Microsoft

Upgrading Visual Basic 6.0 Applications to Visual Basic .NET and Visual Basic 2005

patterns & practices

Microsoft®

Upgrading Visual Basic 6.0 Applications to Visual Basic .NET and Visual Basic 2005

patterns & practices

ISBN 0-7356-2298-1

目次

付録 B

アプリケーション ブロック、フレームワーク、およびその他の開発支援 603

付録 C

ASP のアップグレードに関する概要 623

付録 D

FMStocks 2000 のアップグレード - ケース スタディ 645

序文

Microsoft® Visual Basic® 開発システムは、現在最も広く使用されているプログラミング言語の 1 つです。比較的簡単かつ強力なこの言語は、ビジネス アプリケーションをすばやく効率的に構築するのに最適な言語として、現代の迅速なアプリケーション開発の形成に役立ってきました。

広く使用される製品が皆そうであるように、Visual Basic もまた、コンピュータ テクノロジの世界の変化に適応しながら改良を重ねてきました。そしてその次なる展開となるのが、Visual Basic .NET です。

Visual Basic はバージョンを重ねるたびに、常に変わらない使いやすさと、その時代の最新の手法やテクノロジを提供してきました。しかし、Visual Basic 6.0 がリリースされた後に使用されるようになった新しいテクノロジや手法は、6.0 以前のバージョンの Visual Basic では利用できません。オブジェクト ベースのプログラミングは何年も前から Visual Basic で利用されてきましたが、本当の意味でのオブジェクト指向の機能はありませんでした。分散アプリケーションを開発するための機能も大きく制限されていました。また近年では、Web サービス、モバイル デバイス、.NET などの他のテクノロジの重要性も高まっています。これらの機能やその他の機能のサポートに対するニーズから、Visual Basic の新しいバージョンのリリースが求められていました。

Visual Basic .NET によって Visual Basic プログラミング言語は大きく進化しました。Microsoft .NET Framework の一部となる Visual Basic .NET では、強力で拡張可能なビジネス アプリケーションを構築するための最新テクノロジがすべてサポートされています。オブジェクト指向プログラミングの手法も完全にサポートされるようになり、完全なオブジェクト指向のデザインを実装できます。また、新しいフォーム機能により、機能豊富なデスクトップ アプリケーションをこれまでよりさらに簡単に構築できます。その上、Web フォームによって、Web アプリケーションの作成も同じくらい簡単になっています。

Visual Basic .NET に加えられた変更により、以前のバージョンの一部の機能が使用できなくなりました。新しいバージョンでは、こうした古い機能と同じ結果をより良い方法で実現できますが、こうした変更によって以前のバージョンとの互換性が失われています。このため、以前のバージョンの Visual Basic で作成したアプリケーションを最新のバージョンに自動的にアップグレードすることは不可能です。アップグレードのプロセスを自動化することはできますが、アプリケーション全体を自動的なプロセスだけでアップグレードすることはできません。アップグレード プロセスを完了するためには、手動の作業が必要になります。

このガイドは、アプリケーションを Visual Basic 6.0 から Visual Basic .NET にアップグレードする際に必要な情報を提供することを目的としています。アプリケーションをアップグレードして Visual Basic .NET や .NET Framework の数多くの新機能を活用するために必要なツールや手法を紹介します。

対象読者

このガイドは、Visual Basic 6.0 のアプリケーションやコンポーネントの開発に携わるソフトウェア技術面での意思決定者、ソリューション アーキテクト、およびソフトウェア開発者を対象としています。Visual Basic .NET へのアップグレードに伴う問題やリスクについて理解するのに役立つほか、アップグレードをコスト効果の高い形で成功させるためのアプリケーションの準備の手順や、Visual Basic .NET へのアップグレードが完了した後にアプリケーションを改良するための方法に関するアイデアやアドバイスも紹介しています。

技術面での意思決定者

技術面での意思決定者は、このガイドから、Visual Basic 6.0 アプリケーションを Visual Basic .NET にアップグレードする場合のコストとメリットの分析および評価や、このアップグレードを実証済みのプロセスとツールを使って最適な形で行う方法について、豊富な情報を得ることができます。このガイドでは、.NET にアップグレードするメリットが最も大きいアプリケーションおよびアプリケーションの部分を特定する方法や、アップグレードに伴うリスクやアップグレード プロセスの間に注意深く監視する必要がある問題について学ぶことができます。さらに、アプリケーションのアップグレードが完了した後にアプリケーションに適用できる Visual Basic .NET の改良された機能についても学ぶことができます。

ソリューション アーキテクト

ソリューション アーキテクトはこのガイドで、アップグレード プロジェクトの計画および実行のための最善の方法を学ぶことができます。アップグレード プロジェクトの複雑さやアップグレードの完了に必要な労力を効果的に評価する方法や、アップグレードの完了に多くの手動の処理を必要とする箇所を特定する方法を学べるほか、アップグレード プロジェクトを成功させるためのベスト プラクティスや、それが実際のプロジェクトのタスクやマイルストーンにどのように結び付くのかについての詳細な説明を読むこともできます。

開発者

開発者はこのガイドで、Visual Basic 6.0 の多くの機能に関する情報や、それらを Visual Basic .NET にアップグレードする方法を学ぶことができます。このガイドでは、アップグレードを成功させるための最も効果的な方法を示す段階的なプロセスや、プロセスの一部を自動化するために使用できるツールが紹介されています。また、自動的に変換されない機能に関する情報や、それらの機能を手動で変換するための最善の方法に関する提案事項も含まれています。全体にわたってコード サンプルが提供されているため、自動的にアップグレードされない Visual Basic 6.0 コードや、それを置き換えるために使用できる同等の Visual Basic .NET コードの実例を見ることができます。さらに、アップグレードの完了後にアプリケーションを改良するためのアイデアやアドバイスも含まれています。

前提条件

先にも述べたように、このガイドは、Visual Basic 6.0 アプリケーションを Visual Basic .NET にアップグレードすることを検討している、またはアップグレードすることを既に決定した、ソフトウェア開発者、ソリューション アーキテクト、および技術面での意思決定者を対象としています。このため、ここでは Visual Basic 6.0 についてある程度の知識と経験があることを前提とします。また、.NET Framework、特に Visual Basic .NET についての一般的な知識も必要です。

このガイドは、Visual Basic 6.0、.NET アプリケーション アーキテクチャ、Visual Basic .NET のいずれの入門書でもありません。Visual Basic 6.0 の多くの機能や、Visual Basic .NET および .NET Framework 全般についての基本的な説明も含まれていますが、それらは、これらのテクノロジ間のアップグレードの手法に関する説明をより適切に行うための文脈の中で提供されているにすぎません。これらのテクノロジについての詳しい情報が必要な場合は、各章の終わりにある「詳細情報」を参照してください。

このガイドの使用方法

このガイドのほとんどの章はそれぞれ独立しているため、どのような順序で読んでもかまいません。ただし、第 1 章「はじめに」と第 2 章「アップグレードの成功のためのプラクティス」は順番に読むことをお勧めします。この 2 つの章は、アップグレード プロセスの手順の重要な土台を形成するものであり、このガイドの残りの部分の基礎となります。

このガイドは、システムのデザインと統合を監督するアーキテクチャの専門家から実際のコーディングを行うソフトウェア開発者までさまざまな読者を対象としているため、当然、読者によって各章の重要性が異なります。以下のリストは、各章の要点をまとめたものです。これを参考にして、自分の関心に近いと思われる章を見つけてください。

このガイドは以下の 4 つの部分に分かれています。

- 第 1 ～ 3 章から成る第 1 部「General Upgrade Practices」は、あらゆるアップグレード プロジェクトで成功のために必要となるアップグレード プラクティスに焦点を当てています。Visual Basic 6.0 から Visual Basic .NET へのアップグレードに重点が置かれていますが、大半の情報はどのアップグレード プロジェクトにも適用できます。第 1 部に含まれる章は以下のとおりです。
  - 第 1 章「はじめに」では、Visual Basic .NET にアップグレードする一般的な理由と、アップグレードをコスト効果の高い形で成功させるための方法について検討します。
  - 第 2 章「アップグレードの成功のためのプラクティス」では、計画やアプリケーションの準備からアップグレードを経て .NET でのテストと品質保証に至るアップグレード プロセスの各手順を網羅する、実証済みの方法を紹介します。

- 第 3 章「評価および分析」では、アップグレード プロセスの最初の手順について検討します。ここでは、計画したアップグレード プロジェクトのスコープと複雑さの評価が焦点になります。

- 第 4 ～ 6 章から成る第 2 部「Understanding the Upgrade Process」は、Visual Basic .NET へのコードのアップグレードに関するより技術的な概要となっています。第 2 部に含まれる章は以下のとおりです。
  - 第 4 章「一般的なアプリケーションの種類」では、Visual Basic アプリケーションのさまざまな種類について検討し、それらのアプリケーションのアップグレード方法に関する提案事項、アップグレードをすばやく進める方法、および発生する可能性があるものとして頭に入れておく必要がある障害や問題を紹介します。
  - 第 5 章「Visual Basic のアップグレード プロセス」では、アップグレードを実行するための準備と手順についてより詳しく説明します。この章には、Visual Basic アップグレード ウィザードを使用するための詳細な手順が含まれています。Visual Basic アップグレード ウィザードは、Microsoft Visual Studio® .NET 開発システムのアドインで、アップグレード作業の大半をユーザーに代わって実行します。この章を読むと、アップグレード ウィザードをより効果的に使用して、できるだけ多くのコードを自動的にアップグレードできるようにするには、アプリケーションをどのように準備すればよいのかがわかります。
  - 第 6 章「Visual Basic アップグレード ウィザードについて」では、Visual Basic アップグレード ウィザードの機能と制限について詳しく説明します。

- 第 7 ～ 16 章から成る第 3 部「Manual Upgrade Tasks」では、アップグレード ウィザードで自動的に行うことができず手動で処理しなければならないアップグレードの側面について詳しく説明します。第 3 部に含まれる章は以下のとおりです。
  - 第 7 章「一般的に使用される Visual Basic 6.0 オブジェクトのアップグレード」には、一般的な Visual Basic 6.0 アプリケーションで使用されていて Visual Basic .NET ではサポートされないオブジェクトの多くがまとめられています。これらのオブジェクトを手動でアップグレードするための手法や、アップグレードを正常に完了するための回避策や代替策についての提案事項も含まれています。
  - 第 8 章「一般的に使用される Visual Basic 6.0 言語機能のアップグレード」では、Visual Basic .NET で変更や省略が行われた Visual Basic 6.0 のその他の構成要素を特定します。どのような変更が行われていて、アプリケーションをアップグレードする際にどのように対処すればよいのかを説明します。
  - 第 9 章「Visual Basic 6.0 フォーム機能のアップグレード」では、Visual Basic 6.0 のフォームと、Visual Basic .NET で使用される Windows フォーム パッケージとの違いに焦点を絞ります。フォームに関連するアップグレードの問題点を特定し、それらを克服するための方法を紹介します。
  - 第 10 章「Web アプリケーションのアップグレード」では、Web ベースの Visual Basic 6.0 アプリケーションをアップグレードする方法について説明します。
  - 第 11 章「文字列操作とファイル操作のアップグレード」では、文字列とファイルの操作に特化した情報を提供します。アップグレード ウィザードを適用した場合に自動的にアップグレードされる種類の操作と、Visual Basic .NET で利用できる新しい操作が含まれています。

- 第 12 章「データ アクセスのアップグレード」では、データ アクセス オブジェクト (DAO)、リモート データ オブジェクト (RDO)、および ActiveX データ オブジェクト (ADO) に基づくデータ アクセスのアップグレードについて検討します。アップグレード ウィザードでは処理されないデータベース アクセス コードをすばやくアップグレードするための手法を紹介します。
- 第 13 章「Windows API の使用」では、Windows API 関数呼び出しを使用する Visual Basic 6.0 アプリケーションをアップグレードする際に発生する一般的な問題を取り上げます。一般的な Windows API 関数呼び出しを Visual Basic .NET の新しいメソッドに置き換える方法を説明します。
- 第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」では、この 2 つのバージョンの言語の相互運用のための選択肢を紹介します。Visual Basic .NET アセンブリから Visual Basic 6.0 コンポーネントへのアクセスや、Visual Basic 6.0 で構築したクライアントから .NET アセンブリへのアクセスの手法について検討します。コンポーネント オブジェクト モデル (COM) を使用してこの 2 つの言語を相互運用する場合に発生するさまざまな問題や、アンマネージ環境とマネージ環境での COM の動作のしくみもカバーされています。
- 第 15 章「MTS アプリケーションと COM+ アプリケーションのアップグレード」では、Microsoft Transaction Services (MTS) と COM+ のコンポーネントをアップグレードする方法を取り上げます。また、使われなくなった機能の一部を紹介し、アプリケーションをアップグレードする際にどのように対処すればよいかを説明します。
- 第 16 章「アプリケーションの完成」では、アプリケーションの機能の中心的な要素ではないが、ユーザーに完全なアプリケーションを提供する製品を仕上げるためには欠かせない、アプリケーションのその他の要素を取り上げます。これには、統合ヘルプや製品の配置のアップグレードなどのトピックが含まれます。

- 第 17 ～ 21 章から成る第 4 部「Beyond the Upgrade」では、アップグレードの完了後の作業を取り上げます。第 4 部に含まれる章は以下のとおりです。
  - 第 17 章「アプリケーションの改良の概要」では、Visual Basic .NET でアプリケーションの同等の機能が実現された後に使用できる改良の選択肢について検討します。アプリケーションのアーキテクチャ、デザイン、および実装を改良するための提案事項も含まれています。
  - 第 18 章「一般的なアプリケーションの改良シナリオ」では、フォーム アプリケーションやエンタープライズ サービスなど、個々のアプリケーションの種類に応じた改良の提案事項を提供します。
  - 第 19 章「一般的な Web シナリオのための改良」は第 18 章の続きで、Web ベースのアプリケーションの改良に焦点を当てています。

- 第 20 章「一般的なテクノロジ シナリオ」は、ほとんどすべての種類のアプリケーションに適用できる改良に焦点を当てています。これらの改良には、アプリケーションのセキュリティ、管理性、パフォーマンス、および拡張性や、通信と状態の管理の強化などが含まれます。
- 第 21 章「アップグレード後のアプリケーションのテスト」では、アプリケーションのアップグレード後や必要な改良の実装後に、アプリケーションの正確さや完全さを検証するために使用できる方法について詳しく説明します。

技術面での意思決定者に関連のある章

技術面での意思決定者は、主に、評価と分析、アップグレードのリスクと課題、およびアプリケーションの改良に関する情報に関心があることでしょう。これらの情報は以下の章に含まれています。

- 第1章「はじめに」
- 第3章「評価および分析」
- 第7章「一般的に使用される Visual Basic 6.0 オブジェクトのアップグレード」
- 第8章「一般的に使用される Visual Basic 6.0 言語機能のアップグレード」
- 第9章「Visual Basic 6.0 フォーム機能のアップグレード」
- 第10章「Web アプリケーションのアップグレード」
- 第11章「文字列操作とファイル操作のアップグレード」
- 第12章「データ アクセスのアップグレード」
- 第17章「アプリケーションの改良の概要」
- 第18章「一般的なアプリケーションの改良シナリオ」
- 第19章「一般的な Web シナリオのための改良」
- 第20章「一般的なテクノロジ シナリオ」

ソリューション アーキテクトに関連のある章

ソリューション アーキテクトは、主に、推奨されるアップグレード プロセス、評価と分析、アップグレード プロジェクトを管理するためのベスト プラクティス、このプロセスに関連するリスク、およびアプリケーションの改良に関する情報に関心があることでしょう。これらの情報は以下の章に含まれています。

- 第2章「アップグレードの成功のためのプラクティス」
- 第3章「評価および分析」
- 第4章「一般的なアプリケーションの種類」
- 第7章「一般的に使用される Visual Basic 6.0 オブジェクトのアップグレード」
- 第8章「一般的に使用される Visual Basic 6.0 言語機能のアップグレード」
- 第9章「Visual Basic 6.0 フォーム機能のアップグレード」

- 第 10 章「Web アプリケーションのアップグレード」
- 第 11 章「文字列操作とファイル操作のアップグレード」
- 第 12 章「データアクセスのアップグレード」
- 第 13 章「Windows API の使用 I」
- 第 17 章「アプリケーションの改良の概要」
- 第 18 章「一般的なアプリケーションの改良シナリオ」
- 第 19 章「一般的な Web シナリオのための改良」
- 第 20 章「一般的なテクノロジシナリオ」

ソフトウェア開発者に関連のある章

ソフトウェア開発者は、主に、Visual Basic 6.0 の技術的な機能が Visual Basic .NET のどの機能に当てはまるのかを扱う資料に関心があることでしょう。ここには、既存のツールによって自動的に変換される機能と、手動でアップグレードしなければならない機能が含まれます。また、開発者は、.NET Framework の機能を使ってアプリケーションを改良する方法にも関心があると考えられます。これらの情報は以下の章に含まれています。

- 第 4 章「一般的なアプリケーションの種類」
- 第 5 章「Visual Basic のアップグレードプロセス」
- 第 6 章「Visual Basic アップグレードウィザードについて」
- 第 7 章「一般的に使用される Visual Basic 6.0 オブジェクトのアップグレード」
- 第 8 章「一般的に使用される Visual Basic 6.0 言語機能のアップグレード」
- 第 9 章「Visual Basic 6.0 フォーム機能のアップグレード」
- 第 10 章「Web アプリケーションのアップグレード」
- 第 11 章「文字列操作とファイル操作のアップグレード」
- 第 12 章「データアクセスのアップグレード」
- 第 13 章「Windows API の使用」
- 第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」
- 第 17 章「アプリケーションの改良の概要」
- 第 18 章「一般的なアプリケーションの改良シナリオ」
- 第 19 章「一般的な Web シナリオのための改良」
- 第 20 章「一般的なテクノロジシナリオ」

表記規則

このガイドでは、表 1 に示す表記規則と用語を使用します。

表 1: 表記規則

要素	意味
太字	コマンドやスイッチなど、そのままの形で入力する文字は太字で表記されています。また、メソッド、関数、データ型、データ構造などのプログラミング要素も太字で表記されます (ただし、コード サンプル内では固定幅フォントになります)。これ以外には、ユーザー インターフェイス要素も太字です。
斜体	具体的な値を指定する必要がある変数です。たとえば、*Filename.ext* は、それぞれの場合に有効な任意のファイル名を表します。これ以外に、新しい用語も初出時に斜体で表記されます。
`Monospace font` (固定幅フォント)	コード サンプルです。
%SystemRoot%	Windows オペレーティング システムがインストールされているフォルダです。

フィードバックとサポート

このガイドの内容の正確さには細心の注意を払っています。サンプル コードとプロシージャはすべて個別にテストされています。また、ツール、仕様、および標準の参照と説明はすべてチェックされており、このガイドが公開された時点で正確であると考えられるものになっています。

このガイドについてのご感想およびご意見、内容の誤りについては、vbmigfb@microsoft.com まで電子メールでご連絡ください。

メモ： GotDotNet にはこのガイドのコミュニティ サイトがあります。このサイトには、このガイドで取り上げられている Visual Basic Upgrade Assessment Tool を入手できる「Downloads」セクションや、フィードバック、質問、意見などを投稿できるフォーラムもあります。

主な執筆者

このガイドの執筆と付属の Visual Basic 6.0 Upgrade Assessment Tool の開発は、ArtinSoft の以下のスタッフによって行われました。

- プロジェクトリーダー：Federico Zoufaly
- ガイド：César Muñoz 、Paul Dermody、Manfred Dahmen、Ronny Vargas、Hendel Valverde、José David Araya、Oscar Calvo、Allan Cantillo、Alvaro Rivera、Christian Saborío、Juan Fernando Peña、Xavier Morera、Iván Sanabria
- 評価ツール開発：Rolando Méndez

協力者

Visual Basic 6.0 Upgrade Assessment Tool の開発と『Upgrading Visual Basic 6.0 to Visual Basic .NET and Visual Basic 2005』の執筆は、以下の方々の協力の下に行われました。

- プログラムマネージャ：William Loeffler (Microsoft Corporation)
- プロダクトマネージャ：Eugenio Pace (Microsoft Corporation)
- アーキテクト：Keith Pleas (Guided Design)
- テスト：Edward Lafferty (Microsoft Corporation)、Ashish Babbar、Terrence Cyril J.、Manish Duggal、Chaitanya Bijwe、Arumugam Subramaniyam、Umashankar Murugesan、Dhanaraj Subbian、Tarin R. Shah、Dipika Khanna、Gayatri Patil、Sandesh Pandurang Ambekar (以上 Infosys Technologies Ltd)
- ドキュメントとサンプル：RoAnn Corbisier (Microsoft Corporation)、Tina Burden McGrayne、Melissa Seymour (以上 TinaTech Inc.)、Sharon Smith (Linda Werner & Associates Inc.)、Francisco Fernandez、Paul Henry (以上 Wadeware LLC)

謝辞

このガイドの執筆では、以下の方々から多大な支援をいただきました。ここに感謝いたします。

- Dan Appleman (Desaware Inc.)
- Joe Binder (Microsoft Corporation)
- Rob Copeland (Microsoft Corporation)
- Jackie Goldstein (Renaissance Computer Systems Ltd.)
- Ed Hickey (Microsoft Corporation)
- Billy Hollis
- Edward Jezierski (Microsoft Corporation)
- Chris Kinsman (Vergent Software)
- Deborah Kurata (InStep Technologies)
- Julia Lerman (The Data Farm)
- Rockford Lhotka (Magenic Technologies)
- Christian Nielsen (Volvo Information Technology AB)
- Jay Roxe (Microsoft Corporation)
- Jay Schmelzer (Microsoft Corporation)
- Scott Swigart (Swigart Consulting)
- Visual Basic の MVP (Microsoft Valued Professionals) の方々

詳細情報

Visual Basic 6.0 Upgrade Assessment Tool の詳細およびダウンロードについては、GotDotNet の「Visual Basic 6 to Visual Basic .NET Migration Guide」を参照してください。
URL は http://www.gotdotnet.com/codegallery/codegallery.aspx?id=07c69750-9b49-4783-b0fc-94710433a66d です。

ArtinSoft の詳細については、ArtinSoft の Web サイトを参照してください。
URL は http://www.artinsoft.com です。

1

はじめに

アップグレード プロジェクトに時間、資金、労力を投資する前に、なぜアップグレードが有益なのかを理解することが大切です。また、コストやリスクを最小にするためには、考えられるアップグレード方法を理解することも同じように重要です。この章では、Visual Basic 6.0 アプリケーションを Visual Basic .NET にアップグレードすることの利点について説明し、アプリケーションをアップグレードするために適用できるいくつかの方法を示します。

アップグレード プロジェクトを検討する理由

アプリケーションを Visual Basic 6.0 から Visual Basic .NET にアップグレードするかどうかを決める前に、なぜアップグレードを検討すべきなのかを理解することが重要です。ここでは、アプリケーションをアップグレードする意味と、それに適した時期について説明します。

通常は、Visual Basic 6.0 から Visual Basic .NET にアップグレードしなければならないビジネス要件があります。アプリケーションを Visual Basic .NET にアップグレードする一般的な理由は以下のとおりです。

- アプリケーションを Web 対応にしたり、トレース、柔軟な状態管理、スケーラブルなデータ アクセス、パフォーマンスの向上などの ASP.NET の機能を使用して既存の Web 対応のアプリケーションを拡張したりするため。
- Visual Basic .NET、.NET Framework、および Visual Studio .NET による開発者の生産性の向上の恩恵を受け、強化された開発機能を利用するため。特に、Web サービスのように Visual Basic .NET でより簡単に実現できる新機能が必要な場合など。
- 企業のソフトウェア資産を統合するため。たとえば、新しいアプリケーションが Visual Basic .NET でビルドされている場合、以前のバージョンの Visual Basic でビルドされている他のアプリケーションをアップグレードしなくてはならない場合があります。これによりシステムの統合が強化され、さまざまなプラットフォームに対して専門家を準備しておく必要がなくなります。

- 進行中のビジネス活動のコストを削減するため。たとえば、Visual Basic .NET にアップグレードしたアプリケーションは、拡張性とパフォーマンスが向上し、開発者の生産性が向上するため、通常のビジネス活動のコストを削減できます。
- 以下のような状況でアプリケーションのメンテナンス性を高めるため。
  - アプリケーションの専門家が社内にいない場合。
  - スタッフの異動が頻繁に行われる場合。
  - アプリケーションをサポートするための十分なリソースがない場合。
  - 入手できるドキュメントが限られていたり、最新のものではない場合。

当然、これらの理由の組み合わせがアップグレードの判断に影響を与えます。これらの理由は、*プッシュ*要因 (Visual Basic アプリケーションを新しい環境に合わせてアップグレードしなければならない要因) から*プル*要因 (ビジネスの成長の機会を捉え、新しい顧客をつかみ、製品やサービスを拡充しようとすることから生じる要因) までの、一種のスペクトルと考えることができます。

判断のために必要な最低限の情報

アプリケーションをアップグレードすることが適切かどうかを判断するには、プロジェクトの実現可能性分析を行う必要があります。プロジェクトの実現可能性分析の最初のステップは、Visual Basic 6.0 アプリケーションの現在の状態に関する情報を収集することです。アプリケーションを Visual Basic .NET にアップグレードするかどうかに関して最善の判断を下すためには、分析で以下の情報を扱う必要があります。

- **プロジェクトの目的と優先順位。** プロジェクトの目的が明確にドキュメント化されており、各目的に優先順位が割り当てられていることを確認します。アップグレードによってこれらの目的がどのように満たされるかを明確にします。アップグレードの判断を下す前に、以下の質問に対して答えを見つけておく必要があります。
  - 予想されるアプリケーションの寿命はどのくらいか。一時的なソリューションなのか、長期的なソリューションなのか。ビジネス上のニーズやプロセスは変化し続けているため、予想される寿命が短いアプリケーションは、多くの場合アップグレードの価値がありません。しかし、ビジネスの中核となる機能を持ち、アップグレード プロジェクトの期間よりも十分に長い間その役割を果たし続けるアプリケーションは、アップグレードの候補となります。
  - アップグレードしたアプリケーションが実稼動環境で必要になるのはいつか。アップグレード プロジェクトが、スケジュールで許される期間よりも長くかかる場合は、アプリケーションを置き換える方が適している場合があります。
  - パフォーマンス、セキュリティ、拡張性、その他の潜在的な利点に対して現実的な期待をしているか。また、これらの強化を実現するためには、追加の変更が必要になる場合があることを理解しているか。アップグレード プロセスだけでは、アプリケーションのパフォーマンスや拡張性が向上しない場合があることを理解することが重要です。このような向上は、第 20 章「一般的なテクノロジシナリオ」で説明するアプリケーションの改良の中で実現される場合がよくあります。

- このアプリケーションが、存続期間中にどれだけ多くのユーザーをサポートするか。また、増大するユーザーの要求を満たすために Visual Basic .NET で何が支援されるかを明確に理解しているか。大規模なユーザー ベースを持つアプリケーションを変更すると、これらのユーザーに大きな影響を与えます。しかし、良い方向の変更であれば、肯定的な影響を与えます。Visual Basic .NET のパフォーマンス機能強化や最新のテクノロジにより、ユーザーのアプリケーション エクスペリエンスが改善されます。
- アプリケーションに、他のシステムとの統合や Web からのアクセスなど、新しい機能を追加しようと考えているか。これらの強化もアプリケーション改良の一部であり、アプリケーションをアップグレードする以上の作業が必要になります。

- **アプリケーションのビジネス バリュー。**このアプリケーションで提供される機能は特有なものか、または同じ作業を行うために使用できるサードパーティ製アプリケーションがあるか。アップグレードを計画するアプリケーションが特殊な性質を持ったものであることを実証できる必要があります。

  関連する質問は、ビジネス ニーズに対してアプリケーションが適切であるかという質問です。アプリケーションが、ビジネス ニーズに合った機能を確実に提供していることを実証する必要があります。アプリケーションが長期間使用されていた場合は、アプリケーションの機能が古くなっておらず、ビジネスで必要なプロセスや運用に適していることを確認する必要があります。

- **開発環境。**Visual Basic .NET に移行することは、開発環境を更新することにもなります。開発者は新しい統合開発環境 (IDE) や新しい言語を学ぶ必要があります。また、アップグレードする必要があるツールのソフトウェア ライセンスを購入し、アップグレードしたコードをメンテナンスする必要もあります。

- **開発スキル。**開発者が、.NET Framework 全般と、特に Visual Basic .NET についての知識や経験を持っているか。アプリケーションをアップグレードするのは、新しい言語を学ぶ良い方法ですが、アップグレード プロジェクトを開始する前に、開発者が .NET Framework と Visual Basic .NET についての基本的な知識を得るようにする必要があります。

- **品質保証環境。**アップグレード プロジェクトによって、品質保証手順が影響を受けます (特に自動化されている場合)。テスト環境をアップグレードするために必要な作業についても必ず検討してください。

- **アプリケーションのアーキテクチャと複雑さ。**大企業向けのほとんどのアプリケーションは、企業内の他の複数のシステムに依存しており、サードパーティ ベンダのシステムやライブラリにも依存しています。これらの依存関係の性質を理解し、アップグレードしたアプリケーションで動作させる前に、各システムで必要な変更について必ず検討してください。

また、アップグレードしたアプリケーションがそれに依存する他のアプリケーションに与える影響についても検討してください。Visual Basic .NET に移行することで、そのようなアプリケーションにどのような影響があるかを検討してください。

最後に、アプリケーションの品質は良好か。良好な品質とは、ビジネスの道具としての品質と、それを実装するソース コードの品質の両方を指します。ここでは、ユーザーと開発者から見たアプリケーションの実際の価値を評価します。ユーザーは、アプリケーションが安定しており、作業を正しく行い、使いやすいということを求めています。開発者は、コードが理解しやすく、正確にドキュメント化されており、必要以上に複雑でないアプリケーションを求めています。

以下の 3 つの評価を行うと、アプリケーションのアップグレードに関する判断に必要な情報を収集するのに役立ちます。また、適切なアップグレード プロセスを選択して実施するために必要な作業量を正確に見積もる際にも役立ちます。

- ビジネス バリューの評価
- コードの品質の評価
- 開発環境の評価

ビジネス バリューの評価

アップグレード プロジェクトに時間と資金を投資することを検討している場合は、ビジネスにとっての既存のアプリケーションの機能の重要性を評価することが大切です。アプリケーションをアップグレードした場合、投資に対する利益を得ることができるでしょうか。これは、これまでそのアプリケーションにどれだけ投資してきたかを算出することで示すことができます。また、アプリケーション固有の機能のうち、サードパーティの既製の製品で置き換えることができない機能を明確にします。以下の質問に答えることで、ビジネスにとってのアプリケーションの価値を判断するのに役立ちます。

- アプリケーションの基本的な入力と出力は、従来のファイル形式に基づいているか。簡単に変換できない従来のファイル形式に基づく入出力を使用しているアプリケーションは、アップグレード時に問題となる場合があります。一方、アプリケーションを Visual Basic .NET に移行し、XML などのより一般的な形式にデータをアップグレードすることで、アプリケーションは将来の拡張やテクノロジの変化に強くなります。
- アプリケーションはどのような種類のデータを処理するのか。また、同じジョブを実行できるアプリケーションが他にあるか。重要なビジネス データの処理を行うアプリケーションは、ビジネスにとってきわめて高い価値があります。サードパーティの既製の製品などに、このデータを処理できるアプリケーションがない場合は、アプリケーションを Visual Basic .NET にアップグレードすることでアプリケーションの寿命を延ばすことは論理的なステップです。
- このアプリケーションがユーザーとどのように対話し、ユーザーは人間とコンピュータ ベースのクライアントのどちらなのか。人間のユーザーにとって、整理されたわかりやすいインターフェイスは、アプリケーションの価値を高めます。Visual Basic .NET など、新しい言語バージョンの新しいまたは改良さ

れたユーザー インターフェイス機能により、ユーザーの評価が上がることがあります。同様に、他のアプリケーションと協調して動作するアプリケーションでは、データ転送方法が明確である必要があります。信頼できるデータ形式と通信プロトコルを使用する必要があります。多くの場合、新しい言語では、改良された転送プロトコルやデータ形式をサポートしており、アプリケーション間の通信が簡単で信頼性が向上します。

- アプリケーションに依存している他のアプリケーションがあるか。また、同じデータや機能を提供するサードパーティ製品があるか。他のシステムがそのアプリケーションに依存しており、アプリケーションを置き換えるサードパーティ製品がない場合は、アプリケーションの価値は高くなります。この場合は、アプリケーションの将来を保証することがより重要になります。
- 他では再現することが難しい、アプリケーション固有のビジネス ルールはどれか。他のアプリケーションでは利用できない多数のビジネス ルールを持つアプリケーションは、企業にとって非常に価値が高くなります。
- 企業の現在のビジネス プロセスの中で、このアプリケーションはどのような役割を果たしているか。アプリケーションがなくなったらこれらのプロセスはどうなるか。少数の平凡なタスクを自動化するアプリケーションは、多くの場合それほど価値が高くありません。これに対し、大量のビジネス データを処理するアプリケーションは、非常に価値が高くなります。データ指向のタスクを大量に自動化することで従業員を支援することを意図したアプリケーションでも、そのアプリケーションがユーザーの効率を向上させるなら、価値が高くなります。

アプリケーションをアップグレードするもう 1 つの理由が、総所有コスト (TCO) の削減です。.NET Framework で提供されている多くの機能により、開発者の生産性が向上すると共に、配置とメンテナンスのコストが下がるため、TCO が削減されます。

コードの品質の評価

アプリケーションをアップグレードするかどうかを判断する際に重要な要因の 1 つが、現在のコードのデザインと実装の品質です。アプリケーションのソース コードを詳細に分析することで、コードの品質、デザインの複雑さ、コンポーネントの依存関係についての情報を得ることができます。Visual Basic Upgrade Assessment Tool を使用すると、これらの情報のいくつかを得ることができます。このツールの詳細については、第 3 章「評価および分析」を参照してください。Visual Basic Upgrade Assessment Tool は、このガイドに付属の CD-ROM で提供されています。GotDotNet コミュニティ サイトからダウンロードすることもできます。

この評価ツールは、アプリケーションをアップグレードするために必要な作業量を見積もるのに役立つように作成されていますが、アプリケーションの構造的な複雑さとデザインの品質を理解するのにも役立ちます。ここでは、デザインの品質とは、アプリケーションで必要とされる機能が指定されたときの、コードの実装の適合性を表します。たとえば、何年もの間、異なるプログラマによって何度も変更されたアプリケーションでは、依存関係が過剰に冗長で、使用されていないコードが含まれていることがよくあります。これは、デザインの品質が悪いことを示します。

評価ツールは以下のメトリクスを収集します。

- **サイズのメトリクス。** これには、コードの行数、フォーム、デザイナ、モジュール、クラス、コンポーネント、ユーザー コントロール、データ ソースの数などが含まれます。
- **使用法のメトリクス。** これには、関数、型、プロパティが含まれます。
- **複雑さのメトリクス。** 評価ツールは、アップグレード ウィザードがエラー、警告、および問題 (EWI) を生成するのと同じ内容を認識します。これらの EWI は、それを解決するために必要な具体的な変更を暗に意味しています。評価ツールはこれらの EWI に複雑さを割り当てます。この情報は、報告されたすべての問題に対処するために実行する必要がある作業を見積もるのに役立ちます。
- **依存関係ダイアグラム。** 評価ツールは、各関数とその結果生じるモジュールとファイル間の依存関係のコール グラフを生成します。また、足りないライブラリやファイルの一覧を出力します。この情報を使用して、可能なアップグレード ロードマップ、つまりモジュールをアップグレードする順序を決定してください。

アプリケーションの種類や、それらを構築するために使用できるテクノロジにはさまざまなものがあります。Visual Basic アップグレード ウィザードを使用してアプリケーションをアップグレードすることを検討している場合は、ツールのいくつかの制限事項に注意してください。以下に、アップグレード ウィザードでは一部しかサポートされていないか、まったくサポートされていないアプリケーションや、Visual Basic .NET でサポートされていない機能を使用しているアプリケーションの例を示します。

- COM を介して通信する、いくつもの異なる層のオブジェクトを使用した複雑な分散アプリケーション。
- Visual Basic 6.0 に移行していない Visual Basic 5.0 以前のアプリケーション。
- Microsoft ActiveX® テクノロジ、Web クラス、DHTML を使用したインターネット プロジェクト。これらの開発テクノロジは、どれも Visual Basic .NET ではサポートされていません。
- Visual Basic .NET でサポートされていない Data Environment デザイナに基づくデータベース プロジェクト。
- データ連結を使用してデータソースをコントロールにバインドしているデータベースプロジェクト。
- ActiveX コントロールまたは ActiveX DLL プロジェクト。
- ユーザー コントロールを使用したデータベース クライアント アプリケーション。

これらの機能とアプリケーションは、Visual Basic .NET では直接サポートされませんが、Visual Basic .NET で同じ機能を実現する代替手段 (より優れた手段であることが多い) が必ず存在します。アップグレード ウィザードで自動的にアップグレードされない機能の完全な一覧については、第 7 章から第 11 章を参照してください。

アプリケーションがこれらのシナリオのいずれかに一致する場合は、プロジェクト計画の中に、関係するテクノロジの代替アップグレード手段を明確にして分析するための時間を見込む必要があります。また、ターゲット アーキテクチャをレビュー、デザイン変更、理解するための時間も割り当てる必要があります。

コードの品質は、元のアプリケーションに対する開発作業の状態にも影響を受けます。開発イニシアチブが進行中の場合は、コードは安定していません。コードにはバグ、不完全な機能、またはコンパイルできない無効なコードさえも含まれている可能性があります。

アップグレードに最良の時期は、アプリケーションのコード ベースが安定しており、機能拡張の必要に迫られているときです。この方法では、アップグレード作業と機能拡張の機会を組み合わせることができます。これに対し、アプリケーションに多数の変更を加えている最中は、非常に厳密に管理されたコード管理プロセスを採用しているのでない限り、できるだけアップグレードを避けるべきです。大規模なアプリケーションをアップグレードするには、数週間または数か月かかることがあります。この間、元のコード ベースに対して行われた変更は、アップグレードしたコード ベースにも反映させる必要があります。変化している 2 つのコード ベースのメンテナンスを行うと、エラーが起きやすく、2 つのバージョンのアプリケーション間に不一致が生じる可能性が高くなります。アップグレード中に元のアプリケーションを変更する必要性が生じた場合は、アップグレード対象のモジュールとして安定したものを選び、徐々にアプリケーション全体をカバーすることで、中断を最小化することをお勧めします。

コードの品質は、アップグレードの判断にさまざまな影響を与えます。適切にデザインされ、実装され、テストされ、ドキュメント化されたコード ベースを持つ安定したアプリケーションで、アプリケーションの変更やビジネスの変化によりビジネス ニーズからそれてしまったアプリケーションが、最も優先順位の高いアップグレード候補となります。一方、ビジネス ニーズを満たしており、改良が不要な安定したアプリケーションは、そのままの形にしておくのが適しています。品質が低く、デザインが貧弱で、実装が複雑な、十分にドキュメント化されていないアプリケーションも、アップグレードの有力な候補となります。品質の低いアプリケーションを Visual Basic .NET などの新しい開発言語にアップグレードすることで、これらの欠点を修正する機会が与えられます。これによりメンテナンスが容易になり、アプリケーションが改良される可能性もあります。

開発環境の評価

開発環境を正確に評価するには、開発チームのスキルを考慮する必要があります。アップグレード プロジェクトにおける開発者の生産性は、既存のソース コードや移行先のテクノロジに対する知識を開発者がどれだけ持っているかに依存します。チームの現在のスキルで、アップグレード分析中に見つかった各問題を解決できるかどうかを評価することが必要です。一般に、アップグレード プロジェクトを開始する前に、開発者は Visual Basic .NET のトレーニングをすべて受けることをお勧めします。

また、テスト ケースがあるかどうか、開発チームがアップグレード対象のアプリケーションをどれだけ詳しく理解しているかなど、他の要因も考慮する必要があります。アプリケーションのテスト ケースがきちんと定義されていない場合は、アップグレードしたアプリケーションが完成したことを証明するのが難しくなります。テスト ケースがあ

る場合は、それも Visual Basic .NET にアップグレードする必要があります。これは、自動化されたツールでテストプロセスを実行する場合に特に当てはまります。

開発者にアプリケーションに対する知識がない場合は、開発者がアプリケーションのデザインや実装を理解するために追加の時間を見込む必要があります。アプリケーションに十分なドキュメントが揃っていない場合も、必要な時間が増えます。

アップグレード、再利用、書き換え、置き換えのどれを選ぶべきか

アプリケーションを更新する必要がある場合、大別してアップグレード、再利用、書き換え、置き換えの 4 つの選択肢があります。これらの選択肢について以下で説明します。

- **アップグレード。** アップグレード プロセスを適用し、必要に応じて機能を追加したり、ビジネスの範囲を広げます。アップグレードの利点は、以前の投資の大部分が (特にビジネス ロジックの領域で) 活用できることです。また、アプリケーションを書き換えるのに比べて、バグが混入する可能性が低くなります。特に自動アップグレード プロセスを使用した場合に顕著になります。
- **再利用。** アプリケーションを更新する代わりに、アプリケーションの上に層を追加して、組織内の新しい Visual Basic .NET システムと相互運用できるようにすることが可能です。しかし、統合を目的としてアプリケーションを更新する場合は、Visual Basic 6.0 システムと Visual Basic .NET の両方のシステムをメンテナンスする必要が出てくるため、再利用は有効ではありません。
- **書き換え。** 最初から完全に書き換えること以外にアプリケーションを更新する方法がない場合があります。この場合、元のアプリケーションを破棄して新しく作り直すことになります。ビジネスの運用手順が徐々に変化したことが原因でアプリケーションの妥当性がしだいに失われ、ビジネス ニーズに合わなくなってしまったと判断される場合は、アプリケーションの大部分、またはすべてを書き換えることをお勧めします。アプリケーションを書き換えるもう 1 つの理由は、コードがメンテナンス不可能なほど冗長であり、ドキュメントが古く、アプリケーションが複雑になりすぎたというものです。このような状況は、複数のバージョンのアプリケーションがさまざまな開発者によって作成され、最新版に対して作業できる人が現在ではだれもいない場合に起こります。

  アプリケーションを書き換えることの利点は、最先端のテクノロジを使用して完全に新しいデザインを自由に作成できることです。欠点は、既存のアプリケーションの再開発であっても、新しいアプリケーションを作成することは、他の選択肢よりも費用がかかり、リスクも大きいことです。また、新しいアプリケーションを書き換える場合、既存のアプリケーションに対するすべての投資を無駄にすることになります。
- **置き換え。** アプリケーションを置き換えるのに適したパッケージを探すか、アプリケーションが行っていた作業を外部に委託します。現在の要件に完全に適合するパッケージが存在する可能性は低いため、ビジネス モデルを変更してパッケージに歩み寄る準備をしておいてください。

以下に挙げる指標は、アプリケーションの一部または全部を更新するための選択肢を選ぶ際に役立ちます。

- **アップグレードを選ぶにあたっての指標。**以下のとおりです。
  - 既存のアプリケーションがビジネスニーズに適合していること。
  - 既存のアプリケーションに対して必要な機能変更があまり多くないこと。
  - 新しいアプリケーションではある程度の機能追加が必要で、既存のアプリケーションと密接に統合されている必要がある場合。
  - 既存のアプリケーションでは運用コストが高い場合。
  - 戦略的な理由から .NET へのアップグレードが必要な場合。
  - アプリケーションの将来の構想に Web サービスが含まれる場合。
  - 既存のアプリケーションをメンテナンスするためのリソースを見つけるのが困難で、費用がかかる場合。
- **再利用を選ぶにあたっての指標。**以下のとおりです。
  - ビジネスルールが満たされていること。
  - 既存のアプリケーションの運用コストが低いこと。
  - ロジックと永続的なデータおよびプレゼンテーション層の分離が難しい場合。
  - 単純な Web アクセスだけが必要で、ラッピング ソリューションが使用できる場合。
  - 従来のコードの中核をメンテナンスするための十分なリソースが確保できること。
  - 既製のソフトウェアが既存のアプリケーションの中心となっており、サードパーティのサポートとメンテナンスに依存できる場合。
- **書き換えを選ぶにあたっての指標。**以下のとおりです。
  - ビジネスルールは満足できるものの、アプリケーションに機能を追加する必要がある場合。
  - 既製のソリューションではニーズを十分に満たすことができない場合。
  - 既存のアプリケーションのコードの品質が悪い場合。
  - 既存のアプリケーションのメンテナンスコストが高い場合。
  - アプリケーションを書き換えるための時間、コスト、中断が許容できるものであること。
- **置き換えを選ぶにあたっての指標。**以下のとおりです。
  - 既存のアプリケーションがビジネスニーズから大きく外れている場合。
  - 既製のソリューションに合わせてビジネス モデルを変更することが許容される場合。
  - アプリケーションを置き換えるための時間、コスト、中断が許容できるものであること。

図 1.1 は、これらの指標を図式化したものです。この図を使用するには、まず現在のアプリケーションがビジネス バリューと品質の面でどこに位置しているかを判断する必要があります。

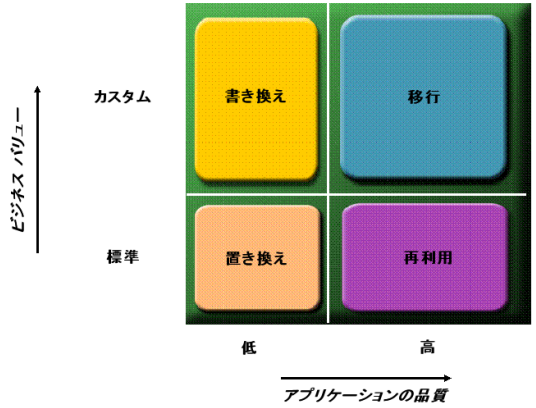

図 1.1

アプリケーションのアップグレードの方法と時期を判断するための視覚的なガイド

左側の垂直軸はビジネス バリューです。アプリケーションが、他のアプリケーション、特にサードパーティの既製の製品が持っている "標準的な" 機能を提供している場合は、この軸の下の方に位置することになります。アプリケーションがカスタム アプリケーションであり、その機能がビジネス ニーズ固有のものである場合は、この軸で上の方に位置することになります。

下側の水平軸は、ユーザーと開発者の両方から見たアプリケーションの品質を表します。アプリケーションが使いにくい、遅い、不安定である、または結果が一貫していない場合は、ユーザーから見たアプリケーションの品質は低くなります。コードが複雑でメンテナンスが難しい場合は、開発者から見た品質は低くなります。どちらの場合も、アプリケーションはアプリケーション品質の軸の左の方に位置することになります。ユーザーと開発者のどちらもアプリケーションに満足している場合は、軸の右の方に位置することになります。

アプリケーションが両方の軸のどこに位置するかがわかれば、アプリケーションを更新するために取るべき最良の方法を判断できます。一般に、"標準的な" 機能を提供しているアプリケーションは置き換えか再利用の候補となり、"カスタムの" 機能を提供しているアプリケーションは書き換えかアップグレードの候補となります。品質の低いアプリケーションは置き換えまたは書き換えるべきですが、品質の高いアプリケーションは再利用またはアップグレードするべきです。

指標がアップグレードを指している場合は、Visual Basic .NET へのアップグレードを選択することが既存のコード ベースを最大限に活用できる方法です。これらの利点のいくつかの詳細については、この章の後半の「Visual Basic 6.0 から Visual Basic .NET への移行」を参照してください。

部分的なアップグレード

これまで、この章ではアプリケーション全体を更新するか何もしないかのどちらかという方向で議論してきました。つまり、アップグレードすることも、そのままにしておくこともできます。また、書き換えたり、置き換えたりすることもできます。しかし、多くのアプリケーションでは、前述の 4 つの方法の中から 1 つを選ぶというのは、簡単な問題

ではありません。場合によっては、アップグレード、再利用、書き換え、置き換えの組み合わせを検討する必要があります。たとえば、アプリケーションのアップグレードと再利用を組み合わせることは可能であり、その場合はアプリケーションの一部だけをアップグレードすることになります。新しいコンポーネントと従来のコンポーネントを統合するために、標準的な相互運用のメカニズムを利用できます。このアプローチは部分的なアップグレードと呼ばれ、アプリケーションの特定の部分では Visual Basic .NET に移行する利点があるものの、他の部分では移行を正当化するのが難しいと判断される場合に、このアプローチが有効です。特に、前述の指標を検討し、アップグレードを選択する指標がアプリケーションの一部だけに該当し、再利用を選択する指標が他の部分に該当している場合は、部分的なアップグレードを検討することをお勧めします。

アプリケーションの更新に部分的なアップグレードが適していると判断される場合、アプリケーション全体をアップグレードするわけではないため、アップグレード作業が少なくて済むと思いがちです。しかし、新しいアプリケーションを Visual Basic 6.0 のままの従来のコンポーネントと統合するには、追加の作業が必要になります。相互運用性の詳細については、第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」を参照してください。

アップグレード方法

アップグレードがアプリケーションを更新する最適な方法だと判断したら、適切なアップグレード方法を選択する必要があります。選択可能な方法の詳細については、第 2 章「アップグレードの成功のためのプラクティス」の「アップグレード方法の選択」を参照してください。

Visual Basic 6.0 から Visual Basic .NET への移行

Visual Basic .NET にいつ移行すべきかを理解することは重要ですが、なぜアップグレードすべきなのかを理解することも同様に重要です。また、アップグレードを開始する前にその利点を理解することも大切です。Visual Basic 6.0 のすべての機能が Visual Basic .NET の機能と 1 対 1 に対応しているわけではなく、すべてのコードを自動的にアップグレードすることはできません。そこで生じる疑問が、機能している Visual Basic 6.0 アプリケーションをなぜ Visual Basic .NET にアップグレードすべきなのかということです。アップグレードの利点を完全に理解するには、まず Visual Basic .NET、.NET Framework、および標準で .NET が組み込まれた開発環境である Visual Studio .NET について理解する必要があります。

生産性の向上

Visual Basic .NET 言語には、開発者の生産性を向上させる、多くの新しい言語機能が含まれています。ここでは、こうした機能のいくつかについて触れ、それがアプリケーション開発においてどのような利益をもたらすかについて説明します。

Visual Basic .NET は、.NET Framework を中心に構築されています。Visual Basic .NET の利点の多くは、.NET Framework 自体から直接もたらされたものです。

Microsoft .NET Framework は、共通言語仕様 (CLS) という仕様に基づいています。この仕様では、Visual Basic を含むすべての .NET 言語が持っていなくてはならない最低限の機能セットを規定するルールが定義されています。この最低限の機能セットには、継承や例外処理などのオブジェクト指向機能が含まれます。これらの機能は、以前のバージョンの Visual Basic にはありません。また、コンパイラが生成する実行可能ファイルの形式が指定されており、これによりアプリケーションがあらゆるプラットフォーム上のあらゆる .NET エンジンで再コンパイルなしに動作することが保証されます。

その結果、コードはプラットフォーム固有のマシン コードではなく、Microsoft 中間言語 (MSIL) と呼ばれる特殊な形式にコンパイルされます。その後 MSIL の実行可能ファイルは、実行時に JIT (Just-In-Time) コンパイルによってプラットフォーム固有のマシン コードにコンパイルされます。このアプローチの利点は、アプリケーションの単一のコンパイルが、現在コードが動作している実際のプロセッサやオペレーティング システムに基づいて実行時に最適化されることです。たとえば、.NET アプリケーションは、標準的な 32 ビット版の Microsoft Windows® XP オペレーティング システムでは 32 ビット アプリケーションとして動作します。同じアプリケーションが、64 ビット版の Microsoft Windows Server™ 2003 オペレーティング システムでは、再コンパイルしなくても 64 ビット アプリケーションとして最適化され動作します。この仕様は公開されているため、新しいプロセッサやオペレーティング システムがリリースされたときに、専用の .NET ランタイム エンジンを作成することが可能です。これは、.NET Framework 向けに作成されたアプリケーションは、再コンパイルやアップグレードを行わなくても、新しいハードウェアやオペレーティング システム上で動作し、最適化されることを意味します。アプリケーションやコンポーネントのマルチプラットフォーム版が得られることから、開発者は既存のコンポーネントの移植に時間を費やすのではなく、新しい問題を解決できます。

さらに、MSIL には、その中に入っているクラスに関するメタデータが含まれます。その結果、開発者は "インクルード ファイル"、タイプ ライブラリ 、インターフェイス定義言語 (IDL) コードなどの、これらのクラスを別のアプリケーションで使用するための追加のファイルを作成する必要がありません。クライアント アプリケーションは、これらの情報を MSIL から自動的に取り出すことができます。これにより、MSIL の共有と使用が非常に簡単になります。既存のクラスを簡単に利用できるため、生産性が向上します。

CLS の実装における中心的なコンポーネントは、共通言語ランタイム (CLR) と呼ばれています。すべての .NET アプリケーションはこのエンジン上で動作し、すべての .NET 言語の間で相互運用と統合が CLR によって可能になります。.NET アプリケーションのリソース (メモリ、スレッド、例外、セキュリティなど) の管理は、CLR によって自動的に処理されます。.NET テクノロジだけを使用して構築されたアプリケーションは、マネージ アプリケーションまたはマネージ コードと呼ばれます。マネージという用語は、そのようなアプリケーションのリソースが CLR によって管理 (マネージ) されるということに由来します。マネージ コードを作成する開発者は、低

レベルなリソース管理ロジックに目を向けるのではなく、アプリケーション ロジックに注力できるため、CLR に作業を任せることによる恩恵を受けることができます。これにより、ソフトウェア開発者はビジネス上の問題を解決することに集中できるため、生産性がはるかに向上します。

データ型のルールは、CLS の共通型システム (CTS) と呼ばれるセクションで定義されています。この仕様では、.NET のすべてのデータ型とその動作が定義されています。その結果、.NET のすべての言語が、記憶要件や機能的な動作がすべての言語で同じである共通の基本データ型のセットを共有します。たとえば、文字列は、Visual Basic .NET モジュール、マネージ Visual C++® ライブラリ、Visual C#® コンポーネント、Visual J#® パッケージのいずれで定義されていても同じです。これによる利点は、Visual Basic .NET アプリケーションから Visual C# ライブラリに渡されたパラメータがライブラリ内で正しく解釈され、戻り値が Visual Basic .NET アプリケーションで期待どおりに動作することが仕様によって保証されていることです。アプリケーションとそれが使用するライブラリが 2 つの異なる言語で記述されているという事実をそれぞれの開発者が意識することはありません。これにより、Visual Basic .NET の開発者は、既存のライブラリが他の言語で開発されている場合でも、それらのライブラリやコンポーネントを簡単に Visual Basic .NET コードに統合できるようになったため、生産性が向上します。

CTS のそれ以外の利点としては、.NET 言語で記述されたあらゆるクラスが他の .NET 言語で記述されたクラスの動作を継承できることです。たとえば、Visual C# で開発された別のクラスの動作を継承した Visual Basic .NET クラスを作成できます。さらに、.NET Framework 自体を Visual Basic .NET から拡張することもできます。これにより、利用できるコンポーネントのセットとその利用方法を大幅に拡張できます。異なる言語間でもコードを再利用できることで、生産性が向上します。

.NET Framework 用に構築されたアプリケーションは、.NET Framework クラス ライブラリ (FCL) と呼ばれる総合的なクラス ライブラリにアクセスできます。これには、あらゆるカテゴリのクライアント アプリケーションやサーバー アプリケーションの開発が可能となる多数のクラスが含まれます。FCL には、セキュリティ、暗号化、Windows フォーム、Windows サービス、Web サービス、Web アプリケーション、I/O、サービス コンポーネント、SQL Server™ や他の多数のデータベース システムへのデータ アクセス、ディレクトリ サービス、グラフィックス、印刷、国際化、メッセージ キュー、メール、一般的なインターネット テクノロジとプロトコル、文化特有のリソース、COM との相互運用性、.NET リモート処理、シリアル化、スレッド、およびトランザクションをプログラミングするためのクラスが含まれます。FCL の利点は、Visual Basic .NET の開発者が、ゼロから始める必要がなく、FCL 上に構築された既存のコンポーネントを使用して拡張できることです。

Visual Basic .NET では、CLR がメモリの管理を行います。特に CLR は、アプリケーションで参照されなくなったオブジェクトのメモリを解放する作業を管理します。この処理はガベージ コレクションと呼ばれ、CLR のメモリが少なくなると自動的に実行されます。この処理の利点は、メモリ リソースを解放するために不要なオブジェクト

を破棄する作業を開発者が行わなくて済むことです。これにより、開発者はリソース管理の問題よりもビジネス上の問題を解決するためにより多くの時間を費やすことができるため、生産性が向上します。

Visual Basic 6.0 では、オブジェクトベースのプログラミング モデルが提供されていますが、完全なオブジェクト指向言語が持つべきいくつかの機能が欠けています。特に、Visual Basic 6.0 では安全な継承がサポートされていません。インターフェイスの継承はサポートされていますが、この制限された形では、オブジェクト指向プログラミングのすべての利点を活用することはできません。たとえば、このタイプの継承ではコードの再利用が不可能で、オブジェクト指向デザインを完全に実装する能力が制限されます。

Visual Basic .NET では、実装の完全な継承がサポートされています。既存のクラスから継承した新しいクラスを作成できます。そのうえ、スーパー クラスを作成するために使用した .NET プログラミング言語が何であっても、実装の継承を行うことができます。これは、継承を通じて既存のクラスを完全に再利用し、オブジェクト指向デザインを正確に実装できるようになったことを意味します。

実装の継承に加え、Visual Basic .NET ではインターフェイスの継承も引き続きサポートされています。インターフェイスは、Interface キーワードで定義します。その後各クラスでインターフェイスを継承し、そのメソッドを実装します。

さらに、既存のフォームを継承したフォームを作成できます。この種の継承はビジュアル継承と呼ばれ、既存のフォームのレイアウト、コントロール、およびコードを新しいフォームに適用できます。このような継承を通じて、開発者は標準的な基本フォームを構築できます。その後、アプリケーションに対して類似した外観とスタイルを持たせるために、これらのフォームをアプリケーションで継承することもできますが、新しいコントロールを追加したり、特定のコントロールの動作を変更したりすることで、フォームを自由に拡張できます。ビジュアル継承を利用すれば、アプリケーションごとに新しいフォームを作成し直すのではなく、既存のフォームを使用してそれを新しいアプリケーション向けにカスタマイズできます。

継承の利点は、コードの再利用が可能になることです。開発者は、既存のクラスやフォームを選び、継承を通じてそれをカスタマイズし、アプリケーションの要件に合わせることができます。その結果、完全に新しいコンポーネントを構築するために費やす時間が減り、生産性が向上します。

Visual Basic .NET では、Windows フォームと呼ばれる新しいフォーム開発システムがサポートされています。Windows フォーム フレームワークは、強力なフォームベースのアプリケーションを簡単に作成できるようにするための機能を提供します。これらの機能には、標準的な Windows アプリケーションのフォーム コントロールすべて (ボタンやテキスト ボックスなど)、フォームを XML サービスに接続するための組み込みサポート、ActiveX コントロールのサポートが含まれます。これらは、Windows フォームで利用できるさまざまな機能のほんの一部です。このフレームワークの利点は、開発者がフォーム アプリケーションをすばやく構築できることです。

コントロールのアンカーリングにより、開発者がフォームのサイズ変更に対処するためのコードを記述する必要がなくなりました。フォーム上のコントロールは、開発者が特に作業を行わなくても、フォームの端から一定の長さを保ち、フォームのサイズ変更に合わせてサイズが変わるように、視覚的に固定することが可能です。これにより開発者は、フォームの操作ではなくビジネスロジックに注力できるため、生産性が向上します。

Visual Basic .NET には、Web アプリケーション用に、Web フォームと XML のサポートが組み込まれています。これらの機能を利用すると、それらをアプリケーションに組み込むために必要な作業が簡単になるため、生産性が向上します。現在機能しているアプリケーションを Web サービスに拡張することもできます。さらに、Visual Studio .NET IDE には、HTML ドキュメント、XML ドキュメントおよびスキーマをすばやく効率的にデザインするのに役立つデザイン ツールが含まれます。これらすべての機能を組み合わせると、Windows ベースのアプリケーションを開発するのと同程度の作業量で Web アプリケーションを開発できる環境が得られます。

Visual Basic .NET 言語自体が持つ有益な機能に加え、標準の .NET IDE である Visual Studio .NET を使用することによる生産性の利点もあります。ここでは、これらの機能のいくつかについて説明します。

Windows フォーム デザイナでは、フォームをすばやくデザインするために必要なすべてのツールが提供されています。ボタン、チェックボックスなどのコントロールは、フォームにドラッグすることで追加できます。

Visual Studio .NET では、Web ページ用の WYSIWYG (what you see is what you get: 見たものが、手に入るもの) のデザイナも提供されています。これにより、開発者は Web ベースのソリューションをすばやくデザインできます。また、前述のように、XML デザイナでは XML 対応のソリューションをすばやく構築するための各種ツールが提供されています。

Visual Studio .NET では、開発者が完了しなければならない未処理タスクが表示されるタスク一覧表示が提供されています。タスク一覧には、未処理タスクを示すための方法として開発者自身がコードに埋め込んだ ToDo コメントが表示されます。タスク一覧ではこれらのコメントの一元化されたビューが提供され、そのタスクに対処する必要があるときに、そのコメントにすばやくジャンプできます。

タスク一覧には、修正が必要なコンパイル エラーも表示されます。Visual Basic .NET コンパイラは、コーディング中もバックグラウンドで処理を継続します。コンパイル エラーは、コードの下の青い波線でリアル タイムに通知されます。タスク一覧のコンパイル エラーの一覧もリアル タイムで更新され、修正が必要なエラーの完全な一覧が表示されます。

タスク一覧は、たとえばコンパイル エラーだけを表示したり、ToDo コメントだけを表示するなど、特定の種類の未処理タスクだけを表示するようにフィルタリングすることもできます。図 1.2 に、Visual Basic .NET ソリューションにおける一般的なタスク一覧を示します。

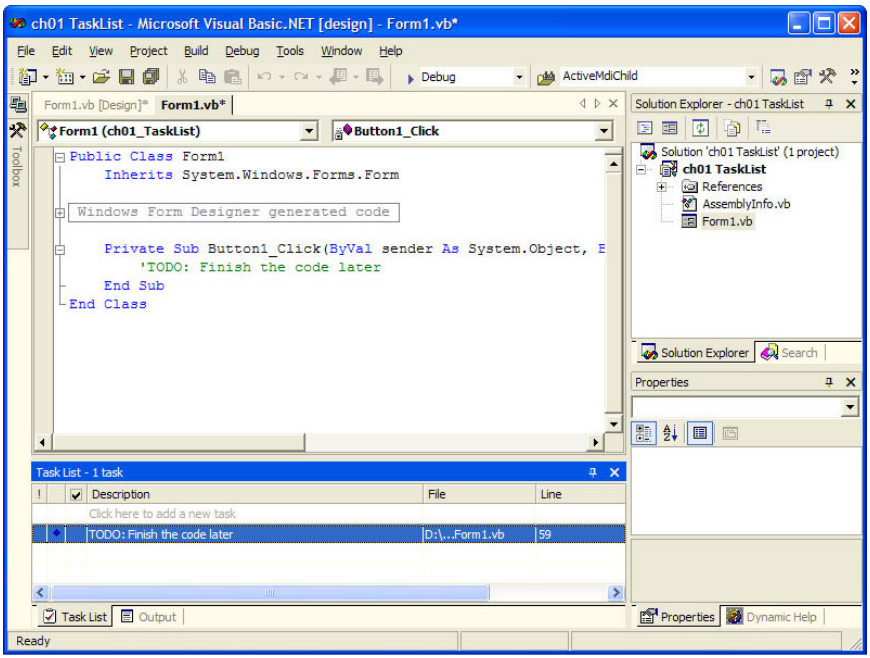

図 1.2

*タスク一覧の*ToDo *コメント*

バックグラウンド コンパイル機能とタスク一覧機能により、終了していないタスクを示すことができるため、開発者の生産性が向上します。

前述のように、.NET Framework は、言語間で相互運用が可能なように設計されています。すべてのデータ型は CLR で統一されるため、ある言語で記述されたコントロールやコンポーネントは、別の言語で簡単に使用できます。Visual Basic 6.0 のユーザー コントロールを Visual C++ プロジェクトで利用しようとして苦労した経験があるなら、この相互運用性の利点を認識できるでしょう。他の .NET 言語で記述されたコンポーネントを簡単に使用できるだけでなく、他の .NET 言語で記述されたクラスから継承することもできます。また、Visual Basic .NET のクラスから継承したコンポーネントを他の言語で作成することもできます。意外にも、この相互運用性によって開発が複雑になるということはありません。Visual Studio .NET では、複数の言語を使用したアプリケーションを開発およびコンパイルするための単一の環境が提供されます。Visual Studio .NET のソリューションは、各プロジェクトが異なる .NET 言語を使用して開発された複数のプロジェクトを含むことができます。統一されたデバッガを使用して、さまざまなコンポーネントが異なる言語で記述されている場合でも、その中をステップ スルーすることができます。また、COM+ サービスをデバッグしたり、SQL Server のストアド プロシージャにステップ インしたりすることもできます。

Visual Studio .NET の新機能が、**[プロセス]** ダイアログ ボックスです。ここから実行中のプロセスにアタッチしてデバッグできます。特定の種類のアプリケーションでは、Visual Studio では提供できない専用の実行環境が必要となります。たとえば、Windows サービスの実行は、Windows サービス マネージャで処理されます。このよう

なアプリケーションは、グラフィカル ユーザー インターフェイスを持っていません。Main メソッドと、デバッグを可能にするための適切なテスト フレームワークを持った Windows サービスを提供することも場合によっては可能ですが、Windows サービスとしての自然な環境でサービスを動作させないと、問題を再現することが不可能な場合も多々あります。[プロセス] ダイアログ ボックスが便利なもう 1 つの状況は、Visual Studio .NET の外でアプリケーションを実行している場合に、再現困難な方法でアプリケーションが予測しない動作をする場合です。Visual Studio .NET のデバッガには、現在実行中のプロセスにデバッガをアタッチするための [プロセス] オプションがあります。この機能により、IDE の外で動作しているアプリケーションをデバッグする手段が提供され、このような状況に対処するための柔軟性が得られます。

Visual Studio .NET は、SQL Server と非常に高いレベルで統合されています。SQL Server にはサーバー エクスプローラを使用して接続できます。接続後は、データベースを参照したり、新しいオブジェクトを作成したりすることができます。具体的には、SQL Server のネイティブなプロシージャ言語である TSQL で記述されたストアド プロシージャを作成および編集できます。作成を終えたストアド プロシージャは、Visual Studio .NET の中から実行できます。Visual Basic .NET のメソッドと同様に、ストアド プロシージャにステップ インしてデバッグできます。ストアド プロシージャ、関数、トリガ、または拡張ストアド プロシージャの実行中にコードの特定の行に達したら SQL Server が Visual Studio .NET に制御を返すようにブレークポイントを設定できます。

Visual Studio .NET では、開発しているソリューションの他のプロジェクトに対する参照を追加できます。これを行うと、Visual Studio は、そのプロジェクトと参照先プロジェクトの依存関係とビルド順序を自動的に計算します。また、参照されるすべてのアセンブリが常に最新であることと、必要なときに適切なディレクトリに存在していることを確認します。

以前のバージョンの Visual Studio にあった イミディエイト ウィンドウは拡張され、コマンド ウィンドウに名前が変更されました。これらのウィンドウを使用して、一時停止中のアプリケーションの変数の値を読み書きしたり、式を評価したり、ステートメントを実行したりすることができます。また、これらのウィンドウを使用して、メニュー、キーボード ショートカット、ツール バーから任意の Visual Studio コマンドを実行できます。たとえば、プロジェクトまたはソリューションに新しい 項目や既存の項目を追加したり、正規表現を使用して検索処理や置換処理を実行したり、ブレークポイントを切り替えたり、ウォッチを追加したり、呼び出し履歴やメモリを調べたり、プログラム内の現在のスレッドの一覧を表示したりするコマンドを発行できます。コマンド ウィンドウでは、ファイル名に対しても IntelliSense がサポートされており、コマンド ウィンドウを効率的に使用できます。

高度な統合

Visual Basic .NET は、.NET Framework 上に構築されたコンポーネントやレガシ コンポーネントと簡単にやり取りできるように設計されています。ここでは、高度な統合によりアプリケーションにどのような利点があるかについて説明します。

アセンブリは .NET における配置の基本単位です。アセンブリは型とリソースのコレクションであり、コンポーネントやアプリケーションなどの機能の論理的な単位を、ダイナミック リンク ライブラリ (DLL) または実行可能ファイル

(EXE) の形で構成します。また、アセンブリは、アセンブリのバージョン番号、それが依存している他のアセンブリの名前、セキュリティのアクセス許可要求など、CLR が使用する情報を保持しています。

アプリケーションのコンポーネント間の依存関係を保持して制御できることの重要な点は、コンポーネントの使用方法を定義したパブリック インターフェイスを公開できることです。これが重要なのは、コンポーネントの開発者がコンポーネントの変更、改良、または問題の修正を行っても、コンポーネントのクライアントによって記述されたコードを変更する必要がないためです。アセンブリの分離により、コンポーネントの開発者は、コンポーネントの事前に定義されたサブセット以外はクライアントから直接使用できないようにすることが可能です。これにより開発者がコンポーネントの実装を変更しても、パブリック インターフェイスが変更されない限り、問題が発生したり、クライアントに対する変更が必要になることはありません。

アセンブリの分離のもう 1 つの利点は、サイドバイサイド実行が可能なことです。サイドバイサイド実行とは、複数のバージョンのアプリケーション、アプリケーションが使用するサードパーティ コンポーネント、または .NET Framework 自体をインストールし、クライアントがどのバージョンのフレームワークやコンポーネントを使用するかを実行時に選択できる機能を指します。その結果、開発者は、新しいバージョンのモジュール、コンポーネント、またはフレームワークのリリースにより、既存のアプリケーションを破壊してしまうことを心配する必要がなくなるため、統合が高度化されます。アセンブリは互いを破壊することがなく、クライアントは参照するアセンブリの正しいバージョンを常に知っています。

Visual Basic .NET には、COM コンポーネントとの相互運用を可能にする機能が含まれます。COM 相互運用機能によって、開発者は、数値型や文字列などのすべての基本データ型に対して、マネージ Visual Basic .NET コードと COM コンポーネントのアンマネージ コードの間にブリッジを作成できます。COM コンポーネントに渡すパラメータのデータ型や、COM コンポーネントからの戻り値のデータ型は、マネージ コードとアンマネージ コードの間で、対応する適切なデータ型に変換されます。このメカニズムはデータ マーシャリングと呼ばれ、アップグレードしたアプリケーションのマネージ コードと、COM コンポーネントのアンマネージ コードの間の明確な相互運用性が保証されます。ユーザー定義型では、マーシャリングを行うカスタム コードが必要になる場合がある点に注意してください。

アプリケーションの拡張性

新しいテクノロジを使用してアプリケーションを拡張することにより、そのアプリケーションに対する投資を活用できます。Visual Basic .NET には、追加のテクノロジを使用してアプリケーションを拡張するプロセスを簡単にする機能が多数用意されています。

Visual Basic .NET では、複数の分散テクノロジがサポートされています。そのようなテクノロジの 1 つが Web サービスです。このオープン スタンダードを使用すれば、開発者はソフトウェア サービスのセットを XML を通じてインターネットに公開できるようになります。.NET Framework には、Web サービスの作成および公開のためのサポートが組み込まれています。このようにして公開されたソフトウェア サービスは、拡張性があり、HTTP、XML、SOAP、および Web サービス記述言語 (WSDL) などのオープンなインターネット標準を採用しているた

め、あらゆるクライアントまたはインターネット対応のデバイスからサービスにアクセスしてそれを利用できます。Web サービスのサポートも ASP.NET で提供されています。

もう 1 つの例が、.NET リモート処理と呼ばれる .NET Framework に固有の新しいテクノロジです。リモート処理は DCOM に似ており、アプリケーション間での通信を可能にします。この通信は、同じコンピュータ上のアプリケーション間でも、同じネットワーク上の異なるコンピュータ上のアプリケーション間でも、異なるネットワーク上の異なるコンピュータ上のアプリケーション間でも行うことができます。.NET リモート処理のサポートは、Visual Basic .NET で提供されています。

信頼性の向上

Visual Basic .NET の新機能のいくつかを利用すると、コンポーネントとアプリケーションの信頼性が向上します。ここでは、これらの機能のいくつかと、なぜ信頼性が向上するかについて説明します。

Visual Basic .NET では、データ型は静的にバインドされます。つまり、変数の型はコンパイル時にわかっている必要があります。この方法を使用することで、コンパイラは特定の変数や値に対して有効な演算だけが適用されることを確認できます。Visual Basic .NET と CTS によって適用されるこの厳密なデータ型のルールは、異なる言語間で高レベルの互換性を提供するためだけのものではありません。これらのルールがあることで、データの不正な使用が原因で発生するエラーが開発ライフサイクルのできるだけ初期の段階、一般にはコンパイル時に検出されるような開発プラットフォームが構築されます。コンポーネント間でのデータ型の相互運用性が向上し、Visual Basic .NET でデータ型のルールが厳密に適用されることで、アプリケーションとコンポーネントの信頼性が高まります。

Visual Basic .NET では、プログラミング エラーを減らすのに役立つ厳密な型チェックがサポートされています。たとえば、プロパティに誤った enum 値を設定したり、変数に互換性のない値を代入したりすると、コンパイラがそれを検出して報告します。また、ADO.NET では、アプリケーションに型指定されたデータセットを追加できます。コードが不正なフィールド名を参照すると、ランタイム エラーではなくコンパイル エラーとして検出されます。エラーの検出が開発サイクルの早い段階であるほど、修正は簡単になります。実行時ではなくコンパイル時に型エラーを検出することで、修正が簡単になります。

アプリケーション コード内で Option Strict On を使用すれば、より厳密な型チェックをコンパイル時に適用することも可能です。このオプションを使用すると、遅延バインディングが完全に禁止され、変数に異なる型の値を代入する際は、必ず変換関数を使用する必要があります。これにより、変換が行われることで原因のわかりにくいエラーが発生するコードの箇所を特定できるようになります。例としては、有効桁数が大きな値を有効桁数が小さな変数に代入した場合の切り捨てによる値の損失 (たとえば、Long 値を Integer 変数に代入した場合) などがこれに該当します。

信頼性の向上につながる Visual Basic .NET のもう 1 つの新機能として、データや動作のカプセル化などオブジェクト指向の概念に準拠することを強制するセキュリティ機能があります。この機能は*タイプ セーフティ*と呼ば

れ、オブジェクトのプライベート メンバへのアクセスを制限することで、メモリ位置を保護します。フィールドやメソッドがプライベートとして宣言されている場合は、それが属するクラス内からのみアクセスできます。

タイプ セーフティという用語は、不正なデータ型に関連するランタイム エラーの数を減らす言語機能を指す、より広い意味も持っています。この文脈では、タイプ セーフティは、適切なデータ型の値に対して演算が実行されることを確認する、コンパイル時のチェックを指します。

これに対し、.NET のセキュリティを保証するために使用されるタイプ セーフティはコンパイラによってチェックされ、検証を使用して実行時に確認することもできます。検証が必要な理由は、アプリケーション内のさまざまなアセンブリのコンパイル方法についてはランタイムが制御できないためですが、コードが検証を迂回するアクセス許可を持っている場合は、この処理を省略できます。このタイプ セーフティにより、クラスのパブリック インターフェイスが尊重され、アセンブリをアプリケーション ドメイン内の他のアセンブリから分離できることが保証されます。これにより信頼性とセキュリティが向上します。他のコードがプライベートとして宣言されたデータを変更したり機能を実行することにより実行に悪影響を与えることがなくなるためです。.NET におけるタイプ セーフティと検証の詳細については、MSDN の『.NET Framework Developer's Guide』の「Type Safety and Security」を参照してください。

構造化例外処理は、アプリケーションの信頼性を向上させる Visual Basic .NET のもう 1 つの新機能です。Visual Basic 6.0 で慣れ親しんだ On Error GoTo エラー キャッチ メカニズムのサポートに加え、Visual Basic .NET では、新しい構造化エラー処理メカニズムである Try/Catch/Finally ブロックと Throw ステートメントがサポートされています。

Try/Catch/Finally 構造では、エラーやその他の例外条件になる可能性があるコードを囲み、考えられる例外それぞれに対するハンドラを指定します。ソフトウェア開発者は、個別のコード行から、コード ブロック全体、関数内のすべてのコードまで、例外処理の粒度を完全に制御できます。また、処理する例外の種類についても、特定の例外から、汎用の catch-all 例外までを制御できます。構造のオプションの Finally 部分には、例外が発生したかどうかにかかわらず実行されるコードを指定できます。

Throw ステートメントは、開発者がエラー状況または例外状況の発生を知らせるための手段です。これを使用すると、開発者は、予期しない状況が発生したことをクライアントに知らせ、その状況をどのように扱うかを判断する機会をクライアントに与えることができます。クライアントはその状況から回復することも、アプリケーションを適切に終了させる方法を提供することもできます。

構造化例外処理は、エラー状態が発生したときに、その状況から回復したり、適切に終了したりするための手段を提供します。これらの方法は、データの損失を防ぐのに役立ち、エラーが発生した場合でもアプリケーションの信頼性が向上します。

セキュリティの向上

アプリケーション セキュリティは、Visual Basic .NET の重要な機能の 1 つです。.NET Framework は、開発者に複雑な作業を強いることのない、高度なセキュリティ機能のセットを備えています。CLR のセキュリティ管理機能は、アプリケーションやそれに依存しているコンポーネントが含まれるアセンブリを他の人が変更するのを防ぎます。また、アクセスが許可されていないメモリにコンポーネントがアクセスできないことが保証されます。

開発者は、構成ファイルを使用するアプリケーションやコンポーネントのアクセス許可を制御できます。また、アプリケーションを実行しているユーザーの特権レベルや、アプリケーション自体の信頼レベルに基づいて、リソースへのアクセスを制御できます。これにより、コードのユーザーがもともとフル アクセス可能である場合でも、信頼されていないコードがリソースにアクセスするのを制限できます。たとえば、信頼できないコードがスーパーユーザー アカウントで実行されていた場合でも、リソースへのアクセスを制限できます。

配置オプションの強化

.NET 以前は、クライアント アプリケーションを配置するためには、一般にレジストリ設定を適用したり、適切なディレクトリに DLL をコピーする必要がありました。そのため、レジストリ設定が壊れたり、別のアプリケーションが DLL を互換性のないバージョンで置き換えたりすると、アプリケーションが動作しなくなる可能性がありました。

Visual Basic .NET には、さまざまなクライアントの配置方法を簡素化するために設計された、多数の新機能があります。これには、次のような機能が含まれています。

- .NET Framework では、アプリケーションの分離がサポートされています。あるアプリケーションの DLL は、他のアプリケーションに影響を与えません。これは、ゼロインパクト インストールと呼ばれます。
- 既定では、コンポーネントと DLL は、アプリケーション用に非公開でインストールされます。これらのコンポーネントを他のアプリケーションから参照できるようにするには、明示的に共有する必要があります。
- 複数のバージョンの DLL を透過的に共有できます。アプリケーションのアセンブリには、使用しているすべての DLL で必要なバージョン情報がすべて含まれます。これにより、複数のバージョンの DLL がターゲットとなるコンピュータに存在している場合でも、アプリケーションは正しいバージョンだけを使用します。
- レジストリ エントリを使用せず、依存関係を持たない自己完結型のアプリケーションおよびコンポーネントを配置できます。必要なのは、適切なファイルをアプリケーションの bin ディレクトリにコピーすることだけです。このようなインストールは、XCOPY インストールと呼ばれます。
- アプリケーションは、広告、発行、修復、オンデマンド インストールなど、Microsoft Windows インストーラで提供されている機能を利用できます。
- インクリメンタル インストール機能を利用すると、ダウンロードするデータのサイズが小さくて済みます。さらに、インストールに失敗すると、ターゲットのコンピュータはインストーラが開始する前の状態に完全に戻ります。

- マージ モジュールと呼ばれる新しい種類のインストーラは、複数のアプリケーションで共有されるコンポーネントをパッケージ化してインストールするための非常に簡単な方法を提供します。マージ モジュール (.msm ファイル) を利用すると、複数の配置プロジェクト間でコンポーネントを共有できます。Visual Studio .NET を使用して独自のマージ モジュールを作成することも、マイクロソフトやサードパーティ ベンダが提供している多数の標準コンポーネント向けに利用可能な既存のマージ モジュールを使用することもできます。マージ モジュールは、一度作成すればそれを使用するすべてのアプリケーションに含めることができます。マージ モジュールは、どのアプリケーションがそのコンポーネントを使用しているか、それを必要とする最後のアプリケーションがいつアンインストールされたかなどの詳細をすべて記録します。マージ モジュールと他のインストール オプションおよび配置オプションについての詳細は、第 16 章「アプリケーションの完成」を参照してください。

Visual Basic .NET において配置機能が強化されたことにより、インストールによって他のアプリケーションのインストールが破壊されることがなくなりました。

パフォーマンスの向上

Visual Basic .NET のいくつかの新機能を使用すると、アプリケーションのパフォーマンスを向上させることができます。ここでは、これらの機能のいくつかと、それを使用してパフォーマンスを向上させる方法について説明します。

典型的な Visual Basic アプリケーションはシングル スレッドであり、同時に 1 つのタスクしか実行できません。Visual Basic .NET ではマルチスレッド アプリケーションがサポートされているため、複数のタスクを同時に実行できます。マルチスレッドによって、以下のようにアプリケーションのパフォーマンスが向上します。

- 他の作業を実行している最中もユーザー インターフェイスをアクティブなままにできるため、プログラムの応答性が高まります。
- アイドル タスクは割り当てられたプロセッサ時間を他のタスクに与えることができます。
- プロセッサを集中的に使用するタスクも、定期的に他のタスクにプロセッサを明け渡すことができます。

たとえば、データベースから読み込んだ情報をクライアントに提供するサーバー アプリケーションがあるとします。シングルスレッドのアプリケーションでは、サーバーは同時に 1 つのクライアントにサービスを提供できます。つまり、複数のクライアントが同時に情報を必要としている場合でも、サービスを受けるのは 1 つだけです。他のクライアントは、最初のクライアントの要求が完了し、自分の順番になるのを待つ必要があります。マルチスレッド アプリケーションでは、サーバーには複数の独立した実行スレッドがあり、個々のクライアントの要求に対して同時にサービスを提供します。その結果、各クライアントは、すぐにサーバーのサービスを受けることができます。

テクニカル サポート

2005 年 3 月の時点で、マイクロソフトによる Visual Basic 6.0 のサポートはメインストリーム フェーズが終了し、延長フェーズに移行しました。現在の延長フェーズのサポートは、2008 年 3 月まで提供され、その時点で Visual Basic 6.0 のサポートは終了します。

これらのサポート ポリシーによる影響は、Visual Basic 6.0 の無償サポートが提供されなくなることです。延長フェーズの間、ユーザーは、インシデントごとに課金される電話およびオンライン インシデント サポート チャネルやプレミア サポートを通じて、引き続き Visual Basic 6.0 のプロフェッショナル プロダクト サポートを受けることができます。同様に、重要な更新と修正プログラムは、延長フェーズの間有償でのみ提供されます。

Visual Basic 6.0 のサポート ポリシーの詳細については、MSDN の Microsoft Visual Basic Developer Center の「Product Family Life-Cycle Guidelines for Visual Basic 6.0」を参照してください。

Visual Basic 6.0 to Visual Basic .NET Upgrade Wizard の利点

アプリケーションをアップグレードする場合は、Visual Studio .NET IDE の Visual Basic 6.0 to Visual Basic .NET Upgrade Wizard を使用するのがアップグレードのための最良の方法です。アップグレード ウィザードを使用してアプリケーションをアップグレードすることで、以下のことが可能になります。

- 拡張とメンテナンスの両方が容易なソースコードを生成できます。
- 元のプログラムに組み込まれている機能はすべてそのままに、機能的に同等なアプリケーションを得ることができます。
- コードを強力かつインテリジェントに自動で再構成できます。
- 元のコードのコメントを残すことができます。
- すべてのアプリケーション コンポーネントの相互参照を維持できます。

Visual Basic .NET でサポートされていないテクノロジをアプリケーションで多数使用していると、これらの利点のいくつかが半減したり、失われたりします。Visual Basic 6.0 Upgrade Assessment Tool は、アプリケーションがそのようなテクノロジに依存していないかどうかを識別するのに役立ちます。またこのガイドでは、Visual Basic .NET でサポートされていないテクノロジに依存している場合の対処方法について、その解決方法を説明しています。

まとめ

Visual Basic 6.0 アプリケーションを Visual Basic .NET にアップグレードするかどうかを決める際には、検討すべきさまざまな要因があります。アプリケーションの中には、.NET テクノロジの恩恵を受けられなかったり、アップグレードのコストや複雑さが大きすぎるために、アップグレードにあまり適していないものがあります。他のアプリケーションでは、利点がコストをはるかに上回るため、現実的に移行が必要になる場合もあります。この章の主なポイントは、アップグレードはアプリケーションごとに検討する必要があるということです。アプリケーションを Visual Basic .NET にアップグレードするための説得力のある理由は多数存在しますが、アプリケーションをそのままにしておくための説得力のある理由もまた同じように存在します。アプリケーションのアップグレードは、意味がある場合にのみ行ってください。アップグレードが正当である場合、アップグレード計画を作成することでリスクとコストを最小限に抑えることができます。このガイドの以降の章では、そのような計画を作成し、実際のアップグレードをできるだけ効率よく行うために役立つツールと技法について説明します。

詳細情報

Visual Basic Upgrade Assessment Tool の詳細またはダウンロード方法については、次の GotDotNet コミュニティサイトにある「Visual Basic 6 to Visual Basic. NET Migration Guide」を参照してください。
http://www.gotdotnet.com/codegallery/codegallery.aspx?id=07c69750-9b49-4783-b0fc-94710433a66d

.NET におけるタイプ セーフティと検証の詳細については、MSDN の『.NET Framework Developer's Guide』の「Type Safety and Security」を参照してください。
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpcontypesafetysecurity.asp

Visual Basic 6.0 のサポートポリシーの詳細については、MSDN の Microsoft Visual Basic Developer Center の「Product Family Life-Cycle Guidelines for Visual Basic 6.0」を参照してください。
http://msdn.microsoft.com/vbasic/support/vb6.aspx

2

アップグレードの成功のためのプラクティス

この章では、アップグレード プロセスについて詳しく検討すると共に、第 1 章で紹介した 2 つの主要概念である同等の機能とアプリケーションの改良についてさらに詳しく説明します。また、この 2 つの概念に対応するアップグレード方法を紹介し、アップグレード プロジェクトを成功させるためのいくつかのベスト プラクティスについて説明します。

同等の機能とアプリケーションの改良

Visual Basic 6.0 から Visual Basic .NET にアプリケーションをアップグレードする際には、元のアプリケーションと同じ外観と機能を備えた Visual Basic .NET バージョンのアプリケーションができた時点でアップグレード プロセスを終了するか、新しい言語で利用可能な新機能をアプリケーションに追加するかを選択できます。ここでは、これら 2 つの方法、つまり同等の機能とアプリケーションの改良について説明します。

同等の機能

同等の機能は、すべてのアップグレード プロジェクトにとって重要な概念です。Microsoft Visual Basic のアップグレードの場合において、同等の機能の実現とは、アップグレードが完了し、まだ新しい機能が追加される前の時点で、Visual Basic 6.0 アプリケーションとまったく同じ機能が Visual Basic .NET アプリケーションに保持されている状態を意味します。同等の機能を実現することにより、アプリケーションのロジックがアップグレード後も変わらないこと、および既存のビジネス ルールがアップグレード後のアプリケーションでも機能することが保証されます。

同等の機能をアップグレード プロジェクトの目標にすると、組織にとって次のようなメリットがあります。

- Visual Basic .NET アプリケーションは元の Visual Basic 6.0 アプリケーションと機能的に同等であるため、アプリケーションのアップグレード プロジェクトの完全な仕様があることになります。つまり、アプリ

ケーションの動作や出力の不一致をすぐに見つけることができ、アップグレード プロジェクトが成功したかどうかを明確に判定できます。

- 機能的に同等なアプリケーションでは、スコープの拡大や変化がないため、プロジェクトのテストフェーズが予測可能で容易なものになります。ただし、同等の機能が実現された後に追加される新機能については、仕様を準備する必要があります。
- 同等の機能の実現を目標とする場合、プロジェクトの進捗状況をわかりやすく明確な方法で追跡できます。アプリケーションの一部をアップグレードするたびに、その機能を元のアプリケーションと比較できます。
- 機能的に同等なアプリケーションでは、既存のユーザーがアップグレード後のアプリケーションに適応する必要がなく、変更管理のプロセスを IT 部門に集約できます。
- 同等の機能の実現は、組織で古い技術から新しい技術に移行するための最も迅速な方法です。また、最も効率的に組織の自律管理を回復し、アプリケーションの通常のインクリメンタルなメンテナンスを再開できます。

アプリケーションの改良

同等の機能が実現されると、.NET バージョンのアプリケーションを使用できるようになりますが、これは、アプリケーションをアップグレードするプロセスの中間ステップと見なされるのが一般的です。第 1 章「はじめに」で述べたように、アプリケーションを .NET Framework にアップグレードすることの最大の利点は、アプリケーションのパフォーマンス、統合、生産性、拡張性、信頼性、およびセキュリティを大幅に強化できる点にあります。この新しいテクノロジを使用してアプリケーションの機能を追加または強化することを、アプリケーションの改良と呼びます。

アプリケーションの改良では、.NET Framework のさまざまな改良点の恩恵を受けるアプリケーションの領域を特定して、それを変更または拡張します。実際、アプリケーションの改良こそが、Visual Basic .NET にアップグレードするそもそもの理由と考えることもできます。最終的には変更することになる同等の機能をアプリケーションで最初に実現することは労力の無駄に見えるかもしれませんが、実際にはそれが、アプリケーションの改良を慎重に行うための確かな基礎となります。

組織の構造とソフトウェアのライフ サイクル

アップグレード プロジェクトの重点を同等の機能の実現に置くか、同等の機能の実現とその後のアプリケーションの改良に置くか、アプリケーションの改良のみに置くかに関係なく、アプリケーションのアップグレードは、標準のソフトウェア開発プロジェクトとして考える必要があります。アプリケーションのアップグレードでは、アプリケーションの設計 (の大半) が既に決定されていて、アプリケーションが既に実装されているとしても、通常の開

発サイクルの段階についての計画が必要です。これらの段階 (期待の管理、設計、実装、およびテスト) は、アップグレード プロセスにとってきわめて重要な要素です。

テストはアップグレード プロジェクトにとって非常に重要であり、また、アプリケーションの以前のバージョンのために開発したテスト プロセスを再利用できる場合があります。たとえば、Visual Basic 6.0 アプリケーションで使用していたテスト資産 (テスト計画、テスト ケース、テスト コードなど) の一部をアップグレード後も使用できる場合があります。これについては、この後で詳しく説明します。

アップグレード プロセスの概要

Visual Basic 6.0 アプリケーションのアップグレードは、簡単な作業ではありません。他の開発プロジェクトと同様に、プロセスの概念を明確にすることが重要です。そうすることにより、限られた時間と予算の中でプロジェクトを遂行するのに役立ちます。

ここで説明するプロセスは、マイクロソフトやそのパートナーによって行われた多くの Visual Basic アップグレード プロジェクトの結果として開発されたものです。この章では、アップグレード プロセスを、アップグレード中のマイルストーンを表す一連のフェーズとして見ていきます。計画など、他のフェーズを開始する前にプロジェクト全体に対して完了させておく必要のあるフェーズもあれば、アプリケーションの各部分に個別に適用でき、その間に他の部分を後のフェーズに進めることができるフェーズもあります。たとえば、段階的なアップグレード方法を選択した場合は、Visual Basic アプリケーションの中に、アップグレードの準備が行われている部分と、既にアップグレードが済んでテストできる状態になっている部分がある場合があります。

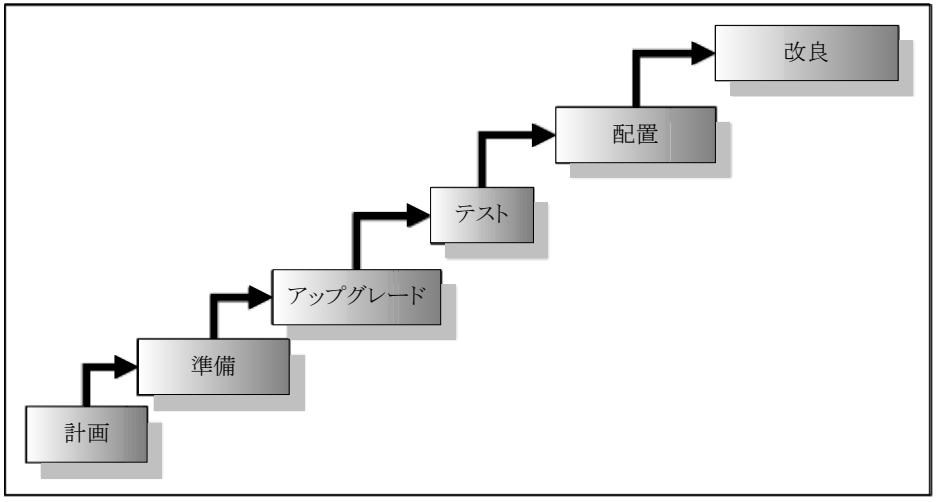

図 2.1

アップグレード プロセスを表す図

アップグレード プロセスの主なフェーズには、計画、準備、アップグレード、テスト、配置、および改良があります。これらのフェーズについては、この後で詳しく検討することにして、ここでは簡単に説明します。

- **計画**。計画は、アップグレード プロジェクトの最初のフェーズです。ここでは、目標、期待、および制約の定義が行われます。計画フェーズでは、アプリケーションのアップグレードに伴うリスクを評価し、必要な手順を決定するために、アップグレードするアプリケーションを評価する必要があります。特に重要となるのが、モジュールをアップグレードする順序の決定です。また、アプリケーションのアップグレードのコストも計画フェーズで評価する必要があります。
- **準備**。2 番目のフェーズでは、アプリケーションをアップグレードに向けて準備します。準備フェーズは必須ではありませんが、ここでアプリケーションを事前に準備しておくとアップグレード フェーズの時間を大幅に節約できるため、このフェーズを省略しないよう強くお勧めします。アプリケーションの準備では、Visual Basic アップグレード ウィザードをできるだけ効率的に使用できるように、アプリケーションの実装の詳細を一部変更することもあります。これにより、アップグレード ウィザードの使用後に必要となる作業を減らすことができます。
- **アップグレード**。アップグレード フェーズでは、アップグレード ウィザードを使用して、できるだけ多くのコードを Visual Basic 6.0 から Visual Basic .NET にアップグレードします。その後、残りのコードを手動でアップグレードします。このフェーズの結果は、Visual Basic 6.0 アプリケーションと機能的に同等な Visual Basic .NET アプリケーションになる必要があります。
- **テスト**。通常は、テストが、アプリケーションの機能が同等であることを実証する根拠となります。ここではアップグレード フェーズの次のフェーズになっていますが、テストはプロセス全体を通じて行う必要があります。これは、アップグレードの初期の段階についても同様です。たとえば、個々のコンポーネントのアップグレードが完了するたびに単体テストを行うことにより、システム全体における動作をテストする前に、それらのコンポーネントに問題がないことを確認できます。
- **配置**。配置でもテストが必要になるため、配置フェーズは、アップグレード フェーズやテスト フェーズと重なる部分がよくあります。アプリケーションが正しく配置されないとテスト フェーズが完了しないプロジェクトでは特にそうです。
- **改良**。最後のフェーズでは、.NET の新機能の実装や、.NET の機能を使用した既存の機能の強化により、機能的に同等なアプリケーションを改良します。新しい機能を追加した場合は、それらについてもテストする必要があることに注意してください。

本格的にプロジェクトを開始する前に、アプリケーションのサブセクションを使ってこのプロセスのテスト ランを行うこともできます。これにより、コストの見積もりを改善できる可能性があるほか、実際のプロジェクトの障害になりそうな問題を見つけてアップグレードのリスクを削減できる場合もあります。

概念実証

プロジェクトの計画を立てる際には、アップグレードの主要な問題を概念実証やプロトタイプを使用してテストする必要性を評価する必要があります。概念実証は、プロジェクトが理想的かどうか、または実現可能かどうかを実証する証拠です。概念実証やプロトタイプは、コーディング時の誤った決定によって時間を無駄にしないようにするのに役立ちます。また、これらを使用すると、ソリューションを分離された状態で検証できます。また、概念実証は、アップグレード ウィザードのアップグレード レポートに表示される最も一般的な問題について、実際のコストを特定するのにも役立ちます。アップグレード ウィザードの詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

アップグレード プロジェクトの概念実証は、アップグレード プロセスの詳細に慣れるための手段として有効です。概念実証を開発する方法としては、練習用のアプリケーションをアップグレードすることをお勧めします。練習用のアプリケーションとしては、アップグレード ウィザードで使用するのに適した単純なアプリケーションを選択することを検討し、他のテクノロジと緊密に統合されているアプリケーションは避けます (詳細については、この後の「アプリケーション分析の実行」を参照してください)。練習用のアプリケーションを選択してアップグレードした後に、アップグレード後のコードで一般的な問題を分析し、それらを解決することによって、貴重な経験をすることができます。この作業は、より複雑な実際のプロジェクトの実行に影響する可能性がある問題を見つけるのに役立ちます。アップグレードの習得では、その他の習得と同様に、扱える量の単純なプロジェクトから始めます。また、練習用のアプリケーションでは、アプリケーションをすばやくアップグレードして実行できると考えられるため、より大きなプロジェクトに取り組むのに必要な自信を得ることもできます。最後に、このアップグレードの練習は、より大規模で複雑なアプリケーションのアップグレードの実現可能性を評価するうえでも有効です。

概念実証によって、より複雑な実際のアプリケーションのアップグレードの実現可能性を確信できたら、アプリケーションで同等の機能が実現された時点から、最後のフェーズであるアプリケーションの改良までのプロセスを確認できるように、.NET Framework の新機能の恩恵を受けるアプリケーションをアップグレードの対象として選択します。選択する際には、アーキテクチャがどのように変更されるのかを考える必要があります。大規模なアプリケーションでは、分断攻略の原則が当てはまり、アプリケーションをモジュール単位でアップグレードする必要があります。たとえば、まずクライアント モジュールから始めて、次にビジネス ロジック層のモジュールをアップグレードし、その次はデータ層のモジュールをアップグレードするという形で、依存関係の階層を上っていきます (アップグレード方法の詳細については、第 1 章「はじめに」を参照してください)。プロセスを進めていく際には、アップグレード後のアプリケーションのモジュールをその都度テストするようにしてください。これにより、先に進む前にコードの品質を確保できます。

アップグレード プロセスについてある程度の経験を積み、アップグレードする実際のアプリケーションを選択したら、プロジェクトのスコープを定義する作業に入ることができます。

メモ：アプリケーションを部分ごとにアップグレードする場合は、相互運用の手法を利用して、まだ Visual Basic 6.0 のままのコンポーネントと、Visual Basic .NET にアップグレードしたコンポーネントが通信できるようにする必要があります。相互運用性の詳細については、第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」を参照してください。

アップグレードの計画

アップグレード プロジェクトの計画には、組織のソフトウェア開発プロジェクトで通常行われるタスクに加えて、以下の作業が含まれている必要があります。

- **プロジェクト スコープの定義**。アップグレード プロジェクトの目的を定義します。どのコンポーネントや機能をアップグレードするのか、アップグレードに何を期待するのかを明確にします。
- **アプリケーション分析の実行**。アップグレードするアプリケーションのプロファイリングを行って、最適なアップグレード方法を決定します。
- **現在のアーキテクチャとターゲット アーキテクチャの評価**。アップグレードするアプリケーションの設計と実装を評価します。Visual Basic .NET への移行の間に変更する必要があるアーキテクチャの側面を明らかにします。
- **新機能の分析と設計**。現在のアプリケーションにない機能を期待する場合は、新機能の分析および設計の計画を立てます。
- **アップグレード方法の選択**。アプリケーションの設計によって、どのアップグレード方法を選択すればよいかがわかるでしょう。アップグレード方法を選択する際には、アプリケーションの中間リリースのための測定可能なマイルストーンとテスト プロセスを作成する必要があります。
- **ソース コードのインベントリの作成**。アップグレードする必要があるソース コードの量を指定します。アプリケーションのコンパイルと実行に必要なアプリケーションのその他の依存関係もすべて含める必要があります。
- **ソース コードの準備**。アップグレードに向けてコードを準備するための時間を十分に確保します。チームやコードの準備が整うまでは、アップグレード フェーズに入りたい気持ちを抑えてください。
- **アップグレードの問題への対処の準備**。アプリケーションの評価の結果に基づいて、アップグレード ウィザードでは完全にアップグレードできない機能のアップグレードを手動で完了するための適正な時間をスケジュールに入れます。また、アップグレードに成功しても .NET 環境で動作が変わる可能性がある機能にかかる時間も必要になります。
- **単体テスト**。アップグレード プロジェクトであっても、低レベルのコードも予想どおりに動作することを確認するために、単体レベルでテストを行う必要があります。これにより、統合テストや受け入れテストの効率が上がります。

- **同等の機能と受け入れの検証**。新しいアプリケーションが実際に元のアプリケーションとまったく同じように (あるいはあらかじめ定義された誤差の範囲で) 動作することを証明する必要がある場合は、同等の機能とアプリケーションの受け入れを判定するために使用する基準を定義する必要があります。
- **新機能の実装とテスト**。アップグレード後のアプリケーションに追加するすべての新機能を実装およびテストするための時間とリソースをスケジュールに入れます。
- **Visual Basic .NET アプリケーションの配置**。新しいアプリケーションを配置して実稼動するまでは、アップグレードが完了したことにはなりません。元のアプリケーションの配置から新しくアップグレードした .NET バージョンの配置への移行に必要な計画を立てます。

プロジェクト スコープの定義

プロジェクト スコープの定義は、プロジェクトの計画においてきわめて重要な段階です。ここでは、プロジェクトの目標と、それらの目標を達成するために処理できるタスクを設定します。また、それらの目標が満たされたかどうかの判定に使用するマイルストーンや基準も定義します。

アップグレード プロジェクトの計画には、他のソフトウェア開発プロジェクトでは通常必要とされない目標がいくつか含まれます。第 1 に、アップグレード後のアプリケーションが元のアプリケーションと機能的に同等かどうかを判定するための基準がプロジェクト スコープに含まれている必要があります。

マイルストーンの達成を判定するために使用する基準を指定し、それらを検証するためのメカニズムを準備する必要があります。まず、テスト ケースの作成から始めます。テスト ケースは、アップグレード プロセスの進捗状況や同等の機能を判定するための重要なツールであり、他のソフトウェア開発プロジェクトと同様に、アプリケーションの品質を高レベルに保つのに役立ちます。

以下の質問に答えることによって、プロジェクト スコープをさらに細かく定義できます。

- **アップグレード後のアプリケーションの最初のバージョンに機能を追加することを期待しているか。**第 1 の目標は、同等の機能の実現であるべきです。機能を追加したい場合は、アプリケーションの改良に関連するタスクの具体的なマイルストーンを定義することによって、この目標とは切り離してください。追加機能は、アプリケーションをアップグレードした結果として自動的に得られるものではありません。そのための計画を立てる必要があります。
- **機能的に同等のアプリケーションのパフォーマンスの向上を期待しているか。**アプリケーションのパフォーマンスは、アップグレードの結果として自動的に向上するものではありません。ただし、パフォーマンスの強化は、アップグレード後のアプリケーションの最初の反復の要件として妥当であるため、早い段階でこれらの要件をきちんと文書化しておく必要があります。

- **セキュリティ、使いやすさ、信頼性などの強化を期待しているか。**これらは、アプリケーションで同等の機能が実現されてから実装する必要がある追加機能です。これらの実装は、アプリケーションの改良に含まれます。
- **配置や国際化の新しいテクニックを利用する予定があるか。**これらの新機能の実装も、アプリケーションの改良に含まれます。したがって、アプリケーションで同等の機能が実現されてから開始する必要があります。
- **Visual Basic .NET や .NET Framework に関連する新しい依存関係と、それらが配置の要件に与える影響を認識しているか。**機能的に同等なアプリケーションの配置は妥当な期待であり、その成功のためには新しい依存関係の計画がきわめて重要になります。
- **アップグレード後のアプリケーションの外観が元のアプリケーションとまったく同じであることを期待しているか。外観が異なる場合の準備はできているか。**アプリケーションの種類によっては外観の変化が避けられない場合もあるため、それらの変化をユーザーのために文書にまとめる計画が必要です。アプリケーションを Visual Basic .NET にアップグレードするときに期待する外観および動作の変化の例については、第 9 章「Visual Basic 6.0 フォーム機能のアップグレード」を参照してください。

アプリケーション分析の実行

アップグレード プロジェクトのリスクを管理し、コストを見積もるうえで重要なステップとなるのが、専門家のチーム (社内のチームであっても社外のチームであってもかまいません) によって行われるアプリケーションの技術的な分析です。これにより、プロジェクトにとって最適なアップグレード方法が決定されます。この作業の目標は、プロジェクトが実際のビジネス ニーズに適合していること、およびプロジェクトへの期待がプロジェクトの結果について現実的なものになっていることを確認することです。この分析では、プロジェクトの時間とコストの見積もりの概略と、アプリケーションのテクニカル プロファイルが作成される必要があります。この概略とプロファイルは、どちらもプロジェクトの成功に欠かせません。

プロファイリングは、アップグレードするアプリケーション コードの洗い出しと分析から始まります。最初に、ライブラリやリソースへの参照も含め、アプリケーション ソース コードのインベントリを作成することをお勧めします。その後、アップグレードの後の段階で対処が必要になる可能性がある問題を洗い出して分類する必要があります。このタスクでは、アップグレード ウィザードを使用して最初にテスト ランを行うことがきわめて有効です。その際には、ソース コードの機能を、自動的に処理される機能と、手動で処理する必要がある機能や、プロセスの後の段階で再設計を検討しなければならなくなる可能性がある機能とに区別する必要があります。

分析の一環として、アップグレード ウィザードによって生成されるアップグレード レポートを確認します。アップグレード レポートには、アップグレード プロセスに関する情報や、アップグレード プロセスで発見された問題の一覧が含まれています。これらの問題に対処しないと、プロジェクトをコンパイルして実行することはできません。アップグレード レポートに示されるさまざまな問題と、それらを解決するための推奨されるソリューションについて調査します。既に述べたように、アップグレード ウィザードを適用する前に Visual Basic 6.0 のコードを準備す

ることによって、対処しなければならない問題の数を減らすことができます。アップグレード ウィザードの使用の詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

アプリケーション分析の目的は、アップグレード プロジェクトの作業量の見積もりに使用する情報を生成し、アプリケーションのテクニカル プロファイルを作成することです。アプリケーション分析の結果には、以下の項目が含まれている必要があります。

- 現在のアーキテクチャとターゲット アーキテクチャ
- アップグレードするインベントリ
- ソースコードのメトリクス
- サポートされない機能の処理
- アップグレード レポート情報の評価

現在のアーキテクチャとターゲット アーキテクチャの評価

分析の一環として、アプリケーションの現在のアーキテクチャと、そのコンポーネントがターゲット アーキテクチャでどのように置き換わるのかを明確に理解する必要があります。たとえば、ActiveX Data Object (ADO) の ADO.NET への置換や、COM コンポーネントの .NET Framework のネイティブ コンポーネントへの置換を計画する場合があります。結果となるターゲット アーキテクチャには、アプリケーションの新しい要件をサポートする新しいコンポーネントも含まれている必要があります。これらの新しいコンポーネントは、新しい要件分析と設計の作成の一部として識別されます。

アップグレード後のアプリケーションが、他の既存の Visual Basic アプリケーション、モジュール、またはコンポーネントや、再利用するがアップグレードはしないと決定したアプリケーションの任意の部分と対話する場合もよくあります。アプリケーションのアーキテクチャの中でこれらの対話を洗い出して、対話するコンポーネントと、相互運用機能を必要とするコンポーネントとを明確に区別する必要があります。また、同等の .NET コンポーネントに置き換えることのできるコンポーネントも洗い出します。

新機能の分析と設計

アップグレード プロジェクトは、既存のアプリケーションに新しい機能を導入するのに理想的な機会です。追加したい機能がある場合は、それらの新機能の要件を文書化し、それらの機能がアプリケーションの既存の機能とどのように対話するのかを指定し、それらをアップグレード後のアプリケーションにどのように含めるのかを設計します。

このガイドは同等の機能の実現に関するものであるため、ここでは新機能の分析と設計の詳細については説明しません。ただし、アプリケーションへの追加を検討する可能性のある新機能については、第 17 ～ 21 章を参照してください。

アップグレード方法の選択

第 1 章「はじめに」では、アプリケーションを更新する必要がある場合に利用可能な方法について検討しました。アプリケーションを更新する方法としてアップグレードが適している判断をしたら、2 つのアップグレード方法を検討する必要があります。1 つは完全アップグレード、もう 1 つは段階的アップグレードです。

完全アップグレードは、アプリケーションのすべてをまとめてアップグレードすることを示します。この場合は、アプリケーション全体がアップグレードされて新しいプラットフォームで実行されるまで、コードは一切リリースされません。

段階的アップグレードでは、アプリケーションの一部を他の部分より先にアップグレードできます。この場合は比較的リスクが少なくなる反面、アプリケーションのアップグレードされた部分とされていない部分の相互運用を可能にするための追加作業が必要になります。

段階的アップグレードを、第 1 章で説明した部分的なアップグレードと混同しないようにしてください。部分的なアップグレードは、アプリケーション全体の更新に代わる方法であり、アプリケーションの一部のみを Visual Basic .NET に移行して、他の部分は Visual Basic 6.0 のまま残すことを意味します。

部分的なアップグレードか全体のアップグレードかに関係なく、アプリケーションをアップグレードすることが決まったら、アプリケーションをまとめてアップグレードするか (完全アップグレード)、部分ごとにアップグレードするか (段階的アップグレード) を選択できます。

完全アップグレード

完全アップグレードでは、アプリケーションのすべてのコンポーネントがまとめてアップグレードされ、配置されます。これは、それぞれのコンポーネントのアップグレードが並行して行われるという意味ではなく、単に、すべてのコンポーネントが .NET に移行されるまで、アプリケーションを実稼働環境に配置する作業は一切行われないということです。この方法ではかなりの労力が必要とされる場合があるため、コストが非常に高くなる可能性があります。コードを Visual Basic 6.0 から Visual Basic .NET にアップグレードする作業だけでなく、すべてのテクノロジを同等の .NET のテクノロジにアップグレードしたり、同等のものがない場合はそれに代わるテクノロジを適用したりする作業も含まれます。アプリケーションで古いテクノロジがあまり使用されていない場合は、完全アップグレードをすばやく低コストで行える可能性があります。アプリケーションが古いテクノロジに依存していればいるほど、アップグレード プロセスにかかるコストと時間が増大します。

この方法には、次のような長所があります。

- **アップグレード後のアプリケーションを .NET のテクノロジやテクニックを使用して改良できます。**この例と方法については、第 17 章「アプリケーションの改良の概要」、第 18 章「一般的なアプリケーションの改良シナリオ」、第 19 章「一般的な Web シナリオのための改良」、および第 20 章「一般的なテクノロジ シナリオ」を参照してください。

- **アップグレード後のアプリケーションを新しい .NET ソリューションに統合してより多くのシナリオで利用できます。**この統合は、新しいアプリケーションの開発に使用するプログラミング言語に関係なく行うことができます。既存のアプリケーションに変更を加える必要もありません。
- **完全にアップグレードされたアプリケーションは、新しいハードウェアや新しいバージョンの Windows で使用できます。**新しいハードウェアや新しいバージョンのオペレーティング システムに完全な .NET Framework エンジンが含まれていれば、アップグレードや再コンパイルをしなくてもアプリケーションを使用できます。

完全アップグレードには、上のような長所がある一方で、次のような短所もあります。

- **Visual Basic .NET へのアップグレードを自動的に、または簡単に行うことができない Visual Basic 6.0 の機能もあります。**Visual Basic .NET にはこうした機能に代わる (多くの場合、より優れた) 機能がありますが、この種のアプリケーションのアップグレードで同等の機能を実現するにはアプリケーションに変更を加える必要があります。こうした変更には時間とコストがかかります。
- **完全にアップグレードされたアプリケーションのユーザー インターフェイスや動作が異なる場合があります。**この場合は、アプリケーションを新しいアプリケーションとして完全にテストする必要があります。また、ユーザーの再トレーニングが必要になる場合もあります。
- **Visual Basic .NET アプリケーションでは、アップグレードするコンピュータに互換性があるバージョンの .NET Framework がインストールされている必要があります。**このため、特に大規模な企業では、IT 専門家による計画と配置の追加作業が必要になります。
- **アプリケーションのアップグレードやアップグレード後のアプリケーションの保守を担当する開発者が、アップグレードのために新しいスキルを学ばなければならない可能性があります。**これには、時間とお金の両面のコストが伴います。

完全アップグレードはコストが高くなる可能性がある反面、アプリケーションを将来に向けて準備できるため、多くの場合、最も望ましいアップグレード方法になります。重要なのは、アップグレード方法の決定を行う前にアプリケーションを十分に評価することです。多くの場合、この評価によって、完全アップグレードの利点がそのコストをはるかに上回ることが実証され、アプリケーションを Visual Basic .NET に移行することがビジネスにとって正しい決断であることがわかります。

段階的アップグレード

アプリケーションのすべての部分が単一の複雑なプロセスでアップグレードされる完全アップグレードとは違って、段階的アップグレードでは、より制御された段階的なアップグレードが可能になり、アプリケーションを部分ごとまたはコンポーネントごとにアップグレードできます。新しくアップグレードされたコンポーネントはその都度ロール アウトでき、アプリケーション全体が完全にアップグレードされるのを待つ必要はありません。この方法を利用できるのは、既存のアプリケーションが、それぞれ別々のプロジェクトとして開発された複数の別個のコン

ポーネントによって構成されている場合だけです。また、アップグレードされたコンポーネントと変更されていないコンポーネントの連携のために、相互運用のテクニックを適用する必要があります。

段階的アップグレードは、完全アップグレードに対する妥当な妥協案となります。既存のアプリケーションが大規模な場合は、段階的アップグレードが最善の選択肢となるでしょう。この方法には、次のような長所があります。

- アプリケーションを段階的にアップグレードすると、アップグレード プロジェクトの進捗やコストをより細かく制御できるようになります。また、アプリケーションが各段階の後で安定した実稼動品質の状態に戻されるため、リスクを最小化することができます。開発チームが各段階で徐々に .NET に習熟するため、チームの生産性が向上し、労力とコストの見積もりも当初より改善されます。
- アップグレード後のコンポーネントでは、多くの場合、アプリケーション全体がアップグレードされるのを待たずに .NET のパフォーマンスやスケーラビリティの機能を活用できます。
- アプリケーションの優先度の低い部分やコストの高い部分のアップグレードを無期限に延期して、結果的に部分的なアップグレードにすることができます。こうした段階的な方法では、アップグレード プロジェクトを開始した後でも、第 1 章「はじめに」の「アップグレード、再利用、書き換え、置き換えのどれを選ぶべきか」で説明したようなアップグレードや再利用が混在したソリューションを自由に選択できます。

段階的アップグレードの短所には、次のようなものがあります。

- COM 相互運用機能に必要なオーバーヘッドが追加されるため、パフォーマンスの向上が完全アップグレードほど顕著ではありません。.NET コンポーネントと COM コンポーネントの間の COM 相互運用機能は、.NET Framework で一緒に機能するマネージ コンポーネントや、Visual Basic 6.0 で一緒に機能する COM オブジェクトに比べて速度が劣ります。ただし、何万ものトランザクション要求を行うアプリケーションなど、COM メソッドの呼び出しを大量に行うアプリケーションでない限り、パフォーマンスの差は気になりません。
- COM コンポーネントに依存しているアプリケーションでは、COM オブジェクトの登録を行わなければならないほか、COM オブジェクトには .NET オブジェクトにはないバージョン管理の問題もあるため、完全に .NET にアップグレードされたアプリケーションに比べて配置やメンテナンスが困難になります。
- 段階的アップグレードでは、.NET コードと古いコードの間の相互運用のメカニズムを提供するために、ラッパーやインターフェイスを実装する必要があります。これらのラッパーはたいてい一時的なものであり、後に破棄されます。

このような短所があるとはいえ、複数のコンポーネントを持つ大規模なアプリケーションでは、段階的アップグレードが最も魅力的な方法となるでしょう。

段階的アップグレードは、さらに垂直アップグレードと水平アップグレードに分類できます。

- **垂直アップグレード**。垂直アップグレードでは、アプリケーションの単一のモジュールを n 層すべてにわたって分離してアップグレードします。アプリケーションの他の部分は変更されません。
- **水平アップグレード**。水平アップグレードでは、アプリケーションの 1 つの層全体をアップグレードします。他の層は変更されません。

段階的アップグレードに対して水平アップグレードと垂直アップグレードのどちらの手法を選択するかは、アプリケーションのアーキテクチャによって決まります。したがって、この方法の選択には、チームのソリューション アーキテクトが大きく関与する必要があります。また、アップグレードするアプリケーションによって、それぞれの方法の長所と短所がわかります。通常は、アプリケーションの設計と実装の評価から、最適な方法が明らかになります。

以降では、アプリケーションのアーキテクチャに最適な方法を選択できるように、水平アップグレードと垂直アップグレードの手法について検討します。

垂直アップグレード

垂直アップグレードでは、アプリケーションの複数の (すべてでない場合は) 層を横断する 1 つ以上のアプリケーション コンポーネントを洗い出して、アップグレードの対象として選択します。基本的には、他の部分との対話ができるだけ少ないアプリケーションの部分を抜き出して、アプリケーションの残りの部分から独立してアップグレードすることになります。

アプリケーションに、他の部分から十分に分離されている部分がある場合は、垂直アップグレードの候補として最適です。通常は、アプリケーションの残りの部分と状態情報をほとんど共有しておらず、システムの残りの部分にほとんど影響を与えずに簡単にアップグレードできる部分が、垂直アップグレードの理想的な候補となります。

以下のシナリオは、垂直アップグレードに適したアプリケーション アーキテクチャを示しています。

- **アプリケーションの層が緊密に統合されている場合**。層の間で ADO レコードセットを多用している場合は、垂直アップグレードを検討する価値があります。多くのアプリケーションでは、切断された ADO レコードセットがデータ層やビジネス層からプレゼンテーション層に渡されます。その後、プレゼンテーション層でレコードセットが反復処理されて、HTML テーブルが生成されます。この種類のアプリケーションでは、アプリケーションの特定の側面に個別に取り組むことができるため、垂直アップグレードが適しています。たとえば、ADO から ADO.NET へのアップグレードや相互運用では特別の配慮が必要になりますが、垂直アップグレードにより、ADO との相互運用の実現に伴う作業を最小限に抑えることができます。
- **アプリケーションの再設計を計画している場合**。アプリケーションの一部を新しいアーキテクチャに垂直アップグレードすると、それがアプリケーションの新しい設計のための最適なテスト ベッドになります。.NET Framework によって、新しいアーキテクチャを構築するための機能の提供が容易になります。たとえば、HttpHandlers を使用すると、これまで ISAPI 拡張機能を使用して行っていたタスクの多くを、はるかに単純なプログラミング モデルを使って行うことができます。

選択した部分のアップグレードが完了した後、新しいマネージ コードとアンマネージ コードの間のインターフェイスが残っている場合、それらは COM 相互運用機能を通じて機能します。アプリケーションの新しい部分と古い部分が連携し、データを共有して、エンド ユーザーやクライアント開発者にシームレスなユーザー エクスペリエンスを提供できるようにするには、ある程度の開発とテストが必要になります。その後、同じ垂直アップグレー

ドの方法を使用して他のアプリケーションをアップグレードしたり、元のアプリケーションのさらに多くの部分を更新したりするための、基本的なインフラストラクチャが完成します。

水平アップグレード

水平アップグレードでは、アプリケーションの 1 つの層全体を Visual Basic .NET にアップグレードします。その他の層はすぐにはアップグレードしません。1 つの層を一度にアップグレードすることによって、.NET Framework がその層に提供している特定の機能を活用できるようになります。多くの場合、アプリケーションのコードを変更する必要はなく、アプリケーションの他の層の操作に影響が及ぶこともありません。どの層を最初にアップグレードするかの決定が、水平アップグレードの最初のステップになります。

中間層の COM コンポーネントは、プレゼンテーション層にほとんど (あるいはまったく) 変更を加えずに Visual Basic .NET にアップグレードできます。水平アップグレードが適しているかどうか、また、適している場合はどの層を最初にアップグレードすればよいかを決定する際には、以下の点を考慮に入れる必要があります。

- **アプリケーションが多数の Web サーバーに配置されている。** Visual Basic .NET アプリケーションでは、各 Web サーバーに共通言語ランタイム (CLR) がインストールされていることが配置の前提条件になります。アプリケーションが Web ファーム構成の多数のサーバーに配置されている場合は、このことが問題になる可能性があります。中間層のサーバーの数が他の層に比べて少ない場合は、水平アップグレードを使用して中間層を最初にアップグレードすることを検討します。
- **アプリケーションが大量の共有コードを使用している。** Visual Basic アプリケーションの中間層で DLL やその他の種類の共有コード、定数、またはカスタム ライブラリが大量に使用されている場合は、この層を最初にアップグレードすると、アプリケーション全体のアップグレードを管理しやすくなります。ただし、その場合は、共有コードを使用している多数のコンポーネントが不安定になるため、テスト プロセスに影響が出ます。
- **アプリケーションの中間層が複雑である。** 中間層に複雑なオブジェクト階層がある場合は、それらを 1 つの単位として扱う必要があります。このような場合は、アプリケーションをどこで分離するかの決定が難しくなります。複雑な中間層の一部のみをアップグレードすると、2 つの環境の間で大量の相互運用機能の呼び出しが必要になることが多く、その結果、パフォーマンスが低下します。

中間層を置き換える場合は、以下の問題も考慮に入れる必要があります。

- クライアント コードに影響を与えることなく透過的に中間層コンポーネントを .NET コンポーネントに置き換えるには、COM コンポーネントの元の GUID と ProgID を維持する必要があります。また、COM コンポーネントを透過的に置き換えようとする場合は、Visual Basic コンポーネントによって生成されるクラス インターフェイスの置き換えを適切に処理する必要があります。
- アップグレード後の中間層コンポーネントから元の Visual Basic コードで使用される ADO レコードセットに返される ADO.NET データセットを変換する必要があります。

- 中間層コンポーネントのための相互運用アセンブリを配置する必要があります。
- この種類のアップグレードの場合、アプリケーションの層の定義によっては、Visual Basic .NET と Visual Basic 6.0 の間の相互運用が広範囲に及ぶ可能性があります。互いに通信する層の一方が .NET アセンブリになったことによって層の間の距離が広がると、アプリケーションで発生するマーシャリングが増加して、パフォーマンスに悪影響が出る可能性があります。

アップグレードの方法が決まったら、アップグレードのためのプロジェクト計画を作成することによって、アップグレード プロセスの効率と成功の可能性を高めることができます。詳細については、この後の「プロジェクト計画の作成」を参照してください。

テスト

選択したアップグレード方法に関係なく同等の機能が実現されていることを確認できるように、単体テストやシステム テストの包括的なテスト計画を作成する必要があります。このテストでは、アップグレードする個々のモジュールや層だけでなく、システム全体も対象になります。アップグレード プロジェクトにおけるテストの詳細については、第 21 章「アップグレード後のアプリケーションのテスト」を参照してください。

ソース コードのインベントリの作成

アップグレードする必要がある関連ソース コードを洗い出します。これには、ライブラリ、リソース、およびコンポーネントを含める必要があります。アプリケーションのコール グラフ ビューを使用してツリー ビュー形式の依存関係ダイアグラムを作成すると、関連する Visual Basic 6.0 プロジェクトやファイルを把握しやすくなります。実行されないコードが見つかった場合は、インベントリに含めるか、部分的にあるいは完全に除外するかを決定する必要があります。ソース コード内の依存関係や実行されないコードを特定するには、Visual Basic 6.0 Upgrade Assessment Tool などのツールを使用すると便利です。これらのツールの詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

さらに、アプリケーションが現在使用しているすべてのコンポーネントを含む、アップグレードするアプリケーション コンポーネントと、アップグレードせずに再利用するアプリケーション コンポーネントを洗い出します。サードパーティ コンポーネントについては、同等のコンポーネントが .NET にあるかどうかを確認します。

ソース コードの準備

アップグレードのためのソース コードの準備に時間をかけることによって、アップグレード プロジェクトの効率を高めることができます。方法については、この後の「Visual Basic 6.0 ソース コードの準備」を参照してください。

アップグレードの問題への対処の準備

発生する可能性がある問題をアップグレードの開始前に洗い出しておくと、アップグレード プロセスでそれらに対処するための準備を整えておくことができます。アップグレードの変更を実装する作業に入ってから問題に気付くよりその方が有効です。ここでは、問題を洗い出すための情報の入手方法について説明します。

ソース コードのメトリクスの取得

アプリケーションのソース コードからメトリクス情報を取得します。この情報には、コードの行数や、プロジェクト、フォーム、デザイナ、モジュール、クラス、コンポーネント、ユーザー コントロール、データ ソースの数などのサイズのメトリクスと、使用されている関数、プロパティ、イベントなどの使用法のメトリクスが含まれている必要があります。

また、アプリケーションで各コンポーネントが使用される頻度についてのメトリクスも取得する必要があります。コンポーネントが使用される方法や頻度を知ることは、アプリケーションのアップグレード後に各コンポーネントを .NET バージョンの同等のコンポーネントに置き換えることができるかどうかの判断に役立ちます。

図 2.2 は、在庫管理のサンプル アプリケーションのソース コードのメトリクスの一部を示しています。ここからソースコードのインベントリに関する情報を得ることができます。

Project	Total Files	Classes	Modules	UserControls	Non upgradable	Designers	Forms
CoreComps	53	45	8	0	0	0	0
StockServer	41	10	25	0	0	0	6
MarketCommAPI	35	24	11	0	0	0	0
StockNegotiator	422	415	7	0	0	0	0
InvPlanServer	30	3	27	0	0	0	0
AccessNet	262	250	12	0	0	0	0
PlanCommShared	158	140	18	0	0	0	0
SOAPConnector	27	3	24	0	0	0	0
InvTasks	115	41	30	6	0	0	38
Total	1,143	931	162	6	0	0	44

図 2.2

在庫管理アプリケーションのソースコードのメトリクスの一部

この図には、データ型とコントロールの使用回数、ソース コードのインベントリ (ツリー ビュー形式)、およびデータ型 (この例では Integer と String) が使用されている具体的な場所に関する情報の一部が示されています。評価ツールでは、これらすべての情報が読みやすい HTML レポートで提供されます。評価ツールの詳細については、第 3 章「評価および分析」を参照してください。

サポートされない機能の洗い出し

Visual Basic .NET ではサポートされない Visual Basic 6.0 の機能をすべて洗い出して、それらの機能がアプリケーションで使用される頻度を書き留めておきます。これらの機能を置き換えることのできる Visual Basic .NET の機能について調べて、そのプロトタイプを作成する必要があります。これは、追加機能が必要な箇所を特定

するよい機会にもなります。以前のバージョンの機能や Visual Basic 6.0 の古い機能と、それらの代わりに使用できる Visual Basic .NET の機能の詳細については、第 7 章「一般的に使用される Visual Basic 6.0 オブジェクトのアップグレード」、第 8 章「一般的に使用される Visual Basic 6.0 言語機能のアップグレード」、および第 9 章「Visual Basic 6.0 フォーム機能のアップグレード」を参照してください。

アップグレード レポート情報の評価

アプリケーションでアップグレード ウィザードを実行すると、レポートが生成されます。このレポートでは、発見されたアップグレードの問題や、それらがアプリケーション内で発生する回数を確認できます。それぞれの種類の問題はタスク番号で識別され、説明、関連するドキュメントへのリンク、および問題を解決するためのステップが記されています。たとえば、タスク番号 1037 は "Late-bound default property reference could not be resolved" という問題に対応し、この問題を解決するためのドキュメントへのリンクがレポートに示されます。

以下の一覧は、アップグレード レポートで明らかになる一般的な問題のよくある原因をまとめたものです。

- Visual Basic 6.0 の主な関数の中には、直接対応する .NET 関数がないものや、アップグレード後に動作が変わるものも少数ですが存在します。ただし、これらの関数についても、Visual Basic .NET で提供されている別の方法を使用することによって、アプリケーションで同じ機能を実現できます。このように別の方法を使用する場合は、通常、手動でコードを変更する必要があり、自動的にアップグレードすることはできません。
- 一部の ActiveX コントロールは Visual Basic .NET ではサポートされていません。
- アップグレード ウィザードによってアップグレード後のコードの言語の変更が検出されると、それらを手動で見直すように求めるメッセージが表示されます。たとえば、Visual Basic 6.0 の既定のプロパティがこれに当てはまります。アップグレード ウィザードは、既定のプロパティを可能な限り解決して、アップグレード後のコードで明示的に展開します。ただし、これらについては、元のコードの既定のプロパティが正しいプロパティに置き換えられていることを確認するために見直す必要があります。

アプリケーションのアップグレードの問題を分析する有効な方法は、最初に問題を次の 2 つのカテゴリに整理することです。

- **アップグレードされない問題およびサポートされない問題**。これらの問題は、サポートされない機能の存在、コントロールのプロパティやメソッドの使用、または手動で処理する必要があるステートメントの存在を示しています。このカテゴリの問題はすべて、アップグレード後のアプリケーションの正常なコンパイルの妨げになります。サポートされない機能への対処が完了すれば、このカテゴリの残りの問題の大半 (ほぼ 95%) は、単純なコード分析によって Visual Basic .NET でのソリューションを見つけることで解決できます。たとえば、COM+ インターフェイスの一部のクラスやメソッド (ObjectContext や Commit など) はアップグレードされませんが、これらと同等のメンバや機能は、.NET 名前空間の EnterpriseServices で見つけることができます。残りの 5% の問題については、アプリケーションのロジックやサポート機能の実装の若干の変更が必要になります。

- **動作の違いの問題**。これらの問題は、アップグレード後のコードに、Visual Basic 6.0 アプリケーションの元の動作とは異なる動作が含まれている可能性があることを示します。これらの問題については、コードに変更を加えなければならない可能性があるため、新しい動作がコードのコンテキストでどのような結果をもたらすのかを手動で確認する必要があります。このカテゴリの問題があってもアップグレード後のアプリケーションは問題なくコンパイルできますが、それらの問題によってランタイム エラーが発生し、アプリケーションが正しく機能しなくなる可能性があります。経験的には、このカテゴリの問題で手動で対処する必要があるのはごくわずかです。これらの問題の説明には動作の違いが記述されているため、アップグレード後のアプリケーションの単体テストの際のガイドにもなります。テスト担当者はこの情報を使用して、予想される動作の違いを把握し、それらの違いによってアップグレード後のアプリケーションで同等の機能を維持できなくなるかどうかをテストできます。このような理由から、アプリケーションの単体テストが完了するまでは、これらをコードから削除するべきではありません。

それぞれのカテゴリについて、タスク番号ごとに問題を調べ、その発生頻度を確認します。まず最初に、最も発生頻度の高い問題について、最善の解決方法と、解決に必要な労力を特定します。これらの問題は最も頻繁に発生するため、最初に取り組む必要があります。次に、その他のアップグレードされない問題やサポートされない問題の解決方法と必要な労力を特定します。最後に、動作の違いに関連する問題の解決方法と必要な労力を特定します。一般に、アップグレードされない問題とサポートされない問題は、動作に関する問題より解決の重要性が高くなります。問題の解決に要する労力を見積もるには、多くの問題を解決できるアップグレードのテスト ランを実行して、その経験から得たデータをベンチマークとして使用すると効果的です。

プロジェクト計画の作成

従来のソフトウェア開発プロジェクトと同様に、アップグレード プロジェクトの以下のフェーズの計画とスケジュールを作成することが重要です。

- プロジェクト管理 (管理タスク)
- アップグレード プロセスおよび .NET のためのトレーニング
- 開発環境のセットアップ (データベースのセットアップ、ソース コードの整理、必要なアプリケーションのインストールなど)
- コードの準備
- アップグレード プロセスの実行
- テスト ケースの変換および検証
- コード テストおよび単体テスト
- 機能テスト
- 情報の転送およびアップグレード後のコードの配置

他のソフトウェア開発プロセスと同様に、各成果物のマイルストーンを達成できるように入念な計画とスケジュールを作成します。このことは、社内および社外の顧客の期待を管理するのに役立ちます。また、アップグレード プロジェクトのスケジュールを作成する際には、プロジェクトで必要な IT サポートを受けられるように、社内で行われている他のプロジェクトのことも考慮に入れる必要があります。

コストの見積もり

ソフトウェア開発プロジェクトのコスト要件の見積もりは、ソフトウェア実装の品質や開発者のプログラミング能力にばらつきがあるために、一般に複雑な作業になります。アップグレード プロジェクトのコスト要件の見積もりについても同様です。アプリケーションのアップグレードに要する労力は、さまざまな要因によって変化します。これらの要因は、アプリケーションのコストにも同じように影響を与えます。したがって、コストの見積もりはアプリケーションによって大きく変化します。

コストの見積もりに影響を与える要因の中には、すべてのアップグレード プロジェクトに共通のものもあります。たとえば次のような要因があります。

- オブジェクト指向の手法で記述されているかどうか、アプリケーションのファミリの間で共有されている共通のサービスのセットにどの程度依存しているか、アップグレードされる部分と Visual Basic 6.0 のまま残る部分の相互依存関係はどの程度かなど、Visual Basic 6.0 コードの手法。
- 元のコードのメンテナンス履歴。それまでに頻繁に修正されてきたアプリケーションは、アップグレードの際にも頻繁な修正が必要になると考えられます。
- アプリケーション要件の複雑さ。
- アプリケーションと他のシステムとの統合の度合い。
- セキュリティの問題。
- 開発チームの規模と経験。
- アプリケーション内で使用されているサードパーティのツールセット。
- テスト インフラストラクチャと品質管理への取り組み。

コストの見積もりに影響を与える可能性があるその他の要因には、コードやアプリケーションのプラットフォームに特有の要因もあります。たとえば、RDO/DAO のデータ連結、遅延バインドされるオブジェクト、ActiveX ドキュメントなどのメカニズムや構成要素が使用されている場合は、コストの見積もりに影響します。ASP コードの場合は、レンダリング関数が (Response.Write の代わりに) 使用されている、1 つのページで複数の言語が使用されている、インクルード ファイルが入れ子になっている、16 ビットの整数に依存している、既定のパラメータが使用されている、GOSUB が使用されているなどの特性が要因に含まれます。

また、以下の作業も、アップグレード プロジェクトの最終的なコストの見積もりに影響を与えます。

- **サポートされない機能の解決**。サポートされない機能に影響を与えるさまざまな問題のコストを見積もる必要があります。たとえば、次のような問題があります。
  - 代替ソリューションの調査のコスト。
  - 選択したソリューションの開発およびテストのコスト。
  - サポートされない機能の置き換えのコストは、その機能がアプリケーションで多く使用されているほど増加します。また、個々の機能やその使われ方によっても増加します。1 つの機能が複数の場所で使用されている場合は、それぞれに注目する必要がある場合もあれば、単純な検索と置換によって解決できる場合もあります。

- **報告された問題の解決**。アップグレード レポートのすべての問題について、それを解決するためのコストを見積もる必要があります。これらのコストを見積もるには、アップグレードのテスト ランを実行することをお勧めします。アプリケーションの選択した部分のアップグレードをテストすることによって、アップグレード ウィザードでどのような種類の問題が検出されるのかがわかります。この情報を使用して、問題の解決に伴うコストの正確な情報を入手できます。また、テスト ランにより、アップグレード プロセスにおけるチームの能力への貴重な洞察も得られます。

- **テスト**。アップグレード後のアプリケーションのテストのコストを見積もる必要があります。テスト ラン (上の項目を参照) からは、必要なテストの要件に関する貴重な情報が得られます。テスト ランを実行することによって、現在のテスト プロセスを使用できるかどうか、または、アップグレード後のアプリケーションで同等の機能が実現されていることを証明するには拡張や調整が必要かを評価できます。また、プロジェクトのアプリケーションの改良フェーズで実装される追加機能のテストのコストも見積もる必要があります。

  さらに、(アプリケーションの更新の場合とは違って) 新機能だけでなくアップグレード後のアプリケーション全体をテストすることになるため、テスト プロセスにかかる時間が長くなったり、プロセスを何度か繰り返す必要が出てきたりする可能性があり、そのような場合は、通常のテストよりコストが高くなるということを認識しておく必要があります。

- **対応する .NET コンポーネントへの Visual Basic 6.0 コンポーネントの手動による置き換え**。アップグレード ウィザードで、対応する .NET コンポーネントに自動的に置き換えることができない各 Visual Basic 6.0 コンポーネントについては、手動による置き換えのコストを見積もる必要があります。その際には、アップグレード後のアプリケーションでその機能が使用される頻度を考慮に入れる必要があります。

- **サードパーティの .NET コンポーネント**。アプリケーションが依存している各サードパーティ コンポーネントについて、アップグレード後のアプリケーションで置き換える予定の .NET コンポーネントのライセンスコストを調べる必要があります。

- **2 次的な機能**。統合ヘルプ システム、インストーラ、マニュアルなどの二次的な機能がアプリケーションにある場合は、それらを新しいアプリケーションに追加するためのコストを考慮に入れる必要があります。

- **配置**。アップグレード後のアプリケーションの配置のコストを見積もる必要があります。
- **新しい要件**。アプリケーションのアップグレードが完了した後に追加する新機能の設計、開発、およびテストのコストを見積もる必要があります。プロジェクトのアプリケーションの改良フェーズのコストの見積もりは、通常のソフトウェア開発プロジェクトのコストの見積もりと似ています。

アプリケーションのアップグレードのコストを見積もる際には、従来のソフトウェア開発プロジェクトのコストの見積もりにおける組織の経験を活用するのが賢明です。従来の開発プロジェクトにおいても、プロジェクトを成功させるために、上に挙げたような一般的な要因や特殊な要因が考慮されていることは明らかです。ただし、一般的なプロジェクト フェーズの中にはアップグレード プロジェクトでは必要ないものもあり、アップグレード プロジェクトのコストの見積もりにはそれらを含めないようにする必要があるので注意してください。

たとえば、従来のソフトウェア開発プロジェクトには、分析、設計、開発、テスト、配置などのフェーズが含まれるのに対し、アップグレード プロジェクトでは、より短い分析フェーズと、アップグレード、テスト、および配置のフェーズが含まれるのが一般的です。アップグレード プロジェクトではソース コード自体が仕様になるため、標準の分析フェーズや設計フェーズが必要になることはほとんどありません。

プロジェクトのアップグレード フェーズのコストの見積もりは、主に、アプリケーション分析の結果によって決まります。つまり、アプリケーション分析で見つかったさまざまなアップグレードの問題の解決のコストによって左右されます。適切に対処された場合、これらの問題の解決のコストは、コードを書き直すコストよりはるかに低くなるはずです。アプリケーションのサポートされない機能や古い機能がコストに与える影響には特に注意を払う必要があります。これらによる影響は、アップグレード プロセスを進める前にソース コードを準備またはクリーンアップすることによって最小限に抑えることができます。また、アップグレードの問題を処理する開発者の能力も慎重に評価する必要があります。

アップグレード プロジェクトのテスト フェーズのコストの見積もりは、主に、アプリケーションのテストに使用するインフラストラクチャやアップグレードするアプリケーションの部分の品質と状態に依存します。同等の機能の実現は、アップグレード後のコードが期待される動作を示すことをテスト ケースで証明できるかどうかに直接関連しているということを覚えておく必要があります。プロジェクトの受け入れの基準も、通常は同等の機能の実現によって定義されます。したがって、自動テストを利用できる場合は、それを .NET Framework で実行できるように修正するために必要な作業を考慮に入れる必要があります。必要に応じてテスト カバレッジを拡張するための予算を割り当てます。また、アップグレード後の .NET アプリケーションでは、Visual Basic 6.0 のリリースで行われる通常の（おそらくは限定的な）テストとは違って、詳細なテストが必要になることを理解しておく必要もあります。アップグレード後のアプリケーションに完全な回帰テスト ケースを適用する際には、通常より時間がかかったり、テストを複数回繰り返す必要が生じたりする可能性があります。これにより、通常のテストのコストよりコストが高くなる可能性があります。一般に、テストのコストには 2 つの部分があります。テスト ケースの作成と、

テスト ケースの実行です。ここでは同等の機能の実現が目標であるため、元のテスト ケースが適切であればテスト ケースの作成のフェーズは省略でき、元のテスト ケースを新しいプラットフォームに合わせて調整するだけで済みます。テスト ケースの実行は、どの開発プロジェクトにおいても、プロジェクトの完了に近づくにつれて繰り返し行う必要があります。

最後となるアップグレード プロジェクトの配置フェーズのコストの見積もりは、従来のソフトウェア開発プロジェクトの配置コストの見積もりと似ています。当然ながら、.NET の配置の問題を理解および処理するためのトレーニングをチームに対して実施する必要があります。ただし、幸いなことに、一般に .NET の配置は、COM ベースのアプリケーションよりはるかに合理化されたメカニズムで構成されています。

ここでの情報をガイドとして使用すれば、アップグレード プロジェクトに必要な作業時間のコストを見積もることができるはずです。最初の見積もりが完成した後は、その数字をさらに正確なものにしたり、必要に応じて調整を加えたりできます。このプロセスは、アップグレード計画にも適用できます。このようにしてプロジェクトのコストを見積もると、従来のソフトウェア開発プロジェクトのコストとアップグレード プロジェクトのコストの最大の違いがテスト フェーズに関連していることがわかります。従来のプロジェクトでは、テスト プロセスが総コストに占める割合は約 20 ～ 30% に相当します。一方、アップグレード プロジェクトのテスト プロセスが総コストに占める割合は、容易に 60 ～ 70% に達してしまいます。ただし、アップグレード プロジェクトに必要とされる全体の作業時間は、従来のソフトウェア開発プロジェクトよりはるかに短くなります。

マイクロソフトやそのパートナーによって実施されたアップグレード テストでは、アップグレード前にコードの準備が適切に行われた場合、Visual Basic 6.0 から Visual Basic .NET へのアップグレードのコストは、開発コスト全体の 15 ～ 25% 程度になるのが一般的でした。この数字では、Visual Basic 6.0 から Visual Basic .NET に移行する Visual Basic 開発者の学習曲線も考慮されています。次に、アプリケーションをアップグレードに向けて準備するためのベスト プラクティスについて検討します。

アップグレードの準備

プロジェクトの計画の段階が完了したら、その計画の実際の実装を開始します。計画の実装における最初のステップは、必ずしも、すぐにアップグレード ウィザードを適用し、このツールによって生成される個々のエラー、警告、および問題 (EWI) を解決することではありません。この方法でもアプリケーションをアップグレードすることはできますが、よほど単純なアプリケーションでない限り、アップグレード プロジェクトの最善の方法とはいえません。

多少でも複雑なアプリケーションの場合は、最初の準備作業として、ソース コードで特殊な共通のプラクティス (既定のプロパティの使用など) を検索してコードを修正した方が良いでしょう。この修正によって、コードがアップグレード ウィザードにとってわかりやすいものになり、その結果、コードを Visual Basic .NET に正しく変換す

るためのウィザードの決定が大幅に改善されます。一見すると、この準備によって作業が増えるように見えるかもしれませんが、このように早い段階でコードを修正しておくことで、最終的にはかなりの作業を節約できます。

アップグレードに向けてソース コードを準備する前に、アプリケーションのアップグレードに使用する開発環境を準備しておくことが重要です。

開発環境の準備

ここでは、アプリケーションのアップグレードに取り組む開発者が必要とする基本的なツール群について説明します。ここで紹介するツールの一部は、他の章で詳しく説明されています。以下の概要では、必要なツールを洗い出して、それらが必要な理由と、詳細情報の参照先について説明します。

Visual Studio 6.0

Visual Studio 6.0 は、アップグレード ウィザードを使用して Visual Basic 6.0 のコードをアップグレードする前に、コードをアップグレードに向けて準備したり、確実にコードをコンパイルするために必要です。この IDE は、任意のコンピュータにインストールできます。アップグレードに必要な他のツールと競合することはありません。

ソースコードの準備の際には、この後に説明する Visual Basic 6.0 Code Advisor によって生成されるタスクの一覧を使用します。これらのタスクでは、ソース コードの修正と再コンパイルが必要になります。この作業はすべて Visual Studio. NET で行う必要があります。

Visual Studio 6.0 ソース コードと共に、この後に説明する Visual Basic 6.0 Code Advisor によって生成されるタスクの一覧を使用します。これらのタスクでは、ソース コードの修正と再コンパイルが必要になります。この作業はすべて Visual Studio 6.0 で行う必要があります。

また、アップグレード ウィザードを実行し、アップグレード レポートを確認した後、Visual Basic 6.0 に戻ってさらなる修正を加えるという作業もよく行われます。

これらの作業は別のコンピュータで行うこともできますが、同じ 1 台のコンピュータですべての作業を行う方が便利かもしれません。特に、元のコードとアップグレード後のコードを切り替えながら作業する場合はその方が便利です。

Visual Basic 6.0 Code Advisor

Visual Basic 6.0 Code Advisor は、Visual Studio 6.0 のアドインです。このツールを使用すると、Visual Basic 6.0 のソース コードをスキャンして、構成可能なコーディング標準に準拠していないプラクティスを見つけることができます。これらの標準は、堅牢でメンテナンスしやすいコードを作成するためにマイクロソフトによって開発されたプラクティスに基づいています。また、コード アドバイザでは、アップグレード ウィザードでコードを Visual Basic .NET に自動的に変換する際の障害となるコードの問題も明らかにされます。

コード アドバイザを使用するには、ダウンロードしてコンピュータにインストールする必要があります。Visual Studio 6.0 と一緒に自動的にインストールされることはありません。コード アドバイザの詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

Visual Studio .NET

Microsoft Visual Studio .NET は、Visual Basic .NET アプリケーションの開発およびメンテナンスに使用する開発環境です。Visual Basic 6.0 プロジェクトを Visual Basic .NET プロジェクトに変換するアップグレード ウィザードを実行する際にも、Visual Studio .NET を使用します。

Visual Studio .NET は、Visual Studio 6.0 では利用できない新機能や強化された機能を数多く含む高度な統合開発環境です。これまで利用してきた機能もほとんど使用できます。

複数の言語およびプラットフォームのアプリケーションを同じ IDE で、さらには同じワークスペース (またはソリューション) の中で構築できます。デスクトップ、サーバー、Web、およびモバイル デバイス用のアプリケーションを、Visual C++、Visual Basic .NET、Visual C#、および Visual J# で構築することが可能です。また、Windows と Web の両方の配置に対応した組み込みの Windows インストーラ テクノロジなど、配置を容易にするすばらしい新機能も用意されています。さらに、DLL のバージョン管理の問題を軽減あるいは解消するアプリケーションのサイドバイサイド実行などの機能によって、メンテナンスもこれまでになく容易になっています。

このバージョンの Visual Studio で追加されたもう 1 つの大きな機能は、Web サービスのサポートです。Visual Basic .NET の他のクラスを作成する作業にわずかな作業を追加するだけで、Web サービスを作成できます。Web サービスの配置およびデバッグのための高度なサポートが用意されているため、開発者は、アプリケーションの構築という最も重要な作業に集中できます。

Visual Studio .NET では使用できない機能もあります。その 1 つが、**エディット コンティニュ**です。この機能を使用すると、デバッガでコードを修正して、アプリケーションを再起動することなくそのまま新しいコードでアプリケーションの実行を続けることができます。これにより、デバッグ中にコードに小さな変更を効率よく加えることができるため、Visual Basic 6.0 では重宝されていました。Visual Studio の将来のバージョンにはこの機能が追加される予定です。

Visual Studio の詳細については、MSDN の Microsoft Visual Studio Developer Center を参照してください。

Visual Basic アップグレード ウィザード

Visual Basic アップグレード ウィザードは、Visual Basic 6.0 のソース コードとプロジェクト ファイルを Visual Basic .NET に自動的に変換するステップを実行するためのツールです。このアップグレード ウィザードは Visual Studio 2003 に組み込まれており、Visual Basic 6.0 のプロジェクトを開こうとすると自動的に実行されます。このツールをインストールまたは実行するための特別なステップは必要はありません。

アップグレード ウィザードの詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

ASP to ASP.NET Migration Assistant

ASP to ASP.NET Migration Assistant は Visual Studio .NET のアドインで、ASP のページとアプリケーションを ASP.NET にアップグレードする際に使用できます。

ASP to ASP.NET Migration Assistant では、ファイル名の変更やハイパーリンクの更新に加えて、ASP と ASP.NET の間の VBScript の構文の違いも修正されます。また、VBScript コードで明示的に型指定されておらず、既定のプロパティの解決を遅延バインドに依存している変数を、正しい型に解決する試みも行われます。既定のプロパティは ASP.NET ではサポートされていません。

ASP to ASP.NET Migration Assistant の詳細については、付録 C の「ASP のアップグレードに関する概要」を参照してください。

Visual Basic 6.0 ソース コードの準備

アップグレードの前に Visual Basic 6.0 でアプリケーションの準備を行うと、ほぼ例外なくプロセスの効率が向上します。Visual Basic 6.0 の一般的なプラクティスによって作成されるコードの中には、正しく読みやすいものであっても、アップグレード ウィザードで Visual Basic .NET に適切に変換できないものがあります。

いくつかのツールを使用すると、コードに対してアップグレード ウィザードを実行した場合に発生する問題の一覧を取得できます。こうした一覧は、元のソース コードで修正する方が容易な箇所（開発者が Visual Basic .NET より Visual Basic 6.0 に詳しい場合は特にその方が容易です）を特定するのに便利です。

アップグレードのためのアプリケーションの準備を開始する際に最初に使用するツールは、既にこの章で説明した Visual Basic 6.0 Code Advisor です。その次に使用するのが、アップグレード ウィザードです。アップグレード ウィザードでは、アップグレード中に見つかったすべてのエラー、警告、および問題を含むレポートが生成されます。このレポートには、コード アドバイザでは洗い出せなかった問題が含まれていることがよくあります。同等の機能が実現されるようにソース コードを最善の形で準備するには、アップグレード レポートで指摘された問題に対処する作業を何度か繰り返す必要がある場合もあります。アップグレード ウィザードによって報告される複数の問題が同じ問題に関連していて、その問題を解決すると多くの問題が同時に解消される場合もよくあります。たとえば、遅延バインドされる変数で既定のプロパティが使用されている場合に、変数の型を明示的に指定すると、既定のプロパティの使用に関連するすべての問題が解消されます。経験的には、アップグレードの前に Visual Basic 6.0 のソース コードで 1 つ問題を修正すると、アップグレード ウィザードによって報告される問題が平均で 5 ～ 8 個解決されます。図 2.3 は、アップグレード レポートからの情報に基づいてコードを修正する際の反復的なプロセスを表しています。

アップグレードのためのコードの準備の詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

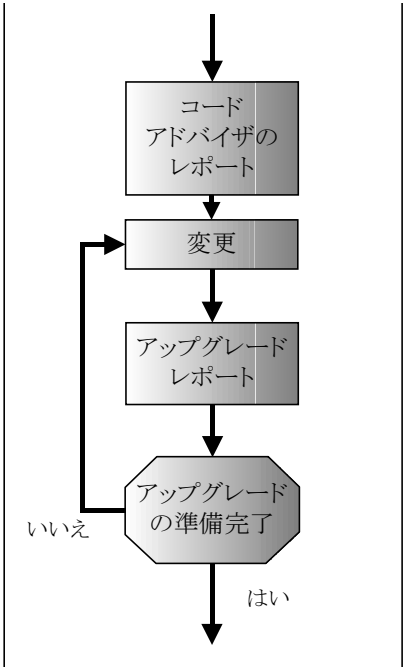

図 2.3

コードの準備のフェーズを表す図

以前のバージョンの Visual Basic で作成されたアプリケーションのアップグレード

アップグレード ウィザードは、Visual Basic 6.0 アプリケーションをアップグレードするように設計されています。バージョン 1.0 ～ 5.0 の Visual Basic で作成されたプロジェクトについては、Visual Basic .NET にアップグレードする前に、まず Visual Basic 6.0 にアップグレードする必要があります。Visual Basic 6.0 より前のバージョンで開発されたプロジェクトは、Visual Basic 6.0 IDE で開いて保存するだけでアップグレードできます。コントロールを Visual Basic 6.0 にアップグレードするかどうかを確認するメッセージが表示されたら、[はい] をクリックします。プロジェクトに Visual Basic 5.0 の ActiveX コントロールが含まれている場合は、Visual Basic 6.0 バージョンに置き換えた方が良い場合がほとんどです。これは、これらのコントロールでは Visual Basic 6.0 のコントロールとは異なるスレッド モデルが使用されているためです。この古い方のスレッド モデルは、Windows フォームではサポートされていません。

バージョン 1.0 ～ 4.0 の Visual Basic で作成された 16 ビットのプロジェクトについては、アプリケーションを Visual Basic 6.0 に変換するには追加の変更が必要な場合があります。一部の VBX コントロールは

自動的に変換されません。また、Win16 Windows API を Win32® の対応する API に置き換える必要もあります。

Visual Basic バージョン 2.0 および 3.0 では、多くの場合に追加のステップが必要になります。Visual Basic 6.0 ではテキスト形式のファイルしか開くことができないのに対し、Visual Basic バージョン 2.0 および 3.0 ではバイナリとテキストの 2 つのファイル形式がサポートされています。これらのプロジェクトをアップグレードする前に、以下の手順に従って、アプリケーション全体がテキスト形式で保存されていることを確認する必要があります。

▶ **Visual Basic 1.0 および 2.0 のファイルをテキスト形式に変換するには**

1. [ファイル] メニューの [名前を付けて保存] をクリックします。
2. [Save] ダイアログ ボックスで、[Save As Text] チェック ボックスをオンにします。

Visual Basic 1.0 ではファイルをバイナリ形式でしか保存できないため、それらのプロジェクトはすべて、いったん Visual Basic 2.0 または 3.0 で開き、テキストとして保存してから、Visual Basic 6.0 に変換する必要があります。プロジェクトを Visual Basic 6.0 に変換した後、Visual Basic .NET へのアップグレードのプロセスを開始できます。

コンパイルの確認

アップグレード ウィザードは、有効な Visual Basic プロジェクトを処理するように設計されています。このため、アップグレードを試みる前に、コードが構文的に正しいかどうかを確認することが非常に重要です。ファイルにエラーがあると、アップグレード ウィザードでコードの解析時に問題が発生して、誤解を招くようなエラー メッセージが表示されたり、さらには予期せぬ結果が発生したりする可能性があります。構文が正しいかどうかは、アップグレード ウィザードを適用する前に Visual Basic 6.0 プロジェクトをコンパイルすることにより確認できます。

プロジェクトをコンパイルすると、必要なソース コードがすべて利用可能かどうかも確認できます。アップグレード ウィザードでは、最初にすべてのコードが分析され、そこで得られた情報を使用して決定が下されます。分析が終わると、コードの変換が開始されます。不足しているファイルがあると、アップグレード ウィザードで誤った決定が下される可能性があります。

アップグレード ウィザードを適用する前に Visual Basic 6.0 で元のソース ベースをコンパイルすることによって、アップグレード ウィザードで最良の選択が行われるために必要な情報がすべて揃っていることを確認できます。アップグレード ウィザードの決定が正しくなればなるほど、同等の機能の実現に必要な手動の作業が減ります。

アプリケーションのアップグレード

コードの準備が完了したら、コードを変換して、機能的に同等な .NET アプリケーションを生成する作業を開始できます。この作業には 2 つのステップがあります。

最初のステップでは、アップグレード ウィザードを使用して、準備した Visual Basic 6.0 のコードを Visual Studio .NET に読み込みます。前述したように、アプリケーションが準備されていれば、ファイルの不足の問題やコンパイルの問題は解決されているため、このステップはほぼ難なく完了します。このステップでアップグ

レード ウィザードによって生成されるコードは、アップグレード後のアプリケーションに対して行われるその後のすべての開発のベースラインとして使用されます。この時点では、生成されたコードのことを*未加工コード*と呼びます。これは、アップグレード ウィザードから生成されたばかりで、まだ手動による変更が行われていないためです。

次のステップでは、未加工コードをコンパイルできるように変更し、完全にアップグレードされずに残っているすべてのコードを、Visual Basic .NET の対応する有効な要素に置き換えます。必要な変更の種類のサンプルについては、この章で前に説明した「アップグレードの問題への対処の準備」を参照してください。

これらの 2 つのステップの詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

アプリケーションが正しくコンパイルされ、アップグレード ウィザードによって報告されたすべての問題への対処が完了したら、テストを開始できます。次に、テストについて説明します。

テストおよび品質保証

テストでは、アプリケーションとそのコンポーネントがそれぞれの仕様どおりに動作することが確認されます。プログラムで期待どおりの処理が行われるか、プログラムが期待どおりの速度で問題なく実行されるかなど、適切なテスト プロセスでは、こうした要件のすべてが満たされていることが確認されます。

継続的なテストは、ソフトウェア開発に欠かせない重要な要素です。これは、アップグレード プロジェクトにおいても同様です。継続的なテストによって、バグを早い段階で見つけたり、その再発を防いだりすることが可能になります。また、開発者が新しいバグを追加していないことを確信できるため、ソフトウェアおよび開発作業への信頼につながります。こうしたことのすべてが時間とお金の節約になるほか、顧客が最終的な製品を受け取った後に経験するバグも少なくなるため、顧客満足度の向上にもつながります。

コードの変更には常にバグが追加される可能性が伴います。アプリケーションがアップグレードされ、新しいバージョンになると、バグを防止するための適切な回帰テストのセットが必要になります。

アップグレード後のアプリケーションのテストの詳細については、第 21 章「アップグレード後のアプリケーションのテスト」を参照してください。

配置

Visual Basic .NET アプリケーションのアップグレードおよびテストが完了したら、完成したアプリケーションの配置を開始します。.NET アプリケーションの配置の主な目的は、Visual Basic 6.0 アプリケーションの配置と同じで、アプリケーションをインストールし、クライアントのコンピュータまたは実稼動サーバーで実行することです。ただし、そのための方法、アプリケーションが持つ依存関係、および更新やセキュリティを処理するためのメカ

ニズムのすべてが、Visual Basic 6.0 の場合とは異なります。その理由となるのが、.NET アセンブリです。.NET アセンブリとは、.NET アプリケーションのための新しいパッケージ化メカニズムです。

.NET アプリケーションを配置する際にまず確認しなければならない新しい依存関係は、配置先のコンピュータに .NET Framework がインストールされているかどうかです。配置先のコンピュータに .NET Framework がインストールされていない可能性がある場合は、ユーザーが何らかの方法で .NET Framework をインストールできるようにすることを検討する必要があります。

アセンブリ

.NET アプリケーションの配置の最小単位は、アセンブリと呼ばれます。アセンブリとは、クラス、構成ファイル、リソース ファイル、およびアプリケーションが実行時に必要とするその他のファイルの集まりです。アセンブリは、実行可能ファイル (EXE) である場合も、ダイナミックリンクライブラリ (DLL) である場合もあります。

アセンブリと以前の実行可能ファイルや DLL との違いは、アセンブリには、アセンブリ自体やその中のクラスに関する追加情報 (メタデータと呼ばれる) が含まれている点にあります。アセンブリに含まれるこの追加情報は、たとえば、同じアセンブリの異なるバージョンを区別するためなどに使用されています。

アセンブリには、これ以外にセキュリティ関連の情報も含まれています。これにより、アセンブリの提供元に間違いがないこと、元の作成者によるリリース以降変更されていないことが保証されます。

プライベート アセンブリおよび共有アセンブリ

既定では、アセンブリは所属先のアプリケーションと同じディレクトリにインストールされます。この種類のアセンブリをプライベート アセンブリと呼びます。プライベート アセンブリはアプリケーション ディレクトリにインストールされるため、同じコンピュータにインストールされている他のアプリケーションに干渉する可能性はありません。

また、共有アセンブリを作成することもできます。共有アセンブリは、グローバル アセンブリ キャッシュと呼ばれるシステム全体のアセンブリのリポジトリにインストールされます。これらのアセンブリは、複数の独立したアプリケーションが使用できます。共有アセンブリには、使用されるディスク領域やメモリを節約できるというメリットがあります。

バージョン管理

グローバル アセンブリ キャッシュには、同じアセンブリの複数のバージョンを簡単にインストールできます。これらのアセンブリは、グローバル アセンブリ キャッシュによって個別に維持されます。名前と GUID (COM への公開のため) は同じでバージョンのみが異なるアセンブリをインストールしても、アプリケーションがコンパイルされたバージョンのアセンブリのみが実行時にリンクされることが .NET Framework によって保証されます。アプリケーションが使用する DLL のバージョンを、アプリケーションを再インストールせずに変更するには、そのアプリケーションの構成ファイルに特殊なディレクティブを挿入する必要があります。これにより、互換性のない

バージョンの DLL がインストールされるため、結果として、コンピュータに既に存在するアプリケーションに新しいアプリケーションが干渉する問題が解消されます。

グローバル アセンブリ キャッシュでは、バージョンが同じでコンパイル対象のアーキテクチャやプロセッサが異なるアセンブリも区別されます。たとえば、32 ビット プラットフォーム用と 64 ビット プラットフォーム用の同じ .NET アセンブリをインストールすると、アプリケーションに最適なアセンブリがグローバル アセンブリ キャッシュによって自動的に選択されます。さらに、64 ビット プロセッサのさまざまなモデルごとにアセンブリを最適化することもできます。この場合も、グローバル アセンブリ キャッシュによって最適なアセンブリが自動的に選択されます。

サイドバイサイド実行

同じアセンブリの複数のバージョンを同じコンピュータにインストールして、異なるアプリケーションの間で共有できるようになったことにより、.NET アプリケーションの複数のバージョンを簡単に同じコンピュータにインストールして、互いに干渉することなくサイドバイサイドで実行できるようになりました。

構成

配置やサイドバイサイド実行が容易になったことには、.NET アプリケーションでは一切の構成情報を Windows レジストリに保持する必要がないという事実も関係しています。このため、レジストリが原因でアプリケーションのバージョン間の衝突が起こることはありません。

.NET アプリケーションの構成情報は、アプリケーション本体と同じディレクトリにある XML ベースの構成ファイルに保持されます。これ以外に、同じアプリケーションのすべてのバージョンに適用されるファイルおよびコンピュータレベルで存在するファイルがあります。

配置の更新

既に述べたように、インストールされたアプリケーションは、通常はコンパイルされたのと同じバージョンのアセンブリにのみリンクされます。ただし、アプリケーションの構成ファイルに構成設定を追加することにより、一緒にインストールされたバージョンの DLL の使用を中止して、更新された DLL の使用を開始するように指定することもできます。このリダイレクトが行われるのは、明示的に指定された場合だけです。これにより、サービス パックやアプリケーションの更新によるインストール後のアプリケーションの変更が容易になります。この状況では、古いバージョンの DLL を削除して新しいバージョンをインストールするだけでは十分ではありません。それではアプリケーションが実行されなくなるだけです。

Microsoft Windows インストーラ

Visual Studio .NET には、さまざまな種類のアプリケーションのインストーラを作成できる特殊なプロジェクト テンプレートが用意されています。

プロジェクトの選択後、インストール手順をカスタマイズできます。たとえば、レジストリの修正、ファイルの種類の追加、ショートカットや任意のファイル (ドキュメントなど) の追加、インストールを続行するためにインストール先

のコンピュータで満たされていなければならない前提条件の指定などを行うことができます。さらに、独自のインストール操作やアプリケーション固有のダイアログ ボックスをインストール プロセスに追加することもできます。

Visual Studio .NET に用意されているセットアップ プロジェクトでは、通常は Windows インストーラ ファイル (.msi) が作成されます。このファイルは、CD や Web サイトに保存したり、ネットワーク経由のインストールのために共有ネットワーク ディレクトリに保存して配布できます。

Visual Studio .NET には、さまざまな種類のアプリケーションをインストールできるように、以下の配置プロジェクトが用意されています。

- **セットアップ プロジェクト**。任意のファイルや操作を追加できる汎用の Windows アプリケーション セットアップ プロジェクトです。
- **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ルを目的とするプロジェクトで、登録や構成の問題を自動的に処理できます。
- **マージ モジュール プロジェクト**。共有コンポーネントを他のインストール プロジェクトに組み込んで配置する場合に使用するプロジェクトです。
- **セットアップ ウィザード**。これはプロジェクトの種類ではなく、既存の Visual Studio .NET アプリケーション プロジェクトから初期のセットアップ プロジェクトを作成できるウィザードです。作成したプロジェクトは後からカスタマイズできます。
- **CAB プロジェクト**。CAB プロジェクトを使用すると、ファイルやプロジェクト出力の集まりを 1 つにまとめて、圧縮したり他のコンピュータにコピーしたりできます。この種類のプロジェクトは、ActiveX コントロールをパッケージ化する際によく使用されます。

アプリケーションの改良

テストが完了すると、元の Visual Basic 6.0 アプリケーションと機能的に同等なアプリケーションを使用できるようになります。しかし、多くの場合、アプリケーションをアップグレードするそもそもの理由は新機能の追加にあります。同等の機能を実現できたら、その構想した結果を実現するためにアプリケーションの改良を開始できます。新しい開発はすべて Visual Basic .NET で行われます。

アプリケーションの改良とは、元のアプリケーションの仕様の範囲を超えてアップグレード後のアプリケーションを強化することです。これには、以前のバージョンの Visual Basic で実装するより Visual Basic .NET で実装する方が簡単な機能を追加する作業が含まれることもよくあります。中には、初めてアプリケーションに実装できるようになる機能 (スレッド化など) もあります。

アプリケーションの改良の詳細については、第 17 章「アプリケーションの改良の概要」、第 18 章「一般的なアプリケーションの改良シナリオ」、第 19 章「一般的な Web シナリオのための改良」、および第 20 章「一般的なテクノロジ シナリオ」を参照してください。

アップグレード プロジェクトの管理

管理に関しては、アップグレード プロジェクトと標準の開発プロジェクトの管理方法の違いを理解することが重要です。ここでは、アップグレード プロジェクトをできるだけスムーズに進めるためのベスト プラクティスやテクニックを多数紹介します。

ここで紹介するプラクティスとテクニックは、数多くの Visual Basic アップグレード プロジェクトの経験に基づいています。マイクロソフトとそのパートナーは、何百万行という Visual Basic 6.0 のコードを Visual Basic .NET にアップグレードする中で、貴重な情報を得てきました。ここでは、この経験が多くのプロジェクトに役立てられるように、その教訓の一部を紹介します。

変更管理

アップグレード プロジェクトでは、通常のアプリケーション開発プロジェクトに比べてソース管理の問題が複雑になります。現在ソース管理システムがない場合は、購入を検討する必要があります。ソース管理システムを使用すれば、同じファイルのセットで作業する複数のプログラマが互いの作業内容を上書きしてしまう問題を大幅に軽減できます。

ソース管理の使用

ソース管理システムとは、同じソース ファイルのセットで作業する複数の開発者から成るグループの作業を管理するソフトウェア プログラムです。これにより、開発者が同じソース ファイルに同時に変更を加えることによって貴重な作業内容が失われる事態を防ぐことができます。

サポートされるファイルの種類やユーザー数などの機能はソース管理システムによってさまざまですが、以下に示す基本的な機能はほとんどのソース管理システムに備わっています。

- **ソース ツリー全体と関連ファイルの格納**。ソース管理システムには、アプリケーションのすべてのソース コードと関連ファイルを格納するディレクトリ構造が含まれています。このディレクトリ構造を*ソース ツリー*と呼びます。ソース ツリーはソース管理システムによって集中管理され、すべての開発者がアクセスできるサーバーに格納されます。
- **ファイルへの排他アクセス**。ユーザーは、ファイルを*チェックアウト*することによって、一時的な排他書き込みアクセスを取得できます。チェックアウトされているファイルを変更できるのは、ファイルをチェックアウトしたユーザーだけです。ファイルがチェックアウトされている間は、他のユーザーがそのファイルをチェックアウトしたり変更したりすることはできません。チェックアウトされる前のバージョンのファイルについては引き続き、読み取りのみ行うことができます。ファイルの変更を終えたユーザーがファイルをチェックインすると、他のすべてのユーザーがその変更を利用できるようになります。その後、別のユーザーがそのファイルをチェックアウトして作業できます。このしくみにより、開発者が別の開発者によって行われた変更を上書きする可能性を大幅に低減できます。

- **バージョン履歴**。ファイルが変更され、アプリケーションの開発が進められていく間、ソース管理システムではファイルの履歴が維持されます。このため、任意の時点におけるアプリケーションの状態を確認できます。この機能によってソース管理システムは、開発者が 1 人の場合でも、ソース コードの以前のバージョンを追跡するための有効なツールとなります。たとえば、ほとんどのシステムではソース ファイルのバージョンにラベルを付けることができるため、特定の顧客が持っているバージョンに対応するソース管理のバージョンを簡単に抽出できます。まったく同じバージョンを使用すれば、バグをより簡単に再現できます。
- **ソース ブランチの作成**。ほとんどのソース管理システムでは、ソース ツリーに複数のブランチを作成できます。各ブランチでは、別のプログラマのチームが、アプリケーションの別のバージョンの作業を同時に行うことができます。これらのバージョンは、ある特定の段階で再び 1 つのブランチへとマージされるのが一般的です。たとえば、先ほどの顧客のケースでは、顧客のバージョンのアプリケーションにはバグがあるため、より新しいバージョンに置き換える必要があると考えられます。しかし、アプリケーションの開発が継続されていてまだ安定していない場合や、現在のバージョンに含まれている新機能を顧客に提供したくない場合は、顧客が持つバージョンのソースからブランチを作成して、そのバージョンの中でバグを修正することができます。その後、このバグの解決策を後からメインのソース ブランチにマージできます。

ソース管理システムを使用することの利点を以下にいくつか紹介します。

- プログラマが互いの作業内容を上書きすることによって失われる作業内容を最小化できます。
- 各ソース ファイルを誰が使用しているのかを常に把握できます。また、それらのファイルに加えられた複数の変更を調整できます。
- 同じアプリケーションで作業を行う複数のプログラマの間の調整が容易になります。
- アプリケーションの以前の任意のバージョンにいつでもアクセスできます。
- 1 人のプログラマによるソース ファイルの変更を開発チームの他のメンバにリリースする方法を制御できます。

多重チェックアウトの危険とファイルのマージ

ソース管理システムの管理者によって、同じソース ファイルを複数の開発者が同時にチェックアウトすることが許可される場合もあります。このような場合は、2 人の開発者が同じファイルに別々の変更を加えてしまう可能性があります。多くのソース管理システムはこの状況に対応しており、これらの変更をマージするためのツールが用意されています。通常は、開発者がお互いに調整して、それぞれソース ファイルの別の箇所を変更することによって影響を最小限に抑えるようにします。

アップグレード ウィザードによって生成されたコードで開発作業を開始する時点では、アプリケーションのソース ファイルの多くはコンパイルされません。また、複雑な依存関係があって、依存先のファイルが先にコンパイルされないとコンパイルできないファイルもあります。

ソース ファイルの多重チェックアウトを許可することによってこの問題を解決したくなります。多重チェックアウトを許可すれば、複数の開発者がそれぞれにソース ファイルで作業して、依存関係のあるファイルがコンパイルされるように問題を解決することができます。しかし、Visual Basic のアップグレード プロジェクトの場合、特にプロジェクトを初めてコンパイルする最初の段階では、これは避けるべきです。同じファイルで作業する 2 人の開発者は、ほとんどの場合、ファイルをコンパイルできるようにするためにはファイルに同じ変更を加えなければならなくなります。その結果、作業が重複したり、場合によっては、コードの同じ行が複数の開発者によって変更されるためにマージが困難になったりします。

ソース ブランチ

既に説明したように、多くのソース管理システムでは、ソース コードに独立したブランチを作成して、アプリケーションの複数のバージョンを独立して開発できます。会社のソース管理ポリシーにもよりますが、たいていは、アップグレード プロジェクトの間にいくつかの異なるソース ブランチを使用することになります。たとえば、次のようなソース ブランチが使用されます。

- **元のソースのブランチ**。通常は、元のソース コードに専用のブランチがあります。元の Visual Basic 6.0 アプリケーションの開発を継続する必要が生じる可能性がある場合は、アプリケーションの元のバージョンを専用のブランチに残して、アップグレード用の新しいブランチを作成する必要があります。
- **コードの準備のブランチ**。既に説明したように、コードの準備では、元の Visual Basic 6.0 ソース コードに変更を加えて、アップグレード ウィザードで使用するための準備を整えます。アプリケーションが有効な Visual Basic 6.0 アプリケーションであることは変わりませんが、ソース コードが変更されるためにバグが混入する可能性があります。

  開発センターによっては、ソース コードに対して行われる変更が厳しく制限されていたり、それらの変更のテスト方法やさらにはチェックイン方法についての規則があったりする場合もあります。たとえば、多くの企業では、ファイルをチェックインする前の段階で早くも最初のテストが要求されることが一般的になっています。これらのテストは、チェックイン テストと呼ばれます。

  このような場合には、コードの準備専用のソース ブランチを作成すると便利です。そうすれば、元のアプリケーションへのすべての変更を開発のメイン ブランチとは別にすることができ、また元のアプリケーションの開発も続けられます。
- **未加工コードと同等の機能のブランチ**。コードの準備が完了したら、アップグレード ウィザードをコードに適用します。これにより、新しい Visual Studio .NET ソリューションが作成されます。既に説明したように、この新しいソリューションを未加工コードと呼びます。

  未加工コードは、元のコードのソース ツリーを含むソース コード ブランチや、準備されたコードのソース ツリーを含むソース コード ブランチから独立した、アプリケーションのまったく新しいバージョンです。アップグレード ウィザードの出力が満足できるものになったら、その後の開発はすべて未加工コードで行われます。

通常は、元のアプリケーションの新しいブランチを使用するのではなく、まったく新しいソース ツリーを作成します。この新しいソース ツリーは、アップグレード プロセスの次のステップとなるコンパイルの基礎として使用されます。

実際、アップグレード ウィザードによってソース ツリーが生成された後のアップグレードの残りのステップは、通常は未加工コードを使用して行われます。その最初のステップとなるのは、元の Visual Basic 6.0 アプリケーションと同等の機能を実現することです。その次のステップでは、新機能の追加や既存機能の強化を含むアプリケーションの改良が行われます。これらの作業はすべて連続しており、他の開発プロセスとまったく同様に、同じソース ツリーで行うことができます。

場合によっては、未加工コードに対して加えられた変更のレポートを作成する必要が生じることもあります。既に述べたように、ソース管理システムには、元の未加工コードのファイルのコピーや、任意の時点のソース コードのコピーを生成するためのメカニズムが備わっています。これらのレポートを使用して、未加工コードのバージョンと機能的に同等なバージョンのファイルを比較することにより、ソース コードに対して加えられた変更を正確に特定できます。

アップグレード ウィザード使用後のプロジェクトの管理

Visual Basic アップグレード ウィザードの使用後には、コンパイルされないソース コード ファイルがアプリケーション内に多数あります。ファイルのコンパイルを管理するための最善の方法を特定するのは、プロジェクトが大規模で複雑な場合には特に、難しい課題となることがあります。明確な目標があっても、その目標を達成する方法も明確であるとは限りません。アップグレード ウィザード使用後のソース コード ファイルのコンパイルの管理に役立つヒントを以下に示します。

- 作業の進行の妨げになる箇所を早い段階ですべて洗い出しておくようにします。1 人の開発者が 1 つのファイルで作業している間、チームの残りのメンバが作業できなくなることがないようにします。
- 他のファイルへの依存関係が最も少なく、アプリケーションへの影響が最も大きいファイルから作業を始めます。
- 同じ問題が複数の開発者によって何度も修正されることがないようにします。
- 開発者がコードに不要な変更を加えることによって混入されるバグの数をできる限り減らします。
- 反復的で手間のかかる作業はできる限り回避します。
- アプリケーションで同等な機能を実現するために必要となる作業をできる限り少なくします。

複雑な依存関係を持つコンパイルされないファイルの処理

未加工コードのファイルを無作為に選んでコンパイルしようとすると、まだコンパイルされていない他の多くのファイルのオブジェクトへの参照が見つかります。このような場合は、そうした参照先のファイルがコンパイルされるまで、そのファイルの作業は完了できません。複数の開発者がそれぞれ別のファイルで作業する場合は、

共通のファイルへの依存関係のために、そのファイルを修正している開発者の作業が終わるまで、他のすべての開発者が待たなければならなくなることがあります。

こうしたシナリオが起こる確率をできるだけ小さくするには、他のファイルへの依存関係が最も少ないファイルを特定して、それらのファイルから修正を開始すればよいのは明らかです。もう 1 つ考慮に入れる必要がある特性は、最初に作業するファイルに依存しているファイルの数です。最初に作業する必要性が最も高いファイルは、他のファイルに一切依存しておらず、他の多くのファイルが依存しているファイルです。このようなファイルから作業を開始すれば、そのファイルがコンパイルされるときに、同時にコンパイルできるようになるファイルの数が最も多くなります。

評価ツールを使用すると、多くの貴重な情報に加えて、コンパイル時にファイルをどのような順序で処理すればよいかのアドバイスも得られます。評価ツールの詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

ファイルのコンパイルの優先順位付け

すべてのファイルがコンパイルされないとアプリケーションで同等の機能が実現されないのは明らかです。開発者の生産性を維持するために、たとえ一部の機能が一時的に失われることになっても、ファイルをコンパイルできるように修正する作業を優先します。言うまでもありませんが、ファイル間に依存関係があると、他のファイルに依存しているファイルのコンパイルはなかなか思うようには進みません。

先に別のファイルがコンパイルできるようにならないとファイルの作業を進められないという場合はよくあります。また、特定のファイルのグループの作業を同時に行わなければならなくなることもあります。これは、ファイルが互いに依存している相互依存関係のためである場合もあれば、依存先のファイルの準備がまだできていないにもかかわらず作業を進めなければならないファイルがあるというだけの場合もあります。このような場合は、通常、反復的なアプローチが、複数のファイルのコンパイル作業を同時に行うための最善の方法となります。

最初のステップでは、ファイル内にあるすべての*ローカル*のコンパイルの問題に対処します。ローカルのコンパイルの問題とは、そのファイル自体の変更によって引き起こされ、そのファイル自体の変更によって修正できる問題です。たとえば、構文の問題、機能の欠如の問題、既定のプロパティの使用などがこれに当てはまります。外部のファイルの問題によって引き起こされている問題については、このステップでは無視してかまいません。

各開発者はそれぞれのファイルで見つかった問題をすべて修正し、修正後のファイルをチェックインして他の開発者が使用できるようにする必要があります。

ローカルのコンパイルの問題への対処がすべて完了したら、*グローバルな (外部の)* コンパイルの問題への対処を開始できます。これらは、別の開発者が編集している別のファイルの問題によって引き起こされている問題です。グローバルなコンパイルの問題は、コンパイルしようとしているファイルの外部にあるファイルを編集しないと解決できないコードの問題です。こうした問題の多くは、開発者がそれぞれのファイルでローカルのコンパイルの問題を解決することによって自動的に解決されます。残った問題については個別に対処する必要があります。

開発者がローカルの問題を解決してファイルをチェックインしたら、他の開発者は、それぞれのファイルでローカルの問題を改めて探す必要があります。あるファイルの変更によって別のファイルの問題が修正されて、かつてはグローバルな問題だった問題が事実上ローカルの問題に変わることがよくあります。

このアプローチでは、ファイルがしだいに正常にコンパイルできる状態へと近づいていきます。このプロセスは、ファイルの外部依存関係が少ないほど容易になります。

ファイルをコンパイルできるようにする早道

ファイルをできるだけ早くコンパイルできるようにするには、まだ完全にアップグレードされていない機能をコメント アウトする方法がよく使用されます。このようにコードをコメント アウトしてコンパイルできるようにする場合は、問題の解決を事実上先送りすることになります。これには、良い面と悪い面があります。

回避しようとしている問題が GUI の表面的な修正である場合や、ファイルやアプリケーションの動作の中で分離された機能に関連している場合は、コメント アウトしても問題ありません。しかしそれが、メッセージのログへの記録、データベースのデータの挿入や変更、特定のデータ財務計算の追加など、通常のテストでは欠けていることに気付くのが難しい基本的な機能である場合は、コメント アウトにはコストがかかる可能性があります。というのも、機能が欠けていることに気付かないまま放置される可能性があり、大分後になってから気付くと修正のコストが高くなるからです。

実際にファイルを早くコンパイルできるようにするために機能をコメント アウトする場合は、そのことを忘れないようにすることが重要です (忘れると、アプリケーションに新たなバグが混入することになります)。そのためには、コメント アウトした内容を記録しておく必要があります。それには、Visual Studio .NET のタスク一覧を使用するか、アプリケーションでその箇所のコードが実行されるたびにメッセージが出力されるようにします。コメント アウトした機能に気付き、後から対処できるような形で記録しておきます。

分断攻略

ほとんどの大きな作業と同様に、古典的な分断攻略の方法は有効なアプローチとなります。この概念をアップグレード プロジェクトに適用するには 2 とおりの方法があります。最初の方法では、プロジェクトまたはファイルに基づいて作業を割り当てます。個々のプロジェクトまたはソース コード ファイルを開発者に割り当て、割り当てられた開発者は、そのファイルを最初にコンパイルする作業に専念します。既に述べたように、評価ツール (このガイドのコミュニティ サイトからダウンロードできます。詳細については、「Preface」の「Feedback and Support」を参照してください) を使用すると、ファイルの作業を行うための最適な順序を特定できます。ここでは、すべてのファイルをコンパイルできるようにすることを目標にする必要があります。作業の割り当てをプロジェクト、ファイルのグループ、または個々のファイルのどのレベルで行うかは、ファイルのサイズによって異なります。

すべてのファイルがコンパイルできるようになると、通常は、解決する必要がある機能の欠如に関する問題が残ります。コメント アウトされたコードや、動作が異なる可能性があるコードとしてマークされた未加工コードがこれに該当します。これらの問題は、通常、Visual Studio .NET の [Tasks] ウィンドウのタスク一覧や、ファイルをコンパイルできるようにする作業の間にコメントアウトされた EWI やその他のコードの一覧に含まれています。

この段階では、引き続きプロジェクトやソース コード ファイルに基づいて作業を割り当てることもできますが、代わりに EWI やタスクに基づいて作業を割り当てることも考えられます。その場合は、開発者に特定の EWI を割り当てたり、選択したタスクの一覧を割り当てたりできます。たとえば特殊な知識を必要とする問題がある場合は、それらの問題を適切な能力を持つ開発者に割り当てることができるため、この方法が有効です。この方法には、開発者に Visual Basic .NET の特定の側面を専門に扱わせることによってプロジェクトの進行を加速できるというメリットがあります。

たとえば次のような作業を、コンパイルするファイルを割り当てる代わりに開発者に割り当てることができます。

- **型が指定されていない変数や Object 型に指定されている変数の型指定。** Visual Studio .NET に付属のアップグレード ウィザードでは、明示的に型指定されていない変数やなんらかの汎用の型を割り当てられている変数の型の推測は行われません (ArtinSoft Visual Basic Upgrade Wizard Companion ではこれが行われます。Upgrade Wizard Companion の詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください)。このため、不適切な型を持つ変数が生成されたり、既定のプロパティの使用を認識できなかったりします。重要な作業として、Object 型にアップグレードされている変数をそれぞれのファイルで見つけて、より適切な型を割り当てる必要があります。
- **正しく変換されなかったコードの手動による修正。** 一般に、アップグレード ウィザードによって正しくアップグレードされなかった機能がファイルにある場合、その機能はアプリケーションの他のファイルでも見つかります。たとえば、印刷 API などの API や、Visual Basic .NET には直接対応する要素がない Windows フォームのコレクションが使用されている場合、それらの機能はアップグレードされず、アプリケーションの複数のファイルで見つかります。これらの問題を修正するには、開発者に特定の API を割り当てます。割り当てられた開発者は、アプリケーション内でそれらの API が使用されている箇所を調べてアップグレードします。
- **ActiveX/OCX コンポーネントのテスト。** ActiveX コンポーネントや OCX コンポーネントの多くは、アップグレード ウィザードによって生成されるラッパーを使って機能しますが、中には機能しないものもあります。どのコンポーネントが機能しなくなるのかがアップグレード プロジェクトの最初にわかっていると便利です。アップグレード プロジェクトを開始する前に、.NET 環境では機能しなくなる ActiveX コンポーネントと OCX コンポーネントを特定します。この作業は、コンポーネントのバグの修正を単純にするために、アプリケーションのコンテキストの外部で行います。これにより、問題が発生した場合に、その原因がコンポーネントとラッパーまたはコンポーネントと .NET Framework の間に互換性がないことによるものであり、アップグレード後のアプリケーションの内部の問題ではないことが保証されます。

- **実行されないコードや冗長な宣言、構造体、および定数の削除。** DLL の機能を使用する際には、必要な DECLARE ステートメントがすべて含まれているコードを Web ページやヘルプ ファイルからコピーして貼り付けることがよくあります。そうすると、そこで提供されている機能は必要ないために、大量の不要なコードが含まれることになります。このような場合は、多数の冗長な外部参照によってコードが混乱したり、アップグレードが余計に困難になったりする可能性があります。これらのステートメントがコードの準備の間に削除されていなかった場合は、任意の時点で未加工コードから削除してかまいません。

アップグレード プロジェクトの障害

コードを入念に準備した場合でも、アップグレード プロセス中に障害に直面することがあります。ここでは、アップグレード プロジェクトの一般的な障害とその対処方法について検討します。

継続的に開発されている Visual Basic 6.0 アプリケーションの管理

企業のテクニカル サポート部門やカスタマ サービス部門の活動が活発で、アプリケーションのユーザー ベースも大きい場合は、アプリケーションの開発が継続的に行われる場合があります。そのような場合は、一般に、アップグレードの間も開発を中断することはできません。この問題に対しては、ソース管理システムでソース ブランチを作成して (あるいは単純にソース コードのコピーを作成して)、アプリケーションが機能的に安定しているある特定の時点のバージョンを使用してアプリケーションをアップグレードするのが主な解決策となります。

アップグレード後のアプリケーションは、Visual Basic 6.0 アプリケーションの古いバージョンと機能的に同等な Visual Basic .NET アプリケーションになります。元のアプリケーションに対してその時点までに行われたすべての変更が十分な詳細さで記録されていれば、すぐに新しい機能を追加して、.NET アプリケーションを最新の状態にできるはずです。

実際には、このシナリオが当てはまるような場合は、ユーザー ベースをアプリケーションの古いバージョンから新しいバージョンに移行するプロセス自体が大変な作業となるため、アプリケーションのアップグレードの間に報告される小さなバグはその陰に隠れてしまいます。

チェックインされるソース コードに対する厳密なコンパイルの要件への対処

ここでの内容が当てはまるのは、既にソース管理システムを使用していて、アプリケーションのソース コードに加えることのできる変更が企業のポリシーによって制限されている場合だけです。このようなポリシーでは、コードをチェックインする前にチェックイン テストを行うことが要求されたり、チェックイン後にコストの高い品質保証サイクルが要求されたりします。このような場合は、ソース管理システムで独立したブランチを作成し、それを使ってコードの準備を行います。これにより、ソース コードに加える変更はメイン ブランチとは無関係になるため、ソースコードを必要に応じて自由にチェックインできます。

共有ファイルを持つ複数の Visual Basic 6.0 プロジェクトの処理

プロジェクト間で共有されているファイルは、アップグレード ウィザードでは共有されていることが認識されないために問題が発生する可能性があります。アップグレード ウィザードはこれらのファイルを、使用しているプロジェクトごとに繰り返しアップグレードします。アップグレード後のファイルは Visual Basic .NET で同じ名前と名前空間を持つため、この問題は簡単に見つけることができます。しかし、コンパイル時まではわかりません。

この場合は、新しい Visual Studio .NET ソリューションを作成して、手動の見直しが必要なすべてのプロジェクトをそこに含めます。そこで、以前は共有されていて現在は別々のプロジェクトに繰り返し出現しているファイルを確認し、それらのファイルが実際に所属するプロジェクトを特定できます。その後、この情報に基づいて参照を修正できます。

この作業を行う際には、アップグレードされたコンテキストの違いによって共有ファイルにわずかな違いが生じている可能性があるので注意してください。

Visual Basic 6.0 プロジェクトの循環依存関係の処理

循環依存関係はプログラミング プラクティスの問題を示すものですが、実際に存在します。これらは、互いに依存している複数のプロジェクトのグループによって引き起こされます。たとえば、2 つのプロジェクトのそれぞれにもう一方の DLL への参照が含まれている場合、それらのプロジェクトは互いに依存しています。

このような場合は、アップグレード後のプロジェクトでも同じ相互依存の問題が発生します。この問題を修正するには、依存関係の根本を突き止めて、循環依存にならないように解決する必要があります。通常は、一方のプロジェクトからもう一方のプロジェクトにコードを移動することによって解決できます。これらの変更は、アップグレードのコードの準備のフェーズで行うことをお勧めします。そうすれば、アップグレード後のアプリケーションのソースコードが改善されます。

部分的なアップグレードの管理

部分的なアップグレードを実行する (アプリケーションの一部のみをアップグレードして、他の部分は Visual Basic 6.0 のまま残す) ことを決断した場合は、このプロセスのソース コード管理をどのように処理するかを決定する必要があります。ここで問題になるのは、明らかに、アプリケーションの一部のコンポーネントは削除されて新しい .NET コンポーネントに置き換わるのに他の部分はそうならないことです。

第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」では、Visual Basic 6.0 と Visual Basic .NET の相互運用について詳しく検討します。部分的なアップグレードを行う場合は、事前にこの章を参照して、相互運用の概念と制限について理解を深めてください。

部分的なアップグレードのソース コードの管理に関しては、選択できる方法が 2 つあります。たいていは、どちらの方法も適用できます。

1 つ目の方法では、COM オブジェクトや DLL などのコンポーネントを .NET の DLL で置き換えます。DLL を使用するコードを変更せずに置き換えを行うことができる場合は、プロセスがはるかに容易になります。この場合は、最終的に古い DLL を置き換える新しい .NET プロジェクトを作成するだけで済みます。現在の DLL と

は別に新しい DLL を作成し、準備ができたら新しい DLL の使用を開始するだけなので、ソース管理システムを使用しているかどうかにかかわらず、単純なプロセスになります。

2 つ目の方法では、COM オブジェクトや DLL などのコンポーネントを使用するクライアントを、対応する .NET の要素に置き換えます。元のクライアントとは別にクライアントを開発し、準備ができたら新しいクライアントを使用します。共有コンポーネントの変更を回避できる場合は、プロセスがはるかに容易になります。コンポーネントを変更しなければならない場合は、コードの準備の際ではなく、アップグレードを開始する前の元のプロジェクトに変更を加えるようにしてください。

履歴情報の使用

すべてのアプリケーションはそれぞれ異なります。アプリケーションでどの種類のアップグレードの問題がいくつ発生するのかを知ることは不可能です。ここでは、何らかの指針を示すために、既に行われた数多くのアップグレードの成功例に基づく一般的な統計をいくつか紹介します。ここでの統計はすべて、最初に Visual Basic 6.0 で Visual Basic .NET へのアップグレードのための準備が行われたプロジェクトのものである点に注意してください。最初に Visual Basic 6.0 でアプリケーションの準備を行わない場合、実際の数字はこれよりはるかに大きくなると予想されます。ただし、アップグレード ウィザードによって大量の問題が報告されても恐れることはありません。原因と修正方法さえわかれば、たいていの問題はごく簡単に解決できます。

問題全体の 88% は、最も一般的な 5 つのアップグレードの問題によって引き起こされています。これらの問題と、その全体に占める割合を表 2.1 に示します。

表 2.1: アップグレードの上位 5 つの問題

問題	割合
Could not resolve default property of object "<objectname>"	52%
Property/method was not upgraded	13%
Property/method/event is upgraded but has a different behavior	12%
COM 式はサポートされていません	7%
Null/IsNull() の使用が見つかりました	4%

1.

