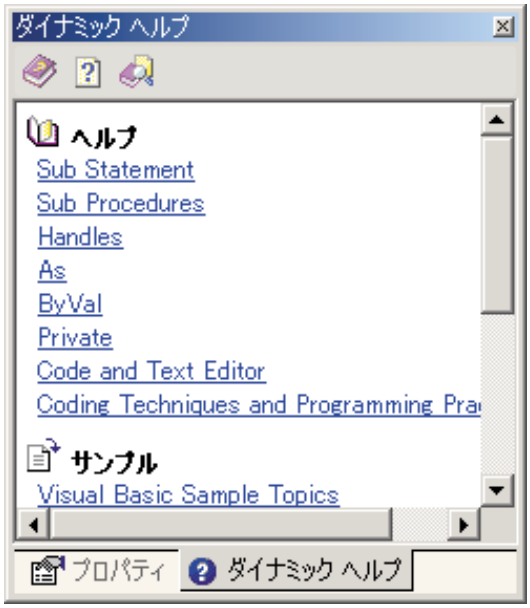


図2：ダイナミックヘルプ

このように、生産性を上げるためのさまざまな新機能を利用できることも、Visual Basic .NETの利点といえます。

Windowsアプリケーションの利点

- **Visual Basic .NETでWindowsアプリケーションを作成する利点**
  Windowsフォーム（リッチなGUIをもつアプリケーションを簡単に開発）
  生産性をUpする開発環境

Visual Basic .NETでは、Windowsフォームを利用して、リッチなユーザーインターフェイス機能を持つアプリケーションを作成できます。

Windowsフォームから、XML Webサービスへアクセスしたり、ADO.NETで取得したデータをコントロールにバインドして利用したりすることができます。また、.NET Frameworkの提供する、より高度な描画機能（GDI+）や、今まで苦労してきたWindows APIの代わりとなる機能を簡単に利用することができます。

Visual Basic .NETになっても、クリックやドラッグ＆ドロップ操作などを中心とした、従来どおりの開発のしやすさは変わりません。

また、Windowsアプリケーション開発の生産性をさらにUpするいくつかの機能が追加されています。

インプレースメニュー編集

インプレースメニューを利用すると、Visual Basic 6.0のときよりも、直感的にメニューを作成することができます（図3）。

図3：メニューの作成がカンタンに

アンカリング

アンカリングの機能を利用すると、ウィンドウの大きさにかかわらず、常に右上からの位置を一定に保つことができるボタンをコーディングなしで簡単に実現できます（図4）。

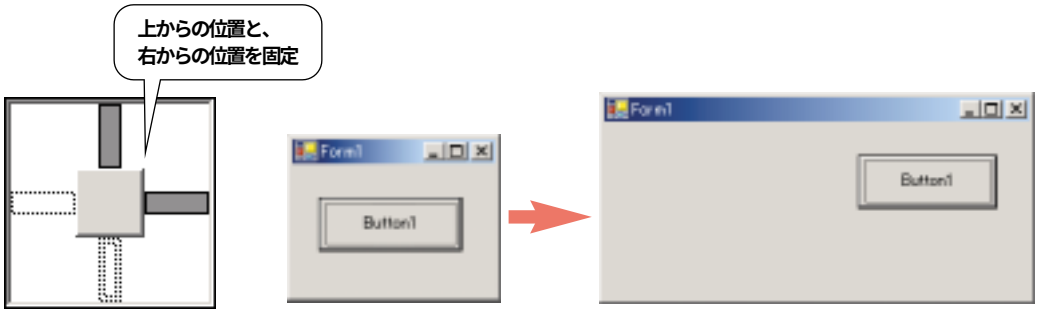


図4：上からの位置と右からの位置を固定すると

タブオーダー

タブオーダーも、フォーカスを移動させたい順にコントロールをクリックしていくだけで簡単に設定できます（図5）。

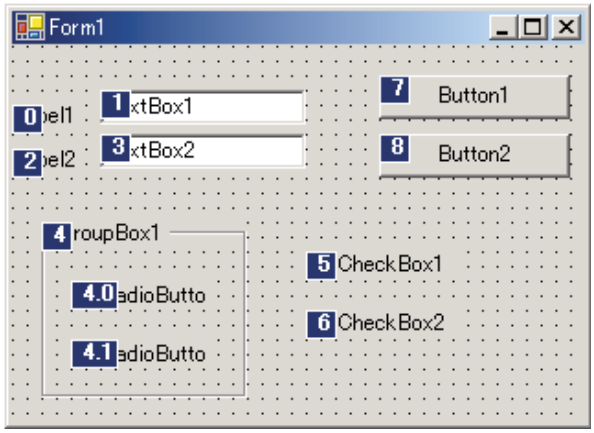


図5：タブオーダーもクリックするだけ

Webアプリケーションの利点

・Visual Basic .NETでWebアプリケーションを作成する利点
.NET Frameworkの利用
Windowsアプリケーションのように簡単に開発

Visual Basic 6.0だけでは、Webアプリケーションを作成することが困難なため、多くの開発者はVisual Basic 6.0でCOMコンポーネントを作成し、それをASP（Active Server Pages）から利用するというプログラムを書いていました。あるいは、すべてASPだけでWebアプリケーションを構築してしまっていた方もいるかもしれません。ASPは非常に簡単にサーバーサイドのプログラミングを実現できる反面、次のような欠点がありました。

・レイアウトとコードが混沌としている
GUI構成部分とコードがすべてASPファイルに含まれているため、作成、保守、デバッグが困難。

・コードの構造化や機能のカプセル化が困難
コードの構造化や機能のカプセル化が難しく、そのためコードがわかりにくくなりがち。いわゆる「スパゲティプログラム」になりやすい。

・ASPはインタープリタ型
スクリプトなので、大きなプログラムではそのパフォーマンスが問題。

・貧弱なGUI
VBのフォームのようなリッチなGUIを、開発環境のサポートなしにHTMLを書くことだけで実現するのはとても困難。

・複数のブラウザに場合分けのコードで対応
多様化するブラウザ事情に、すべて個別のページで対応するのが困難。

・開発環境の非力さ
Microsoftが提供する、最もすぐれたWeb開発環境のVisual InterDevでさえも、Visual Basicユーザーにとっては満足できるものではなかった。

これまでの、Webアプリケーション開発におけるグチをつらつらと書きましたが、.NET Framework＋Visual Basic .NETはこれらをすべて解消します。

・レイアウトとコードが分離できる
ASP.NETは、レイアウトとコードを分離してページを作成する機能を持っている。

・コードの構造化や機能のカプセル化が可能
ASP.NETは、Visual Basicと同様の構造でコーディングできる。

・ASP.NETは非インタープリタ型
MSILという中間コードにコンパイルされた後、実行時にその環境に最適なコードに変換されて実行される。そのため、今まで以上の実行速度が期待できる。このコードはキャッシュされるので、2回目以降はさらに実行速度が高くなる。

・リッチなGUI
Webフォームと豊富なコントロールを利用し、複雑なGUIを簡単に作成できる

（Webフォームやコントロールについては第3章の「Webアプリケーション」を参照）。

- **複数のブラウザに場合分けのコードで対応**

  これらのコントロールはブラウザによって異なるインターフェイスを自動で振り分けて提供する機能がある。

- **Visual Basic 6.0の、さらに上をいく開発環境**

  これからは、Visual Basicの「Windowsアプリケーションを作成する快適さ」でWebアプリケーションを開発することができる。

.NET Framework

・開発と実行の新しいプラットフォーム
.NET言語と共通言語仕様（CLS）
.NET Frameworkクラスライブラリ
共通言語ランタイム（CLR）

Windows上の分散システムをデザインするためのWindows DNAはあまりにも有名です。しかし、このWindows DNAは、プレゼンテーション、ビジネス、データの各層をデザインするための技術がそれぞれ別々に存在していました。つまり、Windows DNAアーキテクチャを利用した分散アプリケーションを作成するには、各層の技術に関して高度な、しかも別々の知識／技術が必要でした。
一方、.NET Frameworkは、これら、すべての層をカバーする開発と実行のプラットフォームを提供します（図6）。
具体的には、

・.NET言語と共通言語仕様（CLS：Common Language Specification）
・.NET Frameworkクラスライブラリ
・共通言語ランタイム（CLR：Common Language Runtime）

が存在します。

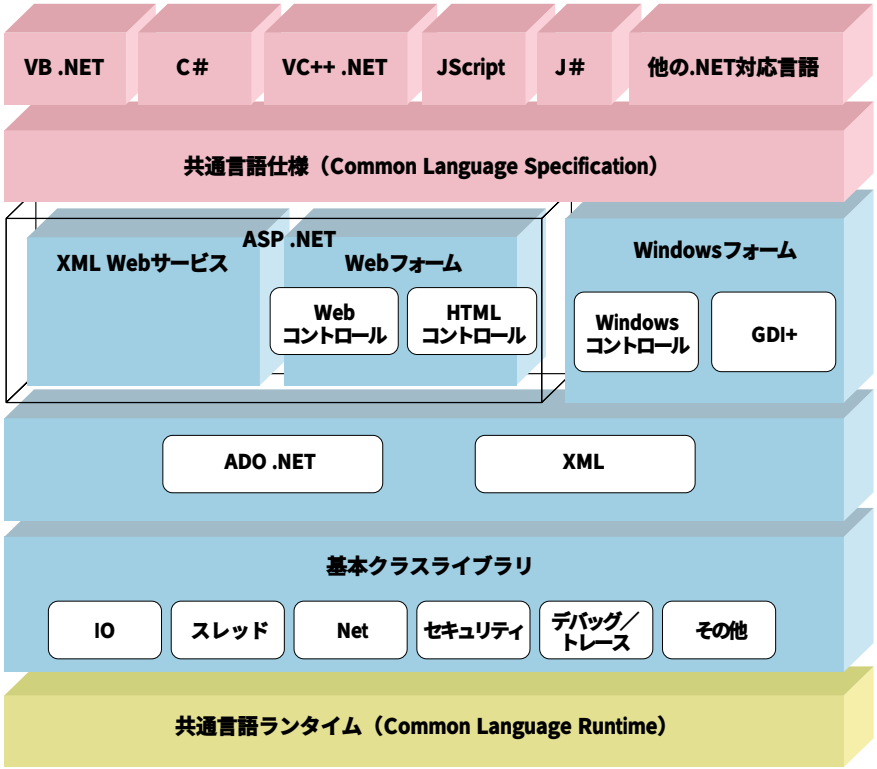


図6：.NET Framework主要コンポーネント

.NET言語と共通言語仕様（CLS：Common Language Specification）

共通言語仕様とは、.NET言語が共通で利用する言語機能のサブセットです。このサブセットに準拠する言語（ここでは.NET言語と呼んでいます）では、言語間の相互利用が容易です。また、.NET Framework対応のアプリケーションは、複数の.NET言語による開発が可能です。

.NET Frameworkクラスライブラリ

.NET Frameworkが提供するクラスライブラリは、ユーザーやプログラムのインターフェイスを提供するライブラリ、データアクセスのためのライブラリ、標準的な機能を提供するライブラリに分けることができます。

インターフェイスの提供は、Windowsアプリケーションだけでなく、Webアプリケーションを利用したユーザーインターフェイスや、XML Webサービスといったプログラミングインターフェイスを Visual Basic .NETで提供できることを意味しています。

また、データアクセスのためのADOやXMLなど、従来COMコンポーネントとして追加提供されていた機能が、.NET Frameworkに組み込まれていることを確認することができます。

標準的なシステムレベルの機能を提供するライブラリは特に基本クラスライブラリと呼ばれます。

共通言語ランタイム（CLR：Common Language Runtime）

共通言語ランタイムは、その名の通り、.NET言語共通のランタイム（実行エンジン）です。メモリ管理やガベージコレクションなど、.NET Frameworkの土台となる変更要素を含んでいます。この利点として、

- **多くをランタイムが提供してくれるため、開発者が多くのコーディングをしなくてもよい**
- **下部のアーキテクチャが隠蔽されることにより、その処理を考えなくてよい**

をあげることができます。しかし、もともとVisual Basic自身がこのような設計思想に基づいて作られていたため、私たちにとっては、大きなニュースではありません（多分この恩恵は、Visual Basicと同じランタイムを利用することになったVisual Basic .NET以外の.NET言語が受けた恩恵でしょう）。

これ以外の利点として、DLL Hellの終焉（「DLL Hellの終焉」参照）や、容易なデプロイメント（配置：「容易なデプロイメント」参照）をあげることができます。

Plus One　ガベージコレクションとVisual Basic

ガベージコレクションは、実はVisual Basic開発者にとっては、必ずしもうれしい追加機能とはいえません。なぜなら、Visual Basicはもともと、オブジェクトを参照している変数がスコープを抜けるなどして参照から外れると、自動的にオブジェクトが破棄されていました。つまり、自動メモリ解放機能であるガベージコレクションのような機能は、備わっていたといえます。

そして、.NET Frameworkの「ガベージコレクション」は、Visual Basicの「ガベージコ

レクションのような機能」と違う点があります。それは、オブジェクトが破棄されるタイミングです。共通言語ランタイムでは、メモリが必要だと判断したときに、ガベージコレクションを行ないます。言い換えると、必要でなければいつまでたってもガベージコレクションが行なわれないわけです。たとえば、後処理（Visual Basic .NETではClass_Terminateイベントの代わりにFinalizeメソッドを利用します）を記述していたとしても、忘れたころに、それが実行されるかもしれないのです。つまり、Visual Basic開発者が期待する、

「参照している変数がスコープを抜けた」→「オブジェクト破棄」→「後処理」

という一連の流れは、期待通りのタイミングで行なわれるとは限らないということを覚えておかなくてはいけません。

DLL Hellの終焉

- DLL Hellとは
- DLL Hellの原因
- そして、DLL Hellの終焉

DLL Hellとは

簡単に言うと、DLL Hell（地獄）とは、DLLのバージョンアップによって、前のバージョンを利用していたアプリケーションが動かなくなってしまうことや、それを避けるためにバージョンごとに別々のDLLファイルを作成した結果、システムディレクトリに同じようなDLLがあふれかえってしまう状況を指します。

DLL Hellの原因

もともと、DLL（Dynamic Link Library）は多くのシステムで利用されている、ディスクやメモリの節約を可能にするコンパイル済みのライブラリを提供するための技術です。

このDLLを使用することで起きるDLL Hellは、COMで解決される予定でした。COMのインターフェイスの決まりに則って正しくCOMを作成し、インストーラを使って正しくインストールしていれば、こういった問題は解決されたはずです。しかし、実際にはこの問題は根絶することはできませんでした。

原因として考えられることは、本来守るべきルール（「COMではインターフェイスを変えてはいけない」「下位互換性を保たなければいけない」など）を守らなかった設計者やプログラマのミスをあげることができます。

もっと基礎的な問題として、きちんとDLLをインストールせずに、ファイルを単純に上書きしてしまったというような原因もあるかもしれません。そこまでひどくないにしても、インストーラがきちんとバージョンをチェックせずにDLLをコピーしたときにも同様の状態になります。

そして、DLL Hellの終焉

この問題を解決するため、“プライベートDLL”という考え方があります。これはアプリケーションが利用するDLLを各々のフォルダなどに配置するというものです。これをSide-By-Side配置と呼びます。これにより、たとえば同じDLLが別のフォルダに配置され、しかも、それらを利用するアプリケーションが起動しているとき、同じ名前の異なるバージョンのDLLがシステム上で同時に実行されることが可能になります。

.NET Frameworkでは、このプライベートDLLの考え方を拡張して、DLL Hellに終わりを告げます。

まず、共通言語ランタイムは、同じDLLの異なるバージョンを管理することができ、それらが同時に実行されることもできるようになっています。

そして、DLLを利用する側でも、コンパイル時に、必要なライブラリのバージョンの情報がコンパイルしたモジュールに含められます。これにより、必ず正しいDLLが利用されることになります。さらに、バージョンポリシーを利用して、そのDLLの最新のバージョンを利用するように指定したり、ある特定のバージョンを指定したりす

