LÊ MINH TRÍ

Microsoft DOT NET

Kỹ năng lập trình ứng dụng với Visual Basic .NET

NHÀ XUẤT BẢN THỐNG KÊ

Microsoft DOT NET

Kỹ năng lập trình ứng dụng với

VISUAL BASIC .NET

LÊ MINH TRÍ

Microsoft DOT NET

Kỹ năng lập trình ứng dụng với

VISUAL BASIC .NET

NHÀ XUẤT BẢN THỐNG KÊ

Microsoft DOT NET

Kỹ năng lập trình ứng dụng với

VISUAL BASIC .NET

Lê Minh Trí

Chịu trách nhiệm xuất bản:
CÁT VĂN THÀNH

Biên tập:
HẠNH NGUYÊN

Trình bày:
NGỌC DIỆP

Bìa:
THANH LONG

In 1 000 cuốn khổ (14.5x20.5)cm tại Xưởng In Trung Tâm Hội Chợ Triển Lam Việt Nam. Giấy phép XB số 192-104/XB-QLXB do Cục Xuất Bản cấp ngày 30/01/2002. In xong và nộp lưu chiểu tháng 9 năm 2002.

MỤC LỤC

Chương 1

Microsoft .NET LÀ GÌ

Tài liệu này được soạn thảo cho các lập trình viên Visual Basic cần chủ động tăng tốc độ khởi tạo Microsoft .NET mới và nhất là cách tạo các ứng dụng trong VB.NET.

Giáo trình này được soạn thảo cho các lập trình viên đã từng sử dụng Visual Basic 6.0 hoặc phiên bản trước đây và bây giờ cần chuyển sang .NET Framework mới với VB.NET. Bạn sẽ tìm hiểu các phương pháp mới về tạo các ứng dụng Windows và Web trong VB.NET. Các khái niệm hướng đối tượng sẽ được trình bày có phương pháp cũng như cách giải quyết nhiều vấn đề của thế giới thực. Sự nổi bật của giáo trình này là việc thực tập viết mã thuần thục, chẳng hạn như các chuẩn đặt tên được chấp nhận về mặt công nghiệp, cách thụt dòng và khả năng dùng lại mã. Tài liệu này được biên soạn tập trung vào sự phát triển ứng dụng doanh nghiệp. Nếu bạn là một lập trình viên ứng dụng cần tìm hiểu cách kết hợp ứng dụng Visual Basic.NET ngay, thì đây là giáo trình đáp ứng cho bạn.

Để có thể sử dụng sách này một cách hiệu quả, bạn cần có một ít kinh nghiệm về Visual Basic. Bạn cũng cần có một máy tính chạy hệ điều hành Microsoft Windows 2000 hoặc phiên bản sau của nó và tất nhiên bạn cần có bản copy của Microsoft .NET CLR và Visual Studio.NET.

Microsoft .NET

- Tìm hiểu về Microsoft .NET Framework
- Tìm hiểu về Common Language Runtime

TỔNG QUAN VỀ MICROSOFT .NET

Nền Microsoft .NET là một cách phát triển các ứng dụng hoàn toàn mới. Mục đích của Microsoft trong việc tạo nền phát triển mới này là để bảo đảm không có vấn đề trở ngại nào về ngôn ngữ bạn sử dụng và nó đều có thể phát triển ứng dụng Internet cũng như Windows mở rộng và đáng tin cậy một cách nhanh chóng và dễ dàng.

Cách Microsoft hoàn thành điều này là phát triển một tập các dịch vụ để các lập trình viên có thể sử dụng, chẳng hạn như một tập các dịch vụ hệ thống phong phú, engine thời gian chạy, framework phát triển và các control phong phú sẽ giúp họ thiết kế cả ứng dụng Windows lẫn Internet.