これらはすべて*些細な*問題です。些細な問題とは、よく発生するが簡単に修正できる問題です。これらのプロジェクトでは、問題の 41% がフォームのデザインで発生し、59% がモジュール、クラス、フォームの分離コード、およびその他のプロジェクト アイテムで発生しています。平均すると、モジュール、クラス、およびフォームの分離コードにはコード 106 行に 1 つ、フォームにはコントロール 5 つに 1 つ、アップグレードの問題が発生します。

アップグレードを実行するためのベスト プラクティス

プロジェクトの実行の制御を維持したり、アップグレードに関連する潜在的なリスクを管理したりするために役立つ推奨事項を以下に示します。

- 一般にソフトウェア アプリケーションは企業の生きた資産であるため、アプリケーションのコードの開発を凍結することは困難です。アプリケーションのアップグレードはできるだけ、アプリケーションのコーディングを重要なバグの解決のみに制限できるときに行うようにしてください。そうすることによって、アップグレード プロセスを加速し、アップグレード後のアプリケーションだけでなく元のアプリケーションにまで変更を加えなければならなくなる可能性を抑え、アップグレード中のソース コードの管理を単純化することができます。開発を中断することが不可能な場合は、特定の時点のソース コードのコピーをアップグレードし、アップグレード中またはアップグレード後にアプリケーションに適用しなければならない変更やバグの修正を記録する必要があります。
- Visual Basic 6.0 アプリケーションを理解している開発者およびテスト担当者と、.NET Framework の経験を持つ開発者およびテスト担当者から成るアップグレード チームを作るようにします。必要な業務のスキルは備えているが技術的な知識は必ずしも持ち合わせていない他のチーム メンバが知識を身に付ける間、この組み合わせのチーム メンバがアップグレードの技術的な側面を処理します。
- アップグレード プロジェクトを優先課題に位置付け、開発者がアップグレード プロジェクトに専念できるようにします。これにより、チームが新しいコード ベースに焦点を絞り、その作業を開始できるようになります。
- 手動で処理する必要があるアップグレードの問題を把握し、分類します。こうした問題の多くはすばやく機械的に修正できますが、中には注意が必要な問題もあります。さまざまなカテゴリの問題を修正するためのチームの能力について、プロジェクトのスケジュールへの影響の可能性も含めて評価します。
- アップグレードの情報をチーム全体で共有します。1 つの解決策が、アップグレード中に見つかった複数の問題のすべての事例に当てはまる場合もあります。開発者が、他の開発者によって実施されている解決策について、常に最新の情報を把握できるようにします。そのためには、知識ベース ソフトウェアを使用する、イントラネットでドキュメントを公開する、定期的にチームでミーティングを行うなどの方法があります。このほか、企業で使用しているその他の知識普及の手段も使用できます。
- 同等の機能の実現を目標にします。Visual Basic .NET アプリケーションが Visual Basic 6.0 アプリケーションと機能的に同等であれば、新機能を追加するための確かな基礎を確保できます。アップグレード後のアプリケーションが、.NET Framework の推奨されるプラクティスに完全には従っていなかったり、完全には最適化されていなかったとしても、その機能を検証することはできます。その後、制御された形で、細かな調整、セキュリティの実装、リモート配置、および .NET Framework によって提供されるその他のさまざまな機能を使ってアプリケーションを強化する作業を開始できます。

- アップグレード中に問題を解決する際には、維持しようとしているのは機能であってコードではないということを頭に入れておく必要があります。元のアプリケーションとまったく同じにするためだけに古い機能を新しいプラットフォームで再実装しようとするより、特定の部分のコード全体を、新しいプラットフォームで提供されている機能に置き換えた方が良い場合もあります。
- テスト フェーズを部分的なテストと完全なテストに分けます。部分的なテストは、コード テストと単体テストとして考えることができます。部分的なテストでは、コンポーネント、クラス、特定のメソッドなど、アプリケーションの分離可能な要素に焦点を絞ります。完全なテストは、システム テストとも呼ばれ、個々のコンポーネントの機能のテスト、アプリケーション全体の機能のテスト、およびアプリケーションと他のアプリケーションとの統合のテストから構成されます。部分的なテストでは、システム全体を統合する前にエラーを見つけることができます。元のアプリケーションにも存在していたにもかかわらず検出されていなかったバグが、アップグレード後のアプリケーションで見つかるのも珍しいことではありません。そうしたエラーは、2 つの環境のアプリケーションの実行方法の違いによって明らかになります。これらのバグは、完全なシステム テストではなく単体テスト (限定的なテスト) の段階で見つかるとより簡単に解決できる場合があります。
- テスト ケースは、アップグレード プロセスの進捗状況を定量化するための優れた測定ツールとなります。また、アプリケーションで同等の機能が実現されたことを確認することもできます。また、他のソフトウェア プロジェクトと同様に、アプリケーションの品質を高レベルに保つうえでも役に立ちます。
- テスト専用のリソースを確保します。専任のテスト担当者がいない場合は、開発者を、アップグレードを実行するグループと、アップグレード後のコードをテストするグループに分けます。これにより、バグの発見の効率が改善されます。コードより業務に詳しいと考えられるメンバにテストを任せるようにします。

陥りやすい問題の回避

アップグレードの最初の段階では、アプリケーションを再設計したい気持ちは抑えるようにしてください。コードをアップグレードに向けて準備する際には、アプリケーションを改善するための方法が簡単に見つかります。この段階でコードを修正し始めると、すぐにアップグレード プロセスを制御できなくなり、スケジュールや予算を維持できなくなります。コンパイルや同等の機能の実現の段階では、再設計に目を奪われず、それらの作業に集中する必要があります。そのアルゴリズムはそれまで機能していたものなので、今のところはそれを堅持します。ただし、その間も、アプリケーションで同等の機能が実現された後に改善できる箇所を把握しておく必要があります。

また、アップグレード ウィザードによって生成されたコードが機能する場合は、それに代わる要素が .NET にあるというだけの理由でコードを改善しようとするのは避けてください。これらの作業を行うのは、アプリケーションの改良の段階に入ってからです。

前にも述べたように、アップグレード プロジェクトの第 1 の目標は同等の機能の実現であるべきです。アプリケーションの強化や最適化の作業を開始するのは、同等の機能が実現されてからです。

まとめ

同等の機能が実現されれば十分なのか、それともアプリケーションに新しい機能を追加する必要があるのかの決定は、アップグレード プロジェクトを開始する前に検討する必要がある重要な決定です。新機能を追加する前にまず同等の機能を実現する作業から開始することはできますが、その後の開発作業を円滑に進めるための下準備をアップグレードの間に行えるように、将来の改良の計画を事前に立てておく必要があります。

アップグレード方法を決定し、アップグレードの計画を立てて、関連するコストと労力を見積もるためには、さまざまなアップグレード方法と、それらをどのようなときに適用すればよいのかを理解することが重要です。アプリケーション全体をアップグレードするのか、それとも一部のみをアップグレードするのか (アプリケーションの一部が Visual Basic 6.0 のまま残るため、相互運用のテクニックを使用する必要があります)、すべてのアップグレードを一度に行うのか、それとも一度に 1 つずつ段階的にコンポーネントをアップグレードするのかといった問いに対しては、アップグレードを開始する前の段階で答えを出さなければなりません。

最後に、アップグレード プロジェクトの成功と効率は、明確に定義された段階から成る系統的なアプローチにかかっています。この章で概要を示したプロセスからは、実際のアップグレード プロジェクト計画の準備や実施に必要な知識を身に付けることができます。

詳細情報

Visual Studio の詳細については、MSDN の Microsoft Visual Studio Developer Center を参照してください。
http://msdn.microsoft.com/vstudio/

3

評価および分析

はじめに

評価および分析では、通常、アプリケーションを Microsoft .NET Framework にアップグレードするための最適な方法、およびアップグレードに要するコストを決定します。この章では、こうした情報を収集するための最適な方法について説明します。

「アップグレードの目的の評価」では、Microsoft Visual Basic .NET へのアップグレードに伴う一般的な期待、およびこうした期待の背後にある現実について説明します。

「データの収集」では、アップグレード プロセスに関する重要な決定を下す際に準備しておくべき情報、およびこうした情報を収集するための最適な方法について説明します。

「アプリケーション分析」では、Visual Basic 6.0 Upgrade Assessment Tool (このガイドに付属) を検討し、このツールを使用して、アプリケーションの統計的、構造的、および機能的な情報を取得する方法について説明します。また、こうした情報を利用して、アプリケーションの最適なアップグレード方法を決定する方法についても解説します。

「労力とコストの見積もり」では、アップグレード プロジェクトに要する時間とコストの見積もり方法について説明します。

各セクションでは、必要な情報の収集方法、およびプロジェクトにおけるこうした情報の価値について説明します。こうした情報の最適な利用方法を理解することで、プロジェクトのコストとリスクを削減し、投資収益 (ROI) を増加させることができます。

アプリケーションのアップグレードのための最適な方法とコストを決定するには、アップグレードの範囲を決定して、適切にプロジェクトを計画し、.NET Framework にアップグレードしたアプリケーションを十分にテストす

る必要があります。アップグレードに関する前提条件と期待についてすべて認識し、これらがすべて正しいことを確認する必要があります。この章を読むことで、この方法について理解できます。

プロジェクトの範囲と優先順位

プロジェクトの範囲を定義するために必要な情報は、アプリケーション分析から得られる最も重要な情報であるだけでなく、アップグレード プロジェクトの労力とコストに最も大きな影響を与える情報でもあります。プロジェクトの範囲を定義する際には、ここで説明する検討事項が役立ちます。

アップグレードが必要なコンポーネントの特定

.NET にアップグレードするアプリケーション部分の決定は、さまざまな要因に影響されます。たとえば、サーバーサイド テクノロジの統合を必要とする一方で、クライアントのアップグレードはあまり重要でない場合があります。あるいは、インターネットまたは Web サービスとの統合の強化や、この機能の追加が必要な場合もあります。このようなシナリオでは、強化または追加する機能に直接関連する部分のみをアップグレードする必要があります。

これらの要件は、アプリケーションの特定のコンポーネントを Visual Basic .NET にアップグレードする必要があることを意味します。通常はコンポーネント間に依存関係があるため、他のコンポーネントもアップグレードする必要があります。ただし、Visual Basic 6.0 と Visual Basic .NET の相互運用性を利用できるため、必ずしもすべてのソース コードをアップグレードする必要はありません。相互運用性の詳細については、第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」を参照してください。アップグレード方法の詳細については、第 1 章「はじめに」を参照してください。

また、アップグレードするアプリケーション部分は、確保できる開発者の数、アップグレードに要する時間、アップグレードの完了までに必要な労力など、利用可能なリソースによって制限される場合があります。

この章の後半では、アプリケーションのコンポーネント間の依存関係を追跡する方法について説明します。アプリケーションのサイズの測定、および個々のコンポーネントのアップグレードに必要な労力の見積もりについても説明します。この情報は、アップグレード プロジェクトの範囲を決定する際に役立ちます。

古いコンポーネントの特定

アップグレードするアプリケーションを評価する際、現在使用されていない機能が見つかることがあります。これは一連の開発チームによって数年かけて開発された古いアプリケーションの場合に特に当てはまります。ビジネスの焦点または運用手順が変更されたため、アプリケーションに追加された機能が後で使用されなくなることがよくあります。多くの場合、アップグレードの前にこれらの機能を削除できます。これにより、作業量を大幅に削減することが可能です。通常は、機能が古ければ古いほど、Visual Basic .NET でサポートされな

い (したがって Visual Basic .NET へのアップグレードが困難な) 以前のバージョンの Visual Basic の動作に依存している可能性が高くなります。

使用されなくなった (アップグレードの必要がない) 機能を特定するために利用できる分析手法がいくつかあります。このような手法には、ユース ケース分析と入出力分析があります。この章の後半では、これら手法について説明します。

アップグレードに関する期待の管理

期待の管理は、あらゆる開発プロジェクトにおいて重要です。アップグレード プロジェクトでは特に重要になります。なぜなら、アップグレードに対する考え方や、アップグレードを行う理由が人によって異なるからです。この章の後半では、人々がアップグレード プロジェクトに対して抱く一般的な期待と、アップグレードによって実現しようとする目的に関する一覧を示します。この一覧は、プロジェクトに関わるさまざまな人々の期待を管理し、予期せぬ問題の発生を回避するのに役立ちます。

計画

効率的なプロジェクト計画を作成するには、実行すべき作業、作業に携わることのできる人員、および作業に要する時間を正確に把握する必要があります。この点については、この章の前半の「プロジェクトの範囲と優先順位」で一部説明しました。以降では、計画作成時に留意すべきその他の検討事項について説明します。Visual Basic 6.0 Upgrade Assessment Tool (以下「評価ツール」) を慎重に適用すると、こうした検討事項を明らかにすることが可能です。

大幅なアップグレード作業を要するコンポーネントの特定

コンポーネントの中には、.NET に容易にアップグレードできるものがあります。たとえば、サードパーティ コンポーネントのソース コードにアクセスする必要がない場合があります。一方、アップグレードにかなりの作業を要する機能に依存するコンポーネントもあります。その例として、DAO (データ アクセス オブジェクト) や RDO (リモート データ オブジェクト) を多数使用してデータにアクセスするコンポーネントが挙げられます。これらのコンポーネントのアップグレードにはより大きなリスクが伴うため、アップグレードの計画段階で特別な注意を払う必要があります。

この章の後半では、評価ツール (このガイドに付属) に関する詳細、および Visual Basic .NET へのアップグレードが困難なコンポーネントをこのツールで特定する方法について説明します。

アップグレードするコード量の測定

コメント行や空行を除いた実際のコードの行数は、アプリケーションの複雑さを評価するための有効な指標です。実際には、コードの行数から労力が直接見積もられることはありません。その代わり、見積もりに影響する

その他の要因に対する重みとして使用されます。詳細については、この後の「時間とコストの見積もり」を参照してください。

たとえば、コードの行数は、アプリケーションのインベントリの作成、アップグレード順序の定義、Visual Basic アップグレード ウィザードの最初の実行時に生成されるアップグレード レポートのレビューなどに要する時間に影響します。

評価ツールを使用すると、アプリケーション全体に含まれるコードの行数、およびアプリケーションを構成する個々のファイルやコンポーネントそれぞれに含まれるコードの行数がわかります。

アップグレード順序の決定

大規模なアップグレード プロジェクトでは、アプリケーション内のコンポーネント間やファイル間に複雑な相互依存関係が通常存在するため、最初にアップグレードするファイル、コンポーネント、およびモジュールを判断できない場合が少なくありません。物理的には、アプリケーションを構成するファイルに依存関係が含まれるため、アプリケーションの一部をデザイン変更する場合を除き、事前に定義した順序に従ってファイルをアップグレードする必要があります。ただし、論理的および戦略的には、ファイル間の依存関係にかかわらず、コンポーネントごとにアプリケーションをアップグレードする方が適切であるといえます。

アプリケーションに含まれる個々のコンポーネントのアップグレード順序は、ユーザー固有のビジネス ニーズや技術的な要件によって決まります。決定された順序に疑問が残る場合は、評価ツールを使用すると、コンポーネント間の物理的な依存関係を明らかにし、コンポーネント レベルでの決定に役立てることができます。予期しない依存関係がコンポーネント間に見つかったり、アプリケーションのデザインまたは実装にかかわる問題が検出されることも珍しくありません。コンポーネントのアップグレード順序を決定した後で、コンポーネントを構成するファイルの数に応じて、個々のファイルのアップグレード順序を決定する必要があります。依存するファイルの数が最も少ないファイルからアップグレードを開始し、それから依存関係の階層に沿って作業を進めていくことをお勧めします。評価ツールを適用することで、各ファイルの正確な依存関係が検出されるだけでなく、その情報に基づいたアップグレード順序が提供されるため、ファイルをアップグレードする際は、評価ツールが大いに役立ちます。

必要な技術的専門知識の評価

実行すべき作業とその順序を計画する際は、特定した各作業の担当者も決定する必要があります。評価ツールが MainReport.xls ファイルに生成するレポートは、こうした意思決定に役立ちます。このファイルの [Config – Resources] タブには、推奨されるリソース カテゴリの一覧が、各カテゴリの説明および 1 時間あたりのコストと共に表示されます。各カテゴリごとに既定のコストが表示されますが、これらの値は企業の実際のコストに合わせて変更できます。

また、MainReport.xls ファイルでは、プロジェクトのアップグレードに必要な作業の種類に関するほぼ完全な情報が提供されます。さらに、これらの作業に必要なリソース カテゴリも提供されます。

たとえば、アップグレード ウィザードによって検出されたエラー、警告、または問題 (EWI) のそれぞれについて、推奨されるリソース (開発者やアーキテクトなど) が関連付けられます。これらのリソースは、EWI を解決する担当者に必要な最低限の経験レベルを表します。これらは評価ツールが提供する既定の推奨値ですが、好ましくない場合は [Config – EWI] タブで変更できます。

また、手動アップグレードを必要とする Visual Basic 6.0 コードのさまざまな機能について、推奨されるリソースが MainReport.xls の [Config – General] タブに表示されます。これらの値についても変更可能であり、推奨値が好ましくない場合は独自のリソースを選択できます。

さまざまなアップグレード作業の処理に必要なリソースの推奨値は、これまでに行われた多数の Visual Basic 6.0 アップグレード作業を基に決定されます。ただし、企業内で利用可能なリソースに応じて状況が異なる場合があります。レポートが構成可能であるため、ユーザーの環境に合わせてリソースを調整できます。

時間とコストの見積もり

時間とコストは、一般的に最も重要な要因であり、従来最も決定が難しい要因でもあります。時間とコストは、利用可能なリソース、開発者のスキル、経験、知識、およびアプリケーションの品質とサイズに依存します。評価ツールによって生成される MainReport.xls ファイルでは、さまざまなアップグレード作業に必要なリソースの種類、各リソースのコスト、およびリソースが必要とする作業時間に関する情報が提供されます。これらの情報は、コストと労力の見積もりに役立ちます。

MainReport.xls ファイルは以下に示す複数のタブから構成されており、アップグレード作業に要するコストと労力の見積もりが各タブに表示されます。

- Effort – Total : 労力とコストの見積もりの合計が表示されます。
- Effort – By Resource Type : 各リソース カテゴリの労力の見積もりが表示されます。
- Effort – By Task : 各アップグレード作業の労力の見積もりが表示されます。
- Effort – EWI : 各 EWI の労力の見積もりが表示されます。

これらの各タブの詳細については、この後の「労力とコストの見積もり」で説明します。

これらのタブに表示される見積もりは、他のタブに含まれる構成可能な値に基づいています。前のセクション (「必要な技術的専門知識の評価」) で説明したように、各リソースのコストを [Config – Resource] タブで変更できます。指定した値が作業レポートで自動的に使用されます。[Config – EWI] タブおよび [Config – General] タブでは、さまざまなアップグレード作業に必要なリソースの種類と時間が表示されます。これらの値はニーズに合わせて変更できます。変更した値は、これらの値に依存する作業レポートに反映されます。したがって、

アップグレード プロジェクトのコストと労力の見積もりが簡素化されることになります。なぜなら、[構成] タブで必要な変更を指定するだけで、変更内容が作業レポートに自動的に反映されるからです。

MainReport.xls ファイルでは、さまざまな見積もりが構成可能であるため、時間の経過に伴い見積もりの正確性を向上させることができます。開発者が参加するアップグレード プロジェクトの数が多ければ多いほど、アップグレードのさまざまな側面に対処するために必要なコストと労力についてより多くの情報を収集できます。こうした情報を利用することにより、今後のプロジェクトについて、評価ツール レポートのコストと労力の見積もりを調整できます。

優先順位の決定

アプリケーション分析におけるもう 1 つの重要な目的は、アップグレード プロジェクトの優先順位の決定です。これが重要なのは、優先順位を決めることで、作業に焦点を絞り、決定を導き出すことができるためです。優先順位は、アップグレード プロジェクトの根底にあるビジネス目標や技術的な目的によって異なります。ただし、アプリケーションの内容や動作に関する分析結果、および利用可能な時間と予算にも一部依存します。

アプリケーションの内容に応じて、優先順位にリスク管理が含まれる場合があります。この場合、アプリケーションの特定部分に関するパイロット プロジェクトや初期テストが必要になります。

1 つのシナリオ例として、大規模なユーザー ベースを処理できるように、パフォーマンスや拡張性の改善を含む新しいビジネス要件が挙げられます。この要件では、現在のアーキテクチャとボトルネックに関する詳細な分析、およびパフォーマンスと拡張性を改善できる Visual Basic .NET の機能についての十分な理解が必要です。第 1 章および第 14 章では、これらの機能の一部について説明しています。ボトルネックを優先させると、アップグレードを迅速に行うことができます。

もう 1 つの例として、新しい Web サービス インターフェイスをできるだけ時間をかけずに追加するというシナリオが挙げられます。相互運用性を利用したソリューションを採用できる場合は、アプリケーションを部分的にアップグレードするだけで十分です。なぜなら、必要なのは Web サービスにコンテンツを提供するコンポーネントのみだからです。他のコンポーネントは、Visual Basic 6.0 のままにしておくことができます。ここでは追加サービスのみを優先しているため、時間とコストという点で最も効率的なソリューションが実現されます。

最後の例は、統合を優先する場合のシナリオです。この場合、部分的なアップグレードは選択できません。また、プロジェクトの初期段階において、アップグレードがより困難なコンポーネントに注意を払う必要があります。なぜなら、そのようなコンポーネントがプロジェクトの遅延要因になる可能性があるからです。

アップグレードの目的の評価

ここでは、Visual Basic 6.0 から Visual Basic .NET にアップグレードする一般的な理由、およびアップグレードプロジェクトに伴う期待について説明します。

開発者の期待と、スポンサー、ユーザー、およびクライアントの期待を明らかにするため、アップグレードの理由を明確にする必要があります。開発者は、アプリケーションのアップグレードが正当な理由に基づくものであることを期待します。また、プロジェクトの目的が達成可能であること、および目的の達成に必要な作業量が明確になっていることを期待します。

こうした期待の大部分は 1 つの単純な事実に起因しています。Microsoft Visual Studio .NET の自動化ツールを使用してアプリケーションをアップグレードすると、新しいアプリケーションのコードの大部分が元のアプリケーションのコードをベースにして生成されます。これはよくある例です。ただし、コードがまったく生成されない場合もあります。コードが生成されない例として、その動作が元のアプリケーションのソース コードではなく、現在利用できないサードパーティライブラリに含まれるものである場合が挙げられます。

以降では、アップグレードの一般的な目的、およびこれらの目的の実現性に関する人々の期待について説明します。それぞれの場合について、期待の背後にある理由を説明します。

ビジネス目標

アップグレード プロジェクトは、企業のビジネス目標に影響を与えます。ここでは、アップグレード プロジェクトに関するいくつかの重要な検討事項と期待について、ビジネス目標の観点から説明します。

組織内の混乱を最小限にする

自動アップグレードを実行すると、元のアプリケーションのルック アンド フィールを新しい Visual Basic .NET アプリケーションでそのまま利用できます。一般的に、これは新しいアプリケーションのためのトレーニングが必要ないという期待につながります。期待どおりになることもありますが、さまざまな理由からこのレベルで同等のアプリケーションを実現できない場合もあります。最も一般的な原因は、.NET で利用できないサードパーティコンポーネントです。サードパーティコンポーネントが利用できない場合は、独自のコンポーネントを作成するか、Visual Basic .NET に類似したコンポーネントに置き換える必要があります。これにより、アプリケーションの動作が若干異なる可能性があります。

パフォーマンスを変更すると、アプリケーションの動作に明白な違いが生じ、ユーザーを混乱させる可能性があります。たとえば、.NET アプリケーションのパフォーマンスを変更すると、ユーザーがこの違いに困惑し、トランザクションが正常に終了したかどうか判断できなくなる場合があります。

評価が終了した後で、不要な機能や使用しない機能を特定します。使用されなくなったコードと機能を削除することで、アップグレードに要する時間を短縮し、新しいアプリケーションをクリーンアップするための良い機会となります。ただし、削除した機能に対するドキュメント化されていない依存関係が他の機能に含まれていることがあるため、このような決定は慎重に行う必要があります。また、一部のユーザーが未報告の機能を使用していたり、当初とは異なる目的でそれらの機能を使用している場合があります。たとえば、本来の目的とは異なる 2 次的なレポートから情報を読み取ることがあります。これは、単に本来使用すべきレポートよりも、このレポートのほうが簡単または迅速に作成できるからです。

その他の混乱要因として、必要とされるテスト手順や配置手順の変更が挙げられます。新しいアプリケーションは Visual Basic 6.0 をベースにしていないため、テスト手順の変更が必要になる場合があります。これは、テストが自動化されている場合に特に当てはまります。また、配置要件の変更により、配置手順の更新が必要になる場合もあります。

最後に、新しいアプリケーションには元のアプリケーションのビジネス ロジックの大部分が含まれますが、これはプログラマが何も学ぶことがないことを意味するわけではありません。Visual Basic .NET は新しい言語です。新しい構文規則や API を備えており、新しいプログラミング方法論をサポートしています。また、Visual Basic 6.0 プログラマの一般的なプラクティスの多くはサポートされなくなりました。元のアプリケーションのアップグレードは、Visual Basic 6.0 プログラマがこの新しい言語を習得するための良い機会となります。ただし、アップグレードを開始するには、Visual Basic .NET についてある程度の知識があることを前提とします。

既存の知的資産の活用

アプリケーション内にカプセル化されたビジネス ロジックは、時間と資金の大規模な投資を表します。そのため、これらのビジネス ロジックをすべて破棄し、新しいビジネス ロジックをゼロから作成することは、一般的に最良の解決方法とはいえません。書き換えではなくアップグレードを選択する理由の 1 つは、元のコードに対して行った投資を可能な限り活かすことにあります。アプリケーションをアップグレードする場合、既存のビジネス ロジックを手作業で新しい言語に書き換える必要はほとんどありません。アップグレードの大部分が自動的に行われるため、元のアプリケーションの開発に携わっていない開発者でもアップグレードを行うことができます。

リスクの削減

以前のアプリケーションが正常に動作し、ユーザーやビジネスのニーズを満たしている場合、新しいプラットフォームに直接アップグレードしても、それに伴うリスクはほとんどないと考えるのが一般的です。同一のビジネス ロジックを単に新しいプラットフォームに移すだけだからです。

古いコードは、冗談で「動くコード」と定義されることがよくあります。このようなコードは、長期間にわたって使用、テスト、および改良されてきました。コードが動くのであれば、書き換える必要はありません。また、新しいコードを記述すると、検出および修正しなければならない新しいバグが発生する可能性があります。元のコードの大部分が自動的にアップグレードされるため、アプリケーションをゼロから作成する場合と比べて、記述し

なければならない新しいコードが少なくて済みます。これにより、アプリケーションに新しいバグを追加してしまうリスクが減少します。

投資収益の最大化

ビジネスでは、直接的または間接的に投資収益を生み出す新しいプロジェクトのみが承認されます。このため、アップグレード プロジェクトには投資収益に関する暗黙の期待が常に伴います。

非常に多くの変数について考慮する必要があるため、従来、ROI の計算は複雑な作業でした。このガイドでは、ROI の計算方法に関するいくつかのポイントを示します。ただし、計算に使用する変数の多くは、ユーザー固有のビジネス プラクティスとアップグレード後のアプリケーションの使用方法に依存します。簡単に言うと、ROI は、アップグレードによって得られた収益の金額とアップグレードによって削減された金額の合計から、アップグレードにかかったコストを差し引いたものです。

ROI = 収益 + 削減した金額 – アップグレードのコスト

アップグレードによって得られる収益は、アップグレード後のアプリケーションと、ビジネスでのアプリケーションの使用方法に完全に依存します。ただし、多くの場合、アプリケーションがアップグレードの直接の結果として提供する新しい機能、サービス、およびパフォーマンスにも関連します。アップグレードのコストは、アップグレードを開始する前に見積もることができます。この点については、後で詳細に説明します。

アプリケーションを独自に分析し、アップグレードによって強化される機能がアプリケーションの運用コストやメンテナンス コストにどのように影響するかを調べる必要があります。Visual Studio .NET によって新しいテクノロジが提供されるため、アップグレード後のアプリケーションが新しい市場を開拓したり、より多くのユーザーに使用される可能性もあります。

.NET Framework と Visual Studio が提供する機能により、ソフトウェアの開発およびメンテナンスが大幅に改善され、アプリケーションの開発およびメンテナンスにかかる総コストが削減されます。アプリケーションの配置、構成、およびメンテナンスが容易になり、必要な管理リソースが少なくて済みます。XML ベースの構成ファイル、新しい配置の実現、および他への影響が少ない配置のためのアクティビティは、配置によってインストールされている他のソフトウェアにダメージを与えたり、他のソフトウェアからダメージを受ける可能性が減少することを意味します。こうした側面から総コストが減少します。

パフォーマンス、セキュリティ、および拡張性が改善されるため、アプリケーションの応答性と安全性が向上します。これにより、ダウンタイムの削減とトランザクションの増加が実現し、ユーザーの満足度が向上します。これらの側面はすべて生産性の向上とビジネス プロセスの効率化につながります。

最後に、マイクロソフトは Visual Basic 6.0 以前のバージョンに対するサポートの終了を予定しているため、Visual Basic 6.0 に関する十分な知識を持つ開発者は今後少なくなると予想されます。これにより、開発者の確保がより困難になり、アプリケーションのメンテナンス コストがますます増加します。また、多くの企業では開発者のトレーニングに多額の投資を行っており、開発者のスキルが最新の状態にあるかどうかについて知りたいと考えています。また、企業が外部の開発者を必要とする場合は、外部から人材をすぐに確保できるか

どうかについても知りたいと考えています。開発者をすぐに確保できない場合は、アップグレードを行わないほうがコストが高くつくという単純な事実から、アップグレードによる ROI が生じることになります。

同等の機能の実現

同等の機能については、第 2 章で詳細に説明しており、アップグレード プロジェクトにおける主要な目的であると提言しています。実際、アプリケーションを新しいプラットフォームに移行するプロジェクトでは、最低限の同等の機能が当然のように期待されます。

ほとんどの場合、これは達成可能な正しい目的であるといえます。ただし、障害がないわけではありません。ユーザーの管理下になく、.NET で利用できないコンポーネントが提供する機能は、書き換えるか、その他のサードパーティコンポーネントや社内で作成したカスタム コンポーネントで置き換える必要があります。

同等の機能を必要とする場合は、すべてのサードパーティ COM コンポーネントとライブラリが .NET と互換性があるかどうか、または .NET バージョンのコンポーネントが利用可能かどうかを調べて、こうした同等の機能が実現可能であることを確認してください。

Visual Basic 6.0 に対する正式なサポートの終了への対応

第 1 章で説明したように、マイクロソフトによる Visual Basic 6.0 の正式なサポートは 2008 年 3 月で完全に終了します。多くの企業は、ベンダがサポートするプラットフォームやツールセットの下で、IT 部門の業務やソフトウェアの運用が行われるようにすることを大きな目的としています。

第 2 章で説明したアプリケーションの部分的なアップグレードではなく、全体的なアップグレードを選択した場合は、Visual Basic 6.0 に依存することがなくなります。部分的なアップグレードを選択した場合は、アプリケーションの特定の部分のみをアップグレードし、Visual Basic 6.0 と Visual Basic .NET との相互運用メカニズムを利用して、アップグレードした部分とアップグレードしない部分を連携させます (詳細については、第 13 章「Windows API の使用」を参照してください)。ただし、このアプローチを選択した場合は、アプリケーションの一部で Visual Basic 6.0 が当然必要になります。マイクロソフトのサポート計画に従う場合は、最終的にアプリケーション全体を Visual Basic .NET に移行することになります。

技術的な目的

アップグレード プロジェクトの背後には技術的な目的が存在します。ここでは、アップグレードを行う技術的な目的のいくつかについて説明します。

新機能の追加

Microsoft .NET Framework などの新しいフレームワークにアプリケーションをアップグレードすることにより、システムを最新のテクノロジで強化できます。これにより、Web Enabling、Web サービス、XML、ADO.NET、ASP.NET など、ますます拡大するユーザーやビジネスのニーズを満たすことができます。また、重要なアプリケーションが、サポートされない古い環境に依存しなくなるため、長期的なサポートと展開のための準備が整うことになります。

これは一般的な期待であり、多くの場合、.NET に移行する第 1 の理由です。ただし、これらの機能は自動的に実現されることを覚えておいてください。キー ワードは *可能にする* です。.NET はこれらのテクノロジの利用を可能にします。アプリケーションを .NET にアップグレードすると、.NET のすべての機能と可能性を利用できるようになります。ただし、これらのテクノロジを利用するための計画を作成し、この計画を実装した開発サイクルを開始する必要があります。

パフォーマンスと拡張性の向上

パフォーマンスと拡張性の向上は、.NET が提供する重要な拡張機能です。多くの場合、アップグレードしたアプリケーションからパフォーマンスと拡張性の向上を期待できます。

コードを .NET にアップグレードする以外の追加作業を行うことなく、アプリケーションのパフォーマンスが大幅に向上する場合があります。たとえば、ホワイト ペーパー「Advantages of Migrating from Visual Basic 6.0 to Visual Basic .NET」(MSDN に掲載) には、ASP と Visual Basic 6.0 のアプリケーションを .NET にアップグレードした開発者に関する記述があります。アップグレード後、同時に利用可能なユーザー数とスループットが 3 倍になりました。コードを .NET で実行できるようにしたこと以外に、コードに加えた変更はありませんでした。

このようなパフォーマンスの向上は、常に自動的に実現されるわけではありません。アップグレード プロジェクトを開始するときに、アプリケーションのデザインについて検討する必要があります。たとえば、Visual Basic 6.0 で行われていることが Visual Basic .NET ではどのように行われるかについて考慮する必要があります。パフォーマンスと拡張性を改善するために、.NET でどのように行われるかを見直すことが必要な場合もあります。たとえば、ActiveX Data Objects (ADO) データ アクセス コードを ADO.NET に書き換えることにより、パフォーマンスの大幅な改善が期待できます。以降の章、特に第 20 章「一般的なテクノロジ シナリオ」では、アプリケーションを Visual Basic .NET にアップグレードする際にパフォーマンスと拡張性を改善するためのポイントを多数紹介します。

開発プロセスの短縮

コードの書き換えの決定は、アプリケーションをゼロから作成することを意味します。これに対し、アップグレードの場合は、元のコードの大部分が自動的に変換されることにより、メリットが得られると考えられます。実際、これにより、書き換える場合と比べて生産性が大幅に向上します。

この大きな違いが生じる主な原因として、書き換えでは新しいアルゴリズムの設計、作成、テスト、およびデバッグが必要なのに対して、アップグレードではアルゴリズムを新しい言語に変換し、コードの動作が元のアプリケーションと同じになることを確認するだけで済むことが挙げられます。

単一のフレームワークへの統合

アプリケーションをアップグレードする一般的な理由の 1 つは、複数の言語や複数のフレームワークを単一の言語およびフレームワークにアップグレードして、フレームワークの統合または統一を図ることにあります。

IT 部門が抱える最も大きな問題の 1 つは、異なるフレームワーク上で動作し、異なるプログラミング言語で記述された、複数のアプリケーションをサポートすることです。これにより互換性や統合に関する問題がアプリケーション間で発生するため、管理コストが非常に大きくなります。一方、単一のフレームワーク上でアプリケーションを実行する場合は、アプリケーションの管理、調整、強化、および配置が容易になり、アプリケーションの統合や相互運用を簡単に実現できます。アプリケーションを単一のプラットフォームに移行する場合の唯一の障害は移行に要する時間と作業です。この障害を克服する唯一の方法は移行作業を自動化することです。

このガイドの内容に即して言うと、.NET Framework をベースとしたアプリケーションと Visual Basic 6.0 をベースとしたアプリケーションが組織内に混在する場合は、Visual Basic 6.0 アプリケーションを Visual Basic .NET にアップグレードした方が良いということです。Visual Basic アップグレード ウィザードを使用すると、アプリケーションを書き換える場合の何分の 1 かの時間とコストで、.NET Framework 上で動作する Visual Basic .NET アプリケーションにアップグレードできます。これにより、Visual Basic 6.0 と .NET Framework の両方をサポートする場合と比べて、アプリケーションのサポートとメンテナンスが容易になります。

配置の簡素化

IT におけるもう 1 つの重要な問題は、組織全体へのアプリケーションの配置です。複雑な配置スキームを使用したアプリケーションでは、複数のコンピュータにインストールするコストが大きくなります。DLL のバージョン競合が発生すると、ユーザーの生産性が低下し、IT 部門のトラブルシューティングのコストが上昇します。

.NET Framework には、アプリケーションの配置プロセスを簡素化するためのアプリケーション配置オプションが用意されています。また、バージョンの競合を防ぐためのオプションも用意されています。これにより、新しいアプリケーションを配置するときに、インストール済みのアプリケーションが壊されることがなくなります。

配置オプションの詳細については、第 16 章「アプリケーションの完成」の「アプリケーションのセットアップのアップグレード」を参照してください。

データの収集

この章の冒頭で述べたように、このガイドの目的はアプリケーションを Visual Basic .NET にアップグレードする最良の方法と、アップグレードに要するコストを特定することです。この点に留意し、ここでは、アプリケーションを調べて、アップグレードの決定に役立つ情報を得るための 2 つの方法について説明します。

最初の方法では、使用方法に基づいてアプリケーションを調べます。この場合は、アプリケーションの実行環境、ユーザー、使用方法、およびユーザーが実際に使用する機能について調べる必要があります。こうした観点からアプリケーションを調べる場合、このコンテキストでのユーザーは、エンド ユーザー (人間) であったり、アプリケーションと連携する他のシステムであったりすることに注意してください。以前に作成され、現在は使用されていない機能がアプリケーションに多数含まれていると考えられる場合は、このアプリケーション分析の手法が特に役立ちます。

2 番目の方法では、その内容に基づいてアプリケーションを調べます。アプリケーションの内容には、アプリケーションが提供する機能、これらの機能を提供するためにアプリケーションが使用するテクノロジ、このテクノロジを処理するために記述されたコード、およびこれらの相互作用が含まれます。したがって、アプリケーションのビルドに使用されたソース コード、ソース コードのサイズと複雑さ、およびソース コードが参照するサードパーティ ライブラリに注目します。また、アプリケーションの構造、デザインが表すサブシステム、アプリケーションと外部システムとの依存関係、およびこれらのサブシステムと外部システムとのやり取りに使用されるメカニズムにも注目します。このような分析を行う際には、このガイドに付属する評価ツールが非常に役立ちます。

以降では、アプリケーションの使用方法を分析する手法と、評価ツールを使用してアプリケーションの要素の一覧を作成する方法について説明します。

アプリケーションの使用方法に関する評価

アップグレード計画を作成する際には、アプリケーションの使用方法に関する評価を行う必要があります。ここでは、この評価を行うための手法であるユース ケース分析と入出力分析について検討します。

ユース ケース分析

ソフトウェア アプリケーションの要件を記述するための一般的な手法は、*ユース ケース*によるものです。ユース ケースとは、システムが使用される特定のシナリオを記述したドキュメントのことです。完全なユース ケースは、ソフトウェア アプリケーションのすべての機能要件を表しており、より詳細な機能仕様とテスト計画のための出発点となります。ここでは、一連のユース ケースを使用して、Visual Basic .NET へのアップグレードが必要な機能セットを定義します。アプリケーションのユース ケースが既に存在する場合は、それらのユース ケー

スを出発点として利用できます。通常、ユース ケースはアプリケーション開発の開始前に作成されます。そのため、元のユース ケースが作成された後、新しい使用シナリオが追加された可能性があることに注意してください。

ユース ケースの例

通常、1 つのユース ケースは、特定のタスクを実行するための一連のステップを記述したテキストで構成されます。たとえば、顧客から受け取った小切手を処理するためにユーザーが実行すると予想されるステップを記述できます。

典型的なユース ケースには以下の情報が含まれます。

- **名前。** ユース ケースを識別します。
- **説明。** ユース ケースに関する記述です。
- **前提条件。** ユース ケースのステップを実行する前に満たされなければならない条件を表します。
- **ステップ。** ユース ケースの実行に必要なステップです。
- **事後条件。** ユース ケースのステップが正常終了した後に満たさなければならない条件を表します。
- **アクタ。** これらのステップを通常実行する主体を表します。
- **バリエーション。** 代替のステップを表します。最終的に、事後条件は各ステップが正常に実行されたことを確認するための方法を表します。

ユース ケースの実際のサイズは、必要な詳細と記述されるタスクの複雑さに応じて、数行のものから数ページにわたるものまでさまざまです。ユース ケースに最低限必要なものは、タスクを識別するための簡潔な名前と、ユース ケースの実行に必要なステップです。名前は簡潔で説明的なものにします。例として、"ドキュメントの印刷"、"口座振替"、"生産レベルの確認"、"レポートの Excel へのエクスポート" などが挙げられます。ステップには、アプリケーションに提供する必要のある情報と、アプリケーションから返される情報を記述します。ただし、必要な場合を除いて、詳細情報は記述しないでください。

効果的なユース ケース作成のためのドキュメントは数多くあり、これらはプロジェクトのアップグレードにも役立ちます。ユース ケースを定義する際に注意すべき点を簡単にまとめます。

- ユース ケース名には説明的で正確な名前を使用します。あいまいな用語やソフトウェア関連の用語は使用しないようにします。ソフトウェアやデータベース関連の用語 (クラス、オブジェクト、SELECT、クエリ) ではなく、アプリケーションが属する業務範囲に即した用語 (口座、顧客、注文書、レポートなど) を使用します。
- 各ユース ケースには最小のタスクを記述します。複数の目的を含むユース ケースや、部分処理を記述したユース ケースは作成しないようにします。

- ユース ケースの範囲を適切に設定します。ユース ケースの範囲を明確にします。各ユース ケースのトリガと期待される結果を追加することにより、ユース ケースの範囲を明確にできます。トリガとは、一般的に、ユース ケースの起動を必要とする、またはユース ケースの起動の原因となるイベントのことです。期待される結果とは、ユース ケースの直接的な結果のことです。
- 元のアプリケーションの要件に対応したユース ケースを使用している場合は、そのユース ケースが現在もなお有効で役に立つことを確認します。古いユース ケースの中には、使用されなくなったタスク、または元の概念とは異なるタスクを記述したものがあります。
- ユース ケースの作成にユーザーを参加させます。最終的には、ユーザーの要件がユース ケースに反映されます。したがって、ユース ケースの作成および検証にユーザーを参加させる必要があります。

新しいユース ケースの作成

元のアプリケーションのデザイン時に作成されたユース ケースが存在することが望ましいですが、そのようなユース ケースが利用できない場合は、以下の方法でユース ケースを作成します。

- アプリケーションを使用中のユーザーにインタビューする。
- フォーカス グループを召集したり、ステークホルダーとのブレーンストーミング セッションを行う。
- アンケートや調査を通じて情報を収集する。
- ユーザーの日常業務を調査する。
- 外部システム、および外部システムのアプリケーションとのやり取りを分析する。

ユース ケースからの情報の取得

代表的なユース ケースを作成したら、コード、コンポーネント、およびプロジェクトを必要な機能を実装したものと必要でないものに分類します。これにより不要なアプリケーション部分のアップグレードを回避でき、コストと労力の削減につながります。

ユース ケースに対応した別のアプリケーションは、一連のシステム テストの基礎として使用することもできます。このシステム テストによって、必要な機能の有無とその品質を確認できます。

入出力分析

コンピュータ ソフトウェアを記述する古典的な方法は、ソフトウェアが受け取るすべてのデータ セットと、ソフトウェアが生成するすべてのデータ セットを一覧にすることです。これは単純ですが、アプリケーションの動作や実用性に関する正式な仕様を作成する場合に有効な手法です。

このガイドでは、アプリケーションの各部分の実装については記述されません。プロトコル、形式、ユーザー インターフェイスなど、アプリケーションと対話するための機能が入出力との関連でのみ記述されます。たとえば、顧客情報をアプリケーションに追加するためにユーザーがやり取りを行うユーザー インターフェイス画面の場合、画面のレイアウトではなくその内容が記述されます。

入力の例

入力とは、従業員、顧客、その他のユーザー、または外部アプリケーションによってアプリケーションに入力される情報のことです。入力の例として以下が挙げられます。

- 顧客情報など、アプリケーションのデータ入力画面にユーザーが手動で入力した生データ。
- 電子メール メッセージ、レポート、リスト、生データなど、アプリケーションの外部で作成され、アプリケーションに送られる任意のドキュメント。
- 他のアプリケーションから受け取った非同期メッセージ。
- インターネットまたはイントラネット経由でのユーザー要求。
- タイマ、アマウント、電子メール メッセージなど、アプリケーションが監視するソフトウェア イベント。
- アプリケーションが監視するハードウェア イベント。

出力の例

出力とは、アプリケーションが生成、画面に表示、印刷、他のアプリケーションに送信、またはハード ディスク、フロッピー ディスク、CD、テープ、ネットワークなどの記憶装置に保存するすべてのものを指します。出力の例として以下が挙げられます。

- 既存のデータに基づく未処理の情報や概要情報を含むレポート。通常、何らかのクエリに対する応答や、スケジュールされたイベントの結果として生成されます。
- 電子メール メッセージやファックスなど、電子的に送信されるドキュメント。
- グラフ、テキスト、棒グラフ、画像、音声などのデータ表現。
- イベントや警告など、透過的にログに記録されるか、または即時対応を要するメッセージ。

入出力からの情報の取得

入出力の一覧を利用すると、ユース ケースを利用する場合と比べ、入出力を処理するコードをより簡単に識別できます。また、使用されなくなったためアップグレードの必要がないコンポーネント、ファイル、またはプロジェクトを特定することにより、アップグレードに要する時間を節約できると考えられる場合、このような入出力の一覧が役立ちます。

入出力はテストを行う場合にも役立ちます。なぜなら、入力と出力のマッピングを作成でき、このマッピングを基にテストを作成して、アップグレード後のソフトウェアの機能の妥当性を確認できるからです。

アプリケーション環境

アプリケーションの開発環境および配置環境は、問題の解決時に行う調査の優先順位と方法を定義する際に重要になります。

たとえば、元の開発チームに参加していない場合は、元の開発、アーキテクチャ、デザイン、実装、および (適切な場合) 製品管理の担当者を把握することが重要になります。アップグレード作業を行うプログラマの経

験について明確に把握することも重要です。なぜなら、プログラマのトレーニングの手配や、プロジェクト管理のために Visual Basic .NET の経験が豊富な人材の採用が必要になる場合があるからです。

配置先では、アプリケーションのサポートおよび管理を行う担当者についても把握する必要があります。アプリケーション テクノロジや配置特性が変更される場合があるため、ここでもトレーニングが必要になります。

どちらの環境でも、必要な指示を得られるように、判断を下す責任者について把握する必要があります。

アプリケーション分析

アップグレード プロジェクトの実行に必要な作業を特定および把握することは、さまざまなツールを必要とする複雑な作業です。このプロセスでは、アプリケーション全体がより管理しやすい複数の要素に分割され、各要素について個別にカウント、解釈、および見積もりが行われます。したがって、アップグレードに必要な作業の見積もりは、各段階で自動化ツールによるサポートが可能な分析プロセスとなります。

アップグレード プロジェクトでは、元のアプリケーションの分析が必要不可欠です。元のアプリケーションを分析することにより、アップグレード対象のアプリケーションについてより深く理解でき、アップグレードの計画および決定のために必要な情報を取得できます。評価ツールは、Recommended Upgrade Order レポートなどアップグレードの実行に役立つ情報を含め、必要なすべての情報を生成します。

Visual Basic 6.0 Upgrade Assessment Tool を使用すると、アプリケーションのアップグレードに必要な作業量を測定できます。このツールは、アプリケーション コンポーネントやコンポーネント間の関係をアップグレードの観点から分析し、アップグレード プロジェクトで多くのリソースを必要とする要素、構成、および機能を評価します。

評価ツールの主な目的は、Visual Basic 6.0 プロジェクトを分析して、Visual Basic .NET へのアップグレードの計画、およびアップグレードに要するコストの見積もりに役立つ情報を取得することです。このツールによって生成される複数のレポートを使用すると、作業の労力やコストをより詳しく計算できます。評価ツールでは、ユーザーが指定した構成値で初期見積もりを変更できるため、組織の特定の環境に合わせてツールを調整できます。

評価ツールは、Visual Basic 6.0 パーサーを使用して元のアプリケーションを分析します。このパーサーでは、ユーザーが指定したすべてのプロジェクトが解析されます。続いて取得されたデータが処理され、アップグレードが困難な使用パターンと機能が特定されます。これらの機能の分類とカウントがすべて終了すると、詳細データや概要データを含む一連のレポートが生成されます。

評価ツールでは、アップグレード ウィザードで Visual Basic 6.0 アプリケーションをアップグレードする際に発生し得る既知の問題の大部分が考慮されます。ただし、完全自動アップグレードが完了した後でのみ検出される問題もあります。評価ツールを使用する際にはこの点に注意する必要があります。これらの問題を検出するには、アップグレード ウィザードでのみ使用できる複雑なパターン認識ルールが必要になります。これらの問題は、その他の問題に対する割合で見積もることができます。評価ツールは、こうした割合での見積もりを Upgrade Issues レポートに出力します。

以降では、評価ツールによって生成される各レポートと、作業を見積もる際に考慮する側面について説明します。

評価ツールの使用

このガイドに付属する評価ツールでは、2 つの Microsoft Excel® 2003 ファイルが生成されます。最初のファイルは MainReport.xls という名前のファイルで、ツールによる詳細な評価結果が含まれます。このファイルを初めて開くと、図 3.1 に示すページが表示されます。

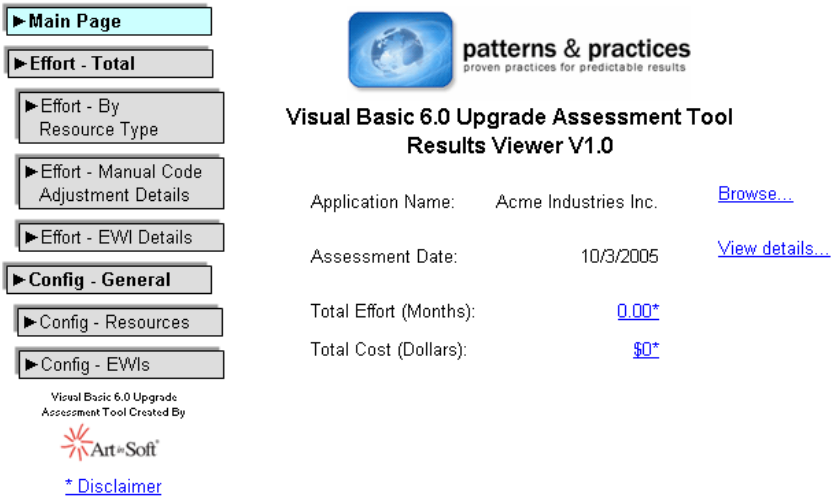

図 3.1
MainReport.xls の Main Page

図 3.2 に示すページからレポートのその他のページに順番にアクセスしていくことができます。

このレポートには、DetailedReport.xls というファイルにある 2 番目のレポートへのリンクが含まれていることがわかります。評価ツールの使用中にコストおよび労力の構成に対して加えた変更を保持できるように、メイン レポートのリンクを変更して、他の評価による詳細レポートにリンクさせることができます。メイン レポート内にあるアプリケーション固有のデータはすべて、特定の詳細レポートにリンクされています。以降では、これらの 2 つのレポートに含まれるさまざまなシートについて説明します。

通常、MainReport.xls ファイルには、すべての構成設定とコストの見積もり、および労力の見積もりに関する概要が含まれます。DetailedReport.xls ファイルには、アプリケーションの内容に関する詳細レポートが含まれます。図 3.2 に、DetailedReport.xls ファイルの最初のページを示します。

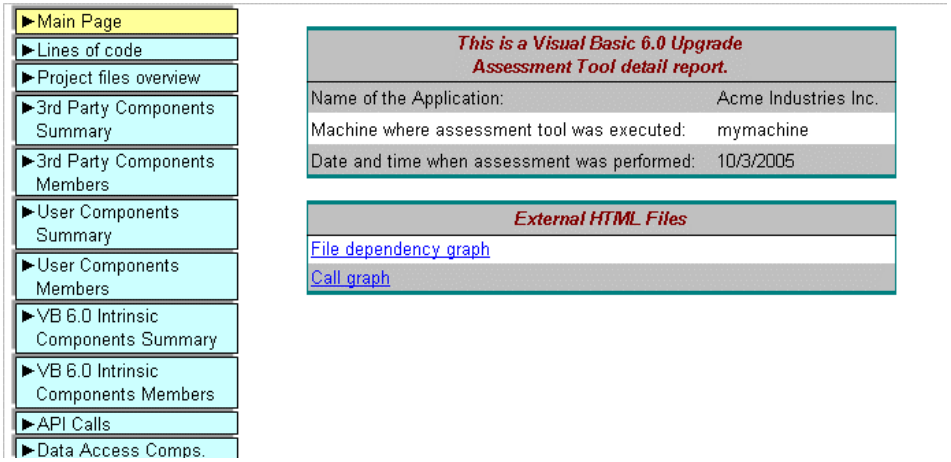

図 3.2
DetailedReport.xls のMain Page

現在のアーキテクチャとターゲット アーキテクチャ

評価ツールは、元のアプリケーションのアーキテクチャとプロジェクトの種類を特定および分析し、アプリケーションが参照するファイルとコンポーネントをすべて抽出します。プロジェクト グループにアクセスする場合は、プロジェクト間の依存関係も分析します。

評価ツールでは、以下の Visual Basic 6.0 プロジェクトの種類を分析できます。

- **標準** EXE。見積もりを実行する際に、平均的な生産性の値に基づいて、標準的な機能が分析されます。
- IIS (**インターネット インフォメーション サービス**) **アプリケーション。**アプリケーションは標準的な方法で分析されます。
- **分散コンポーネントを使用するアプリケーション。**アップグレードに要するコストを見積もる際に、分散機能の使用が特定および累算されます。

Visual Basic 6.0 プロジェクト グループは、リソースと機能を共有できる関連プロジェクトのコレクションです。評価ツールでは、上記のプロジェクトの種類で構成されるプロジェクト グループを処理できます。Recommended Upgrade Order レポートを生成する際に、共有コンポーネントが特定および分析されます。

プロジェクトでのアップグレード パスは、基本コンポーネント間の依存関係によって異なります。評価ツールはこのような依存関係を分析し、推奨されるアップグレード順序を生成します。アップグレード パスに影響するも

う 1 つの要因は、ユーザーが選択するアップグレード方法です。第 2 章「アップグレードの成功のためのプラクティス」で説明したように、水平アップグレードまたは垂直アップグレードを選択できます。

アプリケーション コンポーネントの関係の大部分は、第 4 章「一般的なアプリケーションの種類」で説明したアプリケーションの種類の対応関係に従って、対象のアプリケーションとアプリケーションの全体的なアーキテクチャにそのまま残ります。作業の見積もりに関するこの分析では、プロジェクトの種類またはアーキテクチャの変更のための作業は考慮されません。これらの作業は、同等の機能が実現された後に実行するアップグレード後の作業と考えられるからです。

アップグレードするインベントリ

アプリケーション分析において最初に考慮する点は、アップグレード プロセスに関連するリソースとコンポーネントのインベントリです。このインベントリには、モジュール、サードパーティ コンポーネント、および組み込みコンポーネントが含まれます。これらの一部はユーザー定義コンポーネントです。ユーザー定義コンポーネントについては、コンポーネントを使用するソース コードに加えて、コンポーネントの実装もアップグレードする必要があります。サードパーティ コンポーネントや組み込みコンポーネントについては、それぞれのコンポーネントにアクセスするコードをアップグレードする必要があります。この場合、評価目的の観点から言うと、アプリケーションで使用するクラス メンバやその他の要素のインベントリ作成に関連しています。

図 3.3 のテーブルは、DetailedReport.xls ファイルの一部であり、評価ツールによって生成される Project Files Overview レポートの例を示しています。

Project	Total Files	Classes	Modules	UserControls	PropertyPages	Designers	Forms
LinkerMAC.vbp	139	136	3	0	0	0	0
LinkerUserInt.vbp	111	0	11	0	0	0	100
ConvertUI.vbp	21	1	12	0	0	0	8
LConvertUI.vbp	3	0	0	0	0	0	3
MyLinker.vbp	4	0	0	0	0	1	3
MyNewLinker.vbp	3	0	0	0	0	1	2
Total	281	137	26	0	0	2	116

図 3.3

Project Files Overview *レポート*

このレポートは、評価ツールによって処理されたプロジェクトと、各プロジェクトに含まれるさまざまなモジュールの数を表しています。このレポートを使用すると、さまざまな機能を持つプロジェクトを特定できます。たとえば、ユーザー インターフェイス要素を提供する (フォーム モジュール中心の) プロジェクトや、ビジネス ロジック機能を実装した (クラスとモジュール中心の) プロジェクトなどです。

ユーザー コンポーネントは、アップグレード作業の大部分を占めます。したがって、アップグレードするリソースのインベントリを決定する際には、ユーザー コンポーネント定義と宣言済みインスタンスのインベントリについて考慮する必要があります。図 3.4 に、DetailedReport.xls ファイルの User Components Summary レポートの例を示します。

Component	LOC	Count
MaxBusiness.Font		152
MaxBusiness.cHelpManager	367	101
MaxBusiness.asiento		101
MaxBusiness.cMouseWait	17	64
MaxBusiness.DocumentoTipo		43
MaxBusiness.cAsistente	15	41
MaxBusiness.MovimientoStock		15
MaxBusiness.cColGrilla	13	11
MaxBusiness.DiccTablaPlantilla		10
MaxBusiness.EjercicioContable		10
MaxBusiness.Despacho		10
MaxBusiness.Tercero		9
MaxBusiness.ArbolNodo		9
MaxBusiness.fManfiestoCargaCalibres	264	9

図 3.4

User Components Summary *レポート*

User Components Summary レポートでは、すべてのユーザー定義コンポーネントと、各コンポーネントに含まれるコードの行数 (LOC) が表示されます。このレポートでは、その他のユーザー コードで宣言された各クラスのインスタンス数も表示されます。評価ツールによって生成される見積もりワークシートには、各コンポーネントのメンバの使用方法を示す、ユーザー コンポーネントの詳細レポートも含まれます。

アップグレードするアプリケーション インベントリでは、サードパーティ コンポーネントや Visual Basic 6.0 の組み込みコンポーネントも評価されます。これらのコンポーネントをアップグレードする場合は、アプリケーションのソース コードを処理して、各コンポーネント メンバへのアクセスを特定する必要があります。使用されているすべてのメンバが分析され、対応するメンバが移行先の環境に存在するかどうかを判断します。メンバによっては、Visual Basic .NET に対応するメンバがないものもあります。その場合は手動で .NET にアップグレードする必要があります。図 3.5 に、DetailedReport.xls ファイルの Third Party Component Summary レポートを示します。このレポートには、アプリケーションで使用されるすべてのコンポーネントと、各クラスごとに作成されるインスタンス数が表示されます。

Component	Count
EditLib.fpText	776
vsOcx6LibCtl.vsElastic	576
Threed.SSCommand	270
Threed.SSCheck	229
EditLib.fpDoubleSingle	206
EditLib.fpDateTime	193
vsOcx6LibCtl.vsIndexTab	132
Threed.SSOption	123
ACTIVESKINLibCtl.SkinForm	119
TrueDBGrid60.TDBGrid	111
GridEX20.GridEX	107
TrueDBGrid60.Column	85

図 3.5

Third Party Components Summary *レポート*

Visual Basic 6.0 の組み込みポーネントは、サードパーティ コンポーネントと同様の方法で処理されます。ただし、ほとんどの組み込みコンポーネントは .NET Framework に対応するコンポートを持っており、メンバ アクセスの大部分についてある種の変換を行う必要があります。アプリケーションで使用される組み込みコン

ポーネントのインベントリは、評価ツールによって生成される見積もりワークシートの Visual Basic Intrinsic Components Summary レポートに表示されます。

ユーザー COM オブジェクトもアップグレードするインベントリに含まれます。労力とコストの見積もりでは、このオブジェクトの使用回数が評価されます。宣言元のプロジェクト以外のプロジェクトからユーザー COM オブジェクトにアクセスする場合、このオブジェクトはサードパーティコンポーネントと見なされる点に注意する必要があります。

ユーザー定義コンポーネントの場合は、メンバ宣言やアップグレードのコストが大きい機能の使用など、コンポーネントの内部コードの処理にアップグレード作業の大部分が費やされます。評価ツールはこれらの機能を検出し、生成する見積もりワークシートに表示します。たとえば、以下のような機能が検出されます。

- API 呼び出し。受け取るパラメータのデータ型に加えて、関数名とライブラリが表示されます。レポート内の列には、他のコードでの関数の使用回数が表示されます。
- データ アクセス。ADO、RDO、DAO など、使用されているさまざまなデータ アクセス テクノロジが表示されます。
- COM+。COM+ および Microsoft Transaction Server (MTS) クラスの使用に関する概要と詳細情報が表示されます。

これらのレポートに表示される情報を利用すると、アップグレードの計画時に特別な注意を払う必要があるアプリケーションのコンポーネントを特定できます。たとえば、ADO レコードセットが多数使用されている場合は、影響を受けるコンポーネントをアップグレードする際に、垂直アップグレードを使用します。アップグレード計画の詳細については、第 2 章「アップグレードの成功のためのプラクティス」と第 4 章「一般的なアプリケーションの種類」を参照してください。

ソース コードのメトリクス

通常、アプリケーションのサイズはコードの行数という観点で定義されます。また、アプリケーションのソースコードは、コードの種類に応じて分類およびカウントされます。たとえば、評価ツールを使用すると、ビジュアル コンポーネント宣言に関連するコード行、コメント、空行、およびユーザーが記述したコードを特定できます。アップグレードという観点では、コードの行数はアプリケーションの複雑さを見積もるものとして不十分な点があります。アップグレード ウィザードがサポートする機能のみを使用しているアプリケーションの場合、サイズが大きくてもアップグレード作業は少なくて済みます。一方、アップグレード ウィザードがサポートしない機能を多数使用しているアプリケーションの場合は、サイズが小さくでもアップグレードのコストは大きくなります。それでも、ソース コードのメトリクスを使用して、アップグレード プロセスに含まれるタスクを実行するために必要なサイズと作業を見積もることはできます。これらのタスクはそれほど複雑なものではなく、主にアプリケーションのサイズと、アプリケーションを構成するモジュールおよびコンポーネントの数に依存します。タスクの例とし

ては、アプリケーション リソースのインベントリや移行順序の定義があります。これらの 2 つのタスクは、評価ツールによって生成される Effort – Total レポートで見積もられます。

評価ツールはプロジェクトに含まれるコードの行数を示すレポートを生成します。このレポートを使用すると、ソース コードのメトリクスを把握できます。図 3.6 に、評価ツールによって生成される DetailedReport.xls ファイルのコード行 (LOC) レポートの例を示します。

Project	Total Lines	Visual Lines	Code Lines	Comment Lines	Blank Lines
LinkerDAC.vbp	270,334	0	179,190	53,192	37,952
LinkerUserInt.vbp	214,773	41,065	98,639	51,485	23,584
ConvertUI.vbp	26,337	758	17,362	3,751	4,466
LConvertUI.vbp	1,705	1,458	111	92	44
MyLinker.vbp	4,472	3,543	839	1	88
MyNewLinker.vbp	3,382	2,598	724	0	59
Total	521,003	49,422	296,865	108,521	66,193

図 3.6

コード行(LOC) レポート

評価ツールは、アプリケーション モジュールの内容をすべて分析し、その結果を基にソース コード行を分類します。各行が解析および分類され、処理対象の Visual Basic 6.0 プロジェクトに記述されているビジュアル行、コード行、およびコメント行の数が特定されます。ビジュアル行とは、Visual Studio によって生成された行のことです。

サポートされない機能の処理

アップグレード ウィザードは、言語構造と組み込みコンポーネントの大部分を自動的にアップグレードします。サードパーティ コンポーネントや Visual Basic 6.0 の各種プロジェクトもサポートします。ただし、アップグレード エンジンによって完全にサポートされず、Visual Basic .NET にアップグレードするために、ある程度の手作業が必要になる機能もあります。ArtinSoft が提供する Visual Basic Upgrade Wizard Companion は、アップグレード ウィザードでサポートされない機能の多くをサポートします。

サポートされない機能を含むアプリケーションをアップグレードすると、対象のアプリケーションでさまざまなエラーが発生します。コンパイル エラーやランタイム エラーが発生し、ユーザーによる修正が必要になる可能性があります。その結果、コードを見直し、修正を行い、適切なテストを実行するためのリソースが必要になります。

評価ツールは、コードを分析し、アプリケーションのアップグレード後に問題を発生させるコード パターンを検索することにより、サポートされない機能を検出します。見積もりワークシートの [Config – EWI] タブでは、各問題に関連付けられたコストと労力の値を確認し、特定のニーズに合わせて調整できます。このタブに挿入される初期値は、ArtinSoft がこれまでに行った Visual Basic アップグレード プロジェクトを基に決定されます。図 3.7 に示す MainReport.xls レポート ファイルの [Effort – EWIs] タブを生成する際に、これらの構成値が評価されます。

NAME	MSDN ID	Resource	Occurrences	Hours per Occurrence	Cost per Hour	Total Hours	Total Cost
ParamArray was changed from ByRef to ByVal	1003	DEV	50	0.10	50	5.00	250
Statement was removed. Variables were explicitly declared	1005	DEV	21	0.02	50	0.35	18
Statement is not supported	1014	DEV	34	0.13	50	4.53	227
Declaring a parameter 'As Any' is not supported.	1016	DEV	62	0.05	50	3.10	155
Statement was changed	1021	DEV	78	0.07	50	5.20	260
DoEvents does not return a value.	1022	DEV	14	0.07	50	0.93	47
As statement was removed from ReDim statement	1056	DEV	18	0.03	50	0.60	30

図 3.7
[Upgrade Issues] タブ

評価ツールによって生成される MainReport.xls ファイルの [Effort – EWIs] タブには、検出されたすべての問題、問題の発生回数、問題の複雑さ、および見積もりコストの値が表示されます。アップグレード ウィザードが生成するすべての問題が評価ツールによって検出されるわけではないという点に注意します。ただし、アップグレード作業に大きな影響を与える問題の大部分は検出されます。既に述べたように、[Config – EWIs] タブに含まれる列を使用すると、アップグレードの問題の解決に要する労力の値を調整して、より正確な見積もりにすることができます。

アプリケーションの依存関係

評価ツールは、以下に示すソース コードのさまざまな要素を分析することにより、アプリケーション コンポーネント間の依存関係を特定します。

- **アプリケーションの参照。** ユーザー コンポーネントへの明示的な参照は、ユーザー プロジェクト間に依存関係が存在することを示します。
- **変数宣言。** 異なるモジュールまたはプロジェクトで定義された型を使用して変数を宣言すると、モジュールまたはプロジェクト間に依存関係が発生します。
- **メンバ アクセス。** メンバ アクセスによっても、サブルーチン、関数、またはプロパティ間の依存関係が発生します。

評価ツールは、プロジェクト グループのメンバ間の依存関係を特定します。あるプロジェクトが同じプロジェクト グループ内の別のプロジェクトを参照する場合、評価ツールは参照先プロジェクトをユーザー コンポーネントとして特定し、これらのプロジェクト間の依存関係をユーザーに通知します。

アップグレード順序の定義、テスト、デバッグなど、アップグレード計画のさまざまな側面で、コンポーネント間の依存関係に関する知識が不可欠となります。アップグレード計画の詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

評価ツールは、依存関係の分析に基づいて DetailedReport.xls ファイルに Upgrade Order レポートを生成します。このレポートには、推奨されるファイルのアップグレード順序が表示されます。図 3.8 に、このレポートの例を示します。

Project	File
AdminUI	G:\Depot\Microsoft\VB6Migration\Test\Samples\RichClientSample\AdminUI\frmProductAddition.frm
	G:\Depot\Microsoft\VB6Migration\Test\Samples\RichClientSample\AdminUI\mdiAdmin.frm
	G:\Depot\Microsoft\VB6Migration\Test\Samples\RichClientSample\AdminUI\frmProductUpdate.frm
	G:\Depot\Microsoft\VB6Migration\Test\Samples\RichClientSample\AdminUI\frmCustomerAddition.frm
	G:\Depot\Microsoft\VB6Migration\Test\Samples\RichClientSample\AdminUI\Resources.bas
	G:\Depot\Microsoft\VB6Migration\Test\Samples\RichClientSample\AdminUI\frmSupplierAddition.frm
	G:\Depot\Microsoft\VB6Migration\Test\Samples\RichClientSample\AdminUI\frmSupplierUpdate.frm

図 3.8

*サンプルの*Upgrade Order *レポート*

このレポートは、アプリケーションの各プロジェクトに対応する複数のセクションで構成されています。各グループのファイルは、グループ内のファイル間の依存関係に従って並べられます。各グループの先頭のファイルは、同じグループのユーザー定義ファイルには依存しません。したがって、このファイルは単独でアップグレードおよびテストできます。2 番目のファイルは、最初にアップグレードしたファイルに基づいてアップグレードおよびテストできます。

これらの各ファイルは、機能を追加しながら段階的にアップグレードできます。評価ツールは、Upgrade Order レポートを作成するため、依存するユーザー コンポーネントの数が最も少ないモジュールを最初に検索し、次にこのレベルに含まれるコンポーネントを基に新しい依存レベルを生成します。このように、前のレベルが提供する機能に基づいて各レベルが構築されます。このレポートは利用可能なアップグレード パスを特定するのに役立ちます。水平アップグレードを使用する場合は、推奨されたアップグレード順序に従うことができます。アプリケーションの一部に対して垂直アップグレードを実行する場合は、さまざまなグループのモジュールを同時にアップグレードすることもできます。

不足しているアプリケーション要素

最初のアップグレード準備作業が終了した後、自動アップグレードを実行するコンピュータ上で一部のアプリケーション コンポーネントがまだ使用できない場合があります。これらのコンポーネントを特定したら、アップグレード プロセスを再開する前に環境の変更を行う必要があります。このような追加ステップは時間とリソースを消費します。

評価ツールは不足しているアプリケーション要素を特定します。特定された要素は、見積もりワークシートの Missing Components レポートと Missing Files レポートで確認できます。不足している要素は、分析対象の

プロジェクトに含まれるユーザー定義モジュールと明示的な参照を基に特定されます。

図 3.9 に、DetailedReport.xls ファイルの Missing Components レポートを示します。

Path	Version	Name
c:\windows\system32\stdole.tlb	2.0.0	OLE Automation
c:\windows\system32\comsvcs.dll	1.0.0	COM+ Services Type Library
c:\windows\system32\dx8vb.dll	1.0.0	DirectX 8 for Visual Basic Type Library
mscomctl.ocx	2.0.0	
mschrt20.ocx	2.0.0	
mscomm32.ocx	1.1.0	
mscal.ocx	7.0.0	
tabctl32.ocx	1.1.0	

図 3.9

Missing Components レポート

Missing Components レポートでは、不足しているコンポーネントのファイル名とそのパス、対応するバージョン、および登録名が表示されます。自動アップグレードを実行するコンピュータにこれらのコンポーネントをインストールする必要があります。不足しているコンポーネントの数は、不足している要素の構成データを基に作業の見積もり (評価ツールによって生成される MainReport.xls ファイルの [Effort – By Tasks] タブ) で評価されます。

図 3.10 に、DetailedReport.xls ファイルの Missing Files レポートを示します。

Name	Path	File Type	Project Name
File 1	C:\MyApp\Forms\File1.frm	Form	Project 1
File 2	C:\MyApp\Modules\File2.frm	Module	Project 1
File 3	C:\MyApp\Forms\File3.frm	Form	Project 1
File 4	C:\MyApp\Controls\File4.frm	User Control	Project 1
File 5	C:\MyApp\Forms\File5.frm	Class	Project 1
File 6	C:\MyApp\Modules\File6.frm	Module	Project 2
File 7	C:\MyApp\Forms\File7.frm	Form	Project 2
File 8	C:\MyApp\Forms\File8.frm	Form	Project 2
File 9	C:\MyApp\Forms\File9.frm	Form	Project 2
File 10	C:\MyApp\Controls\File10.frm	User Control	Project 2

図 3.10

Missing Files レポート

このレポートでは、不足しているファイルの名前、パス、およびモジュールの種類が表示されます。不足しているファイルが検出されたプロジェクトも表示されます。不足しているコンポーネントと同様に、不足しているユーザー ファイルも作業の見積もりで評価されます。

労力とコストの見積もり

アプリケーションの最適なアップグレード方法を決定したら、機能的に同等な .NET アプリケーションにアップグレードするために必要な実際の労力を決定する必要があります。これは、アプリケーションの分析および評価における 2 番目の目的です。労力を見積もる場合は、要件の初期分析からアプリケーションの配置までの、アップグレードのすべてのステップを考慮する必要があります。さらに、開発者とユーザーの再トレーニングが必要になる場合もあります。

労力の見積もりを終えると、アップグレードのコストの見積もりに関する最も重要な情報が得られます。ただし、アップグレードのコストはその他の要素にも依存します。そのような要素の 1 つは、アップグレード作業者のコストです。アップグレードにはさまざまな要素が含まれるため、さまざまな経歴を持つ人が必要になります。たとえば、プロジェクト管理者、開発者、テスタ、アーキテクトなどが含まれ、それぞれコストが異なります。

アプリケーションを新しいバージョンに切り替える際には、トレーニング、ハードウェア、ソフトウェア ライセンスなどビジネスの阻害要因になるもの、またはこうした阻害要因を回避するためのコストについても考慮する必要があります。

評価ツールによって生成される作業見積もりレポート ([Effort – Total]、[Effort – By Task]、[Effort – By Resource Type]、および [Effort – EWIs] タブ) では、構成可能なパラメータが提供されます。これらのパラメータを使用すると、プロジェクトのアップグレードに必要な労力と時間の計算を調整できます。作成されるレポートには、アプリケーションのテクノロジに関する詳細が含まれます。

手法の概要

労力という概念をアップグレードのコンテキストで理解する必要があります。各作業の労力は時間単位で示され、標準化された時間を表します。つまり、各作業の労力は、平均的なリソースを使用して実行した場合にその作業に要する時間に相当します。経験豊富なプログラマがその作業を担当した場合は、短時間で終了します。Effort Estimation レポートでは、既定で提供される平均的なリソースに加えて、初級開発者、初級テスタ、上級開発者、上級テスタなど、それぞれ時間単位のコストが異なるパラメータを使用できます。

大部分の作業のコストは、作業を行うのに必要なリソースのコストと、作業を行うのに必要な労力の見積もりから直接算出されます。たとえば、上級開発者が変換作業を担当した場合、COM コンポーネントのアップグレードに要するコストは高くなりますが、結果的に少ない時間で済む可能性があります。一方、初級開発者が担当した場合、上級開発者と比べてコストは安くなりますが、機能的に同等なアップグレードを実現するためには多くの時間が必要になります。より特殊な作業の場合は、各作業の難易度や必要なリソースの種類に関連する作業あたりのコストからコストが算出されます。評価ツールによって生成される MainReport.xls ファイルの [Config – General] タブには、さまざまなアップグレード作業に要するコストと労力の既定値が表示されます。これらの値は、組織の開発能力やコストに可能な限り一致するように調整できます。図 3.11 に、[Config – General] タブを示します。

Migration Task	Effort per occurrence (Hours)	Cost per hour	Resource	Description
1000 lines of code	0.25	50	DEV	Migrate 1000 lines of code assuming no issues reported by the upgrade wizard. This affects the cost only in that large amounts of code increases risks.
Property Page	2.00	50	DEV	The number of hours and the cost per hour to migrate one property page assuming no issues reported by the upgrade wizard.
Designer	3.00	50	DEV	The number of hours and the cost per hour to migrate one designer
Form	3.00	50	DEV	The number of hours and the cost per hour to migrate one Form
3rd-Party Component Declarations	0.50	50	DEV	The number of hours and cost per hour to migrate each 3rd Party Component used in a variable declaration
3rd-Party Component Member Usage	0.10	50	DEV	The number of hours and cost per hour to migrate each use of a member of a 3rd Party Component
User Component Definitions	2.00	50	DEV	The number of hours and the cost per hour to migrate each User Component

図 3.11
[Config – General] タブ

作業によっては、個々のコンポーネントまたはファイルを構成するコードの行数に直接依存するものがあります。このような作業での労力の見積もりは、コードの行数に、ある定数を乗じて算出します。この定数は、これまでに行われたさまざまなアプリケーションに対するアップグレード プロジェクトで実行された作業に基づいています。

ある機能の使用回数からコストが算出される作業の場合は、プロジェクト内でその機能が使用されている回数をカウントし、1 回あたりのアップグレードのコストと労力 (時間単位) を加算することにより、アプリケーションの変換に要する労力の見積もりが計算されます。評価ツールによって生成される MainReport.xls ファイルの [Effort – By Task] タブには、各機能の 1 回あたりのコストと労力、およびその機能のすべての使用回数を計算した合計時間と合計コストが表示されます。図 3.12 に、このタブの例を示します。

Feature	Occurrances	Effort per occurrence (Hours)	Cost per occurrence ($)	Total hours	Total cost	Resource
Non-visual, executable lines of code	294,758	0	0.0125	74	$ 3,684.48	DEV
Property Pages	0	2	100.00	0	$ -	DEV
Designers	2	3	150.00	6	$ 300.00	DEV
Forms	116	3	150.00	348	$ 17,400.00	DEV
Third Party Component Declarations	55	1	25.00	28	$ 1,375.00	DEV
Third Party Component Member Usage	2,173	0	5.00	217	$ 10,865.00	DEV
User Component Declarations	36	2	100.00	72	$ 3,600.00	DEV
User Components Member Usage	2,688	0	5.00	269	$ 13,440.00	DEV
VB Intrinsic Components	18	0	12.50	5	$ 225.00	DEV
VB Intrinsic Components Member	16,556	0	5.00	1,656	$ 82,780.00	DEV
API Calls	27	0	24.00	8	$ 648.00	ARC
ADO Data Access Controls	6	0	5.00	1	$ 30.00	DEV
ADO Data Access Member Usage	250	0	5.00	25	$ 1,250.00	DEV

図 3.12
[Manual Code Adjustment Effort] タブ

各 EWI には労力とコストが関連付けられます。これらの値は、それぞれの問題を他のアップグレードプロジェクトで解決するために必要なステップから算出されます。これらの値は、MainReport.xls ファイルの [Effort – EWIs] タブで構成可能です。図 3.13 に、このテーブルの例を示します。

NAME	MSDN ID	Resource	Effort per occurrence (Minutes)	Effort per occurrence (Hours)	Cost per hour	Functionality
ParamArray was changed from ByRef to ByVal	1003	DEV	6	0.10	50	Use of ByRef modifier.
Statement was removed. Variables were explicitly declared	1005	DEV	1	0.02	50	Use of DefType. Each declaration occurrence should be counted. Use of variables declared by these statements won't be counted.
Statement is not supported	1014	DEV	8	0.13	50	Detects use of unsupported flow control statement On ... GoSub.
Declaring a parameter 'As Any' is not supported.	1016	DEV	3	0.05	50	Use of "As Any" modifier in external declarations.
Statement was changed	1021	DEV	4	0.07	50	Detects use of unsupported member vba.Information.IsMissing.

図 3.13

[Effort – EWIs] タブ

このタブでは、各 EWI に関連付けられた労力とコストの既定値を変更できます。これにより、アップグレードの問題に対処するために必要な労力をより正確に見積もることができます。

フェーズの見積もり

さまざまなフェーズの見積もりは、アップグレード手順の各ステップに直接関連します。見積もりを行う項目を簡単にまとめると、以下のようになります。

- **プロジェクトの開始。** アップグレード プロジェクトのキックオフ ミーティングに要する時間の見積もりです。通常、これは大きな値にはなりません。この項目は、これらのコストの見積もりに明確に定義された目的があることを示すものとなります。また、トレーニング、調査、ソフトウェアやハードウェアの購入など、項目には含まれていなくても、アップグレード プロジェクトを開始する前に行う必要のあるさまざまな作業についても示します。
- **アプリケーションの準備。** 元のアプリケーションの開発環境のセットアップおよび検証に要する時間の見積もりです。元のアプリケーションをコンパイルしてアプリケーションに問題がないことを検証する時間や、後の段階の変換プロセスを加速するためにソースコードを修正する時間が含まれます。
- **アプリケーションの変換。** アップグレード ウィザードの実行や手作業による最終的な修正に要する時間の見積もりです。
- **テストとデバッグ。** 元のテスト ケースの実行やランタイム エラーの修正に要する時間の見積もりです。

- **プロジェクト管理。** スケジュール設定や管理に関連する管理タスクに要する時間の見積もりです。
- **構成管理。** アップグレード プロセス全体を通じて得られる製品の構成管理に要する時間の見積もりです。

詳細については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。

アプリケーションの構成はアップグレードするインベントリを表します。このインベントリは [Project Files Overview] タブに表示されます。アップグレード作業を見積もる際には、アプリケーションのインベントリを最初に考慮する必要があります。このインベントリにより、アップグレードするプロジェクトとモジュールの概要を把握できます。Effort Estimation レポートでは、これらのモジュールの内容に関する詳細情報が提供されます。アプリケーションの構成に関するレポートは、アップグレードのその他の部分を計画したり、さまざまなプロジェクトやモジュールの優先順位を設定する際に重要となります。たとえば、クラス間の依存関係を含む複数のユーザー コントロールがある場合は、これらのコンポーネントを優先してアップグレードし、これらのユーザー コントロールに依存するフォームやクラスをアップグレードするまでに準備を整えておく方が賢明です。

「手法の概要」で説明したように、[Config – EWI] タブにはタスクをアップグレードするために必要な作業の見積もりが表示されます。評価ツールによってこのタブの一部として生成される値は、以下の各ワークシートに影響します。

- Effort – EWIs
- Effort – By Task
- Effort – Total

[Effort – By Task] タブでは、機能/コード項目など作業の見積もりに関連するさまざまな要素が一元管理されます。これらの要素には、プロパティ ページ、デザイナ、サードパーティ コンポーネント、ADO、RDO、COM、COM+ などが含まれます。このタブには各要素に関する概要情報が表示されます。より詳細な情報は DetailedReport.xls ファイルに表示されます。

[Effort – By Task] タブに含まれる作業については、より詳細な情報が用意されており、この詳細のデータを基に見積もりが計算されます。このレポートでは、さまざまな作業を行うために使用するリソースの種類も明示的に指定されます。これにより、各作業で必要とされる経験レベルと、それに伴うコストを把握できます。

Effort – Total ワークシートについて

評価ツールによって生成される Microsoft Excel ワークブックの Effort – Total ワークシートでは、労力とコストの見積もりに関する概要が表示されます。

ワークシートを詳細に見ていく前に注意すべき点があります。それは、実際のプロジェクトの見積もりで使用する前に、このワークシートの値を十分に検討してカスタマイズする必要があるということです。このワークブックでは、アプリケーションのほとんどの要素がカバーされており、アップグレードに必要な労力を示す重みが

各要素に関連付けられています。ただし、一部の要素については重みがゼロになっています。これは、評価ツールでは、見積もりに対してすべての要素が重要であるとは見なされないことを意味します。各要素に割り当てられている労力とコストの重みの値が、組織にとって適切でない場合があります。また、アップグレード プロジェクトはすべて異なっています。これに類似したワークシートが実際のアップグレード プロジェクトで使用されたこともありますが、プロジェクトには考慮すべき新しい条件が常に存在します。また、見積もりワークシートに組み込まれている前提条件がすべてのプロジェクトに適用されるわけではありません。通常、この見積もりをプロジェクトに適用する前に、何らかの調整を行う必要があります。以降では、評価ツールによって Excel ワークブックのコストの見積もりがどのように生成されたかについて説明します。労力はすべて、可能な限り標準的なプロジェクトの見積もりに基づいて算出されます。図 3.14 の数式と重みは、実際のアップグレード プロジェクトでの経験に基づいており、これらの経験を正確に反映しています。

Task	*Effort (Hours)*		*Cost*	*Resource*
Project kick-off	0	$	-	All
Application preparation				
Development environment	0	$	-	DEV
Application resource inventory	0	$	-	DEV,TES
Compilation verification	0	$	-	DEV
Migration order definition	0	$	-	DEV,TES
Upgrade wizard report review	0	$	-	DEV,TES
Total	0	$	-	
Application conversion				
Upgrade wizard execution	0	$	-	DEV,TES
Manual code adjustment	0	$	-	See details
System integration and smoke	0	$	-	DEV,TES
Administrative Tasks	0	$	-	DLE
Total	0	$	-	
Testing and debugging				
Test case creation	0	$	-	STE
Test case execution	0	$	-	TES
Bug management	0	$	-	TES
Regression test	0	$	-	TES
Run-time error correction	0	$	-	DEV
Administrative Tasks	0	$	-	STE
Total	0	$	-	
Project management	0	$	-	PM
Configuration management	0	$	-	CM
Totals	0	$	-	

図 3.14
[Effort –Total] タブ

Effort – Total ワークシートは、アプリケーションの準備、アプリケーションの変換、およびテストとデバッグの 3 つのセクションで構成されています。各セクションについて詳細に説明しますが、以下の点に留意してください。

- [Effort (Hours)] 列の大部分のセルは、[General Configuration] タブの名前付きセルへの参照、または名前付きセルの関数です。変更を行う場合は、Effort ワークシートではなく、General

Configuration ワークシートを変更することをお勧めします。

- 任意のセルまたは隣接する複数のセルに名前を付けて、数式の中で使用できます。これらの名前は、単一の値 (=NameA+NameB など) や値の集合 (=SUM(NameC) など) のように使用されます。
- 整合性を保つため、これらのワークシートと数式は、General Configuration ワークシートまたは EWI Configuration ワークシートのいずれかを一度変更すれば済むように構成されています。General Configuration ワークシートは主に Total Effort ワークシートで参照され、EWI Configuration ワークシートは主に Upgrade Issues Table ワークシートで参照されます。ただし、必要な場合は他のワークシートを変更できます。変更する必要があるもう 1 つのワークシートは Resource Costs ワークシートです。このワークシートでは、アップグレード プロジェクトで作業するさまざまな担当者のコストを指定します。これらの値は国や企業ごとに異なっており、さらには月ごとに異なる場合もあります。したがって、これらの値の見直しが必要になります。

評価ツールによって生成されたワークシートを変更する場合は注意が必要です。評価ツールを再実行すると、新しいワークブックが作成され、ユーザーが行った変更内容が上書きされるおそれがあります。評価ツールを再実行する前に、ワークブック ファイルのバックアップを必ず保存しておくようにしてください。MainReport.xls ファイルに対してのみ変更を行い、このファイルのバックアップを保存しておくことをお勧めします。変更した MainReport ファイルは、任意の DetailedReport.xls ファイルに簡単にリンクさせることができます。これにより、評価ツールを再構成したり、構成情報をレポート間でコピーする必要がなくなります。

アプリケーションの準備

アプリケーションの準備フェーズの詳細については、第 5 章「Visual Basic のアップグレード プロセス」で説明します。ここでは、アプリケーションの準備の定義を拡張し、アップグレードを実行するコンピュータの準備から、最終的にアップグレード ウィザードを実行して新しい Visual Basic .NET アプリケーションのソース ベースとして使用される初期コードを生成するまでのすべての作業を含むものとして解釈します。

別の方法や、アップグレード プロジェクトの承認前に実行されるその他の作業の分析に要する時間は含まれません。アップグレードの決定が既になされており、アップグレード プロジェクトが承認済みであると仮定しています。

また、第 5 章で説明するコードの準備作業も含まれません。この作業には、アップグレードを容易にするための、Visual Basic Code Advisor を実行してコードをクリーンアップする作業などが含まれます。これらの作業を行うことにより、アップグレード全体に要する時間を短縮できますが、このような作業を含めると、見積もりが非常に複雑になります。コードの準備を行ってから、評価ツールを使用することをお勧めします。コードの準備を適切に行うと、評価ツールによる労力の見積もりがコードの準備を行う前と比べて短縮されます。

このフェーズで実行される大部分の作業では、労力とコストの見積もりが固定されているか、アプリケーションに含まれるコードの行数に応じて見積もりが決定されます。これらの作業について説明します。

- **開発環境。**アプリケーションの分析およびアップグレードに使用するアップグレード ウィザードをセットアップします。この労力は固定されており、評価ツールのユーザーが構成できます。
- **アプリケーションのリソース インベントリ。**アプリケーションについての初期調査、およびアプリケーションを Visual Basic 6.0 でコンパイルするために必要なすべてのファイルとツールの収集が含まれます。アプリケーションのアーキテクチャおよびデザインに関する情報を収集するための時間も含まれます。この値はアプリケーションのサイズに依存します。
- **コンパイルの確認。**元のアプリケーションをコンパイルして、必要なファイルがすべて利用可能かどうか確認します。この労力は固定されており、評価ツールのユーザーが構成できます。
- **アップグレード順序の定義。**アップグレード順序は、評価ツールのレポートと、方法や技術的な目的に基づく優先順位によって決定されます。この値もアプリケーションのサイズに依存します。
- **アップグレード ウィザードのレポートの確認。**アプリケーションの準備的なテスト アップグレードにより、システム リソースをテストし、Visual Basic 6.0 で容易に修正できる一般的なエラーとプラクティスを検出します。この作業もアプリケーションのサイズに依存します。

アプリケーションの変換

アプリケーションの変換フェーズには、*未加工コード*を取得して同等の機能を実現するために必要なすべての作業が含まれます。未加工コードとは、アップグレード ウィザードによって生成され、アップグレードの初期コード ベースとして使用される未修正のコードのことです。

このフェーズでは以下の作業を行います。

- **アップグレード ウィザードの実行。**アップグレード ウィザードを実行して、未加工コードを取得します。この労力はアプリケーションのサイズに依存します。
- **問題の解決。**Effort – EWIs ワークシートに表示される問題を手動で解決します。このシートについては、この後で詳しく説明します。
- **コード レビュー。**問題をすべて解決したら、新しいコードをすべて統合し、コンパイルおよびデバッグを実行する必要があります。
- **管理タスク。**スケジュール設定やリソースの割り当てなど、このフェーズで実行される管理タスクです。このフェーズの他の作業に対する割合として計算されます。

テストとデバッグ

この最終フェーズでは、アプリケーションが元のアプリケーションと機能的に同等であることを保証するために必要なすべての品質保証の作業を行います。このフェーズで行うすべての作業は、アプリケーションの変換フェーズの労力の合計に対する割合として計算されます。以下の作業があります。

- **テスト ケースの作成。** 新しいテスト ケースを作成するか、または既存のテスト ケースを調整します。このテストケースは、同等の機能を確認するための品質保証で使用されます。
- **テスト ケースの実行。** アプリケーションの同等の機能を実現するため、テスト時にテスト ケースを複数回実行します。
- **ランタイム エラーの修正。** バグ修正や対処方法など、検出されたランタイム エラーを解決するための作業です。
- **管理タスク。** バグ管理、テスト ケース管理、回帰テストなど、テストとデバッグに関連する管理タスクです。

見積もりについて

既に述べたように、開発プロジェクトの正確な見積もりは非常に難しい作業です。評価ツールによって生成されるレポートと見積もりは、これまでに行われた数多くのアップグレード プロジェクトに基づいており、これらの経験から得られた詳細が可能な限り考慮されます。これまでに説明した 3 つのフェーズ以外の作業は含まれません。

最も重要な作業を特定したら、各作業に必要な時間を決定する必要があります。評価ツールによる見積もりは、開発者の経験に基づく推奨値に過ぎません。レポートで特定されたすべての機能と、これらの機能に割り当てられた時間を検討する必要があります。これらの見積もりが組織の実状を反映していない場合は、値の修正が必要になることがあります。たとえば、その分野に経験がある人材がいないため、より多くの時間が必要になると考えられる場合、または以前にその問題を解決したことがあるため、より短時間で問題を解決できると考えられる場合などです。

予期しない問題の発生についても注意が必要です。評価ツールによって分析されないサードパーティ ライブラリを使用している場合があります。また、Visual Basic 6.0 の機能や動作を通常とは異なる方法で使用しているため、評価ツールによる分析の対象とならない場合もあります。

このため、評価ツールによって生成された値はおおよその見積もりとして捉える必要があります。評価ツールによって特定されたすべての問題を見直し、各問題に割り当てる労力を組織の実状に合わせて調整することにより、より正確な結果を得ることができます。また、見積もりについて開発者の承認を得る必要があります。最終的に作業を実行するのは開発者だからです。

構成設定について

ここでは、評価ツールの構成設定について説明します。これらの各設定は、評価ツールによって生成される MainReport.xls ファイルに表示されます。

Config – General

[Config – General] タブでは、コード 1,000 行ごとの修正および検証など、特定のアップグレードの問題に関する労力の見積もりを構成できます。このタブには以下の列が含まれます。

- Migration Task。この列には、アップグレード時に実行する必要のあるタスクが表示されます。これらのタスクの中には、アプリケーションの特定の特性 (コードの行数など) や、アプリケーションで使用されるコンポーネントの種類 (COM オブジェクトや ADO など) を表しているものがあります。
- Effort per occurrence (Hours)。この列には、1 回のアップグレード作業に必要な時間 (時間単位) を指定します。組織の開発者の能力に合わせてこの値を変更する必要があります。通常、既定値は平均的な値を表します。
- Cost per hour。この列は、これらの問題の 1 つをアップグレードするために必要な 1 時間あたりのコストを表します。この値は、各問題を処理する担当者の技術レベルと給与レベルに応じて大きく異なります。
- Resource。この列は、各問題の解決に割り当てられる推奨のリソースの種類を表します。
- Description。この列には、各アップグレード作業の説明が表示されます。

Config – By Resources

リソースのコストは [Config – By Resources] タブに表示されます。これらのコストは、アップグレードの問題の解決に必要な開発リソースの見積もりを表しています。図 3.15 に、[Config – By Resources] タブの例を示します。

Resource	ID	Cost per hour	Description
Developer	DEV	50	Application developer.
Architect	ARC	80	Senior developer with experience in the design and
Development Leader	DLE	90	Leader of the development team, involved in architecture and
Tester	TES	40	Application tester.
Senior Tester	STE	70	Senior application tester with experience in the original
Project Manager	PM	80	Administrative personnel for scheduling and management.
Configuration	CM	70	Configuration management

図 3.15
[Config –Resources] タブ

[Config – Resources] タブには以下の 4 つの列が含まれます。

- Resource。この列は、アップグレード時に必要とされる、さまざまな経験レベルと技術レベルを表します。
- ID。この列は、リソースの種類の省略形を表します。この値はレポート内のその他のテーブルで使用されます。
- Cost per hour。この列は、各リソースの 1 時間あたりのコストを表します。既に述べたように、この値は組織によって大きく異なるため、組織の実状に合わせて変更する必要があります。
- Description。この列には、各リソースの種類の説明が表示されます。

Config – Fixed Tasks

Config – Fixed Tasks テーブルでは、すべてのアップグレード プロジェクトで実行される一連のプロセスが表示されます。これらのプロセスは、[Config – By Resource] タブに表示されているリソースによって実行されます。図 3.16 に、このタブの例を示します。

Task	Hours	Description
Project Kick-off	2	Time dedicated to the migration project kick-off meeting.
Development Environment Preparation	4	Setup and verification of the development environment of the original application.
Compilation Verification of the Original Application	8	The original application is compiled in order to verify the setup is OK.
Configuration Management	8	Configuration management of the products obtained during the migration project.

図 3.16
[Config – Fixed Task] タブ

[Config – Fixed Tasks] タブには以下の 3 つの列が含まれます。

- Task。この列には、すべてのアップグレード プロジェクトで実行される共通の作業が表示されます。
- Hours。この列には、各作業の実行に必要な時間が表示されます。この値も組織の環境に合わせて変更する必要があります。
- Description。この列には、各作業の説明が表示されます。

Config – EWIs

[Config – EWIs] タブには、EWI のアップグレードの見積もりに関する情報が表示されます。このタブには以下の 7 つの列が含まれます。

- Name。この列には、エラー、警告、および問題の名前が表示されます。
- MSDN ID。この列には、EWI の Microsoft Developer's Network ID が表示されます。
- Resource。この列は、各 EWI の解決に割り当てられる推奨リソースの種類を表します。

- Effort per occurrence (Minutes)。この列には、1 つの EWI をアップグレードするために必要な時間 (分単位) を指定します。組織の開発者の能力に合わせてこの値を変更する必要があります。通常、既定値は平均的な値を表します。
- Effort per occurrence (Hours)。この列は上記の列と同じですが、時間が分単位ではなく時間単位で表示されます。
- Cost per hour。この列は、平均的な開発者が各問題をアップグレードする場合のコストを表します。この値は、各問題を処理する担当者の技術レベルと給与レベルに応じて大きく異なります。
- Functionality。この列には、EWI に関する詳細な説明が表示されます。

まとめ

アップグレード プロジェクトの成功は、アップグレードするプロジェクトの慎重な評価と分析に依存します。これらの作業をすべて適用することにより、正確な計画と予算を作成できます。この章では、これらの作業を実行するために収集する必要のある情報、およびこれらの情報の収集に役立つツールについて説明しました。

詳細情報

アプリケーションのパフォーマンス向上に関する詳細については、MSDN のホワイト ペーパー「Advantages of Migrating from Visual Basic 6.0 to Visual Basic .NET」を参照してください。
http://msdn.microsoft.com/vbasic/productinfo/advantage/default.aspx

ArtinSoft の Visual Basic Companion の詳細については、ArtinSoft の Web サイトを参照してください。
http://www.artinsoft.com/Default.aspx.

4

一般的なアプリケーションの種類

この章では、Visual Basic 6.0 の一般的なアプリケーションの種類について、アップグレード前、アップグレード中、およびアップグレード後の対処方法を説明します。アプリケーションの種類とアプリケーションの実際のアーキテクチャの間の依存関係を、アップグレードの観点から検討します。アプリケーションをアップグレードすることが決まったら、この章を読んで、Visual Basic 6.0 と Visual Basic .NET のアプリケーションの種類の類似点および相違点や、アップグレード後のバージョンで使用できる機能を活用して移行をスムーズに行う方法を学んでください。

アプリケーションの種類の確認とアップグレード

アプリケーションを、同じような要件やアーキテクチャを持つ他のアプリケーションと比較することによって、アプリケーションやそのアップグレードのプロセスについて学ぶことができます。アプリケーションの種類を選択することによって、アプリケーションで解決する問題に最適な機能や特徴のセットを選択することになります。アプリケーションの種類は、対象ユーザー、使用目的、アーキテクチャ、機能などの特徴によって区別されます。

Visual Basic 6.0 では、さまざまなアプリケーションの種類を選択できます。これらは 2 つのカテゴリに分類できます。1 つはアプリケーション コンポーネント、もう 1 つは完全なアプリケーション実行可能ファイルです。

Visual Basic アップグレード ウィザードは、標準実行可能ファイル (.exe)、ActiveX コンポーネント、IIS (インターネット インフォメーション サービス) アプリケーションなど、Visual Basic 6.0 のさまざまなアプリケーションの種類をサポートしています。アップグレード ウィザードにより、元のプロジェクトの種類に応じて、Windows ベースのアプリケーション (.exe) またはクラス ライブラリ (ダイナミック リンク ライブラリ (DLL)) のいずれかを出力する .NET プロジェクトが生成されます。

.NET プラットフォームのサービスと機能を最大限に活用するには、*アップグレード後*のアプリケーションのアーキテクチャと種類を評価する必要があります。たとえば、密結合の (モノリシックな) デスクトップ アプリケーションを、明確に定義された層に複数の独立したコンポーネントを含むアプリケーションに変換すると、Visual Basic .NET が提供する技術を活用できます。この種の改良の詳細については、第 17 章「アプリケーションの改良の概要」を参照してください。

アプリケーションの種類と同等の機能の決定

アプリケーションをアップグレードするときの第 1 の目標は、既存のバージョンと同等の機能をアップグレード後のバージョンで実現することです。表 4.1 は、Visual Basic 6.0 と Visual Basic .NET の、アプリケーション アーキテクチャとプロジェクトの種類の対応関係を示しています。

表 4.1: アプリケーションの種類の対応表

アプリケーション アーキテクチャ	Visual Basic 6.0 のプロジェクトの種類	Visual Basic .NET のプロジェクトの種類
デスクトップ アプリケーション	標準実行可能ファイル	Windows ベースのアプリケーション
Web アプリケーション	IIS (インターネット インフォメーション サービス) アプリケーション	ASP .NET Web アプリケーション
分散アプリケーション	DCOM (分散コンポーネント オブジェクト モデル)、MTS (Microsoft Transaction Services)、または COM+ のサービスを使用するアプリケーション	.NET Framework 名前空間 System Runtime Remoting および System Enterprise Services のコンポーネントを使用するアプリケーション

この章では、各アプリケーション アーキテクチャおよびアプリケーションの種類について、アップグレード プロセスの高レベルな概要を紹介します。

コンポーネントの種類とプロジェクトの種類の決定

Visual Basic アプリケーションは、アプリケーションが必要とする基本機能やサービスを提供するさまざまなコンポーネントで構築されています。こうしたコンポーネントの中には、別のアプリケーション アーキテクチャで使用できるものもあります。たとえば、アウトプロセスの再利用可能なコンポーネント ライブラリは、デスクトップ アプリケーションでも分散アプリケーションでも使用できます。

表 4.2 は、コンポーネントの種類と Visual Basic のプロジェクトの種類の対応を示しています。

表 4.2: コンポーネントとプロジェクトの種類の対応

コンポーネントの種類	Visual Basic 6.0 のプロジェクトの種類	Visual Basic .NET のプロジェクトの種類
再利用可能なライブラリ (アウトプロセス)	ActiveX .exe	直接対応するものはありません。Windows ベースのアプリケーションまたはクラス ライブラリのプロジェクトにアップグレードします。
再利用可能なライブラリ (インプロセス)	ActiveX DLL	クラス ライブラリ
ビジュアル アプリケーション コンポーネント	ActiveX コントロール	Windows コントロール ライブラリ
インターネット アプリケーションでホストされるビジュアル アプリケーション コンポーネント	ActiveX ドキュメント	直接対応するものはありません。UserControl や XML Web フォームなどの .NET Framework コンポーネントを使用して再実装します。

これらのコンポーネントの種類をアップグレードする方法の概要については、この章の後半の「アプリケーション コンポーネント」を参照してください。

デスクトップ アプリケーションと Web アプリケーション

デスクトップ アプリケーションと Web アプリケーションは、アプリケーションを構成するすべてのコンポーネントとソース コードを含む 2 つの基本的な枠組みです。デスクトップ アプリケーションや Web アプリケーションをアップグレードする際には、コンポーネントとその相互接続、各コンポーネントの構造、およびアプリケーションの全体的なアーキテクチャを考慮に入れる必要があります。

ここでは、まずデスクトップ アプリケーションと Web アプリケーションの両方に共通したアップグレードの注意点について説明し、その後、それぞれのアプリケーションの種類に固有の側面に焦点を絞ります。

デスクトップ アプリケーションを .NET Framework にアップグレードする際には、相当な努力を要する場合もあります。アプリケーションのアップグレードが困難になる理由の 1 つは、Visual Basic が .NET Framework のために完全に再設計されていて、関数やコントロールが Visual Basic 6.0 のそれらと厳密に一致しないという点にあります。[1]

アーキテクチャの注意点

第 2 章「アップグレードの成功のためのプラクティス」で説明したように、アップグレードの際には、次の 2 つのいずれかの方法を使用する必要があります。

- **垂直アップグレード**。この方法では、n 層のすべてを通じてアプリケーションの一部を分離し、置き換える必要があります。
- **水平アップグレード**。この方法では、アプリケーションの層全体を置き換える必要があります。

どちらの方法を選択しても同等の機能が実現されていることを確認できるように、単体テストやシステム テストの包括的なテスト スイートを用意する必要があります。このテストでは、アップグレードする個々のモジュール、コンポーネント、または層だけでなく、アプリケーション全体も対象になります。テストできるアプリケーションの部分をプロセスの早い段階でアップグレードすると、複数の開発者または開発チームが並行して作業できるようになったり、問題をプロジェクトの早い段階で修正できるなど、便利な場合があります。ただし、後ほど説明するように、アプリケーションの各部分を常に個別にアップグレードしたりテストしたりできるとは限りません。

アップグレード後のアプリケーションのテストとデバッグの詳細については、第 5 章「Visual Basic のアップグレード プロセス」の「アップグレード後のアプリケーションのテストとデバッグ」と第 21 章「アップグレード後のアプリケーションのテスト」を参照してください。

単一層アプリケーション

単一層 (モノリシック) アプリケーションとは、特定の機能のコードのほとんどまたはすべてが単一のソース コード ファイルに含まれているアプリケーションです。Visual Basic の場合は、機能ロジックのすべてがコントロールのイベント ハンドラ サブルーチンに含まれるのが一般的です。必要なコンポーネントや機能はすべてフォームのグループに含まれており、それらのフォームが、アプリケーションの機能の概念的な単位となります。このようなアプリケーションの開発は、ユーザー インターフェイス駆動型の開発になる傾向があります。アプリケーションで新しい機能が必要になった場合は、既存のフォームに新しいコンポーネントが組み込まれます。たとえば、単一層アプリケーションでデータベース アクセスが必要になると、データ アクセス用のコンポーネントがフォームにプラグインされます。この新たに統合されたコンポーネントは、データベースにアクセスする必要があるすべてのメソッドによって使用されます。

単一層アプリケーションには、データ アクセス、ビジネス ロジック、またはユーザー インターフェイスのための独立した機能層はありません。すべての必要なサービスとコンポーネントが、同じ開発単位 (通常はフォーム) に統合されます。この種のアプリケーションでは、すべてのコンポーネントが (通常は同じファイルまたはプロシージャで) 緊密に結び付いていて、共有されている構造や接続ポイントが数多くあるため、結合が密になります。また、単一層アプリケーションでは、凝集度も低くなります。つまり、異なる概念エンティティを構成するメカニズムが非常に緊密に結び付いているために、概念の境界があいまいになります。この場合、機能に変更を加えると、目的の機能の動作以外の部分にも影響が及ぶ可能性があるため、変更は困難になります。

この種類のアプリケーションをコンパイルすると、ユーザー定義のフォームやモジュールのための必要な情報をすべて含む単一の実行可能ファイルが生成されます。このようなアプリケーションを配置するには、実行可能ファイルと、関連するサードパーティ コンポーネントを、各ユーザーのコンピュータにインストールする必要があります。

ここの残りの部分では、単一層アプリケーションをアップグレードするときに考慮する必要がある問題について説明します。

この例で使用しているのは単一の MDI (マルチ ドキュメント インターフェイス) 子フォームですが、ここで紹介する概念は、複数のフォームやモジュールで構成されたソリューションや、密結合の Web アプリケーションにも当てはまります。

この例のフォーム (Web ページでもかまいません) には、IT ヘルプ デスクで受けた通話の概要が表示されます。ユーザーは、目的のレコードが見つかったら、それに関する詳細情報を要求できます。このフォームには、通話データを含むテーブルのレコードを表示するデータ連結グリッドが含まれています。このグリッドは、ADO (ActiveX データ オブジェクト) データ コントロールをデータ ソースとして使用します。このデータ コントロールは、システムのデータベースとの通信を提供します。フォームには通話の完全なリストが表示されるため、コントロールは非表示になっています。図 4.1 は、このフォームのコンポーネントとアーキテクチャの概要を示しています。

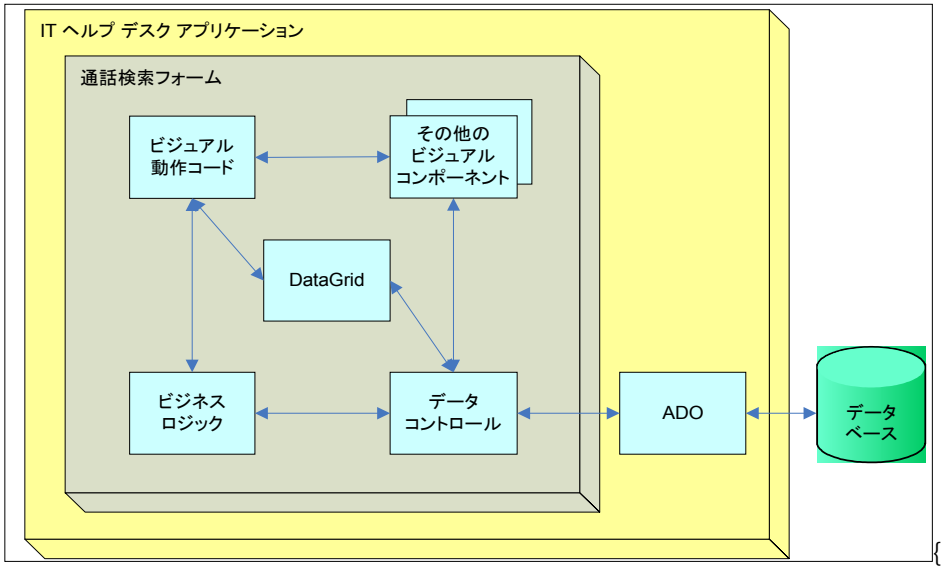

図 4.1

標準的なモノリシック フォームのコンポーネント

外側のボックスは、通話検索フォームを含む単一層アプリケーションを表しています。このフォームには、データ アクセス用のコンポーネント、ビジネス ロジック、およびビジュアル コンポーネントが含まれています。これらは、図の小さなボックスで表されています。図の矢印はコンポーネント間の依存関係を示しています。ビジュアル動作コードは、残りのコンポーネントを管理します。DataGrid コントロールは、ADO データ コントロールを使用して、データベースから抽出した情報を表示します。ビジネス ロジックは、通話の詳細情報が要求されると電話番号を解釈します。ビジネス ロジックとデータ アクセスに関連するボックスは、図では独立したエンティティになっていますが、これらに対応するコードはフォームの残りのコードと混ざり合っている可能性があります。また、すべての機能が通話検索フォームの中に包み込まれていて、このフォームの外部のアプリケーション モジュールからは使用できないという点にも注意する必要があります。

この例では、すべての機能が単一のフォームに完全にカプセル化されているため、すべてのアップグレードと必要な調整が済んでからでないと、フォームをコンパイルしたり、実行したり、テストしたりすることはできません。たとえば、フォームに含まれているビジネス ロジックは、開発者が元のフォームを再構成しない限り、コードの残りの部分と別にアップグレードやテストを行うことはできません。このモノリシックな構造のために、このアプリケーションでは、少しずつアップグレードを重ねたり、機能を分離させてテストを行ったりすることが非常に困難になります。フォームとそのすべての要素は、アプリケーション全体を構成する単一の機能単位としてアップグレードされる必要があります。その意味で、モノリシック アプリケーションのアップグレードは、完全アップグレードの方法に従って行われることになります。

単一層アプリケーションの再設計

単一層アプリケーションをアップグレードする際には、アプリケーションを再設計して、その機能を個別の部分に分割するのが最善の方法になります。この方法では、モジュール性と機能の凝集が実現されるため、アプリケーションの一部に変更を加えたり、機能を追加または削除したり、他のアプリケーションで機能を利用できるようにしたりするのが簡単になります。

垂直アップグレードの方法を使用してアプリケーションの一部を新しいアーキテクチャにアップグレードすると、それが新しいデザインのための有効なテスト ベッドになります。.NET Framework が提供するアーキテクチャ サポートにより、アップグレードした部分をベースとする開発もはるかに容易になります (垂直アップグレードの方法は、2 層アプリケーションや 3 層アプリケーションにも適用できます)。

垂直アップグレードによるアプリケーションの一部のアップグレードが完了したら、相互運用機能を使用して、マネージ コードの一部と既存のアンマネージ コードを統合できます。アプリケーションのどの部分をアップグレードするかによって開発コストが変わってくるため、アップグレードする部分を選択する際には注意が必要です。たとえば、マネージ フォームとアンマネージ ビジネス クラスのグループを統合する場合は、相互運用機能を使用できるため、相互作用する必要があるマネージ フォームとアンマネージ フォームを統合する場合に比べてコストが低くなります。マネージ フォームとアンマネージ フォームを統合する場合は、イベント モデルの違いや、マネージ フォームとアンマネージ フォームのコード間の通信およびマーシャリングのスキームのユーザー定義のために、開発コストが高くなる可能性があります。

アプリケーションの機能のサブセットの統合が完了したら、テストを行う必要があります。これにより、アプリケーションの新しい部分と古い部分が連携し、データを共有して、エンド ユーザーやクライアント開発者にシームレスなユーザー エクスペリエンスを提供できるかどうかを確認できます。このテストに成功すると、基本的なインフラストラクチャが完成します。その後は、アプリケーション コンポーネントの垂直アップグレードを続行できます。

アプリケーションの再構成は、ほとんどの場合、自動アップグレードを実行して同等の機能を実現してから行います。そうすれば、Visual Basic .NET の強化された機能を活用できます。ただし、Visual Basic アップグレード ウィザードやその他の自動アップグレード ツールでサポートされている機能によっては、構造やコンポーネントにいくつかの変更を加えてからでないと、Visual Basic 6.0 のソース コードを自動的にアップグレードできない場合もあります。そのような場合は、アップグレード ウィザードですべての必要なマッピングや変換を適用できるように、コードを再構成したり、サポートされていないクラスをサポートされているクラスに置き換えたりする必要があります。たとえば、DAO (データ アクセス オブジェクト) コンポーネントや RDO (リモート データ オブジェクト) コンポーネントは、若干の再構成を行うことで、Visual Basic アップグレード ウィザードやその他の自動アップグレード ツールでよくサポートされている ADO に変更できます。アップグレード ウィザードでサポートされていない機能やデータ アクセス コンポーネントのアップグレードの詳細については、第 7 ～ 12 章を参照してください。

アップグレード後のアプリケーションの同等の .NET アプリケーション アーキテクチャは、元の Visual Basic 6.0 オブジェクトに相当するスタートアップ オブジェクトを持つ Visual Basic .NET Windows ベース アプリケーション プロジェクトになります (Visual Basic アップグレード ウィザードでは、このプロジェクトのみを含む .NET

ソリューション ファイルが生成されます)。ウィザードで標準的な単一層アプリケーションをアップグレードすると、すべてのライブラリ参照が、対応する .NET Framework 基本クラス ライブラリの一般的な参照に置き換えられます。System 名前空間への参照は、Form クラスの基本機能と XML 機能を提供します。Microsoft.VisualBasic アセンブリは、Visual Basic 6.0 の VBA ライブラリで利用可能な機能と同様の機能を提供します。元のアプリケーションで参照されていた各クラス ライブラリは、それぞれ対応する参照を Visual Basic .NET プロジェクトで持ちます。ActiveX コントロール (.ocx) コンポーネントには、Ax というプレフィックスで始まる追加の参照が含まれます。これは、Windows フォーム コントロール ラッパーに相当します。これらの参照は、aximp ユーティリティによって生成されます。たとえば、コモン コントロールのための MSComctlLib コンポーネントへの参照は、AxMSComctlLib への新しい参照になります。この追加の参照は、すべての Windows フォーム コントロールがサポートするネイティブ メンバと動作のサポートを提供します。

アップグレード後のアプリケーションは、元のアプリケーションと同じ構造を持ちます。アップグレード後のバージョンには、元のアプリケーション コンポーネント間の関係がすべて含まれています。Visual Basic アップグレード ウィザードでは、すべての ActiveX コンポーネントに対して、相互運用ラッパーが必要に応じて自動的に生成されます。たとえば、前に説明した通話フォームの例では、DataGrid コンポーネントと ADO コンポーネントに対してラッパーが生成されます。相互運用機能の詳細については、この章の後半の「アプリケーション コンポーネント」を参照してください。

残りのコンポーネントは .NET Framework のネイティブ クラスにアップグレードされ、ソース コードのほとんどが Visual Basic .NET に変換されるという点に注意することが重要です。ラップされた ActiveX コンポーネントについては、自動ツールの適用後に、手動でネイティブ .NET コンポーネントにアップグレードできます。たとえば、ADO は ADO.NET に、DataGrid コントロールは .NET Framework の DataGrid やサードパーティのデータ グリッドにアップグレードできます。ただし、手動による調整では、完了までにより多くの時間と労力が必要です。

2 層アプリケーション

2 層アプリケーションは、ユーザー インターフェイスとビジネス ロジックを含む 1 つ目の層 (ユーザー インターフェイス層) と、すべてのデータ アクセス コンポーネントを含む 2 つ目の層 (データ層) で構成されています。ユーザーはインターフェイス層とやり取りし、それを使ってすべてのアプリケーション固有の機能を実行します。データ層にはすべてのデータ ソースが含まれており、すべての必要なデータ サービスをユーザー インターフェイス層に提供できるように管理されています。

Visual Basic 6.0 アプリケーションのユーザー インターフェイスには、フォーム、Web ページ、またはその両方を含めることができます。このほか、プログラムによってアクセスできるフロント エンドを含めることもできます。さらに、エンタープライズ アプリケーションでは、リモートからアクセスできるリッチ フォーム クライアントやシン Web クライアントを使用することもできます。

図 4.2 は、前のセクションで説明した通話リストとフィルタ ウィンドウをベースとする 2 層アプリケーションのフォームを示しています。この例では、データ アクセス コンポーネントがフォーム モジュールの一部ではなくなり、代わりに独立して定義されています。これにより、モジュール性とコードの再利用が実現されます。また、

このアーキテクチャでは、単一層アーキテクチャの場合よりアップグレードの選択肢が増えます。これは、層が分離されているために、各層を個別にアップグレードしたりテストしたりできるからです。これにより、より柔軟なアップグレード計画が可能になります。

2 層アプリケーションでは、水平アップグレードと垂直アップグレードの両方が可能です。水平アップグレードを選択する場合は、一方の層全体をアップグレードして、もう一方の層は変更しないまま残しておくことができます。アップグレードした層では、相互運用機能を使用して、アップグレードしていない層と連携できます。後は、時間や予算に応じて残りの層をアップグレードして、対象のアプリケーションに統合できます。このほか、前のセクションで説明した垂直アップグレードの方法を使用することもできます。

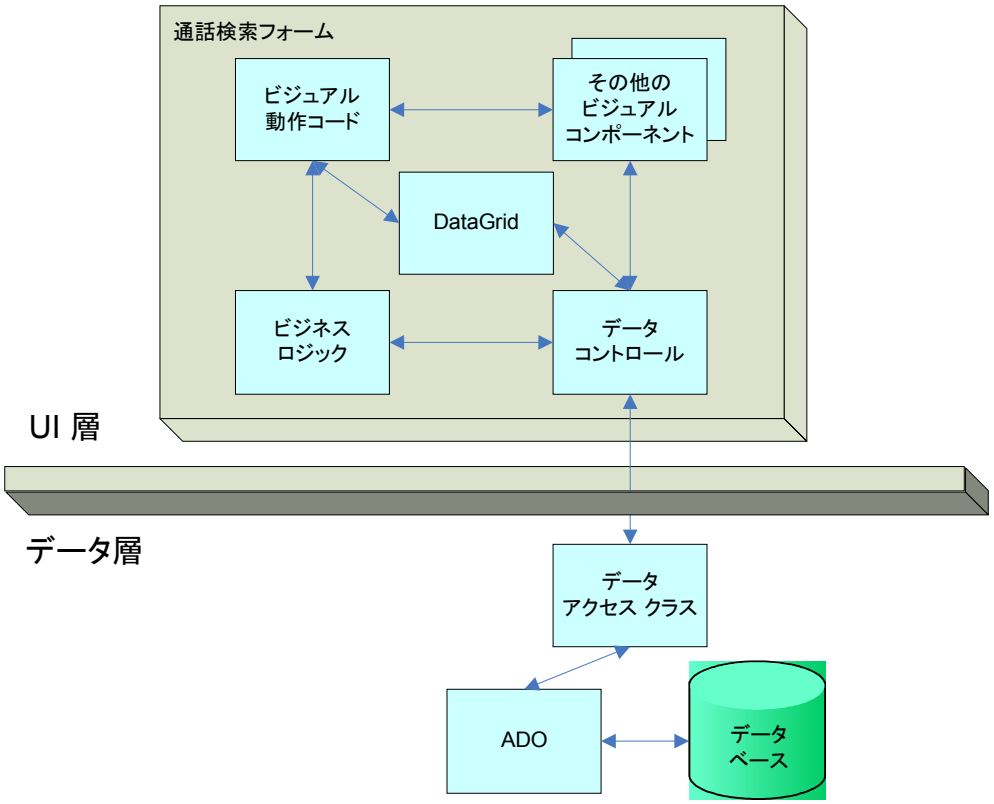

図 4.2

標準的な2 層アプリケーションのアーキテクチャ

2 層アプリケーションをアップグレードすると、アップグレード ウィザードにより、単一層アプリケーションに対して生成されるのと同じようなファイルと参照を含む Visual Basic .NET プロジェクトが生成されます。最大の違いは、データ アクセス クラスがプロジェクト ファイルに含まれることです。このクラスは、他のクラスから独立していて、他の任意のクラスで使用できるサービスを提供します。このクラスには、ラップされたクラスがいくつか含まれています。これらは、基本的なサービスを提供する低レベルのデータ アクセス クラス (ADO) に相当しま

す。ADO を ADO.NET に手動でアップグレードする場合は、このデータ アクセス クラスにも変更を加える必要があります。ただし、アプリケーションがモジュール方式 (疎結合) で設計されている場合は、アプリケーションの残りの部分に加える変更は最小限で済みます。これは、ユーザー定義のデータ アクセス クラスの内部の変更は、それを使用する側には影響しないためです。

3 層アプリケーション

3 層アプリケーションでは、コンポーネントの凝集度は最大になり、層間の結合度は最小になります。このアーキテクチャでは、アプリケーションの異なる側面に対して 3 つの異なる層を使用します。ユーザー インターフェイス層、ビジネス ロジック層、およびデータ層の 3 つです。3 層アプリケーションと 2 層アプリケーションの違いは、ビジネス ロジック層が追加されている点にあります。ビジネス ロジック層には、アプリケーションのルールとアルゴリズムが含まれています。したがって、ユーザー インターフェイス層が担当するのは、ユーザーとアプリケーションの間のやり取りの管理のみになります。ビジネス ロジック層は、さらに複数の論理層に分割して、多層アーキテクチャを作ることもできます。また、ビジネス ロジック層のコンポーネントは、データ層との間に緊密な相互作用を持つこともできます。図 4.3 は、標準的な 3 層アプリケーションを示しています。

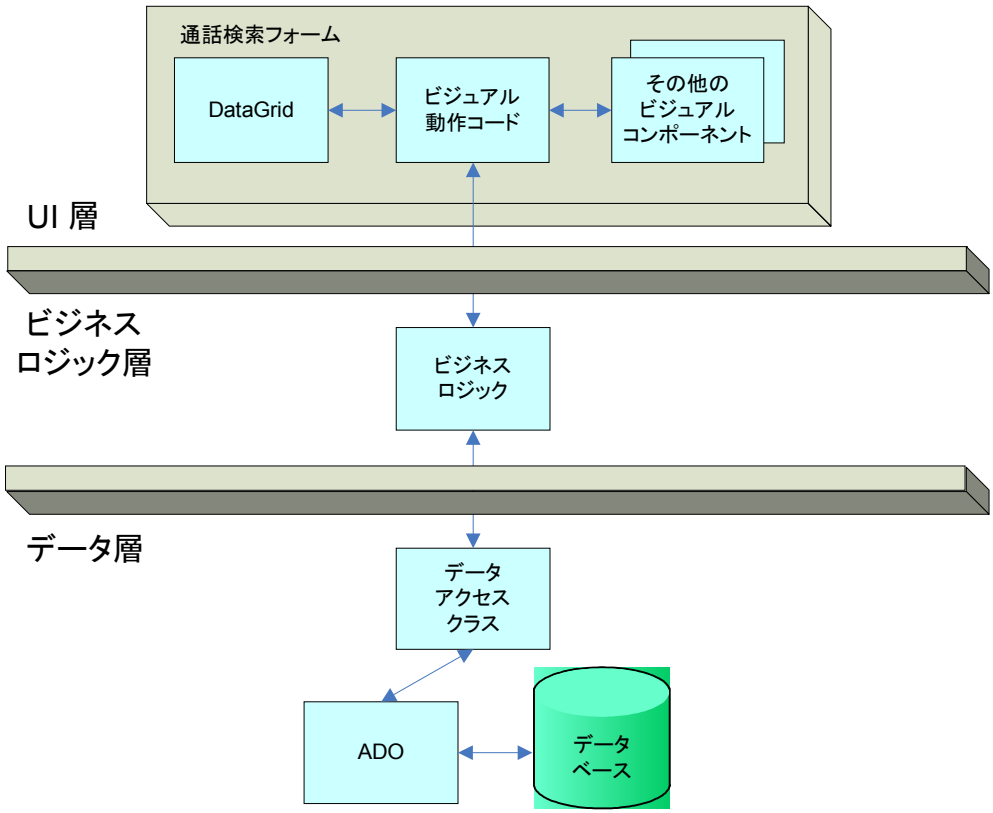

図 4.3

標準的な3 層アプリケーションのアーキテクチャ

3 層アプリケーションでは層がさらに分離されているため、2 層アプリケーションの場合よりさらに柔軟なアップグレードの方法が可能になります。2 層アプリケーションの場合と同様に、3 層アプリケーションでも、水平アップグレードと垂直アップグレードの両方が可能です。水平アップグレードを使用すると、いずれかの層のすべてまたは一部をアップグレードして、その他の層は変更しないまま残しておくことができます。アップグレードしたコードでは、相互運用機能を使用して、まだ変更していないコードにアクセスできます。後は、徐々に他の層をアップグレードしてアプリケーションに統合できます。

3 層アーキテクチャのもう 1 つの重要な特徴は、開発者の専門技術を活かして作業を並行して進められるという点にあります。たとえば、他の開発者がビジネス ロジック層やデータ アクセス層のコンポーネントのアップグレードや調整を行っている間に、特定の開発者または開発チームがモジュール式のユーザー インターフェイス コンポーネントの作業を行うことなどが可能になります。

3 層アプリケーションをアップグレードすると、結果となるアプリケーションの層も 3 つになります。アップグレード ウィザードでは、UI 層で使用されているコンポーネントに対して ActiveX ラッパーが生成されます。UI コンポーネントの中には、ラッパー、カスタム アップグレード、再実装などを必要とするものもあるため、追加の UI テストを実行する必要がある場合も考えられます。ただし、UI 層を先にアップグレードし、相互運用機能を使用して他の層に接続して、プロジェクトの早い段階で UI 層をテストすることによって、関連するリスクを軽減することができます。この方法を使用すると、他の層のアップグレード、テスト、および統合を並行して行うことができます。どの層を最初にアップグレードすればよいかに関するその他のアドバイスについては、第 2 章「アップグレードの成功のためのプラクティス」を参照してください。

アップグレード後の 3 層アプリケーションの配置では、労力の大半がシステム サーバーの設定に費やされます。クライアントがアプリケーションにアクセスできるようにするには、アプリケーション サーバーとデータ サーバーを設定する必要があります。ほとんどのコンポーネントがサーバーにインストールされていて、クライアント コンピュータはインターネット ブラウザを使用してアプリケーションにアクセスするため、クライアントのセットアップの方が簡単です。

デスクトップ アプリケーション

Visual Basic 6.0 デスクトップ アプリケーションを作成するには、標準実行可能プロジェクト テンプレートを使用します。既定では、このテンプレートは空白のフォームで始まります。このフォームが、主要なインターフェイスの 1 つになります。Visual Basic 6.0 の統合開発環境 (IDE) では、補助的なコンポーネントや参照を追加することによって既存の機能を活用できます。

密結合のアーキテクチャを使用する標準的なデスクトップ アプリケーションは、ファット クライアント、ビジネス ロジックを含む中間層 COM コンポーネント (ない場合もあります)、データ アクセス層、およびデータ ストアへの接続で構成されます。この種類のアプリケーションでは、プレゼンテーションとビジネス ロジックはクライアント側にあります。

一般に、デスクトップ アプリケーションには次のような特徴があります。

- **使用方法**。多くの視覚的な手がかりや補助機能がアプリケーションに用意されていて、ユーザーとの間で複雑なやり取りが行われます。
- **アーキテクチャ**。アプリケーションのアーキテクチャは柔軟です。たとえば、単一の層から成る完全に結合されたアプリケーションも、明確に定義されたビジネス ロジック層とデータ アクセス層を持つ完全にモジュール化されたアプリケーションも可能です。
- **必要な機能**。アプリケーションには、ユーザー インターフェイス コンポーネント、基本サービス、およびデータ アクセス機能が必要です。マルチスレッド機能が必要な場合もありますが、アプリケーションが DCOM (分散コンポーネント オブジェクト モデル) や COM+ を使用する場合を除いて、同期したプロセス分散が必要になることはほとんどありません。

Visual Basic 6.0 to Visual Basic .NET Upgrade Wizard では、デスクトップ アプリケーションは自動的に Visual Basic .NET Windows ベース アプリケーションにアップグレードされます。コードの全般的な構成は変更されません。したがって、元のアプリケーション層はそのまま残ります。ただし、デスクトップ アプリケーションをアップグレードする際には、アーキテクチャ関連の問題を考慮に入れる必要があります。たとえば、次のような問題があります。

- **言語の問題**。DLL 関数の外部宣言やスレッド セーフなどの言語の問題を解決するために、手動による追加のアップグレードが必要になる場合があります。たとえば、基本的な API サービスやオブジェクト間の同期を提供するために、クラスを作成したり変更したりする作業などがこれに含まれます。外部宣言は、.NET Framework クラスと新しいインターフェイス クラスにアップグレードできます。さらに参照を追加する必要がある場合もあります。Visual Basic .NET コンポーネント (UserControl を含む) はもともとスレッド セーフなわけではないということに注意する必要があります。複数のスレッドがコンポーネントに同時にアクセスできるため、結果として、Visual Basic 6.0 ではスレッド セーフなコードが、Visual Basic .NET にアップグレードするとスレッドセーフでなくなる可能性があります。Internet Explorer、インターネット インフォメーション サービス、COM+ などのマルチスレッド環境でコンポーネントを実行している場合に、この変更の影響を受けます。[2]この場合は、新しい同期メカニズムを追加して、スレッド セーフの問題を解決する必要があります。

- **フォームの問題**。Dynamic Data Exchange (DDE) とドラッグ アンド ドロップ機能は、アップグレードの際に問題になる可能性があります。DDE は、Visual Basic .NET でも .NET Framework でもサポートされていません。Visual Basic のドラッグ アンド ドロップは過去の機能と見なされるため、アップグレード ウィザードではドラッグ アンド ドロップ関連のコードはアップグレードされません。代わりに、Visual Basic 6.0 のドラッグ アンド ドロップのコードがアップグレード後の Visual Basic .NET プロジェクトに保持されます。このコードはコンパイルされません。したがって、このコードを削除してドラッグ アンド ドロップ機能を取り除くか、OLE のドラッグ アンド ドロップを使用するようにコードを変更する必要があります。これらの問題の詳細については、第 9 章「Visual Basic 6.0 フォーム機能のアップグレード」を参照してください。
- ActiveX コントロールのアップグレード。アップグレード ウィザードは、ほとんどの ActiveX コンポーネントをサポートしています。これらのコンポーネントのアップグレードは、元のコンポーネントの周囲に構築できる相互運用機能を活用して行われます。ただし、アプリケーション コントロールがサポートされていない場合は、アプリケーションの一部の機能 (特にユーザー インターフェイスの機能) を再実装および再構成する必要があります。複雑なサードパーティ コントロールを広範に使用しているアプリケーションでは、単純化したユーザー インターフェイスを使用して限定的なアップグレード テストを実行する必要があります。これにより、完全に、または部分的にサポートされているコントロールの数を特定できます。

> **Visual Basic 2005 の場合**:
>
> Visual Studio 2005 バージョンの Visual Basic アップグレード ウィザードでは、ラップされたコントロールではなく .NET のネイティブ コントロールにアップグレードされる ActiveX コントロールの数が増えています。新たにサポートされたコントロール (Microsoft Windows コモン コントロール ライブラリに含まれています) には、**ToolBar**、**StatusBar**、**ProgressBar**、**TreeView**、**ListView**、**ImageList** などがあります。

アップグレード後の .NET デスクトップ アプリケーションを配置するには、以下の作業を行う必要があります。

- アプリケーションの実行可能ファイルとライブラリ ファイルをクライアント コンピュータにインストールします。
- .NET Framework Assembly Registration Utility (regasm) を使用して、別のアプリケーションと共有する .NET Framework ライブラリ アセンブリを登録します。
- アプリケーションに ActiveX コンポーネントが含まれている場合は、各クライアント コンピュータでそれらを登録します。
- 対応するすべての ActiveX ラッパー アセンブリを配布します。

Web アプリケーション

Visual Basic 6.0 による Web サイトや Web アプリケーションの構築では、ほとんどの場合、以下の技術が使用されます。

- Active Server Pages (.asp)。インタラクティブな Web サーバー アプリケーションを作成するために使用されるサーバー側スクリプト ランタイム環境です。.asp ファイルには、HTML セクション、スクリプト コマンド、および COM コンポーネントを含めることができます。COM コンポーネントは、Web サーバーで実行されて、クライアントに Web ページを表示します。
- Internet Information Server (IIS) **プロジェクト**。Web サーバーで実行され、クライアントからの HTTP 要求に応答する WebClass で構成されたアプリケーションです。WebClass では、アプリケーションの

状態情報を処理できます。IIS プロジェクトのアップグレードの詳細については、付録 C「ASP のアップグレードに関する概要」と第 10 章「Web アプリケーションのアップグレード」を参照してください。

機能的に同等な Visual Studio .NET のサーバー側技術は、ASP.NET です。これは、単なる ASP の新しいバージョンではありません。.NET Framework をベースとして、完全な再設計が行われています。ここでは、ASP コードを ASP.NET にアップグレードする際に考慮する必要がある主な注意点にスポットを当てます。

図 4.4 は、Visual Basic 6.0 Web アプリケーションのアーキテクチャの概要を示しています。クライアント Web ブラウザが IIS に要求を送信し、IIS は Active Server Pages を表示することによって応答します。ASP ファイルは、アプリケーションのビジネス ロジック層とデータ アクセス層の一部を含む COM コンポーネントにアクセスします。これらのコンポーネントは、Web サーバー コンピュータまたはアプリケーション サーバーで実行されます。背後にある大きなボックスは、異なる概念層に機能を提供するプラットフォーム サーバーを表しています。たとえば、Web サーバーでは ASP ファイルの実行可能コードだけでなくプレゼンテーション要素も提供しなければならないため、Web サーバーのボックスには、ビジネス ロジックの機能とユーザー インターフェイスの機能の一部が含まれています。ASP ファイルでは、アプリケーションのロジックとプレゼンテーション要素が 1 つのモジュールに含まれています。Active Server Pages のボックスが UI 層とビジネス ロジック層の境界にまたがっているのはこのためです。

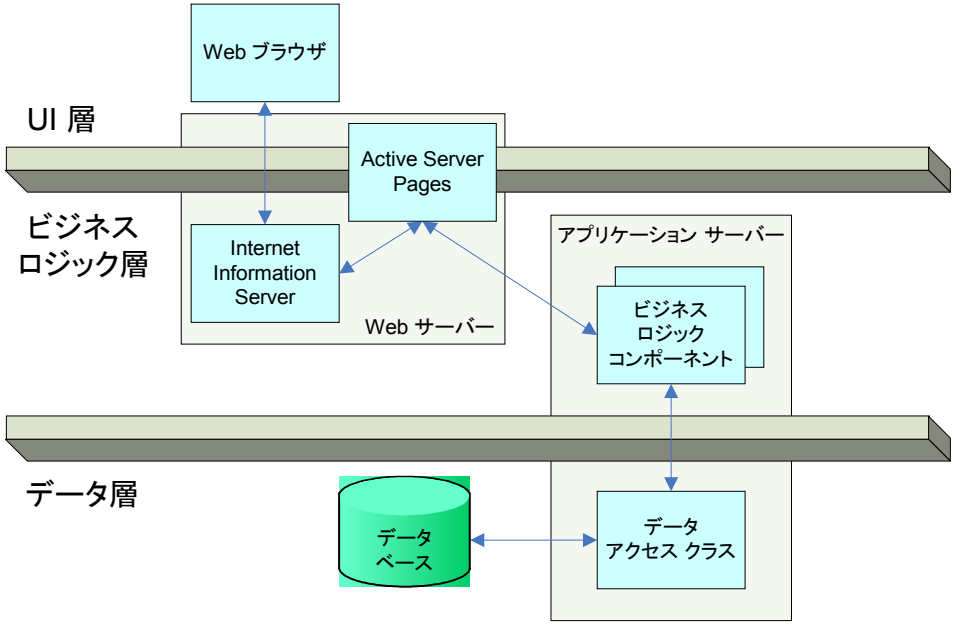

図 4.4

Visual Basic 6.0 Web *アプリケーションのアーキテクチャ*

図 4.5 は、ASP.NET Web アプリケーションのアーキテクチャを示しています。図 4.4 を図 4.5 と比較すると、2 つの技術の基本的な違いや、アップグレードの際に焦点となる部分がわかります。

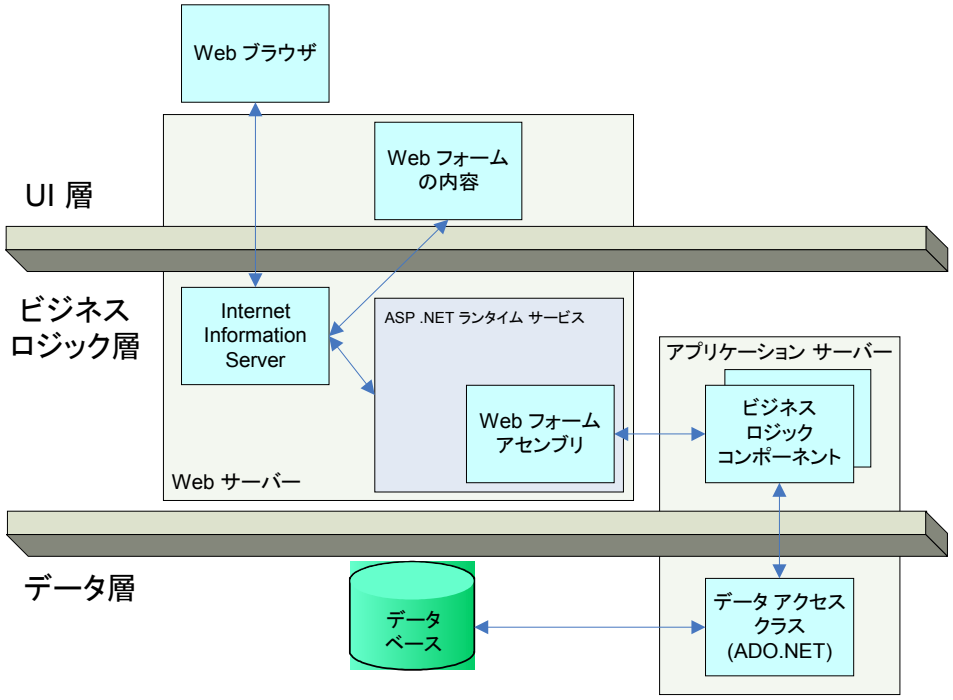

図 4.5

ASP.NET Web アプリケーションのアーキテクチャ

ASP から ASP.NET への移行においては、アーキテクチャの変更のほとんどが、Web サーバーに配置されるコンポーネントで行われています。Web ドキュメントのモデルは完全に変更されています。ASP.NET では、Web ページは Web フォームとして構築されます。Web フォームでは、ドキュメントの内容と、ドキュメントのプレゼンテーション ロジックと動作が、別のファイルに分かれています。したがって、Web ページのビジネス ロジックの側面とユーザー インターフェイスの側面が分離しています。このモデルでは、Web フォームの内容のボックスは UI 層に、Web フォームの実行可能コード (Web フォーム アセンブリ) はビジネス ロジック層に含まれています。ASP.NET ランタイム コンポーネントは、Web セキュリティ、キャッシュ、その他のパフォーマンス機能などの追加のサービスを提供します。

共存

ASP と ASP.NET は同じ Web サーバーに共存できます。ただし、互いに通信しない別のプロセスとして実行されます。

1 つのサイト内の 1 つの Web アプリケーションに、ASP.NET ページと ASP ページの両方を含めることができます。このことには、大規模で機能的に分断されている変化の激しいサイトを部分ごとに ASP.NET に移行する必要がある場合などに、いくつかのメリットがあります。互いに依存していないアプリケーションの別々のセクションを ASP.NET にアップグレードして、アプリケーションの残りの部分と共存させることができます。ASP.NET への移行を長期的な計画として行う場合は、この機会に、アプリケーションのアーキテクチャやデザインをできるだけ多く改良する必要があります。ASP へのアップグレードの詳細については、MSDN の「Migrating to ASP.NET: Key Considerations」を参照してください。

同じ Web サーバーで ASP と ASP.NET の両方のページにアクセスできるため、既存の ASP ページを ASP.NET 互換ページにアップグレードする必要はありません。ただし、アップグレードするとさまざまなメリットがあります。以下にその代表的な例をいくつか紹介します。

- **パフォーマンスの向上**。独立したテストの結果によると、ASP.NET アプリケーションが 1 秒間に処理できる要求の数は、従来の ASP アプリケーションの 2 ～ 3 倍に上ります。パフォーマンスの強化の詳細については、第 1 章「はじめに」を参照してください。
- **安定性の向上**。ASP.NET ランタイムは、プロセスを緊密に監視および管理します。プロセスで問題 (リークやデッドロックなど) が発生した場合、ASP.NET では、新しいプロセスを作成して置き換えることができます。これにより、アプリケーションによる要求の処理を中断させずに済みます。
- **開発者の生産性の向上**。ASP.NET のサーバー コントロールやイベント処理などの機能は、アプリケーションをよりすばやく、より少ないコードで作成するのに役立ちます。また、より簡単にコードを HTML コンテンツから分離できます。

ASP.NET への変換の詳細については、MSDN の「Converting ASP to ASP.NET」を参照してください。

ASP to ASP.NET Migration Assistant

マイクロソフトでは、ASP ページを ASP.NET に自動的に変換するためのツールを無償で提供しています。このツールでは、ASP のすべての機能が自動的に変換されるわけではありませんが、一部のステップを自動化できるため、アップグレード プロジェクトがより簡単になります。移行アシスタントをダウンロードするには、MSDN の Microsoft ASP.NET Developer Center の "ASP to ASP.NET Migration Assistant" を参照してください。

ASP を使用するエンタープライズ アプリケーションをアップグレードする場合は、2 つの異なるアップグレード アシスタントを使用する必要があります。ビジネス ロジック層、データ アクセス層のユーザー定義クラス、およびアプリケーション サーバーで実行される追加のコンポーネントがアプリケーションに含まれている場合は、

Visual Basic アップグレード ウィザードを使用する必要があります。これらの基盤コンポーネントのアップグレードが完了したら、ASP to ASP.NET Migration Assistant を使用して ASP コードを変換できます。

ASP to ASP.NET Migration Assistant の詳細については、付録 C「ASP のアップグレードに関する概要」を参照してください。

ASP.NET への ASP アプリケーションの移植

ASP アプリケーションを ASP.NET にアップグレードする際には、既存の ASP アプリケーションに ASP.NET の機能を組み込む作業にどのくらいの時間をかけるかを決定する必要があります。この作業を行うには、次の 2 とおりの方法があります。

- ASP ページで移行アシスタントを実行した後に手動の変更を適用する。
- ASP.NET Web コントロール、ADO.NET、.NET Framework のクラスなど、.NET の機能の多くを使用できるようにアプリケーションを再構成する。

後者の方法を使用すると、ASP.NET ページがより読みやすく、管理しやすく、機能豊富になりますが、アップグレードの完了までにより多くの時間と労力を必要とします。

主な注意点

既存の ASP ページを ASP.NET ページにアップグレードする際には、以下の点にも注意が必要です。

- API **コアの変更**。ASP の API コアには、少数の組み込みオブジェクト (Request、Response、Server など) とその関連メソッドがあります。いくつかの単純な変更点を除けば、これらの API は ASP.NET でも正しく機能します。
- **構造の変更**。構造の変更とは、ASP ページのレイアウトやコーディング スタイルに影響する変更です。コードが ASP.NET で確実に機能するようにするためには、いくつかの構造の変更を認識しておく必要があります。
- Visual Basic **言語の変更**。VB スクリプトと Visual Basic .NET スクリプトの間には、アプリケーションをアップグレードする前に準備が必要となる変更がいくつかあります。
- COM **関連の変更**。COM はまったく変更されていません。ただし、ASP.NET で使用した場合に COM オブジェクトがどのように動作するのかを把握しておく必要があります。
- **アプリケーションの構成の変更**。ASP では、Web アプリケーションの構成情報はすべてシステム レジストリと IIS メタベースに格納されますが、ASP.NET では、各アプリケーションに固有の Web.config ファイルがあります。
- **状態管理の問題**。アプリケーションで組み込みオブジェクトの Session または Application を使用して状態情報を格納している場合は、ASP.NET でもそれらを何の問題もなく使用できます。追加のメリットとして、ASP.NET では状態の格納場所のオプションが追加されています。

- **基本クラス ライブラリ**。基本クラス ライブラリ (BCL) は、開発するアプリケーションの種類 (ASP.NET アプリケーション、Windows フォーム アプリケーション、Web サービスなど) に関係なく使用できる基本的なビルド ブロックのセットを提供します。BCL を使用すると、アプリケーションを大幅に改良できます。

これらの各注意点の詳細については、付録 C「ASP のアップグレードに関する概要」と、MSDN の「Migrating to ASP.NET: Key Considerations」を参照してください。

アップグレード後の Web アプリケーションの配置では、1 つのクライアントと 1 つ以上のサーバー コンピュータのインストールが必要になります。すべての必要なクライアント側 ActiveX コンポーネントをクライアントにインストールし、サーバー コントロール、コンポーネント ライブラリ、およびデータ プロバイダの完全なセットを使ってサーバーをセットアップする必要があります。

アプリケーション コンポーネント

マイクロソフトは、アプリケーション間の相互作用を実現し、コードの再利用のためのプラットフォームを提供するために、コンポーネント オブジェクト モデル (COM) を開発しました。

Visual Basic 6.0 では、次のような種類の COM コンポーネントを作成できます。

- ActiveX **コード ライブラリ**。COM 実行可能プログラムまたは DLL としてコンパイルできるコンポーネントです。これらのライブラリには、クライアント アプリケーションでインスタンスを作成することによって使用できるクラスが含まれています。これらの COM コンポーネントを作成するために使用するプロジェクト テンプレートは、Visual Basic 6.0 では ActiveX 実行可能ファイルおよび ActiveX DLL と呼ばれています。
- ActiveX **コントロール**。再利用可能なフォームやダイアログ ボックスをすばやく組み立てることができる標準のインターフェイス要素です。
- ActiveX **ドキュメント**。ドキュメント コンテナ内でホストおよびアクティブ化する必要がある COM コンポーネントです。この技術を使用すると、Internet Explorer などの汎用シェル アプリケーションでさまざまな種類のドキュメントをホストできます。

ここでは、サポートされている任意の種類のアプリケーション コンポーネントをアップグレードする際に使用できる一般的な情報を紹介します。

ネイティブ DLL とアセンブリ

Visual Basic 6.0 では、アプリケーション コンポーネントは DLL としてコンパイルされます。DLL には、クライアント アプリケーションによって使用される実行可能コードが含まれています。これらのライブラリは、ファイル名に拡張子 .dll または .exe を持つファイルとして配置されます。ライブラリの要素にクライアント アプリケーションがアクセスできるようにするには、ライブラリの情報をシステム レジストリに登録する必要があります。

アセンブリは、.NET Framework ベースのアプリケーションの主要なビルド ブロックであり、単一の実装単位としてビルド、バージョン管理、および配置されるコンポーネント ライブラリです。すべてのアセンブリに、アセンブリを記述するマニフェストが含まれています。マニフェストには、以下の情報を保持するライブラリのメタデータが含まれています。

- アセンブリの名前、バージョン、カルチャ、およびデジタル署名 (アセンブリがアプリケーション間で共有される場合)
- アセンブリのビルドに使用されたファイルの名前
- アセンブリで使用されているリソースの名前 (アセンブリからエクスポートされるリソースに関する情報を含む)
- 他のアセンブリとのコンパイル時の依存関係
- アセンブリに必要な権限

この情報は、.NET 共通言語ランタイム (CLR) によって、参照の解決、バージョン バインド ポリシーの実施、および読み込まれたアセンブリの整合性の検証に使用されます。CLR では、アセンブリ マニフェストを使用することにより、.NET コンポーネントを事前にシステム レジストリに登録しなくてもクライアント アプリケーションでコンポーネントを使用できます。その結果、アプリケーションの配置が単純化され、バージョン管理の問題が軽減されます。コンポーネント ベースのシステムで発生するバージョン管理と配置の問題は、エンド ユーザーや開発者の間で周知のものとなっています。これらの問題の多くは、COM クラスのアクティブ化に必要なレジストリ エントリに関連しています。.NET Framework のアセンブリを使用すると、こうした配置の問題の多くを解決できます。アセンブリでは、アプリケーションのゼロインパクト インストールが促進されるほか、アプリケーションのアンインストールや複製も簡単になります。

バージョン管理の問題

現在 Win32 アプリケーションで発生しているバージョン管理の問題は 2 つあります。

- アプリケーションの部分ごとにバージョン管理規則を指定したり、それをオペレーティング システムで実施したりできない。現在の方法は下位互換性に依存していますが、下位互換性の保証が困難な場合もよくあります。
- 一緒にビルドされたコンポーネントのセットと実行時に存在するセットの間の一貫性を維持する方法がない。

これらのバージョン管理の問題が組み合わされて、DLL の競合が発生します。DLL の競合とは、アプリケーションをインストールした結果、前のバージョンとの下位互換性が十分に確保されていないソフトウェア コンポーネントや DLL がインストールされて、意図せずに既存のアプリケーションが破壊される状態を指します。いったんこの問題が発生すると、システムのサポートを利用して診断や修正を行うことはできません。

アセンブリのソリューション

バージョン管理の問題や、DLL の競合を引き起こすその他の問題を解決するために、.NET CLR では、以下の目標を達成するための手段として、アセンブリが提供されています。

- さまざまなソフトウェア コンポーネント間のバージョン管理規則を開発者が指定できるようにする。
- バージョン管理規則を実施するためのインフラストラクチャを提供する。
- コンポーネントの複数のバージョンを同時に実行（*サイドバイサイド実行*）できるようにするためのインフラストラクチャを提供する。

.NET と COM の相互運用性

アプリケーション コンポーネントを Visual Basic 6.0 から Visual Basic .NET にアップグレードする際には、.NET CLR の相互運用機能を考慮に入れる必要があります。たとえばサードパーティ コンポーネントをアップグレードする場合には、相互運用ラッパーを使用する、.NET Framework コンポーネントにアップグレードする、Visual Basic .NET 用に設計されたサードパーティコンポーネントにアップグレードするなど、さまざまなメカニズムを利用できます。必要とされる労力、期待される結果、利用可能なリソース、今後のアプリケーション開発計画など、さまざまな要因を考慮に入れて、アプリケーションに適した方法を選択する必要があります。

相互運用ラッパーを使用すると、.NET プログラミング言語から COM コンポーネントにアクセスすることができます（COM クライアントから .NET アセンブリに直接アクセスすることはできません）。Visual Basic 6.0 アプリケーションでは、変換されたコンポーネントとラップされたコンポーネントの組み合わせにアクセスできます。たとえば、COM コンポーネントの相互運用ラッパーを作成し、それらを Visual Basic .NET でテストすることができます。ラップされたコンポーネントが期待どおりに動作した場合は、ラッパーを通じてそのコンポーネントにアクセスできます。また、.NET にアップグレードすることもできます。

▶ COM ラッパーを作成するには

1. Regsvr32 ツール（Regsvr32.exe）を使用して、元の COM コンポーネントを移行先のコンピュータに登録します。
2. タイプ ライブラリ インポータ ユーティリティ（Tlbimp.exe）を使用して、相互運用ラッパーを作成します。
3. コンポーネントが ActiveX コントロールの場合は、Windows フォーム ActiveX コントロール インポータ ユーティリティ（Aximp.exe）を使用して、Windows フォーム ActiveX コントロール インポータを生成します。

COM ラッパーの作成の詳細およびコード例については、第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」を参照してください。

図 4.6 は、相互運用機能がどのようにセットアップされ、それを .NET クライアントがどのように使用するのかを示しています。

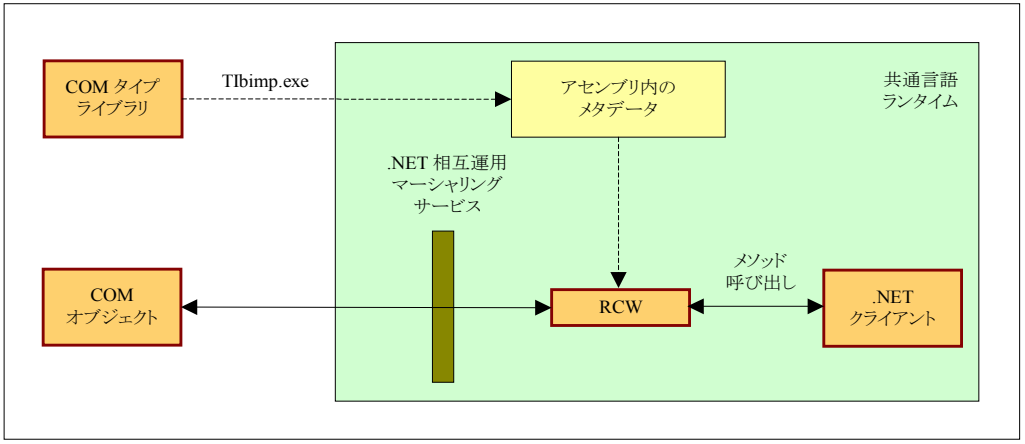

図 4.6

相互運用機能による COM コンポーネントへのアクセス

図 4.6 では、タイプ ライブラリ インポータ ユーティリティ (TIbimp.exe) が COM ライブラリを分析して、型定義をアセンブリ内の同等の定義に変換しています。アセンブリには、元の COM ライブラリに相当するメタデータが含まれます。CLR は、各 COM オブジェクトに対してランタイム呼び出し可能ラッパー (RCW) を作成します。RCW の主な機能は、.NET クライアントと COM オブジェクトの間の呼び出しのマーシャリングです。このマーシャリングでは、2 つの技術が通信できるようにするためのデータ型の変換が行われます。.NET クライアントは、RCW を媒介として使用することで COM ライブラリにアクセスできます。

作成される相互運用ラッパーは、COM コンポーネントをラップする .NET アセンブリです。このアセンブリを別のアプリケーションと共有する場合は、グローバル アセンブリ キャッシュに登録する必要があります。アセンブリをグローバル アセンブリ キャッシュに配置するのは、次のような理由によります。

- **グローバル アセンブリ キャッシュを使用すると、共有アセンブリを集中管理できます。**アセンブリを複数のアプリケーションで使用する場合は、グローバル アセンブリ キャッシュに配置する必要があります。
- **グローバル アセンブリ キャッシュを使用すると、ファイルのセキュリティを強化できます。**多くの管理者は、アクセス制御リスト (ACL) を使用して書き込みアクセスや実行アクセスを制御することにより、WINNT ディレクトリを保護しています。グローバル アセンブリ キャッシュは WINNT ディレクトリにインストールされるため、そのディレクトリの ACL を継承します。
- **グローバル アセンブリ キャッシュを使用すると、サイドバイサイドのバージョン管理が可能になります。**グローバル アセンブリ キャッシュでは、名前が同じでバージョン情報が異なる複数のアセンブリのコピーを保持できます。
- **グローバル アセンブリ キャッシュは CLR が最初に検索する場所です。**共通言語ランタイムは、構成ファイルのコード ベース情報をプローブまたは使用する前に、アセンブリ要求に一致するアセンブリがグローバル アセンブリ キャッシュにないかどうかをチェックします。

ラッパーの詳細については、MSDN の『.NET Framework Developer's Guide』の「Customizing Standard Wrappers」を参照してください。

こうしたメリットがあるとはいえ、必要な場合以外は、アセンブリをグローバル アセンブリ キャッシュにインストールして共有しないようにしてください。一般的なガイドラインとしては、明らかにアセンブリを共有する必要がある場合以外は、アセンブリの依存関係を公開しないようにして、アセンブリをアプリケーション ディレクトリに配置します。また、アセンブリからCOM 相互運用機能やアンマネージ コードにアクセスできるようにするために、アセンブリをグローバル アセンブリ キャッシュにインストールする必要はありません。

アセンブリをグローバル アセンブリ キャッシュにインストールしないことが推奨される場合もあるので注意してください。たとえば、アプリケーションのアセンブリの 1 つをグローバル アセンブリ キャッシュに配置すると、XCOPY インストールを使用してそのアプリケーションを複製したりインストールしたりできなくなります。この場合は、アプリケーションのすべてのアセンブリをグローバル アセンブリ キャッシュに配置する必要があります。アセンブリの使用の詳細については、MSDN の『.NET Framework Developer's Guide』の「Working with Assemblies and the Global Assembly Cache」を参照してください。

アセンブリをグローバル アセンブリ キャッシュにインストールするには、2 つの方法があります。

- Microsoft Windows インストーラ 2.0 を使用する。アセンブリをグローバル アセンブリ キャッシュに追加するための最も一般的な方法です。通常はこの方法をお勧めします。
- Global Assembly Cache tool (Gacutil.exe) を使用する。この方法は開発目的の場合に推奨されます。本番のアセンブリをグローバル アセンブリ キャッシュに追加する際には、この方法を使用しないでください。

再利用可能なライブラリ

Visual Basic 6.0 コンポーネントは、以下の 3 つの場所で実行できます。

- クライアントと同じアドレス空間 (インプロセス)
- 同じコンピュータ (アウトプロセス)
- リモート コンピュータ (アウトプロセス)

インプロセス コンポーネントは、DLL または ActiveX コントロールとして実装できます。コンポーネントを使用する各クライアント アプリケーションは、コンポーネントの新しい インスタンスを開始します。

アウトプロセス コンポーネントは、実行可能ファイルとして実装されて、ローカルまたはリモートのコンピュータの独自のプロセス空間で実行されます。クライアントとアウトプロセス コンポーネントの間の通信では、プロセス境界におけるパラメータと戻り値のマーシャリングが必要になります。アウトプロセス コンポーネントは、1 つのインスタンスで多くのクライアントにサービスを提供できます。グローバル データの共有や、クライアント アプリケーション間の分離も可能です。この種のコンポーネントは、DCOM や COM+ を使って実装できます。この種類のコンポーネントのアップグレードについては、この後の「分散アプリケーション」を参照してください。

ここの残りの部分では、ActiveX コード コンポーネントをアップグレードする方法について説明します。これらのコンポーネントは、Visual Basic 6.0 ActiveX DLL プロジェクト テンプレートに相当します。

ほとんどの場合、コード コンポーネントにはユーザー インターフェイスはありません。代わりにクラスのライブラリで構成されています。クライアント アプリケーションでは、それらのクラスに基づくオブジェクトをインスタンス化できます。コード コンポーネントは、以前のドキュメントでは OLE オートメーション サーバーと呼ばれていました。

ActiveX DLL に対応する .NET プロジェクトは、.NET クラス ライブラリです。Visual Basic アップグレード ウィザードでは、元の ActiveX DLL プロジェクト内のほとんどのコードが自動的にアップグレードされて、対応する種類の Visual Basic .NET プロジェクトが生成されます。自動アップグレードが完了したら、言語の問題を手動で修正する必要があります。サポートされていない言語機能については、第 7 ～ 11 章を参照してください。

Visual Basic .NET クラス ライブラリには、他のプロジェクトと共有できる再利用可能なクラスやコンポーネントが含まれています。この種類のプロジェクトにはウィンドウはないと見なされているため、Windows フォームのクラスは含まれません。Visual Basic 6.0 コード コンポーネント ライブラリを使用してモーダル ダイアログやモードレス ダイアログの標準ライブラリを提供していた場合は、この制約が障害になる可能性があります。ただし、Visual Basic アップグレード ウィザードでは、フォーム オブジェクトがアップグレードされて、コンポーネントの内部にコンパイルされます。これらのフォーム クラスを他のアプリケーションから使用できるようにする必要がある場合は、対応するアクセス レベルを Friend から Public に変更する必要があります。

アプリケーションをさらに改良するには、すべてのビジュアル クラスを別のコンポーネントに移します。これにより、単一のコンポーネントに存在するプレゼンテーション層とビジネス層が分離されて、よりモジュール性の高いアーキテクチャになります。

ActiveX コントロール

ActiveX コントロールは、ユーザー インターフェイス要素を持つ COM コンポーネントです。以前は、OLE コントロールまたは OCX コントロールと呼ばれていました。標準の組み込みコントロールと同じように使用できるため、これらのコントロールを使用して Visual Basic 6.0 のツールボックスを拡張できます。Visual Basic で作成した ActiveX コントロールは、Visual Basic アプリケーション、Microsoft Office ドキュメント、Microsoft Internet Explorer のような Web ブラウザでアクセスする Web ページなど、さまざまなコンテナ アプリケーションで使用できます。

アプリケーションでは、ActiveX コントロールを他の Visual Basic 6.0 コンポーネントと一緒に提供したり、サードパーティの ActiveX コントロールにアクセスして使用したりできます。ActiveX コントロールがアプリケーションのコンポーネントである場合は、Visual Basic アップグレード ウィザードを使用してコントロールのソース コードをアップグレードできます。ActiveX コントロールのソース コードを利用できない場合は、アップグレードした他のコンポーネントに置き換えるか、この章の前半で説明したように、相互運用機能を使用してコントロールにアクセスできます。ここの残りの部分では、ユーザー定義の ActiveX コントロールをアップグレードする方法について説明します。

Visual Basic 6.0 では、ActiveX コントロールは常に、UserControl オブジェクトと、その上に配置される任意のコントロール (*内在コントロール*と呼ばれます) で構成されます。ActiveX コントロールに対応する .NET コンポーネントは、.NET UserControl です。Visual Basic アップグレード ウィザードを使用して ActiveX コントロールをアップグレードすると、アップグレード先のソリューションにクラス ライブラリ プロジェクトが作成されます。このプロジェクトには、元の Visual Basic 6.0 UserControl に相当するクラス宣言を囲む UserControl モジュールが含まれます。この新しいクラスは、System.Windows.Forms.UserControl から継承されます。ContainerControl から継承される System.Windows.Forms.UserControl には、新しいコントロールを作成するための基盤に必要な標準の位置設定コードおよびニーモニック処理コードがすべて含まれています。

ウィザードでは、Visual Basic 6.0 UserControl のさまざまな側面が、次のような形で .NET UserControl に自動的にアップグレードされます。

- **構成ビジュアル要素**。UserControl の基本要素と、対応するすべてのデザイン時プロパティが更新されます。
- **内部ロジック**。コンポーネントの動作やクライアントとのやり取りを制御するコードが更新されます。
- **イベント ハンドラ**。コンポーネント クライアントに含まれているイベント ハンドラ宣言が、新しい .NET UserControl のイベントに合わせて変換されます。UserControl のアップグレードによる変更に合わせて自動的に更新されるのは、元の UserControl と同じアップグレード プロセスでアップグレードされるコンポーネント クライアントのみなので注意してください。他のユーザー アプリケーションがコンパイル済みバージョンの ActiveX コントロールを使用している場合、それらは、ラップされた ActiveX コントロールにアクセスするようにアップグレードされます。この最後のアップグレードを改善するには、ウィザードが同時に処理できるように、必要なコンポーネントをすべて同じアップグレード プロジェクトに含めます。

Visual Basic アップグレード ウィザードは、Visual Basic 6.0 のプロジェクト グループをサポートしていません。ActiveX コントロールが Visual Basic 6.0 のプロジェクト グループに含まれている場合は、次の方法でプロジェクト パッケージをアップグレードする必要があります。

1. アプリケーションを構成するプロジェクトの間の依存関係のリストを入手します。
2. 他のプロジェクトに依存していないプロジェクトを特定し、それらを個別のコンポーネントとしてアップグレードします。
3. 別のプロジェクトに依存している各プロジェクトについて、必要なオブジェクトをすべて依存プロジェクトに複製します。これにより、他のプロジェクト オブジェクトで行われたすべての変換を依存プロジェクトで処理できるようになります。依存プロジェクトをアップグレードします。
4. すべてのプロジェクトをアップグレードしたら、重複しているオブジェクトを削除して、プロジェクト間に正しい依存関係を設定します。元のプロジェクト グループと同じ依存関係になる必要があります。

ユーザー定義の ActiveX コントロールを Visual Basic .NET にアップグレードすると、既定のコントロールと同じようにコントロール ツールボックスに追加できるようになります。ユーザー定義の ActiveX コントロールは、ツールボックスの **[ユーザー コントロール]** タブに表示されます。既定のコントロールは **[全般]** タブに表示されます。アップグレードしたコントロールのインスタンスを .NET Windows フォームに配置したり、.NET フォーム エディタを使用してコントロールのデザイン時プロパティを設定したりできます。

プロパティ ページを使用して、ActiveX コントロールのプロパティを設定するためのカスタム インターフェイスを定義することができます。これにより、**[プロパティ]** ウィンドウを拡張できます。ただし、Visual Basic アップグレード ウィザードでは、この機能はサポートされていません。したがって、プロパティ ページのユーザー インターフェイスを再実装し、プロパティ値とコントロールの状態を同期させて、コントロールのプロパティとダイアログ ボックスのフィールドを関連付ける必要があります。詳細については、第 9 章「Visual Basic 6.0 フォーム機能のアップグレード」を参照してください。

Web ページに埋め込まれた ActiveX コントロール

<Object> HTML タグを使用して、ActiveX コントロールを Web ページに挿入することができます。このタグは、ActiveX コンポーネントの識別子に対応する ClassId パラメータを受け取ります。この識別子は、コンポーネントをクライアント コンピュータに登録して、Web ブラウザで正しくインスタンス化および表示できるようにするために必要です。

アップグレード後の Web アプリケーションで同等の機能を実現するには、埋め込まれているコントロールを Object タグの中に残しておきます。元の ASP アプリケーションにサーバー側コンポーネントが含まれている場合は、相互運用機能を使用してそれらのコンポーネントをラップできます。

埋め込まれているコントロールにプログラムでアクセスすることもできますが、次のような形で、アップグレードするアプリケーションの品質に影響する可能性があります。

- 動的な動作を実現するために別の言語 (VBScript、JScript®、JavaScript、DHTML など) を使用する必要があります。
- アプリケーションの配置に必要なステップや、ダウンロードする必要のあるファイルが増えます。
- コンポーネントで利用できるリソース (アプリケーション サーバーやデータベースのリソースなど) が少なくなります。

ASP.NET は、Web 開発のための別のモデルを提供します。このモデルには、Web 環境専用に作られたコントロールが含まれています。たとえば、次のような種類のコントロールがあります。

- HTML **サーバー コントロール**。プログラミングできるようにサーバーに公開される HTML 要素です。HTML サーバー コントロールは、表示する HTML 要素にきわめて近いマップを持つオブジェクト モデルを公開します。
- Web **サーバー コントロール**。HTML サーバー コントロールより多くの組み込み機能を持つコントロールです。Web サーバー コントロールには、ボタンやテキスト ボックスなどのフォーム型のコントロールと、カレンダーなどの特殊な用途のコントロールがあります。Web サーバー コントロールのオブジェクト モ

デルには、必ずしも HTML 構文が反映されていません。その意味では、HTML サーバー コントロールより抽象的です。

- **検証コントロール**。ユーザーの入力をテストするためのロジックが組み込まれているコントロールです。これらのコントロールを入力コントロールにアタッチすると、ユーザーが入力した内容をテストできます。検証コントロールは、必須フィールドのチェック、特定の値や文字のパターンに対するテスト、値が範囲内に収まっているかどうかの確認などのために使用できます。
- **ユーザー コントロール**。Web フォーム ページとして作成するコントロールです。Web フォーム ユーザー コントロールは、他の Web フォーム ページに挿入できます。これにより、メニューやツール バーなどの再利用可能な要素を簡単に作成できます。

これらのコントロールは、Web ページに埋め込まれた ActiveX コントロールを再実装するための選択肢となります。元の ActiveX コントロールの機能によっては、Web コントロールとして実装する方が便利で適切な場合もあります。たとえば、別のサーバーと通信してデータの取得や処理を行う ActiveX コントロールがアプリケーションにある場合は、一般に、新しい Web サーバー コントロールを作成する方が効率的です。これにより、通信や重い処理をサーバーで実行して、ビジュアル要素を Web クライアントに表示できます。

ASP to ASP.NET Migration Assistant ツールでは、Web ページに埋め込まれた ActiveX コントロールは HTML タグとして扱われ、移行先の ASP.NET ファイルにそのまま保持されます。したがって、Web ページのこれらの要素は自動的に変換されません。必要に応じて後から手動でアップグレードする必要があります。

コンポーネントの再実装は、最初のアップグレード (自動アップグレード) が完了してから行う必要があります。同等の機能が実現された後では、再実装の作業を、アップグレード後のアプリケーションの改良の一環として行うことができます。このプロセスの詳細については、第 17 章「アプリケーションの改良の概要」または MSDN の「Introduction to ASP.NET Server Controls」を参照してください。

ActiveX ドキュメント

ActiveX ドキュメントとは、ドキュメント コンテナ内でホストおよびアクティブ化する必要がある COM コンポーネントです。ActiveX ドキュメントを使用すると、アプリケーションの機能を利用できます。また、アプリケーション固有のデータのコピーを保持したり、配布したりすることもできます。これらのコンポーネントは、HTML ページ上で使用することも、HTML ページの代わりとして使用することもできます。また、ユーザーが ActiveX ドキュメントとアプリケーションや Web サイトのその他のページとの違いを意識することなく、それらの間を移動できるように、コンポーネントを配置できます。

.NET Framework には、ActiveX ドキュメントに相当するコンポーネントの種類はありません。そのため、.NET Framework ベースのコンポーネントを使用して、ActiveX ドキュメントの再設計や再実装を行う必要があります。

Visual Basic .NET にアップグレードする際に考慮する必要がある ActiveX ドキュメントの重要な機能を以下に示します。[3]

- **インターネットによるコンポーネントの自動ダウンロード**。コンポーネントの実行に必要なすべてのコンポーネントをブラウザが見つけてダウンロードできるように、ActiveX ドキュメントへのリンクを作成することができます。
- **オブジェクトのハイパーリンク**。ハイパーリンクに対応したコンテナでは、Visual Basic Hyperlink オブジェクトのプロパティとメソッドを使用して、URL にジャンプしたり、履歴リスト内を移動したりできます。
- **コンテナ ウィンドウとのやり取り**。ActiveX ドキュメントを使用して、コンテナ ウィンドウの追加要素にアクセスできます。たとえば、コンポーネントのメニューをブラウザのメニューに統合することができます。
- **データの格納**。Internet Explorer に ActiveX ドキュメントを配置すると、PropertyBag オブジェクトを通じてデータを格納できます。

ActiveX ドキュメントのソース コードを、アップグレード ウィザードでサポートされている別の種類のモジュールに移す場合は、アップグレード ウィザードを使用して ActiveX ドキュメントの一部をアップグレードできます。たとえば、ActiveX ドキュメントに含まれているソース コードとコントロールを、新しい Visual Basic 6.0 UserControl にコピーできます。UserControl のソース コードの大半が自動的にアップグレードされますが、元の ActiveX ドキュメントのソース コードに含まれているサポートされていない機能は一切アップグレードされません。それらは再実装する必要があります。

ActiveX ドキュメント コンポーネントは、Visual Basic .NET UserControl や XML Web フォームを使用して再実装できます。これらは、さまざまな種類のコンテナでホストすることができます。元の機能の一部は、新しいプラットフォームで再設計して実装しないと失われます。こうした機能のほとんどは、シリアル化インターフェイスや、選択したデザインに応じてサーバーでもクライアントでも実行できる機能など、.NET Framework によって提供されるサービスを使用して実装できます。

分散アプリケーション

分散アプリケーションとは、アプリケーション コンポーネントの一部がリモート コンピュータで実行され、ローカル コンポーネントとリモート コンポーネントの間でやり取りが行われるアプリケーションです。これらのコンポーネントの間で効果的なやり取りが行われるようにするには、通信プロトコル、セキュリティ、リソース ロケータ サービスなど、数多くの基本サービスを使用する必要があります。分散アプリケーションには、優れた拡張性、信頼性、耐障害性など、エンタープライズ環境にとって重要なメリットがあります。

以降では、分散アプリケーションのコンポーネントをアップグレードするときに考慮する必要がある問題について説明します。

DCOM アプリケーション

DCOM (分散コンポーネント オブジェクト モデル) は COM (コンポーネント オブジェクト モデル) の拡張で、別のコンピュータ上のオブジェクトとの通信をサポートします。ローカル エリア ネットワーク (LAN)、ワイド エリ

ア ネットワーク (WAN)、さらにはインターネットを介した通信にまで対応しています。DCOM を使用すると、ユーザーやアプリケーションの必要に応じて、アプリケーションをあらゆる場所に分散できます。

DCOM は COM の拡張であるため、DCOM に移行する際には、COM ベースの既存のアプリケーション、コンポーネント、ツール、および知識を活用することができます。ネットワーク プロトコルの低レベルの詳細は DCOM によって処理されます。

DCOM に相当する .NET 技術とみなされているのはリモート処理です。これは、オブジェクト間の通信を単純化するために作られたフレームワークです。リモート処理では、別のアプリケーション ドメインに存在し、別のコンテキストを持つオブジェクトどうしが、同じコンピュータにあるかどうかや、同じアプリケーション ドメインにあるかどうかさえ関係なく、互いに通信することができます。

リモート処理フレームワークは共通言語ランタイムに組み込まれており、これを使用することによって、高度な分散アプリケーションを構築できます。.NET リモート処理によって提供される機能の一部を以下に示します。[4]

- **プロキシ オブジェクト**。.NET リモート処理では、クライアントがリモート オブジェクトをアクティブ化すると、プロキシ オブジェクトが作成されます。プロキシ オブジェクトは、リモート オブジェクトの代理として機能します。プロキシに対して行われたすべての呼び出しは、正しい リモート オブジェクト インスタンスに転送されます。
- **オブジェクトの受け渡し**。リモート処理フレームワークには、アプリケーション間でのオブジェクトの受け渡しのメカニズムが用意されています。マーシャリングの詳細はこれらのメカニズムによって処理されます。
- **アクティブ化のモデル**。.NET リモート処理では、リモート オブジェクトを複数の方法でアクティブ化できます。
- **ステートレス オブジェクトとステートフル オブジェクト**。.NET リモート処理には、状態の管理やステートレス オブジェクトの管理のためのさまざまな方法が用意されています。
- **チャネルとシリアル化**。.NET リモート処理には、アプリケーションやアプリケーション ドメインの間でメッセージを転送するための転送メカニズムが用意されています。このメカニズムは、.NET チャネル サービスと呼ばれています。.NET によってシリアル化が行われ (オブジェクトをバイト ストリームで転送できます)、フォーマッタを使用してバイト ストリームがエンコード/デコードされます。
- **リースに基づく有効期間**。.NET リモート処理では、*リース*を使用して、クライアントがリモート オブジェクトに接続していられる期間をアプリケーションで制御できます。リースの期限が切れると、オブジェクトの接続が自動的に切断されます。必要な場合にリースを延長する方法も用意されています。
- **IIS でのオブジェクトのホスティング**。.NET リモート処理オブジェクトは、.NET ベースの任意の実行可能ファイルやマネージド サービスでホストできますが、IIS でホストすることもできます。これにより、リモート処理オブジェクトを Web サービスとして公開できます。

分散コンポーネント技術を使用している Visual Basic 6.0 アプリケーションを Visual Basic .NET にアップグレードするには、2 つの方法があります。

- 新しい COM クラスを使用する
- .NET リモート処理を使用する

以降では、それぞれの方法の長所と短所について説明します。

COM を使用して分散アプリケーションをアップグレードする場合

この 1 つ目の方法では、DCOM インフラストラクチャと通信するプロキシとして働く追加の COM クラスを作成する必要があります。マネージ コードでは、COM 相互運用機能を使用してこれらの新しいクラスにアクセスできます。この選択肢には、次のような長所があります。

- **すばやく簡単なアップグレードの実行**。新しい COM ラッパー クラスに必要なのは、DCOM インフラストラクチャと Visual Basic .NET の間の通信リンクを提供することだけです。
- **柔軟なアップグレード パス**。このソリューションは広範なリソースを必要としないため、アプリケーションが複雑な場合は複数の段階に分けて実装できます。
- **コンポーネントの相互作用の早期のテスト**。他のアップグレード作業と並行して、これらのコンポーネントのテストを早い段階でスケジュールできます。

このソリューションは、マネージ コードがアンマネージ コンポーネントやオペレーティング システムのサービスとやり取りする混合ソリューションです。この方法の主な短所を以下に示します。

- **通信の要件の増加**。新しい中間クラスや相互運用ラッパーによって、消費される通信リソースや処理リソースが増加します。
- **新しい技術の採用の制限**。既存技術に依存しているため、.NET Framework が提供する新しい機能やサービスの採用が制限されます。
- **保守性の低下**。新しいクラスによってアプリケーションがより複雑になります。

.NET リモート処理を使用して分散アプリケーションをアップグレードする場合

アップグレードのもう 1 つの選択肢では、アプリケーションを再実装して、.NET リモート サービスを使用します。アプリケーションのロジックは、Visual Basic アップグレード ウィザードを使用してアップグレードできます。その後、.NET リモート サービスを使用して、DCOM を使用するコードを再実装する必要があります。リモート処理を有効にするには、分散オブジェクトが MarshalByRefObject クラスから派生している必要があります。リモート処理は、ServicedComponent から継承される COM+ クラスの組み込み機能です (ServicedComponent はさらに MarshalByRefObject クラスから継承されます)。

MTS と COM+ アプリケーション

単一コンピュータ環境でビジネス ロジック COM コンポーネントを作成および配置できたら、必要に応じてコンポーネントを元のランタイム環境から分離させて、別の管理/実行プラットフォームに配置します。このようにして、アプリケーションで多層アーキテクチャを実現できます。

この種の分離は、拡張性、管理性、説明責任、パフォーマンスなど、さまざまな理由で行われます。マイクロソフトでは、この種類のアーキテクチャを作成するための主要技術として、2 つの技術を提供しています。Microsoft Transaction Server (MTS) と COM+ です。

MTS はトランザクション サービスを提供します。これにより、シングルユーザー アーキテクチャからマルチユーザー アーキテクチャへの移行が促進され、拡張性の高い堅牢なエンタープライズ アプリケーションを構築するためのアプリケーション インフラストラクチャと管理サポートが提供されます。トランザクション管理は多くのアプリケーションにとって不可欠な要素ですが、MTS は、トランザクションを利用しないアプリケーションにも便利なサービスを提供します。

アプリケーションのロジックをカプセル化しているコンポーネントを MTS の管理下で実行し、プレゼンテーション サービスによってさまざまなクライアントから呼び出すことができます。たとえば、次のようなクライアントを利用できます。[5]

- Visual Basic などの COM をサポートする言語で開発された従来のアプリケーション
- Web ブラウザ
- IIS 内で実行される Active Server Pages スクリプト

Visual Basic 6.0 と MTS を使用すると、n 層アプリケーションを構築できます。そのためには、MTS の管理の下に中間層サーバーで実行される COM DLL コンポーネントを開発します。クライアントがこれらの COM DLL を呼び出すと、その要求が Windows によって自動的に MTS に転送されます。

アップグレード作業に関連する MTS のサービスには、このほかに次のようなものがあります。

- コンポーネント トランザクション
- オブジェクト ブローカリング
- リソース プール
- ジャスト インタイム アクティベーション
- 管理

アップグレード方法

アップグレード方法としては、まず、この章の「DCOM アプリケーション」で説明したように、相互運用機能を通じてアクセスできるプロキシ COM クラスを用意する方法があります。ただし、すべての状況でこの方法を使用できるとは限りません。MTS と COM+ のサービスを再実装する必要がある場合もあります。

.NET Framework には、同様の機能を提供するクラスや、COM+ が使用するメカニズムへのインターフェイスを提供するクラスがあります。これらのクラスは、System.EnterpriseServices 名前空間に含まれています。この名前空間は、.NET オブジェクトから COM+ サービスへのアクセスを実現する、エンタープライズ アプリケーションのためのインフラストラクチャを提供します。これにより、エンタープライズ アプリケーションで .NET Framework オブジェクトを使用できるようになります。

MTS と COM+ アプリケーションが使用するコア サービスは、COM+ サービスのタイプ ライブラリ (COMSVCS.DLL) にカプセル化されています。

メモ：ここで取り上げているコンポーネントのアップグレード作業のほとんどは、Visual Basic アップグレード ウィザードの Visual Studio 2005 バージョンでは自動的に行われます。[6]

COM+ Visual Basic 6.0 プロジェクトの全般的な注意点

COM+ アプリケーションの移行先となる .NET アプリケーションを正しく準備および設定できるように、以下の情報を理解しておく必要があります。

System.EnterpriseServices 名前空間には、COM+ とのやり取りに必要なクラスがすべて含まれています。COM+ を有効にするには、移行先の Visual Basic .NET プロジェクトにこの名前空間への参照を含める必要があります。

厳密名ツール (.NET Framework に含まれている Sn.exe) を使用してキー ファイルを生成する必要があります。作成したファイルは、AssemblyKeyFile 属性を含むアセンブリ属性ファイルと共にプロジェクト ファイルに追加します。プロジェクト ファイルは、元の Visual Basic 6.0 プロジェクトと同じ名前を付けてアップグレード出力フォルダに保存します。

.NET サービス インストール ツール (Regsvcs.exe) によるアセンブリの登録を支援するドキュメントやユーティリティを用意して、アップグレード後のコンポーネントをユーザーが配置できるようにする必要があります。その際には、登録ヘルパ API (System.EnterpriseServices.RegistrationHelper.InstallAssembly) を使用できます。

表 4.3 は、MTS と COM+ のコア機能を提供するコンポーネントおよびサービスと、それに対応する .NET Framework の要素を示しています。各コンポーネントのアップグレードの詳細については、第 15 章「MTS アプリケーションと COM+ アプリケーションのアップグレード」を参照してください。

メモ：この表の .NET コンポーネントや名前空間には、それぞれ **System.EnterpriseServices** というプレフィックスを付ける必要があります。たとえば、**CompensatingResourceManager** の完全な名前は、**System.EnterpriseServices.CompensatingResourceManager** です。

表 4.3: MTS と COM+ のコンポーネントに対応する .NET の要素

COM+ コンポーネント/サービス	対応する .NET の要素
Compensating Resource Manager	**CompensatingResourceManager** 名前空間のコンポーネント
オブジェクト プール	**ServicedComponent**
アプリケーション セキュリティ	**SharedPropertyManager**、**SharedPropertyGroup**、**SharedProperty**
オブジェクト コンストラクタ文字列	**ServicedComponent**
トランザクション	トランザクションの機能は、**ServicedComponent** クラスと、**MTSTransactionMode** プロパティの適切なマッピング (**TransactionOption** 列挙値で指定) によって実現されます。

サポートされない機能

COM+ サービスのタイプ ライブラリは、Visual Basic や、COM アプリケーションを構築できるその他の言語で使用されます。このライブラリの中には、Visual C++ 専用に作られていて Visual Basic 6.0 では使用できないメンバも含まれています。たとえば、CServiceConfig クラスはそのようなクラスの 1 つです。こうしたクラスは、.NET Framework にマッピングされていたとしても、Visual Basic 6.0 ではサポートされません。

同様に、Visual Basic 6.0 でサポートされていても、意味的に対応する要素が .NET Framework にないメンバもあります。これらのサポートされないコンポーネントには、セキュリティ、コンテキスト管理、イベント管理、リソース ロケータ、サービス構成、ログ管理などの領域の機能が含まれます。

COM+ のアップグレードおよびサポートされない機能の詳細については、第 15 章「MTS アプリケーションと COM+ アプリケーションのアップグレード」を参照してください。

まとめ

アップグレードする必要があるアプリケーションにどのような種類があるのかを理解することにより、それぞれの種類のアプリケーションをどのようにアップグレードすればよいのかをより深く理解できます。各種類ごとに、アップグレードの容易な機能と、アップグレードの困難な機能があります。潜在的な問題のある箇所を事前に把握しておくことで、それらがアップグレード プロセスに与える影響を軽減できます。

以上でアプリケーションのさまざまなカテゴリについて理解できたので、すべてのアップグレード プロジェクトに適用できる一般的なアップグレード プロセスと手順の学習に入ることができます。次の章では、Visual Basic 6.0 から Visual Basic .NET へのアップグレード プロセスを理解するうえで必要な詳細について説明します。

詳細情報

ASP へのアップグレードの詳細については、MSDN の「Migrating to ASP.NET: Key Considerations」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/aspnetmigrissues.asp) を参照してください。

ASP ページを ASP.NET に自動的に変換する無償ツールについては、MSDN の Microsoft ASP.NET Developer Center の「ASP to ASP.NET Migration Assistant」(http://msdn.microsoft.com/asp.net/migration/aspmig/aspmigasst/default.aspx) を参照してください。

ラッパーの詳細については、MSDN の『.NET Framework Developer's Guide』の「Customizing Standard Wrappers」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconcustomizingstandardwrappers.asp) を参照してください。

アセンブリの使用の詳細については、MSDN の『.NET Framework Developer's Guide』の「Working with Assemblies and the Global Assembly Cache」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconworkingwithassembliesglobalassemblycache.asp) を参照してください。

ASP.NET サーバーコントロールの詳細については、MSDN の「Introduction to ASP.NET Server Controls」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconintroductiontowebformscontrols.asp) を参照してください。

脚注

[1] 『Upgrading Microsoft Visual Basic 6.0 to Visual Basic .NET e-Book』、第 3 章「Upgrading options」、「Selecting projects to upgrade」

[2] 『Upgrading Microsoft Visual Basic 6.0 to Visual Basic .NET e-Book』、第 11 章「Resolving issues with language」、「Making Your Code Thread-Safe」

[3] 『Desktop Applications with Microsoft Visual Basic 6.0, MCSD Training Kit』

[4] 『Upgrading Microsoft Visual Basic 6.0 to Visual Basic .NET e-Book』、第 21 章「Upgrading Distributed Applications」、「Remoting」

[5] 『Web Applications with Microsoft Visual InterDev 6.0, MCSD Training Kit』

[6] 『Visual Basic Upgrade Wizard Whidbey version specification』、「COM+ Services Type Library Migration」

5

Visual Basic のアップグレード プロセス

この章では、Microsoft Visual Basic 6.0 から Visual Basic .NET へのアップグレード プロセスに関する技術的な側面について詳細に説明します。最初の準備作業から最終的なアプリケーションの配置まで、アップグレード プロセスの各ステップについて説明します。Visual Basic アップグレード ウィザードを使用して、アップグレードの大部分を自動化する方法についても説明します。手動で実行する必要のあるタスクや、アップグレードを成功させるための一般的な推奨事項についても紹介します。

手順の概要

アップグレード プロジェクトに含まれる各ステップは、以下の 3 つのグループに分けることができます。

1. アプリケーションの準備
2. アプリケーションのアップグレード
3. アプリケーションのテストとデバッグ

これらのタスク グループを実行する場合、特別な技術と経験が必要になります。各グループに含まれるタスクは、必要なリソースの可用性と特殊性に応じて、ある程度並行して実行することができます。タスク グループでは、リニアな実行順序は定義されません。アプリケーションの異なる部分に対して、複数のタスクを同時に実行できる場合があります。

適用可能な入力および出力を含めた、アップグレード手順の概要を以下に示します。

1. **アプリケーションの準備。** この手順では以下のタスクを実行します。
   a. 開発環境の準備
      入力：ソフトウェア開発およびサード パーティ コンポーネントのインストーラ。
      出力：アップグレードするアプリケーションに対応する、完全な構成のアプリケーション開発環境。
   b. アップグレード ツールの準備

入力：アップグレード ウィザードおよび分析ツールのインストーラ。

出力：元のアプリケーションを実行できる、完全な構成のシステム。

c. アプリケーションのリソース インベントリの取得

入力：元のアプリケーションの仕様およびデザインに関する利用可能な情報。

出力：アップグレードの際に役立つドキュメントのカタログ。

d. コンパイルの確認

入力：適切に構成されたシステム内にある、元の Visual Basic 6.0 アプリケーション。

出力：元のアプリケーションをコンパイル、デバッグ、および実行できるシステム。

e. プロジェクトのアップグレード順序の定義

入力：元のソース コード。

出力：分析ツールに基づいて行われた、アプリケーション コンポーネントの依存関係に関する分析。分析ツールを使用すると、さまざまなコンポーネントのアップグレード順序を計画できます。

f. アップグレード ウィザードのレポートに関する確認

入力：元のアプリケーションのテスト用部分アップグレードに関するアップグレード ウィザードのレポート。

出力：コンピュータのリソースに関する問題の検出とアップグレードに必要な労力の見積もり。

2. **アプリケーションのアップグレード。**この手順では以下のタスクを実行します。

a. Visual Basic アップグレード ウィザードの実行

入力：元の Visual Basic 6.0 ソース コードとアップグレード システム。

出力：Visual Basic .NET で記述された、アップグレード後のアプリケーションの初期コード ベース。このコード ベースには、後で処理する必要があるアップグレードの問題が含まれる可能性があります。

b. アップグレードの進捗状況の確認

アップグレード ウィザードが適切に実行されていることを確認するための制御ステップです。

c. アップグレード ウィザードの実行エラーの修正

アップグレード ウィザードに問題が発生した場合に適用される修正ステップです。

d. 手作業による変更を加えてアップグレードを完了

入力：Visual Basic .NET で記述された、アップグレード後のアプリケーションの初期コード ベース。

出力：コンパイル可能な、アップグレード後の Visual Basic .NET アプリケーション。

3. アップグレード後のアプリケーションのテストとデバッグ。この手順では以下のタスクを実行します。
   a. 元のテスト ケースの実行
      入力:コンパイル可能な、アップグレード後の Visual Basic .NET アプリケーション。
      出力:失敗したテスト ケースの一覧とアプリケーションで検出されたランタイムのバグ。
   b. ランタイム エラーの修正
      入力:コンパイル可能な、アップグレード後の Visual Basic .NET アプリケーション。
      出力:適切に実行できる、アップグレード後の Visual Basic .NET アプリケーション。

この章の残りの部分では、上記の各ステップの詳細について説明します。

アプリケーションの準備

Visual Basic アップグレード ウィザードは、複雑なタスクを自動的に実行して、アップグレード プロセスを容易にするためのツールです。このツールで得られる結果は、元のソース コードに対して適用した準備作業、およびこのアプリケーションが Visual Basic .NET への自動アップグレードにどの程度対応しているかによって影響されます。ここでは、このツールの機能を最大限活用してアップグレードを成功させるために必要な、アプリケーションおよび処理環境の準備に関するガイドラインを提供します。

アプリケーションを適切に準備することで、管理、開発、テストなどさまざまな分野における作業量を大幅に削減できます。アップグレード プロセスの準備は、長期に渡るさまざまな側面が含まれるため、特に重要です。プロセスの初期段階におけるエラーの影響は容易に広がり、作業量の増大に繋がる場合があります。さらに、自動アップグレードが完了した後に問題が見つかると、アプリケーションの一部を再アップグレードする必要があり、アップグレードのその他の段階に遅延を発生させる恐れがあります。

まずは、アップグレードの開発環境の準備から始めることをお勧めします。既存の開発環境は元のソース コードのビルド用に適切に設定されている可能性が高いため、この準備に必要な作業量は比較的少ないものとなります。問題が見つかったとしても、足りないコンポーネントを再インストールすることで解決できる場合がほとんどで、こうしたエラーは問題となるものではありません。アップグレード プロセスを開始する前に、適切な開発環境の準備やコンポーネントの欠落などの問題の修正に時間をかければ、後の段階での時間と労力を大幅に削減することができます。

アップグレードの開発環境の準備が完了したら、ソース コード自体の準備作業に専念できます。この作業には、ソース コードの検査、コンパイルの確認、およびアップグレードに関する一般的な問題の特定が含まれます。

複雑な手順を実行する場合には、アップグレードの各段階における成果物や、コンポーネントの依存関係、アップグレード順序などの重要なステップをアップグレード手順ログにすべて記録しておくと便利です。このログにより、プロセスの内容について明確に理解し、アップグレード プロセスの後の段階で必要となる労力を計

画および測定できるようになります。また、今後のアップグレード プロジェクトの基本としても役立ちます。図 5.1 は、アップグレード手順ログのサンプルを示しています。

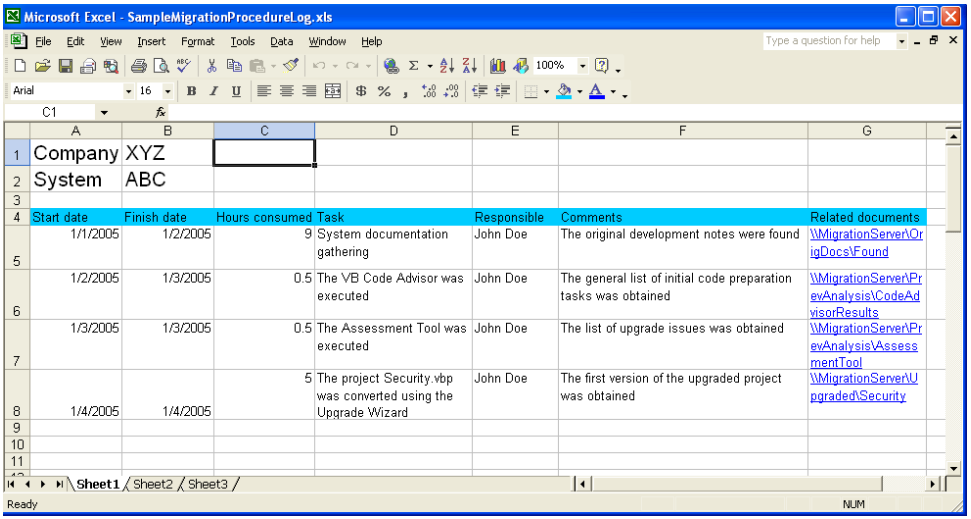

図 5.1

アップグレード手順ログのサンプル

以降では、準備手順の詳細について説明します。

開発環境の準備

アップグレード プロジェクトを開始する前に開発環境を入念に準備しておくことは、アップグレードの成功につながります。元のアプリケーションの作成に使用した開発環境と同じ環境のコンピュータ上でアップグレード プロセスを実行することをお勧めします。これにより元のアプリケーションを簡単に分析でき、初期準備テストを実行できるようになります。

通常の場合、アプリケーション開発に使用されるコンピュータには、アップグレード ウィザードの実行に (必須ではないとしても) 非常に有用な物理リソースが搭載されています。

アップグレード プロセスに影響する環境的な要因には次の 2 種類があります。1 つは、アップグレード プロセスの速度に影響するシステム リソースです。もう 1 つは、アップグレード ウィザードの通常の実行および終了に影響する外部依存関係です。アップグレード プロセスを実行するコンピュータ上に必要なコンポーネントが存在しない場合、例外が発生します。

システム リソース

アップグレード ウィザードはアプリケーションのすべてのグローバル宣言をメモリ内に読み込み、アップグレード プロセスが完了するまで保持します。これは、他のモジュールからアクセスされるメンバの型を解決するために必要となります。ファイルがアップグレードされる際、ファイルはメモリ内に読み込まれ、必要なすべての変換が適用されます。アップグレードが完了すると、結果がディスクに書き込まれ、モジュールのローカル宣言についての情報がアップグレード ウィザードのデータ構造から削除されます。

アップグレード ウィザードは I/O 集約的なアプリケーションではありませんが、言語構造の格納および変換ルールの実行のために大量のメモリおよびプロセッサ リソースを必要とします。20,000 行のコードからなる Visual Basic 6.0 アプリケーションの場合、最大で 160 ～ 200 MB のメモリを消費します。アップグレード プロセスで使用可能なメモリ量は、そのプロセスの速度に大きく影響します。アップグレード ウィザードを適用するコンピュータでは、少なくとも 512 MB のメモリを搭載することをお勧めします。必要なメモリ量はアプリケーションのサイズと複雑さに応じて増加します。100,000 行以上のコードからなる Visual Basic 6.0 アプリケーションをアップグレードする場合、少なくとも 1 GB のメモリが必要です。アプリケーションが数百万行のコードからなる場合には、2 ～ 3 GB のメモリを搭載したコンピュータを使用する必要があります。また、アップグレードするアプリケーションを、より管理しやすい単位に分割すると便利な場合もあります。

CPU については、Intel Pentium 4 クラスまたはそれと同等のプロセッサを推奨します。システムに十分なメモリが搭載されていれば、これらのプロセッサを使用することで、平均的なサイズのアプリケーションを数時間程度で処理できます。

システムのハードドライブには、元のアプリケーションの少なくとも 3 倍のサイズを格納できるだけの空き容量が必要です。これは、すべての一時ファイルと最終的なアップグレード結果を生成するために必要となります。

外部依存関係

標準的な Visual Basic 6.0 アプリケーションでは、サードパーティ コンポーネントが使用されます。プロジェクト設定を介して外部 DLL を明示的に参照するアプリケーションもありますが、インストールされているコンポーネントに基づいてオブジェクトを動的に参照および作成するアプリケーションもあります。