ることもできます。

また、Visual Studio .NETには適切なデプロイメント（配置）を簡単に行なえるインストーラも存在しますので、これを利用することができます。

まとめ

Visual Basic 6.0で正しくCOMコンポーネントを作成し、ディストリビューションウィザードを使ってきちんと配布していた場合、DLL Hellは終わっていたのかもしれません。そのような方にとっては、.NET FrameworkはCOMの代わりにより、確実な新しい方法で、DLL Hellを解消することにしたと捉えることができます。

容易なデプロイメント

- **XCOPYで完了するデプロイメント**
- **推奨：適切なツールによるインストール（推奨）**

Visual Basic 6.0で作成したアプリケーションを配布するときは、多くの場合ディストリビューションウィザードや市販のインストーラ作成ツールを利用してインストーラを作成していました。なぜなら、アプリケーションのデプロイメント（配置）のためにしなくてはならないことがたくさんあったからです。

.NET Frameworkの目標のひとつは、このデプロイメントを単純化することです。そのために、「副作用なしのインストール」と「XCOPYによるデプロイメント」を導入しています。

副作用なしのインストール

.NET Frameworkにおける配置の単位は、「アセンブリ」です。アセンブリは自己記述的な性質（自分自身を説明する能力）を持っています。アセンブリには、そのアセンブリ自身の名前やバージョン情報、提供するクラス／メソッド／プロパティ、そのアセンブリが利用するアセンブリのバージョン情報、などが含まれます。この性質のために、レジストリに必要な情報を登録する必要がなくなり、その結果として別のアプリケーションに影響を与えることがなくなります。

また、Side-By-Sideの導入により、そのアプリケーションが利用するアセンブリを同じディレクトリに含めることができるため、アプリケーションのインストールによって他のアプリケーションに影響を与えることもなくなります。これにより、DLL Hellを回避します。

XCOPYによるデプロイメント

レジストリに依存していないため、デプロイメントの究極の方法として、アセンブリをDOSコマンドのXCOPYでコピーする方法を選択することもできます。

ただ、実際にはユーザーにファイルコピーを強いるのではなく、セットアップツールを利用したグラフィカルなユーザーインターフェイスの提供を推奨します。

Visual Studio .NETによるデプロイメント

Visual Studio .NETには、いくつかのデプロイメント用のプロジェクトテンプレートが用意されています。これらのテンプレートは、「新しいプロジェクト」ダイアログボックスの［セットアップ/デプロイメントプロジェクト］ノードで確認できます（図7）。

- **セットアッププロジェクト── Windowsインストーラ（.msiファイル）を作成します。自動修復機能や、オンラインデマンドインストールなど、高度なインストールおよびアプリケーションの維持を行なうことができます。**
- **Webセットアッププロジェクト── Web アプリケーションのインストーラ（.msiファイル）を作成します。セットアッププロジェクトとの違いは、ファイルの配置される場所です。セットアッププロジェクトは、ファイルをターゲットコンピュー**

タのファイルシステムにインストールし、Webセットアッププロジェクトは、ファイルをWebサーバーの仮想ディレクトリにインストールします。

- **マージモジュールプロジェクト── コンポーネントをパッケージ化したモジュール（.msmファイル）を作成します。これを、Windowsインストーラに含めて、インストールすることができます。これにより、複数のWindowsインストーラで共有利用できます。**
- **CABプロジェクト── WebブラウザにActiveXコントロールなどをダウンロードするキャビネットファイル（.cab）を作成します。**
- **セットアップウィザード**

以上のいずれかのデプロイメントプロジェクトを対話式に作成するためのウィザードです。

これらの、Visual Studio .NETによるインストール以外にも、すでに紹介したXCOPYや、Webアプリケーションの配置のために［プロジェクト］-［プロジェクトのコピー］を利用することもできます。ただし、これらは単純にファイルがコピーされるだけであるため、たとえばIISのディレクトリ設定などを別途手動で行なわなくてはいけません。できるだけ、「コピー」ではなく、Windowsインストーラなどによる「デプロイメント」を行ないましょう。

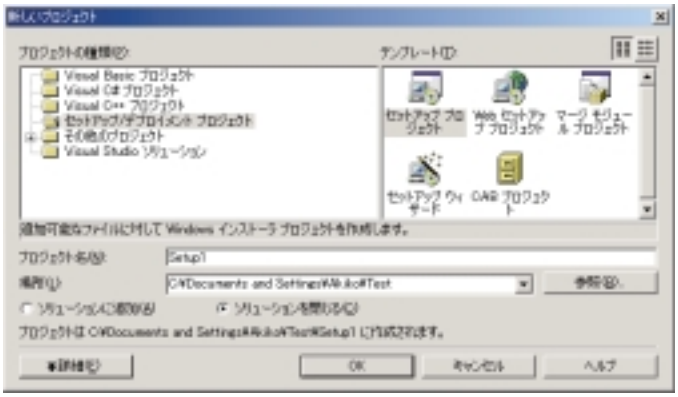


図7：デプロイメント用のプロジェクトテンプレート

第３章

スムーズな移行のためのポイント
～アーキテクチャ編～

Microsoft®
Visual Basic® .net™

アーキテクチャの変更のポイントを知っておくとVisual Basic .NETになったときに、スムーズな移行ができます。また、Visual Basic 6.0でどのようなアーキテクチャを選択しておけばよいかという指針が見つかります。
ここでは、Visual Basic 6.0のユーザーを対象に、「Visual Basic .NETへの移行」にフォーカスしたVisual Basic .NETの特徴を紹介していきます。

変更ポイント

Windows/Webアプリケーションをどのように作成するようになるのか、またコンポーネントはどうなるのか、といった変更ポイントを紹介します。

アップグレードウィザードによる変更点

アップグレードウィザードがどのような変更を加えてくれるかを紹介します。

今、何をしておくべきか？

Visual Basic .NETでの変更に備えて、現在開発中のVisual Basic 6.0のプロジェクトをどのように構成しておけばよいかの指針を紹介します。

関連・補足項目

各項目に関連する開発上のヒントや補足事項を紹介します。

- 変更点だけを効率よく知りたい方は 変更ポイント のみ、または 変更ポイント と 関連・補足項目 をお読みください。
- アップグレードウィザードの効果を知りたい方は アップグレードウィザードによる変更点 をお読みください。
- Visual Basic .NETへの移行を考慮したVisual Basic 6.0のプログラミングのポイントを知りたい方は 変更ポイント と 今、何をしておくべきか？ をお読みください。
- Visual Basic .NETを予習し、スムーズな移行を目指す方は 変更ポイント から 関連・補足項目 まで、すべてお読みください。

Windowsアプリケーション

- **Windowsアプリケーションは.NETでも作成可能**
- **Windowsフォームに自動的にアップグレード**

変更ポイント

Windowsアプリケーションは、Windowsフォームをベースに作成します。ユーザーインターフェイスは、WindowsコントロールやGDI+を利用して整えます。.NETコンポーネント（クラスライブラリやコントロール）を利用することもできますし、XML Webサービスを利用することもできます。

また、従来のCOMコンポーネント（コードコンポーネントやActiveXコントロール）を利用することもできます。ただし、COMと.NETの相互運用にはその境界を越えるためのオーバーヘッドが生じることを考慮に入れる必要があります。

なお、フォームおよびコントロールの変更の詳細は、第5章を参照してください。

アップグレードウィザードによる変更点

Visual Basic 6.0で作成されたWindowsアプリケーションは、アップグレードウィザードによってアップグレードすることができます。アップグレードするためには、Visual Basic 6.0のプロジェクトであることが必要です。それ以前のバージョンのVisual Basicで作成されたプロジェクトは、あらかじめVisual Basic 6.0にアップグレードしておきます。

Visual BasicのフォームはWindowsフォームに、また、フォーム上に配置されたコントロールも、可能な限り対応するWindowsコントロールにアップグレードされます。対応するものがないときには、アップグレードレポートとコメントに従って修正を行なう必要があります。これらの詳細は、第5章を参考にしてください。

COMコンポーネント（コードコンポーネントやActiveXコントロール）を利用しているアプリケーションもアップグレード可能です。このとき、相互運用のためのラッパーが作成され、必要な参照設定が自動的に追加されます。なお、COMコンポーネント自体のアップグレードについては、本章の「コンポーネント」を参照してください。

今、何をしておくべきか？

Visual Basic 6.0では、簡単にリッチなユーザーインターフェイスを備えたWindowsアプリケーションを作成することができました。これらを、ウィザードを利用して容易にVisual Basic .NETにアップグレードできます。また、COMコンポーネントを利用していても問題はありません。簡単にアップグレードできるからです。問題がないどころか、COMコンポーネントをできるだけ利用することをお勧めします。COMコンポーネントなどを利用して、ユーザーインターフェイスとビジネスロジックをできるだけ分けて、アプリケーションを作成しておきましょう。そうすれば、.NETにアップグレードする時、COMコンポーネントを対応する.NETコンポーネントに部分的に置き換

えたり、あるいはコンポーネントはそのままで、それを利用する側のプログラムを置き換えたりするなど、柔軟に対応することができます。

Plus One

Visual Basic .NETでは、Windowsアプリケーションと同じくらい簡単に、Webアプリケーションを作成できます。残念ながら、既存のWindowsアプリケーションを自動でWebアプリケーションにアップグレードするツールはVisual Studio .NETにはありません。しかし、Windowsアプリケーションをアップグレードすることで、コードが.NETに対応するよう書き直されるので、これをコピー＆ペーストして新しいWebアプリケーションのプロジェクトで活用するという方法は有効でしょう。

もちろん、すべてをWebアプリケーションにする必要はありませんし、Webアプリケーションに向かない場合や、移植が非常に困難な場合があります。以下にその例を示します。

複雑なユーザーインターフェイスを持つアプリケーション

ラベルやテキストボックスなど、標準コントロールを利用した簡単なユーザーインターフェイスを持つアプリケーションはWebアプリケーションへの移行も簡単に行なえます。しかし、複雑なインターフェイスや、サードパーティ製のコントロールを利用したアプリケーションはアップグレードが困難な場合が多く、また、Webアプリケーションに向かない場合も多いでしょう。

イベントを継続して利用するアプリケーション

Webフォームは、サーバーコントロールを利用するため、マウスの移動やドラッグなど継続的にイベントを取り続けたいアプリケーションなどは、向いているとはいえません。このような場合、DHTMLなどを利用することも可能ですが、この変更は簡単にはできません。

以上のようなアプリケーションは、そのままWindowsアプリケーションとして利用することをお勧めします。

また、このアップグレードをできるだけ簡単にするために、Visual Basic 6.0のプロジェクトにおいて、ユーザーインターフェイスとビジネスロジックを分けて作成しておくこと、また、できるだけユーザーインターフェイスを薄く作っておくことをお勧めします。

Webアプリケーション

- ASP.NETを利用したWebアプリケーションの開発
- 使い慣れた開発環境と言語で実現可能
- 移行が簡単なのはASP＋COMコンポーネント

変更ポイント

これまでは、Visual Basic 6.0だけでは、Webアプリケーションを作ることが難しかったので、Visual Basic 6.0でコンポーネントを作り、それをASPから呼び出して利用したり、ユーザーインターフェイスはDHTMLを利用したりしていました。つまり、Visual Basicの活躍の場が、中間層の作成のみに限定されていました。

これからは、Visual Basic開発者は、いつもの使い慣れた開発環境と言語を使ってWebアプリケーションを開発することができます。

Visual Basicに限らず、.NETでは、Webフォームを利用することができます。Webフォームには、「HTMLベースのユーザーインターフェイス定義」と「イベント処理などのロジック」の両方が含まれます。これを同じファイルに記述することも、ロジック（コード）を別のファイルに記述して、「ページの背後のコード（コードビハインド）」として提供することもできます。

Visual Basic .NETで、「ASP.NET Webアプリケーション」プロジェクトを作成すると、あらかじめ「WebForm1.aspx」と、コード用のファイル「WebForm1.aspx.vb」が作成されます（図1）。.aspファイル内の「runat="Server"」の記述のあるフォームやコントロールと、.vbファイル内のコードは、ASP.NETランタイムエンジン、つまりサーバーサイドのエンジンにより実行されます。このように、Webフォームを利用したアプリケーションは、基本的にサーバーベースのアーキテクチャを採用しています。とはいえ、Visual Basic .NETでは、フォームにコントロールを貼り付け、それにイベントプロシージャを実装する感覚でデザインしていくことができます。

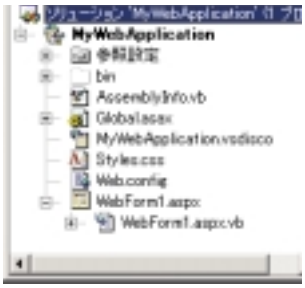


図1：「ASP.NET Webアプリケーション」プロジェクトの作成

Visual Basic .NETでは以下の3種類のコントロールを利用できます。

- HTMLクライアントコントロール
- HTMLサーバーコントロール
- Webサーバーコントロール

HTMLクライアントコントロールは従来からある、いわゆるタグに1対1で対応する、ブラウザ（クライアントサイド）で実現されるコントロールです。

HTMLサーバーコントロールも、いわゆるタグに1対1で対応するコントロールですが、サーバーサイドで実現されます。そのためサーバー側のコードと連携することが可能です。また、HTMLで実現している既存のページをサーバーベースに移行するためにも利用できます。

Webサーバーコントロールは、HTMLコントロールより多くの機能を提供し、以下のような特徴を持ちます。

- **Visual Basic開発者にとって使い慣れたオブジェクトモデル**
- **ブラウザを検出し、ブラウザに最適の出力を調整する**
- **適切なHTMLを生成し、ユーザーインターフェイスを実現する**

これらにより、強力なWebアプリケーションのプレゼンテーション部分をVisual Basicで作ることができるようになったというわけです。

ASP.NETを利用したWebアプリケーションを作成するためには、Visual Basicのアプリケーション開発のノウハウだけでなく、Webの開発知識（HTML、DHTML、XMLの基礎知識、セッション管理の考え方、クライアントサイドスクリプトなど）も必要です。もちろん、ASP.NETの技術習得も必要です。これにはかなりのページを費やすのでここでは解説できません。ASP.NETについては、本誌2002年2月号の別冊付録なども参考にしてください。

アップグレードウィザードによる変更点

Visual Basic 6.0では、Webアプリケーションを作成するためのプロジェクトタイプがいくつかありました。しかしそれらのほとんどは、アップグレードウィザードによって簡単にアップグレードできるというわけではありません。

アップグレードされないプロジェクトは、

- **DHTMLアプリケーション（DHTML＋クライアントサイドDLL）**
- **ActiveXドキュメント**

です。これらは、クライアントベースのアーキテクチャを使っているため、根本的にサーバーベースのアーキテクチャを持つASP.NETに、自動でアップグレードされません。また、手動での変更も簡単ではありません。

また、

- **IISアプリケーション**

は、ウィザードによってWebクラスがASP.NETにアップグレードされます。しかし、手動での修正が必要となるでしょう。

今、何をしておくべきか？

Visual Basic 6.0を利用してWebアプリケーションを作成するときの技術として、

・クライアントサイド
HTMLとクライアントサイドスクリプト

・サーバーサイド
ASP、COMコンポーネント（ここをVisual Basic 6.0で作成）

を利用していれば、スムーズな移行が可能です。

移行の段階には、ASPから.NETコンポーネントを利用することもできますし、COMコンポーネントをASP.NETから利用することもできます。

結論として、Visual Basic .NETへの移行を意識してWebアプリケーションを作成するなら、Windowsアプリケーションの時と同様、できるだけCOMコンポーネントなどを活用し、機能を分割して作成しておくとよいでしょう。そして、ユーザーインターフェイスをDHTMLとASP、コンポーネントの呼び出しをASP、ビジネスロジックをCOMコンポーネントで作成しておくことをお勧めします。これにより、既存の資産を活用し、スムーズな移行を実現することができます。