Điểm chủ yếu của nền .NET được gọi là CLR (Common Language Runtime). Engine vào thời gian chạy này chứa tất cả các kiểu dữ liệu chung và cấu trúc ngôn ngữ lõi mà bất kỳ ngôn ngữ nào cũng cần. Cuối cùng CLR sẽ được phân phối với hệ điều hành Windows, vì vậy nhu cầu thiết lập tích hợp và phức tạp sẽ không còn trong các phiên bản tương lai của Windows. CLR là

framework phát triển hỗ trợ nhiều ngôn ngữ và cung cấp các dịch vụ vào thời gian thiết kế cũng như dịch vụ vào thời gian chạy. Các dịch vụ vào thời gian thiết kế này bao gồm Integrated Development Environment (IDE) và một trình gỡ rối tinh xảo. Hình 1 cho bạn một cái nhìn tổng quát về các thành phần khác nhau của nền .NET so với một nền khác.

Hình 1: Nền .NET

Mức thấp nhất trong nền này là hệ điều hành Windows 32-bit. Trên nó là các dịch vụ chung được lập sẵn trong hệ điều hành mà bạn có thể tương tác. Chúng bao gồm các dịch vụ Message Queuing, COM+ các giao dịch, Internet Information Server (IIS) và Windows Management Instrumentation (WMI). Common Language Runtime trong nền .NET đặt một trình

wrapper chung quanh các dịch vụ này, vì vậy tất cả các ngôn ngữ có thể tương tác với chúng bằng cách dùng một giao diện chung. Ở trên CLR là một Base Framework của các lớp, bao gồm ADO.NET và ASP.NET. ADO.NET có thể được dùng từ cả các WinForm lẫn WebForm.

Kế đến là một loạt các ngôn ngữ được viết phù hợp với Common Language Specification mà CLR được thiết lập. Các ngôn ngữ này gồm hầu hết các ngôn ngữ thông dụng hiện nay, như Visual Basic, VC++, C# (một ngôn ngữ mới), Perl, Python, Eiffel và nhiều ngôn ngữ khác. Khi bạn làm việc với các ngôn ngữ này, hầu như bạn đều có thể sử dụng Visual Studio.NET để cung cấp một IDE tốt cho việc phát triển sử dụng .NET framework.

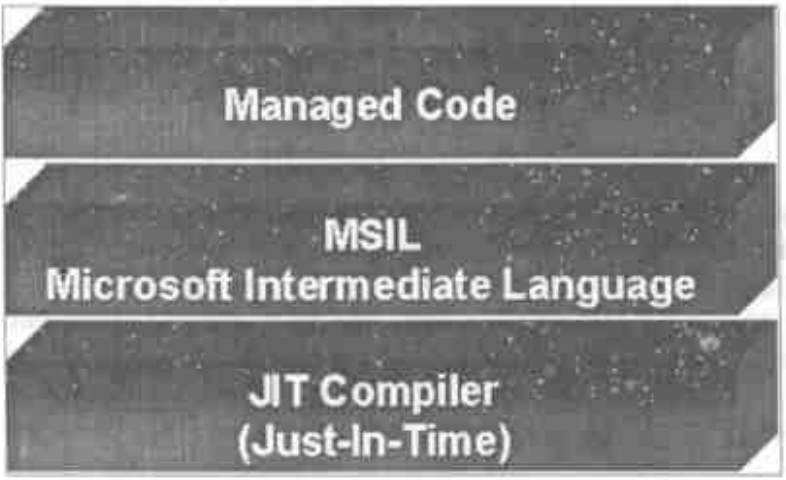
Đặc tả ngôn ngữ chung

Common Language Specification (CLS) là một giao ước cho biết ngôn ngữ tương tác với CLR sẽ ứng xử như thế nào. Khi các ngôn ngữ sử dụng CLS này, chúng được xem là thực hiện "managed code". Managed code là một tập các cấu trúc ngôn ngữ mà tất cả các ngôn ngữ .NET đều phải có, bao gồm các kiểu dữ liệu, xử lý lỗi, metadata, vân vân.

Managed code là mã có thể được trình biên dịch tách thành một tập "bytecodes", được coi như Microsoft Intermediate Language (MSIL hoặc IL). IL này được lưu trữ cùng với một số metadata về chính chương trình vào một file Portable Executable (EXE hoặc DLL).