こうしたアプリケーションをアップグレードする場合、これらの外部コンポーネントのインストールおよび可用性には特別な注意を払う必要があります。アップグレード対象のアプリケーションが参照するすべてのサードパーティ コンポーネントは、インストールされ、利用できる必要があります。プロジェクトから明示的に参照されるコンポーネントをシステムで利用できない場合、アップグレード ツールのコマンドライン バージョンである Visual Basic アップグレード ツールによって以下のメッセージが表示され、実行が停止されます。

例外の発生：参照されたコンポーネントを読み込めませんでした：

NameOfTheComponent.ocx (8.0.0)

プロジェクトをアップグレードする前にこのコンポーネントをインストールする必要があります。

アップグレードを実行する前に、Visual Basic 6.0 および参照されるすべてのコンポーネントをインストールして、アプリケーションのコンパイルと実行を確認することをお勧めします。

同様に、ウィザード バージョンのアップグレード ツール (Visual Studio .NET で使用できます) を使用してアプリケーションをアップグレードする場合、アップグレード ウィザードによって以下のメッセージが表示され、実行が停止されます。

アップグレードの失敗：例外の発生：参照されたコンポーネントを読み込めませんでした：NameOfTheComponent.ocx (2.0.0)

プロジェクトをアップグレードする前にこのコンポーネントをインストールする必要があります。

アップグレードを実行する前に、Visual Basic 6.0 および参照されるすべてのコンポーネントをインストールして、アプリケーションのコンパイルと実行を確認することをお勧めします。

さらに、以下の情報がプロジェクトのアップグレード ログの一部として表示されます。

```
<Issue
    Type = "Global Error"
    Number = "4002"
>Could not load referenced component: NameOfTheComponent.ocx (2.0.0) You need to install this
component before you upgrade the project.It is recommended you install VB6.0, with all referenced
components, and ensure the application compiles and runs before upgrading.</Issue>
```

試用版のサードパーティ コンポーネントをインストールしている場合、コンポーネントによってはインスタンス化されるたびにライセンス情報のダイアログ ボックスが表示されることがあります。このダイアログ ボックスが表示されると、アップグレード ウィザードなどのホスト アプリケーションの実行が停止されます。コンポーネントの一部について試用版をインストールしている場合は、前処理段階が開始されるまでの間、アップグレード ウィザードが表示するメッセージに注意を払い、試用版のコンポーネントによって表示されるダイアログ ボックスに応答できるようにすることをお勧めします。前処理段階が開始されたら、アップグレード ウィザードを無人状態で実行していても、アップグレードされたコード ベースが生成されるまでアップグレードを続行できます。

また、アプリケーション準備の一部として、複雑なユーザー インターフェイスを持つサードパーティ コンポーネントのアップグレードに対し、限定的にテストを行うことも重要です。これらのコンポーネントのコア機能を初期化してアクセスするための、サイズの小さなアプリケーションを作成します。これらのアプリケーションの作成が完了したら、アップグレード ウィザードを使用できるようになり、アップグレード ウィザードによって生成されたコードを評価できます。アップグレード後のコンポーネントのデザインおよび実行時の表示と動作には特別な注意を払う必要があります。この準備的なコンポーネントのテスト段階では、コンポーネント間の相互作用、相互運用ラッパー、および Visual Basic .NET コンポーネントが検証されます。相互運用の詳細については、第 14 章の「Visual Basic 6.0 と Visual Basic .NET の相互運用」を参照してください。アプリケーション全体の自動アップグレードに対して悪影響を及ぼす要素を特定し、この要素をアプリケーションから分離すると便利な場合があります。これらの問題については、一部のコンポーネントで使用されている非標準のインターフェ

イスまたは実装が原因であることがほとんどです。アップグレード ウィザードの実行中にこれらのコンポーネントの 1 つがランタイム エラーを起こす場合、考えられる解決策の 1 つはそのコンポーネントを削除してアプリケーションの残りの部分をアップグレードすることです。そのコンポーネントが提供する機能は、後で Visual Basic .NET で再実装できます。

アップグレード ウィザードの準備

アップグレード プロセスの準備には、元のアプリケーションに関するさまざまな側面を考慮した以下の 3 つの範囲が含まれます。

- **アップグレード ウィザードとアプリケーションの外部環境の準備。**この作業範囲には、開発およびアップグレードに必要なすべてのツールのセットアップが含まれます。詳細については、この章の前半の「開発環境の準備」および「サードパーティ コンポーネント」を参照してください。
- **アプリケーションの評価。**この準備範囲では、アプリケーションの構成、およびアップグレード プロセスの適用時に発生する可能性のある問題について評価します。実際のアップグレードに先立って、労力およびコストの見積もりが生成されます。この準備範囲は、Visual Basic 6.0 Assessment Tool などのさまざまなツールによって支援されます。評価および労力の見積もりの詳細については、第 3 章「評価および分析」を参照してください。
- **元のアプリケーションの調整。**この準備範囲では、アップグレード プロセスを容易にするために元のアプリケーションを調整します。これらの変更によりアップグレード ウィザードによって実行される自動作業が増加し、自動プロセスの終了後に手作業で完了させるタスクが減少します。変更は、ドキュメントやアドバイザ ツールなどのさまざまなソースから得られる初期評価および一般的なアドバイスを基に行います。アプリケーション調整の詳細については、この後の「Visual Basic 6.0 Assessment Tool」および「未使用コンポーネントの削除」を参照してください。

アプリケーションの内部面に対する準備は必須です。これはアップデート後のアプリケーションの品質と、自動アップグレード後に手動で行う必要のある作業量に直接影響します。アプリケーションの内部面の準備に使用されるソフトウェア ツールは、アップグレードするアプリケーションのサイズや複雑さを評価したり、開発者が必要な変更を加える場合に非常に役立ちます。これらのデータは、アップグレードに必要な労力を正確に見積もるための基礎になります。

ここで紹介する各ツールは、コードの評価、労力の見積もり、および元のコードの調整に使用できるさまざまな情報を提供します。これらのツールは、アプリケーションのさまざまなモジュールを解析することにより、アプリケーションの情報をソース コードから直接抽出します。各ツールはそれぞれ異なった分析の視点で、アプリ

ケーションおよびそのアップグレード プロセスに関する補足的な情報を提供します。ここでは以下の各ツールを紹介します。

- Visual Basic 6.0 Assessment Tool は、アプリケーションに関する広範なインベントリを生成します。アップグレードが困難な機能に特に焦点が当てられます。
- Visual Basic 6.0 Code Advisor は、Visual Basic .NET にスムーズに移行するために修正可能な機能を元のアプリケーション内で特定します。

アップグレードの準備に使用されるツールはコードを徹底的に調査しますが、このプロセスに必要な時間はアップグレード プロセス全体としてはわずかなものです。主な理由としては、これらのツールはコードの一部を分析して特定のパターンを識別しますが、変換は行わないためです。これらの準備ツールは、アップグレード プロセスの適用後に得られる結果をすばやく予測および改善するための方法の 1 つとして用意されています。

Visual Basic 6.0 Assessment Tool

Visual Basic 6.0 Assessment Tool は、Visual Basic 6.0 のソースコードを分析し、アプリケーションに関する広範な情報を生成します。このツールが収集する情報の詳細については、第 3 章「評価および分析」を参照してください。

このツールは、複数のプロジェクト ファイルを同時に分析できます。そのため、ユーザーはファイルを個別に選択したり、グループごとに選択したりできます。また、ディレクトリおよびそのサブディレクトリにあるすべての .vbp ファイルを検索して選択することもできます。このツールには重み付けファイルが用意されています。この重み付けファイルを使用すると、複雑さを表す値をアップグレードの各問題に割り当てて、カスタマイズ可能な労力見積もりデータを生成することができます。

評価ツールは、アップグレードの労力の見積もりに使用できるアプリケーション統計やアップグレードの問題を含む、さまざまなレポートを生成します。手作業での調整を要するほとんど (すべてではありません) の問題が特定されます。収集される統計情報には、コントロールの使用、クラス メンバへのアクセス、およびライブラリの存在に関するデータが含まれます。

作業量の見積もりに役立つその他の情報ソースとして、Upgrade Issues テーブルがあります。このレポートには、アップグレードが困難だとされた機能の使用に関するデータが含まれます。このテーブルに含まれる各問題には複雑さを表すレベルが割り当てられ、アップグレードの難しさと手作業での作業量を示します。評価ツールによって検出されてこのテーブルに追加される問題には、COM+ 機能、Visual Basic 6.0 のゲームのパフォーマンス機能、および ActiveX ドキュメントがあります。

評価ツールの詳細については、第 3 章「評価および分析」を参照してください。

Visual Basic 6.0 Code Advisor

Visual Basic 6.0 Code Advisor は Visual Basic 6.0 のアドインであり、アプリケーションのソース コードの確認に使用されます。開発者は、このツールを使用して、アプリケーションが所定のコーディング標準を満たしているかどうか確認できます。このコーディング標準は、堅牢でメンテナンスしやすいコードを作成するためにマイクロソフトによって開発されたベスト プラクティスに基づいています。

また、このツールによって特定された問題は、アプリケーションを Visual Basic .NET にアップグレードする開発者への警告も表しています。元のソース コードでこれらの問題を修正することにより、移行作業をスムーズに行うことができます。コード アドバイザ ツールはアップグレードに関するすべての問題を検出するわけではありませんが、最も一般的な問題を検出し、アップグレード プロセスを高速化するようにデザインされています。正規表現を使用すると、このツールが検出する問題の範囲を拡張できます。

以下のリストは、このツールによって検出された所定のルールのサンプルを示しています。

- **バリアント オブジェクトの遅延バインド。** Object 型として宣言された変数に代入されるオブジェクトは、遅延バインドされます。この型のオブジェクトは任意のオブジェクトに対する参照を保持できます。Object 型の変数をアップグレードする場合、Visual Basic アップグレード ウィザードが変数の特定の型を識別できず、対応するクラス メンバに対するアクセスがアップグレードされません。この問題は、すべての変数宣言で特定の型を指定することによって解決できます。Option Strict On ディレクティブを使用すると、変数の型の指定が強制されます。Visual Basic 6.0 Code Advisor は宣言されていない変数を検出し、Option Strict On を使用するように開発者に求めます。
- **Option Explicit の欠落。** このオプションを宣言しない場合、開発者は宣言していない変数をソース コードで使用できます。未宣言の変数は既定で Variant 型になります。この型の変数には任意の型の値を格納できます。このような変数を使用すると、前の項目で説明した問題と同じ問題がアップグレード時に発生します。Variant として宣言されたオブジェクトの既定のプロパティをコードで使用している場合、アップグレード ウィザードは使用する正しいプロパティを決定できません。
- **Form または Control のソフト バインド。** Form または Control として宣言された変数は、アップグレード時に問題を発生する場合があります。Visual Basic 6.0 では、特定のフォームやコントロールに定義するプロパティとメソッドにこれらの汎用クラスを使用できます。Visual Basic .NET では、Form 変数と Control 変数はサポートされますが、特定の型のフォームやコントロールに固有なプロパティには、アクセスできません。
- **ActiveForm と ActiveControl を使用したソフト バインド。** このパターンを使用すると、前の項目で説明した問題と同じ問題がアップグレード時に発生します。
- **Variant を返す文字列関数。** Visual Basic 6.0 には 2 つのバージョンの文字列関数のグループが用意されています。1 つは Variant を返し、もう 1 つは String を返します (Left や Left$ など)。Variants は Null 値を格納できます。文字列を要求する変数に Null 値を代入すると、デバッグの困難なランタ

イム エラーが発生する場合があります。Visual Basic .NET では、String バージョンの文字列関数のみがサポートされます。

- **OLE コントロールはアップグレードされません。** Visual Basic .NET には、OLE Container コントロールに対応するコントロールがありません。OLE Container コントロールを含むフォームをアップグレードしようとしても、OLE Container コントロールのコードはアップグレードされません。代わりに、OLE Container コントロールが Label コントロールに置き換えられます。これは、OLE Container コントロールを自動的にアップグレードできないことを強調して示しています。
- **Visual Basic .NET には Line コントロールがありません。** Visual Basic .NET には、Line コントロールに直接対応するコントロールがありません。水平または垂直 Line コントロールを含むフォームをアップグレードしようとすると、これらのコントロールが Label コントロールに置き換えられます。水平線や垂直線以外の線も一切アップグレードされません。Visual Basic .NET のグラフィックス関数呼び出しを使用して手作業で置き換える必要があります。
- **プロパティ/メソッド/イベントはアップグレードされません。** Visual Basic .NET には、Visual Basic 6.0 の参照プロパティ、参照メソッド、および参照イベントに直接対応するものがありません。アプリケーションをアップグレードすると、該当するコード行がターゲット コードにそのまま修正されずにコピーされます。これにより、Visual Basic .NET でコンパイル エラーが発生します。

Visual Basic 6.0 Code Advisor の詳細については、MSDN の Microsoft Visual Basic Developer Center に掲載されている「Visual Basic 6.0 Code Advisor」を参照してください。

未使用コンポーネントの削除

元のアプリケーションのライフタイム期間に変更が何度か行われ、各コンポーネント間の依存関係が変更されている可能性があります。また、コンポーネントのコードの一部がアプリケーションの他の部分で使用されなくなった可能性もあります。

アプリケーションの明瞭性と管理性を確保するため、アプリケーションで現在使用されないコンポーネントやコードの一部は削除する必要があります。また、アップグレードの観点から考えると、アップグレード プロセスを開始する前に不要なアプリケーション要素を削除することも重要です。これにより、自動アップグレードが完了した後に発生する、アップグレードの問題が減少すると共に、アップグレードする必要のあるコードの量が削減されます。

評価ツールを使用すると、未使用のコンポーネントを特定できます。ファイルの依存関係グラフにより、アプリケーション ファイル間の関係がレポートされます。このレポートを確認すると、使用されていないファイルを特定できます。サードパーティ コンポーネントや外部 DLL 関数の参照についても、それ以外に削除可能な未使用の要素がないか調べる必要があります。

アプリケーションのリソース インベントリの取得

アプリケーションをある言語から別の言語にアップグレードする場合、複雑さや要件の異なるレベルで多くのタスクを実行する必要があります。異なる環境でホストされた別の言語にプログラミング言語や開発環境を移

行するタスクは、アップグレードにおいて重要な作業となります。ただし、新しいリソースを十分に活用し、アプリケーションを新しいプラットフォームで新しいライフ サイクルに沿って改良していくには、それ以外にいくつかのタスクを実行する必要があります。

アップグレードに必要な労力を見積もる場合、評価ツールによって作成された初期ソース コード インベントリが不可欠です。以前のバージョンのアプリケーションをアップグレードする場合は、ドキュメント、設計図、テスト ケース、仕様書などのリソース インベントリを作成することも重要です。これらのリソースは、アップグレード手順の最終段階で役立ちます。元のテスト ケースは、アップグレード後のアプリケーションをテストする際の基盤となります。ターゲットのアーキテクチャを決定するには、設計図やマニュアルに関する評価が重要になります。この情報は、アプリケーションを高いレベルで見通してアップグレード時に特別の注意が必要な部分や機能を特定するためにも役立ちます。また、アップグレード後のアプリケーションを修正する際には、データベースや全般的なデザイン ドキュメントも役立ちます。たとえば Variant 変数を使用してデータベース フィールドにアクセスしている場合、フィールドのデータ型情報を利用することで、より具体的な型を指定して変数を再宣言できます。

コンパイルの確認

アップグレード ウィザードは構文的に正しいコードを必要とします。構文が正しいかどうかは、アップグレード ウィザードを実行する前に元のアプリケーションをコンパイルすることにより確認できます。元のアプリケーションで発生したコンパイル エラーを修正し、アップグレード ウィザードに適切な入力を行う必要があります。また、元のアプリケーションのコンパイルは、アプリケーションによって参照されるすべてのコンポーネントおよびユーザー定義のソース コードがシステムで使用可能かどうかを確認する際にも役立ちます。プロジェクト グループの場合は、グループ全体および個々のプロジェクトを正しくコンパイルできるかどうか確かめることが重要です。プロジェクト グループのアップグレードについては、次で詳しく説明します。

使用できないコンポーネントを参照しているプロジェクトを開こうとすると、以下のエラー メッセージが表示されます。

> 読み込み時エラー。詳細については 'C:\ReferenceSample\Form1.log' を参照してください。

指定されたファイルを開くと、このエラー メッセージの原因がわかります。サンプルのエラー行を以下に示します。

> 行 13：コントロール TreeView1 のクラス MSComctlLib.TreeView は、読み込まれたコントロール クラスではありません。

このコンパイル エラーを修正して、アップグレード ウィザードでこのアプリケーションをアップグレードできるようにするには、エラーの説明で示されたコンポーネントをインストールする必要があります。この例では、MSComctlLib.TreeView コンポーネントのインストールが必要です。

また、アップグレードを実行するコンピュータ上で元のアプリケーションをテスト実行することもお勧めします。テスト実行により、すべてのコンポーネントが使用可能であることと、アプリケーションが予想通りの動作結果となるかを確認します。元のアプリケーションに存在するすべてのランタイム エラーがターゲット アプリケーショ

ンにもそのまま残る可能性を考慮することは重要です。元のアプリケーションに含まれるエラーについて認識しておくと、ターゲット アプリケーションのテストおよび改良を行う際の基盤となります。詳細については、この後の「アップグレード後のアプリケーションのテストとデバッグ」を参照してください。

プロジェクトのアップグレード順序の定義

Visual Basic 6.0 アプリケーションに含まれるファイルのアップグレード順序は、対応するアプリケーション プロジェクト ファイルで指定されたファイル順序によって決定されます。この順序は、ターゲット アプリケーションのディレクトリで、どの結果ファイルが最初に書き込まれるかも決定します。この情報は、特定のアプリケーションのアップグレードで発生する問題をテストして切り分ける場合に役立ちます。Visual Basic 6.0 アプリケーションは、個別にアクセスおよび管理される複数のプロジェクトから構成できます。また、複数のプロジェクトを 1 つのプロジェクト グループ ファイルで構成することもできます。プロジェクト グループ ファイルを使用してビルドされたアプリケーションのアップグレードの詳細については、この後の「プロジェクト グループのアップグレード」を参照してください。

プロジェクトには、アプリケーションの異なる階層に対応するソース コード ファイルを含めることができます。また、これらの階層を複数のプロジェクトに分散させることもできます。アップグレードを行うには、アプリケーション階層とプロジェクトの構成要素の対応関係について明確に理解することが重要です。プロジェクトとコンポーネントのアップグレード順序は、コンポーネントとアプリケーション階層の関係、およびコンポーネントの依存関係によって決定されます。処理順序を決定する際にもう 1 つ考慮する点は、アップグレード方法の選択です。第 1 章「はじめに」および第 4 章「一般的なアプリケーションの種類」で説明した条件に基づいて、垂直アップグレード方法または水平アップグレード方法のいずれかを選択できます。

アプリケーションをアップグレードする際に推奨される初期アプローチは、アプリケーションのコア コンポーネントを最初に処理することです。コア コンポーネントは、アプリケーションの他の要素にサービスや機能を提供します。もう 1 つのアプローチは、これらの基本コンポーネントをアップグレードの最終段階まで処理せずに残しておくことです。アップグレードの中間結果をテストする場合には、その他のコンポーネントは相互運用のテクニックを使用してこれらの基本コンポーネントにアクセスできます。

すべての依存関係の特定

あるアプリケーション コンポーネントが別のコンポーネントを使用する場合、最初のコンポーネントは 2 番目のコンポーネントに依存する、と言います。依存関係にはさまざまな用途があり、それに応じて依存関係の特定方法が異なります。依存関係は、クラス メンバへのアクセス、メソッドの実行、特定の型の変数宣言、埋め込みコントロールの追加などで使用されます。また、プロジェクト レベルでの依存関係もあり、これらはプロジェクト参照を通じて設定されます。依存関係は推移的です。つまり、コンポーネント A がコンポーネント B に依存し、コンポーネント B がコンポーネント C に依存する場合、コンポーネント A もコンポーネント C に依存するこ

とになります。このとき、B と C の間に直接的な依存関係があり、A と C の間に間接的な依存関係があるため、B は C に直接的に依存し、A は C に間接的に依存していると言います。

先に述べたように、依存関係は、あるコンポーネントが別のコンポーネントを参照またはアクセスする方法に基づいて特定されます。アプリケーションに含まれるコンポーネントが少数の場合、依存関係は容易に特定できます。しかし、コンポーネントの数とその相互関係が増加するにつれて、コンポーネントの依存関係を完全に把握することは、より難しくなります。アップグレードを成功させるには、プロジェクト レベルの依存関係を考慮するだけで十分な場合がほとんどですが、複雑でアップグレードの困難なコンポーネントがプロジェクト内に存在する場合は、これらのコンポーネントについても依存関係を考慮する必要があります。

アップグレードを実行するエンジニアは、コンポーネント間およびプロジェクト間の依存関係についての知識が必須です。この情報は、戦略的優先度、ビジネス価値、使用可能なリソースなどその他の重要な考慮事項と共に、アップグレード作業に集中して最大の結果を得るために使用されます。

評価ツールはファイル依存関係グラフを生成します。このグラフは、依存関係を持たないコンポーネントの特定に役立ちます。このレポートを図 5.2 に示します。

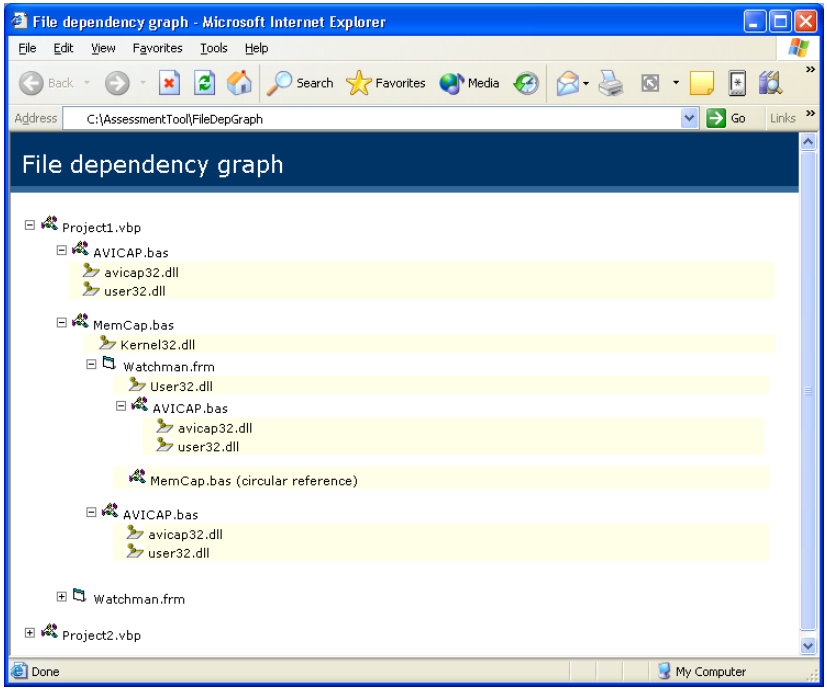

図 5.2

ファイル依存関係グラフのサンプル

このレポートを使用すると、独立したコンポーネントを特定できます。この例では、レポートの最初のプロジェクトである Project1.vbp に、AVICAP.bas、MemCap.bas、および Watchman.frm の 3 つのファイルが含まれています。レポートでは、これらのファイルそれぞれに異なる依存関係が特定されています。Watchman.frm と

AVICAP.bas は MemCap.bas という名前のフォルダに格納されています。これは、MemCap.bas が他のファイルに依存していることを示します。独立したユーザー定義ファイルは、サードパーティ コンポーネントや、user32.dll などのシステム コンポーネントにのみ依存します。このレポート例では、他のユーザー定義ファイルに依存していない AVICAP.bas は独立したコンポーネントです。

コンポーネントのアップグレードについては、独立したコンポーネントを Visual Basic .NET にアップグレードするか、相互運用を使用してこれらのコンポーネントにアクセスするかを最初に決定します。この決定は、第 2 章「アップグレードの成功のためのプラクティス」で説明した費用便益分析に基づいて行う必要があります。コンポーネントのアップグレードが難しくなく、新しい .NET 機能から、より優れたターゲット アーキテクチャを生成するためにメリットが得られるのであれば、コンポーネントをアップグレードすることをお勧めします。しかし、コンポーネントが複雑な場合や、アップグレードから得られる利益がコストに見合わない場合は、これらのコンポーネントを Visual Basic 6.0 のまま残し、相互運用のテクニックを使用して新しいアプリケーションからアクセスする方法が最適なこともあります。

独立したコンポーネントのアップグレードが完了したら、これらのコンポーネントに直接依存するコンポーネントについても同様の方法でアップグレードする必要があります。アプリケーション全体の処理が完了するまで、各レベルの依存コンポーネントに対してこの手順を繰り返します。この方法はアプリケーションのアップグレード処理順序に関する高レベルな考え方の 1 つであり、テスト段階、処理効率、リスクなど (これらについては後で説明します) を考慮してさらに改良することができます。

依存関係が体系化されている場合は、アプリケーションの漸進的なテスト (アップグレードするコンポーネントを徐々に増やしていく) が容易になります。依存関係グラフに従って、異なるテスト マイルストーンをアップグレード計画に定義できます。

アップグレード計画とコンポーネントのアップグレード順序を決定する際には、それ以外に考慮が必要な点があります。それは、アップグレードするアプリケーションの部分と、各コンポーネントに適用するテストとのバランスを取ることです。

可能な方法の 1 つは、複数のアプリケーション コンポーネントからなる 1 グループをアップグレードし、それに対応するアプリケーション機能の単体テストをアップグレードの最後に行うことです。アプリケーション全体のアップグレードが完了したら、システムのテストを開始できます。このアプローチの利点の 1 つは、アップグレードの並列処理力が向上することです。ほとんど調整を行わずに複数のコンポーネントを同時にアップグレードできます。大きな欠点は、テスト段階における問題の切り分けが困難になることです。アップグレードするアプリケーションの複雑さが中程度以下の場合は、この方法をお勧めします。アップグレードの複雑さは、言語の特性とアップグレードするアプリケーションのサイズによって決まります。詳細については、第 3 章「評価および分析」を参照してください。

もう 1 つ考えられる方法は、依存関係グラフの順序に厳格に従ってコンポーネントをアップグレードすることです。これは、各コンポーネントをアップグレードした後、アプリケーション機能を漸進的にテストすることを意味します。この方法の利点の 1 つは、問題を容易に切り分けられることです。ただし、重要な欠点は、プロジェクトの実行時に計画および調整を追加で行う必要があることです。さらに、このアプローチでは並列処理力が低下する可能性があります。複雑なアプリケーションをアップグレードする場合は、この方法をお勧めします。

この章の「アプリケーションの準備」の序文で説明したように、アップグレード プロジェクトのログに重要な決定事項をすべて記録することが大切です。このログには、プロジェクトのアップグレード順序と、順序の決定に使用した情報を記録する必要があります。

アップグレード ウィザードのレポートに関する確認

準備段階の一部として、テスト用にアプリケーションの部分的なアップグレードを実行し、以下の各項目について決定または検出する必要があります。

- **アップグレード プロセスの速度。** アプリケーションの部分的なアップグレードにかかった時間を測定することで、アプリケーション全体のアップグレードに必要な時間を把握できます。この情報は、アップグレード手順計画の作成に役立ちます。
- **システム リソース。** アプリケーションの一部を使用することにより、利用可能なシステム リソース、特にシステム メモリの量を評価できます。システム リソースの使用量は、アプリケーションの複雑さに影響されます。極端なメモリ スワップ (*スラッシング*と呼ばれる) など、システム リソースの過剰使用の兆候が示される場合は、リソースを追加する必要があります。
- **セットアップに関する問題。** テスト用の部分アップグレードでは、使用できないサードパーティ コンポーネントや古いソース コードなど、セットアップに関する一般的な問題を検出できます。
- **その他の一般的なエラー。** アプリケーションの完全アップグレードを実行する前に、ソース コードに関する問題を特定して修正できます。

テスト アップグレード時に生成されるアップグレード ウィザードのレポートを検証すると、自動処理の後に手動で実行する必要のある作業量を把握できます。アップグレードの計画や必要なリソースの準備に関するビジョンも提供されます。また、第 3 章「評価および分析」で説明したように、このレポートをフィルタ処理して分析することもできます。

アップグレード ウィザードでは、プロジェクトのアップグレード中に見つかったエラー、警告、および問題を含むレポートが生成されます。プロジェクトをアップグレードするとアップグレード ウィザードによりほとんどのコードが Visual Basic .NET にアップグレードされますが、一部の要素は自動的に変換されません。これらの言語要素については、自動処理が完了した後に手動で修正する必要があります。以下のコード例は、アップグレード ウィザードを使用してアップグレードしたときに警告が発生する、最も一般的なシナリオの 1 つを示しています。

```
Sub SetObjLabel(obj As Variant)
    obj.Caption = "Initial text"
End Sub
Private Sub Form_Load()
    SetObjLabel Label1
End Sub
```

このコード例では、いくつかの初期値を設定するサブルーチンに Form のコントロールの 1 つが送られます。適用すると、アップグレード ウィザードによって以下のコードが生成されます。

```
Sub SetObjLabel(ByRef obj As Object)
    ' UPGRADE_WARNING: オブジェクト obj.Caption の既定プロパティを解決できませんでした。
    obj.Caption = "Initial text"
End Sub
Private Sub Form1_Load(ByVal eventSender As System.Object, _
            ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    SetObjLabel(Label1)
End Sub
```

このコード例が示すように、ほとんどのコードが自動的に Visual Basic .NET にアップグレードされます。ただし、太字の行は、アップグレード ウィザードが検出した、アップグレードに関する問題を表しています。コメントにはどのような問題であるかが示され、コード行にはアップグレード ウィザードが自動的にアップグレードできないコードが表示されます。このコードは、元のコードベースからアップグレード後のコードにそのままコピーされます。

この問題の原因は、Label オブジェクトに対するオブジェクト変数の遅延バインドにあります。この章の前半で述べたように、アップグレード ウィザードでは遅延バインドを解決できず、Caption プロパティの参照時にアクセスされるオブジェクトの特定の型を決定できません。その結果、アップグレード ウィザードはこのプロパティに対応する式を見つけることができません。このコードをビルドして実行すると、実行時例外が発生します。ほとんどの場合、影響を受ける変数の型をより具体的に指定し、アクセス先のメンバを .NET の対応メンバに置き換えることでこの問題を解決できます。これについての Visual Basic .NET のコード例を以下に示します。

```
Sub SetObjLabel(ByRef obj as System.Windows.Forms.Label)
    obj.Text = "Initial text"
End Sub
```

修正を終えたら、UPGRADE_ISSUE コメントを削除できることに注意してください。

サブルーチン SetObjLabel には、コードの修正に伴い意図的にではなく変更された特別な用途があります。元のアプリケーションの SetObjLabel は、Caption プロパティを持つ任意の種類のコントロールを受け取ることができます。新しいバージョンでは、サブルーチンは Label コントロールのみを受け取ることができます。これは見落とされやすい重要な考慮事項です。元のコードの動作を保持したまま修正を行うには、パラメータの型を System.Windows.Forms.Control に変更します。これは、すべての.NET コントロールの基本となる型で、

Text プロパティを持っています。これにより、任意のコントロールを受け取る機能を復元できます。以下に例を示します。

```
Sub SetObjLabel(ByRef obj as System.Windows.Forms.Control)
    obj.Text = "Initial text"
End Sub
```

メモ : ArtinSoft の Visual Basic Upgrade Wizard Companion には、未宣言の変数の型を自動検出するための高度なメカニズムが用意されています。このメカニズムでは、Visual Basic 6.0 に含まれる大部分の言語要素の使用パターン、および最も可能性の高い関連シンボルのデータ型についてこれらの要素によって提供される情報を基に、ソース コードに対する静的な分析が行われます。また、変数、パラメータ、およびその他遅延バインドされる言語要素のデータ型について詳細な結果を得るために、シンボル間の関係も分析されます。Upgrade Wizard Companion を使用すると、上記のコード例が以下のコードに自動的にアップグレードされます。

```
Sub SetObjLabel(ByRef obj as System.Windows.Forms.Label)
    obj.Text = "Initial text"
End Sub
```

SetObjLabel の追加呼び出しに応じて、オブジェクトのパラメータが **System.Windows.Forms.Control** 型として、またはそれ以外のコントロール型として宣言されます。
Upgrade Wizard Companion およびその他のアップグレード サービスの詳細については、ArtinSoft の Web サイトを参照してください。

以下のリストは、アップグレードに関する最も一般的な問題と、自動アップグレードを実行した後にこれらの問題を解決する方法を示しています。

- **プロパティ <object>.<property> はアップグレードされませんでした。**プロパティを自動的にアップグレードできない場合、この警告が発生します。この問題を修正するには、対処方法を用意して元の機能を置き換える必要があります。アプリケーションの特定の機能の再実装が必要になる場合もあります。
- **<objecttype> オブジェクト <object> はアップグレードされませんでした。**サポートされていないクラスのオブジェクトが宣言および使用されている場合、この警告が発生します。この問題の修正には、追加の開発作業が必要になる場合があります。詳細については、第 7 ～ 11 章を参照してください。
- Use of Null/IsNull detected Null is not supported in Visual Basic .NET. Null は System.DBNull.Value にアップグレードされ、IsNull は IsDBNull に変更されます。この値および関数は、Visual Basic 6.0 のそれらとは動作が異なります。また、Visual Basic .NET では、Null の伝播がサポートされなくなりました。アプリケーション ロジックを見直して、Null の使用方法を変更する必要があります。

- **<functionname> には新しい動作が含まれます。** Visual Basic .NET と Visual Basic 6.0 では、動作の異なる関数があります。したがって、コードを詳細に調べて、アプリケーション ロジックが変更されていないことを確認する必要があります。
- **<object> イベント <variable>.<event> はアップグレードされませんでした。** 一部のイベントは、Visual Basic .NET にアップグレードできません。必要な場合は、対応するメソッドを呼び出してください。
- Could not resolve default property of object '<objectname>'. これは発生する問題の中でも最も一般的なものの 1 つです。値に遅延バインド変数が設定されている場合に発生します。これは、前のコード例で説明した警告です。
- **参照されたコンポーネント <reference> を読み込めませんでした。** このメッセージは、アップグレードするアプリケーションのセットアップに関連しています。ActiveX コントロールやクラス ライブラリなどの参照が見つからない場合に発生します。この問題を修正するには、必要なすべてのコンポーネントをコンピュータにインストールする必要があります。アップグレードを実行する前に、Visual Basic 6.0 および参照されるすべてのコンポーネントをインストールしてアプリケーションのコンパイルと実行を確認することにより、この問題を回避できます。

エラー、警告、および問題の詳細については、MSDN の「Visual Basic 6.0 Upgrading Reference」の左ペインを参照してください。

Visual Basic 2005 の場合 :

Visual Studio 2005 に含まれるアップグレード ウィザードは、以前のバージョンから大幅に変更されています。変更点の中には、Visual Basic 6.0 の機能サポートの追加および新しいアップグレード メッセージに対するサポートの追加が含まれています。アップグレード ウィザードの変更の結果として、(より多くの機能がサポートされるようになったため) エラー、警告、および問題 (EWI) の数が減少し、特定された EWI を解決するために必要な労力の配分がやり直されます。

アプリケーションのアップグレード

Visual Basic アップグレード ウィザードは、アプリケーションをアップグレードするための主ツールです。ここでは、使用可能な実行オプションと、ウィザードを最大活用する方法について説明します。

アプリケーションの自動アップグレードは、アップグレード プロセス全体では中間ステージに相当します。アップグレードを実行する前に、必要なコンポーネントの準備とセットアップを行う必要があります。通常の場合、同等の機能を持つコンパイル可能なアプリケーションを得るためには、自動アップグレード後に手作業による若干の修正が必要になります。その後で、ターゲット アプリケーションの機能拡張を開始することができます。

アップグレード ウィザードにはさまざまなバージョンがあり、アップグレード結果をさらに自動化するための機能をそれぞれ備えています。本書で説明するアップグレード機能の大部分は、Visual Studio .NET (Visual Studio .NET 2003 とも呼ばれる) バージョンのアップグレード ウィザードに適用されます。Visual Studio 2005 バージョンのアップグレード ウィザードでは、ActiveX コントロールに対する追加サポートや COM+ 機能の

アップグレードに対するサポートの改良など追加のアップグレード機能が用意されると共に、パフォーマンスおよび信頼性が向上しています。ArtinSoft Visual Basic Upgrade Wizard Companion は、コンテキストを意識した列挙のアップグレード、コード改良機能、Variant 変数、Object 変数の型の自動検出などの機能をサポートする、追加のアップグレード メカニズムです。このアップグレード ツールと、その他のアップグレード ツールおよびサービスの詳細については、ArtinSoft の Web サイトを参照してください。

ここでは、Visual Studio .NET アップグレード ウィザードを中心に解説します。ただし、適用できる場合には、その他のバージョンのアップグレード ウィザードについても取り上げます。アップグレード ウィザードおよび自動アップグレード プロセスの詳細については、第 6 章「Visual Basic アップグレード ウィザードについて」を参照してください。

Visual Basic アップグレード ウィザードの実行

アップグレード ウィザードは、コマンド ライン ツール、または Visual Studio .NET IDE から利用できるウィザードのいずれかの方法で使用できます。Visual Studio .NET で Visual Basic 6.0 のプロジェクト ファイルを開こうとすると、ウィザード バージョンが自動的に起動します。ここでは、これら両方のアプローチについて説明します。

アップグレードを実行する場合、アップグレードを実行するコンピュータにVisual Basic 6.0 および Visual Basic .NET をインストールすることをお勧めします。これにより、元のアプリケーションとターゲット アプリケーションをテストできるようになり、また、アップグレード ウィザードで必要となる基本コンポーネントがインストールされます。

わかりやすくするために、Visual Basic 6.0 のサンプル プロジェクトを例として使用して、アップグレード ウィザードの適用手順を説明します。このサンプル プロジェクトの名前を SumApp とします。このアプリケーションを実行し、コマンド ボタンをクリックすると、2 つの入力フィールドの値を加算した結果がボックス内に表示されます。このアプリケーションは、加算を実行する前に、ユーザーが適切な数値をボックス内に入力したかどうかを確認します。

Visual Studio .NET からのアップグレード ウィザードへのアクセス

ここでは、Visual Studio .NET IDE のアップグレード ウィザードを使用した、サンプル アプリケーションのアップグレード方法について説明します。プロジェクトをアップグレードするには、以下の手順を実行する必要があります。

1. Visual Studio .NET を開きます。
2. **[ファイル]** メニューの **[開く]** をクリックし、**[プロジェクト]** をクリックします。**[プロジェクトを開く]** ダイアログ ボックスが表示されます。Visual Basic 6.0 プロジェクト ファイルの場所に移動し、目的のプロジェクト ファイルをクリックします。この例では、SumApp プロジェクトを保存した場所に移動し、SumApp.vbp をクリックして **[開く]** をクリックします。

3. ウェルカム ページを読んで、[次へ] をクリックします。

4.[**プロジェクトの種類の選択**] ページでは、適切なオプションが既定で選択されています。[**次へ**] をクリックします。

メモ：このページでは、アップグレードのオプションを設定します。ターゲット プロジェクトについて生成するアプリケーションは、2 種類から選択できます。EXE プロジェクトは実行ファイルを生成するプロジェクトにアップグレードされ、DLL/カスタム コントロール プロジェクトは DLL を生成するプロジェクトにアップグレードされます。標準の EXE プロジェクトおよび DLL プロジェクトの場合、アップグレードの種類は自動的に決定されます。EXE と DLL のハイブリッドのように動作する EXE サーバー プロジェクトなどそれ以外の種類の場合は、Visual Basic .NET の EXE プロジェクトまたは DLL プロジェクトのいずれかにアップグレードできます。この例では、サンプル アプリケーションが標準の EXE であるため、[EXE] オプションが自動で選択され、[DLL/custom projects upgrade] オプションは選択できません。

---

5. [**新しいプロジェクトを作る場所を指定してください**] ページでは、ウィザードによってターゲット ディレクトリの名前が projectname.NET に指定されます。projectname は元のプロジェクト名です。この例では、SumApp.NET という名前が指定されます。[**次へ**] をクリックします。

メモ：出力先のディレクトリが既に存在し、ファイルが格納されている場合は、出力先ディレクトリが空ではないことを示す警告が表示されます。ウィザードから別のディレクトリを選択するように求められます。ディレクトリがまだ存在しない場合は、ディレクトリの作成を確認するための警告が表示されます。ディレクトリを作成する場合は [はい]、別のターゲット ディレクトリを指定する場合は [いいえ] をクリックします。この例では、SumApp.NET フォルダがまだ存在していないため、[はい] をクリックしてターゲット ディレクトリを作成し、ウィザードの次のページに進みます。

---

6. ウィザードの [**アップグレード可能**] ページで、アップグレードの準備が完了したことが通知されます。[**次へ**] をクリックしてアップグレードを開始します。

ウィザードの次のページでは、アップグレード プロセスの状態に関する情報が表示されます。アップグレード ウィザードが現在実行しているタスクが [**状態**] ボックスに表示され、タスク完了までのおよその残り時間が進行状況バーに表示されます。アップグレードに必要な時間は 1 分 (サンプル アプリケーションのように小規模なプロジェクトの場合) から数時間 (数百のフォームやクラスを含むプロジェクトの場合) までさまざまです。処理が完了すると、アップグレードされた新しいプロジェクトが Visual Studio .NET で開きます。サンプルを図 5.3 に示します。

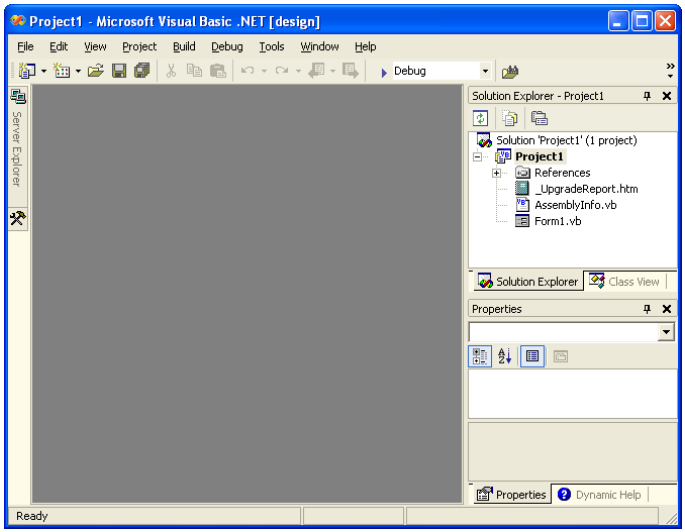

図 5.3

Visual Basic *アップグレード ウィザード適用後のサンプル アプリケーション*

アップグレードされたプロジェクトにはフォーム モジュール Form1 が含まれます。これは、元のプロジェクトの Visual Basic 6.0 フォーム モジュールに対応します。ソリューション エクスプローラ (図 5.3 に示された IDE の右上) で Form1.vb をダブルクリックして、アップグレードされたフォームを開きます。元のフォームと同様な新しいフォームが表示されます。このフォームを図 5.4 に示します。

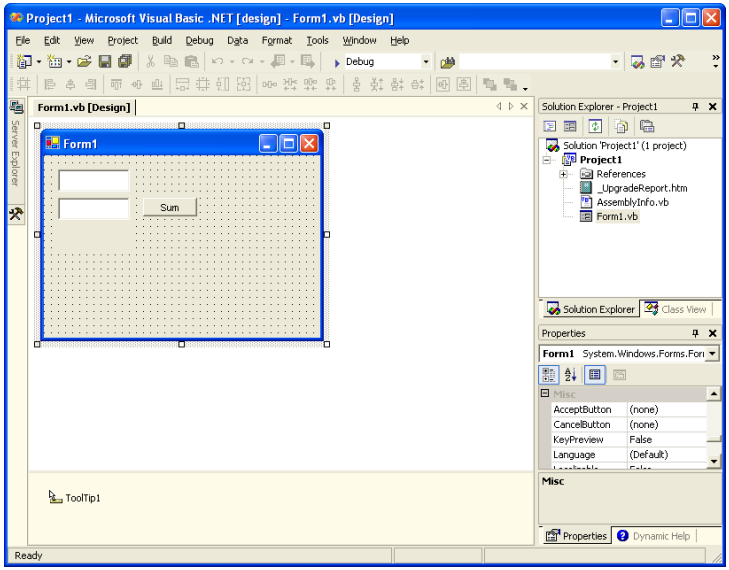

図 5.4

アップグレードされたフォーム

アップグレードされたアプリケーションを実行するには、F5 キーを押します。アップグレードされたプロジェクトを初めてビルドまたは実行すると、ソリューション ファイルの保存を求めるプロンプトが表示されます。.NET ソリューションは、ソリューション エクスプローラの表示ツリーのルートに対応します。ソリューション ファイルは、Visual Basic 6.0 のグループ ファイルに似ています。アップグレードされたプロジェクトはこのソリューション ファイルに格納されます。指定されたソリューション名をそのまま使用し、Enter キーを押して保存します。プロジェクトが自動的にビルドおよび実行されます。Form1 が表示され、ユーザーの入力待ち状態になります。サンプルの入力データから得られた結果を図 5.5 に示します。

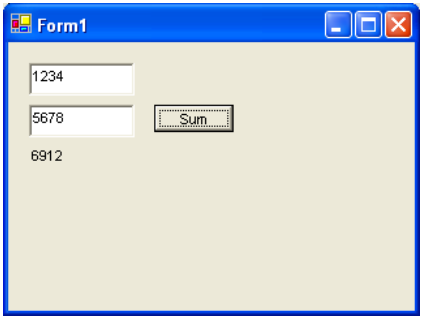

図 5.5

アップグレードされたアプリケーションの実行

コマンド ラインからのアップグレード ツールへのアクセス

VBUpgrade.exe アップグレード プログラムを使用して、コマンド ラインから自動アップグレード ツールを適用することもできます。このプログラムは、Visual Studio .NET でアクセスするアップグレード ウィザードと同じアップグレード エンジンを使用します。既定のインストール ディレクトリを使用している場合、以下のパスでこのプログラムを見つけることができます。

drive:\Program Files\Microsoft Visual Studio .NET version\Vb7\VBUpgrade\

このパスの drive はインストール ディレクトリがあるドライブを表しており、version は Visual Studio .NET のバージョンを表しています。たとえば、このガイドは Visual Studio .NET 2003 IDE を対象としているため、完全パスは次のようになります。

drive:\Program Files\Microsoft Visual Studio .NET 2003\Vb7\VBUpgrade\

既定のディレクトリ以外の場所にアップグレード ツールをインストールした場合、またはプログラムのインストール場所が不明な場合は、Windows エクスプローラを使用してプログラムを検索することができます。以下の説明では、このユーティリティのディレクトリをコマンド検索パスに追加していると想定します。

コマンド ラインのアップグレード ツールを使用すると、アップグレード ウィザードと同じ結果が得られます。プロジェクトをアップグレードするには、プロジェクト ファイルの名前と出力先ディレクトリをコマンドの引数として指定する必要があります。出力先ディレクトリが存在しない場合、アップグレード ツールは自動的にそのディレクトリを作成します。たとえば、以下のステートメントを実行すると、サンプル プロジェクトがアップグレードされ、C:\SumApp\SumApp.vbp to the C:\SumApp\SumApp.NET ディレクトリに保存されます。

```
vbupgrade c:\SumApp\SumApp.vbp /Out c:\SumApp\SumApp.NET
```

ディレクトリ名は適切な場所の名前に置き換える必要があります。

コマンド ライン バージョンのアップグレード ツールにはウィザード ページがありません。さらに、このツールは出力先ディレクトリ内のファイルを一切削除しません。出力先ディレクトリが既に存在し、ファイルが格納されている場合、コマンド ライン ツールは実行を中止してエラーを返します。

アップグレード システムで利用可能なメモリ量が限られている場合は、コマンド ライン ツールの使用をお勧めします。このツールを使用すると、アップグレード ツールの実行時に Visual Studio .NET で必要なシステム リソースが節約されます。システム構成によっても異なりますが、Visual Studio .NET は約 15 MB のメモリを消費します。

コマンド ライン オプション

アップグレード ツールのコマンド ライン バージョンには、アップグレードの実行時に指定できるさまざまなオプションが用意されています。コマンド プロンプトで以下のコマンドを実行すると、このツールで使用可能なオプションの完全な一覧が表示されます。

vbupgrade /?

このコマンドを実行すると、まずウェルカム メッセージが表示され、その後に使用可能なオプション スイッチとその意味が表示されます。

```
Microsoft (R) Visual Basic.NET Upgrade Tool Version 7.00.9238.0
Copyright (C) Microsoft Corp 2000-2001.
Portions copyright ArtinSoft S.A.
All rights reserved.
Usage: VBUpgrade <filename> [/Out <directory>] [/NoLog • /LogFile <filename>] [/Verbose]
[/GenerateInterfaces]
/?                    Display this message
/Out                    Target directory (default is ".\OutDir")
/Verbose                Outputs status and results
/NoLog                  Don't write a log file
/LogFile                Log file name (default is
                        "<ProjectFileName>.log")
/GenerateInterfaces    Generates interfaces for public classes
```

このツールには冗長モードが用意されています。冗長モードでは、ツールが現在実行中の処理を正確に把握することができます。これは、大規模なアプリケーションのアップグレード進行状況を確認する方法として便利です。サンプル アプリケーションのアップグレード中にこのツールを使用すると、以下の内容が表示されます。

```
C:\Program Files\Microsoft Visual Studio .NET 2003\Vb7\VBUpgrade>vbupgrade C:\SumApp.vbp /Verbose
Microsoft (R) Visual Basic.NET Upgrade Tool Version 7.00.9238.0
Copyright (C) Microsoft Corp 2000-2001.
Portions copyright ArtinSoft S.A.
All rights reserved.

Initializing...
Parsing Form1.frm...
```

```
Loading User Type Library Form1
Pre-processing Form1...
Upgrading Form1...
Upgrading Form1.Form1
Upgrading Form1.Command1
Upgrading Form1.Text2
Upgrading Form1.Text1
Upgrading Form1.Label1
Upgrading Form1.Command1_Click
Writing Form1.vb...
Writing Form1.resx...
Writing project file Project1.vbproj...
Writing project file Project1.vbproj.user...
Writing project file _UpgradeReport.htm...
Writing AssemblyInfo.vb...
Writing project file Project1.vbproj...
Writing project file Project1.vbproj.user...
Writing project file _UpgradeReport.htm...
```

上記の例では、Visual Studio .NET の既定のインストール ディレクトリが想定されていることに注意してください。このサンプル出力では、Visual Basic 6.0 プロジェクトのアップグレードで実行される各段階が表示されます。ツールの実行中、出力行の表示に時間がかかるものがあることに注意してください。場合によっては数分かかることもあります。Windows タスク マネージャを使用すると、アップグレード プロセスの状態をチェックできます。この点については次に説明します。

プロジェクト グループのアップグレード

Visual Basic 6.0 では、開発者が複数のプロジェクトを 1 つのプロジェクト グループ ファイルにグループ化できます。グループ ファイルをコンパイルすると、グループ内の指定されたファイルがすべてまとめてコンパイルされます。また、デバッガではすべてのプロジェクトのコードを全域にわたってデバッグできます。アプリケーション内にプロジェクト依存関係が存在するときに、この機能が役立ちます。アップグレード ウィザードはプロジェクト グループをサポートしていません。この主な理由は、すべてのプロジェクトについて同時に管理する必要のあるコンテキスト情報のサイズにあります。

Visual Basic .NET では、プロジェクト グループに相当する機能はソリューションです。1 つのソリューションには、さまざまな言語で記述された複数のプロジェクトと、各プロジェクト間の参照を格納することができます。Visual Basic 6.0 のプロジェクト グループと同様に、これらのプロジェクトはまとめてコンパイルおよびデバッグできます。

中程度以上の複雑さを持つアップグレードされたアプリケーションをテストする場合、元のアプリケーションをテストするときと少なくとも同程度の作業が必要になります。アプリケーション機能のテストには、その他のコンポーネントのアップグレードと並行して実行できるものもあります。こうした並列タスクでは、タスクに応じてプロジェクトのアップグレード順序を調整および定義する必要があります。詳細については、この後の「アップグレード後のアプリケーションのテストとデバッグ」を参照してください。

中間結果のテストで使用する方法に応じて、以下のような順序でプロジェクトをアップグレードします。

- 依存関係を持たないプロジェクトを最初にアップグレードします。コア機能を最初にアップグレードすると、コア コンポーネントのテストから始めて、テストのたびに機能を増やしていくというように、異なるレベルでのテストを実行できます。このアプローチは問題を切り分けできるため、複雑なアプリケーションをアップグレードする場合の用途に適しています。また、アップグレードした機能のデモンストレーションを早期に行う必要がありません。
- 他のプロジェクトから依存されないプロジェクトを最初にアップグレードします。依存関係階層の最上部にあるコンポーネントを最初にアップグレードできます。必要な場合は、他のコンポーネントに対する相互運用を使用して、アプリケーションの機能をテストできます。このアプローチでは、アップグレードした機能のデモンストレーションを早期に行うことができますが、そのために相互運用ラッパーを作成するための追加作業が必要になることがあります。

混合アプローチを採用して、両方のメリットを得ることもできます。

次の例では、最初に説明したアプローチを使用してプロジェクト グループをアップグレードします。サンプルのプロジェクト グループには、2 つのプロジェクトが含まれています。1 つは加算処理の実行に使用するクラスを含む DLL を生成するプロジェクトです。もう 1 つはユーザー入力用のフォームを表示するフロントエンド プログラムで、DLL が提供するサービスを利用します。プロジェクト グループをコンパイルし、生成されたプログラムを実行すると、加算する数値の入力をユーザーに求めるフォームが表示されます。不適切な値を入力すると、DLL 内部でエラーが発生し、そのエラーに関する説明がフォームに表示されます。このサンプル アプリケーションは付属の CD に含まれています。

コア コンポーネントを優先してこのプロジェクト グループをアップグレードするには、以下の手順を実行する必要があります。

1. Group1.vbg ファイル グループを Visual Basic 6.0 で開きます。
2. アップグレード ウィザードを使用して、依存関係を持たないプロジェクトをすべてアップグレードします。各プロジェクトは次のように個別に処理する必要があります。
   a. Visual Basic 6.0 で、Project1 に関連付けられた参照を調べます。プロジェクト ウィンドウで Project1 をクリックします。既定では、これは IDE の右上にあります。**[プロジェクト]** メニューの **[参照設定]** をクリックします。**[参照設定 – Project1.vbp]** ダイアログ ボックスが表示されます。
   b. **[参照設定 – Project1.vbp]** ダイアログ ボックスでは、Project1 が Utilities に依存していることが表示されます。Utilities プロジェクトは他の依存関係を持たないため、このプロジェクトを最初にアップグレードします。大規模なアプリケーションの場合は、この章の前半の「すべての依存関係の特定」で説明した、評価ツールのファイル依存関係グラフを使用すると、この種の依存関係を分析できます。「Visual Studio .NET からのアップグレード ウィザードへのアクセス」の説明に従って、アップグレード ウィザードを使用して Utilities プロジェクトをアップグレードします。

3. Visual Basic .NET を開いた後、依存プロジェクトをアップグレードしてソリューションに追加できます。[ファイル] メニューの [プロジェクトの追加] をポイントし、[既存のプロジェクト] をクリックします。[既存プロジェクトの追加] ダイアログ ボックスが表示されます。Project1.vbp をクリックし、[開く] をクリックしてアップグレードを開始します。「Visual Studio .NET からのアップグレード ウィザードへのアクセス」の説明に従って、既定のアップグレード オプションをそのまま使用します。

4. アップグレードした新しいプロジェクト参照を、対応する .NET プロジェクトで置き換えます。参照を置き換えるには、以下の手順に従ってください。

   a. ソリューション エクスプローラで、Project1 プロジェクト フォルダを開きます。[参照設定] フォルダを開きます。Utilities を右クリックし、[削除] をクリックします。これで古いバージョンの Utilities への参照が削除されます。

   b. Visual Basic .NET バージョンの新しい Utilities プロジェクトへの参照を追加するには、[参照設定] フォルダを右クリックして [参照の追加] をクリックします。[参照の追加] ダイアログ ボックスが表示されます。利用できるプロジェクトを表示するには、[プロジェクト] タブをクリックします。参照を追加するには、Utilities をダブルクリックし、[OK] をクリックします。

メモ： 以上で、Visual Basic 6.0 バージョンの元の Utilities への参照が、アップグレードされた新しいバージョンに置き換えられました。ただし、インターフェイスやデータ型が異なる場合があります。インターフェイスやデータ型が異なると、このコンポーネントにアクセスする場所でコンパイル エラーやランタイム エラーが発生します。手順 3 で Project1 をアップグレードしたとき、アップグレード ウィザードが元のバージョンの Utilities への参照を調べて、サードパーティ製の外部コンポーネントとして処理しました。相互運用ラッパーが作成され、メンバ宣言に対して変換は適用されませんでした。このため、Project1 内におけるこのコンポーネントの使用方法は変更されません。相違は、Utilities ライブラリをアップグレードする際に発生します。アップグレード ウィザードはインターフェイス宣言の一部を変換します。そのため、コンポーネントの使用場所を更新する必要があります。これらの変更には、イベント パラメータやアップグレードされたデータ型が含まれます。

---

5. Project1 内部での Utilities に対するアクセスを確認し、必要に応じて更新します。この例では、Utilities インターフェイスがアップグレード時に変更されなかったため、Project1 で必要な変更はありません。ただし、インターフェイスとデータ型がより複雑な場合は、アプリケーションをコンパイルするために追加作業が必要になることがあります。

6. これで、ソリューションをコンパイル、実行、およびテストする準備が整いました。ソリューション エクスプローラで Project1 を右クリックし、[スタートアップ プロジェクトに設定] をクリックして、Project1 をスタートアップ プロジェクトに設定します。

7. 以上で、これら 2 つのプロジェクトのアップグレードが完了し、関連テストを適用できるようになりました。アプリケーション機能全体に対するこのサブセットのテストおよび修正を終えたら、安定した基盤の上で新しいレベルの機能をアップグレードおよびテストできるようになります。この例では、アップグレードする他のプロジェクトがグループ内にないため、処理は完了です。

また、グループ内の他のプロジェクトから依存されないプロジェクトをまず優先して、このプロジェクトをアップグレードすることもできます。この方法を適用するには、以下の手順を実行する必要があります。

1. Visual Studio .NET を開きます。
2. グループ内の他のプロジェクトから依存されないプロジェクトの 1 つをアップグレードします。このサンプル プロジェクト グループの場合、Project1 に依存するプロジェクトがないため、Project1 を最初にアップグレードします。「Visual Studio .NET からのアップグレード ウィザードへのアクセス」で説明した手順に従ってください。
3. 次に、ステップ 1 でアップグレードしたプロジェクトにサービスを提供するプロジェクトをアップグレードします。この例では、Project1 が使用するプロジェクトは Utilities のみです。Visual Basic .NET で、**[ファイル]** メニューの **[プロジェクトの追加]** をクリックし、**[既存のプロジェクト]** をクリックします。**[既存プロジェクトの追加]** ダイアログ ボックスが表示されます。Utilities をクリックし、**[開く]** をクリックします。既定のアップグレード オプションをそのまま使用します。
4. Project1 は、Visual Basic 6.0 バージョンの Utilities を依然として参照します。この参照を削除して、Visual Basic .NET バージョンにアップグレードした Utilities への参照に置き換える必要があります。Utilities への参照を削除するには、Project1 の参照設定ノードを展開し、Utilities を右クリックして **[削除]** をクリックします。
5. 新しい Utilities プロジェクトへの参照を追加するには、Project1 の **[参照設定]** を右クリックして **[参照の追加]** をクリックします。**[参照の追加]** ダイアログ ボックスの **[プロジェクト]** タブをクリックし、Utilities をダブルクリックして [OK] をクリックします。
6. Utilities への参照を Visual Basic 6.0 バージョンから Visual Basic .NET バージョンに置き換えるときのその他の考慮事項については、前の手順 5 を参照してください。コンポーネントのアップグレード順序の変更にかかわらず、同じ問題が発生する可能性があります。
7. これで、ソリューションをコンパイル、実行、およびテストする準備が整いました。手順 2 の後、元のバージョンの Utilities を相互運用経由で使用して Project1 をテストできるという点に注意してください。

これら 2 つの手順を説明するために使用した例は比較的小さなプロジェクトですが、多数のプロジェクトから構成されるプロジェクト グループに対して、いずれかのアプローチを適用できるということを覚えておいてください。適用するアプローチの選択は、コンポーネントおよびコンポーネント間の依存関係についての入念な分析に基づいて行う必要があります。依存関係を持たないコンポーネントを多数含むプロジェクト グループの場合は、最初に提示したアプローチのほうが適しています。一方、他のコンポーネントから依存されないコンポーネントを多数含むプロジェクト グループの場合は、2 番目のアプローチのほうが適しています。

アップグレードの進捗状況の確認

コマンド ラインからアップグレード ツールを実行すると、次の 2 つのプロセスが起動されます。

- コマンド ライン モードのアップグレード ツールのフロント エンドである、VBUpgrade.exe
- アップグレード エンジンである、VBUD.exe

Visual Studio .NET でアップグレード ウィザードにアクセスした場合も、同じアップグレード エンジンが起動されます。ただし、この場合に起動されるのは VBUD.exe のみです。

コマンド ライン バージョンのアップグレード ツールによって起動された 2 つのアップグレード プロセスが、Windows のタスク マネージャに表示されます。

Visual Studio .NET 2003 バージョンのアップグレード ウィザードは、ソース コード ファイルを 1 つずつ処理し、各結果を一時ファイルとして保存します。この方法ですべてのファイルを処理した後、最終的な Visual Basic .NET ソース コードを出力先ディレクトリに書き込みます。これがユーザーに渡される最終結果となります。ファイルのアップグレード中に問題が発生してアップグレード ツールの実行が強制終了された場合は、それまでに生成された結果の一時ファイルがすべて失われ、最終的な Visual Basic .NET ソース コードは生成されません。こうした状況を避けるため、Visual Basic 2005 バージョンのアップグレード ウィザードと ArtinSoft 製の Visual Basic Upgrade Wizard Companion は、最終的なアップグレード結果をファイルごとに生成します。ファイルが処理された後、最終的なソース コードが書き込まれ、ターゲット プロジェクトとソリューションのファイルが更新されます。これにより、他のファイルのアップグレード中に致命的なエラーが発生した場合でも、最終結果を利用できるようになります。

アップグレード ウィザードの実行中に発生する問題の修正

ここでは、アップグレード ウィザードの実行中に発生する可能性のある一般的な問題について説明します。通常は、必要な情報を指定したら、アップグレード プロセスを無人で実行できます。ただし、大規模なアプリケーションの場合、アップグレード処理に数時間かかることがあり、その間にユーザーが注意する必要のあるさまざまな状況が発生する可能性があります。以下の一覧は、発生する可能性のある最も一般的な問題を示しています。

- **メモリ スワップ。** アプリケーションのアップグレードに使用するコンピュータに、大規模なアプリケーションを処理するための十分なメモリ リソースが搭載されていない場合、オペレーティング システムはメモリをディスクにスワップして、アップグレード プロセスに追加メモリを割り当てようとします。メモリとディスクの間でスワップが頻繁に発生すると、他のタスクよりも多くの処理能力が費やされる場合があります。この状況を、システムが "スラッシング" 状態にあると言います。この状態になると、アップグレード プロセスが完了するまでの時間が大幅に増加することがあります。この問題が発生したときは、自動アップグレードを停止し、システム メモリを追加してから、自動アップグレードを再開することをお勧めします。図 5.6 は、大規模なアプリケーションのアップグレード中にスラッシング状態になったシステムを表しています。すべてのプロセッサ時間がカーネルによって使用され、ほとんどすべてのメモリが使用されていることに注意してください。

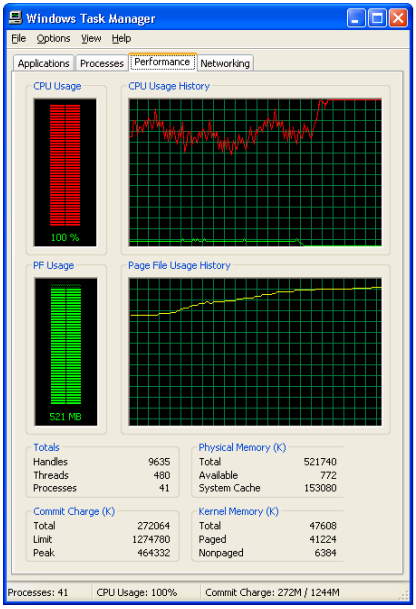

図 5.6

典型的な"スラッシング"シナリオにおけるWindows タスク マネージャのパフォーマンス表示

- **不適切な参照。**アップグレードするプロジェクトが 1 つのアプリケーションに複数含まれている場合、プロジェクトを選択して 1 つずつアップグレードできます。アップグレードの優先順位やその他の要因に応じて、ソース コードの一部を再編成することができます。ソース コードを再編成する際に、ユーザー プロジェクトまたはサードパーティ コンポーネントへの参照を誤って他の参照に変更した場合、メンバやデータ型に関する必要なすべての情報をアップグレード ウィザードが取得できなくなります。これにより、これらのコンポーネントのメンバが使用されるすべての場所で、既定のプロパティの取得に関する警告が発生します。この原因は、これらのメンバがアップグレード ウィザードによって自動的にアップグレードされないことにあります。この問題が発生した場合は、警告の対象となっている要素に関連した、参照およびアクセスを確認する必要があります。
- **サードパーティ コンポーネントの試用版。**アップグレード ウィザードは、プロジェクト内で参照されるすべてのサードパーティ コンポーネントを読み込みます。試用版のコンポーネントの中には、コンポーネントの初期化時にライセンスのダイアログ ボックスを表示して、ユーザーの入力待ち状態になるものがあります。この場合、ダイアログ ボックスが閉じられるまでアップグレード プロセスが中断されることになります。これらのダイアログ ボックスには注意を払い、アップグレード プロセスを続行できるようにする必要があります。
- **アップグレード ウィザードの例外。**アップグレード ウィザードによるファイルの処理中に、致命的な問題が発生することがまれにあります。たとえば、非標準のインターフェイスが使用されているサードパーティ コンポーネントや、その他の条件が原因で例外が発生することがあります。このケースでは、アップグレード ツールの実行が停止します。問題を解決するためには、最初の手順として、例外の

原因となっているファイルおよびソース コードを分離します。分離したら、そのソース コードをコメント アウトできます。問題のあるコードを削除したら、アップグレード プロセスを再開できます。この種の例外については、他の開発者が回避できるように、ニュースグループに報告することをお勧めします。推奨されるニュースグループとして、microsoft.public.dotnet.languages.vb.upgrade 、microsoft.public.dotnet.languages.visualbasic.migration などがあります。

既知の例外について事前に認識して計画を立てることにより、アップグレードをスムーズに行うことができます。

手作業による変更を加えてアップグレードを完了

アップグレード ウィザードを Visual Basic 6.0 プロジェクトに適用し、Visual Basic .NET コードを生成した後、新しい環境でアプリケーションを実行するために、アップグレードされたコードに対する調整が必要になることがよくあります。ここでは、必要となる変更のいくつかについて説明します。

ビルド可能な状態の確認

このプロセスの最初の手順は、アプリケーションを Visual Studio .NET でコンパイルできるようにすることです。アプリケーションをコンパイルする前に、プロジェクト参照を調べて、必要なすべての参照が適切な状態で存在していることを確認する必要があります。ソリューション エクスプローラで、対応するプロジェクトの [参照設定] フォルダを開き、すべての参照が適切に存在していることを確認します。参照の横に警告アイコンが表示されている場合、最も可能性の高い原因として、相互運用ラッパーの作成時、または作成した相互運用ラッパーをプロジェクトに追加したときに発生した問題が考えられます。警告状態を解消するには、参照をいったん削除して、手動で追加し直す必要があります。

プロジェクト参照の確認が完了したら、新しいバージョンのアプリケーションをビルドできるようになります。[ビルド] メニューの [ソリューションのビルド] をクリックします。

Visual Studio .NET IDE を使用すると、検出されたすべてのコンパイル エラーを特定および検索できます。また、検出されたすべての問題が [タスク一覧] に表示されます。一覧で任意の行をダブルクリックすると、問題が発生した特定のコード行に移動できます。[タスク一覧] の項目に表示されている情報を利用し、第 7 ～ 11 章で説明するテクニックを適用することで、問題を修正できます。

アプリケーションのコンパイルが正常に完了したら、アプリケーションのフォームおよびコントロールのデザイン時の表示を確認できます。デザイン時のコードを検証するには、各フォームを開き、すべてのコントロールが正常に表示されていることを確認します。コントロールが表示されない場合は、対応する参照を調べて、サポートされていない機能の有無を確認します。

全体の確認が終了したら、テスト フェーズおよびデバッグ フェーズを開始できます。

次では、アップグレードの自動処理部分で発生する最も一般的な問題について説明します。これらの問題はアップグレード ウィザードによって報告され、Visual Basic .NET でのコンパイル エラーの原因となります。

一般的なコンパイル エラー

ここでは、Visual Basic 6.0 のサポートされない機能に起因する最も一般的なコンパイル エラーについて説明します。アップグレード ウィザードはこれらの問題をアップグレードしません。これは、対応する機能が Visual Basic .NET にはないか、明示的に指定する必要のある情報が元のソース コードでは暗黙的に指定されているためです。各問題について、Visual Basic 6.0 の元のコード例と、自動生成された Visual Basic .NET のコード例を紹介します。また、問題を解決するために必要な修正についても説明します。ここの目的は、アプリケーションを Visual Basic .NET にアップグレードした後に発生する可能性のある問題に関し、それらを修正するための一般的な必要手順とテクニックについて説明することです。サポートされない特定の機能に関する詳細については、第 8 章「一般的に使用されるVisual Basic 6.0 言語機能のアップグレード」を参照してください。

標準的な Visual Basic 6.0 アプリケーションに関し、これまでにアップグレード ウィザードから報告された最も一般的なアップグレードの問題は、既定のプロパティの取得問題です。これは、変数のデータ型に関する情報の欠落が原因で発生します。この問題は、変数の使用方法に応じて、コンパイル エラーまたはランタイム エラーの原因となる可能性があります。たとえば、この問題が発生すると、アップグレード ウィザードによって以下のコメントが生成されます。

```
' UPGRADE_WARNING: オブジェクト MyObj.MyProperty の既定プロパティを解決できませんでした。
```

この問題を修正するには、この章の前半の「アップグレード ウィザードのレポートに関する確認」で紹介したテクニックを適用します。

次では、アップグレード ウィザードによって検出され、ユーザーの修正が必要となる、その他のアップグレードの問題について説明します。

As Any パラメータ型に関する問題

Visual Basic 6.0 で外部 DLL 関数にアクセスする場合、対応する関数のパラメータ宣言で As Any パラメータの追加が必要になることがよくあります。Visual Basic .NET では、このデータ型がサポートされなくなりました。以下の Visual Basic 6.0 コード例は、この変更によって発生する問題を示しています。

```
Private Declare Function SystemParametersInfo Lib "user32" _
    Alias "SystemParametersInfoA" (ByVal uAction As Integer, ByVal uParam As _
    Integer, ByRef lpvParam As Any, ByVal fuWinIni As Integer) As Integer

Private Function GetKeyBoardDelay() As Integer
    Const SPI_GETKEYBOARDDELAY As Integer = 22
    Dim delayVal As Integer
    Call SystemParametersInfo(SPI_GETKEYBOARDDELAY, 0, delayVal, 0)
    GetKeyBoardDelay = delayVal
End Function
```

アップグレード ウィザードを実行すると、以下のコードが生成されます。

```
' UPGRADE_ISSUE: パラメータ 'As Any' の宣言はサポートされません。
Private Declare Function SystemParametersInfo Lib "user32" _
    Alias "SystemParametersInfoA"(ByVal uAction As Short, ByVal uParam As Short, _
    ByRef lpvParam As Any, ByVal fuWinIni As Short) As Short

Private Function GetKeyBoardDelay() As Short
    Const SPI_GETKEYBOARDDELAY As Short = 22
    Dim delayVal As Short
    Call SystemParametersInfo(SPI_GETKEYBOARDDELAY, 0, delayVal, 0)
    GetKeyBoardDelay = delayVal
End Function
```

後者のコードでは、不明な As Any データ型を使用しているため、Visual Basic .NET ではコンパイル エラーが発生します。この問題を修正するには、外部関数 SystemParametersInfo の仕様について調べる必要があります。この関数は、uAction パラメータを基にシステム パラメータの値を取得して、その結果を lpvParam パラメータに格納します。ソース コード内での関数の使用方法に応じて、lpvParam パラメータのデータ型を特定のデータ型に変更する必要があります。この場合、取得されるシステム パラメータはキーボードのオート リピート待ち時間で、期待される結果は Short 型になります。この問題を修正するには、As Any データ型を As Short 型に変更します。

このエラーの修正の詳細については、第 13 章「Windows API の使用」を参照してください。

一般に使用されるオブジェクトのプロパティの変更

以下のソース コードは、Visual Basic 6.0 で正常にコンパイルおよび実行できます。しかし、このコードを Visual Basic .NET にアップグレードすると、ここで示されるように、サポートされないプロパティが原因でコンパイル エラーが発生します。

```
' List1 は現在の Form に埋め込まれたリスト ボックスです。
List1.AddItem "Hello"
List1.ListIndex = List1.NewIndex
```

このコードにアップグレード ウィザードを適用すると、以下のコードが生成されます。

```
' List1 は現在の Form に埋め込まれたリスト ボックスです。
List1.Items.Add("Hello")
' UPGRADE_ISSUE: リスト ボックスのプロパティ List1.NewIndex はアップグレードされませんでした。
List1.SelectedIndex = List1.NewIndex
```

このコードを Visual Basic .NET でコンパイルすると、コンパイル エラーが発生します。ListBox コントロールに最後に追加されたアイテムのインデックスを取得するために、NewIndex プロパティが使用されているためです。しかし、アップグレードされたコードでは NewIndex の値を常に決定できるわけではありません。このエラーは、Add メソッドの戻り値 (新しく追加されたアイテムのインデックス) を保存することによって修正できます。修正したコード例を以下に示します。変更箇所は太字で表示されています。

```
' List1 は現在の Form に埋め込まれた ListBox です。
Dim NewIndex As Integer
NewIndex = List1.Items.Add("Hello")
List1.SelectedIndex = NewIndex
```

GoSub に対するサポートの終了

Visual Basic 6.0 で利用できる GoSub ディレクティブを使用すると、プログラムの実行をサブルーチンまたはラベルに分岐できます。以下のコード例は、Visual Basic 6.0 での On…GoSub の使用を示しています。

```
Function Decide(v As Integer) As String
    On v GoSub case1, case2, case3
case1:
    Decide = "case1"
    Exit Function
case2:
    Decide = "case2"
    Exit Function
case3:
    Decide = "case3"
    Exit Function
End Function
```

このコードにアップグレード ウィザードを適用すると、以下が生成されます。

```
Function Decide(ByRef v As Short) As String
    ' UPGRADE_ISSUE: On...Gosub ステートメントはサポートされていません。
    On v GoSub case1, case2, case3
case1:
    Decide = "case1"
    Exit Function
case2:
    Decide = "case2"
    Exit Function
case3:
    Decide = "case3"
    Exit Function
End Function
```

Visual Basic .NET では、On … GoSub コンストラクトがサポートされなくなりました。アップグレード ウィザードによって生成されたコードをコンパイルすると、コンパイル エラーが発生します。ただし、以下に示すように、問題のあるコンストラクトを削除することにより、このコードを修正できます。

```
Function Decide(ByRef v As Short) As String
    If v = 1 Then
        Decide = "case1"
    ElseIf v = 2 Then
        Decide = "case2"
    ElseIf v = 3 Then
        Decide = "case3"
    End If
End Function
```

マルチドキュメント インターフェイス (MDI) Form の ActiveForm の変更

Visual Basic 6.0 では、MDIForm の ActiveForm プロパティを使用して、アクティブな子フォームを参照できます。このフォームのコントロールは、プロパティとしてアクセスできます。Visual Basic .NET の ActiveForm は、コントロールを持たない汎用型です。

以下のコード例は、Visual Basic 6.0 では正常に動作します。

```
' Form1 という名前の子フォームが MDIForm1 にあると仮定します。Form1 が現在表示されており、
' Label1 という名前の Label コントロールがこのフォームに含まれています。
Dim controlStr As String
controlStr = MDIForm1.ActiveForm.Label1.Caption
```

アップグレード ウィザードを適用すると、以下のコードが生成されます。

```
Dim controlStr As String
' UPGRADE_ISSUE: コントロール Label1 は、汎用名前空間 ActiveMDIChild 内にあるため、
' 解決できませんでした。
controlStr = MDIForm1.DefInstance.ActiveMDIChild.Label1.Caption
```

UPGRADE_ISSUE コメントに記述されているように、ActiveForm が変更されたため、アップグレードされたコードで Label1 コントロールを解決できなくなりました。この問題を修正するには、対応する Form クラスで明示的に型キャストし、アクセスするコントロール メンバを手動でアップグレードする必要があります。これを以下のコード例に示します。

```
Dim controlStr As String
controlStr = CType(MDIForm1.DefInstance.ActiveMdiChild, Form1).Label1.Text
```

ここで提示した問題は、アップグレード ウィザードが生成したコードに対し、手動で適用する必要のある最も一般的な変更を示しています。ただし、包括的な一覧を示すものではありません。アップグレードに関するその他の問題、およびこれらの問題に適用する手動修正の詳細については、第 7 ～ 13 章を参照してください。

アップグレード後のアプリケーションのテストとデバッグ

テスト フェーズおよびデバッグ フェーズの目的は、アプリケーションの実行時に発生する問題を特定および修正することです。ここでは、アップグレードされたアプリケーションのテストおよびデバッグについて説明します。

アップグレード レポートに記載される問題

アップグレード レポートには、以下の種類の問題が記載されることがあり、これらの問題については対処する必要があります。

- **アップグレードに関する問題。** これらはサポートされないクラス メンバや言語要素を示しています。特定された各項目について、機能を見直す必要があり、影響を受ける式の一部は .NET の類似した要素に手動でアップグレードする必要があります。アップグレードした各項目は、シナリオを限定してテストする必要があります。
- **アップグレードの ToDo。** これらはソース コード内でユーザーによる修正が必要な箇所を示しています。これらの項目に含まれる指示に従う必要があります。作業後のコードをテストすることにより、修正が適切に適用されたことを確認できます。
- **アップグレードに関する警告。** これらの項目は、コンパイル可能であるにもかかわらず、ランタイム エラーが発生する可能性のあるソース コードを示しています。こうしたコードについては、十分にテストを行って、ランタイム エラーが発生するかどうか、およびどのようなエラーが発生するかを確認する必要があります。ランタイム エラーの発生を防ぐため、追加の変更作業が必要になる場合もあります。
- **デザイン エラー。** サポートされないデザイン時プロパティを使用すると、これらの問題が発生します。動作が変更されたメンバもこの問題の対象となります。これらのメンバが提供する機能は、再デザインするか、類似した .NET 要素に変更することができます。動作の相違がアプリケーションのコア機能に影響するかどうかを確認するため、コードをテストする必要があります。
- **グローバル警告。** ガベージ コレクションが原因で決定論的なオブジェクト ライフタイムが十分でないなどのグローバルな問題が含まれます。これらの警告の大部分はセットアップに関する問題が原因で、簡単に修正およびテストできます。追加の修正作業が必要なその他の警告もあります。たとえば、ソース コード行がアップグレード ウィザードによって認識されない場合は、その行を手動でアップグレードする必要があります。
- **アップグレードに関する注意。** ソース コードが大幅に変更された場合、またはアップグレードされたコードと元の Visual Basic 6.0 バージョンで動作が一部異なる場合に生成されます。動作の相違がアプリケーションのコア機能に影響するかどうかを確認するため、コードをテストする必要があります。

アップグレードされたすべてのアプリケーション機能について、単体テストおよびシステム テストを実行する必要があります。このフェーズでは、元のアプリケーションの仕様書とテスト ケースが必須です。Visual Basic .NET バージョンのアプリケーションに対し、テスト ケースの調整または追加が必要になる場合もあります。

大規模なアプリケーションをアップグレードする場合は、アプリケーションの一部のテストを、その他の部分のアップグレード中に行う必要が生じることもあります。こうした状況では、アップグレード タスクとテスト タスクの

統合に関して、以下の点を考慮する必要があります。

- コンポーネントまたはプロジェクトのアップグレード順序の問題で、アップグレード前のコンポーネントに依存するコンポーネントのテストが必要になる場合があります。相互運用機能を利用して元のコンポーネントにアクセスすることで、これらアップグレード済みのコンポーネントのテスト作業を進めることができます。このアプローチを使用すると、作業を並行して続けることができます。また、テスト用の相互運用ラッパーを必要に応じて個別に作成できるため、アップグレード フェーズとテスト フェーズの間で必要な調整は少なくなります。主な欠点は、統合されたアプリケーションでは問題の切り分けが困難になることと、最終製品で不要な相互運用ラッパーを作成するための追加作業が必要になることです。
- テストは、コンポーネント依存関係グラフで指定された順序で実行できます。テストごとに 1 つのレベルのアプリケーション機能が検証されます。あるコンポーネントをテストすると、そのコンポーネントが依存するすべてのコンポーネントもテストされます。したがって、そのようなすべてのコンポーネントについて、アップグレードを済ませておく必要があります。より複雑なコンポーネントをテストできるようにするために、最も基本的なコンポーネントをまずアップグレードする必要があります。このアプローチの利点は、作業を漸進的に進められるため、検出されたすべての問題を容易に切り離して修正できることです。主な欠点は、追加的な計画および調整が必要になることと、作業の並列度が低下することです。

2 つのアプローチを混ぜたテスト方法を適用することもできます。最初の方法を使用して一部のアプリケーション機能をアップグレードする一方で、2 番目の方法に従って基本コンポーネントをアップグレードおよびテストできます。これにより、最もコアな機能のアップグレードが進行中であっても、エンド ユーザーにデモンストレーション可能なアップグレード結果を早い時期に取得できます。両ワークフローは最終的に 1 つの中間点に収束します。この中間点では、基本コンポーネントがアップグレードされ、より高レベルの機能を構成するコンポーネントで必要なすべてのサービスが提供されます。混合アプローチの主な欠点は、それぞれのアップグレード方法に適し、追加作業を必要としないアプリケーション機能の細かな要素を特定するのが難しいことです。

ランタイム エラーの修正

アプリケーションのアップグレード完了後に発生するエラーの中で最も検出が困難なのは、実行時にのみ特定可能なエラーです。潜在的なランタイム エラーの多くが、アップグレード ウィザードによって検出されます。ただし、アップグレードしたアプリケーションを実行して、十分にテストした後で初めて検出される問題もあります。このようなエラーを特定する場合には、元のアプリケーションのテスト ケースとデザイン ドキュメントが特に重要になります。

ここの残りの部分では、アップグレード ウィザードによって検出され、ランタイム エラーを発生させる一般的なアップグレードの問題のいくつかについて説明します。これらの問題は手動で修正する必要があります。

ユーザー定義型の変更

Visual Basic 6.0 では、Type キーワードを使用してユーザー定義型を定義できます。Type ブロックに記述されたすべてのメンバは、Visual Basic 6.0 ランタイムによって自動的に初期化されます。以下のコード例は、ユーザー定義型の作成について示しています。

```
Const LF_FACESIZE = 32
Private Type MyLogFont
    fHeight As Integer
    fWidth As Integer
    FaceName(LF_FACESIZE) As Byte
End Type

Sub TestFont()
    Dim a As Byte
    Dim f As MyLogFont
    a = f.FaceName(1)
End Sub
```

固定サイズでの FaceName フィールドの宣言方法および初期化方法に注意してください。

Visual Basic .NET で対応するコンストラクトは Structure です。Type コンストラクトとは異なり、Structure のメンバは Initialize メソッドを使用して明示的に初期化する必要があります。上記のコードにアップグレード ウィザードが適用されると、型が構造体に変換され、Initialize メソッドが自動的に追加されます。アップグレード ウィザードによって生成されたコードを以下に示します。

```
Const LF_FACESIZE As Short = 32
Private Structure MyLogFont
    Dim fHeight As Short
    Dim fWidth As Short
    <VBFixedArray(LF_FACESIZE)> Dim FaceName() As Byte
    ' UPGRADE_TODO: この構造体のインスタンスを初期化するには、Initialize を
    ' 呼び出さなければなりません。
    Public Sub Initialize()
        ReDim FaceName(LF_FACESIZE)
    End Sub
End Structure

Sub TestFont()
    Dim a As Byte
    ' UPGRADE_WARNING: 構造体 f の配列は、
    ' 使用する前に初期化する必要があります。
    Dim f As MyLogFont
    a = f.FaceName(1)
End Sub
```

アップグレードされたこのコードは問題なくコンパイルされます。ただし、アプリケーションを実行して、TestFont を呼び出すと、ランタイム エラーが発生します。これは、Initialize メソッドがまだ呼び出されていないため、構造体の配列メンバが初期化されていないためです。この問題を解決するには、以下の修正を TestFont に適用して、Initialize メソッドを呼び出します。

```
Sub TestFont()
    Dim a As Byte
    Dim f As MyLogFont
    f.Initialize()
    a = f.FaceName(1)
End Sub
```

Initialize メソッドが呼び出されると、型の配列メンバが初期化され、ランタイム エラーは発生しなくなります。

Null および IsNull の変更

Visual Basic .NET と Visual Basic 6.0 では、Null および IsNull の動作が異なります。アップグレードウィザードは Null および IsNull の使用を検出します。Null は、Visual Basic .NET で使用できる値の中で最も近い値である System.DBNull.Value にアップグレードされますが、Visual Basic .NET では Null 値の伝播がサポートされません。これによって動作の相違が生じ、ランタイム エラーが発生します。これを以下のコード例に示します。

```
Sub TestNull()
    dbVal = Null
    res = 5 * dbVal
    If IsNull(res) Then
        ' 何らかの処理を実行します。
    End If
End Sub
```

通常は、クエリによりデータベースにアクセスして dbVal を取得することに注意してください。わかりやすくするため、この例では変数に Null 値を直接割り当てています。

アップグレード ウィザードを使用してこのコードをアップグレードすると、以下のコードが生成されます。

```
' UPGRADE_ISSUE: IsNull 関数はサポートされません。
Sub TestNull()
    Dim res As Object
    Dim dbVal As Object
    ' UPGRADE_WARNING: Null/IsNull() の使用が見つかりました。
    ' UPGRADE_WARNING: Could not resolve default property of
    ' object dbVal.
    dbVal = System.DBNull.Value
    ' UPGRADE_WARNING: Could not resolve default property of
    ' object dbVal.
```

```
    ' UPGRADE_WARNING: Could not resolve default property of
    ' object res.
    res = 5 * dbVal
    ' UPGRADE_WARNING: Null/IsNull() の使用が見つかりました。
    If IsDbNull(res) Then
        ' 何らかの処理を実行します。
    End If
End Sub
```

多数の UPGRADE_WARNING コメントがコード内に挿入されていることに注意してください。このコードは問題なくコンパイルされますが、実行するとランタイム エラーが発生します。

この問題を修正するには、Null 値の伝播と、Null チェック関数に依存する条件について、手動で調整する必要があります。以下のコード例では、修正の必要な箇所を太字で示しています。

```
Sub TestNull()
    Dim res As Object
    Dim dbVal As Object
    dbVal = System.DBNull.Value
    If Not IsDBNull(dbVal) Then
        res = 5 * dbVal
    End If
    If IsDBNull(dbVal) Or IsDBNull(res) Then
         ' 何らかの処理を実行します。
    End If
End Sub
```

配列インデックスの変更

Visual Basic 6.0 では、インデックスの下限が 0 以外の配列を作成できます。Visual Basic .NET では、配列インデックスの下限はすべて 0 にする必要があります。

アップグレード ウィザードはすべての配列の下限を 0 にします。ただし、ソース コードが配列の算出サイズに依存している場合、アップグレードされたバージョンと元のバージョンで配列コードの動作が異なる可能性があります。以下のコード例はこの状況を示しています。

```
Function GetArraySize(a As Variant) As Long
    Size = UBound(a) - LBound(a) + 1
    GetArraySize = Size
End Function
Sub testArraySize()
    Dim a(5 To 10)
    If GetArraySize(a) > 7 Then
        ' 何らかの処理を実行します。
    End If
End Sub
```

アップグレードウィザードを使用してこのコードをアップグレードすると、以下のコードが生成されます。

```
Function GetArraySize(ByRef a As Object) As Integer
    Dim aSize As Object
    ' UPGRADE_WARNING: Could not resolve default property of
    ' object size.
    aSize = UBound(a) - LBound(a) + 1
    ' UPGRADE_WARNING: Could not resolve default property of
    ' object size.
    GetArraySize = aSize
End Function
Sub testArraySize()
    ' UPGRADE_WARNING: 配列の下限が 5 から 0 に変更されました。
    Dim a(10) As Object
    Dim res As Integer
    If GetArraySize(a) > 7 Then
        ' 何らかの処理を実行します。
    End If
End Sub
```

元のバージョンのコードでは、配列のサイズが 6 しかないため、If ブロック内のステートメントが実行されません。ただし、自動アップグレードの適用後は、アップグレードされたコード内の配列サイズが 11 に拡張されるため、If ブロック内のステートメントが実行されます。これはほとんどの場合、望ましくない動作となります。

この特殊な問題を修正するためには、元のコードとアップグレードされたコードを詳しく調べて、アプリケーションの動作に影響を与えることなく、元の配列サイズを保持する必要があります。この例では、アップグレードされた配列はインデックス 0 ～ 5 の 6 つの要素を持つ必要があります。この変更に合わせて、配列サイズに依存するすべてのコードを修正します。インデックスの範囲が変更されているため、要素の特定の位置に依存するコードを見直す必要が生じます。配列範囲を調整したコード例を以下に示します。

```
Function GetArraySize(ByRef a As Object) As Integer
    Dim aSize As Object
    aSize = UBound(a) - LBound(a) + 1
    GetArraySize = aSize
End Function
Sub testArraySize()
    Dim a(5) As Object
    Dim res As Integer
    If GetArraySize(a) > 7 Then
        ' 何らかの処理を実行します。
    End If
End Sub
```

配列範囲の変更に対処するために適用できるその他の方法については、第 8 章「一般的に使用される Visual Basic 6.0 言語機能のアップグレード」を参照してください。

Activate イベントの変更

Form の状態を再計算または更新するためのコードを Visual Basic 6.0 の Activate イベントに記述することは、珍しくありません。Visual Basic 6.0 でこのイベントが発生するのは、アプリケーション内でフォームを切り替えるときだけです。そのような場合に Visual Basic .NET では、相当するイベント Activated が発生しますが、他のアプリケーションから切り替えたときにもこのイベントが発生します。これにより、2 つのバージョン間でランタイムにおける相違が生じることになります。以下の Visual Basic 6.0 ソースコードはこの状況を示しています。

```
Private Sub Form_Activate()
    ' コードの更新
End Sub
```

アップグレード ウィザードを使用してこのコードをアップグレードすると、以下のコードが生成されます。

```
' UPGRADE_WARNING: フォーム イベント Form1.Activate には新しい動作が含まれます。
Private Sub Form1_Activated(ByVal eventSender As System.Object, _
            ByVal eventArgs As System.EventArgs) Handles MyBase.Activated
    ' コードの更新
End Sub
```

問題は、Activated イベントが Visual Basic の Activate イベントよりも頻繁に発生することです。他のアプリケーションからフォーカスを切り替えると、必ず Activated イベントが発生します。その結果、元のコードとは動作の相違が生じることになります。

この相違を修正するには、他のアプリケーションからのアクティブ化と、同じアプリケーション内の他のフォームからのアクティブ化を区別する必要があります。最後にアクティブ化されたアプリケーション フォームを格納するグローバル変数を使用すると、これらを区別できます。この変数は、新しいアプリケーション クラスの Public Shared メンバとして実装できます。フォーカスを別のアプリケーションに切り替えてから元のフォームに戻した場合、このグローバル変数の値は変更されません。この値に基づいて、Activated イベント コードをスキップできます。この状況を示すのが以下のコード例です。

```
Friend Class MyAppGlobals
    Public Shared AppActiveForm = ""
End Class

Private Sub Form1_Activated(ByVal eventSender As System.Object, _
            ByVal eventArgs As System.EventArgs) Handles MyBase.Activated
    If Not MyAppGlobals.AppActiveForm = MyBase.Name Then
        ' コードの更新
        MyAppGlobals.AppActiveForm = MyBase.Name
    End If
End Sub
```

ComboBox コントロール イベントの変更

この例は、アプリケーションの元のバージョンとアップグレードされたバージョンで動作の相違が生じる別の状況を示しています。これらの相違によってランタイム エラーが発生する場合があります。

Visual Basic 6.0 では、ComboBox コントロールには、ユーザーがコントロールにテキストを入力したときに発生する Change イベントがあります。Visual Basic .NET では、このイベントに対応する TextChanged が同じ状況で発生します。ただし、このイベントは、対応するフォームが初期化されるとき (フォームが表示されるよりも前) にも発生します。TextChanged イベントが発生し、フォームが読み込まれるときに ComboBox の初期化が完了していないために、ランタイム エラーの原因となる場合があります。

以下の Visual Basic 6.0 のコード例はこの問題を示しています。

```
Dim myClass1 As Class1

Private Sub Combo1_Change()
    myClass1.hello
End Sub

Private Sub Form_Load()
    Set myClass1 = New Class1
End Sub
```

自動アップグレード後のコードを以下に示します。

```
Dim myClass1 As Class1

'UPGRADE_WARNING: Event Combo1.TextChanged may raise when form is
' initialized.
' UPGRADE_WARNING: ComboBox イベント Combo1.Change は、
' 新しい動作をもつ Combo1.TextChanged にアップグレードされました。
Private Sub Combo1_TextChanged(ByVal eventSender As System.Object, _
            ByVal eventArgs As System.EventArgs) Handles Combo1.TextChanged
    myClass1.hello()
End Sub

Private Sub Form1_Load(ByVal eventSender As System.Object, _
            ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    myClass1 = New Class1
End Sub
```

アップグレードされたコードでは、対応するフォームが初期化されるたびに Combo1_TextChanged イベント ハンドラ メソッドが実行されます。この場合、Form1_Load イベント ハンドラで myClass1 がまだ初期化されていないためにランタイム エラーが発生します。この問題を解決するには、対応する Form クラスに新しい属性を作成する必要があります。この属性は、フォームが初期化されるかどうかを示します。InitializeComponent

メソッドの開始時および終了時に、この属性の値を設定する必要があります。修正したコード抜粋を以下に示します。このコードは Form1 クラスに記述する必要があります。

```
Friend Class Form1
    Inherits System.Windows.Forms.Form

    Private initializing As Boolean
    Dim myClass1 As Class1

    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        initializing = True
        ' InitializeComponent に記述されている元のコードを
        ' ここにコピーしてください。
        initializing = False
    End Sub

    Private Sub Combo1_TextChanged(ByVal eventSender As System.Object, _
                ByVal eventArgs As System.EventArgs) Handles Combo1.TextChanged
        If Not initializing Then
            myClass1.hello()
        End If
    End Sub

    Private Sub Form1_Load(ByVal eventSender As System.Object, _
                ByVal eventArgs As System.EventArgs) Handles MyBase.Load
        myClass1 = New Class1
    End Sub
End Class
```

まとめ

アップグレードを問題なく効率的に行うための最良の方法は、実績のあるアップグレード手順を適用することです。アップグレード前の準備作業を入念に行うと、アップグレード プロセスの後期段階での時間と労力を節約できます。アップグレードに関する一般的な問題を理解し、前述したような潜在的な問題をアップグレード開始前に特定することで、それらをスムーズに解決できます。最後に、ツールを活用してアップグレードをできるだけ自動化することにより、時間と労力を節約できます。この章では、アップグレード方法の各ステップについて説明しました。この方法を適用すると、可能な限り最も効率的な手段で、アプリケーションを問題なくアップグレードできます。

詳細情報

Visual Basic 6.0 Code Advisor の詳細については、MSDN の Microsoft Visual Basic Developer Center に掲載されている「Visual Basic 6.0 Code Advisor」(http://msdn.microsoft.com/vbasic/downloads/codeadvisor/default.aspx.) を参照してください。

Visual Basic Upgrade Wizard Companion およびその他のアップグレード サービスの詳細については、ArtinSoft の Web サイト (http://www.artinsoft.com/) を参照してください。

エラー、警告、および問題の詳細については、MSDN の「Visual Basic 6.0 Upgrading Reference」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vboriVisualBasic60CompatibilityReference.asp) の左ペインを参照してください。

6

Visual Basic アップグレード ウィザードについて

Visual Basic アップグレード ウィザードは、Visual Basic 6.0 で作成されたプロジェクトを Visual Basic .NET で記述された一連のファイルにアップグレードするソース コード自動アップグレード ツールです。このツールを使用すると、使用中の以前のアプリケーションをそのまま保持しながら、.NET Framework を十分に活用できます。

このアップグレード ツールは、ソフトウェア開発者がアプリケーションを手動でアップグレードする場合と同様に動作します。つまり、プログラムの構成やステートメント、式、記号、データ型、リテラル値、変数などを含めたソフトウェア作成と変換のさまざまな側面が考慮されます。アップグレード ウィザードは、このデータを利用して、元のソース コードのアルゴリズムや全体的なスタイルを保持しながら、Visual Basic .NET のソース コードを生成します。開発者が理解し、改善できる形でコードがアップグレードされます。

アップグレード ツールの使用

Visual Basic アップグレード ウィザードについて説明する前に、まず Visual Basic アップグレード ツールについて説明しておく必要があります。このアップグレード ツールは、Visual Basic 6.0 プロジェクトを Visual Basic .NET にアップグレードします。アップグレード ツールでは、プロジェクトが新規作成され、元のプロジェクトの各ファイルが新しいプロジェクトにコピーされて、必要に応じてファイルが変更されます。最後に、ツールによって実行された処理と、アップグレードを完了するために実行する必要がある残りの処理について詳しく説明するレポートが生成されます。

アップグレード ツールは、Visual Studio .NET 開発環境に組み込まれたアップグレード ウィザードまたは .NET Framework のコマンド ライン ツールという 2 つの形式で提供されます。アップグレード ウィザードを呼び出すには、Visual Studio .NET で Visual Basic 6.0 プロジェクト ファイル (.vbp) を開きます。これにより、

アップグレード ウィザードが自動的に起動し、順を追ってアップグレード プロセスの説明が表示されます。Visual Studio .NET がインストールされていない場合、またはコマンド ラインで作業をしたい場合には、アップグレード ツールのコマンド ライン バージョンを起動できます。

アップグレード ツールのウィザード バージョンおよびコマンド ライン バージョンの実行方法の詳細については、第 5 章「Visual Basic のアップグレード プロセス」の「Visual Basic アップグレード ウィザードの実行」を参照してください。

メモ : Visual Basic アップグレード ウィザードは、アップグレード プロジェクトで一般的に使用されるツールのバージョンです。この章全般を通じて、*アップグレード ウィザード*はアップグレード ツールを指す名称として使用されています。ただし、この章の内容はウィザード バージョンとコマンド ライン バージョンの両方に適用されます。

アップグレード ウィザードで実行されるタスク

Visual Basic 6.0 プロジェクトでアップグレード ウィザードを使用すると、アップグレード ウィザードにより、元の Visual Basic 6.0 プロジェクトからアップグレードされたファイルをすべて含む新しい .NET プロジェクトが作成されます。表 6.1 は、Visual Basic 6.0 プロジェクトとそれに対応する .NET プロジェクト、または対応する .NET プロジェクトがない場合の代替プロジェクトを示しています。

表 6.1: Visual Basic 6.0 のプロジェクトの種類に対応する Visual Basic .NET プロジェクト

Visual Basic 6.0	Visual Basic .NET
標準 EXE	Windows アプリケーション。
ActiveX DLL	クラス ライブラリ。
ActiveX EXE	クラス ライブラリ。
ActiveX コントロール	Windows コントロール ライブラリ。
ActiveX ドキュメント	直接対応するものはありません。Visual Basic .NET は ActiveX ドキュメントと相互運用可能です。
DHTML アプリケーション	直接対応するものはありません。ASP.NET Web アプリケーションを使用します。
IIS アプリケーション (WebClass)	直接対応するものはありません。ASP.NET Web アプリケーションを使用します。

メモ : ASP.NET Web アプリケーションの詳細については、第 4 章「一般的なアプリケーションの種類」、第 20 章「一般的なテクノロジ シナリオ」、および付録 C「ASP のアップグレードに関する概要」を参照してください。

コードの変更

プロジェクトのアップグレード時には、アップグレード ウィザードにより、次のような方法でコードが変更されます。

- **言語の変更。** Visual Basic .NET に導入された新しいキーワードや言語構造を使用するように、Visual Basic 6.0 コードがアップグレードされます。可読性やロジック構造、コードの所有権はそのまま保持され、機能を実現するために必要なランタイム関数が追加されます。
- **コア関数。** Visual Basic 6.0 のコア関数に対する参照は、可能な限り、.NET Framework の対応するクラスやメソッドにアップグレードされます。対応する .NET 関数がない場合、または使用する関数をアップグレード ウィザードが判断できない場合には、手動で修正できるようにコード ファイルに注記が追加されます。
- **フォーム。** Visual Basic 6.0 のフォームとコントロールは、対応する .NET Windows フォーム コントロールにアップグレードされます。Windows フォーム コントロールの詳細については、この後の「Visual Basic 6.0 オブジェクト」の「フォーム」を参照してください。
- **Web クラス。** Visual Basic 6.0 Web クラスは .NET Web フォームにアップグレードされます。.NET Web フォームの詳細については、この後の「Visual Basic 6.0 の言語要素」の「Web クラス」を参照してください。
- **データ。** データおよび接続オブジェクトは、対応する Visual Basic NET のランタイム呼び出し可能ラッパー (RCW) にアップグレードされます。これらのコンポーネントを ADO.NET にアップグレードするには、追加の作業が必要になります。
- **ユーザー コントロール。** Visual Basic 6.0 ActiveX ユーザー コントロール コンポーネントは .NET ユーザー コントロールにアップグレードされます。.NET ユーザー コントロールの詳細については、この後の「Visual Basic 6.0 オブジェクト」の「ユーザー コントロール」を参照してください。

重複する機能 (DDE など) や、Visual Basic 6.0 ランタイムに依存する機能 (ScaleMode など) はアップグレードされませんが、警告またはエラーとして表示されます。

参照チェック

プロジェクトを Visual Basic .NET にアップグレードする際に、アップグレード ウィザードはプロジェクト内の他のファイルや外部のライブラリ、コンポーネント、COM オブジェクト、ファイルを参照するコードを調べます。(通常、Visual Basic .NET の規則に適合するようにファイル名の拡張子を変更することで) プロジェクト ファイルの名前を変更すると、新しいファイル名と場所が反映されるように、参照先のファイルをすべて自動的に更新します。アップグレード ウィザードでは、Visual Basic .NET プロジェクトで使用できるように、リソース ファイルやバイナリリソースも自動的に更新されます。

アップグレード ウィザードは、型キャストの生成や、メンバ マッピングの識別、アップグレードしたコードの品質の向上を実現するために、プロジェクト ファイル内で参照される外部コンポーネントをすべてインスタンス化して、コンポーネントに関する情報を取得します。このためには、プロジェクトのファイルによって参照される外部コンポーネントが、アップグレードを実行するコンピュータに適切にインストールされている必要があります。

ArtinSoft Visual Basic Upgrade Wizard Companion

アップグレード ウィザードの拡張バージョンである ArtinSoft Visual Basic Upgrade Wizard Companion でも、データベース スキーマ定義ファイルを解析して、フィールド データ型に関する追加情報を取得したり、より質の高い Visual Basic .NET コードを生成したりすることができます。ArtinSoft Visual Basic Upgrade Wizard Companion やその他の機能拡張を利用することで、アップグレード プロセスを自動化し、さらに短縮できます。ArtinSoft Visual Basic Upgrade Wizard Companion は、ArtinSoft が販売およびサポートしています。価格はアプリケーション ソース コードのサイズによって決まります。詳細については、ArtinSoft の Web サイトを参照してください。

アップグレード レポート

アップグレード ウィザードによりプロジェクトのアップグレードが完了すると、アップグレード レポートが生成されます。このレポートを基に、プロジェクトに必要な変更を手動で実行できます。図 6.1 は、アップグレード レポートの一例を示しています。

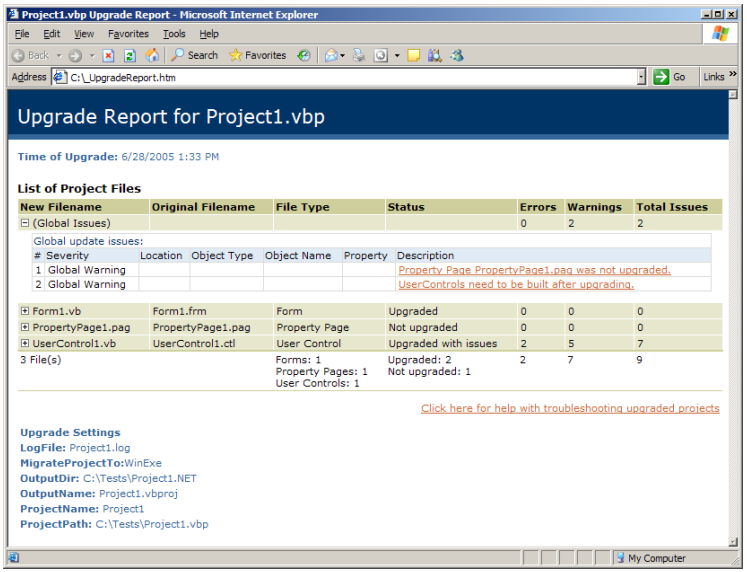

図 6.1

アップグレード レポートの例

アップグレード レポートには、アップグレード プロセスに関する情報や、アップグレード プロセスで発見された問題の一覧が含まれています。これらの問題に対処しないと、プロジェクトをコンパイルすることはできません。アップグレード レポートは HTML 形式で、ソリューション エクスプローラの _UpgradeReport.htm ファイルをダブルクリックすると、Visual Basic .NET 開発環境で表示できます。または、[ファイル] メニューの [ブラウザの選択] をクリックすると、外部の Web ブラウザでも表示できます。

レポートの上部には、全体的な問題と、アップグレードされた各ファイルの一覧が表示されます。各ファイルのセクションを展開すると、対処する必要がある問題の詳細を表示できます。問題ごとに、その問題の重大度、修正する必要があるコードの場所、問題の説明が表示されます。レポートの下部には、アップグレード時に使用された設定や、新しいプロジェクト ファイルの場所などのアップグレードに関する全般的な情報が表示されます。

レポートに加えて、アップグレード後のコードには、変更する必要があるステートメントについて警告するコメントがアップグレード ウィザードによって挿入されます。これらのコメントは、新しい Visual Studio .NET の [タスク一覧] ウィンドウに TO DO タスクとして表示されるため、必要な変更を一目で把握できます。タスク一覧のタスクをダブルクリックすると、関連するコード行が表示されます。アップグレード レポートの各タスクと項目は、オンライン ヘルプのトピックに関連付けられており、コードの変更が必要な理由についての説明と、問題を解決するための手順を参照できます。

サポートされる要素

ここでは、アップグレード ウィザードによって Visual Basic 6.0 の言語構造、ネイティブ ライブラリ、フォーム、および ActiveX オブジェクトがどのように Visual Basic .NET にアップグレードされるかについて詳しく説明します。

Visual Basic 6.0 の言語要素

アップグレード ウィザードは、以下のようなさまざまな Visual Basic 6.0 要素を Visual Basic .NET にアップグレードします。

宣言

アップグレード ウィザードは、Visual Basic .NET の構造に従い、Visual Basic 6.0 宣言のガイドラインを考慮しながら、変数宣言を対応する .NET Framework 型にアップグレードします。

たとえば、以下のような Visual Basic 6.0 コードでは、a と b は暗黙的に Variant データ型で宣言され、c は明示的に 16 ビット整数として宣言されています。

```
Dim a, b, c As Integer
```

Visual Basic .NET では、変数は明示的に宣言する必要があり、Variant データ型は存在しません。アップグレード ウィザードは、.NET Framework で Variant に最も近いデータ型である object データ型を使用して a と b を宣言するためにコードを変更し、Visual Basic .NET で 16 ビット整数と呼ばれる short データ型として c を宣言します。

```
Dim a, b As Object
Dim c As Short
```

ルーチン

プロシージャまたは関数にパラメータを渡すには、参照 (ByRef) または値 (ByVal) を使用する 2 つの方法があります。パラメータが参照によってサブプロシージャに渡される場合、パラメータの値を変更すると、呼び出し側プロシージャによって渡される実際の変数またはオブジェクトに影響します。パラメータが値によって渡される場合、サブプロシージャは元の変数またはオブジェクトのコピーを使用します。したがって、サブプロシージャを変更しても、元の値には影響しません。

パラメータのデータ型が Integer、Long、Boolean、String などの組み込みデータ型の場合、Visual Basic 6.0 では既定で ByRef 方式が使用されます。パラメータが非組み込みデータ型を使用している場合、Visual Basic 6.0 では、既定でパラメータが値によって渡されます。

これとは対照的に、Visual Basic .NET では、特に指定がない限り、すべてのパラメータが値によって渡されます。Visual Basic 6.0 ルーチンのシグネチャにパラメータ キーワードが指定されている場合、アップグレード ウィザードはそれに応じた Visual Basic .NET ルーチンを生成します。Visual Basic 6.0 ルーチンに ByVal または ByRef の指定がない場合、アップグレード ウィザードはパラメータに ByRef を指定します。

以下の Visual Basic 6.0 コードでは、明示的と暗黙的の両方の方法を使用してパラメータが渡されます。

```
Public Sub ByValTestSub(ByVal p1 As String)
    p1 = "Hello World"
End Sub

Public Function TestFunc(p1 As Integer, p2 As Integer) As Integer
    TestFunc = p1 + p2
End Function
```

アップグレード ウィザードは、Visual Basic .NET でも Visual Basic 6.0 と同じようにパラメータが動作するように、参照によって明示的に TestFunc 関数にパラメータを渡します。

```
Public Sub ByValTestSub(ByVal p1 As String)
    p1 = "Hello World"
End Sub

Public Function TestFunc(ByRef p1 As Short, ByRef p2 As Short) As Short
    TestFunc = p1 + p2
End Function
```

Visual Basic 6.0 でパラメータが Optional として宣言されている場合には、パラメータの既定値を指定する必要はありません。たとえば、この Visual Basic 6.0 サブルーチンは、省略可能な String パラメータである p1 を取ります。

```
Public Sub OpTestSub(Optional p1 As String)
End Sub
```

Visual Basic .NET では、省略可能なパラメータを宣言するときには、既定値を指定する必要があります。呼び出し側プロシージャでパラメータの指定がない場合、サブルーチンは変数に既定値を使用します。アップグレード ウィザードは、既定値の指定がない省略可能なパラメータすべてに、自動的に既定値を追加します。

```
Public Sub OpTestSub(Optional ByRef p1 As String = "")
End Sub
```

数値データ型には既定値 0 が使用され、文字列には空の文字列 ("") が使用されます。オブジェクトには、既定値として Nothing が使用されます。

プロパティ

Visual Basic 6.0 では、プロパティは、Property Get、Property Let、および Property Set ステートメントを使用して定義されます。Visual Basic .NET では、これらのステートメントが Get および Set *アクセサ*を使用する新しいプロパティ宣言構文に置き換えられ、プロパティへのアクセスが提供されます。

以下の Visual Basic 6.0 コードでは、Property Get、Property Let、および Property Set ステートメントを使用して、Text という名前のプロパティが公開されます。

```
Private mText As String
Private mObj As Object

'Text プロパティ
Public Property Get Text() As String
    Text = mText
End Property
Public Property Let Text(ByVal Value As String)
    mText = Value
End Property

'Object プロパティ
Public Property Get Obj() as Object
    Obj = mObj
End Property
Public Property Set Obj(ByVal Value As Object)
    Set mObj = Value
End Property
```

アップグレード ウィザードは、これらのステートメントをプロパティごとに 1 つのプロパティ宣言にまとめます。この例では、以下のような 2 つの新しいプロパティ宣言が作成されます。

```
Private mText As String
Private mObj As Object

Public Property Text() As String
    Get
        Return mText
```

```
    End Get
    Set(ByVal Value As String)
        mText = Value
    End Set
End Property

Public Property Obj() As Object
    Get
        Obj = mObj
    End Get
    Set(ByVal Value As Object)
        mObj = Value
    End Set
End Property
```

Visual Basic 6.0 では、Property Let ステートメントと Property Set ステートメントを省略することで、読み取り専用プロパティを作成できます。たとえば、前の例の Text プロパティと Obj プロパティを読み取り専用にするには、以下に示すように Property Set および Property Let コードを削除します。

```
Private mText As String
Private mObj As Object
Private mText_w As String
Private mObj_w As Object

Public Property Get Text() As String
    Text = mText
End Property

Public Property Get Obj() As Object
    Set Obj = mObj
End Property

Public Property Set TextW(value As String)
    mText_w = value
End Property

Public Property Set ObjW(value As Object)
    Set mObj_w = value
End Property
```

Visual Basic .NET では、読み取り専用プロパティは ReadOnly 修飾子を使用して宣言されます。Visual Basic 6.0 の読み取り専用プロパティを Visual Basic .NET にアップグレードすると、アップグレード ウィザードにより、プロパティ宣言に ReadOnly 修飾子が追加されます。書き込み専用プロパティの宣言の場合も同様で、この場合には WriteOnly 修飾子が使用されます。この例では、コードは以下のようになります。

```
Private mText As String
Private mObj As Object
Private mText_w As String
Private mObj_w As Object
```

```
Public ReadOnly Property Text() As String
    Get
        Return mText
    End Get
End Property
Public ReadOnly Property Obj() As Object
    Get
        Obj = mObj
    End Get
End Property
Public WriteOnly Property TextW() As String
    Set(ByVal Value As String)
        mText_w = Value
    End Set
End Property
Public WriteOnly Property ObjW() As Object
    Set(ByVal Value As Object)
        mObj_w = Value
    End Set
End Property
```

イベント ハンドラ

Visual Basic .NET のイベントはデリゲートを基盤としているため、Visual Basic 6.0 のイベントとは処理方法がまったく異なります。

アップグレード ウィザードは、.NET Framework のルールに従って、メソッドのシグネチャを生成し、Visual Basic .NET イベント インフラストラクチャに適合するために必要な Visual Basic キーワードを追加することで、イベントに関する詳細事項を自動的に処理します。

たとえば、以下のように Text プロパティを使用して OnChange イベントを定義する Visual Basic 6.0 クラスの Class1 があるとします。

```
Private mText As String

Public Event OnChange(ByVal Text As String)

Private Sub Change(ByVal Text As String)
    RaiseEvent OnChange(Text)
End Sub

Public Property Get Text() As String
    Text = mText
End Property

Public Property Let Text(ByVal Value As String)
    mText = Value
    Call Change(Value)
End Property
```

Visual Basic 6.0 クライアント アプリケーションがイベントを使用します。アプリケーションには、TextBox コントロールと Button コントロールが含まれており、Instance という名前の Class1 オブジェクトが作成されます。アプリケーションには、フォーム ロード イベント (Form_Load)、ボタン クリック イベント (Command1_Click)、および Class1 変更イベント (Instance_OnChange) が含まれます。ボタンがクリックされると、Class1 のテキスト プロパティが変更され、その結果として Instance_OnChange イベントが発生します。

```
Public WithEvents Instance As Class1

Private Sub Command1_Click()
    Instance.Text = Text1.Text
End Sub
Private Sub Form_Load()
    Set Instance = New Class1
End Sub
Private Sub Instance_OnChange(ByVal Text As String)
   Call MsgBox("Changed to: " + Text)
End Sub
```

アップグレード ウィザードで Class1 を Visual Basic .NET にアップグレードすると、以下のようなコードが生成されます。

```
Option Strict Off
Option Explicit On
Friend Class Class1
    Private mText As String
    Public Event OnChange(ByVal Text As String)

    Private Sub Change(ByVal Text As String)
        RaiseEvent OnChange(Text)
    End Sub

    Public Property Text() As String
        Get
            Return mText
        End Get
        Set(ByVal Value As String)
            mText = Value
            Call Change(Value)
        End Set
    End Property
End Class
```

アップグレード ウィザードは、以下のようにクライアントをアップグレードします。

```
Option Strict Off
Option Explicit On
Friend Class Form1
    Inherits System.Windows.Forms.Form
#Region "Windows Form Designer generated code "
    …
    Public WithEvents Text1 As System.Windows.Forms.TextBox
    Public WithEvents Command1 As System.Windows.Forms.Button
    …
#End Region
#Region "Upgrade Support "
    …
#End Region

    Public WithEvents Instance As Class1

    Private Sub Command1_Click(ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) Handles Command1.Click
        Instance.Text = Text1.Text
    End Sub

    Private Sub Form1_Load(ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) Handles MyBase.Load
        Instance = New Class1
    End Sub

    ' UPGRADE_NOTE: Text は Text_Renamed にアップグレードされました。…
    Private Sub Instance_OnChange(ByVal Text_Renamed As String) Handles _
        Instance.OnChange
        Call MsgBox("Changed to: " & Text_Renamed)
    End Sub
End Class
```

いずれの場合も、アップグレード ウィザードによって Visual Basic 6.0 コードが Visual Basic .NET にアップグレードされ、それ以上の変更は必要ありません。

イベントに関する重要な変更は、パラメータの受け渡しです。Visual Basic 6.0 では、イベント サブルーチンに各パラメータの名前と型が含まれています。Visual Basic .NET では、パラメータは EventArgs オブジェクトに含まれ、このオブジェクトへの参照がイベント処理サブルーチンに渡されます。また、イベント サブルーチンは、そのイベントを起動したオブジェクトへの参照も受け取ります。

ここで、List1 という名前の ListBox コントロールが含まれる Visual Basic 6.0 フォームを例として使用して、イベント処理サブルーチンのパラメータ受け渡しの違いを説明します。List1 がチェック ボックス形式で設定されている場合は、チェックされた項目はイベント ハンドラによって処理されます。これを示すのが以下のコード例です。

```
Private Sub List1_ItemCheck(Item As Integer)
   MsgBox "You checked item: " & Item
End Sub
```

アップグレード ウィザードをこのコードに適用できます。そうすると、新しいイベント ハンドラ形式でアップグレード後のバージョンのコードが作成されます。この結果を示すのが以下のコード例です。

```
' UPGRADE_WARNING: ListBox イベントの List1.ItemCheck の動作が変更されました。
Private Sub List1_ItemCheck(ByVal eventSender As System.Object, _
                ByVal eventArgs As System.Windows.Forms.ItemCheckEventArgs) _
                Handles List1.ItemCheck
   MsgBox("You checked item " & eventArgs.Index)
End Sub
```

元のコードでは、チェックされた項目がパラメータとして直接渡されています。対照的に、アップグレード後のバージョンでは、項目は ItemCheckEventArgs オブジェクト eventArgs のメンバとして渡されます。また、元のバージョンでは必要なパラメータは選択された項目のみでしたが、アップグレード後のバージョンでは、イベント引数オブジェクトと、そのイベントを起動したオブジェクトへの参照が必要になります。

制御構造

Visual Basic .NET 制御構造は、Visual Basic 6.0 の制御構造とは一部がわずかに異なります。たとえば、Wend キーワードは、Visual Basic 6.0 では While ループで終了しますが、Visual Basic .NET では End While になります。アップグレード ウィザードは、Visual Basic 6.0 のフロー ステートメントの機能を保持しながら、ステートメントを自動的にアップグレードします。

以下の Visual Basic 6.0 コードは、If…Then、For…Next、および While…Wend 制御構造を示します。

```
Dim i As Integer
Dim bol As Boolean
Dim value As Integer

bol = True

If bol = True Then
    i = 0
Else
    i = -1
End If
For i = 0 To 10
    value = value + i
Next i

i = 0
While i < 10
    i = i + 1
Wend
```

アップグレード ウィザードは、この制御構造を以下のように Visual Basic .NET にアップグレードします。

```
Dim i As Short
Dim bol As Boolean
Dim value As Short
bol = True

If bol = True Then
    i = 0
Else
    i = -1
End If

For i = 0 To 10
    value = value + i
Next i

i = 0
While i < 10
    i = i + 1
End While
```

モジュールとクラス

アップグレード ウィザードは、モジュールとクラスを完全にサポートします。Visual Basic 6.0 では、モジュール ファイルとクラス ファイルには別々のファイル名拡張子 (モジュールは .bas、クラスは .cls) が付けられ、プロジェクト フォルダ内の特定のフォルダに格納されます。

Visual Basic .NET の場合、モジュール ファイルとクラス ファイルのファイル名拡張子は同じ .vb で、各ファイル内で構造によって区別されます。

アップグレード ウィザードは、ファイル名のアップグレードも自動的に処理します。アップグレード ウィザードは、要素ごとに、特定の構造に基づいて名前を構成します。以下のコード例は、アップグレード ウィザードが名前のアップグレードをどのように処理するかを示します。

```
Module MODULENAME
End Module
```

以下のコードは、クラスの場合の名前のアップグレード処理を示します。

```
Friend Class Class1
End Class
```

インターフェイス

Visual Basic 6.0 では、共通のクラスをインターフェイスとして扱うことができます。以下のような Visual Basic 6.0 クラスの Class1 があるとします。

```
Class1:
Public Sub Test()
    MsgBox "Hello from class 1"
End Sub
```

Class2 は implements キーワードを使用して Class1 を実装します。

```
Class2:
Implements Class1
Public Sub Class1_Test()
    MsgBox "Hello from class 2"
End Sub

Public Sub Test()
    Dim c As Class1
    Set c = New Class1
    c.Test
End Sub
```

アップグレード ウィザードは、Class1 を Visual Basic .NET にアップグレードし、以下に示すように同様の名前のインターフェイスも作成します。

```
Interface _Class1
    Sub Test()
End Interface
Friend Class Class1
    Implements _Class1
    Public Sub Test() Implements _Class1.Test
        MsgBox("Hello from class 1")
    End Sub
End Class
```

アップグレード ウィザードは、クラスとインターフェイスを生成して、Visual Basic 6.0 の動作をレプリケートします。この場合、1 つのオブジェクトをインターフェイスおよび標準のクラスとして使用できます。Class1 を実装し、それを標準のクラスとして使用する Class2 は、以下のようにアップグレードされます。

```
Friend Class Class2
    Implements _Class1
    Public Sub Class1_Test() Implements _Class1.Test
        MsgBox("Hello from class 2")
    End Sub
    Public Sub Test()
        Dim c As _Class1
        c = New Class1
        c.Test()
    End Sub
End Class
```

Web クラス

Visual Basic 6.0 では、WebClass プロジェクト (別名 IIS アプリケーション プロジェクト) を使用して、Active Server Pages (ASP) テクノロジに基づく Web アプリケーションが作成されます。Visual Basic .NET では、ASP.NET Web アプリケーション プロジェクトを使用して、新しい ASP.NET テクノロジに基づく Web アプリケーションが作成されます。

Visual Basic 6.0 WebClass プロジェクトは、アップグレード ウィザードを使用して、ASP.NET Web アプリケーション プロジェクトにアップグレードできます。WebClass プロジェクトのアップグレード プロセスは、他のプロジェクトのアップグレード プロセスと基本的には同じですが、考慮すべき関連の問題がいくつかあります。

WebClass プロジェクトをアップグレードするには、2 つの前提条件があります。

- アップグレード コンピュータにインターネット インフォメーション サービス (IIS) がインストールされ、実行されていること。
- アプリケーションをインストールするための管理者権限があること。

アップグレード ウィザードを使用して WebClass プロジェクトをアップグレードすると、以下のような処理が行われます。

- WebClass プロジェクトのアップグレード時に、アップグレード ウィザードは既定で新しいプロジェクト名の projectname.NET を使用します。projectname は、Visual Basic 6.0 プロジェクトの名前です。この名前は、仮想ディレクトリの命名にも使用されます。この仮想ディレクトリは、IIS でアプリケーションとしても構成され、アプリケーションにとってわかりやすい名前がプロジェクト名になります。
- http://localhost サーバーに仮想ディレクトリ名が既に存在する場合 (アップグレードの問題を解決するために、同じプロジェクトにアップグレード ウィザードを何回も適用すると、名前の重複が起きやすくなります)、アップグレード ウィザードは仮想ディレクトリ名とプロジェクト名の後に番号を追加して、一意性を確保します。
- Visual Basic 6.0 の WebClass プロジェクトを Visual Basic .NET にアップグレードすると、プロジェクトの .asp ファイルが .aspx ファイルにアップグレードされます。ただし、HTML テンプレート ファイル内の .asp ファイルへの参照は自動的には .aspx 参照に変更されません。これは、テンプレート ファイルに WebClass プロジェクトの一部ではない他の .asp ファイルへの参照が含まれている可能性があるためです。
- アップグレード ウィザードは、HTML テンプレート ファイルを新しいプロジェクト ディレクトリにコピーするだけです。他の .html ファイルまたはイメージ ファイルは新しいディレクトリにはコピーされません。
- アップグレード ウィザードでは、HTML ファイルは、既定でコンテンツ ファイルとして Visual Basic .NET プロジェクトに追加されます。WebClass プロジェクトのアップグレード時には、HTML ファイルは埋め込みリソースとして追加されます。アップグレード後に HTML ファイルをプロジェクトに追加する場合は、アプリケーションから HTML ファイルが見えるように、Build Action プロパティを Embedded Resource に設定する必要があります。
- Visual Basic 6.0 コードの Function および Sub プロシージャのスコープ (ProcessTags や Respond) は、WebClass 互換性ランタイムがこれらのプロシージャを実行できるように、Private から Public に変更されます。

- WebClass のプロジェクト、および WebClass プロジェクト内の WebItems と Templates ごとに宣言が追加されます。
- Page_Load イベント プロシージャがプロジェクトに追加され、Visual Basic 6.0 WebClass プロジェクトに関連付けられた WebItems と Templates ごとに WebClass オブジェクトと WebItem オブジェクトが作成されます。
- Page_Load イベント プロシージャには、WebClass 互換性ランタイムの WebClass.ProcessEvents への呼び出しが含まれています。これにより、ランタイムは要求 URL に指定された WebItem を表示できます。このコードは、アップグレード後のプロジェクトに追加される唯一の新しいコードで、Visual Basic 6.0 WebClass ランタイムの基本動作のエミュレートのみを目的として追加されるものです。

メモ : ASP オブジェクトと ASP.NET オブジェクトの間には、プロパティ、メソッド、およびイベントの動作の違いが多数あります。動作の違いの詳細については、第 4 章「一般的なアプリケーションの種類」および付録 C「ASP のアップグレードに関する概要」を参照してください。

遅延バインド

Visual Basic 6.0 と Visual Basic .NET は、いずれも遅延バインドされたオブジェクトをサポートしています。これは、Object データ型として変数を宣言し、実行時にクラスのインスタンスに割り当てる場合の慣例です。ただし、アップグレード プロセスでは、既定のプロパティを解決する場合や、基礎となるオブジェクト モデルが変更され、プロパティやメソッド、イベントをアップグレードする必要が生じた場合に、遅延バインドされたオブジェクトがあると問題が生じることがあります。

たとえば、Label1 というラベルがある Form1 という名前の Visual Basic 6.0 フォームがあるとします。以下の Visual Basic 6.0 のコード例は、ラベルのキャプションを "SomeText" に設定します。

```
Dim o As Object
Set o = Me.Label1
o.Caption = "SomeText"
```

Visual Basic .NET では .NET Framework の Windows フォーム クラスを使用するため、ラベル コントロールの Caption プロパティは Text になります。アップグレード ウィザードは、コードを Visual Basic .NET にアップグレードするときに、すべての Label オブジェクトの Caption プロパティを Text にアップグレードします。ただし、遅延バインドされたオブジェクトには型がないため、アップグレード ウィザードはオブジェクトがどのような目的で使用されるのかや、プロパティを変換する必要があるかどうかについて知ることはできません。このような場合は、アップグレード後に自分でコードを変更する必要があります。

遅延バインドされたオブジェクトを正しくアップグレードするには、2 つの方法があります。

- 事前バインドされたオブジェクトを使用して、以下のように元の Visual Basic 6.0 コードを書き換え、アップグレード ウィザードによって自動的にアップグレードされるようにします。

```
Dim o As Label
Set o = Me.Label1
o.Caption = "SomeText"
```

通常は、一般的な Object データ型ではなく、可能な限り適切なオブジェクト型を使用して変数を宣言する必要があります。

- アップグレード プロセスの後に取得した Visual Basic .NET コードを以下のように変更し、不要な遅延バインドされた変数を削除して、影響を受けたクラス メンバを手動でアップグレードします。

```
Dim o As System.Windows.Forms.Label
Set o = Me.Label1
o.Text = "SomeText"
```

メモ：遅延バインドされた変数の問題を解決するもう 1 つの方法として、ArtinSoft Visual Basic Upgrade Wizard Companion を使用する方法があります。ArtinSoft Visual Basic Upgrade Wizard Companion の型推論機能は、変数、パラメータ、戻り値の最適なデータ型を推測し、Object などの一般的なデータ型の使用を回避します。Object 変数を検出すると、このツールが適切な型を使用して変数を宣言します。したがって、手動による修正は必要ありません。ArtinSoft Visual Basic Upgrade Wizard Companion の詳細については、ArtinSoft の Web サイトを参照してください。

型キャスト

アップグレード ウィザードは、適切な型制限を使用して、型キャスト ステートメントをアップグレードします。たとえば、この Visual Basic 6.0 コードは、16 ビット整数を文字列に変換してから、整数に戻します。

```
Dim i As Integer
Dim s As String

i = 1
s = CStr(i)
i = CInt(s)
```

Visual Basic 6.0 の Integer データ型は、Visual Basic .NET の Short データ型に対応します。アップグレード ウィザードは、以下に示すように、Visual Basic .NET コードの生成時に適切なキャスト ステートメントを使用します。

```
Dim i As Short
Dim s As String

i = 1
s = CStr(i)
i = CShort(s)
```

列挙型

列挙型 (Enum) は、特殊な値型です。列挙型には、名前と、プリミティブ データ型の値を定義する一連のフィールドがあります。

アップグレード ウィザードは、列挙型とその参照を Visual Basic .NET で使用される形式に自動的にアップグレードします。たとえば、以下の Visual Basic .NET コードは、列挙型を宣言および使用します。

```
Public Enum MyEnum
    FirstValue
    SecondValue
    ThirdValue
End Enum
Public Sub TestEnum()
    Dim e As MyEnum
    e = SecondValue
End Sub
```

アップグレード ウィザードは、以下のように、このコードを自動的に Visual Basic .NET に対応するように変更します。

```
Public Enum MyEnum
    FirstValue
    SecondValue
    ThirdValue
End Enum

Public Sub TestEnum()
    Dim e As MyEnum
    e = MyEnum.SecondValue
End Sub
```

ユーザー定義型

Visual Basic 6.0 では、ユーザー定義型 (UDT) を使用して、異なるタイプのデータ項目のグループを作成できます。Visual Basic .NET では、UDT は構造体と呼ばれます。構造体は、1 つ以上のメンバを相互に、または構造体そのものと関連付けます。構造体を宣言すると、複合データ型になり、その型の変数を宣言できます。

UDT とアップグレード ウィザード

アップグレード ウィザードは、自動的に UDT を Visual Basic .NET の構造体にアップグレードします。たとえば、以下の Visual Basic 6.0 コードは、UDT を定義および使用します。

```
Private Type MyType
    x As Integer
    y As Integer
    name As String
End Type
Public Sub TestUDT()
    Dim udt As MyType
    udt.name = "Joe"
    udt.x = 5
    udt.y = 10
End Sub
```

アップグレード ウィザードによって生成される Visual Basic .NET コードでは、Structure キーワードを使用して、UDT を構造体として再定義します。

```
Private Structure MyType
    Dim x As Short
    Dim y As Short
    Dim name As String
End Structure
Public Sub TestUDT()
    Dim udt As MyType
    udt.name = "Joe"
    udt.x = 5
    udt.y = 10
End Sub
```

固定長フィールドを含む UDT のアップグレード

前の例では、ネイティブ データ型フィールドを含む UDT を紹介しました。これらのフィールドのサイズは、プログラミング言語とコンパイル方法によって決まります。通常は元のフィールドに対応する .NET データ型が存在すれば、自動アップグレードは可能ですが、必ずしもそうとは限りません。

UDT には、配列や固定長文字列などの基本データ要素で構成される固定長フィールドが含まれることが多くあります。この種の UDT は通常、第 11 章「文字列操作とファイル操作のアップグレード」の「ユーザー定義データ型を使用した固定長レコードへのアクセス」で説明するように、従来型のフラット ファイルにアクセスする場合に使用されます。

以下の例は、固定長フィールドを含む Visual Basic 6.0 の UDT を示します。

```
Type employee
    name As String * 15
    last_name As String * 20
    department As String * 6
    phone_ext(3) As Long
    salary As Integer
End Type
```

アップグレード ウィザードは、以下のようにこのコードをアップグレードします。

```
Structure employee
    <VBFixedString(15),System.Runtime.InteropServices.MarshalAs( _
    System.Runtime.InteropServices.UnmanagedType.ByValTStr, _
    SizeConst:=15)> Public name As String

    <VBFixedString(20),System.Runtime.InteropServices.MarshalAs( _
    System.Runtime.InteropServices.UnmanagedType.ByValTStr, _
    SizeConst:=20)> Public last_name As String

    <VBFixedString(6),System.Runtime.InteropServices.MarshalAs( _
    System.Runtime.InteropServices.UnmanagedType.ByValTStr, _
    SizeConst:=6)> Public department As String

    <VBFixedArray(3)> Dim phone_ext() As Integer
    Dim salary As Short

    ' UPGRADE_TODO: この構造体を初期化するには、"Initialize" を
    ' 呼び出す必要があります。詳細については、ここをクリックします。
    ' 'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1026"'
    Public Sub Initialize()
        ReDim phone_ext(3)
    End Sub
End Structure
```

データ型が固定長文字列のフィールドは、Visual Basic .NET の構造体で直接宣言することはできません。固定長文字列は、Visual Basic .NET のネイティブ型ではなく、構造体に含めることができないからです。変

数宣言で使用できる 1 つの解決策は、VB6.FixedLengthString 互換性クラスを使用することです。ただし、このクラスはオブジェクトではないため、やはり構造体に含めることはできません。

構造体の問題を解決するには、構造体のフィールドをファイル I/O 用に固定長要素として扱うようにプログラマが指定します。この機能は、.NET のマーシャリング機能に相当し、VBFixedString または VBFixedArray 属性を使用して指定します。これらの属性に渡される引数は、フィールドのサイズを表します。この方法であれば、対応する Visual Basic .NET の構造体を取得できます。前の Visual Basic .NET のコード例は、このマーシャリング属性の使用方法を説明するものです。

メモ:ここで紹介する手法は、ファイル I/O シナリオで固定長文字列と配列を扱う場合にのみ適用できます。相互運用の場合には、固定長フィールドを正しく処理するためには MarshalAs 属性を使用する必要があります。MarshalAs 属性の詳細ついては、第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」を参照してください。

Visual Basic 6.0 ネイティブ ライブラリ (VB、VBA、VBRUN)

Visual Basic 6.0 言語のコア機能の大半を提供する Visual Basic 6.0 ネイティブ ライブラリは、.NET Framework そのものに置き換わりました。.NET Framework では、System 名前空間とその他の空間を通じて、.NET 言語にコア機能を提供します。アップグレード ウィザードは、Visual Basic 6.0 ネイティブ ライブラリへの呼び出しを、対応する .NET Framework オブジェクトおよびクラスへの呼び出しに自動的に置換します。

たとえば、以下の Visual Basic 6.0 コードは、VB、VBA、および VBRUN ライブラリの関数を指定どおりに使用します。

```
Public Sub TestLibraries()
    Dim s As String                  ' Library: VBA
    s = CStr("2005")                 ' Library: VBA

    Dim c As ColorConstants          ' Library: VBRUN
    c = vbBlue                       ' Library: VBRUN

    MsgBox App.EXEName               ' Library: VB
End Sub
```

アップグレード ウィザードによって生成された Visual Basic .NET コードは、.NET Framework のオブジェクトとメソッドを使用します。

```
Public Sub TestLibraries()
Dim s As String                      ' Library: VBA
s = CStr("2005")                     ' Library: VBA

    Dim c As System.Drawing.Color    ' Library: VBRUN
```

```
    c = System.Drawing.Color.Blue      ' Library: VBRUN

    MsgBox(VB6.GetExeName())           ' Library: VB
End Sub
```

Visual Basic 6.0 オブジェクト

ここでは、アップグレード ウィザードが、どのようにフォームやビジュアル コンポーネントをアップグレードするかを説明します。また、Visual Basic 6.0 のグラフィカル コントロールと同等の .NET グラフィカル コントロールの類似点と相違点についても概説します。

フォーム

アップグレード ウィザードは、Visual Basic 6.0 フォーム (拡張子 .frm のファイル) を .NET Framework のフォーム開発システムで、System.Windows.Forms.Form 名前空間のクラスを使用する Windows フォームにアップグレードします。

```
Friend Class Form1
    Inherits System.Windows.Forms.Form
    …
End Class
```

アップグレード後のフォーム ファイルは、他の Visual Basic .NET コード ファイルと同様に、ファイル名に .vb という拡張子が付けられて保存されます。各 Visual Basic 6.0 フォームのプロパティ、メソッド、およびイベント (PME) は、対応する System.Windows.Forms.Form PME にアップグレードされます。たとえば、この Visual Basic 6.0 コードは、ボタンがクリックされると、現在のフォームの 2 つのビジュアル プロパティを変更します。

```
Private Sub Command1_Click()
    Me.Caption = "My Form"
    Me.BackColor = ColorConstants.vbBlue
End Sub
```

アップグレード ウィザードは、自動的にコードをアップグレードし、フォームの既定の外観を Visual Basic .NET の既定に変更します。

```
Option Strict Off
Option Explicit On
Friend Class Form1
    Inherits System.Windows.Forms.Form
#Region "Windows Form Designer generated code "
    …
    ' Windows フォーム デザイナで必要です。
    Private components As System.ComponentModel.IContainer
    Public WithEvents Command1 As System.Windows.Forms.Button

    ' 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。
    ' これは Windows フォーム デザイナを使用して変更できます。
    ' コード エディタは使用しないでください。
```

```
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        Me.components = New System.ComponentModel.Container()
        Me.Command1 = New System.Windows.Forms.Button
        Me.Text = "Form1"
        Me.ClientSize = New System.Drawing.Size(548, 295)
        Me.Location = New System.Drawing.Point(4, 20)
        Me.StartPosition = _
            System.Windows.Forms.FormStartPosition.WindowsDefaultLocation
        Me.Font = New System.Drawing.Font("Arial", 8!, _
            System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, _
            CType(0, Byte))
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.BackColor = System.Drawing.SystemColors.Control
        Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Sizable
        Me.ControlBox = True
        Me.Enabled = True
        Me.KeyPreview = False
        Me.MaximizeBox = True
        Me.MinimizeBox = True
        Me.Cursor = System.Windows.Forms.Cursors.Default
        Me.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.ShowInTaskbar = True
        Me.HelpButton = False
        Me.WindowState = System.Windows.Forms.FormWindowState.Normal
        Me.Name = "Form1"
…
    End Sub
#End Region
#Region "Upgrade Support "
    …
#End Region

    Private Sub Command1_Click(ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) Handles Command1.Click
        Me.Text = "My Form"
        Me.BackColor = System.Drawing.Color.Blue
    End Sub
End Class
```

Visual Basic 2005 の場合 :

Visual Studio 2005 のアップグレード ウィザードは、以前のバージョンのプロパティ、メソッド、およびイベントもサポートします。この追加サポートは、Visual Basic 2005 の新機能の一部を基盤としています。たとえば、Visual Basic 6.0 の Form.Picture プロパティはもともと Form.BackgroundImage からアップグレードされたもので、動作メッセージが異なっていました。BackgroundImageLayout プロパティでは、この Form.Picture プロパティをより正確にアップグレードできます。

リソース

Visual Basic 6.0 では、ファイル名拡張子 .res の付くリソース ファイルをサポートしています。各 Visual Basic 6.0 プロジェクトには、リソース ファイルを 1 つ設定できます。

Visual Basic .NET のリソース ファイルの拡張子は .resx です。リソース ファイルは、XML タグ内のオブジェクトや文字列を指定する XML エントリを構成します。

関連付けられたリソース ファイルがある Visual Basic 6.0 プロジェクトでアップグレード ウィザードを使用すると、Visual Basic の .res ファイルが Visual Basic .NET の .resx ファイルに自動的にアップグレードされます。

フォームのリソース

Visual Basic 6.0 のフォーム バイナリ ファイル (拡張子 .frx のファイル) には、イメージやアイコンなど、メイン プログラム フォームに必要なリソースが含まれています。

Visual Basic .NET では、バイナリ リソースは .resx ファイルにエンコードされます。フォームは、関連するリソース ファイルから自動的にリソースを取得できます。

アップグレード ウィザードは、.frx ファイルを、Form クラスに関連付けられた .resx ファイルにアップグレードします。

ソース コードの再構成

Visual Basic 6.0 では、(MDIForms を含め) すべてのフォームに、Form クラスと同じ名前の事前に宣言されたフォーム ID またはインスタンス変数があります。フォーム ID の宣言は、Visual Basic 6.0 の New キーワードを使用するのとほぼ同じです。つまり、コードの 1 行がフォームを参照し、その値が Nothing の場合、フォームの新しいインスタンスが自動的に作成されます。

```
' まだ作成されていない場合は、インスタンスを作成し、表示します。
Form1.Show
Form1.BackColor = vbBlue
...
set Form1 = Nothing
' 新しいインスタンスを表示します。
Form1.Show
```

アップグレード ウィザードは、DefInstance という名前のパブリック プロパティを作成して、フォームと MDIForms の Visual Basic 6.0 での動作をシミュレートします。前の Visual Basic 6.0 コードの例の場合、アップグレード ウィザードにより、以下のような Visual Basic .NET コードが生成されます。

```
' まだ作成されていない場合は、インスタンスを作成し、表示します。
Form1.DefInstance.Show
Form1.DefInstance.BackColor = vbBlue
...
Form1.DefInstance = Nothing
' 新しいインスタンスを表示します。
Form1.DefInstance.Show
```

Visual Basic 2005 の 場合 :

Visual Basic 2005 では、事前に宣言されたインスタンスとしてフォーム クラス名を使用できます。この変更は、アップグレード ウィザードの Visual Studio 2005 バージョンでサポートされ、前の Visual Basic 6.0 のコード例で使用した場合には以下のような結果が生成されます。

```
' まだ作成されていない場合は、インスタンスを作成し、表示します。
Form1.Show
Form1.BackColor = vbBlue
…
Form1 = Nothing
' 新しいインスタンスを表示します。
Form1.Show
```

また、アップグレード ウィザードは、プロジェクトの各 Form と MDIForm に以下のコードを自動的に追加して、Form の新しいインスタンスの作成をサポートします。

```
#Region " Upgrade Support "
Private Shared m_vb6FormDefInstance As Form1
Private Shared m_InitializingDefInstance As Boolean
Public Shared Property DefInstance As Form1
    Get
        If m_vb6FormDefInstance Is Nothing OrElse m_vb6FormDefInstance.IsDisposed _
            Then
            m_InitializingDefInstance = True
            m_vb6FormDefInstance = New Form1
            m_InitializingDefInstance = False
        End If
        DefInstance = m_vb6FormDefInstance
    End Get
    Set
        m_vb6FormDefInstance = Value
    End Set
End Property
#End Region
```

メモ : "Form1" タイプは、フォーム クラス名に置き換える必要があります。

以下に、SDI フォームをインスタンス化するために必要な Visual Basic .NET コードを示します。

```
Public Sub New()
    MyBase.New

        If m_vb6FormDefInstance Is Nothing Then
            If m_InitializingDefInstance Then
                m_vb6FormDefInstance = Me
            Else
```

```
                Try
                    ' スタートアップ フォームについては、最初に作成されたインスタンスが
                    ' 既定インスタンスになります。
                    If System.Reflection.Assembly. _
                        GetExecutingAssembly.EntryPoint.DeclaringType _
                        Is Me.GetType Then
                        m_vb6FormDefInstance = Me
                    End If
                Catch
                End Try
            End If
        End If

    ' この呼び出しは、Windows フォーム デザイナで必要です。
    InitializeComponent

    ' これが MDI の子の場合は、MDIParent を設定するコードをここに記述します。
    ' これが MDI の子で、プロジェクトの MDIForm で
    ' AuthoShowChildren=True と設定されている場合、フォームを自動的に表示するコードをここに記述します。
End Sub
```

フォームがMDIフォームの場合、Sub Newのコードは少し変わります。

```
Public Sub New()
    MyBase.New
        If m_vb6FormDefInstance Is Nothing Then
            ' この MDI フォームを作成できるインスタンスを 1 つのみに制限します。
            m_vb6FormDefInstance = Me
        End If

    ' この呼び出しは、Windows フォーム デザイナで必要です。
    InitializeComponent
    MDIForm_Initialize_Renamed()     * this call only if required
End Sub
```

アップグレード ウィザードは、すべてのビジュアル コンポーネントとフォームの初期化をInitializeComponentメソッドに移動します。たとえば、Visual Basic 6.0 フォームでは、フォームのキャプションは .frm ファイルに格納されます。

```
Begin VB.Form Form1
    Caption         =   "Form1"
```

アップグレードウィザードは、このコードをInitializeComponentメソッドに移動します。

```
Friend Class Form1
    Inherits System.Windows.Forms.Form
#Region "Windows Form Designer generated code "
…
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        …
        Me.Text = "Form1"
```

```
        …
        End Sub
#End Region
…
End Class
```

Visual Basic 2005 の場合 :

Windows フォームでサポートされる**パーシャル タイプ**では、ユーザー コードは 1 つのファイルに格納され、InItializeComponents や非ユーザー コードはすべて別のファイルに格納されます。これにより、フォームのファイル形式を 2 つの別個のファイルに分割できます。アップグレード ウィザードは、パーシャル タイプを使用するようにフォームをアップグレードします。パーシャル タイプの詳細については、MSDN の「What's New with the Visual Basic Upgrade Wizard in Visual Basic 2005」を参照してください。

測定単位の変換

Visual Basic 6.0 では、ScaleMode プロパティを使用して、既定の twips 単位からフォームの座標系や PictureBox コントロールを変更できます。Visual Basic .NET では、複数の座標系はサポートされておらず、測定はすべてピクセル単位で表す必要があります。プロジェクトを Visual Basic .NET にアップグレードすると、アップグレード ウィザードにより、ビジュアル コンポーネントのデザイン時の座標が twips からピクセルに自動的に変換されます。実行時に ScaleMode プロパティを設定するコードではコンパイル エラーが発生するため、このコードを手動で変更する必要があります。

メモ : アップグレード ウィザードは、ScaleMode プロパティはデザイン時に Twip に設定されているものと想定します。設定が Twip ではない場合、アップグレードは適切に行われず、修正が必要になります。

さらに、アップグレード ウィザードは、Microsoft.VisualBasic.Compatibility.VB6 名前空間のアップグレード メソッドを使用して、コントロールの位置の実行時修正をアップグレードします。たとえば、この Visual Basic 6.0 メソッドは、ボタンの Height プロパティを変更します。

```
Public Sub x()
    Me.Command1.Height = 5
End Sub
```

アップグレード ウィザードは、このコードを以下のように Visual Basic .NET にアップグレードします。

```
Public Sub x()
        Me.Command2.Height = VB6.TwipsToPixelsY(5)
End Sub
```

デザイン時の座標のアップグレードと同様に、アップグレード ウィザードは ScaleMode プロパティはデザイン時に Twip に設定されているものと想定します。Twip に設定されていない場合、メソッドを手動で修正する必要があります。

Visual Basic 6.0 のネイティブ コントロール

.NET Framework には、ほとんどの Visual Basic 6.0 標準ユーザー インターフェイス コントロールに対応するコントロールが用意されていますが、Visual Basic 6.0 コントロールの中には、対応するコントロールが .NET にないものや、名前が異なるものがあります。表 6.2 は、Visual Basic 6.0 ツールボックスの標準コントロールと対応する .NET Framework コントロールを示します。

表 6.2: Visual Basic 6.0 の標準コントロールと対応する .NET Framework のコントロール

Visual Basic 6.0 の標準コントロール	アップグレード ウィザードの出力
PictureBox	System.Windows.Forms.PictureBox。PictureBox コントロールを他のコントロールのコンテナとして使用する場合は、アップグレード ウィザードにより、System.Windows.Forms.Panel にアップグレードされます。
Label	System.Windows.Forms.Label
TextBox	System.Windows.Forms.TextBox
Frame	System.Windows.Forms.GroupBox
CommandButton	System.Windows.Forms.Button
CheckBox	System.Windows.Forms.CheckBox
OptionButton	System.Windows.Forms.RadioButton
ListBox	System.Windows.Forms.ListBox
ComboBox	System.Windows.Forms.ComboBox
HScrollBar	System.Windows.Forms.HScrollBar
VscrollBar	System.Windows.Forms.VScrollBar
Timer	System.Windows.Forms.Timer
DriveListBox	Microsoft.VisualBasic.Compatibility.VB6.DriveListBox
DirListBox	Microsoft.VisualBasic.Compatibility.VB6.DirListBox
FileListBox	Microsoft.VisualBasic.Compatibility.VB6.FileListBox
Shape	該当なし。図形描画には、.NET 共通言語ランタイム (CLR) のクラスを使用します。
Line	該当なし。線の描画には、.NET CLR のクラスを使用します。
Image	System.Windows.Forms.PictureBox
Data	該当なし。Visual Basic .NET のデータ連結の処理方法は Visual Basic 6.0 とは異なります。
OLE	該当なし。
ImageList	System.Windows.Forms.PictureBox

アップグレード ウィザードは、フォームの Shape、Line、Data および OLE コントロールを System.Windows.Forms.Label コントロールにアップグレードし、各ラベルの BackColor プロパティを Red に設定します。

アップグレード ウィザードは、サポートされるビジュアル コンポーネントの機能を Visual Basic .NET にアップグレードします。たとえば、クリック イベント、Label、TextBox、および ComboBox が設定された CommandButton が含まれる Visual Basic 6.0 プロジェクトがあるとします。ボタンがクリックされると、他のコントロールは以下のように変更されます。

```
Private Sub Command1_Click()
    Me.Text1.Text = "Hello World"
    Me.Combo1.AddItem ("Item 1")
    Me.Label1.Caption = "Good bye"
End Sub
```

アップグレード ウィザードは、コンポーネントを Visual Basic .NET にアップグレードし、それに応じてコントロールの変更に使用されるメソッドを変更します。

```
Option Strict Off
Option Explicit On
Friend Class Form1
    Inherits System.Windows.Forms.Form
#Region "Windows Form Designer generated code "
…
    Public WithEvents Combo1 As System.Windows.Forms.ComboBox
    Public WithEvents Command1 As System.Windows.Forms.Button
    Public WithEvents Text1 As System.Windows.Forms.TextBox
    Public WithEvents Label1 As System.Windows.Forms.Label
…
#End Region
#Region "Upgrade Support "
    …
#End Region

    Private Sub Command1_Click(ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) Handles Command1.Click
        Me.Text1.Text = "Hello World"
        Me.Combo1.Items.Add(("Item 1"))
        Me.Label1.Text = "Good bye"
    End Sub
End Class
```

コントロール配列

コントロール配列は、同じ名前、型、およびイベント プロシージャを共有するコントロールのグループのことです。同じコントロール配列でも、要素のプロパティ設定はそれぞれ異なります。

アップグレード ウィザードは、コントロール配列をアップグレードする際に Microsoft.VisualBasic.Compatibility.VB6 名前空間の関数やオブジェクトに依存します。

メモ : Microsoft.VisualBasic.Compatibility 名前空間およびその子の名前空間のクラスは、Visual Basic .NET アップグレード ツールで使用できるように Visual Basic 6.0 から提供されます。ほとんどの場合、これらのクラスが提供する機能は .NET Framework の他のパーツを使用しても実行できます。これらのクラスは、Visual Basic 6.0 のコード モデルが .NET の実装と大幅に異なる場合にのみ必要になります。

表 6.3 は、Visual Basic 6.0 のコントロール配列をアップグレードするために使用する Microsoft.VisualBasic.Compatibility.VB6 名前空間のコントロール配列の一覧です。

表 6.3: Microsoft.VisualBasic.Compatibility.VB6 名前空間で使用可能なコントロール配列

オブジェクト	説明
BaseControlArray	Visual Basic 6.0 コントロール配列をエミュレートするための親クラス。
BaseOcxArray	**ActiveX** コントロールのエミュレートされた配列の親クラス。
ButtonArray	**CommandButton** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
CheckBoxArray	**CheckBox** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
CheckedListBoxArray	**Style** プロパティを **Checked** に設定して、ListBox コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
ComboBoxArray	**ComboBox** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
DirListBoxArray	**DirListBox** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
DriveListBoxArray	**DriveListBox** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
FileListBoxArray	**FileListBox** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
GroupBoxArray	**Frame** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
HScrollBarArray	**HScrollBar** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
LabelArray	**Label** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
ListBoxArray	**ListBox** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
MenuItemArray	**Menu** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
PanelArray	子コントロールを含む **PictureBox** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
PictureBoxArray	**PictureBox** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
RadioButtonArray	**OptionButton** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
TabControlArray	**TabStrip** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
TextBoxArray	**TextBox** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
TimerArray	**Timer** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。
VscrollBarArray	**VScrollBar** コントロールの Visual Basic 6.0 コントロール配列をエミュレートします。

データ環境

Visual Basic 6.0 データ環境モジュールには、ファイル名拡張子 .dsr が付きます。このモジュールには、ADO 接続を作成するための情報と、データにアクセスするためのコマンドが含まれます。

データ環境、その接続、およびコマンドはそれぞれ、Microsoft.VisualBasic.Compatibility.VB6.BaseDataEnvironment、ADODB.Connection、および ADODB.Command 名前空間のクラスを使用して Visual Basic .NET にアップグレードされます。アップグレード ウィザードは、アクティブ デザイナ (.dsr) ファイルを .vb ファイルに変換します。各 .vb ファイルには、Microsoft.VisualBasic.Compatibility.VB6.BaseDataEnvironment から継承されるクラスが含まれます。

データ アクセスの詳細については、第 12 章「データ アクセスのアップグレード」の「Data Environment のアップグレード」を参照してください。

ActiveX

ここでは、アップグレード ウィザードによって、ActiveX コンポーネントとコントロールを使用する Visual Basic 6.0 アプリケーションがどのように Visual Basic .NET にアップグレードされるかについて説明します。

ActiveX コンポーネントのアップグレード

Visual Basic 6.0 アプリケーションで ActiveX コンポーネントを検出すると、アプリケーション ウィザードは、以下の 2 種類のラッパーを各コンポーネントの周囲に作成します。

- **ランタイム呼び出し可能ラッパー** (RCW)。.NET Framework のタイプ ライブラリ インポータ (Tlbimp.exe) は、ActiveX コンポーネントおよびすべての依存アセンブリに対して RCW を作成します。
- **Windows フォーム ラッパー** (WFW)。WFW は、ActiveX コンポーネントのネイティブ プロパティ、メソッド、およびイベントを、System.Windows.Forms.AxHost のプロパティ、メソッド、およびイベントにマージします。これは、System.Windows.Forms.Control から継承され、ActiveX コントロールを完全に機能する Windows フォーム コントロールとして .NET Framework に公開するクラスです。

アップグレード ウィザードは、これらのラッパーを DLL としてプロジェクトに追加します。たとえば、ActiveX: Microsoft カレンダー コントロール (MSCAL.OCX) を Visual Basic 6.0 プロジェクトに追加し、このプロジェクトをアップグレード ウィザードを使用して Visual Basic .NET にアップグレードすると、アップグレード ウィザードにより、AxInterop.MSACAL.dll (WFW) と Interop.MSACAL.dll (RCW) という 2 つのファイルが作成されます。

また、アップグレード ウィザードは、ActiveX コンポーネント、そのプロパティ、メソッド、およびイベントへの参照を自動的にアップグレードして、コンポーネントのプロパティのデザイン時の初期設定を InitializeComponent メソッドに追加します。

この Visual Basic .NET のコード例は、Microsoft カレンダー コントロール (ここでの名前は "Calendar1") を含む 1 つのフォームで Visual Basic 6.0 アプリケーションをアップグレードした後の、アプリケーション ウィザードの出力を示します。

```
Option Strict Off
Option Explicit On
Friend Class Form1
    Inherits System.Windows.Forms.Form
#Region "Windows Form Designer generated code "
    …

' Windows フォーム デザイナで必要です。
    …
    Public WithEvents Calendar1 As AxMSACAL.AxCalendar
    ' 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。
    ' これは Windows フォーム デザイナを使用して変更できます。
    ' コード エディタは使用しないでください。
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        …
        Me.Calendar1 = New AxMSACAL.AxCalendar
        CType(Me.Calendar1, System.ComponentModel.ISupportInitialize).BeginInit()
        …
        Calendar1.OcxState = CType(resources.GetObject("Calendar1.OcxState"), _
            System.Windows.Forms.AxHost.State)
        Me.Calendar1.Size = New System.Drawing.Size(273, 177)
        Me.Calendar1.Location = New System.Drawing.Point(56, 8)
        Me.Calendar1.TabIndex = 0
        Me.Calendar1.Name = "Calendar1"
        Me.Controls.Add(Calendar1)
        CType(Me.Calendar1, System.ComponentModel.ISupportInitialize).EndInit()
    End Sub
#End Region
…
End Class
```

.NET の組み込みコントロールにアップグレードされた ActiveX コントロール

現在のバージョンのアップグレード ウィザードは、Microsoft Tabbed Dialog Control 6.0 ActiveX コンポーネントを .NET Framework コントロールの System.Windows.Forms.TabControl にアップグレードします。

Visual Basic 2005 の場合 :

Visual Basic 2005 バージョンのアップグレード ウィザードでは、この他の ActiveX コントロールでも対応する .NET Framework コントロールにアップグレードできます。

ユーザー コントロール

Visual Basic 6.0 では、ActiveX コントロール プロジェクトとも呼ばれるユーザー コントロールを使用して、ActiveX コントロールを作成します。ユーザー コントロールをコンパイルしたら、Visual Basic 6.0 フォームや Internet Explorer など、ActiveX をサポートする任意のコンテナでホストできます。

Visual Basic .NET では、他のプロジェクトと共有したり、Windows フォーム アプリケーションでホストできる再利用可能なクラスとコンポーネントの作成に、Windows クラス ライブラリ プロジェクトが使用されます。Visual Basic 6.0 の ActiveX DLL プロジェクト テンプレートは、Windows クラス ライブラリ テンプレートに置き換わります。

メモ: Visual Basic 6.0 の ユーザー コントロールとは異なり、Windows クラス ライブラリ コントロールは Internet Explorer ではホストできません。

アップグレード ウィザードは、Visual Basic 6.0 のユーザー コントロールを Windows クラス ライブラリにアップグレードします。プロジェクト全体が ActiveX DLL である場合は、アップグレードは自動的に処理されます。ただし、プロジェクトがユーザー コントロールを含む標準の Visual Basic プロジェクトである場合、アップグレード ウィザードはこれらのコンポーネント用に別個のプロジェクトを作成しません。アプリケーションをアップグレードした後で、クラス ライブラリ プロジェクトを手動で作成し、アップグレード後のアプリケーションから新しいプロジェクトにコンポーネントを移動する必要があります。

Visual Basic 6.0 では、UserControl の値を取得したり、UserControl の値を PropertyBag オブジェクトに保存するのに、ReadProperties イベントと WriteProperties イベントを使用します。Visual Basic .NET では、PropertyBag オブジェクトはサポートされず、ReadProperties イベントと WriteProperties イベントも使用されません。Visual Basic .NET では、PropertyBag 情報を含め、コントロールのシリアル化されたデータを含むプロパティ ページ (.pag ファイルおよび .pax ファイル) はサポートされません。他のサポートされないプロジェクト アイテムと同様に、元の .pag ファイル (存在する場合は .pax ファイルも) はコピー先ディレクトリにコピーされ、None というビルド アクションを使用してプロジェクトに追加されます。**プロパティ ページ**のアップグレードの詳細については、第 9 章「Visual Basic 6.0 フォーム機能のアップグレード」の「プロパティ ページに対する変更の処理」を参照してください。

アップグレード ウィザードは、基本クラスとして System.Windows.Forms.UserControl を使用して、Visual Basic 6.0 ユーザー コントロールの残りの機能を自動的にアップグレードします。式、フロー制御ステートメント、プロパティ、イベント、プロシージャ、および関数は、Visual Basic 6.0 アプリケーションのアップグレードと同じ規則に従ってアップグレードされます。

コンポーネントの詳細ついては、第 4 章「一般的なアプリケーションの種類」、第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」、および第 15 章「MTS アプリケーションと COM+ アプリケーションのアップグレード」を参照してください。

まとめ

アプリケーションを効率的にアップグレードするには、アップグレードを可能な限り自動化するツールを使用する必要があります。

利用できるツールは複数ありますが、最も利用しやすいのは、Visual Basic .NET IDE の一部である Visual Basic アップグレード ウィザードです。このアップグレード ウィザードは、コードのアップグレードのほとんどを

自動的に実行してくれます。また、自動的にアップグレードできないコードの部分を特定し、その問題を解決するための情報を表示できるMSDN Webサイトの参照先を表示します。

アップグレード ウィザードを最大限に利用するには、Visual Basic 6.0の機能の中でも、ウィザードによってサポートされ、自動的にアップグレードされる機能とそうでない機能を把握しておくことをお勧めします。この章では、アップグレード ウィザードによってアップグレードできる機能を取り上げ、ウィザードがアップグレードをどのように実行するかを説明しました。この情報をもとに、アップグレード ウィザードを適用できるように Visual Basic アプリケーションを効率的に準備できます。また、アップグレード後に自分で実行する必要がある作業が何かも理解できます。

アップグレード ウィザードは強力なツールですが、自動的にアップグレードできない機能もあります。以降の章では、アップグレード時に介入が必要な機能の処理に役立つ方法を紹介します。

詳細情報

ArtinSoft Visual Basic Upgrade Wizard Companionの詳細については、ArtinSoftのWebサイト(http://www.artinsoft.com/)を参照してください。

Visual Basic 2005のパーシャル タイプの詳細については、MSDNの「What's New with the Visual Basic Upgrade Wizard in Visual Basic 2005」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/VBUpgrade.asp)を参照してください。

7

一般的に使用される Visual Basic 6.0 オブジェクトのアップグレード

Basic プログラミング言語は、Microsoft GW-BASIC や Microsoft QuickBasic、Microsoft Visual Basic、Microsoft Visual Basic for Applications (VBA) などのさまざまな形式で広く使用されてきました。各製品には、サポートされる型や言語構成要素が異なる独自の Basic プログラミング言語のブランドがあります。これらの製品ではそれぞれ、言語の整理と簡略化が試みられています。

Microsoft Visual Basic .NET では、これまでの機能に代わる独自の型と言語構成要素を追加した新しいバージョンの Basic 言語を提供します。したがって、Visual Basic 6.0 から Visual Basic .NET にアプリケーションをアップグレードすると、アップグレード後のアプリケーションでいくつかの問題が生じる可能性があります。問題によっては、コードを 1 行変更するだけで解決できる場合もあれば、問題が発生した部分のコードを書き換えたり、場合によっては設計を変更する必要がある場合もあります。

同じ問題についてもプログラマによって解決する方法が異なるのと同様に、プログラミングの問題には多数の解決策があります。この章では、問題ごとに、各自のコードに使用できる簡単な解決策を紹介します。ただし、Microsoft .NET Framework では、同じ目的を達成できる別の方法が複数提供されることもしばしばあります。問題によっては、解決に当たる中でもっと適切な方法が見つかる場合があります。ここでは、可能な限り、別の解決策や詳細情報を参照できるリファレンスも併載しています。

一般的な Visual Basic 6.0 アプリケーションは、言語に用意された標準のオブジェクトを使用して構築されています。これらのオブジェクトの中には、Visual Basic .NET で使用が中止されたために、アップグレード時に特別な処理が必要になるものがあります。この章では、Visual Basic .NET でサポートが中止された最も一般的に使用される Visual Basic 6.0 オブジェクトについて説明し、これらのオブジェクトを Visual Basic .NET テクノロジで使用できるようにする方法を解説します。

Visual Basic 2005 の場合 :

Visual Basic .NET 2003 のアップグレード ウィザードでは、ActiveX ラッパーを使用してサードパーティ コンポーネントが変換されます。Visual Basic .NET 2005 のアップグレード ウィザードでは、最も一般的なサードパーティ コンポーネント ライブラリがサポートされるため、ActiveX ラッパーを使用することなく、クラスやコントロールが組み込み .NET コンポーネントにアップグレードされます。

Visual Basic Upgrade Wizard 2005 でサポートされる新しいライブラリと主なコンポーネントは次のとおりです。

- Microsoft Common Dialog Control 6.0: CommonDialog
- Microsoft Internet Controls: WebBrowse
- Microsoft Masked Edit Control 6.0: MaskEdBox
- Microsoft Rich Textbox Control 6.0: RichTextBox
- Microsoft Windows Common Controls 6.0:
  - TreeView
  - StatusBar
  - ProgressBar
  - ImageList
  - ListView
  - ToolBar
- COM+ Services Type Library: ほとんどのクラスをサポート。

さらに、Visual Basic Upgrade Wizard Companion (別名 Visual Basic Companion) では、次のコンポーネントおよび技術に対応するさまざまなレベルのオートメーションがサポートされます。

- ADO から ADO.NET へのアップグレード
- Recordset (ADOR)
- Microsoft Windows Common Controls-2 6.0
- Microsoft Windows Common Controls 5.0
- Microsoft FlexGrid Control 6.0
- Microsoft XML, version 2.6
- ActiveX Threed コントロール
  - SSPanel
  - SSSplitter
  - SSTab
- ComponentOne VSFlexGrid 7.0 (OLEDB および Light)

Visual Basic Upgrade Wizard Companion は、ArtinSoft の Web サイトからダウンロードできます。

App オブジェクトのアップグレード

Visual Basic 6.0 の App オブジェクトは、アプリケーションに関する情報を設定または取得するためのグローバル オブジェクトです。Visual Basic .NET には App オブジェクトに直接対応するオブジェクトはありませんが、ほとんどのプロパティを Microsoft .NET Framework の対応するプロパティにマップできます。たとえば、App オブジェクトのバージョン情報プロパティは、Visual Basic .NET ではアセンブリ属性に置き換えられています。

Visual Basic 6.0 では、[*プロジェクトのプロパティ*] ダイアログ ボックスでバージョン情報を設定できます。このダイアログ ボックスを表示するには、[*プロジェクト*] メニューをクリックして、[*プロジェクトのプロパティ*] をクリックします (*プロジェクト* はプロジェクトの名前です)。バージョン情報を設定するには、[Make] タブをクリックして、[Version Number] 領域に適切な値を入力します。

Visual Basic .NET では、アセンブリ属性を使用します。これを設定するには、変換済みのプロジェクトの AssemblyInfo ファイルを編集します。

たとえば、PictureBox と Label というコントロールが 1 つずつ含まれる Visual Basic 6.0 フォームがあるとします。プロジェクトのバージョンを指定するすべてのプロパティが設定済みであると想定します。フォームの読み込み時には、メジャー、マイナ、リビジョン番号、および実行可能ファイルのパスなどのアプリケーションのバージョンに関する情報を取得するために App オブジェクトが使用されます。これを示すのが以下のコード例です。

```
Private Sub Form_Load()
    If App.PrevInstance = True Then
        MsgBox ("The application is already running!")
        End
    End If

    ' フォーム ラベルにアプリケーションのバージョン番号を表示します。
    lblVersion.Caption = "Version " & App.Major & "." & App.Minor & _
                         "." & App.Revision

    ' 実行中のアプリケーションへのパスを取得することで、アプリケーションの
    ' フォルダに格納されたイメージ ファイルを表示します。
    Picture1.Picture = LoadPicture(App.Path & "\Logo.jpg")
End Sub
```

このコードで Visual Basic アップグレード ウィザードを使用すると、手動で対応する必要があるアップグレードの問題が生じます。Visual Basic .NET には App オブジェクトに直接対応するオブジェクトがないため、アップグレード ウィザードではこのオブジェクトを参照するコードの一部を変換できません。この例の場合、アップグレード ウィザードでは Revision プロパティを変換できないため、App.Revision を使用するコードは変更されないままになります。代わりに、アップグレード ウィザードは、コードの問題が生じた部分に、問題に関するコメントを挿入します。このコードにアップグレード ウィザードを適用した結果は、以下のようになります。

```
' UPGRADE_ISSUE: App プロパティ App.Revision はアップグレードされませんでした。
lblVersion.Text = "Version " & _
    System.Diagnostics.FileVersionInfo.GetVersionInfo( _
        System.Reflection.Assembly.GetExecutingAssembly.Location).FileMajorPart & _
        "." & System.Diagnostics.FileVersionInfo.GetVersionInfo( _
        System.Reflection.Assembly.GetExecutingAssembly.Location).FileMinorPart & _
        "." & App.Revision
```

メモ：このコード例は、読みやすくするために、アップグレード ウィザードによって生成されたコードを修正したものです。このコード例ではコメントとコードを複数行にわたって表示していますが、実際にはコメントとコードはそれぞれ 1 行に生成されます。これは、コードのコンパイルには一切影響しません。

アップグレード ウィザードによって生成されたコードでは、App.Revision プロパティ参照を変換できないため、コードはコンパイルされません。このエラーを修正するには、2 つの方法があります。1 つは、アップグレードできなかったプロパティをコードから削除することです。この方法を用いてコード例から App.Revision を削除すると、コードは以下のようになります。

```
Private Sub Form1_Load(ByVal eventSender As System.Object, _
                       ByVal eventArgs As System.EventArgs) _
                       Handles MyBase.Load
    If (UBound(Diagnostics.Process.GetProcessesByName( _
        Diagnostics.Process.GetCurrentProcess.ProcessName)) > 0) Then
         MsgBox("The application is already running!")
         End
    End If

    lblVersion.Text = "Version " & _
        System.Diagnostics.FileVersionInfo.GetVersionInfo( _
        System.Reflection.Assembly.GetExecutingAssembly.Location _
        ).FileMajorPart & _
        "." & System.Diagnostics.FileVersionInfo.GetVersionInfo( _
        System.Reflection.Assembly.GetExecutingAssembly.Location _
        ).FileMinorPart

    ' 実行中のアプリケーションへのパスを取得することで、
    ' アプリケーションのフォルダに格納された
    ' イメージ ファイルを表示します。
    Picture1.Image = System.Drawing.Image.FromFile(VB6.GetPath & "\Logo.jpg")
End Sub
```

この方法では、アプリケーションは構築できますが、機能の一部が失われることになります。もう 1 つの方法は、目的の機能に代わるコンポーネントを Microsoft .NET Framework で探すことです。前の例の場合、App.Revision プロパティの問題は、Application オブジェクトの ProductVersion プロパティを使用することで修正できます。このプロパティは、アプリケーションの すべてのバージョン番号を 0.0.0.0 形式で返します。代わりのコードは、以下のとおりです。

```
lblVersion.Text = "Version " & Application.ProductVersion
```

ProductVersion プロパティは、アプリケーションのすべてのバージョン番号を返すため、アップグレード ウィザードによって既に変換されていたコードの一部も削除されています。

1 つの整数としてリビジョン値にアクセスするには、表 7.1 に示すメカニズムを使用します。

Visual Basic .NET に対応する機能がある App オブジェクトのプロパティおよびメソッドの包括的な一覧については、MSDN の『Visual Basic Concepts』の「App Object Changes in Visual Basic .NET」を参照してください。表 7.1 は、App オブジェクトのプロパティとメソッドと、これらと同等の機能を実現するための Visual Basic .NET の対応する機能をまとめたものです。

表 7.1: App オブジェクトのプロパティとメソッドと、Visual Basic .NET の対応する機能

Visual Basic 6.0	Visual Basic .NET の対応する機能
Comments	System.Diagnostics.FileVersionInfo.GetVersionInfo(System.Reflection.Assembly.GetExecutingAssembly.Location).Comments
CompanyName	(System.Reflection.Assembly.GetExecutingAssembly.Location).CompanyName
EXEName	VB6.GetEXEName()
FileDescription	System.Diagnostics.FileVersionInfo.GetVersionInfo(System.Reflection.Assembly.GetExecutingAssembly.Location).FileDescription
HelpFile	**HelpFile** プロパティに対応する Visual Basic .NET の機能の詳細については、第 16 章「アプリケーションの完成」の「統合ヘルプのアップグレード」を参照してください。
HInstance	VB6.GetHInstance()
LegalCopyright	System.Diagnostics.FileVersionInfo.GetVersionInfo(System.Reflection.Assembly.GetExecutingAssembly.Location).LegalCopyright
LegalTrademarks	System.Diagnostics.FileVersionInfo.GetVersionInfo(System.Reflection.Assembly.GetExecutingAssembly.Location).LegalTrademarks
LogEvent メソッド **LogMode** **LogPath**	Visual Basic .NET へのログインは、イベント ログによって処理されます。この機能は、**System.Diagnostics.EventLog** クラスを使用することで代用できます。
Major	**System.Diagnostics.FileVersionInfo.GetVersionInfo(System.Reflection.Assembly.GetExecutingAssembly.Location).FileMajorPart** **メモ**: Visual Basic .NET では、バージョン番号の形式が異なります。
Minor	**System.Diagnostics.FileVersionInfo.GetVersionInfo(System.Reflection.Assembly.GetExecutingAssembly.Location).FileMinorPart** **メモ**: Visual Basic .NET では、バージョン番号の形式が異なります。

Visual Basic 6.0	Visual Basic .NET の対応する機能
NonModalAllowed	これは、ActiveX DLL ファイルに関係する読み取り専用プロパティでした。.NET 共通言語ランタイムは、この動作に自動的に対応するため、このプロパティは削除してもかまいません。
OleRequestPendingMsgText **OleRequestPendingMsgTitle** **OleRequestPendingTimeout** **OleServerBusyMsgText** **OleServerBusyMsgTitle** **OleServerBusyRaiseError** **OleServerBusyTimeout**	これらのプロパティは、OLE オートメーションに関連するものです。Visual Basic .NET では OLE オートメーションはサポートされません。対応する機能については、第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」を参照してください。
Path	VB6.GetPath
PrevInstance	UBound(Diagnostics.Process.GetProcessesByName(Diagnostics.Process.-GetCurrentProcess.ProcessName)) > 0
ProductName	System.Diagnostics.FileVersionInfo.GetVersionInfo(System.Reflection.Assembly.GetExecutingAssembly.Location).ProductName
RetainedProject	Visual Basic .NET では、メモリにプロジェクトを保持することはできません。したがって、このプロパティは削除するか、**False** リテラルに置き換える必要があります。
Revision	**Major** および **Minor** と共にこのプロパティを使用する場合には、すべての参照を **Application.ProductVersion** に置き換えることができます。 このプロパティを単独で使用する場合には、番号を **System.Diagnostics.FileVersionInfo.GetVersionInfo(System.Reflection.Assembly.GetExecutingAssembly.Location).FileBuildPart** に置き換えることができます。 これらを呼び出すことで、バイナリ実行可能ファイルに格納された製品のバージョン情報を取得できます。 アセンブリファイルのリビジョン番号を取得するには、**System.Reflection.AssemblyName.GetAssemblyName(System.Reflection.Assembly.GetExecutingAssembly.Location).Version.Revision** を使用します。 **メモ :** Visual Basic .NET では、バージョン番号の形式が異なります。
StartLogging メソッド	Visual Basic .NET へのログインは、イベント ログを介して処理されます。この機能は、**System.Diagnostics.EventLog** クラスを使用することで代用できます。
StartMode	Visual Basic 6.0 では、このプロパティはアプリケーションを ActiveX コンポーネントとして起動するために使用されていました。ActiveX コンポーネントの作成は、Visual Studio .NET ではサポートされていません。この動作をシミュレートするには、メイン フォームのコンストラクタに省略可能なパラメータを追加して、アプリケーションがユーザーによって直接読み込まれる場合の既定値に 0 を使用します。

Visual Basic 6.0	Visual Basic .NET の対応する機能
TaskVisible	この動作をシミュレートするには、**System.Windows.Forms.Form.ShowInTaskBar** を使用するか、タスク一覧に表示されない Windows サービスまたはコンソール アプリケーションにプロジェクトを変更します。
ThreadID	Visual Basic .NET ではスレッド モデルは異なりますが、このプロパティは **AppDomain.GetCurrentThreadId()** メソッドに置き換えることができます。
Title	**System.Reflection.Assembly.GetExecutingAssembly.GetName.Name Set** にはマップしません。Visual Basic .NET では、Microsoft Windowsの [タスクリスト] に表示される名前は、フォームの **Text** プロパティです。
UnattendedApp	このプロパティへの参照は削除してください。Visual Basic .NET の無人アプリケーションには、コンソール アプリケーション プロジェクトを選択してください。

Visual Basic 2005 の場合 :

Visual Studio 2005 には、新しい My 名前空間が含まれています。この名前空間では、最も一般的なプログラミング シナリオを最も簡単に実装できるように設計された、完全な機能を備えたメンバとサービスのセットが提供されます。Visual Basic 6.0 の App オブジェクトの追加サポートは、My.Application 名前空間を通じて得ることができます。Visual Studio .NET のこのバージョンに含まれるアップグレード ウィザードは、CompanyName、EXEName、FileDescription、LegalCopyright、LegalTrademarks、LogEvent、LogPath、Major、Minor などの該当する App メンバをすべて自動的にアップグレードします。
また、My 名前空間では、コンピュータおよびリソース オブジェクトならびに関数を処理するために機能が公開されます。対応するクラスは、My.Computer と My.Resources に含まれています。

たとえば、Visual Basic Upgrade Wizard 2005 が実行する変換処理の 1 つをここで紹介します。

以下は、元の Visual Basic 6.0 のコードです。

```
...
labelText = LoadResString(1)
...
```

以下は、Visual Basic Upgrade Wizard 2003 で生成されたコードです。

```
...
labelText = VB6.LoadResString(1)
...
```

以下は、Visual Basic Upgrade Wizard 2005 で生成されたコードです。

```
...
labelText = My.Resources.str1
...
```

Screen オブジェクトのアップグレード

App オブジェクトと同様に、Visual Basic 6.0 の Screen オブジェクトも、グローバル アプリケーション プロパティを取得および (場合によっては) 設定するためのプロパティとメソッドを提供します。Screen オブジェクトを使用し、MousePointer プロパティを設定して、モデル フォームが表示されているときに砂時計のマウス ポインタを表示したり、アプリケーションのアクティブ コントロールまたはアクティブ フォームを取得したりすることができます。Visual Basic .NET には Screen オブジェクトに直接対応するオブジェクトはありませんが、ほとんどのプロパティを .NET Framework の対応するプロパティにマップできます。

ここでは、マウス ポインタの変更、アクティブ フォームの最大化、および指定の Textbox コントロールがアクティブになった時点でのコントロールのテキストの変更に、Screen オブジェクトを使用している Visual Basic 6.0 の例を使用します。

```
Screen.MousePointer = vbHourglass
Screen.ActiveForm.WindowState = vbMaximized
If Screen.ActiveControl.Name = "Command1" Then
    Screen.ActiveControl.Caption = "Maximized"
End If
Screen.MousePointer = vbDefault
```

アップグレード ウィザードを適用すると、Screen コードの一部のプロパティが変更されずに残り、以下のようなアップグレード警告が表示されます。

```
' UPGRADE_WARNING: Screen プロパティ Screen.MousePointer
' には新しい動作が含まれます。
System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.WaitCursor
System.Windows.Forms.Form.ActiveForm.WindowState = System.Windows.Forms.FormWindowState.Maximized
' UPGRADE_ISSUE: コントロール名は、汎用名前空間 ActiveControl
' 内にあるため、解決できませんでした。
If VB6.GetActiveControl().Name = "txtState" Then
    ' UPGRADE_ISSUE: コントロール キャプションは、汎用名前空間 ActiveControl
    ' 内にあるため、解決できませんでした。
    VB6.GetActiveControl().Text = "Maximized"
End If
' UPGRADE_WARNING: Screen プロパティ Screen.MousePointer には
' 新しい動作が含まれます。
System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.Default
```

明らかになった問題の数は多くても、アップグレード後のコードを修正して問題を解決することは難しくありません。警告で指摘された問題のいくつかは解決できます。また、一部の指示は Microsoft .NET Framework のプロパティに置き換えることができます。たとえば、VB6.GetActiveControl の参照は、Me.ActiveControl に置き換えることができます。このプロパティは、.NET Form クラスの先祖である System.Windows.Forms.ContainerControl クラスに定義されています。このプロパティには、Me または

System.Windows.Forms.Form.ActiveForm を通じてアクセスできます。結果として、Form オブジェクトが返されます。修正後のコードは、以下のようになります。

```
System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.WaitCursor
System.Windows.Forms.Form.ActiveForm.WindowState = _
    System.Windows.Forms.FormWindowState.Maximized

If Me.ActiveControl.Name = "txtState" Then
    Me.ActiveControl.Text = "Maximized"
End If
System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.Default
```

Screen オブジェクトの変更の詳細については、MSDN の『Visual Basic Concepts』の「Screen Object Changes in Visual Basic .NET」を参照してください。

表 7.2 は、Screen オブジェクトのプロパティの包括的な一覧と、Visual Basic .NET の対応する機能をまとめたものです。

表 7.2: Screen オブジェクトのプロパティと、Visual Basic .NET の対応する機能

Visual Basic 6.0	Visual Basic .NET の対応する機能
ActiveControl	System.Windows.Forms.Form.ActiveForm.ActiveControl
ActiveForm	System.Windows.Forms.Form.ActiveForm
FontCount	System.Drawing.FontFamily.Families.Length
Fonts	System.Drawing.FontFamily.Families
Height	System.Windows.Forms.Screen.PrimaryScreen.Bounds.Height

MouseIcon	この機能は、**Cursor** オブジェクトを宣言し、**MouseIcon** のパスで初期化することで、手動でアップグレードする必要があります。
	以下のサンプル コードを参照してください。
	
	`' Visual Basic 6.0`
	`Screen.MouseIcon = _`
	`  LoadPicture("C:\WINDOWS\Cursors\3dwns.cur")`
	
	`' Visual Basic .NET`
	`Me.Cursor = New Cursor("C:\WINDOWS\Cursors\3dwns.cur")`
	
	詳細については、第 9 章「Visual Basic 6.0 フォーム機能のアップグレード」の「Dealing with Changes to the MousePointers Property」を参照してください。

MousePointer	System.Windows.Forms.Form.ActiveForm.Cursor.Current
TwipsPerPixelX	VB6.TwipsPerPixelX
TwipsPerPixelY	VB6.TwipsPerPixelY
Width	**System.Windows.Forms.Screen.PrimaryScreen.Bounds.Width**

Printer オブジェクトのアップグレード

Visual Basic 6.0 では、アプリケーションとシステム プリンタの対話は Printer オブジェクトを通じて可能になります。Printer オブジェクトには、テキスト、行、およびイメージを印刷するための関数が含まれています。ただし、.NET Framework の印刷モデルは、Visual Basic 6.0 の印刷モデルとは大きく異なるため、Printer オブジェクトの使用は中止されました。さいわい、Printer オブジェクトの機能は、System.Drawing.Printing 名前空間に含まれるクラス (特に PrintDocument クラス) で実現できます。

Visual Basic 6.0 の印刷機能をアップグレードするには、主に 2 つの方法があります。1 つは、Printer オブジェクトのメンバを PrintDocument クラスで提供される対応する機能に置き換えることです。通常は、機能の再実装が必要になります。これについては、次のコード例で説明します。もう 1 つの方法は、Visual Basic .NET の PrintDocument クラスに基づいて、.NET に独自の Printer クラスを作成することです。この方法では、.NET の PrintDocument クラスの機能に、Visual Basic 6.0 の Printer オブジェクトで提供される機能の名前をマスクすることができます。この方法についても、後で例を示して説明します。

最初の方法を適用するには、印刷コードを再実装する必要があります。以下の Visual Basic 6. 0 のコード例は、異なるフォント設定で 4 行のテキストを印刷します。

```
Private Sub Form_Load()
    With Printer
        Printer.Print "Normal Line"
        .FontBold = True
        Printer.Print "Bold Line"
        .FontItalic = True
        Printer.Print "Bold and Italic Line"
        .FontBold = False
        .FontItalic = False
        Printer.Print "Second Normal Line"
        .EndDoc
    End With
End Sub
```

このコードは、以下に示すように、PrintDocument クラスのメソッドを使用して、Visual Basic .NET に実装できます。

```
Dim WithEvents prn As New Printing.PrintDocument

Private Sub Form1_Load(ByVal eventSender As System.Object, _
ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    prn.Print()
End Sub

Private Sub prn_PrintPage(ByVal sender As Object, _
ByVal e As System.Drawing.Printing.PrintPageEventArgs) Handles prn.PrintPage
    Dim currFont As Font
```

```
    Dim yPos As Integer
    yPos = 1
    With e.Graphics
        currFont = New Font("Arial", 10, FontStyle.Regular)
        .DrawString("Normal Line", currFont, Brushes.Black, 1, yPos)
        yPos = yPos + currFont.GetHeight

        currFont = New Font("Arial", 10, FontStyle.Bold)
        .DrawString("Bold Line", currFont, Brushes.Black, 1, yPos)
        yPos = yPos + currFont.GetHeight

        currFont = New Font("Arial", 10, FontStyle.Bold Or FontStyle.Italic)
        .DrawString("Bold and Italic Line", currFont, Brushes.Black, 1, yPos)
        yPos = yPos + currFont.GetHeight

        currFont = New Font("Arial", 10, FontStyle.Regular)
        .DrawString("Second Normal Line", currFont, Brushes.Black, 1, yPos)
    End With
End Sub
```

コードの構造が変更され、描画メソッドの数が増えています。.NET Framework では、描画操作の制御と機能が拡張されていますが、アップグレードの観点から見ると、学習曲線があるため、印刷機能のすべてを手動で再実装するには非常に労力がかかります。

Printer オブジェクトのメンバの対応する機能の詳細については、表 7.3 と表 7.4 を参照してください。Visual Basic .NET から PrintDocument コンポーネントを使用して印刷する手順については、MSDN の「Printing with the PrintDocument Component」を参照してください。

もう 1 つのアップグレード方法では、独自の Printer クラスを作成する必要があります。これにより、複数の描画メソッドと、グラフィック オブジェクト (Circle や Line など) のコレクションを統合して、Print メソッドまたは EndDoc メソッドが呼び出されたときにこれらのオブジェクトの印刷に使用される座標を格納できます。PrintDocument クラスの Print メソッドが呼び出されると、PrintPage イベントが発生します。このイベントを使用して、PrinterClass のコレクションに格納されたすべてのオブジェクトとテキストを描画するようにアプリケーションに指示できます。以下の Visual Basic .NET のコードは、このような Printer クラスを作成するための方法を示します。

```
Imports Microsoft.VisualBasic.Compatibility
' Microsoft.VisualBasic をインポートします。
Public Class PrinterClass
    Public Enum FillStyleConstants As Short
        vbFSSolid = 0
        vbFSTransparent = 1
    End Enum
    ' スケール モード定数
    Public Enum ScaleModeConstants As Short
        vbTwips = 1
```

```
    vbPixels = 3
End Enum
Private Structure LineInfo
    Dim pen As System.Drawing.Pen
    Dim p1, p2 As Drawing.Point
End Structure
Private Structure RectangleInfo
    Dim pen As System.Drawing.Pen
    Dim rec As Drawing.Rectangle
    Dim FillStyle As FillStyleConstants
    Dim FillColor As System.Drawing.Color
End Structure
Private Structure PageInfo
    Public Lines() As LineInfo
    Public Circles() As RectangleInfo
End Structure
Private Pages() As PageInfo
Private PageIndex = 0
Private WithEvents InnerPrinter As New System.Drawing.Printing.PrintDocument
Private objPen As New System.Drawing.Pen(System.Drawing.Brushes.Black)
Private intCurrentX As Double = 20
Private intCurrentY As Double = 20
Public ScaleMode As ScaleModeConstants = ScaleModeConstants.vbTwips
Public FillColor As Drawing.Color
Public FillStyle As FillStyleConstants = FillStyleConstants.vbFSTransparent

Public Sub New()
    ReDim Pages(0)
    Pages(0) = New PageInfo
End Sub

' Printer オブジェクトに送られた印刷操作を終了し、
' 印刷デバイスまたはスプーラにドキュメントを解放します。
Public Sub EndDoc()
    Dim j As Integer
    For j = 0 To Pages.Length - 1
        PageIndex = j
        InnerPrinter.Print()
    Next
End Sub
Public Sub Circle(ByVal P As Drawing.Point, ByVal radius As Double, _
                  Optional ByVal [Step] As Boolean = False)
    Circle(P, radius, objPen.Color, [Step])
End Sub
Public Sub Circle(ByVal P As Drawing.Point, ByVal radius As Double, _
                  ByVal Color As Drawing.Color, _
                  Optional ByVal [Step] As Boolean = False)

    Dim diameter As Double

    If [Step] = True Then
        P.X = P.X + CurrentX
        P.Y = P.Y + CurrentY
    End If
```

```
    diameter = radius * 2

    ' CurrentX プロパティと CurrentY プロパティを移動します
    CurrentX = P.X + radius
    CurrentY = P.Y + radius

    P.X = ConvertToPixelsX(P.X)
    P.Y = ConvertToPixelsY(P.Y)
    diameter = ConvertToPixelsX(diameter)

    If IsNothing(Pages(PageIndex).Circles) Then
        ReDim Preserve Pages(PageIndex).Circles(0)
    Else
        ReDim Preserve Pages(PageIndex).Circles(Pages(PageIndex).Circles.Length)
    End If

    Pages(PageIndex).Circles(Pages(PageIndex).Circles.Length - 1).rec = _
        New Drawing.Rectangle(P.X, P.Y, diameter, diameter)
    Pages(PageIndex).Circles(Pages(PageIndex).Circles.Length - 1).pen = objPen
    Pages(PageIndex).Circles(Pages(PageIndex).Circles.Length - 1).FillColor = _
        FillColor
    Pages(PageIndex).Circles(Pages(PageIndex).Circles.Length - 1).FillStyle = _
        FillStyle
End Sub
Private Sub Printer_PrintPage(ByVal sender As Object, _
            ByVal e As System.Drawing.Printing.PrintPageEventArgs) _
            Handles InnerPrinter.PrintPage
    Dim i As Integer
    ' すべての円を描画します
    If IsNothing(Pages(PageIndex).Circles) = False Then
        For i = 0 To Pages(PageIndex).Circles.Length - 1
            Dim x, y As Integer
            e.Graphics.DrawEllipse(Pages(PageIndex).Circles(i).pen, _
                Pages(PageIndex).Circles(i).rec)
            If Pages(PageIndex).Circles(i).FillStyle = _
                     FillStyleConstants.vbFSSolid Then
                e.Graphics.FillEllipse( _
                    New Drawing.SolidBrush( _
                    Pages(PageIndex).Circles(i).FillColor), _
                    Pages(PageIndex).Circles(i).rec)
            End If
        Next
    End If
End Sub
Public Property CurrentX() As Double
    Get
        Return ConvertToPixelsX(intCurrentX)
    End Get
    Set(ByVal Value As Double)
        intCurrentX = ConvertToPixelsX(Value)
    End Set
```

```
    End Property
    Public Property CurrentY() As Double
        Get
            Return ConvertToPixelsY(intCurrentY)
        End Get
        Set(ByVal Value As Double)
            intCurrentY = ConvertToPixelsY(Value)
        End Set
    End Property
    Public Function ConvertToPixelsX(ByVal num As Double) As Double
        Return IIf(ScaleMode = ScaleModeConstants.vbTwips, _
                   VB6.TwipsToPixelsX(num), num)
    End Function

    Public Function ConvertToPixelsY(ByVal num As Double) As Double
        Return IIf(ScaleMode = ScaleModeConstants.vbTwips, _
                   VB6.TwipsToPixelsY(num), num)
    End Function
End Class
```

表 7.3 と表 7.4 は、Visual Basic 6.0 Printer オブジェクトのプロパティとメソッドと、Visual Basic .NET の対応する機能をまとめたものです。直接対応する機能がない場合は、追加情報のリンクを記載してあります。

以下の例は、プリンタ オブジェクトを使用する Visual Basic 6.0 のコードを、前述した独自のプリンタ クラスを使用する Visual Basic .NET のコードに変換する方法を示します。

```
Printer.FillColor = vbBlue
Printer.FillStyle = vbSolid
Printer.Circle (3000, 3000), 1500, vbRed
Printer.EndDoc
```

前の Visual Basic 6.0 のコードでは、青で塗りつぶされた円を印刷し、境界線は赤です。このコードを Visual Basic .NET に変換するには、前述のプリンタ クラスを使用して、以下の手順を実行します。

まず、このプリンタ クラスを .NET プロジェクトに追加します。次に、Micosoft.VisualBasic.Compatibility アセンブリへの参照を追加します。

その後で、Visual Basic .NET モジュールを作成して、以下のコードを追加します。

```
Module Module1
    Public Printer As PrinterClass = New PrinterClass
End Module
```

最後に、独自のプリンタ クラスの新しい定義と矛盾がないように、元の Visual Basic 6.0 のコードにいくつかの変更を行う必要があります。最終的な Visual Basic .NET のコードは以下のようになります。

```
Printer.FillColor = Color.Blue
Printer.FillStyle = PrinterClass.FillStyleConstants.vbFSSolid
Printer.Circle(New Point(3000, 3000), 1500, Color.Red)
Printer.EndDoc()
```

表 7.3: Printer **オブジェクトのプロパティと、Visual Basic .NET の対応する機能**

Visual Basic 6.0	Visual Basic .NET の対応する機能
ColorMode	PrintDocument.PrinterSettings.DefaultPageSettings.Color
Copies	PrintDocument.PrinterSettings.Copies
CurrentX	対応する機能はありません。**Graphics** クラスのさまざまなメソッドの場所や次元の引数に置き換わっています。 このプロパティは、プリンタ クラスの倍精度浮動小数点型変数でシミュレートできます。
CurrentY	対応する機能はありません。**Graphics** クラスのさまざまなメソッドの場所や次元の引数に置き換わっています。 このプロパティは、プリンタ クラスの倍精度浮動小数点型変数でシミュレートできます。
DeviceName	PrintDocument.PrinterSettings.PrinterName
DrawMode	対応する機能はありません。詳細については、MSDN の「Graphics Changes in Visual Basic .NET」を参照してください。
DrawStyle	System.Drawing.Drawing2D.DashStyle
DrawWidth	System.Drawing.Pen.Width
DriverName	対応する機能はありません。プリンタ ドライバは Windows で管理されるため、不要になりました。
Duplex	System.Drawing.Printing.Duplex
FillColor	対応する機能はありません。詳細については、MSDN の「Graphics Changes in Visual Basic .NET」を参照してください。 このプロパティは、プリンタ クラスの **System.Drawing.Color** 変数でシミュレートできます。
FillStyle	この機能は、**Drawing.SolidBrush** を使用して **Draws** オブジェクトを塗りつぶすことで実現できます。
Font	このプロパティは、プリンタ クラスの **System.Drawing.Font** クラスでシミュレートできます。
FontBold	値を取得するには、**System.Drawing.Font.Bold** を使用します。 値を設定するには、**VB6.FontChangeBold** を使用します。
FontCount	System.Drawing.FontFamily.GetFamilies().Length
FontItalic	値を取得するには、**System.Drawing.Font.Italic** を使用します。 値を設定するには、**VB6.FontChangeItalic** を使用します。
FontName	値を取得するには、**System.Drawing.Font.Name** を使用します。 値を設定するには、**VB6.FontChangeName** を使用します。
Fonts	System.Drawing.FontFamily.GetFamilies
FontSize	値を取得するには、**System.Drawing.Font.Size** を使用します。 値を設定するには、**VB6.FontChangeSize** を使用します。
FontStrikethru	値を取得するには、**System.Drawing.Font.Strikeout** を使用します。 値を設定するには、**VB6.FontChangeStrikeout** を使用します。