コンポーネント

- .NETコンポーネント
  クラスライブラリ（コードコンポーネント）
  Windowsコントロール、Webコントロール
- コードコンポーネントは自動アップグレード
- ユーザーコントロールは自動アップグレード不可

変更ポイント

Visual Basic 6.0では「ActiveX DLL」「ActiveX EXE」「ユーザーコントロール」の各プロジェクトでCOMコンポーネントを作成してきました。Visual Basic .NETでは、このCOMコンポーネントに代わるものとして、.NETコンポーネントを作成することができます。

.NETコンポーネントを作成するためには、プロジェクトのテンプレートとして、「クラスライブラリ」あるいは「Windowsコントロールライブラリ」を選択します。

「クラスライブラリ」プロジェクト

Visual Basic 6.0の「ActiveX DLL/EXE」に対応するプロジェクトを作成するには、プロジェクトのテンプレートとして、「クラスライブラリ」を選択します（図2）。このタイプを選択すると、Visual Basic 6.0のクラスモジュールにあたるClass1.vbというファイルがひとつ用意され、ここに次のようなコードが自動的に追加されます。

```
Public Class Class1

End Class
```

図2：クラスライブラリ

このClass…End Classに、プロパティやメソッド、イベントを実装していきます。このファイルは厳密にはVisual Basic 6.0の「クラスモジュール」とは異なります。Visual Basic .NETでは、「ひとつのファイルに複数のクラスを記述」できるからです。Class…End Classの記述がそれを可能にしています。もちろん、従来通り1つのファイルに1つのクラスのみを記述するということも可能ですが、複数のClass…End Classをファイルに追加することで、複数のクラスを1つのファイルで提供し、ファイル数を減らすことができます。

メソッドやイベントについては、その実装方法はほとんど同じです。プロパティの

記述はわかりやすく変更されました。第4章の「プロパティプロシージャ」を参照してください。

従来3つだったスコープ（Private、Friend、Public）に新たな概念が追加されました。追加されたキーワードは、Protectedで、これは「コードの継承」の機能追加によるものです。

「Windowsコントロールライブラリ」プロジェクト

コントロールを作成するには、プロジェクトのテンプレートとして、「Windowsコントロールライブラリ」を選択します（図3）。このタイプを選択すると、Visual Basic 6.0のユーザーコントロールモジュールにあたるUserContorl1.vbというファイルがひとつ用意され、ここに以下のようなコードが自動的に追加されます。

```
Public Class UserControl1
  Inherits System.Windows.Forms.UserControl

#Region " Windows Form Designer generated code "
(略)
#End Region

End Class
```

ユーザーコントロールは、UserControlから継承して作成します。また、#Region…#End Regionまでには、ユーザーインターフェイスを提供するためのコードが必要ですが、Visual Basic .NETではデザイナにより生成させることができます。
「クラスライブラリ」で解説した変更以外の変更点は、マニュアルなどを参考にしてください。

この他、Webコントロールを作成するためには、「Webコントロールライブラリ」プロジェクトを選択します。

図3：Windowsコントロールライブラリ

アップグレードウィザードによる変更点

Visual Basic 6.0で作成した「ActiveX DLL/EXE」プロジェクトは、.NETの「クラスライブラリ」に自動変換されます。ただ、今まで、クラスのプロパティで設定していたことが、属性としてコードに含めることができるようになったものについては、手動で変更が必要です。たとえば、トランザクションの設定などがそれにあたります。

また、ユーザーコントロールのプロジェクトはアップグレードすることができません。

ユーザーコントロールに変更を加えたい場合は、2つの選択肢があります。ひとつは、Visual Basic 6.0を使用してユーザーコントロールを変更し、それをCOMと.NETの相互運用機能を利用して使用する方法です。

そしてもうひとつは、Visual Basic .NETでコントロールを新しく作り直す方法です。これはもちろん大変ですが、Visual Basic .NETならではの新しい機能（動的プロパティの設定など）を利用できます。

Plus One　XML Webサービスの作成

コンポーネントとして提供してきた機能を、XML Webサービスに置き換えて提供してもよいでしょう。

これにより、ファイアウォールを越えたメソッド呼び出しが可能になります。また、データの受け渡しにはXMLが利用されるため、他の言語やプラットフォームとのやり取りがグンと楽になります。

ただし、トランザクションなど、一部サポートされない機能もあります。また、パフォーマンスについても考慮が必要です。

結論としては、トランザクションなどを必要としない、他のプラットフォームと相互運用が必要なケースでは、中間層をXML Webサービスにすることを考慮してもよいということがいえます。

ただし、もちろんこの変更は手動になります。

いったんVisual Basic .NETでコードを.NETにアップグレードしておき、そのコードを「ASP.NET Webサービス」プロジェクトにコピー＆ペーストして活用すると作業が楽になります。

統合デバッガ

統合デバッガにより、COM+コンポーネントの開発サポートが改善されました。ステップインでクライアントとCOM+コンポーネントのデバッグが可能になりました。

さらに、Visual Basic 6.0のCOM+コンポーネントへもステップスルー可能です。ただし、最適化なしの、シンボリックデバッグ情報が必要です。

データアクセス

- .NETではADO.NETを利用
  - 分散アプリケーションに適したモデル
  - XMLに強力に対応
- ADOとADO.NETはコントロールとのバインドが可能

変更ポイント

Visual Basic .NETでは、.NET Frameworkが提供するデータアクセスのためのライブラリであるADO.NETを利用できます。これは、アプリケーションが今まで以上に分散し、企業を超えて、企業間でひとつのアプリケーションを構成する場面などを想定しています。そのため、データソースに常に接続していない、いわゆる「非接続型のデータアクセス」にフォーカスしたモデルとして設計されています。同時に、XMLの機能が強化されていることも特徴のひとつです。さらに、分散アプリケーションで使用したときのパフォーマンスもADOと比べて改善されています。

WindowsフォームやWebフォームでは、ADO.NET（図4）や、それとバインドするためのDataGridコントロールなどを利用することができます。

・DataSet

ADO.NETの最も重要なクラスで、データをキャッシュして利用するために使います。ADOのRecordsetオブジェクトの機能のうち、分散アプリケーションにおいて必要な「非接続型のデータアクセス」を意識したオブジェクトであることがDataSetの特徴です。

・OleDbXXX

「OleDb」で始まるクラスは、OLE DBプロバイダ経由のデータアクセスを実現します。

・SqlXXX

「Sql」で始まるクラスは、直接SQL Serverにアクセスすることができます。

DataGridコントロールにバインドされたデータを、Windowsフォームでは読み書き両用に使えますが、Webフォームでは読み取り専用となります（もちろん相応のコードを書けば、Webアプリケーション上でもデータの更新は可能です）。

Visual Basic .NETでも、ラッパークラスを利用して従来のDAOやRDOやADOも利用

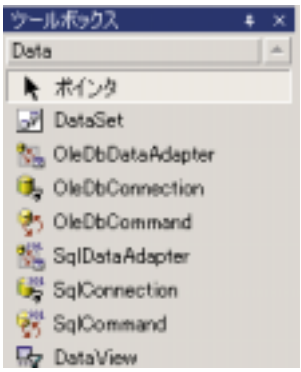


図4：ADO.NET

可能ですが、COMと.NETのオーバーヘッドがあります。また、ADO.NETと同様のデータバインディングが可能なのは、ADOだけです。

アップグレードウィザードによる変更点

コードのアップグレード

DAO、RDOを利用しているVisual Basic 6.0のプロジェクトをアップグレードすると、ラッパークラスが自動的に生成され、それを利用して動作するアプリケーションになります。またADOを利用しているプロジェクトは、システムが提供するADO2.7用のラッパークラス（「Primary Interop Assembly」として提供）を利用するアプリケーションにアップグレードされます。つまり、いずれも特別なコードの変更は不要です。

コントロールのアップグレード

データコントロール（DAOのデータコントロール）はアップグレードされずにエラーを表わす「赤い色のラベル」に変更されます（アップグレードしたときに対応するコントロールがないと、多くの場合この赤色のラベルになります）。

また、RDC（RDOのデータコントロール）はラッパークラスが生成されますが、コントロールへのデータのバインディングはできません。

ADODC（ADOデータコントロール）はVisual Basic 6.0の互換機能として提供されるADODCに置き換えられます。また、バインドしていたコントロール（データグリッドやデータリストコントロールなど）を利用している場合は、それらのためのラッパークラスが追加されます。バインディングの情報もそのままアップグレードされますので、特に修正をしなくてもそのままでアプリケーションが動作します。このままADODCを使うことも可能ですが、あくまでも、Visual Basic 6.0との互換のために用意されたものであるため、将来に渡って存在が保障されるわけではありません。オーバーベッドを考慮する場合や、ADO.NETの機能を利用したい場合は、ADO.NET、および対応するコントロールやコンポーネントに置き換えるとよいでしょう。

今、何をしておくべきか？

ADO.NETは、ADOに似ている部分もありますが、データへのアプローチやアプリケーション設計の方針が異なります。ですから、ADO.NETを利用したアプリケーションを作成する場合、既存のADOのコードを活用するよりも、ADO.NETでイチからコーディングをしたほうが、ADO.NETの特徴を生かしたアプリケーションが作成できるかもしれません。もちろん、データにアクセスするためのADOの知識が無駄になるわけではありません。むしろ、それと比較することでADO.NETの優位性が見えてきて、よりよいコードが書けると思います。実際多くのADO.NETのマニュアルはADOとの比較でそれを語っています。

そのような意味でも、Visual Basic 6.0ではADOを利用してADOに親しんでおき、.NETになったら、ADO.NETを利用することをお勧めします。

関連・補足項目

ADO.NETについては、ぜひ別途習得してください。

第4章

スムーズな移行のためのポイント
～プログラミング編～

Microsoft®
Visual Basic® .net™

プログラミングに関する変更のポイントを知っておくと、Visual Basic .NETを使う際にスムーズな移行ができます。また、今Visual Basic 6.0でどのように書いておけばよいかの指針が見つかります。
ここでは、Visual Basic 6.0のユーザーを対象に、「Visual Basic .NETへの移行」にフォーカスしたVisual Basic .NETの特徴を紹介していきます。

変更ポイント

Visual Basic 6.0から　Visual Basic .NETでどう変わるか、またVisual Basic 6.0からVisual Basic .NETで、文法がどのように変更されるかを紹介します。

アップグレードウィザードによる変更点

アップグレードウィザードがどのような変更を加えてくれるかを紹介します。

今、何をしておくべきか？

Visual Basic .NETでの変更に備えて、現在開発中のVisual Basic 6.0のプロジェクトをどのように構成しておけばよいかの指針を紹介します。

関連・補足項目

各項目に関連する開発上のヒントや補足事項を紹介します。

- 変更点だけを効率よく知りたい方は 変更ポイント のみ、または 変更ポイント と 関連・補足項目 をお読みください。
- アップグレードウィザードの効果を知りたい方は アップグレードウィザードによる変更点 をお読みください。
- Visual Basic .NETへの移行を考慮したVisual Basic 6.0のプログラミングのポイントを知りたい方は 変更ポイント と 今、何をしておくべきか？ をお読みください。
- Visual Basic .NETを予習し、スムーズな移行を目指す方は 変更ポイント から 関連・補足項目 まで、すべてお読みください。

変数宣言の変更

- Option Explicit Onがデフォルト
- Dim x, y, z As String
- ブロック内の変数のスコープ

変更ポイント

Option Explicit（変数宣言の強制）

変数宣言を強制するために、Option Explicit Onを利用します。また、Visual Basic .NETでは「Option Explicit On」がデフォルトになりました。つまり変数宣言を強制されたくない場合には、Option Explicit Offと明示的にソースコードに記述する必要があります。

変数宣言時の型指定

```
Dim x, y, z As String
```

この宣言は、Visual Basic 6.0では、2つのVariant型データ(xとy)と、ひとつのString型データ(z)の生成を意味します。

Visual Basic .NETでは、この宣言で3つのString型データが生成されます。

変数のスコープ

Visual Basic 6.0では、ローカル変数の宣言が含まれている行からプロシージャの末尾まで、ローカル変数を参照することができます。

Visual Basic .NETでは、ブロック内で宣言されている変数は、そのブロック内のみ利用可能です(リスト1)。

```
Option Explicit On
    (略)
Sub Proc1()
        Dim x   As   Integer
        If x = 100 Then
                Dim a As String
                a = "Hello"
        Else
                a = "Hi"  ' VB .NETではエラー
        End If
        a = "Hi"      ' VB .NETではエラー
End Sub
```

aのスコープ

リスト1：VB .NETの変数スコープ

変数宣言が強制されている場合、スコープ外での変数の利用はコンパイルエラーになります。

Ifステートメントの場合はIfの中で宣言したデータを利用できるのはIfからElseまでになります。

ただし、同じ名前の変数の宣言は、プロシージャの中で1度だけしかできません。したがってリスト2のコードはコンパイルエラーになります。

```
Option Explicit On
  (略)
Sub Proc1()
        Dim a As String
        Dim x As Integer
        If x = 100 Then
                Dim a As String
                a = "Hello"
        End If
End Sub
```

リスト2：変数名はプロシージャの中で1度だけ

アップグレードウィザードによる変更点

[Visual Basic 6.0 のコード]

```
Dim x, y, z As String
```

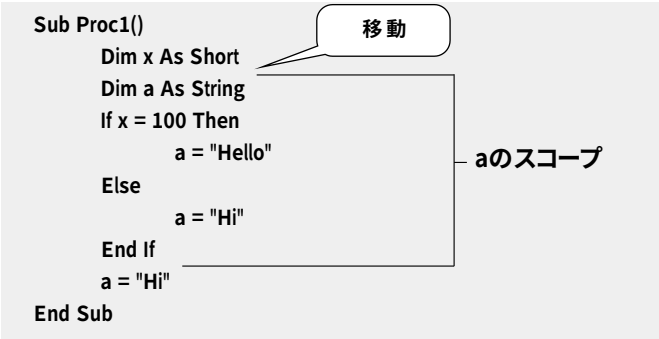
[Visual Basic .NET にアップグレード]

```
Dim x, y As Object
Dim z As String
```

また、Visual Basic 6.0で以下のようにブロック内で宣言していたデータは、

```
Sub Proc1()
        Dim x As Short
        If x = 100 Then
                Dim a As String
                a = "Hello"
        Else
                a = "Hi"
        End If
        a = "Hi"
End Sub
```

リスト3のように、アップグレードウィザードがブロックの外で宣言するように変更してくれます。



リスト3：スコープの変更を考慮したコードの変更（アップグレードウィザード）

今、何をしておくべきか？

Visual Basic 6.0でも変数は適切なデータ型を指定して宣言することが推奨されていました。Visual Basic .NETでは、Option Explicit Onがデフォルトになります。つまり、Visual Basic 6.0でも適切なデータ型を指定しておくことで、Visual Basic .NETでの変更を最小限に抑えることができます。

すべてがオブジェクト型

- 基本データ型（Integer、String）→オブジェクト型
- Variant→Object

変更ポイント

Visual Basic .NETでは、IntegerやStringなど、基本データを含むすべての変数がオブジェクトになりました。また、汎用データ型は、Variantではなく、Objectを使用し、型を指定しない変数のデータ型もObjectとなります。

たとえば以下のように記述することにより、安全性や、他の.NET言語との互換性が高まるだけでなく、変数の持つ便利なメソッドやプロパティを利用することができます。

```
Dim str1 As String, x As Integer
str1 = "ABCDEFG"
x = str1.Length    ' 文字列の長さを取得（StringオブジェクトのLengthプロパティ）
```