NGÔN NGỮ TRUNG GIAN

Ngôn ngữ trung gian (IL) được tạo sao cho trình biên dịch có thể phiên dịch chương trình thành mã cuối cùng có thể được hệ điều hành (OS) và phần cứng diễn dịch. Đây chính là lý do khiến cho nó dễ dàng tạo một trình biên dịch cho bất kỳ hệ điều hành và nền phần cứng nào. Do đó, về lý thuyết .NET có thể làm việc với Unix, Linux hoặc bất kỳ một nền khác.



Hình 2: Managed Code tách thành IL và sau đó được JIT biên dịch

CÁC DỊCH VỤ TRONG .NET

Có nhiều dịch vụ mà nền .NET cung cấp cho người phát triển. Một số dịch vụ đáng kể nhất như sau:

- Memory management

- Threads
- Garbage collection
- Exception handling
- Security
- Application isolation
- Data Interaction
- Encryption Services
- Deployment

Trước đây để có tất cả các dịch vụ tinh xảo này, bạn phải tự xây dựng chúng hoặc phải sử dụng một ngôn ngữ mức cao. Khi tự xây dựng bạn phải mất nhiều thời gian về các mục này và cuối cùng để có một chương trình thực bạn phải mất thời gian gấp đôi. Còn khi sử dụng một ngôn ngữ mức cao, bạn phải gỡ bỏ hệ điều hành để khai thác mạch hoặc nếu có thể sử dụng ngôn ngữ đó thì bảo mật gặp khó khăn.

Với CLR, tất cả các dịch vụ trên đều có thể được dùng từ một ngôn ngữ bất kỳ để làm việc với CLR. Các ngôn ngữ có thể khai thác các dịch vụ này. Vì vậy, bây giờ sự chọn lựa ngôn ngữ của bạn không còn là vấn đề lớn, vì dù sao mọi thứ đều được biên dịch thành cùng Intermediate Language.

Hệ thống kiểu chung

Các kiểu dữ liệu trong .NET không còn được định nghĩa bằng từng ngôn ngữ, thay vào đó chúng thuộc một phần lõi của CLR.

Mỗi kiểu dữ liệu chính là một đối tượng. Mỗi ngôn ngữ có sự chọn lựa để thực thi tất cả chúng hoặc chỉ một tập con.

Các kiểu dữ liệu

Bảng 1 liệt kê các kiểu dữ liệu khác nhau và cho biết các ngôn ngữ nào thực thi từng kiểu dữ liệu ấy.

Kiểu dữ liệu	Mô tả	Sử dụng ngôn ngữ
Byte	Số nguyên không dấu 8 bit	Visual Basic-Byte C#-Byte VC++ - Char
Sbyte	Số nguyên có dấu 8 bit	Visual Basic – NA C# - sbyte VC++ - Signed char
Int16	Số nguyên có dấu 16 bit	Visual Basic – Short C# - short VC++ - short
Int32	Số nguyên có dấu 32 bit	Visual Basic – Integer C# - Internet C++ - int hoặc long
Int64	Số nguyên có dấu 64 bit	Visual Basic – Long C# - long VC++ - _int64
Unint16	Số nguyên không dấu 16 bit	Visual Basic – NA C# - ushort VC++ - unsigned short
Unint32	Số nguyên không dấu 32 bit	Visual Basic – NA C# - uint VC++ - unsigned int hoặc unsigned long
Unint64	Số nguyên không dấu 64 bit	Visual Basic – NA C# - ulong VC++ - unsigned __int64
Single	Số dấu chấm động 32 bit	Visual Basic – Single C# - float VC++ - float