Visual Basic 6.0	Visual Basic .NET の対応する機能
FontTransparent	対応する機能はありません。詳細については、MSDN の『*Visual Basic Concepts*』の「Font Changes in Visual Basic .NET」を参照してください。
FontUnderline	値を取得するには、**System.Drawing.Font.Underline** を使用します。 値を設定するには、**VB6.FontChangeUnderline** を使用します。
ForeColor	System.Drawing.Pen.Color
hDC	**PrintDocument.PrinterSettings.GetHdevmode.ToInt32**
Height	**PrintDocument.DefaultPageSettings.PaperSize.Height**
Orientation	**PrintDocument.DefaultPageSettings.Landscape**
Page	直接対応するものはありません。現在のページ番号は追跡されません。ただし、**BeginPrint** イベントに変数を設定し、**PrintPage** イベントで増分していけば、簡単にページ番号を追跡できます。
PaperBin	PrintDocument.PrinterSettings.PaperSources
PaperSize	PrintDocument.DefaultPageSettings.PaperSize
Port	不要になりました。ポート情報は、**PrintPreviewDialog** コントロールによって自動的に設定されます。
PrintQuality	PrintDocument.DefaultPageSettings.PrinterResolution
RightToLeft	不要になりました。印刷方向は、Windows のローカリゼーション設定によって制御されます。
ScaleHeight	対応する機能はありません。詳細については、MSDN の『*Visual Basic Concepts*』の「Coordinate System Changes in Visual Basic .NET」を参照してください。
ScaleLeft	対応する機能はありません。詳細については、MSDN の「Coordinate System Changes in Visual Basic .NET」を参照してください。
ScaleMode	対応する機能はありません。詳細については、MSDN の「Coordinate System Changes in Visual Basic .NET」を参照してください。
ScaleTop	対応する機能はありません。詳細については、MSDN の「Coordinate System Changes in Visual Basic .NET」を参照してください。
ScaleWidth	対応する機能はありません。詳細については、MSDN の「Coordinate System Changes in Visual Basic .NET」を参照してください。
TrackDefault	直接対応するものはありません。プリンタが既定でも、既定のプリンタが変更された場合に印刷が中止になっていないかどうかを確認するには、**PrinterSettings** クラスの **IsDefaultPrinter** プロパティを使用します。
TwipsPerPixelX	不要になりました。Visual Basic .NET の測定単位は常にピクセルです。
TwipsPerPixelY	不要になりました。Visual Basic .NET の測定単位は常にピクセルです。
Width	PrintDocument.DefaultPageSettings.PaperSize.Height
Zoom	不要になりました。プリンタにズーム機能がある場合、設定は自動的に **[印刷]** ダイアログボックスに公開されます。

表 7.4: Printer **オブジェクトのメソッドと、Visual Basic .NET の対応する機能**

Visual Basic 6.0	Visual Basic .NET の対応する機能
Circle	PrintPageEvents.Graphics.DrawEllipse
EndDoc	**PrintDocument.Print**
KillDoc	**PrintEventArgs.Cancel**
Line	**PrintPageEvents.Graphics.DrawLine**
NewPage	このメソッドは、ページの配列を使用して、ページごとに **PrintDocument** クラスの **Print** メソッドを呼び出してシミュレートする必要があります。
PaintPicture	PrintPageEvents.Graphics.DrawImage
Print	**PrintPageEvents.Graphics.DrawString**
PSet	**PrintPageEvents.Graphics.DrawLine**
Scale	対応する機能はありません。詳細については、MSDN の『*Visual Basic Concepts*』の「Coordinate System Changes in Visual Basic .NET」を参照してください。
ScaleX	対応する機能はありません。詳細については、MSDN の「Coordinate System Changes in Visual Basic .NET」を参照してください。
ScaleY	対応する機能はありません。詳細については、MSDN の「Coordinate System Changes in Visual Basic .NET」を参照してください。
TextHeight	System.Drawing.Graphics.MeasureString
TextWidth	**System.Drawing.Graphics.MeasureString**

Printers コレクションのアップグレード

Visual Basic 6.0 には、システムで使用可能なプリンタに関する情報を返す Printers コレクションがあります。たとえば、特定のプリンタのプリンタ レイアウトを確認できます。Printers コレクションは、Visual Basic .NET ではサポートされていません。印刷モデルが変更されたため、Visual Basic .NET にアップグレードするときには Printers コレクションの参照を手動で調整する必要があります。

以下に示す Visual Basic 6.0 の例では、Printers コレクションを使用して、最初に使用可能なプリンタを探し、そのページ方向を [縦] に設定します。コレクションでプリンタが見つかると、プリンタ デバイスの名前がメッセージ ボックスに表示されます。この例のコードは、以下のようになります。

```
Dim prn As Printer
For Each prn In Printers
   If prn.Orientation = vbPRORPortrait Then
      MsgBox prn.DeviceName
      ' プリンタの検索を中止します。
      Exit For
   End If
Next
```

Visual Basic .NET では、Printers コレクションはサポートされないため、アップグレード ウィザードではこのコードを自動的にアップグレードすることはできません。アップグレード ウィザードを適用すると、以下のようなアップグレードの問題を指摘する警告がコードに表示されます。

```
' UPGRADE_ISSUE: Printers コレクションはアップグレードされませんでした。
For Each prn In Printers
```

Visual Basic .NET で同様の効果を実現するための唯一の方法は、Microsoft .NET Framework で提供されるさまざまなクラスを使用して、独自の Printers コレクションを実装することです。たとえば、新しい Visual Basic .NET モジュールとクラスは、以下のように作成できます。

```
Module PrintersModule
   Public Printers As New PrintersCollection
End Module

Public Class PrintersCollection : Implements IEnumerable
   Private printer As System.Drawing.Printing.PrinterSettings

   Public ReadOnly Property Count() As Integer
       Get
           Return System.Drawing.Printing.PrinterSettings.InstalledPrinters.Count
       End Get
   End Property

   Public ReadOnly Property Item(ByVal Index As Integer) _
                               As System.Drawing.Printing.PrinterSettings
       Get
           printer = New System.Drawing.Printing.PrinterSettings
           printer.PrinterName = _
               System.Drawing.Printing.PrinterSettings.InstalledPrinters.Item(Index)
           Return printer
       End Get
   End Property
   Overridable Function GetEnumerator() As IEnumerator _
                                  Implements  IEnumerable.GetEnumerator
      Dim Count As Integer = _
          System.Drawing.Printing.PrinterSettings.InstalledPrinters.Count
      Dim printersArray(Count) As System.Drawing.Printing.PrinterSettings
      Dim i As Integer
      For i = 0 To Count - 1
          printersArray(i) = New System.Drawing.Printing.PrinterSettings
          printersArray(i).PrinterName = _
              System.Drawing.Printing.PrinterSettings.InstalledPrinters.Item(i)
      Next
      Return printersArray.GetEnumerator()
   End Function
End Class
```

このモジュールとクラスを作成し、アップグレード ソリューションに追加したら、いくつかの細かい修正を加えるだけで、元のコードをアップグレードできます。以下に変更内容を示します (変更点は**太字**部分です)。

```
Dim prn As System.Drawing.Printing.PrinterSettings
For Each prn In Printers
   If prn.DefaultPageSettings.Landscape = False Then
      MsgBox(prn.PrinterName)
      Exit For
   End If
Next
```

Visual Basic .NET で Printers コレクションの機能を置き換える方法についての詳細は、MSDN の『Visual Basic Concepts』の「Printers Collection Changes in Visual Basic .NET」を参照してください。

Forms コレクションのアップグレード

Visual Basic 6.0 の Forms コレクションは、プロジェクトに読み込まれたすべてのフォームのコレクションです。Forms コレクションの最も一般的な用途は、フォームが読み込まれているかどうかの確認、読み込まれたフォームの反復処理、名前を指定したフォームのアンロードです。Visual Basic .NET では Forms コレクションはサポートされませんが、この章で説明した他のオブジェクトと同様に、同様の機能が提供されます。

以下の Visual Basic 6.0 の例は、Forms コレクションを反復処理して、frmSecurity という名前のフォームが開いているかどうかを確認し、開いている場合には、btnInsert、btnUpdate、および btnDelete というボタンを無効にします。

```
Public Sub checkForm()
    Dim privateForm As Form
    Dim tempControl As Control
    For Each privateForm In Forms
        If privateForm.Name = "frmSecurity" Then
            For Each tempControl In privateForm.Controls
                If tempControl.Name = "btnInsert" Or _
                   tempControl.Name = "btnUpdate" Or _
                   tempControl.Name = "btnDelete" Then
                    tempControl.Enabled = False
                End If
            Next
            MsgBox "For security reasons, this form can not be shown"
        End If
    Next
End Sub
```

Visual Basic .NET では Forms コレクションはサポートされないため、アップグレードウィザードでは Forms コレクションを自動的にアップグレードできません。前述のように、このようなコードにアップグレード ウィザードを使用すると、以下のようなアップグレードの問題に関するコメントが生成されます。

```
' UPGRADE_ISSUE: Forms コレクションはアップグレードされませんでした。
For Each privateForm In Forms
```

Forms コレクションの問題にも、Printers コレクションの問題を解決するのと同じ方法を適用できます。つまり、Printers の例で示したのと同じように、Visual Basic .NET の機能をマスクするモジュールとクラスを作成します。この場合、コードは以下のようになります。

```
Module FormsCollection
   Public Forms As New FormsCollectionClass()
End Module

Class FormsCollectionClass : Implements IEnumerable
   Private collec As New Collection()
   Sub Add(ByVal tempForm As Form)
      collec.Add(tempForm)
   End Sub
   Sub Remove(ByVal tempForm As Form)
      Dim itemCount As Integer
      For itemCount = 1 To collec.Count
         If tempForm Is collec.Item(itemCount) Then
            collec.Remove(itemCount)
            Exit For
         End If
      Next
   End Sub
   ReadOnly Property Item(ByVal index) As Form
      Get
         Return collec.Item(index)
      End Get
   End Property
   Overridable Function GetEnumerator() As _
      IEnumerator Implements IEnumerable.GetEnumerator
      Return collec.GetEnumerator
   End Function
End Class
```

FormsCollectionClass は、読み込まれたすべてのフォームをコレクションに格納します。各フォームを作成したら、フォームがコレクションに追加されたことを確認し、フォームを解放した後にはコレクションからフォームが削除されていることを確認する必要があります。たとえば、frmSecurity の New イベントでは、以下のようなコード行を追加することで、frmSecurity をコレクションに追加できます。

```
Forms.Add(Me)
```

同様に、Disposed イベントでは、以下のようなコード行を追加することで、コレクションから frmSecurity を削除できます。

```
Forms.Remove(Me)
```

ソリューションのすべてのフォームについて、New イベントと Disposed イベントに Add 行と Remove 行を追加する必要があります。

新しいモジュールとクラスをアップグレード ソリューションに追加したら、後は一切の変更を加えることなく、元の checkForm サブルーチン コードをそのまま使用できます。

Visual Basic 2005 の場合 :

Forms コレクションは My.Application.OpenForms クラスに実装されているため、新しい Visual Basic 2005 ではこのソリューションは使用できません。これにより、この例で示したようにグローバル メンバを使用したラッパー クラスやモジュールを作成することなく、自動アップグレードを実行できるようになります。

以下に示すように、アップグレード後の Forms コレクション コードは、元のコードとほとんど変わりません。

```
Public Sub checkForm()
   Dim privateForm As System.Windows.Forms.Form
   Dim tempControl As System.Windows.Forms.Control
   For Each privateForm In My.Application.OpenForms
      If privateForm.Name = "frmSecurity" Then
         For Each tempControl In privateForm.Controls
            If tempControl.Name = "btnInsert" Or _
               tempControl.Name = "btnUpdate" Or tempControl.Name = "btnDelete" Then
               tempControl.Enabled = False
            End If
         Next tempControl
         MsgBox("For security this form can not be showed")
      End If
   Next privateForm
End Sub
```

Clipboard オブジェクトのアップグレード

Visual Basic 6.0 には、クリップボードとの間でテキストやグラフィックスをやり取りするための Clipboard オブジェクトがあります。Visual Basic .NET では、Clipboard の操作には System.Windows.Forms.Clipboard 名前空間を使用します。新しい Clipboard クラスは、特定の形式でデータを設定および取得でき、Clipboard オブジェクトの内容のクエリを実行してサポートされる形式を確認できるという点で、Visual Basic 6.0 の Clipboard クラスより柔軟です。ただし、Visual Basic 6.0 Clipboard オブジェクトのコードは自動的にアップグ

レードできないため、この新しい柔軟性を利用するにはそれなりの手間がかかります。とはいえ、同じ機能を Visual Basic .NET で実装するのは簡単です。

たとえば、Text1 という名前の TextBox コントロールが 1 つ含まれる Visual Basic 6.0 フォームがあるとします。以下のコードは、vbCFText 定数を使用してプレーンテキスト形式のみを適用することを指定し、クリップボードとの間で "Hello" というテキストをやり取りします。

```
Clipboard.Clear
Clipboard.SetText "Hello", vbCFText
If Clipboard.GetFormat(vbCFText) Then
   Text1.Text = Clipboard.GetText(vbCFText)
End If
```

アップグレード ウィザードでこのコードをアップグレードすると、Clipboard オブジェクトのコードが変更されずに残り、以下のようなアップグレード警告が表示されます。

```
' UPGRADE_ISSUE: Clipboard メソッド Clipboard.Clear はアップグレードされませんでした。
```

Visual Basic .NET では Clipboard オブジェクト関数はサポートされないため、このアップグレード後のコードは Visual Basic .NET ではコンパイル エラーになります。この問題を修正するには、アップグレードされなかった Clipboard オブジェクトのコードを削除して、Visual Basic .NET の機能に置き換える必要があります。新しい Visual Basic .NET のコードは、以下のようになります。

```
Dim datobj As New System.Windows.Forms.DataObject

datobj.SetData("")
datobj.SetData System.Windows.Forms.DataFormats.Text, "hello"
System.Windows.Forms.Clipboard.SetDataObject(datobj)

If System.Windows.Forms.Clipboard.GetDataObject.GetDataPresent( _
   System.Windows.Forms.DataFormats.Text) Then
   Text1.Text = System.Windows.Forms.Clipboard.GetDataObject.GetData( _
   System.Windows.Forms.DataFormats.Text)
End If
```

表 7.5 に、残りの Clipboard オブジェクト関数と、Visual Basic .NET の対応する機能を示します。DataFormat パラメータは、DataFormats.Text や DataFormats.Rtf などの適切な形式に置き換える必要があります。

Visual Basic .NET の機能で Clipboard コードを書き換えるには、クリップボードに移動したときの内容の形式 (この例の場合のプレーンテキストなど) を知っておく必要があります。表 7.6 は、Clipboard クラスの定数と、Visual Basic .NET の対応する機能を示します。

表 7.5: Clipboard メソッドに対応する Visual Basic .NET の機能

Visual Basic 6.0	Visual Basic .NET の対応する機能
Clear メソッド	Clipboard.SetDataObject("")
Clipboard.GetData	Clipboard.GetDataObject.GetData(*DataFormat*, True)
GetFormat	Clipboard.GetDataObject.GetDataPresent(*DataFormat*)
GetText	Clipboard.GetDataObject.GetData(*DataFormat*, True)
SetData	Dim datobj As New System.Windows.Forms.DataObject datobj.SetData(*DataFormat*, *data*) System.Windows.Forms.Clipboard.SetDataObject(*datobj*)
SetText	**Dim datobj As New System.Windows.Forms.DataObject datobj.SetData(*DataFormat*, *data*) System.Windows.Forms.Clipboard.SetDataObject(*datobj*)**

表 7.6: Clipboard 定数に対応する Visual Basic .NET の機能

Visual Basic 6.0	Visual Basic .NET の対応する機能
vbCFBitmap	System.Windows.Forms.DataFormats.Bitmap
vbCFDIB	System.Windows.Forms.DataFormats.DIB
vbCFEMetafile	System.Windows.Forms.DataFormats.EnhancedMetafile
vbCFFiles	System.Windows.Forms.DataFormats.FileDrop
vbCFLink	対応する機能はありません。**Clipboard** オブジェクトの **vbCFLink** 定数の詳細については、MSDN の『*Visual Basic Concepts*』の「Dynamic Data Exchange Changes in Visual Basic .NET」を参照してください。
vbCFMetafile	System.Windows.Forms.DataFormats.MetafilePict
vbCFPalette	System.Windows.Forms.DataFormats.Palette
vbCFRTF	System.Windows.Forms.DataFormats.Rtf
vbCFText	**System.Windows.Forms.DataFormats.Text**

Visual Basic 2005 の場合 :

My.Computer は、Visual Studio .NET 2005 で提供される My 名前空間の最上位グループの 1 つです。My.Computer は、アプリケーションを実行しているコンピュータのアイテムに関連し、.NET Framework で最も一般的に使用されるオブジェクトのインスタンスへのアクセスを提供します。これらのアイテムの 1 つがクリップボードです。Visual Basic Upgrade Wizard 2005 は、Visual Basic 6.0 Clipboard のメンバを自動的に My.Computer.Clipboard メンバにアップグレードします。

Licenses コレクションのアップグレード

Visual Basic 6.0 では、アプリケーションの実行中に ActiveX コントロールを動的に読み込むことができます。ActiveX コントロールの中には、プログラムで使用するためのライセンスが必要なものがあります。アプリケーションの Licenses コレクションには、読み込まれたすべての ActiveX コントロールのライセンス情報が含まれており、必要に応じてライセンスを個別に追加または削除できます。Visual Basic .NET のライセンス モデルはまったく異なるため、Licenses コレクションを使用するアプリケーションをアップグレードするには、一定の作業が必要になります。

以下のコード例は、Visual Basic 6.0 での一般的な Licenses コレクションの使用例です。この例では、Customers.ListCustomer コントロールのライセンスがチェックされ、確認後にこのコンポーネントの使用が許可されます。ライセンスが設定されていない場合には、メッセージが表示されます。

```
On Error GoTo NoLicense
    Dim customer As VBControlExtender
    Licenses.Add "Customers.ListCustomer"
    Set customer = Me.Controls.Add("Customers.ListCustomer", "ListCustomer")
    customer.Visible = True
    Exit Sub

NoLicense:
    'エラーがこのコントロールのライセンスに関するエラーであることを確認します。
    If Err.Number = 731 Then
      MsgBox "You need a License to dynamically load this ActiveX Customers.ListCustomer control"
    End If
```

ただし、Visual Basic .NET では、ライセンスは直接、実行可能ファイルにコンパイルされます。つまり、実行時にコントロールを動的に追加するためには、コントロールがプロジェクトに存在している必要があります。したがって、ライセンス管理を自動的にアップグレードすることはできません。この場合には、これに代わる方策を Visual Basic .NET で適用する必要があります。

1 つの方法は、プロジェクトにダミー フォームを追加して、このフォームに動的に追加されるすべての ActiveX コントロールを配置することです。このダミー フォームを作成したら、プロジェクト内の任意のフォームに ActiveX コントロールを動的に追加できます。Visual Basic .NET のコードは以下のようになります。

```
On Error GoTo NoLicense
    Dim customer As System.Windows.Forms.AxHost
    customer = New Customers.ListCustomer
    Me.Controls.Add("Customers.ListCustomer", "ListCustomer")
    customer.Visible = True
    Exit Sub

NoLicense:
    ' エラーがこのコントロールのライセンスに関するエラーであることを確認します。
    If Err.Number = 731 Then
        MsgBox("You need a License to dynamically load this ActiveX
```

```
Customers.ListCustomer control")
    End If
```

ライセンス管理は、コードから削除されています。もう不要になったためです。

Controls コレクションのアップグレード

実行時にコントロールを追加または削除できるように、Visual Basic 6.0 には Controls コレクションが提供されています。Controls コレクションは、フォームまたはコンテナ上のコントロールの動的なコレクションです。このコレクションを通じて、実行時にフォーム (またはコンテナ) にコントロールを追加または削除することが可能になります。

たとえば、実行時にフォームに Label を動的に追加する以下のコードを考えてみましょう。

```
Dim labelControl As Control
Set labelControl = Me.Controls.Add("VB.Label", "labelControl")
labelControl.Visible = True
```

このコードは、フォームに Label を追加して、表示します。Visual Basic 6.0 ではコントロールを動的に削除することも簡単です。以下のコード行は、前のコード例で追加された Label を削除します。

```
Me.Controls.Remove "labelControl"
```

Visual Basic 6.0 モデルの大きなデメリットの 1 つが、Microsoft IntelliSense® 技術では Controls コレクションがサポートされない点です。追加するコントロールのメソッド、パラメータ、および ProgID を覚えておく必要があります。

Add メソッドの動作は、Visual Basic 6.0 と Visual Basic .NET では異なるため、Controls コレクションの Add メソッドを使用する Visual Basic 6.0 のコードをアップグレード ウィザードで自動的にアップグレードすることはできません。ただし、Visual Basic .NET には同じ機能を実現する対応する機能があります。Visual Basic .NET で組み込みコントロールを追加または削除するのは比較的簡単です。以下の例は、Visual Basic .NET で実行時にフォームに Label を動的に追加する方法を示します。

```
Dim c As Control
c = New Label()
c.Name = "labelControl"
Me.Controls.Add(c)
```

Visual Basic .NET で使用されるフォーム モデルである Windows フォームでは、コントロールには名前ではなく、番号が付けられます。名前を指定してコントロールを削除するには、Me.Controls コレクションを反復処理して目的のコントロールを探してから、削除する必要があります。以下のコードは、Visual Basic .NET で名前を指定して、新しく追加されたコントロールを削除する方法を示しています。

```
Dim tempControl As Control
For Each tempControl In Me.Controls
   If tempControl.Name = "labelControl" Then
      Me.Controls.Remove(tempControl)
      Exit For
   End If
Next
```

Visual Basic 2005 の場合 :

新しい Visual Basic 2005 の Controls コレクションには、Visual Basic 6.0 と同じ方法でコントロールを削除できるメソッドがあります。したがって、Controls コレクションを反復処理する必要はありません。たとえば、前の反復処理の例を以下のように 1 行のコードに変えることができます。

```
Me.Controls.RemoveByKey(tempControl)
```

Item インデックス付きプロパティを使用すれば、コントロールの数値インデックスを指定することで、コレクション内の 1 つのコントロールにアクセスできます。Visual Basic 2005 では、コントロールの名前を (文字列として) 指定することで、1 つのコントロールにアクセスすることも可能です。

Visual Studio 2005 には、ControlCollection クラスに新しいメソッドも追加されています。このメソッドを使用すれば、Visual Basic 6.0 の ControlArray の機能の一部を実現できます。ControlCollection は、Control クラスの Controls プロパティの種類です。このクラスには、コントロールとそのすべての子コントロールを繰り返し検索して、特定の名前をもつコントロールを探し、一致するすべてのコントロールの配列を返す find() メソッドが含まれています。その他の新しいメソッドとしては、RemoveByKey()、ContainsKey()、IndexOfKey() と、Item プロパティの新しい文字列パラメータがあります。これらの新しいメソッドはすべて、インデックスではなく、コントロールの名前を使用して ControlCollection を操作するための手段を提供します。詳細については、MSDN の『.NET Framework Class Library』の「Control.ControlCollection Class (System.Windows.Forms)」を参照してください。

Visual Basic 2005 の新しい Controls コレクションの詳細については、MSDN の『Visual Basic Concepts』の「Controls Collection for Visual Basic 6.0 Users」を参照してください。執筆時点では、本書の内容は Visual Basic 2005 のプレリリース バージョンに基づいており、今後変更されることがある点をあらかじめご了承ください。

実行時に ActiveX コントロールを動的に追加するには、もう少し作業が必要です。Visual Basic .NET は、ActiveX コントロール用のラッパーを作成します。コントロールを追加するには、その前にこれらのラッパーが存在している必要があります。また、フォームにコントロールを作成するには、ActiveX コントロールのデザインタイム ライセンスも存在している必要があります。Visual Basic .NET では、ライセンスは実行可能ファイルにコンパイルされます。つまり、実行時にコントロールを動的に追加するには、コントロールがプロジェクトに存在している必要があります。1 つの方法は、プロジェクトにダミー フォームを追加して、このフォームに動的に追加されるすべての ActiveX コントロールを配置することです。このダミー フォームを作成したら、プロジェクト内の任意のフォームに ActiveX コントロールを動的に追加できます。以下のコードは、フォームに Windows Common Controls TreeView コントロールを動的に追加する方法を示します (ここでは、TreeView の追加元となるダミー フォームが既にプロジェクトに追加されていることを前提としています)。

```
Dim activexControl As New AxMSComctlLib.AxTreeView()
activexControl.Name = "TreeViewUsuers"
Me.Controls.Add(activexControl)
```

コントロールを追加したら、組み込みコントロールを削除するのと同じ方法で、動的にコントロールを削除できます。作業が必要になるのは、コントロールを追加する場合のみです。

Visual Basic 6.0 と Visual Basic .NET では、Controls コレクションの動作が違うことも考慮しておく必要があります。Visual Basic 6.0 では、このコレクションには、特定の Form に追加されたすべてのコントロールが含まれています。これには、PictureBox コントロールや Frame コントロールなどの他のコンテナ コントロールに追加されたコントロールも含まれます。Visual Basic .NET では、Controls コレクションには、コレクションを所有するコントロールに直接追加されたコントロールのみが含まれます。たとえば、button1 という名前の Button コントロールが含まれる Panel コントロールを Form が含んでいるとすると、button1 は Form に属する Controls コレクションには含まれず、パネルの Controls コレクションに含まれます。つまり、フォームの Controls コレクションの要素を反復処理する場合、Visual Basic 6.0 の場合のようにフォームに含まれるすべてのコントロールが対象になるわけではありません。Controls コレクションの動作の違いの詳細については、MSDN の「Getting Back Your Visual Basic 6.0 Goodies」を参照してください。

以下のコード例は、Visual Basic 6.0 フォームのすべてのコントロールで操作を実行する、代表的な For Each ループを示します。

```
Function ListOfControlNames() As String
    Dim res As String
    For Each c In Me.Controls
        res = res & c.Name & " "
    Next
    ListOfControlNames = res
End Function
```

フォームが図 7.1 に示すようなレイアウトになっていた場合、前例の関数の出力は「Frame1 Command1」になります。

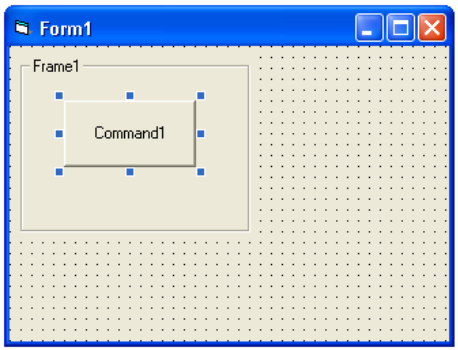

図 7.1

コントロールがネストされたフォーム

ListOfControlNames 関数をアップグレードすると、コードは以下のようになります。

```
Function ListOfControlNames() As String
    Dim c As System.Windows.Forms.Control
    Dim res As String
    For Each c In Me.Controls
        res = res & c.Name & " "
    Next c
    ListOfControlNames = res
End Function
```

Visual Basic .NET では、この関数の出力は「Frame1」になります。このように、Visual Basic 6.0 の場合と結果は明らかに異なります。この違いを修正するには、Form1 に含まれるコントロールを繰返し取得する 2 つの関数を定義する必要があります。以下に、この 2 つの関数を示します。

```
Function MyControls() As ArrayList
    Dim res As New ArrayList
    GetAllControls(Me, res)
    Return res
End Function
Function GetAllControls(ByVal c As Control, ByVal res As ArrayList)
    Dim curControl As Control
    For Each curControl In c.Controls
        res.Add(curControl)
        GetAllControls(curControl, res)
    Next
End Function
```

Me.Controls ではなく、MyControls 関数を使用するように、ListOfControlNames 関数の For Each ループを変更する必要があります。コントロールは反復的に取得されるため、コントロールの処理順序が変わる場合があることに注意してください。コントロールの取得順序にコードが依存する場合には、Visual Basic 6.0 のコードの機能に対応するように調整する必要があります。

まとめ

Visual Basic .NET での変更により、以前のバージョンの一部のオブジェクトが使用できなくなりました。これらのオブジェクトをアップグレードする上での作業を軽減できるように、互換性ライブラリが用意されています。互換性ライブラリにオブジェクトがない場合でも、新しいオブジェクトや関数を通じて、ほぼ同じ機能が Visual Basic .NET でも提供されています。この章では、使用が中止された Visual Basic 6.0 の一般的なオブジェクトについて説明し、これらのオブジェクトを Visual Basic .NET の機能に置き換える方法を解説しました。

Visual Basic 6.0 と Visual Basic .NET の間では、オブジェクト以外にも変更点があります。その他の言語機能の中にも変更されたり、使用が中止されたものがあります。次の章では、一般的に使用される言語機能の中で、アップグレード ウィザードでは自動的にアップグレードできない機能のアップグレード方法について説明します。

詳細情報

Ed Robinson、Robert Ian Oliver、Michael Bond『Upgrading Microsoft Visual Basic 6.0 To Microsoft Visual Basic .NET』Redmond: Microsoft Press, 2001、ISBN: 073561587X。MSDN (http://msdn.microsoft.com/vbrun/staythepath/additionalresources/upgradingvb6/) でも参照できます。

ArtinSoft Visual Basic Upgrade Wizard Companion の入手方法、およびその他のアップグレード ツールとサービスの詳細については、ArtinSoft の Web サイト (http://www.artinsoft.com/) を参照してください。

Visual Basic .NET に対応する機能がある App オブジェクトのプロパティとメソッドの包括的な一覧については、MSDN の『Visual Basic Concepts』の「App Object Changes in Visual Basic .NET」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vxconChangesToAppObjectInVisualBasicNET.asp) を参照してください。

Screen オブジェクトの変更点の詳細については、MSDN の『Visual Basic Concepts』の「Screen Object Changes in Visual Basic .NET」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vxconChangesToScreenObjectInVisualBasicNET.asp) を参照してください。

PrintDocument コンポーネントを使用した Visual Basic .NET からの印刷方法の詳細については、MSDN の「Printing with the PrintDocument Component」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconprintingwithprintdocumentcontrol.asp) を参照してください。

詳細については、MSDN の『Visual Basic Concepts』の「Graphics Changes in Visual Basic .NET」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbcongraphicschangesinvisualbasicnet.asp) を参照してください。

Controls コレクションの動作の違いの詳細については、MSDN の「Getting Back Your Visual Basic 6.0 Goodies」(http://msdn.microsoft.com/vbasic/using/columns/adventures/default.aspx?pull=/library/en-us/dnadvnet/html/vbnet05132003.asp) を参照してください。

Visual Basic 6.0 の FontTransparent プロパティに対応する機能を見つける方法の詳細については、MSDN の『Visual Basic Concepts』の「Font Changes in Visual Basic .NET」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconfontchangesinvisualbasic60.asp) を参照してください。

Visual Basic 6.0 の ScaleHeight プロパティに対応する機能を見つける方法の詳細については、MSDN の『Visual Basic Concepts』の「Coordinate System Changes in Visual Basic .NET」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconcoordinatesystemchangesinvisualbasicnet.asp) を参照してください。

Visual Basic .NET で Printers コレクション機能を代用する方法の詳細については、MSDN の『Visual Basic Concepts』の「Printers Collection Changes in Visual Basic .NET」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vxconprinterscollectionchangesinvisualbasicnet.asp) を参照してください。

インデックスではなく、コントロールの名前を使用して ControlCollection を操作するための Visual Studio 2005 メソッドの詳細については、MSDN の『.NET Framework Class Library』の「Control.ControlCollection Class (System.Windows.Forms)」(http://msdn2.microsoft.com/library/f4w0yshb(en-us,vs.80).aspx) を参照してください。

Visual Basic 2005 の新しい Controls コレクションの詳細については、MSDN の「Controls Collection for Visual Basic 6.0 Users」(http://msdn2.microsoft.com/library/7e4daa9c(en-us,vs.80).aspx) を参照してください。

Clipboard オブジェクトの vbCFLink 定数の詳細については、MSDN の『Visual Basic Concepts』の「Dynamic Data Exchange Changes in Visual Basic .NET」(http://msdn.microsoft.com/library/en-us/vbcon/html/vbcondynamicdataexchangechangesinvisualbasicnet.asp) を参照してください。

8

一般的に使用される Visual Basic 6.0 言語機能のアップグレード

一般的に使用されるオブジェクトの他にも、Microsoft Visual Basic 6.0 言語の複数の機能が、Visual Basic .NET では使用が中止されて、直接サポートされなくなりました。ただし、Visual Basic .NET では、サポートされなくなった機能の多くに対応する機能が用意されています。この章では、サポートされなくなった言語機能と、Visual Basic .NET で同じ目的を達成するための代替機能について説明します。

この章を読む際には、高い品質保証の必要性に十分留意してください。これは、特に Visual Basic アップグレード ウィザードによってアップグレードされたコードすべてに当てはまります。ウィザードによって作成された Visual Basic .NET のソース コード ファイルは、エラーすることなくコンパイルできる場合もあります。しかし、アップグレード後のコードが元のコードと異なる動作をしたり、実行時例外が発生し、問題の箇所を修正しない限りプログラムが失敗したりする可能性があります。各項目では、適宜、確認の必要なこうした問題点について記載します。

既定のプロパティに関する問題の解決

Visual Basic 6.0 の既定のプロパティでは、オブジェクト名のみを使用して設定または取得するプロパティを指定することによって、コードを簡略化できます。このため、object.property 完全修飾名の代わりにオブジェクト名のみをコード内で使用しても、オブジェクトで一般的に使用されるプロパティを参照できます。しかし、Visual Basic .NET では、既定のプロパティがサポートされなくなりました。

既定のプロパティが含まれているコードのアップグレードに、アップグレード ウィザードを使用すると、既定のプロパティの使用箇所にプロパティ名が挿入されて、コードが変更されます。これは、各オブジェクト参照に対する既定のプロパティをウィザードが判断できる場合にのみ行われます。遅延バインドされたオブジェクト

の場合など、既定のプロパティとして参照されているプロパティをアップグレード ウィザードが判断できないときは、コードが変更されずに、解決が必要な問題を示す警告コメントが挿入されます。

この状況に対処するには、少なくとも 2 つの方法があります。1 つは、元のコード ベースを変更して、既定のプロパティを使用せず、すべてのプロパティを完全な名前で明示的に参照することです。もう 1 つの方法は、アップグレード後のコードを変更して、ウィザードによって特定された既定のプロパティの問題を、手動で修正します。ここでは、2 つ目の方法について説明します。

以下の Visual Basic 6.0 のコード例は、既定のプロパティへのアクセスに遅延バインドされたオブジェクトと事前バインドされたオブジェクトを使用した場合の、アップグレード結果の違いを示します。

```
Private Sub CopyButton_Click()
   EarlyBoundCopy Text1, Text2
   LateBoundCopy Text1, Text3
End Sub

' このメソッドのパラメータは、TextBox として明示的に定義されています。
Private Sub EarlyBoundCopy(sourceCtrl As TextBox, destCtrl As TextBox)
   destCtrl.Text = sourceCtrl
End Sub

' このメソッドのパラメータは、variant (既定の型) です。
Private Sub LateBoundCopy(sourceCtrl, destCtrl)
   destCtrl.Text = sourceCtrl
End Sub
```

上記のコードにアップグレード ウィザードが適用されると、以下に示すように、コード内の 2 か所に実行時の警告が挿入されます。太字の項目が、アップグレード ウィザードによって挿入されたアップグレード警告コメントで、既定のプロパティの解決試行時に検出された問題を示しています。

```
Private Sub CopyButton_Click(ByVal eventSender As System.Object, _
   ByVal eventArgs As System.EventArgs) Handles CopyButton.Click
   EarlyBoundCopy(Text1, Text2)
   LateBoundCopy(Text1, Text3)
End Sub

' このメソッドのパラメータは、TextBox として明示的に定義されています。
Private Sub EarlyBoundCopy(ByRef sourceCtrl As  _
   System.Windows.Forms.TextBox, ByRef destCtrl As _
   System.Windows.Forms.TextBox)
   destCtrl.Text = sourceCtrl.Text
End Sub

' このメソッドのパラメータは、variant (既定の型) です。
Private Sub LateBoundCopy(ByRef sourceCtrl As Object, _
```

```
   ByRef destCtrl As Object)
   ' UPGRADE_WARNING: オブジェクト destCtrl.Text の既定プロパティを
   ' 解決できませんでした。
   ' UPGRADE_WARNING: オブジェクト sourceCtrl の既定プロパティを
   ' 解決できませんでした。
   destCtrl.Text = sourceCtrl
End Sub
```

アップグレード ウィザードによって生成されたコードは、コンパイルには成功しますが、LateBoundCopy の呼び出し時に実行時例外が発生します。これは、既定のプロパティの扱いが Visual Basic .NET では異なるためです。既定のプロパティにはパラメータが必要です。既定のプロパティのサポート方法が変更されたため、ランタイムが sourceCtrl を文字列としてキャストしようとした結果、InvalidCastException が発生します (文字列ではなく TextBox であるため)。Visual Basic .NET のパラメータ化された既定のプロパティの詳細については、MSDN の『Visual Basic Language Concepts 』の「Default Properties Changes in Visual Basic」を参照してください。

メモ : アップグレード ウィザードによって生成された、既定のプロパティに関するすべての UPGRADE_WARNING コメントを、注意深く調べてください。ソース コードの問題を修正しないと、ハンドルされていない実行時例外の発生によってプログラムが失敗する可能性が高くなります。

アップグレード ウィザードによって生成されたコードを変更するには、オブジェクト名に正しいプロパティを手動で追加する必要があります。このコード例では、sourceCtrl に適したプロパティは Text プロパティです。

```
Private Sub LateBoundCopy(ByRef sourceCtrl As Object, _
   ByRef destCtrl As Object)
   destCtrl.Text = sourceCtrl.Text
End Sub
```

アップグレード後のコードに対するこの変更方法は、アップグレード ウィザードが事前バインドされたオブジェクトに対して行う処理とまったく同じです。その場合は、ウィザードによって適切な既定のプロパティが特定され、アップグレード後のコードに追加されます。可能な限り、コードでは事前バインドを使用することをお勧めします。

カスタム コレクション クラスに関する問題の解決

コレクション クラスを使用すると、オブジェクトをグループ化できます。グループ化は、通常、関連するオブジェクトの組織と処理を強化するために行います。第 7 章では、Controls コレクションや Printers コレクションなど、Visual Basic 6.0 で使用可能な標準コレクション クラスの一部について説明しました。しかし、独自のカスタム コレクションを構築することもできます。多くの場合、これには Visual Basic 6.0 クラス ビルダを使用します。

クラス ビルダで作成したカスタム コレクションは、アップグレード ウィザードでアップグレードできます。ただし、どの程度の手動調整が必要であるかは、クラス ビルダで生成された後のコレクション コードに加えられた変

更によって異なります。

カスタム コードがほとんどない かまったくない コレクションの場合、アップグレード ウィザードによってほぼすべてのアップグレード作業が行われ、結果コードの確認を促す ToDo コメントのみが挿入されます。簡単な変更を加えるだけで、最終結果が元のコレクションと同様に動作します。

たとえば、以下のような Visual Basic 6.0 コレクションがあるとします。

```
' コレクションが含まれるローカル変数
Private myCollec As Collection

Public Sub Add(obj As Object)
    mCol.Add obj
End Sub

Public Sub Insert(obj As Object, Key As Variant)
    mCol.Add obj, Key
End Sub

Public Property Get Item(Index As Variant) As Object
    ' コレクション内の要素を参照するときに使用します。
    ' vntIndexKey にはコレクションへの Index または Key が含まれます。
    ' このため、Variant として宣言されています。
    ' 構文 : Set myVar = x.Item(xyz) or Set myVar = x.Item(5)
  Set Item = myCollec(Index)
End Property

Public Property Get Count() As Long
    ' コレクション内の要素の数を取得するときに
    ' 使用します。構文 : Debug.Print x.Count
    Count = myCollec.Count
End Property


Public Sub Remove(vntIndexKey As Variant)
    ' コレクションから要素を削除するときに使用します。
    ' vntIndexKey には Index または Key が含まれます。このため、
    ' Variant として宣言されています。
    ' 構文 : x.Remove(xyz)


    myCollec.Remove vntIndexKey
End Sub


Public Property Get NewEnum() As IUnknown
    ' このプロパティを使用すると、このコレクションを
    ' For...Each 構文で列挙できます。
    Set NewEnum = myCollec.[_NewEnum]
End Property
```

```
Private Sub Class_Initialize()
    ' このクラスの作成時にコレクションを作成します。
    Set myCollec = New Collection
End Sub


Private Sub Class_Terminate()
    ' このクラスの終了時にコレクションを破棄します。
    Set myCollec = Nothing
End Sub
```

上記のコレクション コードにアップグレード ウィザードが適用されると、同様に動作する Visual Basic .NET のコードが生成されます。ただし、コレクションの NewEnum プロパティはコメント アウトされ、新しいメソッド GetEnumerator に置換されます。アップグレード ウィザードによって生成される、アップグレード後のコードを以下に示します。

```
Friend Class MyCollection
    Implements System.Collections.IEnumerable
    ' コレクションが含まれるローカル変数
    Private myCollec As Collection

    Public Sub Add(ByRef obj As Object)
        Dim mCol As Object
             ' UPGRADE_WARNING: Could not resolve default property
             ' of object mCol.Add.
        mCol.Add(obj)
    End Sub

    Public Sub Insert(ByRef obj As Object, ByRef Key As Object)
        Dim mCol As Object
             ' UPGRADE_WARNING: Could not resolve default property
             ' of object mCol.Add.
        mCol.Add(obj, Key)
    End Sub

    Public ReadOnly Property Item(ByVal Index As Object) As Object
        Get
            ' コレクション内の要素を参照するときに使用します。
                ' vntIndexKey にはコレクションへの Index または Key が
            ' 含まれます。このため、Variant として宣言されています。
            ' 構文 : Set abc = x.Item(xyz) or Set abc = x.Item(5)
            Item = myCollec.Item(Index)
        End Get
    End Property

    Public ReadOnly Property Count() As Integer
        Get
            ' コレクション内の要素の数を取得するときに
            ' 使用します。構文 : Debug.Print x.Count
            Count = myCollec.Count()
        End Get
```

```
    End Property


    ' UPGRADE_NOTE: NewEnum プロパティがコメント アウトされました。
    ' Public ReadOnly Property NewEnum() As stdole.IUnknown
        ' Get
            ' このプロパティを使用すると、このコレクションを
            ' For...Each 構文で列挙できます。
            ' NewEnum = myCollec._NewEnum
        ' End Get
    ' End Property

    Public Function GetEnumerator() As System.Collections.IEnumerator _
                    Implements System.Collections.IEnumerable.GetEnumerator
            ' UPGRADE_TODO: コレクション列挙子を返すには、コメントを外して
            ' 以下の行を変更してください。
        ' GetEnumerator = myCollec.GetEnumerator
    End Function


    Public Sub Remove(ByRef vntIndexKey As Object)
        ' コレクションから要素を削除するときに使用します。
        ' vntIndexKey には Index または Key が含まれます。このため、
        ' Variant として宣言されています。
        ' 構文 : x.Remove(xyz)


        myCollec.Remove(vntIndexKey)
    End Sub


    ' UPGRADE_NOTE: Class_Initialize は Class_Initialize_Renamed にアップグレードされました。
    Private Sub Class_Initialize_Renamed()
        ' このクラスの作成時にコレクションを作成します。
        myCollec = New Collection
    End Sub
    Public Sub New()
        MyBase.New()
        Class_Initialize_Renamed()
    End Sub


    ' UPGRADE_NOTE: Class_Terminate は Class_Terminate_Renamed にアップグレードされました。
    Private Sub Class_Terminate_Renamed()
        ' このクラスの終了時にコレクションを破棄します。
        ' UPGRADE_NOTE: オブジェクト myCollec をガベージ コレクトするまでこのオブジェクトを
                ' 破棄することはできません。
        myCollec = Nothing
    End Sub
    Protected Overrides Sub Finalize()
        Class_Terminate_Renamed()
        MyBase.Finalize()
    End Sub
End Class
```

NewEnum プロパティが GetEnumerator メソッドに置換されたため、追加の作業が必要な場合があります。この置換の確認が確実に行われるよう、アップグレード ウィザードによって、GetEnumerator コードに UPGRADE_TODO コメントが挿入され、この列挙子を実際に返すコードの各行がコメント アウトされます。その結果、基になるコレクションの列挙子メンバを返すだけで十分か、それとも元のコレクションがカスタマイズされているため追加のカスタム コードが必要かを、コードを評価して判断する必要が生じます。ほとんどの場合、このコードの各行からコメントをはずすだけで完成します。確認作業を必須とすることによって、アップグレード ウィザードによる生成結果が元の Visual Basic 6.0 コレクションと同様に動作するかどうかを、必ず意識的に決定することになります。

一般的に使用される関数およびオブジェクトに対する変更への対応

Visual Basic .NET では、Visual Basic 6.0 の多くの関数およびオブジェクト用に互換性ライブラリが用意されており、アプリケーションの機能的動作を維持しながらのアップグレードが可能です。アップグレード ウィザードによって、適宜、元のコード内の関数が、互換性ライブラリにある Visual Basic .NET の対応する関数に置換されます。また、アップグレード ウィザードでそのような置換が行われた場合、Visual Basic 6.0 互換性ライブラリ名前空間 (Microsoft.VisualBasic.Compatibility) への参照がソリューションに追加されます。

メモ : この章を通して、*互換性ライブラリ*という用語は、Visual Basic 6.0 互換性ライブラリ (Microsoft.VisualBasic.Compatibility) の省略名として使用します。

互換性ライブラリは、アップグレードに必要な作業量を最小限にします。このライブラリは、元の Visual Basic 6.0 関数の機能的動作の複製を試みます。しかし、ライブラリのメンバにアップグレードされた機能は、アプリケーションをさらに広く Visual Basic .NET に移行させるための、アップグレードの第一段階として捕らえるべきです。互換性ライブラリの一部の機能は、Microsoft .NET Framework にも用意されています。アップグレードにはこのように複数の方法がありますので、投入できる時間とリソースに応じて選択してください。一般的に、互換性ライブラリを使用するよりも、.NET Framework に機能をアップグレードする方が、冗長コンポーネントの数が削減されるため、より良いデザインと参照方法を持つアプリケーションが生成されます。

例として、以下のような Visual Basic 6.0 のコードがあるとします。

```
Private Sub cmdUpdate_Click()
    Dim currentTime As Date

    currentTime = Now
    lblDate.Caption = Format(currentTime, "dddd, mmm d yyyy")
    lblTime.Caption = Format(currentTime, "hh:mm:ss AMPM")
End Sub
```

このサンプル コードでは、ボタンをクリックするたびに、フォームのラベルが現在の日付と時刻に更新されます。Format 関数を使用して、日付と時刻を書式設定しています。

このコードにアップグレード ウィザードを適用すると、以下に示すように、Format 関数の互換性ライブラリのバージョンが参照されます。太字の項目は、互換性ライブラリを使用するためにアップグレード ウィザードによって加えられた変更を示します。

```
Private Sub cmdUpdate_Click(ByVal eventSender As System.Object, _
                            ByVal eventArgs As System.EventArgs) _
                            Handles cmdUpdate.Click
    Dim currentTime As Date

    currentTime = Now
    lblDate.Text = VB6.Format(currentTime, "dddd, mmm d yyyy")
    lblTime.Text = VB6.Format(currentTime, "hh:mm:ss AMPM")
End Sub
```

このコードは正しくコンパイルされ、元のコードと同様に動作します。

互換性ライブラリへの参照に代わる方法として、同様の動作を実現する、純粋な Visual Basic .NET の対応する関数を見つけることも考えられます。前の例では、互換性ライブラリの代わりに、String クラスで提供されている Format 関数の Visual Basic .NET バージョンを使用できます。変更後の Visual Basic .NET のコードは次のとおりです。太字の項目は、互換性ライブラリへの依存を削除するために必要な変更です。

```
Private Sub cmdUpdate_Click(ByVal eventSender As System.Object, _
                            ByVal eventArgs As System.EventArgs) _
                            Handles cmdUpdate.Click
    Dim currentTime As Date

    currentTime = Now
    lblDate.Text = String.Format("{0:dddd, mmm d yyyy}", currentTime)
    lblTime.Text = String.Format("{0:hh:mm:ss tt}", currentTime)
End Sub
```

この関数の改訂されたバージョンは、元の関数と同様に動作しますが、互換性ライブラリに依存していません。

互換性ライブラリには、1 つの関数に対して Visual Basic .NET の対応する関数が複数存在する場合もあります。たとえば、前の例を書き換えるには、VB6.Format を、currentTime オブジェクトが属する DateTime クラスの ToString メソッドに置換する方法もあります。このように変更したサンプル コードを以下に示します。ここでも、太字の項目は、互換性ライブラリへの依存を削除し、String.Format を置換するために適用される変更です。

```
Private Sub cmdUpdate_Click(ByVal eventSender As System.Object, _
                            ByVal eventArgs As System.EventArgs) _
                            Handles cmdUpdate.Click
    Dim currentTime As Date

    currentTime = Now
    lblDate.Text = currentTime.ToString("dddd, mmm d yyyy")
    lblTime.Text = currentTime.ToString("hh:mm:ss tt")
End Sub
```

互換性ライブラリには、一般的に使用される多くの Visual Basic 6.0 関数およびオブジェクトに対する置換関数が含まれています。関数およびオブジェクトの完全な一覧については、Visual Basic .NET ヘルプの「VisualBasic.Compatibility Namespace Reference」を参照してください。DateTime.ToString および String.Format の使用の詳細については、DateTime クラスおよび String クラスに関する Visual Basic .NET ヘルプを、それぞれ参照してください。

TypeOf に対する変更への対応

Visual Basic 6.0 では、実行時に TypeOf 句を使用してオブジェクトの型を確認できます。オブジェクトの型に基づいてアクションを選択するには、この句を判断ステートメント (つまり If) で使用します。オブジェクトが指定された型である場合はこの句が True と評価され、そうでない場合は False と評価されます。

Visual Basic .NET でも TypeOf キーワードがサポートされますが、動作が変更されました。主要な変更は次の 2 点です。

1. Visual Basic 6.0 では、ユーザー定義型が、TypeOf 句で使用できるオブジェクトの型と見なされています。Visual Basic .NET では、ユーザー定義型 (構造体とも呼ばれる) がオブジェクトの型と見なされなくなったため、TypeOf 句で使用できません。
2. Visual Basic 6.0 では、インターフェイスの実装によってのみ、継承が可能です。Visual Basic .NET では、本当の意味での継承がサポートされています。このため、クラスには他のクラスからの継承が可能です。これは TypeOf 句にも影響を及ぼします。変数の型ではなく、変数が参照するオブジェクトの型に基づいて、答えが導かれるためです。Visual Basic .NET の継承については、第 17 章「アプリケーションの改良の概要」の「オブジェクト指向の機能の利用」を参照してください。

これらのポイントを説明するため、Visual Basic 6.0 アプリケーションでの TypeOf 句の使用例を以下に示します。

```
Private Type MyType
   Data as Integer
End Type

' この例では、ユーザー定義型で TypeOf を使用します。
Sub CheckMyType()
```

```
   Dim demo as MyType
   If TypeOf demo Is MyType Then
      MsgBox "demo is of type MyType"
   End If
End Sub

' この例では、Visual Basic 型で TypeOf を使用します。
Sub ControlProcessor(MyControl As Control)
   Dim message as String
   If TypeOf MyControl Is CommandButton Then
      Message = "A button was clicked."
   ElseIf TypeOf MyControl Is CheckBox Then
      Message = "A checkbox was changed."
   ElseIf TypeOf MyControl Is TextBox Then
      Message = "A label was clicked."
   End If
   MsgBox Message
End Sub
```

前のコードにアップグレード ウィザードを適用することが可能で、アップグレード後のバージョンが生成されます。ただし、コードには UPGRADE_WARNING コメントがいくつか挿入され、ユーザー定義型の場合はコードがコンパイルできません。アップグレード ウィザードによって生成されるコードを、以下に示します。

```
Private Structure MyType
   Dim a As Short
End Structure

Sub CheckMyType()
   Dim demo As MyType
   ' UPGRADE_WARNING: TypeOf には新しい動作が含まれます。
   If TypeOf demo Is MyType Then
      MsgBox "demo is of type MyType"
   End If
End Sub

Sub ControlProcessor(ByRef MyControl As System.Windows.Forms.Control)
   ' UPGRADE_WARNING: TypeOf には新しい動作が含まれます。
   If TypeOf MyControl Is System.Windows.Forms.Button Then
      ' UPGRADE_WARNING: TypeName には新しい動作が含まれます。
      System.Diagnostics.Debug.WriteLine("You passed in a " & TypeName(MyControl))
   ' UPGRADE_WARNING: TypeOf には新しい動作が含まれます。
   ElseIf TypeOf MyControl Is System.Windows.Forms.CheckBox Then
      ' UPGRADE_WARNING: TypeName には新しい動作が含まれます。
      System.Diagnostics.Debug.WriteLine("You passed in a " & TypeName(MyControl))
   ' UPGRADE_WARNING: TypeOf には新しい動作が含まれます。
   ElseIf TypeOf MyControl Is System.Windows.Forms.TextBox Then
      ' UPGRADE_WARNING: TypeName には新しい動作が含まれます。
      System.Diagnostics.Debug.WriteLine("You passed in a " & TypeName(MyControl))
   End If
End Sub
```

ユーザー定義型の問題は変更が必要です。Visual Basic .NET ではユーザー定義型で TypeOf 演算子を使用できないためです。したがって、MyType をテストする If ステートメント内の TypeOf 句がアップグレードウィザードで残されているにもかかわらず、このコードのコンパイル時にエラーが発生します。このコードを変更する際は、TypeOf キーワードを削除し、ターゲットの型を持つ新しいオブジェクトを作成して、テスト オブジェクトとターゲット オブジェクトの型を比較するために GetType メソッドを使用する必要があります。変更後のコードを以下に示します。

```
Private Sub cmdUserDefined_Click(ByVal eventSender As System.Object, _
                                 ByVal eventArgs As System.EventArgs) _
                                 Handles cmdUserDefined.Click
    Dim demo As MyType
    Dim test As MyType
    If demo.GetType Is test.GetType Then
       MsgBox("MyType")
    End If
End Sub
```

この方法は、必要に応じてすべてのユーザー定義型に使用できます。

他の問題は、Visual Basic .NET の継承が原因の警告です。この 2 番目のコンテキストでは、TypeOf を使用しても、コンパイルおよび実行が可能です。しかし、それでもコードを注意深く確認してください。継承が原因となり、オブジェクトの型がオブジェクトの祖先クラスと一致する場合に、TypeOf が True を返す可能性があります。たとえば、前の If 構成要素内で Object 型のテストが導入されたため、予期しない結果が起きる場合があります。これを以下に示します。

```
Sub ControlProcessor(ByRef MyControl As System.Windows.Forms.Control)
   ' UPGRADE_WARNING: TypeOf には新しい動作が含まれます。
   If TypeOf MyControl Is System.Windows.Forms.Button Then
      ' UPGRADE_WARNING: TypeName には新しい動作が含まれます。
      System.Diagnostics.Debug.WriteLine("You passed in a " & TypeName(MyControl))
   ElseIf TypeOf MyControl Is Object Then
      System.Diagnostics.Debug.WriteLine("You passed in an Object")
   ' UPGRADE_WARNING: TypeOf には新しい動作が含まれます。
   ElseIf TypeOf MyControl Is System.Windows.Forms.CheckBox Then
      ' UPGRADE_WARNING: TypeName には新しい動作が含まれます。
      System.Diagnostics.Debug.WriteLine("You passed in a " & TypeName(MyControl))
   ' UPGRADE_WARNING: TypeOf には新しい動作が含まれます。
   ElseIf TypeOf MyControl Is System.Windows.Forms.TextBox Then
      ' UPGRADE_WARNING: TypeName には新しい動作が含まれます。
      System.Diagnostics.Debug.WriteLine("You passed in a " & TypeName(MyControl))
   End If
End Sub
```

前の例の新しい ElseIf ブランチによって、Button 以外のすべてのコントロールに対して "Object" というメッセージが表示されます。すべてのコントロールが Object クラスを継承するためです。このように、TypeOf 句を使用する場合は、アップグレード後のコードを注意深く確認しないと、予期しない分岐が実行される可能性があります。

Visual Basic 6.0 の Enum 値への参照のアップグレード

Visual Basic 6.0 の定義済み列挙子値を参照しているコードをアップグレードする際には、特別な注意が必要です。アップグレード ウィザードでは、列挙子への参照が、同じ値を表す Visual Basic .NET の新しいオブジェクトに置換される場合がありますが、これは単純な列挙子値ではありません。以降の例でこれを説明します。

Windows フォームまたはコントロールのプロパティを表す定数を Visual Basic 6.0 で定義すると、プロジェクトのアップグレード後にコードをコンパイルできない場合があります。以下のような Visual Basic 6.0 のコードがあるとします。

```
Const myGreenColor = vbGreen
```

この単純なステートメントでは、緑色を表すための定数値を定義しています。Visual Basic 6.0 の定義済み列挙子値 vbGreen の値に割り当てられています。アップグレード ウィザードを適用すると、以下の Visual Basic .NET のコードが生成されます。

```
' UPGRADE_NOTE: myGreenColor は Constant から Variable に変更されました。
Dim myGreenColor As System.Drawing.Color = System.Drawing.Color.Lime
```

まず気付くのは、宣言が定数宣言から変数宣言に変更されたことです。なぜ、このように変更されたのでしょう。それは、アップグレード ウィザードによって参照が Visual Basic .NET での緑色の表現に置換されたためです。ただし、System.Drawing.Color.Lime は定数値ではなく、非定数式です。これは固定の数値ではなく、System.Drawing.Color オブジェクトを返すプロパティです。

アップグレード後の Visual Basic .NET プロジェクトのコード ウィンドウで [Lime] を右クリックしてから [定義へ移動] をクリックすると、System.Drawing.Color.Lime の宣言を表示できます。System.Drawing.Color.Lime の定義が**オブジェクト ブラウザ**で以下のように表示されます。

```
Public Shared ReadOnly Property Lime() As System.Drawing.Color
```

この変更の影響で、Visual Basic 6.0 の定義済み列挙子値を参照しているコードのアップグレード時に、コンパイラ エラーが起きる場合があります。たとえば、元のコードで、赤、緑、および青の値を含む RGBColors という名前の列挙子を作成したとします。コードで以下のように列挙子値を使用して、赤、青、黒、および黄を組み合わせたカスタムの背景をラベル用に作成します。

```
Enum RGBColors
   myDarkRose = vbRed + vbBlue
   myWhite = vbBlue + vbBlack + vbYellow
End Enum
```

```
Private Sub frmCollection_Load()
   Label1.BackColor = RGBColors.myDarkRose
   Label2.BackColor = RGBColors.myWhite
End Sub
```

このコードにアップグレード ウィザードを適用すると、以下が生成されます。

```
Enum RGBColors
    myDarkRose = System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Red) _
                  + System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Blue)
    myWhite = System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Blue) _
              + System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Black) _
              + System.Drawing.ColorTranslator.ToOle(System.Drawing.Color.Yellow)
End Enum

Private Sub frmCollection_Load(ByVal eventSender As System.Object, _
                   ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    Label1.BackColor = _
                System.Drawing.ColorTranslator.FromOle(RGBColors.myDarkRose)
    Label2.BackColor = System.Drawing.ColorTranslator.FromOle(RGBColors.myWhite)
End Sub
```

色オブジェクト myDarkRose および myWhite の宣言が、定数、リテラル、列挙のいずれでもない値に初期化されているため、各行で "定数式が必要です。" というコンパイラ エラーが発生します。

これらの問題を修正するには、2 つの方法があります。1 つ目の方法として、独自の定数値を定義して、各列挙子の Visual Basic .NET プロパティと置き換えることができます。また、非定数値を直接使用することもできます。以降では、これらの方法について個別に説明します。

独自の定数値の定義

コードを修正する最も簡単な方法では、アップグレード ウィザードで生成された Red、Blue、Black、Yellow の各オブジェクトの代わりに、定数値を使用します。たとえば、Visual Basic 6.0 の定数 vbRed、vbBalck、vbWhite、および vbBlue の値を持つ定数を宣言できます。こうすれば、列挙子のメンバをこれらの新しい定数で初期化できます。この方法のコード例を以下に示します。

```
Const RedBlue As Integer = &HFFs + &HFF0000
Const BlueBlackYellow As Single = &HFF0000 + &H0s + &HFFFFs

Enum RGBColors
    myDarkRose = RedBlue
    myWhite = BlueBlackYellow
End Enum

Private Sub frmCollection_Load(ByVal eventSender As System.Object, _
```

```
                ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    Label1.BackColor = _
            System.Drawing.ColorTranslator.FromOle(RGBColors.myDarkRose)
    Label2.BackColor = System.Drawing.ColorTranslator.FromOle(RGBColors.myWhite)
End Sub
```

非定数値の使用

この問題のもう 1 つの解決方法では、RGBColors のメンバが使用されているすべてのインスタンスを、対応する System.Drawing.Color に置換します。定数値と非定数プロパティとの違いのため、この方法を取るには、追加の調査や作業が必要な場合があります。たとえば、前のコード例では、2 つの列挙子値を足すことによって背景色の値を計算しています。System.Drawing.Color プロパティ Red および Blue を代わりに使用した場合、この方法で背景色を直接計算することはできません。2 つのオブジェクトを足すことはできないためです。しかし、意味のある数値をオブジェクトに割り当てるようなメソッドまたは関数を使用できれば、その数値を計算に使用できます。数値からオブジェクトへの変換関数を使用すれば、数値計算の結果を、計算値を表すオブジェクトに戻すことができます。ただし、これは、そのような変換関数が存在する場合にのみ可能です。

色オブジェクトの場合、System.Drawing.ColorTranslator クラスの ToOle メソッドを使用することによって、意味のある色数値を取得できます。ToOle メソッドは、System.Drawing.Color.Red などの色オブジェクトから、これに対する RGB 色値を取得します。たとえば、ToOle(System.Drawing.Color.Red) は、RGB 値 255 または 16 進数値 FF を返します。Visual Basic 6.0 の vbRed 定数が同じ値を持つのは偶然ではありません。新しい色を計算した後、ColorTranslator.FromOle メソッドを使用して、結果を色オブジェクトに変換し直すことができます。

以下のコード例では、元のコードと同じ結果が得られるように、System.Drawing.Color プロパティを使用して例を変更する方法を示します。

```
' Form ファイルの冒頭に Imports ステートメントを含めます。
Imports System.Drawing
…
Private Sub frmCollection_Load(ByVal eventSender As System.Object, _
                ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    Label1.BackColor = ColorTranslator.FromOle(ColorTranslator.ToOle(Color.Red) + _
                       ColorTranslator.ToOle(Color.Blue))
    Label2.BackColor = ColorTranslator.FromOle(ColorTranslator.ToOle(Color.Blue) + _
                       ColorTranslator.ToOle(Color.Black) +  _
                       ColorTranslator.ToOle(Color.Yellow))
End Sub
```

独自の定数を定義するか、コード内で非定数値を直接使用するかは、各自の判断によります。元のコードで定数値を他の定数値と組み合わせて使用している場合は、独自の定数値を定義して、コードで使用する方が理にかなっています。一方、定数値をプロパティに単純に割り当てている場合は、非定数値を直接使用する方が理にかなっています。

配列に対する変更への対応

Visual Basic 6.0 では、Option Base 宣言を使用するか、配列宣言でインデックス範囲を指定することによって、配列の開始インデックスを指定できます。Visual Basic .NET では、すべての配列の開始インデックスが 0 です。その結果、Option Base 1 宣言も、配列インデックス範囲の指定も、サポートされていません。

以下のような Visual Basic 6.0 配列宣言があるとします。

```
Option Base 1
Dim VariantArray(5) As Variant '5 elements (1-5)

' 配列宣言で配列インデックス範囲を指定します。
Dim StringArray(1 To 100) As String    '100 elements
Dim LongArray(15 To 25) As Long        '11 elements
Dim IntArray(-10 To 10) As Integer     '21 elements
```

前の宣言はいずれも、Visual Basic .NET ではサポートされていません。すべての配列宣言で、開始インデックスが 0 である必要があります。このため、配列宣言の開始インデックスが 0 以外であるコードは、調整する必要があります。そのような宣言が含まれたコードにアップグレード ウィザードを適用すると、アップグレード後のコードには、宣言のある箇所に UPGRADE_WARNING コメントが挿入されます。さらに、元のコードの開始インデックスに応じて、変更される宣言もあれば、変更されない宣言もあります。

以下のコード例は、前の例の宣言にアップグレード ウィザードを適用した結果を示します。

```
' UPGRADE_WARNING: 配列 VariantArray の下限が 1 から 0 に変更されました。
Dim VariantArray(5) As Object '6 elements (0-5)

' 配列宣言で配列インデックス範囲を指定します。
' UPGRADE_WARNING: 配列 StringArray の下限が 1 から 0 に変更されました。
Dim StringArray(100) As String '101 elements
' UPGRADE_WARNING: 配列 LongArray の下限が 15 から 0 に変更されました。
Dim LongArray(25) As Integer '26 elements
' UPGRADE_ISSUE: 宣言の型がサポートされていません : 下限が
' 0 未満の配列
Dim IntArray(-10 To 10) As Short 'Invalid array declaration
```

メモ：前に示したアップグレード後のコード内のコメントは、各配列に含まれる要素数の変化を反映して、アップグレード後に変更されています。実際のコメントは、元の Visual Basic 6.0 のコードから変更されません。

各問題を順に説明します。

Option Base 1 を使用しているコードに対しては、アップグレード ウィザードで以下の UPGRADE_WARNING メッセージが発行されます。

```
' UPGRADE_WARNING: 配列 VariantArray の下限が
' 1 から 0 に変更されました。
```

この UPGRADE_WARNING は、下限が指定されていない配列宣言ごとに追加されます。配列境界が変更された影響で、VariantArray は、0 ～ 5 にインデックス付けされた 6 つの要素を持つようになりました。元のコードで開始インデックスが指定されていないすべての配列も同様です。アップグレード後のバージョンはすべて、インデックス 0 で始まり、元の終了インデックスで終わります。

正の下限インデックスを持つインデックス範囲が指定されている配列宣言も、同様に変更されています。アップグレード ウィザードによって、宣言から下限インデックスが削除され、終了インデックスだけが残されます。こうして、アップグレード後の StringArray 宣言および LongArray 宣言には、終了インデックスだけが残されています。この影響で、アップグレード後の配列内の要素数が、元の開始インデックスと同じ数だけ増加しています。たとえば、開始インデックスを 1 として宣言されていた StringArray は、インデックス 0 から始まるようになり、要素が 1 つだけ増えています。元の開始インデックスが 15 であった LongArray は、アップグレード後の配列ではインデックス 0 から始まり、結果的に配列要素が 15 追加されています。

これらの各例では、アップグレード後の宣言をコンパイルおよび実行できます。ただし、元のコードで配列の下限に対する動的参照を使用している場合、注意が必要です。アップグレード後の配列に元の配列より多くの要素が含まれているためです。LongArray に項目カウンタを格納する、以下のような Visual Basic 6.0 ループがあるとします。

```
Dim i As Integer
Dim Index As Integer
i = 1
For Index = LBound(LongArray) To UBound(LongArray)
   LongArray(Index) = i
   i = i + 1
Next
```

このコードをアップグレードしても、アップグレード ウィザードによる変更はありません。しかし、動作は異なります。元のコードでは 11 個だけの要素 (インデックス 15 ～ 25) の格納が行われたのに対し、アップグレード後のバージョンでは 26 個の要素 (インデックス 0 ～ 25) の格納が行われるためです。

負の下限を持つ Visual Basic 6.0 配列宣言は、アップグレード ウィザードによって変更されません。ただし、配列宣言での配列インデックス範囲の指定がサポートされなくなったため、アップグレード後のコードにそのような宣言があると、コンパイル時にエラーが生成されます。したがって、そのような配列宣言は、0 の下限を使用するように手動で変更する必要があります。さらに、配列を使用していたコードを変更して、負のインデックス参照を削除する必要がある場合もあります。

前に示した例の 4 番目の配列宣言を見てください。IntArray の配列宣言の開始インデックスは -10 です。この宣言はアップグレード ウィザードによって変更されず、その結果、アップグレード後のコードはコンパイルできません。新しいコードの配列宣言を手動で変更して、元のコードと同数の要素が含まれるようにする必要があります。この例のインデックス -10 ～ 10 の配列には、21 個の要素があります。新しい宣言は以下のようにする必要があります。

```
Dim IntArray(20) As Integer
```

この新しい宣言にある配列要素の長さは、インデックス 0 ～ 20 の 21 個です。

また、負の配列インデックスへの参照がないかどうかを確認する必要もあります。たとえば、元のコードに、IntArray の内容を処理するインデックスがハードコーディングされた、以下のようなループがあるとします。

```
Dim i As Integer
For i = -10 To 10
   IntArray(i) = i
Next
```

配列宣言を調整すればコードのコンパイルは可能ですが、前のループの実行を試みると、実行時例外 System.IndexOutOfRangeException が発生します。-10 が、この配列の配列インデックスの範囲外だからです。変更方法はニーズによって異なります。たとえば、この配列に値 -10 ～ 10 を格納する必要がある場合、コードを調整して配列インデックスを 10 オフセットするようにできます。これを以下に示します。

```
Dim i As Integer
For i = -10 To 10
   IntArray(i + 10) = i
Next
```

配列インデックスが使用されているすべての箇所でこのようなコードの置換を実行して、負のインデックスが絶対に使用されないようにしてください。

また、配列の代わりに、.NET Framework で提供されている新しいコレクションを使用することもできます。ArrayList、SortedList、Queue、Stack、HashTable などです。これらのコレクションの一部を使用すると、キーまたはインデックスを使って要素にアクセスできます。このため、元のインデックス構造をコレクション内の項目のキー値として割り当てて、新しいゼロベースのインデックスの代わりに、これらのキーを使用して引き続

きこれらの項目を参照できます。たとえば、IntArray の配列宣言を SortedList コレクションに置換して、元のインデックスをキーとして項目を格納できます。これを以下に示します。

```
Dim IntList As New Collections.SortedList(20)
Dim i As Integer
For i = -10 To 10
   IntList.Add(i,i)
Next
```

Visual Basic .NET で使用可能な新しいコレクションの詳細については、MSDN の『.NET Framework Class Library』の「System.Collections Namespace」を参照してください。

従来の Visual Basic 言語機能

Visual Basic 6.0 は、多くの言語機能について、Visual Basic の最初のバージョンとのコード互換性を維持しています。これらの機能には、DefInt、VarPtr、GoSub…Return などの要素があります。これらを始めとする機能は、Visual Basic 6.0 での使用が推奨されていません。

Visual Basic .NET では、これらの機能の一部が削除されました。わかりにくいコード スタイルにつながり、現在のビジネス環境で主流になりつつある大規模分散アプリケーション開発には適さないためです。

これらの機能が削除されることになったのは、可能な限り迅速かつ有益な方法でこの言語の進歩を図るためです。DefInt、DefStr、DefObj、DefDbl、DefLng、DefBool、DefCur、DefSng、DefDec、DefByte、DefDate、DefVar、GoTo、Imp、および Eqv の Visual Basic 6.0 言語機能は、アップグレード ウィザードによってアップグレードされますが、アプリケーションで使用しないことをお勧めします。コードのわかりやすさとメンテナンスのしやすさの面からも、これらの機能の使用を避けることをお勧めします。DefType ステートメントはアップグレード ツールによって自動的にアップグレードされ、ターゲット コードに生成されるコメントで、DefType ステートメントが削除されたことと、影響を受ける変数の明示的宣言が追加されたことが示されます。アップグレード ウィザードは可能な限りこの状況を処理しますが、Visual Basic .NET 言語でサポートされなくなった古い機能は、アップグレード前に元のソース コードから削除することをお勧めします。

アップグレード ウィザードによってアップグレードされないため、開発者が削除する必要がある機能は、GoSub…Then、LSet、VarPtr、ObjPtr、StrPtr、Null、および Empty です。表 8.1 は、旧形式の機能と、アップグレード時に適用できる推奨代替機能の一覧です。

表 8.1: Visual Basic .NET で旧形式とされたキーワード、関数、およびステートメント

Visual Basic 6.0	Visual Basic .NET の代替機能
文字列関数	ドル記号 ($) を追加した文字列を返す Visual Basic の関数は、オーバーロードされたメソッドに置き換えられました。**Variant** を返す関数は、**Object** を返すオーバーロードされたメソッドに置き換えられました。
Array	配列の宣言で {} を使用して、配列に値を割り当てることができます。
As Any	Visual Basic .NET では declare のオーバーロードがサポートされているため、データ型を明示的に使用する必要があります。
Calendar	**System.Globalization** 名前空間で使用可能です。
Currency	**Decimal** データ型に置き換えられました。**ToString("C")** メソッドを使用して、現在のカルチャ設定に応じた通貨として、**Decimal** 値を表示できます。
Date	データ型 (**System.DateTime** へのマップ) としてはサポートされていますが、4 バイトのデータ値を返す関数ではなくなりました。8 バイトの CLR 形式で日付を返すには、**System.DateTime** 構造体の **Today** プロパティを使用してください。
Debug.Assert、Debug.Print	**System.Diagnostics** 名前空間のメソッドに置き換えられました。
Def<*Type*>	サポートされなくなりました。すべての変数に明示的宣言が必要です。
DoEvents	**System.Windows.Forms.Application** クラスのメソッドに置き換えられました。
Empty、Null、IsEmpty、IsNull	すべて、**Nothing** キーワードによって処理されます。これらの条件を確認するステートメントも削除されました。データベースの null のテストには、**IsDBNull** 関数または **System.DBNull.Value** プロパティを使用できます。
Eqv、Imp	等号 (=) 演算子を **Not** および **Or** と組み合わせて使用してください。たとえば、**A Imp B** は **(Not A) Or B** に置き換えてください。
GoSub	サポートされなくなりました。アップグレード前に元のコードからすべて削除してください。
IsMissing	Visual Basic .NET では、すべての省略可能なパラメータに既定値が必要です。
IsObject	**IsReference** に置き換えられました。
Let、Set	パラメータのない既定のプロパティがサポートされなくなったため、これらのステートメントが削除されました。
LSet、RSet	**System.String** クラスの **PadRight** メソッドおよび **PadLeft** メソッドに置き換えられました。引き続きサポートはされていますが、**LSet** を使用してユーザー定義型 (UDT) をコピーすることはできなくなりました。Visual Basic .NET で UDT に相当するのは構造体です。構造体を直接コピーすることはできませんが、メンバを 1 つずつコピーすれば同じ結果を実現できます。 UDT を使用したファイル レコードへのアクセスの詳細については、第 11 章「文字列操作とファイル操作のアップグレード」の「ユーザー定義データ型を使用した固定長レコードへのアクセス」、および第 6 章「Visual Basic アップグレード ウィザードについて」の「ユーザー定義型」を参照してください。

Visual Basic 6.0	Visual Basic .NET の代替機能
PSet、Scale	サポートされなくなりました。**System.Drawing** に同等の機能があります。
Option Private Module	**Private** キーワードを使用して、任意のモジュールをプライベートとして設定できます。
Time	**System.DateTime** 構造体の **TimeOfDay** プロパティに置き換えられました。**Date** データ型は **Double** として表されません。代わりに、**DateTime** 構造体として表されます。
Type	**Structure** ステートメントに置き換えられました。
VarType	引き続き存在しますが、特定のデータ型の **GetTypeCode** メソッドに置き換えてください。
Variant	ユニバーサル データ型である **Object** に置き換えられました。
Wend	**End While** に置き換えられました。

アドインのアップグレード

Visual Basic の拡張性オブジェクト モデルでは、アドインの使用により開発を容易にできます。アドインは、拡張性モデルにおいてオブジェクトとコレクションを使用してプログラムで作成するツールで、Visual Basic 統合開発環境 (IDE) をカスタマイズおよび拡張します。ある意味で、アドインは Visual Basic IDE に "スナップ留め" されます。作成方法にかかわらず、アドインの主な目的は、開発環境内で、手動で実行するには困難、単調、または時間のかかる作業を自動化することです。アドインは、時間と労力を省くための、Visual Basic プログラミング環境用自動化ツールです。

アドインは Visual Studio .NET でも引き続きサポートされていますが、異なる点があるため、Visual Basic 6.0 からのアップグレード時にプロジェクトを変更する必要があります。ここでは、アドイン コード例を使用して、必要な調整のいくつかを説明します。

ここで使用するアドイン例は単純ですが、通常のアドインのアップグレードで必要となる基本的な変更の多くを説明するのに十分な機能を備えています。アドイン プロジェクトは LabAddIn という名前で、Connect という名前の**クラス モジュール**と、プロジェクトと同じ名前のリソース ファイルとが含まれています。このリソース ファイルには、ID 101 に関連付けられたビットマップが含まれています。アドインをコンピュータに登録した後は、Visual Studio 6.0 を読み込むたびに、Visual Basic の標準ツール バーにアドインのボタンが表示されます。このボタンをクリックすると、Option Explicit ディレクティブが、現在のソース コード ドキュメントの冒頭にクリッ

プボード オブジェクトとして挿入されます。Visual Basic 6.0 アドインの作成と登録の詳細については、MSDN の「Creating a Basic Add-In」を参照してください。

アドインの Connect クラス モジュールのコードは、以下のとおりです。

```
Implements IDTExtensibility

Dim MenuCommandBar As Office.CommandBarControl
Public WithEvents MenuHandler As CommandBarEvents
Dim CurrentVBInstance As VBIDE.VBE
Dim LastButton As Long
Private Sub IDTExtensibility_OnAddInsUpdate(custom() As Variant)
' アドインの必須プロシージャ
End Sub
Private Sub IDTExtensibility_OnConnection(ByVal VBInst As Object, _
            ByVal ConnectMode As VBIDE.vbext_ConnectMode, _
            ByVal AddInInst As VBIDE.AddIn, custom() As Variant)

    Set CurrentVBInstance = VBInst
    CurrentVBInstance.CommandBars("Standard").Visible = True
    LastButton = CurrentVBInstance.CommandBars("Standard").Controls.Count
    Set MenuCommandBar = CurrentVBInstance.CommandBars("Standard").Controls.Add(1, _
                                                        , , LastButton)
    Clipboard.SetData LoadResPicture(101, vbResBitmap)
    MenuCommandBar.PasteFace
    MenuCommandBar.ToolTipText = "Lab AddIn"
    Set Me.MenuHandler = CurrentVBInstance.Events.CommandBarEvents(MenuCommandBar)
End Sub

Private Sub IDTExtensibility_OnDisconnection(ByVal RemoveMode As _
            VBIDE.vbext_DisconnectMode, custom() As Variant)
    CurrentVBInstance.CommandBars("Standard").Controls.Item(LastButton).Delete
End Sub

Private Sub IDTExtensibility_OnStartupComplete(custom() As Variant)
    ' アドインの必須プロシージャ
End Sub

Private Sub MenuHandler_Click(ByVal CommandBarControl As Object, _
            handled As Boolean, CancelDefault As Boolean)
    Screen.MousePointer = vbHourglass
    On Error GoTo ErrorHandler
    Call CurrentVBInstance.ActiveCodePane.CodeModule.InsertLines(1, _
         "Option Explicit" + Chr(13))
    Screen.MousePointer = vbDefault
    Exit Sub
ErrorHandler:
    MsgBox Err.Description, vbCritical
    Screen.MousePointer = vbDefault
End Sub
```

このコードにアップグレードウィザードを適用すると、以下が生成されます。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("Connect_NET.Connect")> _
Public Class Connect
    Implements VBIDE.IDTExtensibility

    Dim MenuCommandBar As Microsoft.Office.Core.CommandBarControl
    Public WithEvents MenuHandler As VBIDE.CommandBarEvents
    Dim CurrentVBInstance As VBIDE.VBE
    Dim LastButton As Integer
    Private Sub IDTExtensibility_OnAddInsUpdate(ByRef custom As System.Array) _
                Implements VBIDE.IDTExtensibility.OnAddInsUpdate
        ' アドインの必須プロシージャ
    End Sub
    Private Sub IDTExtensibility_OnConnection(ByVal VBInst As Object, _
                                ByVal ConnectMode As VBIDE.vbext_ConnectMode, _
                                ByVal AddInInst As VBIDE.AddIn, _
                                ByRef custom As System.Array) _
                                Implements VBIDE.IDTExtensibility.OnConnection

        CurrentVBInstance = VBInst
        CurrentVBInstance.CommandBars("Standard").Visible = True
     LastButton = CurrentVBInstance.CommandBars("Standard").Controls.Count
     MenuCommandBar = CurrentVBInstance.CommandBars("Standard").Controls.Add( _
                         1,  ,  , LastButton)
        ' UPGRADE_ISSUE: Clipboard メソッド Clipboard.SetData はアップグレードされませんでした。
        Clipboard.SetData(VB6.LoadResPicture(101, VB6.LoadResConstants.ResBitmap))

        ' UPGRADE_WARNING: Could not resolve default property of object
        ' MenuCommandBar.PasteFace.
        MenuCommandBar.PasteFace()
        MenuCommandBar.ToolTipText = "Lab AddIn"
        Me.MenuHandler = CurrentVBInstance.Events.CommandBarEvents(MenuCommandBar)
    End Sub

    Private Sub IDTExtensibility_OnDisconnection( _
            ByVal RemoveMode As VBIDE.vbext_DisconnectMode, _
            ByRef custom As System.Array) _
            Implements VBIDE.IDTExtensibility.OnDisconnection
        ' UPGRADE_WARNING: Could not resolve default property of object
        ' CurrentVBInstance.CommandBars().Controls.Item().Delete.
        CurrentVBInstance.CommandBars("Standard").Controls.Item(LastButton).Delete()
    End Sub

    Private Sub IDTExtensibility_OnStartupComplete(ByRef custom As System.Array) _
                            Implements VBIDE.IDTExtensibility.OnStartupComplete
        ' アドインの必須プロシージャ
    End Sub
```

```
    Private Sub MenuHandler_Click(ByVal CommandBarControl As Object, _
                                  ByRef handled As Boolean, _
                                  ByRef CancelDefault As Boolean) _
                                  Handles MenuHandler.Click

        ' UPGRADE_WARNING: Screen プロパティ Screen.MousePointer には新しい動作が含まれます。
        System.Windows.Forms.Cursor.Current =  _
           System.Windows.Forms.Cursors.WaitCursor
        On Error GoTo ErrorHandler
        Call CurrentVBInstance.ActiveCodePane.CodeModule.InsertLines(1, _
                        "Option Explicit" & Chr(13))
        ' UPGRADE_WARNING: Screen プロパティ Screen.MousePointer には新しい動作が含まれます。
        System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.Default
    Exit Sub
ErrorHandler:
        MsgBox(Err.Description, MsgBoxStyle.Critical)
        ' UPGRADE_WARNING: Screen プロパティ Screen.MousePointer には新しい動作が含まれます。
        System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.Default
    End Sub
End Class
```

このコードに関するコンパイル エラーが、タスク一覧に表示されます。この章で既に説明したとおり、Clipboard オブジェクトが変更されたことによるエラーです。問題の命令を以下のように変更すれば、エラーを修正できます。

```
' UPGRADE_ISSUE: Clipboard メソッド Clipboard.SetData はアップグレードされませんでした。
Clipboard.SetData(VB6.LoadResPicture(101, VB6.LoadResConstants.ResBitmap))
```

Visual Basic .NET での Clipboard の変更に対処するには、このステートメントを以下のコードに置換します。

```
Dim d As New System.Windows.Forms.DataObject(VB6.LoadResPicture (101, _
            VB6.LoadResConstants.ResBitmap))
System.Windows.Forms.Clipboard.SetDataObject(d)
```

Clipboard コードの修正に加えて、VBIDE ライブラリおよび Microsoft.Office.Core ライブラリへの参照も修正する必要があります。これらは、Visual Basic IDE 機能へのアクセスを有効にするライブラリです。これらのライブラリは、.NET で代替ライブラリに置き換えられました。これらのライブラリの新しいバージョンを使用するには、まずこれらの参照を削除する必要があります。次に、EnvDTE、Extensibility、Office の各ライブラリへの参照を追加します。これらのライブラリは、Visual Studio .NET の拡張性機能を有効にして、IDE の現在のインスタンスにアクセスできるようにします。**[プロジェクト]** メニューの **[参照の追加]** をクリックすると、**[参照の追加]** ダイアログ ボックスにアクセスできます。

参照にこれらの変更を加えると、VBIDE ライブラリ内の関数呼び出しが原因で、新しいコンパイル エラーがコード内に発生します。アップグレードの次のステップでは、これらのエラーを修正するため、参照するようにしたライブラリ内の呼び出しを代替バージョンに置換します。たとえば、以下のようなコード行でエラーが発生します。

```
Implements VBIDE.IDTExtensibility
```

以下に示す 2 行の Visual Basic .NET のコードに置換すると、このエラーを修正できます。

```
Implements Extensibility.IDTExtensibility2
Implements EnvDTE.IDTCommandTarget
```

この変更によって VBIDE ライブラリの拡張性インターフェイスが置換され、Visual Studio .NET IDE のグラフィカル インターフェイスの諸要素を変更できるようになります。

これらの変更だけでは、まだアップグレードが完了しません。次のステップでは、EnvDTE、Extensibility、System.Runtime.InteropServices の各ライブラリをインポートします。これらのライブラリをインポートするには、Connect クラスの宣言の前に以下のコード行を追加します。

```
Imports EnvDTE
Imports Extensibility
Imports System.Runtime.InteropServices
```

表 8.2 に、OnAddInsUpdate、OnConnection、OnDisconnection、OnStartUpComplete の各イベントで必要なコード置換を示します。

表 8.2: コード イベントの置換表現

元のコード	置換表現
VBIDE.IDTExtensibility	Extensibility.IDTExtensibility2
VBIDE.vbext_ConnectMode	Extensibility.ext_ConnectMode
VBIDE.AddIn	Object
VBEIDE.vbext_DisconnectMode	Extensibility.ext_ConnectMode

変更が適用された結果のコードは以下のとおりです。

```
Private Sub IDTExtensibility_OnAddInsUpdate(ByRef custom As System.Array) _
            Implements Extensibility.IDTExtensibility2.OnAddInsUpdate

Private Sub IDTExtensibility_OnConnection(ByVal VBInst As Object, _
            ByVal ConnectMode As Extensibility.ext_ConnectMode, _
            ByVal addInInst As Object, ByRef custom As System.Array) _
            Implements Extensibility.IDTExtensibility2.OnConnection

Private Sub IDTExtensibility_OnDisconnection( _
            ByVal RemoveMode As Extensibility.ext_DisconnectMode, _
            ByRef custom As System.Array) Implements _
            Extensibility.IDTExtensibility2.OnDisconnection

Private Sub IDTExtensibility_OnStartupComplete(ByRef custom As System.Array) _
            Implements Extensibility.IDTExtensibility2.OnStartupComplete
```

アップグレード後のアドインには、OnBeginShutdown イベント ハンドラが含まれている必要があります。以下のコード例を使用できます。

```
Public Sub IDTExtensibility_OnBeginShutdown(ByRef custom As System.Array) _
          Implements Extensibility.IDTExtensibility2. OnBeginShutdown
   ' アドインに必須
End Sub
```

以下のコードを Visual Studio .NET IDE で使用すると、接続コマンドが処理されて、プラグインに送信されます。これによってプラグインとの接続が可能になり、対応するコントロールがユーザー インターフェイスに展開されます。

```
Public Sub Exec(ByVal cmdName As String, _
          ByVal executeOption As vsCommandExecOption, ByRef varIn As Object, _
          ByRef varOut As Object, ByRef handled As Boolean) _
          Implements IDTCommandTarget.Exec
    handled = False
    If (executeOption = vsCommandExecOption.vsCommandExecOptionDoDefault) Then
        If cmdName = "LabAddIn.Connect" Then
            handled = True
            Exit Sub
        End If
     End If
End Sub

Public Sub QueryStatus(ByVal cmdName As String, _
          ByVal neededText As vsCommandStatusTextWanted, _
          ByRef statusOption As vsCommandStatus, _
          ByRef commandText As Object) _
          Implements IDTCommandTarget.QueryStatus
    If neededText = _
          EnvDTE.vsCommandStatusTextWanted.vsCommandStatusTextWantedNone Then
        If cmdName = "LabAddIn.Connect" Then
            statusOption = CType(vsCommandStatus.vsCommandStatusEnabled + _
                                 vsCommandStatus.vsCommandStatusSupported, _
                                 vsCommandStatus)
        Else
            statusOption = vsCommandStatus.vsCommandStatusUnsupported
        End If
    End If
End Sub
```

新しいコードにも、コンパイル エラーが 2 つ存在します。以下に示すように、削除された VBIDE の代わりに EnvDTE ライブラリを参照することによって、これらを解決できます。

```
Public WithEvents MenuHandler As CommandBarEvents
Dim CurrentVBInstance As DTE
```

次に、正しいアドイン プロジェクト名を使用するように ProgId クラス属性を変更する必要があります。自動的なアップグレード プロセスでは、以下のクラス宣言命令が使用され、この属性に Connect_NET.Connect が設定されます。

```
<System.Runtime.InteropServices.ProgId("Connect_NET.Connect")>
```

属性を正しい名前に変更してください。この例では、名前を LabAddIn.Connect に変更します。前の例が以下のように変更されます。

```
<System.Runtime.InteropServices.ProgId("LabAddIn.Connect")>
```

最後に、MenuHandler.Click イベントの以下の行を変更する必要があります。

```
Call CurrentVBInstance.ActiveCodePane.CodeModule.InsertLines(1, _
                            "Option Explicit" & Chr(13))
```

以下の命令を使用して、前のコード行を置換してください。

```
Dim Txt As TextDocument = CurrentVBInstance.ActiveDocument.Object()
Txt.CreateEditPoint().Insert("Option Explicit On" & Chr(13))
```

この最後の変更によって、プロジェクトの構築の準備が整いました。これで、アドインが正しくコンパイルできるように、また、Visual Studio .NET IDE のプラグインとして使用できるようになりました。実行するには、登録して機能を使用可能にする必要があります。アドインを登録するには、以下の手順を実行します。

1. アドイン用のダイナミックリンクライブラリ (DLL) を構築します。
2. コマンド プロンプトを開きます。
3. コマンド プロンプトで「regasm name.dll」と入力します。name はアドイン名です。
4. レジストリを編集します。regedit コマンドを使用して、レジストリ エディタを起動できます。
5. キー HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\7.1\AddIns を見つけます。キー AddIns がまだ作成されていない場合は、作成する必要があります。
6. ProgId クラス属性と同じ名前のキーを追加します。
7. 新しいキーに、文字列の値 FriendlyName および Description を追加します。
8. FiendlyName オプションに、アドインの表示名を設定します。
9. Description オプションに、アドインを説明するコメントを設定します。
10. Visual Studio .NET を読み込みます。
11. [ツール] メニューの [アドイン マネージャ] をクリックします。
12. アドインを一覧から選択して、Visual Studio .NET IDE に追加します。

これらの手順によって、Visual Basic 6.0 IDE と同様に Visual Studio .NET IDE をカスタマイズできます。

まとめ

Visual Basic 6.0 言語の複数の機能が、Visual Basic .NET で変更または削除されました。しかし、これらの機能が含まれるコードをアップグレードできないという意味ではありません。多くの場合にコードを手動で調整する必要があるという意味です。

機能変更への対処方法を理解するには、どの機能が変更されたのかを知ることが重要です。変更の内容やアップグレードへの影響を知ることも、同じく重要です。

この章に記載された各機能には、手動での修正が必要です。この章では、これらの問題への対処方法について説明しています。これらの問題の解決には複数の方法があり得ます。ある方法がふさわしくない場合は、Web 検索でより適切な解決方法が見つかる可能性があります。

詳細情報

Visual Basic .NET のパラメータ化された既定のプロパティの詳細については、MSDN の『Visual Basic Language Concepts』の「Default Property Changes in Visual Basic」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vaconDefaultProperties.asp) を参照してください。

Visual Basic .NET で使用可能な新しいコレクションの詳細については、MSDN の『.NET Framework Class Library』の「System.Collections Namespace」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemcollections.asp) を参照してください。

Visual Basic 6.0 アドインの作成と登録の詳細については、MSDN の「Creating a Basic Add-In」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon98/html/vbconcreatingaddin.asp) を参照してください。

9

Visual Basic 6.0 フォーム機能のアップグレード

多くの Microsoft Windows ベースのアプリケーションはフォームに基づいています。Microsoft Visual Basic の主要な利点の 1 つは、フォームを簡単に構築できることです。このため、Microsoft Visual Basic は迅速なアプリケーション開発の共通言語となっています。

Visual Basic 6.0 以降、基本的なフォーム アーキテクチャに大きな変更が加えられています。Visual Basic .NET のフォーム アプリケーションは、Windows フォーム アーキテクチャで作成されます。Windows フォームは、Visual Basic 6.0 で使用できるアーキテクチャに比べ、拡張性の高いアーキテクチャですが、このアーキテクチャの違いにより、アプリケーションのアップグレード時には調整が必要です。この章では、フォームにおけるいくつかの変更点、およびフォーム アプリケーションをアップグレードする際に必要な作業について説明します。

グラフィックス操作に対する変更の処理

Visual Basic 6.0 には、フォーム上に線や形状を描画するための多数のグラフィックス メソッドがあります。また、他のコントロールに描画メソッドを適用できる、PictureBox コントロールおよび Image コントロールもあります。Visual Basic .NET での描画操作の実行方法にはいくつかの変更が加えられています。以降では、これらの変更点について説明します。

Visual Basic .NET での Line コントロールの削除

Visual Basic 6.0 の Line コントロールを使用すると、フォームに線分を描画できます。Visual Basic .NET には対応するコントロールはありません。アップグレード ウィザードが適用されるときに、垂直および水平の Line コントロールは、Windows フォームの Label コントロールに置き換えられます。このとき、Text プロパティは空の文字列に、BorderStyle プロパティは None に設定され、BackColor、Width、および Height の各プロパティは元のコントロールと一致するように設定されます。垂直、水平のいずれでもない Line コントロールはアップグレードされません。それらの線は、アップグレード ウィザードによって Label コントロールに置き換えられます。このとき、BackColor プロパティが Red に設定されて該当箇所が強調表示されます。

Visual Basic .NET に組み込まれたグラフィックス関数を使用して、Line コントロールを Paint イベントで描画した線に置き換えることができます。次の例では、Visual Basic .NET で対角線を描画する方法を示します。

```
Private Sub Form1_Paint(ByVal sender As Object, _
        ByVal e As System.Windows.Forms.PaintEventArgs) _
        Handles MyBase.Paint
    ' Pen オブジェクトを作成します。
    Dim pen As New Drawing.Pen(System.Drawing.Color.Brown, 1)
    e.Graphics.DrawLine(pen, 0, 0, 100, 100)
    pen.Dispose()
End Sub
```

Line コントロールを使用するよりも DrawLine メソッドを使用する方が多くの制御ができますが、Line コントロールは垂直または水平ではない線には必要です。また、Line コントロールは、BorderWidth や BorderStyle などの属性値に対して既定以外の値が設定されている線にも必要です。さらに、Label コントロールは、Line などのグラフィックス オブジェクトとは大きく異なります。このコントロールはアップグレードされたプロジェクトで最初は同じように表示される場合もありますが、動作は完全に同じではありません。Line コントロールを Label コントロールにアップグレードする方法は、DrawLine メソッドを使用して再実装するための一時的なソリューションである点に注意してください。

Visual Basic .NET での Shape コントロールの削除

Visual Basic .NET には、Visual Basic .6.0 の Shape コントロールに対応するコントロールがありません。コード内の長方形または正方形の Shape コントロールは、Windows フォームの Label コントロールに置き換えられます。このとき、BorderStyle は FixedSingle に設定され、BackColor、Width、および Height の各プロパティは元のコントロールと一致するように設定されます。楕円および円の Shape コントロールはアップグレードできません。このコントロールは、プレースホルダとして BackColor プロパティが Red に設定された Label コントロールに置き換えられます。

Shape コントロールは、Paint イベントの描画メソッドに置き換えることができます。この方法はどの Shape コントロールにも使用できますが、楕円および円の Shape コントロールでは必須です。以下のコード例は、Visual Basic .NET で円を描画する方法を示します。

```
Private Sub Form1_Paint(ByVal sender As Object, _
                        ByVal e As System.Windows.Forms.PaintEventArgs) _
                        Handles MyBase.Paint
   ' Pen オブジェクトを作成します。
   Dim pen As New Drawing.Pen(System.Drawing.Color.Brown, 1)
   e.Graphics.DrawEllipse(pen, 0, 0, 100, 100)
   pen.Dispose()
End Sub
```

メモ : 元のアプリケーションと同じ外観のレイアウトを得るには、ZOrder メソッドを使用することが必要になる場合があります。

表 9.1 は、Visual Basic 6.0 のグラフィックスのプロパティとメソッドと、Visual Basic .NET の対応する機能をまとめたものです。

表 9.1: グラフィックスのプロパティとメソッド

Visual Basic 6.0	Visual Basic .NET の対応する機能
AutoRedraw プロパティ	対応する機能はありません。永続的なグラフィックスを作成するには、**Paint** イベント ハンドラにグラフィックス メソッドを記述します。
Circle メソッド	**Graphics.DrawEllipse** メソッド
ClipControls プロパティ	対応する機能はありません。オブジェクトの一部分だけを再描画するには、**Graphics.SetClip** メソッドを使用して **Paint** イベント ハンドラにクリップを作成します。
Cls メソッド	**Graphics.Clear** メソッド
CurrentX プロパティ	各種グラフィックス メソッドの **x** パラメータ。たとえば、**DrawRectangle**(pen, x, y, width, height) のようにします。
CurrentY プロパティ	各種グラフィックス メソッドの **y** パラメータ。たとえば、**DrawRectangle**(pen, x, y, width, height) のようにします。
DrawMode プロパティ	**Pen.Color** プロパティ
DrawStyle プロパティ	**Pen.PenType** プロパティ
DrawWidth プロパティ	**Pen.Width** プロパティ
FillColor プロパティ	**SolidBrush.Color** プロパティ
FillStyle プロパティ	**Pen.Brush** プロパティ
HasDC プロパティ	対応する機能はありません。GDI+ では、デバイス コンテキストは不要になりました。
HDC プロパティ	対応する機能はありません。GDI+ では、デバイス コンテキストは不要になりました。
Image プロパティ	**Image** プロパティ。Visual Basic 6.0 の **Image** プロパティはハンドルを返しましたが、Visual Basic .NET では、**System.Drawing.Image** オブジェクトを返します。
Line メソッド	**Graphics.DrawLine** メソッド
PaintPicture メソッド	**Graphics.DrawImage** メソッド
Point メソッド	対応するフォームまたはコントロールはありません。ビットマップの場合は、**Bitmap.GetPixel** メソッドを使用します。
Print メソッド	**Graphics.DrawString** メソッド
Pset メソッド	対応するフォームまたはコントロールはありません。ビットマップの場合は、**Bitmap.SetPixel** メソッドを使用します。
TextHeight プロパティ	**Graphics.MeasureString** メソッド。このメソッドは、指定したフォントで描画した場合の、指定された文字列のサイズ (ピクセル単位) を表す **System.Drawing.SizeF** 構造を返します。**SizeF.Height** プロパティは、テキストの高さを取得するために使用できます。
TextWidth プロパティ	**Graphics.MeasureString** メソッド。このメソッドは、指定したフォントで描画した場合の、指定された文字列のサイズ (ピクセル単位) を表す **System.Drawing.SizeF** 構造を返します。**SizeF.Width** プロパティは、テキストの幅を取得するために使用できます。

PopupMenu メソッドに対する変更の処理

Visual Basic 6.0 の Menu コントロールの中で最も頻繁に使用される機能の 1 つは PopupMenu メソッドです。このメソッドは、MDIForm または Form オブジェクトにポップアップ メニューを表示します。ポップアップ メニューは、現在マウスがある場所または指定された座標に表示できます。通常、このメソッドは、マウス イベントと組み合わせて使用します。

ユーザーがフォームを右クリックしたときにカーソル位置にポップアップ メニューを表示するコード例を次に示します。この例では、1 つ以上のメニュー項目がある mnuFile という名前の Menu コントロールを含むフォームがあることを前提にしています。このフォームのコードには、次の MouseDown イベント ハンドラが含まれます。

```
…
Private Sub Form_MouseDown (Button As Integer, Shift As Integer, _
                            X As Single, Y As Single)
   If Button = 2 Then
      PopupMenu mnuFile
   End If
End Sub
…
```

コードを Visual Basic .NET にアップグレードしても、PopupMenu 命令は変わりませんが、次に示すようなアップグレード警告のマークが付きます。

```
' UPGRADE_ISSUE: Form メソッド Form1.PopupMenu はアップグレードされませんでした。
```

コードをアップグレードすると、Visual Basic .NET でコンパイル時エラーが発生します。同じ機能を得るためには、アップグレードしたコードを、ContextMenu オブジェクトを使用する命令に置き換える必要があります。この方法を示すコード サンプルを次に示します。

```
Private Sub Form1_MouseDown(ByVal eventSender As System.Object, _
        ByVal eventArgs As System.Windows.Forms.MouseEventArgs) _
        Handles MyBase.MouseDown
    Dim Button As Short = eventArgs.Button \ &H100000
    Dim Shift As Short = _
        System.Windows.Forms.Control.ModifierKeys \ &H10000
    Dim X As Single = VB6.PixelsToTwipsX(eventArgs.X)
    Dim Y As Single = VB6.PixelsToTwipsY(eventArgs.Y)
    If Button = 2 Then
        Dim popupmenu As ContextMenu
        popupmenu = New ContextMenu
        popupmenu.MergeMenu(mnuFile)
        popupmenu.Show(Me, New Point(eventArgs.X, eventArgs.Y))
    End If
End Sub
```

上記のコード例では、ContextMenu オブジェクトを使用してポップアップ メニューの表示を可能にしています。MergeMenu 命令によって、mnuFile Menu コントロール内の項目を追加します。

元のコードで、指定された座標にポップアップ メニューを表示していた場合は、Show メソッドで目的の座標を指定できます。たとえば、元のポップアップ メニュー コードは次のようになります。

```
PopupMenu mnuFile, , 10, 10
```

この場合、Visual Basic .NET では表示座標を次のように指定できます。

```
popupmenu.Show(Me, New Point(10, 10))
```

Visual Basic 6.0 のポップアップ メニューのすべての機能が Visual Basic .NET で使用できるわけではありません。flags および boldcommand 引数に対応する引数はありません。Visual Basic .NET 内のメニュー項目は、所有者が描画したメニューを使用しない限り、.NET Framework のバージョン 1.1 では太字に設定できません。ただし、バージョン 2.0 にはこの機能が追加されています。ポップアップ メニューの位置は Show メソッドのパラメータで指定された位置に完全に依存し、ポップアップ メニューはマウスの左ボタンを使用した場合にのみマウスのクリックに反応します。太字の項目など追加の視覚効果を Visual Basic .NET で得るためには、MenuItem.OwnerDraw プロパティを使用してメニュー項目を所有者が描画するように指定します。この手法には、MenuItem.DrawItem イベントのユーザー定義のイベント ハンドラ、および Graphics オブジェクトを使用する必要があります。詳細については、MSDN の『.NET Framework Class Library』の「MenuItem.DrawItem Event」を参照してください。

ClipControls プロパティに対する変更の処理

クリップは、フォームが表示されるときに描画されるフォームまたはコンテナの各部分 (Frame コントロールまたは PictureBox コントロールなど) を決定するプロセスです。フォームおよびコントロールのアウトラインは、メモリ内に作成されます。Windows オペレーティング システムではこのアウトラインを使用して、背景などいくつかの部分を、TextBox コントロールのコンテンツなどの他の部分に影響を与えずに描画します。クリッピング領域はメモリ内に作成されるため、このプロパティを False に設定すると、フォームを描画または再描画する時間を短縮できます。

ClipControls プロパティは、デザイン時に設定できます。実行時に、このプロパティは、Paint イベントのグラフィックス メソッドでオブジェクト全体を再描画するか、新たに公開された領域だけを再描画するかどうかを決定する値を返します。また、Windows オペレーティング システムでオブジェクトに含まれるグラフィックス以外のコントロールを除いたクリッピング領域を作成するかどうかを決定します。

Visual Basic .NET は、ClipControls プロパティをサポートしていません。アプリケーションをアップグレードし、このプロパティをコードで参照する場合、このプロパティは変更されず、次のようなアップグレード警告のマークが付きます。

```
' UPGRADE_ISSUE: Form プロパティ Form1.ClipControls はアップグレードされませんでした。
```

この問題を修正するには、このプロパティへの参照を、Graphics.SetClip メソッドへの呼び出しに置き換える必要があります。Paint イベントにイベント ハンドラを指定して、描画するコントロールに関連付けられた Graphics オブジェクトにアクセスできます。このオブジェクトを使用して、SetClip メソッドを呼び出し、クリッピング領域を指定できます。

これは、追加の作業を伴う若干高度な手法です。カスタムの描画ルーチンを作成する前に、アップグレードしたアプリケーションの描画パフォーマンスをテストする必要があります。ClipControls プロパティは、機能が制限されているハードウェアでパフォーマンスを向上させるために使用されます。現代の高性能コンピュータおよびビデオ カードの多くは、表示のレンダリングにクリッピングの必要はありません。アップグレードするアプリケーションは、クリッピング領域の低レベル処理を行わなくても現在のターゲット システムで十分なパフォーマンスを発揮できる可能性があります。この場合は、アップグレードの代わりに、ClipControls プロパティに関連するコードを削除できます。

ドラッグ アンド ドロップ機能

ドラッグ アンド ドロップ機能は、最新のフォームベース アプリケーションの重要な機能です。Visual Basic 6.0 と Visual Basic .NET では、ドラッグ アンド ドロップ機能に多くの重要な違いがあります。ここでは、ドラッグ アンド ドロップ機能の違いがアプリケーションに与える影響をより理解できるようにその違いの概要を示します。

Visual Basic 6.0 のドラッグ アンド ドロップ機能

Visual Basic 6.0 は、2 種類のドラッグ アンド ドロップ操作をサポートしています。1 つは、単一のフォーム内のコントロール間での項目の移動をサポートする標準のドラッグ アンド ドロップ機能です。もう 1 つは、アプリケーション間での項目の移動をサポートする OLE ドラッグ アンド ドロップ機能です。ここでは、OLE ドラッグ アンド ドロップ機能に重点を置いて説明します。

標準の Visual Basic 6.0 コントロールは、OLE ドラッグ アンド ドロップ機能に対してさまざまな種類のサポートを行います。

表 9.2 は、標準の Visual Basic 6.0 コントロールと手動および自動のドラッグ アンド ドロップ操作に対するサポートをまとめたものです。

表 9.2: Visual Basic 6.0 のコントロール : OLE ドラッグ アンドドロップ機能のサポート

コントロール	OLEDragMode	OLEDropMode
TextBox、PictureBox、Image、RichTextBox、MaskedBox	VbManual、vbAutomatic	vbNone、vbManual、vbAutomatic
ComboBox、ListBox、DirListBox、FileListBox、DBCombo、DBList、TreeView、ListView、ImageCombo、DataList、DataCombo	VbManual、vbAutomatic	vbNone、vbManual
Form、Label、Frame、CommandButton、DriveListBox、Data、MSFlexGrid、SSTab、TabStrip、Toolbar、StatusBar、ProgressBar、Slider、Animation、UpDown、MonthView、DateTimePicker、CoolBar	サポートされません。	vbNone、vbManual

図 9.1 に、Visual Basic 6.0 での OLE ドラッグ アンドドロップ操作のライフ サイクルの概要を示します。

ドラッグ アンドドロップ機能は、Visual Basic .NET でもサポートされますが、違いがあります。次のセクションでは、Visual Basic .NET のドラッグ アンドドロップ機能について詳しく説明します。

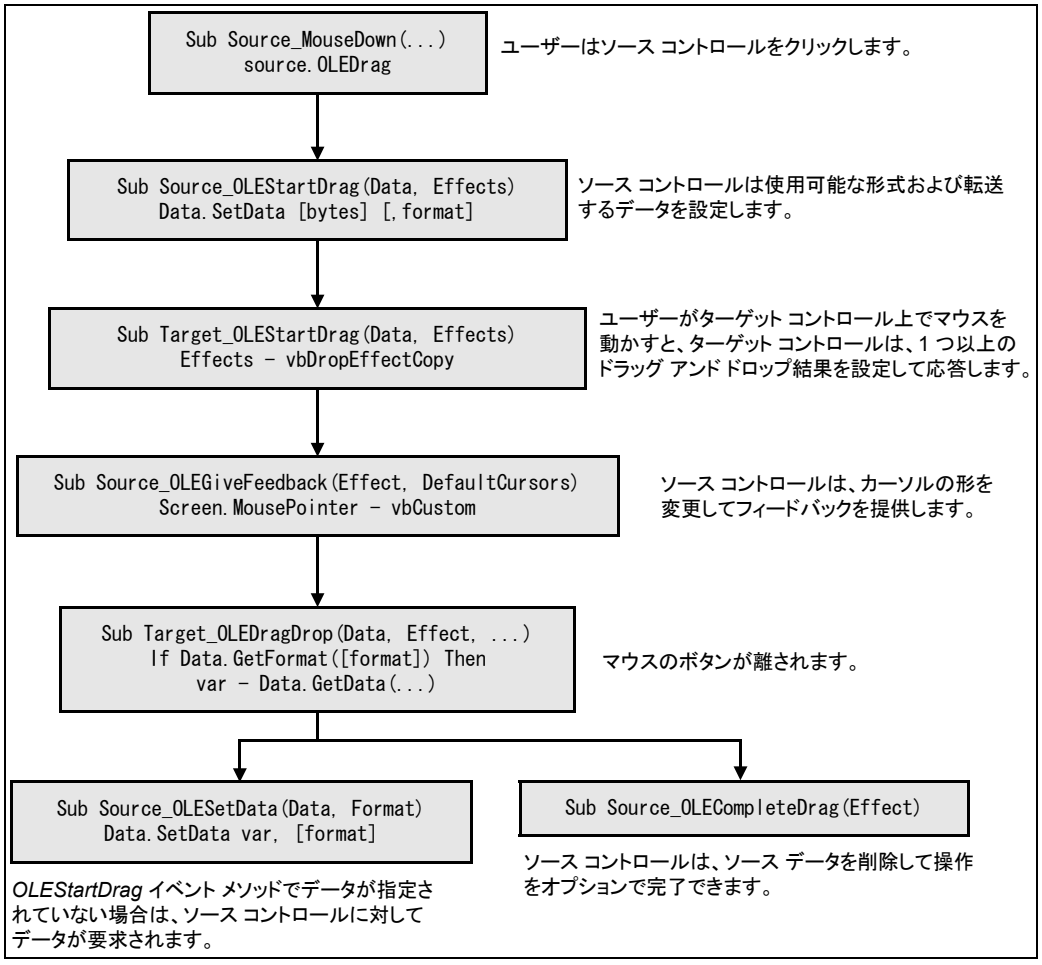

図 9.1

Visual Basic 6.0 *のドラッグ アンドドロップ操作のライフ サイクル*

Visual Basic .NET のドラッグ アンドドロップ機能

Visual Basic .NET では、ドラッグ アンドドロップ操作は単一のフレームワークに統合されています。コントロール間のドラッグ アンド ドロップ操作は、アプリケーション間のドラッグ アンド ドロップ操作と同じように処理されます。この変更によって、新しい開発ではプログラミングの負担が軽くなりますが、既存のアプリケーションに対してはドラッグ アンドドロップ コードを再作成する必要があります。

図 9.2 に、Visual Basic .NET でのドラッグ アンドドロップ操作のライフ サイクルの概要を示します。このサイクルは、コントロール間およびアプリケーション間両方のドラッグ アンドドロップ操作に適用されます。

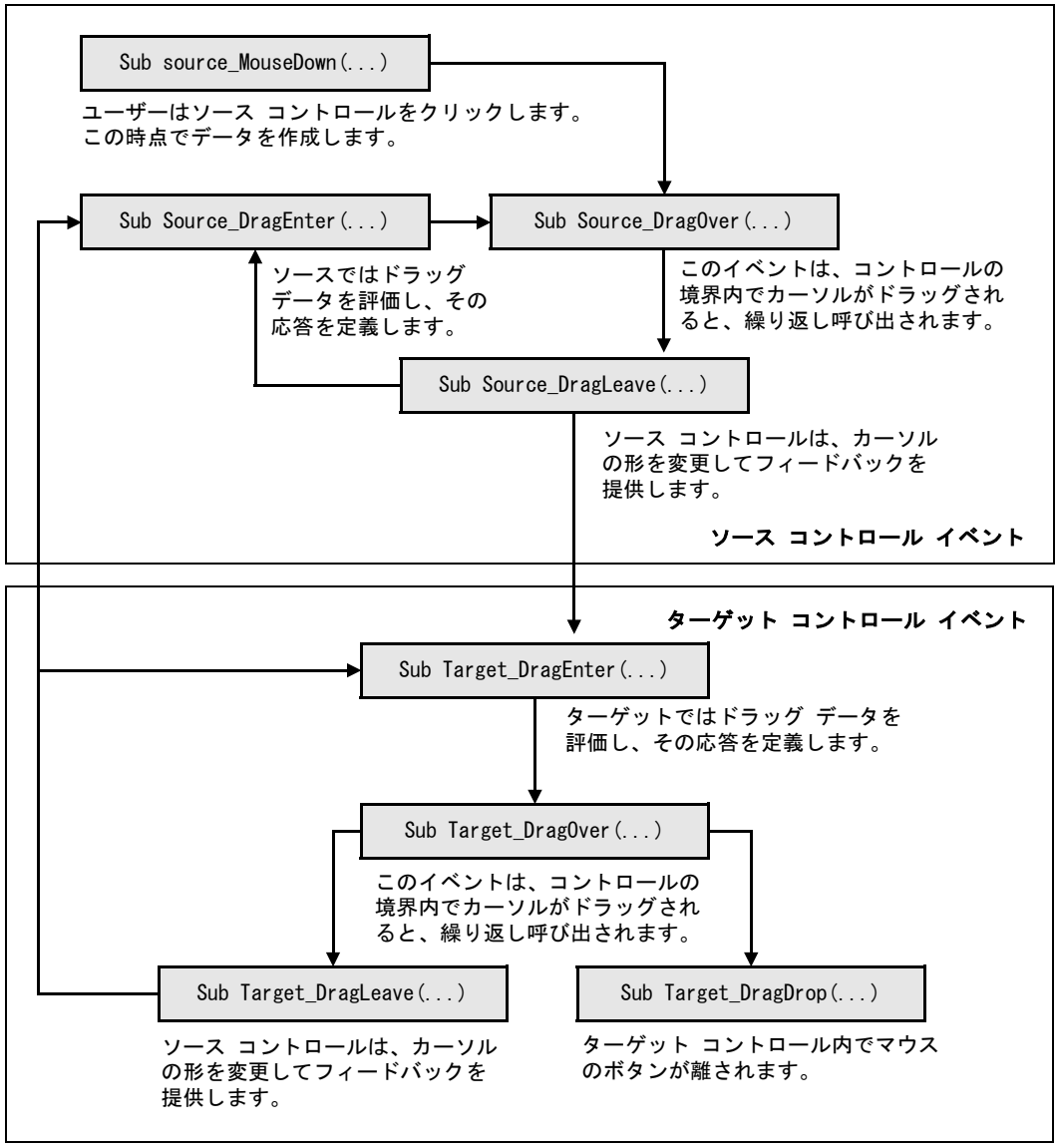

図9.2

Visual Basic .NET *のドラッグ アンド ドロップ操作のライフ サイクル*

Visual Basic .NET でドラッグ アンド ドロップ コードを機能させるために必要となる変更は非常に重要です。ドラッグ アンド ドロップ プログラミング モデルにおける変更により、アップグレード ウィザードでは修正が自動的に行われません。代わりに、メソッドは変更されず、Visual Basic .NET ドラッグ アンド ドロップ イベント モデルにロ

ジックを実装する必要があります。以下ののコード例は、Visual Basic 6.0 でドラッグ アンド ドロップ操作を処理する方法を示しています。

```
Private Sub MyPictureBox_MouseDown(Button As Integer, Shift As Integer, _
          X As Single, Y As Single)
    MyPictureBox.OLEDrag
End Sub

Private Sub MyPictureBox_OLEDragDrop(Data As DataObject, Effect As Long, _
          Button As Integer, Shift As Integer, X As Single, Y As Single)

    If Data.GetFormat(vbCFFiles) Then
        Dim i As Integer
        For i = 1 To Data.Files.Count
            Dim ImagePath As String
            Dim Index As Integer

            ImagePath = Data.Files(i)
            Index = ImageIndexOf(ImagePath)
            If Index = -1 Then
                Select Case UCase(Right(ImagePath, 4))
                    Case ".BMP", ".JPG", ".GIF"
                        ImageList.AddItem ImagePath
                        ImageList.ListIndex = ImageList.ListCount - 1

                        Set MyPictureBox.Picture = LoadPicture(ImagePath)
                End Select
            Else
                ImageList.ListIndex = Index
            End If
        Next
    End If

End Sub

Private Sub MyPictureBox_OLEStartDrag(Data As DataObject, AllowedEffects As Long)
    Data.Files.Clear
    Data.Files.Add ImageList
    Data.SetData , vbCFFiles
    AllowedEffects = vbDropEffectCopy
End Sub

Private Function ImageIndexOf(ImagePath As String) As Integer
    Dim i As Integer

    For i = 0 To ImageList.ListCount
        If ImageList.List(i) = ImagePath Then
            ImageIndexOf = i
            Exit Function
        End If
    Next
    ImageIndexOf = -1
End Function
```

Visual Basic 6.0 コードをアップグレードするのではなく、コードを Visual Basic .NET のドラッグ アンド ドロップ イベント モデルに置き換える必要があります。Visual Basic .NET イベント モデルでは、別の方法で情報の受け渡しが行われます。以下のコード例は、前述の Visual Basic .NET の例と同じドラッグ アンド ドロップ操作を実装する方法を示しています。

```
Private Sub MyForm_Load(ByVal eventSender As System.Object, _
            ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    MyPictureBox.AllowDrop = True
    Application.DoEvents()
End Sub

Private Sub MyPictureBox_DragDrop(ByVal sender As Object, _
            ByVal e As System.Windows.Forms.DragEventArgs) _
            Handles MyPictureBox.DragDrop
    ' 既存のイメージを消去し、すべてのリソースが解放されているかどうかを
    ' 確認します。
    If Not MyPictureBox.Image Is Nothing Then
        MyPictureBox.Image.Dispose()
        MyPictureBox.Image = Nothing
    End If

    Dim Images() As String = e.Data.GetData(DataFormats.FileDrop, True)

    Dim i As Integer
    Dim Index As Integer

    For i = 0 To Images.Length - 1
        Index = ImageIndexOf(Images(i))
        If Index = -1 Then
            ImageList.SelectedIndex = ImageList.Items.Add(Images(i))
        Else
            ImageList.SelectedIndex = Index
        End If
    Next

    ' 最後に追加されたイメージに画像を設定します。
    MyPictureBox.Image = Image.FromFile(ImageList.SelectedItem)
End Sub

Private Sub PictureDisplay_DragEnter(ByVal sender As Object, _
            ByVal e As System.Windows.Forms.DragEventArgs) _
            Handles MyPictureBox.DragEnter
    If e.Data.GetDataPresent(DataFormats.FileDrop) Then
        Dim FileType As String
        Dim file() As String = e.Data.GetData(DataFormats.FileDrop, True)

        FileType = UCase(Microsoft.VisualBasic.Right(file(0), 4))
        Select Case FileType
            Case ".BMP", ".JPG", ".GIF"
                e.Effect = DragDropEffects.Copy
            Case Else
                e.Effect = DragDropEffects.None
```

```
            End Select
        Else
            e.Effect = DragDropEffects.None
        End If
    End Sub

    Private Sub PictureDisplay_MouseDown(ByVal sender As Object, _
                ByVal e As System.Windows.Forms.MouseEventArgs) _
                Handles MyPictureBox.MouseDown
        Dim file(0) As String
        file(0) = ImageList.SelectedItem

        Dim d As New DataObject(DataFormats.FileDrop, file)
        ImageList.DoDragDrop(d, DragDropEffects.Copy)
    End Sub

    Private Function ImageIndexOf(ByRef ImagePath As String) As Short
        Dim i As Short

        For i = 0 To ImageList.Items.Count
            If VB6.GetItemString(ImageList, i) = ImagePath Then
                ImageIndexOf = i
                Exit Function
            End If
        Next
        ImageIndexOf = -1
    End Function
```

この例を見れば、Visual Basic .NET のドラッグ アンド ドロップ機能を置き換えるためにどのような作業が必要であるかがわかります。

MousePointer および MouseIcon プロパティに対する変更の処理

Visual Basic 6.0 では、MousePointer および MouseIcon プロパティを使用して、カスタム アイコン、カーソル、またはさまざまな定義済みマウス ポイントのいずれかを表示できます。マウス ポインタを変更することで、さまざまな動作または可能な動作をユーザーに伝えることができます。たとえば、バックグラウンドで長時間タスクが処理されていることや、コントロールまたはウィンドウのサイズが変更可能であること、または特定のコントロールがドラッグ アンド ドロップ操作をサポートしないことをユーザーに通知できます。カスタム アイコンまたはマウス ポインタを使用して、アプリケーションの状態や機能に関する幅広い視覚情報を表現できます。各ポインタ オプションは、整数値の設定によって表されます。

実行時に、整数値または Visual Basic マウス ポインタ定数を使用してマウス ポインタの値を設定できます。たとえば、次のコード例では、マウス ポインタを砂時計の表示に設定しています。

```
Form1.MousePointer = vbHourglass
```

カスタム アイコンまたはカーソルを使用するには、MousePointer プロパティを vbCustom に設定し、アイコン ファイルまたはカーソル ファイルを MouseIcon プロパティに読み込みます。以下のコード例は、カスタム イメージへのマウス ポインタの設定を示します。

```
Form1.MousePointer = vbCustom
Form1.MouseIcon = LoadPicture("c:\Icons\EW_06.CUR")
```

Visual Basic 6.0 の定義済みマウス ポインタのアップグレードは自動的に行われます。ただし、カスタムのマウス ポインタには追加の作業が必要です。そのようなコードをアップグレード ウィザードでアップグレードすると、コードは変更されず、次のようなアップグレード警告のマークが付きます。

```
' UPGRADE_ISSUE: Form プロパティ Form1.MousePointer はカスタム マウス ポインタを
' サポートしません。

' UPGRADE_ISSUE: Form プロパティ Form1.MouseIcon はアップグレードされませんでした。
```

コードをアップグレードすると、Visual Basic .NET でコンパイル エラーが発生します。同じ機能を得るには、次に示すように、このコードを、System.Windows.Forms.Cursor オブジェクトを使用するコードに置き換える必要があります。

```
Me.Cursor = New Cursor("c:\Icons\EW_06.CUR")
```

カスタムのマウス ポインタは Visual Basic .NET でのデザイン時にサポートされません。MousePointer プロパティは Cursor プロパティに置き換えられ、MouseIcon プロパティは存在しなくなります。ただし、カスタムのマウス ポインタは Load イベントで読み込むことができます。

表 9.3 は、Visual Basic 6.0 のマウス ポインタ定数に対応する Visual Basic .NET の項目をまとめたものです。

表 9.3: マウス ポインタ定数

Visual Basic 6.0	値	Visual Basic .NET の対応する機能
vbDefault	0	Windows.Forms.Cursors.Default
vbArrow	1	Windows.Forms.Cursors.Arrow
vbCrosshair	2	Windows.Forms.Cursors.Cross
vbIbeam	3	Windows.Forms.Cursors.IBeam
vbIconPointer	4	Windows.Forms.Cursors.Default
vbSizePointer	5	Windows.Forms.Cursors.SizeAll
vbSizeNESW	6	Windows.Forms.Cursors.SizeNESW
vbSizeNS	7	Windows.Forms.Cursors.SizeNS

Visual Basic 6.0	値	Visual Basic .NET の対応する機能
vbSizeNWSE	8	Windows.Forms.Cursors.SizeNWSE
vbSizeWE	9	Windows.Forms.Cursors.SizeWE
vbUpArrow	10	Windows.Forms.Cursors.UpArrow
vbHourglass	11	Windows.Forms.Cursors.WaitCursor
vbNoDrop	12	Windows.Forms.Cursors.No
vbArrowHourglass	13	Windows.Forms.Cursors.AppStarting
vbArrowQuestion	14	Windows.Forms.Cursors.Help
vbSizeAll	15	Windows.Forms.Cursors.SizeAll
vbCustom	99	直接対応するものはありません。このセクションに示す例を使用してコードを置き換えます。

プロパティページに対する変更の処理

Visual Basic 6.0 のプロパティページでは、Visual Basic の**プロパティブラウザ**の制限を回避できます。たとえば、プロパティ ページを使用して、ユーザーが色のコレクションを色リスト ユーザー コントロールに追加できるようにします。プロパティページでは、Visual Basic 6.0 の**プロパティブラウザ**の機能では実現できない、コレクションを管理するコードを作成できます。一方、Visual Basic .NET の**プロパティブラウザ**を使用すると、すべての .NET 変数の型またはクラスを編集することができます。プロパティページは必要なくなりました。

ユーザー コントロールのプロパティページは、アップグレード ウィザードによって自動的にアップグレードされません。コントロールのプロパティ ページに公開するために使用していた値をコントロールで編集するためには、ユーザー コントロールにプロパティを追加する必要があります。Visual Basic .NET **プロパティブラウザ**は、コントロールのプロパティのタイプを自動的に認識し、ユーザーが**プロパティ ブラウザ**でそれらのプロパティを編集できるのが利点です。色コレクションの場合は、単にプロパティをユーザー コントロールに追加するだけで、Visual Basic .NET **プロパティブラウザ**に色コレクション エディタが示されます。

このメカニズムを示すのが以下のコード例です。Windows フォームのユーザー コントロールに次のコードを追加します。

```
Dim ColorsCollection() As Color
Property Colors() As Color()
    Get
        Return ColorsCollection
    End Get
    Set(ByVal Value As Color())
        ColorsCollection = Value.Clone
    End Set
End Property
```

ユーザー コントロールをフォームに追加すると、コントロールに Colors という名前の新しいプロパティが設定されます。このプロパティにアクセスすると、コレクション エディタが表示され、コレクション内に含まれているすべての色が示されます。コレクションの項目の色を変更しようとすると、Visual Studio .NET で、図 9.3 に示すような色エディタが自動的に表示されます。

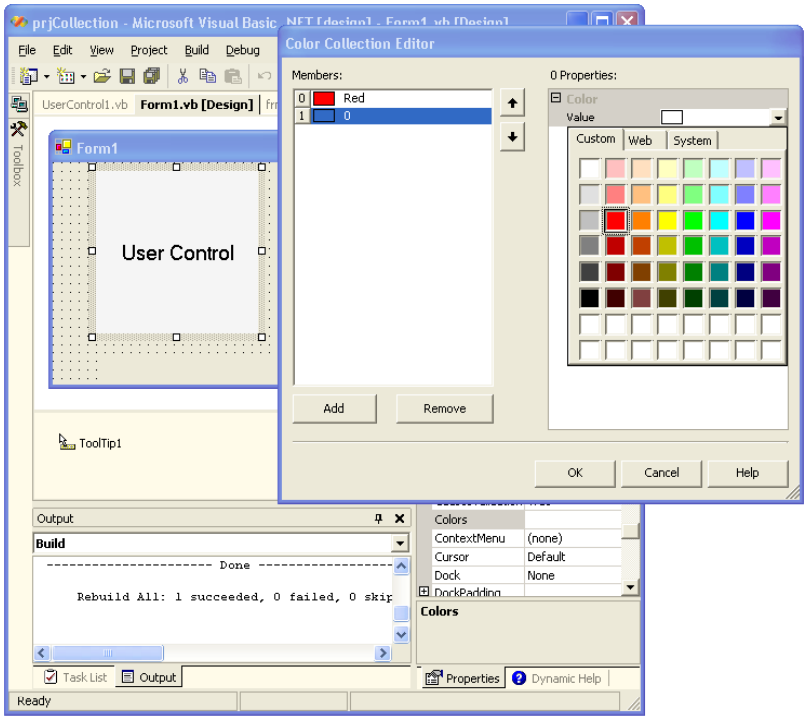

図 9.3

ユーザー コントロールへの色コレクションの追加

プロパティ ページではなく**プロパティ ブラウザ**にプロパティを表示すると、コントロールのすべてのプロパティが表示されるので目的の色を見つけやすいという利点があります。これを、コントロールを使用するクライアントには機能が表示されない、Visual Basic 6.0 のプロパティ ページと比較してみてください。

Visual Basic 6.0 では、プロパティ ページに含まれているデータは通常 PropertyBag オブジェクトを使用して維持されます。ReadProperties イベントおよび WriteProperties イベントは、UserControl の値を取得したり、その値を PropertyBag オブジェクトに保存するために使用されます。Visual Basic .NET では、PropertyBag オブジェクトはサポートされず、ReadProperties イベントと WriteProperties イベントも使用されません。

ただし、オブジェクトのデータは System.Runtime.Serialization および System.IO 名前空間を使用して維持できます。.NET でシリアル化を行うには、Serializable クラス属性を使用する必要があります。Stream オブジェクトおよび BinaryFormatter オブジェクトまたは SoapFormatter オブジェクトを使用して、対応するオブジェクトをインスタンス化するときにファイルからオブジェクトのデータを取得できます。Stream オブジェクトおよび

BinaryFormatter オブジェクトを使用して、オブジェクトを破棄する前にオブジェクトの値をファイルに書き込むことができます。オブジェクトのデータの永続化に関する詳細については、MSDN の「Property Bag Changes in Visual Basic .NET」を参照してください。

Visual Basic .NET でプロパティを変更および管理するには、**プロパティ エディタ**および**デザイナ**を使用します。プロパティを編集するときに、ビジュアル デザイナはダイアログ ボックスまたはドロップダウン リストを使用して指定されたエディタの新しいインスタンスを作成する必要があります。エディタの基本クラスは System.Drawing.Design.UITypeEditor です。これは、System.ComponentModel.EditorAttribute と共に使用します。たとえば、この Visual Basic .NET コードで MyImage クラスを作成すると、エディタとして ImageEditor を指定する EditorAttribute のマークが付きます。

```
<Editor("System.Windows.Forms.ImageEditorIndex, System.Design", _
    GetType(UITypeEditor))> _
Public Class MyImage
    …
End Class 'MyImage
```

Designer 属性 (System.ComponentModel.DesignerAttribute) は、コンポーネントのデザイン時サービスの実装に使用するクラスを指定します。以下の Visual Basic .NET コードでは、MyForm という名前のクラスを作成しています。MyForm には 2 つの属性があります。1 つは、このクラスが DocumentDesigner を使用することを指定する DesignerAttribute で、もう 1 つは Form カテゴリを指定する DesignerCategoryAttribute です。

```
<Designer("System.Windows.Forms.Design.DocumentDesigner, System.Windows.Forms.Design.DLL", _
    GetType(IRootDesigner)), DesignerCategory("Form")> _
Public Class MyForm
    Inherits ContainerControl
    …
End Class 'MyForm
```

Visual Basic 6.0 のプロパティ ページと同等の機能を Visual Basic .NET に実装するには、いくつかの .NET Framework クラスおよびメンバ属性を使用する必要があります。最初に考慮するクラスは、System.Drawing.Design.UITypeEditor です。このクラスは、デザイン時環境のカスタム エディタを実装するために使用または派生できる基本機能を提供します。カスタム エディタは、開発者が指定したデータ型に適用でき、単純なテキスト値エディタでは不十分な場合に役に立ちます。以下のコード例は、デザイン時のプロパティ値エディションの色の選択肢を示す UITypeEditor から派生したクラスを示します。これらの手法の詳細な例については、MSDN の『.NET Framework QuickStart』の「.NET Samples – Windows Forms: Control Authoring」を参照してください。

```
…
Public Class FlashTrackBarValueEditor
    Inherits UITypeEditor
    Private edSvc As IWindowsFormsEditorService

    ' このメソッドを使用して、ユーザー インターフェイス、ユーザー入力処理
    ' および値の代入を処理します。
```

```
    Public Overloads Overrides Function EditValue(ByVal context As _
        ITypeDescriptorContext, ByVal provider As IServiceProvider, _
        ByVal value As Object) As Object
        If (Not (context Is Nothing) And Not (context.Instance Is Nothing) And _
            Not (provider Is Nothing)) Then
            edSvc = CType(provider.GetService( _
                GetType(IWindowsFormsEditorService)), IWindowsFormsEditorService)
            If Not (edSvc Is Nothing) Then
                ' ここで、対応するコントロールを視覚的に更新できます。
                Dim trackBar As FlashTrackBar = New FlashTrackBar
                SetEditorProps(CType(context.Instance, FlashTrackBar), TrackBar)
                edSvc.DropDownControl(trackBar)
                ' 通常、影響を受けるプロパティでこの割り当てを実行するには、
                ' 追加の検証が必要です。
                value = trackBar.Value
            End If
        End If
        Return value
    End Function

    ' このメソッドを使用して、[プロパティ] ウィンドウに、エディタが使用するエディタ スタイルの
    ' タイプを通知します。
    Public Overloads Overrides Function GetEditStyle( _
        ByVal context As ITypeDescriptorContext) As UITypeEditorEditStyle
        If (Not (context Is Nothing) And Not (context.Instance Is Nothing)) Then
            Return UITypeEditorEditStyle.DropDown
        End If
        Return MyBase.GetEditStyle(context)
    End Function
End Class
…
```

前のクラスは、UserControl のプロパティの 1 つに対してカスタム エディタを指定するために使用できます。この例では、FlashTrackBar という名前の UserControl が、濃淡色を示す進捗状況バーと同様の機能を提供するように定義されています。この UserControl には、前に定義されたカスタム エディタを使用してデザイン時に値が変更される開始色と終了色を示す 2 つのプロパティがあります。プロパティのカスタム エディタを指定するには、次に示すように、EditorAttribute 属性クラスを使用する必要があります。

```
…
<Category("Flash"), _
Editor(GetType(FlashTrackBarDarkenByEditor), GetType(UITypeEditor)), _
DefaultValue(200)> _
Public Property DarkenBy() As Byte
    Get
        Return myDarkenBy
    End Get
    Set(ByVal Value As Byte)
        …
    End Set
End Property
…
```

コンポーネントのデザイン時サービスは、DesignerAttribute および DesignerCategoryAttribute クラス属性を使用して指定できます。たとえば、次のコード例では、DocumentDesigner クラスのインスタンスがデザイン時の動作と MyDialog コントロールのビューを提供することを指定しています。DocumentDesigner クラスは、入れ子にされたコントロールをサポートするコントロールのデザイン モードの動作を拡張するために使用される基本デザイナ クラスで、スクロール メッセージを処理することができます。このクラスは、影響を受けるコントロールのデザイン モード ビューも提供できます。

```
<Designer("System.Windows.Forms.Design.DocumentDesigner, _
System.Windows.Forms.Design.DLL", GetType(IRootDesigner)), _
DesignerCategory("Form")> _
Public Class MyDialog
    Inherits ContainerControl
    ...
End Class
```

OLE コンテナ コントロールに対する変更の処理

OLE コンテナ コントロールは、実行時にオブジェクトをフォームに動的に追加するために使用されます。たとえば、OLE コンテナを使用して、埋め込みの Microsoft Word ドキュメントを含むドキュメントを作成したり、リンクされた Word ドキュメントを含むコントロールを作成したり、データベース内の OLE オブジェクトにバインドすることができます。

Visual Basic .NET は、OLE コンテナ コントロールをサポートしていません。そのため、アップグレード ウィザードでは、OLE コンテナ コントロールをアップグレードできません。代わりに、OLE コンテナ コントロールは、コードの再作成が必要なことを強調表示する赤いラベルに置き換わります。

図 9.4 に、埋め込みの Microsoft Word ドキュメントを保持する埋め込みの OLE コンテナ コントロールを使用する Visual Basic 6.0 アプリケーションを示します。

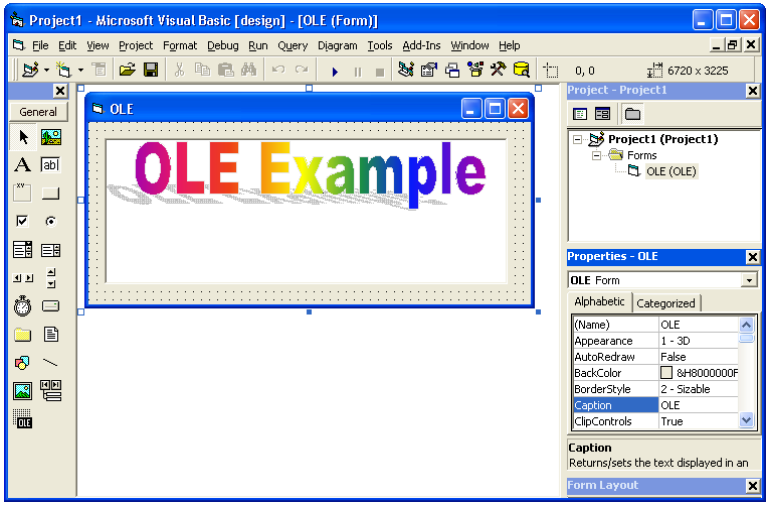

図 9.4

Visual Basic 6.0 のリンクされたオブジェクトを含む OLE コンテナ コントロール

図 9.5 に、アップグレード ウィザード適用後の Visual Basic .NET バージョンを示します。

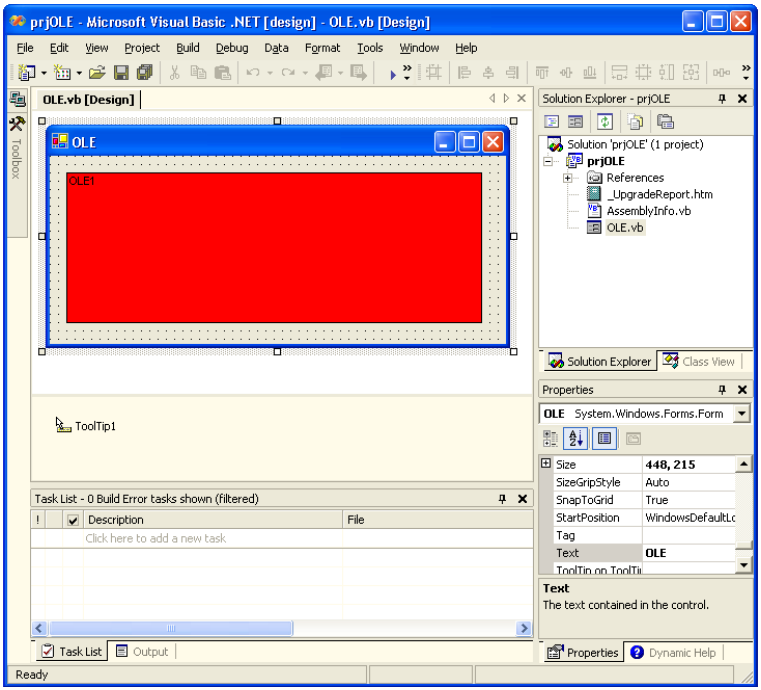

図 9.5

アップグレード ウィザード適用後の OLE コンテナ コントロール

OLE コンテナ コントロールが表示されるたびに、アップグレード ウィザードで生成されたコードには、UPGRADE_ISSUE コメントのマークが付きます。そのコメントの例を以下に示します。

```
' UPGRADE_ISSUE: Ole メソッド OLE1.CreateLink はアップグレードされませんでした。
```

OLE コンテナを置き換えるには、OLE コンテナの代わりに WebBrowser ActiveX コントロールを使用してフォームに Word ドキュメントを表示します。このコントロールを通じて、コントロール内に表示されているドキュメントを表示、編集、および保存できます。

図 9.6 に、OLE コンテナを WebBrowser ActiveX コントロールに置き換えて、Word ドキュメントを表示する Visual Basic .NET コードの更新バージョンを示します。

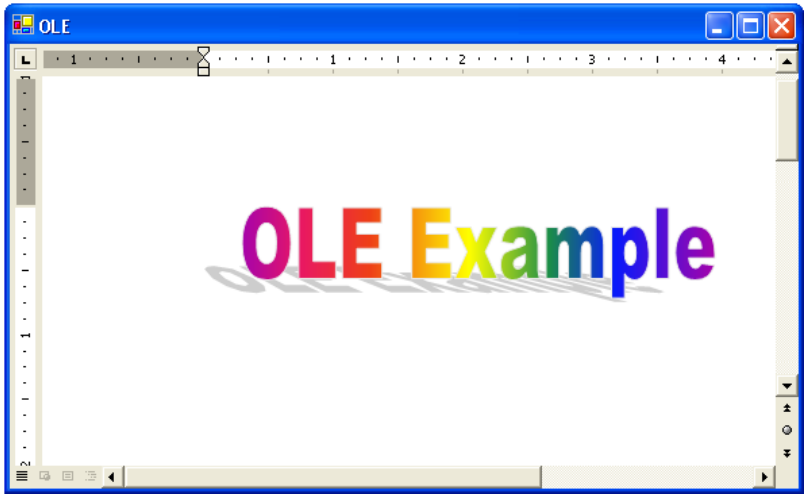

図 9.6

WebBrowser コントロールを使用して表示される Microsoft Word ドキュメント

▶ OLE コンテナを WebBrowser ActiveX コントロールに置き換えるには

1. OLE コンテナのプレースフォルダとしてアップグレード ウィザードにより挿入された赤いラベルを削除します。
2. WebBrowser コントロールをフォームに追加します。これを実行するには、ToolBox 内を右クリックし、[Add/Remove Items] をクリックして [COM コンポーネント] タブをクリックしてから、[ツールボックスのカスタマイズ] コンポーネントピッカーで [Microsoft WebBrowser Control] をクリックします。
3. コントロールを追加した後に、次のコードを Form_Load サブルーチンに (またはドキュメントにバインドする場所に) 追加します。

```
Me.AxWebBrowser1.Navigate("C:\Temp\LinkedDoc.doc")
```

この例のファイル名を、リンクしているオブジェクトの場所に必ず置き換えてください。これらの手順を実行すると、ソリューションをリビルドできるようになります。

この方法は、Microsoft Internet Explorer で表示可能なほとんどのオブジェクト (HTML ファイル、Word ドキュメント、スプレッドシート、GIF、JPEG ファイルなど) に対して機能します。

コントロール配列に対する変更の処理

Visual Basic 6.0 には、フォーム上のコントロールを管理するためのコントロール配列があります。コントロール配列を使用すると、同じ名前と型を共有する一連のコントロールのイベント プロシージャを共有できます。また、コントロール配列は、一連のコントロールを繰り返し表示したり、実行時にコントロールを追加したりするためのメカニズムも提供します。

コントロール配列には制限がありました。コントロール配列には同じ型のコントロールのみ挿入することができます。そのため、たとえば入力ボックスの 1 つがマスクされたエディット コントロールである場合、そのコントロールはテキスト ボックスとして同じコントロール配列内に含めることはできません。

Visual Basic 6.0 ではコントロール配列は一貫していません。コントロール配列は完全なコレクションでも完全な配列でもありません。このため、Visual Basic .NET ではコントロール配列は提供されません。代わりに、Visual Basic .NET には、制限なしでコントロール配列のすべての利点を備えた豊富な機能があります。

Visual Basic 6.0 およびそれより前のバージョンで、開発者がコントロール配列を使用していた主な理由は以下の 3 つです。

1. 1 つのイベント ルーチンを複数のコントロールにフックできる。
2. For Each ループで複数のコントロールに 1 つのコレクションとしてアクセスできる。
3. フォームでコントロールを動的に追加および削除できる。

これらの各タスクは、イベント処理、コレクションおよび動的なコントロール作成で使用できるメカニズムを使用して Visual Basic .NET で実行できます。これらの手法については、以降で説明します。詳細については、MSDN の「Getting Back Your Visual Basic 6.0 Goodies」を参照してください。

イベント処理

Visual Basic 6.0 のコントロール配列では、コントロール配列内のすべてのコントロールに対して 1 組のイベント プロシージャを定義できます。Visual Basic .NET では、この機能は、デザイン時に定義され、実行時に作成されたコントロールに対して実行できます。イベント処理メカニズムを使用すると、コントロール配列を必要とせずに、イベント ハンドラ プロシージャをコントロールで共有できます。以下のコード例は、Visual Basic .NET のイベント ハンドラを示します。

```
Private Sub ProcessGotFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
                        Handles Text1.GotFocus
   ' イベント処理コードがここに入ります。
End Sub
```

Visual Basic .NET イベント ハンドラは Handles キーワードを使用して、ハンドラが管理するイベントを定義します。前述の例では、Text1 の GotFocus イベントが、ProcessGotFocus サブルーチンで処理されています。イベ

ント ハンドラを他のコントロールと共有する必要がある場合は、他のイベントを Handles 句に追加する必要があります。この方法を示すサンプル コードを次に示します。

```
Private Sub ProcessGotFocus(ByVal sender As Object, ByVal e As System.EventArgs) _
                        Handles Text1.GotFocus, Text2.GotFocus, MaskBox1.GotFocus
    ' イベント処理コードがここに入ります。
End Sub
```

これで、ProcessGotFocus サブルーチンは、Text1、Text2、および MaskBox1 の各コントロールの GotFocus イベントを処理します。Handles 句を使用すると、さまざまな種類のコントロール、および、ハンドラ プロシージャ パラメータがイベント宣言のパラメータに対応している限り、さまざまなイベントに対してもイベント ハンドラを共有できます。

コレクションとしてのコントロール配列へのアクセス

Visual Basic .NET では、コントロールのグループの格納に使用できるコレクションおよびその他の種類の構造を広範囲にサポートします。共通のプロパティを更新したり、コントロールの動作を変更するためには、そのようなセット内のコントロールをループ処理すると便利です。

Visual Basic .NET にはコントロール配列がないため、ユーザーはオブジェクトのコレクションを明示的に作成し、必要なすべてのオブジェクトを追加する必要があります。この方法では、ユーザーは現在のニーズに従ってさまざまなオブジェクト グループを作成することができます。

以下のコード例は、For Each ループでコレクションとして使用できる ArrayList クラスの 3 つの Label コントロールをグループ化する方法を示します。この例は、コードが、3 つの Label コントロールで構成されるフォームの一部であることを前提にしています。Demo_Load プロシージャは、すべてのラベルに対して UseMnemonic プロパティを True に設定します。

```
Dim labelCollection As ArrayList
Private Sub GroupLabels()
   labelCollection = New ArrayList
   labelCollection.Add(Label1)
   labelCollection.Add(Label2)
   labelCollection.Add(Label3)
End Sub

Private Sub Demo_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
   Dim obj As Control
   For Each obj In labelCollection
      Dim curLabel As Label
      curLabel = CType(obj, Label)
      curLabel.UseMnemonic = True
   Next
End Sub
```

Visual Studio .NET には、特定のフォームに含まれているすべてのコントロールを格納するコレクションが用意されています。このコレクションの管理は、自動的に行われます。これは、Form クラスの Controls プロパティとして公開され、その型は ControlsCollection です。このコレクションの詳細については、第 7 章「一般的に使用される Visual Basic 6.0 オブジェクトのアップグレード」の「Controls コレクションのアップグレード」を参照してください。

コントロールの動的な追加

Visual Basic 6.0 のコントロール配列のもう 1 つの重要な目的は、実行時にコントロールを簡単に追加できるようにすることです。コントロール配列をデザイン時に作成し、同じ型のコントロールを実行時にその配列に追加することができます。Visual Basic .NET では、実行時にコントロールを追加するためのコントロール配列は必要ありません。Controls コレクションを使用すれば、実行時にどの型のコントロールでも追加することができます。これについては、次のコード例に示します。

元の Visual Basic 6.0 アプリケーション ロジックに従ってコントロールがフォームに動的に追加される場合は、新しいコントロールを既存のイベント ハンドラに動的に接続する必要があります。これは、AddHandler ステートメントを使用して実行できます。このステートメントは、コントロールで発生したイベントを、イベントを処理できる特定のプロシージャに連結します。この手法の例を、次の Visual Basic .NET のコード例に示します。

```
Sub Addbutton()
    Dim newButton As New System.Windows.Forms.Button
    AddHandler newButton.Click, AddressOf General_Click_Handler
    Me.Controls.Add(newButton)
End Sub
Private Sub General_Click_Handler(ByVal sender As System.Object, _
ByVal e As System.EventArgs)
    System.Windows.Forms.MessageBox.Show("A control was clicked")
End Sub
```

AddButton プロシージャは、新しい Button コントロールを作成し、その Click イベントを General_Click_Handler イベント ハンドラに接続します。次に、新しいコントロールを対応するフォームの Controls コレクションに追加し、フォームにボタンを表示します。

Visual Basic .NET は、Visual Basic .NET 互換ライブラリによるコントロール配列をサポートしています。この場合、Visual Basic 6.0 機能の一部を維持しながらアップグレード プロセスを簡略化できます。ただし、第 7 章「一般的に使用される Visual Basic 6.0 オブジェクトのアップグレード」の「Controls コレクションのアップグレード」で説明するように、.NET Framework の代わりに互換ライブラリを使用するには、使用可能なリソースおよびアプリケーションをさらに拡張した場合の利点に従って分析を行う必要があります。

Visual Studio .NET には、Visual Basic 6.0 コントロール配列と同様の機能 (デザイン時のサポートを含む) を提供するサード パーティのコンポーネントがあります。これらのコンポーネントの 1 つである ControlArray コントロールは、.NET Framework Windows フォーム Web サイトで入手できます。このコンポーネントを使用すると、ControlArray イベント ハンドラを共有するコントロールの論理グループをフォーム上に作成することができます。また、このコンポーネントでは、System.Windows.Forms.Control インターフェイスを使用して、ControlArray コンテナを通じて含まれているすべてのコントロールのプロパティを同時に設定したりメソッドを呼び出したりできます。

DDE 機能に対する変更の処理

かつては、共有ファイルを使用せずにアプリケーション間でデータを受け渡す方法の 1 つとして Dynamic Data Exchange (DDE) が使用されていました。バージョン 1 以降 Visual Basic では DDE のサポートが行われてきましたが、DDE は COM に置き換えられています。

COM では、より効率的で拡張性の高い方法でアプリケーション間でのデータの共有やメソッドの呼び出しを行うことができます。パブリック データやメソッドを公開する多くの Windows ベース アプリケーションが COM を利用してこの操作を行っているため、DDE は、Windows ベースのアプリケーションでパブリック データを取得したりパブリック メソッドを呼び出したりする主要な方法ではありません。結果として、COM を、多くの Windows ベースのアプリケーションとの通信手段として使用できます。

アプリケーションを Visual Basic .NET にアップグレードした後に、すべての DDE 通信を COM に置き換えることを強くお勧めします。たとえば、Visual Basic 6.0 アプリケーションは LinkExecute を呼び出して、サーバー アプリケーションとの DDE 対話でコマンドを送信することができます。この場合は、サーバー アプリケーションに関するドキュメントまたはタイプ ライブラリを参照して、DDE コマンドが提供している機能を 1 つ以上の COM メソッドに置き換えることができるかどうかを確認することをお勧めします。その機能が使用可能な場合は、COM 参照を Visual Basic .NET プロジェクトに追加し、対応するメソッドを呼び出すことができます。

Visual Basic .NET と .NET Framework のいずれも DDE をサポートしていません。DDE を使用して別のアプリケーションと通信する必要がある場合は、次の 2 つの方法のいずれかを使用して Visual Basic .NET アプリケーションに DDE を実装する必要があります。

- DDE 対話を管理する Visual Basic 6.0 に組み込まれたユーザー定義の ActiveX EXE サーバーと相互運用する。
- DDE 関連の Windows API 関数を宣言および実装する。

Visual Basic 6.0 DDE コードを再使用する場合は、Visual Basic 6.0 ActiveX EXE プロジェクトを作成し、パブリック クラスを追加して、必要な DDE 関連の情報を Visual Basic .NET アプリケーションと交換できます。たとえば、

Visual Basic .NET コードで DDE LinkExecute 操作を実行する場合は、次のように Visual Basic 6.0 で作成された DDE ヘルパ クラスを作成します。

1. Visual Basic 6.0 ActiveX EXE プロジェクトを作成します。
2. パブリック メソッドを、2 つのパラメータ (ServerTopic および ExecCommand) をとる LinkExecute という名前のクラス (デフォルト名は Class1) に追加します。
3. たとえば、Form1 という名前のフォームを、ActiveX EXE プロジェクトに追加します。
4. たとえば、Text1 という名前の TextBox をフォームに追加します。
5. 次のコードを、Class1 の LinkExecute メソッドに追加します。

```
Dim f As New Form1
f.Text1.LinkMode = 0                'None
f.Text1.LinkTopic = ServerTopic
f.Text1.LinkMode = 2                'Manual
f.Text1.LinkExecute ExecCommand
```

6. デフォルト名が Project1.dll の ActiveX EXE サーバーを構築します。

クラス、フォーム、およびコントロールには、よりふさわしい名前をプロジェクトに付けることが可能であり、またその必要があります。ただし、LinkExecute メソッドの名前は、既存の DDE コードと一貫性を持たせておくために変更できません。

Visual Basic .NET で LinkExecute 操作を実行するためには、Project1.dll への COM 参照を、Visual Basic .NET プロジェクトの References リストに追加し、次のコードを入力します。

```
Dim DDEClientHelper As New Project1.Class1Class()
DDEClientHelper.LinkExecute("MyDDEServer•MyTopic", "MyCommand")
```

Visual Basic .NET アプリケーションに DDE を実装するには、Windows DDE 関連の API 関数を呼び出すことが必要です。最低限 DdeInitialize および DdeUninitialize メソッドを呼び出す必要があります。DdeInitialize を呼び出すには、DdeCallbackProc を実装する必要があります。さらに、その他多数の DDE 関連の API 関数を呼び出して接続を確立し、データを送信する必要があります。たとえば、LinkExecute を実行する機能を実装するには、8 つ以上の他の DDE 関連の API 関数を呼び出す必要があります。このタスクを実行するためには、Windowsの メッセージング アーキテクチャおよびメモリの管理についても深く理解する必要があります。これは、Visual Basic コードの Windows タイプを表す方法 (ハンドルや構造など) に加えて必要な知識です。この知識を習得するには、相当の努力が必要です。DDE 関連の Windows API の詳細については、MSDN の「Dynamic Data Exchange Management Functions」を検索してください。コードにおける Windows API 関数呼び出しでの Declare ステートメントの使用に関連する問題については、第 13 章「Windows API の使用」の「データ型の変更」を参照してください。

まとめ

多くの Visual Basic 6.0 アプリケーションはフォームを使用します。フォーム アーキテクチャは、Visual Basic .NET では .NET Framework に準拠する Windows フォーム パッケージとして完全にやり直しを行っています。その結果、強固なフォームベース アプリケーションを構築するための柔軟なアーキテクチャが使用できる代わりに、Visual Basic 6.0 の一部の機能は手動でアップグレードする必要があります。

この章で説明した手法を実行すると、Visual Basic .NET ではサポートされていないフォーム機能と同等の機能を実現することができます。ただし、アプリケーションを動かすフォームの再設計を検討するのが理想的です。フォームの再設計を検討している場合は、MSDN に記載されている Windows フォーム パッケージに関する豊富な情報およびチュートリアルを参照できます。手始めに MSDN の Microsoft .NET Framework Developer Center の「Building Windows Forms Applications」を参照することをお勧めします。

詳細情報

MenuItem.DrawItem イベントの詳細については、MSDN の『.NET Framework Class Library』の「MenuItem.DrawItem Event」を参照してください。
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwindowsformsmenuitemclassdrawitemtopic.asp

オブジェクトのデータの永続化に関する詳細については、MSDN の「Property Bag Changes in Visual Basic .NET」を参照してください。
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vbtchpropertybagchangesinvisualbasicnet.asp

コントロールの作成手法の詳細については、MSDN の『.NET Framework QuickStart』の「.NET Samples – Windows Forms: Control Authoring」を参照してください。
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpqstart/html/cpsmpNETSamples-WindowsFormsControlAuthoring.asp

コントロール配列の詳細については、MSDN の「Getting Back Your Visual Basic 6.0 Goodies」を参照してください。
http://msdn.microsoft.com/vbasic/using/columns/adventures/default.aspx?pull=/library/en-us/dnadvnet/html/vbnet05132003.asp

ControlArray コントロールをダウンロードするには、次の .NET Framework Windows Web サイトを参照してください。
http://www.windowsforms.net/default.aspx?tabindex=6&tabid=47&ItemID=16&mid=142

Windows フォーム パッケージの詳細については、MSDN の Microsoft .NET Framework Developer Center の「Building Windows Forms Applications」を参照してください。
http://msdn.microsoft.com/netframework/programming/winforms/

10

Web アプリケーションのアップグレード

Visual Basic 6.0 には、Web のプログラミングをサポートする機能がいくつかあります。たとえば、Microsoft インターネット インフォメーション サービス (IIS) アプリケーション (Web クラス)、DHTML アプリケーション、ActiveX ドキュメント、Web ページにダウンロード可能な ActiveX コントロールなどがあります。

Visual Basic .NET は、ASP.NET Web アプリケーション、XML Web サービスなどの機能によって Web プログラミングをサポートできるように構築されています。Visual Basic .NET は新しいアーキテクチャの上に構築されているため、Visual Basic 6.0 の Web 機能はサポートされないか、大幅に変更されているかのどちらかです。ただし、Web プログラミングの知識があれば、新しい Web テクノロジに迅速に移行することができます。

Visual Basic 6.0 では、IIS アプリケーションは Active Server Pages (ASP) モデルを使用して、IIS で実行されるアプリケーションを作成していました。Visual Basic .NET では、ASP.NET テクノロジにより、Web フォーム ページを使用してアプリケーションを作成したり、XML Web サービスを使用してコンポーネントを作成したりすることができます。これらのテクノロジにより、Web のプログラミングは、Visual Basic 6.0 の Windows のプログラミングに非常に似たものになっています。

Visual Basic 6.0 の DHMTL アプリケーションは、Dynamic HTML オブジェクト モデルおよび Visual Basic コードを使用して、Web ブラウザでユーザーが実行する操作に応答できるアプリケーションを作成していました。Visual Basic .NET の Web フォームは DHTML モデルで展開され、より多彩でダイナミックなユーザー インターフェイス機能に加えて、クライアント側の検証機能も提供します。

Visual Basic 6.0 の ActiveX ドキュメントは、Visual Basic .NET ではサポートされません。Visual Basic .NET Web アプリケーションから ActiveX ドキュメントと相互運用することはできますが、開発は Visual Basic 6.0 で維持する必要があります。

Visual Basic 6.0 と同様、Visual Basic .NET では、Web ページにダウンロード可能な ActiveX コントロールを作成したり、アプリケーションで既存の ActiveX コントロールを使用することができます。

この章では、Web アプリケーションをアップグレードする際に対応する必要がある変更点について説明します。

ActiveXドキュメントのアップグレード

Visual Basic 6.0 で使用できる 1 つの機能に、ActiveXドキュメントを作成する機能があります。ActiveXドキュメントは、Web ブラウザ内に表示できるフォームです。ActiveXドキュメントには、組み込みのビューポート スクロール、ハイパーリンク、およびメニュー ネゴシエーションなどの機能があります。

ActiveXドキュメントは、Visual Basic .NET ではサポートされません。代わりに、Web ユーザー コントロールを使用して、ActiveX ドキュメントと同じ効果を得ることができます。このため、ActiveX ドキュメントを使用するアプリケーションを扱う場合は、次の 3 つのアップグレード オプションを使用できます。

- Visual Basic .NET Web アプリケーションから相互運用性を利用して ActiveXドキュメントを使用することができますが、開発は Visual Basic 6.0 で維持する必要があります。Visual Basic .NET Web フォームからVisual Basic 6.0 ActiveXドキュメントへ移動、またはVisual Basic 6.0 ActiveXドキュメントからVisual Basic .NET Web フォームへ移動できます。
- ActiveXドキュメントを Web ユーザー コントロールとして再作成できます。ActiveXドキュメントの動作をシミュレートするには、Web フォーム内に新しいコントロールを含める必要があります。
- 元の ActiveX ドキュメントに対応する Windows フォーム コントロールを作成し、そのコントロールを Microsoft Internet Explorer でホストできます。このアプローチでは、ソースとターゲット両方のコンポーネントで同様の機能を利用できますが、配置や互換性の点では制限があります。

ActiveXドキュメントから Web ユーザー コントロールへのアップグレードについて、docCalculate という名前のユーザードキュメントを含むActiveXドキュメントプロジェクトを例にとります。図 10.1 に示すように、このプロジェクトには、3 つの TextBox コントロール、Line コントロールと Button コントロールが 1 つずつ含まれています。

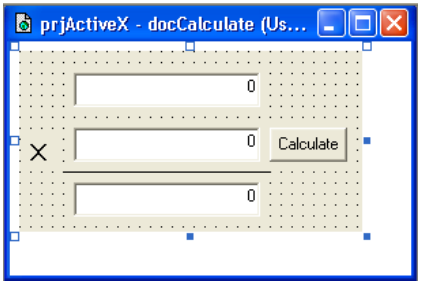

図 10.1

Visual Basic 6.0 ActiveX ドキュメント アプリケーションの例

Button のコードでは、一番上と 2 つ目の TextBox コントロールの値を乗算し、その結果を 3 つ目の TextBox コントロールに表示します。イベント コードは次のようになります。

```
Option Explicit
Private Sub btnCalculate_Click()
    On Error GoTo Exception
    If (txtFirstNumber.Text <> "") Then
```

```
        If (txtSecondNumber.Text <> "") Then
            Dim res As Single
            res = CSng(txtFirstNumber.Text) * CSng(txtSecondNumber.Text)
            txtResult.Text = CStr(res)
        End If
    End If
    Exit Sub
Exception:
    txtResult.Text = "####ERROR####"
End Sub
```

このプロジェクトをアップグレード ウィザードで変換しようとしても、ユーザー ドキュメントはアップグレードされません。ただし、その変更されないままのドキュメントが、アップグレードされたプロジェクトにコピーされます。このコードは、Sub Main が prjActiveX 内に見つからないことを示すコンパイル エラーを生成します。また、その結果として得られるプロジェクトは、ActiveX ドキュメントと同じ動作を示さない Windows フォーム プロジェクトになります。

このコードをアップグレードするには、最初に、ActiveX ドキュメント内に存在するユーザー ドキュメントごとに 1 つのユーザー コントロールを作成します。推奨される命名方法は、各コントロールの名前を、ユーザー ドキュメントの元の名前の前に "UC" を付けた名前に設定することです。これで、各 ActiveX ドキュメントのコントロールとコードを、それぞれのユーザー コントロールにコピーできます。

次に、ActiveX ドキュメント プロジェクトをアップグレード ウィザードを使用して変換します。これによってユーザー コントロールだけがアップグレードされるため、追加の手順を実行してドキュメント プロジェクトを作成する必要があります。ASP.NET Web アプリケーションをソリューションに追加すると、これを実現できます。推奨される命名方法は、元の ActiveX ドキュメントの名前の前に "Web" を付けて使用することです。この例では、WebprjActiveX が推奨される名前になります。ASP.NET Web アプリケーションをソリューションに追加すると、Visual Studio .NET では、IIS に同じ名前の仮想ディレクトリが作成されます。また、元の Visual Basic 6.0 プロジェクト内の各ユーザー ドキュメントのプロジェクトに Web フォームを追加する必要があります。各 Web フォームの推奨される命名方法は、ユーザー ドキュメントの元の名前を使用することです。この例では、1 つの Web フォームだけが追加され、その名前は docCalculate.aspx に設定されます。

元の Visual Basic 6.0 プロジェクト内の各ユーザー ドキュメントのプロジェクトに Web ユーザー コントロールを追加する必要があります。これらの各コントロールの名前は、元の名前と同じ名前に設定します。この例では、1 つの Web ユーザー コントロールだけが追加され、その名前は docCalculate になります。すべての Web ユーザー コントロールが追加されたら、それぞれの Web ユーザー コントロールでユーザー ドキュメントのデザインをコピーできます。この例では、グリッド レイアウト パネルをコントロールに追加し、3 つのテキスト ボックス コントロール、ボタン コントロールとラベル コントロールをそれぞれ 1 つずつ追加できます。各コントロールの名前は、元の Visual Basic 6.0 プロジェクトで使用していた名前と同じになります。図 10.2 に、最終的な Visual Basic .NET フォームの例を示します。

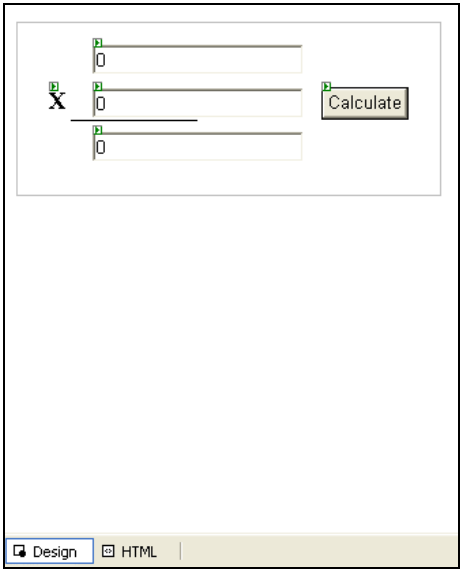

図 10.2

Visual Basic 6.0 ActiveX *ドキュメント アプリケーション*の Visual Basic .NET *へのアップグレードの例*

最後に、ユーザー コントロールでアップグレードされたコードを、それぞれの Web ユーザー コントロールにコピーする必要があります。変換されたコードがコピーされた後に、各 Web ユーザー コントロールを適切な Web フォームに追加できます。プロジェクトを再構築し、発生した問題を修正する必要があります。その後、ドキュメントは Internet Explorer で表示できるようになります。

Web クラスのアップグレード

Visual Basic 6.0 では、WebClass プロジェクト (IIS アプリケーション プロジェクトとも呼ばれます) を使用して ASP テクノロジに基づく Web アプリケーションを作成します。Visual Basic .NET では、ASP.NET Web アプリケーション プロジェクトを使用して、新しい ASP.NET テクノロジに基づく Web アプリケーションが作成されます。Visual Basic 6.0 の WebClass プロジェクトが Visual Basic .NET にアップグレードされると、WebClass プロジェクトは ASP.NET Web アプリケーション プロジェクトに変換されます。

Visual Basic 6.0 WebClass プロジェクトには、要求間で WebClass のインスタンスを維持するために使用できる StateManagement プロパティがあります。これは、デザイン時に StateManagement プロパティを wcRetainInstance に設定して実行されます。インスタンスを終了するには、ReleaseInstance メソッドを使用します。

Visual Basic .NET では、ASP.NET Web アプリケーションに StateManagement プロパティがないため、アプリケーションの状態を管理するためのモデルが大きく異なります。このため、状態管理に関連するすべてのコードを置き換える必要があります。これを行うための多くのオプションがあります。クライアントに (たとえば、直接にページまたは Cookie で) 情報を維持するオプションや、ラウンド トリップ間でサーバーに情報を格納するためのオプションがあります。

インスタンスをクライアントに格納する場合は、次のいくつかの手法を使用できます。

- **ビュー ステート。**control.ViewState プロパティは、同じページの複数の要求間で値を保持するための辞書を提供します。この情報は自動的に格納されます。ページが処理されると、ページおよびコントロールの現在の状態が文字列にハッシュされ、非表示フィールドとしてページに保存されます。ページがサーバーに戻されると、ページはページの初期化時にビュー ステート文字列を解析し、ページにプロパティ情報を復元します。
- **非表示フォーム フィールド。**ブラウザで表示されない非表示フィールドをフォーム上に作成できます。この手法を使用すると、標準コントロールで行うのと同じようにプロパティを設定できます。ページがサーバーに送信されると、非表示フィールドの内容が、他のコントロールの値と共に HTTP フォーム コレクションで送信されて、非表示フィールド経由でページに直接に情報を格納できます。
- Cookie。Cookie を使用して特定のクライアント、セッション、またはアプリケーションに関する情報を格納できます。Cookie はクライアント デバイスに保存されます。その後にブラウザがページを要求するときには、その要求情報と共に Cookie 内の情報が、ブラウザによって送信されます。サーバーは Cookie を読み込み、必要な値を抽出できます。
- **クエリ文字列**。クエリ文字列を使用していくつかの状態情報を維持できますが、ブラウザおよびクライアント デバイスの容量に制限されます。これにより、URL の長さには 255 文字の制限が課せられます。ページ処理中にクエリ文字列の値を使用できるようにするには、HTTP GET メソッドを使用してページを送信する必要があります。このため、HTTP POST メソッドに応答してページが処理される場合には、このオプションは利用できません。

または、最良のオプションがサーバーに情報を格納することであると判断した場合は、次のいずれかの方法を使用できます。

- **アプリケーションの状態**。ASP.NET では、アプリケーションの状態 (アクティブな各 Web アプリケーションの HttpApplicationState クラスのインスタンス) を使用して値を保存できます。アプリケーションの状態は、Web アプリケーション内のすべてのページからアクセスできるグローバルなストレージ メカニズムであり、サーバー ラウンド トリップ間およびページ間で維持する必要がある情報を格納する場合に有用です。アプリケーションの状態は、特定の URL に対する各要求中に作成されるキー値の辞書構造です。この構造に情報を追加し、ページ要求間で格納できます。
- **セッション状態。**ASP.NET では、セッション状態 (アクティブな各 Web アプリケーションの HttpSessionState クラスのインスタンス) を使用して値を保存できます。
- **データベース サポート**。大量の情報を格納しているときに、データベース テクノロジを使用してページの状態を維持できます。データベースへの格納は、長期間の状態、またはサーバーを再起動する必要がある場合でも維持する必要がある状態を保持する場合特に有用です。データベース アプローチは多くの場合、Cookie と組み合わせて使用します。

WebClass プロジェクトをアップグレードする場合、Visual Basic 6.0 コードの Function と Sub プロシージャ (ProcessTab や Respond など) は、WebClass 互換性ランタイムがこれらのプロシージャを実行できるように、スコープが Private から Public に変更されます。

Visual Basic 6.0 の WebClass イベントの中には、ASP.NET でサポートされていないイベントがあります。たとえば、Initialize、BeginRequest、EndRequest、Terminate などです。これらのイベント プロシージャは、アップグレード ウィザードでアップグレードされますが、実行時に呼び出されることはありません。アップグレードした後で、これらのイベントのどのコードも、対応する ASP.NET イベント (Init や Unload など) に移動する必要があります。

アップグレード ウィザードは、プロジェクトにいくつかの宣言も追加します。WebClass に 1 つ、および元のプロジェクトの WebItems と Templates に 1 つずつ追加されます。Page_Load イベント プロシージャがプロジェクトに追加され、まず WebClass オブジェクトが作成されます。次に、元のプロジェクトに関連付けられた WebItems と Templates ごとに WebItem オブジェクトが作成されます。最後に、Page_Load イベント プロシージャには、WebClass 互換性ランタイムの WebClass.ProcessEvents への呼び出しが含まれています。これにより、ランタイムは要求 URL に指定された WebItem を表示できます。このコードは、アップグレード後のプロジェクトに追加される唯一の新しいコードで、Visual Basic 6.0 WebClass ランタイムの基本動作のエミュレートのみを目的とするものです。

まとめ

Web ベースのアプリケーションをアップグレードするには、手動の作業が必要になります。Visual Basic .NET でサポートされない ActiveX ドキュメントは再作成し、Web ユーザー コントロールを使用して同様の機能を再作成する必要があります。ただし、Web クラスは、Visual Basic アップグレード ウィザードおよび WebClass 互換性ランタイムを使用して部分的に自動でアップグレードできます。この章で説明した手法を利用すると、このようなタイプの Web ベース アプリケーションを Visual Basic .NET にアップグレードする際の問題を解決することができます。

11

文字列操作とファイル操作のアップグレード

一般的な Microsoft Visual Basic 6.0 アプリケーションでは、ある種の文字列操作を実行する必要があります。また、タスクの実行中にアプリケーションでファイルを使用または処理することも珍しくはありません。

Visual Basic .NET にアップグレードする場合、文字列操作やファイル操作を実行するコードをアップグレードするには 2 つの方法があります。1 つ目は、アップグレード ウィザードを使用して自動的にコードをアップグレードする方法です。アップグレード後のコードは、Visual Basic 互換性ライブラリを通じて、元のコードと同じ動作を実行します (詳細については、第 8 章「一般的に使用される Visual Basic 6.0 言語機能のアップグレード」を参照してください)。もう 1 つは、これらの操作を新しい Visual Basic .NET 機能に切り替えることです。これには、ウィザードを使用した場合よりも多くの作業が必要になりますが、互換性ライブラリに依存する必要はなくなります。

この章では、文字列操作とファイル操作を Visual Basic .NET にアップグレードする方法を説明します。ここでは、互換性ライブラリを利用する方法と、新しい機能に切り替える方法の両方を説明し、参考となる追加情報の参照先を紹介します。

アップグレード ウィザードで処理される操作

最も簡単な方法は、アップグレード ウィザードに作業を任せることです。ここでは、アップグレード ウィザードによって自動的に処理される機能について説明します。

自動的にアップグレードされる文字列操作

最も基本的な文字列操作は、文字列の連結とサブストリングの取得です。文字列操作には最も簡単な種類の操作があり、それらは最も頻繁に実行されている操作でもあります。以下のコード例は、左、中央、右のサブストリングを取得する方法と、文字列を連結する方法を示しています。

```
Private Sub Command1_Click()
    Dim length As Integer
' 文字列を均等に分けます。
    Dim chunk As Integer

    length = Len(Label1.Caption)
    chunk = length / 3
    lblLeft.Caption = Left(Label1.Caption, chunk)
    lblMid.Caption = Mid(Label1.Caption, chunk + 1, chunk)
    lblRight.Caption = Right(Label1.Caption, chunk)
    labelRandom.Caption = lblMid.Caption & lblRight.Caption & lblLeft.Caption
End Sub
```

このコードにアップグレード ウィザードを適用した場合、生成されたコードは問題なく、ビルドおよび実行できます。アップグレード ウィザードによって生成されるコードは以下のようになります。

```
Option Strict Off
Option Explicit On
Imports VB = Microsoft.VisualBasic
…
    Private Sub Command1_Click(ByVal eventSender As System.Object, _
        ByVal eventArgs As System.EventArgs) Handles Command1.Click

        Dim length As Short
        Dim chunk As Short

        length = Len(Label1.Text)
        chunk = length / 3
        lblLeft.Text = VB.Left(Label1.Text, chunk)
        lblMid.Text = Mid(Label1.Text, chunk + 1, chunk)
        lblRight.Text = VB.Right(Label1.Text, chunk)
        labelRandom.Caption = lblMid.Text & lblRight.Text & lblLeft.Text
    End Sub
…
```

Import ステートメントで Microsoft.VisualBasic 名前空間がインクルードされています。この名前空間には、この例で使用した文字列関数を含む Visual Basic .NET のコア機能が含まれています。Visual Basic 6.0 の文字列関数は、アップグレード ウィザードによって、この名前空間の対応する関数にアップグレードされます。このコードを実行すると、元のコードとまったく同じ動作が実行されます。

Microsoft.VisualBasic 名前空間に含まれる文字列操作の一覧は、この名前空間のヘルプで参照できます。

自動的にアップグレードされるファイル操作

ファイルの入出力は、Microsoft.VisualBasic ライブラリを通じてサポートされます。ここでは、アップグレード ウィザードが Visual Basic 6.0 アプリケーションのファイル入出力をどのように処理するかを説明します。

テキスト ファイルへのアクセス

Visual Basic 6.0 には、ファイルやディレクトリへのアクセスを管理する複数の I/O コマンドがあります。該当するコマンドには、Open、Close、Reset、Get、Put、Print、Write、Input、LineInput、Lock、Unlock、Width などがあります。

以下の Visual Basic 6.0 のコード例は、テキスト ファイルを開く、ファイルの内容を 1 文字ずつ読み込む、読み込んだ文字を**イミディエイト** ウィンドウに送る、というファイル操作の使用方法を示しています。

```
Dim FileName as String
Dim MyChar as String

FileName = "C:\Temp\TextBox.txt"
' ファイルを開きます。
Open FileName For Input As #1
' EOF までループします。
Do While Not EOF(1)
' 1 文字取得します。
   MyChar = Input(1, #1)
' イミディエイト ウィンドウに出力します。
   Debug.Print MyChar
Loop
Close #1
```

このコード例を実行するには、FileName 変数を有効な名前に設定しておく必要があります。

このコードは、アップグレード ウィザードで自動的にアップグレードできます。以下に、ウィザードによって生成されるコードを示します。

```
Dim FileName As String
Dim MyChar As String

FileName = "C:\Temp\TextBox.txt"
' ファイルを開きます。
FileOpen(1, filename, OpenMode.Input)
' EOF までループします。
Do While Not EOF(1)
' 1 文字取得します。
```

```
    MyChar = InputString(1, 1)
' イミディエイト ウィンドウに出力します。
    System.Diagnostics.Debug.WriteLine(MyChar)
Loop
FileClose(1)
```

このコードは、互換性ライブラリのサポートにより、元のコードと同じようにコンパイルおよび実行されます。前述のように、リソースに余裕があれば、Microsoft .NET Framework のコア機能のみを使用して出力を書き換えることも可能です。Visual Basic .NET でファイルを操作する場合は、ストリームを使用することをお勧めします。ストリームとは、読み取り、書き込み、シークを可能にする一連のバイトの総称です。ストリームは、ファイル、ネットワーク接続、メモリなどさまざまなものに関連付けることができます。Visual Basic .NET では、暗号ストリームを読み書きすることもできます。

アップグレード ウィザードによって生成された Visual Basic .NET コードは、テキスト ファイル関数ではなく、ストリームを使用して書き換えることができます。詳細については、この後の「ストリームを使用したファイル I/O の改善」を参照してください。

ユーザー定義データ型を使用した固定長レコードへのアクセス

Visual Basic 6.0 アプリケーションでフラット ファイルを通じてレガシ システムと対話するケースは珍しくありません。Visual Basic 6.0 には、ランダム、バイナリ、シーケンシャルなどのファイル アクセス モードを含め、この種の対話向けのツールが用意されていました。ここでは、ランダム ファイル アクセス コードを Visual Basic .NET にアップグレードするために必要なステップを説明します。

以下の Visual Basic 6.0 のサンプル コードには、レコードを格納するためのメソッドと、そのレコードをフラット ファイルから読み込むためのメソッドが含まれています。

```
Type employee
    name As String * 15
    last_name As String * 20
    department As String * 6
    phone_ext(3) As Long
    salary As Integer
End Type

Private Sub Form_Load()
    WriteRecord 3
    ReadRecord 3
End Sub
Sub ReadRecord(recordnum As Integer)
    Dim fileh As Integer
    Dim emp As employee
    fileh = FreeFile
    Open "C:\recordtest.dat" For Random As fileh Len = 73
    ' レコードは recordnum で指定された位置から読み込まれます。
    Get fileh, recordnum, emp
```

```
    MsgBox ("Employee: " & emp.name & " " & emp.last_name & _
    ", Dept: " & emp.department)
    Close fileh
End Sub
Sub WriteRecord(recordnum As Integer)
    Dim fileh As Integer
    Dim emp As employee
    fileh = FreeFile
    Open "C:\recordtest.dat" For Random As fileh Len = 73
    emp.name = "John"
    emp.last_name = "Doe"
    emp.department = "First aids"
    emp.phone_ext(0) = 123
    emp.salary = 1000

    ' レコードは recordnum で指定された位置に保存されます。
    Put fileh, recordnum, emp
    Close fileh
End Sub
```

employee ユーザー定義型 (UDT) には、UDT がファイル内に格納されている場合に、ランダム方式での UDT へのアクセスを可能にする固定長フィールドのみが含まれています。以下のコード サンプルは、アップグレード ウィザードを使用して取得された出力を、アップグレードの警告に従ってユーザーが調整したコードを示しています。

```
Structure employee
    <VBFixedString(15),System.Runtime.InteropServices.MarshalAs( _
    System.Runtime.InteropServices.UnmanagedType.ByValTStr, _
    SizeConst:=15)> Public name As String

    <VBFixedString(20),System.Runtime.InteropServices.MarshalAs( _
    System.Runtime.InteropServices.UnmanagedType.ByValTStr, _
    SizeConst:=20)> Public last_name As String

    <VBFixedString(6),System.Runtime.InteropServices.MarshalAs( _
    System.Runtime.InteropServices.UnmanagedType.ByValTStr, _
    SizeConst:=6)> Public department As String

    <VBFixedArray(3)> Dim phone_ext() As Integer
    Dim salary As Short

    ' UPGRADE_TODO: この構造体を初期化するには、"Initialize" を
    ' 呼び出す必要があります。詳細については、ここをクリックしてください。
    ' 'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1026"'
    Public Sub Initialize()
        ReDim phone_ext(3)
    End Sub
End Structure

Private Sub Form1_Load(ByVal eventSender As System.Object, _
ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    WriteRecord(3)
```

```
    ReadRecord(3)
End Sub
Sub ReadRecord(ByRef recordnum As Short)
    Dim fileh As Short
    Dim emp As employee
    emp.Initialize()
    fileh = FreeFile
    FileOpen(fileh, "C:\recordtest.dat", OpenMode.Random, , , 73)
    ' レコードは recordnum で指定された位置から読み込まれます。
    ' UPGRADE_WARNING: Get は FileGet にアップグレードされ、新しい動作が指定されています。
    ' 詳細については、ここをクリックしてください。
    ' 'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1041"'
    FileGet(fileh, emp, recordnum)
    MsgBox("Employee: " & emp.name & " " & emp.last_name & ", Dept: " & _
    emp.department)
    FileClose(fileh)
End Sub
Sub WriteRecord(ByRef recordnum As Short)
    Dim fileh As Short
    Dim emp As employee
    emp.Initialize()
    fileh = FreeFile
    FileOpen(fileh, "C:\recordtest.dat", OpenMode.Random, , , 73)
    emp.name = "John"
    emp.last_name = "Doe"
    emp.department = "First aids"
    emp.phone_ext(0) = 123
    emp.salary = 1000
    ' レコードは recordnum で指定された位置に保存されます。
    ' UPGRADE_WARNING: Put は FilePut にアップグレードされ、新しい動作が指定されています。
    ' 詳細については、ここをクリックしてください。
    ' 'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1041"'
    FilePut(fileh, emp, recordnum)
    FileClose(fileh)
End Sub
```

employee 構造体を初期化して、phone_ext フィールドのサイズを設定するには、太字で示したステートメントを使用して Initialize メソッドを呼び出す必要があります。FilePut および FileGet メソッドの呼び出しの前に、動作が異なるというアップグレードの警告が含められていることに注意してください。この警告は、関数が動的配列または文字列を引数として受け取った場合のことを指しています。この場合、2 バイト長の記述子が追加され、取得されたファイルのサイズは異なります。ただし、この例では固定長フィールドが使用されているため、この動作の違いは該当しません。したがって、警告を削除できます。

文字列操作とファイル操作の手動による変更

どのプログラミングにも言えることですが、さまざまな文字列操作やファイル操作を実行する方法は 1 つとは限りません。前のセクションでは、アップグレード ウィザードが Visual Basic 6.0 から Visual Basic .NET に文字列およびファイル コードをアップグレードする際に自動的に適用される変更について説明しました。ただし、この言語の新しいバージョンには、ファイル操作のパフォーマンスや管理性を改善し、ファイル操作を簡潔にする多数の強力な文字列およびファイル メソッドとクラスが用意されています。ここでは、その一部を紹介します。

StringBuilder を使用した文字列の置換

Visual Basic 6.0 と Microsoft Visual Studio .NET では、文字列は不変です。つまり、文字列を宣言し、文字列に値を割り当てると、メモリに保持された文字列の値は変わることはありません。以下のコード例は、この特性による影響を示しています。

```
Dim sMyString As String
sMyString = "Hello"
sMyString = sMyString + " World"
```

前のコードでは、1 つの文字列が変更されているように見えますが、実際には 3 つの文字列が作成されています。最初の文字列は、文字列を宣言する第 1 行で作成されます。2 つ目の文字列は、sMyString 変数に "Hello" という値が割り当てられたときに実行され、3 つ目の文字列は、文字列と "World" という値が連結されたときに作成されます。

このような文字列の特性には欠点があります。初心者の場合、簡単なタスクを実行するために実際に必要なメモリよりも多くのメモリを使用することになります。ガベージ コレクタによるクリーンアップが必要なオブジェクトがあと 2 つ残っています。また、オブジェクトの作成、オブジェクトのコピーの作成、そして古い文字列のメモリからの除去などに余分な CPU サイクルが必要になるため、パフォーマンスのオーバーヘッドも生じます。

プログラムの実行中に格納されている値が頻繁に変更されない場合は、String 変数を使用することも可能です。一定数の文字列が連結される前例のような場合には、String データ型を使用するのが妥当です。

文字列が関与する操作の数が事前にわからない場合、また多数の変更が見込まれる場合は、文字列値が変更されたときにもオブジェクトが新規作成されることがない StringBuilder クラスを使用するのが最適です。文字列にテキストを追加するなどの操作には、操作ごとにメモリに新しいオブジェクトを作成するという面でのオーバーヘッドが少ないことから、String 連結よりも、StringBuilder を使用した方がパフォーマンスは向上します。

Visual Studio .NET では、StringBuilder クラスは System.Text 名前空間にあります。以下のコード例は、StringBuilder クラスを使用するように前述のコードを変更する方法を示しています。

```
Dim sMyString As New StringBuilder("Hello ")
sMyString.Append(" World")
```

元のコード例とは対照的に、このコードでは StringBuilder オブジェクトが変更されるたびに新しいオブジェクトが作成されることはありません。StringBuilder で内容変更操作が実行されても、メモリ内の該当するデータは変更されません。StringBuilder から String の内容をフェッチする必要が生じた場合でも、toString() メソッドを使用すればこの情報を取得できます。

以下の文字列置換関数に示すように、1 つの文字列変数を変更する一連の文字列操作に StringBuilder を使用すると、パフォーマンスが飛躍的に向上します。

```
Public Shared Function ReplaceSymbols(str As String) As String
    Dim tmpStrB As New StringBuilder(str)
    tmpStrB.Replace("#"c, "!"c, 15, 29)
    tmpStrB.Replace("!"c, "o"c)
    tmpStrB.Replace("cat", "dog")
    tmpStrB.Replace("dog", "fox", 15, 20)
    Return tmpStrB.ToString()
End Function
```

つまり、以前の Visual Basic 6.0 コードで使用されている String のすべてのインスタンスを Visual Basic .NET では StringBuilders を使用するように変更する必要があるということでしょうか。いいえ、文字列を変更しない場合、または連結する文字列が少数の場合は、文字列を変更する必要はありません。これに対し、多数の操作（連結、大文字小文字の変更、文字の置換/挿入など）を実行する場合は、String インスタンスを StringBuilders に変更した方が効率的です。この方法に従えば、アプリケーションで**文字列**操作によるパフォーマンスの低下に悩まされることはありません。

複雑な文字列操作の正規表現への置換

Visual Basic .NET を含む Visual Studio .NET プログラミング言語は、正規表現を十分に活用できる能力を備えています。.NET Framework の正規表現エンジンへのアクセスを提供するクラスを利用するには、System.Text.RegularExpressions 名前空間をコードにインポートする必要があります。

正規表現を使用すれば、プログラマは一般的な方法で検索できるパターンを定義できます。これが非常に役立つのにはいくつかの理由があります。正規表現を使用することで、これまで多数の行が必要だった作業も数行のコードで実行できるようになります。したがって、初心者にとっては、特定のタスクを実行するために

必要なコード記述作業が大幅に軽減されます。以下のコード例は、ある文字パターンが 1 つの文字列に出現する数を検索する Visual Basic .NET で書かれた関数を示しています。

```
Function findOccurrences(ByVal sHaystack As String, ByVal sNeedle As String) _
        As Integer
    Dim iPosition As Integer
    Dim iCount As Integer
    For i As Integer = 1 To sHaystack.Length
        iPosition = InStr(i, sHaystack, sNeedle)
        If (iPosition > 0) Then
            iCount = iCount + 1
            i = iPosition
        End If
    Next i
    Return iCount
End Function
```

正規表現を利用すると、この関数は以下のように簡潔に書き換えることができます。

```
Function findOccurrences2(ByVal sHaystack As String, ByVal sNeedle As String) _
        As Integer
    Return Regex.Matches(sHaystack, sNeedle).Count
End Function
```

この 2 つの関数を比較すれば、2 つ目の関数の方が最初の関数よりも必要なコード行が少ないことは明らかです。一時変数を現在の結果に格納する必要もなく、プロシージャのループ セクションを記述する必要もありません。つまり、このコードは行数が減っただけでなく、より効率的で理解しやすくなっています。前のコード例は非常に単純な例です。ただし、問題がもっと複雑な場合でも、何百行ものコードを簡単な式に変換できます。

異なるオプションで同じ検索を実行するように関数を変更する場合でも、正規表現のオプションを変更するだけで済みます。たとえば、関数で大文字小文字を区別しないで検索を実行する場合は、Matches 関数にオプション パラメータとして RegexOptions.IgnoreCase を渡すだけです。正規表現を使用しない場合は、元の findOccurences メソッドのコードに多数の変更を加える必要があります。RegularExpressions 名前空間には、複雑なプロシージャを文字列で実行するうえで役立つ多数のオプションがあります。この名前空間で使用可能なさまざまな列挙値の詳細については、MSDN の『.NET Framework Class Library』の「System.Text.RegularExpressions Namespace」を参照してください。

アップグレード後のコードで複雑な入力検証や複雑な文字列変更を処理する場合は、正規表現の使用を検討することをお勧めします。たとえば、パスワードは最低でも 8 文字で、少なくとも数字と特殊文字を 1 つずつ含んでいなければならないという複雑な要件が定義された Visual Basic 6.0 パスワード検証ルーチンをアップグレードするとします。正規表現を使用しない場合、ユーザーが選択したパスワードを解析するために、この関数のコードには多数の行を記述する必要があります。この種のアルゴリズムを記述するために使用でき

る Visual Basic 6.0 のプロシージャを使用した場合は、コードは非常に読みずらく、必要以上に複雑になってしまいます。RegularExpression 名前空間のクラスを使用して、正規表現をうまく使いこなせば、たった 1 行のコードで検証を実行できます。

Regex.Replace 関数も、本来なら長いコード行を書く必要がある複雑な文字列操作を、ほんの数行で処理するのに役立ちます。以下のコード例では、関数に渡す姓名の順序が逆になります。

```
' 入力は (last_name,first_name) 形式であると想定します。
Function switchPlaces(ByVal sInput As String) As String
   return Regex.Replace(sInput, "(?<last>.+\D),(?<first>.+\D)", _
        "${first} ${last}")
End Function
```

Replace 関数は、1 つの入力文字列と、入力文字列で検出されるパターンを指定する正規表現を受け取ります。最後の引数は、見つかったインスタンスの要素をどのように置換するかを指定するものです。置換パターンにある特殊な構造体は、文字エスケープと代入だけです。たとえば、置換パターンの a*${test}b は、文字列 "a*"、"test" キャプチャグループと一致するサブストリング (存在する場合)、文字列 "b" の順に挿入します。その他の例として、$123 は、グループ番号 123 (10 進数) と一致する最後のサブストリングを代入し、${name} は (?<name>) グループと一致する最後のサブストリングを代入します。正規表現の詳細については、MSDN の『.NET Framework General Reference』の「Regular Expression Language Elements」を参照してください。

このコードは、正規表現を使用していかに文字列操作を簡略化できるかを示すもう 1 つの例です。パラメータは、正規表現を使用して解析され、取得されたサブストリングの格納には後方参照が使用されています。一致する文字列は、メソッド呼び出しで指定された形式に基づいて置換されます。

.NETでの正規表現の実装では、関数を使って複雑な文字列操作を実行するよりも、効率的で管理しやすい方法で正規表現を記述できます。正規表現に慣れるまでには少し時間がかかるかもしれませんが、複雑な文字列操作を難なく効率的に実行できるようになることで、その努力は必ず報われます。

ストリームを使用したファイル I/O の改善

アップグレード ウィザードでは、.NET に用意されている Visual Basic 6.0 互換性ライブラリを使用して、ほとんどのファイル アクセス方式をアップグレードできます。ただし、アップグレード ウィザードによって生成される結果では、Visual Basic 互換性ライブラリのメソッドが使用されます。ファイル アクセスをアップグレードする以外の方法としては、Visual Basic .NET の新しいファイル アクセスを使用する方法があります。.NET では ストリームを使用することをお勧めします。

たとえば、以下のようなコードがあるとします。

```
Dim FileName As String
Dim MyChar As String

FileName = "C:\Temp\TextBox.txt"
FileOpen(1, filename, OpenMode.Input)          ' ファイルを開きます。
Do While Not EOF(1)                            ' EOF までループします。
    MyChar = InputString(1, 1)                 ' 1 文字取得します。
    System.Diagnostics.Debug.WriteLine(MyChar) ' イミディエイト ウィンドウに出力します。
Loop
FileClose(1)
```

前述のように、このコードは、アップグレード ウィザードで自動的にアップグレードできます。ただし、ストリームを使用して同じ効果を得るためには、操作を手動で書き換える必要があります。

ほとんどの新規開発プロジェクトでは、.NET でファイルからデータを読み書きするにはストリームを使用するのが最善の方法です。ストリームとは、読み取り、書き込み、シークを可能にする一連のバイトの総称です。ストリームは、ファイルや他のソース (ネットワーク接続など) と関連付けることができ、非常に最適化されるため、互換性ライブラリを使用したファイル アクセスよりも高速です。

元の Visual Basic 6.0 ファイルのコード サンプルは、Visual Basic .NET のテキスト ファイル関数ではなく、ストリームを使用して書き換えることができます。修正したコードは以下のようになります。