最初はこの変化に戸惑うこともあるかもしれませんが、使ってみるととても便利で、直感的なプログラムができる（しかも、安全に）ということに気づくでしょう。

アップグレードウィザードによる変更点

基本データ型は、対応するデータ（オブジェクト）に変更されます。といってもデータ型名に変更のないものも多いので、見た目には変化がない場合がほとんどです。

```
[Visual Basic 6.0のコード]
Dim str1 As String
Dim birthday As Date

[Visual Basic .NETにアップグレード]
Dim str1 As String
Dim birthday As Date
```

しかし、実際のデータ型は変更しています。

また、Visual Basic .NETでは通貨型（Currency）はなくなりますが、代わりに大きな数値を扱うことができるDecimalを利用します。そのため、Visual Basic 6.0の、

```
Dim cur As Currency
```

は、

```
Dim cur As Decimal
```

に変更されます。

もちろん、Variantや型指定なしの宣言はObjectになります。

[Visual Basic 6.0のコード]

```
Dim x As Variant
Dim y
```

[Visual Basic .NETにアップグレード]

```
Dim x As Object
Dim y As Object
```

整数型については、本章の「Short、Integer、Long」を参照してください。

今、何をしておくべきか？

必要に応じてアップグレードウィザードが変更を行なってくれるので、きちんと変数宣言をして整数を利用していれば問題はありません。変数を利用するとき（悪い意味で）テクニカルな使い方をしていると、Visual Basic .NETで変更が必要になります。たとえば、Booleanのデータを数値として利用したり、Visual Basic 6.0のDate型の内部構造がDouble型である（PlusOne参照）ことを利用（悪用？）したり、といった使い方です。

関連・補足項目

共通型システム（CTS：Common Type System）

.NET Frameworkでは、すべての言語間で共通の型システムを使用します。ここで扱うデータは、基本データ型も含め、すべてオブジェクトです。共通型システムを利用することで、言語間の型の問題が解決され、相互に利用しやすくなります。

値型と参照型

「すべてオブジェクト」といいましたが、実はデータは値型（オブジェクト自身）か参照型（オブジェクトへの参照）のいずれかに分類できます。値型に分類されるのは、基本データ型（ただし、StringとObjectを除く）と列挙型と構造体（ユーザー定義型）です。それ以外は参照型です。

```
Dim x, y As Integer
x = 10
y = x
```

上のコードでは、yに10という値が代入されています。この後、xやyを変更しても互いに影響は与えません。

それは、xとyが値型だからです。

一方、次のプログラムでは、objStudentXとobjStudentYが①で作成した同一のオブジェクトを参照します。

```
Dim objStudentX, objStudentY As clsStudent
objStudentX = New clsStudent()  ―――――――――――――― ①
objStudentY = objStudentX
```

そのため、一方の変更が他の参照に影響を与えます。

つまり、次のコードでは、「Akiko」が表示されます。

```
objStudentX.FName = "Akiko"
MsgBox(objStudentY.FName)
```

これが参照型の特徴です。

また、値型と参照型は初期化にも違いがあります。値型の場合は、変数宣言時にオブジェクトが作成されます。一方、参照型では変数が宣言されただけでは無効な状態（Nullが設定されている状態）になっています。Newキーワードでオブジェクトを生成し、それを代入することではじめて参照がセットされます。

Plus One　Date型

Visual Basic 6.0ではDate型の内部はDouble型でした。そのため、日付の格納と操作にDoubleデータ型を使用することができました。

Visual Basic .NET では、日付は内部的にDoubleとしては格納されないため、そのような使い方をしていると修正が必要になります。

たとえば、次のコードはVisual Basic 6.0では有効ですが、Visual Basic .NETでそのまま利用するとコンパイルエラーになります。

```
Dim dbl As Double
Dim dat As Date
dat = Now
dbl = dat                    ' Double型の変数に Date型の値を代入
dbl = DateAdd("d", 1, dbl)   ' Double型データを日付関数で使用
dat = CDate(dbl)             ' DateによるDouble型データのDate型への変換
```

.NET Frameworkには、DoubleとDateの間で変換を行なうための ToOADateおよびFromOADate関数があるので、これを利用して型変換を行なうことも可能です。ちなみに、上記のコードはアップグレードウィザードによって次のように変換されます。

```
Dim dbl As Double
Dim dat As Date
dat = Now
dbl = dat.ToOADate
dbl = DateAdd(Microsoft.VisualBasic.DateInterval.Day, 1, System.Date.FromOADate(dbl)).ToOADate
dat = System.Date.FromOADate(dbl)
```

今回の場合は、幸運にも適切にアップグレードされましたが、常にプログラムの意図が正しく解釈されるとは限りません。

Visual Basic .NETでの不必要な変更を避けるために、日付の格納には必ず Date型を使用するようにして、操作の際にDouble型として扱うのは避けてください。

Short、Integer、Long

- Shortは16ビット
- Integerは32ビット
- Longは64ビット

変更ポイント

整数データは表1のように変わりました。

	Visual Basic 6.0	Visual Basic .NET	C#	マネージドC++
16ビット	Integer	Short	short	short
32ビット	Long	Integer	int	int
64ビット	(none)	Long	Long	_int64

表1：.NETで変更された整数データ

アップグレードウィザードによる変更点

アップグレードウィザードによって、

```
[Visual Basic 6.0のコード]
Dim x As Integer     ' 16ビット
Dim y As Long        ' 32ビット

[Visual Basic .NETのコード]
Dim x As Short       ' 16ビット
Dim y As Integer     ' 32ビット
```

というように、大きさが同じ対応するデータ型に変更されます。

今、何をしておくべきか？

変数の大きさの変更にともない、32ビットCPUでの最適化を図るためにLongを使っていた場合には、Integerへの変換が必要です。

もちろん、アップグレードウィザードを使えば、適切にアップグレードしてくれるので、問題はありません。

Plus One

この変更により、SQL Serverのデータ型と同じ大きさになったため、バグが少なくなるという（うれしい）副作用もあります。

デフォルトプロパティ

・デフォルトプロパティは使えない

変更ポイント

Visual Basic 6.0 では、デフォルトプロパティを使うことができました。
たとえば、

```
Text2.Text = Text1.Text
```

というコードは、デフォルトプロパティを省略して、

```
Text2 = Text1
```

と、コーディングすることができました。Visual Basic .NETでは「デフォルトプロパティ」は使わなくなりました（PlusOne参照）。なお、例外については、関連・補足項目 を参照してください。

アップグレードウィザードによる変更点

アップグレードウィザードは、デフォルトプロパティを自動解決してくれます。
Visual Basic 6.0の、

```
Text2 = Text1        ' OKなパターン
```

というコードは、自動的に、

```
Text2.Text = Text1.Text
```

にしてくれます。
また、Visual Basic 6.0の、

```
Dim obj As TextBox                    ' これもOKなパターン
Set obj = Form1.Text1
MsgBox obj
```

というコードも、アップグレードウィザードが最後の行を、

```
MsgBox obj.Text
```

としてくれるので、変換後に何もしなくても動作します。
ただし、この「自動解決」をしてくれない場合もあります。それは、Visual Basic 6.0で書かれた以下のようなコードです。

```
Dim obj As Object          ' これは×なパターン
Set obj = Form1.Text1
MsgBox obj
```

変数をObject型として宣言し、実行時にクラスのインスタンスを代入するというこのパターンは、いわゆる実行時バインディングになっていて、オブジェクトが特定できません。そのためデフォルトプロパティの自動解決も行なえず、このままではエラーになってしまいます。そこで変換後、コードの手動変更が必要となります。

今、何をしておくべきか？

オブジェクト変数の実行時バインディングは避けましょう。

また、アップグレードウィザードを通さず、コピー＆ペーストでVisual Basic 6.0のコードを再利用しようと思うのであれば、デフォルトプロパティの利用も避けておくと無難です。

関連・補足項目

デフォルトプロパティ使用不可の例外

コレクションのようなパラメータ付きのデフォルトの使用は可能です。たとえば、

```
rs("CompanyName").Value          ' OK
```

は、Visual Basic 6.0でも、Visual Basic .NETでも、

```
rs.Fields("CompanyName").Value
```

のFieldsコレクションを省略したものとして解釈されます。

Plus One　なぜ、デフォルトプロパティがなくなったのでしょう？

たとえば、Visual Basic 6.0では、オブジェクトの代入にSetステートメントを利用しました。オブジェクトXとYがあったとき、

```
Set Y = X
```

は、X（オブジェクト）をYに代入することを意味します。そして、Xのプロパティを代入したいときは、

```
Y.プロパティ = X.プロパティ
```

と書くか、あるいは、このプロパティがデフォルトである場合は省略でき、

```
Y = X
```

と記述することができます。

Visual Basic .NETでは、オブジェクトの代入にも変数の代入にもSetステートメントを使う必要がなくなりました。

```
Y = X
```

と記述した際、（参照型であれば）Xが参照しているオブジェクトへの参照がYにセットされます。一方、「Xのプロパティ」を「Yのプロパティ」代入したいときには、

```
Y.プロパティ = X.プロパティ
```

と明示します。これにより、プログラムが明確になり、あいまいさがなくなるという利点があります。

プロパティの変更

- **CaptionプロパティはTextプロパティに**

変更ポイント

いくつかのプロパティが変更されています。

代表例は、フォームやラベルコントロールの文字列を設定するCaptionプロパティが、Textプロパティになったことです。ほかにも、リストボックスやコンボボックスコントロールのListIndexプロパティがSelectedIndexプロパティに変更されています。

これらは、他の.NET言語とプロパティ名をそろえるためです。

アップグレードウィザードによる変更点

アップグレードウィザードによって、これらのプロパティを利用しているコードはすべて自動的に変更されます。

```
[Visual Basic 6.0のコード]
Label1.Caption = "Hello"

[Visual Basic .NETにアップグレード]
Label1.Text = "Hello"
```

ですから、ほとんどの場合、問題はありません。

ただし、実行時バインディングを利用していると、正しく変更されません。たとえばVisual Basic 6.0で、

```
Dim obj As Object
Set obj = Form1.Label1
obj.Caption = "Hello"
```

というコードを記述していた場合、「obj」の型が特定できないため、変更してくれません。そのため、手動でCatpionプロパティをTextプロパティに変更しなくてはなりません。

今、何をしておくべきか？

Visual Basic .NETで手動変更をなくすためには、上のような実行時バインディングではなく事前バインディングを使用します。つまり、あらかじめ、

```
Dim obj As Label
Set obj = Form1.Label1
obj.Caption = "Hello"
```

のようにVisual Basic 6.0でコーディングしておくことをお勧めします。

このようにしておけば、「obj」の型がラベルであることがわかるので、アップグレードウィザードがCaptionプロパティをTextプロパティに変更してくれます。

オブジェクトを扱う変数は、Object型やVariant型として宣言するのではなく、可能な限り具体的なオブジェクトとして宣言するようにしてください。

Plus One よいプログラムを作成するために

Visual Basic .NETでは、「適切なデータ型を宣言し、正しくデータを利用するプログラミング」が求められます。なぜかというと、厳密な型チェックを行なうことでエラーの入り込みにくいプログラミングを実現するためです。

このことは、Visual Basic 6.0でも推奨されていたことでした。ですから、Visual Basic 6.0でも、きちんとデータ型を指定してプログラミングをすることで、よりエラーの入り込みにくいプログラムを作ることができ、同時にVisual Basic .NETでの変更を軽減できます。

また、どうしても、実行時バインディングを利用したい場合には、明示的な変換関数を使用することをお勧めします。これにより、コードの意図がわかりやすくなり、プロジェクトの Visual Basic .NETへの移行をスムーズにします。

Visual Basic .NETには、データ型の宣言を強制するための「Option Strict On」というオプションがあります。これは、「Option Explicit」と同様、コードの先頭に記述します。

これにより、次の事柄ができなくなり、実行の安全性が高まります。

- **明示的なキャスト（型変換）なしで、データを異なる型に代入する**
- **実行時バインディング**
- **Asなしの宣言**
- **Object型オブジェクトの使用制限（=、<>、Is、TypeOf…Is以外での使用不可）**

コードで「Option Strict」を指定しない場合、プロジェクトのプロパティ（［共通プロパティ］-［ビルドプロパティ］）で設定可能です。この設定のデフォルトは「Off」になっています。適切なコーディングの強制のために頁単位で「Option Strict On」の記述、またはプロジェクト単位での設定をお勧めします。

配列のインデックス

・配列のインデックスは0から始まる

変更ポイント

Visual Basic 6.0では、任意の上限、下限を持つ配列を宣言することができました。また、Option Baseステートメントで下限を指定しなかった時のデフォルトを「0（デフォルト）」または「1」に指定することもできました。

Visual Basic .NETでは、他の.NET言語との相互利用性を重視し、インデックスの下限は必ず「0」、つまり0オリジンの配列になります。上限については、従来と変更はありません（PlusOne参照）。

また、「インデックスは必ず0から」になるため、Option Baseステートメントはなくなりました。

```
Dim array(5) As Integer
```

は、Visual Basic 6.0でもVisual Basic .NETでも、インデックスが0～5までの要素数6の配列を宣言したことになります。

ちなみに、Visual Basic 6.0で記述可能だった、

```
Dim array（10 to 12） As Integer
```

という宣言は、Visual Basic .NETではできません。

アップグレードウィザードによる変更点

アップグレードウィザードにより「Option Base ステートメント」は削除されます。

また、インデックスの下限が0以外の配列は0オリジンの配列に変更され、アップグレードコメントが付加されます。

```
[Visual Basic 6.0のコード]
Dim array(10 To 12) As Integer          ' 要素数 3

[Visual Basic .NETにアップグレード]
Dim array(12) As Short                  ' 要素数 13
```

このときの要素数に注目してください。確かにこの配列を利用していたプログラムは引き続き配列を利用できますが、このままでは使われていない無駄な領域が存在することになります!!