Double	Số dấu chấm động 64 bit	Visual Basic – Double C# - double VC++ - double
Boolean	True (1) hoặc False (0)	Visual Basic – Boolean C# - bool VC++ - bool
Object	Kiểu cơ sở của một lớp bất kỳ hoặc kiểu dữ liệu	Visual Basic – Object C# - object VC++ - Object
Char	Kí tự Unicode (16 bit)	Visual Basic – Char C# - Char VC++ - __wchar_t
String	Chuỗi các kí tự Unicode	Visual Basic – String C# - string VC++ - String
Decimal	Giá trị thập phân 96 bit	Visual Basic – Decimal C# - decimal VC++ - Decimal

***Bảng 1:** Kiểu dữ liệu.*

Các kiểu ứng dụng .NET

Với nền .NET bạn có thể xây dựng nhiều kiểu ứng dụng khác nhau. Bên dưới là danh sách các kiểu ứng dụng khác nhau:

- Windows Applications
- Windows Services
- Web Applications
- Web Services
- Class Libraries

- WinForms Custom Controls
- WebForms Custom Controls

CÁC ỨNG DỤNG WINDOWS

Ứng dụng Windows sử dụng một client UI (giao diện người dùng) phong phú chạy trên hệ điều hành Windows. Nó sử dụng engine WinForms để tạo giao diện người dùng. Các ứng dụng Windows rất thích hợp cho các ứng dụng CSDL (cơ sở dữ liệu) Client/Server, chương trình đồ họa và điều khiển số. Hoặc một chương trình bất kỳ cần có một giao diện người dùng phong phú.

WINFORMS

Windows Form là engine form mới để phát triển Windows. Bạn có thể sử dụng WinForms với một ngôn ngữ CLR bất kỳ. WinForms có thể chuyển từ một ngôn ngữ này sang một ngôn ngữ khác tương đối dễ dàng. WinForms đi cùng với một tập các control chuẩn được lập sẵn và bạn cũng có thể xây dựng các control riêng.

WINDOWS SERVICES

Windows Service là một ứng dụng chạy dưới Windows NT hoặc Server Windows 2000 hay Advanced Server. Các kiểu ứng dụng này thường không có giao diện người dùng, thay vào đó chúng được dùng để thực hiện một dịch vụ liên tục không có bất kỳ tương tác nào từ người dùng. Hệ điều hành sẽ bắt đầu ứng dụng Windows Service khi hệ điều hành khởi động. Bạn nên

cung cấp một context bảo mật cho dịch vụ này. Bạn có thể tạo các kiểu ứng dụng rất dễ dàng với .NET.

WEB APPLICATIONS

Web Application là một tập các trang HTML tương tác chạy các chương trình trên Web server. Các ứng dụng Web có thể có giao diện người dùng (UI) hoặc không.
ASP.NET

ASP.NET là phiên bản tăng cường của ASP. Nếu bạn đã lập trình bằng ASP trước đây, bạn sẽ thấy chuyển sang lập trình ASP.NET rất dễ. Việc chuyển các ứng dụng cũ của bạn có thể thấy hơi khó khăn, vì vậy bạn nên viết lại khi tiến hành. ASP và ASP.NET có thể chạy cùng nhau trong IIS.

ASP.NET cung cấp các dịch vụ được cải thiện nhiều, chẳng hạn như:

- Quản lý khai báo phiên tốt hơn.
- Xử lý các form web tốt hơn.
- Hiệu suất mạnh hơn 2 tới 3 lần ASP.
- Sử dụng mã đã biên dịch.
- Kĩ thuật lập cahce tốt hơn.

ASP.NET có cả các WebForm lẫn các WebService để phát triển ứng dụng UI hoặc ứng dụng không có UI.

WEB FORMS

WebForms là thành phần tương đương Internet của WinForms. Bạn có thể xây dựng một giao diện người dùng đích Internet bằng cách dùng WebForms. WebForms có thể sử dụng HTML chuẩn bằng cách dùng các control HTML lập sẵn trong IDE của Visual Studio hoặc bạn có thể sử dụng các control phía máy chủ để phát sinh HTML cho bạn.