```
Dim fs As FileStream
Dim MyChar(1) As Char
Dim reader As System.IO.StreamReader

fs = New FileStream("C:\Temp\TextBox.txt", FileMode.Create)
fs.Seek(0, SeekOrigin.Begin)
reader = StreamReader(fs, System.Text.Encoding.ASCII)
Do While reader.Peek <> - 1
   reader.Read(MyChar, 0, 1)
   System.Diagnostics.Debug.WriteLine(MyChar(0))
Loop
reader.Close()
```

このコード サンプルを見るとわかるように、ストリームからの読み取りは非常に単純です。ファイル ハンドルに関連付けられたファイルを開くのではなく、StreamReader オブジェクトを作成してファイルを開きます。ファイルを開いたら、StreamReader メソッドの Read を使用して、次の文字を取得します。Peek プロパティにより、1 つ先の文字を参照して、ファイルの最後に到達したかどうか (Peek が –1 を返した場合) を判断できます。ファイルの内容を読み込んだら、Close メソッドでストリームを閉じます。

Visual Basic 2005 の場合 :

System.IO.File.ReadAll メソッドは、ファイルを開く、内容を文字列変数に読み込む、ファイルを閉じる、という機能を 1 つのメソッドで提供します。

```
Dim path As String
Dim fileContents As String
fileContents = File.ReadAll(path)
```

ファイル書き込みのための対応するメソッドは、System.IO.File.WriteAll です。

ファイルの内容の読み込みには、StreamReader クラスの Read、ReadBlock、ReadLine、および ReadToEnd メソッドが使用されます。StreamWriter クラスは、ストリームを書き込むためのメソッドを提供します。書き込みプロセスは、読み込みの場合とほぼ同じです。まず、StreamWriter オブジェクトを作成し、適切なメソッドを使ってストリームに書き込んでから、ストリームを閉じます。ストリームへの書き込みには、StreamWriter クラスの Write メソッドと WriteLine メソッドを使用します。

従来の Visual Basic 6.0 のシーケンシャル I/O には、ストリーミングと共通する特性があります。ストリーミングにも、バイナリ I/O と共通する部分があります。前の例で示したように、ストリーミング機能によるシーケンシャル アクセスは、FileStream クラスと StreamWriter クラスを使用して実現できます。ファイル アクセスには ASCII エンコードが使用されることに注意してください。これは、.NET 文字列では Unicode エンコードが使用されていますが、Visual Basic 6.0 を意識して書かれたファイルにアクセスするには、ASCII エンコードを使用する必要があるためです。

ストリームでは、現在の位置が保持されます。これは、次の操作を実行する場所へのポインタになります。StreamWriter オブジェクトと StreamReader オブジェクトはいずれも、操作を実行するたびに現在の位置を更新します。

Visual Basic 6.0 シーケンシャル アクセスに関して、ストリームには重要な制限事項があります。区切りデータを読み込む機能が組み込まれていないことです。区切りデータをストリームで処理する場合は、プログラマがデータを解析する必要があります。

ファイルのシーケンシャル書き込みは、文字列と行終端子の書き込みが可能な StreamWriter.WriteLine メソッドを使用して処理します。書き込み操作中にファイルに任意の量のデータを配置できるファイルのバイナリ書き込みには、StreamWriter.Write メソッドを使用します。このメソッドを使用すると、現在の位置に異なるデータ型を書き込むことができます。

Stream クラスは System.IO 名前空間にあります。また、この名前空間には、ディレクトリ処理などの他のファイル操作用のクラスも含まれています。以下のコード例は、Directory クラスの GetDirectories メソッドを使用して、C:\Temp ディレクトリにあるサブディレクトリの一覧を取得する方法を示しています。

```
Dim directories() As String
Dim i As Integer
directories = System.IO.Directory.GetDirectories("C:\temp")
For i = 0 To directories.Length - 1
    System.Diagnostics.Debug.WriteLine(directories(i))
Next
```

System.IO 名前空間の詳細については、Visual Basic .NET のドキュメントを参照してください。

ファイル システム オブジェクト モデルを通じたファイル アクセス

Visual Basic .NET でファイル アクセスやファイル操作を実行するためのもう 1 つの選択肢が、ファイル システム オブジェクト (FSO) モデルです。このモデルは、ファイルやフォルダ (ディレクトリ) をオブジェクト指向的な方法で操作するためのオブジェクトおよびメソッドを提供します。このモデルは、Visual Basic Scripting タイプ ライブラリ (Scrrun.dll) を通じて提供されます。

FSO モデルを使用して、ファイルやフォルダの作成または削除、ファイルやフォルダに関する情報 (パス情報など) の取得、ファイルやフォルダのコピーおよび移動などの操作を実行できます。さらに、このモデルでは、テキスト ファイルを読み書きするためのオブジェクトを作成できる TextStream というクラスも使用できます。バイナリ ファイルの読み書きは、FSO モデルではサポートされていません。

以下のコード例は、ファイルやディレクトリへのアクセスに FSO モデルを使用する非常に簡単な方法を示しています。この方法では、FSO モデルを使いやすくするために、FileSystemObject オブジェクトを作成します。さらに、TextStream オブジェクトを作成して、コンソールに内容を表示するテキスト ファイルを開きます。

```
Private Sub Command1_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles Command1.Click

    Dim fs As Scripting.FileSystemObject
    Dim inFile As Scripting.TextStream
    Dim filename As String

    ' ファイル名を取得します。
    …
    ' ファイル操作用の FileSystemObject を作成します。
    fs = CreateObject("Scripting.FileSystemObject")
    ' 読み込むテキスト ファイルを開きます。
    inFile = fs.OpenTextFile(filename, Scripting.IOMode.ForReading)
    ' テキスト ファイルの内容をコンソールに表示します。
    While Not inFile.AtEndOfStream
```

```
        System.Console.WriteLine(inFile.ReadLine)
    End While
    inFile.Close()

    ' ファイルを一時ディレクトリにコピーします。
    fs.CopyFile(filename, "C:\Temp\", True)
End Sub
```

FileSystemObject のメソッドとプロパティを使用して、一時ファイル名や一時フォルダ名の作成、特定のファイルまたはフォルダの存在の確認、ファイルやフォルダの削除などの操作を実行することもできます。スクリプト ライブラリを通じて使用できるファイル アクセスの FSO モデルについて、ここですべてを説明することはできません。ファイル アクセスの FileSystemObject モデルの詳細、または Visual Basic .NET で使用可能な複数のファイル アクセス モデルから適切なモデルを選択するうえでのヒントについては、Visual Basic .NET ヘルプの「Accessing Files with FileSystemObject」と「Choosing Among File I/O Options in Visual Basic .NET」を参照してください。これらのドキュメントは、MSDN でも参照できます。

まとめ

Visual Basic .NET での変更により、以前のバージョンの一部の機能が使用できなくなりました。これらの機能をアップグレードする際の作業を軽減できるように、互換性ライブラリが用意されています。互換性ライブラリに機能がない場合でも、新しいオブジェクトや関数を通じて、ほぼ同じ機能が Visual Basic .NET でも提供されています。この章では、使用が中止された Visual Basic 6.0 の一般的な機能について説明し、これらの機能を Visual Basic .NET の機能に置き換える方法を解説しました。

詳細情報

RegularExpressions 名前空間で使用可能なさまざまな列挙値の詳細については、MSDN の『.NET Framework Class Library』の「System.Text.RegularExpressions Namespace」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemtextregularexpressions.asp です。

正規表現の詳細については、MSDN の『.NET Framework General Reference』の「Regular Expression Language Elements」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconregularexpressionslanguageelements.asp です。

ファイルアクセスの FileSystemObject モデルの詳細、または Visual Basic .NET で使用可能な複数のファイルアクセスモデルから適切なモデルを選択するうえでのヒントについては、Visual Basic .NET ヘルプまたは MSDN の「Accessing Files with FileSystemObject」と「Choosing Among File I/O Options in Visual Basic .NET」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vbconintroductiontofilesystemobjectmodel.asp
および
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vatskchoosingamongfileiooptionsinvisualbasicnet.asp です。

12

データ アクセスのアップグレード

ほぼすべてのビジネス アプリケーションにはなんらかのデータ アクセスがあります。概念上、多くのビジネスがデータに基づいていることを考えると、これはごく当然とも言えます。顧客情報、注文情報、在庫情報、購入情報など、ビジネス情報はすべて、容易に取得でき、かつ更新しやすい方法で格納する必要があります。データベースに格納された情報を扱いやすくするために、多くの Visual Basic アプリケーションが開発されました。

Visual Basic には、データにアクセスするための 3 つのインターフェイスとして、ADO (ActiveX データ オブジェクト)、RDO (リモート データ オブジェクト)、および DAO (データ アクセス オブジェクト) があります。データ アクセス インターフェイスはデータ アクセスのさまざまな要素を表すオブジェクト モデルです。Visual Basic を使用することによって、接続、ステートメント ビルダ、および任意のアプリケーション向けに返されるデータをプログラムで制御できます。

これまでの Visual Basic では、新しいバージョンがリリースされるたびにデータ アクセスが拡張されてきました。Visual Basic .NET でも同様です。この最新バージョンでも ADO とデータ連結が引き続きサポートされています。DAO と RDO もサポートされていますが、これらのテクノロジのデータ連結はサポートされていません。Visual Basic .NET では、拡張機能として ADO.NET という新しい種類のデータ モデルが導入されました。

この章では、3 つの主要なデータ アクセス テクノロジ (ADO、DAO、および RDO) をアップグレードする方法と、Visual Basic 6.0 のデータ レポートを Visual Basic .NET の Crystal Reports に変換する方法について説明します。このガイドでは、アップグレード後のアプリケーションには既存のアプリケーションと同等の機能が実現されていることを前提条件にしています。そのため、この章では ADO.NET へのアップグレードについては説明しません。ADO.NET の詳細については、第 20 章「一般的なテクノロジ シナリオ」の「ADO から ADO.NET へのアップグレード」を参照してください。

一般的な注意点

Visual Basic アプリケーションのなかには、SQL Server や Microsoft Access などのデータベース システムにデータを格納するアプリケーションが数多く存在します。これらのアプリケーションは、データ連結を使用する ADO、データ連結を使用しない RDO など、それぞれが異なる方法を使ってデータにアクセスします。データ アクセスのコードをアップグレードする場合は、事前にコードを検証し、使用されているデータ アクセスの種類を確認してください。データ アクセスのコードをアップグレードする方法は、使用されているテクノロジと、データ連結が使用されているかどうかに依存します。この章では、データ アクセスをアップグレードする方法について検証します。

ADO (ActiveX データ オブジェクト)

Microsoft ActiveX データ オブジェクト (ADO) は当初、Microsoft Internet Information Server (IIS) のデータ アクセス インターフェイスとして導入されました。ADO は、シンプルかつ容易にデータにアクセスできるインターフェイスです。ADO は、多くのツールと言語に対応できるインターフェイスへのニーズの増加に伴い、RDO と DAO の最も優れた部分を包括するインターフェイスとして拡張され続けています。最終的には、現在最も広く使用されているデータ アクセス インターフェイスである RDO と DAO に取って代わるインターフェイスです。ADO は多くの面で RDO と DAO に似ています。たとえば、ADO では RDO と DAO に似た言語規則が使用されます。ADO が提供するセマンティクスはよりシンプルであるため、今日の開発者にとって習得しやすくなっています。

ADO は、OLE DB へのアプリケーション レベルでのインターフェイスとして設計されています。OLE DB はマイクロソフト社が提供する、最新かつ最も強力なデータ アクセスのパラダイムです。OLE DB は、すべてのデータ ソースへの高パフォーマンスのアクセスを提供します。ADO と OLE DB の組み合わせは Universal Data Access 戦略の基盤を構築します。OLE DB はすべてのデータ ソースに対する普遍的なアクセスを可能にします。ADO を使用することで、開発者はより容易にデータ アクセスをプログラムすることができます。ADO は OLE DB に基づいて構築されているため、普遍的なデータ アクセスを提供する OLE DB の豊富なインフラストラクチャを活用できる利点があります。

ADO オブジェクト モデルはプログラム可能なオブジェクトのコレクションを定義します。これらのオブジェクトは、Visual Basic のように COM とオートメーションの両方をサポートする任意のプラットフォームで使用することができます。ADO オブジェクト モデルは、OLE DB の最も一般的に使用される機能を公開するように設計されています。

ADO を使用することで、データ オブジェクト、データ連結、および ADO データ環境のような ADO デザイン時ツールを操作することができます。アプリケーションのユーザーがランタイムの動作をデータ アクセスの別コンポーネントとみなすことがありますが、この場合、ランタイムの動作は 3 つのコンポーネントのそれぞれの要素として考えることができます。以降では、これらの 3 コンポーネントをそれぞれ検証し、Visual Basic 6.0 と Visual Basic .NET での違いについて説明します。

ADO データ連結のアップグレード

データ連結は、Visual Basic 6.0 で最も強力にデータ アクセスを使用するものの 1 つです。データ連結を使用すると、コントロールのプロパティをデータベース内の特定のフィールドにバインドすることができます。バインドされたコントロール プロパティ (テキスト ボックスの Text プロパティなど) がコントロールのユーザーによって変更された場合、コントロールは値が変更されたことをデータベースに通知します。その後、コントロールは、現在のレコードの関連するフィールドを更新するように要求します。このプロセスが発生すると、それに対しデータベースは、正常に更新できたかどうかをコントロールに返します。コントロール内のデータ連結の主なプロパティは、DataChanged、DataField、DataFormat、DataMember、および DataSource です。

ここで説明するアップグレード シナリオでは、プロジェクトに Microsoft ActiveX データ オブジェクトへの参照が追加され、かつ少なくとも 1 つのフォームに ADO **データ コントロール** が含まれ、かつ DataSource プロパティ、DataMember プロパティ、および DataField プロパティのいずれかのデータ連結を使用するコントロールを 1 つ以上持つアプリケーションすべてが対象となります。

通常、ADO コードの多くは、アップグレード ウィザードで自動的にアップグレードされます。手動で最小限の変更を加えることで、Visual Basic 6.0 で動作していたアプリケーションとデータ アクセス テクノロジを再現することができます。たとえば、アップグレード ウィザードによって自動的に参照される Interop.ADODB.dll の代わりに、ADODB.NET アセンブリ (Adodb.dll) への参照を追加する必要がある場合があります。

Visual Basic .NET は ADO データ連結だけをサポートします。DAO または RDO データ連結はサポートされていません。ADO データ連結は、Visual Basic 6.0 で ADO データ連結を管理する COM ライブラリに組み込まれているため、サポートされています。Visual Basic .NET では、互換性ライブラリである Microsoft.VisualBasic.Compatibility.Data に同等の機能が含まれています。Visual Basic .NET では、このライブラリがデータ連結を管理します。この結果、プロパティ、メソッド、およびイベントは、同じ言語の 2 つのバージョンの間で互換性を持つことになるため、ADO コントロールおよび Data Environment を自動的にアップグレードすることができます。

アップグレード ウィザードは、Visual Basic 6.0 と同じ機能を維持するために、いくつかのサポート関数を追加します。その理由は、Visual Basic .NET では、コード内で参照されるすべてのオブジェクトが使用前にインスタンス化される必要があるためです。アップグレード ウィザードが追加するコードによってこの条件は確実に実装されます。また、デザイン時に作成されたデータ連結がコード内に含まれる場合、アップグレード ウィザードは ADO とのデータ連結を実現するために使用されるバインド変数とプロシージャを追加します。

例として、TexBox を含むフォームを考察してみます。ADO データ コントロールは TextBox コントロールの DataSource プロパティとして設定され、TextControl の DataField プロパティには Title が指定されています。アップグレード ウィザードを使ってこのようなフォームをアップグレードする場合、データ連結が Visual

Basic .NET でも動作するように、アップグレード後のコードには変数とプロシージャが追加されます。たとえば、コードは以下のようになります。

```
Private ADOBind_Adodc1 As VB6.MBindingCollection

Public Sub VB6_AddADODataBinding()
    ADOBind_Adodc1 = New VB6.MBindingCollection()
    ADOBind_Adodc1.DataSource = CType(Adodc1, msdatasrc.DataSource)
    ADOBind_Adodc1.Add(Text1, "Text", "Title", Nothing, "Text1")
    ADOBind_Adodc1.UpdateMode = VB6.UpdateMode.vbUpdateWhenPropertyChanges
    ADOBind_Adodc1.UpdateControls()
End Sub
Public Sub VB6_RemoveADODataBinding()
    ADOBind_Adodc1.Clear()
    ADOBind_Adodc1.Dispose()
    ADOBind_Adodc1 = Nothing
End Sub
```

ただし、実行時に設定された組み込みコントロール内のデータ連結のアップグレードには問題があります。これらのコントロールはネイティブな Visual Basic .NET へアップグレードされますが、アップグレード ウィザードでは DataSource、DataMember、および DataField をアップグレードしません。デザイン時のデータ連結もサポートされません。アップグレード ウィザードはバインド変数またはプロシージャを追加しないため、アップグレード後のコードを変更する必要があります。

labAuthor という名前を持つラベル コントロールと、Adodc1 という名前を持つ ADO データ コントロールを含むフォームのあるプロジェクトの場合を例として、この状況を説明します。Adodc1 は Microsoft Access データベースである Northwind に接続されています。次のコード例に示すように、実行時に ADO データ コントロールを DataSource プロパティに設定し、DataField には EmployeeID を指定します。

```
Private Sub Form_Load()
    Set Me.labAuthor.DataSource = Adodc1
    Me.labAuthor.DataField = "EmployeeID"
End Sub
```

このコードをアップグレードした場合、アップグレード後のコードは修正する必要があります。アップグレード後のコードは以下のようになります。

```
Private Sub frmADO_Load(ByVal eventSender As System.Object, _
            ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    ' UPGRADE_ISSUE: ラベル プロパティ labAuthor.DataSource はアップグレードされませんでした。
    Me.labAuthor.DataSource = Adodc1
    ' UPGRADE_ISSUE: ラベル プロパティ labAuthor.DataField はアップグレードされませんでした。
    Me.labAuthor.DataField = "EmployeeID"
End Sub
```

アップグレード ウィザードは、データ連結が機能するために必要な変数やプロシージャを自動的に作成しません。そのためのコードは手動で追加する必要があります。アップグレード後は、ADO データ コントロールを

使用し、ラベル コントロールがバインドするために必要な Text プロパティを追加し、問題のある命令を削除する必要があります。現在の例での、修正済のコードを次に示します。

```
Private ADOBind_Adodc1 As VB6.MBindingCollection

Public Sub VB6_AddADODataBinding()
    ADOBind_Adodc1 = New VB6.MBindingCollection
    ADOBind_Adodc1.DataSource = CType(Adodc1, msdatasrc.DataSource)
    ADOBind_Adodc1.Add(labAuthor, "Text", "EmployeeID", Nothing, "labAuthor")
    ADOBind_Adodc1.UpdateMode = VB6.UpdateMode.vbUpdateWhenPropertyChanges
    ADOBind_Adodc1.UpdateControls()
End Sub

Public Sub VB6_RemoveADODataBinding()
    ADOBind_Adodc1.Clear()
    ADOBind_Adodc1.Dispose()
    ADOBind_Adodc1 = Nothing
End Sub
```

これらのサブルーチンと変数を追加後、元のコードの DataSource プロパティと DataField プロパティが割り当てられていた場所で、適切なサブルーチンを呼び出す必要があります。現在の例では、これらのプロパティは Load Event コード内に割り当てられていました。適切な調整の例としては、以下のコードのようになります。

```
Private Sub frmADO_Load(ByVal eventSender As System.Object, _
                        ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    VB6_AddADODataBinding()
End Sub
```

これらの変更後は、アップグレード後の Visual Basic .NET でのコードの動作が、Visual Basic 6.0 でのコードの動作と同等になります。

ADO データ連結を使用しないプロジェクト

前のセクションでは、データ連結を使用するコードをアップグレードする方法と、Visual Basic .NET で同じ動作を実現するために必要な変更について説明しました。ただし、データベースへのアクセスには、データ連結を必要としないものもあります。

アップグレードするアプリケーションのプロジェクトが、Microsoft ActiveX データ オブジェクトを参照していても、ADO データ コントロールを含まない場合は、ここで説明するアップグレード シナリオを使用します。この場合、すべてのデータ アクセスは実行時に直接管理されます。

データ連結を使用せずにデータベースにアクセスするプロジェクトを自動的にアップグレードする場合、アップグレード後のコードを変更する必要はありません。アップグレード ウィザードは、ADO を含む Microsoft.VisualBasic.Compatibility ライブラリへの参照を自動的に追加します。すべての ADO コードは元の Visual Basic 6.0 プロジェクトと同じように動作し、修正する必要はありません。

ただし、アップグレード後のコードと元のソース コードでは、レコードセット内のフィールドにアクセスする方法が異なります。Visual Basic 6.0 では、一般的に、短縮形のコーディング規約 (RecordsetName!FieldName) を使ってフィールドを参照します。Visual Basic .NET では、このコードを展開して、既定のプロパティを解決する必要があります。アップグレード後には、得られるコードは多少異なって見えます。次のコード例でもそのようになっています。

```
Dim cn As New Connection
Dim rs As Recordset
cn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Temp\Northwind.mdb"
Set rs = cn.Execute("Select * From Products where ProductID = 1")
txtProduct.Text = rs!ProductName
txtCompany.Text = rs!SupplierID
rs.Close
cn.Close
```

この Visual Basic 6.0 のコードは特定の製品の製品名と供給業者 ID を取得します。この情報は 2 つの TextBox コントロールに割り当てられています。このコードをアップグレードすると、新しいコードは次の例のようになります。

```
Dim cn As New ADODB.Connection
Dim rs As ADODB.Recordset
cn.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Temp\Northwind.mdb")
rs = cn.Execute("Select * From Products where ProductID = 1")
txtProduct.Text = rs.Fields("ProductName").Value
txtCompany.Text = rs.Fields("SupplierID").Value
rs.Close()
cn.Close()
```

アップグレード ウィザードを使用すると、レコードセットのフィールドが展開されることに注意してください。

Data Environment のアップグレード

Data Environment は、Visual Basic 6.0 の機能です。この機能は、プログラムによるランタイム データ アクセスを作成するための、対話的なデザイン時環境を提供します。Visual Basic .NET は、データベース アプリケーションを構築するための、より強力な環境を提供しますが、基盤となるデータ アクセス ライブラリの概念が異なります (Visual Basic 6.0 は ADO を使用しますが Visual Basic .NET は ADO.NET を使用)。そのため、Visual Basic 6.0 の Data Environment と Visual Basic .NET のデータ アクセス機能との間には、直接的なマッピングはありません。

Data Environment を使用することによって、Visual Basic 6.0 の開発者は次のタスクを実現することができます。

- ADO 接続オブジェクトの作成
- ストアド プロシージャ、テーブル、ビュー、類義語、および SQL ステートメントに基づく ADO コマンド オブジェクトの作成

- コマンド オブジェクトのグループ化、または 1 つ以上のコマンド オブジェクトの組み合わせに基づく、階層化されたコマンドの作成
- Data Environment にホストされるオブジェクトのプロパティを読み取り、設定するためのコードの作成と実行
- Data Environment に含まれるコマンドの、プログラム的なランタイム メソッドとしての実行
- Data Environment にホストされたコマンドに対するフォーム コントロールのバインド
- コマンド階層内の値を自動的に算出する集約関数の作成

アップグレード ウィザードを使って Data Environment を使用するアプリケーションをアップグレードすると、次の変更が加えられます。

- Visual Basic 6.0 プロジェクト内の Data Environment ごとに、その Data Environment の名前の先頭に DataEnvironment を付けた名前を持つクラスが作成され、モジュール内に Data Environment の名前を持つ変数がインスタンス化されます。
- Data Environment オブジェクトによってホストされる接続とレコードセットごとに、同じ名前を持つ Public WithEvents メンバ変数が作成されます。
- コマンドごとに同じ名前を持つパブリック メソッドが作成されます。このメソッドは、対応するコマンドを Visual Basic 6.0 で使用する場合と同じ方法で使用できます。

ここに例として使用するアプリケーションには、DE という名前を持つ Data Environment が含まれます。この Data Environment には Access という名前の接続が 1 つ含まれ、Orders という名前の**コマンド**も含まれます。アップグレード ウィザードを使ってこのアプリケーションをアップグレードすると、次のコード例に示すようなクラスが作成されます。

```
Module DataEnvironment_test_Module
    Friend DE As DataEnvironment_DE = New DataEnvironment_DE()
End Module

Friend Class DataEnvironment_DE
    Inherits VB6.BaseDataEnvironment
    Public WithEvents Access As ADODB.Connection
    Public WithEvents rsOrders As ADODB.Recordset
    Private m_Orders As ADODB.Command

    Public Sub New()
        MyBase.New()
        Dim par As ADODB.Parameter
        Dim connectStr as String

        Access = New ADODB.Connection()
        connectString = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source=C:\Temp\Northwind.mdb;Persist Security Info=False;"
        Access.ConnectionString = connectString
        m_Connections.Add(Access, " Access")
        m_Orders = New ADODB.Command()
```

```
        rsOrders = New ADODB.Recordset()
        m_Orders.Name = "Orders"
        m_Orders.CommandText = "SELECT OrderID, OrderDate FROM Orders"
        m_Orders.CommandType = ADODB.CommandTypeEnum.adCmdText
        rsOrders.CursorLocation = ADODB.CursorLocationEnum.adUseClient
        rsOrders.CursorType = ADODB.CursorTypeEnum.adOpenStatic
        rsOrders.LockType = ADODB.LockTypeEnum.adLockReadOnly
        rsOrders.Source = m_Orders
        m_Commands.Add(m_Orders, "Orders")
        m_Recordsets.Add(rsOrders, "Orders")
    End Sub

    Public Sub Orders()
        If Access.State = ADODB.ObjectStateEnum.adStateClosed Then
            Access.Open()
        End If
        If rsOrders.State = ADODB.ObjectStateEnum.adStateOpen Then
            rsOrders.Close()
        End If
        m_Orders.ActiveConnection = Access
        rsOrders.Open()
    End Sub
End Class
```

コード内の Data Environment の用途すべてには同じフォーマットと同じ動作が維持されますが、この新しいクラスと Data Environment オブジェクトには大きな違いがあります。新しいクラスは Visual Basic 6.0 とは異なり、視覚的に操作することができません。コード内にデータ連結を使用する Data Environment オブジェクトが使用されている場合、データ連結は正常に動作しますが、新しいコマンドを視覚的に追加することはできません。次に、アップグレード プロセスについて説明します。

データ連結を使用する Data Environment のアップグレード

Visual Basic 6.0 の Data Environment は、データと連動するコントロールを Data Environment オブジェクトに直接バインドするためのデータ ソース インターフェイスを公開します。そのためには、コントロールの DataSource プロパティに Data Environment オブジェクトの名前を指定し、DataMember プロパティには Data Environment オブジェクトによってホストされる**コマンド** オブジェクトの名前を指定します。この**コマンド** オブジェクトは、取得するデータを含むレコードセットのソースとなります。DataField プロパティにはデータの取得元となる列の名前を指定します。Data Environment ランタイムはコントロールがデータ ソースとして使用されていることを検知し、バインドされたコントロールを含む Visual Basic 6.0 フォームを表示する前に次の動作を始めます。

- ユーザー インターフェイス コントロールにバインドされたコマンドが使用するすべてのデータベース接続を開きます。
- レコードセットを作成するのに必要なコマンドすべてを実行します。
- DataMember プロパティと DataField プロパティに従って、すべてのユーザー インターフェイス コントロールを、前のタスクで取得したレコードセットにバインドします。

Visual Basic .NET によって使用される WinForms ライブラリは ADO.NET データ アクセス ライブラリ向けに最適化されているため、WinForms コントロールを ADO レコードセットに直接バインドすることはできません。WinForms と ADO 間のバインドを可能にするために、アップグレード ウィザードは Microsoft.VisualBasic.Compatibility.Data ライブラリの MBindingCollection オブジェクトと MBinding オブジェクトを使用します。アップグレード ウィザードは、コントロールのデータ連結を追加および削除するために、新しいプロシージャを 2 つ追加します。ただし、Data Environment のコマンドに対する視覚的な管理方法は失われます。データとデータ連結を視覚的に管理するには、ADO コードを ADO.NET に移行する必要があります。

データ アクセス オブジェクトとリモート データ オブジェクト

ここでは、DAO または RDO テクノロジを使ったプロジェクトを Visual Basic .NET にアップグレードする方法について説明します。ただし、アップグレードする方法を定義する前に、これらのテクノロジが Visual Basic 6.0 に使用されている理由について理解することが重要です。

DAO は、Microsoft Access によって使用される Microsoft Jet データベース エンジンを公開した最初のオブジェクト指向インターフェイスでした。DAO を使用することによって、Visual Basic 開発者は Access テーブルやその他のデータベースに、ODBC を使って直接接続することができます。DAO は単一システム アプリケーション、または小規模なローカルの配置に最も適しています。

RDO は ODBC へのオブジェクト指向データ アクセス インターフェイスです。RDO は、DAO の容易に使用できるスタイルを提供し、ODBC の下位レベルの性能と柔軟性をほぼすべて公開するインターフェイスを提供します。RDO の制限として、Jet または ISAM データベースへのアクセスが劣ることと、既存の ODBC ドライバからしかリレーショナル データベースにアクセスできないことが挙げられます。しかし RDO には、SQL Server や Oracle など、多くの有名なリレーショナル データベースの開発者から最もよく採用されているインターフェイスとしての実績があります。RDO は、ストアド プロシージャのより複雑な側面や複雑な結果セットにアクセスするために必要なオブジェクト、プロパティ、およびメソッドを提供します。

多くの場合、DAO と RDO データ アクセス テクノロジを .NET Framework に変換するには、最初にこれらのデータ アクセス テクノロジを ADO に置き換え、その後これらを ADO.NET にアップグレードするのが最適な方法です。この方法によって、DAO と RDO テクノロジをアップグレードする最終的な目標を、複数の中間的な目標に分割することができ、これらの中間的な目標によってさまざまなレベルの機能が段階的に提供されます。

データ連結のアップグレードの注意点

DAO と RDO は以前から存在します。DAO データ連結は Visual Basic 3.0 で導入され、RDO データ連結は Visual Basic 4.0 で導入されました。Visual Basic の開発チームは当初、これらのデータ連結の形式をフォームのパッケージに組み込んで実装しました。この実装によってシームレスな統合が可能となりましたが、データ連結テクノロジと Visual Basic フォームが結合される結果となりました。

Visual Basic .NET では、再設計されたフォーム パッケージである Windows フォームによって Visual Basic フォームが置き換えられました。Windows フォームの設計者達は、DAO と RDO のデータ連結を Windows フォームに組み込まない決断を下したため、DAO と RDO のデータ連結はサポートされていません。

このため、これらのテクノロジを使用するアプリケーションがアップグレードできるのかどうか、既存のアプリケーションに変更を加えずにアップグレードできるのかどうか、あるいは DAO ベースまたは RDO ベースのコードを ADO に変更できるのかどうかについて疑問があるかもしれません。次に、これらの点について説明します。

Visual Basic .NET の DAO と RDO

Visual Basic .NET でも引き続き DAO と RDO を使用することができますが、これらのテクノロジとデータ連結を併用することはできません。アプリケーションを Visual Basic .NET にアップグレードするときに、これらのテクノロジを保持するための作業を考慮すると、アップグレードする前に、Visual Basic 6.0 でこれらのテクノロジを ADO に置き換えたほうがよりスムーズに作業できる場合もあります。そのためには、コード内のデータ連結を再実装する必要があります。しかし、Visual Basic 6.0 ではこれらのテクノロジがそれぞれ非常に似ているため、元のコード ベースには最小限の変更を加えることだけが必要となります。元のコードの DAO と RDO を ADO に置き換えたら、残るアップグレード作業のほぼすべてはアップグレード ウィザードによって自動的に処理されます。

アップグレード処理には、データ コントロールの ADO データ コントロールへの置き換えと、DAO と RDO テクノロジの ADO テクノロジへの置き換えが含まれます。以降では、これらの置き換えの詳細について説明します。

データ コントロールから ADO データ コントロールへの置き換え (Visual Basic 6.0)

組み込みデータ コントロールの実装は、Microsoft Jet データベース エンジンを使ってデータにアクセスします。Microsoft Jet データベース エンジンは Microsoft Access に使用されるデータベース エンジンです。このテクノロジは多くの標準的な形式のデータベースに対するシームレスなアクセスを提供し、コードを書かずに、データを処理するアプリケーションを作成することを可能にします。

このコントロールと DAO テクノロジを併用することによって、フォーム内のコントロールにデータをバインドすることができます。ただし、Visual Basic .NET ではデータ コントロールがサポートされていないため、このコントロールはアップグレード ウィザードによって自動的にアップグレードされません。代わりに、アップグレード ウィザードはデータ コントロールをフォーム上のラベルに置き換えます。アップグレードで問題が生じている箇所を示すために、このラベルの BackColor プロパティは**赤**に設定されます。

この問題は、アップグレード ウィザードを使用する前に対処しておくことが、最善の解決策です。そのためには、最初に Visual Basic 6.0 のプロジェクトに ADO データ コントロールを追加することによって、データ コントロールを ADO データ コントロールに置き換えます。Visual Basic 6.0 のプロジェクトで ADO データ コントロールを使用できるようになったら、コードおよびフォーム デザイン内の各データ コントロールを ADO データ コントロールに置き換えます。

ADO データ コントロールの ConnectionString プロパティには、データ コントロールの Connect および DatabaseName プロパティに指定されている、データベースまたはデータ プロバイダを設定します。たとえ

ば、データコントロールの Connect プロパティに Access が指定され、DatabaseName に C:\Temp\Northwind.mdb が指定されている場合は、次に示す手順に従って新しい ADO データ コントロールの接続文字列を構築します。

▶ ADO コントロールに対応する接続文字列を構築するには

1. ConnectionString プロパティのプロパティページを開き、[ビルド] をクリックします。
2. [プロバイダ] タブの [データ リンク プロパティ] から、Microsoft Jet プロバイダ (3.51 または 4.0) を選択します。選択するバージョンはコンピュータにインストールされている Microsoft Office のバージョンに依存します。たとえば、Microsoft Office 97 がインストールされている場合はバージョン 3.51 を選択します。それ以外の場合はすべてバージョン 4.0 を選択します。適切なプロバイダを選択したら [Next] ボタンをクリックします。
3. [接続] タブで、使用する Microsoft Access データベースまで移動するか、適切なデータベースへのパスを設定します。接続するデータベースを指定したら [データ リンク プロパティ] の [OK] ボタンをクリックし、[プロパティ ページ] の [OK] ボタンをクリックします。

ConnectionString を設定したら、RecordSource に ADO データ コントロールが返す値と型を指定します。これは、RecordSource プロパティの [プロパティ ページ] で設定できます。ここでは 2 つの値を同時に設定できます。図 12.1 に [プロパティ ページ] の例を示します。

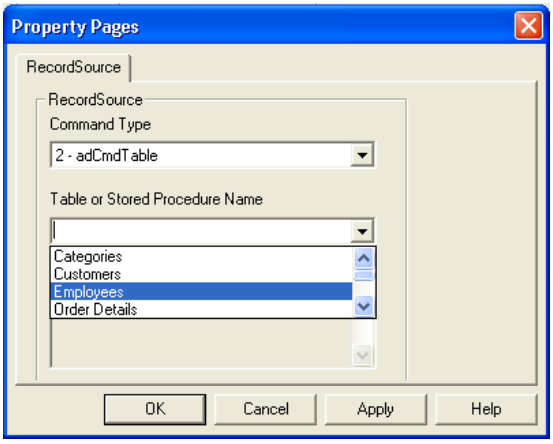

図 12.1

*ADO データコントロールの*RecordSource *プロパティを表す* [*プロパティ ページ*]

この手順が完了したら、フォームから元のデータ コントロールを削除することができます。アップグレード プロセスを開始する前に、この時点でアプリケーションを再構築、再テストし、すべての機能が正常に動作していることを確認してください。

DAO と RDO の ADO への置き換え (Visual Basic 6.0)

ADO は DAO と RDO を継承します。ADO 2.0 は機能的に RDO に似ており、通常これらの 2 つのモデル間には直接マッピングが存在します。ADO は DAO および RDO によって使用されるオブジェクト モデルを "平坦化" します。つまり、ADO に含まれるオブジェクトの数はより少なくなり、より多くのプロパティ、メソッド (および引数)、およびイベントが含まれます。たとえば、ADO には、ODBC ドライバ マネージャと hEnv インターフェイスを公開する、rdoEngine オブジェクトおよび rdoEnvironment オブジェクトと同等の機能がありません。インターフェイスが ODBC OLE DB サービス プロバイダを経由する場合があるにもかかわらず、ADO から ODBC データ ソースを作成することはできません。

DAO モデルおよび RDO モデルに含まれる機能の多くが単一オブジェクトに統合されているため、よりシンプルなオブジェクト モデルが実現されています。ただし、最初は、これによって、DAO と RDO にマップする適切な ADO オブジェクト、コレクション、プロパティ、メソッド、またはイベントを見つけるのに、困難を感じる場合もあります。また、DAO と RDO とは異なり、ADO オブジェクトは ADO のオブジェクト階層のスコープ外に作成することもできます。

ただし、DAO で使用できる機能には、現在の ADO にはサポートされていない機能もあることに注意してください。代わりに ADO は、OLE DB データ ソース、リモート処理、および DHTML テクノロジとやり取りするための RDO スタイルの機能をより密接にサポートします。

現在の ADO はデータ定義言語 (DDL)、ユーザー、または DAO のグループをサポートしないため、DAO は ADO に置き換えないでください。ただし、例外が 2 つあります。ODBCDirect を使用するアプリケーションの場合と、Jet データベース エンジンに依存しない、または DDL を使用しない、クライアント サーバー アプリケーションの場合です。

データ連結を使用しない DAO と RDO のアップグレード

アップグレード プロジェクトで、データ連結を使用しない DAO および RDO コードの扱いは比較的シンプルです。DAO と RDO は COM ライブラリとして実装されているため、コードの多くは Visual Basic .NET でも Visual Basic 6.0 と同じように動作します。たとえば、次に示す DAO コードのサンプルは、Biblio データベースを開き、Select ステートメントを実行し、Authors の一覧を返します。その後、メッセージ ボックス内に Author の名前が表示されます。

```
Dim dbsExample As Database
Dim rstExample As Recordset
Dim fldExample As Field

Set dbsExample = OpenDatabase("c:\temp\Biblio.mdb")
Set rstExample = dbsExample.OpenRecordset("Authors", dbOpenDynaset)
Set fldExample = rstExample.Fields("Author")
MsgBox CStr(fldExample.Value)
dbsExample.Close
```

次に、アップグレード ウィザードによって自動的にアップグレードされたコード例を示します。

```
Dim dbsExample As DAO.Database
Dim rstExample As DAO.Recordset
Dim fldExample As DAO.Field

dbsExample = DAODBEngine_definst.OpenDatabase("c:\temp\Biblio.mdb")
rstExample = dbsExample.OpenRecordset("Authors", DAO.RecordsetTypeEnum.dbOpenDynaset)
fldExample = rstExample.Fields("Author")
MsgBox(CStr(fldExample.Value))
dbsExample.Close()
```

アップグレード ウィザードによってアップグレードされた後のコードは、見かけは Visual Basic 6.0 のコードとほぼ同じですが、より明示的です。また、アップグレード ウィザードは Microsoft.VisualBasic.Compatibility ライブラリへの参照を自動的に追加するため、アップグレード後のコードは Visual Basic .NET でも正常に動作し、手作業で修正を加える必要がありません。

Visual Basic .NET とその他すべてのデータ アクセス テクノロジが大きく異なる点は、レコードセット (または RDO の結果セット) のフィールドにアクセスする方法です。Visual Basic 6.0 では、一般的に、短縮形のコーディング規約 (RecordsetName!FieldName) を使ってフィールドを参照します。Visual Basic .NET では、このコードを展開して、既定のプロパティを解決する必要があります。これはアップグレード ウィザードによって自動的に処理されます。この場合、アップグレード後のコードは多少異なって見えます。以下のような Visual Basic 6.0 のコードがあるとします。

```
Dim rstExample As Recordset
Dim strExample As String
strExample = rstExample!Author
```

このコードでは、rstExample レコードセットの著者の名前を fldExample フィールドに割り当てています。このコードに対してアップグレード ウィザードを実行すると、次に示すコード例に似た結果となります。

```
Dim rstExample As DAO.Recordset
Dim strExample As String
strExample = rstExample.Fields("Author").Value
```

rstExample!Author が rstExample.Fields("Author").Value に展開されている点に注意してください。アップグレード ウィザードはこのような展開を自動的に処理します。

だたし、Visual Basic .NET では RDO と DAO のデータ連結がサポートされていない点に注意してください。このテクノロジがコード内に使用されている場合、アプリケーションのアップグレード後でコードを正常に実行するためのオプションを検討する必要があります。

DAO (データ アクセス オブジェクト) のアップグレード

ここでは、DAO テクノロジを ADO にアップグレードするときの注意点について説明します。

データ連結を使用した DAO

ここで説明するアップグレード シナリオでは、データ コントロールを含む少なくとも 1 つのフォームおよび Microsoft DAO ライブラリへの参照が、アプリケーションのプロジェクトに含まれます。また、フォームのコントロールのうち少なくとも 1 つは、コントロールの DataSource プロパティと DataField プロパティを通じてデータ コントロールにバインドされています。

Visual Basic .NET では DAO データ連結がサポートされていないため、このようなアプリケーションをアップグレードする場合は、アップグレード後のコードを手動で変更する必要があります。実行時に DAO を使用するコードはアップグレード ウィザードによって適切にアップグレードされます。アップグレード ウィザードは DAO ライブラリへのラッパーを使用しますが、プロジェクト内のデータ連結を保持する場合は、データ コントロールに関連するすべてのコードを手動で再コーディングする必要があります。

そのため、プロジェクトを Visual Basic .NET にアップグレードするときは、事前にデータ コントロールすべてを ADO データ コントロールに置き換えるのが最も効率的な方法です。ADO データ コントロールはデータ コントロールに似ていて、Visual Basic .NET でもサポートされています。このコントロールを置き換えると、コードに加える変更を最小限にできます。データ コントロールをアップグレードする方法の詳細については、この章の「RDO リモート データ コントロールから ADO データ コントロールへの置き換え (Visual Basic 6.0)」を参照してください。

データ連結を使用しない DAO

アプリケーションが、プロジェクトに DAO ライブラリへの参照を含む場合でも、データ コントロールを含むフォームがなく、さらに、すべてのデータ アクセスが実行時に直接管理されている場合は、ここで説明するアップグレード シナリオを使用します。

このようなアプリケーションを Visual Basic .NET にアップグレードする場合、アップグレード ウィザードは新しいプロジェクトへのラッパーとして DAO ライブラリを追加します。その結果、DAO データ オブジェクトを使用するコードを手動で変更する必要がなくなり、コードの動作は Visual Basic 6.0 での動作と同じになります。

RDO (リモート データ オブジェクト) のアップグレード

ここでは、RDO テクノロジを ADO にアップグレードするときの注意点について説明します。

データ連結を使用した RDO

ここで説明するアップグレード シナリオは、Microsoft リモート データ オブジェクトを参照するプロジェクトが含まれるアプリケーションに適用します。これらのアプリケーションには、データ連結用のリモート データ オブジェクトを含むフォームが少なくとも 1 つ存在し、かつフォームのコントロールのうち少なくとも 1 つは、DataSoure プロパティと DataField プロパティを使用するリモート データ コントロールによるデータ連結を使用します。

このようなアプリケーションをアップグレードすると、リモート データ コントロールもアップグレードされたかのように見えます。しかし、実行時に新しいコントロールは読み取り専用であり、結果間を移動することはできません。この問題を修正するには、アップグレード後のプロジェクトのリモート データ コントロールを Microsoft.VisualBasic.Compatibility.Data ライブラリ内の ADODC コントロールに変更します。ただし、この方

法には大量の作業が必要になることがあります。そのため、アプリケーションをアップグレードする場合は、事前に Visual Basic 6.0 アプリケーション内の RDO テクノロジを ADO テクノロジに置き換えるとより効率的です。この方法の詳細については、この章の「RDO リモート データ コントロールから ADO データ コントロールへの置き換え (Visual Basic 6.0)」を参照してください。

データ連結を使用しない RDO

ここで説明するアップグレード シナリオは、Microsoft リモート データ オブジェクトを参照するプロジェクトを含むアプリケーションに適用します。また、すべてのデータ アクセスが実行時に管理され、RDO データ コントロールを含まないアプリケーションでも適用対象になる場合があります。

アップグレード ウィザードは RDO ライブラリへの参照を作成するため、このようなアプリケーションでは RDO コードが自動的にアップグレードされます。アプリケーションのコードには以前の RDO 命令が保持されるため、Visual Basic 6.0 と同等の機能が維持されます。

RDO から ADO への置き換え (Visual Basic 6.0)

RDO は ADO に似たマッピングを持つため、アプリケーションをアップグレードする場合は、事前に RDO を ADO に置き換えておくことをお勧めします。以下に示すシナリオでは、RDO を使ってさまざまなデータ アクセスを実装する方法と、同じ動作を ADO を使って実装する方法について説明します。以下の例は、アプリケーションのプロジェクトに Microsoft ADO 2.0 への参照が含まれていることを前提としています。

データベースへの接続の確立

RDO で接続を開くには接続文字列とパラメータを定義する必要があります。RDO で rdoQuery を作成するときに接続は必要ありませんが、rdoResultset オブジェクトを最初に作成するときには接続が必要になります。以下のコード例は、RDO を使って接続を開くコードです。

```
Dim WithEvents cn As rdoConnection
Dim cnB As New rdoConnection
Const ConnectString = "uid=myname;pwd=mypw;driver={SQL Server};" & _
    "server=myserver;database=pubs;dsn=''"
```

この接続文字列は、特定の SQL Server にアクセスします。また、データ ソース名 (DSN) を使用せずに接続を開くことを ODBC に許可します。この例は、標準的な引数をすべて使用する、典型的な ODBC 接続文字列の例です。

次に、フォーム Load イベント コードを使って、この例をさらに検証します。このコードでは、カーソル ドライバの種類とログインのタイムアウトが定義されています。RDO の既定のカーソルの種類は、SQL Server 上でサーバー側のカーソルを起動する rdUseIfNeeded です。ただし、次の例では、rdUseNone を指定することによってこの既定値をオーバーライドしています。rdDriverNoPrompt フラグは、ユーザー ID およびパスワードが一致しなかった場合にアプリケーションがエラーを生成することを示します。

```
Private Sub Form_Load()
    Set cn = New rdoConnection
    With cn
        .Connect = ConnectString
        .LoginTimeout = 10
        .CursorDriver = rdUseNone
        .EstablishConnection rdDriverNoPrompt
    End With
```

次の例が示すように、Load イベント内の 2 つ目の接続はクライアントバッチ更新を実行します。

```
    With cnB
        .Connect = ConnectString
        .CursorDriver = rdUseClientBatch
        .EstablishConnection
    End With
End Sub
```

接続操作が完了すると最後のイベントが実行されます。このイベントは接続が開かれたときに発生するエラーを処理します。このイベントによって、処理を実行する前に接続が確立されたかどうか検証することができます。たとえば、接続が開いている場合にだけ使用できるボタンなどを有効にすることができます。これを以下のコード例に示します。

```
Private Sub cn_Connect(ByVal ErrorOccurred As Boolean)
    If ErrorOccurred Then
        MsgBox "Could not open connection", vbCritical
    Else
        RunOKFrame.Enabled = True
    End If
End Sub
```

ただし、ADO を使ってデータベースへの接続を確立するには、まず ADODB オブジェクトから参照される ADO オブジェクト セットを作成する必要があります。これらのオブジェクトは、接続を開き、結果セットを生成するための特定なプロパティを設定するために、後で使用されます。次のコードに例を示します。

```
Dim cn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim cnB As New ADODB.Connection
Dim Qy As New ADODB.Command
```

次の行では、前の RDO の例で作成した接続文字列に似た接続文字列を作成します。これらの例では共に、時間を節約し、パフォーマンスを向上するために、DSN を使用しない形式の ODBC 接続が使用されています。

```
Const ConnectString= "uid=myname;pwd=mypw;driver={SQL Server};" & _
    "server=myserver;database=pubs;dsn=''"
```

変数と接続文字列を作成したら、フォームの Load イベント内でデータベースへの ADO 接続を開くことができます。その方法を以下に示します。

```
Private Sub Form_Load()
    With cn
        ' DNS を使用しない接続を確立します。
        .ConnectionString = ConnectString
        .ConnectionTimeout = 10
        .Properties("Prompt") = adPromptNever
        ' これは、ADO の既定の確認モードです。
        .Open
    End With
    With cnB
        .ConnectionString = ConnectString
        .CursorLocation = adUseClient
        .Open
    End With
End Sub
```

定数の先頭に rd ではなく ad が付くこと以外は、変換後のコードと RDO のコードは似ています。たとえば、rdDriverNoPrompt は adPromptNever に変換されています。ADO では無確認が既定となっているため、確認時の動作を指定する必要はありません。この動作を変更する場合は、ADO プロパティ コレクションを使って確認動作を定義してください。RDO では OpenConnection 引数を使って動作を設定することができます。ADO では Properties ("Prompt") プロパティを設定します。また、ADO では、既定で、カーソル ドライバなしが既定になるため、RDO での CursorDriver = rdUseNone のような、カーソル ドライバを使用しない場合のその指定の必要はありません。

基本的なクエリの実行

RDO では、OpenResultset メソッドを使ってクエリを実行することができます。このメソッドを使用すると Resultset が返され、すべての結果にアクセスすることができます。次の例では、SQL ステートメントに基づく Resultset を返す方法を示します。Resultset を作成するには接続が開いている必要があります。

```
…
Private Sub RunButton_Click()
    Dim rs As rdoResultset
    Set rs = cn.OpenResultset("select * from titles where title like '%h'")

    ' ここでクエリから取得した Resultset に対して操作を実行します。

    rs.Close
End Sub
…
```

ADO でクエリを実行するには、データベースへの接続を使って、レコードセットの Open メソッドを使用する必要があります。次に示すイベントプロシージャは、前の RDO のコード例によく似ています。ただし次の例では、rdoConnection オブジェクトの OpenResultset メソッドを使用する代わりに、SQL クエリと ADO 接続オブジェクトを引数に取る、ADO の Open メソッドを新しく使用します。また、行セットを返さないものであれば、RDO でと同様に ADO 接続オブジェクトの Execute メソッドを使用することもできます。

```
Private Sub RunButton_Click()
   Dim rs As New ADODB.Recordset
   rs.Open "select * from titles where title like '%h'", cn
   Set MyMSHFlexGrid.Recordset = rs
   rs.Close
End Sub
```

ADO では、このクエリの実行とレコードセットの処理を非同期で実行することができます。rs.Open に adFetchAsynch オプションを指定すると、ADO ではカーソルプロバイダがバックグラウンドで自動的にレコードセットを作成します。

MSHFlexGrid コントロールでの結果セットの表示

次に示すコード例では、カスタム ActiveX コントロールの ShowData メソッドを使って、結果セットからのデータを RDO テクノロジを使った MSHFlexGrid コントロールに表示します。このコードは rdoColumns プロパティ内の名前に基づいてグリッドを構築し、グリッドを初期化して、データ表示ができるように準備をします。結果セットの rdoColumns プロパティをインデックスするために OrdinalPosition プロパティが使用されている点にも注目してください。また、MSHFlexGrid コントロールに行を追加するのに、rdoResultset オブジェクトの GetClipString メソッドが使用されています。

```
...
Public Function ShowData(Resultset As rdoResultset) As Variant
    Dim cl As rdoColumn
    Static GridSetup As Boolean
    Dim MaxL As Integer
    Dim rsl As rdoResultset
    Dim Rows As Variant
    On Error GoTo ShowDataEH
    Set rsl = Resultset
    If GridSetup = False Then
        FGrid1.Rows = 51
        FGrid1.Cols = rsl.rdoColumns.Count
        FGrid1.Row = 0
        For Each cl In rsl.rdoColumns
            FGrid1.Col = cl.OrdinalPosition - 1
            FGrid1 = cl.Name
            If rsl.rdoColumns(cl.OrdinalPosition - 1).ChunkRequired Then
                MaxL = 1
            Else
                MaxL = rsl.rdoColumns(cl.OrdinalPosition - 1).Size + 4
            End If
```

```
            If MaxL > 20 Then MaxL = 20
            FGrid1.ColWidth(FGrid1.Col) = TextWidth(String(MaxL, "n"))
        Next cl
        GridSetup = True
    End If
    FGrid1.Rows = 1     'Clear Grid of data (except titles)
    FGrid1.Rows = 51
    FGrid1.Row = 1
    FGrid1.Col = 0
    FGrid1.RowSel = FGrid1.Rows - 1
    FGrid1.ColSel = FGrid1.Cols - 1
    FGrid1.Clip = rsl.GetClipString(50, , , "-")

ExitShowData:
    FGrid1.RowSel = 1
    FGrid1.ColSel = 0
    Exit Function

ShowDataEH:
    Select Case Err
        Case 40022:
            FGrid1.Clear
            Resume ExitShowData
        Case 13
            FGrid1.Text = "< >"
            Resume Next
        Case Else
            MsgBox "Could not display data: " & Err & vbCrLf & Error$
            Resume ' ExitShowData
    End Select
End Function
…
```

次に示すコードは、RDO コントロールに基づくカスタム ActiveX コントロールの ShowData メソッドを実装する、比較用のコード例ですが、このコードには ADO が使用されています。ADO では RDO の GetClipString メソッドが GetString メソッドによって置き換えられています。その結果として生成される Variant 配列を後で解析する必要があるため、このルーチンのパフォーマンスは著しく低下します。

また、RDO とは異なり、列のタイトルを取得するときに、Fields コレクションのインデックスとして OrdinalPosition を使用することができません。この問題を解決するには、操作する列を指定するときに、代わりに新しい整数カウンタを使います。列に収まらない TEXT データ型フィールドと IMAGE データ型フィールドを検索するには、DefinedSize プロパティと ActualSize プロパティを使用します。これらの新しいプロパティを使用することによって、特定のフィールド値の詳細をより容易に特定することができます。

次の例には、参照するデータ列に BLOB 型データが含まれていた場合に、そのデータに対応するためのコードが追加されています。

```
Public Function ShowData(Resultset As Recordset) As Variant
    Dim cl As Field
```

```
    Static GridSetup As Boolean
    Dim MaxL As Integer
    Dim Op As Integer
    Dim rsl As Recordset
    Dim rows As Variant
    On Error GoTo ShowDataEH
    Set rsl = Resultset

    If GridSetup = False Then
        FGrid1.rows = 51
        FGrid1.Cols = rsl.Fields.Count
        FGrid1.Row = 0
        Op = 0
        For Each cl In rsl.Fields
            FGrid1.Col = Op
            FGrid1 = cl.Name
            If rsl.Fields(Op).DefinedSize > 255 Then
                MaxL = 1
            Else
                MaxL = rsl.Fields(Op).ActualSize + 4
            End If
            If MaxL > 20 Then MaxL = 20
            FGrid1.ColWidth(FGrid1.Col) = TextWidth(String(MaxL, "n"))
            Op = Op + 1
        Next cl
        GridSetup = True
    End If
    FGrid1.rows = 1
    FGrid1.rows = 51
    FGrid1.Row = 1
    FGrid1.Col = 0
    FGrid1.RowSel = FGrid1.rows - 1
    FGrid1.ColSel = FGrid1.Cols - 1
    With FGrid1
        ' また、ここでは次の記述の代わりに ADO2 の GetString メソッドを
        ' 使用することもできます。
        FGrid1.Clip = rsl.GetString(adClipString, 50, , , "-")
    End With

ExitShowData:
    FGrid1.RowSel = 1
    FGrid1.ColSel = 0
    Exit Function

ShowDataEH:
    Select Case Err
        Case 3021:
            FGrid1.Clear
            Resume ExitShowData
        Case 13, Is < 0
            rows(j, i) = "< >"
            Resume 'Next
        Case Else
            MsgBox "Could not display data: " & Err & vbCrLf & Error$
```

```
            Resume ' ExitShowData
    End Select
End Function
```

RDO 2.0 を ADO 2.0 に変換する方法の詳細については、MSDN の「Visual Basic Concepts: Converting from RDO 2.0 to ADO 2.0」を参照してください。

RDO リモート データ コントロールから ADO データ コントロールへの置き換え (Visual Basic 6.0)

リモート データ コントロール テクノロジを含むアプリケーションをアップグレードすると、アップグレード ウィザードによってコントロールがアップグレードされたかのように見えます。しかし、新しいコントロールは、実行時に読み取り専用となります。この場合、結果間を移動することができません。そのため、アプリケーションをアップグレードする場合は、事前に Visual Basic 6.0 アプリケーション内の RDO テクノロジを ADO テクノロジに置き換えることをお勧めします。

以下の例は、リモート データ コントロールをアップグレードするときに使用できる方法を示します。この例には RDC1 という名前を持つリモート データ コントロールと、OrderId という名前を持つ TextBox コントロールが含まれます。

この例の RDC1 は Orders テーブルに格納されている注文情報をすべて取得します。Orders テーブルは Access という名前の DSN を使用する Northwind Microsoft Access データベースの一部です。コントロールのプロパティに設定された値の一覧を表 12.1 に示します。

表 12.1: アップグレード例のリモート データ コントロールのプロパティ

RemoteData コントロールのプロパティ	値
Connect	DSN=Access
CursorDriver	0-rdUseIfNeeded
DataSourceName	MS Access データベース
Password	<空白>
SQL	Select * from Orders
UserName	Admin

TextBox コントロール内では DataSource プロパティに RDC1 が指定され、DataField プロパティには OrderID が指定されています。

ADO データ コントロールを使用するには、次の手順に従います。

▶ ADO データ コントロールの使用

1. 「データ コントロールから ADO データ コントロールへの置き換え (Visual Basic 6.0)」の説明に従って、プロジェクトに ADO への参照を追加します。
2. フォームに ADO データ コントロールを追加し、適切な名前を指定します。この例では ADORDC1 です。

3. ADO データ コントロールの ConnectionString プロパティに、RemoteData コントロール内で使用される Access DSN を指定します。[プロパティ ページ] を使ってこれらの値を設定する方法を図 12.2 に示します。

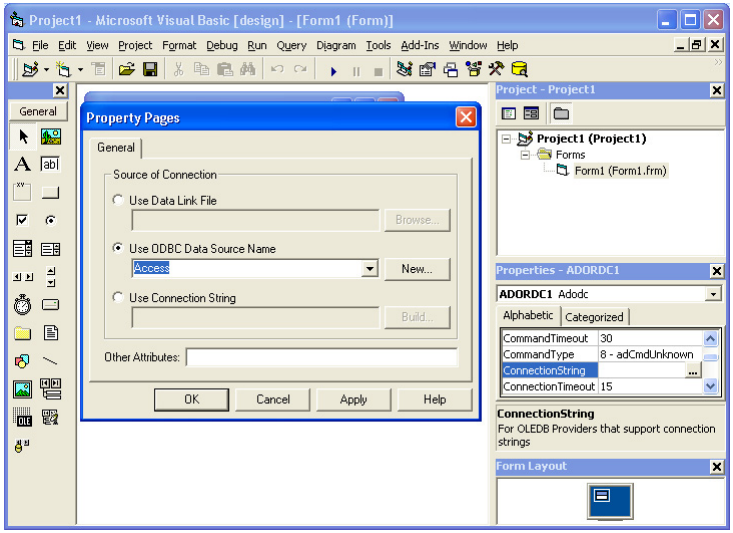

図 12.2

[プロパティ ページ] から ADO データ コントロールの ConnectionString を設定する

4. ADO データ コントロールの UserName プロパティに Admin を指定します。
5. ADO データ コントロールの RecordSource プロパティにクエリ "select * from Orders" を指定します。
6. RemoteData コントロールを削除します。
7. TextBox コントロールの DataSource プロパティの値を ADORDC1 に変更します。

これらの手順が完了したら、アップグレード プロセスを開始する前に、アプリケーションを再構築、再テストし、すべての機能が正常に動作していることを確認してください。

次に、DAO テクノロジと RDO テクノロジすべてを ADO にアップグレードする方法について説明します。

カスタム データ アクセス コンポーネント

カスタム データ アクセス コンポーネントとは、データ アクセス テクノロジをカプセル化するコンポーネントです。カスタム データ アクセス コンポーネントがカプセル化するデータ アクセス テクノロジには、ADO、DAO、DLL 内の RDO、アプリケーションが使用する OCX 内の RDO などがあります。これらのコンポーネントとしては、特定の言語で内部的に開発されたコンポーネント、または他企業から購入した、必ずしもソース コードが得られるとは限らないコンポーネントが考えられます。カスタム データ アクセス コンポーネントを使用するアプリケーションのアップグレード方法は複数あります。その中から採択される方法は、コンポーネントのソース コードを参照できるかどうかと、コンポーネントの開発に使用されている言語によって異なります。

アップグレードするアプリケーションは、次に挙げる3つのシナリオのどれか1つに該当する場合があります。

- カスタム コンポーネントのソース コードがない。
- カスタム コンポーネントのソース コードがあり、開発言語は Visual Basic 6.0 である。
- カスタム コンポーネントのソース コードはあるが、開発言語が Visual Basic 6.0 ではない。

これらの場合には、次の 2 つのアップグレード方法のどちらかを使用することができます。

- アプリケーションをアップグレードし、コンポーネントにはラッパーを使用する。
- アプリケーションとコンポーネントを個別にアップグレードし、古いコンポーネントを Visual Basic .NET ベースの新しいコンポーネントに置き換える。

次に、各アップグレード方法について説明します。

コンポーネントの .NET バージョンへのアップグレード

アプリケーションに使用されているカスタム データ アクセス コンポーネントのソース コードを参照することができ、かつそのコンポーネントの開発言語が Visual Basic 6.0 である場合、これらのコンポーネントの多くは Visual Basic .NET にアップグレードすることができます。通常、この作業はアプリケーションの残りの部分のアップグレードとは切り分けることができます。この方法は段階的アップグレードと呼ばれ、第 2 章「アップグレードの成功のためのプラクティス」に説明されています。ただし、これらのコンポーネントのアップグレード方法を決定する要素は、コンポーネントを構築するのに使用されているデータ アクセス テクノロジに大きく依存します。

既に説明したように、DAO/RDO を使うコードをアップグレードするときは、最初に Visual Basic 6.0 で書かれたコード内にある DAO/RDO を ADO に置き換えるのが最も容易な方法です。ただし、コンポーネントが直接カスタム データベース ファイルにアクセスする場合は、データ アクセスを Microsoft SQL Server などのデータベース サーバーへの接続に置き換えたほうが効果的な場合もあります。これらのシナリオのいずれにも、アップグレードを実行する前にコードを準備する手間がかかる欠点があります。また、この準備の結果としてコンポーネントの動作が変わる可能性もあります。しかし、動作の問題点を解決するときには新しいデータ アクセス機能を追加する機会も得られるので、良い結果を導く場合もあると言えます。また、データ アクセス コンポーネントを Visual Basic .NET にアップグレードすることによって、最新のデータ アクセス テクノロジである ADO.NET が使用できる機会にもなります。ADO.NET は Visual Basic .NET に完全に統合されており、データベースへの容易なアクセスを提供します。

アップグレードするカスタム データ アクセス コンポーネントをサード パーティから購入した場合、新しいバージョンのコンポーネントがリリースされている可能性があります。この場合、その新しいバージョンは Visual Basic .NET によって開発されたものである可能性もあります。新しいバージョンのコンポーネントが Visual Basic .NET によって開発されたものである場合、アプリケーションのアップグレードに必要な作業の量はかなり削減できる可能性があります。

COM 相互運用機能の利用とカスタム データ アクセス コンポーネント

アプリケーションに使用されているカスタム データ アクセス コンポーネントのソース コードがない場合、または Visual Basic 6.0 以外の言語によってコードが開発されている場合は、アップグレード ウィザードを使ってアプリケーションをアップグレードする方法をお勧めします。アップグレード ウィザードはデータ アクセス コンポーネントへのラッパーを作成します。このラッパーによって既存のステートメントを使用することが可能になり、COM 相互運用機能を利用して、Visual Basic .NET でも Visual Basic 6.0 と同じ機能を維持することができます。

この方法には、アップグレード後のコードに対し、データ アクセス コンポーネントのために必要となる変更が最小限にとどめられる利点があります。また、アプリケーションには Visual Basic 6.0 のときと同じ機能が維持されます。この方法には、コンポーネントに引き続き古いテクノロジが使用されてしまう欠点と、COM 相互運用機能を使用することによって発生するオーバーヘッドがアプリケーションのパフォーマンスを低下させる欠点があります。COM 相互運用機能の詳細については、第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」を参照してください。

カスタム コンポーネントのソース コードを参照することができ、かつカスタム コンポーネントが Visual Basic 6.0 によって開発されている場合にもこの方法を使用することができますが、そのシナリオの場合にコンポーネントを Visual Basic .NET にアップグレードする方法については、前の「コンポーネントの .NET バージョンへのアップグレード」を参照してください。

混合データ アクセス テクノロジのアップグレード

ADO、RDO、DAO、カスタム データ アクセスなど、アプリケーションに異なるテクノロジが使用されている場合、アップグレードを開始する前に、すべてのデータ アクセスを 1 つのテクノロジに統合する必要があります。この場合、ADO を使用すると、アップグレード ウィザードを使って自動的にアップグレードすることができるため、最も効果的です。アップグレードを自動化することによって、アップグレード後のコードに手動で加える変更の数を減らすことができます。

別の方法として、各テクノロジを個別にアップグレードすることもできます。そのためには、アプリケーションが使用する各テクノロジ用に Visual Basic 6.0 のプロジェクトを新しく作成する必要があります。各プロジェクトは個別にアップグレードする必要があります。すべてのプロジェクトをアップグレードした後で、Visual Basic .NET ですべてのプロジェクトを 1 つのアプリケーションに統合する必要があります。

データ レポートから Crystal Reports への変換

Visual Basic 6.0 は Microsoft データレポート デザイナを提供します。Microsoft データレポート デザイナは、階層付きバンド レポートを作成する機能を備えた多機能なデータ レポート ジェネレータです。Data Environment デザイナなどのデータ ソースと組み合わせて使用することによって、異なる複数のリレーショナル テーブルからレポートを作成することができます。印刷用レポートの作成に加え、レポートを HTML ファイルまたはテキスト ファイルにエクスポートすることもできます。

Visual Basic .NET の標準的なレポート ツールは Crystal Reports です。このツールを使用することによって、アプリケーションの用途に合わせた強力なレポートを作成することができます。Crystal Reports を使用することには多くのメリットがあります。

Crystal Reports を使用するメリットの 1 つとして、Microsoft データ レポート (.dsr ファイル) を Crystal レポート (.rpt ファイル) に変換するときに、比較的容易な自動処理を使用できることが挙げられます。データ レポートを変換すると、同等の Crystal レポート が生成されます。変換後のレポートは元のレポートと同じデザインとデータ アクセスを持ちます。データベースへの接続とレポートの全般的な構造は、似たモデルに基づいて保持されます。

ただし、Crystal Reports ツールは、データを取得するためにパラメータを使用するクエリを含むデータ レポートを変換することはできません。たとえば、次のクエリを含むデータ レポートは自動変換できません。

```
SELECT Products.* FROM Products WHERE (ProductID = ?)
```

その場合、Crystal レポート形式のレポートを手動で再実装することをお勧めします。Crystal レポートを作成する方法の詳細については以下を参照してください。

- MSDN の「Crystal Reports for Visual Studio .NET」の「Reports in Windows Application」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/crystlmn/html/crconreportsinwindowsapplications.asp)
- MSDN Magazine の「Add Professional Quality Reports to Your Application with Visual Studio .NET」(http://msdn.microsoft.com/msdnmag/issues/02/05/Crystal/)

自動変換に Crystal Reports を使用できる場合は、次に示す複数のステップに従って処理を実行します。

1. 最初に、関連する Visual Basic 6.0 プロジェクト、フォーム、およびデータ アクセス コンポーネントすべてを Visual Basic .NET にアップグレードします。
2. 次に、データ レポートを Crystal Reports にアップグレードします。
3. 最後に、アップグレード後のコードと変換後の Crystal レポートが連携できるように、プロジェクトに変更を加えます。

データ レポートをアップグレードする方法を次の例に示します。この例では、Visual Basic 6.0 プロジェクトに ProductReport という名前を持つレポートが含まれます。このレポートは、Microsoft Office に付随するサンプル データベースである Northwind データベース内の製品に関連する情報を表示するのに使用します。データベースへの接続は DataBase という名前を持つ DataEnvironment オブジェクトによって実行され、フォームは製品を表示するのに使用されます。フォームには 2 つのボタンが含まれ、ユーザーはレポートを表示 (btnShow)、またはエクスポート (btnFile) することができます。

[Show Report] ボタンをクリックすると画面にレポートが表示され、[Export Report] ボタンをクリックするとレポートがテキスト ファイルにエクスポートされます。次に、各イベントのコードを示します。

```
Private Sub btnFile_Click()
```

```
    ProductReport.ExportReport rptKeyText, "c:\temp\Reports\Products.doc"
End Sub

Private Sub btnShow_Click()
    ProductReport.Show
End Sub
```

アップグレード ウィザードを使ってプロジェクトをアップグレードした場合、DataReport はアップグレードされません。代わりに新しいプロジェクト内にそのままコピーされます。フォーム内のコードと DataEnvironment 内のコードがアップグレードされ、この時点で一般的なアップグレード プロジェクトと同じ量の作業が発生します。

コード ベースをアップグレードしたら、次の段階では DataReport を Crystal レポートに変換します。DataReport を変換するには次の手順に従います。

▶ DataReport を Crystal レポートに変換する

1. Visual Studio IDE の **[ソリューション エクスプローラ]** ペインの **[新しい項目の追加]** をクリックし、アップグレード後のプロジェクトに Crystal レポートを追加します。新しいレポートを追加すると、[Crystal Report Gallery] ダイアログ ボックス (図 12.3) が表示されます。

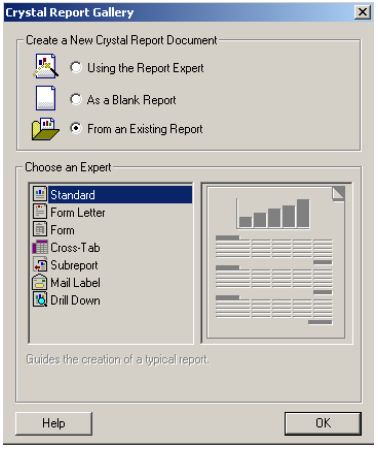

図 12.3
[Crystal Report Gallery] *ダイアログ ボックス*

2. [Crystal Report Gallery] ダイアログ ボックスの [From an Existing Report] をクリックします。
3. [Search] ダイアログ ボックスで DataReport ファイルまで移動し、選択します。DataReport ファイルは Visual Basic 6.0 アプリケーションのソース コードの一部です。適切なファイルを選択して [OK] をクリックします。

これらの手順を完了すると、Crystal Report Engine は自動的に DataReport オブジェクトをアップグレードします。アップグレード処理が完了したら、アップグレード ウィザードによってプロジェクトにコピーされた DataReport を削除することができます。

DataReport を Crystal レポートに変換する最後の手順では、新しく作成された Crystal レポートと連携できるように、アップグレード後のプロジェクトに変更を加えます。そのために、プロジェクトに次の参照を追加します。

- CrystalDecisions.Shared
- CrystalDecisions.ReportSource
- CrystalDecisions.CrystalReports.Engine
- CrystalDecisions.Windows.Forms

[Show Report] ボタンの動作を複製するには、プロジェクトに新しいフォームを追加する必要があります。この例で追加するフォームの名前は DataReport です。このフォームには Crystal Report Viewer を追加する必要があります。Crystal Report Viewer の Dock プロパティに Fill を指定し、Modifiers プロパティには Public を指定します。また、名前として Viewer を指定することもできます。このようにフォームとビューアを追加した後、レポートが新しいフォーム内に表示されるようにボタンのイベント ハンドラ コードを調整します。元のコードに対してアップグレード ウィザードを実行すると、[Show Report] ボタンのコードで次の警告が表示されます。

```
Dim ProductReport As Object
' UPGRADE_WARNING: オブジェクト ProductReport.Show の既定プロパティを解決できませんでした。
ProductReport.Show()
```

アップグレード時のこの問題を解決するには、新しいレポートとフォーム用の変数を宣言する必要があります。その後、Crystal Report Viewer の ReportSource プロパティにレポートを割り当てる必要があります。次に修正後のコードの最終版を示します。

```
Dim ProductReport As New ProductReport
Dim report As New DataReport
report.Viewer.ReportSource = ProductReport
report.Show()
```

これらの変更を加えると、[Show Report] ボタンをクリックしたときの Crystal レポートの動作が、元の Visual Basic 6.0 のレポートの動作と同じになります。

[ファイルのエクスポート] ボタンの動作を複製する場合にも、アップグレード後のコードを手動で調整する必要があります。この場合、Crystal レポートの Export 関数を使用する必要があります。元の Visual Basic 6.0 の [ファイルのエクスポート] ボタンのイベント ハンドラに対してアップグレード ウィザードを実行すると、結果は次のコード例のようになります。

```
Dim ProductReport As Object
' UPGRADE_WARNING: オブジェクトの既定プロパティを解決できませんでした。
ProductReport.ExportReport.
ProductReport.ExportReport(MSDataReportLib.ExportKeyConstants.rptKeyText,
"c:\temp\Reports\Products.doc")
```

アップグレード時のこの問題を解決するには、Crystal レポート用の変数を宣言する必要があります。レポートをテキスト ファイルにエクスポートするには、次のコードに示すように、イベント ハンドラ内の ExportReport 関数の代わりにCrystal レポートの Export メソッドを呼び出す必要があります。

```
Dim ProductReport As New ProductReport
ProductReport.ExportToDisk(ExportFormatType.RichText, _
    "c:\temp\Reports\Products.doc")
```

まとめ

なんらかのデータ アクセスを使用する Visual Basic 6.0 アプリケーションについて、多くの場合、アップグレードが必要になります。この章が提供する方法を使用することによって、データ アクセス コードをアップグレードすることができます。

詳細情報

Crystal レポートを作成する方法の詳細については以下を参照してください。

- MSDN の「Crystal Reports for Visual Studio .NET」の「Reports in Windows Application」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/crystlmn/html/crconreportsinwindowsapplications.asp)
- MSDN Magazine の「Add Professional Quality Reports to Your Application with Visual Studio .NET」(http://msdn.microsoft.com/msdnmag/issues/02/05/Crystal/)

ArtinSoft Visual Basic .NET Ready プログラムの詳細については、ArtinSoft の Web サイト (http://www.artinsoft.com/iproducts/vb6todotnet/laboratories.asp) を参照してください。

RDO 2.0 を ADO 2.0 に変換する方法の詳細については、MSDN の「Converting from RDO 2.0 to ADO 2.0」(http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon98/html/vbconconvertingfromrdotoado.asp) を参照してください。

13

Windows API の使用

Windows API は、すべての Windows アプリケーションの構築基盤です。Visual Basic アプリケーションも例外ではありません。Visual Basic は、言語とフォーム パッケージを通じて、Windows API を一連の使いやすいステートメント、コンポーネント、およびコントロールに抽象化します。多くの場合、Visual Basic アプリケーションは Windows API 関数を直接呼び出さずに構築することができます。ただし、Visual Basic の言語とフォーム パッケージが利用可能な Windows API 関数をすべて表しているわけではないため、Visual Basic では Windows API 関数の直接呼び出しをサポートしています。これはバージョン 1.0 から変わりません。これを利用し、Visual Basic ランタイムでは実装できない機能をアプリケーションに追加することができます。

Visual Basic .NET では、Windows API 関数を以前と同じ方法で宣言して呼び出せるようにすることで、この機能をさらに推し進めています。さらに、これらの API の多くへのアクセスが、より包括的な .NET Framework クラス ライブラリを通じて公開されました。これらの API 関数呼び出しをアップグレードする方法は 2 つあります。

1 つ目の方法は、Windows API を引き続き使用することです。この方法では、相互運用のテクニックを使用して API にアクセスする必要があります。この方法では、API 関数呼び出しをアップグレードするための労力は最小限で済みますが、アンマネージ コードへの依存が生じます (マネージ コードの詳細については、第 1 章「はじめに」の「生産性の向上」を参照してください)。

2 つ目の方法は、Windows API の呼び出しを、Visual Basic .NET の代替機能で置き換えることです。この方法では、API 関数呼び出しをアップグレードするための作業は余分に必要ですが、アンマネージ ライブラリに依存しない、完全なマネージ コードができあがります。

この章では、Windows API を使用するためのこれら 2 つのアプローチについて説明します。ここでは、Declare ステートメントの作成方法に影響を与える言語の変更点に注目し、Windows API 関数を引き続き使用できるようにします。また、.NET Framework に含まれているクラスとメソッドを使用し、アプリケーションで実行する Windows API 呼び出しを補足または置換する方法も示します。

データ型の変更

Visual Basic .NET には、複数の Windows API 関数に影響を与えるデータ型の変更が含まれています。これらの変更には、Integer データ型および Long データ型に関連するものと、固定長文字列に関連するものがあります。Visual Basic .NET にアップグレードするときは、Windows API 関数が適切に呼び出されるように注意する必要があります。

Integer データ型と Long データ型の変更

.NET Framework との一貫性を維持するため、Visual Basic 6.0 と Visual Basic .NET の間で、Integer データ型と Long データ型の領域サイズが変更されました。これらの変更は、数値型の Integer および Long を含む Windows API 関数宣言のほとんどに影響を与えます。Visual Basic 6.0 では、Integer 型は 16 ビットで、Long 型は 32 ビットです。Visual Basic .NET では、Integer 型は 32 ビットで、Long 型は 64 ビットです。Visual Basic .NET では、Short という新しいデータ型が加わります。このデータ型は、サイズ的には Visual Basic 6.0 の Integer 型を置き換えるものです。Visual Basic .NET では、Windows API 関数の新しい Declare ステートメントを作成するときに、この違いに注意する必要があります。これまで Long だったパラメータ型またはユーザー定義型のメンバは、Integer として宣言する必要があります。これまで Integer 型として宣言していたメンバは、Short 型として宣言する必要があります。

例として、以下に示す Visual Basic 6.0 の Declare ステートメントを見てください。このステートメントには、Integer 型のパラメータと Long 型のパラメータが両方含まれています。

```
Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) _
    As Integer
```

これに対応する Visual Basic .NET の宣言を以下に示します。

```
Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Integer) _
    As Short
```

Visual Basic アップグレード ウィザードは、コード内のすべての変数宣言を、適切なサイズを使用するように自動的に変更します (コード内の変更は**太字**で示されます)。データ型のサイズを考慮する必要があるのは、新しい Declare ステートメントを作成しているとき、または既存のステートメントを変更しているときだけです。

固定長文字列の変更

Visual Basic 6.0 には、Visual Basic .NET でサポートされていない固定長文字列データ型があります。コード内の固定長文字列宣言は、Visual Basic .NET で用意されている固定長文字列ラッパー クラスにアップグレードする必要があります。以下のような Visual Basic 6.0 の関数宣言があるとします。

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Long) As Long
Function GetUser()
    Dim Ret As Long
    Dim UserName As String
    Dim Buffer As String * 25
    Ret = GetUserName(Buffer, 25)
    UserName = Left$(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox (UserName)
End Function
```

アップグレード ウィザードは、このコードを以下のようにアップグレードします。

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA"(ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
Function GetUser() As Object
    Dim Ret As Integer
    Dim UserName As String
    Dim Buffer As New VB6.FixedLengthString(25)
    Ret = GetUserName(Buffer.Value, 25)
    UserName = Left(Buffer.Value, InStr(Buffer.Value, Chr(0)) - 1)
    MsgBox(UserName)
End Function
```

このコードは、元の Visual Basic 6.0 コードと同じように動作しますが、Visual Basic 6.0 互換性ランタイムに依存します。もう 1 つのアプローチは、この Visual Basic 6.0 互換性ランタイムへの依存を取り除くというものです。それには、アップグレード前にコードに対して準備作業を行うか、またはアップグレード ウィザードによって生成された Visual Basic .NET コードを変更します。

アップグレード前に Visual Basic 6.0 コードに対して準備作業を行うには、以下に示すように、固定長文字列ではなく、長さが 25 に明示的に設定された通常の文字列を使用すると、コードを書き換えることができます。

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Long) As Long
Function GetUser()
    Dim Ret As Long
    Dim UserName As String
    Dim Buffer As String
    Buffer = String$(25, " ")
    Ret = GetUserName(Buffer, 25)
    UserName = Left$(Buffer, InStr(Buffer, Chr(0)) - 1)
    MsgBox (UserName)
End Function
```

または、アップグレード後の Visual Basic .NET コードを以下のように変更することにより、Visual Basic 6.0 互換性ランタイムへの依存を取り除くこともできます。

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias _
"GetUserNameA"(ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
Function GetUser() As Object
   Dim Ret As Integer
   Dim UserName As String
   Dim Buffer As New String(" ", 25)
   Ret = GetUserName(Buffer, 25)
   UserName = Left(Buffer, InStr(Buffer, Chr(0)) - 1)
   MsgBox(UserName)
End Function
```

どちらの方法でも、Visual Basic 6.0 互換性ランタイムへのコードの依存は軽減されます。

変数型 "As Any" のサポート廃止

Visual Basic 6.0 では、As Any 変数型を使用して、パラメータ型を宣言することができます。この宣言を使用すると、任意のデータ型の引数を渡し、呼び出しの実行時に Visual Basic 6.0 が適切な情報を渡すようにすることができます。これにより柔軟性は向上しますが、API 関数を呼び出すときに引数の型チェックが実行されなくなります。したがって、互換性のない引数型を渡すと、アプリケーションが実行時例外を発生させる可能性があります。

Visual Basic .NET では、Windows API のパラメータを As Any 変数型で宣言することをサポートしていません。ただし、同じ API 関数を複数回宣言し、それぞれの宣言で同じパラメータに異なるデータ型を使用することで、似たような効果を得ることができます。

Windows API 関数の SendMessage は、Visual Basic 6.0 コードで API 関数を宣言するときに As Any 型を使用する例の 1 つです。送信している Windows メッセージによって、必要なパラメータ型は異なります。たとえば、WM_SETTEXT メッセージでは、最後のパラメータとして文字列を渡す必要があります。これに対し、WM_GETTEXTLENGTH メッセージでは、最後のパラメータとして数値 0 を渡す必要があります。これらのメッセージを両方処理するために、Visual Basic 6.0 では SendMessage の Declare ステートメントを以下のように作成します。

```
Private Declare Function SendMessage Lib "user32" _
Alias "SendMessageA" _
   (ByVal hwnd As Long, _
    ByVal wMsg As Long, _
    ByVal wParam As Long, _
    ByVal lParam As Any) As Long
```

最後のパラメータが As Any 型を使用して宣言されていることに注目してください。以下のコードは、この関数の 2 つの異なる呼び出しを示しています。それぞれの呼び出しでは、最後のパラメータに異なるデータ型を渡しています。この例では、SendMessage を使用して現在のフォームのキャプションを設定し、さらに設定したキャプションの長さを取得します。

```
Dim TextLen As Long
Const WM_SETTEXT = &HC
Const WM_GETTEXTLENGTH = &HE

' 最後のパラメータに String 型の値を使用して SendMessage を呼び出します。
SendMessage Me.hwnd, WM_SETTEXT, 0, "My text"
' 最後のパラメータに数値を使用して SendMessage を呼び出します。
TextLen = SendMessage(Me.hwnd, WM_GETTEXTLENGTH, 0, 0&)
```

同じ機能を Visual Basic .NET で実装するには、SendMessage 関数の Declare ステートメントを複数作成します。SendMessage API の場合は、以下に示すように、Declare ステートメントを 2 つ作成し、最後のパラメータに String と Integer の 2 つの異なるデータ型を使用します。

```
Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
   (ByVal hwnd As Integer, _
    ByVal wMsg As Integer, _
    ByVal wParam As Integer, _
    ByVal lParam As String) As Integer

Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
   (ByVal hwnd As Integer, _
    ByVal wMsg As Integer, _
    ByVal wParam As Integer, _
    ByVal lParam As Integer) As Integer
```

メモ : この章の「固定長文字列の変更」で説明したように、他のパラメータ型を Long 型から Integer 型に変更する必要もあります。

この例の場合、対応する Visual Basic .NET コードでは、SendMessage を使用して現在のフォームのキャプションを設定し、さらに設定したキャプションの長さを取得します。

```
Dim TextLen As Integer
Const WM_SETTEXT = &HC
Const WM_GETTEXTLENGTH = &HE

' 最後のパラメータに String 型の値を使用して SendMessage を呼び出します。
SendMessage(Me.Handle.ToInt32, WM_SETTEXT, 0, "My text")
' 最後のパラメータに数値を使用して SendMessage を呼び出します。
TextLen = SendMessage(Me.Handle.ToInt32, WM_GETTEXTLENGTH, 0, 0)
```

同じ関数の Declare ステートメントを複数作成するのは、As Any 変数型を使用するよりも手間がかかります。しかし、同じ関数名を使用するという柔軟性と、呼び出し時に型チェックが実行されるという利点を両方得ることができます。

API 関数へのユーザー定義型の受け渡し

Visual Basic 6.0 のユーザー定義型を API 関数に渡すと、そのユーザー定義型を含むメモリへのポインタが渡されます。API 関数は、Visual Basic で宣言されたとおりの順序で、ユーザー定義型のメンバを確認します。しかしこれは、Visual Basic .NET には当てはまりません。ユーザー定義型を宣言した場合、メンバの順序がコード内でそのまま維持されるという保証はありません。共通言語ランタイム (CLR) は、関数に渡されるユーザー定義型にとって最も効率的な方法で、ユーザー定義型のメンバを並べ替えることがあります。メンバがコードで宣言されているとおりに渡されることを保証するには、マーシャリング属性を使用する必要があります。

たとえば、2 バイトの Boolean 型 (VARIANT_BOOL) パラメータを受け取る MyFunction という名前の API 関数 (C または C++ で記述) をコードで呼び出している場合、MarshalAs 属性を使用して、この API 関数が必要とするパラメータ型を指定することができます。通常、Boolean 型のパラメータは 4 バイトを使用して渡されます。しかし、MarshalAs 属性のパラメータとして UnmanagedType.VariantBool を含めると、2 バイトの VARIANT_BOOL 引数としてマーシャリングされる (渡される) ようになります。以下のコードは、API 関数宣言に MarshalAs 属性を適用する方法を示しています。

```
Declare Sub MyFunction Lib "MyLibrary.dll" _
   (<MarshalAs(UnmanagedType.VariantBool)> ByVal MyBool As Boolean)
```

MarshalAs 属性は **System.Runtime.InteropServices** 名前空間に含まれています。**MarshalAs** を修飾子なしで呼び出すには、以下に示すように、**System.Runtime.InteropServices** の **Imports** ステートメントをモジュールの先頭に追加する必要があります。

```
Imports System.Runtime.InteropServices
```

API 関数に渡すすべての構造体は、StructLayout 属性を使用して宣言し、Windows API との互換性を保証する必要があります。StructLayout 属性は多数のパラメータを受け取りますが、互換性を保証するために最も重要な属性は、LayoutKind と CharSet の 2 つです。たとえば、構造体のメンバを宣言されているとおりの順序で渡す必要があることを指定するには、LayoutKind 属性を LayoutKind.Sequential に設定します。文字列パラメータが ANSI 文字列としてマーシャリングされることを保証するには、CharSet 属性を CharSet.Ansi に設定します。

メモ: .NET Framework は外部の API とやり取りするときに Unicode 文字列を使用する、ということを覚えておくことが重要です。ANSI エンコーディングや、基になるオペレーティング システムによって決められるエンコーディングなど、異なる文字列マーシャリング方法を使用することが必要になることもあります。

以下のコードは、外部関数の宣言例を示しています。この例では、関数が ANSI 文字列を受け取るため、ANSI の **CharSet** 属性が必要です。

```
Declare Function CharUpperA Lib "user32" (<MarshalAs(UnmanagedType.LPStr)> _
ByVal s As String) As String
```

この関数は、ANSI 文字列を 1 つ受け取り、すべての文字を大文字に変換した文字列を返します。ただし、同じ関数の Unicode 版が API に用意されている場合は、.NET Framework からアクセスするときはそちらを使用するのが好ましい選択です。Unicode 版は、エンコーディング間の変換が発生しないため、パフォーマンスに優れています。またマーシャリングが不要なため、以下の例に示すように、宣言が簡素化されます。

```
Declare Unicode Function CharUpperW Lib "user32" (ByVal s As String) As String
```

一般に、可能な限り Unicode 版の API 関数を使用し、また文字列型の引数には **<MarshalAs(UnmanagedType.LPStr)>** ではなく **<MarshalAs(UnmanagedType.LPWStr)>** を使用するようにします。API との対話に必要な構造体フィールドにも、同じ理論を適用します。可能な限り Unicode 版の構造体を使用し、また構造体と文字列フィールドのマーシャリング属性として **<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi)>** ではなく **<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Unicode)>** を使用するようにします。

EnumFontFamilies という名前の API 関数を呼び出していると仮定します。この API は、GDI32 という名前の DLL に含まれていて、LOGFONT という名前の構造体をパラメータとして受け取ります。LOGFONT 構造体は、整数型およびバイト型のフィールドを複数と、バイト型の固定長配列を 1 つ含み、Visual Basic 6.0 で以下のように宣言されます。

```
Public Const LF_FACESIZE = 32
Type LOGFONT
        lfHeight As Long
        lfWidth As Long
        lfEscapement As Long
        lfOrientation As Long
        lfWeight As Long
        lfItalic As Byte
        lfUnderline As Byte
        lfStrikeOut As Byte
        lfCharSet As Byte
        lfOutPrecision As Byte
        lfClipPrecision As Byte
        lfQuality As Byte
        lfPitchAndFamily As Byte
        lfFaceName(LF_FACESIZE) As Byte
End Type
```

アップグレードウィザードは、この宣言を以下のような類似の宣言にアップグレードします。

```
Structure LOGFONT
   Dim lfHeight As Integer
   Dim lfWidth As Integer
   Dim lfEscapement As Integer
   Dim lfOrientation As Integer
   Dim lfWeight As Integer
   Dim lfItalic As Byte
   Dim lfUnderline As Byte
   Dim lfStrikeOut As Byte
   Dim lfCharSet As Byte
   Dim lfOutPrecision As Byte
   Dim lfClipPrecision As Byte
   Dim lfQuality As Byte
   Dim lfPitchAndFamily As Byte
   <VBFixedArray(LF_FACESIZE)> Dim lfFaceName() As Byte
   Public Sub Initialize()
       ReDim lfFaceName(LF_FACESIZE)
   End Sub
End Structure
```

この宣言が類似というのは、Visual Basic 6.0 の宣言と完全に同じものにするための属性が欠けているためです。具体的には、レイアウトを指定する属性が構造体に欠けています。また、固定長配列であることを示す属性が lfFaceName メンバに欠けています。

CLR がこの構造体のメンバを宣言されたとおりの順序で渡すという保証はありません。たとえば、CLR は lfFaceName メンバが先頭になるように構造体をメモリ内で並べ替える可能性があります。メンバの順序が保持されることを保証するには、マーシャリング属性を追加して、CLR が関数に構造体を渡すときに、宣言された順序を使用するように強制する必要があります。次のコードは、これを行う方法を示しています。

```
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi)> _
Structure LOGFONT
   Dim lfHeight As Integer
   Dim lfWidth As Integer
   Dim lfEscapement As Integer
   Dim lfOrientation As Integer
   Dim lfWeight As Integer
   Dim lfItalic As Byte
   Dim lfUnderline As Byte
   Dim lfStrikeOut As Byte
   Dim lfCharSet As Byte
   Dim lfOutPrecision As Byte
   Dim lfClipPrecision As Byte
   Dim lfQuality As Byte
   Dim lfPitchAndFamily As Byte
   <VBFixedArray(LF_FACESIZE)> Dim lfFaceName() As Byte
   Public Sub Initialize()
```

```
        ReDim lfFaceName(LF_FACESIZE)
    End Sub
End Structure
```

lfFaceName メンバに関連して、もう 1 つ問題があります。この章で既に触れたように、Visual Basic .NET は固定長配列をネイティブにサポートしません。ただし、固定長文字列を必要とする API 関数に配列を含む構造体を渡すときは、バイト型メンバを文字列型の配列に変更し、MarshalAs 属性を含めて、固定サイズのバイト配列を渡すように CLR に指示することができます。それには、構造体メンバ(この場合は lfFaceName)を Byte 型の配列ではなく、Char 型の配列として宣言します。MarshalAs 属性には UnmanagedType.ByValArray 引数と SizeConst 引数を含める必要があります。変更後の構造体宣言は以下のようになります。

```
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Ansi)> _
Structure LOGFONT
    Dim lfHeight As Integer
    Dim lfWidth As Integer
    Dim lfEscapement As Integer
    Dim lfOrientation As Integer
    Dim lfWeight As Integer
    Dim lfItalic As Byte
    Dim lfUnderline As Byte
    Dim lfStrikeOut As Byte
    Dim lfCharSet As Byte
    Dim lfOutPrecision As Byte
    Dim lfClipPrecision As Byte
    Dim lfQuality As Byte
    Dim lfPitchAndFamily As Byte
    <MarshalAs(UnmanagedType.ByValArray, SizeConst:=32)> _
    Dim lfFaceName() As Char
    Public Sub Initialize()
        ReDim lfFaceName(LF_FACESIZE)
    End Sub
End Structure
```

StructLayout 属性と MarshalAs 属性を使用することで、この構造体は Visual Basic 6.0 の対応する Type…End Type 構造体と同じようにメモリに格納され、API 関数に渡されるようになります。

メモ:アップグレード ウィザードは、構造体に含まれている固定長配列を誤ってアップグレードします。アップグレード ウィザードは、アップグレート後の固定長配列を Char 型ではなく、Byte 型と宣言します。また ByValArray 属性ではなく、VBFixedArray 属性を適用します。コード内のこれらの箇所は、すべて手動で修正する必要があります。

"AddressOf" 機能の変更

EnumFontsFamilies など、一部の Windows API 関数は、コールバック関数へのポインタを必要とします。この API 関数を呼び出すと、指定したコールバック関数が Windows によって呼び出されます。EnumFontsFamilies の場合は、利用可能なフォントごとにコールバック関数が呼び出されます。

Visual Basic 6.0 では、関数ポインタ パラメータを Long として宣言し、32 ビットのポインタを表すことにより、コールバック関数ポインタを受け取る Windows API 関数を宣言することができます。コードでは、API 関数を呼び出すと共に、サブルーチン名を AddressOf キーワードで修飾することによって、コールバック関数として機能するサブルーチンを渡します。

Visual Basic .NET は AddressOf キーワードを引き続きサポートしますが、32 ビットの整数を返すのではなく、デリゲートを返します。デリゲートは、関数またはクラス メンバへのポインタを宣言することを可能にする、Visual Basic .NET の新しいデータ型です。これは、コールバック関数のポインタをパラメータとして受け取る Windows API 関数の Declare ステートメントを引き続き作成できることを意味します。違いは、パラメータ型を 32 ビット整数として宣言するのではなく、関数パラメータをデリゲート型として宣言する必要があるということです。

以下の Visual Basic 6.0 コードは、AddressOf を使用してコールバック関数へのポインタを渡す方法を示しています。

```
Public Const LF_FACESIZE = 32
Type LOGFONT
    lfHeight As Long
    lfWidth As Long
    lfEscapement As Long
    lfOrientation As Long
    lfWeight As Long
    lfItalic As Byte
    lfUnderline As Byte
    lfStrikeOut As Byte
    lfCharSet As Byte
    lfOutPrecision As Byte
    lfClipPrecision As Byte
    lfQuality As Byte
    lfPitchAndFamily As Byte
    lfFaceName(LF_FACESIZE) As Byte
End Type

Declare Function EnumFontFamilies Lib "gdi32" Alias "EnumFontFamiliesA" _
    (ByVal hDC As Long, ByVal lpszFamily As String, _
    ByVal lpEnumFontFamProc As Long, LParam As Integer) As Long
Declare Function GetDC Lib "user32" (ByVal hWnd As Long) As Long
Declare Function ReleaseDC Lib "user32" (ByVal hWnd As Long, _
    ByVal hDC As Long) As Long

Dim FontList As New Collection
```

```
Function EnumFontFamProc(lpNLF As LOGFONT, lpNTM As Long, _
         ByVal FontType As Long, LParam As Integer) As Long
    Dim FaceName As String
    Dim FullName As String
    FaceName = StrConv(lpNLF.lfFaceName, &H40)
    FontList.Add Left$(FaceName, InStr(FaceName, vbNullChar) - 1)
    EnumFontFamProc = 1
End Function

Sub FillListWithFonts(LB As ListBox)
    Dim font As Variant
    Dim hDC As Long
    LB.Clear
    hDC = GetDC(LB.hWnd)
    EnumFontFamilies hDC, vbNullString, AddressOf EnumFontFamProc, 0
    For Each font In FontList
      LB.AddItem font
    Next
    ReleaseDC LB.hWnd, hDC
End Sub
```

アップグレードウィザードは、このコードを以下のコードにアップグレードします。

```
Public Const LF_FACESIZE As Short = 32
Structure LOGFONT
    Dim lfHeight As Integer
    Dim lfWidth As Integer
    Dim lfEscapement As Integer
    Dim lfOrientation As Integer
    Dim lfWeight As Integer
    Dim lfItalic As Byte
    Dim lfUnderline As Byte
    Dim lfStrikeOut As Byte
    Dim lfCharSet As Byte
    Dim lfOutPrecision As Byte
    Dim lfClipPrecision As Byte
    Dim lfQuality As Byte
    Dim lfPitchAndFamily As Byte
    <VBFixedArray(LF_FACESIZE)> Dim lfFaceName() As Byte

    Public Sub Initialize()
        ReDim lfFaceName(LF_FACESIZE)
    End Sub
End Structure

Declare Function EnumFontFamilies Lib "gdi32" Alias _
    "EnumFontFamiliesA"(ByVal hDC As Integer, ByVal lpszFamily As String, _
    ByVal lpEnumFontFamProc As Integer, ByRef LParam As Short) As Integer
Declare Function GetDC Lib "user32" (ByVal hWnd As Integer) As Integer
Declare Function ReleaseDC Lib "user32" (ByVal hWnd As Integer, _
    ByVal hDC As Integer) As Integer

Dim FontList As New Collection
```

```
Function EnumFontFamProc(ByRef lpNLF As LOGFONT, ByRef lpNTM As Integer, _
         ByVal FontType As Integer, ByRef LParam As Short) As Integer
    Dim FaceName As String
    Dim FullName As String
    FaceName = _
        StrConv(System.Text.UnicodeEncoding.Unicode.GetString(lpNLF.lfFaceName), _
        &H40s)
    FontList.Add(Left(FaceName, InStr(FaceName, vbNullChar) - 1))
    EnumFontFamProc = 1
End Function

Sub FillListWithFonts(ByRef LB As System.Windows.Forms.ListBox)
    Dim font As Object
    Dim hDC As Integer
    LB.Items.Clear()
    hDC = GetDC(LB.Handle.ToInt32)
    'UPGRADE_WARNING: AddressOf EnumFontFamProc の delegate を追加する
    EnumFontFamilies(hDC, vbNullString, AddressOf EnumFontFamProc, 0)
    For Each font In FontList
        LB.Items.Add(font)
    Next font
    ReleaseDC(LB.Handle.ToInt32, hDC)
End Sub
```

アップグレード後のコードにはいくつかの問題があります。このコードをエラーなしでコンパイルして実行するには、これらの問題を修正する必要があります。1 つ目の問題は、構造体 LOGFONT に関連しています。この構造体にはマーシャリング属性を適用し、メモリに格納して API 関数に渡す方法を指定する必要があります。マーシャリング属性の適用については、この章で既に説明した「API 関数へのユーザー定義型の受け渡し」を参照してください。

2 つ目の問題は、固定バイト配列の文字列への変換に関連しています。このエラーは以下の行で発生します。

```
FaceName = _
    StrConv(System.Text.UnicodeEncoding.Unicode.GetString(lpNLF.lfFaceName), _
    &H40S)
```

この行を、以下の行に変更します。

```
FaceName = New String(lpNLF.lfFaceName)
```

最後の問題は、AddressOf 式に関連しています。このコンパイル エラーが発生するのは、Integer がデリゲート型ではないために、AddressOf 式を Integer に変換できないからです。このエラーは以下のステートメントで発生します。

```
EnumFontFamilies(hDC, vbNullString, AddressOf EnumFontFamProc, 0)
```

このエラーを修正するには、EnumFontFamilies の Declare 宣言を、Integer 型ではなく、EnumFontFamProc パラメータのデリゲート型を受け取るように変更する必要があります。EnumFontFamProc 関数のデリゲート宣言を作成するための最も簡単な方法は、以下の手順を実行することです。

1. コールバック関数として機能するサブルーチン宣言をコピーし、コード内に貼り付けます。
2. 宣言の先頭に Delegate キーワードを挿入します。
3. Delegate という単語を関数名に追加することにより、関数名を変更します。デリゲート名は一意である必要があります。

上記の例では、以下の関数シグネチャをコピーし、Declare ステートメントを含むコード セクションに貼り付けることにより、デリゲート宣言を作成することができます。

```
Function EnumFontFamProc(ByRef lpNLF As LOGFONT, ByRef lpNTM As Integer, _
    ByVal FontType As Integer, ByRef LParam As Short) As Integer
```

関数宣言の先頭にキーワード Delegate を挿入し、元の関数名に Delegate を追加して名前を変更します。EnumFontFamProc は EnumFontFamProcDelegate になります。以下のように、EnumFontFamProc の構造、名前、およびパラメータを備える関数のデリゲート宣言が完成します。

```
Delegate Function EnumFontFamProcDelegate(ByRef lpNLF As LOGFONT, _
        ByRef lpNTM As Integer, ByVal FontType As Integer, _
        ByRef LParam As Short) As Integer
```

コールバック関数のデリゲート宣言が用意できたら、以下に示すように、EnumFontFamilies の Declare ステートメントのパラメータ型を、Integer からデリゲート型 EnumFontFamProcDelegate に変更する必要があります。

```
Declare Function EnumFontFamilies Lib "gdi32" Alias "EnumFontFamiliesA" _
       (ByVal hDC As Integer, ByVal lpszFamily As String, _
        ByVal lpEnumFontFamProc As EnumFontFamProcDelegate, _
        ByRef LParam As Short) As Integer
```

これらの変更により、コードはエラーなしでコンパイルされて実行され、元の Visual Basic 6.0 コードと同じ結果を生成するようになります。

ObjPrt 関数、StrPtr 関数、および VarPtr 関数のサポートの廃止

Visual Basic 6.0 では、オブジェクト、文字列、あるいは変数またはユーザー定義値の 32 ビット アドレスを取得することができます。それには、ドキュメント化されていないヘルパ関数である ObjPtr、StrPtr、および VarPtr をそれぞれ使用します。Visual Basic .NET はこれらの関数をサポートしておらず、いかなるデータ型のメモリ アドレスの取得も許可していません。一見すると、これは制約が厳しすぎるように思えます。しかし、この制約による利点は重要です。メモリ管理を CLR に任せることにより、開発者はメモリ リソースの割り当てと解放につい

て考える必要がなくなり、アプリケーションとビジネス ロジックに専念することができるようになります。また、この制約によってアプリケーションの信頼性とセキュリティも向上します。マネージ アプリケーションのメモリ領域は、互いに独立しています。アプリケーションがポインタを誤って (または故意に) 使用し、別のアプリケーションのメモリ領域にアクセスすることはもはや不可能です。

基になるメモリを制御する必要がある場合は、.NET Framework に System.IntPtr という名前のポインタ型が用意されています。System.IntPtr は、ポインタまたはハンドルを表すために使用されるプラットフォーム固有のデータ型です。このデータ型を、アンマネージ メモリ操作用のメソッドを含む System.Runtime.InteropServices.Marshal クラスと組み合わせて使用すると、特定のデータ型のポインタを取得することができます。たとえば、AllocHGlobal、GetComInterfaceForObject、および OffSetOf の各関数は、いずれもメモリへのポインタを返します。また、GCHandle 構造体を使用し、ガベージ コレクタによって管理されるメモリに含まれている要素へのハンドルを取得することもできます。さらに、ガベージ コレクタにオブジェクトを単一のメモリ位置に固定させ、そのオブジェクトが削除されるのを防ぐことにより、メモリへのポインタを取得することもできます。

以下の "Module1" という名前の Visual Basic 6.0 モジュール例を考えてみましょう。このモジュールは、CopyMemory を呼び出して、コピー元のユーザー構造体をコピー先の構造体にコピーします。このとき、VarPtr を使用してコピー元の変数とコピー先の変数のメモリ アドレスを取得します。

```
Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" _
   (ByVal lpDest As Long, ByVal lpSource As Long, ByVal cbCopy As Long)

Type MyType
    valid As Boolean
    x As Long
    y As Long
End Type

Sub Test()
   Dim source As MyType
   Dim dest As MyType

   source.valid = True
   source.x = 10
   source.y = 10

   CopyMemory ByVal VarPtr(dest), _
              ByVal VarPtr(source), LenB(source)

   MsgBox "Valid = " & dest.valid & ": X = " & dest.x & " - Y = " & dest.y
End Sub
```

以下のコードは、Visual Basic アップグレード ウィザードの出力です。

```
Option Strict Off
Option Explicit On
Module Module1
    Declare Sub CopyMemory Lib "kernel32"  Alias "RtlMoveMemory"(ByVal lpDest As Integer, ByVal lpSource
As Integer, ByVal cbCopy As Integer)


    Structure MyType
        Dim valid As Boolean
        Dim x As Integer
        Dim y As Integer
    End Structure


    Sub Test()
        Dim source As MyType
        Dim dest As MyType

        source.valid = True
        source.x = 10
        source.y = 10

        ' UPGRADE_ISSUE: LenB 関数はサポートされません。
        ' UPGRADE_ISSUE: VarPtr 関数はサポートされません。
        CopyMemory(VarPtr(dest), VarPtr(source), LenB(source))

        MsgBox("Valid = " & dest.valid & ": X = " & dest.x & " - Y = " & _
                    dest.y)
    End Sub
End Module
```

▶ 完全なアップグレードを実行するには

1. InteropServices 名前空間をインポートし、Marshal クラスを使用してアンマネージ メモリを操作できるようにします。

```
Imports System.Runtime.InteropServices
```

2. 以下の変数を作成して初期化します。

```
Dim addressOfSource As IntPtr
Dim addressOfDest As IntPtr
Dim MyStrucSize As Integer

MyStrucSize = Marshal.SizeOf(GetType(MyType))
addressOfSource = IntPtr.Zero
addressOfDest = IntPtr.Zero

addressOfDest = Marshal.AllocHGlobal(MyStrucSize)
addressOfSource = Marshal.AllocHGlobal(MyStrucSize)
```

addressOfSource 変数と addressOfDest 変数は、MyType の 2 つのインスタンスのアドレスを格納するために使用します。MyStrucSize 変数は、MyType 構造体のサイズを格納するために使用します。IntPtr.Zero は、ゼロに初期化されたポインタまたはハンドルを表す、読み取り専用のフィールドです。Marshal.AllocHGlobal メソッドは、GlobalAlloc を使用することにより、プロセスのアンマネージ メモリからメモリ ブロックを割り当てます。

3. コピー元の MyType インスタンスから、アドレス変数によって表されるアンマネージ メモリ ブロックに、データをマーシャリングします。

```
Marshal.StructureToPtr(source, addressOfSource, True)
```

4. アンマネージ メモリを使用して source から dest にデータをコピーします。

```
CopyMemory(addressOfDest.ToInt32, addressOfSource.ToInt32, MyStrucSize)
```

5. アンマネージ メモリ ブロック addressOfDest からマネージ オブジェクト dest にデータをマーシャリングします。

```
dest = Marshal.PtrToStructure(addressOfDest, GetType(MyType))
```

6. 以下のコード例に示すように、割り当てられたメモリを解放します。

```
Marshal.FreeHGlobal(addressOfSource)
Marshal.FreeHGlobal(addressOfDest)
```

Visual Basic 6.0 コードに対応する Visual Basic .NET コードを以下に示します。

```
Option Strict Off
Option Explicit On

Imports System.Runtime.InteropServices

Module Module1
    Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (ByVal lpDest As Integer, ByVal lpSource
As Integer, ByVal cbCopy As Integer)

    Structure MyType
        Dim valid As Boolean
        Dim x As Integer
        Dim y As Integer
    End Structure

    Sub Test()
        Dim source As MyType
        Dim dest As MyType

        Dim addressOfSource As IntPtr
        Dim addressOfDest As IntPtr
        Dim MyStrucSize As Integer
```

```
        MyStrucSize = Marshal.SizeOf(GetType(MyType))
        addressOfSource = IntPtr.Zero
        addressOfDest = IntPtr.Zero

        source.valid = True
        source.x = 10
        source.y = 10

        addressOfDest = Marshal.AllocHGlobal(MyStrucSize)
        addressOfSource = Marshal.AllocHGlobal(MyStrucSize)

        Marshal.StructureToPtr(source, addressOfSource, True)

        CopyMemory(addressOfDest.ToInt32, addressOfSource.ToInt32, MyStrucSize)

        dest = Marshal.PtrToStructure(addressOfDest, GetType(MyType))

        MsgBox("Valid = " & dest.valid & ": X = " & dest.x & " - Y = " & dest.y)

        Marshal.FreeHGlobal(addressOfSource)
        Marshal.FreeHGlobal(addressOfDest)
    End Sub
End Module
```

ObjPtr や VarPtrArray など、他の似たような Visual Basic 6.0 関数でも、同じアプローチを使用することができます。

メモ:アンマネージ メモリ操作を使用する場合は、メモリ リソースの割り当てと解放を手動で行う必要があるということに注意してください。マネージ メモリ操作の使用については、次の「API 呼び出しの Visual Basic .NET への移行」を参照してください。

API 呼び出しの Visual Basic .NET への移行

API 呼び出しを Visual Basic 6.0 から Visual Basic .NET にアップグレードするときのもう 1 つの方法は、Declare ステートメントではなく、Visual Basic .NET の代替機能を使用することです。このアプローチの利点は、Visual Basic .NET のすべての代替機能は、API がアンマネージの場合でも、完全なマネージ コードになるということです。API の多くは、それらに対応するものが Visual Basic .NET または .NET Framework に存在します。

Visual Basic .NET の代替機能による API 関数の置き換えは、ケースバイケースで行う必要があります。以下の例は、いくつかの API 呼び出しを Visual Basic .NET の対応する処理にアップグレードする方法を示しています。この例では、FindFirstFile API 関数と FindNextFile API 関数を使用して、特定のディレクトリのファイルを一覧表示します。

これらの API 関数の Visual Basic 6.0 の Declare ステートメントを以下に示します。

```
Private Const MAX_PATH = 260
Private Const ERROR_NO_MORE_FILES = 18&
```

```
Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Type WIN32_FIND_DATA
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    nFileSizeHigh As Long
    nFileSizeLow As Long
    dwReserved0 As Long
    dwReserved1 As Long
    cFileName As String * MAX_PATH
    cAlternate As String * 14
End Type

Private Declare Function FindFirstFile Lib "kernel32" _
    Alias "FindFirstFileA" (ByVal lpFileName As String, _
    lpFindFileData As WIN32_FIND_DATA) As Long
Private Declare Function FindNextFile Lib "kernel32" Alias _
    "FindNextFileA" (ByVal hFindFile As Long, _
    lpFindFileData As WIN32_FIND_DATA) As Long
```

以下のコードは、これらの関数を Visual Basic 6.0 アプリケーションで使用する方法を示しています。ListFilesAndDirs プロシージャは、ディレクトリ パスをパラメータとして受け取り、そのパスに含まれているすべてのディレクトリとファイルを一覧表示します。ListFilesAndDirs プロシージャの Visual Basic 6.0 コードを以下に示します。

```
Public Sub ListFilesAndDirs(ByVal DirPath As String)
    Dim FirstFile As Long
    Dim NextFile As Long
    Dim FileData As WIN32_FIND_DATA
    Dim pos As Integer

    DirPath = DirPath & "*.*"
    FirstFile = FindFirstFile(DirPath, FileData)
    If FirstFile <> -1 Then
        NextFile = FindNextFile(FirstFile, FileData)
        While (NextFile <> ERROR_NO_MORE_FILES) And (NextFile <> 0)
            pos = InStr(FileData.cFileName, vbNullChar)
            If pos = 0 Then
                Debug.Print FileData.cFileName
            Else
                If pos > 1 Then
                    Debug.Print Left$(FileData.cFileName, pos - 1)
                End If
            End If
            NextFile = FindNextFile(FirstFile, FileData)
            Wend
    End If
End Sub
```

Visual Basic 6.0 コードをアップグレード ウィザードによって変換したら、いくつかの手順を実行し、API 呼び出しへの依存を取り除いて、それらを Visual Basic .NET の対応する機能で置き換える必要があります。最初の手順では、Declare ステートメントと構造体をすべて取り除きます。これは、API 呼び出しを使用していないからです。

2 番目の手順では、FindFirstFile API 関数と FindNextFile API 関数の呼び出しを、.NET Framework 関数の System.IO.Directory.GetFiles と System.IO.Directory.GetDirectories で置き換えます。これらの Visual Basic .NET 関数は文字列型の配列を返すため、新しい Visual Basic .NET 関数は以下のようになります。

```
Dim List As String() = System.IO.Directory.GetFiles(DirPath, "*.*")
Dim ListofDirs As String() = _
    System.IO.Directory.GetDirectories(DirPath, "*.*")
pos = List.Length
ReDim Preserve List(List.Length + ListofDirs.Length - 2)
ListofDirs.CopyTo(List, pos - 1)
```

これらの Visual Basic .NET 関数がパターン パラメータ Path および Search を受け取ることに注目してください。そのため、Visual Basic 6.0 の以下の行をコードから取り除く必要があります。

```
DirPath = DirPath & "*.*"
```

また、これらの Visual Basic .NET 関数は文字列の配列を返すため、最初の If ステートメントと While ステートメントを、以下のような For ステートメントで置き換えることができます。

```
For i = 0 To list.Length - 1
```

最後の手順として、ファイル名を取得して出力するコードを変更する必要があります。これは、構造体を使用していないからです。以下に Visual Basic 6.0 コードを示します。

```
FileName = list(i)
pos = FileName.LastIndexOf("\\")
If pos > 1 Then
    FileName = FileName.Substring(pos + 2, FileName.Length-pos-2)
End If
System.Diagnostics.Debug.WriteLine(FileName)
```

対応する Visual Basic .NET コードを以下に示します。

```
Public Function ListFiles(ByVal DirPath As String) As String()
    Dim pos As Short
    Dim i As Integer
    Dim FileName As String

    Dim List As String() = System.IO.Directory.GetFiles(DirPath, "*.*")
    Dim ListofDirs As String() = _
        System.IO.Directory.GetDirectories(DirPath, "*.*")
    pos = List.Length
    ReDim Preserve List(List.Length + ListofDirs.Length - 2)
```

```
    ListofDirs.CopyTo(List, pos - 1)

    For i = 0 To List.Length - 1
        FileName = List(i)
        pos = FileName.LastIndexOf("\\")
        If pos > 1 Then
            FileName = FileName.Substring(pos + 2, FileName.Length-pos-2)
        End If
        System.Diagnostics.Debug.WriteLine(FileName)
    Next
End Function
```

この例は、Windows API への依存を取り除くために利用できる API の置き換えの一部を示しているに過ぎません。Win32 関数と同じような機能を提供する Microsoft .NET Framework バージョン 1.0 または 1.1 の API の一覧については、MSDN の「Microsoft Win32 to Microsoft .NET Framework API Map」を参照してください。

まとめ

この章では、Windows API 上に構築された Visual Basic 6.0 アプリケーションを Visual Basic .NET にアップグレードする方法を、2 つの異なる方法を使用して示しました。

1 つ目の方法では、Windows API を Visual Basic .NET コード内で引き続き使用します。このアプローチを適用するときは、API 呼び出しの正しい動作を保証するために、適切にデータ型を更新し、マーシャリング属性を適用する必要があります。

2 つ目の方法では、Windows API の呼び出しを、Visual Basic .NET の対応する関数呼び出しで置き換えます。このアプローチでは、1 つ目の方法よりもアップグレード プロセス中の作業が多くなります。しかし、アプリケーションが旧式のアンマネージ API に依存する度合いは軽減されます。

いずれかの方法を使用することにより、Windows API 上に構築された Visual Basic 6.0 アプリケーションと同じ機能を、Visual Basic .NET で実現することができます。

詳細情報

アップグレード プロセスを開始する前にアプリケーションに対して行う準備作業の詳細については、MSDN の「Preparing Your Visual Basic 6.0 Applications for the Upgrade to Visual Basic .NET」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvb600/html/vb6tovbdotnet.asp です。

Win32 関数と同じような機能を提供する Microsoft .NET Framework バージョン 1.0 または 1.1 の API の一覧については、MSDN の「Microsoft Win32 to Microsoft .NET Framework API Map」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/win32map.asp です。

14

Visual Basic 6.0 と Visual Basic .NET の相互運用

第 1 章「はじめに」では、アプリケーションを Microsoft Visual Basic .NET にアップグレードする際に利用可能なオプションについて紹介しました。そこでは、各アプローチの課題とリスクについての概要を示し、特定のアプリケーションに最適なアップグレード方法を選択するために役立つガイドラインを提供しました。第 1 章で説明したように、アップグレード処理を付加的に進めたり、部分的なアップグレードを行うことで、リスクを軽減し、課題の深刻度合いを小さくすることができます。これらのアップグレード方法では、Visual Basic 6.0 モジュールと Microsoft .NET Framework アセンブリを相互運用させる必要があります。また、Visual Basic .NET モジュールを Visual Basic 6.0 モジュールと相互運用させる必要がある場合もあります。従来の Visual Basic 6.0 の実行環境と、Visual Basic .NET で使用される管理された実行環境では、さまざまな違いがあります。これらの違いが原因で、Visual Basic 6.0 コード モジュールから .NET アセンブリを直接起動することはできません。そこで、.NET アセンブリを共通言語ランタイム (CLR) で管理されたコンテキスト内で引き続き実行しつつ、Visual Basic 6.0 のコード モジュールにアクセス可能にするには、あるメカニズムを使用する必要があります。

アプリケーションをアップグレードする前に、適切な機能分析を必ず行うようにしてください。これにより、アプリケーションの機能的なコンポーネントを識別し、アップグレード プロセス全体に対するより効果的な計画を作成できます。また、アップグレードに必要な作業量を減らすために削除すべき冗長なコード モジュールや、使用されていないコード モジュールが明確になることから、アプリケーションの改良にも役立ちます。Visual Basic .NET にアップグレードする必要があるコンポーネントと、(一時的または永久に) Visual Basic 6.0 に残しておいてかまわないコンポーネントを明確にした後で、コンポーネント間の効果的な相互運用を作成し始めることができます。

この章では、Visual Basic 6.0 のコード モジュールと .NET アセンブリ間で相互運用を実現する方法につい

て説明します。相互運用は、任意の .NET 言語で記述されたアセンブリで実現できますが、この章では、Visual Basic .NET だけに焦点を当てます。

メモ：この章に掲載されている Visual Basic 6.0 のコード サンプルを実行するには、Microsoft Service Pack 6 for Visual Basic 6.0 がインストールされ、正しく機能している必要があります。この サービス パック は、MSDN からダウンロードできます。

Visual Basic 6.0 クライアントからの .NET アセンブリの呼び出し

.NET Framework は、COM と相互運用できるように基本から設計されました。COM コンポーネントから .NET アセンブリを呼び出したり、逆に .NET アセンブリから COM コンポーネントを呼び出すことができます。さらに、.NET アセンブリは、COM コンポーネントと同様に参照できるようにビルドすることができます。そのためには、COM 呼び出しが可能となるようにアセンブリをビルドします。これにより、アセンブリの機能とインターフェイスを他のコードから参照できるようになります。

COM 呼び出し可能なアセンブリは、Visual Basic 6.0 アプリケーションから、通常の COM コンポーネントであるかのように参照できます。この場合は、正しいアセンブリのタイプ ライブラリ (.tlb ファイル) を参照し、必要な機能にアクセスするだけで、簡単に相互運用が実現できます。しかし、このアプローチには欠点があります。それについてはこの章の後半で説明します。

Visual Basic .NET クライアントからの Visual Basic 6.0 ライブラリの呼び出し

前述のように、Visual Basic .NET アプリケーションは COM コンポーネントとやり取りすることができます。この機能には、Visual Basic 6.0 で作成された COM コンポーネントが含まれます。

Visual Basic .NET クライアントから Visual Basic 6.0 コードにアクセスするには、Visual Basic 6.0 コードを COM コンポーネントとして公開し、Visual Basic .NET クライアントから COM コンポーネントとして参照します。

相互運用を実現する方法

Visual Basic 6.0 と Visual Basic .NET の間で相互運用を実現するのは、難しいことではありません。実現するには 2 つのアプローチがあります。

最初のアプローチは、COM を通じて直接アクセスすることです。このアプローチでは、参照される側の .NET アセンブリを COM 呼び出し可能なアセンブリとして作成することが必要です。

2 つ目のアプローチは、最初のアプローチを拡張したものです。COM 呼び出し可能なアセンブリを作成する代わりに、.NET 言語を使用して、アクセスする機能に対する COM 呼び出し可能ラッパーを作成します。このアプローチでは、アクセスするアセンブリに対する特別なビルド要件はありませんが、ラッパー自体は COM 呼び出し可能としてビルドする必要があります。

相互運用を実現するには、満たす必要がある要件がいくつかあります。次に、これらの要件について説明し、その後で前述した相互運用を実現するためのさまざまなアプローチについて説明します。

アクセス要件

Visual Basic 6.0 からアクセスするすべての .NET コンポーネントは、システムの相互運用機能に登録する必要があります。登録はグローバルなレベルで行われ、コンポーネントはすべての Visual Basic 6.0 アプリケーションから使用可能になります。アセンブリは、.NET アプリケーションの主要なビルド ブロックであり、COM クライアントが使用できるさまざまなクラスが含まれています。アセンブリを登録することで、登録したアセンブリ内のクラスが、システム上のすべての Visual Basic 6.0 アプリケーションから使用可能になります。ネイティブなダイナミック リンク ライブラリとアセンブリの詳細については、第 4 章「一般的なアプリケーションの種類」を参照してください。

.NET アセンブリを登録し、COM 呼び出し可能にするには、タイプ ライブラリ (.tlb) を作成し、対応するエントリをシステムのレジストリに追加することが必要です。幸いにも、これは、適切なビルド構成オプションを設定することで、Visual Studio .NET IDE の中から簡単に実行できます。

▶ COM 相互運用機能にコンポーネントを登録するには

1. COM 呼び出し可能にするクラスが含まれている Visual Basic .NET プロジェクトを開きます。
2. [**プロジェクト**] メニューの [**プロパティ**] をクリックします。
3. [**構成プロパティ**] フォルダの [**ビルド**] をクリックします。
4. [COM **相互運用機能の登録**] チェック ボックスをオンにし、[OK] をクリックします。
5. [**ファイル**] メニューの [**すべてを保存**] をクリックし、プロジェクトの構成変更を保存します。
6. [**ビルド**] メニューの [**ソリューションのビルド**] をクリックします。

この手順に従うことで、Visual Basic .NET の DLL が作成され、COM 相互運用機能に登録されます。アセンブリの AssemblyName プロパティの値が、COM として登録される名前になります。このプロパティを変更するには、以下の手順を実行します。

▶ アセンブリの AssemblyName プロパティを変更するには

1. ソリューション エクスプローラでプロジェクト名を右クリックし、[**プロパティ**] をクリックします。
2. [**共通プロパティ**] フォルダの [**全般**] をクリックします。
3. [**アセンブリ名**] ボックスに、アセンブリの名前を指定します。
4. [OK] をクリックして変更を保存します。

アセンブリは、.NET Framework と共に配布されているアセンブリ登録ツール (Regasm.exe) を使用して、Windows のコマンド ラインからも登録できます。ビルド中にアセンブリを登録するには前述の手順を使用でき

るため、コマンド ラインでの登録が必要になるのは、一般に配置の際だけです。Regasm.exe ユーティリティを使用してアセンブリを登録するためのコマンドは以下のとおりです。

regasm Assembly.dll /tlb: Assembly.tlb

このコマンドはアセンブリを登録し、通常の COM オブジェクトとして参照できるようにするためのタイプ ライブラリを生成します。斜体の項目は、プロジェクトごとの適切な名前に置き換えます。/tlb オプションを使用して、生成するタイプ ライブラリ ファイルを指定します。

メモ：アセンブリ間で名前が矛盾しないように注意してください。新しいアセンブリを登録する前に、必ずそのアセンブリの古いバージョンの登録を解除してください。

アセンブリの登録を解除する必要がある場合は、次の regasm コマンドを使用します。

regasm /u Assembly.dll

Visual Basic 6.0 アプリケーションが実行時に .NET アセンブリを見つけるためには、アセンブリはグローバル アセンブリ キャッシュに格納されているか、アプリケーションの実行可能ファイルがあるフォルダに格納されている必要があります。アセンブリを別の場所にインストールし、かつ Visual Basic 6.0 アプリケーションへグローバルにアクセスできるようにするには、Regasm.exe ユーティリティで /codebase オプションを使用します。このオプションにより、共通言語ランタイムが対応するアセンブリを検出するために使用するファイル パスを、ユーザーが指定できるようになります。なお、/codebase オプションの詳細はこのガイドの範囲を超えています。また、COM 呼び出し可能なアセンブリは、アプリケーションを配置する際に、グローバル アセンブリ キャッシュに格納することをお勧めします。

COM との相互運用のための要件

.NET のマネージ型 (クラス、インターフェイス、構造体、または列挙型など) を COM クライアントに公開するには、以下の要件を満たしている必要があります。

- **マネージ型はパブリックである必要があります。**相互運用機能に登録してタイプ ライブラリにエクスポートできるのは、アセンブリ内のパブリック型だけです。したがって、COM から参照できるのは、パブリック型だけです。

  マネージ型は、COM に公開しない機能を他のマネージ コードに公開します。たとえば、パラメータ化されたコンストラクタ、スタティック メソッド、および定数フィールドは、COM クライアントには公開されません。また、相互運用マーシャリングを使用すると動作に違いが生じる場合があることに注意してください。

- **メソッド、プロパティ、フィールド、およびイベントはパブリックである必要があります。**パブリック型のメンバも、COM から参照する場合、パブリックである必要があります。アセンブリ、パブリック型、またはパブリック型のパブリック メンバを参照の参照は、ComVisibleAttribute を適用することで制限できます。既定では、すべてのパブリック型とメンバが参照可能です。

- **型には、COM からアクティブになるパブリックな既定のコンストラクタが定義されている必要があります。**パブリックなマネージ型は COM から参照できます。ただし、パブリックな既定のコンストラクタ (引数のないコンストラクタ) がない場合、COM クライアントはその型を作成できません。他のクラス メソッド

から間接的に作成された場合など、他の手段でアクティブにされた型は、COM クライアントで使用することができます。

- **型を抽象型にすることはできません。** COM クライアントおよび .NET クライアントは、いずれも抽象型を作成できません。
- **特定の Visual Basic .NET の機能を利用すると、メソッドが COM 呼び出し可能でなくなります。** 共有の (静的な) Visual Basic .NET メソッドは、COM オブジェクトから呼び出すことができません。また、メソッドの引数と戻り値は、Integer 型や String 型などの単純なデータ型である必要があります。メソッドの引数や戻り値が、COM 相互運用機能での利用を意図していない型の場合は、メソッドを Visual Basic 6.0 から呼び出すことはできません。また、Visual Basic .NET のメソッドはオーバーロードできません。COM はメソッドのオーバーロードをサポートしていないため、Visual Basic 6.0 では、各オーバーロードの名前は、元の名前と異なる自動的に生成された名前になります。.NET ラッパー クラスを作成する際には、メソッドに一意の名前を付けることをお勧めします。

また、COM にエクスポートすると、マネージ型の継承階層が平坦化されます。その結果、基本クラスのメンバは、派生クラスの中で宣言されているかのように公開されます。

Visual Basic 6.0 から .NET アセンブリへの直接アクセス

.NET アセンブリを Visual Basic 6.0 から呼び出すプロセスは、非常に簡単です。一般的なプロセスは以下のとおりです。

1. アクセスするアセンブリを特定します。
2. アセンブリのタイプ ライブラリを検出します。

メモ：COM クライアントからアセンブリにアクセスするには、アセンブリが Assembly Registration ユーティリティ (RegAsm.exe) を使用して登録されている必要があります。アセンブリの登録の詳細については、この章の「アクセス要件」を参照してください。

---

3. Visual Basic 6.0 アプリケーションで、アセンブリの目的のタイプ ライブラリへの参照を追加します。
4. 必要な機能をアセンブリから呼び出します。

このアプローチの欠点は、Visual Basic .NET と .NET Framework の機能の一部にしかアクセスできない可能性があるということです。その理由は以下のとおりです。

- すべての .NET アセンブリが COM 呼び出し可能なわけではありません。ComVisible 属性では、COM クライアントに対するアセンブリのメンバのアクセス可能性を制御します。あるアセンブリでこの属性に False が設定されていると、そのアセンブリ内のすべてのパブリック型が参照不可となり、COM クライアントにアクセスできなくなります。ComVisible 属性は、アセンブリ、インターフェイス、クラス、含まれているメンバに個別の方法で設定できます。

- Visual Basic .NET で使用されているオブジェクト指向プログラミングのパラダイムには、Visual Basic 6.0 で使用できないものがあります。たとえば、.NET クラスのいくつかは抽象クラスであり、使用するにはサブクラスが必要になりますが、これは Visual Basic 6.0 では不可能です。
- Visual Basic 6.0 では、.NET 例外を部分的にしかサポートしていません。.NET アセンブリは、エラー状態を通知するメカニズムとして例外を使用するように作成できますが、Visual Basic 6.0 では、構造化されたオブジェクト指向の例外として例外をサポートしていません。そのため、.NET コンポーネントを呼び出した後でエラー状態を検出するには、Visual Basic 6.0 コンポーネントの場合と同様に、呼び出しの直後または On Error Goto ステートメントを使用して、例外の発生に関して明確に確認する必要があります。これにより、.NET のエラー管理機構で得られる利点が減少します。

この欠点にもかかわらず、このアプローチは、過度に複雑で費用がかかる完全なアップグレードに対する、最適な代替方法です。

Visual Basic 6.0 から .NET アセンブリにアクセスするプロセスについては、以下の手順で詳しく説明します。この例のプロジェクトは、.NET System タイプ ライブラリを参照します。これには、.NET コードで使用される基礎的なクラスと基本クラスが含まれています。また、Visual Basic 6.0 と Visual Studio .NET (または .NET Framework のいずれかのバージョン) が同じシステムにインストールされていることを前提としています。

▶ Visual Basic 6.0 アプリケーションから .NET アセンブリにアクセスするには

1. アクセスするアセンブリを登録します。これにより、Visual Basic 6.0 が .NET アセンブリのインスタンスを生成して、そのアセンブリにアクセスできるようになります。以下の手順に従ってアセンブリを登録します。
   a. Windows でコマンド プロンプトを開きます。既定では、コマンド プロンプトのショートカットは、**[スタート]** メニューの **[プログラム]** サブメニューまたは **[すべてのプログラム]** サブメニュー (Windows のバージョンによって異なります) の、**[アクセサリ]** カテゴリにあります。
   b. 現在のディレクトリを、System.dll ファイルがある .NET Framework ディレクトリに変更します。それには、コマンド プロンプトで次のコマンドを入力します。
   cd DriveLetter:\Windows\Microsoft.NET\Framework\v1.1.4322.

   メモ：*DriveLetter* は .NET Framework がインストールされているドライブを表し、名前 v1.1.4322 はインストールされている .NET Framework のバージョンに対応しています。実際の名前はシステムによって異なることがあります。どこにインストールされているか分からない場合は、ファイル システムで Microsoft.NET\Framework フォルダを検索します。

   c. コマンド プロンプトで次のコマンドを入力し、System.dll ファイルを登録します。RegAsm System.dll.
2. Visual Basic 6.0 を起動します。
3. **[新しいプロジェクト]** ダイアログ ボックスで、[Standard.exe] をクリックして新しいプロジェクトを作成します。
4. **[プロジェクト]** メニューの **[参照設定]** をクリックします。**[参照設定]** ダイアログ ボックスが表示され、アプリケーションから参照できる COM コンポーネントと登録済みの COM 呼び出し可能な .NET アセンブリを選択できるようになります。

5. [参照設定] ダイアログ ボックスで、[参照] をクリックします。
6. [参照の追加] ダイアログ ボックスで、System.tlb ファイルがあるディレクトリに移動します。このディレクトリは、ステップ 1.b で識別されたディレクトリと同じである必要があります。
7. 参照するアセンブリの .tlb ファイルをクリックし、[開く] をクリックします。この例では、[System.tlb] をクリックします。
8. [使用できる参照] ボックスで、参照する Visual Basic .NET ライブラリのチェック ボックスをオンにし、[OK] をクリックします。この例では、[System.dll] チェック ボックスをオンにします。

これらのステップを完了すると、アセンブリに含まれているすべてのインターフェイスとパブリック クラスを Visual Basic 6.0 プロジェクトから参照できるようになります。たとえば、System タイプ ライブラリを参照することで、System.WebClient クラスのインスタンスを作成できます。その後、次のサンプル プログラムに示すように、そのオブジェクトを通じて、downloadFile メソッドなどの任意の System.WebClient メソッドを呼び出すことができます。

```
Dim downloader As System.WebClient
Dim targetFile As String
Dim localFile As String

targetFile = _
    "http://www.msdn.com/library/toolbar/3.0/images/banners/" & _
    "msdn_masthead_ltr.gif"
Set downloader = New System.WebClient
downloader.downloadFile targetFile, App.Path & "\msdnLogo.gif"
```

メモ：Visual Basic 6.0 の開発環境内でこのサンプル プログラムを実行するには、システムに Visual Studio 6.0 Service Pack 6 がインストールされている必要があります。サービス パックがインストールされていない状態で Visual Basic 6.0 IDE 内でサンプル プログラムを実行すると、オートメーション ランタイム エラーが発生します。その場合、サンプル プログラムを正常に実行するには、アプリケーションの .exe ファイルを実行して、開発環境の外側で実行する必要があります。このサービス パックは、MSDN からダウンロードできます。

Visual Basic 6.0 アプリケーションから .NET の機能を呼び出すのは、非常に簡単です。参照するアセンブリが COM 呼び出し可能な場合は、Visual Basic 6.0 から非常に簡単にその機能にアクセスできます。

Visual Basic 6.0 プロジェクトから .NET アセンブリを簡単に参照できるものの、アプリケーション パフォーマンスの面で代価を払わなくてはなりません。Visual Basic 6.0 と Visual Basic .NET の間で実行される COM の相互運用機能は、相互運用しないコンポーネントでは不要な追加の手順が必要になります。これらの手順には、データのマーシャリング、呼び出し規約の調整、レジスタ保護、スレッド処理、例外処理のフレーム管理などがあります。そのため、Visual Basic 6.0 から Visual Basic .NET を呼び出す際のオーバーヘッドは、Visual Basic 6.0 から Visual Basic 6.0 を直接呼び出すのに比べて大きくなります。アプリケーションから呼び出す回

数や頻度を最小限に抑えると、追加のオーバーヘッドによる影響が少なくなります。パフォーマンスの改善についての詳細は、MSDN の『Improving .NET Application Performance and Scalability』の第 7 章「Improving Interop Performance」を参照してください。

.NET での相互運用ラッパーの作成

.NET Framework Class Library (FCL) に直接アクセスすると、機能面でできることが制限される場合があります。既に述べたように、Visual Basic 6.0 はオブジェクト指向ではありません。そのため、この方法を使ってインポートしたクラスの動作をカスタマイズできない場合があります。たとえば、利用したいクラスが抽象クラスであり、それを使用するにはサブクラスが必要な場合に特に当てはまります。また、すべての .NET アセンブリが COM 呼び出し可能なわけではありません。これにより、追加の手順を実行せずに Visual Basic 6.0 のコードから直接アクセスできるクラスが制限されます。そのような場合には、別のアプローチを利用できます。つまり、Visual Basic .NET で COM 呼び出し可能なラッパー オブジェクトを作成し、Visual Basic 6.0 からはラッパー オブジェクトを参照する方法です。ラッパーを作成するには、ラッパー クラスのタイプ ライブラリ (.tlb) ファイルを作成し、そのクラスを COM 相互運用機能に登録するなどの複数の手順が含まれます。

System.Text.StringBuilder クラスの Visual Basic .NET ラッパーを簡単に作成して、必要な文字列処理操作を行うことができます。また、ラッパーを利用すると、.NET の機能を正しく使用するために Visual Basic 6.0 から渡さなくてはならないパラメータを減らすことができます。次に示すコードは、Visual Basic .NET のラッパー クラスの例です (このクラスの定義を格納するファイルの名前は、このコード例の後に示す Visual Basic 6.0 コードに合わせるために、wrappers とする必要があります)。

```
Imports System.text

<ComClass(StringBuilderWrapper.ClassId, _
    StringBuilderWrapper.InterfaceId, StringBuilderWrapper.EventsId)> _
Public Class StringBuilderWrapper

#Region "COM GUIDs"
    Public Const ClassId As String = "963BBA03-8FA9-45F3-94FC-2E68D4940515"
    Public Const InterfaceId As String = "E7B75972-6A52-4E7D-96E1-9E7DBBBACBC7"
    Public Const EventsId As String = "48760051-DD7F-4EA1-B612-9CD7F2F1BAD3"
#End Region
    Private stringb As StringBuilder
    Public Sub New()
        stringb = New StringBuilder
    End Sub
    Public Sub SetValue(ByVal s As String)
        stringb = New StringBuilder(s)
    End Sub
    Public Sub Append(ByVal s As String)
        stringb.Append(s)
    End Sub
    Public Overrides Function ToString() As String
```

```
        Return stringb.ToString()
    End Function
End Class
```

クラスをコンパイルして COM 相互運用機能に登録すると、対応する .tlb ファイルを Visual Basic 6.0 プロジェクトの参照にインポートできるようになります。以下の Visual Basic 6.0 のコード例は、ラッパー クラスの機能にアクセスする方法を示しています。ラッパーが、StringBuilder クラスの非常に限られた機能だけをサポートしている点に注意してください。このラッパーが提供しているのは、Visual Basic 6.0 クライアントに必要な機能のみです。

```
Private Sub TestStringOperations()
    Dim s As New wrappers.StringBuilderWrapper
    s.SetValue "This is "
    s.Append "a test"
    MsgBox s.ToString
End Sub
```

Visual Basic .NET アセンブリを COM 呼び出し可能にするには、COM 相互運用機能に登録する必要があります。そのためには、.tlb ファイルを作成し、対応するエントリをシステムのレジストリに追加します。幸いにも、これは、適切なビルド構成オプションを設定することで、Visual Studio .NET IDE の中から簡単に実行できます。次の手順では、コンポーネントを COM 相互運用機能に登録する方法について詳しく説明します。

▶ Visual Basic .NET コンポーネントを COM 相互運用機能に登録するには

1. Visual Basic .NET ラッパー クラス用のプロジェクトを開きます。
2. [**プロジェクト**] メニューの [**プロパティ**] をクリックします。
3. [**構成プロパティ**] フォルダの [**ビルド**] をクリックします。
4. [COM **相互運用機能の登録**] チェック ボックスをオンにし、[OK] をクリックします。
5. [**ファイル**] メニューの [**すべてを保存**] をクリックし、プロジェクトの構成変更を保存します。
6. [**ビルド**] メニューの [**ソリューションのビルド**] をクリックします。

この手順に従うことで、Visual Basic .NET の DLL が作成され、COM 相互運用機能に登録されます。アセンブリの AssemblyName プロパティの値が、COM として登録される名前になります。このプロパティを変更するには、対応するプロジェクトのプロパティにアクセスし (ソリューション エクスプローラでプロジェクト名を右クリックします)、[**プロパティ**] をクリックします。[**共通プロパティ**] フォルダの [**全般**] をクリックします。

メモ：ラッパー クラスを登録する際には、名前が矛盾しないように注意してください。各アセンブリ名が一意であることを確認し、新しいアセンブリを登録する前に、必ずそのアセンブリの古いバージョンの登録を解除してください。

Visual Basic 6.0 で使用する Visual Basic .NET ラッパーを作成することは、COM 呼び出し可能としてビルドされているものを制限することなく、Visual Basic 6.0 から Visual Basic .NET アセンブリにアクセスする場合の最良の選択肢です。ラッパーを使用すると、Visual Basic .NET から使用可能なすべての .NET アセンブリにアクセスできます。また、ラッパーを使用すると、必要がない .NET アセンブリを何度も実行しなくて済むため、コードを適切にモジュール化することができます。さらに、固有のクラスを使用して FCL の機能をマスクし、インターフェイスを変えずにラッパーの実装を置き換えることもできます。これにより、クライアント コードとの互換性を心配をせずにラッパーのコードを進化させることができます。

あらゆる相互運用手順と同じように、.NET ラッパー クラスを開発する際の注意点があります。前述したように、欠点 (アセンブリの機能の一部にしかアクセスできないことがあるという点) の他に、留意すべき問題がいくつかあります。

- コードがうまくモジュール化されていない場合は、ラッパーを実装する際に追加作業が必要になることがあります。
- コードのリファクタリングは、投資に見合った効果が得られる可能性はあるものの、時間がかかります。
- ラッパーの設計と計画は重要になります。ラッパーで網羅する機能が多すぎたり少なすぎたりしないようにします。どちらの場合も、リスクを高めて、機能的なラッパーを作成するために必要な作業が増えます。

コマンド ラインからの登録

前述のように、ラッパー コンポーネントを作成した後は、すべての Visual Basic 6.0 アプリケーションにグローバルにアクセスできるように、ラッパー コンポーネントをシステムに登録する必要があります。Visual Studio .NET IDE の中からコンポーネントを登録するプロセスについては、前述しました。ただし、アセンブリは、.NET Framework と共に配布されているアセンブリ登録ツール (RegAsm.exe) を使用して、Windows のコマンド ラインからも登録できます。ここでは、そのための手順について説明します。

この例では、説明のために、仮のモジュール名を使用します。斜体の項目は、プロジェクトごとの適切な名前に置き換えます。

▶ コマンド ライン ツールを使用して .NET アセンブリを登録するには

1. Visual Basic .NET のコンパイル コマンド vbc.exe を使用して、ラッパー クラスをコンパイルします。

```
vbc assemblyinfo.vb sourcecode.vb /r:System.dll
   /target:library /out:wrappers.dll
```

/r オプションは、コンパイルで System.dll の参照を指定します。/target オプションは、ライブラリ (.dll) の生成を指定します。/out オプションは、出力ファイルの名前を指定します。

2. アセンブリを登録し、通常の COM オブジェクトとして参照できるようにするためのタイプ ライブラリを生成します。

```
regasm wrappers.dll /tlb:wrappers.tlb
```

/tlb オプションを使用して、生成するタイプ ライブラリ ファイルを指定します。

メモ： アセンブリを登録する際には、名前が矛盾しないように注意してください。各アセンブリ名が一意であることを確認し、新しいアセンブリを登録する前に、必ずそのアセンブリの古いバージョンの登録を解除してください。

アセンブリに対応する .tlb ファイル情報をシステム レジストリから削除する必要がある場合は、次の regasm コマンドを使用します。

```
regasm /u wrappers.dll
```

データ型のマーシャリング

マーシャリングは、クロス プラットフォームの呼び出しの実行を可能にする方法で、パラメータと戻り値のパックとアンパックを行う処理です。前述のように、Visual Basic 6.0 と Visual Basic .NET の間で相互運用を実現するのに重要な要素は COM です。異なるスレッド アパートメントにあるコンポーネントが対話する際にデータ型のマーシャリングが問題にならないのは、COM が存在しているためです。.NET の相互運用マーシャリングにより、マネージ メモリとアンマネージ メモリ内の異なるデータ型の間で対話が可能になります。相互運用マーシャリングは、同じ COM アパートメント内で関数が呼び出されるときに、CLR のマーシャリング サービスによる実行時に行われます。異なる COM アパートメントまたは異なるプロセス内のマネージ コードとアンマネージ コードの間で対話が行われるときは、相互運用マーシャラと COM マーシャラの両方が実行されます。

ほとんどの Visual Basic のデータ型は、マネージ メモリとアンマネージ メモリで同様に表されます。相互運用マーシャラは、これらの型を自動的に処理します。一部の型は、Visual Basic .NET にアップグレードするときに大きな違いがあります。あいまいなデータ型は、単一のマネージ型にマッピングされる複数のアンマネージ表現があるか、または文字列や配列のサイズなどの型情報を利用できない可能性があります。

カスタム マーシャリング

データ型の内部的な特性に関する仮定や依存があると、カスタム マーシャリングが必要になる場合があります。一般に当然とみなされる特性は、内部的な表現、要素のサイズ、要素の位置です。データ型に関して特別な仮定を行っているコードに依存した Visual Basic 6.0 アプリケーションの一部をアップグレードするときは、2 つのコード部分で .NET の相互運用機能を通じて対話が必要になる場合があります。このようなシナリオでは、アプリケーションのうち Visual Basic .NET にアップグレードした部分は、アップグレードしていないコンポーネントとの互換性を保つ必要があります。その場合、コンポーネント間で対話するデータ型のいくつかで、ユーザー定義マーシャリングが必要になります。

このような状況で、必要なデータ変換を行うために推奨されるアプローチは、これらの変換を行うためのコードすべてを含んだラッパー クラスを記述することです。このクラスは、マーシャリング ラッパー クラスと呼ばれ、古いインターフェイスと新しいインターフェイス間の橋渡しを行います。これにより、特定のインターフェイスを期待しているクライアントが、異なるインターフェイスを実装しているコンポーネントと連携できるようになります。

ほとんどの場合、マーシャリング ラッパー クラスを使用すれば、すべての対話と互換性の問題に対処できます。しかし、宣言的な、低レベルの手法を使用してマーシャリングを扱う必要がある場合には、カスタム マーシャラを使用できます。

パラメータまたは関数の戻り値に対してカスタム マーシャリングを適用するには、MarshalAsAttribute 属性を使用します。カスタム マーシャラはこの属性によって識別され、ICustomMarshaler インターフェイスを実装して、適切なラッパーを .NET CLR に提供します。.NET CLR は、カスタム マーシャラの MarshalNativeToManaged メソッドと MarshalManagedToNative メソッドを呼び出し、相互運用するコンポーネント間の呼び出しを扱うための適切なラッパーをアクティブにします。詳細については、MSDN の『.NET Framework Developer's Guide』の「Custom Marshaling」を参照してください。

次の例は、Visual Basic 6.0 にアップグレードしていないコンポーネントとの互換性を保つために、カスタム相互運用マーシャリングを使用しなくてはならない場合を示したものです。元のアプリケーションには、Converter という名前の COM クラスがあり、そこで Visual Basic 6.0 のデータ型 Currency と値をやり取りする関数 ConvertCurrency が定義されています。この関数は、財務計算を行うために、アプリケーションの残りの部分で使用されます。ConvertCurrency の元の宣言を以下に示します。

```
Public Function ConvertCurrency(amount As Currency, curFrom As String, _
                curTo As String) As Currency
    ConvertCurrency = amount * GetConversionFactor(curFrom, curTo)
End Function
```

Converter クラスは、次のようにしてクライアント コードで使用されます。

```
Private Sub Form_Load()
    Dim converter As New Utils.Converter
```

```
    Dim res As Currency
    Dim amount As Currency
    amount = GetInitialAmount()
    res = converter.ConvertCurrency(amount, "DOL", "EUR")
End Sub
```

次のクラスは、Converter クラスを Visual Basic .NET に自動的にアップグレードして得られた結果を基にしています。

```
<System.Runtime.InteropServices.ProgId("Converter_NET.Converter"), _
ClassInterface(ClassInterfaceType.AutoDual)> Public Class Converter
    Public Function ConvertCurrency(ByRef amount As Decimal, _
                    ByRef curFrom As String, ByRef curTo As String) As Decimal
        ConvertCurrency = amount * _
                          GetConversionFactor(curFrom, curTo)
    End Function
End Class
```

クラス メンバの宣言を Visual Basic 6.0 で使用できるように、クラス属性 ClassInterface(ClassInterfaceType.AutoDual) は手動で追加されています。アップグレードの結果得られたプロジェクトは、COM 相互運用機能に自動的に登録されます。このクラスを COM クライアントに公開するためには、ComClassAttribute 属性も使用しなくてはならない場合があります。上の例では、ClassInterfaceType.AutoDual 属性だけを使用して簡略化しています。

アプリケーションのアップグレード計画で、前述の Form_Load の例に示したように、アンマネージ COM クライアントが新しい Converter クラスにアクセスすると決定したとします。この計画に従えば、元の Converter クラスに対する参照は新しいマネージ クラスの参照に置き換える必要があります。しかし、その結果得られるアプリケーションをコンパイルすると、ConvertCurrency を実行したときに、"Compile error: Function or interface marked as restricted, or the function uses an Automation type not supported in Visual Basic" というエラー メッセージが表示されます。

問題は、COM クライアントが、Decimal データ型を、サポートされていない Variant 型として判断することです。この問題を解決するには、次に示すように、Visual Basic .NET コンポーネントの定義でカスタム マーシャリングを使用する必要があります。

```
<System.Runtime.InteropServices.ProgId("Converter_NET.Converter"), _
ClassInterface(ClassInterfaceType.AutoDual)> Public Class Converter
    Public Function ConvertCurrency( _
                    <MarshalAs(UnmanagedType.Currency)> ByRef amount As Decimal _
                    ByRef curFrom As String, ByRef curTo As String) _
                    As <MarshalAs(UnmanagedType.Currency)> Decimal
        ConvertCurrency = amount * _
                          GetConversionFactor(curFrom, curTo)
    End Function
End Class
```

MarshalAsAttribute 属性は、関数パラメータと関数の戻り値の 1 つが、COM の Decimal ではなく通貨データ型としてマーシャリングされる必要があることを示します。この調整を行うことで、Visual Basic 6.0 クライアントから引き続きコンポーネントを使用できます。

エラー管理

前述のように、.NET では構造化例外処理のアプローチを使用して、予期されるエラー状態と予期されないエラー状態の両方に対して通知と応答を行います。これは、Visual Basic 6.0 の場合には当てはまりません (OnError ハンドラを使用すれば、例外処理と同様と考えられる処理が可能になりますが、動作は同じではありません)。

Visual Basic .NET と Visual Basic 6.0 の間の相互運用は、COM ベースのメカニズムを使用して実現されるため、HRESULT という名前の、Visual Basic 6.0 コードと .NET アセンブリ間でやり取りされる固有の値があります。これは背後で行われていますが、たしかに存在しています。HRESULT の値を調べてエラー状態の存在を確認する処理は、CLR 自体によって行われます。したがって、この値は、どちらの側のクライアントコードにも直接公開されません。Visual Basic 6.0 では、Err.Number プロパティを通じてこの値にアクセスすることができます。Visual Basic .NET では、元の例外が標準の例外かどうかに応じてこの値の扱いが異なります。標準の例外の場合は、Visual Basic .NET クライアントは標準の .NET 例外を受け取ります。カスタム例外の場合、Visual Basic .NET クライアントは、HRESULT の値に応じて ErrorCode プロパティが設定された汎用例外を受け取ります。

Visual Basic 6.0 での .NET 例外のキャッチ

HRESULT によって、.NET アセンブリで例外が発生したことが通知されたときに、Visual Basic 6.0 で On Error 句が宣言および定義されていれば、それが実行されます。実際のエラー状態は、Visual Basic 6.0 に組み込みの Err オブジェクトを使用し、Description プロパティと Number プロパティを検査することで調べることができます。

メモ : 共通の .NET 例外から Visual Basic 6.0 のエラー コードへの定義済みのマッピングがあり、発生したシステム エラーの種類を決定するために使用できます。マッピング メカニズムの詳細については、MSDN の「HRESULTs and Exceptions」を参照してください。

たとえば、2 つの引数を使って除算を行う次に示すようなアセンブリを使用するとします。

Visual Basic .NET コンポーネント

```
Imports System.IO
Imports System.Runtime.InteropServices

<ComClass(DivisorTool.ClassId, DivisorTool.InterfaceId, DivisorTool.EventsId)> _
```

```
Public Class DivisorTool

#Region "COM GUIDs"
    Public Const ClassId As String = "D45B5767-62C9-4871-90DA-EFB56E2F8860"
    Public Const InterfaceId As String = "AA43EC7A-54B0-4E82-BC54-00087E27645C"
    Public Const EventsId As String = "9C4D24A8-F325-409F-8A00-62DE8B938F1D"
#End Region

    Public Sub New()
        MyBase.New()
    End Sub
    Public Function Divide(ByVal term As Double, _
                    ByVal divisor As Double) As Double
        Dim res As Decimal
        res = (term / divisor)
        Return res
    End Function
End Class
```

アセンブリをビルドし、登録して、Visual Basic 6.0 プロジェクト内で参照すると、次に示す例と同様の Visual Basic 6.0 コードを使用して例外情報にアクセスできるようになります。

Visual Basic 6.0 クライアント

```
Private Function Perform_Division() As Double
    On Error GoTo Error_Handler
    Dim div As New DivisorTool
    Perform_Division = div.Divide(10, 0)
    Exit Function

Error_Handler:
    MsgBox "Error : " & Err.Description, _
           vbOKOnly, "(code = " & CStr(Err.Number) & ")"
End Function
```

Divisor.Divide() 関数が除数 0 を指定して呼ばれた場合、System.OverflowException は .NET コンポーネントから生成されて、Visual Basic 6.0 クライアント コードに反映されます。そして、Visual Basic 6.0 内のオーバーフローのエラー状態を通常確認するときに確認します。これは既知のシステム エラーであるため、例外からエラー コードへの内部的なマッピング機構は、あるコード層と別のコード層の間でエラーに一貫性があることを確認します。

アプリケーションで定義された例外は若干異なります。.NET 例外の形で発生したアプリケーション エラーを大雑把に処理するには、Visual Basic 6.0 の Err オブジェクトの Number プロパティを調べ、この値に基づいて例外パスを選択する*必要があります*。この方法は、オーバーフローやゼロによる除算の例外など、標準の例外にも適用できます。これについて次の例で説明します。

コードの簡単な変更でさまざまなエラー状態を検出できる方法を示すために、ここで前述の例を使用します。あるビジネス ロジック上の理由から、値 3 以外のすべての除数が指定可能であり、3 以外のすべての項が指定可能だとします。それによって、ルーチンは、除数が 3 の場合や項が 3 の場合に異なる例外を発生するように記述されます。次のコード例にこれを示します。

Visual Basic .NET コンポーネント

```
Imports System.IO
Imports System.Runtime.InteropServices

<ComClass(DivisorTool.ClassId, DivisorTool.InterfaceId, DivisorTool.EventsId)> _
Public Class DivisorTool

#Region "COM GUIDs"
    Public Const ClassId As String = "D45B5767-62C9-4871-90DA-EFB56E2F8860"
    Public Const InterfaceId As String = "AA43EC7A-54B0-4E82-BC54-00087E27645C"
    Public Const EventsId As String = "9C4D24A8-F325-409F-8A00-62DE8B938F1D"
#End Region

    Public Sub New()
        MyBase.New()
    End Sub
    Public Function Divide(ByVal term As Double, _
                    ByVal divisor As Double) As Double
        If divisor = 3 Then
            Dim e As System.Runtime.InteropServices.COMException = _
            New System.Runtime.InteropServices.COMException( _
                "Bad Divisor", &H80000001)
            Throw e
        ElseIf term = 3 Then
            Dim e As System.Runtime.InteropServices.COMException = _
            New System.Runtime.InteropServices.COMException( _
                "Bad Term", &H80000002)
            Throw e
        Else
            Return (term / divisor)
        End If
    End Function
End Class
```

16 進数値 &H80000001 と &H80000002 は、Visual Basic 6.0 で例外として受信される HRESULT エラー コードに対応します。これらの値の下位 16 ビットは検出されたエラー コードに対応します。この場合のエラー コードは 1 と 2 です。

メモ：HRESULT 値に含まれている各フィールドの説明については、MSDN の「HRESULT」を参照してください。

Visual Basic 6.0 クライアント アプリケーションがこのクラスを参照し、Divide 関数を呼び出した場合、エラー ハンドラがその処理方法を決定するためには、発生したエラー コードを正しく判断する必要があります。これを以下のコード例に示します。

Visual Basic 6.0 クライアント

```
Private Function Perform_Division() As Double
    On Error GoTo Error_Handler
    Dim div As New DivisorTool
    Perform_Division = div.Divide(10, 0)
Exit Function

Error_Handler:
    If (Err.Number And &H7FFF) = 1 Then
        ' "bad divisor" エラーへの対処
    ElseIf (Err.Number And &H7FFF) = 2 Then
        ' "bad term" エラーへの対処
    Else
        ' それ以外の未知のエラー
        MsgBox "Error : " & Err.Description, _
            vbOKOnly, "(code = " & CStr(Err.Number) & ")"
    End If
End Function
```

Err.Number と 16 進数値 &H7FFF の間でビットごとの And 演算を行うことで、元のエラー コードを取り出す必要があります。.NET の相互運用機能によって COM クライアントに送信されるエラー コードの最初のビットがオンになり、元のエラー コードを得るために And 演算によってこのビットがオフになります。

上記のコード例に示したように、このアプローチによって相互運用で必要なコード量とコード全体の複雑さが増します。エラー ハンドラの複雑さが増すため、予測しない結果が生じたときに、どのエラー条件が原因なのかを正確にデバッグするのが難しくなります。

.NET アセンブリのドキュメントにどのエラー コード (Err.Number の値など) がどのエラー状態に対応するのかが記述されていないと、さらにリスクが高まります。これによって、.NET コンポーネントで発生した例外のいくつかを Visual Basic 6.0 クライアント コードで効果的に処理するのが難しくなります。

Visual Basic .NET での Visual Basic 6.0 で生成した OnError 条件のキャッチ

逆の場合、つまり Visual Basic 6.0 のコード モジュールで発生したエラー条件を .NET アセンブリで検出して対処する方法も分析する必要があります。ここでも COM が関係しており、共通のシステム エラーを処理するという点ではいくつかの利点があります。

共通のシステム エラーでは、マーシャリング層が自動的にこれらのエラー状態を適切な .NET 例外に変換します。つまり、Visual Basic 6.0 のサブルーチンを .NET から呼び出し、try/catch/finally ブロックを使用して、コード モジュールが生成した通常のシステム エラーを検出することができます。

たとえば、前のセクションに示した DivisorTool の Visual Basic 6.0 バージョンを考えてみます。このコンポーネントを .NET アセンブリで使用し、除数に 0 を指定して Divide 関数を呼び出すと、Visual Basic 6.0 コンポーネントが **ゼロによる除算**エラーを生成します。このエラーは、.NET 側で自動的に System.DivideByZeroException に変換されるため、クライアント コード内で簡単にシステム例外を確認し、適切にエラーを処理することができます。

この方法のコード例を以下に示します。

Visual Basic 6.0 コンポーネント

```
Public Function Divide(ByVal term As Double, ByVal divisor As Double) As Double
    Divide = term / divisor
End Function
```

Visual Basic .NET クライアント

```
Private Function Perform_Division( _
    ByVal term As Double, _
    ByVal divisor As Double) As Double

    Dim x As Double
    Dim div As VB6Module.DivisorTool
    div = New VB6Module.DivisorTool
    Try
        Return div.Divide(term, divisor)
    Catch e As System.DivideByZeroException
        System.Windows.Forms.MessageBox.Show("Division by 0")
        Return 0
    End Try
End Function
```

Visual Basic .NET のコード例に示すように、Visual Basic 6.0 の関数 Divide の実行は、try/catch ブロックで囲まれており、System.DivideByZeroException に対して Catch 句が指定されています。

前述のように、アプリケーション固有のエラーは、一般的なエラーの処理よりも若干複雑です。Visual Basic 6.0 のアプリケーション エラーは Visual Basic .NET コード内で InteropServices.COMException として生成されます。このコード例は上の例を拡張したもので、除数または項に 3 が指定された場合のアプリケーション例外を追加しています。

Visual Basic 6.0 コンポーネント

```
Public Function Divide(ByVal term As Double, ByVal divisor As Double) As Double
    If divisor = 3 Then
        Err.Raise 1, "Project1", "Bad divisor", "", ""
    ElseIf term = 3 Then
        Err.Raise 2, "Project1", "Bad term", "", ""
    Else
```

```
        Divide = term / divisor
    End If
End Function
```

Visual Basic .NET クライアント

```
Private Function Perform_Division( _
    ByVal term As Double, _
    ByVal divisor As Double) As Double

    Dim x As Double
    Dim div As VB6Module.DivisorTool
    div = New VB6Module.DivisorTool
    Try
        Return div.Divide(term, divisor)
    Catch e As System.DivideByZeroException
        System.Windows.Forms.MessageBox.Show("Division by 0")
        Return 0
    Catch e As System.Runtime.InteropServices.COMException
        System.Windows.Forms.MessageBox.Show("App Error")
    End Try
End Function
```

上のコード例の Visual Basic .NET クライアントコードでは、2 つの Visual Basic 6.0 アプリケーション エラーを区別していないため、エラー条件を適切に処理することができません。上の例では、"bad divisor error" (エラー番号 1) と "bad term error" (エラー番号 2) を区別できません。

エラー状態を区別するには、COMException.ErrorCode の値を使用します。これにより、Visual Basic .NET クライアント コードが、Visual Basic 6.0 コンポーネントのコードで Err.Raise ステートメントで指定されたエラー コード値を使って、エラー条件を区別するためのメカニズムが提供されます。

次のコード例は、Visual Basic .NET クライアント コードの拡張バージョンです。この例では、エラー コードの値を調べ、アプリケーション固有のエラー状態に適切に対処します。

Visual Basic .NET クライアント (個別のエラー処理を行う例)

```
Private Function Perform_Division( _
    ByVal term As Double, _
    ByVal divisor As Double) As Double

    Dim x As Double
    Dim div As VB6Module.DivisorTool
    div = New VB6Module.DivisorTool
    Try
        Return div.Divide(term, divisor)
    Catch e As System.DivideByZeroException
        System.Windows.Forms.MessageBox.Show("Division by 0")
```

```
        Return 0
    Catch e As System.Runtime.InteropServices.COMException
        If (e.ErrorCode And &HFFFF) = 1 Then
            ' "Bad Divisor" エラーへの対処
        ElseIf (e.ErrorCode And &HFFFF) = 2 Then
            ' "Bad Term" エラーへの対処
        Else
            System.Windows.Forms.MessageBox.Show("App Error")
        End If
    End Try
End Function
```

上のコード例に示すように、例外オブジェクトから実際のエラー コードを取り出すには、ビット演算を行う必要があります。ErrorCode と 16 進数値 FFFF (&HFFFF) の間でビットごとの And 演算を行い、ErrorCode 値の下位 16 ビットを確認します。これらのビットには、Visual Basic 6.0 の Err.Raise 呼び出しで生成された実際のエラー コードが格納されています。下位 16 ビットを調べることで、Visual Basic 6.0 コンポーネントが生成した正確なエラーを特定し、アプリケーション固有のエラー状態に適切に反応することができます。このようにして、エラー処理に関しても、.NET 境界を越えてビジネス ロジックの一貫性を保つことができます。

COM イベントのシンク

Visual Basic 6.0 からのイベントを COM クライアントでシンクするには、RaiseEvent ステートメントを使用します。また、これらのイベントを、COM コンポーネントが生成したかのように .NET コードでキャッチして処理することができます。

次の Visual Basic 6.0 のコード例は、イベントの生成方法を示します。また、その後の Visual Basic .NET のコード例は、イベントをキャッチして処理する方法を示します。Visual Basic 6.0 のイベント プロデューサ コードは、DoDivide 関数が呼び出されると必ず Divide イベントを生成し、DoMultiply 関数が呼び出されると Multiply イベントを生成します。Visual Basic .NET で記述されたコンシューマ側のコードは、イベント ハンドラの OnDivide と OnMultiply によって、発生したイベントをキャッチして処理します。

Visual Basic 6.0 のイベント プロデューサ

```
Option Explicit

Public Event Divide(ByVal x As Double, ByVal y As Double)
Public Event Multiply(ByVal x As Double, ByVal y As Double)

Public Function DoDivide(x As Double, y As Double) As Double
    RaiseEvent Divide(x, y)
    DoDivide = x / y
End Function

Public Function DoMultiply(x As Double, y As Double) As Double
    RaiseEvent Multiply(x, y)
```

```
    DoMultiply = x * y
End Function
```

Visual Basic .NET のイベントコンシューマ

```
Option Explicit
Option Strict

Imports System
Imports System.Runtime.InteropServices

Namespace EventConsumer

    Public Class Consumer

        Dim WithEvents myProducer As New VB6Module.Producer

        Private Sub ConsumeTest()
            myProducer.DoDivide(1,3)
            myProducer.DoMultiply(5,7)
        End Sub

        Private Sub OnDivide(x As Double, y As Double) _
            Handles myProducer.Divide
            Console.WriteLine("Division: {0}", x / y)
        End Sub

        Private Sub OnMultiply(x As Double, y As Double) _
            Handles myProducer.Multiply
            Console.WriteLine("Multiplication: {0}", x * y)
        End Sub

    End Class
End Namespace
```

同様に、Visual Basic .NET コンポーネントで Delegate ディレクティブを使用してイベントを生成することができます。これらのイベントは、Visual Basic 6.0 クライアント コード内で WithEvents ディレクティブを使用してキャッチすることができます。これを示すのが以下のコード例です。このコード例では、イベント プロデューサ コードは上の例のイベント プロデューサの Visual Basic .NET バージョンであり、イベント コンシューマ コードは上の例のイベントコンシューマの Visual Basic 6.0 バージョンです。

Visual Basic .NET のイベント・プロデューサ

```
Option Explicit On
Option Strict On

Namespace EventProducer
    <ComClass(Producer.ClassId, Producer.InterfaceId, Producer.EventsId)> _
    Public Class Producer
```

```
#Region "COM GUIDs"
        Public Const ClassId As String = "591AC4AC-949A-45E6-AE5F-FFB5B7635E9B"
        Public Const InterfaceId As String = _
        "D3DC65BE-F49E-4356-8308-80B9B6198139"
        Public Const EventsId As String = "58D8EC7F-821A-453A-BFC0-6AF8A8D43D1A"
#End Region

        Public Event Divide(ByVal x As Double, ByVal y As Double)
        Public Event Multiply(ByVal x As Double, ByVal y As Double)

        Public Sub New()
            MyBase.New()
        End Sub

        Public Sub SendDivide(ByVal x As Double, ByVal y As Double)
            RaiseEvent Divide(x, y)
        End Sub

        Public Sub SendMultiply(ByVal x As Double, ByVal y As Double)
            RaiseEvent Multiply(x, y)
        End Sub

        ' このクラスにカスタム関数を追加し、SendXXXX を呼び出してイベント通知を送信します。
    End Class
End Namespace
```

Visual Basic 6.0 のイベントコンシューマ

```
Public WithEvents myProd As Producer

Private Sub Class_Initialize()
    Set myProd = New Producer
End Sub

' イベント名とシグネチャでイベントとメソッドを一致させます。
Private Sub myProd_Divide(ByVal x As Double, ByVal y As Double)
    MsgBox "Division: " & (x / y)
End Sub

Private Sub myProd_Multiply(ByVal x As Double, ByVal y As Double)
    MsgBox "Multiplication: " & (x * y)
End Sub

Public Sub ProducerTest()
    myProd.SendDivide 1, 2
    myProd.SendMultiply 3, 5
End Sub
```

上のコード例に示すように、SendMultiply 関数と SendDivide 関数を定義した後で、必要に応じてイベント受信待ち状態の COM クライアントに対してイベントを生成できるようになります。この規則は、Visual Basic 6.0

クライアントにも適用されます。これは、Visual Basic 6.0 が、COM の相互運用機能を通じて Visual Basic .NET のイベントプロデューサに接続するためです。

Visual Basic 6.0 で記述されたコンポーネントと Visual Basic .NET で記述されたコンポーネントの間には CLR が存在するため、各ラン タイム環境の間でのイベント処理は互いに透過的です。イベントをキャッチする側のコンポーネントにとっては、イベントが .NET のマネージ コンポーネントで生成されたものであっても、Visual Basic 6.0 のアンマネージ コンポーネントで生成されたものであってもかまいません。CLR の COM 相互運用レイヤで詳細を処理するため、アップグレードが簡単になります。

OLE オートメーション呼び出しの同期

Visual Basic 6.0 の App オブジェクトでは、プロパティ グループを提供します。このグループを使用して、OLE オートメーション サーバーとの対話と同期に必要なパラメータをそのメソッドの 1 つを実行している間に指定することができます。

最初のプロパティセットは、失敗したオートメーション呼び出し要求をアプリケーションが再試行する回数に関係します。指定された回数だけ試行すると、カスタマイズ可能なダイアログ ボックスがアプリケーション内で自動的に表示されます。このダイアログ ボックスは、OLE サーバーのビジー状態をユーザーに通知します。この動作は、プロパティ OleServerBusyMsgText、OleServerBusyMsgTitle、OleServerBusyRaiseError、および OleServerBusyTimeout で制御できます。

2 番目のプロパティセットは、OLE オートメーション要求の完了をアプリケーションが待つ時間に関係します。このプロパティ グループは、上のグループと同様に機能し、OleRequestPendingMsgText、OleRequestPendingMsgTitle、および OleRequestPendingTimeout を含みます。

Visual Basic .NET では、これらのプロパティによって提供される機能に相当するものがないため、現在使用できるメカニズムを使って実装し直す必要があります。

この機能の目的は、ユーザーがアプリケーションの状態を知ることができる、応答性に優れたアプリケーションを作成することです。Visual Basic .NET で同等の機能を提供する 1 つの方法は、.NET のマルチスレッド機能を使用することです。

次のコード例は、Visual Basic .NET を使用して、OLE オートメーション サーバーの非同期呼び出しを行う方法を示します。これによりアプリケーションは、アプリケーションと保留中の要求の状態に関する有益な情報を待ち、ユーザーに提供できます。

Visual Basic 6.0 コンポーネント

```
Private Sub Form_Load()
    App.OleRequestPendingMsgText = "Pending"
    App.OleRequestPendingTimeout = 500
    App.OleServerBusyMsgText = "Busy"
    App.OleServerBusyRaiseError = False
End Sub
```

```
Private Sub Command1_Click()
    Dim c As New ComponentExe.ClassComExe
' このメソッドは実行にかなり時間がかかります。
    c.myMethod
    MsgBox "myMethod ComExe finished"
End Sub
```

上の Form_Load メソッドは、アプリケーションが ClassComExe.myMethod メソッドの実行を待っているときに表示されるダイアログ ボックスに影響を与えるカスタム パラメータを設定します。Visual Basic .NET では、アプリケーションがまだ実行中であるかどうかをユーザーに知らせるために、別の方法でこの機能を実装する必要があります。

次のコード例では、別のスレッドで実行される中間的なクラスを使用して ClassComExe.myMethod メソッドを実行し、メイン アプリケーションはユーザーに応答し続けることができるようになっています。非同期呼び出しのための補助クラスが、処理の完了をアプリケーションに通知する方法に注意してください。補助クラスはイベントを生成し、メイン アプリケーションがそれを処理します。

Visual Basic .NET コンポーネント

```
Dim WithEvents AsynchCallerObj As AsynchCaller

Class AsynchCaller
    Public Event ThreadDone()
    Sub DoCall()
        Dim c As New ComponentExe.ClassComExe
        c.foo()
        RaiseEvent ThreadDone()
    End Sub
End Class

Private Sub Form1_Load(ByVal eventSender As System.Object, _
ByVal eventArgs As System.EventArgs) Handles MyBase.Load
    ' コードはサポートされていないためユーザーによって削除されました。
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
    AsynchCallerObj = New AsynchCaller
    Dim Thread As New System.Threading.Thread(AddressOf AsynchCallerObj.DoCall)
    Thread.Start()
    ' ここで進行状況モニタを起動し、プロセスの状態についてユーザーに通知することも
    ' 可能です。
End Sub

Sub AsynchCallerDone() Handles AsynchCallerObj.ThreadDone
    MsgBox("foo ComExe finished")
End Sub
```

リソースの処理

共通言語ランタイム (CLR) は、リソース処理メカニズムを備えており、プログラマが手動でメモリ管理を行わなくても済むようになっています。このメカニズムはガベージ コレクションと呼ばれ、.NET の不可欠な部分となっています。

Visual Basic .NET オブジェクトと Visual Basic 6.0 オブジェクトは、New キーワードで作成します。通常、オブジェクトを使用する前には、初期化が必要です。初期化には、ファイルのオープン、データベースへの接続、内部構造のメモリのようなその他のシステム リソースの取得などのタスクが含まれます。Visual Basic .NET では、コンストラクタと呼ばれるプロシージャを使用して、新しいオブジェクトの初期化を制御します。

オブジェクトの有効期間は、オブジェクトがスコープを離れ、CLR によって解放されると終了します。Visual Basic .NET では、デストラクタと呼ばれるプロシージャを使用して、システム リソースの解放を制御します。コンストラクタとデストラクタで、システム リソースを効率的に使用する、堅牢で予測可能なクラスの作成を支援します。

Visual Basic .NET のコンストラクタとデストラクタ

Visual Basic .NET の Sub New プロシージャと Sub Finalize プロシージャは、それぞれコンストラクタ メソッドとデストラクタ メソッドです。これは、Visual Basic 6.0 で使用されている Class_Initialize メソッドと Class_Terminate メソッドに代わるものです。Class_Initialize と違い、Sub New コンストラクタは、クラスの作成時に一度だけ実行でき、同じクラスまたは派生クラスいずれかのコンストラクタの最初の行でしか明示的に呼び出すことができません。

オブジェクトを解放する前に、CLR は、Sub Finalize プロシージャが定義されているオブジェクトの Finalize メソッドを自動的に呼び出します。Finalize メソッドには、オブジェクトが破棄される直前に実行する必要がある、ファイルのクローズ、状態情報の保存、アンマネージ コンポーネント (たとえば、Visual Basic 6.0 で作成された COM コンポーネント) の解放などのコードを含めることができます。Sub Finalize の実行には時間がかかるため、COM コンポーネントが含まれている場合など、オブジェクトを明示的に解放する必要がある場合にだけ Sub Finalize メソッドを定義します。オブジェクト参照に Nothing が設定されるとすぐに実行される Visual Basic 6.0 の Class_Terminate とは異なり、通常は、オブジェクトがスコープを失ってから Visual Basic .NET が Finalize デストラクタを呼び出すまでには遅延があります。

Visual Basic .NET では、Dispose と呼ばれる 2 つ目の種類のデストラクタが使用可能です。これは、割り当てられたリソースをすぐに解放する必要がある場合に明示的に呼び出します。すべての .NET クラスは、割り当てられたアンマネージ リソースを解放するために、IDisposable インターフェイスを実装できます。このインターフェイスでは、コンポーネントのインスタンスが不要になったときにコンポーネントの利用者が呼び出す Dispose メソッドが定義されます。Dispose の実装には、割り当て済みのメモリ、ファイル ハンドル、データベース接続などの限られたシステム リソースをすぐに解放するためのコードを含めることができます。

ガベージ コレクション

.NET では、実行中のコードからコンポーネントにアクセスできなくなったときにコンポーネントのデストラクタを実行するのは、ガベージ コレクション メカニズムの役目です。この条件は、コンポーネントに対するすべての参照が解放されるか、実行中のすべてのコードと無関係なオブジェクトに参照が属している場合に満たされます。

.NET Framework は、参照トレース ガベージ コレクションと呼ばれる手法を使用して、使用されなくなったリソースを定期的に解放します (ガベージ コレクションの詳細については、MSDN の「Object Lifetime: How Objects Are Created and Destroyed」を参照してください)。Visual Basic 6.0 では、同様の目的で参照カウントと呼ばれるメカニズムを使用しています。これらのメカニズムには主に以下の違いがあり、相互運用を正常に実現させるにはこれらの違いを考慮する必要があります。

- **非決定性オブジェクトの有効期限。** CLR は、システム リソースが一定のレベルまで減少すると、早くオブジェクトを破棄するようになり、逆に、豊富になると、オブジェクトをゆっくりと破棄するようになります。このスキームの結果、Visual Basic .NET では、実際にいつオブジェクトが破棄されリソースが解放されるかを事前に決定することはできません。この場合、.NET オブジェクトの有効期限は非決定的であると言います。Visual Basic .NET では、オブジェクトがスコープを外れてすぐに Finalize デストラクタが実行されない場合があるこという共通認識がある間は、この動作が開発プロセスに影響を与えることはありません。
- **Nothing 値の代入。** Visual Basic 6.0 では、対応するオブジェクト変数のどれかに Nothing を代入することで、プログラマがオブジェクトの参照カウントに影響を与えることがあります。参照カウントがゼロになると、オブジェクト リソースはすぐに解放されます。Visual Basic .NET では、Nothing 値を代入しても、すぐに解放が実行されることはありません。すぐに解放する必要がある場合は、Dispose メソッドを実装して、必要な時に明示的に呼び出す必要があります。

メモ：System.Runtime.InteropServices.Marshal クラスでは、COM オブジェクトに関連付けられている参照カウントを操作し、オブジェクトの有効期限を手動で決定するためのメソッドが提供されています。これらのメソッドには、AddRef、Release、および ReleaseComObject があります。ReleaseComObject メソッドは、.NET コンポーネント内で割り当てられたアンマネージ リソースを解放するためによく利用されます。これにより、ガベージ コレクタがリソースを消去するのを待たずに、COM オブジェクトが強制的に解放されます。

まとめ

部分的で段階的なアップグレード方法により、アプリケーションをアップグレードする際のリスクを最小限に抑えつつ、新しい言語機能や技術の恩恵を受けることができます。これらの方法を実施するにあたり、Visual Basic 6.0 コンポーネントが新しい Visual Basic .NET コンポーネントと対話できることが重要な要件となります。

Visual Basic .NET の COM 相互運用機能を使用すると、部分的なアップグレードが簡単に実現できます。複雑な Visual Basic 6.0 コンポーネントや、アップグレード費用がかかりすぎるコンポーネントは Visual Basic 6.0 のままにし、より簡単なコンポーネントを Visual Basic .NET にアップグレードします。COM 相互運用機能により、古いコンポーネントと新しいコンポーネントが互いにシームレスに対話することができ、Visual Basic .NET の新機能と共に Visual Basic 6.0 コンポーネントを活用することができます。この章で説明した手法は、この相互運用の実現に役立ちます。

詳細情報

MSDN から Microsoft Service Pack 6 for Visual Basic 6.0 をダウンロードするには、次の URL を参照してください。
http://msdn.microsoft.com/vstudio/downloads/updates/sp/vs6/sp6/default.aspx

パフォーマンスの改善についての詳細は、MSDN の『Improving .NET Application Performance and Scalability』の第 7 章「Improving Interop Performance」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt07.asp です。

カスタム マーシャリングの詳細については、MSDN の『.NET Framework Developer's Guide』の「Custom Marshaling」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconCustomMarshaling.asp です。

共通の .NET 例外から Visual Basic 6.0 のエラー コードへのマッピングの詳細については、MSDN の「HRESULTs and Exceptions」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconhresultsexceptions.asp です。

HRESULT 値に含まれている各フィールドの説明については、MSDN の「HRESULT」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/APISP/html/sp__mapi1book_c_type_hresult.asp です。

ガベージ コレクションの詳細については、MSDN の「Visual Basic Language Concept」の「Object Lifetime: How Objects Are Created and Destroyed」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vaconFinalizeDestructors.asp です。

15

MTS アプリケーションと COM+ アプリケーションのアップグレード

Microsoft Transaction Server (MTS) は、主にマルチユーザー サーバー アプリケーションでの使用を目的とした技術です。MTS は、分散トランザクションのサポートに加えて、COM のセキュリティ モデルを拡張して、拡張性サービス、接続管理、スレッド プール、および管理構造を提供しています。これによって、拡張性の高いサーバー アプリケーションを構築して配置するための簡単な方法が利用できるようになり、中間層で動作する COM オブジェクトに適した分散対応環境が使用可能になります。Microsoft Transaction Server がインストールされている場合は、中間層のコンポーネントとオブジェクトは、トランザクションに関係があるかどうかにかかわらず、MTS 環境内で実行することをお勧めします。

COM+ は、COM (コンポーネント オブジェクト モデル) と MTS インフラストラクチャを拡張したものです。以前のアプリケーションでは、あるサービスは COM に、他のサービスは MTS に依存していましたが、これら 2 つのプログラミング モデルを統合することで、アプリケーションの開発、配置、デバッグ、メンテナンスが統一され、分散アプリケーションの開発が容易になります。

この章では、Visual Basic 6.0 の COM+ アプリケーションと MTS アプリケーションを Visual Basic .NET にアップグレードする方法について説明します。

Visual Basic 6.0 での MTS/COM+ の使用

Visual Basic 6.0 と MTS を使用すると、n 層アプリケーションを構築できます。Visual Basic 6.0 を使用して、MTS の管理の下に中間層サーバーで実行される COM DLL コンポーネントを開発します。クライアントがこれらの COM DLL を呼び出すと、その要求が Windows オペレーティング システムによって自動的に MTS に転送されます。アップグレード作業に関連する MTS のサービスには、このほかに次のようなものがあります。

- コンポーネントトランザクション
- オブジェクトブローカリング

- リソースプール
- ジャストインタイム アクティベーション
- 管理

アップグレード方法としては、相互運用機能を通じてアクセスするプロキシ COM クラスを使用することが考えられます。この方法は、あるシナリオには適していますが、プロキシ クラスはいつも使用可能というわけではなく、適していないこともあります。場合によっては、代わりに .NET Framework を使用して、MTS サービスや COM+ サービスを実装し直す必要があります。

.NET Framework の System.EnterpriseServices 名前空間には、同様の機能を提供し、COM+ が使用するメカニズムへのインターフェイスを提供するクラスがあります。これにより、COM+ サービスへのアクセスを維持しつつ、.NET オブジェクトを使用したエンタープライズ アプリケーションの構築が可能になります。

以下に、Visual Basic 6.0 を使用して記述され、MTSTransactionMode プロパティが RequiresTransaction に設定された COM+ コンポーネントのコード例を示します。

```
Public Sub transferfunds(acc1 As String, acc2 As String, _
    amount As Long)

    On Error GoTo ErrHandler

    withdrawFunds acc1, amount
    depositFunds acc2, amount
    GetObjectContext.SetComplete

    Exit Sub
ErrHandler:
    GetObjectContext.SetAbort
    Err.Raise Err.Number, Err.Source, Err.Description, _
        Err.HelpFile, Err.HelpContext
End Sub

Private Sub withdrawFunds(acc As String, amount As Long)
    ' 引き出しの実行...
End Sub

Private Sub depositFunds(acc As String, amount As Long)
    ' 預け入れの実行...
End Sub
```

Visual Basic .NET での COM+ の使用

.NET Framework は、分散トランザクション、コンポーネントのインスタンス管理、ロール ベース セキュリティ、キュー コンポーネント、およびイベントを提供するにあたり、COM+ に依存しています。
System.EnterpriseServices 名前空間では、COM+ サービスを使用するために必要なすべての型、クラス、オブジェクトが提供されており、実質的に COM+ が .NET Framework の一部になっています。

COM+ サービスを使用する .NET コンポーネントは、*サービス コンポーネント*と呼ばれます。以下のコード例は、Visual Basic .NET のクラスとしてのサービス コンポーネントを示しており、2 つの銀行口座の間で資金を振り替えます。前の例と同じように見えますが、トランザクション サービスを使用していないため、機能的には対応していない点に注意してください。不足している機能は少し後で追加します。

```
Imports System.EnterpriseServices

Public Class MyComponent
    Inherits ServicedComponent

    Public Sub transferfunds(ByRef acc1 As String, _
        ByRef acc2 As String, ByRef amount As Integer)

        On Error GoTo ErrHandler

        withdrawFunds(acc1, amount)
        depositFunds(acc2, amount)

        Exit Sub
    ErrHandler:
        Err.Raise(Err.Number, Err.Source, Err.Description, _
           Err.HelpFile, Err.HelpContext)
    End Sub

    Private Sub withdrawFunds(ByRef acc As String, _
        ByRef amount As Integer)
        ' 引き出しの実行...
    End Sub

    Private Sub depositFunds(ByRef acc As String, _
         ByRef amount As Integer)
        ' 預け入れの実行...
    End Sub

End Class
```

このクラスを構築するには、System.EnterpriseServices 名前空間への参照を追加する必要がある点に注意してください。

サービス コンポーネントを作成する際には、次の点を考慮してください。

- サービス コンポーネントのインスタンスが不要になった場合は、そのコンポーネントが使用していたリソースを解放するために、クライアントは Dispose() メソッドを呼び出す必要があります。
- サービス コンポーネントは、既定のコンストラクタ (引数のないコンストラクタ) を含んでいる必要があります。
- 静的メソッドは、リモートから呼び出したり、クラスの特定のインスタンスに関連付けることはできません。このため、トランザクションやオブジェクト プールなどの COM+ サービスを利用することはできません。

- トランザクションや Windows 認証などの、コンピュータ間でやり取りされるサービスは、DCOM を使用した場合にのみ実現できます。
- 実行時には、COM+ アプリケーションを実行しているユーザーは、アンマネージ コードを実行するための権限を持っている必要があります。

以下のコード例は、前のセクションで示した Visual Basic 6.0 のコード例をいくらか複雑にしたものです。この例では、コードは .NET サービス コンポーネントにアップグレードされています。

```
Option Strict Off
Option Explicit On
Imports System.EnterpriseServices

<Transaction(TransactionOption.Required), EventTrackingEnabled()> _
Public Class Account
    Inherits ServicedComponent

    Public Sub transferfunds(ByRef acc1 As String, _
        ByRef acc2 As String, ByRef amount As Integer)
        On Error GoTo ErrHandler

        withdrawFunds(acc1, amount)
        depositFunds(acc2, amount)
        ContextUtil.SetComplete()

        Exit Sub
    ErrHandler:
        ContextUtil.SetAbort()
        Err.Raise(Err.Number, Err.Source, Err.Description, _
           Err.HelpFile, Err.HelpContext)
    End Sub

    Private Sub withdrawFunds(ByRef acc As String, _
        ByRef amount As Integer)
        ' 引き出しの実行...
    End Sub

    Private Sub depositFunds(ByRef acc As String, _
         ByRef amount As Integer)
        ' 預け入れの実行...
    End Sub
End Class
```

ここで最初に重要な点は、System.EnterpriseServices 名前空間への参照です。この参照により、エンタープライズ アプリケーションのためのインフラストラクチャが利用可能になり、.NET オブジェクトが COM+ サービスにアクセスできるようになります。

次に、このクラスを COM+ アプリケーションとして使用できるように、ServiceComponent クラスを拡張してあります。System.EnterpriseServices 名前空間にある ServiceComponent クラスは、COM+ サービスを使用するすべてのクラスの基本オブジェクトです。

最後に、Transaction 属性と EventTrackingEnabled 属性は、それぞれ、Account クラスが使用できるトランザクションの種類 (この例ではトランザクションは Required です) を示し、Account クラスのイベント トラッキングを使用可能にしています。

ServicedComponent オブジェクトを配置して使用するためには、対応するコンポーネントに、アセンブリの ID (必須の単純なテキスト名とバージョン番号、およびオプションのカルチャ情報) と公開キーおよびデジタル署名から成る厳密な名前で署名する必要があります。この作業を行うには、厳密名ツール (Sn.exe) を使用します。これは、.NET Framework に添付されているコマンド ライン ツールであり、ファイル名の拡張子が .snk のキー ペア ファイルを作成します。生成されたファイルは、コンポーネントのソース コード フォルダにコピーする必要があります。この章では、必要な手順にはこの重要なステップが含まれていますが、厳密名ツールの詳細については、MSDN の『.NET Framework Tools』の「Strong Name Tool (Sn.exe)」を参照してください。

ServicedComponent クラスを使用する際には、以下のコード例に示すようなアセンブリ属性を使用して、追加の構成オプションを指定する必要があります。

```
<Assembly: AssemblyKeyFile("MyAssembly.snk")>
```

AssemblyKeyFileAttribute クラスは、厳密名ツール (Sn.exe) を使用して生成され、厳密な名前を生成するために使われるキー ペアが格納されたファイルを指定するために使用します。これらの属性はクラス ファイルに追加することもできますが、アセンブリ属性の格納には、すべての Visual Basic .NET プロジェクトに含まれている AssemblyInfo.vb ファイルを指定することをお勧めします。

一般的な注意点

ここでは、サービス、アプリケーション、配置、それらが Visual Basic .NET へのアップグレードによってどのような影響を受けるかなど、COM+ に関係する一般的な問題点のいくつかについて説明します。

Visual Basic 2005 の場合 :

Visual Studio 2005 に含まれる Visual Basic アップグレード ウィザードは、Visual Basic 6.0 の Microsoft Transaction Server プロジェクトと COM+ サービス プロジェクトを、Visual Basic 2005 の EnterpriseServices プロジェクトにアップグレードします。この章の「COM+ イベント」と「メッセージ キューとキュー コンポーネント」で説明している作業を除き、この章で説明されているほとんどの作業は、アップグレード ウィザードで自動的に実行されます。自動的に実行されない作業は、手作業でのアップグレードが必要となります。

また、Visual Basic 2005 アップグレード ウィザードを使用した後で、アプリケーションを配置して構成するための作業も手作業で行う必要があります。

COM+ アプリケーションの種類

以下に、4 つの基本的な COM+ アプリケーションの種類を示します。

- **ライブラリ アプリケーション。**ライブラリ アプリケーションは、作成されるクライアントのプロセスで動作します。ライブラリ アプリケーションでは、ロール ベース セキュリティは使用できますが、リモート アクセスやキュー コンポーネントはサポートしていません。
- **サーバー アプリケーション。**サーバー アプリケーションは、COM+ が作成する専用のプロセスで動作します。ライブラリ アプリケーションとサーバー アプリケーションのパフォーマンスの違いや制約は、プロセス間通信が主な原因です。サーバー アプリケーションは、すべての COM+ サービスをサポートできます。
- **アプリケーション プロキシ。**アプリケーション プロキシは、クライアントがサーバー アプリケーションにリモートからアクセスできるようにするための登録情報が含まれるファイル セットです。
- **プレインストールされた COM+ アプリケーション。**COM+ には、内部機能を処理するプレインストールされたアプリケーションのセットが含まれています。プレインストールされたアプリケーションは、**コンポーネント サービス**管理ツールの **COM+ アプリケーション** フォルダに一覧表示されますが、変更や削除はできません。次のアプリケーションが含まれています。
  - .NET Utilities
  - Analyzer Control Publisher Application
  - COM+ Explorer
  - COM+ QC Dead Letter Queue Listener
  - COM+ Utilities
  - IIS In-Process Applications
  - IIS Out-Of-Process Pooled Applications
  - System Application

これらのアプリケーションは COM+ インフラストラクチャに属していて、特定の言語に結びつけられていないため、Visual Basic 6.0 のコードをアップグレードする際に考慮すべき特別な要素はありません。

SOAP サービスの使用

クライアント アプリケーションがリモート サーバー上で実装されているメソッドを呼び出せると便利な場合があります。メソッドは、リモート サーバー上に格納された変動する情報を使用することもあります (たとえば、特定の通貨に対する現在の為替レートを返すメソッドなど)。また、開発者が、メソッドを使用するすべてのアプリケーションを再配置せずに、メソッドの実装をアップグレードしたい場合があります。

XML Web サービスは、IIS などの Web サーバー上で公開され、HTTP を使用してアクセスされます。これらの HTTP パケットには、サーバー上で実装されているメソッドへの呼び出しの入出力パラメータが含まれており、SOAP でエンコードされています。XML Web サービスを使用するには、サービスが公開されている URL と呼び出したいメソッドの名前を調べ、メソッドに対して入力パラメータを指定する必要があります。

メモ：SOAP サービスの機能をサポートしているオペレーティング システムは、Microsoft Windows 2003、Microsoft Windows 2000、および Microsoft Windows XP です。SOAP サービスは、Microsoft Windows NT ではサポートされていません。

SOAP

XML Web サービスの基礎にあるインフラストラクチャを理解することは有益ですが、COM+ を使用すると、XML Web サービスの作成と使用が簡単になります。

あらゆる COM+ アプリケーションは、XML Web サービスとして公開できます。これによりクライアントは、アプリケーションの構成されたコンポーネントの既定のインターフェイス内のメソッドを、リモートから呼び出すことができるようになります。コンポーネント サービス管理ツールを使用すると、SOAP を使用してコンポーネントのメソッドを呼び出す IIS 仮想ルート ディレクトリを作成できます。コンポーネントを記述する際には、公開するメソッドを既定のインターフェイスに追加し、コンポーネントをサーバーの COM+ カタログに構成する以外は、特別なプログラミングは必要ありません。ネットワーク インターフェイスを通じて通信するためのコードや、SOAP を解析するためのコードを記述する必要はありません。

COM+ アプリケーションを XML Web サービスとして公開する際、XML Web サービスで使用可能なすべてのメソッドの構文に関する詳細情報は、Web サービス記述言語 (WSDL) を使用して自動的に公開されます。クライアントは、この情報を使用して XML Web サービスと通信します。

COM+ では、リモート XML Web サービスにアクセスして使用するための方法として、次の 2 つがあります。

- 既知のオブジェクト (WKO) モードを使用すると、COM+ を使用せずに作成したサービスや、Microsoft Windows を使用していないサービスであっても、WSDL を使用して構文を公開している XML Web サービスのすべてにアクセスできます。
- クライアント起動オブジェクト (CAO) モードは、COM+ アプリケーションを公開することで作成された XML Web サービスにアクセスするためにのみ使用されます。CAO モードでは、現在の SOAP 標準ではサポートされていない常設の接続を使用することで、パフォーマンスが向上します。

どちらの方法でも、ネットワーク インターフェイスを通じて通信するためのコードや、SOAP を解析するためのコードを記述しなくても、クライアント アプリケーションから XML Web サービスのメソッドを呼び出すことができます。

以下のコードは、COM+ と SOAP の例です。この例は、コンポーネントを Web サービスとして自動的に公開するために、ApplicationActivation 属性を SoapVRoot 要素と共に使用する方法を示します。

Visual Basic .NET で COM+ を使用するには、厳密な名前を使用してコンポーネントに署名する必要があります。厳密な名前で必要なキーを生成するには、次のコマンドを使用します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

```
sn -k SOAPServices.snk
```

以下のコードは、COM+/SOAP の例です。

```
Imports System
Imports System.Reflection
Imports System.Runtime.InteropServices
Imports System.EnterpriseServices

Namespace SOAPServices

    Public Interface ICalc
        Function Add(ByVal Value1 As Integer, ByVal Value2 As Integer)
            As Integer
    End Interface