そこで、効率よくメモリを利用するために、手動でコードを変更しなくてはなりません。

今、何をしておくべきか？

このような手動による変更をしなくてすむようにするためには、Visual Basic 6.0のプログラムでも、できるだけ0オリジンの配列を利用しておくとよいでしょう。

関連・補足項目

Visual Basic .NETで、0オリジンでない配列を使いたいときは、配列ラッパークラスを使うこともできます。ただし、配列ラッパークラスはネイティブな配列よりも低速であるだけでなく、配列型のパラメータを取る .NET Framework関数に渡したり、C# .NETまたはVC++ .NETのプログラムに渡したりすることができないため、お勧めできません。

Plus One

Visual Basic 6.0からの変更点として、Visual Basic .NETでは、他の言語との相互運用性を重視しインデックスの下限は「0」となりましたが、上限はVC++ .NETやC# .NETとは異なります。

各言語での配列宣言の例は次の通りです。

```
[C# .NETの場合]
int [] ar = new int[5];              // 要素数 5（インデックスは0〜4）

[VC++ .NETの場合]
System::Int32 ar[] = _gc new System::Int32[5];   // 要素数 5（インデックスは0〜4）

[Visual Basic .NETの場合]
Dim ar(5) As Integer                 ' 要素数 6（インデックスは0〜5）
```

固定長文字列

・固定長文字列はサポートされない

変更ポイント

固定長文字列は、（基本データ型としては）サポートされなくなりました。それは、配列と同様に他の .NET言語との互換性を持たせるためです。

アップグレードウィザードによる変更点

固定長文字列は、Visual Basic固有の「VB互換文字列（固定長）」に変換されます。

```
[Visual Basic 6.0のコード]
Dim str1 As String * 10

[Visual Basic .NETにアップグレード]
Dim a As VB6.FixedLengthString = New VB6.FixedLengthString(10)
```

このようにVisual Basic .NETには、固定長文字列の動作を提供する互換性クラスが用意されているため利用できます。そのため、アクセスが低速であることと、.NETの他の言語との互換性の問題を除けば、大きな問題はないでしょう。

今、何をしておくべきか？

もちろん、今から固定長文字列をできるだけ利用しないようにしておけば、Visual Basic .NETでも速度や互換性の問題は発生しません。

関連・補足項目

固定長文字列をユーザー定義型の中で利用する場合に発生する問題に関しては、「ユーザー定義型（構造体）の中の配列と固定長文字列」を参照してください。

構造体の中の配列と固定長文字列

- **構造体の宣言時**
  配列の要素数は指定できない
  固定長文字列は使えない

変更ポイント

Visual Basic 6.0では、ユーザー定義型の要素として、配列を宣言するとき、その要素数を指定できました。また、固定長文字列も指定できました。

ユーザー定義型は、Visual Basic .NETでは構造体（Structure）といいます。この構造体の宣言時に、配列の要素数の指定や、固定長文字列を使うことはできなくなりました。

アップグレードウィザードによる変更点

アップグレードウィザードは、これらの変更ポイントを自動で変更してくれます。

次のようなVisual Basic 6.0で作ったユーザー定義型をアップグレードすると、

```
[Visual Basic 6.0のコード]
Private Type MyType
        MyArray(5) As Integer
        MyString As String * 100
End Type
```

次のような、構造体になります。

```
[Visual Basic .NETにアップグレード]
Private Structure MyType
        Dim MyArray() As Short
        <VBFixedString(100), System.Runtime.InteropServices.Marshal As _
         System.InteropServices.UnmanagedType.ByValTStr, SizeConst:=100)>
        Dim MyString As String
        Public Sub Initialize()
                ReDim MyArray(5)
        End Sub
End Structure
```

この構造体には、配列の要素数を初期化するためのユーザー定義関数（Initialize）が含まれています。この関数を利用して配列の要素を初期化するよう、TODOコメントも追加されるので、このコメントを参考に、配列の要素数を初期化します。

また、固定長文字列については、可変長文字列に変更されていますが、固定配列やマーシャリングのための属性が追加されているので、固定長文字列のように利用できます。

今、何をしておくべきか？

この文法の変更に対処する方法はふたつあります。

たとえば、Visual Basic .NETでの変化に備えて、あらかじめ固定長文字列を利用しないようにすることです。たとえば、以下のように可変長文字列にしておき、必要に応じて配列の要素数を指定する初期化処理を行ないます。

```
Private Type MyType
        MyArray() As Integer
        MyFixedString As String
End Type

Sub Bar()
        Dim MyVariable As MyType
        ReDim MyVariable.MyArray(5) As Integer
        MyVariable.MyFixedString = String$(100, " ")
End Sub
```

あるいは、初期化のための関数を作成し、それを利用して要素数を指定してもよいでしょう。

そして、もうひとつの対処法は、Visual Basic .NETになってからアップグレードウィザードでできる限り変換後、必要に応じて手動で修正するという方法です。

このいずれかの選択になります。

Dim x As New MyClass

- **Dim x As New MyClassの動作が変更**

変更ポイント

Visual Basic .NETでは、このコードによる動作が変更されました。

Visual Basic 6.0では、このコードのある行ではオブジェクトは生成されずに、それを利用するタイミングに、オブジェクトがすでに作られているかどうかをチェックし、なければ作成されるというものでした。これによる利点と欠点は次の通りです。

利点（Visual Basic 6.0）

必ずオブジェクトが存在することが保証される（Nothingを設定してオブジェクトが破棄されたとしても、オブジェクトを再利用しようとすると、再作成される）

欠点（Visual Basic 6.0）

オーバーヘッド（オブジェクトが利用されるたびに毎回チェックが入る）

明示的なオブジェクトの管理がしにくい

Visual Basic .NETでは、

```
Dim x As New MyClass()
```

は、

```
Dim x As MyClass
x = New MyClass()
```

を1行で書き表わしたものにすぎません。そのため、Visual Basic 6.0とは異なり、Newされるのは後にも先にも1回だけです。もちろんオーバーヘッドもありません。

アップグレードウィザードによる変更点

アップグレードウィザードではコードは変更されません。ここでポイントとなるのは、コードは変更されないけれども、その意味は違ってしまっているという点です。

Visual Basic 6.0でこのコードを使っている場合、上記の利点（常にオブジェクトがあることが保障されている）を期待している場合が多いでしょう。しかし、Visual Basic .NETではその限りではないことに注意しなくてはいけないのです。もし、Visual Basic 6.0で「オブジェクトがあることを常に期待するプログラム」を書いているときは、Visual Basic .NETでは変更が必要になります。

今、何をしておくべきか？

原則として、Visual Basic 6.0では、

```
Dim x As New MyClass
```

はオーバーヘッドが大きいので、使うべきではありません。どうしても、そのようなプログラムが便利だという場合は、Visual Basic .NETになったときに、コードを変更する覚悟が必要です。

Plus One

Visual Basic .NETでも、常にオブジェクトが存在することを期待するプログラミングを実現したい場合、例外処理を利用することもできます。Visual Basic .NETでは、オブジェクトを利用しようとした際に、オブジェクトが見つからないと例外を検出するので、必要に応じて

```
x=New MyClass
```

を行なうというコードを書くこともできます。

ＶＢ定数

- **VB定数は、名前や値が変更されている**

変更ポイント

Visual Basic の組み込み定数（VB定数）は、その名前と値の多くが変更されています。たとえば、

```
vbMaximized
      ↓
System.Windows.Forms.FormWindowState.Maximized（Enum）

vbHourglass
      ↓
System.Windows.Forms.Cursors.WaitCursor（Cursorオブジェクト）
```

などのように、vbXXXではなく他の言語と共通のオブジェクトに変更になっています。

アップグレードウィザードによる変更点

VB定数は適切に変更されます。

```
Form1.MousePointer = vbHourglass
Form1.MousePointer = 11
```

などは、いずれも、

```
Form1.DefInstance.Cursor = System.Windows.Forms.Cursors.WaitCursor
```

のように、適切に変更されます。

しかし、VB定数の代わりにVB定数に相当する値を使用している場合、アップグレードが行なわれないこともあります。たとえばVisual Basic 6.0で、

```
x = 11
Form1.MousePointer = x
```

と記述したコードは、

```
x = 11
Form1.DefInstance.Cursor = x
```

のように、値はそのままで、しかも、アップグレードコメントなども付加されません。

このままの状態で変更せずにおくと、xのデータ型次第ではビルドエラーか実行時エラーが発生します。

今、何をしておくべきか？

Visual Basic 6.0でも推奨されているとおり、きちんとVB定数を使用していれば、Visual Basic .NETでの手動変更を最小限に抑えることができます。

プロシージャ

- 引数のデフォルトはByVal
- プロシージャのStaticキーワードはサポートされない
- プロシージャの呼び出しには“ () ”を付ける

変更ポイント

引数のデフォルトはByVal

Visual Basic 6.0では、特に指定をしない場合、引数はByRefとなりましたが、Visual Basic .NETではByValになります。Visual Studio .NETでは、混乱を防ぐために、型を指定しないと自動的にByValが追加されるようになっています。

Staticプロシージャ

Visual Basic 6.0でプロシージャにStaticキーワードを指定すると、プロシージャ内のローカル変数がStatic変数になり、変数をコール間で共有することができました。Visual Basic .NETではプロシージャに対してStaticキーワードを指定することはできなくなりました。Static変数にしたいときは、変数宣言時に個別に指定します。

プロシージャの呼び出し

プロシージャの呼び出しの時は引数を必ず“ (”と“) ”で囲みます。引数がない場合は、()は任意ですが、Visual Studio .NETのコードエディタは自動的に()を追加します。

アップグレードウィザードによる変更点

引数のタイプを指定しない場合、Visual Basic 6.0ではByRefを意味します。そのため、Visual Basic 6.0による、

```
Sub Proc1(ByVal x As Integer, ByRef y As Integer, z As Integer)
        z = x + y
End Sub
```

というコードの第3引数の「z」は、アップグレードウィザードによって、以下のようにByRefが明示的に追加されます。

```
Sub Proc1(ByVal x As Short, ByRef y As Short, ByRef z As Short)
        z = x + y
End Sub
```

これは、何も書いていないと、Visual Basic .NETではByValを意味してしまうためです。
また、これを呼び出すための、

```
Proc1 a, b, c
```

は、()が追加され次のように変換されます。

```
Proc1(a, b, c)
```

Plus One

Visual Basic では、関数名を利用して戻り値を設定していました。

```
<関数名> = <戻したい値>        （Visual Basic 6.0とVisual Basic .NET）
```

Visual Basic .NETもこの方法をサポートしていますが、それに加え、Returnキーワードを利用して、

```
Return <戻したい値>            （Visual Basic .NET）
```

とコーディングすることができるようになりました。これにより、コードの意味が明確になります。

また、Returnキーワードを利用すると、実行コードを生成するときに最適化がより適切に行なわれるため、パフォーマンスの面でも優れているといえます。関数名をReturnに変えるだけでよいのですから、ぜひ置き換えを行なってください。

プロパティプロシージャ

- プロパティ構文が変更
- アクセスレベルは、SetとGetでそろえる
- デフォルトプロパティはコード中で指定（パラメータ付きのみ）
- Friendは微妙に変更

変更ポイント

プロパティ構文が変更になりました。Visual Basic .NETでは、

```
Public Property MyProp1() As Object
        Get
                ' プロパティを取得する処理
        End Get
        Set
                ' プロパティを設定する処理
        End Set
End Property
```

と、プロシージャを一体化して書きます。もちろん、取得専用または設定専用のプロパティ（関連・補足項目 参照）を作成することもできます。Visual Basic 6.0では、プロパティのデータ型によってSetまたはLetを使い分けていましたが、すべてSetになります。

このコードの書き方からも予想がつくと思いますが、SetとGetのアクセスレベルをそろえる必要があります。これは、意味的にも正しいことですので、異なるアクセスレベルに設定できないことは、問題ではないでしょう。

また、Visual Basic 6.0でのデフォルトプロパティの設定は、「ツールの属性」メニューからプロパティを「既定値」に設定することで行なっていました。そのため、コードを見ただけでは、デフォルトプロパティがどれであるかがわかりませんでした。Visual Basic .NETでは、コードの中にDefaultキーワードを挿入することで設定できるようになりました。

```
Default Public Property MyProp22(ByVal index As Integer) As Object
        Get
                ' プロパティを取得する処理
        End Get
        Set
                ' プロパティを設定する処理
        End Set
End Property
```

ただし、Visual Basic .NETではパラメータのないプロパティをデフォルトプロパティに設定することはできないことを忘れないでください。

アップグレードウィザードによる変更点

できる限りアップグレードしてくれます。たとえば、

```
Private mvarMyProp1 As Variant

Private Public Property Let MyProp1(ByVal vData As Variant)
        ' プロパティを設定する処理
End Property

Public Private Property Get MyProp1() As Variant
        ' プロパティを取得する処理
End Property
```

というVisual Basic 6.0のコードは、

```
Private mvarMyProp1 As Object

Public Property MyProp1() As Object
        Get
                ' プロパティを取得する処理
        End Get
        Set(ByVal Value As Object)
                ' プロパティを設定する処理
        End Set
End Property
```

とアップグレードされます。
次のような変更をチェックしてみて下さい。

- **SetとGetの一体化**
- **LetをSetに変更**
- **アクセス指定子が異なる場合、Publicに統一**

デフォルトプロパティの指定は、手動で追加して下さい。

今、何をしておくべきか？

この変更に備えて、特にしておくことはありません。

関連・補足項目

取得専用のプロパティにはReadOnlyキーワード、設定専用のプロパティにはWriteOnlyキーワードを指定します。
Set…End Setを記述しないときにReadOnlyキーワードを忘れるとエラーになります。また、ReadOnlyキーワードがあるときに、Set…End Setがあってもエラーになります。
Get…End Getを記述しないときにWriteOnlyキーワードを忘れるとエラーになりま

す。また、WriteOnlyキーワードがあるときに、Get…End Getがあってもエラーになります。

Plus One

Friendの効果が微妙に変化しました。

Visual Basic 6.0では、同一プロジェクトから参照したいが、外部に公開したくないプロパティなどにFriendと付けました。Visual Basic .NETでは、同一アセンブリ内から参照したいが、外部に公開したくないプロパティなどにFriendと付けます。

この2つは、ほとんど同じように見えます。たしかにVisual Studio .NETを利用している場合は同じといってよいかもしれません。しかし、厳密には、Friendはアセンブリ内であれば参照できますので、たとえば、Visual Basic .NETのFriendプロパティを同一アセンブリのC# .NETから利用できる、ということになるのです。これが「微妙」な変化です。

COMコンポーネントの利用

・プロジェクトをアップグレードすると自動的に利用可能に

変更ポイント

Visual Basic .NETからCOMコンポーネントを利用することができます。COMコンポーネントを利用するためには「参照設定」を行ない、オブジェクトを生成して利用します（作業ベースではVisual Basic 6.0とほとんど変わりはありません）。

Visual Basic .NETでは、COM以外の一般のオブジェクトを作成する場合、Newキーワードを利用し、CreateObject関数は使用しません。そして、COMのオブジェクトを生成する場合は、NewキーワードまたはCreateObject関数を利用します。

アップグレードウィザードによる変更点

COMコンポーネントを利用しているVisual Basic 6.0のプロジェクトをアップグレードすると、COMコンポーネントを利用するための参照がプロジェクトに自動的に追加されます。そのため、ほとんどの場合、手動でコードを変更することなくCOMコンポーネントを利用できます。

たとえば、次のようなCOMコンポーネントを利用するVisual Basic 6.0のコードは、

```
Dim x As MyProject.MyClass
Set x = New MyProject.MyClass
x.MyMethod1
```

アップグレードウィザードによって、次のように変更されます。

```
Dim x As MyProject.MyClass
x = New MyProject.MyClass()
x.MyMethod1()
```

なお、CreateObject関数はそのままで、変更は行なわれません。

今、何をしておくべきか？

「COMコンポーネント」と「それを利用するプログラム」は両方、またはどちらか一方のみをVisual Basic .NETにアップグレードして利用できます。COMで部品化しておくことで、段階的な.NET化が可能になります。

ただし、一方ずつアップグレードする場合、Visual Basic .NETのコード（マネージドコード）とCOMのコード（アンマネージドコード）との相互運用のため、オーバーヘッドが生じてしまいます。

関連・補足項目

アップグレードウィザードウィザードを利用しない場合も、COMコンポーネントを簡単に利用できます。

Visual Studio .NETの「ソリューションエクスプローラ」からプロジェクト名を右クリックして、表示されるメニュー（図1）から「参照の追加」を選択します。

「参照の追加」ダイアログの「COM」タブ（図2）から利用したいCOMコンポーネントを選択すると、必要に応じてCOMを利用するためのラッパークラスを生成してくれます。図3はこのとき表示されるメッセージです。

図1：「参照の追加」を選択

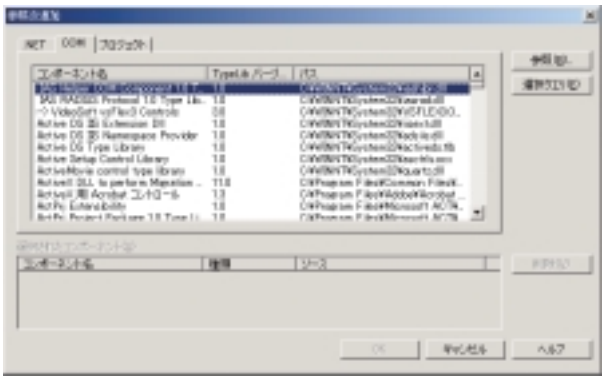


図2：利用したいCOMコンポーネントを選択

図3：ラッパークラスの生成

APIの利用

- **Windows APIに対応する.NET Frameworkクラスライブラリの利用（推奨）**
- **引き続き各種APIを利用することも可能**

変更ポイント

ほとんどの場合、Windows APIには、対応する.NET Frameworkが提供するクラスライブラリがありますので、書き換えることができます。しかし、対応するものがない場合やDLLとしてサードパーティなどから提供されるAPI関数などを使いたい場合、今までと同様にそれらを利用することもできます。