Các control phía máy chủ có khả năng mềm dẻo hơn, bao gồm khả năng nối kết nguồn dữ liệu và phát sinh nhiều HTML, nếu không bạn sẽ phải mất thời gian viết mã. Các control phía máy chủ có thể phát sinh HTML 3.2 chuẩn hoặc có các đuôi mở rộng IE cho giao diện người dùng phong phú hơn. Khi bạn sử dụng các control phía máy chủ, chúng có giao diện lập trình phong phú mà bạn có thể đáp ứng.

WEB SERVICE

Web Service sẽ trình bày các phương thức có thể được một giao diện HTTP gọi. Vì các phương thức này có thể được HTTP gọi, một ngôn ngữ bất kỳ và một hệ điều hành bất kỳ có thể gọi các phương thức này. XML và SOAP là các công cụ bạn dùng để gọi các dịch vụ này. Tuy nhiên, với các công cụ .NET, bạn chỉ cần sử dụng các lớp lập sẵn và xây dựng các Web Service. Bạn không bao giờ thấy hoặc sử dụng các đặc tả XML hoặc SOAP.

Có nhiều thứ bạn có thể làm với các Web Service, bao gồm:

- Gửi trở lại quyền sử dụng thẻ tín dụng
- Gửi trở lại tình trạng hàng gửi

- Gửi trở lại các xác nhận đặt hàng
- Gửi trở lại bản yết giá chứng khoán
- Gửi trở lại thông tin danh mục/sản phẩm

ADO.NET

Một trong các dịch vụ mới được lập sẵn trong .NET framework là một kiểu truy xuất dữ liệu hoàn toàn mới. Kiểu dữ liệu mới này được gọi là ADO.NET. Dù nó cùng tên với phương thức truy xuất dữ liệu cũ được gọi là ADO, nhưng rất khác nhau. Bạn có thể nhận biết một số đối tượng tương tự Connection và Command trong ADO, nhưng chúng có các đặc tính và phương thức khá khác nhau bạn cần cảnh giác.

Sự khác biệt lớn nhất giữa hai kiểu là cách dữ liệu được lưu trữ trong bộ nhớ sau khi đọc nó từ CSDL. Trong ADO, các Recordset được lưu trữ ở định dạng nhị phân. Trong ADO.NET, bây giờ chúng được lưu trữ như XML. Một sự thay đổi lớn khác của ADO.NET là không cần kết nối. ADO.NET không duy trì bất kỳ các kết nối nào mở sau khi đọc dữ liệu vào.

Có một số đối tượng mới, chẳng hạn như ADOCommand, ADODataSetCommand, ADODataReader và DataSet. Đối tượng Recordset không còn và tất cả các bản ghi được đọc từ đối tượng ADODataReader hoặc DataSet.

DataSets cho bạn đọc ở một số bảng vào một đối tượng. Sau đó bạn có thể thiết lập các mối quan hệ giữa các bảng trong bộ nhớ, bổ sung, hiệu chỉnh và xóa dữ liệu trong các bảng này, rồi lưu trữ dữ liệu trở lại CSDL. Schema cho các bảng này cũng được đọc và lưu trữ ở định dạng XML.

ADO.NET có khả năng đọc và ghi XML sẵn có. Ngoài ra, bạn có thể sử dụng OLEDB hoặc các phần mềm sẵn có để đọc dữ liệu từ các nguồn dữ liệu, chẳng hạn như SQL Server, Oracle hoặc Access.

Security Service

Bảo mật trong .NET lan tỏa khắp tất cả framework. Bảng 2 trình bày một số kiểu bảo mật khác nhau mà bạn có thể thiết lập cho các ứng dụng.