    Public Class Calc
        Inherits ServicedComponent
        Implements ICalc

        Public Function Add(ByVal Value1 As Integer, _
            ByVal Value2 As Integer) As Integer Implements ICalc.Add

            Return (Value1 + Value2)
        End Function
    End Class
End Namespace
```

この例を実行するには、AssemblyInfo.vb ファイルに以下の行を含める必要があります。

```
Imports System.EnterpriseServices
<Assembly: ApplicationName("SOAP Services")>
<Assembly: ApplicationActivation(ActivationOption.Server, SoapVRoot := _
    "SOAPServices")>
<Assembly: AssemblyKeyFile("SOAPServices.snk")>
```

.NET での COM+ アプリケーション プロキシ

別のコンピュータからリモートで COM+ サーバー アプリケーションにアクセスするには、クライアント コンピュータは、プロキシ/スタブの DLL や、DCOM インターフェイス リモート処理用のタイプ ライブラリなど、サーバー アプリケーションの属性のサブセットがインストールされている必要があります。このサブセットは、*アプリケーション プロキシ*と呼ばれます。

サービス コンポーネントを作成してサーバーに登録する際、アプリケーション プロキシを作成するために特別なコードを記述する必要はありません。ServicedComponent クラスは継承ツリーに MarshalByRefObject を含んでいるため、リモート クライアントからアクセスできます。ただし、COM+ アプリケーション プロキシに含まれているデータが実際のサーバー コードに含まれないように、インターフェイス定義とクラスの実装を分離するように習慣付けることをお勧めします。

MTS/COM+ サービスのアップグレード

COM+ コンポーネントをアップグレードするためにアップグレード ウィザードを使用して手作業で変更する方法は、ほとんどが簡単な作業ですが、心に留めておくべき注意点がいくつかあります。

この章では、コード例を使用して、この点について詳しく説明します。各シナリオでは、Visual Basic 6.0 コンポーネント、アップグレード ウィザードをコンポーネントに適用した結果得られた出力、Visual Basic .NET へのアップグレードを完成させるために必要な手作業での変更内容について説明します。

メモ：この章で使用している .NET コンポーネントのコード サンプルを実行するには、まずコンポーネントを COM 相互運用機能に登録する必要があります。.NET コンポーネントを登録する方法については、第 14 章「Visual Basic 6.0 と Visual Basic .NET の相互運用」の「.NET での相互運用ラッパーの作成」を参照してください。

COM+ のシナリオ例

このシナリオでは、Visual Basic 6.0 サーバー コンポーネントの関数を公開します。このコンポーネントを COM+ アプリケーションとしてコンポーネント サービスに追加して、クライアント アプリケーションから使用します。まずアップグレード ウィザードを使用してサーバー コンポーネントをアップグレードし、必要な手作業での変更を行って、Visual Basic .NET の対応するコンポーネントを作成します。シナリオの最後でクライアント コンポーネントをアップグレードします。

以下に示す非常に単純化されたコード例は、Visual Basic 6.0 で記述された、COMTest という名前のサーバーコンポーネントを含んでいます。この例のコンポーネントでは、トランザクションを使用する必要はありません。

```
Private OpCounter As Integer

Public Sub Init()
    OpCounter = 0
End Sub

Public Function Add(ByVal value1 As Long, ByVal value2 As Long) _
    As Long

    Dim SumResult As Long
    SumResult = value1 + value2
    OpCounter = OpCounter + 1
    add = SumResult
End Function
```

コンポーネントの中心となるのが、Long 型の 2 つのパラメータを受け取る加算関数です。初期化メソッド Init も公開します。

以下のコード サンプルは、COM+ アプリケーション内にあるサーバー コンポーネントをインスタンス化する Visual Basic 6.0 アプリケーションの例です。

```
Public Sub Execute()
    Dim MyCOM As Object
    Dim result As Long

    Set MyCOM = CreateObject("Test.COMTest")
    MyCOM.Init
    result = MyCOM.Add(10, 5)

End Sub
```

このシナリオでは、サーバー コンポーネント モジュールの名前は COMTest であり、ダイナミック リンク ライブラリの名前は Test です。COM+ アプリケーションでオブジェクトをインスタンス化するには、Test.COMTest をパラメータとして CreateObject を呼び出します。

サーバー コンポーネントのアップグレード

アップグレード ウィザードを使用してサーバー コンポーネントをアップグレードすると、以下の出力が得られます。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")>
Public Class COMTest
    Private OpCounter As Short

    Public Sub Init()
        OpCounter = 0
    End Sub

    Public Function Add(ByVal value1 As Integer, ByVal value2 As Integer) _
        As Integer

        Dim SumResult As Integer
        SumResult = value1 + value2
        OpCounter = OpCounter + 1
        add = SumResult
    End Function
End Class
```

アップグレード ウィザードは、モジュールの機能のほとんどを正しくアップグレードしますが、完全に機能する正確なサーバー コンポーネントとするには、いくつか軽微な変更を行う必要があります。

▶ 機能的に対応するコンポーネントを作成するには

1. プロジェクトに System.EnterpriseServices への参照を追加し、Imports ステートメントを指定します。

```
Imports System.EnterpriseServices
```

2. ServicedComponent クラスから継承するようにクラスを変更します。

```
Public Class COMTest
    Inherits ServicedComponent
```

3. COMTest コンポーネントの厳密な名前を作成します（厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください）。

```
sn -k COMKey.snk
```

4. サーバー コンポーネントと、作成したキー ペア ファイルをリンクします。そのためには、AssemblyInfo.vb ファイルに AssemblyKeyFile 属性を追加します。

```
<Assembly: AssemblyKeyFile("COMKey.snk")>
```

上記の変更を行った結果を以下に示します。

```
Option Strict Off
Option Explicit On
Imports System.EnterpriseServices

<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")>
Public Class COMTest
    Inherits ServicedComponent

    Private OpCounter As Short
    Public Sub Init()
        OpCounter = 0
    End Sub

    Public Function Add(ByVal value1 As Integer, ByVal value2 As Integer) _
        As Integer

        Dim SumResult As Integer
        SumResult = value1 + value2
        OpCounter = OpCounter + 1
        add = SumResult
    End Function
End Class
```

手作業での変更をすべて終えたら、コンポーネントを構築し、新しい COM+ アプリケーションを作成して、DLL をアプリケーションに追加します。

心に留めておくべき非常に重要な注意点があります。Visual Basic 6.0 コンポーネントでは、Instancing 属性が Multiuse に設定され、他のアプリケーションがクラスからオブジェクトを作成できるようになっています。コンポーネントの 1 つのインスタンスが、このようにして作成された任意の数のオブジェクトを提供できます。アウトプロセス コンポーネントは、複数のクライアントに対して複数のオブジェクトを提供できます。インプロセス コン

ポーネントは、クライアントおよびそのプロセス内の他のあらゆるコンポーネントに対して、複数のオブジェクトを提供できます。この動作はコードに依存しないため、コンポーネントやクライアントのソース コードを変更しなくても、COM+ アプリケーションの起動方法を切り替えることができます。

これに対し、Visual Basic .NET のコンポーネントは、既定ではライブラリのアクティブ化を使用します。したがって、COM+ アプリケーションをサーバーとしてアクティブ化するよう指定すると、コンポーネントを作成しようとしたときに、.NET クライアント アプリケーションがクラッシュします。.NET コンポーネントのアクティブ化の方法を指定するには、AssemblyInfo.vb ファイルで ApplicationActivation 属性を使用して、コンポーネントのソースコードを変更する必要があります。

- サーバー コンポーネントでは、以下に示すように使用します。

```
<Assembly: System.EnterpriseServices.ApplicationActivation _
    (System.EnterpriseServices.ActivationOption.Server)>
```

- ライブラリ コンポーネントでは、以下に示すように使用します。

```
<Assembly: System.EnterpriseServices.ApplicationActivation _
    (System.EnterpriseServices.ActivationOption.Library)>
```

クライアント アプリケーションのアップグレード

クライアント アプリケーションのアップグレードは、サーバー コンポーネントのアップグレードよりも簡単です。以下のコード例は、アップグレード ウィザードで生成された出力を示します。

メモ : 以下のコード例およびこの章全体で、アップグレード ウィザードで自動的に生成された長いコード行とコメントのいくつかは、読みやすく理解しやすいように再フォーマットしてあります。そのため、実際の出力はこの章のコードといくらか異なる場合があります。

```
Public Sub Execute()
    Dim COM As Object
    Dim result As Integer

    COM = CreateObject("Test.COMTest")
    ' UPGRADE_WARNING: Could not resolve default property of
    ' object COM.init
    COM.Init()
    ' UPGRADE_WARNING: Could not resolve default property of
    ' object COM.Add
    result = COM.Add(10, 5)
End Sub
```

以下のステートメントは、変更が必要な唯一のステートメントです。

```
COM = CreateObject("Test.COMTest")
```

CreateObject メソッドで使用するパラメータは、COM+ アプリケーション内の .NET コンポーネントの名前と一致する必要があります。使用する正しい名前については、サーバー コンポーネントの ProgID 属性を参照してください。

```
COM = CreateObject("COMTest_NET.COMTest")
```

最終的なクライアント アプリケーションは以下のとおりです。

```
Public Sub Execute()
    Dim COM As Object
    Dim result As Integer

    COM = CreateObject("COMTest_NET.COMTest")
    COM.Init()
    result = COM.Add(10, 5)
End Sub
```

Visual Basic 2005 の場合：

Visual Studio 2005 で提供されるバージョンの Visual Basic .NET アップグレード ウィザードでは、この章で説明している手作業の変更のほとんどが自動的に実行されます。

COM+ Compensating Resource Manager

COM+ Compensating Resource Manager (CRM) サービスを使用すると、アプリケーション リソースを Microsoft 分散トランザクション コーディネータ (DTC) トランザクションに統合できます。

COM+ サービス ライブラリでは、CRMClerk、CRMRecoveryClerk、ICrmFormatLogRecords、ICrmMonitorLogRecords、tagCrmLogRecordRead などの、CRM サービスを開発者がうまく利用するためのいくつかのクラスとインターフェイスが提供されています。これらのサービスは .NET Framework の System.EnterpriseServices.CompensatingResourceManager 名前空間でも提供されています。

開発者は、CRMClerk クラスを使用してログ関連の操作を行うことができます。.NET Framework での対応するクラスは System.EnterpriseServices.CompensatingResourceManager.Clerk です。

Visual Basic 6.0 の開発者は、ICrmCompensatorVariants インターフェイスを実装してログ関連の問題を扱うことができます。Visual Basic .NET で同様の機能を得るには、ICrmCompensatorVariants をサポートしているクラスを System.EnterpriseServices.CompensatingResourceManager.Compensator から継承する必要があります。

CRM は、実装が必要な 2 つの COM コクラス Worker と Compensator から成ります。Worker クラスは変更を開始し、Compensator クラスは変更をコミットまたはロールバックします。

ワーカー コンポーネントのアップグレード

以下のコード例は、COMTest という名前の Visual Basic 6.0 クラス内で定義された CRM ワーカー コンポーネントを示します。

```
Public Clerk As CRMClerk

Public Sub Register()
    On Error GoTo ErrorHandler
    Set Clerk = New CRMClerk

    Clerk.RegisterCompensator "MyCompensator.Comp", _
        "This is my Compensator", CRMREGFLAG_ALLPHASES
Exit Sub

ErrorHandler:
    ' エラー処理コードがここに入ります。
End Sub
```

以下の例は、アップグレード後のコードを示します。

```
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
Public Class COMTest
    Public Clerk As COMSVCSLib.CRMClerk

    Public Sub Register()
        On Error GoTo ErrorHandler
        Clerk = New COMSVCSLib.CRMClerk

        Clerk.RegisterCompensator("MyCompensator.Comp", _
            "This is my Compensator", _
            COMSVCSLib.tagCRMREGFLAGS.CRMREGFLAG_ALLPHASES)
    Exit Sub

    ErrorHandler:
        ' エラー処理コードがここに入ります。
    End Sub
End Class
```

▶ 機能的に対応するコンポーネントを作成するには

1. プロジェクトに EnterprisesServices への参照を追加し、対応する Imports ステートメントをクラスに追加します。

```
Imports System.EnterpriseServices
```

2. ServicedComponent クラスから継承するようにクラスを変更します。

```
Inherits ServicedComponent
```

3. COMSVCSLib.CRMClerk への参照はすべて削除します。.NET Framework の対応するクラスでこれらの参照が置き換わるためです。
4. Visual Basic 6.0 の Clerk オブジェクトを.NET の CompensatingResourceManager.Clerk オブジェクトで置き換えます。
5. COMSVCSLib.CRMClerk コンストラクタの呼び出しと RegisterCompensator メソッドの呼び出しを削除して、Clerk クラスのコンストラクタの呼び出し 1 つに置き換えます。Clerk コンストラクタのシグネチャは以下のとおりです。

```
Public Sub New( _
    ByVal compensator As String, _
    ByVal description As String, _
    ByVal flags As CompensatorOptions _
    )
```

6. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します（厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください）。

```
sn -k MyKey.snk
```

これらの変更を行った結果に得られる、正しくアップグレードされたコンポーネントを以下に示します。

```
Imports System.EnterpriseServices

<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
    Public Class COMTest
    Inherits ServicedComponent

    Public Clerk As CompensatingResourceManager.Clerk

    Public Sub Register()
        On Error GoTo ErrorHandler
        Clerk = New CompensatingResourceManager.Clerk("", "", _
            CompensatingResourceManager.CompensatorOptions.AllPhases)
        Exit Sub

    ErrorHandler:
        ' エラー処理コードがここに入ります。
    End Sub
End Class
```

コンペンセータコンポーネントのアップグレード

以下のコード例は、Visual Basic 6.0 の CRM コンペンセータコンポーネントを示します。

```
Option Explicit

Implements ICrmCompensatorVariants

Dim CrmLogControl As ICrmLogControl

Private Function ICrmCompensatorVariants_AbortRecordVariants( _
    pLogRecord As Variant) As Boolean

End Function

Private Sub ICrmCompensatorVariants_BeginAbortVariants( _
    ByVal bRecovery As Boolean)

End Sub

Private Sub ICrmCompensatorVariants_BeginCommitVariants( _
    ByVal bRecovery As Boolean)

End Sub

Private Sub ICrmCompensatorVariants_BeginPrepareVariants()
End Sub

Private Function ICrmCompensatorVariants_CommitRecordVariants( _
    pLogRecord As Variant) As Boolean

    ICrmCompensatorVariants_CommitRecordVariants = False
End Function

Private Sub ICrmCompensatorVariants_EndAbortVariants()
End Sub

Private Sub ICrmCompensatorVariants_EndCommitVariants()
End Sub

Private Function ICrmCompensatorVariants_EndPrepareVariants() _
    As Boolean

    ICrmCompensatorVariants_EndPrepareVariants = True
End Function

Private Function ICrmCompensatorVariants_PrepareRecordVariants( _
    pLogRecord As Variant) As Boolean

End Function

Private Sub ICrmCompensatorVariants_SetLogControlVariants( _
```

```
    ByVal pLogControl As COMSVCSLib.ICrmLogControl)
    Set CrmLogControl = pLogControl
End Sub
```

アップグレード ウィザードを使用してこのコンポーネントをアップグレードすると、以下の Visual Basic .NET のコードが得られます。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
Public Class COMTest Implements COMSVCSLib.ICrmCompensatorVariants

    Dim CrmLogControl As COMSVCSLib.ICrmLogControl

    Private Function ICrmCompensatorVariants_AbortRecordVariants( _
        ByRef pLogRecord As Object) As Boolean Implements _
        COMSVCSLib.ICrmCompensatorVariants.AbortRecordVariants
    End Function

    Private Sub ICrmCompensatorVariants_BeginAbortVariants(_
        ByVal bRecovery As Boolean) Implements _
        COMSVCSLib.ICrmCompensatorVariants.BeginAbortVariants

    End Sub

    Private Sub ICrmCompensatorVariants_BeginCommitVariants(_
        ByVal bRecovery As Boolean) Implements _
        COMSVCSLib.ICrmCompensatorVariants.BeginCommitVariants

    End Sub

    Private Sub ICrmCompensatorVariants_BeginPrepareVariants() _
        Implements _
        COMSVCSLib.ICrmCompensatorVariants.BeginPrepareVariants

    End Sub

    Private Function ICrmCompensatorVariants_CommitRecordVariants( _
        ByRef pLogRecord As Object) As Boolean Implements _
        COMSVCSLib.ICrmCompensatorVariants.CommitRecordVariants

        ICrmCompensatorVariants_CommitRecordVariants = False
    End Function

    Private Sub ICrmCompensatorVariants_EndAbortVariants() Implements _
        COMSVCSLib.ICrmCompensatorVariants.EndAbortVariants
    End Sub

    Private Sub ICrmCompensatorVariants_EndCommitVariants() _
        Implements COMSVCSLib.ICrmCompensatorVariants.EndCommitVariants
    End Sub

    Private Function ICrmCompensatorVariants_EndPrepareVariants() _
```

```
        As Boolean Implements _
        COMSVCSLib.ICrmCompensatorVariants.EndPrepareVariants

        ICrmCompensatorVariants_EndPrepareVariants = True
    End Function

    Private Function ICrmCompensatorVariants_PrepareRecordVariants( _
        ByRef pLogRecord As Object) As Boolean Implements _
        COMSVCSLib.ICrmCompensatorVariants.PrepareRecordVariants
    End Function

    Private Sub ICrmCompensatorVariants_SetLogControlVariants( _
        ByVal pLogControl As COMSVCSLib.ICrmLogControl) Implements _
        COMSVCSLib.ICrmCompensatorVariants.SetLogControlVariants

        CrmLogControl = pLogControl
    End Sub
End Class
```

▶ アップグレードを完成させるには

1. プロジェクトに EnterpriseServices への参照を追加し、以下の Imports ステートメントをクラスに追加します。

```
Imports System.EnterpriseServices.CompensatingResourceManager
```

2. 次に、Compensator クラスから継承するようにクラスを変更します。

```
Inherits Compensator
```

Compensator クラスは、すべての Compensating Resource Manager (CRM) コンペンセータの基本クラスです。

3. プロジェクトから COMSVCSLib への参照をすべて削除します。
4. 表 15.1 に従ってメソッドを置き換えます。

表 15.1: Visual Basic 6.0 のメソッドとそれに対応する System.EnterpriseServices.CompensatingResourceManger 名前空間のメソッド

Visual Basic 6.0 のメソッド	.NETの System.EnterpriseServices. CompensatingResourceManager の対応するメソッド
ICrmCompensatorVariants_ AbortRecordVariants	Compensator.AbortRecord
ICrmCompensatorVariants_ BeginAbortVariants	Compensator.BeginAbort
ICrmCompensatorVariants_ BeginCommitVariants	Compensator.BeginCommit

Visual Basic 6.0 のメソッド	.NETの System.EnterpriseServices.CompensatingResourceManager の対応するメソッド
ICrmCompensatorVariants_BeginPrepareVariants	Compensator.BeginPrepare
ICrmCompensatorVariants_CommitRecordVariants	Compensator.CommitRecord
ICrmCompensatorVariants_EndAbortVariants	Compensator.EndAbort
ICrmCompensatorVariants_EndCommitVariants	Compensator.EndCommit
ICrmCompensatorVariants_EndPrepareVariants	Compensator.EndPrepare
ICrmCompensatorVariants_PrepareRecordVariants	Compensator.PrepareRecord
ICrmCompensatorVariants_SetLogControlVariants	サポートされません。Compensator の読み取り専用プロパティ Clerk を使用してください。
COMSVCSLib.ICrmLogControl	アップグレードの必要はありません。Compensator の読み取り専用プロパティ Clerk を使用してください。

5. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

```
sn -k MyKey.snk
```

以下に示すコードは、Visual Basic .NET の最終的なコンペンセータです。

```
Option Strict Off
Option Explicit On

Imports System.EnterpriseServices.CompensatingResourceManager

<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
    Public Class COMTest
    Inherits Compensator

    Public Overrides Function AbortRecord(ByVal rec As LogRecord) _
        As Boolean

    End Function

    Public Overrides Sub BeginAbort(ByVal fRecovery As Boolean)
    End Sub
```

```
    Public Overrides Sub BeginCommit(ByVal fRecovery As Boolean)
    End Sub

    Public Overrides Sub BeginPrepare()
    End Sub

    Public Overrides Function CommitRecord(ByVal rec As LogRecord) _
        As Boolean

        CommitRecord = False
    End Function

    Public Overrides Sub EndAbort()
    End Sub

    Public Overrides Sub EndCommit()
    End Sub

    Public Overrides Function EndPrepare() As Boolean
        EndPrepare = True
    End Function

    Public Overrides Function PrepareRecord(ByVal rec As LogRecord) _
        As Boolean

    End Function
End Class
```

COM+ オブジェクト プール

Visual Basic 6.0 を使用して開発されたコンポーネントはプールできません。その理由は、COM+ オブジェクトプールではマルチスレッド アパートメント (MTA) コンポーネントが必要ですが、Visual Basic 6.0 のコンポーネントは、シングルスレッド アパートメント モデルを使用するためです。ただし、Visual Basic 6.0 アプリケーションは、IObjectControl インターフェイスを使用して、オブジェクトをプールできるかどうかを確認できます。

Visual Basic 6.0 では、ObjectControl クラスが IObjectControl と同じ機能を提供します。Visual Basic .NET では、System.EnterpriseServices 名前空間の ServicedComponent クラスが同等の機能を提供します。したがって、ServicesComponent クラスから継承している Visual Basic .NET クラスはプールすることが可能です。

オブジェクトをプール可能にするには、ObjectPooling 属性を使用してクラスを修飾し、System.EnterpriseServices 名前空間への参照を追加します。また、プール済みオブジェクトはサーバー アプリケーションでのみ動作し、ライブラリとして動作するように構成することはできません。以下の Visual Basic .NET クラスでは、オブジェクト プールの最小サイズは 2、最大サイズは 4 です。

```
Imports System.EnterpriseServices
Imports System.Windows.Forms
```

```
<ObjectPooling(2, 4), _
    System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
    Public Class COMTest
    Inherits ServicedComponent
    Public counter As Integer = 0

    <AutoComplete()> Public Sub IncrementCounter() As Integer
        counter = counter + 1
        IncrementCounter = counter
    End Sub

    Protected Overrides Function CanBePooled() As Boolean
        Return True
    End Function
End Class
```

通常の状況下では、Visual Basic 6.0 でプールされないオブジェクトは、Visual Basic .NET でもプールする必要はありません。オブジェクト プールは、機能的に同等であることを超えた新機能であり、完全にテストする必要があります。

COM+ アプリケーションのセキュリティ

COM+ アプリケーションでは、リソースに対するアクセスを制御することができます。ロールを使用することで、開発者はアプリケーションの認証ポリシーを管理上構成し、どのユーザーがどのリソースにアクセスできるかを選択できます。必要に応じてメソッド レベルでの指定も可能です。また、ロールは、アプリケーションでより細かなアクセス制御が必要な場合に、コード内でのセキュリティ チェックを強制するためのフレームワークを提供します。

ロール ベース セキュリティは、開発者が、ある特定のコンポーネントへの呼び出しチェーンで、上流にあるすべての呼び出し元に関するセキュリティ情報を取得できる一般的なメカニズムの上に構築されています。

Visual Basic 6.0 の開発者は、SecurityCallContext クラスを使用してユーザーロール情報を扱うことができます。Visual Basic 6.0 では、SecurityCallContext の特定のインスタンスを宣言しなくても、IGetSecurityCallContext インターフェイスで提供される GetSecurityCallContext グローバル関数を使用し、返されたオブジェクトにアクセスすることで、SecurityCallContext のメンバにアクセスできます。

Visual Basic .NET の開発者は、System.EnterpriseServices.SecurityCallContext の静的インスタンスの CurrentCall プロパティを通じて、GetSecurityCallContext と同等の機能を利用できます。

Visual Basic 6.0 の開発者は、SecurityCallers と SecurityIdentity を使用して、セキュリティで保護されたアプリケーションの呼び出し元に関する情報を操作できます。Visual Basic .NET では、System.EnterpriseServices 名前空間に含まれている SecurityCallers クラスと SecurityIdentity クラスが同等の機能を提供します。

セキュリティオブジェクトを使用する Visual Basic 6.0 コンポーネントのアップグレードは、非常に簡単な作業です。以下のコード例は、セキュリティ機能が有効か無効かを確認する Visual Basic 6.0 のコンポーネントです。

```
Public Function SecurityEnabled(ByVal caller As String) As Boolean
    Dim SecContext As SecurityCallContext

    On Error GoTo ErrorHandler

    Set SecContext = GetSecurityCallContext
    SecurityEnabled = SecContext.IsSecurityEnabled
Exit Function

ErrorHandler:
    ' エラー処理コードがここに入ります。
End Function
```

アップグレードウィザードを使用してこのコンポーネントをアップグレードすると、以下の Visual Basic .NET のコードが得られます。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
Public Class COMTest
    Public Function SecurityEnabled(ByVal caller As String) _
        As Boolean

        Dim SecContext As COMSVCSLib.SecurityCallContext

        On Error GoTo ErrorHandler

        SecContext = _
            COMSVCSLibGetSecurityCallContextAppObject_definst. _
            GetSecurityCallContext
        SecurityEnabled = SecContext.IsSecurityEnabled
    Exit Function

    ErrorHandler:
        ' エラー処理コードがここに入ります。
    End Function
End Class
```

▶ アップグレードを完成させるには

1. プロジェクトに EnterpriseServices への参照を追加し、対応する Imports ステートメントをクラスに追加します。

```
Imports System.EnterpriseServices
```

2. ServicedComponent クラスから継承するようにクラスを変更します。

```
Inherits ServicedComponent
```

3. プロジェクトから COMSVCSLib への参照をすべて削除します。
4. Visual Basic 6.0 の SecurityCallContext オブジェクトを .NET の System.EnterpriseServices.SecurityCallContext で置き換えます。
5. Visual Basic 6.0 の GetSecurityCallContext メソッドを .NET の静的メソッド SecurityCallContext.CurrentCall で置き換えます。

```
System.EnterpriseServices.SecurityCallContext.CurrentCall()
```

6. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

```
sn -k MyKey.snk
```

以下のコード例は、最終的なコードです。

```
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
Public Class COMTest
  Inherits ServicedComponent
  Public Function SecurityEnabled(ByVal caller As String) As Boolean

      Dim SecContext As System.EnterpriseServices.SecurityCallContext

      On Error GoTo ErrorHandler

      SecContext =
          System.EnterpriseServices.SecurityCallContext.CurrentCall()
      SecurityEnabled = SecContext.IsSecurityEnabled
    Exit Function

    ErrorHandler:
        ' エラー処理コードがここに入ります。
    End Function
End Class
```

COM+ Shared Property Manager

COM+ Shared Property Manager (SPM) は、オブジェクトの共有された一時的な状態を管理するために使用できます。分散環境では、同時実行の問題と名前の競合の問題があるため、グローバル変数は使用できません。SPM では、共有プロパティ グループを提供し、その中に含まれる共有プロパティに対して一意の名前空間を確立することで、名前の競合がなくなります。また、同期メカニズムとしてロックとセマフォを実装しています。

シングル プロセッサ サーバーでも、クライアントの台数が少なければ、状態を保持するためのメカニズムとして、Shared Property Manager は非常にうまく動作します。ただし、あまり拡張性が高くないため、ユーザー

が同時かつ頻繁に使用するアプリケーションでは、データベースを使用して状態情報を保持することをお勧めします。

Visual Basic 6.0 では、SPM 関連の機能を実装するためのクラスは、COM+ サービスのタイプ ライブラリの SharedPropertyManager 、SharedPropertyGroup、および SharedProperty の各クラスです。Visual Basic .NET でこれらのクラスに対応するクラスは、System.EnterpriseServices 名前空間に含まれており、同じ名前を持っているため明確に識別できます。

以下の Visual Basic 6.0 コンポーネントは、SPM オブジェクトを使用しています。

```
Public Function SPMTest(ByVal strGrpName As String, _
    ByVal strPrpName As String, ByVal vntPrpValue As String, _
    ByRef blnExists As Boolean)

    Dim spmMgr As COMSVCSLib.SharedPropertyGroupManager
    Dim spmGrp As COMSVCSLib.SharedPropertyGroup
    Dim spmPrp As COMSVCSLib.SharedProperty

    Set spmMgr = New COMSVCSLib.SharedPropertyGroupManager
    Set spmGrp = spmMgr.CreatePropertyGroup(strGrpName, LockSetGet, _
        Process, blnExists)
    Set spmMgr = Nothing
    Set spmPrp = spmGrp.CreateProperty(strPrpName, blnExists)

    spmPrp.Value = vntPrpValue
    SPMTest = spmPrp.Value

    Set spmPrp = Nothing
    Set spmGrp = Nothing
End Function
```

コンポーネントをアップグレードした後のクラスには、System.EnterpriseServices 名前空間にある、.NET Framework で提供された Security クラスを使用して解決することが必要なステートメントがいくつか含まれています。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
Public Class COMTest
    Public Function SPMTest(ByVal strGrpName As String, _
        ByVal strPrpName As String, ByVal vntPrpValue As String, _
        ByRef blnExists As Boolean) As Object

    Dim spmMgr As COMSVCSLib.SharedPropertyGroupManager
    Dim spmGrp As COMSVCSLib.SharedPropertyGroup
    Dim spmPrp As COMSVCSLib.SharedProperty

    spmMgr = New COMSVCSLib.SharedPropertyGroupManager

    spmGrp = spmMgr.CreatePropertyGroup(strGrpName, _
        COMSVCSLib.__MIDL___MIDL_itf_autosvcs_0408_0002.LockSetGet, _
        COMSVCSLib.__MIDL___MIDL_itf_autosvcs_0408_0003.Process, _
```

```
        blnExists)

    ' UPGRADE_NOTE: オブジェクト spmMgr をガベージ コレクトするまでこの
    ' オブジェクトを破棄することはできません。
    spmMgr = Nothing

    spmPrp = spmGrp.CreateProperty(strPrpName, blnExists)

    spmPrp.Value = vntPrpValue

    ' UPGRADE_WARNING: Could not resolve default property of
    ' object SPMTest
    SPMTest = spmPrp.Value

    ' UPGRADE_NOTE: オブジェクト spmPrp をガベージ コレクトするまで
    ' このオブジェクトを破棄することはできません。
    spmPrp = Nothing

    ' UPGRADE_NOTE: オブジェクト spmGrp をガベージ コレクトするまで
    ' このオブジェクトを破棄することはできません。
    spmGrp = Nothing
    End Function
End Class
```

前に述べたように、SPM インフラストラクチャで使用していたクラス名は、.NET でも同じです。

▶ アップグレードを完成させるには

1. プロジェクトに EnterpriseServices への参照を追加し、対応する Imports ステートメントをクラスに追加します。

```
Imports System.EnterpriseServices
```

2. ServicedComponent クラスから継承するようにクラスを変更します。

```
Inherits ServicedComponent
```

3. プロジェクトから COMSVCSLib への参照をすべて削除します。
4. Visual Basic 6.0 の LockSetGet 定数を、それに対応する .NET の PropertyLockMode.SetGet で置き換えます。

```
System.EnterpriseServices.PropertyLockMode.SetGet
```

5. Visual Basic 6 の Process 定数を、それに対応する .NET の PropertyReleaseMode.Process で置き換えます。

```
System.EnterpriseServices.PropertyReleaseMode.Process
```

6. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

```
sn -k MyKey.snk
```

以下のコード例は、最終的なコードです。

```
Option Strict Off
Option Explicit On

Imports System.EnterpriseServices

<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
    Public Class COMTest
    Inherits ServicedComponent

    Public Function SPMTest(ByVal strGrpName As String, _
        ByVal strPrpName As String, ByVal vntPrpValue As String, _
        ByRef blnExists As Boolean) As Object

        Dim spmMgr As SharedPropertyGroupManager
        Dim spmGrp As SharedPropertyGroup
        Dim spmPrp As SharedProperty

        spmMgr = New SharedPropertyGroupManager
        spmGrp = spmMgr.CreatePropertyGroup(strGrpName, _
            PropertyLockMode.SetGet, PropertyReleaseMode.Process, _
            blnExists)
        spmMgr = Nothing
        spmPrp = spmGrp.CreateProperty(strPrpName, blnExists)
        spmPrp.Value = vntPrpValue

        SPMTest = spmPrp.Value

        spmPrp = Nothing
        spmGrp = Nothing
    End Function
End Class
```

COM+ オブジェクト コンストラクタ文字列

COM+ オブジェクト コンストラクタ文字列は、コンポーネントに対して管理上指定される初期化文字列です。これらのオブジェクト コンストラクタ文字列は、特定の作業向けに後からカスタマイズできるような汎用性を持った単一のコンポーネントを記述するために使用できます。言い換えれば、パラメータ化されたオブジェクトコンストラクタを作成できます。

Visual Basic 6.0 の開発者は、IObjectConstruct インターフェイスと IObjectConstructString インターフェイスを実装することで、この COM+ の機能を使用できます。Visual Basic .NET では、System.EnterpriseServices 名前空間の ServicedComponent クラスの Construct メソッドが、同等の機能を提供します。構成文字列をサポートする Visual Basic .NET クラスは、System.EnterpriseServices.ServicedComponent から継承する必要があります。ServicedComponent クラスの Construct メソッドをオーバーライドすることで、IObjectConstruct の Construct メソッドをエミュレートできます。

以下の Visual Basic 6.0 コンポーネントは、IObjectConstruct インターフェイスを実装しています。

```
Implements COMSVCSLib.IObjectConstruct

Private MyConnString As String

Private Sub IObjectConstruct_Construct(ByVal objConstructor As Object)

    Dim cs As COMSVCSLib.IObjectConstructString
    Set cs = objConstructor
    MyConnString = cs.ConstructString

End Sub
```

アップグレード ウィザードを使用してこのコンポーネントをアップグレードすると、以下の Visual Basic .NET のコードが得られます。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
Public Class COMTest_
    Implements COMSVCSLib.IObjectConstruct

    Private MyConnString As String

    Private Sub IObjectConstruct_Construct(ByVal objConstructor _
        As Object) Implements COMSVCSLib.IObjectConstruct.Construct

        Dim cs As COMSVCSLib.IObjectConstructString
        cs = objConstructor
        MyConnString = cs.ConstructString

    End Sub
End Class
```

▶ アップグレードを完成させるには

1. プロジェクトに EnterpriseServices への参照を追加し、対応する Imports ステートメントをクラスに追加します。

```
Imports System.EnterpriseServices
```

2. ServicedComponent クラスから継承するようにクラスを変更します。

```
Inherits ServicedComponent
```

3. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

```
sn -k MyKey.snk
```

4. 以下で説明する、構成要素によるアプローチか属性によるアプローチまたはその両方に従います。

構成要素によるアプローチ

ServicedComponent クラスは、IObjectConstruct インターフェイスを仮想メソッドとして実装しています。そこから派生したサービス コンポーネントは、このコード例に示すように Construct メソッドをオーバーライドできます。

```
Option Strict Off
Option Explicit On

Imports System.EnterpriseServices

<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
    Public Class COMTest
    Inherits ServicedComponent

    Private MyConnString As String

    Protected Overrides Sub Construct(ByVal objConstructor As String)
        MyConnString = objConstructor
    End Sub

End Class
```

コンポーネントの [Activation] タブの [Enable object construction] チェック ボックスがオンになっていると、コンポーネントのコンストラクタが呼び出された後で、Construct メソッドが呼び出されます。これにより、コンポーネントに構成文字列が提供されます。

属性によるアプローチ

ConstructionEnabled 属性を使用することでも、構成文字列のサポートを有効にし、既定の構成文字列を提供できます。

```
Option Strict Off
Option Explicit On
```

```
Imports System.EnterpriseServices

<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest"), _
    ConstructionEnabled(True, [Default]:="My string")> _
    Public Class COMTest
    Inherits ServicedComponent

End Class
```

ConstructionEnabled 属性には 2 つのパブリック プロパティがあります。Enabled を使用すると、コンポーネントを登録した後で、コンポーネント サービス エクスプローラでのサービス コンポーネントの構成文字列のサポートが有効になり、Default は文字列の初期値です。

COM+ トランザクション

Visual Basic 6.0 では、MTSTransactionMode プロパティを使用して、ユーザー クラスのトランザクション動作を設定できます。このプロパティは、Microsoft Transaction Server で動作するコンポーネントでのみ使用され、コンポーネントが MTS の外で動作する場合は影響がありません。

Visual Basic .NET では、System.EnterpriseServices 名前空間を使用して、.NET クラスでトランザクション機能を使用できます。Visual Basic .NET クラスがトランザクション クラスとして扱われるようにするには、ServicedComponent から継承し、適切なトランザクション属性を追加する必要があります。Visual Basic 6.0 のトランザクション クラスは、サービス コンポーネントにアップグレードする必要があります。

MTSTransactionMode の値は、COM+ コンポーネントが要求する自動トランザクションの種類が含まれる System.EnterpriseServices.TransactionOption .NET Framework 列挙に対応します。MTSTransactionMode プロパティには、各グループの値の対応を表す表 15.2 に示した任意の定数値を設定できます。

表 15.2: MTSTransactionMode の値と System.EnterpriseServices.TransactionOption 名前空間での対応する値

MTSTransactionMode の値	System.EnterpriseServices.TransactionOption での対応する値
NotAnMTSObject (0)	Disabled
NoTransactions (1)	NotSupported
RequiresTransaction (2)	Required
UsesTransaction (3)	Supported
RequiresNewTransaction (4)	RequiresNew

次の情報を使用して、トランザクション クラスを参照する Visual Basic .NET プロジェクトを作成します。

- トランザクション クラスは ServicedComponent から継承します。
- トランザクション クラスにトランザクション属性を追加します。
- GetObjectContext の呼び出しを削除し、代わりに ContextUtil クラスを使用します。

Visual Basic 6.0 の MTSTransactionMode 属性は、あるクラスでトランザクションが必要かどうかを制御します。この属性は、System.EnterpriseServices.TransactionOption の対応する値にマップする必要があります。

COM+ トランザクション関連のクラスはすべて Visual Basic 6.0 でサポートされており、ほぼすべてのクラスについて .NET Framework に対応するクラスがあります。唯一の例外は ITransaction インターフェイスで、Visual Basic .NET で意味的に対応するクラスはありません。

この章の「Visual Basic 6.0 での MTS/COM+ の使用」で説明したように、単純な COM+ サーバー コンポーネントにトランザクションのサポートを追加すると、以下の Visual Basic 6.0 のコードが得られます。

MTSTransactionMode プロパティの値が RequiresTransaction に設定されている点と、サーバーの名前が COMTest である点に注意してください。

```
Private Sub transferfunds(acc1 As String, acc2 As String, _
    amount As Long)
    On Error GoTo ErrHandler

    withdrawFunds acc1, amount
    depositFunds acc2, amount
    GetObjectContext.SetComplete

    Exit Sub
ErrHandler:
    GetObjectContext.SetAbort
    Err.Raise Err.Number, Err.Source, Err.Description, _
        Err.HelpFile, Err.HelpContext
End Sub

Private Sub withdrawFunds(acc As String, amount As Long)
   ' 引き出しの実行...
End Sub

Private Sub depositFunds(acc As String, amount As Long)
   ' 預け入れの実行...
End Sub
```

アップグレード ウィザードを使用してこのサーバー コンポーネントをアップグレードすると、以下の Visual Basic .NET のコードが得られます。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
Public Class COMTest
    Public Sub transferfunds(ByRef acc1 As String, _
        ByRef acc2 As String, ByRef amount As Integer)
        On Error GoTo ErrHandler

        withdrawFunds(acc1, amount)
        depositFunds(acc2, amount)
        ' UPGRADE_WARNING: Could not resolve default property of
        ' object GetObjectContext.SetComplete.
        GetObjectContext.SetComplete()

        Exit Sub
    ErrHandler:
        ' UPGRADE_WARNING: Could not resolve default property of
        ' object GetObjectContext.SetAbort.
        GetObjectContext.SetAbort()
        Err.Raise(Err.Number, Err.Source, Err.Description, _
           Err.HelpFile, Err.HelpContext)
    End Sub

    Private Sub withdrawFunds(ByRef acc As String, _
        ByRef amount As Integer)
        ' 引き出しの実行...
    End Sub

    Private Sub depositFunds(ByRef acc As String, _
         ByRef amount As Integer)
        ' 預け入れの実行...
    End Sub
End Class
```

▶ アップグレードを完成させるには

1. プロジェクトに EnterpriseServices への参照を追加し、対応する Imports ステートメントをクラスに追加します。

```
Imports System.EnterpriseServices
```

2. ServicedComponent クラスから継承するようにクラスを変更します。

```
Inherits ServicedComponent
```

3. 次の 2 つの方法のいずれかを使用して、GetObjectContext 警告に対処します。

   a. 1 つ目の方法は、System.EnterpriseServices.ContextUtil クラスを使用することです。このクラスは、GetObjectContext に対応する .NET Framework クラスです。ContextUtil クラスは、COM+ オブジェクトのコンテキストに関する情報を取得します。

   ```
   ContextUtil.SetComplete()

   ContextUtil.SetAbort()
   ```

   b. 2 つ目の方法は、System.EnterpriseServices.AutoCompleteAttribute を使用します。この属性は、属性メソッドを AutoComplete オブジェクトとしてマークします。メソッド呼び出しが正常に返されると、トランザクションは自動的に SetComplete を呼び出します。メソッド呼び出しが例外をスローすると、トランザクションは中止されます。

4. TransactionOption.Supported 属性を .NET クラスに追加します。これを行う理由は、Visual Basic 6.0 サーバー コンポーネントで MTSTransactionMode プロパティを RequiresTransaction に設定しているためです。

   ```
   System.EnterpriseServices.Transaction(TransactionOption.Supported)
   ```

5. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

   ```
   sn -k COMKey.snk
   ```

最終的な Visual Basic .NET のコードを以下に示します。ここでは、ContextUtil クラスを使用しています。

```
Option Strict Off
Option Explicit On

Imports System.EnterpriseServices

<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest"), _
    System.EnterpriseServices.Transaction(TransactionOption.Supported) > _
    Public Class COMTest
    Inherits ServicedComponent

    Public Sub transferfunds(ByRef acc1 As String, _
        ByRef acc2 As String, ByRef amount As Integer)
        On Error GoTo ErrHandler

        withdrawFunds(acc1, amount)
        depositFunds(acc2, amount)
        ContextUtil.SetComplete()
```

```
        Exit Sub
    ErrHandler:
        ContextUtil.SetAbort()
        Err.Raise(Err.Number, Err.Source, Err.Description, _
           Err.HelpFile, Err.HelpContext)
    End Sub

    Private Sub withdrawFunds(ByRef acc As String, _
        ByRef amount As Integer)
        ' 引き出しの実行...
    End Sub

    Private Sub depositFunds(ByRef acc As String, _
        ByRef amount As Integer)
        ' 預け入れの実行...
    End Sub
End Class
```

以下のコード例は、AutoComplete 属性を使用した場合の例です。

```
Option Strict Off
Option Explicit On

Imports System.EnterpriseServices

<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest"), _
    System.EnterpriseServices.Transaction(TransactionOption.Supported)> _
    Public Class COMTest
    Inherits ServicedComponent

    <AutoComplete()> Public Sub transferfunds(ByRef acc1 As String, _
        ByRef acc2 As String, ByRef amount As Integer)
        On Error GoTo ErrHandler

        withdrawFunds(acc1, amount)
        depositFunds(acc1, amount)

        Exit Sub
    ErrHandler:
        Err.Raise(Err.Number, Err.Source, Err.Description, _
           Err.HelpFile, Err.HelpContext)
    End Sub

    Private Sub withdrawFunds(ByRef acc As String, _
        ByRef amount As Integer)
        ' 引き出しの実行...
    End Sub

    Private Sub depositFunds(ByRef acc As String, ByRef amount As Integer)
        ' 預け入れの実行...
    End Sub
End Class
```

その他の COM+ 機能

Visual Basic 6.0 での COM+ の機能には、アップグレード ウィザードでサポートされていないものがあります。アップグレード ウィザードでこれらの機能が見つかっても、自動的にはアップグレードされないため、手作業でアップグレードする必要があります。

表 15.3 に、Visual Basic 6.0 の COM+ オブジェクトと、それに対応する Visual Basic .NET のオブジェクトを示します。これらのオブジェクトはアップグレードで使用可能です。その他のオブジェクトは、オブジェクトの置き換えではなく、より大規模な変更が必要となります。

表 15.3: Visual Basic 6.0 の COM+ オブジェクトに対応する Visual Basic .NET のオブジェクト

Visual Basic 6.0 オブジェクト	対応する Visual Basic .NET オブジェクト
サービス構成 : CserviceConfig	System.EnterpriseServices.ServiceConfig
コンテキスト : IMTxAS	System.EnterpriseServices.ContextUtil
例外とエラー : Error_Constants	System.EnterpriseServices 名前空間の以下のクラスを使用してください。 RegistrationErrorInfo RegistrationException ServicedComponentException
COM+ アプリケーションの種類 : tagCOMPLUS_APPTYPE	System.Runtime.InteropServices.IMoniker

表 15.4 に、これらのオブジェクトのアップグレード方法に関する情報が記載されている参照先を示します。

表 15.4: Visual Basic 6.0 の COM+ オブジェクトをアップグレードするための情報がある参照先

アップグレード対象の Visual Basic 6.0 オブジェクト	この章の参照すべきセクション
COM+ イベント : COMEvents, ComServiceEvents, COMSVCSEVENTINFO, ComSystemAppEventData, IEventServerTrace, IMtsEventInfo, CoMTSLocator, MtsGrp	「COM+ イベント」
Compensating Resource Manager (CRM): ICrmFormatLogRecords, ICrmMonitorLogRecords, CRMRecoveryClerk	「COM+ Compensating Resource Manager」
セキュリティ : IsecurityProperty, SecurityCertificate, SecurityIdentity, SecurityProperty	「COM+ アプリケーションのセキュリティ」と「COM+ のセキュリティ」
SOAP: SoapMoniker	「SOAP サービスの使用」

アップグレード対象の Visual Basic 6.0 オブジェクト	この章の参照すべきセクション
トランザクション：TransactionContext, TransactionContextEx	「コンテキスト コンポーネント」
MSMQ: MessageMover	「メッセージ キューとキュー コンポーネント」
プール：PoolMgr	「COM+ オブジェクト プール」

COM+ のセキュリティ

COM+ では、COM+ アプリケーションを保護するために使用可能なセキュリティ機能がいくつか提供されています。以下の Visual Basic 6.0 コンポーネントは、SecurityCallContext オブジェクトと SecurityIdentity オブジェクトを使用して、コンポーネントの直接の呼び出し元に関する情報を取得します。メソッド GetSecurityCallContext は、直接の呼び出し元を取得するために使用する現在のセキュリティ コンテキストを返します。

```
Public Function DirectCallerInfo() As String
    Dim ctx As SecurityCallContext
    Dim idx As SecurityIdentity
    Dim AccName, AuServ, ImpL, AuLev As String

    Set ctx = GetSecurityCallContext
    Set idx = ctx.Item("DirectCaller")

    AccName = "Account Name: " & idx("AccountName") & " -- "
    AuServ = "Authentication Service: " & CLng(idx("AuthenticationService")) & _
        " -- "
    ImpL = "Impersonation Level: " & CLng(idx("ImpersonationLevel")) & " -- "
    AuLev = "Authentication Level: " & CLng(idx("AuthenticationLevel"))

    DirectCallerInfo = AccName & AuServ & ImpL & AuLev
End Function
```

アップグレード後の Visual Basic .NET コードには、いくつかの警告と COMSVCSLib ライブラリへの参照が含まれています。COM+ アプリケーションとしてエクスポート可能な .NET の機能アセンブリを取得するには、これらの問題を解決する必要があります。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> Public Class COMTest
    Public Function DirectCallerInfo() As String
        Dim ctx As COMSVCSLib.SecurityCallContext
        Dim idx As COMSVCSLib.SecurityIdentity
        Dim AuServ, AccName, ImpL As Object
        Dim AuLev As String
```

```
        ctx = _
            COMSVCSLibGetSecurityCallContextAppObject_definst.GetSecurityCallContext
        idx = ctx.Item("DirectCaller")

        ' UPGRADE_WARNING: Could not resolve default property of object idx().
        ' UPGRADE_WARNING: Could not resolve default property of object AccName.
        AccName = "Account Name: " & idx("AccountName") & " -- "
        ' UPGRADE_WARNING: Could not resolve default property of object idx().
        ' UPGRADE_WARNING: Could not resolve default property of object AuServ.
        AuServ = "Authentication Service: " & CInt(idx("AuthenticationService")) & _
           " -- "
        ' UPGRADE_WARNING: Could not resolve default property of object idx().
        ' UPGRADE_WARNING: Could not resolve default property of object ImpL.
        ImpL = "Impersonation Level: " & CInt(idx("ImpersonationLevel")) & " -- "
        ' UPGRADE_WARNING: Could not resolve default property of object idx().
        AuLev = "Authentication Level: " & CInt(idx("AuthenticationLevel"))

        ' UPGRADE_WARNING: Could not resolve default property of object ImpL.
        ' UPGRADE_WARNING: Could not resolve default property of object AuServ.
        ' UPGRADE_WARNING: Could not resolve default property of object AccName.
        DirectCallerInfo = AccName & AuServ & ImpL & AuLev
    End Function
End Class
```

▶ 機能的に対応するコンポーネントを作成するには

1. プロジェクトに EnterpriseServices への参照を追加し、対応する Imports ステートメントをクラスに追加します。

```
Imports System.EnterpriseServices
```

2. ServicedComponent クラスから継承するようにクラスを変更します。

```
Inherits ServicedComponent
```

3. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

```
sn -k MyKey.snk
```

5. COMSVCSLib への参照をすべて削除します (COMSVCSLib セキュリティ クラスの名前は EnterpriseServices 名前空間内でも同じであるため、クラス名を変更する必要はありません)。

6. SecurityCallContext.CurrentCall 静的メソッドを使用して、現在のセキュリティ コンテキストを取得します。

```
System.EnterpriseServices.SecurityCallContext.CurrentCall
```

7. DirectCaller を使用して、オブジェクトのインスタンスを返します。

```
SecurityCallContext.DirectCaller
```

8. SecurityIdentity クラスに含まれているそれぞれのプロパティから、アカウント名、認証サービス、偽装レベル、および認証レベルの値を取得します。

```
SecurityIdentity.AccountName
SecurityIdentity.AuthenticationService
SecurityIdentity.ImpersonationLevel
SecurityIdentity.AuthenticationLevel
```

これらの手順を例に適用した後のコードは、以下のようになります。

```
Option Strict Off
Option Explicit On

Imports System.EnterpriseServices

<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
    Public Class COMTest
    Inherits ServicedComponent
    Public Function DirectCallerInfo() As String
        Dim ctx As SecurityCallContext
        Dim idx As SecurityIdentity
        Dim AuServ, AccName, ImpL, AuLev As String

        ctx = SecurityCallContext.CurrentCall
        idx = ctx.DirectCaller

        AccName = "Account Name: " & idx.AccountName & " -- "
        AuServ = "Authentication Service: " & idx.AuthenticationService.ToString() _
            & " -- "
        ImpL = "Impersonation Level: " & idx.ImpersonationLevel.ToString & " -- "
        AuLev = "Authentication Level: " & idx.AuthenticationLevel.ToString

        DirectCallerInfo = AccName & AuServ & ImpL & AuLev
    End Function
End Class
```

コンテキスト コンポーネント

ここでは、トランザクション コンテキスト オブジェクトについて説明すると共に、Visual Basic .NET で使用するために Commit メソッドと Abort メソッドをアップグレードする方法について説明します。

この Visual Basic 6.0 コンポーネントは、TransactionContext オブジェクトを使用して、.NET セキュリティ コンポーネントのインスタンスを作成します。このアップグレードについては、「COM+ のセキュリティ」で説明しました。トランザクション中にエラーが発生しなければ、変更がコミットされます。エラーが発生した場合は、トランザクションは中止されます。

```
Public Function TestTC() As String
    Dim ctx As TransactionContext
    Dim Com As Object
    Dim str As String

    On Error GoTo ErrorHandler

    Set ctx = New TransactionContext
    Set Com = ctx.CreateInstance("COMTest_NET.COMTest")
    str = Com.DirectCallerInfo
    ctx.Commit
    TestTC = str
Exit Function

ErrorHandler:
    ctx.Abort
End Function
```

アップグレード ウィザードを使用してこのサーバー コンポーネントをアップグレードすると、以下の Visual Basic .NET のコードが得られます。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> Public Class COMTest
    Public Function TestTC() As String
        Dim ctx As COMSVCSLib.TransactionContext
        Dim Com As Object
        ' UPGRADE_NOTE: str は str_Renamed にアップグレードされました。
        Dim str_Renamed As String

        On Error GoTo ErrorHandler

        ctx = New COMSVCSLib.TransactionContext
        Com = ctx.CreateInstance("COMTest_NET.COMTest")
        ' UPGRADE_WARNING: Could not resolve default property of object
        ' Com.DirectCallerInfo.
        str_Renamed = Com.DirectCallerInfo
        ctx.Commit()
        TestTC = str_Renamed
    Exit Function
```

```
    ErrorHandler:
        ctx.Abort()
    End Function
End Class
```

▶ アップグレードを完成させるには

1. プロジェクトに EnterpriseServices への参照を追加し、対応する Imports ステートメントをクラスに追加します。

   ```
   Imports System.EnterpriseServices
   ```

2. ServicedComponent クラスから継承するようにクラスを変更します。

   ```
   Inherits ServicedComponent
   ```

3. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

   ```
   sn -k MyKey.snk
   ```

5. COMSVCSLib への参照をすべて削除します。
6. 以下のステートメントを削除します。

   ```
   Dim ctx As COMSVCSLib.TransactionContext
   ctx = New COMSVCSLib.TransactionContext
   ```

7. CreateObject メソッドを使用して、「COM+ のセキュリティ」で作成した Visual Basic 6.0 のセキュリティ コンポーネントをインスタンス化します。

   ```
   CreateObject("Test.COMTest")
   ```

8. TransactionContext の Commit メソッドと Abort メソッドを、それぞれ System.EnterpriseServices.ContextUtil の SetComplete メソッドと SetAbort メソッドで置き換えます。

   ```
   ContextUtil.SetComplete()
   ContextUtil.SetAbort()
   ```

最終的な Visual Basic .NET のコードは以下のとおりです。

```
Option Strict Off
Option Explicit On

Imports System.EnterpriseServices
```

```
<System.Runtime.InteropServices.ProgId("COMTest_NET.COMTest")> _
    Public Class COMTest
    Inherits ServicedComponent

    Public Function TestTC() As String
        Dim Com As Object
        Dim str_Renamed As String

        On Error GoTo ErrorHandler

        Com = CreateObject("Test.COMTest")
        str_Renamed = Com.DirectCallerInfo
        ContextUtil.SetComplete()
        TestTC = str_Renamed
        Exit Function

ErrorHandler:
        ContextUtil.SetAbort()
    End Function
End Class
```

COM+ イベント

COM+ イベント システムでは、疎結合イベント (LCE) システムの概念が採用されています。このイベント システムでは、イベントの受信者 (*サブスクライバ*と呼ばれます) が、イベントの提供者 (*パブリッシャ*と呼ばれます) に対して変数を宣言する必要がありません。代わりに、サブスクライバとパブリッシャのどちらも、COM+ イベント クラスの形式のインターフェイス 定義に依存します。

図 15.1 に、COM+ イベント システムの概念を示します。

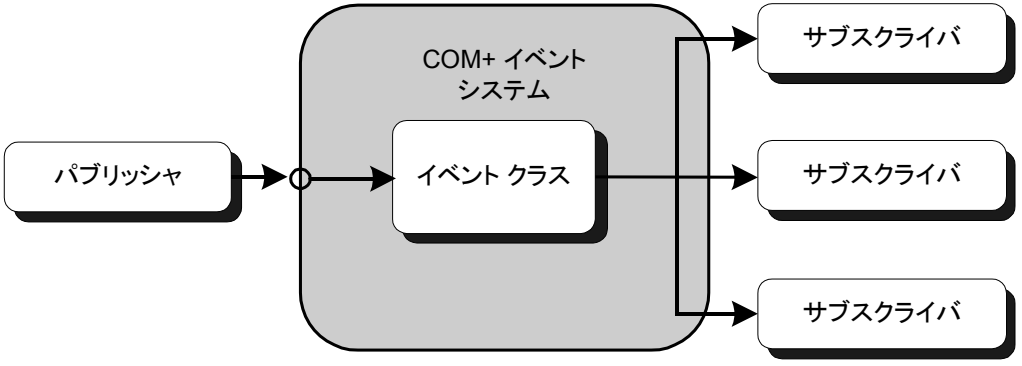

図 15.1
COM+ イベント システムの概念図

イベント クラスが定義しているインターフェイスをパブリッシャが呼び出し、サブスクライバが実装することで、イベントを分離するための主なメカニズムが形成されています。[Install new event class(es)] オプションを選択して、イベントコンポーネントを [Component Services] ダイアログ ボックス内にインストールする必要があります。

イベント クラスをインストールしたら、新しいイベントを受信するサブスクライバを作成し、イベント クラスで定義されているインターフェイスを実装する必要があります。

パブリッシャ (イベントを発行するオブジェクト) は、標準モジュール、ダイナミック リンク ライブラリ、または [Component Service] ダイアログ ボックスの [Install new class(es)] オプションをクリックして作成した通常の COM+ アプリケーションのいずれかとなります。クライアント層のユーザー アプリケーションは、この後イベントを取得するサブスクライバとなります。

以下のシナリオに、Visual Basic 6.0 の COM+ イベントのアップグレード プロセスを示します。

イベント コンポーネント

Visual Basic のイベント クラスは、他のインターフェイスと同様に定義します。生成するイベントの関数シグネチャを定義するだけです。実際には、イベント クラスは抽象クラスに埋め込まれたインターフェイスにすぎません。このコード例に示すように、モジュールは COM+ サービスのタイプ ライブラリへの参照を含んでいる必要があります。

```
Public Sub ContactInfoUpdated(name As String)
End Sub
```

イベント パブリッシャ

パブリッシャの作成も単純な作業です。イベントを生成するには、パブリッシャはイベント クラスのインスタンスを作成し、公開されているメソッドを呼び出します。プロジェクトに EventComponent と COM+ サービスのタイプ ライブラリへの参照を追加する必要があります。また、パブリッシャが、必ず COM+ カタログ内で COM+ アプリケーションとしてホストされるようにします。

```
Public Sub UpdateContact(name As String)
    ' ContactInfo を更新します。
    ' ...

    Dim e As EventComponent.EventClass
    Set e = CreateObject("EventComponent.EventClass")
    e.ContactInfoUpdated (name)
End Sub
```

イベント サブスクライバとテスト

イベント クラスをインストールしたら、新しいイベントを受信するサブスクライバを作成し、イベント クラスで定義されているインターフェイスを実装する必要があります。この実装で、サブスクライバはイベントに対するアクションを定義します。

サブスクライバに加えて、以下のコードにはイベントをテストするためのステートメントが含まれています。プロジェクトは、2 つのボタンを持った EXE アプリケーションである必要があります。

▶ プロジェクトと 2 つのボタンを持った EXE アプリケーションを作成するには

1. 以下のコードを使用してイベントにサブスクライブします。

```
Private Sub Command2_Click()
    Subscribe
End Sub
```

2. 以下のコードを使用してイベントをテストします。

```
Private Sub Command1_Click()
    Update "joe"
End Sub
```

3. 以下のコードを使用して、プロジェクトに EventComponent、PublisherComponent、COM+ サービスのタイプ ライブラリ、COM+ 管理のタイプ ライブラリへの各参照を追加します。

```
Implements EventComponent.EventClass

Private Sub Command1_Click()
    Update "joe"
End Sub

Private Sub Command2_Click()
    Subscribe
End Sub

Public Sub Update(name As String)
    Dim COM As Object
    Set COM = CreateObject("PublisherComponent.Publisher")
    COM.UpdateContact (name)
End Sub

Public Sub Subscribe()
    Dim subscriptions As ICatalogCollection
    Dim s As ICatalogObject
    Dim comAdm As COMAdmin.COMAdminCatalog

    Set comAdm = New COMAdmin.COMAdminCatalog
    Set subscriptions = comAdm.GetCollection("TransientSubscriptions")
    Set s = subscriptions.Add
    ' この例では、説明のために CLSID 値がハードコーディングされています。
```

```
        ' この CLSID 値は、Event クラスが含まれているライブラリを登録したときに
        ' 生成されます。値が生成された後は、レジストリから
        ' 読み込むことができます。
        s.Value("EventCLSID") = "{2D080D19-C950-4B0A-9009-67C30CF5DCEA}"
        s.Value("Name") = "Form subscription"
        s.Value("SubscriberInterface") = Me
        subscriptions.SaveChanges
    End Sub

    Private Sub EventClass_ContactInfoUpdated(name As String)
        MsgBox name
    End Sub
```

アップグレード ウィザードを使用して各モジュールをアップグレードすると、以下の Visual Basic .NET のコードが得られます。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("EventClass_NET.EventClass")> Public Class EventClass
    Public Sub ContactInfoUpdated(ByRef name As String)
    End Sub
End Class
```

イベント パブリッシャのコードを以下に示します。

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId("Business_NET.Business")> Public Class _
    Business
    Public Sub UpdateContact(ByRef name As String)
        Dim EventComponent As Object
        ' ContactInfo を更新します。
        ' ...

        Dim e As EventComponent.EventClass
        e = CreateObject("EventComponent.EventClass")
        ' UPGRADE_WARNING: Could not resolve default property of object
        ' e.ContactInfoUpdated. 詳細については、'ms-
        ' help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword=vbup1037'
        ' をクリックしてください。
        e.ContactInfoUpdated(name)
    End Sub
End Class
```

最後に、イベント サブスクライバとテストのコードを以下に示します。

```
Option Strict Off
Option Explicit On

Friend Class Form1
    Inherits System.Windows.Forms.Form
```

```
    Implements EventComponent.EventClass
#Region "Windows Form Designer generated code "
    ...
#End Region
#Region "Upgrade Support "
    ...
#End Region

    Private Sub Command1_Click(ByVal eventSender As System.Object, ByVal eventArgs _
        As System.EventArgs) Handles Command1.Click
        Update_Renamed("joe")
    End Sub

    ' UPGRADE_NOTE: Update は Update_Renamed にアップグレードされました。
    ' UPGRADE_NOTE: name は name_Renamed にアップグレードされました。
    Public Sub Update_Renamed(ByRef name_Renamed As String)
        Dim COM As Object

        COM = CreateObject("BusinessComponent.Business")
        ' UPGRADE_WARNING: Could not resolve default property of object
        ' COM.UpdateContact.
        COM.UpdateContact(name_Renamed)

    End Sub

    Public Sub Subscribe()
        Dim subscriptions As COMAdmin.ICatalogCollection
        Dim s As COMAdmin.ICatalogObject
        Dim comAdm As COMAdmin.COMAdminCatalog

        comAdm = New COMAdmin.COMAdminCatalog
        subscriptions = comAdm.GetCollection("TransientSubscriptions")
        s = subscriptions.Add
        ' UPGRADE_WARNING: Could not resolve default property of object s.Value().
        ' この例では、説明のために CLSID 値がハードコーディング
        ' されています。
        ' この CLSID 値は、Event クラスが含まれているライブラリを登録したときに
        ' 生成されます。値が生成された後は、レジストリから
        ' 読み込むことができます。
        s.Value("EventCLSID") = "{2D080D19-C950-4B0A-9009-67C30CF5DCEA}"
        ' UPGRADE_WARNING: Could not resolve default property of object s.Value().
        s.Value("Name") = "Form subscription"
        ' UPGRADE_WARNING: Could not resolve default property of object s.Value().
        s.Value("SubscriberInterface") = Me
        subscriptions.SaveChanges()

    End Sub

    Private Sub Command2_Click(ByVal eventSender As System.Object, ByVal eventArgs _
        As System.EventArgs) Handles Command2.Click
        Subscribe()
    End Sub
```

```
    ' UPGRADE_NOTE: name は name_Renamed にアップグレードされました。
    Private Sub EventClass_ContactInfoUpdated(ByRef name_Renamed As String) _
        Implements EventComponent.EventClass.ContactInfoUpdated
        MsgBox(name_Renamed)
    End Sub
End Class
```

Visual Basic 6.0 の COM+ イベントの機能を実現するには、アップグレード後のすべてのプロジェクトを変更する必要があります。以下のコンポーネントでは、.NET で COM+ イベントを構築していることを分かりやすくするために、いくつかの名前を変更しています。

イベント インターフェイス

イベントコンポーネントはクラスライブラリであり、イベント インターフェイス インフラストラクチャを定義します。この特定のコンポーネントについては、ServicedComponent クラスを拡張する必要はありませんが、アセンブリに厳密な名前を追加する必要があります。

▶ アセンブリに厳密な名前を追加するには

1. 厳密名ツール (Sn.exe) を使用して、コンポーネントのキー ペア ファイルを作成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

   ```
   sn -k MyKey.snk
   ```

2. AssemblyInfo.vb ファイルに AssemblyKeyFile 属性を追加し、サーバー コンポーネントとキー ペア ファイルをリンクします。

   ```
   <Assembly: AssemblyKeyFile("MyKey.snk")>
   ```

以下のコード例は、イベントコンポーネントを示します。

```
Public Interface IEvent
    Sub ContactInfoUpdated(ByVal name As String)
End Interface
```

イベント クラス

次のステップは、前述のインターフェイスを実装するイベント クラスを作成することです。これについて、以下の手順で詳しく説明します。

▶ イベント クラスを作成するには

1. イベント インターフェイスへの参照を追加します。
2. インターフェイスを実装します。
3. プロジェクトに EnterpriseServices への参照を追加し、対応する Imports ステートメントをクラスに追加します。

   ```
   Imports System.EnterpriseServices
   ```

4. ServicedComponent クラスから継承するようにクラスを変更します。

```
Inherits ServicedComponent
```

5. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

```
sn -k MyKey.snk
```

7. クラスに EventClass 属性を追加します。この属性は、属性クラスをイベント クラスとしてマークします。イベント クラスに対するメソッド呼び出しは、実装には渡されず、代わりにイベント サブスクライバに渡される点に注意してください。
8. クラスに EventTrackingEnabled 属性を追加します。この属性は、コンポーネントのイベント トラッキングを有効にします。

```
Imports System.EnterpriseServices

<EventClass(), EventTrackingEnabled()> _
Public Class COMEventClass
    Inherits ServicedComponent
    Implements EventInterface.IEvent

Public Sub ContactInfoUpdated(ByVal name As String) Implements
EventInterface.IEvent.ContactInfoUpdated
    End Sub
End Class
```

パブリッシャ

次に、COM+ アプリケーションでホストされるパブリッシャ コンポーネントを作成します。これについて、以下の手順で詳しく説明します。

▶ COM+ アプリケーションでホストされるパブリッシャ コンポーネントを作成するには

1. イベント インターフェイスへの参照を追加します。
2. プロジェクトに EnterpriseServices への参照を追加し、対応する Imports ステートメントをクラスに追加します。

```
Imports System.EnterpriseServices
```

3. ServicedComponent クラスから継承するようにクラスを変更します。

```
Inherits ServicedComponent
```

4. 厳密な名前を使用してコンポーネントに署名するために、対応する AssemblyKeyFile 属性を AssemblyInfo.vb ファイルに追加します。次のコマンドを実行してキー ファイルを生成します (厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください)。

```
sn -k MyKey.snk
```

以下のコードは、Publisher クラスの例です。

```
Imports System.EnterpriseServices

Public Class Publisher
    Inherits ServicedComponent

    Public Sub Update(ByVal name As String)
        Dim evt As New EventClass.COMEventClass
        evt.ContactInfoUpdated(name)
    End Sub
End Class
```

イベントとパブリッシャの準備が整ったら、サブスクライバ/テスト アプリケーションをアップグレードする必要があります。このアプリケーションは、Visual Basic 6.0 アプリケーションと同じく 2 つのボタンを持った Windows フォーム アプリケーションです。

▶ アプリケーションを作成するには

1. イベント インターフェイスへの参照を追加します。
2. COM+ 管理のタイプ ライブラリへの参照を追加します。
3. Type.GetTypeFromProgID メソッドを使用してイベント クラスの型を取得し、その後 GUID プロパティを使用して COM を取得します。

   ```
   Type.GetTypeFromProgID().GUID
   ```

   Subscribe メソッド内のステートメントが COM+ 管理のタイプ ライブラリのオブジェクトを使用するため、それ以上の変更を行う必要はありません。

以下のコード例は、アプリケーション コードを示します。

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Implements EventInterface.IEvent

#Region " Windows Form Designer generated code "
    ...
#End Region

#Region "Upgrade Support "
    ...
#End Region
```

```
    Public Sub Update_Renamed(ByRef name_Renamed As String)
        Dim COM As Object
        COM = CreateObject("PublisherComponent_NET.Publisher")
        COM.UpdateContact(name_Renamed)
    End Sub

    Public Sub Subscribe()
        Dim subscriptions As COMAdmin.ICatalogCollection
        Dim s As COMAdmin.ICatalogObject
        Dim comAdm As COMAdmin.COMAdminCatalog

        comAdm = New COMAdmin.COMAdminCatalog
        subscriptions = comAdm.GetCollection("TransientSubscriptions")
        s = subscriptions.Add
        s.Value("EventCLSID") = "{" & _
            Type.GetTypeFromProgID("EventClass.COMEventClass").GUID.ToString() &
"}"
        s.Value("Name") = "Form subscription"
        s.Value("SubscriberInterface") = Me
        subscriptions.SaveChanges()
    End Sub

    Public Sub ContactInfoUpdated(ByVal name As String) Implements _
        EventInterface.IEvent.ContactInfoUpdated
        MsgBox(name)
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
        System.EventArgs) Handles Button1.Click
        Subscribe()
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As _
        System.EventArgs) Handles Button2.Click
        Update_Renamed("Joe")
    End Sub
End Class
```

メッセージ キューとキュー コンポーネント

メッセージ キュー (MSMQ とも呼ばれます) は、アプリケーション間の柔軟で信頼性の高い通信をサポートするプラットフォームを構成します。さまざまな程度の信頼性を持つネットワーク内のアプリケーション間でメッセージを送信できる通信環境を実装することで、さまざまなアプリケーションを統合できます。メッセージ キューを使用すると、システム内に信頼性が低いコンポーネントがあっても、異なる時点で動作するアプリケーションが、異機種ネットワークを通じて、高い信頼性で通信することができます。

タスクは、それが呼び出された時点とは非同期に起動され、タスクが終了して結果が生成されるのを待つことはありません。この実行方法を使用することで、アプリケーションの応答性が高まり、使用可能なリソースが有効に利用できるようになります。この処理モードは、COM+ キュー コンポーネントまたはメッセージ キューを使用して実現できます。

メッセージ キューを使用すると、アプリケーションはデータをキューに格納できます。キューは、アプリケーションが間接的に通信できるようにするためのメッセージ ストアです。キューは、不要になるまでメッセージを保持します。キューに格納されているデータは、生成されたアプリケーションで取得することも、他のアプリケーションで取得することもできます。メッセージ キューのキューは、メッセージを取得したサーバー プロセスによって非同期的に実行されるタスクを表すメッセージを受け取ることができます。

信頼性が低い分散コンピューティング環境では、指定された時間でトランザクションに関係するすべてのサーバーが使用できない可能性があります。たとえば、顧客の注文書は、必要なリモート SQL サーバーが使用可能になるまで、一時的にメッセージ ストアに保存する必要があります。メッセージ キューは、このような状況を念頭に設計および開発されています。

最近までは、メッセージ キューで提供されているサービスにアクセスする主な方法は、MSMQ COM API と C API を使用する方法でした。COM API は、Visual Basic 6.0 などの COM クライアントから使用できますが、COM の相互運用機能を使用すれば Visual Basic .NET からもアクセスできます。ただし、.NET Framework の採用により、System.Messaging 名前空間では、管理された環境で MSMQ にアクセスするための新しい便利な方法が提供されています。System.Messaging 名前空間の詳細については、MSDN の『.NET Framework Class Library』の「System.Messaging Namespace」を参照してください。

メッセージ キューは、次のようなさまざまな種類のキューをサポートしています。

- **パブリック キュー。** これらのキューには、だれでもアクセスできます。通常、パブリック キューは、システム管理者によって作成および構成されます。
- **専用キュー。** これらのキューは、アクセス対象のキューの完全パスに関するデータを持っているアプリケーションからアクセスできます。
- **トランザクション キュー。** これらのキューは、トランザクション メッセージのターゲットとして使用できます。トランザクションの一部としてメッセージを送信することで、順番に配信されること、一度だけ配信されること、および送信先のキューから正常に取得されることが保証されます。これらの条件のうちどれかが満たされない場合、トランザクションは不完全となり、.NET Framework の MessageQueueTransaction.Abort メソッドが呼び出されて、トランザクション全体がロールバックされます。トランザクション キューと Microsoft Transaction Server (MTS) を組み合わせて使用して、メッセージを MTS トランザクションの一部にすることができます。.NET でのトランザクション キューの使用の詳細については、MSDN の『.NET Framework Class Library』の「MessageQueue.Transactional Property」を参照してください。

System.Messaging 名前空間では、メッセージ キューで提供される機能のほとんどに開発者がアクセスできるクラスのグループが提供されています。表 15.5 に、MSMQ COM API で使用可能なさまざまなクラスをアップグレードする際の推奨事項を記載します。

表 15.5: メッセージ キューの COM API クラスを Visual Basic .NET にアップグレードする方法

MSMQ COM API クラス	推奨されるアップグレード方法
MSMQApplication メッセージ キュー アプリケーション オブジェクトを実装します。グローバルな機能を提供します。	対応するクラスはありません。再実装が必要です。
MSMQCoordinatedTransactionDispenser DTC トランザクション ディスペンサを実装します。新しい DTC トランザクションの作成をサポートします。	対応するクラスはありません。再実装が必要です。
MSMQDestination メッセージ キューの送信先を表します。	メッセージの送信先は、System.Messaging.MessageQueue オブジェクト自体が示します。この Send メソッドは、送信先オブジェクトを受け取りません。MSMQ COM API では、MSMQMessage クラスで Send メソッドが定義されていましたが、このメソッドはメッセージ キューの一部となりました。
MSMQEvent 送信する非同期イベントを表します。非同期メッセージの到着を通知するために使用します。	非同期メッセージの到着通知を取得するには、System.Messaging.MessageQueue.PeekCompleted イベントを使用することができます。
MSMQManagement 送信キューとターゲット キューの共通の管理機能をカプセル化します。	対応するクラスはありません。再実装が必要です。
MSMQMessage メッセージを表します。	System.Messaging.Message
MSMQOutgoingQueueManagement 送信キューの管理機能をカプセル化します。	対応するクラスはありません。再実装が必要です。
MSMQQuery パブリック キューを見つけるために使用されるメッセージ キューの検索機能を提供します。	クエリ条件は System.Messaging.MessageQueueCriteria で指定します。クエリは System.Messaging.MessageQueue.GetPublicQueues メソッドで実行します。
MSMQQueue メッセージの取得をサポートしている開いているキューを表します。	System.Messaging.MessageQueue のサブセットです。
MSMQQueueInfo キューを表します。キューを作成、削除、開くために使用します。	System.Messaging.MessageQueue のサブセットです。

MSMQ COM API クラス	推奨されるアップグレード方法
MSMQQueueInfos MSMQQuery.LookupQueue で生成されたキューのコレクションを表します。	System.Messaging.MessageQueueEnumerator
MSMQQueueManagement ターゲット キューの管理機能をカプセル化します。	対応するクラスはありません。再実装が必要です。
MSMQTransaction メッセージ キュートランザクション オブジェクトを実装します。	System.Messaging.MessageQueueTransaction
MSMQTransactionDispenser メッセージ キュートランザクション ディスペンサを実装します。	新しいトランザクションは、コンストラクタ System.Messaging.MessageQueueTransaction.New で作成します。トランザクション オブジェクトは、MessageQueue.Send と MessageQueue.Receive のパラメータとして使用します。

以下の例は、通信媒体としてメッセージ キューのパブリック キューを使用した、メッセージの送信者とメッセージの受信者のプロシージャの Visual Basic 6.0 のソースコードを示します。

```
Private Sub Send_Click()
    Dim dest As New MSMQDestination
    Dim msg As New MSMQMessage
    dest.ADsPath = _
        "LDAP://CN=MSMQTest,CN=msmq,CN=MyComputer,CN=Computers,DC=MyDomain,DC=com"
    dest.Open

    msg.Body = "This is a test"
    msg.Send dest
End Sub

Private Sub Receive_Click()
    Dim QueueQuery As New MSMQQuery
    Dim QueueCollection As New MSMQQueueInfos
    Dim QueueInfo As New MSMQQueueInfo
    Dim MsgQueue As New MSMQQueue
    Dim msg As New MSMQMessage

    Set QueueCollection = QueueQuery.LookupQueue(, , "MSMQTest")
    QueueCollection.Reset
    Set QueueInfo = QueueCollection.Next()
    Set MsgQueue = QueueInfo.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)
    Set msg = MsgQueue.Receive()
    MsgBox "Received message: " & msg.Body
End Sub
```

Send_Click イベント ハンドラで使用されるパス文字列 ADsPath に含まれているさまざまな部分に注意してください。指定した引数は、パブリック キュー名 (MSMQTest)、メッセージ キュー プロバイダ (msmq)、サーバー名 (MyComputer)、パスがメッセージ キューを参照することを示すインジケータ (Computers)、およびドメイン名 (MyDomain) です。上の例では、コンピュータ管理構成アプリケーションを使用してシステム管理者が作成したパブリック キューを使用しています。キューを作成する手順については、次のセクションで説明します。

メモ：この手順を実行するには、コンピュータにメッセージ キューがインストールされている必要があります。メッセージ キューのインストールの詳細については、MSDN の「How to Install MSMQ 2.0 to Enable Queued Components」を参照してください。

▶ パブリック キューを作成するには

1. Windows のタスク バーで、**[スタート]** をクリックし、次に **[コントロール パネル]** をクリックします。
2. [コントロール パネル] で、**[管理ツール]** をダブルクリックします。
3. **[コンピュータの管理]** をダブルクリックします。
4. [サービスとアプリケーション] フォルダをクリックします。
5. [Message Queuing] フォルダをダブルクリックします。
6. [パブリック キュー] フォルダを右クリックし、次に [New\Public Queue] をクリックします。
7. [新しいオブジェクト] ウィンドウで、[Queue name] ボックスに「MSMQTest」と入力し、[OK] をクリックします。

このコード サンプルを、以下の Visual Basic .NET のコードに手作業でアップグレードできます。

```
Private Sub Send_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles Send.Click

    Dim msgQueue As MessageQueue
    Dim msg As New Messaging.Message
    msgQueue = New MessageQueue(".\MSMQTest")
    msg.Formatter = New XmlMessageFormatter
    msg.Body = "This is a test"
    msgQueue.Send(msg)
End Sub

Private Sub Receive_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Receive.Click

    Dim msgQueue As MessageQueue
    Dim msg As Messaging.Message
    Dim types(0) As String
    msgQueue = New MessageQueue(".\MSMQTest")
    msg = msgQueue.Receive()
    types(0) = "System.String"
```

```
    msg.Formatter = New XmlMessageFormatter(types)
    MessageBox.Show("Received message: " & msg.Body)
End Sub
```

異なる形式 .\MSMQTest を使用してメッセージ キューを識別している点に注意してください。これは、現在のコンピュータ上にあるパブリック メッセージを指定します。新しい形式は「マシン名\キュー名」です。MessageQueue クラスの詳細については、MSDN の『.NET Framework Class Library』の「MessageQueue Class」を参照してください。もう 1 つの重要な違いは、オブジェクトをメッセージ本体にシリアル化したり、メッセージ本体から逆シリアル復元したりするフォーマッタを示す Message.Formatter プロパティを使用している点です。このフォーマッタにより、Visual Basic 6.0 から送信されるメッセージとはメッセージの内容が異なります。Visual Basic .NET から送信されたメッセージを Visual Basic 6.0 アプリケーションで受信することはできますが、メッセージの内容を読み取るには追加の処理が必要です。Visual Basic 6.0 アプリケーションは、メッセージで本来送信された情報を取得するために、内部的な XML 形式を解析する必要があります。逆方向で通信する際にも追加の処理が必要です。Visual Basic 6.0 で記述されたアプリケーションと Visual Basic .NET で記述されたアプリケーションが通信する必要がある場合に推奨される方法は、MSMQ の通信層全体を一貫した状態に保つことです。たとえば、アプリケーションの通信層のすべてを Visual Basic 6.0 または Visual Basic .NET で記述します。その場合、他の言語で記述されたコンポーネントとローカルにやり取りする新しい通信クラスを作成することが必要になる場合があります。

COM+ キュー コンポーネント サービスは、メッセージ キューを使用して、非同期でコンポーネントを呼び出して実行する効率的な方法を提供します。非同期クラスを System.EnterpriseServices.ServicedComponent から継承することで、このような処理を比較的簡単に設定できます。

以下の例に示すように、MaxListenerThreads プロパティは、**キュー コンポーネント**のリスナ スレッドの同時最大数を示します。この例は、メッセージを非同期で表示する QueuedComp クラスをサーバーに実装する方法を示します。また、リモートの**キュー コンポーネント** に対して DispMsg メソッドを呼び出すクライアント メソッドも示します。以下のコード例に、その方法を示します。

初めに、サーバー コードを示します。

```
Imports System.Reflection
Imports System.EnterpriseServices
Imports System
Namespace QueuedCompDemo
    Public Interface IQueuedComp
        Sub DispMsg(msg As String)
    End Interface
    <InterfaceQueuing(Interface := "IQueuedComp")> _
    Public Class QueuedComp
    Inherits ServicedComponent Implements IQueuedComp
        Public Sub DispMsg(msg As String) implements _
```

```
        IQueuedComp.DispMsg
            MessageBox.Show(msg, "Processing message")
        End Sub
    End Class
End Namespace
```

サーバー プロジェクトでは、AssemblyInfo.vb ファイルに以下の行を含める必要がある点に注意してください。

```
<Assembly: ApplicationName("QueuedCompDemoSvr")>
<Assembly: ApplicationActivation(ActivationOption.Server)>
<Assembly: ApplicationQueuing(Enabled := True, _
            QueueListenerEnabled := True)>
<Assembly: AssemblyKeyFile("QueuedCompDemoSvr.snk")>
```

次に、クライアントコードを示します。

```
Protected Sub Send_Click(sender As Object, e As System.EventArgs) _
Handles send.Click
    Dim iQc As IQueuedComp = Nothing
    Try        iQc = CType(Marshal.BindToMoniker( _
        "queue:/new: QueuedCompDemo.QueuedComp"), _
        IQComponent)
    Catch l as Exception
        Console.Writeline("Caught Exception: " & l.Message)
    End Try
    iQc.DispMsg("Message test")
    Marshal.ReleaseComObject(iQc)
End Sub
```

サーバー コンポーネントは、厳密な名前で署名する必要がある点に注意してください。厳密な名前の詳細については、この章の「Visual Basic .NET での COM+ の使用」を参照してください。

まとめ

MTS テクノロジと COM+ テクノロジを使用すると、ビジネス ニーズを満たす分散アプリケーションを構築できます。Visual Basic 6.0 を使用してこれらのアプリケーションが構築されている場合は、Visual Basic .NET にアップグレードできます。そのためには、COM+ アプリケーションの種類、SOAP、COM+ の配置などのいくつかの環境変数を考慮する必要がありますが、アップグレード方法と関連する問題を理解することで、これらの重要なテクノロジのアップグレード作業を円滑に行うことができます。この章で説明した手法を適用すれば、現在ある Visual Basic 6.0 の MTS アプリケーションと COM+ アプリケーションを、可能な限り少ない作業で Visual Basic .NET にアップグレードできます。

詳細情報

厳密名ツールの詳細については、MSDN の「.NET Framework Tools Strong Name Tool (Sn.exe)」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpgrfstrongnameutilitysnexe.asp です。

System.Messaging 名前空間の詳細については、MSDNの「.NET Framework Class Library System.Messaging」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemmessaging.asp です。

.NET でのトランザクション キューの使用の詳細については、MSDN の『.NET Framework Class Library』の「MessageQueue.Transactional Property」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemmessagingmessagequeueclasstransactionaltopic.asp です。

メッセージ キューのインストールの詳細については、MSDN の「How to Install MSMQ 2.0 to Enable Queued Components」を参照してください。
URL は http://support.microsoft.com/default.aspx?scid=kb;en-us;256096 です。

MessageQueue クラスの詳細については、MSDN の『.NET Framework Class Library』の「MessageQueue Class」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemmessagingmessagequeueclasstopic.asp です。

16

アプリケーションの完成

アップグレード プロセスを完了する前に、エンド ユーザーがより簡単にアプリケーションを使用できるようにアプリケーションのいくつかの側面を修正する必要があります。これには、配置を改善するためのアセンブリの再構成、統合ヘルプのアップグレード、実行時の依存関係の管理、および配置オプションへの対処が含まれます。これらの修正は自動的には処理できないため、手動で行う必要があります。

この章では、これらの修正を完了するために必要な情報を紹介します。

アセンブリの分割

アセンブリは、.NET Framework アプリケーションの主要なビルド ブロックであり、単一の実装単位としてビルド、バージョン管理、および配置されるコンポーネント ライブラリです。すべてのアセンブリに、アセンブリを記述するマニフェストが含まれています。

システム アーキテクトは、効率と信頼性の両方を確保できるように、アプリケーションの配置方法を慎重に計画する必要があります。アプリケーションおよび配置先プラットフォームのさまざまな側面を考慮に入れて、各アセンブリを実行するための適切な場所を選択します。以降では、考えられる選択肢の概要と、各選択肢の長所と短所について説明します。アセンブリとその機能の詳細については、第 4 章「一般的なアプリケーションの種類」の「ネイティブ DLL とアセンブリ」を参照してください。

アセンブリを分割しない

コンポーネントの凝集度が低いにもかかわらずアプリケーションやコンポーネントが単一のアセンブリとして実装されていると、開発、配置、およびサポートの各側面に悪影響が及ぶ可能性があります。このような場合にアプリケーションの一部が変更されて、その部分の新しいバージョンをインストールしなければならなくなると、重大な配置の問題になります。Visual Basic 6.0 ではこの種の構成がサポートされていましたが、これは、開発者が定義できるユーザー クラスが 1 つのファイルにつき 1 つだけだったためです。

Visual Basic .NET では、1 つのアセンブリに複数のクラスを含めることができます。このため開発者は、高い凝集度を持つ関連クラスのグループを作成できます。これにより、機能グループ全体をまとめて管理できるため、間違ったバージョンのコンポーネントが含まれることによってアプリケーションが破損するリスクを抑えることができます。

逆に、アセンブリをたった 1 つのコンポーネントやクラスのコンテナとして使用すると、アプリケーションが必要とするシステム リソースが増加します。これは、コンポーネントが増えるとアセンブリが増え、アセンブリが増えるたびに .NET 共通言語ランタイムの処理の作業負荷が増加するからです。

アプリケーション層による分割

小規模および中規模のプロジェクトでは、アプリケーションの各層に対応するアセンブリへとコンポーネントをまとめることができます。アプリケーション層によるアセンブリの分割は、アプリケーションのさまざまな層が物理的に分割されていることの自然な結果です。アプリケーション層については、第 4 章「一般的なアプリケーションの種類」の「デスクトップ アプリケーションと Web アプリケーション」を参照してください。

1 つの層を、1 つまたは複数の層に対応する複数のアセンブリの間でさらに分割する場合、その主な理由となるのはアセンブリのサイズと複雑さです。アセンブリのサイズが大きく、一部の機能がほとんど使用されない場合は、アセンブリを機能層に従って分割できます。配置やシステム管理の問題が軽減されるのであれば、プレゼンテーション層、ビジネス ロジック層、およびデータ アクセス層を同じアセンブリに含めることも検討する必要があります。

機能による分割

アプリケーションが複雑になってくると、アセンブリを機能別のグループにまとめる必要が出てきます。たとえば、ビジネス ロジックとデータ アクセス機能を 1 つのアセンブリにまとめて複数のアプリケーションで共有することができます (異なる層にある複数のアセンブリを共有するより簡単です)。アセンブリを機能で分割すると、新機能を開発する開発者が、その新機能と同じ概念レベルで動作するブロックを使用できるため、デザインの面でも有効です。

統合ヘルプのアップグレード

Visual Basic 6.0 アプリケーションにヘルプ システムのサポートが含まれている場合 (ユーザーが F1 キーを押すとヘルプ ドキュメントが表示される場合など)、Visual Basic アップグレード ウィザードで自動的にアップグレードすることはできません。HelpFile プロパティ (ヘルプ ドキュメントの名前と場所を指定するために使用されます) と HelpContextID プロパティ (ヘルプ ファイルの特定のトピックをフォーム上の各コントロールに関連付けるために使用されます) は、変更されずに残ります。また、Visual Basic .NET ではヘルプ システムが変更されているため、これらのプロパティが検出されるたびにコンパイル エラーが生成されます。たとえば、Visual Basic .NET ではヘルプ サポートはフォームごとに実装され、1 つ以上の HelpProvider コンポーネン

トがフォームに追加されます。特定のトピックにアクセスするには、デザイン時に各フォームやコントロールの HelpKeyword プロパティと HelpNavigator プロパティを使用するか、実行時に SetHelpKeyword メソッドと SetShowHelp メソッドを使用します。

以下の Visual Basic 6.0 コードには、2 つの TextBox コントロールが含まれています。アプリケーションのヘルプ ファイルとヘルプ トピックのプロパティを設定する App オブジェクトは、実行時に設定されます。

メモ:この例をテストする場合は、特定のコンテキスト ID を持つセクションを含むコンパイル済み HTML ヘルプ ファイル (.chm) を用意する必要があります。このコード サンプルで使用されているファイルの名前は MyHelp.chm です。このファイルには 2 つのセクションが含まれています。1 つ目のセクションのコンテキスト ID は 1000 で、2 つ目のセクションのコンテキスト ID は 1001 です。このヘルプ ファイルのパスとヘルプ コンテキスト ID の定数を、用意したファイルに合わせて置き換える必要があります。

```
' MyHelp.chm の実際のコンテキスト番号です。
' ヘルプ コンテキスト ID の定数を定義します
Const HelpUserName = 1000
Const HelpPWD = 1001

Private Sub Form_Load()
    App.HelpFile = "C:\MyHelp.chm"

    Text1.HelpContextID = HelpUserName
    Text2.HelpContextID = HelpPWD
End Sub
```

アップグレード ウィザードを使用してこのコードをアップグレードすると、結果は次のようになります。

```
Option Strict Off
Option Explicit On
Friend Class Form1
    Inherits System.Windows.Forms.Form
    …
    Const HelpUserName As Short = 1000
    Const HelpPWD As Short = 1001

    Private Sub Form1_Load(ByVal eventSender As System.Object, _
                           ByVal eventArgs As System.EventArgs) Handles MyBase.Load
        'UPGRADE_ISSUE: App プロパティ App.HelpFile はアップグレードされませんでした。
        App.HelpFile = "C:\MyHelp.chm"

        'UPGRADE_ISSUE: TextBox プロパティ Text1.HelpContextID はアップグレードされませんでした。
        Text1.HelpContextID = HelpUserName
        'UPGRADE_ISSUE: TextBox プロパティ Text2.HelpContextID はアップグレードされませんでした。
        Text2.HelpContextID = HelpPWD
    End Sub
End Class
```

問題のあるステートメントは元のコードのまま変更されていない点に注意してください。プロジェクトをビルドする前にこれらのコンパイル エラーに対処する必要があります。

実行時のヘルプの統合

実行時のヘルプ エラーを修正し、Visual Basic .NET アプリケーションにヘルプ サポートを追加するには、次の手順に従ってください。

▶ 実行時にヘルプを統合するには

1. ヘルプを表示するフォームに HelpProvider コンポーネントを追加します。図 16.1 はその方法を示しています。

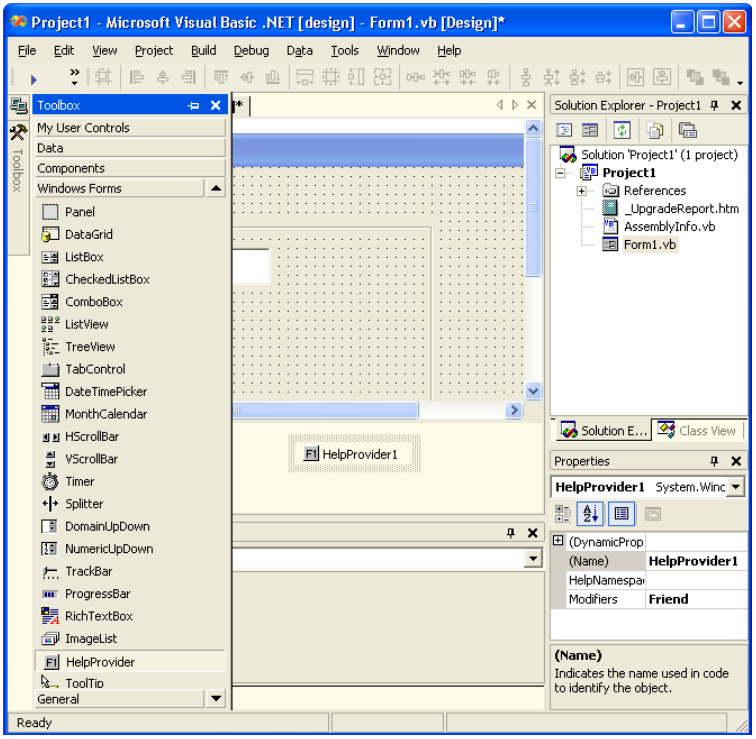

図 16.1

フォームにHelpProvider コンポーネントを追加

2. App オブジェクトの HelpFile プロパティを HelpProvider の HelpNamespace プロパティに置き換えます。そのためには、次の行を置き換えます。

```
App.HelpFile = "C:\MyHelp.chm"
```

この行を次の行に置き換えます。

```
Me.HelpProvider1.HelpNamespace = "C:\MyHelp.chm"
```

3. HelpContextID プロパティは、ヘルプ ファイルの構成がわからないとアップグレードできません。Visual Basic 6.0 では、関連するコンテキスト番号を使用しますが、Visual Basic .NET では、必要な情報を含むドキュメントの名前を使用する必要があります。この例では、2 つの定数を使用して、表示する特定のトピックを識別しています。定数 HelpUserName はドキュメント UserName.htm を表示し、定数 HelpPWD は PWD.htm を表示します。

   アップグレード後のコードで、これらの定数を参照している行を、適切なページを参照するように変更します。まず、各定数の型を Short から String に変更し、その後、値を適切なドキュメントの名前に置き換えます。これは、以下のコード例のようになります。

```
Const HelpUserName As String = "UserName.htm"
Const HelpPWD As String = "PWD.htm"
```

4. 各オブジェクトの HelpContextID プロパティを、手順 1. で追加した HelpProvider オブジェクトの SetHelpKeyword、SetHelpNavigator、および SetShowHelp の各メソッドに置き換えます。変更後のコードは次のようになります。

```
Me.HelpProvider1.SetHelpKeyword(Me.Text1, HelpUserName)
Me.HelpProvider1.SetHelpNavigator(Me.Text1, HelpNavigator.Topic)
Me.HelpProvider1.SetShowHelp(Me.Text1, True)

Me.HelpProvider1.SetHelpKeyword(Me.Text2, HelpPWD)
Me.HelpProvider1.SetHelpNavigator(Me.Text2, HelpNavigator.Topic)
Me.HelpProvider1.SetShowHelp(Me.Text2, True)
```

これらの変更が完了すると、新しいバージョンのアプリケーションで元のバージョンと同じようにヘルプが表示されるようになります。

デザイン時のヘルプの統合

Visual Basic 6.0 アプリケーションにヘルプを組み込むためのテクニックはもう 1 つあります。前の例では、実行時にヘルプを統合する方法を紹介しましたが、Visual Basic 6.0 の [Project Properties] ダイアログ ボックスのプロパティを設定することによって、デザイン時にヘルプ ファイルを含めることもできます。図 16.2 はその方法を示しています。

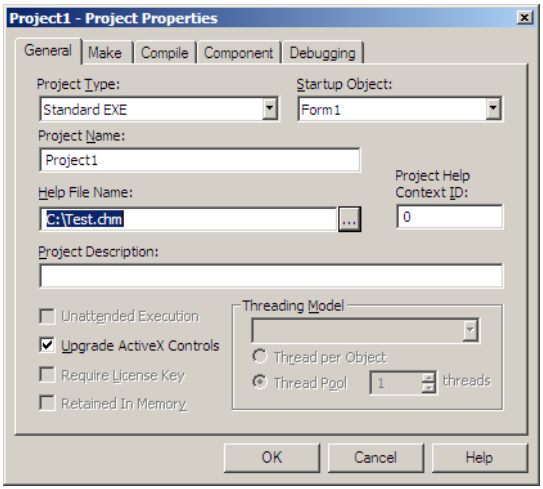

図 16.2

[Project Properties] *ダイアログ ボックスで*Visual Basic 6.0 *プロジェクトのヘルプ ファイルを指定*

この場合は、プロパティ ウィンドウでコントロールの HelpConmtextID プロパティを変更する必要があります (図 16.3 を参照)。

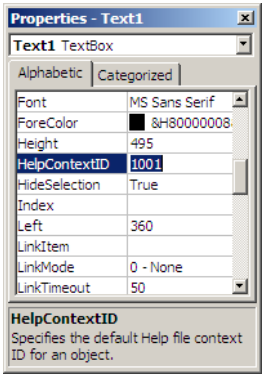

図 16.3

Visual Basic 6.0 *コントロールの*HelpContextID *を設定*

この例は、2 つの TextBox を持つ、前の例のフォームに基づいています。これを Visual Basic 6.0 で実行できるようにするには、[Project Properties] ダイアログ ボックスの Help File Name プロパティに有効なヘルプ ファイルのパスを追加する必要があります。その後、各 TextBox の HelpContextID プロパティを、このヘルプ ファイル内の有効なエントリに設定します。これにより、実行時に値を設定した前の例の場合と同じように、値が正しく設定されます。

Visual Basic .NET Upgrade Wizard でこの例をアップグレードすると、アプリケーションはコンパイルも実行もできますが、ヘルプは利用できるようになりません。アップグレード後のアプリケーションでヘルプを利用でき

るようにするには、実行時にヘルプを統合するための手順と同様の手順を実行する必要があります。

▶ デザイン時にヘルプを統合するには

1. HelpProvider コンポーネントを追加します。
2. HelpProvider の HelpNamespace プロパティを変更して、ヘルプ ファイルへのパスを設定します (図 16.4 を参照)。

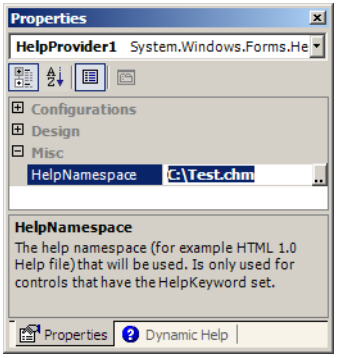

図 16.4

HelpProvider.HelpNamespace *プロパティのパスを指定*

3. HelpKeyword としてトピック ID を指定し、HelpNavigator の値を Topic に設定します (図 16.5 を参照)。

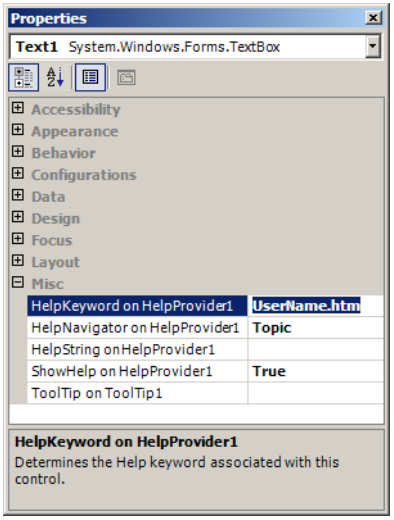

図 16.5

Visual Basic .NET *コントロールのヘルプ プロパティを設定*

HelpProvider クラスは、コントロールのプロパティを次のように拡張することによって、コントロールのヘルプを提供します。

- HelpKeyword プロパティは、HelpNamespace によって指定されているヘルプ ファイルからそのコントロールに関連するヘルプを取得するためのキーの情報を提供します。
- HelpNavigator プロパティは、指定したコントロールのヘルプをヘルプ ファイルから取得するときに使用するヘルプ コマンドを指定します。
- HelpString プロパティは、指定したコントロールに関連付けられている文字列を指定します。
- ShowHelp プロパティは、指定したコントロールに対してヘルプが表示されるかどうかを指定します。

詳細については、MSDN の『.NET Framework Class Library』の「HelpProvider Class」を参照してください。

WinHelp から HTML へのアップグレード

Visual Basic .NET がサポートしているのは HTML ヘルプ形式だけです。したがって、既存のヘルプ プロジェクトが Windows ヘルプ プロジェクト (.hpj ファイル) に基づいている場合は、ソース コードをアップグレードする前に変更する必要があります。

▶ Windows ヘルプを HTML 形式にアップグレードするには

1. Visual Basic 6.0 の一部として含まれている HTML Help Workshop ツールを開きます。
2. 新しいプロジェクトを作成します。
3. [Convert WinHelp project] をクリックし、[Next] をクリックします。
4. 最初のフィールドに、変換する Windows ヘルプ ファイルの名前を入力します。
5. その次のテキスト フィールドに、HTML ファイルを保存するフォルダの名前を入力します。または、[Browse] をクリックしてフォルダを選択し、プロジェクトの名前を入力します。
6. [Finish] をクリックすると、変換が開始されます。HTML Help Workshop ツールは、WinHelp プロジェクト (.hpj) ファイルを HTML ヘルプ プロジェクト (.hhp) ファイルに、WinHelp トピック (.rtf) ファイルを HTML ヘルプ トピック (.htm、.html) ファイルに、WinHelp 目次 (.cnt) ファイルを HTML ヘルプ目次 (.hhc) ファイルに、WinHelp インデックスを HTML ヘルプ インデックス (.hhk) ファイルに、それぞれ変換します。WinHelp のアート ファイル (.bmp ファイルまたは .wmf ファイル) は、対象のブラウザ プログラムが .png イメージ ファイルをサポートしていない場合 (Internet Explorer 3.0 など) は HTML イメージ ファイル (.gif ファイルまたは .jpeg ファイル) に、サポートしている場合 (Internet Explorer 4.0 など) は .png イメージ ファイルに変換されます。
7. 変換プロセスが完了したら、[Compile] メニュー オプションを使用してプロジェクトをコンパイルします。これにより、コンパイル済み HTML ヘルプ (.chm) ファイルが作成されます。

コンテキスト ヘルプの統合

フォームの WhatsThisButton プロパティと WhatsThisHelp プロパティを使用することによって、Visual Basic .NET アプリケーションにコンテキスト (ポップアップ) ヘルプを実装できます。これらのプロパティを True に設定すると、[What's This] ボタンが表示されます。そのプロパティの値を次のように設定します。

- ControlBox プロパティを True に設定します。
- BorderStyle プロパティを Fixed Single または Sizable に設定します。
- MinButton と MaxButton を False に設定するか、BorderStyle プロパティを Fixed Dialog に設定します。

アップグレード ウィザードでは WhatsThisButton プロパティと WhatsThisHelp プロパティはアップグレードされませんが、Visual Basic .NET では、ポップアップ ヘルプの機能は Windows フォームの HelpButton プロパティによって実装されます。Visual Basic .NET でヘルプ ボタンが表示されるのは、HelpButton プロパティが True に設定され、MaximizeBox プロパティと MinimizeBox プロパティがどちらも False に設定されている場合だけです。

実行時の依存関係

Visual Basic 6.0 アプリケーションは、*ブートストラップ* ファイルと呼ばれる最低限のファイルのセットがないとインストールできません。このほか、実行可能ファイル (.exe)、データ ファイル、ActiveX コントロール、ダイナミック リンク ライブラリ (.dll) ファイルなどのアプリケーション固有のファイルも必要です。

必要なアプリケーション ファイルは、次の 3 つのカテゴリに大きく分けることができます。

- **ランタイム ファイル**。アプリケーションがインストール後に正しく機能するために必要なファイルです。これらのファイルは、すべての Visual Basic 6.0 アプリケーションに必要です。Visual Basic プロジェクトのランタイム ファイルには、Msvbvm60.dll、Stdole2.tlb、Oleaut32.dll、Olepro32.dll、Comcat.dll、Asycfilt.dll、および Ctl3d32.dll があります。
- **セットアップ ファイル**。エンド ユーザーのコンピュータで標準のアプリケーションをセットアップするために必要なファイルです。これには、セットアップ 実行可能ファイル (Setup.exe と Setup1.exe)、セットアップ ファイル リスト (Setup.lst)、およびアンインストール プログラム (St6unst.exe) が含まれます。セットアップの詳細については、この後の「アプリケーションのセットアップのアップグレード」を参照してください。
- **アプリケーション固有のファイル**。アプリケーションを実行するために必要な、アプリケーションに固有のファイルです。アプリケーションの実行可能ファイル、データ ファイル、アプリケーションで使用される ActiveX コントロールなどがこれに含まれます。そのほか、アプリケーションに含まれている ActiveX コントロールが使用するライブラリやその他のファイルなど、プロジェクトが依存しているその他のファイルもこれに含まれます。

Visual Basic .NET では、アプリケーションに依存関係もあります。ランタイム ファイルは、.NET Framework ランタイムと Microsoft Data Access Components (MDAC) にカプセル化されています。アプリケーションをインストール先のコンピュータで実行するには、.NET Framework と MDAC がインストールされていることを事前に確認する必要があります。

セットアップ ファイルとアプリケーション固有のファイルは Visual Basic .NET でも必要ですが、Visual Basic .NET ではデプロイメント プロジェクトの作成方法が異なるため、セットアップ ファイル リストは変更されています。次の「アプリケーションのセットアップのアップグレード」では、これらの要件について説明します。

アプリケーションのセットアップのアップグレード

プロジェクトの自動アップグレードと手動による変更が完了したら、新しいアプリケーションの配布方法について検討する必要があります。古いインストーラはもう機能しませんし、Package and Deployment ウィザードで新しいアプリケーションのインストーラを作成することもできません。しかし、Visual Basic .NET には、セットアップ/デプロイメント プロジェクトという新しいプロジェクトの種類が用意されています。これを使用して、アプリケーションを配布するための Windows インストーラや CAB ファイルを作成できます。

セットアップ/デプロイメント プロジェクト オプションを使用してインストーラを作成するプロセスは、Visual Basic 6.0 セットアップ プロジェクトを使用してインストーラを作成し、作成されたインストーラを一切カスタマイズしていない場合に最も簡単になります。配置のカスタマイズを必要としない単純なプロジェクトでは、デプロイメント プロジェクトで実行可能ファイルと製品名を指定するだけで済みます。しかし、元のプロジェクトのインストーラでダイアログ ボックスの追加や標準のダイアログ テキストの変更などのカスタマイズが必要だった場合は、セットアップ/デプロイメント プロジェクトを使用する際にも同様のカスタマイズを行う必要があります。

以降では、Visual Studio .NET のセットアップ/デプロイメント プロジェクトを使用してアップグレード後のアプリケーションのインストーラを作成する方法と、必要に応じてインストーラをカスタマイズする方法を説明します。

新しいインストーラの作成

プロジェクトのインストーラを作成するには、デプロイメント プロジェクトを含めるソリューションでいくつかのステップを実行する必要があります。まず最初のステップでは、アプリケーションのソリューションに新しいセットアップ/デプロイメント プロジェクトを追加します。

▶ アプリケーションにセットアップ/デプロイメント プロジェクトを追加するには

1. ソリューション エクスプローラでツリーの [ソリューション] ノードを右クリックし、メニューの **[新しい項目の追加]** をクリックします。
2. **[新しいプロジェクトの追加]** ダイアログ ボックスで、セットアップ プロジェクトの名前を入力します。アプリケーションの他の部分と別にしておきたい場合は、インストーラ プロジェクトの場所を指定することもできます。図 16.6 は、一般的な **[新しいプロジェクトの追加]** ダイアログ ボックスを示しています。

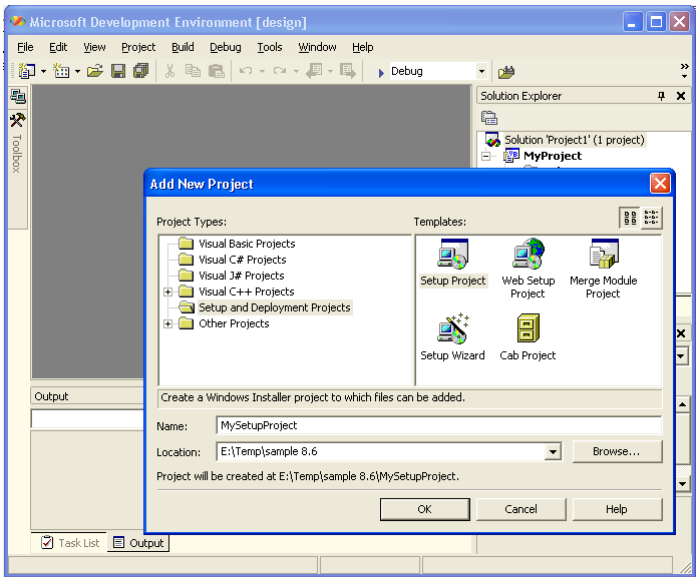

図 16.6

[新しいプロジェクトの追加] ダイアログ ボックス

3. プロジェクトの情報を入力し、[OK] をクリックします。プロジェクトがソリューションに追加されます。

セットアップ プロジェクトをソリューションに追加したら、ソリューション内のプロジェクトをインストーラに関連付ける必要があります。そのためには、プロジェクトをプロジェクト出力としてアプリケーション フォルダに追加する必要があります。プロジェクト出力には、プロジェクトをビルドすると生成されるファイルが含まれます。通常は、実行可能 (.exe) ファイルやライブラリ (.dll) ファイルで構成されます。

▶ プロジェクトをセットアップに追加するには

1. ソリューション エクスプローラを使用してセットアップ プロジェクトに移動します。
2. **[プロジェクト]** メニューの **[追加]** をクリックし、**[プロジェクト出力]** をクリックします。**[プロジェクト出力グループの追加]** ダイアログ ボックスが表示されます (図 16.7 を参照)。ダイアログ ボックスの一番上のドロップダウン リストには、ソリューション内のプロジェクトが表示されます。その下には、インストーラに追加できるプロジェクト出力のカテゴリが一覧表示されます。

   通常のインストールでは、ドロップダウン リストでアプリケーション プロジェクトを選択し、グループ リストで **[プライマリ出力]** を選択します。このほかに、インストーラを準備するときに使用する構成 ([Debug .NET] や [Release .NET] など) を指定することもできます。

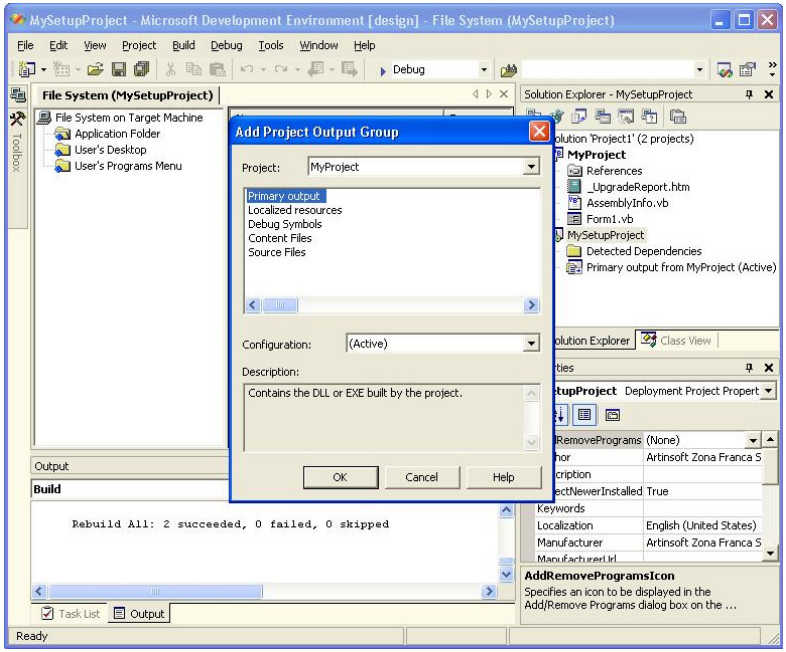

図 16.7

[プロジェクト出力グループの追加] ダイアログ ボックス

該当するプロジェクト出力をすべてインストーラ プロジェクトに追加すると、標準のアプリケーション インストーラをビルドできるようになります。

セットアップ プロジェクトの [ファイル システム] ビューには 3 つのフォルダがあります。[アプリケーション フォルダ]、[ユーザーのデスクトップ]、および [ユーザーのプログラム メニュー] の 3 つです。[アプリケーション フォルダ] は、アプリケーションの実行可能ファイルをインストールするインストール先コンピュータのパスを指定します。既定では、このパスの値は Program Files\Manufacturer\ProductName になります。Manufacturer は、Visual Studio .NET をインストールしたときに指定した会社名、ProductName は、セットアップ プロジェクトに使用した名前です。

これらの値を変更するには、[アプリケーション フォルダ] の DefaultLocation プロパティを使用するか、セットアップ プロジェクトの Manufacturer プロパティと ProductName プロパティを使用します。ユーザーへの影響をできるだけ小さくするために、元のインストーラで使用されていたのと同じ値を使用するようにしてください。

[ユーザーのデスクトップ] フォルダは、インストール時にユーザーのデスクトップに配置するファイルやショートカットを指定します。[ユーザーのプログラム メニュー] フォルダは、アプリケーションのインストール時にユーザーのプログラム メニューに追加する項目を指定します。

これで、セットアップ プロジェクトをビルドできます。セットアップ プロジェクトをビルドすると、アプリケーションの配布に必要なすべてのファイルが作成されます。これには、Setup.exe、Setup.ini、および ProductName.msi の各ファイルが含まれます。Setup.exe は、ProductName.msi と Windows インストーラ ブートストラップ アプリケーションのラッパーです。Windows インストーラ ブートストラップ アプリケーションは、

インストール先のコンピュータに正しいバージョンの Windows インストーラがなかった場合にそれをインストールします。セットアップ プロジェクトによって生成されるファイルはすべて必要なファイルであり、1 つのパッケージとして配布する必要があります。

インストーラのカスタマイズ

ほとんどの場合、既定のオプションを含む標準のインストーラでは、アプリケーションの配置の要件を十分に満たすことはできません。必要に応じて、ダイアログ ボックス、テキスト、ヘルプ ファイル、およびその他の要素をセットアップ プログラムに追加する必要があります。Visual Basic 6.0 でこれを行うには、セットアップ スクリプト ファイルか、ウィザードで使用される Setup1.exe プロジェクト テンプレートを修正しなければなりませんでした。

Visual Basic .NET では、プロジェクトのプロパティと、Visual Studio .NET に用意されているエディタを使用して、これらの変更を行うことができます。以下のエディタを使用できます。

- **ファイル システム エディタ**。インストール ディレクトリや、[スタート] メニューとデスクトップの項目を指定できます。
- **レジストリ エディタ**。インストール時にインストール先のコンピュータに追加するレジストリ キーと値を指定できます。
- **ファイルの種類エディタ**。インストール先コンピュータにおけるアプリケーションのファイルの種類の関連付けを指定できます。それらのファイルの種類に対する可能なアクションを識別するための動詞を指定することもできます。
- **ユーザー インターフェイス エディタ**。インストール プロセスにダイアログ ボックスを追加することができます。
- **カスタム動作エディタ**。インストール時に実行する追加の動作を指定できます。
- **起動条件エディタ**。インストールを開始する前にインストール先コンピュータで満たされている必要がある条件を指定できます。

個々のエディタの詳細については、MSDN の「Editors Used in Deployment」を参照してください。

ファイル システム エディタを使用すると、インストール先コンピュータに配置されるファイルへのすべての参照の管理、ファイルをコピーするための条件の設定、インストールされたファイルへのショートカットの作成、指定したディレクトリへのファイルの追加などを行うことができます。これらのファイルに対して指定したパスは、インストール時にインストール先コンピュータで使用されます。

レジストリ エディタを使用すると、インストール時にインストール先コンピュータで特定のレジストリ キーを操作できます。また、アップグレード後のアプリケーションのインストールを、前のバージョンのインストール時の動作に合わせてカスタマイズすることもできます。たとえば、前のバージョンのインストーラで、インストール プロセスの間に製品のバージョンを HKLM\Software\Manufacturer\Version のキーとして追加していた場合は、レジストリ エディタを使用すると、ツリーを展開していくだけでその製造元を見つけることができます。製造元が見つかったら、新しいキーを追加して、名前を Version に設定します。その後、このキーに新しい文字列値を追加して、製品のバージョンの値を設定します。図 16.8 はこの方法を示しています。

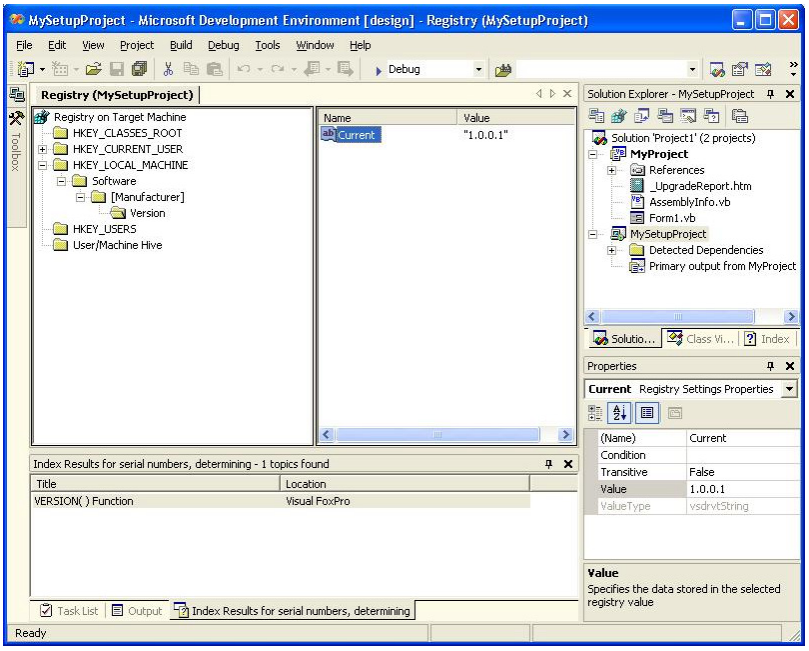

図 16.8

デプロイメント プロジェクトでレジストリ エディタを使用してバージョン番号のキーを作成

元のインストーラにダイアログを追加していた場合は、ユーザー インターフェイス エディタを使用して、新しいインストーラに同様のダイアログを追加する必要があります。ユーザー インターフェイス エディタには、インストール時に表示して追加の情報を提示または収集できる定義済みのユーザー インターフェイス ダイアログ ボックスがいくつか用意されています。

追加できるダイアログ ボックスを以下に示します。

- **[チェックボックス]**。インストール時に最大で 4 つの選択肢をユーザーに提示して、選択結果の値を返します。詳細については、MSDN の「Checkboxes User Interface Dialog Box」を参照してください。
- **[インストールの確認]**。インストールを開始する前に、インストールをキャンセルしたり、前のダイアログ ボックスに戻って変更を加えたりするための機会をユーザーに与えます。詳細については、MSDN の「Confirm Installation User Interface Dialog Box」を参照してください。
- **[ユーザー情報]**。名前、会社名または組織名、製品のシリアル番号などの情報の入力をユーザーに求めます。詳細については、MSDN の「Customer Information User Interface Dialog Box」を参照してください。
- **[完了]**。ユーザーにインストールの完了を通知します。詳細については、MSDN の「Finished User Interface Dialog Box」を参照してください。
- **[インストール アドレス]**。アプリケーション ファイルをインストールする Web の場所をユーザーが選択できるようにします。詳細については、MSDN の「Installation Address User Interface Dialog Box」を参照してください。

- [**インストール フォルダ**]。アプリケーション ファイルをインストールするフォルダをユーザーが選択できるようにします。詳細については、MSDN の「Installation Folder User Interface Dialog Box」を参照してください。
- [**使用許諾契約書**]。ユーザーに使用許諾契約書を提示して、確認と同意を求めます。詳細については、MSDN の「License Agreement User Interface Dialog Box」を参照してください。
- [**進行状況**]。インストールの進行状況をユーザーに通知します。詳細については、MSDN の「Progress User Interface Dialog Box」を参照してください。
- [**オプション ボタン**]。インストール時に最大で 4 つの相互排他的な選択肢をユーザーに提示して、ユーザーが選択した値を返します。詳細については、MSDN の「RadioButtons User Interface Dialog Box」を参照してください。
- [**注意事項**]。正誤表や最新情報など、ユーザーがアプリケーションを実行する前に知っておく必要がある追加情報を提示します。詳細については、MSDN の「Read Me User Interface Dialog Box」を参照してください。
- [**ユーザーの登録**]。ユーザーが登録情報を送信できます。これは、用意した実行可能ファイルを使用して行われます。詳細については、MSDN の「Register User User Interface Dialog Box」を参照してください。
- [**スプラッシュ**]。イメージを表示します。通常は、ロゴや商標情報を表示するために使用されます。詳細については、MSDN の「Splash User Interface Dialog Box」を参照してください。
- [**テキストボックス**]。インストール時に最大で 4 つのテキスト入力フィールドを表示して、ユーザーが入力した内容を返します。詳細については、MSDN の「Textboxes User Interface Dialog Box」を参照してください。
- [**ようこそ**]。序文テキストと著作権情報を表示します。詳細については、MSDN の「Welcome User Interface Dialog Box」を参照してください。

これらのダイアログ ボックスの詳細については、MSDN の「Deployment Dialog Boxes」を参照してください。

たとえば、以前のインストーラのインストール プロセスに [注意事項] ダイアログ ボックスが含まれていた場合に、Visual Basic .NET でもこれを含めるには、以下の手順を実行します。

1. 注意事項の内容をファイルに保存します。
2. ユーザー インターフェイス エディタを使用して、[注意事項] ユーザー インターフェイス ダイアログ ボックスをプロジェクトに追加します。
3. ダイアログ ボックスの ReadmeFile プロパティに、先ほど保存したファイルのパスを設定します。

すべてのエディタは、[ソリューション エクスプローラ] ペインの一番上にあるパネルを使用して開くことができます (図 16.9 を参照)。

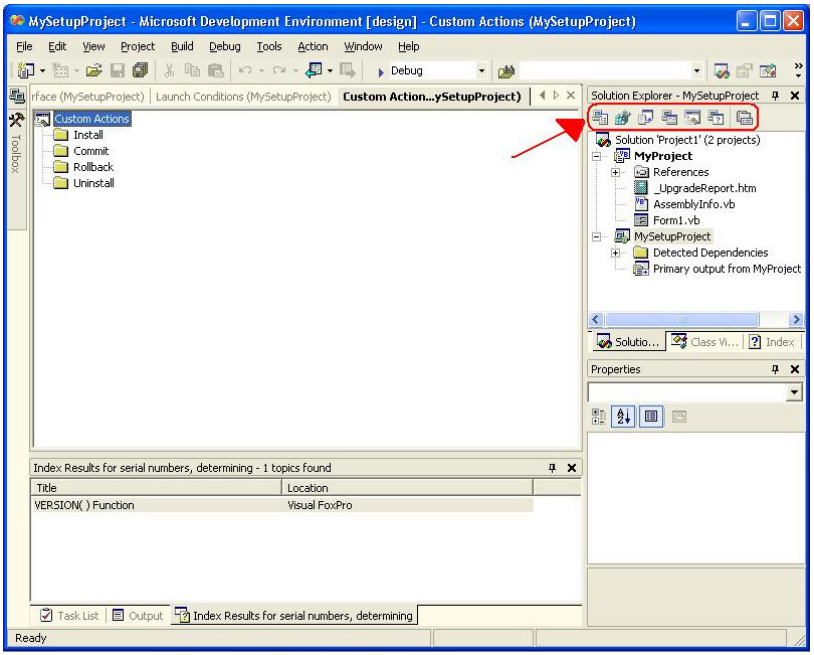

図 16.9

[ソリューション エクスプローラ] ペインにおける、配置のダイアログ ボックス エディタ

セットアップ プロジェクトのプロパティを使用すると、インストーラをさらにカスタマイズすることができます。作成者の名前、製品名、製造元の名前、製品の説明、バージョン番号などの情報を、プロパティを使用して設定できます。図 16.10 は、セットアップ プロジェクトのプロパティ ページの例です。

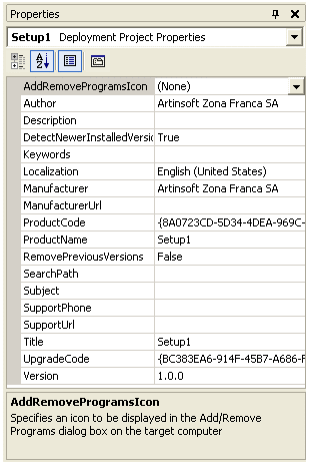

図 16.10

セットアップ プロジェクトのプロパティ ページの例

マージ モジュール

インストーラの作成におけるもう 1 つの重要なステップが、マージ モジュールの作成です。マージ モジュールを使用すると、COM プロジェクトなど、複数のアプリケーションによって共有されるコンポーネントをインストールすることができます。マージ モジュール (.msm) は、すべてのファイル、リソース、およびレジストリ エントリと、共有ファイルをインストールするためのセットアップ ロジックを含む単一のパッケージです。

マージ モジュールは、Visual Studio .NET で利用できるセットアップ/デプロイメント プロジェクトのオプションで、ソリューションに追加することができます。.dll ファイル、コントロール、リソース、コンポーネントなど、開発者以外は使用できないようにするものはすべて、マージ モジュールにパッケージ化する必要があります。このモジュールは、後から Windows インストーラに追加してエンド ユーザーに配布できます。

1 つのマージ モジュールに複数のコンポーネントを追加することもできますが、原則としては、コンポーネントごとに別のマージ モジュールを作成して不要なファイルを配布しないようにすることをお勧めします。なお、マージ モジュール ファイルで配布されていないサードパーティの共有コードをインストーラに追加する必要がある場合は、SharedLegacyFile プロパティをそれに合わせて設定する必要があります。

たとえば、Visual Basic 6.0 で作成された COM コンポーネントをアプリケーションで使用している場合 (ハイブリッド アプリケーションの場合) は、マージ モジュールが重要となる典型的なケースです。というのも、この種類のアプリケーションの配置には、Visual Basic 6.0 ランタイム マージ モジュールを含める必要があるからです。このマージ モジュールは、インストール先のコンピュータに Visual Basic 6.0 ランタイムのファイルがない場合にそれらを自動的にインストールするために使用します。この種類のアプリケーションのセットアップ プロジェクトを作成すると、参照されている COM コンポーネントが Visual Studio .NET によって自動的にプロジェクト出力として追加されます。インストール時には、Windows インストーラが COM コンポーネントを登録します。しかし、Visual Basic 6.0 ランタイム マージ モジュールは自動的には追加されないため、手動で追加する必要があります。以下の手順に示すような簡単なプロセスで追加できます。

▶ Visual Basic 6.0 マージ ファイルを手動で追加するには

1. **ソリューション エクスプローラ**で、**セットアップ** プロジェクトを選択します。
2. **[プロジェクト]** メニューの **[追加]** をポイントし、**[マージ モジュール]** をクリックします。
3. **[モジュールの追加]** ダイアログ ボックスで、Visual Basic 6.0 マージ モジュール (Msvbvm6.msm) を見つけます。既定では、\Program Files\Common\Merge Modules ディレクトリにインストールされています。このファイルがシステム上にない場合は、MSDN の Microsoft Visual Studio Developer Center の「Windows Installer Merge Modules」からダウンロードできます。ファイルを選択し、**[OK]** をクリックします。

アンマネージ コンポーネント (COM .dll など) の依存関係が Visual Studio .NET によって自動的に検出されないこともあります。そのような場合は、インストーラを作成する前に、プロジェクトの依存関係の種類を特定す

る必要があります。その依存関係の種類に基づいて、デプロイメント プロジェクトを作成するときに正しい解決方法を選択できます。

たとえば、別のプロジェクトの一部としてしかインストールできないコンポーネント (Internet Explorer の一部としてインストールされる Web ブラウザ コントロール (Shdocvw.dll) など) を参照している場合は、依存関係が自動的に検出されません。この場合は、このコンポーネントをデプロイメント プロジェクトから除外する必要があります。その後、起動条件を追加して、インストール先のコンピュータでこのコンポーネントの有無をチェックし、コンポーネントが見つからない場合はインストールを中止するようにします。この場合、エンド ユーザーは、アプリケーションをインストールする前に、このコンポーネントを提供する製品をインストールする必要があります。

このほか、公開されない依存関係があるアンマネージ コンポーネントを追加する場合も、依存関係が自動的に検出されません。たとえば、Microsoft Foundation Classes (MFC) では、ローカライズされたサテライト ファイルは依存関係として含まれません。この場合は、考えられる依存関係をすべて特定して、デプロイメント プロジェクトに含める必要があります。コンポーネントのドキュメントをチェックしたり、コンポーネントの作成者に問い合わせるなどして、依存関係の完全なリストを入手する必要があります。

マージ モジュールの詳細については、MSDN の以下の記事を参照してください。

- 「Installer Package Files and Merge Modules」
- 「Merge Module Properties」
- 「Walkthrough: Creating and Consuming a Merge Module」

Web 配置

ASP .NET アプリケーションを配置するには、Microsoft Visual Studio .NET の Web セットアップ プロジェクトを使用します。このプロジェクトを使用すると、従来のメディアではなく Web を通じてアプリケーションを配布するための Windows インストーラを作成できます。配置を使用して Web サーバーにファイルをインストールすると、登録や構成の問題が自動的に処理されるため、単純にファイルをサーバーにコピーするより有効です。

この種類のセットアップ プロジェクトでは、この章の「インストーラのカスタマイズ」で説明したのと同じエディタを使用します。ただし、インストール Web サイトについて必要な情報を追加するための小さな違いがいくつかあります。違いが最も大きいのはファイル システム エディタで、ページを格納する Web フォルダの構造と、配置先 Web サイトの属性を変更するために使用できるプロパティしか表示されません。

ファイル システム エディタで使用できる主なプロパティを以下に示します。

- Virtual Directory。配置先 Web サイトの名前を保持します。
- Port。Web サイトが応答するポートの番号を指定します。
- AllowDirectoryBrowsing。配置先の Web ディレクトリが参照可能かどうかを指定します。

- ExecutePermissions。配置先 Web サイトの実行モードを指定します。
- IsApplication。配置先 Web サイトにアプリケーションを作成する必要があるかどうかを指定します。

また、起動条件エディタでは、配置先のコンピュータでインターネット インフォメーション サービスが利用可能かどうかをチェックする**起動条件**が Visual Studio .NET によって自動的に追加されます。

COM+ の配置

コンポーネント サービス管理ツールでは、ステージング コンピュータやプロダクション コンピュータへのサーバー アプリケーションのインストール、クライアント コンピュータへのアプリケーション プロキシのインストール、アプリケーションの削除や更新など、システム管理者によって行われる COM+ の配置作業の多くを自動化できます。

コンポーネント サービス管理ツールを使用して行う必要がある COM+ アプリケーション インストール作業には、次のようなものがあります。

- **新しい COM+ アプリケーションの作成**。この手順を実行するには、インストール先コンピュータのシステム アプリケーションの Administrator ロールのメンバである必要があります。
- **COM+ サーバー アプリケーションのエクスポート**。コンポーネント サービス管理ツールを使用すると、他の 1 つ以上のサーバー コンピュータにインストールするために COM+ サーバー アプリケーションをエクスポートすることができます。たとえば、アプリケーションの開発やテストを行ったステージング コンピュータから個々のサイトのプロダクション コンピュータに、コンポーネント サービス管理ツールを使用して COM+ アプリケーションをエクスポートできます。
- **COM+ サーバー アプリケーションのインストール**。COM+ アプリケーションをインストールするには、COM+ アプリケーション インストール ウィザードを使用するか、コンピュータの COM+ アプリケーション フォルダを開いた状態で、Windows エクスプローラからコンポーネント サービス管理ツールの詳細ペインにアプリケーション ファイルをドラッグします。
- **COM+ アプリケーション プロキシのエクスポート**。COM+ アプリケーション エクスポート ウィザードは、COM+ アプリケーション プロキシをエクスポートします。クライアント コンピュータにプロキシをインストールすると、クライアント アプリケーションでプロキシの情報を使用して、プロダクション コンピュータで実行されている COM+ サーバー アプリケーションを見つけて、DCOM を使用してアクセスできます。
- **COM+ アプリケーション プロキシのインストール**。アプリケーション プロキシは Windows インストーラ (.msi) ファイルとして作成されるため、簡単にクライアント コンピュータにインストールできます。アプリケーション プロキシをインストールすると、クライアント アプリケーションがクライアント コンピュータから COM+ アプリケーションにリモート アクセスするために必要な登録情報が提供されます。
- **COM+ アプリケーションの削除**。既存のアプリケーションが古くなったり使用されなくなったりして、削除する必要が出てくることもあります。アプリケーションを削除すると、アプリケーションに含まれているコンポーネントも削除されます。これらのコンポーネントがその他のリソース (データベース接続、デー

タ ファイルやテキスト ファイル、IIS 仮想ルート構成など) に依存している場合、それらのリソースは、コンポーネント サービス管理ツール以外のツールやユーティリティを使用して削除する必要があります。

- COM+ **アプリケーションの複製**。すべての COM+ アプリケーションとその構成をコンピュータ (マスタ) から別のコンピュータ (複製先) に複製する必要がある場合があります。複製は、次のような状況で便利です。
  - Web ファーム
  - フェールオーバー クラスタ
  - プロダクション コンピュータへのステージング

マスタ コンピュータ上の各アプリケーションをエクスポートしてインストールする作業を繰り返すことによって、手動でアプリケーションを複製することもできますが、この方法は面倒でエラーも起こりやすいため、COM+ アプリケーションの複製ユーティリティ (comrepl.exe) を使用した方が簡単でしょう。このユーティリティは、Microsoft Windows 2000 以降のオペレーティング システムに含まれています。この複製ユーティリティは、マスタ コンピュータから複製先のコンピュータにすべての COM+ アプリケーションをコピーし、コンピュータ全体の COM+ 構成のすべてを複製します。

アプリケーション プロキシ

コンポーネント サービス管理ツールを使用すると、COM+ サーバー アプリケーションを簡単にアプリケーション プロキシとしてエクスポートすることができます。COM+ でアプリケーション プロキシを生成するには、サーバー アプリケーションのすべてのコンポーネントがインポートではなくインストールされていることが重要です。これにより、必要な登録情報がすべてアプリケーションに含まれていることが保証されます。

COM+ によって生成されるアプリケーション プロキシは、Windows インストーラのインストール パッケージです。インストールが完了すると、クライアント コンピュータのコントロール パネルの [プログラムの追加と削除] にアプリケーション プロキシが (Windows インストーラ オーサリング ツールを使用して .msi ファイルが変更されていない限り) 表示されます。

COM+ インストール パッケージ

コンポーネント サービス管理ツールまたは COM+ 管理ライブラリを使用して、COM+ アプリケーションとアプリケーション プロキシの Windows インストーラ インストール パッケージを作成できます。その後、これらのパッケージを使用して、COM+ アプリケーションをサーバー コンピュータとクライアント コンピュータに配置できます。

コンポーネント サービス管理ツールは、エクスポートするアプリケーションを構成するクラス、その属性、およびグローバル レベルの属性を特定し、その情報を使用して、以下の内容を持つ単一の .msi ファイルを生成します。

- COM 登録情報を含む Windows インストーラ テーブル
- アプリケーションの属性を含む .apl ファイル
- COM+ アプリケーションのクラスによって実装されるインターフェイスを記述する .dll ファイルとタイプ ライブラリ

配置オプション

配置のためのコンポーネントのパッケージ化は、コンポーネント ベースのアプリケーションの作成における重要な側面の 1 つであり、慎重に検討する必要があります。COM+ アプリケーションの配置では、個々の分散コンポーネントが配置および実行されるアプリケーション層を考慮に入れる必要があります。COM+ コンポーネントの種類や構成に関する決定は、パフォーマンス、信頼性、耐障害性など、アプリケーションのさまざまな側面に影響します。COM+ コンポーネントの種類 (サーバー コンポーネントかライブラリ コンポーネントか) は、配置オプションの選択の決定的な要素になります。

サーバー コンポーネントは専用のプロセスに分離されます。サーバー コンポーネントが停止しても、他のサーバー アプリケーションには影響しません。一方、ライブラリ コンポーネントは、直接の呼び出し元のプロセスに読み込まれ、互いに依存する状態になります。

配置オプションを評価する際にまず考慮する必要があるのは、呼び出しのパフォーマンスです。異なるプロセス間の呼び出しは、同じプロセス内の呼び出しに比べてコストが高くなります。考慮する必要があるもう 1 つの側面は、コンポーネントの呼び出しの際に行われるセキュリティ検証です。セキュリティ資格情報が検証されるのは、プロセス間呼び出しの場合だけです。ライブラリ コンポーネントは、ホスト プロセスのセキュリティ レベルを使用します。

一般に、COM+ ライブラリ アプリケーションには、サーバー アプリケーションに含まれるメイン COM+ コンポーネントが使用するユーティリティを含めるようにします。こうすることで、プロセス間呼び出しの量を制御できます。

メイン アプリケーション コンポーネントは、必要に応じてライブラリ コンポーネントを呼び出す COM+ サーバー コンポーネントにします。アプリケーション コンポーネントの量は、保守性、セキュリティ、および信頼性の問題を考慮して調整する必要があります。

配置の問題とアプリケーションのアーキテクチャの決定は、互いに影響する関係にあります。アプリケーションのライフ サイクル全体を通じて、アーキテクチャと配置のオプションを必要に応じて改良および変更する必要があります。配置スキーマを選択する際には、このことを考慮に入れる必要もあります。

まとめ

完全にアップグレードされたアプリケーションをユーザーに提供するためには、コードのアップグレードだけでなく、ヘルプや配置機能のアップグレードも必要になります。またその際には、アプリケーションの破損を招くコンポーネントのバージョン間の競合ができるだけ起こらないように、配置機能を管理する必要もあります。

この章では、アプリケーションのヘルプ システムを Visual Basic .NET にアップグレードする際に必要な情報や、新しくアップグレードしたアプリケーションを配置する際に利用できるさまざまな選択肢を紹介しました。アプリケーションをアップグレードする際にこれらの手順やテクニックを使用することによって、アップグレード プロセスを完了することができます。

詳細情報

ダイアログ ボックスの詳細については、MSDN の「Deployment Dialog Boxes」を参照してください。
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsintro7/html/vbconDeploymentDialogs.asp

HelpProvider クラスの詳細については、MSDN の『.NET Framework Class Library』の「HelpProvider Class」を参照してください。
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwindowsformshelpproviderclasstopic.asp

エディタの詳細については、MSDN の「Editors Used in Deployment」を参照してください。
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsintro7/html/vbcontheunifieddeploymenteditor.asp

Visual Basic 6.0 マージ モジュールのダウンロードについては、MSDN の Microsoft Visual Studio Developer Center の「Windows Installer Merge Modules」を参照してください。
http://msdn.microsoft.com/vstudio/downloads/updates/sp/vs6/sp5/mmoverview.asp

マージ モジュールの詳細については、MSDN の以下の記事を参照してください。

- 「Installer Package Files and Merge Modules」
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsinstal/html/veconinstallerpackagefilesmergepackagefiles.asp
- 「Merge Module Properties」
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsinstal/html/veovrmergepackageproperties.asp
- 「Walkthrough: Creating and Consuming a Merge Module」
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsintro7/html/vxwlkwalkthroughcreatingconsumingmergemodule.asp

17

アプリケーションの改良の概要

アプリケーションのアップグレードは、同等の機能を実現したところで終える必要はありません。実際、Microsoft Visual Basic .NET にアップグレードすることの大きなメリットの 1 つは、Visual Basic の以前のバージョンでは実装することが困難な新しい機能を追加できることです。アプリケーションを Visual Basic .NET に移行すると、新しい可能性が開けます。

アプリケーションの改良は、アプリケーションへの新しい機能の追加や、既存の機能の向上のプロセスです。アプリケーションを改良するために、まず新しい言語 (または少なくとも新しいバージョンの言語) にアップグレードする必要があるとは限りませんが、それが役に立つことはよくあります。言語またはフレームワークの古いバージョンは、新たに登場したテクノロジを取り入れるには制約が多すぎることがあります。

Visual Basic .NET アプリケーションを改良するための可能性を列挙し、それぞれの実現方法について詳しく説明すると、それだけで 1 冊の本になります。そこで、第 17 章 (この章)、第 18 章、第 19 章、および第 20 章では、アプリケーションに加えることができる改良の種類について説明し、特定の可能性を追求する場合の詳細情報の参照先を示します。これらの章では、実際に技術的な内容をすべて提供するのではなく、一般的な改良のシナリオについて説明します。ただし、この情報から、改良することができるアプリケーションの側面のロードマップを得ることができます。

開発者は、どの部分を、どの程度改良するかを決定する必要があります。ポイントは、Visual Studio .NET または Visual Studio 2005 の機能を羅列するのではなく、有益かつ現実的なロードマップを作成することです。ある種の改良については、前提となる要件や注意点を挙げておくことをお勧めします。この章と以降の章により、意思決定を下したり、作業の方向性を決定したりするために必要な情報が得られます。またこれらの章では、決定に従って作業するために必要な詳細事項の参照先も示します。

対象読者

アプリケーションを改良するには、製品チームのすべてのメンバが関与する必要があります。

技術的な意思決定者は、特定の改良による恩恵が、その改良を実現するための作業コストに見合うかどうかを判断する必要があります。第 17 ～ 20 章では、アプリケーションに加えられる改良について説明するだけでなく、改良の利点についても説明します。この情報を使用することで、技術的な意思決定者は、改良が企業にもたらす恩恵と、それを実現するためのコストを比較し、適切な判断を下すことができます。このガイドのこの部分で提供する情報は、改良を実施するためのビジネス ケースの作成に使用することもできます。

コンポーネント、アプリケーション、およびシステムのデザインを担当するソリューション アーキテクトは、デザインの改善に欠かせないソフトウェアの構築およびリファクタリングについての説明を得ることができます。これらのトピックはきわめて広範であり、第 17 ～ 20 章までで完全に説明することはできないため、追加リソースの参照先を示します。

ソフトウェア開発者は、Microsoft .NET Framework で使用可能な多数の機能やテクノロジの一部と、アプリケーションのパフォーマンスと拡張性を向上させるためにそれらを適用する方法についての入門的な説明を得ることができます。また、個々の機能やテクノロジについての詳細情報を得るための追加リソースの参照先も示します。

アーキテクチャ、デザイン、および実装の改良

小規模な改良計画であっても、新しいアーキテクチャ、デザイン、および実装は必要です。アーキテクチャのトピックは、システム全体とシステムの基盤に影響を与えます。また、他のシステムとの関係にも影響を与えます。デザインを改良すると、コードが整理され、システムのどの部分を再利用でき、どの部分を新たに作成する必要があるのかを判断しやすくなります。実装を改良すると、コードのパフォーマンス、読みやすさ、および保守性がわずかに向上します。

ここでは、これらの改良の側面について、それぞれ詳しく説明します。

アーキテクチャの改良

アーキテクチャの概念は古く、ソフトウェア業界で頻繁に使用されています。アーキテクチャは、新しいテクノロジを使用したプログラミングで重要な役割を果たします。アーキテクチャの作成は、.NET Framework で拡張性の高いアプリケーションの開発を計画するときの主要な作業です。

ソフトウェア アーキテクチャとは何か、またソフトウェア アーキテクチャをどのように使用または実装するかについては、意見が分かれています。一部の人たちは、システムをパーツに分解するための最上位レベルのブレイクダウン、または開発者によるシステム デザインの共通の理解という観点で、アーキテクチャを捉えています。最近最も主流になっている説明では、アーキテクチャをパターンの集まりと捉えています。「アーキテクチャ」という用語の解釈に関係なく、ソフトウェア開発におけるアーキテクチャの役割はきわめて重要です。

.NET Framework は、アーキテクチャに重点を置くことの必要性への確かな一歩を提供します。このフレームワークの多くの部分は、ソフトウェア開発の特定のパターンに対処するようにデザインされています。これらのパターンとその応用を理解することは、フレームワークの適切な利用と、デザイン サイクルおよび開発サイクルの簡略化に役立ちます。

オブジェクト指向デザインは、クラスを中心に構成されています。アプリケーションのコンテキストから外れた単一のクラスは、意味を持たないことがあります。アーキテクチャにおけるデザイン パターンは、リレーションシップを表します。デザイン パターンをオブジェクトに適用すると、アプリケーション全体および個々のコンポーネントが意味が持つようになります。

アーキテクチャの作成は、複雑なソリューションやエンタープライズ アプリケーションを構築するための重要なプラクティスです。アプリケーションはビルと捉えることができます。構造的なデザインを行わずにビルを建築するのは非現実的であり、問題があります。またアーキテクチャは、複雑さを管理するためにも役立ちます。ビルが大きくなれば、アーキテクチャの重要性も増します。アーキテクチャのパターンは、アプリケーション全体に意味を与え、解決する必要がある問題を理解しやすくします。強固なアーキテクチャ パターンに基づいて構築された計画があれば、アプリケーションをより迅速に開発することができます。入念に構築されたアプリケーションは、アーキテクチャの作成が適切に行われていれば、メンテナンスがより簡単になります。また、機能の拡張や、変更の実装もさらに容易になります。既知のパターンに基づいてパーツを組み合わせるのは、プログラマの仕事です。これにより、アーキテクトとプログラマの役割は、2 つの異なる機能に分かれます。

.NET Framework は、アーキテクチャへの新たな注目を促します。アプリケーションは本質的に堅牢性が高まり、開発およびデザインのための強化されたアプローチを提供するフレームワーク上に構築されるようになりました。.NET Framework は、堅実な原則と方法に基づき、構造と目的の真の意味を維持しながらも、複雑な状況に容易に適合する開発を可能にします。

アーキテクチャが出現したのは、主に 1990 年代の初めのことです。当時はクライアント サーバー アプリケーションが台頭していました。これらのアプリケーションでは、デザイン、実装、および配置の段階で、今日のアプリケーションよりもはるかに多くの管理作業が必要でした。

21 世紀になり、.NET などの新しいテクノロジが登場しました。これらのテクノロジは、Web 開発環境を追加した、アプリケーション開発用の多機能なフレームワークを提供しました。したがって、これらのテクノロジが揃った今、クライアント/サーバー アーキテクチャに戻り、.NET Framework アプリケーションの将来の開発のための適切な方向に進むことが必要です。

オブジェクト指向の機能の利用

Visual Basic .NET は、C++ や C# などのその他の言語で使用可能なオブジェクト指向の機能のほとんどをサポートする、完全なオブジェクト指向言語です。これらの機能には、継承、ポリモーフィズム、メンバのオーバーロードなどがあります。ここでは、Visual Basic .NET 言語のこれらの機能の概要と、その使用法についての提案事項を示します。オブジェクト指向のデザイン テクニックと開発テクニックについての完全な説明は、このガ

イドの範囲を超えています。詳細については、MSDN の「Object-Oriented Programming in Visual Basic」を参照してください。

オブジェクト指向のデザインとプログラミングは、私たちの身の回りの世界を理解してモデル化し、そのモデルを実装するための優れた方法として提唱されています。したがってオブジェクト指向のデザインとプログラミングは、より自然な表現を提供し、コードの品質と保守性を向上させます。こうした機能には、カプセル化、継承、ポリモーフィズムなどがあります。

カプセル化

この機能は、クラス メンバのグループを 1 つの概念的なユニットとして扱えるようにします。カプセル化を使用すると、開発者は実装の詳細を隠しつつ、明確に定義された機能セットをクライアントに公開することができます。

継承

この機能は、以前に定義されたクラスまたはインターフェイスに基づいて新しいクラスを作成する機能です。新しいクラス (*派生*クラスと呼ばれる) は、継承されるクラス (*基本*クラスと呼ばれる) の機能を拡張し、より具体的な問題を解決します。派生クラスは、基本クラスの既存のメンバ (変数、メソッド、およびプロパティを含む) をすべて備えます。継承したメソッドの動作が派生クラスにとって不適切または不正な場合、派生クラスでは継承したメソッドの定義を変更し、それらをより適切なメソッドにすることができます。このプロセスを*オーバーライド*と呼びます。

継承の古典的な例として、Shape 基本クラスと、特定の形状を表す Rectangle や Circle などのサブクラスがあります。Shape クラスには、形状の面積を計算する Area メソッドがあります。各サブクラスでは、それぞれの形状に適するように Area の定義をオーバーライドすることができます。

以下のサンプル クラス定義を使用して、継承の例を示します。

```
' 基本クラス
Public Class Shape
    Public Overridable Function Area() As Double
        ' 汎用の形状面積のコードがここに入ります。
    End Function
End Class

Public Class Circle ' 派生クラス
    Inherits Shape  'Inheritance

    …
    Overrides Public Function Area() As Double
        ' 円の面積を計算するコードがここに入ります。
    End Function
End Class
```

上記のクラス定義では、Shape クラスと Circle クラスの間に継承関係が確立されています。Shape クラスの定義の Overridable キーワードは、Shape を継承するすべてのクラスで、このメソッドを変更できることを示します。この例では、キーワード Overrides が使用されていることでわかるように、Circle クラスで Area 関数を変更しています。既存の機能で十分であれば、基本クラスで提供されている機能を派生クラスでオーバーライドする必要はありません。

なお、継承プロパティは、他のクラスに推移します。クラス A がクラス B の基本クラスであり、クラス B がクラス C の基本クラスである場合、クラス C はクラス A も継承します。このような場合、クラス A はクラス C の*祖先*クラスと呼ばれます。クラス C のオブジェクトは、クラス A のメソッドと変数をすべて備え、Overridable と示されているクラス A の任意のメソッドをオーバーライドすることができます。

継承によってアプリケーションのデザインは改善され、開発者による既存のコード リソースの再利用や修正が可能になります。

インターフェイス

インターフェイスは、クラスで実装する必要があるプロパティ、メソッド、およびイベントを指定します。インターフェイスを使用すると、開発者はクライアントが把握する必要がある情報 (メソッドの名前、戻り値の型、パラメータ リストなど) を、実装の詳細から切り離すことができます。これにより、既存のコードを危険にさらすことなく、拡張した実装をインターフェイス用に開発できるため、互換性のリスクが軽減されます。

Visual Basic 6.0 では、既存のインターフェイスを実装することは可能ですが、インターフェイスを独自に定義することはできません。Visual Basic .NET では、Interface ステートメントが導入され、開発者がインターフェイスをクラスと異なるエンティティとして定義できるようになりました。クラスでインターフェイスを実装するには、Implements キーワードを使用します。重要なのは、.NET Framework のクラスでは、異なるコンテキストに適した複数のインターフェイスを実装できるということです。また、インターフェイスを実装することにより、インターフェイス (基本クラスとして機能) とそれを実装するクラス (インターフェイスから派生) の間に継承関係が確立されるということも重要です。

インターフェイスは、クラスの継承で起こりうる問題を解消します。つまり、実装後にデザインに変更を加えるときに、コードが破壊される可能性がなくなります。クラスを最初に公開するときは、デザイン上の判断を行う必要があります。デザイン上の仮定を変更する必要がある場合、アプリケーションのコードを変更するのは安全ではない可能性があります。

以下のコード例は、ファイルの暗号化に使用されるクラスに対応する Visual Basic .NET インターフェイスを示しています。

```
Interface ICipher
    Sub SetKey(ByVal keyBase As String)
    Sub Encrypt(ByVal inputFile As String, ByVal outputFile As String)
    Sub Decrypt(ByVal inputFile As String, ByVal outputFile As String)
End Interface
```

このクラスの実装例については、第 20 章「一般的なテクノロジ シナリオ」の「暗号化の使用」を参照してください。インターフェイスの詳細については、MSDN の「Interfaces in Visual Basic .NET」を参照してください。

ポリモーフィズム

この機能を継承と組み合わせて使用すると、メソッドの呼び出しに使用されたオブジェクトに応じて、プログラムで適切なバージョンのメソッドを呼び出すことができます。概念上、基本クラスは機能のコア セットを提供します。派生クラスは、このコア セット上に、機能の一部を特殊化することによって構築されます。基本クラスを参照する変数を使用すると、基本クラスまたは任意の派生クラスのオブジェクトを参照できます。この参照を通じてオーバーライドされたメソッドを呼び出すと、実際に呼び出されるのは、変数の型に基づいたメソッドではなく、オブジェクトの型に基づいたメソッドになります。以下の例は、これを最もよく表しています。

```
Public Function ShowArea(shape as Shape)
    Dim msg as String

    msg = "Area = " & shape.Area
    MsbBox(msg)
End Function

…
    Dim s as new Shape()
    Dim c as new Circle()
' 形状バージョンの面積の計算が呼び出されます
    ShowArea(s)
' 円バージョンの面積の計算が呼び出されます
    ShowArea(c)
…
```

ポリモーフィズムにより、きわめて汎用的なプログラミングが可能になります。開発者は、機能が派生クラスでどのように特殊化されるかを考えることなく、提供される機能に基づいて、問題に対するソリューションを記述できます。上記のコード例の場合、開発者が把握している必要があるのは、Area メソッドが存在するということだけです。このメソッドが異なる形状の場合にどのように計算されるかという詳細は、重要ではありません。ランタイムは、オブジェクトの型に基づいて、呼び出すメソッドのバージョンを管理します。

機能のオーバーロード

あるメンバが同じ名前で複数回宣言され、宣言ごとに異なる引数を持つ場合、そのメンバはオーバーロードされています。この性質を利用し、開発者は機能を汎用的にプログラミングできます。たとえば、Encrypt という関数を、異なるデータ型を受け取って、暗号化した値を返すようにデザインすることができます。こうすれば、暗号化した変数が必要なすべての箇所で、Encrypt 関数を使用することができます。Visual Basic 6.0 では、予想されるデータ型ごとに、名前の異なる関数を定義する必要があります。

オーバーロード、オーバーライド、およびシャドウは、メンバを同じ名前で宣言する場合に使用可能な、よく似た概念ですが、以下に示すように、それぞれが適する特定の状況には重要な違いがあります。

- オーバーロードは、異なるメンバが同じ名前を持ち、異なる数の引数または異なるデータ型の引数を受け取る場合に適用できます。
- オーバーライドは、派生クラスで定義されているメンバが、祖先のメンバと同じデータ型の引数を、同じ数だけ受け取る必要がある場合に適用できます。
- シャドウは、継承したメンバをローカルに置き換える必要がある場合に適用できます。

ビジュアル継承

継承が特に有効であることが証明されているもう 1 つのシナリオは、ビジュアル フォームの継承です。この種の継承を使用すると、既存のフォームのレイアウト、コントロール、およびコードを、新しいフォームに適用できます。このような継承を通じて、開発者は標準的な基本フォームを構築できます。その後、アプリケーションに対して類似したルック アンド フィールを持たせるために、これらのフォームをアプリケーションで継承することもできますが、新しいコントロールを追加したり、特定のコントロールの動作を変更したりすることで、フォームを自由に拡張できます。ビジュアル継承を利用すれば、アプリケーションごとに新しいフォームを作成し直すのではなく、既存のフォームを使用してそれを新しいアプリケーション向けにカスタマイズできます。

実装のレイヤ化

レイヤ化は、開発者が自らのアプリケーションを優れた n 階層モデル アプリケーションとして表現するときに使用する共通の用語です。開発者はしばしば、レイヤ構造になっていることを理由に、アプリケーションがいかに適切に構築されているかを誇らしげに主張します。これは好ましい出発点ではありますが、適切にデザインされたソリューションであることを暗示するわけではありません。

ここでは、改良に向けてアプリケーションを準備するうえでレイヤ化がどう役立つのかも含めて、レイヤ化について詳しく説明します。

レイヤは、機能の積み重ねを表します。上位のレイヤは、下位のレイヤに依存して、必要な情報やメソッドを提供します。単一のレイヤは、アプリケーション層に含まれる論理的な機能グループを表します。アプリケーション層は、通常はデータベース サーバーなどの物理リソースに連結される機能に対応します。このように、アプリケーション層は、その層によって提供される機能を構築する 1 つ以上の論理的なレイヤによって構成されます。アプリケーション層とその分類の詳細については、第 4 章「一般的なアプリケーションの種類」の「アーキテクチャの注意点」を参照してください。

開発作業では、レイヤ化に向けてボトムアップやトップダウンなどのいくつかのアプローチを取ります。ただし、これらのアプローチは必ずしもすべてのアプリケーションに適用できるわけではありません。すべてのアプリケーションは特定の問題に対するソリューションを表します。あるアプリケーションには適用できるものは別のア

プリケーションには適用できません。重要なのは、レイヤを常に上位のレイヤの構成要素と捉えること、また下位のレイヤは上位のレイヤに依存して動作すべきではないということを覚えておくことです。

通常、レイヤのグループは、一貫した集合体を表します。たとえば、アプリケーションをプレゼンテーション層、ビジネス ロジック層、データ層など、レイヤのグループで構成されるさまざまな層に分割するという考え方をよく知っている人もいるはずです。各層は、アプリケーションの一貫したユニットを表し、ユーザーへの情報の表示やビジネス ロジックの表現などの単一の目的で使用されます。

レイヤの欠点は、インターフェイスの変更の連鎖により、作業が増える可能性があるということです。画面またはページにフィールドを追加すると、プレゼンテーション層のレイヤを変更するだけでなく、ビジネス層やデータ層でも変更が必要になることがあります。

レイヤの利点は、置換が可能だということです。つまり、あるレイヤを取り出して、インターフェイスに準拠しているがまったく異なるレベルの機能を提供する別のレイヤを提供することができます。これは単体テストに非常に役立ちます。また、異なるビジネス ロジックやデータベース アクセスを必要とするがその他の変更は必要としないアプリケーションにも便利です。レイヤは慎重にデザインすることが重要だということを覚えておいてください。不要なレイヤは、パフォーマンスに悪影響を与えることがあります。

ほとんどのビジネス アプリケーションで明確に定義されている 3 つの階層があります。これらの階層は以下のとおりです。

- **プレゼンテーション層**。この層は、アプリケーションのユーザー インターフェイス (UI) を提供します。
- **ドメイン (ビジネス) ロジック層**。この層は、アプリケーションの機能を実装します。
- **データ層**。この層は、外部データベースへの情報の格納と、外部データベースからの情報の取得を行います。

考慮すべき他の階層を持つ別のアーキテクチャも存在しますが、アプリケーションを大まかに捉えると、これら 3 つの異なる階層に分割されていることがわかります。

なお、n 階層アプリケーションという用語は、この分野で最も誤用されている用語の 1 つです。多くの人は、レイヤ化されたアプリケーションを n 階層アプリケーションと考えています。実際には、この用語は、ビジネス ロジック層が異なるコンピュータで実行できる複数のレイヤに分かれているアプリケーションにのみ使用できます。事実、"階層" という用語は、論理的な分離ではなく、物理的な分離を表します。一方、"レイヤ" という用語は、論理的な分離を表します。

デザイン パターン

デザイン パターンは、実際のアプリケーション開発で何度も直面するソフトウェア デザインの問題に対する、繰り返されるソリューションです。デザイン パターンは、オブジェクトのデザインと対話に関連しています。その目的は、頻繁に直面するプログラミングの課題に対し、洗練されて再利用可能なソリューションを中心にした通信プラットフォームを提供することです。

フレームワーク、ライブラリ、およびアプリケーション ブロックの使用

最も有益なパターンには、フレームワークを記述するものがあります。これらのパターンは、ソフトウェア アーキテクチャの幅広い再利用を促進するフレームワークを抽象的に記述したものと捉えることができます。同様に、フレームワークは、デザインおよびコードの直接的な再利用を促進するパターンを具体的に実現したものと捉えることができます。パターンとフレームワークの違いの 1 つは、パターンが言語に依存しない方法で記述されるのに対し、フレームワークは通常特定の言語で実装されるということです。ただし、パターンとフレームワークは高い相乗効果を持つ概念であり、どちらか一方がもう一方に従属するというものではありません。次世代のオブジェクト指向フレームワークは多数のパターンを明示的に具現化し、パターンは、フレームワークの形態や内容を文書化するために幅広く使用されます。

デザイン パターンの導入とマーケティングは、ソフトウェア開発に小さな革命を起こしました。開発者は、問題を抽象化し、多数の定義済みデザイン パターンのいずれかに問題が合致するケースを見極めることをお勧めします。共通のパターンという観点で考えることで、開発者は過去の開発作業で得た知識や実装経験をより簡単に適用することができます。デザイン パターンの観点で考えることによってソフトウェア開発に固有の問題がすべて解決するわけではありませんが、これがソフトウェアを開発するためのきわめて強力かつ効果的な方法であることは証明されています。

パターンの中には、きわめて頻繁に、まったく同じ方法で使用されることも多いために、単なるパターンを超えた存在になるものもあります。これらのパターンは共通コード ライブラリへと進化し、ほぼすべてのプロジェクトに含められます。これは今に始まったことではありません。共通コード ライブラリは、プログラミングと同じくらいの長い歴史を持っています。プログラマは皆、独自の共通コード ライブラリを所有しています。これは開発チームでも同じです。共通のタスクを実行するための優れた方法を発見したら、それを形式化し、他の人が使える場所に置いておくことは良いことです。これこそがマイクロソフトのアプリケーション ブロックであり、共通のタスクの実装を簡素化するコード ライブラリなのです。アプリケーション ブロックは、共通タスクの実装を簡素化して標準化する、集約された、一般には小さな抽象化レイヤです。

たとえば、データへのアクセスは、今日のアプリケーションではきわめて一般的なタスクです。データにアクセスする方法は多数ありますが、開発者は長年の経験から、それらの方法のうち効果的なものはごくわずかだということを学んできました。Data Access Application Block (DAAB) の開発者は、共通のデータベース アクセス パターンを見つけ出し、それらを小規模な一貫したインターフェイスに抽象化しました。このインターフェイスによって、データベース アクセスから面倒な作業の大半が取り除かれます。ADO.NET を通じてデータに直接アクセスするために必要なコードの量に比べると、DAAB を使用した作業はわずかなものです。DAAB は独立したクラスに実装されているため、多くの異なるアプリケーションできわめて簡単に、そのまま使用できます。

.NET で現在使用できるアプリケーション ブロックは以下のとおりです。

- Asynchronous Invocation Application Block for .NET。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/PAIBlock.asp です。

- Caching Application Block。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/CachingBlock.asp です。
- Configuration Application Block。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cmab.asp です。
- Data Access Application Block。
  URL は http://msdn.microsoft.com/library/en-us/dnbda/html/daab-rm.asp です。
- Exception Management Application Block。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/emab-rm.asp です。
- Logging and Instrumentation Application Block。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/Logging.asp?frame=true です。
- Security Application Block。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/security1.asp です。
- Smart Client Offline Application Block。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/offline.asp です。
- Updater Application Block – バージョン 2.0。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/updater.asp です。
- User Interface Process (UIP) Application Block – バージョン 2.0。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/uipab.asp です。
- User Interface Process Application Block for .NET。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/uip.asp です。

これらのアプリケーション ブロックは、いずれもソース コードおよびサンプルと共に提供されており、無料でダウンロードして使用できます。

アプリケーション ブロックの詳細については、付録 B の「アプリケーション ブロック、フレームワーク、およびその他の開発支援」の「Visual Basic .NET アプリケーション ブロックの使用」を参照してください。

パターンへのリファクタリング

リファクタリングとは、最新のソフトウェア エンジニアリング品質基準に応じて既存のアプリケーションを変換することを目的とした、反復的なプロセスです。アプリケーションにリファクタリングを適用すると、特に明瞭性、管理性、および冗長性削減の点で、アプリケーションの特性が向上します。リファクタリングでは、変数名やサブルーチン名を変更するといった小規模な変更を行うこともあれば、機能を単一のコンポーネントに統合したり、既存のコンポーネントを、機能をより論理的に分離する複数のコンポーネントに分割するといった、大規模な変更が必要になる場合もあります。

リファクタリングのための最良のアプローチは、少しずつ行うことです。これは、リファクタリング プロセスで欠陥が生じないようにするために役立ちます。小規模なリファクタリング プロセス、たとえば問題のある変数名の特定などは、ほんの数分で終わることがあります。大規模なリファクタリング プロセス、たとえば統合または分割が可能な機能の洗い出しなどは、数時間から数日、場合によっては数週間もかかることがあります。ただし、広範または集中的なリファクタリングでも、少しずつ作業を進めるのが最善であることに変わりはありません。

リファクタリングを適用する理由はいくつかあります。以下の理由は、その中で最も一般的なものです。

- **コードの修正や新しいコードの追加の簡素化**。共通のコードのグループ化や分離により、機能を集中化できます。機能を変更する必要がある場合は、1 つの箇所で済ませることができます。
- **既存のコードのデザインの改善**。リファクタリングを行うと、コードが明快になり、企業の開発標準を適用するのが容易になります。また、開発者はリファクタリングを利用することで、システムの異なるモデルやデザインを容易に開発およびテストし、どれが企業のニーズに最も適しているかを特定することができます。
- **コードについてのより良い理解**。リファクタリングによってコードが明快になり、整理されるため、人間の読み手が機能の主要な概念を識別して理解するのが容易になります。
- **拡張および統合機能の改善**。コードを組織的かつ体系的に拡張できるため、統合と再利用を容易にする標準インターフェイスを定義することができます。

パターンへのリファクタリングは、既存のアプリケーションをソフトウェアのデザイン パターンに応じて変換することを暗に意味しています。パターンは、類似したソフトウェア デザインのグループを抽象化したものに基づいています。この抽象概念は、さらに改良され、最終的には元のアプリケーションの関連する特性のほとんどを含むモデルとなります。パターンは、実際のアプリケーション デザインに適用した論理的な帰納プロセスによって取得されます。またソフトウェア デザイン パターンは、優れたデザイン原則や標準によっても強化されています。これにより、ユーザーはパターンを適用することで、優れた製品を得ることができます。

パターンは、大規模なアプリケーションの作成と、繰り返し発生するデザインの問題を証明済みで効率の良い手引きにより解決し、ユーザーを支援することができます。アプリケーションをデザイン パターンに応じてリファクタリングした場合、最終的なアプリケーションの構造は、適用したパターンから派生したものと理解することができます。これにより、将来の開発およびメンテナンスのリスクが軽減されます。

既存のコードをパターンにリファクタリングするための労力は、コード自体の必要性と比較して評価する必要があるということを覚えておいてください。重要なのは、パターンを使用しようと努力することにより、総合的なデザイン能力は向上しますが、基本的なコーディング技術と同じく、デザイン能力も学習して養っていくものだということです。

デザイン パターンの詳細については、MSDN の「Microsoft Patterns」を参照してください。

実装

アプリケーションを改良すると、最終的には洗い出した新機能または改善点を実装することが必要になります。ここでは、アプリケーションの改良における実装の問題のいくつかについて説明します。

API 呼び出しの .NET Framework の組み込み関数での 置き換え

Visual Basic 6.0 のいくつかの制約により、多くのプログラマは、Windows API で使用できるさまざまな関数を使用することを余儀なくされていました。Visual Basic 6.0 開発者は、レジストリの操作、ウィンドウ名の検索、特殊なフォルダの名前の検索などの操作では、Windows API を使用せざるを得ませんでした。なぜなら、これらのタスクを実行する方法が他になかったからです。

Microsoft Visual Basic .NET では、Visual Basic 6.0 開発者に、Windows API に頼らざるを得なくしていた障壁がすべて解消されました。Windows API で特定のタスクを実行していたすべての関数について、.NET の組み込みの関数を使用して同じ結果を得る方法が存在します。

相互運用サービスを利用することによって Windows API を引き続き使用することは可能ですが、.NET アプリケーションでは、通常は .NET Framework に含まれている関数を使用することをお勧めします。.NET アプリケーションを Visual Basic 6.0 と同じ API で動かすために必要な作業の量は、予想以上に多くなることがあります。これらの API を .NET で呼び出すのは、Visual Basic 6.0 のときほど単純ではありません。また、Visual Basic 6.0 からのデータ型の変更を API の型仕様に準拠させるために必要な修正を加えるために、余分なオーバーヘッドが生じる可能性があります。さらに、.NET アプリケーションで API 呼び出しを使用すると、アプリケーションはすぐに Windows オペレーティング システムの特定のバージョンに固有のアプリケーションに限定されてしまいます。

Visual Basic .NET で Windows API を使用しても害はありませんが、.NET の組み込みの関数を使用して API 呼び出しを置き換えると、コードの移植性と互換性が向上することがあります。最終的な結果は同じかもしれませんが、コードの一部を別の .NET 言語やオペレーティング環境に移行する必要が生じた場合は、Windows API ではなく、.NET 関数を使用することにより、制約を受けることがなくなることがわかります。

Windows API と、.NET で置き換えが可能な関数の完全な一覧については、MSDNの「Microsoft Win32 to Microsoft .NET Framework API Map」を参照してください。

レジストリ API の .NET Framework の組み込み関数での置き換え

Visual Basic 6.0 内での組み込みのレジストリ操作は、SaveSetting、GetSetting、GetAllSettings、および DeleteSettings の 4 つの関数にある程度制限されていました。これらの関数を使用してレジストリを編集することは可能ですが、アクセスできるサブツリーは HKEY_CURRENT_USER\Software\VB および VBA に制限されます。この制限により、多くのプログラマは Visual Basic 6.0 内から Windows API を使用してレジストリ変更タスクを実行しています。

.NET Framework では、Windows のレジストリ API と同じレジストリ操作機能を提供する 2 つの新しいクラスが追加されました。これらのクラスは Registry クラスと RegistryKey クラスで、Microsoft.Win32 名前空間にあります。

Visual Basic 2005 の場合:

Visual Studio 2005 では、My.Computer.Registry オブジェクトが追加され、さらに My.Computer.Registry が指す Microsoft.Win32.Registry オブジェクトに 2 つの新しいメソッドが追加されたことにより、レジストリ アクセスが簡素化されました。これらのオブジェクトは、ローカル コンピュータ上のレジストリへのアクセスを提供します。2 つの新しいメソッドは、GetValue() と SetValue() です。これらのメソッドを使用すると、最初にレジストリ ツリー内を移動することなく、レジストリの任意のキーを読み書きすることができます。詳細については、MSDN の『Visual Basic Language Reference』の「My. Computer.Registry Object」を参照してください。

Registry クラスは、レジストリにある標準の基本キーのセットを提供します。Registry クラスを使用すると、作業を行うルート キーを定義できます。基本キーを定義したら、RegistryKey メソッドを使用して必要なレジストリ操作を実行できます。

RegistryKey クラスのメソッドは、レジストリ内のサブキーの値を追加、削除、または変更するための手段を提供します。Windows のレジストリ API 関数を使用して実行する一般的な操作のほとんどについて、同等の機能を持つ .NET レジストリ メソッドが存在します。たとえば、RegCreateKey、RegOpenKeyEx、RegDeleteKey のいずれかの API 呼び出しを使用していた場合は、RegistryKey クラスのメソッドである CreateSubKey、OpenSubKey、または DeleteSubkey をそれぞれ新たに使用することができます。これらの .NET メソッドは、対応する API と同じ機能を提供し、API より使い方が簡単です。

Registry クラスと RegistryKey クラスで利用可能なプロパティとメソッド、それらの使い方、およびそれらをアプリケーションに組み込む方法の詳細については、MSDN の『.NET Framework Class Library』にある以下のリソースを参照してください。

- 「Registry Class」
- 「RegistryKey Class」

Registry クラスおよび RegistryKey クラスを使用すると、Visual Basic .NET 内からレジストリを操作できます。最終的な結果は、Visual Basic 6.0 内から Windows API のレジストリ関数を使用したときと同じです。ほとんどの場合、これらの操作を実行するために必要なコードは、対応する API を使用するときのコードよりも短くてわかりやすくなります。

シリアル化

データベースにアクセス権がないにもかかわらずアクセスする必要があるアプリケーションをコーディングするときは、プログラムで使用するデータを後で取得できるように格納する必要があることがよくあります。.NET が登場する前、データをシリアル化するには、カスタム コードが必要でした。たとえば、プログラマがクラスまたはデータ構造の情報をシリアル化する必要がある場合は、クラスまたは構造の各インスタンスを XML ドキュメント

に書き出すプロシージャを記述していました。通常、このコードをプログラミングするには膨大な作業が必要であり、実装したシリアル化が正しく機能することを確認するためのテストも必要でした。幸い、このタスクを .NET で実行するには、カスタム コードの記述や、テスト、デバッグの必要はありません。なぜなら、このタスクは入念に計画されており、きわめて簡単に実装できるからです。

.NET を使用する場合は、さまざまなシリアル化スキームをアプリケーションで使用できます。それぞれの手法には長所と短所があり、どの手法を選ぶかは、データを格納するための要件によって決まります。

1 つ目の手法では、System.XmlSerialization 名前空間にあるクラスを使用します。具体的には XmlSerializer クラスです。このクラスを使用すると、コード内のデータの XML 形式へのシリアル化および逆シリアル化が簡単にできます。まず XmlSerializer のインスタンスを作成し、シリアル化するクラスの型をコンストラクタに渡す必要があります。その後、データを格納する FileStream 型のオブジェクトを作成し、先ほど定義した XmlSerializer インスタンスの Serialize メソッドを呼び出すことができます。Employee というクラスと、このクラスの myEmployee という名前のインスタンスが存在すると仮定すると、クラスのインスタンスをシリアル化するコードは以下のようになります。

```
Imports System.Xml.Serialization
Imports System.IO
…
Dim mySerializer As New XmlSerializer(GetType(Employee))
Dim myData As New FileStream("myData.xml", FileMode.Create, FileAccess.Write)
mySerializer.Serialize(myData, myEmployee)
…
```

XmlSerializer クラスを使用したデータの逆シリアル化は、シリアル化によく似ています。ただし、取得した情報を、最初にシリアル化した型と同じ型のクラスのインスタンスに割り当てる必要があります。以下のコードは、XmlSerializer クラスを使用した逆シリアル化プロシージャを示しています。

```
Imports System.Xml.Serialization
Imports System.IO
…
Dim myDeserializer As New XmlSerializer(GetType(Employee))
Dim myData As New FileStream("myData.xml", FileMode.Open, FileAccess.Read)
Dim myEmployee As New Employee
myEmployee = myDeserializer.Deserialize(myData)
…
```

従業員クラスがプライベート プロパティを持つ場合、それらのプロパティは XML 形式にはシリアル化されません。XML ドキュメントにシリアル化されるのはパブリック プロパティだけです。この種のシリアル化を、*浅いシリアル化*といいます。

浅いシリアル化と異なり、.NET の 2 つ目のシリアル化手法であるバイナリ シリアル化では、プライベート プロパティとパブリック プロパティの両方を格納することができます。この手法は System.Runtime.Serialization 名前空間にあり、XMLSerializer の方法よりも若干多くのコーディングが必要です。

バイナリ シリアル化を使用してシリアル化するクラスでは、<Serializable()> 属性を宣言している必要があります。これにより、以下のコードの例に示すように、オブジェクトのシリアル化が可能なことがシリアル化オブジェクトに伝えられます。

```
<Serializable()> _
Public Class Employee
    Public Name As String
    ' 他のクラス プロパティがここに入ります。
End Class
```

これで、先ほど XmlSerializer クラスを使用したときとよく似た方法で、バイナリ シリアル化を使用できます。まず、データを格納するための BinaryFormatter オブジェクトを宣言する必要があります。その後、以下に示すように FileStream オブジェクトと BinaryFormatter クラスの Serialize メソッドを使用して、情報をファイルに格納できます。

```
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.IO
…
Dim mySerializer As New BinaryFormatter
Dim myData As New FileStream("myData.bin", FileMode.Create, FileAccess.Write)
mySerializer.Serialize(myData, myEmployee)
…
```

BinaryFormatter クラスを使用してデータを逆シリアル化するプロセスは、XmlSerializer を使用する方法とよく似ています。これは自習用に残しておきます。

ここで説明した 2 つのクラスを使用すると、.NET アプリケーションでのデータのシリアル化および逆シリアル化がきわめて簡単にできます。XmlSerializer を使用する手法の方が作業は少なくて済みますが、実行できる処理の点では制約が多くなります。前述のように、この手法ではクラスのプライベート プロパティはシリアル化されません。さらに、たとえば IDictionary インターフェイスを実装するクラスなど、この手法ではシリアル化できないオブジェクトもあります。XmlSerializer クラスを使用して制約に直面した場合は、それを BinaryFormatter で解決できるかどうかを調べてみてください。

.NET コレクションを使用したパフォーマンスの最適化

System.Collections 名前空間には、クラス内で関連性の深いオブジェクトのセットを実装するために使用可能な一連のオブジェクトを定義する、一連のクラスとインターフェイスがあります。この名前空間には、ArrayList、Stack、Hashtable、Queue など、すぐに使えるコレクションが含まれていますが、独自のコレクション クラスの作成に使用できる IEnumerable などのインターフェイスも含まれています。

配列を使用してインスタンスのグループを扱うのではなく、クラスでコレクションを実装すると、将来的なトラブルが大幅に軽減される場合があります。コレクションを使用すると、ランタイム エラー、無効なインデックス、配列の長さの追跡など、配列を使用することによって通常注意が必要になる点をすべて気にする必要がなくなります。またコレクションを実装すると、継承やポリモーフィズムなどの他の概念を将来容易に取り入れることができるという意味で、オブジェクト指向の好ましいプラクティスに従うことができます。さらにこのアプローチに従うと、安定したコードを再利用することにより、将来的にエラーが回避されます。

以下の例は、System.Collections 名前空間で使用可能なクラスを使ってコレクションを実装する方法を示しています。魚屋の在庫管理を任されていると仮定してください。この場合、魚が多すぎて追跡に失敗してしまうことが予想されるため、Fish クラスを実装し、さらに Fishes コレクション クラスを作成して、店舗で販売するさまざまな種類の魚をすべて追跡することをお勧めします。この機能を実現するには、Fishes クラスで System.Collections.CollectionBase を継承します。

```
Public Class Fishes
Inherits System.Collections.CollectionBase
```

次に、以下に示すように、CollectionBase 抽象クラスの add メソッドと remove メソッド、および Item プロパティを実装する必要があります。

```
…
Sub add(ByVal aFish As Fish)
    List.Add(aFish)
End Sub

Sub remove(ByVal index As Integer)
    If (index > 0 & index < List.Count - 1) Then
        List.RemoveAt(index)
    End If
End Sub

ReadOnly Property Item(ByVal index As Integer) As Fish
    Get
        Return List.Item(index)
    End Get
End Property
…
```

add メソッドと remove メソッド、および Item プロパティを実装すると、Fishes クラスを使用した要素の特定のインデックスへの追加および削除により、魚をきわめて簡単に追跡できます。

```
…
Dim gerald As New Fish("Greg", "Guppy", "Orange")
Dim nemo As New Fish("Nemo", "Clown Fish", "Orange/White")
```

```
Dim myFishCollection As New Fishes
myFishCollection.Add(gerald)
myFishCollection.Add(nemo)
myFishCollection.remove(0)
MsgBox(myFishCollection.Item(0).getName)
…
```

このように、コレクションを実装することができるクラスを構築する手順は単純です。Fishes クラスは、グループとしてまとめるオブジェクトに要素を追加したり削除したりするための簡単な方法を提供できるようになります。

クラスでコレクションを実装したくはないが、配列のサイズを変更しようとしている場合は、System.Collections 名前空間の ArrayList クラスを使用することをお勧めします。関連するオブジェクトのセットの挿入と削除では、配列は要素の直接的な追加や削除をサポートしません。配列の末尾で要素を挿入または削除しようとする場合は、この操作を Redim ステートメントを使用して実行する必要があります。これにより、通常はパフォーマンスが低下します。その他の部分で要素を挿入または削除するには、ArrayList を使用してタスクを実行するようにしてください。

1 つ注意が必要なのは、HashTables や ArrayLists などのクラスは、タイプ セーフではないということです。これらのオブジェクトは System.Object 型の要素を保持するため、現在保持している型に関係なく、あらゆる型を受け入れます。たとえば、現在ブール型の値を保持している ArrayList がある場合、Integer 型のオブジェクトを追加しても、コンパイラはコンパイル エラーを検出しません。このことはアプリケーションをランタイムに実行するまで発見されません。ランタイムにおいて、型のキャストでエラーが発生します。

Visual Basic .NET での COM オブジェクトの作成

Visual Basic .NET の CreateObject() 関数は、COM オブジェクトのインスタンスを作成して返します。この関数は、COM コンポーネントとして公開されているクラスでのみ使用できます。

この関数の使い方は以下のとおりです。

```
Public Shared Function CreateObject( _
   ByVal ProgId As String, _
   Optional ByVal ServerName As String = "" _
) As Object
```

最初のパラメータ ProgId は、作成するオブジェクトのプログラム ID を表す文字列です。2 番目の省略可能なパラメータは、オブジェクトが作成されるネットワーク サーバーの名前です。空の文字列を渡した場合は、ローカル コンピュータが使用されます。

CreateObject() 関数を使用するときは、オブジェクトの作成を変数のインスタンスに割り当てる方法について考える必要があります。以下の例では、cnADO 変数は .NET 共通言語ランタイム (CLR) によって遅延バイン

ドされます。

```
Dim cnADO as Object
Set cnADO = CreateObject("ADODB.Recordset")
```

この例の最初の行で参照されている変数は、任意の型のデータを指すことができます。柔軟性はありますが、Object 変数が遅延バインドされるため、パフォーマンスが犠牲になるという欠点があります。アプリケーションを実行した後、CLR は型チェックとメンバの参照をランタイムに行う必要があります。これがパフォーマンスのオーバーヘッドを生むのは明らかです。変数宣言を事前バインドした場合は、コンパイラが型チェックとメンバの参照をコンパイル時に実行できるため、このオーバーヘッドは発生しません。

遅延バインドに頼る代わりに、インスタンス化する COM クラスのタイプ ライブラリへの参照を追加することができます。ほとんどの場合は、Dim ステートメントと主要な相互運用アセンブリを使用して COM クラスをインスタンス化する方がより効率的であり、コードも遅延バインドを使用するコードより読みやすくてメンテナンスしやすくなります。以下の例は、CreateObject() メソッドを使用せずに Recordset のインスタンスを作成する方法を示しています。

```
Dim cnADO as ADODB.Recordset
```

リモート サーバー上でオブジェクトを作成する必要がある場合は、アプリケーションがリモート サーバーからアクセス可能であれば、CreateObject() 関数を使用してこの処理を行うことができます。そのためには、サーバー名を 2 番目のパラメータとして渡します。以下はそのコード例です。

```
Sub CreateRemoteExcelObj()
   Dim xlApp As Object
   xlApp = CreateObject("Excel.Application", "\\YourServerName")
   MsgBox(xlApp.Version)
End Sub
```

まとめ

一般に、Visual Basic .NET と .NET Framework で得られる利点を活用するには、新しい機能やテクノロジを使用して、アップグレード後のアプリケーションを改良する必要があります。この章では、アプリケーションを Visual Basic .NET にアップグレードした後にアプリケーションのアーキテクチャ、デザイン、および実装を改良する方法に関する、初歩的な情報をいくつか提供しました。

この章と以降の章で説明している可能性は、出発点に過ぎません。Visual Basic .NET と .NET Framework を詳しく学習すれば、アプリケーションの企業にとっての価値を将来に向けて高めるために、アプリケーションに追加することができるテクノロジや機能がもっと見つかります。これらの可能性を見つけるための手始めとして、次の章では、一般的なアプリケーション シナリオのための改良について説明します。その次の章では、Web ア

プリケーションのための改良を紹介します。最後に、セキュリティ機能やパフォーマンス機能など、技術的なニーズに基づいた改良に関する章で、一連の説明をまとめます。

詳細情報

オブジェクト指向のデザイン テクニックと開発テクニックについての完全な説明は、このガイドの範囲を超えています。オブジェクト指向デザインの詳細については、MSDN の「Object-Oriented Programming in Visual Basic」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vbconprogrammingwithobjects.asp です。

インターフェイスの詳細については、MSDN の「Interfaces in Visual Basic .NET」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vaconinterfaces.asp です。

デザイン パターンの詳細については、MSDN の「Microsoft Patterns」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/MSpatterns.asp. です。

Windows API と、.NET で置き換えが可能な関数の完全な一覧については、MSDN の「Microsoft Win32 to Microsoft .NET Framework API Map」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/win32map.asp です。

レジストリ オブジェクトの詳細については、MSDN の「Visual Basic Language Reference」の「My.Computer.Registry Object」を参照してください。
URL は http://msdn2.microsoft.com/library/sykcb9xf(en-us,vs.80).aspx です。

Registry クラスと RegistryKey クラスで利用可能なプロパティとメソッド、それらの使い方、およびそれらをアプリケーションに組み込む方法の詳細については、MSDN の『.NET Framework Class Library』にある以下のリソースを参照してください。

- 「Registry Class」。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfmicrosoftwin32registryclasstopic.asp です。
- 「RegistryKey Class」。
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfmicrosoftwin32registrykeyclasstopic.asp です。

18

一般的なアプリケーションの改良シナリオ

考えられるアプリケーションの改良に関して議論するには、典型的なアプリケーション シナリオについて考察するのが最も良い方法といえるでしょう。この観点から、改良するアプリケーションの種類に応じた改良方法を見つけることができます。

アプリケーションのカテゴリは多数あり、それぞれに改良の余地があります。この章では、Windows ベースのアプリケーション (フォームベースのアプリケーション) と、ビジネス アプリケーション (エンタープライズ サービス) の 2 つのカテゴリについて取り上げます。

Windows アプリケーションと Windows フォーム スマート クライアント

スマート クライアントは、ローカル リソースを活用し、分散したデータ ソースにインテリジェントに接続することで、適応性、応答性、対話性に優れた、配置と管理が容易なクライアント アプリケーションです。スマート クライアントは、一般にインターネットを通じて接続されたシステムであり、ユーザーのローカル アプリケーションとリモート アプリケーションの連携を可能にします。たとえば、画像認識アプリケーションを実行するスマート クライアントが、インターネットを経由してリモート データベースと通信し、データベースから画像パターンを収集して画像認識処理で使用することが可能です。スマート クライアントを定義する特性は以下のとおりです。

- オフラインでの動作が可能です。インターネットに接続されていない状態でもデータを処理できます。
- 中央のサーバーからネットワークを経由してリアルタイムで配置と更新を行うことができます。
- Web サービス上に構築されるため、複数のプラットフォームと言語をサポートします。
- デスクトップ コンピュータやポータブル コンピュータを含め、インターネットに接続可能なほぼすべてのデバイス上での動作が可能です。

スマート クライアントは、ホスト デバイスで提供される機能を最大限に活用するように構築できます。また、これらのデバイスの標準的なユーザーに最良のユーザー エクスペリエンスを提供するように調整できます。

アーキテクチャの改良

スマート クライアントは、リッチ クライアント モデルが持つ利点と、シン クライアントが持つ管理の容易性を備えていますが、従来のリッチ クライアント アプリケーションよりもはるかに高い柔軟性も備えています。スマート クライアント ソリューションは、複数のクライアント アプリケーションの機能で構成され、それぞれのアプリケーションが互いに連携してユーザー機能を提供します。このような複合アプリケーションは、クライアント側のソフトウェア リソースを一貫したソリューションに統合したり、既存のアプリケーションの機能を拡張してスマート クライアント機能を提供します。

以下の特性は、従来のリッチ クライアント アプリケーションで提供される機能を超えた、スマート クライアントによって提供される機能の手引きになります。クライアント アプリケーションがこれらの特性を持っている場合は、そのアプリケーションをスマート クライアントと呼ぶことができます。

- **ローカル リソースの利用。** スマート クライアントは必ずクライアント上にコード ステートメントを持っており、ローカル リソースの利用が可能になっています。ローカルの CPU や GPU、ローカルのメモリやディスク、または電話やバーコード/RFID リーダーなどの、クライアントに接続されている任意のローカル デバイスを利用します。また、Microsoft Office アプリケーションや、連携が可能なインストール済みアプリケーションなどのローカル ソフトウェアを利用することもあります。
- **接続。** スマート クライアントはスタンドアロンで動作することはなく、必ず大規模な分散ソリューションの一部を構成します。たとえば、アプリケーションはデータや基幹業務 (LOB) アプリケーションへのアクセスを提供する多数の Web サービスと連携します。アプリケーションのメンテナンスを支援し、配置と更新のサービスを提供するサービスにアクセスすることもあります。
- **オフライン機能。** スマート クライアントは、インターネットに接続されていても接続されていなくても動作します。ローカル コンピュータ上で動作することは、スマート クライアントの大きな利点の 1 つです。ユーザーがインターネットに接続されていないときでも動作するように作成できます。

スマート クライアントは、ローカルのキャッシュと処理を利用して、ネットワーク接続が使用できない間や接続が制限されている間の操作を可能にします。この機能は、モバイル接続のコスト、待ち時間、速度という点で特に有用です。

オフライン機能はモバイルのシナリオだけで役立つわけではありません。デスクトップ ソリューションでも、バックグラウンドのスレッドでサーバー システムを更新するために、オフライン アーキテクチャの利点を生かすことができます。これによりユーザー インターフェイスは応答を続けることができ、全体的なエンドユーザー エクスペリエンスが改善されます。このアーキテクチャでは、サーバーからスマート クライアントにユーザー インターフェイスを送る必要がないため、コストとパフォーマンスの面でも利点があります。

スマート クライアントは、他のシステムと必要なデータだけをバックグラウンドでやり取りするため、他のシステムとやり取りするデータの量が削減できます。物理的に結線されたクライアント システムでも、この帯域幅の削減により大きなメリットが得られます。ユーザー インターフェイス (UI) の表示処理がリモート システムで行われないため、UI の応答性が高まります。

インテリジェントなインストールと更新

スマート クライアントでは、従来のリッチ クライアント アプリケーションよりもはるかにインテリジェントに配置と更新が管理されます。Microsoft .NET Framework を使用すると、単純なファイル コピーや HTTP によるダウンロードなど、さまざまな手法を使用してアプリケーションを配置できます。アプリケーションは実行中でも更新が可能で、URL をクリックすることで必要に応じて配置できます。.NET Framework は、アプリケーションとそれに関連するアセンブリの整合性を保証する強力なセキュリティ メカニズムを備えています。

自己更新と ClickOnce

ClickOnce は、Windows フォーム アプリケーションを Web アプリケーションと同じように簡単に配置できるようにするための、新しいアプリケーション配置テクノロジです。このテクノロジは .NET Framework 2.0 で使用可能であり、Visual Studio 2005 に含まれています。

ClickOnce を使用すると、Web ページ上のリンクをクリックするのと同じくらい簡単に Windows フォーム アプリケーションを実行できます。アプリケーションを配置または更新するには、管理者はサーバー上のファイルを更新するだけでよく、すべてのクライアントを個別に操作する必要がありません。

ClickOnce アプリケーションでは、以下の理由により本質的に他への影響が小さくなっています。

- アプリケーションは完全に自己完結型であり、ユーザーごとにインストールされます。そのため、管理者権限は必要ありません。
- ClickOnce アプリケーションは自己完結型であるため、他のアプリケーションを壊してしまうことがありません。ただし、アプリケーションのインストール時に、ドライバのインストールなど、危険を伴う処理を行う必要がある場合は、Microsoft Windows インストーラを使用することをお勧めします。

ClickOnce アプリケーションは、Web サーバー、ファイル サーバー、DVD などを通じて配置できます。インストールすることもできます。インストールすると、[スタート] メニューと [プログラムの追加と削除] にエントリが作成されるか、キャッシュに格納されて動作します。さらに、ClickOnce には、アプリケーションの更新を自動的に確認するように構成する方法がいくつかあります。また、アプリケーション中で ClickOnce API (System.Deployment) を使用し、いつ更新するかを制御することもできます。

クライアント デバイスの柔軟性

.NET Framework と .NET Compact Framework は、スマート クライアント アプリケーションを構築するための共通の環境を提供します。複数のバージョンのスマート クライアントが存在することもよくあります。その場合、各スマート クライアントは特定の種類のデバイスをターゲットとしており、デバイス固有の機能を利用して、デバイスの使用目的に合った機能を提供します。

新しいテクノロジ

ここでは、Visual Basic .NET で提供されている新しいテクノロジのうち、Windows クライアントとスマート クライアントに関係するものについて説明します。

ドッキングとアンカーリングによるサイズ変更ロジックの置き換え

Visual Basic 6.0 では、サイズが変更されたフォーム上でコントロールを正しく表示するには、Resize イベントに対するカスタム コードを記述する必要がありました。これにより、サイズ変更が可能なアプリケーションで、フォーム自体のサイズに関係なく、フォーム上のコントロールの元の配置を維持できるようになります。しかしこの方法では、フォームごとにカスタム コードを記述する必要があるため、コストがかかります。フォームが数個程度であれば大きな問題ではありませんが、大規模なアプリケーションでは、このカスタム コードを実装するために非常に時間がかかります。

Visual Basic .NET では、フォーム中のコントロールの Docking プロパティと Anchoring プロパティを設定することで同じ結果を得られるため、コードを記述する必要がありません。ここでは、これらのプロパティを効率的に使用し、.NET アプリケーションでフォームのサイズにかかわらず正しくコントロールを表示する方法について説明します。

コントロールの Anchor プロパティは、コントロールが画面上に動的に配置されるように変更できます。コントロールがフォームに固定されている状態でフォームのサイズが変更されると、フォーム上のコントロールの位置は、それを保持しているアンカーの位置から相対的に同じになるように維持されます。既定では、図 18.1 に示すように、Anchoring プロパティは左上に設定されています。

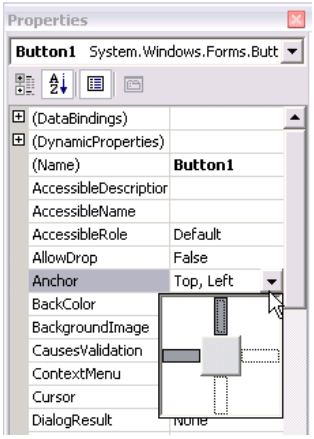

図 18.1

フォーム上のボタンの Anchor プロパティ

ダーク グレーのアンカーが現在設定されているアンカーです (上と左)。アンカーを設定したり設定を解除したりするには、そのアンカー バーをクリックします。

図 18.2 は、下部にアンカーが設定されたボタンのあるフォームの、フォームのサイズを変更する前後の様子を示したものです。

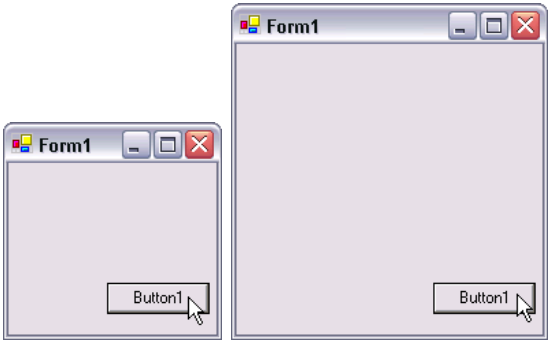

図 18.2

アンカーが設定されたボタンのあるフォームのサイズ変更

図 18.2 で、左のフォームは初期状態のフォームで、右のフォームがサイズ変更後のフォームです。アンカーリングにより、フォームのサイズが変更されても、フォーム上の相対的なボタンの位置が自動的に維持されます。