アップグレードウィザードによる変更点

Windows APIを利用するプログラムをアップグレードすると、可能な限り変更を行なってくれます。

- **データ型などのアップグレード（Long→Integerなど）**
- **固定長の文字列は、VB互換文字列に変更される→そのままAPIに渡すことができる**

また、変更してくれるが修正が必要なものとしては、属性情報の追加などがあります。これは必要に応じて追加してください。

- **属性などの（一部）追加**
  （例）Ansi、Unicode キーワードを使って、文字列をどのコードで渡すかをオプションとして宣言可能

アップグレードウィザードが対応していないものとしては、

- **対応する.NET Framework クラスライブラリを利用するコードへの変更**
- **As Any（複数のDeclare宣言などに手動で対応）**
- **スレッド作成、Windowsサブクラス化、メッセージキューのフックなどの操作を実行するAPIの使用（ただし、Visual Basic .NETではランタイムエラーを発生する場合もある）**

などがあります。

以上のように、.NET Frameworkを利用するように変更されるわけではないので、必要に応じて手動で変更を行なう必要があります。

今、何をしておくべきか？

Windows APIの使用に関しては、Visual Basic 6.0のプロジェクトで、.NETに向けてしておくことはありません。Visual Basic .NETに移行する過程で、対応する.NET Frameworkクラスライブラリを利用するように変更しましょう。

VC++ 6.0などを利用した自作のDLLをVisual Basic 6.0から呼び出しているときなどは、そのDLLを作成するときに、すでに紹介したデータ型などを考慮しておくと移行がスムーズでしょう。

Plus One

多くのAPIはVisual Basic 6.0と同様に使用できますが、データ型の調整が必要な場合があります。たとえば、整数型についてはアップグレードウィザードによって自動的に変更されます。

また、場合によっては、Visual Basic .NETのほうがAPIへの文字列の受け渡しを適切に行なえます。ANSIおよびUnicodeキーワードを使って、文字列をどのコードで渡すかをオプションとして宣言することができるからです。

エラー処理（例外処理）

- エラー処理はそのまま利用可能
- 新しい例外処理に変更してもよい

変更ポイント

Visual Basic 6.0同様、以下のようなVisual Basic固有のエラー処理はVisual Basic .NETでも利用することができます。

```
Sub Proc1
        On Error GoTo ErrProc
        ' エラーの起きそうな処理
        Exit Sub
ErrProc:
        ' エラー処理コード
        Resume Next
End Sub
```

もちろん、

```
On Error Resume Next
```

も利用できます。

これらに加え、Visual Basic .NETでは以下のような「例外処理」を利用することもできます。

```
Sub Proc1
        Try
                ' エラーの起きそうな処理
        Catch
                ' エラー処理コード
        End Try
End Sub
```

例外処理は、

- 見やすい
- 柔軟な例外処理が可能
- ネストが可能
- 例外オブジェクトの利用

などの特徴を持ちます。これにより、より高度なエラー処理が可能になります。ただし、同じプロシージャの中で、エラー処理と例外処理の両方を利用することはできません。

アップグレードウィザードによる変更点

Visual Basic固有のエラー処理はVisual Basic .NETでも利用できるため、アップグレードウィザードによる変更は行なわれません。

今、何をしておくべきか？

エラー処理に関しては、Visual Basic 6.0のプロジェクトで特に注意することはありません。

関連・補足項目

ResumeやResume Nextの機能は非常に魅力的ですが、エラー処理によってエラーが回避されない場合、エラートラップとエラー処理のループを抜け出せずに、結局アプリケーションを強制終了させなければならない状況に陥ることがあります。また、Resumeを行なうためには非常にコストがかかります。このような理由から、Visual Basic .NETでは、例外処理のほうをお勧めします。

レガシー機能

・Visual Basic .NETでなくなる機能はVisual Basic 6.0でも（できるだけ）使わない

変更ポイント

Visual Basic .NETにはなくなるVisual Basic 6.0の機能

・On～GoTo／GoSub、GoSub／Return
・VarPtr、ObjPtr、StrPtr
・LSetによるユーザー定義型変数の代入
・DefBool、DefByte、DefInt、DefStrなど

これらの機能を利用したことがない方は、このページは一切読む必要はありません。

アップグレードウィザードによる変更点

意味的に対応する処理に変更されるものもあります。たとえば、Visual Basic 6.0で記述された以下のようなOn…GoToは、

```
Private Sub Proc1()
        Dim number As Integer
        number = 2
        On number GoTo LineX, LineY
LineX:
        ' 処理
LineY:
        ' 処理
End Sub
```

アップグレードウィザードによって、次のようなSelect Caseステートメントに変換されます。

```
Private Sub Proc1()
        Dim number As Short
        number = 2
        Select Case number
                Case Is < 0
                        Error(5)
                Case 1
                        GoTo LineX
                Case 2
                        GoTo LineY
        End Select
LineX:
        ' 処理
LineY:
        ' 処理
End Sub
```

このように、非常に冗長になっていることがおわかりいただけると思います。
また、Visual Basic 6.0で使用していた次のようなDefxxxは、

```
DefStr A-C          ' AからCで始まる変数をString型で扱う
Adata1 = "Hello"
```

アップグレードウィザードによって次のように変換されます。

```
Dim Adata1 As String
Adata1   =   "Hello"
```

変換されるからといって、Visual Basic 6.0でのこれらの使用をお勧めするわけではありません。
また、これ以外のレガシー機能は、ウィザードによっても変更されず、そのままではビルドエラーになります。

今、何をしておくべきか？

このようなレガシーな機能は、できる限りVisual Basic 6.0でも使用を避けるようにしてください。
たとえば、On…GoToを使用する代わりに、Visual Basic 6.0でもあらかじめ、適切なSelect…Caseを利用しておきましょう。また、Defxxxステートメントを使うのは避け、変数を明示的に宣言するようにしましょう。
これにより、Visual Basic .NETへのスムーズな移行が行なえます。

第5章

スムーズな移行のためのポイント
～フォームとコントロール編～

Microsoft®
Visual Basic® .net™

.NET Frameworkでは、XML WebサービスやWebフォームが注目されていますが、Windowsアプリケーションを作成するためのWindowsフォームやコントロールの機能アップも見逃せません。さらに、今まで同様、いや今まで以上にその重要性が増したともいえます。
というのは、今まで、「Windowsアプリケーションを利用した、リッチで柔軟なユーザーインターフェイスをもつクライアントアプリケーションを提供したい」と思っても、配置の問題で、別の、たとえばWebベースのクライアントを選択せざるを得ないという状況がありました。しかし、.NET Frameworkでは、容易なデプロイメント（第2章参照）が可能になったため、配置はそれほど大きな問題ではなくなったからです。
また、Visual Basic 6.0で作成した既存のWindowsアプリケーションは、アップグレードウィザードによりアップグレードされるため、それをもとに適切に修正することで比較的簡単に新しいWindowsアプリケーションを作成できます。
ここでは、フォームとコントロールを利用したVisual Basic 6.0からの変更点を、「Visual Basic .NETへの移行」にフォーカスして紹介していきます。

変更ポイント

Visual Basic 6.0からVisual Basic .NETで、フォームやコントロールがどのように変更になるかを紹介します。

アップグレードウィザードによる変更点

アップグレードウィザードがどのような変更を加えてくれるかというパターンを紹介します。

今、何をしておくべきか？

Visual Basic .NETでの変更に備えて、現在開発中のVisual Basic 6.0のプロジェクトをどのように書いておけばよいかの指針を紹介します。

関連・補足項目

各項目に関連する開発上のヒントや補足事項を紹介します。

- 変更点だけを効率よく知りたい方は 変更ポイント のみ、または 変更ポイント と 関連・補足項目 をお読みください。
- アップグレードウィザードの効果を知りたい方は アップグレードウィザードによる変更点 をお読みください。
- Visual Basic .NETへの移行を考慮したVisual Basic 6.0のプログラミングのポイントを知りたい方は 変更ポイント と 今、何をしておくべきか？ をお読みください。
- Visual Basic .NETを予習し、スムーズな移行を目指す方は 変更ポイント から 関連・補足項目 まで、すべてお読みください。

フォームの変更

- **自動アップグレード**
  VB固有のフォーム→Windowsフォーム
- **フォームデザイナやプロパティウィンドウによるコード生成**
- **Windowsフォーム／コントロールはクラス**

変更ポイント

Windowsフォーム

Visual Basic 6.0で利用していたフォームは、Visual Basic固有のフォームでした。Visual Basic .NETでは、Frameworkが提供するWindowsフォームを利用します。このフォームは、System.Windows.Forms.Formから継承して作られます。

というと難しく聞こえるかもしれませんが、実際には従来と同様、［プロジェクト］-［Windowsフォームの追加］メニューから簡単にWindowsフォームをプロジェクトに追加できます。さらに、自分で作成したフォームをテンプレートとして、それを継承した新しいフォームを作成する、といったこともできます。

フォームデザイナやプロパティウィンドウによるコードの生成

フォームやコントロールを利用するには、これらを生成するための定義やプロパティを設定するためのコードを記述しておくことが必要があります。

というと面倒そうに思ってしまうかもしれませんが、実際の作業としては従来同様、フォームデザイナを利用してフォームやコントロールをデザインし、プロパティウィンドウでプロパティを設定します。これにより、自動的に定義や設定がコードとして生成されるのでコーディングの必要はありません。しかも、これらは、最初は隠されているので（図1)、必要に応じて行頭の［＋］の部分をクリックして展開し、生成されたコード参照することができます（図2)。図2の中のコメントからもわかるように、

```
⊞ Windows Form Designer generated code
```

図1：VB .NETが生成したコードは隠されている

```
#Region " Windows Form Designer generated code "
    Public Sub New()
        MyBase.New()
        ' この呼び出しは Windows フォーム デザイナで必要です。
        InitializeComponent()
        ' InitializeComponent() 呼び出しの後に初期化を追加します。
    End Sub
    ' Form は dispose をオーバーライドしてコンポーネント一覧を消去します。
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub
    ' Windows フォーム デザイナで必要です。
    Private components As System.ComponentModel.Container
    ' メモ : 以下のプロシージャは、Windows フォーム デザイナで必要です。
    ' Windows フォーム デザイナを使って変更してください。
    ' コード エディタは使用しないでください。
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
        Me.Text = "Form2"
    End Sub
#End Region
```

図2：生成されたコードを表示

禁止されている部分はコードエディタから修正できませんが、その他についてはコードエディタからも修正可能です。

アップグレードウィザードによる変更点

Visual Basic 6.0で作成したWindowsアプリケーションをアップグレードすると、Visual BasicフォームはWindowsフォームに自動的にアップグレードされます。

また、このときフォームやコントロールの定義やプロパティ設定をコードとしても生成してくれるため、対応するオブジェクト、プロパティ、メソッド、イベントが存在する場合には、多くの部分がアップグレードされます。なお、変更されるプロパティ、メソッド、イベントについては、本章の「フォームのプロパティ、メソッド、イベント」を参照してください。

関連・補足項目

VB固有のフォームからWindowsフォームへの変更に関するもうひとつのポイントは、「Windowsフォームやコントロールはクラスである」ということです。図3はフォームとコントロールの階層を表わしています。この図では、下位のクラスは上位のクラスを継承して存在します。このように上位クラスを継承して新しいクラスを作ることを“派生”といいます。派生すると、基本的に下位のクラスは上位のクラスの機能＋αの機能を提供します。

Windowsフォームもコントロールもいずれも、元はControlクラスから派生しています。Controlクラスは、WindowsフォームとグラフィカルなUIをもつコントロールに共通の機能を提供します。ScrollableControlクラスでは、その名前の通り、スクロールに関する機能が追加され、ContainerControlでは子コントロールを管理する機能が追加されます。これらのクラスから派生するFormクラスや独自のWindowsコントロールを作成するためのUserControlクラスはこれらの機能をすべてもつことになります。さらに各プロジェクトで、Formクラスから派生して作られたFormクラスを作成し、利用します。しかし、これらはあくまでもクラス（定義）に過ぎないので、これらを利用するためには実際にはnewステートメントなどを利用してインスタンス（実体）を生成する必要があります。これらは、フォームデザイナが自動生成するコードで確認することができます。もちろん、わたしたちがコーディングをする必要はありません。

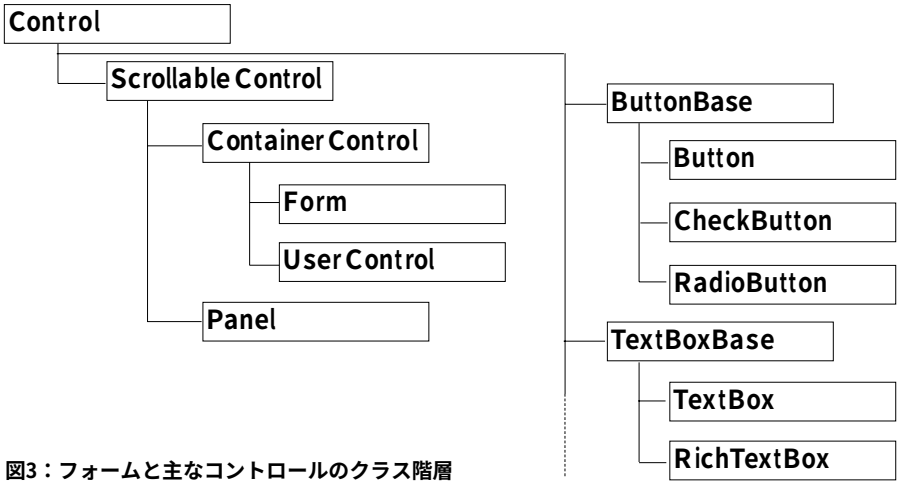


図3：フォームと主なコントロールのクラス階層

フォームのプロパティ、メソッド、イベント

- **主な変更点**
  - **デフォルトボタン、キャンセルボタン**
  - **座標の単位の変更**
  - **グラフィックメソッド**
  - **イベントとイベントプロシージャ**

変更ポイント

プロパティ、メソッド、イベントのいくつかはそのまま利用できますが、変更されているものもあります。ここでは、重要な変更点を紹介します。

デフォルトボタン、キャンセルボタン

デフォルトボタンとキャンセルボタンの設定は、Visual Basic 6.0ではボタン側の設定（DefaultButton、CancelButtonプロパティ）でした。Visual Basic .NETでは、これがフォーム側の設定になりました。フォームのAcceptButtonプロパティに設定されたボタンがデフォルトボタンに、CancelButtonプロパティに設定されたボタンがキャンセルボタンになります。

座標の単位の変更

Visual Basic 6.0では座標の単位（ScaleMode）をセンチ、インチなどが選択でき、デフォルトはTwipでした。Visual Basic .NETでは、座標の単位はPixelだけになります。

グラフィックメソッド

Visual Basic固有フォームには、Circle、CLS、PSet、Line、および Pointなどのグラフィック関連のメソッドがありました。しかし、これに対応するメソッドは、Windowsフォームにはありません。しかし、その代わりにより強力な、新しいGDI+（グラフィックスコマンドのセット）が提供されます（PlusOne参照）。これにより、今までWin32 APIを利用しなければならなかったような、高度なグラフィック機能が実現できるようになります。

また、WindowsフォームはForm.PrintFormメソッドをサポートしていません。

イベントとイベントプロシージャ

フォームの初期化処理のためにNewプロシージャを、また、終了処理のためにDisposeプロシージャを利用します。Finalizeイベントは、ガベージコレクションが行なわれるまで発生しませんが、Disposeプロシージャに記述した処理は、フォームを破棄するとすぐに実行されます。