両方の縁から同じ距離を保つために、コントロールの対向するアンカーリング設定を行うと、コンテナの高さや幅が変更されたときに、コントロールのサイズが変わります。たとえば、ボタンの上と下のアンカーリング属性が設定されている場合にフォームの高さを大きくすると、ボタンの高さも変わります。コントロールのアンカーを左と右に設定してフォームの幅を変更した場合も同じように動作し、両方の縁からの距離が保たれるようにボタンの幅が変わります。図 18.3 に、異なる Anchoring プロパティを持つ 3 つのボタンと、フォームの幅と高さを変更した後の各ボタンの状態を示します。

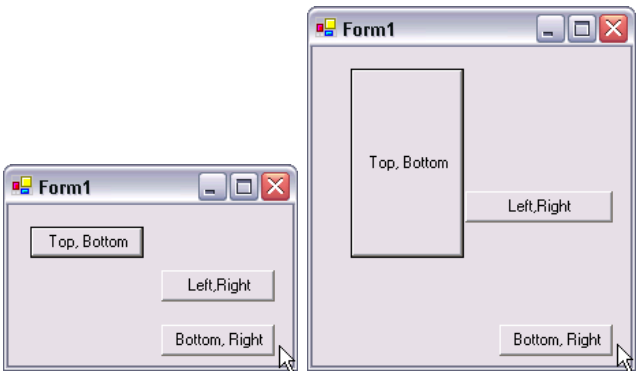

図 18.3

対向するアンカーリング

図 18.3 で、左のフォームは初期状態のフォームで、右のフォームがサイズ変更後のフォームです。

コントロールの Docking プロパティを使用すると、それを包含するフォームまたはコンテナの端にコントロールを固定できます。ドッキング動作は、.NET の前からいくつかのコントロールで使用可能でした。Visual Basic 6.0 の Toolbar コントロールでは、それ自体をフォームの上端に固定することでドッキング動作を行っていました。Visual Basic .NET では、多数のコントロールの Docking プロパティを設定して、フォームのサイズが変更されたときにコンテナの内部にコントロールを固定できます。

コントロールの Docking プロパティを変更すると、フォームのサイズが変更されたときの固定特性を設定できます。表18.1 に、コントロールに対して設定可能なさまざまな Docking プロパティを示します。

表 18.1: Docking プロパティ

Dock プロパティ	説明
Top	コントロールの上端が親コンテナの上端に固定されます。
Bottom	コントロールの下端が親コンテナの上端に固定されます。
Left	コントロールの左端が親コンテナの左側にドッキングされます。
Right	コントロールの右端が親コンテナの右側にドッキングされます。
Fill	コントロールのすべての端が親の端にドッキングされ、コントロールのサイズがそれに応じて変更されます。
None	コントロールはドッキングされません。

コントロールをコンテナの左側または右側にドッキングすると、コントロールの高さはコンテナの高さと同じになります。同様に、コンテナをコンテナの上端または下端にドッキングすると、コンテナの幅はそれを包含するコンテナの幅と同じになります。ドッキングされた複数のコントロールが同時に存在すると、2 番目のコントロールが最初のコントロールの横にドッキングされます。たとえば、図 18.4 で、"2-Dock = Right" というラベルのボタンは左にドッキングされ、"1-Dock = Top" というラベルの付いたボタンの上端に固定されます。これは、後者のボタンのドッキング プロパティを最初に設定したためです。

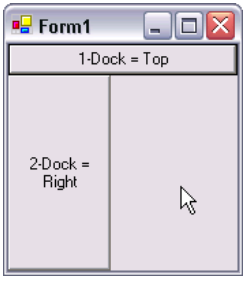

図 18.4

フォーム上のボタンに適用された Docking プロパティ

2 つまたはそれ以上のコントロールが同じコンテナの中にあり、同じ Docking プロパティを共有している場合、そのドッキング動作は特別な動作になります。互いに重なって配置される代わりに、互いに並んで配置されます。Dock プロパティが最初に設定されたコンテナが、親コンテナの端に最も近いコンテナになります。

図 18.5 に、フォーム上の異なる境界に Docking プロパティが設定されたいくつかのコントロールを示します。フォームのサイズが変更されたときに Docking プロパティがどのように影響するかに注意してください。

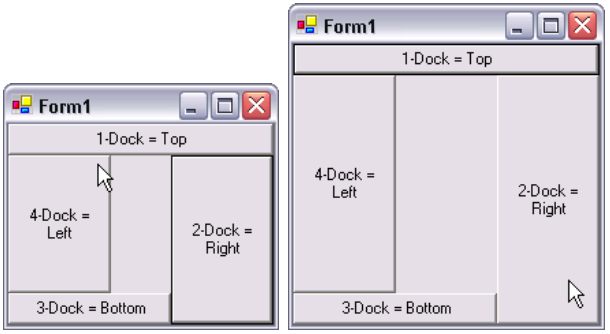

図 18.5

サイズが変更されたフォームに適用されたドッキング

コントロールの Docking プロパティと Anchoring プロパティを使用することで、フォーム (またはコンテナ) が変更されたときのコントロールの動作をより細かく制御できます。コンテナのサイズが変更になった場合にコントロールを配置するためのコードを記述する必要はありません。アンカーリングとドッキングの背景にある考え方は、最初は理解しにくいかもしれませんが、何度か練習とテストを行うことで、サイズ変更イベントとカスタムコードに頼ることなく、これらのプロパティを最適に実装する方法を学ぶことができます。

組み込みコントロール

アプリケーションの GUI を強化する Visual Basic 6.0 のコントロールの多くは、ActiveX ライブラリをインポートすることで使用可能でした。たとえば、ImageList、Treeview、または Toolbar を使用する場合、Microsoft コモン コントロールの .ocx への参照を追加することで簡単に実現できました。この参照を追加すると、これらのきわめて便利なコントロールがアプリケーションで利用できるようになりました。

これらのコンポーネントの多くは、Microsoft Visual Basic アップグレード ウィザードによって自動的に .NET にアップグレードされますが、Visual Basic .NET の組み込みコントロールにアップグレードされるわけではありません。代わりに、ActiveX コンポーネントにアップグレードされます。唯一の例外は、Microsoft Tabbed Dialog Control とカスタム ユーザー コントロールです。たとえば、Toolbar コントロールを Visual Basic .NET にアップグレードすると、図 18.6 に示すように、ActiveX コントロールへのラッパー参照がプロジェクトの参照に追加されます。

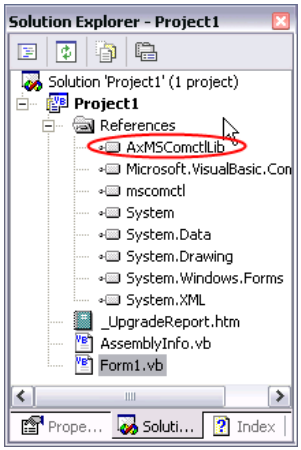

図 18.6

Visual Basic アップグレード ウィザードによって自動的に追加された ActiveX コントロールへの参照

Visual Basic 6.0 からアップグレードされた ActiveX コントロールのほとんどは Visual Basic .NET にアップグレードしても正常に動作しますが、機会があればネイティブ Windows フォームで置き換えることをお勧めします。必ずしも使う必要のない ActiveX コントロールへの参照を削除することで、プロジェクトのメンテナンスが容易になります。さらに、参照を削除することで、ActiveX コントロールをユーザーのマシンにインストールする際の配布や登録について気にかける必要がなくなります。多くの場合、ネイティブ Windows コントロールのプログラミング作業は、ActiveX コントロールのプログラミングにかかわる作業よりも簡単です。

ActiveX コントロールをアップグレードする際の難易度は、アップグレード対象の ActiveX コントロールに依存します。たとえば、Add メソッドでノードを追加する TreeView コントロールまたは ListView コントロールを使用していると、作業にオーバーヘッドが生じる可能性があります。Add メソッドのパラメータの署名が .NET で変更されている可能性があり、Visual Basic 6.0 コードに実装していたロジックと同じロジックを実現するには、多くの場合コードを確認する必要があります。

ImageList コントロールや Toolbar コントロールなどの ActiveX コントロールでは、これらの項目が保持しているコレクションのインデックスが Visual Basic .NET で変更されている点に注意してください。Visual Basic での最小のインデックスは 1 ですが、Visual Basic .NET での最小のインデックス位置は 0 から始まります。実行時に誤ったインデックスを参照しないように、このようなケースを確認して修正する必要があります。たとえば、以下のコードに示す Visual Basic 6.0 のツール バー イベントを見てください。

```
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
    Select Case Button.Index
    Case 1:
        MsgBox "Button Click 1 Event"
    End Select
End Sub
```

組み込みコントロールを使用して Visual Basic .NET で同じ動作を実現するには、デザイナから古い ToolBar の参照を削除した後で、上記のコードを変更し、Visual Basic .NET での正しいインデックスに対応させる必要があります。また、ButtonClick イベントの署名も変更されており、以下のように適切に書き換える必要がある点にも注意してください。

```
Private Sub ToolBar1_ButtonClick(ByVal sender As Object, ByVal e As _
            System.Windows.Forms.ToolBarButtonClickEventArgs) _
            Handles ToolBar2.ButtonClick
    Select Case ToolBar2.Buttons.IndexOf(e.Button + 1)
        Case 1
            MsgBox("Button Click 2 Event")
    End Select
End Sub
```

また、ActiveX のイベントおよびメソッドの特定の署名が、Windows コントロールで使用されていたものとは異なる点にも注意する必要があります。同等の機能を維持するには、すべてのイベントを見直し、場合によっては、イベントを作成し直して、意図したとおりに動作するようにする必要があります。

すべての ActiveX コンポーネントをネイティブ Windows フォーム コントロールに移植するために必要な作業は、使用していたコントロールと、それに対して行っていた操作の複雑さによって異なります。実際には、基本的な操作のほとんどは .NET 側にも完全に備わっています。複雑なプロシージャはそれほど簡単に実装できない場合もありますが、実現不可能というわけではなく、単により多くの作業が必要になるというだけです。Microsoft Visual Studio 2005 では、Visual Basic アップグレード ウィザードがほとんどの ActiveX コンポーネントを Windows フォーム コントロールに自動的にアップグレードします。

ErrorProvider コントロールを使用した視覚的なフィードバックの提供

アプリケーションがユーザーに入力を求める際には、ユーザーが無効な情報を入力する危険性が必ずあります。この理由は、単純なミスから、コードの脆弱性を悪用しようという悪意を持った試みまでさまざまです。エラーの理由に関係なく、コード中で入力を確認する必要があります。

ユーザーが無効なデータを入力したことがわかったら、次に行うべきことは、ユーザーに対してエラーを訂正するよう促すことです。Visual Basic 6.0 では、この作業を行う最も一般的な方法は、MsgBox 関数を使用することでした。このアプローチはさまざまな目的でうまく機能しますが、この入力エラーの検証処理を改良する方法がいくつかあります。

メッセージ ボックスを使用してエラーを表示する方法の最大の欠点は、場合によってはテキスト メッセージでは不十分であるという点です。たとえば、フォームに多数の入力フィールドがあり、複数のエラーが発生した場合、メッセージ ボックスを閉じた後では、ユーザーがすべてのエラーを覚えていない可能性があります。ユーザーは、入力の問題を解決するためにエラーを何度も確認しなければなりません。

Visual Basic .NET では、ErrorProvider コントロールを使用することで、エラーを報告し、エラーの場所を最善の方法でユーザーに伝えることができます。このコントロールを使用した場合とメッセージ ボックスのアプローチを使用した場合の主な違いは、ユーザーにエラーを警告する手段です。テキスト メッセージで表示する代わりに、フォームの入力検証コントロールの中で注意が必要なものに、問題の内容と場所を正確に知らせるためのアイコンが表示されます。エラー通知アイコンの上にマウスを置くと、さらなる支援のためのツールヒントがエラーを解決するための手順と共に表示されます。

ErrorProvider コントロールは、現在のコントロールに簡単に組み込むことができます。ErrorProvider を使用することで、同じことを実現するためにカスタム実装を使用するよりも、効率的でエラーが起こりにくくなります。

以下に、ErrorProvider コントロールの使用例を示します。図 18.7 に示すように、データベースに格納するユーザー情報を要求する単純なフォームがあるとします。

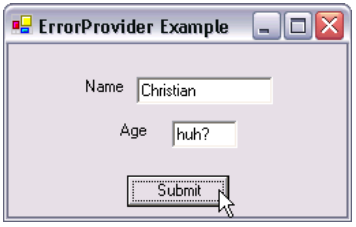

図 18.7

無効な入力データを指定したサンプル入力フォーム

このフォームには明らかな間違いがあり、ユーザーに警告する必要があります。そのためには、ErrorProvider コントロールを使用して、問題の場所と内容をユーザーに知らせます。ErrorProvider コントロールを追加するには、図 18.8 に示すように、[ツール] ペインにアクセスし、対応するツール アイコンを選択します。

図 18.8

ErrorProvider *コントロールの選択*

以下のコードを記述するだけで、問題がある場所をユーザーに警告できます。

```
…
If Not validAge(txbAge.Text) Then
    textErrorProvider.SetError(txbAge, "Age should only be a number")
End If
…
```

SetError 関数は、エラーがあることを示したいコントロールと、ユーザーがマウスをエラー アイコン上に置いたときに表示されるテキストをパラメータとして受け取ります。図 18.9 は、無効な入力が行われ、ErrorProvider に対してマークを付けるように通知した後のフォームを示しています。

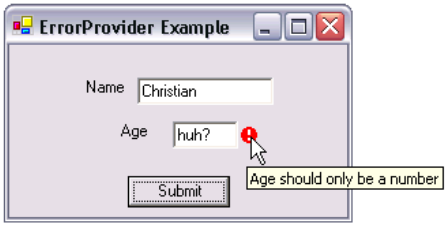

図 18.9

ErrorProvider によるテキスト ボックス コントロールの入力ミスの通知

ErrorProvider コントロールは実装が非常に簡単で、エラーの表示という点ではメッセージ ボックスよりも優れています。このコントロールは問題のある場所を明確に表示し、完全にカスタマイズ可能です。この強力な機能の詳細については、MSDN の『.NET Framework Library』の「ErrorProvider Class」を参照してください。

FlowLayoutPanel と TableLayoutPanel

Microsoft Visual Studio 2005 では、コントロールの配置を容易にするための 2 つの新しいコンテナ (FlowLayoutPanel と TableLayoutPanel) が追加されています。これらのコントロールと以前使用可能であった Panel コンテナ コントロールの違いは、その中にコントロールを配置する方法です。この 2 つの新しいコントロールを使用すると、その中に含まれるコントロールを表示する方法と場所を完全にカスタマイズできます。どちらのコントロールも、Microsoft Visual Studio .NET の [ツールボックス] ペインのコンテナ カテゴリにアクセスすることにより、フォームに追加できます (図 18.10 を参照)。

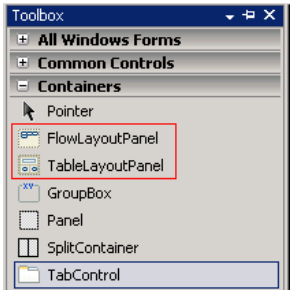

図 18.10

[ツールボックス] ペインの FlowLayoutPanel と TableLayoutPanel

FlowLayoutPanel を使用すると、コントロールをコンテナに追加する際の自動的な配置が容易になります。パネルにコントロールを追加すると、コンテナに合わせて上下および左右に移動します。これは、いくつかのコントロール (複数のテキスト ボックスやラベルなど) の集まりを完全に揃えたいような場合に便利です。図 18.11 では、Label1 と最下段のテキスト ボックス コントロールが最初に追加されています。コントロールを追加していくと、既存のコントロールがそれぞれ下方向に移動します。

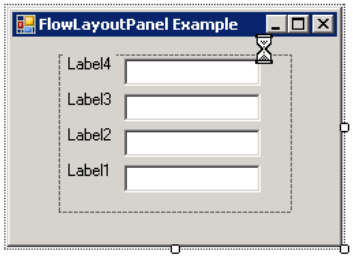

図 18.11

FlowLayoutPanel *の例*

FlowDirection プロパティの値を変更することで、FlowLayoutPanel 内でコントロールが移動する順序を変更できます。表 18.2 と図 18.12 に、FlowLayout プロパティに設定できる値とその動作を示します。

表 18.2: FlowLayoutPanel プロパティ

FlowLayoutPanel プロパティ	説明
LeftToRight (既定値)	新しいコントロールを追加すると、既存のコントロールの右側に追加されます。**WrapContents** に **true** (既定値) が設定されていると、中のコントロールが下方向にも移動します。
TopDown	新しいコントロールは、既存のコントロールの下に配置されます。行が複数あり列が 1 つしかない HTML の表に似ています。
RightToLeft	新しいコントロールを追加すると、パネル内の既存のコントロールの左側に追加されます。これは、**LeftToRight** と逆の動作です。
BottomUp	新しいコントロールは、既存のコントロールの上に配置されます。

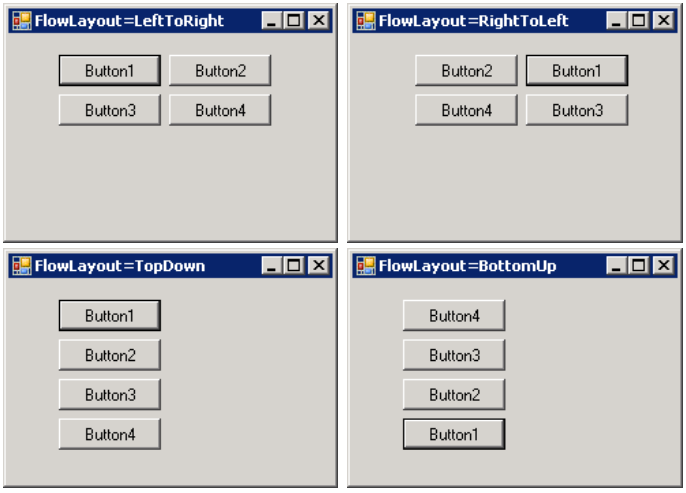

図 18.12

FlowLayoutPanel *の自動配置のバリエーション*

パネル内のコントロールの配置をさらに細かく制御する場合は、TableLayoutPanel を使用できます。このコントロールは、HTML の表と同様に、行と列を使用してコントロールを配置します。明らかな違いは、パネル内の各セルにはコントロールを 1 つしか配置できない点です。

TableLayoutPanel を使用すると、迅速で効率的な実装が可能になり、アプリケーションの GUI 部分を扱う際に長いことプログラマを苦しめてきた問題が解決されます。このコントロールを使用すると、コントロールの正確な配置を簡単に行うことができます。行と列と数を定義して配置を開始するだけです。配置済みの既存のコントロールの間に新しいコントロールが必要になった場合は、既存の表のレイアウトに新しい列を追加し、それに応じて項目を移動するだけです。

TableLayoutPanel は、デザイン時に完全にカスタマイズ可能です。ショートカット メニューにアクセスすることで、列や行の追加、列や行の削除などの操作を行ったり、強力な**列**と**行**のエディタにアクセスできます。図 18.13 に、TableLayoutPanel を編集するためにデザイン時のショートカット メニューで提供されているさまざまなオプションを示します。

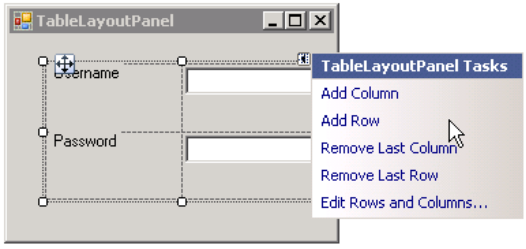

図 18.13

TableLayoutPanel のショートカット メニュー

Windows XP のルック アンド フィール

通常の Windows ユーザーであれば、システム全体でのテーマの利用についてよく知っていると思います。テーマを使用するためのオプションは既定でオンになっており、ユーザーはコンピュータ上で利用可能なさまざまなテーマとバリエーションを使用できます。

.NET Framework バージョン 1.0 で作成されたアプリケーションはテーマをサポートしていません。.NET Framework バージョン 1.1 からは、Windows XP 視覚スタイルが利用可能です。この動作は、Visual Studio 2005 の出荷時点でサポートされ、作成されたすべてのフォームが既定でテーマをサポートするようになります。図 18.14 に、ラジオ ボタンとボタン コントロールのある Visual Basic .NET アプリケーションを示します。

図 18.14

テーマをサポートしていない .NET アプリケーション

図 18.14 のボタンは、Windows 2000 のテーマを使用しているように見えます。Windows XP スタイルのテーマではボタンの角が丸くなっており、境界がライト ブルーになりますが、この図のボタンはそのようになっていません。

.NET Framework バージョン 1.1 以降を使用している場合は、テーマをサポートするように Visual Basic .NET アプリケーションのコードを変更する作業は非常に簡単です。それには、スタティック メソッド EnableVisualStyles を実行する方法と、マニフェスト ファイルを使用する方法の 2 つがあります。

EnableVisualStyles メソッドは .NET Framework バージョン 1.1 で採用されたもので、このメソッドを使用するとアプリケーションが視覚スタイルをサポートできるようになります (オペレーティング システムが視覚スタイルをサポートしている場合)。メソッドを呼び出すことで、アプリケーション内のすべてのコントロールが視覚スタイルを使用して表示されます。したがって、アプリケーションでコントロールを表示する前にこのメソッドを呼び出す必要があります。これを示すために、前の例の InitializeComponent() メソッド呼び出しの前に以下のコード例を追加します。

```
Application.EnableVisualStyles()
Application.DoEvents()
' この呼び出しは、Windows フォーム デザイナで必要です。
InitializeComponent()
```

さらに、FlatStyle プロパティをサポートしているすべてのコントロールでこのプロパティに System を設定します。同じアプリケーションを再度実行すると、図 18.15 に示すように、元の例とはまったく異なる外観になります。

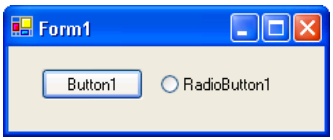

図 18.15

標準的な Windows XP のルック アンド フィールを持った .NET アプリケーション

アプリケーションにテーマを適用するために使用可能なもう 1 つの方法は、マニフェスト ファイルを作成する方法です。マニフェスト ファイルには、どのバージョンのコモン コントロール ライブラリを使用するかをアプリケーションに指示する情報が含まれます。アプリケーションでバージョン 6.0 のコモン コントロール ライブラリを使用するように指定すると、すべてのコントロールが選択したテーマに応じて表示されるようになります。

マニフェストファイルを作成するには、以下のテキストをファイルにコピーします

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
    version="1.0.0.0"
    processorArchitecture="X86"
    name="Executable Name"
    type="win32"
/>
<description>Application description</description>
<dependency>
    <dependentAssembly>
        <assemblyIdentity
            type="win32"
            name="Microsoft.Windows.Common-Controls"
            version="6.0.0.0"
            processorArchitecture="X86"
            publicKeyToken="6595b64144ccf1df"
            language="*"
        />
    </dependentAssembly>
</dependency>
</assembly>
```

このファイルを、アプリケーション名に拡張子 ".manifest" を追加した名前を付けて保存します。たとえば、アプリケーションの名前が WindowsApplication1.exe の場合、WindowsApplication1.exe.manifest という名前で、実行可能ファイルがあるディレクトリにマニフェスト ファイルを保存します。また、すべてのコントロールの Flatstyle プロパティに System を設定する必要があります。マニフェスト ファイルを使用してアプリケーションを起動すると、図 18.15 に示すように、現在オペレーティング システムが使用している Windows XP スタイルのテーマでコントロールが表示されます。

前述の方法には、それぞれ長所と短所があります。マニフェスト ファイルを使用すれば、アプリケーションでコードを記述する必要はありませんが、ユーザーがマニフェスト ファイルを削除してしまうと、アプリケーションがテーマなしで表示されるおそれがあります。EnableVisualStyles プロシージャは実装が簡単ですが、ソースコードの変更が必要です。

アプリケーションでテーマをサポートすべきかどうかは、アプリケーションをどの程度完全なものに見せたいかによります。前述のように、Windows XP ではテーマが既定でオンになっています。テーマをサポートしていないアプリケーションは、未完成または完全にサポートされていないように見えるため、ユーザーに誤った印象を与えてしまう可能性があります。

ローカリゼーションとリソース

Visual Basic 6.0 と Visual Basic .NET のどちらもアプリケーションのローカリゼーションをサポートしています。つまり、さまざまな言語向けにアプリケーションの表示を変えるためのメカニズムがサポートされています。単一のコード ベースを使用して、英語やスペイン語など任意の言語のアプリケーションを配布できます。

アプリケーションを実行する際の言語を動的に決定する場合は、現在のロケールを判定するために Visual Basic 6.0 でいくつかの Win32 関数を呼び出す必要があります。これを示すのが以下のコードです。

```
' ロケール情報を取得するための Win32 関数を宣言します。
Private Declare Function GetLocaleInfo Lib "kernel32" Alias "GetLocaleInfoA" _
(ByVal Locale As Long, ByVal LCType As Long, ByVal lpLCData As String, ByVal cchData As Long) As Long
Private Declare Function GetUserDefaultLCID Lib "kernel32" () As Long

' 現在のロケールの 3 文字の省略形を返します。
Function GetLocale() As String
Static locale_name As String
    If locale_name = "" Then
        Dim length As Long
        Dim locale_id As Long
        Dim buf As String * 1024
        locale_id = GetUserDefaultLCID()
        length = GetLocaleInfo(locale_id, &H3, buf, Len(buf))
        locale_name = Left(buf, length - 1)
    End If
    GetLocale = locale_name
End Function
```

Visual Basic .NET では、この同じ情報を現在のスレッドの CultureInfo オブジェクトから読み込むことができます。その方法を以下のコードに示します。

```
' 現在のロケールの 3 文字の省略形を返します。
Function GetLocale() As String
    Return Thread.CurrentThread.CurrentUICulture.ThreeLetterWindowsLanguageName()
End Function
```

リソース ファイル

Visual Basic 6.0 では、アプリケーションが使用する言語固有のデータを格納するために、リソース ファイルが使用されます。このデータには、文字列、イメージ、その他の種類のリソースが含まれます。1 つの Visual Basic 6.0 プロジェクトには、1 つのリソース ファイルのみを含めることができます。

アプリケーションにローカリゼーション機能を追加する方法の 1 つは、サポートするさまざまな言語に対する全データの各バージョンを、1 つのリソース ファイルに格納することです。これを行うと、ローカライズされたデータをたとえば別の文字列テーブルに格納することで分離できます。ただし、1 つの実行可能ファイルに同時

に格納できる文字列テーブルは 1 つだけであるため、すべての文字列を 1 つのテーブルに格納し、適切なオフセットを使用してアクセスするのがより一般的です。

```
' ローカライズされたリソースで使用するオフセットを決定します。
Private Function GetOffsetForLocale() As Long
    GetOffsetForLocale = 100
    If GetLocale() = "ENC" Then
        GetOffsetForLocale = 200
    ElseIf GetLocale() = "ESC" Then
        GetOffsetForLocale = 300
    End If
End Function

Private Sub Form_Load()
    Picture1.Picture = LoadResPicture(GetOffsetForLocale() + 1, LoadResConstants.vbResBitmap)
    Label2.Caption = LoadResString(GetOffsetForLocale() + 1)
End Sub
```

また、言語ごとに別々のリソース ファイルに格納することもできます。この場合、リソースファイルは独立した DLL にコンパイルされます。実行時に、ローカライズされた必要なリソースが含まれている DLL を動的に読み込みます。

```
Dim objResoures As Object
Set objResoures = CreateObject("MyApp" & Hex$(GetLocale()) & ".Resources")
If objResoures Is Nothing Then
    objResoures c = CreateObject("MyAppENU.Resources") ' 既定のロケール
End If
' objResoures からリソースを読み込みます。
```

Visual Studio .NET では、Visual Studio 6.0 よりも、はるかに統合されたローカリゼーションのサポートを提供しています。フォームの Localizable プロパティに True を設定すると、Visual Studio はそのフォームに対するリソース ファイルを作成し、さまざまな言語のテキスト文字列を別々のリソース ファイルに格納します。これらのリソース ファイルは、サテライト DLL に自動的にコンパイルされます。これらの DLL は、ユーザーのコンピュータの現在のロケールに応じて、実行時に動的かつ自動的に読み込まれます。また、より細かく制御する必要がある場合は、この処理をすべて手動で行うこともできます。この方法の詳細については、MSDN の『Visual Basic and Visual C# Concepts』の「Walkthrough: Localizing Windows Forms」を参照してください。

Visual Studio .NET では、これらのリソース ファイルにイメージを追加することは直接サポートしていません。しかし、.NET Framework SDK では、イメージをサポートする ResEditor というリソース エディタのソースコードが提供されています。また、すべてのイメージをリソースとしてメイン アセンブリに埋め込むこともできます。しかし、さまざまなロケール向けの多数のイメージがあり、インストールごとに 1 つのロケールしか必要でない場合は、この方法は過剰ともいえます。

リソースが含まれているアセンブリをより細かく制御するには、System.Resources 名前空間の ResourceManager クラスを使用する必要があります。リソースが含まれているアセンブリごとにこのクラスのインスタンスを別々に作成し、このクラスを使用してアセンブリからのすべてのリソースの読み込みを管理します。ResourceManager を作成してアセンブリからリソースを取得するには、以下のようなコードを使用します。

```
' 名前空間 System.Reflection と System.Resources を
' インポートする必要があります。

' リソースが格納されているアセンブリへの参照を
' 取得します。
Dim myResourceAssembly As Assembly
myResourceAssembly = Assembly.Load("ResourceAssembly")

' ResourceManager を作成します。
Dim myManager As New ResourceManager("ResourceNamespace.myResources", myResourceAssembly)

Dim myString As System.String
Dim myImage As System.Drawing.Image
myString = myManager.GetString("StringResource")
myImage = CType(myManager.GetObject("ImageResource"), System.Drawing.Image)
```

ツールヒント

Visual Basic 6.0 では、コントロールの ToolTipText プロパティを設定することで、コントロールにツールヒントを関連付けることができました。コントロールの ToolTipText プロパティに空でない文字列が設定されていると、マウスをそのコントロールの上に置いたときにテキストが表示されます。フォームのツールヒントを無効にするには、そのフォームのすべてのコントロールの ToolTipText プロパティに空の文字列を設定する必要がありました。Win32 の呼び出しを使用しない場合、Visual Basic 6.0 のプログラマが使用できる機能としてはこれ以外にありませんでした。

Visual Basic .NET では、ツールヒントは若干違った動作をします。各フォームにはフォーム自身の ToolTip コンポーネントがあります。ツールヒントは、ToolTip コンポーネントの SetToolTip メソッドを呼び出すことでコントロールにアタッチされます。

さらに、Visual Basic .NET では、ツールヒントの動作を簡単にカスタマイズできるいくつかの新機能が使用できます。

ツールヒントの無効化

Visual Basic .NET では、1 つのコマンドでフォーム内のすべてのツールヒントを無効にすることができます。Active プロパティは、この ToolTip コンポーネントに登録されているツールヒントを表示するかどうかを決定します。Visual Basic 6.0 でこれを実現するには、フォーム内のすべてのコンポーネントに対してこれを繰り返し、各コントロールのツールヒントを確認する以外に方法がありませんでした。Visual Basic .NET でもそれは可能ですが、ToolTip コンポーネントの Active プロパティに False を設定する方が効率的です。

```
' Visual Basic 6.0
For Each Control in Me.Controls
    Control.ToolTipText = ""
Next

' Visual Basic .NET
toolTip1.Active = False
```

ツールヒントの遅延の変更

Visual Basic .NET の新機能に、フォームのツールヒントに関する遅延を変更できる機能があります。コントロールの上にマウスを置いてからツールヒントが表示されるまでの時間を変更するには、ToolTip.InitialDelay プロパティを変更します。このプロパティの既定の時間は 0.5 秒です。ツールヒントが表示されてから消えるまでの時間を変更するには、AutoPopDelay プロパティを変更します。このプロパティは、既定では 5 秒に設定されています。ただし、ツールヒントが適用される領域からマウスが離れると、ツールヒントはすぐに消えます。ReshowDelay プロパティは、フォームの周囲でポインタを移動させたときに、以降にツールヒントが表示されるまでの時間を決定します。

前述のすべてのプロパティは、別のプロパティを使用して適切に設定できます。そのプロパティの名前は AutomaticDelay です。このプロパティを設定すると、それと等しい値が自動的に InitialDelay プロパティにコピーされ、その 10 倍の値が AutoPopDelay プロパティにコピーされ、1/5 の値が ReshowDelay プロパティにコピーされます。これを以下のコード例に示します。

```
' 遅延プロパティをミリ秒単位で設定します。
toolTip1.AutomaticDelay = 500
' ツールヒントを表示するまでの時間の長さを指定します。
toolTip1.InitialDelay = 500
' ツールヒントの表示を取り消すまでの時間の長さを指定します。
toolTip1.AutoPopDelay = 5000
' 別のコントロールに移動した後でツールヒントを表示するまでの時間の長さを指定します。
toolTip1.ReshowDelay = 100
```

Visual Basic 2005 の場合：

Visual Studio 2005 では、ツールヒントの色を変更したり、ツールヒント全体を表示できるようにするなど、ツールヒントをカスタマイズする機能が追加されています。詳細については、MSDN の『.NET Framework Class Library』の「ToolTip Class」を参照してください。

ビジネス コンポーネント (Enterprise Services)

.NET Framework では、きわめて拡張性の高いソリューションを構築するためのエンタープライズ サービスが提供されています。エンタープライズ サービス コンポーネントで必要なプロパティをコード自体から設定したり、コードから配置の設定を行うことが可能です。これにより、コード内でこれらすべての設定を扱えるようになるため、開発者にとっては大きな利点があります。

ここでは、.NET Framework Enterprise Services を利用して、アップグレード後のアプリケーションのアーキテクチャを改良するためのガイドラインについて説明します。

アプリケーション アーキテクチャの改良

Enterprise Services で使用可能なプロパティについて理解する必要があります。Enterprise Services では COM+ カタログのプロパティが以下の 2 つのカテゴリに分類されます。

- Transactions など、主に開発者が指定するプロパティ
- Identity など、主に管理者が指定するプロパティ

開発者が指定するプロパティは、System.EnterpriseServices 名前空間の属性として公開されており、これらの属性をアセンブリ、クラス、インターフェイス、またはメソッドに適用することでアセンブリに追加します。

管理者が指定するプロパティは、アセンブリを登録した後でコンポーネント サービス管理ツールを使用するか、COM+ の管理インターフェイスを使用して設定します。

Enterprise Services は、トランザクション、プーリング、キューなどのテクノロジが必要な複雑なアプリケーションを構築する際に非常に便利な、強力なテクノロジです。

Enterprise Services の最も混乱を招く側面の 1 つが、いつそれを使用するかという点です。既存のシステムで広く COM+ を使用している場合は、アプリケーションを .NET にアップグレードする際に Enterprise Services を使用することをお勧めします。しかし、現在 COM+ を使用していない場合は、どのようにして使い始めるべきか、その利点は何かについて疑問に思うかもしれません。以降では、これらの点について詳しく説明します。

Enterprise Services と属性の使用

エンタープライズ アプリケーションの作業を開始する前に、エンタープライズ アプリケーションに必要とされる基本的な属性がいくつかあります。ここではその属性について説明します。

エンタープライズ サービス コンポーネント用の .NET クラスを作成するには、COM+ サービスを利用するために、クラスは System.ServicedComponent クラスから派生させる必要があります。このためには、.NET Enterprise Services 名前空間をインポートする必要があります。特別な用途の .NET 名前空間には、対応するアセンブリへの参照が既定では含まれていないものがあるため、これらのアセンブリへの参照を新しく追加

することも必要です。System.EnterpriseServices 名前空間もこれに該当するため、対応するアセンブリへの参照を追加しなければなりません。

▶ このアセンブリへの参照を追加するには

1. ソリューション エクスプローラで、参照を追加するプロジェクトを選択します。
2. [参照設定] フォルダを右クリックし、[参照の追加] をクリックします。
3. [.NET] タブをクリックします。図 18.17 に示すように、コンポーネント一覧で System.EnterpriseServices を探してダブルクリックします。

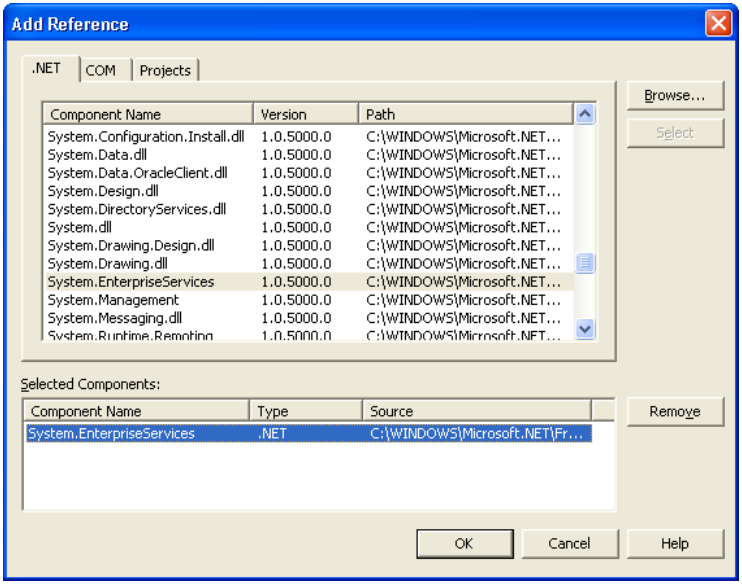

図 18.16

System.EnterpriseServices *アセンブリへの参照の追加*

4. [OK] をクリックします。

参照を追加すると、アプリケーションに System.EnterpriseServices 名前空間をインポートし、System.EnterpriseServices.ServicedComponent からクラスを拡張することが可能になります。これを以下のコードに示します。

```
Imports System.EnterpriseServices
Public Class _myClass
Inherits System.EnterpriseServices.ServicedComponent
```

COM+ アプリケーションの要件に応じて、クラスに対していくつかの属性を指定できます。以下のコードは、クラスに対して属性を宣言する方法を示しています。

```
<ObjectPooling()> Public Class _myClass
```

以降では、Microsoft .NET Framework を使用することで、Enterprise Services を利用するために使用可能な特性と機能について説明します。

COM+ は、主に Microsoft Transaction Server (MTS) と Microsoft Message Queuing (MSMQ とも呼びます) を COM の基本フレームワークに取り込んだものです。基本的には COM、MTS、メッセージ キュー、COM の拡張機能、MTS の拡張機能を統一した上でその他のサービスが追加されています。

これらのサービスには、自己記述的なコンポーネント、キュー コンポーネント、イベント、セキュリティ、トランザクション、負荷分散が含まれます。

自己記述的なコンポーネント

.NET Framework では、コンポーネントの記述の一部となる COM コンポーネントの属性を開発者が指定できます。これらのクラス属性は、コンポーネントに適用される実行時の特性です。システム管理者は、管理ツールを使用して属性を指定できます。開発者はデザイン時に .NET Framework のクラス属性を使用して、自己記述的なコンポーネントに対してこれらの特性を指定できます。たとえば、設計者は以下に示すように、コンポーネントでトランザクションが必要なことを指定します。

```
<Transaction>Public Class _myClass
    Inherits System.EnterpriseServices.ServicedComponent
    <AutoComplete()> Public Sub MyRoutine()

    End Sub
End Class
```

この特性により、クラスは、トランザクションのコミットやロールバックなどのトランザクション管理を手動で宣言しなくても、トランザクションを管理できるようになります。この定義によりデータベース操作が自動的に検出され、コードを追加しなくてもすべてのトランザクションが透過的に処理されます。

コンポーネントをインスタンス化すると、属性を渡すことができます。属性には構成ファイルを指定することもでき、その中で、データベース システムやアプリケーション サーバーを含め、使用する必要があるシステム リソースをコンポーネントに対して指示します。指定可能な属性クラスの種類の詳細については、MSDN の『.NET Framework Class Library』の「System.EnterpriseServices Namespace」を参照してください。

新しいテクノロジ

以降では、アプリケーションで使用可能な Enterprise Services 関連の新しいテクノロジについて説明します。

スレッドの使用

COM+ は、開発者に代わってスレッドを管理します。どの COM コンポーネントにも、コンポーネントの開発時に指定可能な ThreadingModel 属性があります。このプロパティは、コンポーネントの各オブジェクトをメソッド実行用のスレッドに割り当てる方法を決定します。

スレッドは、2 種類のアパートメントに関連付けることができます。

- シングルスレッド アパートメント (STA)
- マルチスレッド アパートメント (MTA)

STA は、1 つのスレッドだけを実行することを指定します。STA は、メッセージを受信してメソッド呼び出しをアパートメント内のオブジェクトにディスパッチする非表示ウィンドウとして実装されます。1 つの STA には複数のオブジェクトが存在でき、1 つのプロセスには複数の STA が存在できます。

MTA には実行スレッドのプールがあります。このシナリオでは同期を考慮する必要があります。セマフォ、クリティカル セクション、ミューテックスなどのオペレーティング システムの同期メカニズムが使用できます。MTA は、1 プロセスにつき 1 つに制限されています。

マルチスレッド化されたアパートメントに存在するすべての COM オブジェクトは、同じアパートメントに属するスレッドからのメソッド呼び出しを直接受け取ることができます。マルチスレッド アパートメント内のスレッドは、*フリースレッディング*と呼ばれるモデルを使用します。

以下のように、さまざまな種類のプロセスを定義できます。

- 1 つの STA のみで構成されるプロセスは、シングルスレッド プロセスと呼ばれます。
- 2 つ以上の STA を含み、MTA を含まないプロセスは、アパートメント モデル プロセスと呼ばれます。
- 1 つの MTA を持ち、STA を持たないプロセスは、フリースレッド プロセスと呼ばれます。
- 1 つの MTA と、1 つ以上の STA を持つプロセスは、混合モデル プロセスと呼ばれます。

実際には、すべてのプロセスがアパートメント モデル プロセスであり、アパートメントが 1 つのスレッドを持つのか、複数のスレッドを持つのかだけが異なります。スレッディング モデルは、プロセスではなくアパートメントに適用されます。オブジェクトのクラスに適用することもできますが、DLL などのコンポーネントではなく、DLL の中のオブジェクト クラスに適用されます。1 つの DLL の中には、異なるスレッディング モデルを持つ複数のクラスが存在できます。

プロセスでフリースレッド アパートメントを 1 つだけ使用している場合は、同じプロセスでアパートメント モデルとフリースレッド モデルの両方を使用できます。また、複数のシングルスレッド アパートメントを持つことができます。STA 内のオブジェクトの呼び出しは Win32 によって自動的に同期され、アパートメントの境界を越える場合にはデータのマーシャリングが必要になります。スレッディング モデルの詳細については、MSDN の「Processes, Threads, and Apartments」を参照してください。

スレッド ニュートラル アパートメント モデル (TNA) を使用すると、コンポーネントは自動的にフリースレッドまたはアパートメントとしてマークされます。このモデルを使用すると、コンポーネントは、呼び出し元のスレッドと同じスレッド タイプを継承します。

あるスレッドが COM オブジェクトに含まれているメソッドを実行し、メソッドが新しいオブジェクトを作成すると、MTS は現在のスレッドを中断し、新しいスレッドを作成して新しいオブジェクトを処理します。TNA モデルでは、アパートメントは複数のスレッドを持つことができます。

以下のコード例は、Visual Basic 2005 でスレッドのアパートメント状態を設定する方法を示しています。SetApartmentState および GetApartmentState メソッドは、Visual Basic 2005 のみでサポートされます。

```
Imports Microsoft.VisualBasic
Imports System
Imports System.Threading

Public Class ApartmentTest

    Shared Sub Main()

        Dim newThread As Thread = New Thread(AddressOf ThreadMethod)
        newThread.SetApartmentState(ApartmentState.MTA)

        ' ApartmentState は一度しか設定できないため、
        ' 次の行は無視されます。
        newThread.SetApartmentState(ApartmentState.STA)

        Console.WriteLine("ThreadState: {0}, ApartmentState: {1}", _
            newThread.ThreadState, newThread.GetApartmentState())

        newThread.Start()

        ' newThread が開始するのを待ってスリープします。
        Thread.Sleep(300)
        Try
            ' newThread はスリープ中であるため、例外が発生します。
            newThread.SetApartmentState(ApartmentState.STA)
        Catch stateException As ThreadStateException
            Console.WriteLine(vbCrLf & "{0} caught:" & vbCrLf & _
                "Thread is not In the Unstarted or Running state.", _
                stateException.GetType().Name)
            Console.WriteLine("ThreadState: {0}, ApartmentState: " & _
                "{1}", newThread.ThreadState, newThread.GetApartmentState())
        End Try

    End Sub

    Shared Sub ThreadMethod()
        Thread.Sleep(1000)
    End Sub

End Class
```

COM 相互運用機能と .NET でのスレッド プログラミングの詳細については、MSDN の『.NET Framework Developer's Guide』の「Interop Marshaling Overview」を参照してください。

トランザクションの使用

COM+ のトランザクション機能は、MTS を再度パッケージ化したものです。COM+ では、各ビットを個別に取得/設定するためのメソッド SetComplete、EnableCommit、SetAbort、および DisableCommit が定義された新しいインターフェイスが提供されています。

.NET では、分散トランザクションも自動トランザクションまたは宣言型のトランザクションと捉えることができます。パフォーマンスは、アプリケーションでトランザクション管理を行うための方法に関する意思決定プロセスの 1 つの側面に過ぎません。パフォーマンス以外にも、開発のしやすさ、柔軟性、将来の拡張性などの検討事項があります。

SqlTransaction オブジェクトまたは OleDbTransaction オブジェクトを使用してトランザクションを管理する場合は、トランザクションを自分で管理することになります。オブジェクト間でトランザクションをどのようにやり取りするかや、どのオブジェクトにトランザクションのルートとして Commit や Rollback を呼び出す役割を持たせるかについて検討する必要があります。これは、最初は難しい問題ではないように思われるかもしれません。しかし、異なるトランザクションが必要なオブジェクトがいくつもありデータベース トリガを使用している場合や、そのオブジェクト間に依存関係がありコードでの処理が複雑な場合はどうなるでしょうか。この作業は多少煩雑になりますが、トランザクションを使用すれば、どのような場合にオブジェクトがトランザクションのルートになり、どのような場合にルートでないかを決定するための特別なロジックが不要になります。

Visual Basic 2005 の場合：

.NET Framework バージョン 2.0 では、新しいトランザクション プログラミングのパラダイムが提供されています。この機能は System.Transactions 名前空間に含まれており、非同期処理、イベント、セキュリティ、同時実行管理、相互運用などの機能を提供します。

System.Transactions の詳細については、MSDNの「Introducing System.Transactions in the .NET Framework 2.0」を参照してください。

もう 1 つの重要な注意点は、アプリケーションの柔軟性です。在庫テーブルが別のデータベースに移動された場合はどうすればよいでしょうか。在庫オブジェクトがリモート サーバーに移動された場合はどうすればよいでしょうか。オブジェクトの 1 つがトランザクション メッセージ キューにディスパッチする場合はどうすればよいでしょうか。このような場合、手動のトランザクション管理を使用して構築されたシステムでは問題が発生します。トランザクション管理を行うために最初から Enterprise Services を使用して構築されたシステムでは、このような不測の事態が起きてもコードを変更する必要はありません。いずれの場合も、システムはすべてのリソース マネージャ (SQL データベースや、メッセージ キューなどのトランザクション リソース) 間でトランザクションを自動的に管理してやり取りします。さらに、Enterprise Services を使用すると、Compensating Resource Manager (CRM) を使用して、分散トランザクションに参加するコンポーネントを作成できます。これにより、トランザクションの結果がわかったときにビジネス ロジックを実行する新しい機会が得られます。

パフォーマンス

.NET Enterprise Services と COM+ を使用したときに得られるパフォーマンスと堅牢性の向上は、これらのテクノロジによって得られる拡張性に直接関係しています。拡張性が高いシステムでは、クライアントからの要求の量が増えても、クライアントが認識するパフォーマンスと応答時間が低下しません。COM+ では、負荷分散、オブジェクト プール、改良されたトランザクション管理などの機能によってこれが実現されます。

.NET Enterprise Services のパフォーマンスの詳細については、MSDN の「.NET Enterprise Services Performance」を参照してください。

キュー

メッセージ キュー (MSMQ とも呼ばれます) は、アプリケーション間の柔軟で信頼性の高い通信をサポートするプラットフォームを構成します。タスクは、それが呼び出された時点とは非同期に起動され、タスクが完了して結果が生成されるのを待つことはありません。この実行方法を使用することで、アプリケーションの応答性が高まり、使用可能なリソースが有効に利用できるようになります。この処理モードは、COM+ キュー コンポーネントとメッセージ キューを使用して実現できます。メッセージ キューの詳細については、第 15 章「MTS アプリケーションと COM+ アプリケーションのアップグレード」の「メッセージ キューとキュー コンポーネン」を参照してください。

通常の実行モードを使用している場合、COM+ コンポーネントのメソッドの呼び出し元は、メソッドが返るまで待ちます。これに対し、コンポーネントがキュー モードで使用されている場合は、クライアント側のスタブがコンポーネントとのやり取りを保存し、コンポーネントが解放されると、トランザクションの管理と復旧のための適切なプロパティなどと共に、メッセージ キューのメッセージとしてやり取りが配信されます。送信先キューがメッセージを受け取ると、やり取りが解釈されて実行されます。

アプリケーション内にキュー コンポーネントを作成する場合は、まずキュー コンポーネントを通じて非同期で呼び出したいインターフェイスを宣言し、そのインターフェイスに InterfaceQueuing 属性を設定します。このインターフェイスのメソッドでは、戻り値や ByRef 引数を使用できません。

```
Imports System.EnterpriseServices

<InterfaceQueuing()> Public Interface IClass1
    Sub MyProcess(ByVal i As Integer)
End Interface
```

キュー インターフェイスはパブリックとして宣言する必要があります。そうしないと、アセンブリを COM+ に登録したときに、キュー インターフェイスが登録されません。

このアセンブリがキュー コンポーネントであることを指定する必要があります。これを行うには、アセンブリ ファイルで ApplicationQueuing 属性を使用します。

属性が宣言されていると、このキュー コンポーネントが特別な動作をしなくてはならないことがアセンブリで指定されます。この設定は、コンポーネント内のすべてのオブジェクトに適用されます。

```
<Assembly: ApplicationQueuing()>
```

その後、サーバー アプリケーションでこのインターフェイスを実装します。ApplicationQueuing 属性は、サーバー アプリケーションに対してキューで待機するよう指示します。以下のコードは、キュー インターフェイスの実装方法を示しています。

```
Public Class Class1
    Inherits EnterpriseServices.ServicedComponent
    Implements IClass1

    Public Sub MyProcess(ByVal i As Integer) Implements IClass1.MyProcess
        ' コードを追加します。
    End Sub
End Class
```

キュー コンポーネント インフラストラクチャはシリアル化可能なオブジェクトをサポートしないという点に注意してください。そのため、以下の状況を考慮する必要があります。

- 単純な型 (int、double、string など) を渡す場合
- XML シリアライザを使用してオブジェクトを XML にシリアル化し、文字列として渡す場合
- バイナリ シリアライザを使用してオブジェクトを バイナリにシリアル化し、バイト配列 (単純な型) として渡す場合

COM+ のセキュリティ

COM+ には、MTS で提供されているものと同様のセキュリティ機能があり、Windows XP で提供される新しいセキュリティ インフラストラクチャも追加されています。アプリケーション、コンポーネント、およびメソッド レベルでアクセス制御を設定できます。ユーザーは、ロールに関連付けられているユーザー アカウントとグループを制御できます。これらの機能により、開発者は、基となるプラットフォームで提供されるさまざまな種類のサービスに基づいて、論理的なセキュリティ モデルを定義できます。

Microsoft Active Directory は、Windows XP での主要なセキュリティ コンポーネントです。公開キー インフラストラクチャ (PKI) と NT LAN Manager (NTLM) もサポートされていますが、Active Directory は主なセキュリティ サービスとして Kerberos を使用します。Active Directory では PKI Kerberos 拡張が実装されており、X.509 証明書を使用してユーザーを認証することが可能です。Kerberos のチケット保証チケット (TGT) とセッション キーは、X.509 証明書の 1 つが取得済みの場合に取得できます。

負荷分散

COM+ の負荷分散アーキテクチャを使用すると、オブジェクトの作成を使用率が低いコンピュータに転送するルーターをサーバー上で定義できます。オブジェクトが作成されると、ステートレスな場合であってもメソッド呼び出しは同じオブジェクトに対して行われます。一般的なすべてのタスクに対してウィザードがあり、管理用に 1 つの [Application Explorer] ウィンドウがあります。

COM+ イベント

COM+ イベント サービスは、自動化された疎結合型のイベント システムであり、さまざまなパブリッシャからのイベント情報が COM+ カタログ内に保存されます。サブスクライバはこの保存されたイベントに対して問い合わせを行い、受け取りたいイベントを選択できます。イベントの詳細については、第 15 章「MTS アプリケーションと COM+ アプリケーションのアップグレード」の「COM+ イベント」を参照してください。

.NET の System.EnterpriseServices 名前空間の詳細については、MSDN の『.NET Framework Class Library』の「System.EnterpriseServices Namespace」を参照してください。

まとめ

一般に、Visual Basic .NET と .NET Framework で得られる利点を活用するには、新しい機能やテクノロジを使用して、アップグレード後のアプリケーションを改良する必要があります。この章では、アプリケーションの種類ごとに、考えられるアプリケーションの改良のための推奨事項を説明しました。アプリケーションが、Windows アプリケーション、ビジネス アプリケーション、他のカテゴリのアプリケーションの場合でも、ここで説明した情報によって、アプリケーション カテゴリの観点からどのような改良を進めたらよいかがわかるでしょう。

Web アプリケーションも改良が可能です。次の章では、Web アプリケーションで考えられる改良について概説します。

詳細情報

ErrorProvider コントロールの詳細については、MSDN の『.NET Framework Library』の「ErrorProvider Class」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwindowsformserrorproviderclasstopic.asp です。

ローカライズの詳細については、MSDN の『Visual Basic and Visual C# Concepts』の「Walkthrough: Localizing Windows Forms」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbwlkWalkthroughLocalizingWindowsForms.asp?frame=true です。

ツールヒントの詳細については、MSDN の『.NET Framework Class Library』の「ToolTip Class」を参照してください。
URL は http://msdn2.microsoft.com/library/ecft989x(en-us,vs.80).aspx です。

指定可能な属性クラスの種類の詳細については、MSDN の『.NET Framework Class Library』の「System.EnterpriseServices Namespace」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystementerpriseservices.asp です。

スレッディング モデルの詳細については、MSDN の「Processes, Threads, and Apartments」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/com/html/cb62412a-d079-40f9-89dc-cce0bf3889af.asp です。

COM 相互運用機能と .NET でのスレッド プログラミングの詳細については、MSDN の『.NET Framework Developer's Guide』の「Interop Marshaling Overview」を参照してください。
URL は http://msdn2.microsoft.com/library/eaw10et3(en-us,vs.80).aspx です。

System.Transactions の詳細については、MSDNの「Introducing System.Transactions in the .NET Framework 2.0」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/introsystemtransact.asp です。

.NET Enterprise Services のパフォーマンスの詳細については、MSDN の「.NET Enterprise Services Performance」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncomser/html/entsvcperf.asp です。

.NET の System.EnterpriseServices 名前空間の詳細については、MSDN の『.NET Framework Class Library』を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystementerpriseservices.asp です。

19

一般的な Web シナリオのための改良

Web アプリケーションは、現代のコンピューティングでは一般的に使用されています。Web アプリケーションを使用することで、企業はデータやサービスを中央の場所で管理しながら、世界中のあらゆる場所からのアクセスを許可することができます。そのため、Web アプリケーションは作成するアプリケーションの種類として人気があります。

Visual Basic .NET は、堅牢で強力な Web アプリケーションの迅速な構築を可能にするクラス、オブジェクト、および言語機能を多数提供します。この章では、これらの機能の概要を示し、アプリケーションでこれらの機能をどのように活用するか判断できるようにします。

Web アプリケーションと ASP.NET

ASP.NET を使用すると、Web フォームや XML Web サービスを使用する Web アプリケーションおよび Web コンポーネントを迅速に作成することができます。Visual Basic .NET の Web フォームは DHTML モデルを拡張し、より多彩で動的なユーザー インターフェイス機能に加えて、クライアント側の検証機能も提供します。Visual Basic .NET では、ASP.NET Web アプリケーション プロジェクトを使用して Web アプリケーションを作成します。ASP.NET では、パフォーマンス、状態管理、スケーラビリティ、構成、配置、セキュリティ、出力キャッシュ制御、Web ファーム サポート、Web サービス インフラストラクチャなどが ASP よりも大幅に改善されています。

開発者は、ASP.NET アプリケーションを作成するときに Web フォームまたは Web サービスを使用するか、これらを好きなように組み合わせることができます。Web フォームと Web サービスは、いずれも同じインフラストラクチャによってサポートされています。このインフラストラクチャは、認証方式を使用したり、頻繁に使用するデータをキャッシュしたりすることを可能にし、またアプリケーションの構成をカスタマイズします。具体的には、以下のことが可能です。

- Web フォームを使用すると、フォームベースの強力な Web アプリケーションを構築することができます。これらのページを構築するときは、ASP.NET のサーバー コントロールを使用して共通の UI 要素を作成し、それらを共通のタスク用にプログラムすることができます。これらのコントロールを使用すると、再利用可能な組み込みコンポーネントまたはカスタム コンポーネントから Web フォームを迅速に構築し、ページのコードを簡素化することができます。

- XML Web サービスは、サーバー機能にリモートでアクセスする手段を提供します。Web サービスを使用すると、企業はデータまたはビジネス ロジックへのプログラム インターフェイスを公開することができます。これらのインターフェイスは、クライアント アプリケーションやサーバー アプリケーションが取得して操作することができます。Web サービスは、HTTP および XML メッセージングなどの標準を使用してデータをファイアウォール経由で移動することにより、クライアント/サーバーまたはサーバー/サーバーのシナリオでデータを交換することを可能にします。Web サービスは、特定のコンポーネント テクノロジやオブジェクト呼び出し規則に縛られません。そのため、任意の言語で記述し、任意のコンポーネント モデルを使用し、任意のオペレーティング システムで実行されるプログラムから、Web サービスにアクセスすることができます。

以下に、ASP.NET の主要な機能を示します。

- ASP.NET は、Web 開発者がアプリケーション レベルで実行されるロジックを記述するために利用できるモデルを提供します。開発者はこのコードを、Global.asax テキスト ファイル内またはアセンブリとして配置されるコンパイル済みクラスに記述することができます。このロジックにはアプリケーション レベルのイベントを含めることができますが、開発者は Web アプリケーションのニーズに合うように、このモデルを容易に拡張することができます。
- ASP.NET は、アプリケーション状態機能およびセッション状態機能を提供します。
- ASP の以前のバージョンに含まれていた ISAPI と同等に強力なプログラミング インターフェイスとして API を使用する必要がある上級開発者のために、ASP.NET は IHttpHandler インターフェイスと IHttpModule インターフェイスを提供します。IHttpHandler インターフェイスを実装すると、IIS Web サーバーの下位レベルの要求および応答と対話できるようになり、また ISAPI 拡張機能とほぼ同じ機能を、より単純なプログラミング モデルで得ることができます。IHttpModule インターフェイスを実装すると、アプリケーションに対して実行されるすべての要求に関与するカスタム イベントを含めることができます。IHttpHandler インターフェイスと IHttpModule インターフェイスの詳細については、この後の「HTTP モジュール」を参照してください。
- すべての ASP.NET コードは、解釈されるのではなく、コンパイルされます。そのため、事前バインド、厳密な型指定、およびネイティブ コードへのジャスト イン タイム (JIT) コンパイルが可能です。また、ASP.NET では簡単に要素を分解することができます。つまり、開発者は開発しているアプリケーションに関係のないモジュール (たとえば、セッション モジュール) を削除することができます。また ASP.NET は、拡張キャッシュ サービス (組み込みサービスとキャッシュ API の両方) を提供します。さらに ASP.NET にはパフォーマンス カウンタも付属しており、開発者やシステム管理者は、それらを監視して新しいアプリケーションをテストしたり、既存のアプリケーションのデータを収集したりすることができます。
- ASP.NET は TraceContext クラスを提供します。このクラスを使用すると、ページを開発するときに、カスタム デバッグ ステートメントを記述することができます。これらのステートメントは、ページまたはアプリケーション全体のトレースを有効にしたときにだけ表示されます。また、トレースを有効にすると、要求に関する詳細がページに追加されます。この詳細を、アプリケーションのルート ディレクトリ に格納されたカスタム トレース ビューアに追加するように指定することも可能です。
- .NET Framework と ASP.NET は、Web アプリケーションのための既定の承認方式と認証方式を提供します。アプリケーションのニーズに応じて、これらの方式の削除、方式への追加、または方式の置き換えを簡単に行うことができます。

- ASP.NET の構成設定は、人間が読み書きできる XML ベースのファイルに格納されます。アプリケーションごとに独自の構成ファイルを持たせることができます。また、要件に合うように構成方式を拡張することができます。

アーキテクチャの改良

ここでは、ASP.NET サブシステムおよびインフラストラクチャの概要を示します。図 19.1 に、ASP.NET のセキュリティシステム間の関係を示します。

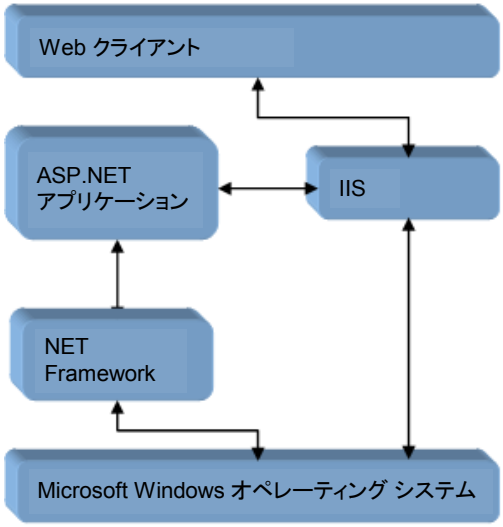

図 19.1

ASP.NET のセキュリティシステムの関係

図 19.1 に示すように、Web クライアントは Microsoft インターネット インフォメーション サービス (IIS) を使用して ASP.NET アプリケーションと通信します。IIS は要求を解釈し、オプションで要求を認証します。[Allow Anonymous] アクセスが有効にされている場合、認証は発生しません。また IIS は、要求されたリソース (ASP.NET アプリケーションなど) を検索し、クライアントが認証された場合は、適切なリソースを返します。

ASP.NET は、高性能な Web アプリケーションの設計と実装を可能にする、さまざまな機能とツールを備えています。こうした機能およびツールとして以下が挙げられます。

- ASP のプロセス モデルから改良されたプロセス モデル
- ASP.NET ページの変更を自動的に検出し、変更されたページを動的にコンパイルし、新たにコンパイルされたファイルを以降の要求で再利用できるように格納する機能
- ASP.NET 独自のパフォーマンス カウンタ
- Web アプリケーション テスト ツール

ASP.NET モデルには、パフォーマンスの強化点が数多く組み込まれています。具体的には、HTTP 要求の処理に関連する強化点は 2 つあります。

- ASP.NET ページが最初に要求されたとき、Page クラスのインスタンスが動的にコンパイルされます。ASP の以前のバージョンでは、ページ コードは要求に対し、ページ上で現れる順序で解釈されました。共通言語ランタイムは、ASP.NET のマネージ ページ コードを、処理サーバーのネイティブ コードに実行時に JIT コンパイルします。
- Page インスタンスが最初の要求に対してコンパイルされると、そのインスタンスがサーバー上にキャッシュされます。そのページに対して以降に要求があるたびに、キャッシュされたクラスのインスタンスが実行されます。最初の要求の後に Page クラスが再コンパイルされるのは、ページの元のソースまたはページのいずれかの従属関係が変更されたときだけです。

さらに、ASP.NET はサーバー変数などの内部オブジェクトをキャッシュし、ユーザー コードのアクセスを高速化します。.NET Framework の一部である ASP.NET は、共通言語ランタイム (CLR) が提供するパフォーマンスの強化点の恩恵を受けます。これには、前述の JIT コンパイルや、シングルプロセッサ コンピュータとマルチプロセッサ コンピュータの両方に対して微調整された共通言語ランタイムなどがあります。

残念ながら、これらの強化点をもってしても、アプリケーションが大量の HTTP 要求を同時に処理したときにコードのパフォーマンスが低下するという問題を解決できるわけではありません。アプリケーションがユーザーの要求を満たすことを確認するには、アプリケーションをテストする必要があります。

マスタ ページ

Microsoft ASP.NET 2.0 のマスタ ページを使用すると、Web アプリケーションの複数のコンテンツ ページに同じページ レイアウトを適用することができます。マスタ ページは、Web サイトの一貫した外観とスタイルを作成するための方法を提供します。

ほとんどの Web アプリケーションのページには、ロゴ、ナビゲーション メニュー、著作権情報などの標準的な要素があります。これらの要素を、単一のマスタ ページにすべて配置することができます。このマスタ ページに基づいてアプリケーションのコンテンツ ページを作成すれば、すべてのコンテンツ ページに同じ標準要素が自動的に含まれます。

この機能は、サイト レベルのページ テンプレート、コンテンツをきめ細かく置き換えるためのシステム、ページで使用するテンプレートについてのプログラム的かつ宣言的な制御、および統合されたデザイナ サポートを提供するテンプレート メカニズムを表しています。このメカニズムは、2 つの概念的要素で構成されます。つまり、マスタ ページとコンテンツ ページです。マスタ ページはコンテンツ ページのテンプレートとして機能し、コンテンツ ページは "埋める" 必要があるマスタ ページの各部分に挿入するコンテンツを提供します。マスタ ページは、本質的には標準の ASP.NET ページです。ただし、拡張子は .master で、<%@ page %> ディレクティブの代わりに <%@ master %> ディレクティブを使用します。このマスタ ページ ファイルは、他のページのテンプレートとして機能するため、通常は最上位の HTML 要素、メイン フォーム、ヘッダー、フッターなどを含みます。マスタ ページ内では、コンテンツ ページによってページ固有の内容を提供する場所に、

ContentPlaceHolder コントロールのインスタンスを追加します。マスタ ページの詳細については、MSDN の「ASP.NET Master Pages Overview」を参照してください。

HTTP モジュール

HTTP モジュールと HTTP ハンドラは、ASP.NET アーキテクチャに欠かせない要素です。実行された各要求は、複数の HTTP モジュール (たとえば、認証モジュールとセッション モジュール) によって処理された後、単一の HTTP ハンドラによって処理されます。ハンドラが要求を処理した後、その要求は HTTP モジュール間を逆方向に流れます。モジュールはハンドラの実行の前後に呼び出されます。モジュールは以下の機能を果たします。

- モジュールを使用すると、開発者は個々の要求をインターセプトしたり、要求に関与したり、要求を修正したりすることができます。
- モジュールは IHttpModule インターフェイスを実装します。このインターフェイスは System.Web 名前空間にあります。

HTTP モジュールを使用し、アプリケーションに送信された各 HTTP 要求に事前処理と事後処理を追加することで、ASP.NET アプリケーションを拡張することができます。たとえば、アプリケーション用のカスタム認証機能が必要な場合、最良の手法は、要求の受信時にその要求をインターセプトし、カスタム HTTP モジュールで処理することです。

HTTP モジュールの構成

HTTP モジュールは ASP.NET の構成ファイルで構成されます。これらの構成ファイルは XML ベースで、任意のテキスト エディタまたは XML エディタで編集することができます。他の .NET 構成ファイルと同様に、ASP.NET の構成ファイルも、さまざまなタグによって識別されるセクションに分割されています。<httpModules> 構成セクション ハンドラは、アプリケーション内の HTTP モジュールを構成する役目を担います。このセクション ハンドラは、コンピュータ レベル、サイト レベル、またはアプリケーション レベルで宣言することができます。以下の例は、典型的な <httpModules> セクション ハンドラの構成方法を示しています。

```
<configuration>
    <system.web>
        <httpModules>
            <add type="[COM+ Class], [Assembly]" name="[ModuleName]" />
            <remove type="[COM+ Class], [Assembly]" name="[ModuleName]" />
            <clear />
        </httpModules>
    </system.web>
</configuration>
```

上で示されているように、<httpModules> タグは、メインの <configuration> セクション内の <system.web> サブセクションの後に含める必要があります。<add> タグは、HttpModule をアプリケーションに追加することを示します。<remove> タグは、HttpModule をアプリケーションから削除することを示します。<clear> タグは、すべての HttpModule をアプリケーションから削除します。

HTTP モジュールの作成

HTTP モジュールを作成するには、IHttpModule インターフェイスを実装する必要があります。IHttpModule インターフェイスには、以下のシグネチャを含む 2 つのメソッドがあります。

```
void Init(HttpApplication);
void Dispose();
```

Init メソッドは、モジュールが自らを HttpApplication オブジェクトにアタッチするときに呼び出されます。Dispose メソッドは、モジュールが HttpApplication オブジェクトからデタッチされるときに呼び出されます。Init メソッドと Dispose メソッドは、HttpApplication によって公開されるさまざまなイベントにモジュールを結び付けるための機会を表します。これらのイベントには、要求の開始、要求の終了、認証の要求などが含まれます。HTTP モジュールの詳細については、MSDN の「HTTP Modules」を参照してください。

Web サービス

Web サービスは、分散アプリケーションの構築における重要な進化のステップです。現在、旧式のシステムはばらばらのブロックとして存在しており、互換性がないこともよくあります。そのため、企業対企業または企業対クライアントの統合はほぼ不可能です。なぜなら、双方の間に技術的な壁が数多く存在するからです。共通テクノロジの重要性が叫ばれているのはこのためです。Web サービスを使用すると、旧式のシステムを Web 上で利用できるようになり、既存のソフトウェアを再利用すると共に、システム統合の可能性を開くことができます。ここでは、Web サービス テクノロジの概要を示し、新たにアップグレードしたアプリケーションに Web サービス テクノロジを組み込む方法について説明します。

Web サービスの汎用的な定義や、Web サービスがもたらす要件やサービスについての最小限の標準は存在しません。一般に受け入れられているのは、Web サービスとはネットワーク内の異なるコンピュータ間の対話をサポートするように設計されたソフトウェア システムであるということです。

Web サービス アプリケーションとコンポーネントの間で行われる通信は、XML 形式でエンコードされます。XML は複数のプラットフォームでサポートされるオープンな標準なので、Web サービスはプラットフォームに依存せず、異なる言語やプラットフォームに基づくアプリケーション間の通信に使用することができます。たとえば、Visual Basic .NET アプリケーションは Java アプリケーションと通信することができ、Windows アプリケーションは UNIX アプリケーションと通信することができます。

その他に、Web サービスには以下のような機能があります。

- 公に公開されるインターフェイス。このインターフェイスのすべてのメンバは、クライアント アプリケーションまたはコンポーネントからアクセスすることができます。このインターフェイスは、XML 形式を使用することにより、標準言語で記述することができます。この用途で最も広く受け入れられている言語は、Web サービス記述言語 (WSDL) です。

- コンポーネントが提供するサービスを公開するメカニズム。提供されるサービスに興味のある関係者は、これらを検索することができます。現在利用できる Web サービス ディレクトリで最も広く受け入れられているのは、Universal Description, Discovery, and Integration (UDDI) です。

Web サービスにおける最も重要な技術革新の 1 つは、XML のリモート プロシージャ コールの転送媒体としての使用です。伝統的なリモート プロシージャ コール (RPC) などの従来の技術では、そのような用途に HTTP を使用していました。XML を標準として使用することで、異なるアプリケーションがデータやサービス呼び出しを容易に通信することができます。公に利用可能な Web サービスの例については、マイクロソフトの Web サイトの「Microsoft MapPoint Web Service」を参照してください。

.NET Framework クラス ライブラリには、ASP.NET の Web サービス オブジェクトへのアクセスを提供する WebService クラスがあります。このクラスは、XML Web サービスの基本クラスとして使用することができます。WebService クラスの詳細については、MSDN の「WebService Class」を参照してください。

Web サービスの利点

Web サービス テクノロジは、システムを企業内レベルで統合するためのすばらしい機会を提供します。また、顧客対クライアントの通信と対話を改善するためにも役立ちます。企業の観点から見ると、その他の利点には以下のようなものがあります。

- 最新情報にアクセスするための新しい機会が得られることによる、顧客満足の向上
- オンライン購入のプロセスが改善されることによる、売上の増加
- 電話センターへの過剰な呼び出しの負担を減らすことによる、コストの削減
- 店舗および企業間での情報共有

アーキテクチャの改良

次に、Web サービスがもたらすアーキテクチャの改良について説明します。

サービス インターフェイス

ソフトウェア サービスをインターネット経由で提供するというのは、新しい考え方ではありません。過去 10 年間にわたり、多くの企業が、ウイルス スキャナ アップデートやウイルス定義ファイルなど、情報を特定のクライアントにインターネット経由で提供するソフトウェア システムを生み出してきました。では、なぜ新しい Web サービスはソフトウェア業界でそれほど重要なのでしょうか。Web サービスには、他にどのような新しいテクノロジやプロトコルが含まれているのでしょうか。

これらの疑問に対する答えは、標準化に関連しています。XML や SOAP などの新しい標準を使用すると、複数のプラットフォームのための基盤を設定し、それらのプラットフォーム間で通信することが可能になります。今日、ソフトウェアコンポーネントは、構造化された言語 (XML) と SOAP を使用することで、他のコンポーネントと通信することができます。

以下の標準は、今日 Web サービスで使用されている最も一般的な標準です。

- **新しい標準**。これには以下のものが含まれます。
  - 業界データ交換のための XML アグリーメント。
  - セキュリティ。これには認証が含まれます (Microsoft WS-Security はその一例です)。
  - トランザクション。これには一連の相互に依存したアクションの管理が含まれます (XAML と OASIS BTP はその一例です)。
  - プロセス。これには複雑なビジネス対話の記述が含まれます (マイクロソフトの WS-Routing はその一例です)。
- **既存の標準**。これには以下のものが含まれます。
  - SOAP。このプロトコルは Web サービスを要求します。
  - Web サービス記述言語 (WSDL)。Web サービスの処理と使い方を文書化するための標準です。
  - Universal Description, Discovery, and Integration (UDDI)。自動検索用に WSDL ドキュメントをディレクトリ内に一覧するための標準です。
- **確立済みの標準**。これには以下のものが含まれます。
  - 伝送制御プロトコル/インターネット プロトコル (TCP/IP)。インターネットのネットワーク プロトコルです。
  - ハイパーテキスト転送プロトコル (HTTP)。このプロトコルは、サーバーとクライアントの間でデータをハイパーテキストまたはハイパーメディアで転送するために使用します。
  - ファイル転送プロトコル (FTP)。この方式では、2 つのインターネット サイト間でファイルを移動します。
  - 簡易メール転送プロトコル (SMTP)。このプロトコルは、コンピュータ間で電子メール メッセージを送信するために使用します。
  - 統一リソース識別子 (URI)。この方式は、テキストのページ、ビデオ クリップやサウンド クリップ、静止画や動画、プログラムのいずれであるかに関係なく、Web 上のコンテンツの任意の位置を識別するために使用します。
  - データ交換のための XML 標準。

Web サービスの探索

前に示した標準に加えて、マイクロソフトは .NET での Web サービスの構築で重要な役目を果たす、2 つの新しいコンポーネントを導入します。それが .asmx ファイルと .disco ファイルです。

.asmx ファイルは、マネージ コードで作成された Web サービスのアドレス指定可能なエントリ ポイントを表します。このファイルに HTTP 経由でアクセスする方法により、受信する応答の種類が決まります。たとえば、Web サービスの URL に「?WSDL」を加えると、Web サービスにより適切な WSDL ファイルが直ちに作成されて返されます。以下のコード例は、Web サービスによって公開されるメソッドへの参照を持つ HTML ページです。

```
<?xml version="1.0" encoding="utf-8" ?>
  <head>
    <link rel="alternate" type="text/xml" href="Service1.asmx?disco"/>
    <title>Service1 Web Service</title>
  </head>
  <body>
    <div id="content">
      <p class="heading1">Service1</p><br>
      <span>
          <p class="intro">The following operations are supported.
          For a formal definition, please review the
          <a href="Service1.asmx?WSDL">Service Description</a>.</p>
              <ul>
              <li>
                <a href="Service1.asmx?op=HelloWorld">HelloWorld</a>
              </li>
              <p>
              </ul>
      </span>
  </body>
```

Web サービスの探索は、Web サービスの記述を見つけて問い合わせるプロセスです。このプロセスを、Web サービスにアクセスする前の予備的なステップとして実行します。Web サービス クライアントは、この探索プロセスで Web サービスの存在、Web サービスの機能、および Web サービスと正しく対話する方法を学習します。探索ファイルは、.disco というファイル名拡張子を持つ XML ドキュメントです。Web サービスごとに探索ファイルを作成する必要はありません。以下のコードは、前の例のセキュリティ Web サービスのサンプル探索ファイルです。

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.xmlsoap.org/disco/">
  <contractRef ref="http://tempuri.org/WebService2/Service1.asmx?wsdl"
docRef="http://tempuri.org/WebService2/Service1.asmx" xmlns="http://
schemas.xmlsoap.org/disco/scl/" />
  <soap address="http://tempuri.org/WebService2/Service1.asmx" xmlns:q1="http://
tempuri.org/WebService2/Service1" binding="q1:Service1Soap" xmlns="http://
schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

このファイルに Serivce1.disco という名前を付け、Web サービスと同じディレクトリに保存することができます。Web サービスの登録、探索、および Web サービスへのアクセスの詳細については、この後の「Web サービスの使用」を参照してください。

Web サービスを /webreference/machinename ディレクトリで作成する場合は、動的探索を有効にするのが最善の方法です。動的探索では、/webreference/machinename のすべてのサブディレクトリで、すべての *.disco ファイルが自動的にスキャンされます。

```
<?xml version="1.0" ?>
    <dynamicDiscovery xmlns="urn:schemas-dynamicdiscovery:disco.2000-03-17">
</dynamicDiscovery>
```

探索ファイルを分析することにより、Web サービスがシステム上のどこに存在するのかを知ることができます。残念ながら、.disco ファイル方式と動的探索方式では、いずれも Web サービスの正確な URL を事前に把握している必要があります。探索ファイルが見つからない場合、Web サービスを検索することはできません。この問題を解決する方法の 1 つは、Web サービスを発行し、潜在的なユーザーが検索できるようにすることです。Universal Description, Discovery, and Integration (UDDI) は、既存の Web サービスを発行するために使用できるメカニズムについて記述する、新しい仕様です。UDDI は、インターネット ベースのオープンな仕様です。この仕様は、企業が好みのアプリケーションを使用して、相手企業をすばやく、簡単に、かつ動的に検索して対話できるようにするための基盤として使用できます。

Web サービスの作成

Web サービスの作成手順は簡単です。最も重要なのは、何を発行するのかをきちんと理解しておくことです。

Visual Studio .NET で Web サービスを作成したとき、コードは以下のように、特別な属性を持つ単純なクラスになります。

```
Imports System.Web.Services

<System.Web.Services.WebService(Namespace := "http://tempuri.org/WebService2/Service1")> _
Public Class Service1
    Inherits System.Web.Services.WebService
    <WebMethod()> _
    Public Function HelloWorld() As String
        Return "Hello World"
    End Function

    <WebMethod()> _
    Public Function HelloMe() As String
        Return "Hello Me"
    End Function

End Class
```

Web サービスを作成するには、まず Visual Studio .NET または好みのテキスト エディタを使用し、.asmx ファイルを作成します。前のコード例で示されているように、Web サービスは通常のクラスとして作成されます。このクラスは、System.Web.Services.WebServices クラスを継承します。<WebMethod()> 属性が前にあるメソッドは、そのメソッドを Web サービス経由でアクセスできるようにすることを示します。

前のコード例では、以下の部分に注目してください。

- Web サービス経由でアクセスできる 2 つのメソッドには、<WebMethod()> 属性が前宣言されています。
- System.Web.Services.WebService 属性は、Web サービスが発行される名前空間を指定します。この例では、http://tempuri.org/WebService2/Service1 という URL が、Web サービスの実際の URL と置

き換えられるプレースホルダです。

- この Web サービスの名前は、クラス名で示されている Service1 になります。

.asmx ファイルを作成し、Web からアクセス可能なディレクトリに格納したら、Web ブラウザを使用してページに直接移動することにより、メソッドを表示することができます。

これで Web サービスの作成者としての仕事は終わりです。Web サービスを作成し、発行する準備が整いました。Web サービスを発行する前に、この Web サービスで発行されるメソッドが、複雑なオブジェクトやシリアル化されないオブジェクトを受け取ることはできないということに注意してください。こうしたオブジェクトを受け取るメソッドがコードに含まれている場合は、メソッドの実装を変更するか、またはオブジェクトをシリアル化可能にして、Web サービスからアクセスできるようにする必要があります。

次に、Web サービスの使用方法について説明します。

Web サービスの使用

Web サイトで Web サービスを使用するには、クライアントの Web サイトと Web サービスを提供する Web サイトの間で、比較的複雑で短い通信を実行する必要があります。

Web サイトで Web サービスを使用できるようにするには、まずアプリケーションに Web 参照を追加する必要があります。そのための最善の方法は、Visual Studio .NET IDE を使用することです。以下の手順例では、この章の「Web サービスの作成」で作成した Web サービスへの参照を追加します。

▶ アプリケーションへの Web 参照の追加

1. クライアント アプリケーションを開きます。
2. [**プロジェクト**] メニューで、[Web **参照の追加**] をクリックします。
3. [Web **参照の追加**] ダイアログ ボックスで、[Web services on the local machine] をクリックします。
4. 利用可能な Web サービスの一覧で、[Service1] をクリックします。
5. [Web **サービス名**] というラベルが付いたテキスト ボックスに、この新しい参照の名前を入力します。この例では、「HelloWorldWS」と入力します。この名前は、Visual Basic .NET コードから Web サービスを参照するために使用します。
6. [**参照の追加**] をクリックします。

Web 参照の追加に関する詳細については、MSDN の「Publishing and Discovering Web Services with DISCO and UDDI」を参照してください。

アプリケーションに Web 参照を追加したら、そのアプリケーションから Web サービスを使用することができます。使用側アプリケーションでは、作成側のどのメソッドを呼び出すかを判断する必要があります。入力パラメータがある場合は、この章の「Web サービスの作成」で説明したとおり、それらのパラメータを XML またはその他のシリアル化可能な形式に変換し、渡せるようにする必要があります。使用側から作成側に対して

HTTP 要求が実行され、呼び出すメソッドが指定され、SOAP 要求、QueryString、または POST ヘッダーによってパラメータが渡されます。

作成側は送られてきた要求を受信し、入力パラメータをアンパッケージ化して、指定されたクラスの適切なメソッドを呼び出します。このメソッドは、完了すると値を返します。その値がパッケージ化され、使用側に HTTP 応答で返されます。使用側はこの応答を受信し、戻り値をアンパッケージ化して、Web サービスの呼び出しを完了します。

Web サービスを使用しているときは、HTTP のメッセージ交換の形式を考慮するのは避けたいところです。この問題を解決するには、プロキシ クラスを使用します。プロキシ クラスは、使用側のプログラム (または Web ページ) と作成側の実際の Web サービスの間の中間的なステップとして機能します。作成側の Web サービスのメソッドごとに、プロキシ クラス内にメソッドが存在します。プロキシ クラスの役目は、メッセージを渡すための複雑なタスクをすべて行うことです。これにより、複雑な部分がクラス内に実質的に隠されます。Web ページまたは Windows アプリケーションでは、このプロキシ クラスのメソッドを単純に呼び出すだけでよく、Web サービスの呼び出しに関連する基盤の形式に配慮する必要はなくなります。Web サービスの作成と使用の詳細については、MSDN の「Consuming Web Services」を参照してください。

プロキシを自分で作成する必要はありません。実際には、Visual Studio によってこの作業のすべてが自動的に行われます。[参照の追加] ボタンをクリックすると、Visual Studio により、HelloWorldWS.Service1 という名前のプロキシが作成されます。この新たに参照された Web サービスで処理を実行するには、プロキシ クラスのインスタンスを作成し、通常のクラスのときと同じように、プロキシ クラスのメソッドを呼び出すだけです。以下のコードでは、この例の Web サービスの HelloWorld() メソッドを呼び出し、Windows フォームの Label1 という名前のラベルに結果を表示します。

```
Private Sub Button1_Click(ByVal sender As System.Object,_
        ByVal e As System.EventArgs) Handles Button2.Click
    Dim ws As HelloWorldWS.Service1
    ws = New HelloWorldWS.Service1
    Label1.Text = ws.HelloWorld()
End Sub
```

新しいテクノロジ

次に、Web サービスを通じて利用できる新しいテクノロジについて詳しく説明します。

Web Services Enhancements (WSE)

Web Services Enhancements (WSE) は、.NET Framework と関連テクノロジを補い、さまざまな Web サービス仕様の実装を提供する .NET クラス ライブラリです。

WSE は、Visual Studio .NET や.NET Framework を使用する開発者がセキュリティ ポリシーを適用したり、長時間実行される安全な会話を確立したり、セキュリティトークンを取得および確認したりするのを支援することにより、安全な Web サービスの開発と配置を簡素化するように設計されています。本稿の執筆時点で、WSE の最新バージョンは 2.0 です。このバージョンでは、前述の機能をより簡単に適用するために、WSE

の以前のバージョンから多くの点が強化されています。新たに承認された WS-Security の OASIS 標準バージョンに対する WSE 2.0 のサポートなど、これらの強化点のいくつかは、WSE 2.0 が安全な SOAP メッセージを WSE 1.0 と交換できないことを意味しています。また WSE 2.0 では、WSE 1.0 で使用していた WS-Routing にとって代わる WS-Addressing を使用します。ただし、WSE 2.0 アセンブリを WSE 1.0 アセンブリと同じアプリケーションで使用し、単一のアプリケーションで WSE 1.0 と WSE 2.0 の両方のWeb サービスと通信することが可能です。

Web Services Security

WS-Security は、それ自体が完全なセキュリティ ソリューションというわけではありません。WS-Security は、メッセージの送信側と受信側の間でセキュリティ情報を交換するためのプロトコルです。開発者は、適切なセキュリティソリューションを設計し、再生攻撃などの潜在的な驚異に対処する必要があります。

Web Services Enhancements (WSE) は、以下の資格情報に基づいたセキュリティトークンをサポートします。

- Kerberos バージョン 5 チケット
- ユーザー名とパスワード
- X.509 証明書

添付ファイル

WSE 2.0 は、SOAP エンベロープ外での SOAP メッセージへのファイルの添付をサポートします。これらのファイルは XML としてシリアル化されません。これは、大きなテキスト ファイルまたはバイナリ ファイルを送信するときに便利です。なぜなら、XML シリアル化はきわめて高コストな処理であり、元のファイルよりもはるかに大きいファイルが生成される可能性があるからです。Direct Internet Message Encapsulation (DIME) は、WSE 2.0 が SOAP メッセージとその添付ファイルを単一のメッセージにカプセル化するために使用する、軽量なバイナリ メッセージ形式です。

WS-Routing

WS-Routing は、サービス ネットワークの参加者が、送信元から宛先にメッセージを移動するために使用する仕様です。WS-Routing は SOAP ルータの心臓部です。SOAP ルータはしばしば中間物として機能し、SOAP ルータ間で、また他のサービス ネットワーク参加者に対して、情報を渡します。

多くの場合、ルータ以外の参加者 (SOAP サーバーなど) も WS-Routing 仕様を実装します。これにより、SOAP サービスの出力を別のサービスへ入力として直接送ることが可能になります。

WS-Routing では、ルーティング テーブルを使用して、メッセージがたどるパスを判断します。これらのルーティング テーブルを更新するために、WS-Referral という別のプロトコルを使用します。

WS-Referral は、SOAP ルータでルートを追加または削除するために使用できるメッセージの形式を記述します。

まとめ

Web アプリケーションは、Web を介してデータを交換し、サービスをリモートで呼び出すための手段を提供します。Web アプリケーションを使用すると、データおよびサービスへのアクセスを、通常のデスクトップ アプリケーションよりもはるかに多くのユーザーに提供することができます。この可能性が大きな意味を持つことが、Web アプリケーションと Web サービスが増えるきっかけとなっています。

アプリケーションが現在 Web アクセスに対応していない場合、それらのアプリケーションを Web サービスで強化したり、Web サービスに変えたりするのは、決して不可能ではありません。この章では、アプリケーションを Web 上で使用できるようにするために Visual Basic .NET で利用できる Web テクノロジや言語機能について説明してきました。

アプリケーションの進歩についてのもう 1 つの視点は、あらゆるアプリケーションにおいて好ましい技術的な機能を考慮することです。次の章では、アプリケーションの安全性、管理性、および拡張性を高めるために役立つ改良点を見ていきます。

詳細情報

マスタ ページの詳細については、MSDN の「ASP.NET Master Pages Overview」を参照してください。
URL は http://msdn2.microsoft.com/library/wtxbf3hh.aspx です。

HTTP モジュールの詳細については、MSDN Magazine の「HTTP Modules」を参照してください。
URL は http://msdn.microsoft.com/msdnmag/issues/02/05/asp/default.aspx です。

公に利用可能な Web サービスの例については、Microsoft.com の「Microsoft MapPoint Web Service」を参照してください。
URL は http://www.microsoft.com/mappoint/products/webservice/default.mspx です。

WebService クラスの詳細については、MSDN の『.NET Framework Class Library』の「WebService Class」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemWebServicesWebServiceClassTopic.asp です。

Web 参照の追加に関する詳細については、MSDN Magazine の「Publishing and Discovering Web Services with DISCO and UDDI」(http://msdn.microsoft.com/msdnmag/issues/02/02/xml/default.aspx) を参照してください。

Web サービスの作成と使用の詳細については、MSDN の「Consuming Web Services」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdk/htm/ebiz_prog_webservices_yydh.asp です。

20

一般的なテクノロジ シナリオ

多くのテクノロジ シナリオは、アプリケーションの目的に関係なくすべてのアプリケーションに当てはまります。これらのシナリオには、セキュリティ、管理性、およびパフォーマンスの要件が含まれます。アプリケーションで過失や悪意による操作からシステムを保護できるかどうかは、適切なセキュリティが提供されているかどうかにかかっています。アプリケーションの管理は、ユーザーのコンピュータにアプリケーションをインストールするだけで終わるものではありません。コンピュータのハードウェアのパフォーマンスが向上し続けるにつれて、そこで実行されるアプリケーションへの要求も増加します。つまり、今日のアプリケーションにとって、パフォーマンスおよび拡張性は重要な考慮事項になります。

この章では、これらのシナリオのそれぞれについて検討し、Microsoft Visual Basic .NET でどのような対処がなされているのかを説明します。

アプリケーション セキュリティ

Microsoft .NET Framework のセキュリティ機能では、デザインの一環として、開発者がセキュリティで保護されたコードをより簡単に記述できるようになっています。ここでは、アプリケーションのセキュリティを強化するために使用できる Visual Basic .NET の機能をいくつか紹介します。

ID および認証の使用

セキュリティは、.NET アプリケーションを構築する際に考慮する必要がある最も重要な機能の 1 つです。アプリケーション開発者は、アプリケーションが実稼働環境で使用される際に発生する可能性があるさまざまな状況を考慮に入れる必要があります。セキュリティ攻撃がごく当たり前に見られる現在、組織内で実行されるアプリケーションのセキュリティの確保は誰もが望むところとなっています。.NET Framework には、セキュリティの状況を管理するためのクラスが用意されています。これにより、アプリケーションにロールベースのセキュリティを簡単に実装できます。

セキュリティの実装と実施には、認証と承認という 2 つの部分があります。これらは、IPrincipal インターフェイスと IIdentity インターフェイスに基づくカスタム プリンシパル クラスとカスタム ID クラスを実装することによって実現されます。

ID はユーザーを表し、ユーザーに関する情報 (ユーザー名など) をプロパティに保持します。ID を扱うクラスは、System.Security.Principal 名前空間にあります。この名前空間には、GenericIdentity と WindowsIdentity という 2 つのクラスが含まれています。これらのクラスを使用して、ユーザーのプロパティを特定できます。この名前空間には、これ以外に、カスタム ID を作成する際に使用できる IIdentity インターフェイスも含まれています。カスタム ID では、ID の基本型を個々のアプリケーションのニーズに合わせて拡張できます。

IIdentity インターフェイスの実装の例を以下に示します。

```
Import System.Threading
Import System.Security.Permissions
Public Class MyIdentity
    Implements System.Security.Principal.IIdentity

    Public Shared strAuthenticationTypeString As String = "NTLM"
    Public strUserName As String
    Private _userInfo As Hashtable

    Public ReadOnly Property AuthenticationType() As String _
           Implements System.Security.Principal.IIdentity.AuthenticationType
        Get
            Return strAuthenticationTypeString
        End Get
    End Property

    Public ReadOnly Property IsAuthenticated() As Boolean _
           Implements System.Security.Principal.IIdentity.IsAuthenticated
        Get
            Return True
        End Get
    End Property

    Public ReadOnly Property Name() As String _
           Implements System.Security.Principal.IIdentity.Name
        Get
            Return _userInfo(strUserName)
        End Get
    End Property

    Private Sub New(ByVal userInfo As Hashtable)
        Me._userInfo = userInfo
    End Sub

    Public Shared Function strCreateMyIdentity(ByVal userInfo As Hashtable) _
                  As MyIdentity
        Return New MyIdentity(userInfo)
    End Function

End Class
```

UserIdentity 型は IIdentity インターフェイスを実装します。このインターフェイスでは、以下の 3 つのプロパティを実装する必要があります。

- Name。ID の名前を返します。共有関数の strCreateMyIdentity() を呼び出して、すべてのユーザー情報を含むハッシュテーブルを渡す必要があります。これにより、ID のインスタンスが返されます。
- IsAuthenticated。ユーザーが認証されているかどうかを表す値を返します。匿名アクセスを許可している場合は、匿名ユーザーに対してこのプロパティを false に設定します。
- AuthenticationType。認証の種類を返します。WindowsIdentity は NTLM を返し、GenericIdentity は空の文字列か、GenericIdentity をインスタンス化した際に指定した種類を返します。

Principal オブジェクトは、ユーザーが (アクティブな認証メカニズムによって) 属するすべてのロールを保持します。IPrincipal インターフェイスを実装する .NET クラスはすべて有効な Principal オブジェクトになります。IPrincipal インターフェイスは、Identity プロパティ (基になる Identity オブジェクトを返します) と IsInRole メソッドを公開します。

以下のコード例は、IPrincipal インターフェイスを実装する方法を示しています。このクラスは、上で定義した IIdentity クラスを使用しています。

```
Import System.Threading
Import System.Security.Permissions
Public Class MyPrincipal
    Implements System.Security.Principal.IPrincipal

    Private Groups As Hashtable
    Private Rights As Hashtable
    Private MyIdentity1 As MyIdentity

    Public ReadOnly Property Identity() As System.Security.Principal.IIdentity _
                   Implements System.Security.Principal.IPrincipal.Identity
        Get
            Return MyIdentity1
        End Get
    End Property

    Public Function IsInRole(ByVal role As String) As Boolean _
           Implements System.Security.Principal.IPrincipal.IsInRole
        Return Groups.ContainsValue(role)
    End Function

    Public Function HasPermissions(ByVal Permission As String) As Boolean
        Return Rights.ContainsValue(Permission)
    End Function

    Private Sub New(ByVal SecGroups As Hashtable, ByVal SecRights As Hashtable, _
                ByVal userInfo As Hashtable)
        Me.Groups = SecGroups
        Me.Rights = SecRights
        MyIdentity1 = MyIdentity.strCreateMyIdentity(userInfo)
    End Sub
```

```
    Public Shared Function CreateSecPrincipal(ByVal SecGroups As Hashtable, _
                ByVal SecRights As Hashtable, ByVal userInfo As Hashtable) _
                As MyPrincipal
        Return New MyPrincipal(SecGroups, SecRights, userInfo)
    End Function

    Protected Overrides Sub Finalize()
        MyBase.Finalize()
    End Sub
End Class
```

Security.Principal 型は IPrincipal インターフェイスを実装します。このインターフェイスでは、Identity プロパティと IsInRole() メソッドを実装する必要があります。また、Security.Principal 型にはプライベート コンストラクタがあり、この型のインスタンスを作成できないようになっています。共有関数 CreateSecPrincipal() を呼び出して、ユーザー情報とユーザーが属するロールを含むハッシュ テーブルと、現在の実装におけるユーザーのセキュリティ権限 (アクセス許可) を渡す必要があります。

この型のコンストラクタは、カスタム ID クラスから CreateUserIdentity() を呼び出してユーザー情報を渡し、受け取った ID を保持します。CreateSecPrincipal() は、セキュリティ プリンシパルのインスタンスを返します。Identity プロパティは、セキュリティプリンシパルに関連付けられている ID オブジェクトを返します。

これらのクラスの定義が完了したら、次のコード例のようにそれらを使用できます。

```
Import System.Threading
Import System.Security.Permissions
Public Class Example
    Public Shared Sub Main()

        Dim MyID As New System.Security.Principal.GenericIdentity("hvalverde")
        Dim roles As String() = {"Manager", "Administrator"}
        Dim GP As New System.Security.Principal.GenericPrincipal(MyID, roles)


        Thread.CurrentPrincipal = GP
        Try
            doSomething()
        Catch e As Exception
            Console.WriteLine("Error:")
            Console.WriteLine(e.ToString())
        End Try
        GP = New System.Security.Principal.GenericPrincipal(MyID, _
            New String() {"Administrator"})
        Thread.CurrentPrincipal = GP
        doSomething()
    End Sub
```

```
    <PrincipalPermissionAttribute(SecurityAction.Demand, Role:="Administrator")> _
    Shared Sub doSomething()
        Console.WriteLine("You are Adm")
    End Sub
End Class
```

.NET Framework クラス ライブラリには、WindowsPrincipal と GenericPrincipal という 2 つの Principal オブジェクトが含まれています。WindowsIdentity と WindowsPrincipal の組み合わせを使用する場合、Principal のロール リストは、その Windows ユーザーが属する Windows グループを使用して作成されます。

プリンシパル オブジェクトと ID オブジェクトを使用して、ユーザーを認証し、各ユーザーのロールに応じてアプリケーションの機能へのアクセスを承認 (または拒否) することができます。

暗号化の使用

.NET Framework には、よく知られているアルゴリズムとその一般的な使用方法 (ハッシュ、暗号化、デジタル署名の生成など) をサポートする一連の暗号オブジェクトが用意されています。これらのオブジェクトは、こうした基本的な機能をより複雑な操作 (ドキュメントの署名や暗号化など) に簡単に組み込めるようにデザインされています。

暗号オブジェクトは、.NET Framework によって内部のサービスをサポートするために使用されますが、暗号のサポートを必要とする開発者も使用できます。.NET Framework には、そうした標準の暗号アルゴリズムや暗号オブジェクトの多くの実装が用意されています。

.NET Framework の System.Security.Cryptography 名前空間には、さまざまな暗号サービスが用意されています。たとえば次のような暗号アルゴリズムがサポートされています。

- **Rivest Shamir Adelman (RSA) およびデジタル署名アルゴリズム (DSA) の公開キー (非対称) 暗号化。**非対称アルゴリズムは固定バッファを処理します。暗号化と復号化には公開キー アルゴリズムが使用されます。
- Data Encryption Standard (DES)、TripleDES、および RC2 の秘密キー (対称) 暗号化。対称アルゴリズムは可変長バッファを変更するために使用され、定期的なデータ入力に対して 1 つの操作を行います。
- **Message Digest 5 (MD5) と Secure Hash Algorithm (SHA1) のハッシュ。**MD5 は一方向のハッシュ アルゴリズムです。可変長の入力データに対して常に 128 ビットのハッシュ値が生成されます。

.NET Framework には、さまざまな状況で使用できる多彩な暗号化アルゴリズムが用意されています。対称アルゴリズムは、ファイルや通信チャネルなどのデータのストリームの暗号化に使用でき、比較的パフォーマンスに優れています。非対称暗号化は、最も強力な保護を実現できる反面、一般に対称暗号化より低速です。

以下の例は、ファイルの暗号化と復号化のためのメソッドを含む Cipher クラスを示しています。この例では、DES 暗号化アルゴリズムを使用しています。このアルゴリズムは、暗号化と復号化に 1 つのキーを使用するため、対称暗号化アルゴリズムのカテゴリに属します (DES 暗号化アルゴリズムの詳細については、Ius Mentis の Web サイトの「The DES Encyption Algorithm」を参照してください)。Cipher クラスには、アルゴリズムで使用するキーを指定する SetKey というメソッドがあります。このクラスを使用する場合は、元のキーを安全な秘密の場所に保管する必要があります。ファイルの復号化や暗号化にはこのキーが必要になります。この例からもわかるように、Decrypt メソッドは、cipherObj.CreateEncryptor の代わりに cipherObj.CreateDecryptor を呼び出す以外は Encrypt メソッドと同じです。chainVec 変数は、暗号化または復号化の処理方法を指定するための追加のパラメータとして使用されており、暗号化された前のブロックの情報の一部を現在のブロックに挿入することによってさらなる保護を提供します。この機能は、*チェーン暗号*と呼ばれています。

```
Imports System.IO
Imports System.Text
Imports System.Security.Cryptography
Public Class Cipher
    Public encryptKey(7) As Byte
    Private chainVec() As Byte = {&HAB, &HCD, &HEF, &H12, &H34, &H56, &H78, &H90}

    Sub SetKey(ByVal keyBase As String)
        Dim encoder As New ASCIIEncoding
        Dim arrByte(7) As Byte
        Dim i As Integer = 0
        Dim res() As Byte
        encoder.GetBytes(keyBase, i, keyBase.Length, arrByte, i)

        Dim SHAHash As New SHA1CryptoServiceProvider
        res = SHAHash.ComputeHash(arrByte)

        '暗号化キーを更新します
        For i = 0 To 7
            encryptKey(i) = res(i)
        Next i
    End Sub

    Sub Encrypt(ByVal inputFile As String, ByVal outputFile As String)
        Try
            Dim dataBuffer(4096) As Byte
            Dim totFileLen As Long
            Dim BytesWrittenTot As Long = 8
            Dim blockSize As Integer
            Dim fileIn As New FileStream(inputFile, _
            FileMode.Open, FileAccess.Read)
            Dim fileOut As New FileStream(outputFile, _
            FileMode.OpenOrCreate, FileAccess.Write)
            Dim cipherObj As New DESCryptoServiceProvider
            Dim encryptedStream As New CryptoStream(fileOut, _
                cipherObj.CreateEncryptor(encryptKey, _
                    chainVec), CryptoStreamMode.Write)
```

```
            fileOut.SetLength(0)
            totFileLen = fileIn.Length

            While BytesWrittenTot < totFileLen
                blockSize = fileIn.Read(dataBuffer, 0, 4096)
                encryptedStream.Write(dataBuffer, 0, blockSize)
                BytesWrittenTot = Convert.ToInt32 _
                (BytesWrittenTot + blockSize / _
                cipherObj.BlockSize * cipherObj.BlockSize)
            End While

            encryptedStream.Close()
        Catch e As Exception
            '例外を処理します
        End Try
    End Sub

    Sub Decrypt(ByVal inputFile As String, ByVal outputFile As String)
        Try
            Dim dataBuffer(4096) As Byte
            Dim totFileLen As Long
            Dim BytesWrittenTot As Long = 8
            Dim blockSize As Integer
            Dim fileIn As New FileStream(inputFile, _
            FileMode.Open, FileAccess.Read)
            Dim fileOut As New FileStream(outputFile, _
            FileMode.OpenOrCreate, FileAccess.Write)
            Dim cipherObj As New DESCryptoServiceProvider
            Dim encryptedStream As New CryptoStream(fileOut, _
            cipherObj.CreateDecryptor(encryptKey, _
            chainVec), CryptoStreamMode.Write)

            fileOut.SetLength(0)
            totFileLen = fileIn.Length

            While BytesWrittenTot < totFileLen
                blockSize = fileIn.Read(dataBuffer, 0, 4096)
                encryptedStream.Write(dataBuffer, 0, blockSize)
                BytesWrittenTot = Convert.ToInt32( _
                BytesWrittenTot + blockSize / _
                cipherObj.BlockSize * cipherObj.BlockSize)
            End While

            encryptedStream.Close()
        Catch e As Exception
            '例外を処理します
        End Try
    End Sub
End Class
```

アプリケーションに暗号化を組み込む場合は、Microsoft パターンとプラクティスの Security Application Block を使用することを検討してみてください。このアプリケーション ブロックでは、システムの要件に基づいてセキュリティオプションを選択できます。Security Application Block は、データベースを使用してユーザーの認証と承

認を行う場合、ロールとプロファイルの情報を取得する場合、ユーザー プロファイルの情報をキャッシュする場合など、さまざまな状況で使用できます。Security Application Block の特徴を以下に示します。

- 標準のタスクを実行するために記述しなければならない定型コードを減らすことができます。
- アプリケーション内および企業全体の両方で、一貫したセキュリティ プラクティスを維持するのに役立ちます。
- 提供されるさまざまな機能領域で一貫したアーキテクチャ モデルが使用されているため、開発者が容易に学習できます。
- アプリケーションの一般的なセキュリティの問題の解決に利用できる実装が用意されています。
- 拡張可能です (セキュリティ プロバイダのカスタム実装をサポートしています)。

セキュリティと暗号化の詳細については、MSDN の「Security Application Block」、「Cryptography Application Block」、および「Cryptography Simplified in Microsoft .NET」を参照してください。

アプリケーションの管理性

.NET Framework では、アプリケーションの管理性を向上させる機能強化がなされています。以降では、これらの機能強化について説明します。

構成ファイルの使用

構成ファイルを使用して .NET アプリケーションを構成できます。アプリケーションを再コンパイルする必要はありません。これらの構成ファイルでは、以前のバージョンの Visual Basic で .ini ファイルや、後のレジストリ設定を使用して行ったのと同様に、アプリケーションの変数や構成値を格納できます。さらに、Visual Basic .NET の構成ファイルでは、ビジュアル コンポーネントのプロパティを保持することもできるため、テキスト ファイルを編集するだけでこれらのコンポーネントを動的に変更できます。

構成ファイルに含まれる情報は、それを使用するアプリケーションの種類によって異なります。.NET クライアント アプリケーションの構成ファイルは .NET サーバー アプリケーションのものとは 含まれるセクション、属性、および値が大きく異なります。

一般に、リモート サーバー アプリケーションと通信するクライアント アプリケーションを構成する際には、以下の作業を行う必要があります。

- **クライアント アプリケーションのセキュリティ ポリシーの構成。** 既定では、イントラネットで実行されるアプリケーションはイントラネット内の他のサーバーにアクセスすることはできません。必要なアクセス許可を設定して、クライアントが他のサーバーにアクセスできるようにする必要があります。

- **アセンブリ バインドの構成**。アプリケーションでサードパーティの共有コンポーネントを使用している場合は、クライアント アプリケーションで使用するコンポーネントのバージョンを指定する必要がある場合があります。
- **リモート処理の構成**。クライアント アプリケーションにサーバーのアドレスを指定する必要があります。この作業は、.NET Framework Configuration Tool を使用して行うことができます。このツールの使用方法の詳細については、MSDN の「.NET Framework Configuration Tool (Mscorcfg.msc)」を参照してください。

構成設定の適用が完了したら、設定後の構成ファイルを、アプリケーションを使用するコンピュータに配置する必要があります。この例での配置先はクライアント コンピュータです。以下の XML サンプルは、ここで説明した構成設定を指定しています。

```
<configuration>
    <runtime>
        <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
            <dependentAssembly>
                <assemblyIdentity name="MySpecificVersionComponent"
                                  publicKeyToken="e9b4c4996039ede8"
                                  culture="en-us"/>
                <publisherPolicy apply="no"/>
            </dependentAssembly>
        </assemblyBinding>
    </runtime>

   <system.runtime.remoting>
      <application name = "myApplication">
         <client url = "tcp://server/clientApplication"
                 displayName="Client Application">
            <activated type = "ClientClassInstance,ClientClass"/>
         </client>
      </application>
   </system.runtime.remoting>
</configuration>
```

dependentAssembly ノードでは、クライアント アプリケーションが依存しているコンポーネントが指定されています。

サーバー アプリケーションの構成ファイルには、リモート クライアントに公開するオブジェクト、クライアント側オブジェクトの有効期限、および解放されるオブジェクトのポーリング間隔の設定が含まれるのが一般的です。

構成 (.config) ファイルは、定義された構造を持つプレーンな XML ファイルです。通常は、アプリケーションの実行可能 (.exe) ファイルと同じフォルダにあります。Visual Studio .NET からアプリケーションに構成ファイルを追加するには、**[ファイル]** メニューの **[新しい項目の追加]** をポイントし、**[アプリケーション構成ファイル]** をクリックします。

構成ファイルには、構成情報を設定する論理データ構造である XML 要素が含まれています。たとえば、<runtime> 要素は、<runtime>*子要素*</runtime> という形式で指定されます。構成設定は、あらかじめ定義された属性を使用して指定します。これらの属性は、対応する要素の中の名前/値のペアです。

次の例では、<codeBase> 要素の version 属性と href 属性を指定しています。この要素は、アセンブリの場所を示します。構成ファイルの内容では大文字と小文字が区別されるので注意してください。

```
<codeBase version="2.0.0.0"
          href="http://www.litwareinc.com/myAssembly.dll"/>
```

構成ファイルの <appSettings> セクションに新しいエントリを追加して、以下の内容を定義することができます。

- **ユーザー定義の設定**。アプリケーションが使用する構成設定です。アプリケーションはこれらの設定を System.Configuration.AppSettingsReader クラスから読み取ります。
- **動的プロパティ**。構成ファイルのエントリから値を割り当てられる Windows フォーム コントロールのプロパティです。これらのプロパティの値は、.config ファイルを修正することによって変更できます。変更を適用するのにアプリケーションを再コンパイルする必要はありません。

次の例は、アプリケーションの構成ファイルに含まれているデータを AppSettingsReader を使用して読み取る方法を示しています。

```
…
Dim myReader As System.Configuration.AppSettingsReader()
Dim myInstance As New MyClass()
myInstance.StringProp = CType(myReader.GetValue("DynamicClass.StringProp", _
GetType(System.String)), String)
…
```

AppSettingsReader.GetValue メソッドは、構成ファイルのエントリのキーと、データの解析に使用する型を受け取っています。次の例は、この例の構成ファイルの内容を示しています。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <appSettings>
      <add key="DynamicClass.StringProp" value="This is a test">
   </appSettings>
</configuration>
```

一般に、構成ファイルのデータを手動で取得するには、以下の手順を実行する必要があります。

1. System.Configuration.AppSettingsReader のインスタンスを作成します。
2. AppSettingsReader.GetValue メソッドを使用して、指定したキーの値を取得します。
3. GetValue メソッドによって返されるオブジェクトを適切なデータ型に変換します。

コンピュータレベルのアプリケーション構成ファイルとセキュリティの設定の詳細については、MSDN の「Application Configuration Files」、「Application Configuration Scenarios」、「System.Configuration Namespace」、および「Configuration Application Block」を参照してください。

配置と更新の機能の使用

アプリケーションを Visual Basic .NET にアップグレードすると、.NET Framework に用意されている配置プロセスの機能を利用できるようになります。.NET Framework では、次に示すようなさまざまな方法でアプリケーションを配置できます。

- **XCOPY 配置**。.NET Framework 上に構築されたアプリケーションを配置するための最も簡単な方法です。XCOPY を使用するには、アセンブリをインストール パスにコピーします。アプリケーションでこれ以外にインストールする必要がある追加ファイルがない場合や、クライアント コンピュータでカスタマイズを行う必要がない場合に適しています。
- **セットアップ プロジェクト**。.NET Framework には、XCOPY 配置より強力なアプリケーション配置方法も用意されています。セットアップ プロジェクトを使用すると、アプリケーションとすべての必要なファイルを配置できます。また、配置先ファイル システムの詳細な構造も提供されます。さらに、以下についても指定できます。
  - 配置先コンピュータで実行するカスタム動作
  - 起動条件
  - レジストリ設定
  - インストール プロセスで表示するメッセージや情報
- **ノータッチ デプロイメント**。この方法では、Web サーバーでアプリケーションのアセンブリをオンラインで使用できます。配置先のコンピュータ (クライアント コンピュータ) は、アプリケーションを実行するたびにサーバーに接続して、最新のバージョンのアセンブリをダウンロードできます。アセンブリは、.NET ランタイムとクライアント コンピュータのインターネット ブラウザによって信頼されているサーバーまたは Web ディレクトリにコピーされます。クライアント コンピュータがブラウザでその URL を使用して Web ディレクトリにアクセスすると、.NET Framework が必要に応じてアセンブリをダウンロードします。ノータッチ デプロイメントでは、クライアント コンピュータにサーバーへの永続的な Web 接続が必要です。

セットアップ プロジェクトと .NET の配置の詳細については、MSDN の「Setup Projects」と「No-Touch Deployment in the .NET Framework」を参照してください。

パフォーマンス カウンタの使用

.NET Framework には、最適化や診断の際に非常に便利な一連のパフォーマンス カウンタが用意されています。Visual Studio .NET のサーバー エクスプローラでは、定義済みのカウンタのセットを使用できます。これらは、アプリケーションにドラッグ アンド ドロップすると使用できます。これらの定義済みパフォーマンス カウンタを使用すると、たとえば、アプリケーションの次のような側面に関する貴重なデータを取得できます。

- .NET 共通言語ランタイム (CLR) の状態 (メモリ、セキュリティ、例外、およびその他の状態データ)
- システム (プロセス、スレッド、およびその他の情報)
- 物理デバイス (メモリ、プロセッサ、およびその他のデバイス)
- SQL Server (データベース、ロック、統計、およびその他の情報)

アプリケーションを Visual Basic .NET にアップグレードすると、アプリケーションのパフォーマンス カウンタを使用できるようになります。詳細については、MSDN の『.NET Framework General Reference』の「Performance Counters」を参照してください。

トレースとログの記録の使用

.NET Framework のトレースとログの記録のメカニズムは、System.Diagnostics 名前空間の Trace と Debug という 2 つのクラスを通じて利用できます。これらのクラスには、同じプロパティとメソッドが含まれています。両者の違いは、Debug はアプリケーションが**デバッグ** モードで実行されているときにしか利用できないのに対し、Trace は**デバッグ** モードと**リリース** モードの両方で利用できる点にあります。つまり、Debug クラスを使用して記述したコードは、**リリース**構成では実行できないことになります。

Trace や Debug を使用すると、システム イベント、エラー、および障害に関するログ情報を出力できます。出力の条件を指定することもできます。条件を指定すると、Trace と Debug は、指定した条件が満たされた場合 (式が true に評価された場合) にのみメッセージを生成します。

TraceListener クラスを使用すると、Trace や Debug の呼び出しによって生成される出力を、[出力] ウィンドウ、システムの**イベント ログ**、またはテキスト ファイルにリダイレクトすることができます。メッセージの生成を無効にしたり抑制したりするには (メッセージの重要度に応じて異なります)、System.Diagnostics の BooleanSwitch クラスと TraceSwitch クラスを使用します。

Trace と Debug の詳細については、MSDN の『.NET Framework Class Library』の「Trace Class」と「Debug Class」を参照してください。

アプリケーションのパフォーマンスと拡張性

実際に使用するアプリケーションでは、パフォーマンスと拡張性が非常に重要になります。そして多くの場合、これらの一方の要因が他方の要因に影響を及ぼします。したがって、現在のニーズにとってどちらがより重要かを決定する必要があります。

アプリケーションには、全体のパフォーマンスを左右する部分があります。こうした部分は、頻繁に呼び出される関数やループに含まれている可能性があります。これらの部分で実行されるタスクの量が増加すると、アプリケーション全体のパフォーマンスが目に見えて低下します。アプリケーションの全体的なパフォーマンスを改善するには、まず、こうしたパフォーマンスのボトルネックを洗い出しておくことがきわめて重要です。アプリケーション開発の間には、開発プロセスの最後にリソースのパフォーマンス チューニングの作業を行わなくても済むように、潜在的なパフォーマンスの問題を評価および特定する必要があります。

ソフトウェア開発におけるパフォーマンスの考慮事項の詳細については、MSDN の『Improving .NET Application Performance and Scalability』を参照してください。

.NET アプリケーションを開発する際には、考慮する必要がある一般的なパフォーマンスの問題がいくつかあります。この章では、中でも最も重要なものをいくつか取り上げます。

例外処理の注意点

例外処理は、.NET Framework 上のアプリケーションの開発における重要な側面の 1 つです。例外処理を使用すると、アプリケーションを容易にクラッシュさせてしまうようなエラーから簡単かつ適切に回復することができます。ただし、例外処理はシステム リソースの点ではコストの高い処理になるため、アプリケーションであまりむやみに使用しないようにする必要があります。

文字列処理の注意点

文字列が変更されると、元の文字列はガベージ コレクトされて、変更後の文字列を保持する新しいオブジェクトが作成されます。これは、変更回数が少ないうちは問題にならないかもしれませんが、極端に多くなるとシステムの負担になります。

.NET Framework には、文字列オブジェクトを操作する場合に使用するようにデザインされた特殊なクラスである StringBuilder クラスが含まれています。StringBuilder クラスには、文字列の内容を変更するためのメソッドが含まれています。このクラスは、System.Text 名前空間に含まれています。詳細については、第 11 章「文字列操作とファイル操作のアップグレード」の「StringBuilder を使用した文字列の置換」を参照してください。

データベース アクセスの注意点

.NET Framework では、データベース アクセスのチューニングのために、必要な機能以外は使用しないようにすることと、非接続型のアプローチを使用してアプリケーションをデザインすることが推奨されています。このアプローチでは、1 つの接続を長時間開いたまま保持する代わりに、複数の接続を次々に確立します。また、マイクロソフトでは、最大限のパフォーマンスが得られるように、クライアントからデータベースに直接接続するのではなく n 層のアプローチを使用することを推奨しています。.NET Framework の多くのテクノロジが多層アーキテクチャを利用できるよう最適化されているため、これをデザインの基本方針の一部と考えるようにしてください。

この以外にも、以下の推奨事項を考慮に入れる必要があります。

- 汎用のプロバイダではなく最適なマネージ プロバイダ (SQL 専用のプロバイダなど) を使用します。
- 後の移動のためにデータを格納する必要がない場合は、DataSet オブジェクトではなく DataReader オブジェクトを常に使用します。DataReader は DataSet よりオーバーヘッドが少ないため、パフォーマンスが向上します。
- マルチプロセッサ コンピュータに対しては Mscorsvr.dll を使用します。このライブラリが提供する最適化では、プロセッサの数に応じたスケーリングが行われます。
- 可能な限りストアド プロシージャを使用するようにします。ストアド プロシージャは最適化されていて、解釈、コンパイル、クライアントコンピュータへの送信なども必要ないため、パフォーマンスが向上します。
- 動的な接続文字列の使用に注意します。接続プール (後ほど説明します) は、一意の接続文字列に基づいて作成されます。動的な接続文字列を使用している場合は、同じ接続に対して別の接続文字列が使用される可能性がありますが、接続プールでは別な接続文字列は認識されません。このため、パフォーマンスを向上させる機会を逃すことになります。
- 自動生成されるコマンドは使用しないようにします。これらのコマンドは、実行のためのメタデータを取得するために頻繁にサーバーに接続しなければなりません。接続が行われるたびにパフォーマンスが低下します。
- 古い ADO のデザインに注意します。コマンドを実行するたびに、クエリで指定したすべてのレコードが返されます。
- データセットをコンパクトに保つようにします。データセットには必要なレコードのみを格納するようにします。データセット内のレコードが増えると、それらを格納するためにより多くのメモリが必要になります。これは、使用されないレコードについても同様です。
- 可能な限りシーケンシャル アクセスを使用します。大きなデータ型では特に重要です。シーケンシャル アクセスでは読み取るデータの単位が小さくなるため、大きなデータ型を読み取る場合の待ち時間が発生しません。

データベース アクセスとパフォーマンスの詳細については、MSDN の「Performance Tips and Tricks in .NET Applications」を参照してください。

マルチスレッドと BackgroundWorker コンポーネント

マルチスレッド (フリースレッディング) とは、複数の処理のスレッドを同時に実行するプログラムの機能です。

マルチスレッドは、コンポーネント プログラミングにおいて強力なツールとなります。マルチスレッド コンポーネントを作成すると、バックグラウンドで複雑な計算を実行しながら、ユーザー インターフェイスではいつでもユーザーの入力に応答できます。

.NET Framework には、コンポーネントをマルチスレッド化するための選択肢が複数用意されています。その 1 つが、System.Threading 名前空間の機能です。また、コンポーネントの非同期パターンを使用すること

もできます。BackgroundWorker コンポーネントは非同期パターンの実装の 1 つで、この高度な機能を簡単に使用できるようにコンポーネントにカプセル化しています。

マルチスレッドと BackgroundWorker コンポーネントの詳細については、MSDN の「BackgroundWorker Component Overview」と「Asynchronous Pattern for Components」を参照してください。

キャッシュ

ASP.NET には、新機能の 1 つとして、ページやアプリケーションのデータをキャッシュするためのシステムがあります。このためアプリケーションでは、ユーザーがページを表示するたびにコストの高いプロセスを繰り返す必要がありません。キャッシュはページ単位で行うことも、ユーザー コントロール単位で行うこともできます。

Web ページのパフォーマンス (主に速度面のパフォーマンス) はユーザーの満足度を左右します。Web サイトのトラフィックの増加を目指しているなら、遅い Web ページを放置しておくわけにはいきません。

スループットを改善するための方法の 1 つが、Web ページのキャッシュを使用することです。キャッシュとは、簡単に言うと、頻繁に使用される項目をメモリに格納することです。キャッシュは十分にテストされており、パフォーマンスを改善するための手法として確立されています。HTML ページをキャッシュすると、ページの読み込み速度を改善できます。これに対して、キャッシュのより間接的な例となるのが接続プールです。接続プールを使用すると、接続を効率的に管理して、異なるサービス要求の間で共有できます。接続プールでは、接続要求を受信するたびに、その要求を満たすために使用できるアイドル状態の接続がないかどうかがチェックされます。したがって、新しい接続が作成されるのは、既存の接続を利用できない場合だけです。.NET Framework は、要求に応じて生成される ASP.NET 応答ページのキャッシュや、個々のオブジェクトのメモリへの格納をサポートしています。

ASP.NET ページのキャッシュの主要な概念を以下に示します。

- 有効期限ポリシーの設定方法を把握します。
- キャッシュの場所をどこに設定すればよいかを把握します。
- どのような場合にページの複数のバージョンをキャッシュするのかを把握します。
- どのような場合にASP.NET ページの一部をキャッシュするのかを把握します。
- どのような場合にアプリケーション要求をキャッシュするのかを決定します。
- キャッシュから項目が削除された場合は必ずアプリケーションに通知します。

キャッシュの詳細については、MSDN の『.NET Framework Developer's Guide』の「Caching ASP.NET Pages」を参照してください。

通信と状態の管理

インターネットやイントラネットに配置するアプリケーションを構築する際には、アプリケーションがオープンかつ拡張可能であることが重要になります。この目標を達成するには、さまざまなプラットフォームや、環境に配置されている他のアプリケーションと通信および接続できるように、アプリケーションをデザインする必要があります。

この接続の実現においては、プロトコルと通信テクノロジがきわめて重要な役割を果たします。コンピュータ間の通信に使用される主要なプロトコルは分散コンポーネント オブジェクト モデル (DCOM) とリモート メソッド呼び出し (RMI) ですが、異なるネットワーク間の通信やインターネットでの通信が必要な場合にこれらのプロトコルを使用するのは簡単ではありません。このため、SOAP などの他のプロトコルがより確実で効率的な選択肢となっています。

アプリケーションは、1 つのコンピュータだけに配置されるようにデザインされるものではありません。優れたアプリケーションは、多くのコンピュータや多くのネットワークに配置されます。それらのコンピュータが通信できるようにするためには、セキュリティの実装という点でも、コンピュータ間およびネットワーク間のやり取りを促進することの複雑さという点でも、プロトコルとインフラストラクチャの構成が重要になります。

以降では、最もよく知られ、よく使用されている通信プロトコルと通信テクノロジについて、それぞれに固有のセキュリティ実装を中心に説明します。

DCOM から HTTP への移行

DCOM は、リモート プロシージャ コール (RPC) に基づくテクノロジです。ここでは、DCOM に基づいたアプリケーションを HTTP に移行するために必要なアップグレード プロセスの概要について説明します。

DCOM を使用していると、ファイアウォールを通過しなければならない場合に問題が生じる可能性があります。この問題は、DCOM で使用する必要があるポートが通常は無効にされているために発生します。しかし、RPC ポート マッパー (ポート 135) を有効にすると、データ開示攻撃を受けるおそれがあります。

さらに、DCOM をトンネリング モード (DCOM over HTTP) で使用する場合も、侵入者が利用できる秘密のチャネルとなる可能性があるため、セキュリティ上の問題になります。

以降では、マーシャリング、Web プロキシ、およびサービス エージェント テクノロジの使用など、こうしたセキュリティ上の問題を回避するための手法について説明します。

マーシャリングの使用

オブジェクトから別のオブジェクトに送信されるデータをパッケージ化するプロセスを *マーシャリング*と呼びます。マーシャリングは、スレッド間通信のために情報を変換するプロセスです。COM では、オペレーティング システム プロセスやコンピュータなどのコンテキストの境界にまたがってデータを移動する場合にマーシャリングが使用されます。

マーシャリングでは、マーシャリングされたオブジェクトの先頭にデータ型の情報を追加するのが一般的です。これにはさまざまな方法があります。以下は、そのうちの 3 つのアプローチの概要です。

- **XML マーシャリング**。このアプローチでは、XML のノードと属性を使用して、型の情報と、値を保持する XML テキストを表します。結果の XML はプレーン テキストとして送信することも、送信されたオブジェクトを再構築できる構造体として送信することもできます。
- **テキスト マーシャリング**。このアプローチでは、プレーン テキストを使用してデータを表現します。受信側のアプリケーションで単純な解析を行うことによってオブジェクトを再構築できます。
- **バイナリ マーシャリング**。このアプローチでは、データ型とサイズの情報を含むバイナリ ヘッダーを使用します。

XML マーシャリングとテキスト マーシャリングは判読が可能ですが、一般にパフォーマンスが劣ります。XML やテキストからオブジェクトを構築するプロセスは、オブジェクトのサイズが大きかったり数が多かったりするとコストが高くなります。一方、バイナリ マーシャリングは非常に効率的ですが、判読は不可能です。パフォーマンスのニーズと判読のニーズを比較検討して、それぞれの状況に合った最適なアプローチを決定する必要があります。

Web プロキシの作成

Web プロキシは、クライアント コンピュータに配置されるコンポーネントです。Web サービスへの要求の発行と結果の解釈をクライアントのために透過的に行って、クライアントが Web サーバーと簡単に通信できるようにします。

Web プロキシは、SOAP プロトコルを使用して XML Web サービスと通信します。Visual Studio .NET または Web サービス記述言語ツール (WSDL または Wsdl.exe) を使用すると、.NET Framework 用の Web プロキシを自動的に作成できます。これらのツールでは、目的の Web サービスに問い合わせを行って、そのサービスとやり取りする方法を確認できます。この問い合わせの結果に基づいてプロキシ クラスが作成されます。作成されたプロキシ クラスをクライアント アプリケーションで使用すると、Web サービスと簡単にやり取りできます。クラスを作成するときに使用する言語を指定できるため、たとえば、対象言語として Visual Basic .NET を指定すると、結果のクラス ファイルを Visual Studio やその他のソース コード エディタで表示して、必要に応じて後から修正することができます。その後、そのプロキシ クラスをクライアント アプリケーションに追加できます。

サービス エージェントの使用

サービス エージェントとは、アプリケーションの他のサービスとの連携を支援するサービスです。サービス エージェントは、しばしば、対象サービスのプロバイダによって提供されています。エージェントは、サービスを使用するアプリケーションにトポロジ的に近いところで実行され、サービスに送信する要求の準備と、サービスからの応答の解釈の両方をサポートします。

サービス エージェントにはいくつかの利点があります。たとえば、サービス インターフェイスが単純化されるため、統合が容易になります。これにより、Web サービスを、ローカルのインプロセス オブジェクトを使用する場合と同じくらい簡単に使用できるようになります。WSDL で生成されるプロキシを使用すると同じ効果を簡単に実現できますが、サービス エージェントには、WSDL ファイルのプロキシでは実現できない利点もあります。以下にその例を示します。

- **エラー処理**。サービスによって生成される可能性があるエラーを認識するサービス エージェントをデザインできます。これにより、統合の作業が非常に簡単になります。
- **データ処理**。サービス エージェントは、サービスからのデータを正しく把握してキャッシュすることができます。これにより、サービスの応答時間の大幅な短縮やサービスの負荷の軽減が実現でき、アプリケーションで非接続時 (オフライン時) の処理が可能になります。
- **要求の検証**。サービス エージェントでは、サービス要求の入力ドキュメントが正しいかどうかを送信前にチェックできます。これにより、ラウンドトリップによる遅延やサーバーの負荷を発生させることなく明らかなエラーを見つけることができます。ただし、この場合もサービスではすべての要求を受信時に検証する必要があります。
- **インテリジェントなルーティング**。一部のサービスでは、エージェントを使用して、要求を内容に基づいて特定のサービス インスタンスに送信できます。

また、サービス エージェントは、オンライン モードとオフライン モードの両方の動作をサポートしています。リモート サービスに到達できなくなるとそれを検出して、オフライン モードに切り替えます。その際、エージェントを使用しているアプリケーションの介入は一切必要ありません。これは、第 18 章「一般的なアプリケーションの改良シナリオ」の「Windows アプリケーションと Windows フォーム スマート クライアント」で説明したように、スマート クライアントのユーザーにとっては非常に大きな利点になります。

System.Messaging によるメッセージ キューの置き換え

System.Messaging 名前空間には、ネットワーク上のメッセージ キューへの接続、監視、および管理や、メッセージの送信、受信、およびピークのためのクラスが用意されています。この名前空間で提供される機能では、開発者は主に 2 つのクラスを通じてメッセージを送受信できます。1 つは Message、もう 1 つは MessageQueue です。Message クラスでは、送信するすべてのデータが、メッセージのルーティング、フォーマット、およびセキュリティを制御するプロパティと共にカプセル化されます。MessageQueue クラスは、個々のキューを表現するために使用されます。このクラスには、Send() や Receive() など、キューを操作するためのメソッドが含まれています。

メッセージ キューと System.Messaging 名前空間を使用することにより、.NET 開発者は、高い拡張性と信頼性を備えた、共通のキューの名前のみを共有する疎結合のアプリケーションを作成できます。

メッセージ キュー API アクセスの System.Messaging 名前空間へのアップグレードの詳細については、第 15 章「MTS アプリケーションと COM+ アプリケーションのアップグレード」の「メッセージ キューとキュー コンポーネント」を参照してください。

メモ : MSMQ 3.0 には、**System.Messaging** 名前空間では利用できない機能もあるので注意してください。たとえば、メッセージ キューでは、キューに含まれているメッセージの量に関する情報を提供できます。この機能が必要な場合は、.NET Framework のその他の機能 (キューのメッセージの量をトレースする .NET Framework の **ServicedComponent** など) を使用して実装する必要があります。

System.Messaging 名前空間の詳細については、MSDN の『.NET Framework Class Library』の「System.Messaging Namespace」を参照してください。

ODBC/OLE DB Data Access Components のアップグレード

Microsoft Data Access Components (MDAC) を使用すると、リレーショナル データ ソースや XML データ ソースなど、さまざまなソースのデータに接続およびアクセスできます。MDAC で利用できるコンポーネントには、ActiveX データ オブジェクト (ADO)、Open Database Connectivity (ODBC)、OLE DB などがあります。これらのコンポーネントを適切なプロバイダおよびドライバと組み合わせて使用することによって、さまざまな種類のデータ ソースにアクセスできます。

ここでは、.NET Framework の ODBC データ アクセス コンポーネントおよび OLE DB データ アクセス コンポーネントの概要について説明します。ADO.NET については次で詳しく説明します。

ODBC コンポーネントと OLE DB コンポーネントは次のように説明できます。

- ODBC。Microsoft Open Database Connectivity (ODBC) インターフェイスは C プログラミング言語のインターフェイスであり、これによりアプリケーションでさまざまなデータベース管理システムのデータにアクセスできます。この API を使用するアプリケーションでアクセスできるのはリレーショナル データ ソースのみです。
- OLE DB。OLE DB は、さまざまなデータ ストアの多様なデータにアクセスするための低レベルの COM インターフェイスの包括的なセットです。OLE DB プロバイダを使用して、データベース システム、ファイル システム、メッセージ ストア、ディレクトリ サービス、ワークフロー、およびドキュメント ストアのデータにアクセスできます。

.NET Framework データ プロバイダには、こうした広範な低レベルのデータ アクセス コンポーネントより有利な点が数多くあります。まず、データ アクセス アーキテクチャの単純化により、機能を犠牲にすることなくパフォーマンスが向上します。また、.NET Framework データ プロバイダは、CLR が提供するマネージ環境の中で実行されるため、COM コンポーネントとのやり取りは必要ありません。

Visual Basic 6.0 で ODBC データ ソースにアクセスするにはさまざまな手法があります。たとえば、ODBC 接続を持つデータ アクセス オブジェクト (DAO) を使用することができます。ODBCDirect DAO オプションを使用すると、リモートの ODBC データ ソースにアクセスすることもできます。また、リモート データ オブジェクト (RDO) を使用して ODBC データ ソースにアクセスすることもできます。RDO は、ODBC API の上にコード レイヤを実装します。これ以外に、RDO が提供する ODBC ハンドルを通じて直接 ODBC API とやり取りするという方法も考えられます。

Visual Basic 6.0 では、OLE DB へのプログラマ インターフェイスである ADO を使用して OLE DB データ ソースにアクセスできます。このインターフェイスは、OLE DB のほとんどすべての機能を公開します。

ODBC データ ソースや OLE DB データ ソースにアクセスする Visual Basic 6.0 アプリケーションを Visual Basic .NET にアップグレードするには、データ ソースとのやり取りに使用されるプログラマ インターフェイスをアップグレードする必要があります。このインターフェイスは、データ ソースによって DAO、RDO、または ADO のいずれかになります。DAO、RDO、および ADO のコードをアップグレードする方法については、第 12 章「データ アクセスのアップグレード」を参照してください。ただし、リソースがアップグレード可能な場合は、すべてのデータ アクセス インターフェイスを ADO.NET にアップグレードすることをお勧めします。

基になるデータ プロバイダ コンポーネントをアップグレードするには、.NET Framework データ プロバイダのいずれかを使用する必要があります。ここの残りの部分では、このプロセスについて説明します。

ODBC .NET Data Provider

.NET Framework データ プロバイダは、データベースとの接続、コマンドの実行、および結果の取得に使用されます。取得した結果は直接処理することも、ADO.NET の DataSet に格納することも、別のアプリケーション層を通じて送信することもできます。

ODBC .NET Data Provider は、.NET アプリケーションで参照可能な追加コンポーネントです。ネイティブ ODBC ドライバへのアクセスを提供するこのコンポーネントは、以下のドライバを含むすべての ODBC ドライバに準拠するようデザインされています。

- Microsoft SQL ODBC ドライバ
- Oracle 用 Microsoft ODBC ドライバ
- Microsoft Jet ODBC ドライバ

これらのドライバはそれぞれマイクロソフトによってテストされており、ODBC .NET Data Provider で使用できることが確認されています。

.NET Framework Data Provider for ODBC は、ODBC データ ソースを使用する中間層アプリケーションで推奨されます。このデータ プロバイダは .NET Framework のバージョン 1.0 には含まれていないので注意してください。ただし、このコンポーネントはダウンロードすることができます。詳細については、Microsoft Download Center の「ODBC .NET Data Provider」を参照してください。

このデータ プロバイダは、Microsoft.Data.Odbc 名前空間を使用します。アプリケーションからこのコンポーネントにアクセスできるようにするには、対応するアセンブリを参照として Visual Basic .NET プロジェクトに追加する必要があります。

次のコード例は、ODBC SQL Server データ プロバイダ用の接続文字列を設定し、その接続を使用して単純なコマンドを作成する方法を示しています。

```
…
Dim cn As OdbcConnection = New OdbcConnection _
    ("DRIVER={SQL Server};SERVER=MySQLServer;UID=sa;" & _
    "PWD=mypassword;DATABASE=northwind;")
Dim mystring As String = "select * from MyTable"
Dim cmd As OdbcCommand = New OdbcCommand(mystring)
cn.Open()
…
cn.Close()
…
```

OLE DB .NET Data Provider

OLE DB .NET Data Provider は、COM 相互運用機能を介した OLE DB へのネイティブ アクセスを提供します。ローカルトランザクションと分散トランザクションの両方をサポートしています。

ADO.NET でのテストが完了しているデータプロバイダを以下に示します。

- Microsoft OLE DB Provider for SQL Server
- Microsoft OLE DB Provider for Oracle
- OLE DB Provider for Microsoft Jet

このデータプロバイダは、バージョン 6.5 以前の Microsoft SQL Server を使用する中間層アプリケーションで推奨されます。.NET Framework Data Provider for OLE DB によって使用される OLE DB インターフェイスをサポートする他のすべての OLE DB プロバイダもサポートされます。このデータ プロバイダは、Microsoft Access データベースを使用する単一層アプリケーションでも推奨されます。OLE DB インターフェイスの詳細については、MSDN の『.NET Framework Developer's Guide』の「OLE DB Interfaces Used by the .NET Framework Data Provider for OLE DB」を参照してください。

次のコード例は、OLE DB SQL Server データプロバイダ用の接続文字列を設定し、その接続を使用して単純なコマンドを作成する方法を示しています。

```
…
Dim cn As OleDbConnection = New OleDbConnection( _
    "Provider=SQLOLEDB;Data Source=localhost;" & _
    "Integrated Security=SSPI;Initial Catalog=northwind")
Dim mystring As String = "select * from MyTable"
Dim cmd As OleDbCommand = New OleDbCommand(mystring)
cn.Open()
…
cn.Close()
…
```

.NET Framework から Oracle データベースへのアクセス

Microsoft Data Access Components (MDAC) には、ODBC インターフェイスおよび OLE DB インターフェイスを使用して Oracle データベースにアクセスするコンポーネントが用意されています。MDAC の現在のリリースでも Oracle ODBC データ プロバイダはサポートされていますが、新しいアプリケーションやアップグレードしたアプリケーションでは、.NET でサポートされている別の Oracle データ プロバイダを使用することをお勧めします。

以下のコード例のように、適切な接続文字列を使用することによって、ODBC .NET Data Provider で Oracle データ ソースにアクセスできます。

```
Dim cn As OdbcConnection cn= New OdbcConnection ( _
    "Driver = {Microsoft ODBC for Oracle};" & _
    "Server=myOracleserver;uid=myuid;pwd=mypwd")
```

OLE DB .NET Data Provider を使用するという選択肢もあります。次のコードは、この方法でアップグレードする場合の接続文字列のサンプルを示しています。

```
Dim cn As OleDbConnection = New OleDbConnection( _
    "Provider=MSDAORA.1;User ID=myUID;password=myPWD; " & _
    "Data Source=myOracleServer;Persist Security Info=False")
```

.NET Framework Data Provider for Oracle

バージョン 8.1.7 以降の Oracle データベースを使用している中間層または単一層のアプリケーションをアップグレードする際には、.NET Framework Data Provider for Oracle を使用することをお勧めします。このデータ プロバイダを使用すると、Oracle クライアント接続ソフトウェアを通じて Oracle データ ソースにアクセスできます。ローカル トランザクションと分散トランザクションがサポートされています。

このデータ プロバイダに対応するクラスは System.Data.OracleClient 名前空間にあり、System.Data.OracleClient.dll アセンブリに含まれています。このデータ プロバイダの機能を使用するには、このアセンブリを参照として追加する必要があります。.NET Framework Data Provider for Oracle は、.NET Framework のバージョン 1.0 には含まれていません。ただし、MSDN の Download & Code Center からダウンロードできます。

ADO から ADO.NET へのアップグレード

ADO は OLE DB プロバイダを通じてさまざまなソースのデータへのアクセスやそれらのデータの操作を可能にする一連のクラスです。ADO では、クライアント/サーバー アプリケーションと Web ベースのアプリケーションの両方を作成できます。

ADO が提供するコア サービスには次のようなものがあります。

- データ プロバイダへの接続

- コマンドの実行またはストアド プロシージャの呼び出し
- データの検索とナビゲーション
- データの取得

Visual Studio .NET には、分散、信頼性、および拡張性という今日のアプリケーションの要件を考慮に入れてゼロから再設計された、データ アクセスのための新しいクラスのコレクションが用意されています。この新しいデータ アクセス モデルが ADO.NET です。ADO.NET には、ADO の機能に加えて、次のような利点があります。

- **相互運用性**。ADO.NET ではすべてのデータが XML 形式で送信され、あらゆるプラットフォームで誰もが読むことのできる構造化テキストドキュメントとして提供されます。
- **拡張性**。ADO.NET では非接続データセットの使用が推進されており、自動接続プールがパッケージの一部としてバンドルされています。
- **生産性**。ADO.NET を使用すると、全体的な開発時間を短縮できます。たとえば、型指定された DataSet は、開発作業の迅速化とバグのないコードの作成に役立ちます。
- **パフォーマンス**。ADO.NET では非接続データセットを使用できるため、データベース サーバーがボトルネックになることがなく、アプリケーションのパフォーマンスが向上します。

ADO.NET には、データベースに依存しないデータ コンテナを実装する DataSet クラスと DataTable クラスが含まれています。また、SQL や OLE DB のコマンド、マネージ データ プロバイダと接続、データ リーダー、データ アダプタなどのデータベース指向のツールも用意されています。ADO.NET を使用すると、アプリケーションでデータをそのソース、形式、物理的な場所に関係なく処理できるようになります。ADO.NET は、データベース中心の ADO のアプローチとは違う、データ中心のデザインを特徴とする新しいオブジェクト モデルを提供します。

ADO から ADO.NET にスムーズに移行するためには、まず最初に、ADO.NET が ADO とは大きく異なることを理解する必要があります。また、ADO.NET の基本的な考え方を理解する必要もあります。

ADO.NET の概要

ADO.NET を使用するとデータ ソースに統一された方法でアクセスできます。ADO.NET では、Microsoft SQL Server、Oracle、および OLE DB や XML を通じて公開されるその他のデータ ソースのデータにアクセスできます。アプリケーションでは、ADO.NET を使用してこれらのデータ ソースにアクセスし、データを取得、操作、および更新できます。

ADO.NET のコンポーネントは、データ操作からデータ アクセスの要素を抽出するようにデザインされています。これは、ADO.NET の 2 つの中心的な機能グループによって実現されています。その 1 つは DataSet コンポーネント、もう 1 つは、Connection、Command、DataReader、DataAdapter などのオブジェクトを含むコンポーネントのセットである .NET Framework データ プロバイダです。

ADO.NET DataSet は、ADO.NET の非接続型アーキテクチャのコア コンポーネントです。DataSet は、データ ソースに依存しないデータ アクセスのために明確にデザインされています。DataSet には、データの行と列から成る 1 つ以上の DataTable オブジェクトのコレクションや、DataTable オブジェクト内のデータに関する主キー、外部キー、制約、およびリレーションシップの情報が含まれています。

ADO.NET アーキテクチャのもう 1 つのコア要素となるのが .NET Framework データ プロバイダです。データ プロバイダ コンポーネントは、データの操作と、前方参照のみで読み取り専用の高速なデータ アクセスのために明確にデザインされています。Connection オブジェクトは、データ ソースへの接続を提供します。Command オブジェクトは、データの戻し、データの変更、ストアド プロシージャの実行、パラメータ情報の送信や取得などのためのデータベース コマンドへのアクセスを提供します。DataReader は、データ ソースからのデータのパフォーマンスの高いストリームを提供します。最後の DataAdapter は、DataSet オブジェクトとデータ ソースの間の橋渡しを行います。DataAdapter は、Command オブジェクトを使用してデータ ソースで SQL コマンドを実行することにより、DataSet にデータを読み込んだり、DataSet 内のデータに加えられた変更でデータ ソースを更新したりします。

ADO と ADO.NET の違い

ADO のデータのメモリ内での表現は Recordset ですが、ADO.NET では DataSet と表現されます。Recordset が単一のテーブルとして機能するのに対し、DataSet は 1 つ以上のテーブルのコレクションになっています。このように、データセットでは、基になるデータベースの構造を模倣することができます。DataSet は、複数の異なるテーブルや、それらの間のリレーションシップに関する情報を保持できるため、自己リレーション テーブルや多対多のリレーションシップを持つテーブルなど、レコードセットよりはるかに豊富なデータ構造を保持することができます。一方、Recordset を置き換えるためのより直接的な選択肢となるのが DataTable オブジェクトです。Recordset と同様に単一のテーブルに対応する DataTable クラスは、Recordset をアップグレードする際の第 1 の候補となります。ここでは、より強力なコレクションである DataSet の使用に焦点を絞って説明します。

アプリケーション間で ADO.NET DataSet を送信するのは、ADO の非接続 Recordset を送信するよりはるかに簡単です。ADO の非接続 Recordset をコンポーネントからコンポーネントに送信するには、COM のマーシャリングを使用します。ADO.NET でデータを送信するには、DataSet を使用します。DataSet では、XML ストリームを送信できます。

ADO では、ADO MoveNext メソッドを使用して、レコードセットの各行をシーケンシャルにスキャンできます。ADO.NET では、行がコレクションとして表されるため、すべてのコレクションと同じようにテーブル内をループすることができ、序数や主キー インデックスを使用して特定の行にアクセスできます。DataRelation オブジェクトは、マスタ レコードと詳細レコードに関する情報を保持します。このオブジェクトが提供するメソッドを使用すると、作業中のレコードに関連付けられているレコードを取得できます。

ADO.NET では、接続を開くのは Select や Update などのデータベース操作を実行する間だけです。データセットに読み込んだ行を使って作業できるため、データ ソースへの接続を維持する必要はありません。ADO の Recordset でも非接続アクセスを利用できますが、ADO は主に接続アクセス向けにデザインされています。

ADO と ADO.NET の非接続処理には大きな違いが 1 つあります。ADO では OLE DB プロバイダを呼び出すことによってデータベースと通信するのに対し、ADO.NET ではデータ アダプタを通じてデータベースと通信し、データ アダプタが、OLE DB プロバイダまたは基になるデータ ソースによって提供される API を呼び出します。

ADO コンポーネントと ADO.NET コンポーネント

ここでは、最もよく使用される ADO オブジェクトと、ADO.NET オブジェクトでそれらのオブジェクトに対応するものについて説明します。以下の提案事項は、ADO から ADO.NET にアップグレードするときのコード変換のガイドラインとしてのみ使用するようにしてください。これらのコンポーネントの間には、動作の違いがある場合があります。したがって、これらの提案に従う場合は、そのコンポーネントのドキュメントをよく調べて、新しい動作が現在のニーズに合っているかどうかを確認することをお勧めします。

ADODB.Connection のアップグレード

このクラスは、SQL Server データベースへの開いている接続を表します。これに対応する ADO.NET の要素は、SqlClient のコンテキストでは SQLConnection、OleDb のコンテキストでは OleDbConnection です。

ADODB.Command のアップグレード

Command オブジェクトは、データベース内のデータに対して一般的なコマンドを実行するために使用されます。ADODB.Command は、SqlCommand または OleDbCommand に直接対応します。これは ADO で最も単純なオブジェクトの 1 つであるため、両方のクラスの間で対応するメンバを簡単に見つけることができます。

ADODB.Parameter のアップグレード

このクラスは、ADODB.Command に対するパラメータを表します。ADO.NET での動作も基本的に同じで、SQL コマンドや OLEDB コマンドのパラメータとして機能します。

ADODB.Recordset のアップグレード

ADODB.Recordset の機能は、ADO.NET の複数のクラスにまたがっています。ADO.NET には、データベースからの情報を管理するためのクラスが 3 つあります。

- DataReader。データベースからのデータの、高速で、前方参照のみの、読み取り専用のストリームを提供します。
- DataTable。メモリ内のデータの 1 つのテーブルを表します。同種の行のグループを格納および管理できます。
- DataSet。データのリレーショナルなメモリ内表現を提供します。これは、データベースから完全に切断された状態で機能するデータの完全なセットです。DataSet を作成するには DataAdapter が必要

です。このアダプタを使用して、DataSet が更新された場合にデータ ソースを更新できます。

ほとんどの場合は、DataTable を使用することによって、Recordset をより直接的にアップグレードできます。Recordset に同じ種類の単一のデータのグループが格納されている場合は、DataTable がアップグレードの最適な選択肢になります。ただし、アプリケーションで複数のテーブルや結果の階層を使用する場合は、DataSet を使ってアップグレードすることをお勧めします。いずれにしろ、ADODB.Recordset オブジェクトは、新しい DataSet クラスか DataTable クラスのオブジェクトに手動でアップグレードする必要があります。

メモ：Visual Basic Upgrade Wizard Companion では、Recordset から DataSet への自動的な移行がサポートされています。また、最もよく使用される ADO コンポーネントの一般的な使用パターンもサポートされています。Visual Basic Upgrade Wizard Companion およびその他の移行ツールや移行サービスの入手方法については、ArtinSoft の Web サイトを参照してください。

ADO を短期間でも使用したことがある場合は、おそらく Recordset のあらゆる情報が頭に入っていて、データ アクセスとはそのように機能するものだと考えてしまいます。ADO.NET ではこれに変更が加えられていますが、ADO.NET のコア データ オブジェクトである DataSet を使用した方が有利な点や、それによって改善される点もあります。以下は、DataSet に関する重要な基本情報の概要です。

- DataSet は、テーブル、リレーションシップ、およびビューのすべてを含むリレーショナル データベース全体をメモリ内で表現できます。
- DataSet は、元のデータ ソースへの継続的な接続なしに機能するようにデザインされています。
- DataSet のデータは要求に応じて読み込まれるのではなく、すべて一度に読み込まれます。
- DataSet には、カーソルの種類という概念はありません。
- DataSet には、現在のレコード ポインタはありません。For Each ループを使用してデータ間を移動することができます。
- DataSet に多数の編集内容を格納して、1 回の操作で元のデータ ソースに書き込むことができます。
- DataSet は汎用のオブジェクトですが、ADO.NET のその他のオブジェクトにはデータ ソースごとにさまざまなバージョンがあります。

メモ：DataTable クラスは .NET Framework のバージョン 2.0 で大幅に拡張されています。これらの拡張は、このクラスをスタンドアロンで使用できるようにすることを目標としています。たとえば、DataTable オブジェクトをシリアル化する機能が DataTable クラスに追加されています。

上の一覧からわかるように、ADO と ADO.NET の間の最も重要な変更は、Recordset と DataSet の違いに関連しています。この方法でアップグレードする場合は、まず ADO と ADO.NET の違いについてさらに詳しく

学ぶことをお勧めします。ADO.NET の詳細については、以下の記事を参照してください。

- MSDN の『.NET Framework Developer's Guide』の「Overview of ADO.NET」
- MSDN の『Visual Basic and C# Concepts』の「Comparison of ADO.NET and ADO」
- Microsoft Download Center の「Microsoft ADO.NET (Core Reference): Sample Code」
- Microsoft Download Center の「MSDN TV: First Look at ADO.NET 2.0」

まとめ

アプリケーションの開発は、ただ特定のビジネスの問題を解決すればよいというものではありません。堅牢かつ強力なアプリケーションを構築するためには、実際のビジネスの要件とは別だが欠かすことのできない重要な機能をいくつか追加しなければなりません。まず、重要な操作に対しては、ユーザーが確実に正しく識別および認証されるようにすることが重要です。また、パフォーマンスの高いアプリケーションを作成し、必要に応じて簡単に拡張できるようにして、アプリケーションを長期にわたって有効に活用できるようにします。これ以外に、アプリケーションを最新の状態に保ち、簡単に構成できるようにすることも重要です。この章では、アプリケーションをアップグレードする際に考慮する必要があるこれらのシナリオやその他の同様のシナリオについて紹介しました。Visual Basic .NET には、これらの状況に、より簡単に対処できる新機能が数多く用意されているので、この機会に、アプリケーションに新機能を追加 (または既存の機能を拡張) することを検討してみてください。

詳細情報

DES 暗号化アルゴリズムの詳細については、Ius Mentis の Web サイトの「The DES Encyption Algorithm」を参照してください。
URL は http://www.iusmentis.com/technology/encryption/des/ です。

セキュリティと暗号化の詳細については、MSDN の以下の記事を参照してください。

- 「Security Application Block」
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/security1.asp です。
- 「Cryptography Application Block」
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/crypto1.asp です。
- 「Cryptography Simplified in Microsoft .NET」
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/cryptosimplified.asp です。

.NET Framework Configuration Tool の使用方法の詳細については、MSDN の『.NET Framework Tools』の「.NET Framework Configuration Tool (Mscorcfg.msc)」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpconNETFrameworkAdministrationToolMscorcfgmsc.asp です。

アプリケーションを Visual Basic .NET にアップグレードした後に利用できるアプリケーションのパフォーマンスカウンタの詳細については、MSDN の『.NET Framework General Reference』の「Performance Counters」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/gngrfperformancecounters.asp です。

.NET Framework Configuration Tool の使用方法の詳細については、MSDN の『.NET Framework Tools』の「.NET Framework Configuration Tool (Mscorcfg.msc)」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cptools/html/cpconNETFrameworkAdministrationToolMscorcfgmsc.asp です。

アプリケーション構成ファイルの詳細については、MSDN の『.NET Framework Developer's Guide』の「Application Configuration Files」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconapplicationconfigurationfiles.asp です。

System.Configuration 名前空間の詳細については、MSDN の『.NET Framework Class Library』の「System.Configuration Namespace」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemconfiguration.asp です。

アプリケーション構成シナリオの詳細については、MSDN の『.NET Framework Developer's Guide』の「Application Configuration Scenarios」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconnetapplicationconfigurationscenarios.asp です。

Configuration Application Block の詳細については、MSDN の「Configuration Application Block」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/config.asp です。

セットアップ プロジェクトの詳細については、MSDN の「Setup Projects」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsintro7/html/vbconSetupProjects.asp です。

.NET の配置の詳細については、MSDN の「No-Touch Deployment in the .NET Framework」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vbtchNo-TouchDeploymentInNETFramework.asp です。

Trace クラスと Debug クラスの詳細については、MSDN の『.NET Framework Class Library』の「Trace Class」と「Debug Class」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemdiagnosticstraceclasstopic.asp
および
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemDiagnosticsDebugClassTopic.asp です。

ソフトウェア開発におけるパフォーマンスの考慮事項の詳細については、MSDN の『Improving .NET Application Performance and Scalability』を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenet.asp です。

データベース アクセスとパフォーマンスの詳細については、MSDN の「Performance Tips and Tricks」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dotnetperftips.asp です。

マルチスレッドと BackgroundWorker コンポーネントの詳細については、MSDN の「BackgroundWorker Component Overview」を参照してください。
URL は http://msdn2.microsoft.com/library/8xs8549b(en-us,vs.80).aspx です。

マルチスレッド プログラミングの詳細については、MSDN の「Asynchronous Pattern for Components」を参照してください。
URL は http://winfx.msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_fxmclicc/html/792aa8da-918b-458e-b154-9836b97735f3.asp です。

キャッシュの詳細については、MSDN の『.NET Framework Developer's Guide』の「Caching ASP.NET Pages」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconaspoutputcache.asp です。

System.Messaging 名前空間の詳細については、MSDN の『.NET Framework Class Library』の「System.Messaging Namespace」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemmessaging.asp です。

.NET Framework Data Provider for ODBC コンポーネントをダウンロードするには、Microsoft Download Center の「ODBC .NET Data Provider」を参照してください。
URL は http://www.microsoft.com/downloads/details.aspx?FamilyID=6ccd8427-1017-4f33-a062-d165078e32b1&displaylang=en です。

OLE DB インターフェイスの詳細については、MSDN の『.NET Framework Developer's Guide』の「OLE DB Interfaces Used by the .NET Framework Data Provider for OLE DB」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconoledbinterfacessupportedbyoledbnetdataprovider.asp です。

.NET Framework Data Provider for Oracle をダウンロードするには、MSDN の Download & Code Center を参照してください。
URL は http://msdn.microsoft.com/downloads です。

Visual Basic Upgrade Wizard Companion およびその他の移行ツールや移行サービスの入手方法については、ArtinSoft の Web サイトを参照してください。
URL は http://www.artinsoft.com です。

ADO.NET の詳細については、以下の記事を参照してください。

- MSDN の『.NET Framework Developer's Guide』の「Overview of ADO.NET」
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconoverviewofadonet.asp です。
- MSDN の『Visual Basic and C# Concepts』の「Comparison of ADO.NET and ADO」
  URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconadopreviousversionsofado.asp です。
- Microsoft Download Center の「Microsoft ADO.NET (Core Reference): Sample Code」
  URL は http://www.microsoft.com/downloads/details.aspx?FamilyID=bae2de67-0062-4bf5-b120-f970865be92e&DisplayLang=en です。
- Microsoft Download Center の「MSDN TV: First Look at ADO.NET 2.0」
  URL は http://www.microsoft.com/downloads/details.aspx?FamilyID=df7f7dc1-512f-4d07-b04f-17dde0fd318a&DisplayLang=en です。

21

アップグレード後のアプリケーションのテスト

アップグレード プロジェクトのテストは開発プロジェクトのテストと同様にきわめて重要ですが、アップグレード後のアプリケーションのテストに必要な労力は、同等の要件を最初から作成する新規アプリケーションの場合に比べるとはるかに少なくて済みます。

アップグレード後のアプリケーションのテストでは、テスタのアップグレード後アプリケーション テストの熟練度に応じて 2 つの方法から選択できます。1 つ目の方法は、2 つ目と比べて経験が必要ですが、労力は少なくて済みます。

1 つ目の方法では、アップグレード ウィザードによって自動的にアップグレードされたコードは、エラー、警告、問題 (EWI) がある箇所以外は正しく動作すると想定します。この方法では、EWI の修正のテストに労力を集中することができます。ただし、EWI が修正されたとしても、通常の開発プロジェクトで見つかるものとはパターンのまったく異なる機能的な欠陥が残る可能性があります。アップグレード後のアプリケーション テストの経験が豊富なテスタは、コードとアプリケーションのデザインをレビューすることによって、機能的な欠陥を見つけることができます。これについての標準的なプロセスやガイドラインはなく、手順はすべて経験をよりどころとします。

2 つ目の方法は、従来型のテスト方法とテスト プロセスに近いもので、1 つ目の方法よりもテストの労力がかかります。この方法は、アップグレード プロジェクトのテスト経験の浅いテスタにお勧めします。この章では、2 つ目の方法に重点を置いて説明します。

アップグレード対象のアプリケーションの規模が大きくなるほど、テストにかかるコストと労力の削減幅は小さくなります。テストの労力とコストは、アプリケーションのサイズに加えて、アップグレード方法、アップグレード計画、採用するテスト方法、および Visual Basic アップグレード ウィザードや ASP to ASP.NET Migration Assistant から示されるアップグレードの問題点の数と複雑さに左右されます。アップグレード後のアプリケー

ションのテストに選択できるテスト方法はいくつかあります。テスト方法は、それぞれのソフトウェア開発手法に応じたテストのやり方を基にします。どの方法でも、中心となるテスト プロセスは共通していますが、テスト プロセスをアップグレード プロセスに組み込む方法は異なります。それぞれのアップグレード方法は、いずれかのテスト方法で "最適" な方法となる可能性があります。

この章では、各アップグレード方法でどのテスト方法が "最適" であるかを説明します。テスト方法、テスト プロセス、テスト計画、およびテストのさまざまな作業について取り上げます。

Fitch & Mather Stocks 2000

この章で紹介する概念を説明するために、Fitch & Mather Stocks 2000 というアプリケーションを使用します。Fitch & Mather Stocks 2000 (FMStocks 2000) は、Microsoft Windows 2000 向けに開発された ASP/Visual Basic 6.0 アプリケーションで、インターネット上の公開株式取引の Web サイトをシミュレートします。オンライン株式取引のシナリオには、オンライン書店を追加する拡張を行っています。FMStocks 2000 を構成するユース ケースは以下のとおりです。

1. 既存ユーザーのログオン
2. 新規ユーザーのログオン
3. ログアウト
4. 記号または会社の検索
5. 口座残高の表示
6. ポートフォリオの表示
7. ポートフォリオ グラフの作成
8. 株式の購入
9. 株式の売却
10. ショッピング カートの表示
11. 商品の閲覧
12. チェックアウト

FMStocks 2000 のビジネス ロジック層を構成するプロジェクトは以下のとおりです。

- FMStocks_Bus.vbp。このプロジェクトには、オンライン株式取引機能に対応するビジネス コードが入っています。Account、Broker、Ticker、および Version の 4 つのクラス モジュールで構成されます。FMStocks_DB コンポーネントを参照します。
- FMSStore_Bus.vbp。このプロジェクトには、オンライン書店機能に対応するビジネス コードが入っています。Product および ShoppingCart の 2 つのクラス モジュールで構成されます。FMSStore_DB コンポーネントおよび FMSStore_EvtSub2 コンポーネントを参照します。
- FMSStore_Events.vbp。このプロジェクトでは、オンライン書店機能の外部注文処理イベントを作成するインターフェイスを定義します。

- FMSStore_EvtSub2.vbp。このプロジェクトでは、オンライン書店機能の外部注文処理イベントを実装します。FMSStore_Events.vbp プロジェクトで定義される FMSStore_Events コンポーネントを参照します。

FMStocks 2000 のデータ アクセス層を構成するプロジェクトは以下のとおりです。

- FMSStore_DB.vbp。このプロジェクトには、オンライン書店機能に対応するデータ アクセス コードが入っています。Product および ShoppingCart の 2 つのクラス モジュールで構成されます。FMStocks_DB.vbp プロジェクトで定義される FMSStore_DB コンポーネントを参照します。
- FMStocks_DB.vbp。このプロジェクトには、オンライン株式取引機能に対応するデータ アクセス コードが入っています。Account、Broker、DBHelper、Position、Ticker、Product、および ShoppingCart の 7 つのクラス モジュールで構成されます。

テストの目的

テストの目的は、アップグレード後のアプリケーションがアップグレードの目的を果たしているかどうかを確認することです。アップグレードの目的が同等の機能を実現することのみである場合、テストの目的は機能的な等価性を試すことです。しかし、パフォーマンスやセキュリティの改善のためにアプリケーションをアップグレードする場合、パフォーマンスとセキュリティをテストに含める必要があります。さらに、同等の機能にとどまらず、アプリケーションに新しい機能を加えている場合、その新機能の機能性、パフォーマンス、およびセキュリティのテストにも労力を割かなくてはなりません。アップグレードの目的の 1 つにグローバリゼーションも含まれる場合は、そのテストも必要です。

このガイドの第 1 の目的は、Microsoft Visual Basic 6.0 から Microsoft Visual Basic .NET へのアップグレードで同等の機能を達成するにはどのようにするかであるため、この章では機能の等価性のテストを中心に取り上げます。

テスト プロセス

テスト プロセスには、採用するテスト方法やアップグレード方法にかかわらず、実施すべき一連の基本手順があります。テストの中心となるこの基本手順は、以下のとおりです。

- テスト計画とテスト コードの作成
- テスト環境の作成
- デザインのレビュー
- コードのレビュー
- 既存のテストケースおよびコードの修正、または追加のテストケースおよびコードの作成
- 単体テスト (ホワイト ボックス テスト) の実施
- ブラック ボックス テストの実施
- プロファイリング (ホワイト ボックス テスト) の実施

上記の手順をアップグレード後のアプリケーションに対して実行し、機能仕様に準拠していること、すなわち元の Visual Basic 6.0 バージョンのアプリケーションと同等の機能を実現していることを検証します。

オプションで、テスト プロセスにデザイン レビュー、グローバリゼーション テスト、パフォーマンス テスト、セキュリティテストを含めることもできます。これらは、アップグレードの目的に含まれる場合にのみ必要となります。

以降では、それぞれの手順について説明します。

テスト計画とテスト コードの作成

テスト計画では、アップグレード後のアプリケーションのテストに使用するテスト ケースをドキュメント化します。これは、テスト コードの作成とテストの実施に必要となる実際のテストの労力を計画して見積もる際に役立ちます。また、テスト計画は、特に何度か繰り返したり仕様変更があったりする場合に、テストとその結果を追跡するためにも有効です。通常、テスト計画はアップグレードのライフ サイクルの早期の段階で作成します。テスト計画の作成には、以下のような情報が必要です。

- アップグレード プロジェクトの計画
- ユース ケース
- 機能仕様
- 元の Visual Basic 6.0 ソース コードまたはアップグレード後の Visual Basic .NET ソース コード

テスト ケースの作成方法は、実行するテストの種類によって異なります。デザイン レビュー、ホワイト ボックス テスト、およびコード レビューのテスト ケースは、ソース コードを見直すことによって作成します。ブラック ボックス テストのテスト ケースは、ユース ケースと機能要件に基づいて作成します。アップグレード プロジェクトでは、アプリケーションの Visual Basic 6.0 バージョン用のテスト ケースが既にある場合、このテスト ケースを Visual Basic .NET バージョン用にアップグレードできます。

テスト プロセスの各手順に対応するテスト ケースを調べる方法については、この章で後述する該当のトピックで説明します。ただし、アップグレードが進むに従い、既存のテスト ケースの変更や新しいテスト ケースの組み込みのためにテスト計画を更新する可能性があることに注意してください。また、アップグレード時に行われる機能仕様の変更によって、テスト計画を修正することもあります。

一般に、テスト計画は詳細テスト計画 (DTP) と詳細テスト ケース (DTC) という 2 つのドキュメントから構成されます。DTP は、各テスト ケースの重要度に基づいてインデックス付けや優先順位付けをしたテスト ケースの一覧です。各テスト ケースの要約の情報も含まれます。この情報には、テスト ケースを用いてテストする機能、プロジェクト、またはコンポーネントの簡潔な説明があります。DTC にはテスト ケースに関する詳細な情報が記載され、DTP のインデックスにマップされます。テスト ケースに関する詳細な情報には、テスト ケースを実行するための手順、期待される結果、実際の結果、テスト ケース実行の状態 (成功/失敗)、テスト ケースの実行に必要なデータ/リソースなどがあります。DTC に記載されたテスト ケースを実行するたびに、実際の結果およびテスト実行の状態の情報で DTC を更新する必要があります。また、DTP と DTC のドキュメントはテスト ケースを変更する際にも更新します。

表 1 は、FMStocks 2000 のログイン ユース ケースのブラック ボックス テスト用のテスト計画を示しています。

表 1: FMStocks 2000 のログイン ユース ケースのブラック ボックス テスト用テスト計画

シナリオ 1		既存ユーザーのログイン手順の機能をテストする
優先度		高
コメント		
1.1	高	ログインとホーム ページの両方が正しい形式で表示されることをテストする。つまり、すべてのリンク、フォント、コンテンツが既存の Visual Basic 6.0 アプリケーションと同じになっている。
1.2	高	ユーザーが有効な電子メールの名前とパスワードを入力した場合に、ユーザーがホーム ページにリダイレクトされることをテストする。
1.3	高	SQL Server が稼動していない場合に、"Cannot open database connection." というメッセージが 表示されることをテストする。
1.4	高	入力された電子メールの名前またはパスワードが無効である場合に、"Invalid e-mail and password combination.Please try again." というメッセージが表示されることを テストする。

表 2 は、FMStocks 2000 のログイン機能のブラック ボックス テスト用サンプル テスト ケースを示しています。テスト ケースには、"必要なデータ"、"実際の結果"、"テストは OK か (Y/N)" という列も含まれています。ここではこれらの情報がないため、表に載せていません。

表 2: FMStocks 2000 のログイン機能テスト用サンプル テスト ケース

テスト ケース	優先度	テストする条件	実行の詳細	期待される結果
1.1	高	ログインとホーム ページの両方が正しい形式で表示されることをテストする。つまり、すべてのリンク、フォント、コンテンツが既存の Visual Basic 6.0 アプリケーションと同じになっている。	ログインとホーム ページを Visual Basic 6.0 バージョンの FMStocks 2000 の対応するページと比較し、次の点を検証する。 コンテンツ Web ページの全般的なルック アンド フィール 入力フィールドの場所 フォント 他ページへのリンク 機能	ユーザーが、アップグレード後のバージョンと Visual Basic 6.0 バージョンの FMStocks のログインとホーム ページの間で、ルック アンド フィール、フォント、機能、コンテンツなどの違いを感じない。
1.2	高	ユーザーが有効な電子メールの名前とパスワードを入力した場合に、ユーザーがホーム ページにリダイレクトされることをテストする。	ログイン ページで次の電子メールの名前とパスワードを入力する。 電子メール : ta450 パスワード : ta	ユーザーはサイトに入力でき、ホーム ページにリダイレクトされる。

実際にテストを行うためのテスト コードは、テスト ケースが完成した後で作成します。一般にテスト コードは、テスト ケースの実行を自動化するコード、サンプル テスト プロジェクト、テスト スタブ、プロファイリングおよびモニタリング コード、テスト プロセスをサポートするその他のコードで構成されます。テスト コードは、ホワイト ボックス テストとブラック ボックス テストのいずれにも必要です。

テスト環境の作成

テスト環境は、通常、テスト対象であるアップグレード後のコンポーネントまたはアプリケーション、そのコンポーネントまたはアプリケーションの実行に必要な基本ソフトウェア、テスト ツール、およびテスト コードから構成されます。アップグレード後の Visual Basic .NET アプリケーションのテストでは、テスト環境として以下のものが必要です。

- テスト対象であるアップグレード後のアプリケーションまたはコンポーネント
- アプリケーションの実行に必要となる、次に挙げるような基本ソフトウェア
  - .NET Framework 1.1 および Visual Studio .NET 2003
  - Visual Basic 6.0 (反復型のテスト方法またはテスト駆動型のアップグレード方法の場合)
- デザイン レビューおよびコード レビューのツール (FxCop など) とドキュメント

- 単体テスト ツール (NUnit など)
- 次に挙げるようなプロファイリング ツール
  - CLR Profiler
  - Enterprise Instrumentation Framework (EIF)
  - Windows Management Instrumentation (WMI)
- 次に挙げるようなパフォーマンス テスト ツール
  - Application Center Test (ACT) (ASP.NET アプリケーション用)
  - パフォーマンス カウンタ

テスト ツールの詳細については、この後の「Visual Basic .NET アプリケーションのテスト用ツール」を参照してください。

アプリケーションが配置される環境によってアプリケーションの動作が変わる可能性があるため、特にブラック ボックス テストでは、テスト環境はターゲット/実稼働環境とほぼ同等である必要があります。

デザインのレビュー

この手順はオプションであり、デザインの最適化がアップグレードの目的でない場合や、Microsoft .NET Framework に固有の機能 (パフォーマンスやセキュリティの拡張機能など) を利用するためにアプリケーションをアップグレードする場合には実行しません。デザイン レビューを行う理由をいくつか挙げます。

- デザインが機能仕様を満たすことを検証するため。この検証は、追加する新しい機能、およびアプリケーションに加えた改良に対してのみ実施します。この検証では、機能仕様のそれぞれの機能項目をアプリケーションのデザインが満たすことができるかどうかを確認します。
- パフォーマンスと拡張性の観点から、デザインが最適な方法を実現していることを検証するため。この検証は、パフォーマンスの向上がアップグレードの目的の 1 つである場合、または .NET 固有の機能を利用するためにアプリケーションをアップグレードする場合にのみ実施します。詳細については、MSDN の『Improving .NET Application Performance and Scalability』の第 4 章「Architecture and Design Review of .NET Application for Performance and Scalability」を参照してください。
- セキュリティの観点から、デザインが最適な方法を実現していることを検証するため。この検証は、セキュリティの機能拡張がアップグレードの目的の 1 つである場合にのみ実施します。セキュリティの機能拡張のためのデザイン レビューの詳細については、MSDN の「Architecture and Design Review for Security」を参照してください。
- デザインがグローバリゼーションの最適な方法を実現していることを検証するため。この検証は、グローバリゼーションがアップグレードの目的である場合にのみ実施します。
- デザインがコードの保守性の最適な方法を実現していることを検証するため。

コードのレビュー

アップグレード後のアプリケーションのコード レビューは、実際には、ホワイト ボックス テストの一部です。しかし、コード レビューには多くの側面があるため、ホワイト ボックス テストと一緒ではなく、独立した手順として扱うだけの価値があります。

コード レビューのテスト ケースは、単体テストのテスト ケースと同時に作成します。コード レビューのテスト ケースを作成するには、デザイン ドキュメントが必要です。あるいは、Visual Basic 6.0 のコードを分析して、デザインをドキュメント化します。評価ツールを使用すると、Visual Basic 6.0 のコードに関する分析情報が得られます。

アップグレード後のコードのレビューでは、以下のことを行います。

- 実装がデザインに適合していることの検証。この検証は、アップグレード後のアプリケーションで改良された機能、またはアップグレード後のアプリケーションでのデザイン変更に対してのみ実施します。テスタには、実装がデザインに準拠していることを検証するために、デザイン ドキュメントを使用する必要があります。
- 名前付けの標準とコメントの標準がコード内で守られていることの検証。Visual Basic と .NET Framework のコードで推奨される標準のコーディング規約がいくつかあり、大文字使用のスタイル、大文字小文字の区別、インデント、頭字語および省略語など、さまざまなガイドラインを定めています。また、組織にも独自のコーディング標準がある場合があります。コーディング標準に準拠すると、コードの理解と保守が容易になります。

  Visual Basic .NET のコードで推奨されるコーディング標準の詳細については、MSDN の「Program Structure and Code Conventions」を参照してください。
- パフォーマンスと拡張性のガイドラインがコード内で守られていることの検証。この検証は、パフォーマンスの向上がアップグレードの目的である場合、またはパフォーマンスに影響を及ぼす可能性のある改良がアップグレード後のアプリケーションに適用されている場合にのみ実施します。

  パフォーマンスと拡張性のためのコード レビューの詳細については、MSDN の『Improving .NET Application Performance and Scalability』の第 13 章「Code Review: .NET Application Performance」を参照してください。
- セキュリティで保護されるコードを記述するためのガイドラインが守られていることの検証。この検証は、セキュリティの機能拡張がアップグレードの目的である場合、またはセキュリティに影響を及ぼす可能性のある .NET Framework テクノロジを利用してアップグレード後のアプリケーションの改良を行っている場合にのみ実施します。セキュリティのコード レビューは、セキュリティ ガイドラインに準拠していないために、セキュリティの脆弱性が生じたコードの領域を特定するために役立ちます。コード レビュー中にテスタが特に注意を払うのは、たとえば以下の領域です。
  - ユーザー名、パスワード、接続文字列など、ハードコーディングされた情報のレビュー

- 特権が必要な操作とセキュリティで保護されたリソースに対して適切なアクセス許可が設定されていて、任意の他のコードからのアクセスに対してアクセス許可を要求することのレビュー
- バッファ オーバーフローのレビュー
- クロスサイト スクリプティング攻撃の脆弱性のレビュー

セキュリティのコード レビューの詳細については、MSDN の『Improving Web Application Security: Threats and Countermeasures』の第 21 章「Code Review」を参照してください。

- グローバリゼーション関連のガイドラインに準拠していることの検証。この検証は、グローバリゼーションがアップグレードの目的の 1 つである場合にのみ実施します。グローバリゼーションの最適な方法の詳細については、MSDN で入手できる『.NET Framework Developer's Guide』の「Best Practices for Developing World Ready Applications」または「Best Practices for Globalization and Localization」を参照してください。
- 実装がコードの保守性のガイドラインに準拠していることの検証。実装のレビューでは、オブジェクト指向の原則が守られていることを検証します。コード レビュー中に、到達できないコードや不明瞭なコードを欠陥として検出または特定します。
- コードに適切な例外処理が実装されていることの検証。検証する必要がある例外処理のガイドラインは、たとえば以下のものです。
  - 例外処理コードのコントロールはアプリケーションのロジックを制御すべきではありません。
  - 例外ハンドラは、エラー メッセージのログ記録、リソースのクリーンアップ、例外メッセージのカスタマイズ、カスタム例外内での例外のラップなど、なんらかの値を追加します。ドキュメント化されている正当な理由以外で例外を処理するべきではありません。
  - 例外のスローにはコストがかかるため、絶対に必要な場合を除いて、例外を再スローすべきではありません。
  - 例外を受け取って何も処理しないことは避けるべきです。例外は、その機能で別の処理を要求しない限りは、呼び出し履歴を介して外部の層に戻せるようにします。
  - 例外ハンドラの Finally ブロックでは、リソースを適切に消去する必要があります。
  - 例外ハンドラのエラー メッセージをログに記録することをお勧めします。
  - 一般的な例外の前に、特定の例外を処理する必要があります。
  - カスタム アプリケーション例外は、標準の例外よりも優先されるべきです。

例外管理のアーキテクチャの詳細については、MSDN の『Exception Management Architecture Guide』を参照してください。

- エラーのシナリオや追加のテストのシナリオの特定。コード レビュー中には、コードを調べて、特にリソースの競合、リソースの使用率の上昇、無効な入力などに関連するエラーのシナリオを見つけ出します。たとえば、メソッド内のループが上限値として正数を指定する場合にのみ正しく動作し、負数ではエラーとなる、などです。ループの上限数が引数としてメソッドに渡される場合、引数として負数が渡されるとメソッドがエラーになることがあります。これをコード レビュー時に検出した場合、欠陥をログに記録し、このシナリオについての単体テスト用テスト ケースを作成します。

  一般に、リソース管理の競合、デッドロック、無効な入力などの問題の発生が疑われるコードの領域がある場合、該当コードを中心として単体テスト、プロファイリング、またはブラック ボックス テストを容易にするテスト ケースを作成します。

  以下の条件の追加のテスト シナリオを見つけるために、アップグレード後のコードもレビューします。

  - 境界条件
  - 無効な入力
  - スレッドセーフ条件とデッドロック条件

単体テスト (ホワイト ボックス テスト) の実施

ホワイト ボックス テストは、テスト対象のコンポーネントまたはアプリケーションの実装の詳細とプログラム ロジックの分析に基づくテストで、潜在的なエラーのシナリオを特定します。ホワイト ボックス テストには、アップグレード後のアプリケーションの単体テストとプロファイリングが含まれます。プロファイリングについては、後で説明します。

単体テストでは、1 つのコンポーネントをテストします。アップグレード後のアプリケーションのコンポーネントは、単体テストを実施して、ループ、条件構成要素、内部サブルーチンなどでエラーのシナリオを検出します。また、単体テストで重視するのは、アップグレード EWI が適切に修正されていて、機能に影響を与えたり、新しい欠陥を生じさせたりすることがないかどうかのテストです。単体テストは、アップグレード後のコンポーネント内のメソッドの動作が、元の Visual Basic 6.0 バージョンのコンポーネントの対応するメソッドと同様であるかどうかを確認するためにも実施します。

単体テストのテスト ケースは、テスト方法に応じて Visual Basic 6.0 のソース コードまたはアップグレード後の Visual Basic .NET のソース コードのレビューによって、また、アップグレード EWI の評価ツール レポートのレビューによって作成します。テスト ケースとテスト コードは、テスト計画の段階で作成しますが、コード レビューの段階で修正する可能性があります。コード レビューの段階で、追加のテスト ケースとテスト コードが加えられることもあります。

アップグレード プロジェクトでは、アプリケーションの Visual Basic 6.0 バージョン用の単体テストのテスト ケースがある場合、このテスト ケースを Visual Basic .NET バージョン用にアップグレードして再使用します。同様に、既存の Visual Basic 6.0 のテスト コードも Visual Basic .NET にアップグレードします。

単体テストは、通常、開発者ツールを利用して自動化されます。.NET Framework での単体テストの自動化を支援するツールの一例は、NUnit です。これについては、この章の「Visual Basic .NET アプリケーションのテスト用ツール」で説明します。

表 3 は、FMStocks 2000 の単体テスト用のサンプルのテスト計画とテスト ケースを示しています。

表 3: FMSStore_Cart.ShoppingCart クラスの単体テストのテスト計画

シナリオ 1		Class FMSStore_Cart.ShoppingCart
優先度		高
コメント		
1.1	高	Public Function GetByKey(ByVal AccountID As Integer, ByVal SKU As Integer) As ADODB.Recordset
1.2	高	Public Sub Add(ByVal AccountID As Integer, ByVal SKU As Integer)
1.3	高	Public Sub Buy(ByVal AccountID As Integer)
1.4	高	Public Function ListByAccount(ByVal AccountID As Integer) As ADODB.Recordset
1.5	高	Public Sub SetQuantity(ByVal AccountID As Integer, ByVal SKU As Integer, ByVal Quantity As Short)
1.6	高	Public Function TotalByAccount(ByVal AccountID As Integer) As ADODB.Recordset
1.7	高	Public Sub EmptyShoppingCart(ByVal AccountID As Integer)

表 4 は、FMSStore_Cart.ShoppingCart クラスの単体テスト用のサンプル テスト ケースを示しています。これ以外に "テストは OK か (Y/N)" という名前の列がありますが、ここではエントリがないため表に載せていません。

表 4: FMSStore_Cart.ShoppingCart **クラスのサンプル テスト ケース**

テスト ケース	優先度	テストする条件	実行の詳細	必要な データ
1.1a	高	Public Function GetByKey(ByVal AccountID As Integer, ByVal SKU As Integer) As ADODB.Recordset - すべてのパラメータが有効な範囲内。	すべての入力パラメータが有効な範囲内。 **入力パラメータ:** AccountID = 5249 SKU = 1004009 **予測される出力:** フィールドに次の値を持つ RecordSet Quantity = 1 SKU = 1004009 Price = 29.95 Description = The Secrets of Investing in Technology Stocks **実際の出力:**	NUnit テスト ケース: GetByKey_Valid
1.1b	高	Public Function GetByKey(ByVal AccountID As Integer, ByVal SKU As Integer) As ADODB.Recordset - AccountID パラメータが有効な範囲より大。	AccountID パラメータが有効な範囲より大。 **入力パラメータ:** AccountID = 5250000 SKU = 1004009 **予測される出力:** 空の RecordSet **実際の出力:**	NUnit テスト ケース: GetByKey_AccountInValid

ブラック ボックス テスト

ブラック ボックス テストでは、テスト対象のコンポーネントまたはアプリケーションの実装の詳細は不明です。ブラック ボックス テストは、エンドユーザー/クライアント エクスペリエンスをシミュレートし、エンドユーザーまたはクライアントのアクションに対するアップグレード後のアプリケーションの応答をテストします。ブラック ボックス テストの目的には、以下のものがあります。

- アップグレード後のアプリケーションが元の Visual Basic 6.0 バージョンのアプリケーションと同等の機能を持つことをテストする。
- アップグレード後のアプリケーションがアップグレードの目的を達成していることをテストする。
- アップグレード後のアプリケーションが、通常の使用および例外的な使用のシナリオに適切に応答することをテストする。アップグレード後のアプリケーションが、無効な入力に対してわかりやすいエラー

メッセージを出し、エラーを診断するのに十分な情報をユーザーに提供していることをテストする。

- 必要に応じて、パフォーマンスとセキュリティの機能をテストする。

ブラック ボックス テストには、以下のすべてが含まれます。

- **検証テスト**。コンポーネントまたはアプリケーションが、ドキュメント化された仕様と顧客/クライアントのニーズを満たしていることをテストします。
- **外部インターフェイス テスト**。コンポーネントの public でアクセス可能なメソッドとプロパティが期待される動作を示していることをテストします。
- **境界条件テスト**。パラメータの境界値をテストします。たとえば、メソッドがパラメータとして 0 より大きい整数値を必要とする場合、境界条件テストでは境界近くの -1、0、1 などの値をテストします。
- **破壊テスト**。アプリケーションのエラーが必ず起こるが、アプリケーションがエラーを適切に処理できる必要があるような極端な状態のシナリオをテストします。このテストには、リソースにアクセスするためのアクセス許可を削除する、コンポーネントからアクセスするデータベースを削除する、ネットワークを使用できないようにするなど、アプリケーションの実行環境の一部を除去したり削除したりすることも含まれます。アプリケーションは、データの損失やデータの状態の永続的な変更を引き起こさずにこのようなシナリオを処理することを期待されます。また、本来の実行環境が完全に復元された後で元に戻す機能も必要です。
- **負荷テスト**。アップグレード後のアプリケーションが、リソース (メモリ、I/O、プロセッサ、ネットワーク使用率など) の割り当て分を超過せずに、求められるパフォーマンスの目標を達成していることを検証するテストです。負荷テストは、アプリケーションの改良の場合、またはパフォーマンスの向上がアップグレードの目的である場合にのみ実行します。
- **ストレス テスト**。アプリケーションに通常の値を超える負荷をかけた場合の、アップグレード後のアプリケーションの動作を検証します。高い負荷のかかった状態でアプリケーションがエラーを適切に処理できることを検証するのが目的です。ストレス テストは、パフォーマンスの向上がアップグレードの目的である場合、または新しい .NET Framework テクノロジを利用してアプリケーションの改良を行っている場合にのみ実施します。
- **セキュリティ テスト**。アップグレード後のアプリケーションにセキュリティの脆弱性があるかどうかをテストします。セキュリティ テストのテスト ケースには、アップグレード後のアプリケーションのデザイン レビューを基に作成した脅威モデルと対応策を使用します。セキュリティ テストでは、対応策がアプリケーションに対する脅威に効果的に対処することを検証します。また、すべての脅威が特定されているかどうかも確認します。セキュリティ テストには、バッファ オーバーフローを引き起こす入力や SQL インジェクションの入力など、セキュリティを弱体化させる可能性のあるあらゆる種類の入力を使用します。セキュリティテストは、セキュリティがアップグレードの目的である場合に実施します。

- **グローバリゼーション テスト**。アップグレード後のアプリケーションがグローバリゼーションに対応しているかどうかを検証するテストです。グローバリゼーション テストは、グローバリゼーションがアップグレードの目的である場合に実施します。

アプリケーションが配置される環境によってアップグレード後のアプリケーションの動作が変わる可能性があるため、ブラック ボックス テストでは、テスト環境はターゲット環境にできるだけ近いものにする必要があります。したがって、動作が変わらないように、テスト環境はアプリケーションの配置環境と一致させます。

ブラック ボックス テストは、要件、機能仕様、およびユース ケースに基づいて実施します。ブラック ボックス テストのテスト ケースは、テスト計画の作成時にドキュメント化しておきます。

表 5 は、FMStocks 2000 の "株式の購入" ユース ケースのブラック ボックス テスト用のテスト計画を示しています。

表 5: FMStocks 2000 の "株式の購入" ユース ケースのブラック ボックス テスト用テスト計画

シナリオ 1		株式の購入の機能をテストする
優先度		高
コメント		
8.1	高	画面左側の "株式の購入" リンクをクリックすると、[Buy A Stock] ページが表示されることをテストする。
8.2	高	[Buy A Stock] ページの外観、ルック アンド フィール、フォント、その他のすべてのユーザー インターフェイスが ASP の対応するページと同じであることを検証する。
8.3	高	ユーザーがチッカーと有効な株式数を入力した後、株式がユーザーに割り当てられ、その株式の市場価格がユーザーの勘定の借方に記入されることを検証する。
8.4	高	無効なチッカーが入力された場合に、"The ticker symbol you entered is not valid. Please try again." というメッセージが表示されることを 検証する。
8.5	高	無効な株式数 (負の値や小数など) が入力された場合、トランザクションは生成されずに、"Invalid number of shares." というメッセージが 表示されることを検証する。
8.6	高	入力された株式数を購入するために十分な資金がユーザーの勘定にない場合、トランザクションは生成されずに、"Don't have enough cash in your account to purchase the requested shares." というメッセージが表示されることを検証する。
8.7	高	SQL Server が稼動していない場合に、"Cannot open database connection." というメッセージが 表示されることをテストする。
8.8	高	トランザクションの途中で SQL 接続が切断された場合でも、トランザクションが存在していることをテストする。

表 6 は、FMStocks 2000 の "株式の購入" ユース ケースのブラック ボックス テスト用のサンプル テスト ケースを示しています。テスト ケースには、"必要なデータ"、"実際の結果"、"テストは OK か (Y/N)" という列も含まれています。ここではこれらの情報がないため、表に載せていません。

表 6: FMStocks 2000 の "株式の購入" ユース ケースのブラック ボックス テスト用サンプル テスト ケース

テスト ケース	優先度	テストする条件	実行の詳細	期待される結果
1.1	高	ログインとホーム ページの両方が正しい形式で表示されることをテストする。つまり、すべてのリンク、フォント、コンテンツが既存の Visual Basic 6.0 アプリケーションと同じになっている。	ログインとホーム ページを Visual Basic 6.0 バージョンの FMStocks 2000 の対応するページと比較し、次の点を検証する。 コンテンツ Web ページの全般的なルック アンド フィール 入力フィールドの場所 フォント 他ページへのリンク 機能	ユーザーが、アップグレード後のバージョンと Visual Basic 6.0 バージョンの FMStocks のログインとホーム ページの間で、ルック アンド フィール、フォント、機能、コンテンツなどの違いを感じない。
1.2	高	ユーザーが有効な電子メールの名前とパスワードを入力した場合に、ユーザーがホーム ページにリダイレクトされることをテストする。	ログイン ページで次の電子メールの名前とパスワードを入力する。 電子メール : ta450 パスワード : ta	ユーザーはサイトに入力でき、ホーム ページにリダイレクトされる。

ホワイト ボックス テスト – プロファイリング

プロファイリングは、実行中にアプリケーションに関する統計情報 (アプリケーションの時間とリソースの使用状況、実際にたどる実行ツリーなど) を収集するプロセスです。この情報は、たとえば、最適化によって速度やリソース使用状況の改善が見込めるコンポーネントを特定するなど、さまざまな目的のために分析できます。

ホワイト ボックス テストのコンテキストでアップグレード後のアプリケーションをプロファイリングして、リソース管理、メモリ割り当て、リソース競合、コード カバレッジ、およびデッドロックが原因の問題とエラーのシナリオを特定します。リソース管理、デッドロック、リソース競合の問題の場合、アプリケーション全体に対するプロファイリ

ングは行いません。ブラック ボックス テストまたはコード レビューの際に特定され、これらの問題の発生が疑われるアプリケーション箇所に対してのみ行います。

アップグレード後のアプリケーションのプロファイリングでは、実行時のアプリケーションを監視します。コード カバレッジ テストと共にプロファイリングを使用して、実行されないコード (1 回も呼び出されず、実行されないコード) を検出できます。コード カバレッジ テストでは、実行ツリー全体をたどるテスト ケースを実行します。このようなテスト ケースは、考えられるすべての実行パスを少なくとも 1 回は通るようにデザインされます。コンポーネントまたはアプリケーションの追加機能は必須機能の動作に影響を与えないため、コード カバレッジ テストは、機能的な等価性の観点からは欠かせないものではありません。コード カバレッジ テスト時は、アプリケーションがプロファイリングされて、呼び出されるメソッドが特定されます。コード カバレッジ テストは考えられるすべての実行パスをたどるため、このテスト中に 1 回も呼び出されないメソッドがあれば、実行されないコードとして特定されます。

また、プロファイリングではリソース (ネットワーク I/O、ディスク I/O、メモリ使用率、CPU など) の過度の使用を明らかにする目的でも実行されます。リソースの過度な使用とは、コードが他のアプリケーションと比べて不相応にリソースをブロックしていることを示します。プロファイリングは、そのようなコードの検出に役立ちます。この種類のプロファイリングは、アップグレード後のアプリケーションが過度にメモリを消費する場合や、ブラック ボックス テストで実行完了に期待された時間より長くかかる場合に実行します。そのような場合、問題の原因と考えられる分離したコンポーネントに限定してプロファイリングを行い、原因を特定することができます。

また、メモリ リークやヒープの断片化がないことを確認するために、コードをプロファイリングすることもできます。この種類のプロファイリングは、この章で扱う範囲を超えています。.NET 共通言語ランタイム プロファイラを使用したメモリのプロファイリングの詳細については、MSDN の『Improving .Net Application Performance and Scalability』の「How To: Use CLR Profiler」を参照してください。

テスト方法の概要

テスト方法は、たとえば、ウォーターフォール モデルや反復型開発モデルなど、それぞれのソフトウェア開発手法に応じたテストのやり方を基にします。これらの方法は、異なる種類のアップグレード プロジェクトに適用することもできます。テスト プロセスは異なる方法の間でもきちんと整合性がとれていますが、テスト計画とテストの労力は方法によって違います。ここでは、アップグレード プロジェクトに適用できる複数のテスト方法について詳しく説明します。

ウォーターフォール型の手法に基づくテスト方法

テスト方法の 1 つの選択肢は、ウォーターフォール型開発手法に基づくものです。この方法では、アプリケーション全体のうち 1 つのプロジェクトをアップグレードした後で、アップグレード後のプロジェクトの単体テストを実施します。アプリケーションのすべてのプロジェクトをアップグレードした後、アップグレード後の全プロジェクト/コンポーネントを統合して、稼働するバージョンのアプリケーションを作成し、機能の等価性のシステム テストを実施します。このテスト方法を図 21.1 に示します。

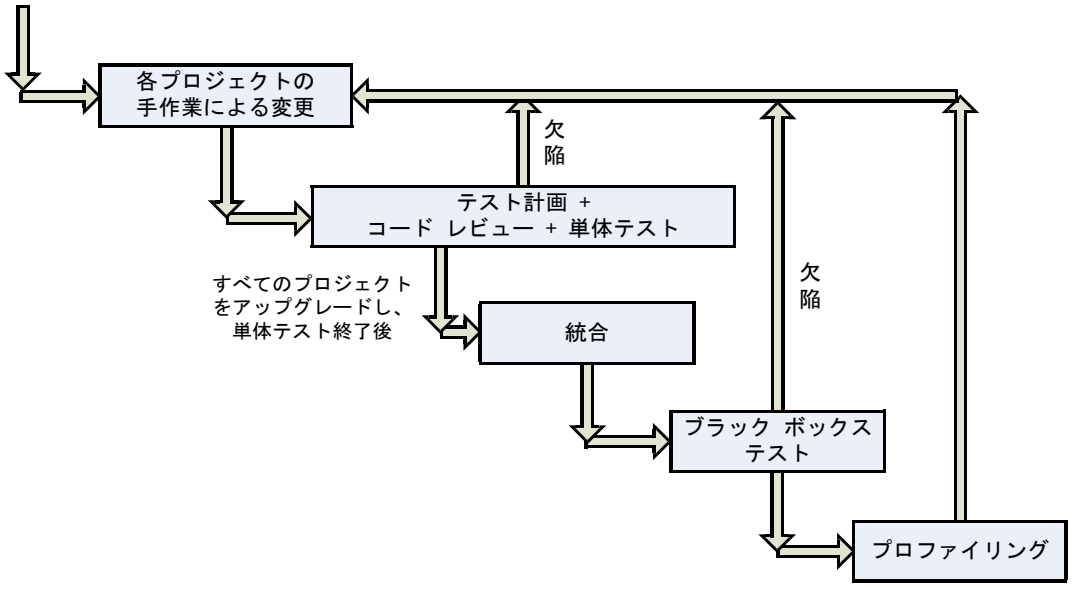

図 21.1

ウォーターフォール アップグレード プロセスを表す図

このテスト方法は、完全アップグレード方法に最適です。完全アップグレード方法では、統合して完全な稼働バージョンを作成する前に、アプリケーションのすべてのコンポーネント/プロジェクトをアップグレードします。次に、このアプリケーション全体のシステム テストで機能の等価性を検査します。完全アップグレード方法の詳細については、第 2 章「アップグレードの成功のためのプラクティス」の「完全アップグレード」を参照してください。