フォームの構築、表示〜破棄に関連するイベントの対応を表1に示します。

これらのイベントはVisual Basic 6.0に比べ、名前が明確になりました。ただし、その意味やタイミングが多少変更されている場合もあるので確認が必要です。たとえば、Visual Basic 6.0のActivateイベントとDeactivateイベントは、アプリケーション内でのアクティブ／非アクティブの切り替えのときにのみ発生していましたが、Visual Basic .NETのActivatedイベントとDeactivateイベントは、他のアプリケーションから切り替えられたときにも発生します。

DblClickは、DoubleClickイベントに名前が変わっていますが、意味は同じです。

このように、従来のイベントは、ほとんどが利用可能です。ただし、LinkxxxやOLExxxなど、提供されなくなる機能、実現方法が変わる機能などに対応するイベントはVisual Basic .NETには存在しません。

その一方で、新しいイベントがかなり増えているため、今まで以上に詳細な動作設定が可能となります。

Visual Basic 6.0	Visual Basic .NET
Initializeイベント	(Newプロシージャ)
Loadイベント	Loadイベント
Activateイベント	Activatedイベント
Deactivateイベント	Deactivateイベント
QueryUnloadイベント	Closingイベント
Unloadイベント	Closedイベント
Terminateイベント	(Disposeプロシージャ)
	Finalizeイベント

表1：フォーム構築、表示～破棄に関連するイベント

アップグレードウィザードによる変更点

デフォルトボタン、キャンセルボタン

すでにコントロールに設定されているデフォルトボタンとキャンセルボタンの設定は、適切にWindowsフォームのAcceptButtonプロパティおよびCancelButtonプロパティに反映されます。

座標の単位の変更にともなう変更

座標系はTwipがデフォルトであったことが考慮され、フォームの座標の単位をTwipであると仮定して、コントロールなどが配置されます。

グラフィックメソッド

Visual Basic 6.0のグラフィックメソッドと、Visual Basic .NETでのグラフィックのための新しいオブジェクトモデルは大きく異なるため、これらのメソッドはアップグレードされません。

イベントとイベントプロシージャ

対応するイベントプロシージャに適切に置き換えられます。たとえば、ActivateはActivatedイベントプロシージャに置き換えられます。また、対応するイベントがない場合には、そのプロシージャを呼び出すためのコードが追加されるなどして、できる限りのアップグレードを行なってくれます。

たとえば、Initialize、Terminateイベントに対応するものがないため、それぞれ、Form_Initialize_Renamed()、Form_Terminate_Renamed()に置き換えられ、それを呼び出すためのコードがNew、Disposeプロシージャに追加されます。

名前が変更されていない場合でも、イベント発生のタイミングや意味が微妙に変更されているものもあるので、アップグレードコメントにしたがって変更点を確認する必要があります。

今、何をしておくべきか？

グラフィック関連の機能はほとんど利用できなくなるため、今から心がけておくことはありません。ただ、座標単位に関しては、上述したように、アップグレードの際、ScaleModeはデフォルトのTwipであると仮定してピクセル値に変換します。つまり、ScaleModeを変更していない場合には、フォーム上のコントロールが正しいサイズにアップグレードされます。アプリケーション内で、できるだけデフォルトのScaleModeであるTwipを使用しておくと、Visual Basic .NETでの手直しが少なくてすみます。

関連・補足項目

プロパティ、メソッド、イベントは、そのクラスだけでなく、そのクラスの継承元のクラス（これを基本クラスといいます）で定義されています。

紹介したクラス階層を理解しておくと、各クラスの提供する機能の理解に役立ちます。

Plus One　GDI+

GDI（Windows APIが提供する描画機能）の.NETバージョンがGDI+です。GDI+は.NET Frameworkの一部として提供され、強力な描画機能をサポートします。

たとえば、基本的な図形（四角形、楕円、多角形など）や、文字列、ビットマップやカーソルの描画を実現します。また、文字列をビットマップで描画したり、グラデーションを指定して図形を描画したりする「効果」を指定することができます（リスト1・図4）。

ポイントは、常に文字列や図形が描画されるためには、“FormのPaintイベントに描画処理を記述する”ことです。このイベントを利用することで、アイコン化して戻されたときや、別のウィンドウに隠れていたウィンドウが復活したときに適切に再描画されます。

```
Private Sub Form1_Paint(ByVal sender As Object, _
                        ByVal e As System.Windows.Forms.PaintEventArgs) Handles MyBase.Paint
    Dim grph As Graphics = e.Graphics
    ' Dim grph As Graphics = Me.CreateGraphics()                    ' Paintイベント以外の時
    Dim brsh As Brush                                              ' 普通のブラシ
    Dim gbrsh As Drawing.Drawing2D.LinearGradientBrush             ' グラデーションブラシ
    Dim fnt As Font                                                ' フォント
    Dim p1 As New Point(10, 10), p2 As New Point(600, 300)
    ' 描画ツールを準備します
    brsh = New TextureBrush(New Bitmap("c:\winnt\シャボン.bmp"))
    gbrsh = New Drawing2D.LinearGradientBrush(p1, p2, Color.Blue, Color.Red)
    fnt = New Font("Arial Black", 50, Drawing.FontStyle.Bold)
    ' 描画します
    grph.DrawString("Visual Basic.NET", fnt, brsh, 10, 10)
    grph.FillRectangle(gbrsh, 40, 110, 640, 30)
End Sub
```

リスト1：文字列をビットマップで表示し、図形にグラデーションをかける

図4：リスト1の実行結果

ダイアログボックス

・フォームのShowメソッドとShowDialogメソッド
DialogResultプロパティの活用

変更ポイント

Visual Basic 6.0でフォームを表示する場合、Showメソッドを利用していました。そして、引数でvbModeless（モードレスフォーム）またはvbModal（モーダルフォーム）を指定しました。

Visual Basic .NETでもShowメソッドを利用してフォームを表示することができます。しかし、このShowメソッドには引数を指定することができません。常にモードレスになります。モーダルフォームを表示したい場合は、ShowDialogメソッドを利用します。

```
Dim frmSub As New Form2 ()
frmSub.ShowDialog()
```

もうひとつの変更点は、ユーザーが選んだボタンの検出です。Visual Basic 6.0では、ユーザーがダイアログ上でクリックしたボタンを、呼び出し側のフォームで検出するために、フォームレベル変数やグローバル変数などを利用するなど、コーディングが必要でした。

Visual Basic .NETでは、これが改善され呼び出された側のコードは不要になります。ダイアログ上に配置したボタンコントロールのDialogResultプロパティを、OKやCancel、Ignore（無視）などに設定します。これで、ボタンとの関連付けが行なわれます。

この設定をしておくと、そのボタンがクリックされたときに、自動的にダイアログが非表示になり、その設定した値を呼び出し元で参照可能になります。上記のコードに続けて次のコーディングをします。

```
Select Case frmSub.DialogResult
        Case DialogResult.OK
                ' OKのときの処理
        Case DialogResult.Cancel
                ' キャンセルのときの処理
End Select
```

その後、

```
frmSub = Nothing
```

でオブジェクトを破棄します。なお、ボタンとの関連付けは、コーディングでも可能です。

または、ダイアログ側で、

```
Me.DialogResult = DialogResult.Ignore
```

としても、同様に動作します。この例では、Ignoreを設定したことになります。

アップグレードウィザードによる変更点

アップグレードウィザードによって適切にアップグレードされます。Showメソッドでモーダルを指定した場合は、ShowDialogになります。ただし、このアップグレードされたコードは、

- **DialogResultプロパティの機能は利用せずに、ダイアログをコーディングで非表示または、クローズする**
- **冗長なコードが多い（オブジェクトの存在を確認しオブジェクトが存在しないと作成）**

という欠点を持っています。そのためお勧めできません。
アップグレード後に修正されることをお勧めします。

標準コントロール

- ほとんどの標準コントロールは、対応するWindowsコントロールがある
- サポートされなくなるコントロールもある

変更ポイント

Visual Basic 6.0の標準コントロールの多くは、対応するWindowsコントロールがあります。いくつかは、強化／変更されたり、削除されたり、また多くの効果的な新しいコントロールが追加されたりしています。

メニュー

Visual Basic 6.0のときには、Menuコントロールのみがあり、これをメインメニューとショートカットメニュー（コンテキストメニュー）として利用していました。Windowsフォームには、メニューバーを提供するためのMainMenuと、ショートカットメニューを提供するためのContextMenuという2つのメニューコントロールがあります。またメニューの項目を追加すると、MenuItemコントロールが追加され、MainMenuによってコレクションとして管理されます。このMainMenuはフォームにひとつですが、ContextMenuはフォームで複数用意でき、フォームやコントロールなどに関連付けをするとショートカットメニューをコーディングなしでも簡単に実現できます。

メニューの編集も、従来のメニュー独自のメニューエディタではなく、インプレースメニューの機能（図5）によりフォームデザイナ上で視覚的に作成することができるようになりました。

図5：フォームデザイナ上で視覚的にメニューを作成

Timerコントロール

Visual Basic 6.0では、TimerコントロールのIntervalプロパティを「0」に設定することにより、タイマーを無効にすることができました。Visual Basic .NETでは、Intervalプロパティの最小値は「1」になり、タイマーを無効にするためにIntervalプロパティを利用することはできなくなります。Intervalプロパティは、あくまでもIntervalの設定のみに使うことになり、混乱を防ぐことができます。

Frameコントロール

Frameコントロールの代わりに、GroupBoxコントロール（またはPanelコントロール）を利用します。FrameとGroupBoxコントロールは、ほぼ同じ機能で、GroupBoxコントロールには境界線を削除する機能がありません。境界線のないコントロール、つまりコンテナとしての役割のみを果たしGUIを持たない枠組みを利用したいときにPanelコントロールを利用します。

サポートされなくなるコントロール

次のような、グラフィック関係のコントロールは、サポートされなくなります。

- **Shapeコントロール**
- **Lineコントロール**

このほか、OLEコンテナコントロール、Dataコントロール、Imageコントロールもサポートされなくなります。ImageコントロールはPictureコントロールの軽量な代替として利用されていましたが、現在はリソースの節約はそれほど深刻でないため、これからはPictureコントロールを利用することになりました。

コントロール配列

Visual Basic 6.0には、同じ名前で、インデックスにより識別されるコントロール配列がありました。これらは、プロパティは別々に設定でき、イベントプロシージャを共有できるという機能がありました。Visual Basic .NETでは、コントロール配列は利用できなくなります。ただし、コントロール配列の代わりになるコレクションや配列、また強化されたイベントの機能などを利用することができます。

アップグレードウィザードによる変更点

メニュー

Visual Basic 6.0のMenuコントロールをメインメニューとして利用していた場合、MainMenuコントロールにアップグレードされるので、そのまま利用可能です。一方、ContextMenuとしては使用することができないので、作り直しが必要です。

Timerコントロール

Enabledプロパティ、Intervalプロパティなどがそのままアップグレードされます。ただし、Intervalプロパティの最小値が「1」であるため、この値が「0」のときは自動的に「1」に変更されます。つまり、Visual Basic 6.0ではIntervalプロパティが「0」に設定されていたためTimerイベントが発生しなかった場合でも、1ミリ秒ごとにTimerイベントが発生するアプリケーションに変わってしまいます。そのため、このような場合には、アップグレード後に手直しが必要になります。

Frameコントロール

Frameコントロールは、GroupBoxコントロールに置き換わります。もちろん、Frame上に配置されたオプションボタンなどがある場合なども、すべて適切にアップグレードされます。ただし、境界線を「なし」にしていた場合、GroupBoxには対応する機能がないため、Panelコントロールに置き換わります。また、これにはタイトルを表示する機能がないので、必要に応じて修正します。

サポートされなくなるコントロール

サポートされなくなる次のコントロールは、同じ高さを持つ赤色のラベルに変更されます。

- **Shapeコントロール**
- **Lineコントロール**
- **OLEコンテナコントロール**
- **データコントロール**

アップグレードレポートやコメントにしたがって適切な修正が必要です。水平／垂直のLineコントロールは、見た目は水平線／垂直線のように見えますが、実際にはラベルなので適切に修正するとよいでしょう。

また、データコントロールを利用したプロジェクトをアップグレードすると、DAOを利用するための参照設定やコードが追加されるので、必要であれば利用することもできます。不要であれば削除しましょう。

Imageコントロールは、Pictureコントロールにアップグレードされます。

コントロール配列

Visual Basic 6.0のコントロール配列をアップグレードすると、Visual Basic 6.0互換機能を利用して、コントロール配列を利用するプログラムに置き換えられ、従来と同様の動作をします。

今、何をしておくべきか？

Timerコントロール

Timerのイベントを発生させないようにするために、「Intervalプロパティ = 0」ではなく、「Enabledプロパティ = False」を利用します。この本来の使い方をしていれば、ウィザードによるアップグレード後の手直しは不要です。

関連・補足項目

タイマーコントロールのような不可視コントロールや、メニューコントロールは、フォーム上ではなく、フォームデザイナの下部の「不可視コントロール専用領域」（図6）に配置されます。とくにVisual Basic .NETでは、不可視のコントロールが増えるので、別領域に配置されることで、デザイン時に邪魔にならず、効率よく開発ができます。

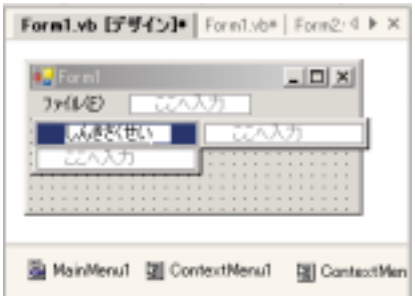


図6：不可視コントロールやメニューコントロールは「不可視コントロール専用領域」に配置される

アップグレードウィザードでアップグレードすると、不可視コントロールは、自動的にフォームデザイナの下部の「不可視コントロール専用領域」に配置されます。

Plus One 新しい便利なコントロールの紹介

さまざまな効果的なコントロールが提供されています。ここではそのいくつかを紹介します。

日付に関連するコントロールは、今までもサードパーティ製あるいはCommonControlとして追加されていましたが、より強力なコントロールがWindowsコントロールとして提供されます。

DateTimePickerコントロール

コンボボックスの右のドロップダウン矢印をクリックすると、カレンダーが表示され、その一覧から日付を選択することができます（図7）。

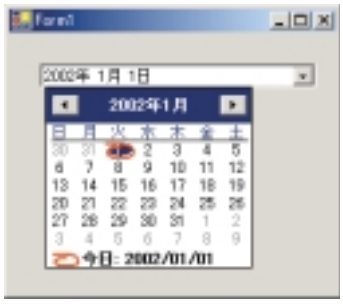


図7：DateTimePickerコントロール

MonthCalendarコントロール

1ケ月分のカレンダーを表示します（図8）。

図8：カレンダーコントロール

プロバイダコントロール

他のコントロールと連携してパワーを発揮するさまざまなコントロールがあります。これらのコントロールをフォームに追加すると、不可視コントロールが表示されるペインに並びます。そして、すべてのコントロールにそのプロバイダコントロールが提供するプロパティが追加されます。

ErrorProviderコントロール

コントロールに関連するエラーを簡単に表示できます。たとえば、入力が必須の項目に、データが未入力のまま［OK］ボタンをクリックした場合、エラーのアイコン（赤いエクスクラメーションマーク）を表示し、そこにマウスと近づけるとエラーがツールヒントとして表示されるというような（図9）、そんなGUIを簡単に実現できます。もちろん、このアイコンを変更することも可能です。

図9：ErrorProviderコントロール

HelpProvider、ToolTipコントロール

HelpProviderコントロールを利用すると、［F1］キーを押したときに、コントロールに関連付けられた状況依存のヘルプを表示することができます。また、ToolTipをフォームに追加すると、各コントロールに「ToolTip1[注1]のToolTip」というプロパティが追加されるので、ここでプロパティにツールヒント文字列を設定します。

注1）ToolTip1はToolTipに付けられるデフォルトの名前です

標準コントロールのプロパティ

- 変更されたプロパティ
- 強化されたプロパティ

変更ポイント

Visual Basic 6.0の標準コントロールの多くは、多くの共通するプロパティを持ちます。ここではこれらのうち、Visual Basic .NETで強化／変更されたものを中心に紹介します。

Textプロパティ

もっとも大きなプロパティの変更は、フォームやラベルコントロールに文字列を表示するためのCaptionプロパティが、すべてTextプロパティに変更・統一されたことでしょう。もう、Captionプロパティは使いません。常にTextプロパティを使います。

Locationプロパティ

位置を指定、参照するLeftとTopプロパティの代わりに、Locationプロパティを利用して同時に指定、参照できます。このLocationプロパティは、X（Leftに相当）とY（Topに相当）をメンバとしてもつ構造体で、GDI+（グラフィックスコマンドのセット）のメソッドで利用されます。なお、LeftとTopプロパティは、（今のところは）コード中で利用できますが、プロパティウィンドウには表示されません。プロパティウィンドウのLocationプロパティの［+］をクリックして展開すると、XとYが参照できます。

Sizeプロパティ

サイズを指定するWidthとHeightも、その代わりにSizeプロパティというWidthとHeightをメンバとして持つ構造体を利用します。Locationと同様、2つの要素を同時に設定でき、GDI+のメソッドで利用されます。そして現在はもともとのプロパティ（Width、Height）をコード中で利用可能ですが、（展開しなければ）プロパティウィンドウに表示されないということも同じです。

AnchorとDockプロパティ

Anchorプロパティを利用すると、配置のときに、上下左右の各方向からの位置を固定することができます。たとえば、上からの位置と右からの位置を固定して指定する（図10）と、フォームがリサイズされたとしても、常に指定したフォームの“辺”からの位置が固定されます（図11）。

また、Dockプロパティを利用すると、コントロールのAlignを指定することができます（図12）。常に、上、下、あるいは右、左の辺にドッキングするコントロールを利用できます（図13）。

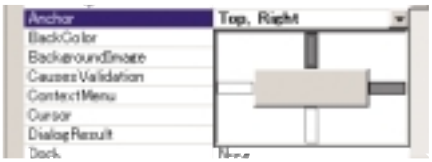


図10：上と右からの位置を固定する

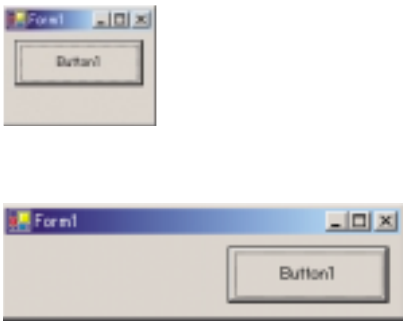


図11：ウィンドウをリサイズしても右上からの位置は変わらない

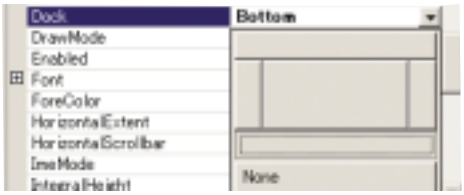


図12：Dockプロパティを利用してコントロールのAlignを指定

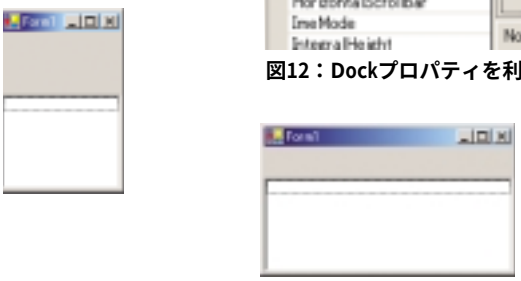


図13：リサイズしても、コントロールは常にぴったり下方に配置

ZOrderプロパティ

コントロールの重なり順を指定するために、ZOrderプロパティを使いますが、これに加え、Visual Basic .NETでは、この重なり順を変更するメソッドが追加されました。BringToFrontメソッドを利用すると最前面、SendToBackメソッドで最背面に表示することができます。

アップグレードウィザードによる変更点

Textプロパティ、Locationプロパティ、Sizeプロパティなどは、それぞれCaptionプロパティ、Left／Topプロパティ、Width／Heightプロパティをもとに、適切にアップグレードされます。Captionプロパティが正しくアップグレードされないパターンなど、利用時の注意事項については、第4章の「プロパティの変更」を参考にしてください。

関連・補足項目

コントロールを配置したときに、最初に自動的に付けられる名前がVisual Basic 6.0のものから変わっているコントロールがあります。たとえば、CommandButtonコントロールは、Buttonコントロールと呼ばれるようになりました。これにともない、デフォルトで付けられる名前が、「Command1」から「Button1」に変わっています。また、TextBoxコントロールのデフォルトで付けられる名前も、「TextBox1」から、「Text1」に変わります。しかし、この変更は、開発においては大きな問題ではないでしょう。

ActiveXコントロール

- **ActiveXコントロールも引き続き利用可能**
  ラッパークラス、参照設定が必要
  オーバーヘッドあり

変更ポイント

ActiveXコントロールは、相互運用機能を提供するラッパークラスがあれば、利用可能です。Visual Basic .NETでは、このラッパークラスが自動生成されるため簡単に利用できます。ツールボックスを右クリックすると表示されるショートカットメニューから［ツールボックスのカスタマイズ］をクリックして表示されるダイアログ（図14）を利用して、ActiveXコントロールをツールボックスに追加します。コントロールをフォーム上に配置すると、必要な参照設定も追加され、簡単に利用できるようになります。

しかし、すでに紹介したように、COMと.NET間の相互運用には、オーバーヘッドが避けられません。できるだけ対応するWindowsコントロール、または.NET Frameworkコンポーネントを使ってください。また、サードパーティ製、あるいは自作のActiveXコントロールも、同じ機能をもつ.NETバージョンのコントロールに置き換えていくことをお勧めします。

自作のActiveXコントロールのアップグレードについては、第3章の「コンポーネント」を参照してください。

アップグレードウィザードによる変更点

ActiveXコントロールを利用したプロジェクトをアップグレードすると、ActiveXコントロールを利用するために必要なラッパークラスが自動的に生成され（図15）、必要な参照設定が追加されます（図16）。そのままActiveXコントロールを利用することも可能ですが、オーバーヘッドを考慮し、対応するコントロールがあれば、修正するとよいでしょう。

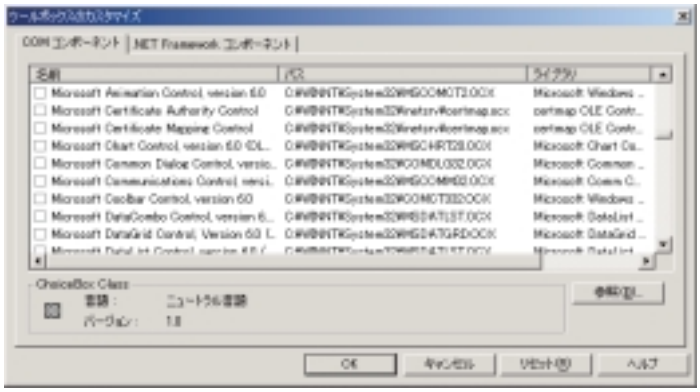


図14：「ツールボックスのカスタマイズ」ダイアログ

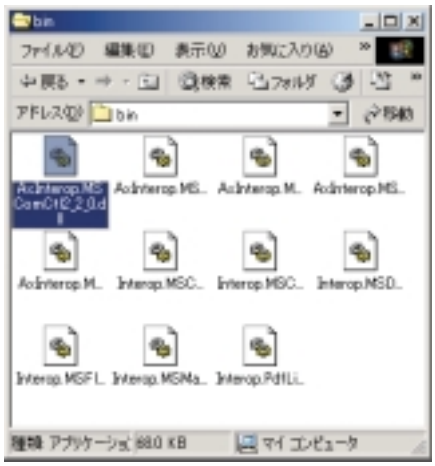


図15：アップグレードウィザードによって自動的にラッパークラスが生成される

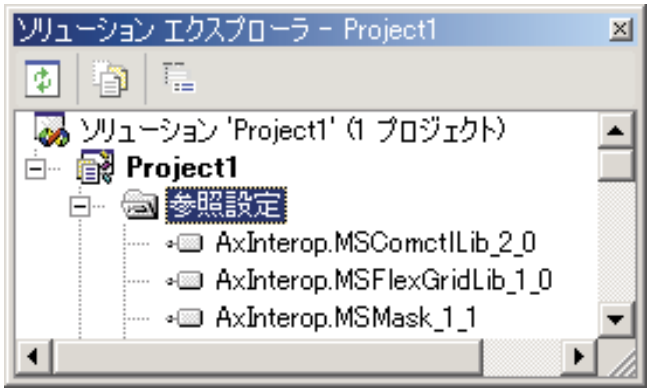


図16：参照設定も自動的に行なわれる

Plus One

Visual Studio .NETによる自動変換以外にも、コマンドラインからAXImp.exe（ActiveX Control Importer）を利用することで、手動で相互運用のためのラッパーを作成することも可能です。

その他の変更

- 特殊オブジェクト
- ドラッグ＆ドロップ
- DDEはサポートされなくなる

変更ポイント

特殊オブジェクト

特殊オブジェクトはなくなります。そして、それに代わるパワーアップされた機能、またはVisual Basic 6.0互換機能が提供されるものを紹介します。

Appオブジェクト

アプリケーションに関する情報は、.NET Frameworkクラスライブラリにある Assemblyクラスが提供します。または、Visual Basic 6.0互換機能（VB6.XXX()）を利用することもできます。

たとえば、App.Pathプロパティで取得していた「実行時のパス名」は、System.Reflection.Assembly.GetExecutingAssembly.Locationプロパティ、または、VB6.GetPath()メソッドで取得できます。互換クラスはオーバーヘッドがあるため、前者がお勧めです。

Debugオブジェクト

対応する機能が.NET Frameworkクラスライブラリにより提供されています。たとえば、Debug.Printメソッドで実現していたデバッグウィンドウへの表示は、System.Diagnostics.Debug.WriteLine()メソッドを利用できます。

Screenオブジェクト

スクリーンに関する情報は、.NET Frameworkクラスライブラリにある Screenクラスが提供しますが、Visual Basic 6.0のScreenオブジェクトと共通するプロパティ、メソッドは多くありません。

Errオブジェクト

Errオブジェクトは引き続き利用可能です。例外処理との関連については、第4章の「エラー処理（例外処理）」を参照してください。

Clipboardオブジェクト

クリップボードとのやり取りは、.NET Frameworkクラスライブラリにある Clipboardクラスが提供します。Visual Basic 6.0のClipboardオブジェクトより多くの機能、多くのクリップボード形式をサポートします。

Printerオブジェクト

プリンタへの出力機能は、.NET Frameworkクラスライブラリにあるprintdocumentクラスが提供します。また、これ以外にもプリンタやページの設定、印刷ダイアログやプレビューの機能を提供するクラスを利用できます。

ドラッグ＆ドロップ

オブジェクトモデルが大きく変更されました。

DDE

サポートされなくなりました。

アップグレードウィザードによる変更点

特殊オブジェクト

特殊オブジェクトは自動アップグレードされるものとされないものがあります。

Appオブジェクト

たとえば、App.ExeNameはVB6.GetExeName()に、App.PathはVB6.GetPath()にそれぞれ変更されます。オーバーヘッドを考慮し、アップグレード後、手動でAssemblyクラスのメソッド、プロパティに修正するとよいでしょう。

Debugオブジェクト

たとえば、Debug.PrintはSystem.Diagnostics.Debug.WriteLine()に変更されます。

Screenオブジェクト

対応するメソッドに置き換えられるものもあります。たとえば、Screen.Heightプロパティは、System.Windows.Forms.Screen.PrimaryScreen.Bounds.Heightプロパティに変更されます。

Errオブジェクト

そのまま利用できるため、変更はありません。

以上のように、対応するメソッドなどがあれば、それに自動アップグレードされます。しかし、ClipboardオブジェクトやPrinterオブジェクトのように、オブジェクトモデルが大きく変更されているオブジェクトについては、自動ではアップグレードされません。

ドラッグ＆ドロップ

オブジェクトモデルが大きく異なるため、Visual Basic 6.0のドラッグ&ドロッププロパティとメソッドはアップグレードされません。

DDE

サポートされないため、すべてコメントアウトされます。

Plus One ドラッグ＆ドロップのサンプル

「本日のランチ」「ドリンク」「ケーキ」をドラッグし、購入リストにドロップすると、リストに追加されるアプリケーションです（図17）。

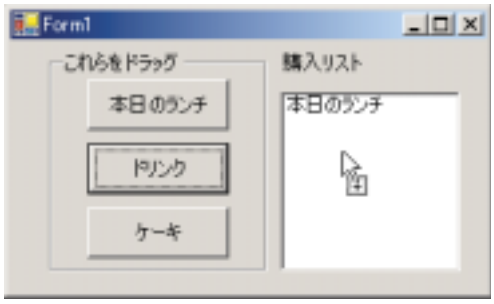


図17：［本日のランチ］ボタンをドラッグし、右側のリスト内にドロップ

サンプルの作り方

①コントロールを配置します。
②プロパティを設定します。ListBoxコントロールのAllowDropプロパティをTrueに設定します。
③コーディングします。リスト2を追加すれば、できあがり！

```
' ボタン上でマウスをダウンするとドラッグ開始 （Button2、3も同様に記述）
Private Sub Button1_MouseDown(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) _
                    Handles Button1.MouseDown
        Button1.DoDragDrop(Button1.Text, DragDropEffects.Copy Or DragDropEffects.Move)
End Sub

' リスト上にドラッグ中のマウスが入ってくると期待したデータ形式どうかをチェック→アイコンを変更
Private Sub ListBox1_DragEnter(ByVal sender As Object, ByVal e As System.Windows.Forms.DragEventArgs) _
                    Handles ListBox1.DragEnter
        If (e.Data.GetDataPresent(DataFormats.Text)) Then
                e.Effect = DragDropEffects.Copy
        Else
                e.Effect = DragDropEffects.None
        End If
 End Sub

' リスト上でドロップしたとき、リストにアイテム（ButtonのText）を追加
Private Sub ListBox1_DragDrop(ByVal sender As Object, ByVal e As System.Windows.Forms.DragEventArgs) _
                    Handles ListBox1.DragDrop
        ListBox1.Items.Add(e.Data.GetData(DataFormats.Text).ToString)
End Sub
```

リスト2：ドラッグ&ドロップサンプルのコード

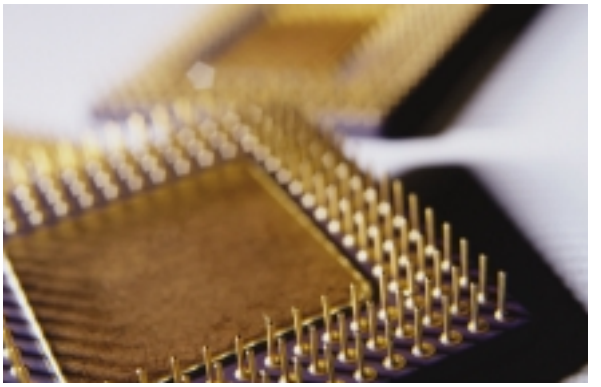
Visual Basic 6.0ユーザーのための Visual Basic .NET移行ガイド ～やっぱりVBが好き～

日本電気株式会社 Eラーニング事業部
山崎明子

● この小冊子の内容は、2002年1月現在のものです。
● 製品内容については予告なく変更することがあります。

製品に関するお問い合わせは次のインフォメーションをご利用ください。
■インターネットホームページ　http://www.microsoft.com/japan/msdn/
■カスタマーインフォメーションセンター　東京　03-5454-2300
大阪　06-6347-9300
※電話番号のおかけ間違いにご注意ください。

Microsoft®
マイクロソフト株式会社
〒151-8533　東京都渋谷区笹塚 1-50-1 笹塚NAビルディング



msdn®
Microsoft® Developer Network

Microsoft®

3008-SE2