完全アップグレード方法では、アプリケーションの稼働バージョンは、アップグレードのライフ サイクルの後の段階まで作成されません。このため、完全アップグレード方法はウォーターフォール モデルに基づくテスト方法に適しています。しかし、これは短所でもあります。問題の切り分けはシステム レベルでのほうが単体レベルよりも難しくなるため、システム テスト フェーズで発生する問題は修正にコストがかかるのです。このため、このテスト方法は、コンポーネント数が少ない比較的単純なアプリケーションに限って使用することをお勧めします。

アップグレード後のアプリケーションが、たとえば ADO を ADO.NET に置き換えるなど、.NET 機能を使用した改良を行っている場合、その改良はデザイン フェーズで行う必要があります。システム テスト フェーズでデザインに関連する欠陥が発見された場合は修正にコストがかかるので、改良を伴うアプリケーションはこのテスト方法に適していません。さらに、このテスト方法は、アプリケーションの不確定要素がテスト フェーズにまで引き継がれてしまった場合には対処できません。このテスト方法では、前のアップグレード フェーズで問題が

すべて確定し、テスト フェーズまで不確定要素を残していないことが要求されます。アプリケーションに一部不確定要素がある改良が含まれている場合、この方法を採用するのは最善ではありません。

このテスト方法は、FMStocks 2000 のアップグレードに最適な方法です。評価ツールから生成されるソースコードのメトリクスとプロジェクト ファイルの概要によると、FMStocks 2000 は完全なアップグレードが可能な中規模サイズのアプリケーションです。FMStocks 2000 は、22 個のファイル、3 つのプロジェクト グループ、および 6 つのプロジェクトで構成されており、その合計コード行数は 4,943 行です。FMStocks 2000 アプリケーションのアップグレードの詳細については、付録 D の「FMStocks 2000 のアップグレード – ケース スタディ」を参照してください。このケース スタディでは、ここで示す方法に従ったこのアプリケーションのテストについても詳細に説明しています。

この方法の一般的なテスト プロセスを以下に示します。

1. アップグレード計画時およびアップグレード方法を決定した後で、機能仕様または元の Visual Basic 6.0 バージョンのアプリケーションに基づき、テスト計画およびブラック ボックス テストのテスト ケースを作成します。Visual Basic 6.0 バージョンのアプリケーション用のテスト計画とテスト ケースが既にある場合、このテスト計画とテスト ケースを Visual Basic .NET バージョンのアプリケーション用にアップグレードします。単体テストのテスト ケースとテスト計画はこの段階では作成されていないため、評価レポートから単体テストの労力の仮見積もりを設定します (ただし、Visual Basic 6.0 バージョンのアプリケーション用の単体テスト ケースがあれば、見積もりの基準として役立ちます)。
2. 各アップグレード プロジェクトに対して、自動化されたアップグレードを実行します。
3. EWI の修正を中心に、各アップグレード プロジェクトに手作業で変更を適用します。
4. アップグレード後のプロジェクトの単体テスト用のテスト計画、テスト ケース、およびテスト コードを作成します。
5. 必要に応じて、デザイン レビューを実施します。
6. コード レビューを実施します。
7. コード レビューに基づいて、単体テストまたはブラック ボックス テスト用の追加のテストケースおよびテスト コードを変更または追加します。
8. EWI に重点を置き、単体テストを実施します。
9. すべてのプロジェクトをアップグレードして単体テストを終えたら、統合して稼働するアプリケーションを作成します。
10. ブラック ボックス テストを実施します。
11. ブラック ボックス テストによってリソース管理やデッドロックの問題が疑われる場合、プロファイリングを実施し、問題を特定します。
12. コード レビュー、単体テスト、ブラック ボックス テスト、またはプロファイリングで検出された欠陥を修正すると、コード レビュー、単体テスト、ブラック ボックス テスト、およびプロファイリングのサイクル全体をやり直す回帰テストが必要です。

反復型の手法に基づくテスト方法

アップグレード プロジェクトのテスト方法のもう 1 つの選択肢は、反復型開発手法に基づくものです。このテスト方法は、ウォーターフォールに基づくテスト方法の欠点を克服していて、複雑なアプリケーションの完全アップグレード プロジェクト、および段階的アップグレード プロジェクトに最適です。その理由は後で説明します。

段階的アップグレード方法では、各プロジェクトコンポーネントをアップグレードしてシステムに統合します。システムは、相互運用のテクニックを介してやり取りする、.NET のアップグレード後のプロジェクトと Visual Basic 6.0 プロジェクトから構成されます。各プロジェクトをアップグレードして、単体テストを実施します。単体テストに合格したら、システムに統合して、システム全体の機能の等価性をテストします。この方法では、稼働するシステムが常に存在することになります。

この方法では、ウォーターフォールに基づくテスト方法を複数回繰り返します。各プロジェクトをアップグレードしてシステムに統合するたびに 1 回の反復を実施します。その後で、システムの機能の等価性を検査します。このテスト方法は、段階的アップグレード方法と、数多くのコンポーネントを持つ複雑なプロジェクトの完全アップグレード方法に最適です。このテスト方法は、従来のテスト方法と比べると、テスト チームと開発チーム間での綿密な調整が必要です。特にテスタは、アップグレード プロセスの各段階でどのコンポーネントがアップグレードされるかを知るために、アップグレード計画を把握しておく必要があります。

既に説明したように、完全アップグレード方法では、稼働するシステムが作成されるのはプロジェクトの終わりのいくつかのフェーズのみです。統合後のアプリケーションで出現するバグは単体テストで発見されるバグよりも切り分けが難しいため、システム テスト フェーズなど、アップグレード サイクルの最後のフェーズでバグが検出されると、修正にコストがかかります。反復型のテスト方法では、各反復が終わるたびにバグを発見し、修正処置をとることができます。各反復での変更は独立しているため、各反復の終わりで検出されるバグは、アップグレードが完了した後で検出されるバグよりも修正のコストが少なくて済みます。したがって、大規模で複雑なプロジェクトのアップグレードでは、反復型モデルに基づくテスト方法を採用することをお勧めします。ここで、テスト チームがアップグレード後のプロジェクトの 1 つの単位を受け取ってテストし、システムに統合後、段階的アップグレード方法の場合と同様にシステム全体の機能の等価性をテストするとします。統合のタスクはオーバーヘッドを伴いますが、システムが十分にテストされることが保証されます。

この方法は、アップグレード サイクルの終わりのいくつかのフェーズで重大な欠陥を発見するリスクを軽減しますが、完全アップグレード方法では、各プロジェクトがアップグレードされるたびにシステム統合のオーバーヘッドがかかります。既にアップグレードと統合が完了しているプロジェクトに変更がある場合にも、オーバーヘッドがかかります。したがって、テスト チームは効率的な変更管理プロセスを実装して、反復の間にシステムに加える変更を管理する必要があります。

FMStocks 2000 アップグレードは、このテスト方法に適切な候補ではありませんが、この方法を適用する場合、アップグレードとテストの手順は以下のようになります。

1. ASP ページを ASPX ページにアップグレードして、ビジネス コンポーネントと統合します。
2. ASPX ページは、ASP ページとの機能の等価性をテストします。
3. FMStocks_Bus.vbp プロジェクトをアップグレードして、システムと統合します。アップグレードされた Visual Basic .NET バージョンの FMStocks_Bus コンポーネントを、稼働システム内の Visual Basic 6.0 バージョンの FMStocks_Bus と置き換え、これからアップグレードされる FMStocks_DB コンポーネントと相互運用できるようにします。
4. アップグレード後の FMStocks_Bus プロジェクトの単体テストを実施し、その後、統合したシステムの機能の等価性をテストします。
5. FMSStore_Bus.vbp プロジェクトをアップグレードします。アップグレード後の Visual Basic .NET バージョンを、システム内の元の Visual Basic 6.0 バージョンと置き換えます。コンポーネントのアップグレード後のバージョンが、FMSStore_DB および FMSStore_EvtSub2 コンポーネントと相互運用できるようにします。
6. FMStocks 2000 アプリケーションの別のプロジェクトに対して、アップグレード プロセスを続行します。各プロジェクトをアップグレードした後で単体テストを実施して、システム全体と統合し (これからアップグレードする Visual Basic 6.0 コンポーネントがある場合、相互運用のテクニックを使用)、システム全体の機能の等価性をテストします。これを、すべてのプロジェクトのアップグレードが完了するまで、アプリケーションの各プロジェクトについて繰り返します。

この方法のテスト プロセスを以下に示します。

1. アップグレード計画時およびアップグレード方法を決定した後で、機能仕様または元の Visual Basic 6.0 バージョンのアプリケーションに基づき、テスト計画およびブラック ボックス テストのテスト ケースを作成します。Visual Basic 6.0 バージョンのアプリケーションの分析とレビューに基づいて、単体テスト用のテスト ケースとテスト コードを作成します。Visual Basic 6.0 バージョンのアプリケーション用のテスト計画、テスト ケース、およびテスト コードが既にある場合、Visual Basic .NET バージョンのアプリケーション用にアップグレードします。
2. 1 つの反復サイクルで、アップグレード プロジェクトの自動化されたアップグレードを実行します。
3. EWI の修正を中心に、アップグレード後のプロジェクトに手作業で変更を行います。
4. アップグレード後のプロジェクトをシステムに統合します。
5. 必要に応じて、デザイン レビューを実施します。
6. コード レビューを実施します。
7. コード レビューに基づいて、単体テストまたはブラック ボックス テスト用の追加のテストケースおよびテスト コードを変更または追加します。
8. EWI の修正に重点を置き、単体テストを実施します。
9. ブラック ボックス テストを実施します。

10. ブラック ボックス テストによってリソース管理やデッドロックの問題が疑われる場合、プロファイリングを実施し、問題を特定します。
11. 残りの反復サイクルについて、手順 2 から 手順 10 を繰り返します。
12. コード レビュー、単体テスト、ブラック ボックス テスト、またはプロファイリングで検出された欠陥を修正すると、コード レビュー、単体テスト、ブラック ボックス テスト、およびプロファイリングのサイクル全体をやり直す回帰テストが必要です。

アジャイル手法に基づくテスト方法

アップグレード対象のアプリケーションが大規模で非常に複雑な場合 (複数のプロジェクトまたはプロジェクト グループで構成されている、プロジェクトまたはコンポーネントの間に複数の相互参照を含む、外部システムへの依存関係を持つなど)、これまでに説明したテスト方法はあまり効果的でなく、信頼性も高くありません。これは、すべてのビジネス COM コンポーネントを Web サービスに置き換えるなど、.NET テクノロジを利用して Visual Basic 6.0 アプリケーションを改良する場合にも当てはまります。この状況は、改良の要件と仕様が明確に定義されていない場合には、さらに複雑になります。

複雑さに対応できる十分な堅牢性を備え、変更をすばやくスムーズに取り込むテスト方法が、この状況では要求されます。アジャイル開発手法に基づくテスト方法は、この状況に最適です。

アジャイル手法は、反復しながら進行する点で反復型の手法に似ていますが、反復の期間がもっと短くなっています。また、この期間は、(反復型の手法の場合のような) 計画された目標ではなく、厳密な配信期限と見なすのが一般的です。反復型の手法の開発、テスト、フィードバックの原則が反映されています。テスト駆動型開発は、アジャイル手法の中心となる実践です。テスト駆動型開発は、テスト ケースを作成してからテスト ケースに応じた機能を構築し、その後実装をリファクタリングする手法です。この場合、開発サイクルが非常に短くなります。テスト駆動型開発の原則をアップグレード プロセスに拡大すると、単一の機能をテストするテスト ケースを使用し、その機能のすべてのテスト ケースに合格するまでコードを修正する必要があります。この後でリファクタリングを行います。このプロセスがテスト駆動型アップグレード プロセスです。

テスト駆動型アップグレード プロセスでは、テスト ケースはシステムの機能仕様を反映しています。アプリケーションの要件または仕様が変わらない限り、テスト ケースは変更しません。アップグレードはテスト ケースに適合するように、一度に 1 機能ずつ実行します。テスト駆動型アップグレード プロセスは、フィードバックの概念を重視しています。アップグレード フェーズの途中で、周期的にテストの反復が挿入されます。つまり、アップグレードを繰り返すたびに、続いて 1 回のテストを行います。1 回のテストは、その前のアップグレードに対するフィードバックとなります。1 回の反復で徐々に機能をアップグレードし、それに続くテストでは、追加された

機能とシステム全体の機能をテストします。不合格になるテスト ケースが、目的のアップグレード機能に関連していないテスト ケースのセットのみになるまでは、アプリケーションの他の部分をアップグレードしません。

テスト駆動型アップグレード プロセスでは、テスト ケースの作成は、実際のアップグレードまたは改良作業を開始する前、プロジェクトの早期のフェーズで行います。アップグレード後のコードは、テスト ケースに応じて変わります。テスト ケースには、単体テストとシステム テストのケースが含まれます。テスト ケースは、アプリケーションの機能全体を対象範囲とし、さまざまなユース ケースやアプリケーションの機能に基づいてサブセットに分類されます。既に述べたように、アップグレード フェーズ全体は、複数回の反復から成ります。1 つのアップグレードの反復では、テスト ケースのサブセットを満足するようにアプリケーションの一部のみをアップグレードし、アプリケーションの機能全体のサブセットをアップグレードします。次に、テスト ケースの完全なセットを用いてテストを実施することによって、アップグレード後のコードについてフィードバックを行います。満足すべきテスト ケースのサブセット内の任意のテスト ケースに不合格となった場合、不合格のテスト ケースに合格するようにアップグレード後のコードを変更します。ターゲットであるテスト ケースのサブセットに含まれるテスト ケースすべてに合格するまで、反復は完了しません。アップグレードした機能に関連付けられたすべてのテスト ケースに合格したら、コードをリファクタリングして、デザインの重複や実行されないコードを取り除き、識別子や名前、コメントを整えます。その後、アップグレードの反復の次のサイクルを開始します。アップグレード フェーズの次の反復で、既にアップグレードが済んだ機能の上に重ねて、アプリケーションの別の部分をアップグレードします。この反復の後には、再度、テストの反復とコード リファクタリングが続きます。次の反復で機能を追加し、追加された機能をテストします。アップグレード後のコードがテストの反復時に中断した場合、現在およびそれ以前の反復に関連付けられたこれらのテスト ケースのみを修正するようコードを変更します。機能のアップグレードは、これらのテスト ケースに合格するまでは完了したものと見なされません。後続のアップグレードの反復は、これが完了した後でないと実施できません。アップグレード プロセスの途中で要件に変更があった場合は、それに応じてテスト ケースを更新して、アップグレード後のコードでテスト ケースを再度実行します。テスト ケースを変更しているので、一部のテスト ケースに不合格となることがあります。不合格のテスト ケースに合格するようにアップグレード後のコードを変更する必要があります。これがテスト駆動型アップグレード プロセスの実施方法です。

このようなアップグレードとテストの周期的なサイクルを重ねる方法では、アップグレード フェーズの途中で要件または仕様が変更されても影響はほとんどなく、容易に取り込むことができます。このため、このテスト方法では変化に柔軟に対応できます。さらに、アプリケーションの改良の場合は特に、要件と仕様がアップグレードの進行に伴って進化する可能性があります。テスト駆動型アップグレードは効率的に変更を取り込むことができるため、アップグレード前に詳細に計画をたてるオーバーヘッドが軽減されます。アプリケーションの改良では、アーキテクチャの変更の範囲を指定しますが、アーキテクチャの変更方法を詳細に記述する必要はありません。アップグレード後のコードの欠陥または未決の問題がすべて解決されるまで次のアップグレードの反復に進めないようにすると、複雑なアプリケーションでもコードの堅牢性が保証されます。

アップグレードとテストを複数回反復するという類似点に加えて、テスト駆動型アップグレードと反復型モデルに基づくテスト方法の他の類似点として、アップグレードとテストのフェーズを通じて、アップグレード後のアプリケーションの稼働するバージョンが保持されることがあります。ただし、テスト駆動型アップグレード方法と反復型モデルに基づくテスト方法の違いは、アップグレードとテストのフェーズを通じて、アップグレードされるシ

ステムが機能を保つ方法にあります。反復型モデルに基づくテスト方法では、アップグレードされていないコンポーネント間の相互運用によってシステムは機能を保持します。ここでは、システムを機能させるための相互運用性に関するコードを記述するオーバーヘッドがかかります。対照的に、テスト駆動型アップグレード方法にはこのオーバーヘッドはありません。代わりに、独立して稼働できる複数の機能単位にシステムを分割して、それぞれの機能単位をアップグレードし、既存のアップグレード済み機能単位と統合します。アプリケーションを複数の機能単位に分割するには、さまざまなユース ケースまたはアプリケーションの機能に基づいて、コンポーネントまたはプロジェクトを分類する必要があります。たとえば、FMStocks 2000 プロジェクトはさまざまなユース ケースと以下のようにマップできます。

- 既存ユーザーのログイン
  - FMStocks_Bus.vbp
  - FMStocks_DB.vbp
- 新規ユーザーのログイン
  - FMStocks_Bus.vbp
  - FMStocks_DB.vbp
- ログアウト
  - FMStocks_Bus.vbp
- 記号または会社の検索
  - FMStocks_Bus.vbp
  - FMStocks_DB.vbp
- 口座残高の表示
  - FMStocks_Bus.vbp
  - FMStocks_DB.vbp
- ポートフォリオの表示
  - FMStocks_Bus.vbp
  - FMStocks_DB.vbp
- ポートフォリオ グラフの作成
  - FMStocks_Bus.vbp
  - FMStocks_DB.vbp
- 株式の購入
  - FMStocks_Bus.vbp
  - FMStocks_DB.vbp

- 株式の売却
  - FMStocks_Bus.vbp
  - FMStocks_DB.vbp
- ショッピング カートの表示
  - FMSStore_Bus.vbp
  - FMSStore_DB.vbp
- 商品の閲覧
  - FMSStore_Bus.vbp
  - FMSStore_DB.vbp
- チェックアウト
  - FMSStore_Bus.vbp
  - FMSStore_DB.vbp
  - FMSStore_EvtSub2.vbp
  - FMSStore_Events.vbp

FMStocks 2000 は、テスト駆動型アップグレードに適切な候補ではありませんが、テスト駆動型アップグレード方法を使用してアップグレードする場合、手順は以下のようになります。

1. ASP ページを ASPX ページにアップグレードします。プロジェクト FMStocks_Bus.vbp および FMStocks_DB.vbp をアップグレードします。
2. アップグレード後の FMStocks_DB および FMStocks_Bus プロジェクトの単体テストを実施します。
3. 以下の機能/ユース ケースの機能の等価性をテストします。
   - 既存ユーザーのログイン
   - 新規ユーザーのログイン
   - ログアウト
   - 記号または会社の検索
   - 口座残高の表示
   - ポートフォリオの表示
   - ポートフォリオ グラフの作成
   - 株式の購入
   - 株式の売却
4. プロジェクト FMSStore_Bus.vbp、FMSStore_EvtSub2.vbp、FMSStore_Events.vbp、および FMSStore_DB.vbp をアップグレードします。
5. アップグレード後のプロジェクト FMSStore_Bus.vbp、FMSStore_EvtSub2.vbp、FMSStore_Events.vbp、および FMSStore_DB.vbp の単体テストを実施します。

6. 以下のユース ケースのシステム テストを実施します。
   - ショッピング カートの表示
   - 商品の閲覧
   - チェックアウト
   - 既存ユーザーのログイン
   - 新規ユーザーのログイン
   - ログアウト
   - 記号または会社の検索
   - 口座残高の表示
   - ポートフォリオの表示
   - ポートフォリオ グラフの作成
   - 株式の購入
   - 株式の売却

Visual Basic .NET アプリケーションのテスト用ツール

Visual Studio .NET の現在のバージョンでは、Visual Basic .NET アプリケーションのテストの自動化に特化したツールは提供されていません。しかし、テストの自動化のためのツールが他に複数あります。ここでは、そのツールの一部について説明します。

NUnit

NUnit は、.NET Framework でアプリケーションの単体テストの実行に広く普及しているツールです。NUnit では、Visual Basic .NET を含む .NET Framework 言語に適用できる単体テストのフレームワークを提供します。NUnit の詳細とダウンロードについては、NUnit の Web サイトを参照してください。

FxCop

FxCop ツールは、コード レビューの自動化に使用するツールです。FxCop ツールには、この章の「コードのレビュー」で示したガイドラインに似た、さまざまな定義済み規則のセットがあります。ツールではアプリケーションのバイナリを解析して、定義済み規則に準拠した実装であるかどうかを判別します。これらの規則はカスタマイズ可能で、機能を拡張できます。FxCop の詳細については、.NET Framework Community の Web サイトで「FxCop Team Page」を参照してください。

Application Center Test (ACT)

Application Center Test は、パフォーマンス テスト ツールです。Web サーバーに負荷をかけ、ASP.NET アプリケーションのパフォーマンスをテストするようにデザインされています。ACT では、サーバーへの複数の接続を開くことによって数多くのユーザーがいるような状況を作り出せるので、同時に数千のユーザーが Web サイトにアクセスする実地環境をシミュレートできます。ACT を使用して、Web アプリケーションの負荷テストとストレス テストを実施できます。さらに、プログラム可能な動的テストを用いて機能テストを実行できます。ACT は、Visual Studio .NET の Enterprise Edition に付属しています。ACT の詳細については、MSDN の「Microsoft Application Center Test 1.0, Visual Studio .NET Edition」を参照してください。

Visual Studio Analyzer

Visual Studio Analyzer は、分散アプリケーションの分析とデバッグに使用するパフォーマンス分析ツールです。このツールでは、高レベルでコンポーネントを分析し、分散環境で問題のあるコンポーネントを特定するために利用できる情報を提供します。Visual Studio Analyzer の詳細については、MSDN の「Visual Studio Analyzer」を参照してください。

Trace クラスと Debug クラス

.NET Framework には、実行時の有用なデバッグ メッセージを印刷するために使用できる 1 組のクラスが付属していて、アプリケーションの流れを確認し、異常な動作を見つける助けとなります。このクラスは Debug および Trace で、いずれも System.Diagnostics 名前空間に定義されています。

両クラスはよく似ていて、いくつかの静的メソッドを含んでいます。任意のテキスト メッセージを印刷するメソッド、テスト表現が true と評価される場合にのみメッセージを印刷するメソッド、ある条件が true と評価されない場合は処理を中断して次の動作の指示をユーザーに求めるメソッド、単純なインデント コマンドを用いてテキスト メッセージをフォーマットするメソッドなどです。

2 つのクラス間の唯一の違いは、既定では Debug メソッド呼び出しはアプリケーションの Release ビルドにコンパイルされませんが、Trace メソッド呼び出しは Release ビルドにコンパイルされることです。この動作を変更するには、コンパイラのコマンド ラインでコンパイラ ディレクティブを定義し (または、あまり望ましくありませんが、ユーザー コード内に)、任意の時点で Trace および Debug メソッド呼び出しを指定します。このデバッグ メカニズムは、C プログラミング言語から流用した条件付きコンパイルという技法を使用します。コンパイラ ディレクティブを定義しない場合、Trace クラスと Debug クラスを使用しても、最終的なコードにコンパイルされないため、最終リリースへのパフォーマンス上の影響はありません。最終的なリリース バージョンでのパフォーマンスを懸念せずに同量のトレース情報をコードに入れることができるため、これはとても便利な方法です。

さらに、これらのクラスの利点として、構成によって、外部構成ファイルを使用して生成する情報の量を増やしたり減らしたりできることがあります。これにより、生成される情報を調整できます。トレース情報を含むアプリケーションをリリースすることもできますが、必要になるまで表示されないようにトレースをオフにしなければなりません。

TraceContext クラス

TraceContext クラスは、Trace クラスと Debug クラスに似た機能を提供しますが、この機能は ASP.NET アプリケーションでのみ利用できます。このクラスに含まれるメソッドは多くありませんが、ASP.NET によって生成された HTML ページに条件付きテキストを追加し、ブラウザでの表示用にフォーマットできます。ページには要求に関する情報が自動的に挿入され、Web アプリケーションの問題のデバッグに役立ちます。

CLR Profiler

CLR Profiler は、アプリケーションごとのメモリ使用率をプロファイリングするためのツールです。CLR Profiler は、マネージ アプリケーション、マネージ ヒープ、ガベージ コレクタの間の相互作用をプロファイリングします。CLR Profiler は、メモリ リークを特定するために使用します。CLR Profiler の詳細については、MSDN の『Improving .NET Application Performance and Scalability』の「How To: Use CLR Profiler」を参照してください。

Enterprise Instrumentation Framework (EIF)

EIF は、.NET アプリケーションのインストゥルメント化に使用し、プロファイリングとトレースを可能にします。プロファイリングとトレースの実行後、アプリケーションのメトリクスが取得されます。EIF は、イベント ログ機能、Event Tracing for Windows (ETW)、および Windows Management Instrumentation (WMI) をカプセル化します。EIF は、エラー イベント、警告イベント、および情報イベントを発行する手段を提供します。ビジネス固有のイベントを発行することもできます。EIF を使用して、コード カバレッジ テストのプロファイリングを行います。リソース競合または無効な入力によって引き起こされる、機能上の抜け穴または仕様の欠陥のシナリオを特定するプロファイリングにも使用します。

EIF の使用方法の詳細については、MSDN の『Improving .NET Application Performance and Scalability』の「How To: Use EIF」を参照してください。

パフォーマンス カウンタ

.NET Framework で利用でき、テスト要件を満たす上で役立つもう 1 つの Windows サービスは、パフォーマンス カウンタです。Windows は、.NET ランタイムやオペレーティング システムの一部の多様な側面を監視しています。ユーザーが独自のパフォーマンス カウンタを定義し、パフォーマンス モニタ ツール (Perfmon.exe) や独自のアプリケーションからこのカウンタを監視することもできます。

.NET のパフォーマンス カウンタの詳細については、MSDN の『.NET Framework General Reference』の「Performance Counters」を参照してください。

カスタム パフォーマンス カウンタの作成および使用方法の詳細については、MSDN の patterns & practices ガイド『Improving .NET Application Performance and Scalability』の、以下の How To を参照してください (How To へのリンクは左側のペインにあります)。

- 「How To: Monitor the ASP.NET Thread Pool Using Custom Counters」
- 「How To: Time Managed Code Using QueryPerformanceCounter and QueryPerformanceFrequency」
- 「How To: Use Custom Performance Counters from ASP.NET」

まとめ

テストは、おそらく、アップグレード プロジェクトを含むソフトウェア開発プロジェクトで最も重要な部分です。綿密に計画されて実施されたテスト プロセスは、バグの特定と、ソフトウェアの正当性の検証に役立ちます。

アップグレード開発のテストで選択できるオプションは、すべての種類のソフトウェア開発で使用できるものと同じです。実際に、テスト方法自体は、さまざまなソフトウェア開発手法に基づいています。アプリケーションのアップグレード時に適用を選択する特定のテスト方法は、アップグレードするアプリケーションの特性に左右されます。アップグレードの最後にテストを実施するか、アップグレード プロセス中の反復の段階ごとに実施するかは、アプリケーション自体のサイズと複雑さによって決まります。小規模で単純なアプリケーションでは、アップグレードが完了するまで待つ方が、オーバーヘッドを最小にするので効率的です。より大規模で複雑なアプリケーションでは、個々のコンポーネントをアップグレードするたびにテストを行う、つまり、反復型のテストを行うのが良い方法です。これらの方法ではテスト パスが増えるためオーバーヘッドも増大しますが、修正が容易なアップグレードの早期の段階にバグやその他の問題を切り分けやすいという利点があります。

テスト プロセスの手順を明確に理解すると、テストが効率的になり、結果の精度も高まります。

この章では、アップグレード後のアプリケーションのテスト方法を決定するための詳細情報と、テストの実施に必要な手順について説明しました。

詳細情報

パフォーマンスのアップグレードの詳細については、MSDN の『Improving .NET Application Performance and Scalability』の第 4 章「Architecture and Design Review of .NET Application for Performance and Scalability」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenetchapt04.asp です。

セキュリティの機能拡張のためのデザインレビューの詳細については、MSDN の「Architecture and Design Review for Security」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secmod/html/secmod78.asp です。

Visual Basic .NET のコードで推奨されるコーディング標準の詳細については、MSDN の「Program Structure and Code Conventions」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vbconProgrammingGuidelinesOverview.asp です。

パフォーマンスと拡張性のためのコードレビューの詳細については、MSDN の『Improving .NET Application Performance and Scalability』の第 13 章「Code Review: .NET Application Performance」を参照してください。
URL は http://msdn.microsoft.com/library/en-us/dnpag/html/ScaleNetChapt13.asp です。

セキュリティのコードレビューの詳細については、MSDN の『Improving Web Application Security: Threats and Countermeasures』の第 21 章「Code Review」を参照してください。
URL は http://msdn.microsoft.com/library/en-us/dnnetsec/html/THCMCh21.asp です。

グローバリゼーションの最適な方法の詳細については、MSDN で入手できる『.NET Framework Developer's Guide』の「Best Practices for Developing World Ready Applications」または「Best Practices for Globalization and Localization」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconbestpracticesforglobalapplicationdesign.asp

または
http://msdn.microsoft.com/library/en-us/vsent7/html/vxconBestGlobalizationPractices.asp です。

例外管理のアーキテクチャの詳細については、MSDN の『Exception Management Architecture Guide』を参照してください。
URL は http://msdn.microsoft.com/library/en-us/dnbda/html/exceptdotnet.asp です。

.NET 共通言語ランタイムプロファイラを使用したメモリのプロファイリングの詳細については、MSDN の『Improving .Net Application Performance and Scalability』の「How To: Use CLR Profiler」を参照してください。
URL は http://msdn.microsoft.com/library/en-us/dnpag/html/scalenethowto13.asp です。

NUnit の詳細とツールのダウンロードについては、NUnit の Web サイトを参照してください
URL は http://www.nunit.org です。

FxCop の詳細については、.NET Framework Community の Web サイトで「FxCop Team Page」を参照してください。
URL は http://www.gotdotnet.com/team/fxcop/ です。

ACT の詳細については、MSDN の「Microsoft Application Center Test 1.0, Visual Studio .NET Edition」を参照してください。
URL は http://msdn.microsoft.com/library/en-us/dnbda/html/exceptdotnet.asp です。

Visual Studio Analyzer の詳細については、MSDN の「Visual Studio Analyzer」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsavs70/html/veoriVisualStudioAnalyzerInBetaPreview.asp です。

CLR Profiler の詳細については、MSDN の『Improving .NET Application Performance and Scalability』の「How To: Use CLR Profiler」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenethowto13.asp です。

EIF の使用方法の詳細については、MSDN の『Improving .NET Application Performance and Scalability』の「How To: Use EIF」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenethowto14.asp です。

.NET のパフォーマンス カウンタの詳細については、MSDN の『.NET Framework General Reference』の「Performance Counters」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/gngrfperformancecounters.asp です。

カスタム パフォーマンス カウンタの作成および使用方法の詳細については、MSDN の patterns & practices ガイド『Improving .NET Application Performance and Scalability』の、以下の How To を参照してください (How To へのリンクは左側のペインにあります)。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/scalenet.asp です。

- 「How To: Monitor the ASP.NET Thread Pool Using Custom Counters」
- 「How To: Time Managed Code Using QueryPerformanceCounter and QueryPerformanceFrequency」
- 「How To: Use Custom Performance Counters from ASP.NET」

付録

A

関連トピックの参照先

アップグレード プロジェクトに関する技術的な詳細に加えて、アップグレードを行う際に留意すべき、副次的ですが重要なトピックがあります。この付録のトピックはこのガイドの範囲を超えていますが、重要であるためここで取り上げています。ここに挙げたトピックは、このガイドでは直接説明していないため、オンラインで入手可能な情報へのリンクを記載しています。

Visual Basic 6.0 Resource Center

Visual Basic .NET は Visual Basic 言語の将来バージョンですが、Visual Basic 6.0 での作業を好む開発者もまだ数多くいます。Microsoft Visual Basic 6.0 Resource Center は、このような開発者のためにあります。

リソース センターには、Visual Basic 6.0 の開発者を対象とした記事、リンク、ダウンロードが掲載されています。チュートリアル、解説、ダウンロード可能なコンポーネントなどもあります。

提供している内容は、純粋な Visual Basic 6.0 アプリケーションとコンポーネントにとどまりません。このサイトでは、Visual Basic .NET のコンポーネントや機能を Visual Basic 6.0 で使用する方法についての情報も提供しています。ここには、Visual Basic 6.0 コンポーネントを Visual Basic .NET で使用する方法の詳細も含まれます。また、Visual Basic 6.0 から Visual Basic .NET への移行に役立つリソースも用意されています。

コーディング標準

標準を定義してすべてのコードに一貫して適用すると、コードの理解とメンテナンスがはるかに容易になります。一般に、このような標準には以下が含まれます。

- 変数、サブルーチンなどのプログラマが定義する識別子の命名規則
- インデントのスタイル
- モジュール、クラス、プログラムの構造
- コメントとドキュメントのスタイル

多くの企業では、独自の標準を定義して開発者に配布します。現在標準が存在していない場合は、定義して利用することを慎重に検討するようお勧めします。Visual Basic .NET コードのコーディング標準の定義の詳細については、MSDN の『Visual Basic Language Concepts』の「Program Structure and Code Conventions」を参照してください。

ファイル I/O オプションの選択

第 11 章「文字列操作とファイル操作のアップグレード」では、フラット ファイル アクセスをアップグレードする方法について詳しく説明しました。また、Microsoft Visual Basic 互換性ライブラリの使用や、ファイル ストリームの使用など、さまざまな方法を説明しました。

これらの方法から選択するのは難しいかもしれません。どのような場合に互換ライブラリではなくストリームを選択すべきかについての説明は、このガイドの範囲を超えています。詳細については、MSDN の『Visual Basic Language Concepts』の「Choosing Among File I/O Options in Visual Basic .NET」を参照してください。

詳細情報

Microsoft Visual Basic 6.0 Resource Center (VBRun) は、MSDN にあります。
URL は http://msdn.microsoft.com/vbrun/default.aspx です。

Visual Basic .NET コードのコーディング標準の定義については、MSDN の「Program Structure and Code Conventions」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vbconProgrammingGuidelinesOverview.asp. です。

I/O オプションの選択方法の詳細については、MSDN の「Choosing Among File I/O Options in Visual Basic .NET」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcn7/html/vatskchoosingamongfileiooptionsinvisualbasicnet.asp です。

付録

B

アプリケーション ブロック、フレームワーク、およびその他の開発支援

Microsoft Visual Basic .NET と Microsoft .NET Framework は、開発者が新しいアプリケーションを作成したり、既に存在するアプリケーションを強化したりするのを支援する新機能を数多く備えています。この付録では、アプリケーション ブロック、フレームワーク、および Visual Basic .NET に組み込まれたその他の開発支援をいくつか紹介します。

Visual Basic .NET アプリケーション ブロックの使用

マイクロソフトは、.NET Framework 用のアプリケーション ブロックをいくつか提供しています。アプリケーション ブロックは、.NET Framework 上にレイヤを追加することにより、特定の主要領域をカプセル化します。またアプリケーション ブロックは、アプリケーションの迅速な開発を支援し、好ましい慣例を促進します。アプリケーション ブロックは、.NET Framework で作業を行うための最善の方法をカプセル化します。アプリケーション ブロックを再利用することで、冗長なコードを減らすことができます。

アプリケーション ブロックは、Visual Studio プロジェクトとして配布される C# クラスおよび Visual Basic .NET クラスです。これらのクラスは、ASP.NET Web アプリケーションを含む、すべての .NET アプリケーションで使用できます。アプリケーション ブロックは、ソース コードとサンプル アプリケーションを含む便利で強力なツールです。これらを使用することで、アプリケーションの管理性、拡張性、および効率を向上させることができます。

以下の一覧に、利用可能なアプリケーション ブロックとツールキットを示します。

- Caching Application Block。Caching Application Block は、柔軟で拡張可能なキャッシュ機構を提供する、Enterprise Library のコンポーネントです。このアプリケーション ブロックは、クライアント側およびサーバー側の .NET 開発プロジェクトで使用できます。
- Configuration Application Block。Configuration Application Block は、さまざまなデータ ソースの構成情報を読み書きするアプリケーションを簡単に構築することを可能にする、Enterprise Library のコ

ンポーネントです。このアプリケーション ブロックには、アプリケーション構成データの編集と表示を支援するグラフィカル ツールも含まれています。

- Cryptography Application Block。Cryptography Application Block は、.NET アプリケーションに暗号機能を含める作業を簡素化する、Enterprise Library のコンポーネントです。このアプリケーション ブロックは、Data Protection API (DPAPI) と対称型の暗号化およびハッシュに対する単純なインターフェイスを提供します。また、Enterprise Library 構成ツールを使用してキー管理を簡素化します。
- Data Access Application Block。Data Access Application Block は、.NET アプリケーションでデータ アクセス レイヤを構築するときに作成、テスト、および維持する必要があるカスタム コードの量を軽減する、Enterprise Library のコンポーネントです。
- Exception Handling Application Block。Exception Handling Application Block は、アプリケーションの論理層で一貫した例外処理ポリシーを実装する作業を簡素化する、Enterprise Library のコンポーネントです。例外のログ記録や、例外の種類のラッピングまたは置換などのタスクを実行するために、例外ポリシーを構成することができます。
- Guidance Automation Toolkit。Guidance Automation Toolkit は、フレームワーク、コンポーネント、パターンなどの再利用可能なアセットについて、豊富かつ統合されたユーザー エクスペリエンスをアーキテクトが作成することを可能にする、Visual Studio 2005 の拡張機能です。完成したガイダンス パッケージは、開発者がアーキテクチャ ガイダンスに沿ったかたちでソリューションを構築することを支援するテンプレート、ウィザード、およびレシピで構成されます。
- Logging and Instrumentation Application Block。Logging and Instrumentation Application Block は、開発者がログ記録およびトレース呼び出しをアプリケーションに加えることを可能にする、Enterprise Library のコンポーネントです。ログ メッセージとトレース メッセージは、指定したデータ シンクに送ることができます。これには、イベント ログ、テキスト ファイル、および Windows Management Instrumentation (WMI) が含まれます。このアプリケーション ブロックは、Enterprise Instrumentation Framework (EIF) の後継にあたります。
- Security Application Block。Security Application Block は、Microsoft .NET Framework の機能の上に構築される Enterprise Library のコンポーネントで、認証や承認を実行したり、ロール メンバシップを確認したり、プロファイル情報にアクセスしたりするのを支援します。
- Smart Client Offline Application Block。Offline Application Block は、スマート クライアント ユーザーがオフラインで作業しているときでもシームレスな体験を得られるようにします。このアプリケーション ブロックは、ネットワークの存在または不在を検出するためのアプローチを提供します。また、要求されたデータをキャッシュしてアプリケーションがオフラインのときに機能できるようにし、アプリケーションが再びオンラインになったときにクライアント アプリケーションの状態とデータをサーバーと同期します。

- Updater Application Block バージョン 2.0。Updater Application Block は、中央の場所に配置されるクライアント アプリケーション アップデートを検出、ダウンロード、および適用するのために使用できる、.NET Framework コンポーネントです。Updater Application Block を使用すると、ユーザーがほとんど関与しないか、またはまったく関与せずに、スマート クライアント アプリケーションを最新の状態に保つことができます。また、Updater Application Block を拡張し、ファイルをダウンロードしたり、配置後の構成タスクを実行したりするためのカスタム クラスを使用することもできます。
- User Interface Process Application Block – バージョン 2.0。User Interface Process Application Block は、ユーザー インターフェイス プロセスを開発するための単純ながらも拡張可能なフレームワークを提供します。このアプリケーション ブロックは、制御フローと状態管理をユーザー インターフェイス レイヤから抽出し、ユーザー インターフェイス プロセス レイヤに移します。

アプリケーション ブロックの詳細とダウンロード方法については、MSDN の「patterns & practices Guidance: Application Blocks」を参照してください。

"My" ファサードの構築

Visual Basic .NET の My 機能は、アプリケーションとそのランタイム環境に関連する情報および既定のオブジェクト インスタンスへのアクセスを提供します。この情報は、IntelliSense を介して検出可能な形式で整理され、用途に応じて論理的に描写されます。

メモ : My ファサード機能は、Visual Studio 2005 でのみ使用できます。

My の最上位レベルのメンバは、オブジェクトとして公開されます。各オブジェクトは、共有メンバを持つ名前空間またはクラスのように機能し、一連の関連するメンバを公開します。

情報および一般に使用される機能へのアクセスを提供する中心的なオブジェクトは、以下のとおりです。

- My.Application。このオブジェクトは、現在のアプリケーションまたは DLL に関連するプロパティ、メソッド、およびイベントを提供します。このオブジェクトは、コンソール アプリケーションと Windows フォーム アプリケーションでのみ使用できます。
- My.Computer。このオブジェクトは、オーディオ、時計、キーボード、ファイル システムなどのコンピュータ コンポーネントを操作するためのプロパティを提供します。このオブジェクトは、すべての種類の Visual Basic .NET アプリケーションで使用できます。
- My.User。このオブジェクトは、現在のユーザーに関する情報へのアクセスを提供します。このオブジェクトは、すべての種類の Visual Basic .NET アプリケーションで使用できます。

以下のコード例は、My を使用して情報を取得する方法を示しています。

```
…
' アプリケーションの完全なコマンド ラインを示すメッセージ ボックスを表示します。
MsgBox(My.Application.CommandLineArgs)

' フォルダ内のサブ フォルダの一覧を取得します。
My.Computer.FileSystem.GetDirectories _
    (My.Computer.FileSystem.SpecialDirectories.MyDocuments, True,"*Logs*")
…
```

My 機能オブジェクトは、以下の機能も提供します。

1. My.Forms。このオブジェクトは、現在のプロジェクトで宣言されている各 Windows フォームのインスタンスにアクセスするために使用できるプロパティを提供します。このオブジェクトは、Windows フォーム アプリケーションでのみ使用できます。
2. My.Log。このオブジェクトは、アプリケーションのログ リスナにイベントと例外の情報を書き込むために使用できるプロパティとメソッドを提供します。このオブジェクトは、Web アプリケーションでのみ使用できます。
3. My.Request。このオブジェクトは、要求されたページの System.Web.HttpRequest オブジェクトを取得します。このオブジェクトは、Web アプリケーションでのみ使用できます。
4. My.Response。このオブジェクトは、System.Web.UI.Page に関連付けられている System.Web.HttpResponse オブジェクトを取得します。このオブジェクトを使用し、HTTP 応答データをクライアントに送信することができます。このオブジェクトは、応答に関する情報も含みます。このオブジェクトは、Web アプリケーションでのみ使用できます。
5. My.Resources。このオブジェクトは、アプリケーションのリソースにアクセスするために使用できるプロパティとクラスを提供します。このオブジェクトは、Web アプリケーションでは使用できません。
6. My.Settings。このオブジェクトは、アプリケーションの設定にアクセスするためのプロパティとメソッドを提供します。このオブジェクトは、Web アプリケーションでは使用できません。
7. My.WebServices。このオブジェクトは、現在のプロジェクトによって参照される各 XML Web サービスの単一のインスタンスの作成と、そのインスタンスへのアクセスに使用できるプロパティを提供します。このオブジェクトは、Windows フォーム アプリケーションでは使用できません。

My 機能の詳細については、MSDN の「Development with My」を参照してください。

Visual Studio .NET スニペットの構築

Visual Studio 2005 では、約 500 個の IntelliSense コード スニペットからなるコード ライブラリが Visual Basic .NET に含まれています。これらのコード スニペットは、アプリケーションにそのまま挿入することができます。各スニペットは、カスタム例外の作成、電子メール メッセージの送信、円の描画などの完全なプログラミング タスクを実行します。IntelliSence コード スニペットは、数回のマウス クリックでソース コードに挿入できます。

定義済みのコード スニペットを使用することに加えて、ビジネス ニーズに適したユーザー定義スニペットを作成することもできます。これらのスニペットはライブラリに追加し、必要なときに使用することができます。

IntelliSense コード スニペットを利用すると、以下のようなことを実施でき、それにより生産性を向上できます。

- コード サンプルの検索に費やす時間を削減する。
- よく知らない機能の使い方を学習するために必要な時間を削減する。
- 記述済みのコードを再利用する。

コード ライブラリを利用すると、以下のようなことを実施でき、それにより生産性を向上できます。

- 500 個の記述済みスニペット (またはタスク) のいずれか 1 個をコード エディタに挿入する。
- プロジェクトで再利用できる新しいタスクを作成する。
- 新しいタスクを作成し、ワークグループや同僚と共有する。
- タスクを編集する。
- 追加のタスクをサード パーティからダウンロードする。

これらのコード ブロックは Visual Studio 全体で使用できます。これらを追加するには、コード エディタのショートカット メニューを使用するか、または XML コード ファイルを Windows エクスプローラからソース コード ファイルにドラッグします。Visual Studio コード スニペットの詳細については、MSDN の「Introduction to IntelliSense Code Snippets」を参照してください。

モバイル アプリケーションと .NET Compact Framework

PDA (Personal Digital Assistant) や携帯電話などのモバイル コンピューティング デバイスの普及により、さまざまなモバイル デバイスで横断的に動作するアプリケーションを開発者が記述するニーズが高まっています。ここでは、マイクロソフトのモバイル テクノロジの概要を示し、モバイル アプリケーションを作成するための重要な環境として .NET Compact Framework を紹介します。

マイクロソフトのモバイル テクノロジの概要

モバイル デバイスの市場は、ここ数年で急速に拡大しています。この傾向は今後も続くことが予想されます。業界の専門家は、2008 年までに融合端末は 1 億台以上、携帯電話利用者は 20 億人以上になると予想しています[1]。これらの数字がモチベーションとなり、企業はモバイル デバイス用の新しいアプリケーションを開発し、既存のデスクトップ アプリケーションの新しいバージョンをモバイル市場に対応させています。

マイクロソフトは、モバイル コンピューティング向けに一連の広範なテクノロジを提供しています。これらのテクノロジを統合したり、採用したりすることにより、モバイル市場に革新的なソリューションを提供することができます。以下の一覧に、マイクロソフトのモバイル関連の取り組みに含まれる主要なテクノロジを示します[2]。

- Windows Mobile® 5.0。この開発キットは、さまざまなデバイスに対して一貫したプラットフォームを提供します。この開発キットには、モバイル アプリケーションを開発するための、ツール、API の共通セッ

ト、一様なデータ サービス、共通インストーラ、およびアプリケーション セキュリティ モデルが含まれています。

- Windows CE .NET。モバイル デバイス用に特別に仕立てられた、フル装備のオペレーティング システムです。
- .NET Compact Framework。.NET Framework から派生したもので、マネージ コード インフラストラクチャと汎用クラス ライブラリを提供します。これらは、PDA や携帯電話などのデバイス用のモバイル アプリケーションを開発するために必要な、最も共通の機能を提供します。
- SQL Server CE。エンタープライズ データ管理機能をモバイル デバイスに拡張するアプリケーションを迅速に開発するために使用される、コンパクトなリレーショナル データベースです。SQL Server と一貫する開発モデルおよびAPIを備えています。
- ASP.NET モバイル コントロール。かつて Microsoft Mobile Internet Toolkit (MMIT) と呼ばれていたこのテクノロジは、.NET Framework と Visual Studio .NET の機能を拡張し、ASP.NET でさまざまなモバイル デバイスにマークアップ コンテンツを提供できるようにすることによって、モバイル Web アプリケーションを構築します。
- Windows XP Embedded。Windows XP Embedded は、Windows NT Embedded 4.0 の後継です。Windows XP Professional と同じバイナリ ファイルに基づいた Windows XP Embedded を使用すると、信頼性が高く、完全な機能を備えた接続デバイスを迅速に開発できます。
- Microsoft Windows XP Tablet PC Edition。Microsoft Windows XP Tablet PC Edition オペレーティング システムは、PC の進化における次の段階と考えられている Tablet PC で実行されます。Tablet PC は、今日のノート型コンピュータのすべてのパフォーマンスと機能を、超軽量な形態で提供します。これには、標準的な Windows アプリケーションの完全なバージョンを実行する能力が含まれます。

.NET Compact Framework の概要

.NET Compact Framework は、リソースが限られたデバイス用に機能絞り込みを行って最適化された、.NET Framework の再設計版です。.NET Compact Framework は、.NET Framework の完全版で利用できる機能とクラスのサブセットを提供します。.NET Compact Framework の最も重要な機能は、豊富なクラス ライブラリ、自動メモリ管理、および CPU への非依存性です。

.NET Compact Framework を使用すると、開発者は情報の表示、収集、処理、および転送を行うアプリケーションを作成できます。これらのアプリケーションで使用するデータは、ローカルでも、リモートでも、ローカルとリモートの組み合わせでもかまいません。

モバイル アプリケーションの開発は、.NET Compact Framework によって大幅に簡素化されます。.NET Compact Framework は、開発者が通常の Windows アプリケーションと同じような方法でアプリケーションを作成し、それらをさまざまな種類のデバイスに配置することを可能にする、統一されたインフラストラクチャを提

供します。そのため Visual Studio .NET 開発者は、モバイル アプリケーションをさまざまなデバイスに移植できます。これらのアプリケーションは、Visual C# .NET または Visual Basic .NET で構築できます。

.NET Compact Framework には主要なコンポーネントが 2 つあります。それは共通言語ランタイム (CLR) と .NET Compact Framework クラス ライブラリです。CLR は、コードの実行を管理する役割を担い、メモリ管理、スレッド管理、安全管理などのコア サービスを提供します。.NET Compact Framework は、新しいアプリケーションの基盤として使用されるクラスのコレクションです。

以下では、このフレームワークのいくつかの機能に注目します。.NET Compact Framework のさらなる詳細については、MSDN の Smart Client Developer Center の「.NET Compact Framework」を参照してください。

含まれているコンポーネント

.NET Compact Framework には、System.Windows.Forms 名前空間と System.Drawing 名前空間で定義されているクラスのサブセットが含まれています。これらのクラスを使用すると、モバイル アプリケーションのユーザー インターフェイスをすばやく構築できます。

重要なのは、デスクトップ アプリケーションのモバイル版を作成すると、ユーザー インターフェイス動作の一部が変更されたり、制限されたりするということです。ただし、eMbedded Visual Basic などの従来のモバイル アプリケーション開発環境で作成したアプリケーションをアップグレードすると、元の機能のほとんどが新しい .NET Compact Framework でサポートされて改善されます。

Windows フォームのコンパクト版には、フォームに対するサポートと、.NET Framework に含まれているコントロールのほとんどに対するサポートが含まれています。サード パーティ コンポーネント、ビットマップ、およびメニューを含めることもできます。以下の一覧に、.NET Compact Framework に含まれているコントロールを示します。

- Button
- CheckBox
- ComboBox
- Control
- ContextMenu
- Cursor および Cursors
- DataGrid
- DomainUpDown
- Form
- HScrollBar
- ImageList
- InputPanel
- Label
- ListBox

- ListView
- MainMenu
- Message Window (コンパクト フレームワークに固有)
- NumericUpDown
- OpenFileDialog
- Panel
- PictureBox
- ProgressBar
- RadioButton
- SaveFileDialog
- StatusBar
- TabControl
- TabPage
- TextBox
- Timer
- ToolBar
- TrackBar
- TreeView
- VScrollBar

モバイル デバイスに存在するサイズおよびパフォーマンスの制約に .NET Framework を適応させるため、プロパティ、メソッド、およびイベントの一部は、このフレームワークのコンパクト版から削除されています。既存のコントロールからの継承により、追加の機能を定義することが可能です。

.NET Compact Framework は、.NET Framework と同じ追加機能や、モバイル デバイス用に特別に設計された追加機能を含む名前空間を提供します。これらの名前空間は、データ アクセス、XML サービス、Web サービス、GDI、IrDA、および Bluetooth に対するサポートを提供します。

削除された機能

.NET Framework から .NET Compact Framework を作成するにあたっては、モバイル デバイスのリソース制約への適合を行いました。アプリケーションをモバイル プラットフォームに移植するときは、.NET Compact Framework を拡張し、.NET Framework から削除された追加機能を構築することが可能です。特定のアプリケーションのニーズに応じて .NET Compact Framework クラスを拡張するときは、モバイル デバイスで利用可能な限られたリソースの使用を最適化します。さらに、さまざまな開発コミュニティから入手した他のコンポーネントを、移植したアプリケーションで使用することもできます。サード パーティ コンポーネントの詳細については、IntelliProg の Web サイトを参照してください。

以下の一覧に、.NET Compact Framework に含まれていない主な機能を示します。

- **メソッド オーバーロード**。.NET Framework で使用できるオーバーロードされたメソッド定義の多くは、.NET Compact Framework のサイズを減らすために省略されました。
- **コントロール**。一部のコントロールは .NET Compact Framework から省略されました。これらのコントロールのいくつかの機能は、通常はモバイル デバイスでは不要です。なくなったコントロールの代わりを補うものとして、Windows CE API からアクセスできるシステム コンポーネントに置き換えることが可能です。
- **XML 機能**。XPath 名前空間と XSLT (Extensible Stylesheet Language Transformation) コンポーネントによって提供される解析機能および検索機能は省略されました。
- **データベース サポート**。データ アクセス コンポーネントは減らされ、SQL Server CE コンポーネントで置き換えられました。eMbedded Visual Basic と異なり、.NET Compact Framework はローカル データ ストア (CEDB または Pocket Access とも呼ばれます) へのアクセスをサポートしません。
- バイナリ シリアル化。BinaryFormatter クラスと SoapFormatter クラスは省略されました。
- **Windows レジストリへのアクセス**。Microsoft.Win32.Registry 名前空間は削除されました。この名前空間は、Windows CE API の対応する機能で置き換える必要があります。
- **セキュリティ**。ロール ベース セキュリティの機能は、.NET Compact Framework から削除されました。さらに、アンマネージ コードへのセキュリティで保護されたアクセスの検証は省略されました。
- **XML Web サービス**。クッキーのサポートは省略され、暗号機能は制限されています。
- **印刷**。印刷のサポートは省略されました。IR ポート通信およびデスクトップ アプリケーションとの対話を、データ印刷のための代替方法として使用できます。
- GDI+。Windows CE は GDI+ をネイティブにサポートしないため、この機能は省略されました。
- **リモート処理**。この機能は .NET Compact Framework に含まれていません。

既定のプロジェクト設定

新しいモバイル アプリケーションを作成するときや、アプリケーションを別のモバイル環境またはデスクトップ環境から移植するときは、モバイル Visual Basic .NET アプリケーションに対して定義される既定のプロジェクト設定を知っておくことが重要です。

Visual Studio .NET は、すべての種類のアプリケーションを構築するための一貫した均一の開発環境を提供します。これには、モバイル アプリケーションも含まれます。

Visual Studio .NET IDE 内では、モバイル Visual Basic .NET アプリケーションをデスクトップ Visual Basic .NET アプリケーションと区別するのはほぼ不可能です。全般的なプロジェクト設定 (メイン メニューの **[プロジェクト]** をクリックし、**[プロパティ]** をクリックすることによってアクセス可能) は同じように構成され、似たような値を持っています。モバイル Visual Basic .NET プロジェクト設定に含まれている唯一の新しいセクションは、**[デバイス]** セクション ([プロジェクトの設定] ウィンドウの **[共通プロパティ]** から **[デバイス]** タブでアクセス可能) です。この新しいセクションは、新しいアプリケーションまたは移植したアプリケーションの配置先プラットフォームを示します。このセクションを使用すると、モバイル アプリケーションをさまざまなデバイスに移植できます。また、アプリケーションをさまざまなデバイス エミュレータでテストすることもできます。

モバイル アプリケーションとデスクトップ アプリケーションのもう 1 つの重要な違いは、すべての新しいプロジェクトに既定で追加される参照のグループです。基本的なアプリケーションを機能させるために必要な最小限のアセンブリを提供するため、デスクトップ アプリケーションの方が含まれる参照は多くなります。モバイル アプリケーションは使用可能なリソースの点で制約が厳しいため、新しいモバイル アプリケーション プロジェクトに既定で追加されるコンポーネントは少なくなります。また、使用可能なコンポーネントも概して少なくなります。使用可能なコンポーネントの一覧については、この前の「含まれているコンポーネント」を参照してください。

モバイル アプリケーション用の追加の参照設定をプロジェクトに含めることができます。

▶ モバイル アプリケーション用の参照を追加するには

1. ソリューション エクスプローラで **[参照設定]** フォルダを右クリックし、**[参照の追加]** をクリックします。**[参照の追加]** ダイアログ ボックスが表示されます。
2. **[参照の追加]** ダイアログ ボックスで、モバイル アプリケーション用の目的の参照をプロジェクトに追加します。

eMbedded Visual Basic からの移植

Microsoft eMbedded Visual Basic は、Visual Basic の縮小版です。eMbedded Visual Basic は、Visual Basic 開発者がこの言語の知識を生かして Windows CE ベースのアプリケーションを開発できるように設計されています。それより新しいモバイル デバイス、つまり Windows Mobile ベースの Smartphone や Pocket PC 2003 ベースのデバイスなどではサポートされていません。代わりに、マイクロソフトはこれらのデバイス用のモバイル アプリケーションや組み込みアプリケーションを、.NET Compact Framework を使用して構築することを推奨しています。これには、eMbedded Visual Basic アプリケーションを .NET Compact Framework 用の Visual Basice .NET に移植する作業が含まれます。

アプリケーションを eMbedded Visual Basic から .NET Compact Framework 用の Visual Basice .NET に移植すると、以下のような大きな利点が得られます。

- **パフォーマンスの向上**。Visual Basic .NET コードは、eMbedded Visual Basic コードのように解釈されるのではなく、コンパイルされます。そのため、アプリケーションのパフォーマンスが向上します。
- **広範なクラス ライブラリ**。.NET Compact Framework は、モバイル アプリケーションで実行される一般的なタスクのほとんどに対応するクラスを提供します。この環境では、eMbedded Visual Basic よりも多くのコンポーネントとクラスを使用できます。

- **改善された開発ツール**。Visual Studio .NET は、開発者の生産性を向上させるツールと機能により、これまでにない開発環境を提供します。デスクトップ アプリケーションの開発とモバイル アプリケーションの開発に同じ IDE を使用するため、開発者は種類の異なるアプリケーションを開発するために、複数の IDE に慣れ親しむ必要はありません。
- **改善された言語**。Visual Basic .NET は、オブジェクト指向プログラミング、例外処理、厳密な型指定など、プログラミング言語の進歩を取り入れるように再設計された言語です。

eMbedded Visual C++ など、その他の eMbedded 言語に基づくアプリケーションを移植するときは、アプリケーションについてのより深い分析と理解が必要です。高いパフォーマンス、最小限の作業セット、および低レベルのデバイス制御が最優先課題なら、開発者は、eMbedded Visual C++ を引き続き使用する必要があります。開発期間短縮と一貫したプログラミング モデルが主要な関心事なら、Visual Studio .NET と .NET Compact Framework を使用するのが最良の選択肢です。

モバイル アプリケーションを自動的にアップグレードするツールはありませんが、他のツールを使用して、eMbedded Visual Basic アプリケーションを .NET Compact Framework に移植するプロセスを加速することができます。eMbedded Visual Basic 言語は Visual Basic 6.0 から派生し、VBScript (ASP ページで使用される言語) と似ています。したがって、このガイドで説明した自動 Visual Basic アップグレード ツールを使用して、モバイル アプリケーションの一部をアップグレードすることができます。このアプローチのガイドラインとして、以下の一般的な手順を使用してください。この手順は、移植するアプリケーションの各部分に適用できます。

▶ モバイル アプリケーションの一部をアップグレードするには

1. アプリケーションまたはアプリケーションの一部を、Visual Basic 6.0 で機能するように修正します。eMbedded Visual Basic は Visual Basic の機能のサブセットを使用するため、アプリケーションの Visual Basic 6.0 版では、通常使用できるすべての機能のうちのいくつかを省略したサブセットを使用することになります。
2. Visual Basic アップグレード ウィザードを使用し、Visual Basic 6.0 アプリケーションを Visual Basic .NET にアップグレードします。
3. Visual Studio .NET で新しい Visual Basic .NET モバイル アプリケーションを作成し、前の手順で取得した Visual Basic .NET ファイルを含めます。
4. 解決されていないアップグレードの問題をすべて修正し、必要な調整を加えて、アプリケーションが機能するようにします。
5. 新しいアプリケーションをテストします。

eMbedded Visual Basic ベースのアプリケーションを Visual Basic .NET に手動で移植する方法の詳細については、MSDN の「Moving from eMbedded Visual Basic to Visual Basic .NET」を参照してください。

デスクトップ アプリケーションのモバイル版の作成

デスクトップ アプリケーションに基づく新しいモバイル アプリケーションを作成することに決めたときは、元のアプリケーションのいくつかの要素を移植したり見直したりする必要があります。これらの要素には、ユーザー インターフェイスやデータ アクセス、さらにはターゲット デバイスの低レベルの制御も含まれます。ここでは、デスクトップ アプリケーションをモバイル アプリケーションに移植するために必要な手順の概要を示します。また、アプリケーションを移植するために使用できるツールをいくつか紹介し、移植を容易に行うための手順について詳しく説明します。

移植するアプリケーションとして、以下の 2 つのアプリケーション タイプのいずれかを選ぶことができます。

- **モバイル アプリケーション**。元のアプリケーションの配置先プラットフォームがモバイル デバイスの場合、モバイル環境の制約は既に考慮されています。これにより、必要な作業の大半は、特定の変換に関する問題の解決に集約されるため、.NET Compact Framework へのアップグレードが容易になります。それ以外の場合は、アプリケーションを再設計し、モバイル デバイスでの使用に適したものにする必要があります。
- **デスクトップ アプリケーション**。デスクトップ アプリケーションのモバイル版を作成することが目的の場合は、まず予備的な適合性の調査を行う必要があります。おそらく元のアプリケーションの機能のかなりの部分は、大幅な修正を要するか、場合によってはモバイル版から削除しなければならなくなります。ほとんどの場合、この種のモバイル アプリケーションは、デスクトップ アプリケーションを補完するものとして機能します。これらのアプリケーションは、モバイル コンピューティングに必要十分な、元の機能のサブセットを含むことになります。

すべての Visual Basic 移植プロジェクトと同じく、アプリケーションをモバイル プラットフォームに移植するには、アプリケーションの準備に始まり、テストとデバッグに終わる、一連の手順を実行する必要があります。推奨する移植手順については、第 5 章「Visual Basic のアップグレード プロセス」を参照してください。この章ではアップグレード プロセスについて説明しますが、その内容は移植プロジェクトにも同じように当てはまります。

ここでは、アプリケーションをコンパイルしてモバイル プラットフォームで実行できるように移植するために必要な初期手順を示します。低レベルのアップグレード、およびモバイル アプリケーションのテストとデバッグについての詳しい情報は、本書の範疇を越えています。ただし、これらのプロセスに関する詳細情報がある場所については、後にこの章で示します。

以下では、Visual Basic .NET アプリケーションを .NET Compact Framework に移植するために使用できるガイドラインを示します。

ユーザー インターフェイスの移植

アプリケーションをモバイル プラットフォームにアップグレードするときに大幅な変更が必要になることが予想される要素は、そのアプリケーションのユーザー インターフェイスです。モバイル デバイス用のビジュアル インターフェイスを構築するために使用できるコンポーネントは、デスクトップ アプリケーションで使用できるコンポーネントに比べて、はるかに小さくなります。

アプリケーションのユーザー インターフェイスは、アプリケーションをモバイル プラットフォームにアップグレードするときに、大幅な変更が必要になる可能性があります。モバイル デバイス用のビジュアル インターフェイスを構築するために使用できるコンポーネントは、デスクトップ アプリケーションで使用できるコンポーネントよりもはるかに小さいサイズになります。これは、モバイル アプリケーションがデスクトップ版と視覚的に異なるものになることを暗示しています。モバイル アプリケーションの動作も異なるものになる可能性があります。これは、アプリケーションの機能の一部を、異なるビジュアル要素を使用して実装する必要があるからです。さらに、もう 1 つの視覚的な違いとして、モバイル デバイスのディスプレイのサイズがあります。Pocket PC や Windows CE デバイスの小さなサイズを考えると、利用可能なスペースを効率的に使用するように、ユーザー インターフェイスを見直して調整する必要があります。一般に、モバイル アプリケーションの外観とスタイルは、元のデスクトップ アプリケーションと異なるものになります。

移植プロセス

以下のプロセスは、Visual Basic .NET アプリケーションのフォームをモバイル プラットフォームで機能するように移植するために必要な手順を示しています。このプロセスは、アプリケーションのユーザー インターフェイスと基本機能を変換するために加える必要がある典型的な調整を示しています。

図 1 に、変換前のフォームの例を示します。

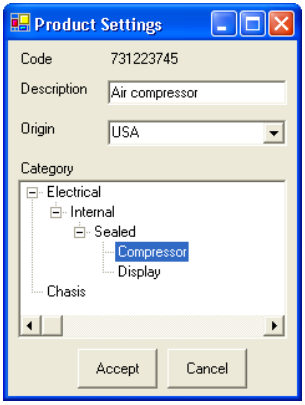

図 1

モバイル デバイスで使用するために移植するサンプル Visual Basic .NET フォーム

このフォームは Form2.vb という名前のファイルに含まれており、いくつかのサンプル製品設定を表しています。ユーザーはこれらの設定を変更し、データベースで更新することができます。以下の手順を実行することで、アプリケーションの最初のバージョンを作成することができます。そうして得られたフォームは、モバイル プラットフォームで実行できます。

この例では、元のアプリケーション ファイルは C:\MobileAppTest ディレクトリにあります。必要に応じて、このディレクトリをシステム上の実際のファイルの場所に置き換えてください。

▶ フォームをコンパクト フレームワーク版に変換するには

1. 移植するアプリケーションを作業ディレクトリにコピーします。.NET Compact Framework で機能させるため、新しいファイルのいくつかに修正を加える必要があります。したがって、元のコードを保存しておくために、作業にはアプリケーションのコピーを使用することが最善です。この例では、元のアプリケーション ファイルを C:\MobileAppTest ディレクトリにコピーしています。
2. Visual Studio .NET を開きます。
3. 以下の手順に従って、Visual Studio .NET で新しい Visual Basic .NET モバイル アプリケーションを作成します。
   a. [**ファイル**] メニューの [**新規作成**] をクリックし、[**プロジェクト**] をクリックします。
   b. [**テンプレート**] リストで、[Smart Device Application] をクリックします。[Smart Device Application Wizard] ウィンドウが表示されます。このウィザードを使用し、ターゲットにするプラットフォームと、作成するプロジェクトの種類を選択します。この例では、プラットフォームとして [Windows CE] を選択し、プロジェクトの種類として [Windows **アプリケーション**] を選択します。
4. 新しいプロジェクトが Visual Studio .NET に表示されます。既定では、Form1 という名前の新しいフォームが含まれています。ここでは元のアプリケーションのフォームをモバイル プロジェクトにインポートするので、Form1 は不要です。以下の手順を実行し、Form1 を削除します。
   a. ソリューション エクスプローラで [Form1] を右クリックし、[**削除**] をクリックします。
   b. [Delete Confirmation] ダイアログ ボックスで [**はい**] をクリックし、Form1 を削除します。
5. モバイル アプリケーション プロジェクトに変換するファイルを追加します。以下の手順に従って、ファイルを変換します。
   a. ソリューション エクスプローラでプロジェクト フォルダを右クリックし、ポップアップ メニューで [**追加**] をクリックして、[**既存項目の追加**] をクリックします。[**既存項目の追加**] ダイアログ ボックスが表示されます。
   b. 変換するファイルを探し、そのファイルを選択します。この例では、C:\MobileAppTest にある Form2.vb ファイルを選択します。
6. [**タスク一覧**] にタスクの一覧が表示されます。この一覧には、存在しないプロパティ、メソッド、およびイベントによって引き起こされる問題に関連するタスクが含まれています。これらの問題を解決するための一般的な方法は、各タスクを選択し、そのタスクが参照するソース コードを調べることです。そのうえで、プロパティ、メソッド、またはイベントが必要か否かに応じて、コードを修正するか、または削除します。表 1 に、ユーザー インターフェイス機能に関連する一般的な問題とソリューションを示します。

表 1: .NET フォームを Compact Framework フォームに変換するときのユーザー インターフェイスの問題とソリューション

エラーの説明	ソリューション
SuspendLayout は **SmartDeviceApplication1.Form2** のメンバではありません。	ステートメントを削除します。
Name は **System.Windows.Forms.*SpecificControl*** のメンバではありません。**System.Windows.Forms.Control** から派生したすべてのコントロールに当てはまります。	ステートメントを削除します。動的に作成されたコントロールを識別することがどうしても必要な場合は、変数を維持して目的のコントロールを参照し、それを比較で使用して特定のコントロールを見つけることができます。
TabIndex は **System.Windows.Forms.SpecificControl** のメンバではありません。**System.Windows.Forms.Control** から派生したすべてのコントロールに当てはまります。	ステートメントを削除します。
この引数の数を受け付ける **New** がないため、オーバーロードの解決に失敗しました。	使用可能なメソッド オーバーロードのいずれか 1 つを使用して、対応するコントロール コンストラクタの呼び出しを変更します。通常は、フォームの **Load** イベント ハンドラまたはその他の初期化コードで適切なコントロールのメソッドまたはプロパティにアクセスすることにより、元の機能を複製することができます。
Method または **Property** は **System.Windows.Forms.SpecificControl** のメンバではありません。	**InitializeComponent** メソッドに含まれているコントロールの初期化では、サポートされていないメソッドまたはプロパティにアクセスできません。ほとんどの場合は、コントロールの初期化後に他の使用可能なメソッドにアクセスすることにより、サポートされていないメンバの機能を複製することができます。
ResumeLayout は **SmartDeviceApplication1.Form2** のメンバではありません。	ステートメントを削除します。

7. プロジェクトの Startup オブジェクトを必要に応じて変更します。次の手順に従ってください。
   a. Visual Studio の**メイン** メニューで [**プロジェクト**] をクリックし、[SmartDeviceApplication1] プロパティをクリックします。
   b. [**共通プロパティ**] フォルダの [**全般**] タブをクリックします。
   c. Startup オブジェクト フィールドで、スタートアップ フォームまたはメソッドの名前を選択します。この例では、[Form2] を選択します。

8. .NET Compact Framework で使用するリソース ファイルを修正します。.NET Compact Framework は異なるクラスを使用してリソース ファイルを読み取ります。リソース ファイルを修正するには、Form2.resx ファイルを変更し、適切な .NET Compact Framework リソース リーダー クラスでクラス名を置き換えます。この変更はテキスト エディタまたは Visual Studio .NET IDE を使用して加えることができます。元のリソース ファイルを以下に示します。

```
<resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms,
    Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
```

このファイルの内容に以下の変更を加えます。

```
<resheader name="reader">
    <value>System.Windows.Forms.Design.CFResXResourceReader, System.CF.Design,
    Version=7.0.5000.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a</value>
</resheader>
```

9. 同じような変更を Form2.resx ファイルに加え、リソース ライター クラスを修正します。以下のコードは元のファイルを示しています。

```
<resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=1.0.5000.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
```

このファイルに以下の変更を加えます。

```
<resheader name="writer">
    <value>System.Windows.Forms.Design.CFResXResourceWriter, System.CF.Design, Version=7.0.5000.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a</value>
</resheader>
```

10. .NET Framework と .NET Compact Framework ではその他のクラスも異なるため、リソース ファイルで指定されているその他の型の修正が必要になることもあります。これらのクラスは、.NET Compact Framework の対応するクラスに変更する必要があります。たとえば、System.Drawing.Size クラスを変更する必要があります。このクラスは .resx ファイルで頻繁に使用されます。以下のコードは元の .resx ファイルを示しています。

```
"System.Drawing.Size, System.Drawing, Version=1.0.5000.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"
```

この行を、以下のコードに示すように変更する必要があります。

```
"System.Drawing.Size, System.CF.Drawing, Version=7.0.5000.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"
```

11. アプリケーションをコンパイルします。それには、**メイン** メニューで **[ビルド]** をクリックし、**[ソリューションのビルド]** をクリックします。前の 2 つの手順を使用し、リソースファイルのその他のエラーを修正します。

これでアプリケーションを配置し、対応するモバイル デバイスまたは適切なエミュレータで実行することができます。図 2 に、変換後のフォームを Windows CE エミュレータで実行した画面を示します。

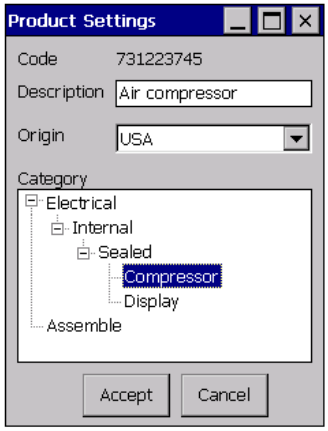

図 2

Windows CE *エミュレータで実行した移植後のアプリケーション*

ソース言語とターゲット言語が Visual Basic .NET なので、アプリケーションのソースコードのほとんどは問題なくコンパイルされて実行されます。問題の大半は、アプリケーションでデータ アクセスや GDI+ などのサポート外のライブラリ コンポーネントを使用するときに生じます。以下では、データ アクセス機能を実装するためのアドバイスを示します。アプリケーションのサポート外の機能にコメントを付け、後で機能を再設計または再実装するための出発点として使用できる作業基盤を作成することができます。

サーバー アプリケーションとの同期

モバイル アプリケーションを他のアプリケーションと統合するには、以下のパターンのいずれかを使用します。

- **デバイス データ対サーバー データ**。このパターンでは、モバイル デバイス上のデータベースがエンタープライズ データベース サーバーと直接同期されます。この方法は、同期するデータに関し、追加の検証または変換が不要な場合に使用できます。この種の同期は、SQL Server CE .NET のリモート データ アクセスとマージ レプリケーションを使用して実現できます。

- **デバイス ロジック対サーバー ロジック**。このパターンでは、モバイル アプリケーションがサーバー コンピュータ上のアプリケーションに接続します。ほとんどの同期にはビジネス ロジックが必要なので、これが最も頻繁に使用される方法です。この種の同期は、Web サービスを使用して実現できます。
- **デバイス ロジック対サーバー データ**。このパターンでは、モバイル アプリケーションはデータ リポジトリに直接接続し、アプリケーション ロジックに基づいて必要なデータを取得することができます。この種の同期は、SQL Server .NET の Data Provider を使用して実現できます。

eMbedded Visual Basic または Visual Basic .NET のデスクトップ アプリケーションを .NET Compact Framework に移植するときは、新しいプラットフォームによって提供される広範な通信機能と、サード パーティによって提供される追加のオプションに注目する必要があります。

同期機能を実装するとき、最も一般的な選択肢の 1 つは、SQL Server CE のマージ レプリケーションを使用することです。このテクノロジを使用すると、SQL Server CE サブスクリプション データベースに格納されたデータを、データの発行者と同期させることができます。モバイル デバイスがエンタープライズ ネットワークに永続的に接続しておらず、都合のよいときに同期を実行する必要がある場合は、このオプションを使用します。ほとんどの場合、eMbedded Visual Basic アプリケーションを使用するときは、これらの条件を満たすことができます。

.NET Compact Framework の SqlCeReplication クラス (System.Data.SqlServerCe 名前空間に含まれています) は、SQL Server CE に対応するデータ プロバイダ コンポーネントです。このクラスを使用すると、開発者は、発行されたデータベースとのマージ レプリケーションをプログラム的にアクティブにすることができます。データベースの発行の詳細については、MSDN の「Replication with SQL Server 2000 Windows CE Edition」を参照してください。

以下の例は、SqlCeReplication クラスを使用して、モバイル アプリケーションのデータを発行されたデータベースと同期させる方法を示しています。

```
…
Sub SynchronizeDB()
    Dim rep As New SqlCeReplication( _
        "http://myserver/pubsmr/sscesa20.dll", _
        "CEDBTest", "ceDBtest", _
        "MYSERVER", _
        "pubs", _
        "pubs", _
        "Testing", _
        " Data Source=\My Documents\PubsMR.sdf")
    Try
        ' データベース ファイルが既に存在するかどうかを確認します。
        If (System.IO.File.Exists("\My Documents\PubsMR.sdf")) Then
            ' 新しいサブスクリプションを追加し、ローカル データベース ファイルを作成します。
            rep.AddSubscription(AddOption.CreateDatabase)
        End If
        rep.Synchronize()
    Catch ex As SqlCeException
```

```
            MessageBox.Show(ex.Message)
        Finally
            rep.Dispose()
            MessageBox.Show("Replication complete")
        End Try
    End Sub
    …
```

SqlCeReplication クラスの Synchronize メソッドは、最初に呼び出されたときに、ローカルの SQL Server CE データベース用にデータの初期スナップショットを取得します。データを初期化し、サブスクリプションをセットアップした後は、Synchronize を新たに呼び出すと、データに適用された変更だけが送受信されます。

モバイル データベースとサブスクリプションを作成した後は、System.Data.SqlServerCe 名前空間に含まれているクラスを使用して、データの更新、挿入、および削除を行うことができます。すべての変更は、Synchronize メソッドを次回呼び出したときに適用されます。

eMbedded Visual Basic アプリケーションが Pocket PC プラットフォームをターゲットとしている場合は、Pocket PC プラットフォームの一部である Pocket Access データベースへのアクセスを可能にするサード パーティ コンポーネントを検討することが重要です。eMbedded Visual Basic で記述されたアプリケーションは、ADO の Windows CE 実装 (ADOCE) を使用することにより、Pocket Access データベースにアクセスすることができます。ただし、この COM コンポーネントに .NET Compact Framework からアクセスするのは容易ではなく、サード パーティ コンポーネントを代替として使用する必要があります。追加のデータ アクセス コンポーネントの詳細については、MSDN の「Pocket Access and the .NET Compact Framework」を参照してください。

詳細情報

アプリケーション ブロックの詳細とダウンロード方法については、MSDN の「patterns & practices Guidance: Application Blocks」を参照してください。
URL は http://msdn.microsoft.com/practices/guidetype/AppBlocks/default.aspx です。

My 機能の詳細については、MSDN の「Development with My」を参照してください。
URL は http://msdn2.microsoft.com/library/5btzf5yk(en-us,vs.80).aspx です。

Visual Studio コード スニペットの詳細については、MSDN の「Introduction to IntelliSense Code Snippets」を参照してください。
URL は http://msdn2.microsoft.com/library/18yz4be4(en-us,vs.80).aspx です。

.NET Compact Framework の詳細については、MSDN の Smart Client Developer Center にある「.NET Compact Framework」を参照してください。
URL は http://msdn.microsoft.com/mobility/prodtechinfo/devtools/netcf/ です。

サードパーティコンポーネントの詳細については、IntelliProg の Web サイトを参照してください。
URL は http://www.intelliprog.com/ です。

eMbedded Visual Basic ベースのアプリケーションを Visual Basic .NET に手動で移植する方法の詳細については、MSDN の「Moving from eMbedded Visual Basic to Visual Basic .NET」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnppcgen/html/fromemb.asp です。

データベースの発行の詳細については、MSDN の「Replication with SQL Server 2000 Windows CE Edition」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/replsql/replimpl_6pet.asp です。

追加のデータアクセスコンポーネントの詳細については、MSDN の「Pocket Access and the .NET Compact Framework」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnroad/html/road10222003.asp です。

脚注

[1] Enterprise Wireless Email Market, 2004-2008:
http://www.gii.co.jp/english/rd25765_wireless_email.html

[2] Understanding Mobile & Embedded:
http://msdn.microsoft.com/mobility/understanding/

付録

C

ASP のアップグレードに関する概要

ここ数年、Web 開発者は Active Server Pages (ASP) テクノロジを使用して、Web サイトや Web アプリケーションの開発を行ってきました。Microsoft .NET Framework は、開発者が高度な Web アプリケーションを構築するための新しいインフラストラクチャを提供します。この新しいインフラストラクチャは ASP.NET と呼ばれます。

ASP.NET は単なる新バージョンの ASP ではありません。.NET Framework をベースとした信頼性と拡張性に優れた Web アプリケーションを構築するための堅牢なインフラストラクチャを提供できるようアーキテクチャが一新されています。

ASP アプリケーションを ASP.NET に移植する際には、既存の ASP アプリケーションに ASP.NET の新しい機能を組み込む作業にどのくらいの時間をかけるかを決定する必要があります。選択肢の 1 つは、まず ASP to ASP.NET Migration Assistant を使用して ASP ページを ASP.NET に自動的に変換し、次に手動で変更を加えて作業を完了することです。もう 1 つの選択肢は、アプリケーションのデザインを最初からやり直し、ASP.NET Web コントロール、Microsoft ADO.NET、Microsoft .NET Framework クラスなど .NET の多数の新機能を利用することです。後者の方法を使用すると、ASP.NET ページがより読みやすく、管理しやすく、機能豊富になりますが、単に移行アシスタントを使用する場合と比べてアップグレードの完了までに多くの時間と労力を必要とします。

既存の ASP ページを ASP.NET ページに変換する作業は、ファイル名の拡張子を .asp から .aspx に変更するだけの単純なものではありません。ASP アプリケーションをアップグレードする際に考慮すべき重要な点として以下が挙げられます。

- **API コアの変更**。ASP の API コアには、少数の組み込みオブジェクト (Request、Response、Server など) とその関連メソッドがあります。いくつかの変更点を除けば、これらの API は ASP.NET でも正しく機能します。

- **構造の変更**。構造の変更は、Active Server Pages のレイアウトやコーディング スタイルに影響します。コードが ASP.NET で確実に機能するようにするためには、いくつかの構造の変更を認識し、これらの変更に対処する必要があります。
- Visual Basic **言語の変更**。Visual Basic Scripting Edition VBScript と Microsoft Visual Basic .NET スクリプトの間では、アップグレード時に考慮が必要ないくつかの変更があります。
- COM **関連の変更**。.NET Framework および ASP.NET の導入による COM の変更はありません。ただし、COM オブジェクト自体や、ASP.NET から COM オブジェクトを使用した場合の動作について考慮する必要がないというわけではありません。考慮すべき基本的な問題がいくつかあります。
- **アプリケーションの構成の変更**。ASP では、Web アプリケーションの構成情報はすべてシステム レジストリと IIS メタベースに格納されますが、ASP.NET では、各アプリケーションに固有の構成ファイルがあります。
- **状態管理の問題**。ASP.NET でも、組み込みオブジェクトの Session または Application を使用して状態情報を格納できます。追加のメリットとして、ASP.NET では状態を格納するためのオプションがいくつか追加されています。
- **データ アクセス**。アップグレード作業の際に考慮すべきもう 1 つの重要な点は、データ アクセスです。ADO.NET では、データの格納およびアクセスのための強力な新機能が用意されています。

ASP と ASP.NET は同じ Web サーバー上に共存できます。つまり、Web サイト、またはサイト内の Web アプリケーションに ASP.NET ページと ASP ページの両方を格納できます。これは、頻繁に変更される大規模なサイトを部分ごとに ASP.NET に移行する場合に非常に役立ちます。また、ASP.NET への移行作業を長期的な投資として進める場合は、この機会を利用してアプリケーションのアーキテクチャやデザインをできるだけ多く改良することもできます。詳細については、MSDN の「Migrating to ASP.NET: Key Considerations」を参照してください。

メモ： ASP.NET は、Microsoft Windows 2000、Microsoft Windows 2003、および Microsoft Windows XP Professional の各オペレーティング システムでサポートされています。ASP.NET アプリケーションは、Microsoft Windows NT ではサポートされていません。詳細については、MSDN の Microsoft Visual Studio Developer Center の「System Requirements for Visual Studio .NET 2003」を参照してください。

同じ Web サーバーから ASP ページと ASP.NET ページの両方にアクセスできるため、既存の ASP ページを ASP.NET に変換する必要はありません。ただし、ASP.NET に変換することにより、多くのメリットを得ることができます。その一部として以下の点が挙げられます。

- **パフォーマンスの向上**。マイクロソフトが行ったテストの結果によると、ASP.NET アプリケーションが 1 秒間に処理できる要求の数は、従来の ASP アプリケーションの 2 ～ 3 倍に上ります。
- **安定性の向上**。ASP.NET ランタイムはプロセスを詳細に監視および管理します。プロセスでリークやデッドロックなどの不正処理が発生した場合、ランタイムは新しいプロセスを生成して問題が発生した

プロセスを置き換えることができます。これにより、アプリケーションは要求を継続的に処理できます。

- **開発者の生産性の向上。** サーバー コントロールやイベント処理などの ASP.NET の新機能は、アプリケーションをよりすばやく、より少ないコードで作成するのに役立ちます。また、従来よりも簡単にコードを HTML コンテンツから分離できます。
- **配置の簡素化。** ASP.NET アプリケーションのインストールは、ノー タッチ デプロイメントにより、大幅に簡素化されています。サーバーにファイルをコピーするだけでアプリケーションを配置できます。また、Web サーバーを再起動することなく、アプリケーションを更新できます。DLL を更新すると、ASP.NET が変更を自動的に検出し、新しいコンポーネントの使用を開始します。
- **セキュリティの向上。** ASP.NET は、.NET Framework や Microsoft インターネット インフォメーション サービス (IIS) と連携して Web アプリケーション全体のセキュリティを確保します。ASP.NET の主要なセキュリティ機能である認証や許可を利用すると、ユーザーの ID を確認したり、認証された ID に付与されているアクセス許可を管理したりすることができます。また、ASP.NET では、コード アクセス セキュリティ、ロールベースのセキュリティ、暗号化サービス、セキュリティ ポリシー管理など、.NET Framework に組み込まれているすべてのセキュリティ機能にアクセスできます。

詳細については、MSDN の「Converting ASP to ASP.NET」を参照してください。

この付録では、ASP to ASP.NET Migration Assistant を使用した、ASP から ASP.NET へのアップグレード プロセスについて概要を説明します。特に、準備、自動アップグレード、手作業による変更、テスト、一般的な問題とその解決方法などについて説明します。ASP から ASP.NET へのアップグレード プロセスに関する完全な説明はこのガイドの範囲を超えています。ただし、このガイドで提供する情報はアップグレードの手引きとして役立つはずです。

プロセスの概要

アップグレード プロジェクトに含まれる各ステップは、以下の 3 つのフェーズに分けることができます。

1. アプリケーションの準備
2. アプリケーションのアップグレード
3. アップグレード後のアプリケーションのテストとデバッグ

通常、これらのタスク グループは、スキルや経験が異なる別々のチームによって実行されます。各グループに含まれるタスクは、必要なリソースの可用性と特殊性に応じて、ある程度並行して実行できます。タスク グループでは、リニアな実行順序は定義されません。アプリケーションの異なる部分に対して、複数のタスクを同時に実行できる場合があります。

以降では、この章で説明するアップグレード手順を簡単にまとめ、各手順の入力と出力を示します。

アプリケーションの準備

1. 開発環境の準備

   入力：ソフトウェア開発およびサードパーティコンポーネントのインストーラ。

   出力：アップグレードするアプリケーションに対応する、完全な構成のアプリケーション開発環境。

2. インターネット インフォメーション サービス (IIS) および IIS と .NET Framework の統合の確認

   入力：インターネット インフォメーション インストーラおよび Visual Studio .NET インストーラ。

   出力：ASP.NET を動作させるには、IIS が正常に実行されており、Microsoft .NET Framework と完全に統合されている必要があります。

3. アプリケーションのリソース インベントリの取得

   入力：アプリケーションの仕様およびデザインに関する利用可能な情報。

   出力：アップグレードの際に役立つドキュメントのカタログ。

4. コンパイルの確認

   入力：適切に構成されたシステム内にある、元の ASP アプリケーション。

   出力：元のアプリケーションをコンパイル、デバッグ、および実行できるシステム。

5. プロジェクトのアップグレード順序の定義

   入力：元のソース コード。

   出力：分析ツールに基づいて行われた、アプリケーション コンポーネントの依存関係に関する分析。分析ツールを使用すると、さまざまなコンポーネントのアップグレード順序を計画できます。

6. アップグレード ツールの準備

   入力：アップグレード ウィザードおよび分析ツールのインストーラ。

   出力：元のアプリケーションを実行できる、完全な構成のシステム。

アプリケーションのアップグレード

1. ASP to ASP.NET Migration Assistant の実行

   入力：元の ASP ページと、移行アシスタントの実行可能ファイル。

   出力：ASP .NET で記述された、アップグレード後のアプリケーションの初期コード ベース。このコード ベースには、後で対処する必要があるアップグレードの問題が含まれる可能性があります。

2. アップグレードの進捗状況の確認

   移行アシスタントが適切に実行されているかどうかを確認するための制御ステップです。

3. 手作業による変更を加えて変換を完了
   入力：ASP .NET で記述された、アップグレード後のアプリケーションの初期コード ベース。
   出力：コンパイル可能な、アップグレード後の ASP.NET アプリケーション。

アップグレード後のアプリケーションのテストとデバッグ

1. 元のテスト ケースの実行
   入力：コンパイル可能な、アップグレード後の ASP.NET アプリケーション。
   出力：失敗したテスト ケースの一覧とアプリケーションで検出されたランタイムのバグ。
2. ランタイム エラーの修正
   入力：コンパイル可能な、アップグレード後の ASP.NET アプリケーション。
   出力：適切に実行できる、アップグレード後の ASP.NET アプリケーション。

この章の残りの部分では、上記の各ステップの詳細について説明します。

ASP to ASP.NET Migration Assistant について

ASP to ASP.NET Migration Assistant は、ASP ページや ASP アプリケーションの ASP.NET への変換を支援するように設計されています。プロセス全体が自動化されるわけではありませんが、アップグレードに必要なステップの一部が自動化されるため、プロジェクトを迅速にアップグレードできます。移行アシスタントは Visual Basic 6.0 アップグレード ウィザードとは関係がなく、Visual Studio .NET には付属していませんが、無料で使用できます。ASP to ASP.NET Migration Assistant のダウンロードについては、MSDN の Microsoft ASP.NET Developer Center を参照してください。

移行アシスタントで実行されるタスク

ASP to ASP.NET Migration Assistant を使用して ASP プロジェクトをアップグレードする場合、大きく分けて 3 つのタスクがこのツールによって実行されます。

- ツールは必要な ASP.NET ファイルを自動的に生成します。これらのファイル名には、.aspx という拡張子が付けられます。同時に、アップグレード対象のファイルへの参照を含む ASP ファイルまたは HTML ファイルをプロジェクト内で調べ、これらのファイルをアップグレードして参照先を新しく作成したファイルに変更します。include ディレクティブ、<script> タグで囲まれたコード、href 属性、Response.Redirect 命令と Application.execute 命令、<form> タグの action 属性など、ファイル内の複数の箇所で他のファイルへの参照が見つかる場合があります。

  ASP では、<!-- #include ... --> タグや <script src="filename"> タグを使用して、HTML コード、スクリプト コード、またはその両方を含むファイルをインクルードできます。移行アシスタントは、拡張子に関係なくこれらのファイルをすべて更新して、ASP.NET 構文を使用します。また、これらのファイル内で宣

言され、参照を含む ASP ファイルによって使用されているすべての変数および関数を解決します。ただし、ASP.NET では、純粋なスクリプト コードを含むファイルをインポートするために使用できるのは <script src="filename"> だけです。これらのファイルに HTML コードや <% … %> で囲まれたスクリプト コードが含まれる場合は、コンパイル エラーが発生します。

- VBScript では型宣言を使用できませんが、ASP.NET では型宣言が要求されます。そのため、ツールはコードを調べ、変数が使用されている方法を分析して CreateObject() メソッドによるオブジェクトの作成などの単純なケースのデータ型を推測し、対応する宣言を追加します。これにより、移行アシスタントによって生成されるコードの量が増え、既定のプロパティを簡単に拡張できるようになります。
- ASP では、レンダ タグ (<% … %>) の中でサブルーチンや変数を宣言し、サーバー スクリプト内でステートメントを実行できます。一方、ASP.NET では、スクリプト タグ (<script … > … </script>) の中で宣言を行い、レンダ タグの中でステートメントを実行する必要があります。したがって、ツールは、以下のように VBScript コードを適切なコード ブロックに移動します。
  - レンダ タグ <% … %> の中で宣言された関数およびサブルーチンは、グローバル <script> タグに移動され、その language 属性が "VBScript" に、runat 属性が "server" にそれぞれ設定されます。
  - <script runat="server"> タグ内のステートメントとコメントはレンダ タグ内に移動されます。
  - レンダ タグ内の変数および定数の宣言は、ファイルの先頭の <script> タグに移動されます。また、ファイル内で使用される未宣言の変数もこのタグにすべて移動されます。

移行アシスタントの制限

移行アシスタントによる ASP コードのアップグレードにはいくつかの制限があります。その例としては以下が挙げられます。

- 移行アシスタントは、ASP プロジェクトの重要なコンポーネントであるセキュリティ モデルを変換しません。ASP.NET と IIS のセキュリティ機能を利用するには、手作業でアプリケーションの再構成を行う必要があります。ASP.NET のセキュリティの詳細については、MSDN の『.NET Framework Developer's Guide』の「ASP.NET Web Application Security」を参照してください。
- ASP.NET では、1 つのページ内に複数の言語を使用してサーバー側の処理を実行することができません。ASP to ASP.NET Migration Assistant は、VBScript 以外のスクリプト言語で記述されたサーバー側コードの変換または分析を行わず、他の言語で記述されたサーバー側コードが出現する前に、変換されたファイルエラー通知を追加します。

メモ：この制限は、複数の言語で記述されたクライアント側コードには影響しません。コードが正しく記述されている限り、移行アシスタントは言語の違いに関係なくクライアント側コードを変換できます。

- ASP to ASP.NET Migration Assistant は、複雑なレンダリング関数を変更しません。これは Visual Studio .NET IDE のデザイナ ビューでのページ表示機能に影響し、コンパイル エラーの原因となります。この制限は、単純なレンダリング関数には影響しません。

アプリケーションの準備

正しい準備が行われない場合、ASP to ASP.NET Migration Assistant を使用したとしても、ASP アプリケーションの ASP.NET へのアップグレードは複雑でエラーの発生しやすいプロセスになります。プロセスの初期段階で発生したエラーは、その後の段階にも伝播し、作業量が増えることになります。また、アプリケーションの一部を再度アップグレードしなければならないことがあり、アップグレードのその他のプロセスの遅延につながります。ASP to ASP.NET Migration Assistant を使用してアップグレードを実行する前に、特定の手順を踏んで ASP アプリケーションのアップグレード準備を行うことにより、管理、開発、テストなど各分野での作業量を大幅に削減できます。

最初に実行すべきステップは、インターネット インフォメーション サービス (IIS) やサードパーティ コンポーネントなど、システム関連のコンポーネントのアップグレード準備です。通常、これらのコンポーネントの準備、アップグレードのその他の部分に対する影響の把握、およびこの段階で発生する問題の修正はコストの高い作業ではありません。こうした外部環境の準備を終えたら、ソース コード、アプリケーションのコンパイル、一般的なアップグレードの問題など、アプリケーションの内部に専念できます。

環境の準備

可能な場合は、元のアプリケーションの作成に使用した開発環境と同じ構成のコンピュータでアップグレード プロセスを実行します。これにより、元のアプリケーションの分析が簡単になり、初期準備テストの実行や、依存関係などの問題の特定が容易になります。

アップグレード プロセスに影響する環境的な要因には次の 2 種類があります。1 つは、アップグレード プロセスの速度に影響するコンピュータ リソースです。もう 1 つの要因は移行アシスタントの通常の実行と終了に影響します。たとえば、プロセスを実行するコンピュータに外部依存関係が存在しない場合、移行アシスタントの実行中に例外が発生します。

システム リソース

移行アシスタントは、アプリケーションの VBScript コードをすべてメモリ内に読み込み、これらのコードを変換して必要なすべての .aspx ファイルを作成します。したがって、言語構造の格納や変換ルールの実行のために、大量のシステム メモリやプロセッサ リソースが必要になる場合があります。

少なくとも 512 MB のシステム RAM を搭載したコンピュータで移行アシスタントを実行することをお勧めします。100,000 行以上のコードや、レンダ タグから <script> タグへの移動が必要なコードを大量に含む複雑な ASP アプリケーションをアップグレードする場合は、少なくとも 1 GB の RAM を搭載したコンピュータを使用することをお勧めします。このような特性を持つアプリケーションをアップグレードする場合、平均で 3 ～ 5 時間の処理時間が必要になります。また、アプリケーションをより管理しやすい複数の単位に分割すると便利な場合もあります。

システムのハード ドライブには、アップグレード後の ASP.NET ファイルと、生成されるすべての一時ファイルを格納するために、元のアプリケーションの少なくとも 3 倍のサイズの空き容量が必要です。ファイルの格納先ディレクトリに対する適切なユーザー権限がアップグレードを実行する開発者に付与されていることを確認してください。

サードパーティ コンポーネント

ASP アプリケーションの多くでは、サードパーティ コンポーネントが何らかの形で使用されています。アプリケーションはプロジェクト設定を介して外部 DLL を明示的に参照することも、CreateObject などの命令を使用して、インストールされているコンポーネントに基づいてオブジェクトを動的に作成することもできます。

外部コンポーネントに依存するアプリケーションをアップグレードする場合は、アップグレードで使用するコンピュータにこれらのコンポーネントがインストールされており、それらが利用できることを確認する必要があります。ASP to ASP.NET Migration Assistant がコンポーネントにアクセスできない場合、アップグレード後のコードでそのコンポーネントを使用するすべての命令に問題が発生します。たとえば、アプリケーションが TRE コンポーネントを参照しており、アップグレードに使用しているコンピュータにこのコンポーネントがインストールされていない場合、移行アシスタントは以下のようなメモを ASP.NET コードに追加します。

```
' UPGRADE_NOTE: The 'TRE.DLL' object is not registered in the
' migration machine.
obj = CreateObject("TRE.DLL")
```

メモ : 試用版のサードパーティ コンポーネントによっては、コンポーネントがインスタンス化されるたびにライセンス情報のダイアログ ボックスが表示されることがあります。このダイアログ ボックスが表示されると、ホスト アプリケーション (この場合は ASP to ASP.NET Migration Assistant) の実行が停止されます。試用版のサードパーティ コンポーネントを使用している場合は、アップグレード プロセスの実行中に表示されるダイアログ ボックスに応じてツールが生成するメッセージに注意する必要があります。

アプリケーションの準備の一部として、複雑で例外的なコンポーネントを特定し、限定的なテストを実行する必要があります。これらのコンポーネントのコア機能をインスタンス化してアクセスするための、サイズの小さなアプリケーションを作成します。これらのアプリケーションを作成したら、移行アシスタントを実行してその結果を評価します。その際には、アップグレード後のコンポーネントのデザイン時および実行時における表示と動作に特に注意します。

IIS と仮想ディレクトリ

新しい ASP.NET アプリケーションの実行に使用する仮想ディレクトリを IIS が正しく作成できるようにするには、バージョン 5.0 以降の IIS と Microsoft FrontPage® Server Extensions がアップグレード プロセスに使用するコンピュータにインストールされていることを確認します。アップグレードを実行する前に ASP アプリケーションをテストすることにより、コンピュータがアプリケーションのすべての要件を満たしていることを確認できます。

ツール

ASP to ASP.NET Migration Assistant は Microsoft Visual Studio .NET のアドインです。したがって、アップグレードに使用するコンピュータに Visual Studio をインストールする必要があります。

また、移行アシスタントはさまざまなバージョンの Visual Basic アップグレード ツールを使用します。アップグレード ツールには、ASP ページ内で使用されているすべての VBScript コードの解析および変換を行う Visual Basic Upgrade Wizard Companion があります。Visual Basic Upgrade Wizard Companion の詳細については、ArtinSoft の Web サイトを参照してください。

移行アシスタントのためのコードの準備

コードに対していくつかの変更を行うことにより、移行アシスタントによる自動変換が改善され、アップグレード後に手動で加える変更の数を減らすことができます。以降では、これらの変更の中で最も重要なものについて説明します。

参照されるすべての COM コンポーネントの登録

コードの分析で最初に実行するステップは、アプリケーションが使用するすべての COM コンポーネントの特定です。COM コンポーネントの一覧を作成したら、アップグレードに使用するコンピュータにこれらのコンポーネントが正しくインストールおよび登録されていることを確認し、ライセンスやアクセス許可に関する問題を解決します。

コンパイルの確認

コードが構文的に正しいかどうかを確認することは非常に重要です。ファイルに構文エラーがあると、移行アシスタントによってコードが解析されるときに問題が発生し、予期しない結果になる可能性があります。

また、コード レビューを行うことにより、プロジェクトで必要なファイルがすべてあるかどうかを確認できます。移行アシスタントは、ASP プロジェクトを変換する前にコードをすべて分析し、収集した情報を使用してアップグレードの手順を決定します。不足しているファイルがあると、ツールによる決定に悪影響を及ぼす可能性があります。

注意すべき一般的な間違いとして、以下が挙げられます。

- **不足ファイル。**include 命令は非常に便利な機能で、この命令を使用すると ASP 開発者は仮想パスまたは物理パスを使用して他のファイルを参照できます。ASP.NET へのアップグレード時によくある失敗の 1 つは、これらの参照先のファイルがアップグレードに使用するコンピュータの適切な場所にコピーされていないことです。たとえば、以下のように ASP ページで Variables.asp ファイルをインクルードしていると仮定します。

```
<!-- #Include Virtual="http://ServerName/myVirtualRoot/Lib/Variables.asp" -->
```

ASP ページをアップグレードする前に Variables.asp が仮想ディレクトリ http://ServerName/myVirtualRoot/Lib/ にコピーされていない場合、移行アシスタントによってページの include 命令の前にアップグレードに関するメモが追加されます。以下に例を示します。

```
<% 'UPGRADE_NOTE: The
    'http://ServerName/myVirtualRoot/Lib/Variables.asp' file was
    not found in the migration directory. Copy this link in your
    browser for more:
    ms-help://MS.MSDNVS/aspcon/html/aspup1003.htm %>
<!-- #Include Virtual="http://ServerName/myVirtualRoot/Lib/Variables.asp" -->
```

この問題を解決するには、参照されるファイルを適切なディレクトリにコピーするか、ファイルが実際に格納されているローカル パス名または統一リソース識別子 (URI) にパスを変更して、アップグレードに関するメモを削除します。

- **不適切な HTML タグ。**ページ内に不適切な HTML タグが含まれる場合、移行アシスタントがページを完全に解析できないため、アップグレードに関するメモが生成されます。これは命令タグの閉じ引用符を記述し忘れた場合に発生します。以下に例を示します。

```
<!--#include file="./Include/Table.asp -->
```

引用符の記述漏れがあると、移行アシスタントがページのインクルード ファイルを特定および分析できなくなるため問題が発生します。このタグを含むページに移行アシスタントを適用した場合、以下のようなメモが生成されます。

```
<% 'UPGRADE_NOTE: '#INCLUDE' tag is malformed. Copy this link in your browser for
    more: ms-help://MS.MSDNVS/aspcon/html/aspup1004.htm %>
<!--#include file="./Include/Table.asp -->
```

この問題を解決するには、HTML タグを修正して、アップグレードに関するメモを削除します。

```
<!--#include file="./Include/Table.asp" -->
```

ファイルの修正が完了したら、すべてのインクルード ファイルが正しく認識されるように、移行アシスタントを再実行します。

- **未登録の COM オブジェクト。**移行アシスタントを使用してファイルを変換する場合は、参照されるすべての COM オブジェクトを、変換を実行するコンピュータに正しくインストールする必要があります。アップグレード対象の ASP ファイルによって参照される COM オブジェクトに移行アシスタントがアクセスできない場合、アップグレードに関するメモが生成されます。たとえば、以下の命令が含まれる ASP ページについて考えてみましょう。

```
<%
     Set myObject = CreateObject("MyObjectCOM.MyClass")
%>
```

移行アシスタントの実行時に MyObjectCOM コンポーネントがコンピュータに登録されていない場合、以下のメモが生成されます。

```
<%
    ' UPGRADE_NOTE: The 'MyObjectCOM' object is not registered in the migration
        machine. Copy this link in your browser for more:
        ms-help://MS.MSDNVS/aspcon/html/aspup1016.htm
    myObject = New MyObjectCOM.MyClass
%>
```

ASP.NET ページでのランタイム エラーを回避するには、MyObjectCOM コンポーネントを正しくインストールし、移行アシスタントを再実行してコードを修正します。

循環参照の削除

ASP (および ASP.NET) の柔軟性および再利用性の鍵となるのがインクルード機能です。インクルード機能を利用すると、#include 命令または <script> タグの scr 属性を使って他のファイルのコードをページに簡単に追加できます。ただし、この機能の使い方を誤ると循環参照が発生する場合があります。循環参照は複数のファイルを相互にインクルードしたときに発生します。循環参照は ASP ページの実行に影響しませんが、ASP.NET では循環参照が許可されません。循環参照を含むページに対して移行アシスタントを実行すると、ツールのパフォーマンスに影響し、コードの一部が変換されない場合があります。コード内で発生する可能性のある循環参照を特定し、共有コードを新規ファイルに移動してこれらの循環参照を削除します。その後、新規ファイルを元のファイルにインポートできます。

サーバー側での複数スクリプト言語の回避

ASP ページを変換する前に考慮しなければならない点がもう 1 つあります。それは、ASP では VBScript や JavaScript など複数の言語をスクリプトで使用できるのに対し、ASP.NET では同一ページ内のサーバー側スクリプトを複数の言語で記述できないということです。したがって、ページ内のサーバー側スクリプトは 1 種類の言語で記述する必要があります。

移行アシスタントが解析できるのは VBScript 言語のみであるため、サーバーで実行されるすべてのコードを VBScript に変更することをお勧めします。これにより、Visual Basic .NET が提供するすべての新機能をアップグレード後のコードで利用できるようになります。他のスクリプト言語を使用することもできます。ただし、他の言語を使用すると、コードが新しい ページにコピーされるだけでアップグレードされません。ASP と ASP.NET の違いのため、アップグレード後の ASP.NET ページでの動作が以前と異なる場合があります。

アプリケーションのアップグレード

適切な準備手順を実行したら、実際のアップグレード プロセスを開始することができます。ここでは、アップグレードを実現するためのステップについて説明します。

アップグレード オプション

ASP と Visual Basic 6.0 の両方のコンポーネントを使用したエンタープライズ アプリケーションをアップグレードする場合、ASP to ASP.NET Migration Assistant と Visual Basic 6.0 to Visual Basic .NET Upgrade Wizard の両方が必要になることがあります。ビジネス ロジック層、データ アクセス層のユーザー定義クラス、またはアプリケーション サーバー上で実行される Visual Basic 6.0 で記述された追加コンポーネントがアプリケーションに含まれる場合は、Visual Basic 6.0 to Visual Basic .NET Upgrade Wizard を使用してこれらのアプリケーション要素をアップグレードする必要があります。これらの基盤コンポーネントのアップグレードが完了したら、ASP to ASP.NET Migration Assistant を使用して ASP コードを処理できます。

ASP to ASP.NET Migration Assistant の使用

ASP to ASP.NET Migration Assistant は、Visual Studio .NET IDE のウィザードまたはコマンド ライン ツールとして使用できます。以降では、これらのアプローチについて説明します。

ウィザードの使用

Visual Studio .NET を使用して ASP アプリケーションを変換するには、[**ファイル**] メニューの [**開く**] をポイントして [**変換**] をクリックします。次に、使用可能なコンバータの一覧から [ASP to ASP.NET Migration Assistant] をクリックします。Visual Studio .NET でソリューションを開いている場合は、[Add to current solution] をクリックして結果を現在のソリューションに追加できます。または、[Create new solution] をクリックして、結果を別のソリューションに保存することもできます。

このウィザードは、ASP ファイルをアップグレードするための手順として、以下の 7 つのページで構成されています。

1. 最初にウェルカム ページが表示されます。[次へ] をクリックして手順を開始します。
2. 2 番目のページでは、アップグレードするファイルが格納されているディレクトリを指定します。変換対象のファイルが格納されているディレクトリへのパスを入力するか、[参照] をクリックして [ファイルを開く] ダイアログ ボックスからディレクトリを検索します。目的のディレクトリが見つかったら、[次へ] をクリックします。
3. 3 番目のページでは、プロジェクトの名前を指定します。新しい ASP.NET アプリケーションの名前を入力して、[次へ] をクリックします。
4. 4 番目のページでは、アップグレード後のプロジェクトを格納するターゲット ディレクトリを指定します。新しい ASP.NET プロジェクトを作成するディレクトリへのパスを入力するか、[参照] をクリックしてダイアログ ボックスからディレクトリを検索します。ターゲット ディレクトリを指定したら、[次へ] をクリックします。
5. 5 番目のページでは、.asp から .aspx にアップグレードする絶対参照アドレスまたは統一リソース識別子 (URI) アドレスを指定します。通常、ASP to ASP.NET Migration Assistant は、アップグレードされた新しいファイルを参照するように相対アドレスを変換します。たとえば、
   <A href="../../OtherDirectory/Default.asp"> ... は
   <A href="../../OtherDirectory/Default.aspx"> ... に変換されます。この変換をすべての URI アドレスに適用するのは好ましくありません。一部の URI アドレスはアップグレード対象ではない外部リソースを参照しており、これらのアドレスについては変更する必要がないからです。このような状況の URI や絶対アドレスを制御するには、ウィザードのこのページでアップグレードするアドレスを指定する必要があります。
   これらの参照を指定したら、[次へ] をクリックして続行します。
6. 6 番目の [アップグレード可能] ページでは、アップグレード手順を開始する準備ができたことが通知されます。[次へ] をクリックしてアップグレードを開始します。
7. 7 番目のページでは、移行アシスタントによるファイルのアップグレードの進捗状況を示すインジケータが表示されます。進捗状況を示すインジケータがフルになると、アップグレード手順が完了します。

コマンド ライン ツールの使用

ASPUpgrade.exe コマンド ライン ツールには、ウィザードと同じ機能があります。ASPUpgrade.exe のコマンド ライン構文を以下に示します。

```
ASPUpgrade[.exe] DirectoryName [/out DirectoryName] [/nolog | /logfile filename ]
[/Verbose] [/ProjectName] [/ForceOverwrite]
```

各オプションの意味を表 1 に示します。

表 1: ASPUpgrade.exe のオプション

引数	オプション	説明
DirectoryName		必須。アップグレードする ASP ファイルのパスです。
	/out *DirectoryName*	ASP.NET プロジェクトの作成先フォルダのパスを指定します。このオプションを指定しない場合、\OutDir が既定のパスとして使用されます。
	/verbose	アップグレード中のすべての出力をコマンド プロンプトに表示します。
	/nolog	アップグレード中にログ ファイルを作成しません。
	/logfile *filename*	指定された *filename* (必要に応じて完全パス名を指定できます) を使用してログ ファイルを作成します。パスとファイル名を指定しない場合、ASP.NET プロジェクトと同じフォルダに既定のファイル名 (*ProjectName*.log) でログ ファイルが作成されます。ここで *ProjectName* はプロジェクト ファイルの名前です。
	/? または **/help**	コマンド構文オプションの一覧を表示します。

コマンド ライン バージョンのツールを使用する場合は、以下の点に注意する必要があります。

- 引数では大文字と小文字が区別されません。
- スペースを含むパスは二重引用符で囲む必要があります。
- ターゲット ディレクトリが存在しない場合は、移行アシスタントがディレクトリを作成します。
- ターゲット ディレクトリが既に存在する場合、移行アシスタントはこのディレクトリの内容を上書きするかどうかをユーザーに確認します。
- ユーザーにはターゲット ディレクトリの書き込み権限が必要です。
- /NoLog オプションと /LogFile オプションを同時に使用することはできません。

手作業による変更を加えてアップグレードを完了

アプリケーションのアップグレード完了後に発生するエラーの中で最も検出が困難なのは、ランタイム エラーです。移行アシスタントは、発生する可能性のあるランタイム エラーの多くを検出でき、ユーザーがこれらのエラーを修正できるようにコード内の該当箇所にマークを付けます。ただし、移行アシスタントによって検出されない問題もあります。これらの問題は、アップグレード後のアプリケーションを実行して、十分にテストした後で初めて検出されます。元のアプリケーションのテスト ケースとデザイン ドキュメントが利用できる場は、テスト フェーズで非常に役立ちます。

ASP to ASP.NET Migration Assistant によって検出され、手動で修正を行わなければならない一般的なアップグレードの問題を以下に示します。

- **動作が変更された機能。**ASP 関数の中には ASP.NET で動作が変更されたものがあります。これらの ASP.NET 関数を使用するようにコードをアップグレードするときに、移行アシスタントはアップグレードに関する警告をコードに追加します。これにより、新しいメソッドを適切にテストできるようになります。たとえば、以下のコードでは、ASP の Date 関数を使用して現在の日付を取得し、その日付をブラウザに表示しています。

```
<script Language=VBScript runat=server>
Sub PrintDate()
    Dim myVar
    myVar = Date
    Response.Write("<strong>Current Date</strong> = " & CStr(myVar))
End Sub

PrintDate
</script>
```

このコードを ASP.NET に変換すると、以下のコードが生成されます。

```
<script language="VB" runat="server">
Sub PrintDate()
    Dim myVar As Date
    ' UPGRADE_WARNING: Date was upgraded to Today and has a
    ' new behavior.
    myVar = Today
    Response.Write(("<STRONG>Current Date</STRONG> = " & CStr(myVar)))
End Sub
</script>

<%
    PrintDate()
%>
```

メモ：上記のコードを含め、この章で提示するすべてのコードでは、ASP to ASP.NET Migration Assistant によって自動的に生成された長いコード行やコメントの一部を変更しています。これはコードを読みやすく理解しやすくするためです。実際のコードとは若干異なる場合があります。

ほとんどの場合、アプリケーションをテストするだけで、新しい関数を含むアプリケーションが正常に動作しているかどうかを検証できます。ASP.NET アプリケーションが ASP バージョンのときと同じように動作する場合は、警告を削除します。この例では、新しい関数である Today は ASP の Date 関数と同じ形式で現在の日付を返します。したがって、以下に示すように、必要なのは警告を削除することだけです。

```
<script language="VB"  runat="server">
Sub PrintDate()
    Dim myVar As Date
```

```
    myVar = Today
    Response.Write(("<STRONG>Current Date</STRONG> = " & CStr(myVar)))
End Sub
</script>

<%
    PrintDate()
%>
```

- **サポートされない機能。** ASP のプロパティ、メソッド、およびイベントの一部は ASP.NET に存在しません。また、これらの動作を新しい機能でシミュレートすることもできません。移行アシスタントがこれらの機能を検出すると、そのまま変更しないで残しておき、アップグレードに関するメモをコードに追加して問題の発生をユーザーに通知します。たとえば、この ASP コードでは、ASP.NET でサポートされない Calendar プロパティが使用されています。

```
<script Language=VBScript runat="server">

Sub ChangeCalendar()
    if (Calendar = vbCalGreg) then
        Calendar = vbCalHijri
    else
        Calendar = vbCalGreg
    end if
End Sub

ChangeCalendar
</script>
```

このコードを変換すると、以下の ASP.NET コードが生成されます。

```
<script language="VB" runat="server">

Sub ChangeCalendar()
    ' CONVERSION_TODO: The 'vbCalGreg' identifier was not declared,
    ' because it is a constant in 'VBA.VbCalendar' library. Copy
    ' this link in your browser for more:
    ' ms-its:C:\Program Files\ASP to ASP.NET Migration Assistant\
    ' AspToAspNet.chm::/1020.htm
    ' UPGRADE_ISSUE: Calendar プロパティはサポートされていません。 Copy this
    ' link in your browser for more:
    ' 'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1039"'
    If (Calendar = vbCalGreg) Then
        ' UPGRADE_ISSUE: Calendar プロパティはサポートされていません。
        ' CONVERSION_TODO: The 'vbCalHijri' identifier was not
        ' declared, because it is a constant in 'VBA.VbCalendar'
        ' library. Copy this link in your browser for more:
        ' ms-its:C:\Program Files\ASP to ASP.NET Migration Assistant\
        ' AspToAspNet.chm::/1020.htm
        Calendar = vbCalHijri
    Else
```

```
        ' UPGRADE_ISSUE: Calendar プロパティはサポートされていません。
        ' CONVERSION_TODO: The 'vbCalGreg' identifier was not declared,
        ' because it is a constant in 'VBA.VbCalendar' library.
        ' Copy this link in your browser for more:
        ' ms-its:C:\Program Files\
        ' ASP to ASP.NET Migration Assistant\AspToAspNet.chm::/1020.htm
        Calendar = vbCalGreg
    End If
End Sub

</script>
<%
ChangeCalendar()
%>
```

移行アシスタントによって、プロパティ、メソッド、またはイベントが対応する ASP.NET のそれぞれに変換できないことが報告された場合は、以下の 2 つの選択肢があります。アプリケーションへの影響が少ない場合は、該当する命令を削除できます。または、新しい言語でアプリケーションに類似した機能を提供する対処方法を探すこともできます。たとえば、ASP の Calendar プロパティの代わりに、ASP.NET の System.Globalization 名前空間を使用できます。この名前空間には、言語、国または地域、使用されるカレンダー、日付、通貨、および数値の形式、文字列の並べ替え順序など、カルチャ関連の情報を定義するクラスが含まれます。

- **レンダリング関数。** ASP では、複数のレンダ タグの間に通常の HTML を挿入してメソッドを定義できます。ただし、ASP.NET では、<script> タグの開始タグと終了タグの中でしかメソッドを宣言できません。たとえば、以下に示すように、ASP コードの helloWorld メソッドは、HTML を間に挟む 2 つのレンダ タグで開始および終了します。

```
<%Sub helloWorld()%>
    <h1>Hello World</h1>
<%End Sub %>
```

このコードを変換すると、レンダリング関数が移行アシスタントによって変更されず、代わりにエラーが生成されます。

```
<% 'UPGRADE_NOTE: Rendering Functions are not supported. Copy this
    link in your browser for more:
    ms-help://MS.MSDNVS/aspcon/html/aspup1008.htm %>
<%Sub helloWorld()%>
    <h1>Hello World</h1>
<%End Sub %>
```

このエラーを解決するには、メソッド全体を一組の <script> タグ (開始タグと終了タグ) の間に移動し、Response.Write メソッドを使用して HTML 情報をページに送信する必要があります。前の例を書き

換えると以下の正当なASPコードになり、移行アシスタントによって正常に処理されます。

```
<script language=VBScript runat=Server>
Sub helloWorld()
    Response.Write("<h1>Hello World</h1>")
End Sub
</script>
<% helloWorld %>
```

- **複数のメソッド宣言。**ASP では、同一のメソッドを複数回宣言できます。これは最後に記述された宣言のみが使用され、残りの宣言が無視されるためです。ただし、ASP.NET では、同一の署名を持つメソッドを複数回宣言することはできません。同一の署名を持つ複数のメソッド宣言が移行アシスタントによって検出された場合、最後の宣言を除くすべての宣言がコメント アウトされます。たとえば、このASPコードでは、MySub 関数が 2 回宣言されています。

```
<%
    Sub MySub()
        Dim y
        y = "String Value"
        Response.Write y
    End Sub

    Sub MySub()
        Dim y
        y = 1
        Response.Write y
    End Sub

    MySub
%>
```

このコードを移行アシスタントで変換すると、最初のメソッド宣言がコメント アウトされ、アップグレードに関するメモが追加されます。生成されたASP.NETコードを以下に示します。

```
' UPGRADE_NOTE: All function, subroutine and variable declarations
' were moved into a script tag global. Copy this link in your browser
' for more:
' ms-help://MS.MSDNVS/aspcon/html/aspup1007.htm
<script language="VB" runat="Server">
    ' UPGRADE_NOTE: Subroutine 'MySub' was commented. Copy this
    ' link in your browser for more:
    ' ms-help://MS.MSDNVS/aspcon/html/aspup1009.htm
    ' Sub MySub()
    '     Dim y
    '     y = "String Value"
    '     response. write y
```

```
    ' End Sub

    Sub MySub()
        Dim y As Integer
        y = 1
        response. write(y)
    End Sub

</script>

<%
    MySub()
%>
```

アップグレードに関するメモとコメント アウトされたコードをプロジェクトからクリアするには、ページからこれらを削除します。

- **スクリプト タグ内で到達できないコード。**<script> タグを使用すると、次のいずれかの方法でコードをページに追加できます。1 つは、ページの <script> タグの中 (開始タグと終了タグの間) に直接コードを入力する方法です。もう 1 つは、使用するコードを含む外部ファイルを開始タグの src 属性で指定する方法です。一組の <script> タグに外部ファイルへの参照とコードの両方が含まれる場合、ASP は src 属性で指定されたファイルのコードのみを使用し、開始タグと終了タグの間のコードを無視します。たとえば、以下の ASP コードを IIS が処理する場合、FileName.vb ファイルからコードがインポートされ、ページの MySub 宣言は無視されます。

```
<script Language=VBScript src="FileName.vb" runat=server>
Sub MySub()
    dim myVar
    myVar = myVar + 1
    Response.Write("SUM = " & CStr(myVar))
End Sub
</script>
```

このページを移行アシスタントで変換すると、以下のようなアップグレードに関するメモが生成されます。

```
<%'UPGRADE_NOTE: Code inside the Script tag was ignored by the upgrade tool. Copy this link in
your browser for more: ms-its:C:\Program Files\ASP to ASP.NET Migration
Assistant\AspToAspNet.chm::/1006.htm  %>
<%'UPGRADE_NOTE: Language element 'SCRIPT' was migrated to the same language element but still
may have a different behavior. Copy this link in your browser for more: ms-its:C:\Program
Files\ASP to ASP.NET Migration Assistant\AspToAspNet.chm::/1011.htm  %>
<script language="VB" src="FileName.vb" runat=server>
Sub MySub()
    Dim myVar
    myVar = myVar + 1
    Response.Write("SUM = " & CStr(myVar))
End Sub
</script>
```

ASP と同様に、ASP.NET は <script> タグの開始タグと終了タグの間のコードを無視します。これは使用する外部ファイルが src 属性で指定されているためです。このコードを使用する場合は、別の <script> タグにコードを移動する必要があります。

```
<script language="VB" src="FileName.vb" runat=server></script>
<script language="VB" runat=server>
Sub MySub()
    Dim myVar
    myVar = myVar + 1
    Response.Write("SUM = " & CStr(myVar))
End Sub
</script>
```

このコードが不要な場合は、ページからコードを削除して、外部ファイルを参照する空の <script> タグのみを残します。

```
<script language="VB" src="FileName.vb" runat=server></script>
```

アップグレード後のアプリケーションのテストとデバッグ

テスト フェーズおよびデバッグ フェーズの目的は、アプリケーションを ASP.NET に変換した後に発生する問題を特定および修正することです。特定が容易ですぐに修正できる問題もありますが、アプリケーションの実行時にのみ発生する問題もあります。

ASP to ASP.NET Migration Assistant は、アプリケーションの変換後に対処する問題の一覧を含む変換レポートを生成します。このレポートには以下のような問題が示されます。

- **変換に関する問題。** サポートされないクラス メンバや言語要素に関する問題がある場合に生成されます。問題のある式の一部を .NET の類似した要素に手動で変換する必要があります。各問題について該当する式の機能を確認し、限定的なテストを実行します。
- **ToDo。** コードがユーザーの介入を必要とする場合に生成されます。アップグレードしたソース コードに統合した後、項目内の指示に従ってテストする必要があります。
- **ランタイムの警告。** コンパイル可能であるにもかかわらず、ランタイム エラーが発生する可能性のあるコードによって生成されます。このコードは十分にテストする必要があります。
- **グローバル警告。** ガベージ コレクタが原因でオブジェクトのライフタイムが十分でないなどのグローバルな問題が含まれます。これらの警告の大部分はセットアップに関する問題が原因で、簡単に修正およびテストできます。移行アシスタントによって認識されないため、手作業による変換が必要なソースコード行など、より多くの修正作業を要する警告もあります。

- **メモ。**ソース コードが大幅に変更された場合、またはアップグレードされた ASP.NET コードと元の Visual Basic 6.0 バージョンで動作が異なる場合に生成されます。動作の相違がアプリケーションのコア機能に影響するかどうかを確認するため、コードをテストする必要があります。

元のアプリケーションの仕様書とテストケースを使用して ASP.NET バージョンのアプリケーション用に新しいテスト ケースを作成し、アップグレード後のアプリケーションのすべての機能について単体テストとシステム テストを実行する必要があります。

大規模なアプリケーションの場合は、アプリケーションを小さなモジュールに分割して、各モジュールを個別にテストすることをお勧めします。最初の段階では、ASP.NET アプリケーションの最も基本的なコンポーネント、つまり他のユーザー定義コンポーネントに依存しないコンポーネントをテストできます。必要に応じて基本コンポーネントのテストと調整を終えたら、これらのコンポーネントに依存するコンポーネントを検証できます。このアプローチに従うと、作業を漸進的に進めることができ、検出された問題を容易に切り離して修正できます。主な欠点は、追加的な計画および調整が必要になることと、作業の並列度が低下することです。ASP と ASP.NET を同じ Web サーバー上で同時に実行できる点を考慮する必要があります。これにより、テストを終えた既知の機能に基づいて、アプリケーションの機能を漸進的にテストできます。たとえば、受け取った注文の問い合わせを処理し、要求に応じて詳細レポートを生成する Web アプリケーションをテストする場合、詳細レポート生成する ASP.NET コンポーネントをテストしながら、それ以外の機能を ASP で実行できます。このアプローチは、機能や作業負荷に関する予備テストを行う場合に役立ちます。ASP コンポーネントと ASP.NET コンポーネントの間で直接的な通信が行われない場合に限り、このアプローチを実行できます。なぜなら、これらのコンポーネントは別々のサーバー プロセス内で実行されるからです。

配置

アプリケーションのテストが完了し、企業の標準に従って実行を開始したら、次のステップはアプリケーションを他のコンピュータに自動的に配布するための新しい インストーラを作成することです。Visual Studio .NET の配置機能は、配置ソリューションを自動的に構築し、登録や構成に関する問題を処理します。

▶ ASP.NET アプリケーションを配置するには

1. Web セットアップ プロジェクトを作成します。
2. セットアップ プロジェクト内の Web サイトの構造を作成します。
3. ASP.NET プロジェクトをプロジェクト出力として配置プロジェクトに追加します。
4. ソリューションをビルドします。

ビルドされた ASP.NET ソリューションの配置は、アプリケーションを配置するサーバーの管理者ユーザー権限を持つユーザーが行う必要があります。インストーラの作成に関する詳細については、第 16 章「アプリケーションの完成」を参照してください。

詳細情報

ASP から ASP.NET へのアップグレードの詳細については、MSDN の「Migrating to ASP.NET: Key Considerations」および「Converting ASP to ASP.NET」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/aspnetmigrissues.asp
および
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/convertasptoaspnet.asp
です。

ASP.NET のシステム要件の詳細については、MSDN の Microsoft Visual Studio Developer Center の「System Requirements for Visual Studio .NET 2003」を参照してください。
URL は http://msdn.microsoft.com/vstudio/productinfo/sysreqs/default.aspx です。

ASP to ASP.NET Migration Assistant のダウンロードについては、MSDN の Microsoft ASP.NET Developer Center を参照してください。
URL は http://msdn.microsoft.com/asp.net/migration/aspmig/aspmigasst/default.aspx です。

ASP.NET のセキュリティの詳細については、MSDN の『.NET Framework Developer's Guide』の「ASP.NET Web Application Security」を参照してください。
URL は http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconASPNETWebApplicationSecurity.asp です。

VB Companion の詳細については、ArtinSoft の Web サイトを参照してください。
URL は http://www.artinsoft.com/pr_vbcompanion.aspx です。

付録

D

FMStocks 2000 のアップグレード - ケース スタディ

このケース スタディでは、Visual Basic 6.0 Upgrade Assessment Tool と Visual Basic アップグレード ウィザードに加え、Fitch & Mather Stocks 2000 (FMStocks 2000) アプリケーションをリファレンス アプリケーションとしているガイダンスの章を利用して、Microsoft Visual Basic 6.0 アプリケーションを機能的に同等な Visual Basic .NET アプリケーションにアップグレードするプロセスを紹介します。FMStocks 2000 アプリケーションを Visual Basic .NET にアップグレードする際に発生する問題について解説し、その解決策を特定し、その解決策に至った経緯を説明していきます。このケース スタディの目的は、Visual Basic .NET へのアップグレードに関するすべての問題を明らかにすることではなく、問題を特定して解決するためのアプローチを提示することです。

また、FMStocks 2000 をサンプル アプリケーションとして評価ツールとアップグレード ウィザードを試してみる場合には、導入ガイドとしての役割も果たします。これにより、開発者が実際のアップグレード プロジェクトに着手する前に、それぞれのツールとアップグレード プロセスに関する理解を深めておくことができます。

アップグレード プロセスのデモンストレーションのため、アップグレード ウィザードを使用して FMStocks 2000 を Visual Basic .NET にアップグレードし、FMStocks_AutomatedUpgrade という名前を付けました。その後、FMStocks_AutomatedUpgrade を手作業で修正して機能的に同等な .NET アプリケーションを作成し、自動生成されたコードと区別するため FMStocks_NET という名前を付けました。FMStocks_AutomatedUpgrade と FMStocks_NET は、それぞれ FMStocks_AutomatedUpgrade.msi および FMStocks_NET.msi というファイル名で付属の CD に収録されています。また、このガイドの GotDotNet コミュニティ サイトから入手することもできます。

FMStocks 2000 の概要

Fitch & Mather Stocks 2000 (FMStocks 2000) は、Microsoft Windows 2000 向けに開発された Microsoft Windows Distributed interNet Applications Architecture (Windows DNA) のサンプル アプリケーションであり、インターネット上の公開株式取引の Web サイトをシミュレートします。このオンライン株式取引のシナリオは、オンライン書店の追加により拡張されています。

FMStocks 2000 は、プレゼンテーション層、ビジネス ロジック層、およびデータ層で構成されています。プレゼンテーション層は ASP を使用し、分散 COM を介してビジネス ロジック層と対話します。ビジネス ロジック層の主要なコンポーネントは FMStocks_Bus と FMSStore_Bus で、オンライン株式取引とオンライン書店に必要なサービスがそれぞれに組み込まれています。データ層は、中間層に存在するデータ アクセス コンポーネントとデータベースで構成されます。データ アクセス コンポーネントには FMStocks_DB と FMStore_DB があり、これらのコンポーネントがデータベース (Microsoft SQL Server 2000) とビジネス ロジック コンポーネントのデータ交換を処理します。

FMStocks 2000 の特徴を以下に示します。

- パフォーマンスと拡張性に優れています。
- 比較的簡単な COM+ プログラミング形式を利用することにより、コンポーネントを簡単に開発できるようになっています。
- オンライン株式取引のシナリオに書店を追加して拡張しています。より複雑な状態管理とショッピング カート機能が実現され、購入を始めとする粒度の大きいトランザクションの最中もサイトの可用性を高く保てるようになっています。
- Excel および Windows CE を利用した無線アクセスで Microsoft Office 2000 をサイトに接続できるようにして、クライアントの利便性を高めています。
- クリアテキストのクッキーを排除し、重要なビジネス コンポーネントを Web サーバーから分離することにより、アプリケーションのセキュリティを高めています。

FMStocks 2000 をケース スタディの題材とする理由

FMStocks 2000 は、Visual Basic 6.0 から Visual Basic .NET へのアップグレードに関するケース スタディの題材に適した多数の機能を備えています。こうした機能として以下が挙げられます。

- COM+ 相互運用と MTS
- Office 2000 との統合
- エラー ログ
- インターフェイスの使用
- Windows API の呼び出し
- ActiveX データ オブジェクト (ADO) の使用

FMStocks 2000 のセットアップ

Fitch & Mather Stocks 2000 (FMStocks 2000) アプリケーションは、ソース コードも含め、FMStocks 2000 の Web サイトにあるダウンロード セクションから入手できます。アプリケーションをインストールするには、インストール セットアップ プログラムをダウンロードして実行します。インストール ウィザードの指示に従って、アプリケーションのインストール先を指定します。

アプリケーションのビルドと実行の詳細については、<インストール ディレクトリ>\FM Stocks 2000\documentation フォルダにある "FMStocks2000_Setup.doc" を参照してください。

FMStocks_AutomatedUpgrade のセットアップ

アップグレード ウィザードによって生成された未加工コードは FMStocks_AutomatedUpgrade に格納されています。お手元の未加工コードをこのケース スタディで生成された未加工コードと比較したい場合は、ケース スタディで生成されたコードをコミュニティ サイトからダウンロードできます。未加工コードは FMStocks_AutomatedUpgrade.msi ファイルに格納されています。ファイルの内容のインストール先は C:\Program Files\ FMStocks_AutomatedUpgrade です。インストール ファイルを実行すると、アップグレード ウィザード レポートと未加工コードがインストールされ、自分のアップグレードの進捗状況と照らし合わせながらケース スタディを進められるようになります。

FMStocks_NET のセットアップ

このケース スタディの最後に完成する機能的に同等なコードは、FMStocks_NET に格納されています。このコードは GotDotNet コミュニティ サイトから入手できます。ファイル名は FMStocks_NET.msi です。このファイルを実行すると、FMStocks 2000 と機能的に同等なバージョンである FMStocks_NET がインストールされます。ファイルの内容のインストール先は C:\Program Files\FMStocks_NET フォルダです。評価ツールのレポートと、機能的に同等な最終コードがインストールされるため、このケース スタディの作成時に生成された結果と自分のアップグレード結果を比較することができます。同じフォルダに入っている FMStocks_NET_Setup.doc に、FMStocks_NET の詳しいセットアップ方法が記載されています。

FMStocks 2000 の評価と分析

FMStocks のアップグレードの主な目的は、同等の機能の実現とアップグレード プロセスの高速化でした。同等の機能を実現するためには、Visual Basic 6.0 バージョンの FMStocks の機能を分析する必要がありました。アップグレード プロセスの高速化という 2 つ目の目的を果たすには、自動アップグレードに加え、アップグレードの見積もり作業量やアップグレードに付随するタスク、リスク、問題を定義した具体的なアップグレード戦略とアップグレード計画が必要でした。アップグレードの詳細計画を作成するためには、Visual Basic 6.0 バージョンの FMStocks の内容を完全に分析する必要がありました。具体的には、アプリケーションの主要な

サブシステムとその相互関係、アプリケーションが提供する機能、アプリケーションで使用されているテクノロジ、完全なソースコード、外部システムおよびサードパーティライブラリに対する依存関係など、FMStocksの構造とデザインに関する分析が行われました。

同等の機能を実現するためのアップグレード計画とテスト計画の段階では、Visual Basic 6.0 バージョンのFMStocks の機能の分析とドキュメント化を行ううえでユース ケース分析が役立ちました。アプリケーションの完全な分析には Visual Basic 6.0 Upgrade Assessment Tool を使用しました。この評価ツールのおかげで、アップグレード計画を立案するためのデータを必要に応じて取得することができ、アップグレード プロセスの高速化という目的が達成しやすくなりました。

FMStocks の構造の概要

FMStocks 2000 の構造は、付属のマニュアル、評価ツールによる分析結果、およびアプリケーションに関する高度なコード レビューで確認しました。FMStocks 2000 は、プレゼンテーション層、ビジネス ロジック層、およびデータベース層の 3 層構成になっています。ビジネス ロジック層を構成するプロジェクトは以下のとおりです。

- FMStocks_Bus.vbp。このプロジェクトには、オンライン株式取引機能に対応するビジネス コードが入っています。Account、Broker、Ticker、および Version の 4 つのクラス モジュールで構成されます。
- FMSStore_Bus.vbp。このプロジェクトには、オンライン書店機能に対応するビジネス コードが入っています。Product および ShoppingCart の 2 つのクラス モジュールで構成されます。
- FMSStore_Events.vbp。オンライン書店機能の外部注文処理イベントを定義するインターフェイスです。
- FMSStore_EvtSub2.vbp。オンライン書店機能の外部注文処理イベントを実装するプロジェクトです。

データ アクセス層を構成するプロジェクトは以下のとおりです。

1. FMSStore_DB.vbp。このプロジェクトには、オンライン書店機能に対応するデータ アクセス コードが入っています。Product および ShoppingCart の 2 つのクラス モジュールで構成されます。
2. FMStocks_DB.vbp。このプロジェクトには、オンライン株式取引機能に対応するデータ アクセス コードが入っています。Account、Broker、DBHelper、Position、Ticker、Product、および ShoppingCart の 8 つのクラス モジュールで構成されます。

上記の各プロジェクトは、以下のプロジェクト グループにグループ化されます。

1. FMS2000_Core.vbg。このプロジェクト グループには、FMStocks_Bus.vbp と FMStocks_DB.vbp が含まれます。
2. FMS2000_Store.vbg。このプロジェクト グループには、FMSStore_Bus.vbp と FMSStore_DB.vbp が含まれます。
3. FMS2000_Events.vbg。このプロジェクト グループには、FMSStore_Events.vbp と FMSStore_EvtSub2.vbp が含まれます。

ユース ケースの概要

ユース ケース分析は、アップグレードとテストの際に同等の機能を実現する目的で FMStocks アプリケーションの機能仕様を分析し、ドキュメント化する過程で必要になりました。FMStocks の場合は、現在使用している Visual Basic 6.0 バージョンの FMStocks アプリケーションからユース ケースを導き出しました。このケース スタディはアプリケーションの機能向上を伴うものではないため、新しいユース ケースは作成しませんでした。FMStocks 2000 を構成するユース ケースは以下のとおりです。

- 既存ユーザーのログイン
- 新規ユーザーのログイン
- ログアウト
- 記号または会社の検索
- 口座残高の表示
- ポートフォリオの表示
- ポートフォリオ グラフの作成
- 株式の購入
- 株式の売却
- ショッピング カートの表示
- 商品の閲覧
- チェックアウト

表 1 は、FMStocks 2000 の株式売却機能のサンプル ユース ケースを表しています。

表 1: FMStocks 2000 のサンプル ユース ケース

ユース ケース ID	UC-09
名前	株式の売却
アクタ	アプリケーション ユーザー
説明	現在のポートフォリオを表示し、売却する株式と売却株数を選択するようにユーザーに求める。売り注文を出す。
トリガ	なし
前提条件	ユーザーはシステムにログオンしている必要がある。
事後条件	なし
標準フロー	1. ユーザーが [**株式の売却**] リンクをクリックする。 2. アプリケーションがユーザーの現在の持ち株一覧を表示する。 3. ユーザーが売却する株式とその数量を選択する。 4. アプリケーションが新しいトランザクションを作成して、関連するデータベース フィールドを更新する。 5. 売り注文の確認後、注文受領画面を表示する。
代替フロー	入力した株数が既存の株数よりも多い場合は、次のメッセージを表示する。 "Unable to place order. You tried to sell more shares than you own."
例外	なし
包含関係	なし
優先度	高
使用頻度	中
ビジネス ルール	なし
特殊要件	なし
仮定	なし
注意点	なし
ユース ケース ID	UC-09

FMStocks 2000 のユース ケースはドキュメント化されており、付属の CD に収録されています。

アップグレード インベントリの取得

FMStocks アプリケーションの分析で最も重要なステップは、アプリケーションに含まれるコンポーネント、クラス、モジュール、およびユーザー コントロールの数の統計を得ることでした。インベントリに関する情報は FMStocks の構造の複雑さを評価するうえで有用であり、それはさらに、アップグレード計画の作成や、時間と労力の見積もりに役立ちました。各プロジェクト グループのインベントリの分析には評価ツールを使用しました。評価ツールによって生成された DetailedReport.xls の Project Files Overview ワークシートには、各プロジェクトに含まれるファイルの一覧とその分類 (クラス、モジュールなど) が表示されています。分析した

FMStock プロジェクトはビジネス層コンポーネントまたはデータ アクセス層コンポーネントであったため、フォームやデザイナは含まれていませんでした。Project Files Overview ワークシートから収集した各プロジェクトのデータを表 2 ～ 4 に示します。実際のワークシートには、ユーザー コントロール、アップグレードできないファイル、デザイナ、およびフォームの各列が含まれることに注意してください。今回のワークシートではこれらの列に値が入っていないため、表には記載していません。

表 2: FMS2000_Core プロジェクト グループ

プロジェクト	ファイルの総数	クラス数	モジュール数
FMStocks_Bus	5	4	1
FMStocks_DB	9	8	1
合計	14	12	2

表 3: FMS2000_Store プロジェクト グループ

プロジェクト	ファイルの総数	クラス数	モジュール数
FMStocks_Bus	3	2	1
FMStocks_DB	3	2	1
合計	6	4	2

表 4: FMS2000_Events プロジェクト グループ

プロジェクト	ファイルの総数	クラス数	モジュール数
FMStocks_Events	1	1	0
FMStocks_EvtSub_OrderProc	1	1	0
合計	2	2	0

Project Files Overview ワークシートのデータにより、移行作業の中心となるプロジェクト グループが FMS2000_Store であることがわかりました。アップグレード計画およびテスト計画の最終決定にあたっては、この点を考慮に入れました。

サードパーティ コンポーネントのアップグレードは、アップグレード作業全体のかなりの割合を占めました。アップグレード ウィザードで一度にアップグレードできるプロジェクトは 1 つだけです。このため、1 つのプロジェクトをアップグレードする際に、元のバージョンのプロジェクトから参照される ActiveX コンポーネントが相互運用 DLL でラップされました。参照先の ActiveX コンポーネントも .NET にアップグレードされていたため、相互運用 DLL は必要ありませんでした。したがって、アップグレードしたプロジェクトごとに、ActiveX コンポーネントへの参照を手作業で修正して .NET コンポーネントへの参照にアップグレードする必要がありました。.NET コンポーネントと ActiveX コンポーネントでは名前とプログラム ID が異なるため、コードを 1 行ずつ確認し、呼び出しを ActiveX コンポーネントから .NET コンポーネントに変更する必要がありました。この作業

では、評価ツールによって生成された Third Party Components Summary レポートと Third Party Components Members レポートが役立ちました。Third Party Components Summary レポートには、インスタンス化されたサードパーティ コンポーネントとそのインスタンス数が表示されます。表 5 の情報は、FMS2000_Store プロジェクト グループに関する Third Party Components Summary レポートから得られたものです。

表 5: FMS2000_Store に関する Third Party Components Summary レポート

コンポーネント	数
FMSStore_Events.ShoppingCart	1
FMStocks_DB.DBHelper	2
ADODB.Recordset	2

このレポートは、FMS2000_Store プロジェクト グループが FMSStore_Events コンポーネントの ShoppingCart クラス モジュール、FMStocks_DB コンポーネントの DBHelper クラス モジュール、および ADODB ライブラリのレコードセットに依存していることを示しています。Third Party Components Members レポートには、コード内で使用されているサードパーティ コンポーネントのメンバとその数が表示されます。以下の表は、FMS2000_Store プロジェクト グループに関する Third Party Components Members レポートを示しています。

表 6: FMS2000_Store に関する Third Party Components Members レポート

メンバ	数
FMSStore_Events.ShoppingCart.ExecuteBuy	1
FMStocks_DB.DBHelper.RunSP	4
ADODB.Recordset.BOF	2
FMStocks_DB.DBHelper.RunSPReturnRS	6
ADODB.Recordset.EOF	2

これらのレポートを利用すると、サードパーティ コンポーネントの検索と、各コンポーネントのインスタンスに対する適切な変更を容易に行うことができます。

FMStocks 2000 のインベントリ情報の収集に利用したその他のレポートは、Data Access Components レポート、API Calls レポート、User Components Members レポート、Intrinsic Components Members レポート、および User Com Objects Members レポートです。これらのレポートの情報を表 7 ～ 10 に示します。

表 7: Data Access Components レポート

ADO データ連結	
コントロール名	**コントロールの種類**
データが見つかりません。	

使用されている ADO データ メンバ	
メンバ	**数**
ADODB.Command.CommandText	9
ADODB.Recordset.BOF	1
ADODB.Recordset.Open	5
ADODB.Recordset.Close	1
ADODB.Parameters.Append	8
ADODB.Recordset.AddNew	1
ADODB.Recordset.ActiveConnection	2
ADODB.Command.Execute	5
ADODB.Recordset.MoveNext	2
ADODB.Fields.Append	5
ADODB.Command.CreateParameter	8
ADODB.Recordset.Delete	1
ADODB.Recordset.Fields	5
ADODB.Recordset.MoveFirst	1
ADODB.Recordset.CursorLocation	5
ADODB.Command.Parameters	8
ADODB.Recordset.UpdateBatch	2
ADODB.Command.ActiveConnection	23
ADODB.Recordset.EOF	3
ADODB.Recordset.Value	5
ADODB.Command.CommandType	9

表 8: API Calls レポート

関数名	パラメータリスト	パラメータ数	ライブラリ	使用回数
RegisterEventSource			Advapi32.dll	2
DeregisterEventSource			Advapi32.dll	2
ReportEvent			Advapi32.dll	2
GetLastError			kernel32	0
CopyMemory			kernel32	2
GlobalAlloc			kernel32	2
GlobalFree			kernel32	2
GetComputerNameAPI			kernel32	2

表 9: User Components Members レポート

メンバ	数
FMStocks_DB.Version.Version	1
FMStocks_Bus.Helpers.NullsToZero	19
FMStocks_DB.Helpers.NullsToZero	2
FMStocks_DB.Position.ListForSale	1
FMStocks_Bus.FMStocks_TransactionType.Insufficient_Funds	2
FMStocks_DB.Version.ConnectionString	1
FMStocks_DB.Ticker.GetPrice	3
FMStocks_DB.DBHelper.RunSP	1
FMStocks_DB.DBHelper.RunSPReturnInteger	7
FMStocks_DB.Helpers.GetKey	1
FMStocks_DB.Helpers.GetVersionNumber	1
FMStocks_DB.DBHelper.GetConnectionString	2
FMStocks_DB.Helpers.GetValue	2
FMStocks_DB.Ticker.ListByTicker	1
FMStocks_DB.Account.GetAccountInfo	1
FMStocks_Bus.Helpers.RaiseError	13
FMStocks_DB.Helpers.mp	34
FMStocks_DB.Account.Add	1
FMStocks_DB.Position.ListSummary	1

メンバ	数
FMStocks_DB.TxNew.SetTxType	4
FMStocks_Bus.FMStocks_TransactionType.Not_Enough_Shares	2
FMStocks_DB.Account.Summary	1
FMStocks_DB.Position.ListForAdjustment	1
FMStocks_DB.Account.VerifyUser	1
FMStocks_DB.Version.ComputerName	1
FMStocks_DB.Broker.Buy	2
FMStocks_DB.Tx.AddBuyOrder	1
FMStocks_DB.Ticker.GetFundamentals	1
FMStocks_DB.Ticker.VerifySymbol	1
FMStocks_Bus.Helpers.GetComputerName	1
FMStocks_DB.DBHelper.RunSPReturnRS	8
FMStocks_DB.DBHelper.RunSPReturnRS_RW	1
FMStocks_DB.Helpers.RaiseError	30
FMStocks_DB.Broker.Sell	2
FMStocks_DB.Tx.AddSellOrder	1
FMStocks_DB.Tx.GetByID	1
FMStocks_Bus.Helpers.GetVersionNumber	1
FMStocks_DB.Ticker.ListByCompany	1
FMStocks_DB.Helpers.GetComputerName	1

表 10: User Com Objects Members レポート

メンバ	数
FMStocks_DB.DBHelper.RunSP	1
FMStocks_DB.DBHelper.RunSPReturnInteger	7
FMStocks_DB.DBHelper.GetConnectionString	2
FMStocks_DB.DBHelper.RunSPReturnRS	8
FMStocks_DB.DBHelper.RunSPReturnRS_RW	1

ソース コードのメトリクスの取得

FMStocks 2000 のアップグレード計画を作成し、作業時間と作業コストを見積もるには、各プロジェクトに含まれるソース コードの行数を知る必要がありました。ソース コードの行数の取得には評価ツールを使用しました。DetailedReport.xls の Lines of Code ワークシートには、各プロジェクトの行数と、各コード行の分類 (ビジュアル行、コメント行、空行、およびコード行) が表示されます。アップグレードするコンポーネントにはユーザー インターフェイスが含まれないため、ビジュアル行の行数はごくわずかでした。アップグレードのコストと労力の見積もりではコメント行と空行が無視されるため、コード行の分類が役立ちました。FMS2000_Core プロジェクト グループのコード行数を表すレポートの情報を表 11 に示します。

表 11: FMS2000_Core プロジェクト グループに関する Lines of Code レポート

プロジェクト	行の総数	ビジュアル行	コード行	コメント行	空行
FMStocks_Bus	900	32	556	94	218
FMStocks_DB	1118	64	715	82	257
合計	2,018	96	1,271	176	475

FMStocks 2000 の行の総数は 4,943 行です。

サポートされない機能の処理

アップグレード ウィザードでは、Visual Basic 6.0 のすべての言語機能とテクノロジが Visual Basic .NET にアップグレードされるわけではありません。これらの言語機能やテクノロジの中には、.NET でサポートされないものがあるからです。アップグレード ウィザードではアップグレードされない Visual Basic 6.0 の機能は、手動でアップグレードする必要があります。自動アップグレードした FMStocks 2000 に手作業で修正を加えて同等の機能を実現するのは、評価ツールがなければ容易ではありませんでした。評価ツールは、自動アップグレードに関する問題の検出と評価に効果を発揮しました。評価ツールの DetailedReport.xls には Upgrade Issues というワークシートがありますが、このワークシートには、アップグレード ウィザードでは Visual Basic .NET にアップグレードされない Visual Basic 6.0 の機能、これらの機能がアプリケーションで使用されている回数、および各機能のアップグレード コストが表示されます。同等の機能を実現するためにどの程度の手作業が必要になるかを評価するうえで、この情報が役に立ちました。

表 12: FMS2000_Core プロジェクト グループに関する Upgrade Issues テーブル

名前	MSDN ID	使用回数
パラメータ "As Any" の宣言はサポートされません。	1016	8
オブジェクト変数を Nothing に設定してもオブジェクトは破棄されません。ガベージ コレクトするまでこのオブジェクトを破棄することはできません。	1029	99
オブジェクトの既定のプロパティを解決できませんでした。	1037	46
.NET では、関数には新しい動作が含まれます。	1041	22
Null/IsNull() の使用が見つかりました。	1049	5
Visual Basic 6.0 と Visual Basic.NET との違いにより、一部のオブジェクトまたはコレクションをアップグレードできません。	2068	4
一部のオブジェクトをアップグレードできません。これらのオブジェクトのプロパティまたはメソッドはアップグレードされたコードにコピーされましたが、基になるオブジェクトがアップグレードされなかったためコンパイルされません。	2069	6

メモ： 実際のレポートの "Guidance topic" の値は、このガイドのオンライン版にリンクされます。各リンクをクリックすると、該当する問題の処理方法を説明する章が表示されます。

アプリケーションの依存関係の特定

アップグレード順序を定義し、アップグレード計画とテスト計画を作成するには、コンポーネントおよびプロジェクトの相互関係を知る必要がありました。プロジェクトおよびコンポーネントの依存関係と相互関係は、評価ツールを使用することで簡単に分析できました。評価ツールには、分析したプロジェクト グループごとにコール グラフと依存関係グラフを生成する機能があります。図 1 と図 2 は、FMS2000_Core プロジェクト グループについて生成された依存関係グラフとコール グラフをそれぞれ表しています。

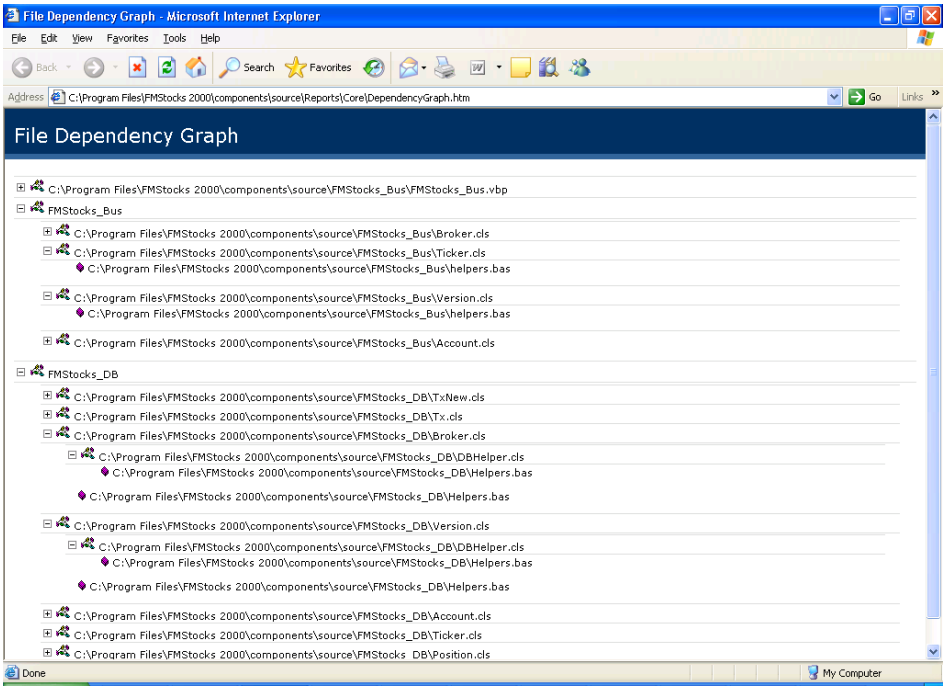

図 1

FMS2000_Core プロジェクト グループの依存関係グラフ

このファイル依存関係グラフには、FMStocks_Bus プロジェクトと FMStocks_DB プロジェクトに属するファイルが表示されています。FMStocks_Bus プロジェクトのファイルは、同じプロジェクトに属する Helpers.bas ファイルに依存しています。FMStocks_DB プロジェクトのファイルは、FMStocks_DB プロジェクトに属する Helpers.bas ファイルと DBHelper.cls ファイルに依存しています。

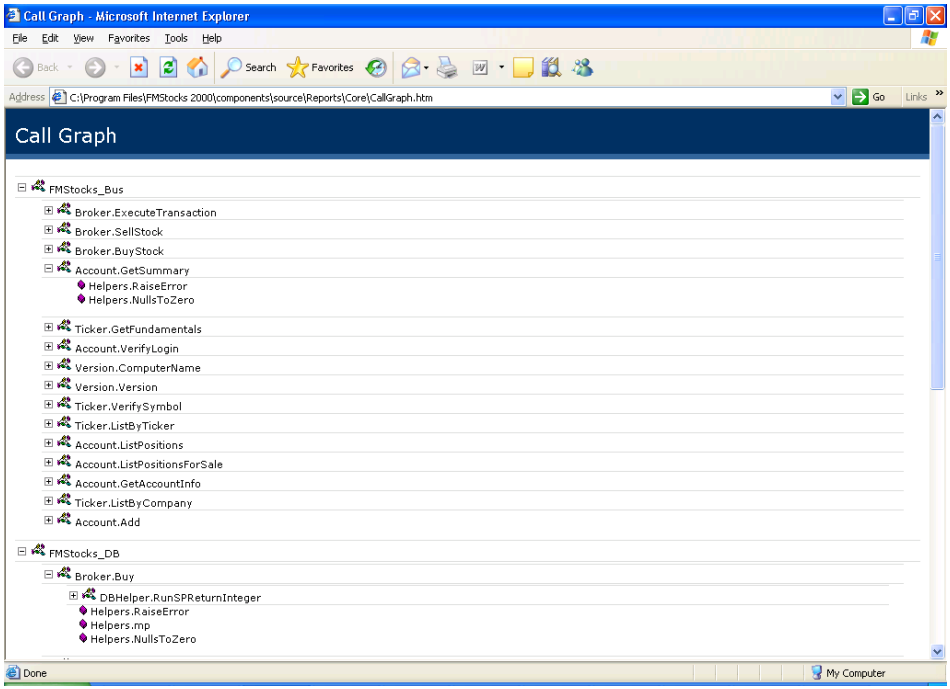

図 2

FMStocks_DB *プロジェクト*と FMStocks_Bus *プロジェクトのコール グラフ*

このコール グラフには、FMStocks_Bus プロジェクトと FMStocks_DB プロジェクトに属するクラス モジュールのメソッドが表示されています。このコール グラフでは、FMStocks_Bus プロジェクトの Account クラス モジュールの GetSummary メソッドが、Helpers モジュールの RaiseError メソッドと NullsToZero メソッドを呼び出しています。同様に FMStocks_DB プロジェクトでは、Broker クラス モジュールの Buy メソッドが、Helpers モジュールの RaiseError メソッド、mp メソッド、および NullsToZero メソッドに依存しています。

このようなファイル依存関係は、評価ツールでも特定されます。評価ツールでは、DetailedReport.xls に Upgrade Order レポートが生成されます。FMS2000_Core プロジェクト グループに関する Upgrade Order レポートの情報を表 13 に示します。

表 13: FMS2000_Core に関する Upgrade Order レポート

プロジェクト	依存グループ	ファイル
FMStocks_Bus	依存グループ 1	C:\Program Files\FMStocks 2000\components\source\FMStocks_Bus\Helpers.bas
FMStocks_DB	依存グループ 1	C:\Program Files\FMStocks 2000\components\source\FMStocks_DB\Helpers.bas

Upgrade Order レポートの情報は、ファイル依存関係グラフと同じデータを反映しています。FMStocks_Bus プロジェクト ファイルは Helpers.bas ファイルに依存しています。FMStocks_DB プロジェクトについても同様です。

標準的かつ直接的なアプローチは、他のコンポーネントやファイルに依存しないコア (基本) コンポーネントまたはファイルを最初にアップグレードし、続いてそれらに依存するコンポーネントやファイルをアップグレードすることです。前述の依存関係を基に FMStocks_Db プロジェクトと FMStocks_Core プロジェクトのアップグレード順序を定義すると、他のどのクラス モジュールやファイルよりも先に Helpers モジュールをアップグレードすることになります。

FMStocks 2000 アプリケーションの分析と評価のプロセスにかかった作業時間はおよそ 12 時間でした。これだけの労力で、32 ページの ASP、および 22 個の Visual Basic ファイルに含まれる 4,943 行の Visual Basic コードの分析と評価を終えることができました。

FMStocks 2000 のアップグレード

初期評価が終了したら、アップグレードの実際の作業を開始できます。ここでは、FMStocks 2000 を Visual Basic 6.0 から Visual Basic .NET にアップグレードする際に実行する手順について詳細に説明します。

アップグレードの計画

FMStocks 2000 のアップグレード計画段階は、サブ段階、つまり、アップグレード方法の決定、アップグレード順序の定義、およびコストや労力の見積もりを含むアップグレード スケジュールの作成で構成されていました。FMStocks 2000 の評価および分析段階で得られたデータをアップグレード計画段階の入力として使用しました。アップグレード計画段階では、約 4 時間の作業時間が必要でした。

アップグレード方法の決定

FMStocks 2000 のアップグレードでは、アップブレード方法として完全アップブレード方法を採用しました。完全方法を採用した理由は以下のとおりです。

- 完全アップグレードはすばやく低コストです。
- FMStocks 2000 は、22 個のファイル、3 つのプロジェクト グループ、および 6 つのプロジェクトで構成されており、その合計コード行数は 4,943 行です。ソース コードのメトリックやプロジェクト ファイルの概要から判断すると、FMStocks は明らかに中規模アプリケーションであり、配置前に完全にアップグレードできます。また、段階的アップグレードではなく完全アップグレードを選択したことに起因する問題発生の可能性も高くありません。

- Upgrade Issues レポートでは、Visual Basic .NET でサポートされていない 261 個の問題が報告されています。ただし、確認した問題は、廃止予定のテクノロジ (DAO、ROD、DDE など) に関連するものではありません。そのため、.NET に完全アップグレードしたアプリケーションを配置する際に問題が発生する可能性はほとんどありません。したがって、このアップグレード プロジェクトでは、個々のコンポーネントをアップグレードするたびにシステム テストを実施する必要はありません。結果的に、FMStocks は段階的アップグレードに適したアプリケーションではありません。
- 段階的アップグレードでは、Visual Basic 6.0 コードと .NET コードの間の相互運用のメカニズムを提供するために、ラッパーやインターフェイスを実装する必要があります。これらのラッパーはたいてい一時的なものであり、後に破棄されます。ラッパーの作成に伴うオーバー ヘッドが (特に COM コンポーネントに依存するラッパーの場合)、FMStocks 2000 の段階的アップグレードを魅力のない方法にしています。さらに、COM コンポーネントと .NET アセンブリとの相互運用性が必要な場合、アプリケーションの配置にはさらに作業と時間を必要とします。
- テスト方法としてテスト駆動型アップグレードを採用しました。つまり、アップグレード後に各コンポーネントの単体テストを実施しました。これにより、各コンポーネントのアップグレード後にシステム テストを実施する必要性が低下しました。

アップグレード順序の定義

完全アップブレード方法を採用したにもかかわらず、アップグレード後に各コンポーネントの単体テストを行う必要があったため、依然としてアップグレード順序を定義する必要がありました。アップグレード順序を定義するために、DetailedReport.xls のコール グラフ、ファイル依存関係グラフ、および Upgrade Order レポートの検討と、プロジェクト参照の検証を行いました。取得したデータは以下のとおりです。

- FMSStore_EvtSub_OrderProc プロジェクトは FMSStore_Events プロジェクトに依存します。
- FMSStore_DB プロジェクトは FMStocks_DB プロジェクトの DBHelper クラス モジュールに依存します。
- FMSStore_Bus プロジェクトは FMSStore_DB プロジェクトおよび FMSStore_EvtSub_OrderProc プロジェクトに依存します。
- FMStocks_Bus プロジェクトは FMStocks_DB プロジェクトに依存します。

他のコンポーネントやプロジェクトに依存しない基本コンポーネントやプロジェクトをアップグレード順序リストの最初にしました。次にアップグレードしたのは、基本コンポーネントやプロジェクトに依存するコンポーネントやプロジェクトです。依存するコンポーネントやプロジェクトの数が多いプロジェクトほど、アップグレード リストの後方になっています。

上記のデータとアップブレード方法を基に、プロジェクト レベルでのアップグレード順序を以下のように定義しました。

1. FMSStore_Events プロジェクト および FMStocks_DB プロジェクト
2. FMSStore_EvtSub_OrderProc プロジェクトおよび FMStocks_Bus プロジェクト
3. FMSStore_DB プロジェクト
4. FMSStore_Bus プロジェクト

アップグレード スケジュールの作成

FMStocks 2000 のアップグレード スケジュールを作成するには、以下の手順を実行する必要がありました。

- さまざまなアップグレード作業の特定
- 評価ツールによって生成された見積もりレポートの収集
- プロジェクト要件に合わせた労力とコストの見積もりレポートの修正

第 5 章「Visual Basic のアップグレード プロセス」で説明したように、FMStocks 2000 のアップグレードでは以下のアップグレード作業が特定されました。

- **アプリケーションの準備**。この手順では以下のタスクを実行します。
  - 開発環境の準備
  - コンパイルの確認
- FMStocks 2000 **のアップグレード**。この手順では以下のタスクを実行します。
  - アップグレード ウィザードの実行
- アップグレード ウィザード レポートの分析と、サポートされない機能の手動アップグレード
  - アップグレード管理タスクの実行
- **アップグレード後のアプリケーションのテスト**。この手順では以下のタスクを実行します。
  - テスト ケースの作成
  - 自動テストの作成
  - 単体テスト用のテスト ケースの実施
  - 機能等価性を実現するためのシステム テスト
  - バグの管理
  - 回帰テスト
  - バグの修正
  - テスト管理タスクの実行

次に実行したステップは、評価ツールによって生成された見積もりレポートからのデータの収集です。FMStocks2000_Core プロジェクト グループに関する見積もりレポートの情報を表 14 に示します。

表 14: FMStocks2000_Core の見積もりレポート

タスク	作業時間 (時間)	コスト	リソース
アプリケーションの準備			
開発環境	4	$200	DEV
アプリケーションのリソース インベントリ	0	$7	DEV
コンパイルの確認	2	$100	DEV
移行順序の定義	0	$3	DEV
アップグレード ウィザードのレポートの確認	0	$5	DEV
合計	6	$315	
アプリケーションの変換			
アップグレード ウィザードの実行	0	$3	DEV
コードの手動調整	18	$1,032	
システムの統合とスモーク テスト	4	$221	DEV
管理タスク	1	$60	DLE
合計	23	$1,316	
テストとデバッグ			
テスト ケースの作成	3	$210	STE
テスト ケースの実行	4	$160	TES
問題の解決	5	$234	TES
管理タスク	1	$77	STE
合計	13	$681	
プロジェクト管理	2	$165	**PM**
構成管理	1	$70	**CM**
合計	45	$2,547	

見積もりレポートは FMStocks 2000 用にカスタマイズされました。以下の構成値は、このレポートのカスタマイズ用に変更されています。

- Config – Fixed Tasks。元のアプリケーションのコンパイル確認時間が 8 時間から 2 時間に変更されました。これは 2,018 行のコードを含む単一のプロジェクト グループであるため、コンパイルの確認時間が削減されました。さらに、FMStocks 2000 は必要なすべての参照を含む作業アプリケーションでした。評価ツールによって生成された DetailedReport.xls から収集したデータによると、このプロジェクト グループが参照するサードパーティ コンポーネントは 1 つだけであり、不足しているコンポーネントやファイルはありませんでした。したがって、不足している参照が原因でコンパイル エラーが発生する可能性はほとんどありませんでした。こうした分析の結果、元のアプリケーションのコンパイル確認に要する見積もり時間が 8 時間から 2 時間に縮小されました。
- Config – Fixed Tasks。構成管理の時間が 8 時間から 1 時間に変更されました。プロジェクト グループのサイズ (2,018 行のコード)、完全アップグレード方法の選択、および特定されたアップグレード問題の数 (191) によって、ソース コードのコード チャーンに 1 時間の構成管理作業のみ必要であることが示されました。その結果、労力とコストの見積もりレポートがプロジェクト要件に合わせて変更されました。
- Config – Dependent Tasks。テスト ケースの実行の比率が 1% から 10 % に増加しました。これは、手動および自動の単体テストやその結果分析に加えて、機能等価性を実現するための機能テストをテストの実施作業で行う必要があったためです。

表 15 は、FMStocks 2000 アプリケーションを構成する 3 つのプロジェクト グループ (FMS2000_Core、FMS2000_Events、および FMS2000_Store) の累積した労力とコストの見積もりを示しています。どのようにしてこの累積レポートに至ったのかを示す重要なポイントのいくつかを以下に示します。

- 「アプリケーションの準備」の「開発環境」における 4 時間の作業全体は、FMS2000_Core プロジェクト グループで作成されるレポートに含まれます。そのため、残りのプロジェクト グループの開発環境の準備のための時間数は 0 時間になります。ここで、他のプロジェクト グループで開発環境の追加準備が必要ないと仮定しました。
- FMS2000_Events プロジェクト グループのサイズ (52 行のコード) がその他のプロジェクト グループと比べて非常に小さいため、このプロジェクト グループの構成値が大幅に変更されました。固定タスクのコンパイルの確認では、1 時間の作業時間が FMS2000_Events プロジェクト グループに割り当てられました。同様に、テスト ケースの実施の比率が 5% に変更されました。

表 15: 変更された見積もりレポート

タスク	作業時間 (時間)	コスト	リソース
アプリケーションの準備			
開発環境	4	$200	**DEV**
アプリケーションのリソース インベントリ	0	$7	**DEV**
コンパイルの確認	3	$150	**DEV**
移行順序の定義	0	$3	**DEV**
アップグレード ウィザードのレポートの確認	0	$5	**DEV**
合計	7	$365	
アプリケーションの変換			
アップグレード ウィザードの実行	0	$3	**DEV**
コードの手動調整	24	$1,443	
システムの統合とスモーク テスト	6	$293	**DEV**
管理タスク	1	$81	**DLE**
合計	31	$1,820	
テストとデバッグ			
テスト ケースの作成	4	$280	**STE**
テスト ケースの実行	5	$200	**TES**
問題の解決	7	$350	**TES**
管理タスク	2	$140	**STE**
合計	18	$970	
プロジェクト管理	3	$226	**PM**
構成管理	3	$210	**CM**
合計	66	$3,381	

この時点で判明した最も重要な点の 1 つは、プレゼンテーション層の ASP ページから ASP.NET へのアップグレードのための見積もりでした。ASP.NET への変換の必要があった ASP ページは 32 ページでした。この方法は、ASP to ASP.NET Migration Assistant を使用して自動アップグレードを実行することでした。ただし、一部の調査と見積もりは事前に以下のように行いました。

- ASP.NET Migration Assistant によってアップグレードされない機能を事前に特定する方法がなかったため、問題の機能を特定するためにすべての ASP コードを分析しました。この分析を基に、FMStocks の主要な機能を含む 5 つの ASP ページをサンプルとして選び、移行アシスタントでこれらのページをアップグレードしました。次に移行アシスタントによって生成された変換レポートを分析し、移行アシスタントからスローされる可能性のあるアップグレード問題を選択し、それらの複雑さを分析しました。アップグレード問題の最終的な見積もり数は 15 でした。
- 移行アシスタントがスローする可能性のある各問題の複雑さを分析した結果、問題 1 回当たりの解決に必要な平均作業時間は 15 分と測定されました。Visual Basic のアップグレード問題と比べて 1 回の作業時間を高く設定した理由の 1 つは、Visual Basic アップグレード ウィザード レポートとは異なり、ASP to ASP.NET Migration Assistant Conversion レポートではアップグレードの問題の発生したコード行が特定されないためです。したがって、問題の発生箇所を特定するための追加作業が必要になります。

この分析と、ASP ページから ASP.NET へのこれまでのアップグレード経験を基に、表 16 に示す見積もりを算出しました。

ASP から ASP.NET への移行に関する見積もりについて注意すべき点は以下のとおりです。

- 付録 C で説明したように、ASP to ASP.NET Migration Assistant はエラーの発生しやすいツールであるため、ランタイム エラーの修正時間の値を高く設定しました。
- Visual Basic コードのコンパイル確認タスクの代わりに、ASP ページの実行確認タスクを追加しました。このタスクでは、ASP ページの実行、ASP ページ内のリンクや URL、および適切な COM コンポーネントへの呼び出しを確認します。

FMStocks 2000 アプリケーション全体の最終的なコストと労力の見積もりは以下のとおりです。

- 合計見積もり作業時間：90.5 時間
- 合計見積もりコスト：$4,723

表 16: FMStocks 2000 ASP ページを ASP.NET にアップグレードするためのコストと労力の見積もり

タスク	作業時間 (時間)	コスト	リソース
アプリケーションの準備			
開発環境	0	$0	**DEV**
アプリケーションのリソース インベントリ	1	$50	**DEV**
ASP ページの実行の確認	2	$100	**DEV**
合計	3	$150	
アプリケーションの変換			
移行アシスタントの実行	0	$3	**DEV**
コードの手動調整	4	$200	
システムの統合とスモーク テスト	2	$100	**DEV**
管理タスク	0.5	$45	**DLE**
合計	6.5	$345	
テストとデバッグ			
テスト ケースの作成	3	$140	**STE**
テスト ケースの実行	4	$80	**TES**
バグ管理	1	$50	**TES**
回帰テスト	1	$50	**TES**
ランタイム エラーの修正	7	$307	**DEV**
管理タスク	1	$70	**STE**
合計	13	$697	
プロジェクト管理	1	$80	**PM**
構成管理		1	$70
合計	24.5	$1,342	

アプリケーションの準備

FMStock 2000 アプリケーションのアップグレード準備では、以下のタスクを実行しました。

1. 開発環境の準備：FMStocks 2000 のセットアップのドキュメントに従って、IIS 5.0 や SQL Server 2000 などの FMStocks 2000 で必要なソフトウェアをインストールし、次に FMStocks 2000 をインストールしました。Visual Studio .NET 2003、.NET Framework 1.1、および Visual Basic 6.0 Upgrade Assessment Tool もインストールしました。さらに、Microsoft Excel 2003 などの評価ツールで必要なソフトウェアをインストールしました。
2. コンパイルの確認および ASP ページの実行の確認：FMStock 2000 をインストールした後、ビジネス コンポーネントのソース コードもインストール フォルダにコピーしました。次にビジネス コンポーネントのプロジェクト グループがコンパイルされて、登録されているかどうかを確認しました。次に ASP ページを検証するため、登録した COM コンポーネントに対して ASP ページを実行しました。

FMStocks 2000 の準備段階では、合計で約 7 時間の作業時間が必要でした。

自動アップグレードの実行

実際のアップグレード プロセスの最初の手順は、自動アップグレード手順から始まります。ここでは、FMStocks 2000 のアップグレードにおけるこの経験内容について説明します。

アップグレード ウィザードによる Visual Basic 6.0 から Visual Basic .NET へのアップグレード

自動アップグレードの実行には、Visual Studio .NET 2003 の Visual Basic アップグレード ウィザードを使用しました。このケース スタディの前半で定義したアップグレード順序に従ってプロジェクト ファイルをアップグレードしました。アップグレード ウィザードを使用してプロジェクトをアップグレードした後、次のプロジェクトをアップグレードする前に、アップグレードしたプロジェクトを手作業で修正して単体テストを実施しました。

図 3 は、FMSStore_Bus プロジェクトのアップグレード ウィザード レポートを示しています。

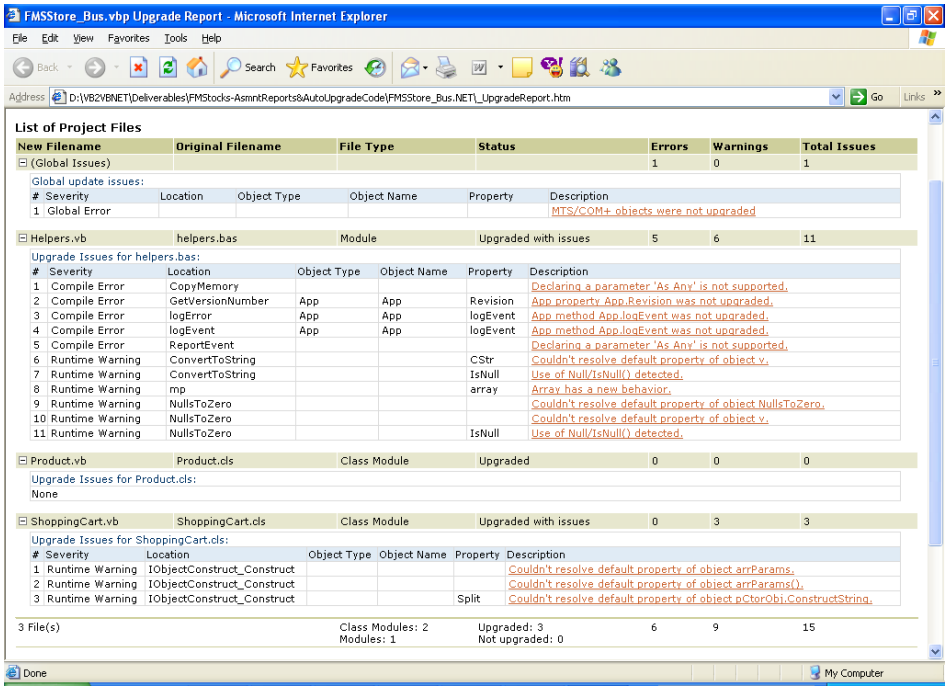

図 3

FMSStore_Bus *プロジェクトのアップグレード ウィザード レポート*

移行アシスタントによる ASP から ASP.NET へのアップグレード

FMStocks 2000 の自動アップグレードの最終段階では、ASP to ASP.NET Migration Assistant を使用して ASP ページ プロジェクトを ASPX ページにアップグレードしました。

図 4 は、ASP to ASP.NET Migration Assistant によって生成された変換レポートを示しています。

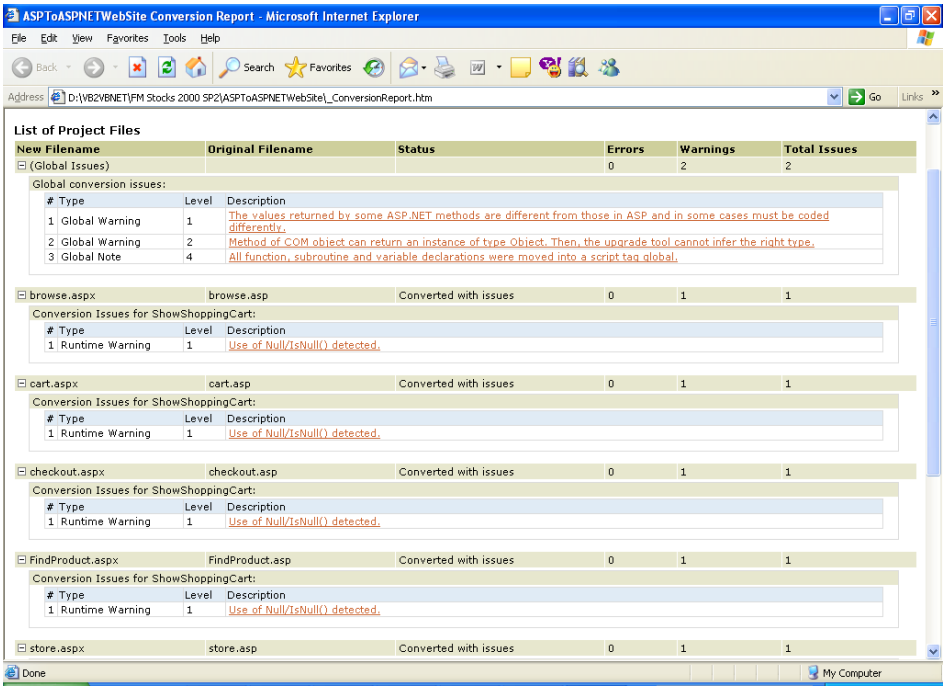

図 4

FMStocks 2000 ASP ページの ASP to ASP.NET Migration Tool Conversion Report

Visual Basic アップグレード ウィザードと ASP to ASP.NET Migration Tool の両方のアプリケーションを含む、自動アップグレードには、約 1 時間の作業が必要でした。

アップグレードの手動調整の適用

自動アップグレード手順を実行した後、次のプロジェクトをアップグレードする前に、各プロジェクトを手作業でアップグレードしました。問題の特定には、自動アップグレード段階で生成されたアップグレード ウィザード レポートを使用しました。『Upgrading Visual Basic 6.0 Applications to Visual Basic .NET and Visual Basic 2005』ガイドを基に、アップグレード問題の解決策を決定しました。適切な解決策の特定には、評価ツールによって生成された DetailedReport.xls の Upgrade Issues レポートも使用しました。このレポートには、アップグレード ウィザードによって特定された問題の大部分が表示されます。また、問題の解決策が詳細に説明されているガイド内の各セクションへのリンクも提供されます。このようにレポートを使用することによって、アップグレード問題を簡単に検出して解決できました。問題を解決した後、アップグレードされたコードを見直してクリーン アップを行いました。(自動アップグレード時にアップグレード ウィザードによって挿入された) コメント行を変更または削除し、アップグレード バージョンの FMStock 2000 に合わせて (変数名などの) 識別子を変更しました。

ASP ページの手動アップグレードでは、ASP to ASP.NET Migration Assistant はアップグレード問題の検出および解決のガイドとしてあまり役に立ちませんでした。そのため、移行アシスタントによってアップグレードされた ASPX コードを行単位ですべて見直し、アップグレード問題を特定および解決する必要がありました。

メモ : FMStocks 2000 のビジネス COM+ コンポーネントを Visual Basic 6.0 から .NET にアップグレードしたとき、COM+ コンポーネントの ProgID が自動アップグレード中に変更されました。たとえば、Visual Basic 6.0 の FMStocks_Bus.Account コンポーネントが Visual Basic .NET の Account_NET.Account にアップグレードされました。ただし、こうした変更を ASPX ページに反映させる必要がありました。手動アップグレード時に ASPX ページの各 ProgID を手作業で変更することが必要でした。

FMStocks 2000 の手動アップグレードでは、合計で約 38 時間の作業時間が必要でした。ASPX ページをすべて見直して、アップグレード問題を手動で特定および解決するための追加作業が必要になりました。

機能テスト

アップグレードした .NET バージョンの FMStocks 2000 をテストする目的は、機能等価性を検査することでした。FMStocks 2000 のテストでは、以下のテスト プロセスを実行しました。

- テスト計画およびテスト ケースの作成
- 単体テスト
- ブラック ボックス テスト
- ホワイト ボックス テスト

FMStocks のアップグレードを開始する前に、単体テストおよびブラック ボックス テスト用のテスト計画とテスト ケースを作成しました。テスト計画の作成に必要なデータは、ユース ケース分析と FMStock 2000 の元の Visual Basic 6.0 コードから取得しました。

単体テストでは、アップグレードしたプロジェクトの各クラスに含まれるパブリック メソッドをテストして、アップグレードしたプロジェクトの各メソッドの動作が元のアプリケーションの対応するメソッドの動作と一致していることを確認しました。各パブリック メソッドのテスト ケースでは、有効範囲内の値、境界値、有効範囲外の値の順に、さまざまな値が入力パラメータとしてメソッドに渡されます。NUnit を使用して単体テスト用のテスト ケースをすべて自動化しました (Nunit による単体テストの詳細については、Nunit の Web サイトを参照してください)。表 17 は、FMSStore_Cart.ShoppingCart クラスのテスト計画を示しています。

表 17: FMSStore_Cart.ShoppingCart **クラスの単体テストのテスト計画**

シナリオ 1		Class FMSStore_Cart.ShoppingCart
優先度		高
コメント		
1.1	高	Public Function GetByKey(ByVal AccountID As Integer, ByVal SKU As Integer) As ADODB.Recordset
1.2	高	Public Sub Add(ByVal AccountID As Integer, ByVal SKU As Integer)
1.3	高	Public Sub Buy(ByVal AccountID As Integer)
1.4	高	Public Function ListByAccount(ByVal AccountID As Integer) As ADODB.Recordset
1.5	高	Public Sub SetQuantity(ByVal AccountID As Integer, ByVal SKU As Integer, ByVal Quantity As Short)
1.6	高	Public Function TotalByAccount(ByVal AccountID As Integer) As ADODB.Recordset
1.7	高	Public Sub EmptyShoppingCart(ByVal AccountID As Integer)

表 18 は、FMSStore_Cart.ShoppingCart クラスの単体テスト用のサンプル テスト ケースを示しています。これ以外に "テストは OK か (Y/N)" という名前の列がありますが、エントリがないため表に載せていません。

表 18: FMSStore_Cart.ShoppingCart **クラスのサンプル テスト ケース**

テスト ケース	優先度	テストする条件	実行の詳細	必要なデータ
1.1a	高	Public Function GetByKey(ByVal AccountID As Integer, ByVal SKU As Integer) As ADODB.Recordset - すべてのパラメータが有効な範囲内。	すべての入力パラメータが有効な範囲内。 **入力パラメータ:** AccountID = 5249 SKU = 1004009 **予測される出力:** フィールドに次の値を持つ Recordset Quantity = 1 SKU = 1004009 Price = 29.95 Description = The Secrets of Investing in Technology Stocks **実際の出力:**	NUnit テスト ケース: GetByKey_Valid

テスト ケース	優先度	テストする条件	実行の詳細	必要なデータ
1.1b	高	Public Function GetByKey(ByVal AccountID As Integer, ByVal SKU As Integer) As ADODB.Recordset - AccountID パラメータが有効な範囲より大。	AccountID パラメータが有効な範囲より大。 **入力パラメータ:** AccountID = 5250000 SKU = 1004009 **予測される出力:** 空の Recordset **実際の出力:**	NUnit テスト ケース: GetByKey_AccountInValid

ブラック ボックス テストの目的は、アプリケーション レベルでの機能等価性を検査することです。その結果、FMStocks 2000 アプリケーションのユース ケース分析からブラック ボックス テスト用のテスト計画とテスト ケースを取得しました。表 19 は、ログイン ユース ケースのブラック ボックス テスト用のテスト計画を示しています。

表 19: ログイン ユース ケースのブラック ボックス テスト用テスト計画

シナリオ 1		既存ユーザーのログイン手順の機能をテストする
優先度		高
コメント		
1.1	高	ログインとホーム ページの両方が正しい形式で表示されることをテストする。つまり、すべてのリンク、フォント、コンテンツが既存の Visual Basic 6.0 アプリケーションと同じになっている。
1.2	高	ユーザーが有効な電子メールの名前とパスワードを入力した場合に、ユーザーがホーム ページにリダイレクトされることをテストする。
1.3	高	SQL Server が稼動していない場合に、"Cannot open database connection." というメッセージが 表示されることをテストする。
1.4	高	入力された電子メールの名前またはパスワードが無効である場合に、"Invalid e-mail and password combination.Please try again." というメッセージが表示されることを 検証する。

表 20 は、ログイン機能とユース ケースのブラック ボックス テスト用のサンプル テスト ケースを示しています。テスト ケースには、"必要なデータ"、"実際の結果"、"テストは OK か (Y/N)" という列も含まれています。ここではこれらの情報がないため、表に載せていません。

表 20: ログイン機能のブラック ボックス テスト用テスト ケース

テスト ケース	優先度	テストする条件	実行の詳細	期待される結果
1.1	高	ログインとホーム ページの両方が正しい形式で表示されることをテストする。つまり、すべてのリンク、フォント、コンテンツが既存の Visual Basic 6.0 アプリケーションと同じになっている。	ログインとホーム ページを Visual Basic 6.0 バージョンの FMStocks 2000 の対応するページと比較し、次の点を検証する。 コンテンツ Web ページの全般的なルックアンドフィール 入力フィールドの場所 フォント 他ページへのリンク 機能	ユーザーが、アップグレード後のバージョンと Visual Basic 6.0 バージョンの FMStocks のログインとホーム ページの間で、ルック アンド フィール、フォント、機能、コンテンツなどの違いを感じない。
1.2	高	ユーザーが有効な電子メールの名前とパスワードを入力した場合に、ユーザーがホーム ページにリダイレクトされることをテストする。	ログイン ページで次の電子メールの名前とパスワードを入力する。 電子メール : ta450 パスワード : ta	ユーザーはサイトに入力でき、ホーム ページにリダイレクトされる。

ホワイト ボックス テストの目的は、ブラック ボックス テストで検出されなかったコード内のエラーのシナリオを見つけて、アップグレード問題を適切に解決することでした。ホワイト ボックス テストは、以下のタスクで構成されていました。

- エラーのシナリオを検出するためのコード レビュー (特に、ブラック ボックス テストで検出されなかった可能性のある、ループ、条件ステートメント、および COM コンポーネントとの相互運用性の場合)
- 例外処理やログ記録が適切に実装されているかどうかを確認するためのコード レビュー
- 単体テスト用の追加テスト シナリオを検出するためのコード レビュー
- アップグレード問題が適切に解決されているかどうかのレビュー
- 問題を発生させる可能性のある中間結果を見つけるためのコードのプロファイリングとテスト

以降では、単体テストおよびブラック ボックス テストの実行時に発生したエラーや問題について説明します。

適切にアップグレードされなかったインターフェイス

FMStocks 2000 の Visual Basic 6.0 コードでは、FMStore_Events プロジェクトの ShoppingCart クラス モジュールで以下のようなインターフェイスが定義されています。

```
' これはイベント インターフェイスであり、インターフェイスの定義のみを目的としているため、
' 実装は行われていません。
Public Sub ExecuteBuy(ByVal AccountID As Long)
End Sub
```

これは、本体のないメソッド定義だけの空のクラス モジュールとして、Visual Basic 6.0 でインターフェイスを定義する方法です。.NET では、Interface キーワードを使用してインターフェイスを定義します。ただし、アップグレード ウィザードでこのコードをアップグレードした場合、アップグレード ウィザードは ShoppingCart クラス モジュールを他のクラス モジュールと区別できないため、以下のようにしてアップグレードされます。

```
Option Strict Off
Option Explicit On

<System.Runtime.InteropServices.ProgId(_
    "ShoppingCart_NET.ShoppingCart")> Public Class ShoppingCart

 ' これはイベント インターフェイスであり、インターフェイスの定義のみを目的としているため、
 ' 実装は行われていません。

 Public Sub ExecuteBuy(ByVal AccountID As Integer)
 End Sub
End Class
```

FMStocks 2000 の元のコードでは、FMStore_EvtSub2 プロジェクトの ShoppingCart.cls クラス モジュールで以下のようにインターフェイスが実装されています。

```
Option Explicit

Implements FMSStore_Events.ShoppingCart

Private Sub ShoppingCart_ExecuteBuy(ByVal AccountID As Long)
    Dim objSC As FMSStore_Bus.ShoppingCart

    Set objSC = New FMSStore_Bus.ShoppingCart
    objSC.EmptyShoppingCart AccountID

End Sub
```

ただし、アップグレードした .NET コードでは、以下に示すように、同一のクラス モジュール内でインターフェイスが定義および実装されます。

```
Option Strict Off
Option Explicit On

Public Interface _ShoppingCart
End Interface

<System.Runtime.InteropServices.ProgId( _
    "ShoppingCart_NET.ShoppingCart")> _
Public Class ShoppingCart
    Implements _ShoppingCart
    Implements _ShoppingCart

    Private Sub ShoppingCart_ExecuteBuy(ByVal AccountID As Integer)
        Dim objSC As _ShoppingCart

        objSC = New FMSStore_Bus.ShoppingCart
        objSC.EmptyShoppingCart(AccountID)
    End Sub
End Class
```

アップグレード時に Response に変更された Request

FMStocks 2000 の元のコードには、t_head.asp ファイルに以下のような ASP コードが記述されています。

```
if Request.Cookies("Account") <> "" then
g_AccountID = Request.Cookies("Account")
else
g_AccountID = 0
end if
```

このコードは、ASP to ASP.NET Migration Assistant によって t_head.aspx ファイル内の以下のコードにアップグレードされました。

```
If Not IsNothing(Request.Cookies.Item("Account")) Then
g_AccountID = IIf(IsNothing(Response.Cookies.Item("Account").Value), "",
Request.Cookies.Item("Account").Value)
Else
g_AccountID = 0
End If
```

このアップグレード コードを実行したところ、「入力文字列の形式が正しくありません」というエラーが発生しました。これは、Account の値が Request オブジェクトではなく Response オブジェクトから取得されたためです。したがって、上記のコードを以下のように修正しました。

```
If Not IsNothing(Request.Cookies.Item("Account")) Then
g_AccountID = IIf(IsNothing(Request.Cookies.Item("Account").Value), "",
Request.Cookies.Item("Account").Value)
Else
g_AccountID = 0
End If
```

ByRef でパラメータを渡すときに発生する例外

ASPX ページの分離コード ファイルでは、パラメータが参照で渡された場合に一部の関数が例外をスローします。たとえば、PortFolio.aspx の Item(ByRef text As String, ByRef align As String) 関数は、プロバイダがこの関数をサポートしていない、またはこの関数が読み取り専用であることを示す例外をスローします。

これらのパラメータを参照ではなく値で渡すように変更することで、この問題を解決しました。

レコードセットの型形式例外

アップグレードした ASPX ページで、recordset("<fieldname>") を Response.Write と共に使用するか、または割り当てステートメント内で使用すると、「型形式例外」が発生しました。Response.Write または割り当てステートメントの recordset("<fieldname>") を recordset("<fieldname>").Value に変更することで、この問題を解決しました。

Response.Cookies.Item("Account").Value

t_head.asp ファイルには以下のような ASP コードが表示されます。

```
<% if Request.Cookies("Account") <> "" then  %>
```

このコードは、ASP.NET Migration Assistant によって、以下のような ASP.NET コードに変換されました。

```
<%If Not IsNothing(Request.Cookies.Item("Account")) Then%>
```

この変換により、動作に変更がありました。Default.aspx ページでは t_head.aspx ページをインクルードしています。Logout.aspx では、対応する ASP ページと同様に、Account アイテムの値が空の文字列 ("") に設定されます。

```
Response.Cookies.Item("Account").Value = ""
```

Account アイテムの値を "" に設定した後、Logout.aspx ページはユーザーを Default.aspx ページにリダイレクトします。Default.aspx ページでは、Account アイテムの値 "" が条件ステートメント IsNothing() で検証されます。ただし、テスト中、Account アイテムの値 "" が比較条件に渡されたため、例外がスローされました。以下に示すように、Logout.aspx で Account アイテムの値を "" ではなくゼロに設定するように変更することで、この問題を解決しました。

```
Response.Cookies.Item("Account").Value = 0
```

他の ASPX ページにリダイレクトするときに発生した Err.Number 5

アップグレードした FMStocks 2000 の ASPX ページがユーザーを別の ASPX ページにリダイレクトしたときに、エラー (Err.Number=5, Err.Description="Thread was being aborted.") が発生しました。

エラー ハンドラ ブロックでこのエラーがキャッチされたため、リダイレクト先の ASPX ページの代わりにエラーメッセージが表示されました。

エラーの原因は、次の行に示す変換された ASPX ページにありました。

```
If Err.Number <> 0 Then
```

この問題を解決するために、上記の行を以下のように修正しました。

```
If Err.Number <> 0 and Err.Number <> 5 Then
```

FMStocks 2000 の機能テストでは、合計で約 46 時間の作業時間が必要でした。ASPX ページのランタイムエラーを修正するために、追加作業が必要になりました。

まとめ

FMStocks 2000 の完全アップグレードには、合計で約 95 時間の作業時間を要しました。最終的なアップグレード コストは $5,023 でした。

コストが増加した原因は、移行アシスタントを使用して ASP ページから ASP.NET にアップグレードするための追加作業が必要になったため、ASP ページから ASPX ページへのアップグレード コストが見積もりコストを $300 超過したことでした。この作業は Visual Basic Upgrade Assessment Tool で評価されなかったため、評価ツールが生成したレポートにコストが反映されませんでした。ASP ページのアップグレードで手作業による追加作業が必要になったにもかかわらず、ASP ページを ASPX ページに完全に書き直す場合と比べると、移行アシスタントを使用したアップグレードは、依然として作業時間の短縮になっています。

評価ツール、アップグレード ウィザード、および Visual Basic 6.0 to Visual Basic.NET ガイドを併用することにより、Visual Basic 6.0 バージョンの FMStocks 2000 アプリケーションを首尾よく効率的に Visual Basic .NET と ASP.NET にアップグレードできることが最終的に証明されました。

詳細情報

NUnit による単体テストの詳細については、NUnit の Web サイトを参照してください。
URL は http://www.nunit.org です。