Kiểu bảo mật	Mô tả
Access Control	Các đối tượng bảo mật, chẳng hạn như các file, các khóa đăng ký và các đối tượng dịch vụ thư mục.
Security Support Provider Interface	Thiết lập các kết nối có xác nhận.
Logon Authenticated	Lọc mật khẩu, đăng nhập Windows và xác nhận bảo mật cục bộ
Certificate Services and Components	Cấp và quản lý các giấy chứng nhận
Cryptography	Crypto API
Smart Card	Xác nhận của Smart-Card
Policy Management	Thiết lập bằng lập trình và quản lý chính sách bảo mật cục bộ

Bảng 2: *Có nhiều tùy chọn bảo mật trong .NET.*

CÁC NGÔN NGỮ CHO .NET

Có nhiều ngôn ngữ có thể dùng cho nền .NET. Chúng bao gồm:

- Visual Basic
- C#
- VC++
- JScript
- Java – (Của Rational Corp.)
- Eiffel
- Smalltalk
- Cobol
- Perl
- Nhiều ngôn ngữ khác ...

Tại sao di trú sang .NET

Dù có lẽ bạn không muốn cố di trú bất kỳ ứng dụng cũ nào của mình, nhưng chắc chắn bạn muốn phát triển các ứng dụng mới với .NET.

.NET thực sự khai thác tốt nhất tất cả các công nghệ bạn đã sử dụng cho tới nay và cải tiến chúng. Các công nghệ, chẳng hạn

như COM, Internet, XML và SOAP, và các ngôn ngữ RAD, như Visual Basic.

.NET cung cấp một cơ sở hạ tầng tốt để phát triển. Common language Runtime cung cấp các dịch vụ mà tất cả các ngôn ngữ phát triển có thể sử dụng. Ngoài ra, .NET framework có khả năng chuyển sang các nền khác. Sự phát triển cho các thiết bị di động cũng khá tốt.

Còn có nhiều tính năng tích hợp giữa các ngôn ngữ, chẳng hạn như tính thừa kế, gỡ rối, các kiểu dữ liệu chung, Windows form và Web form. Việc tạo các ứng dụng web chưa bao giờ thuận tiện hơn với .NET và các khả năng nền chéo bằng cách dùng Web Services tạo truyền thông với các ứng dụng khác Windows mạnh.

Việc cài đặt và gỡ bỏ các chương trình bây giờ dễ dàng hơn nhiều so với COM. Việc cài đặt có thể được thực hiện bằng cách dùng chỉ một lệnh copy và gỡ cài đặt đơn giản như xóa các file. Đương nhiên, điều này xảy ra khi .NET framework và ngôn ngữ chung vào thời gian chạy đã được cài đặt rồi.

Các lợi ích của việc sử dụng .NET

.NET cung cấp các ưu điểm cho mọi người quan tâm đến tiến trình phát triển ứng dụng.

ĐỐI VỚI NGƯỜI DÙNG

Họ có thể nhận thông tin nhanh hơn bằng cách dùng Web Services và bằng cách cộng tác qua Internet. Thông tin cá nhân của họ có thể được lưu trữ vào một CSDL trung tâm, nhưng họ có

thể truy xuất nó từ máy tính ở nơi làm việc hoặc máy tính gia đình.

ĐỐI VỚI NGƯỜI PHÁT TRIỂN

.NET sẽ cho phép họ phát triển các kiểu ứng dụng mà người dùng đòi hỏi nhanh hơn trước đây. Với một loạt các công cụ hệ thống thông dụng có thể làm việc với một phạm vi các thiết bị rộng, chúng sẽ có thể viết mã một lần và cho chuyển giao trên một phạm vi thiết bị rộng.

Thay vì phải tìm hiểu các công cụ phức tạp, chẳng hạn như XML và SOAP, .NET sẽ bao bọc các giao diện thành một giao diện dễ dàng sử dụng cấu trúc hướng đối tượng. Điều này sẽ tăng nhanh việc tìm hiểu cho các người phát triển mới trong khi cho phép các người phát triển cấp cao phát triển các ứng dụng web mất ít thời gian hơn trước đây.

Vì việc cài đặt dễ dàng hơn nhiều, nó cũng tăng nhanh thời gian đưa các ứng dụng của họ ra thị trường.

ĐỐI VỚI NHÀ QUẢN LÝ

Các nhà quản lý luôn luôn phải đối diện với quá nhiều project, không có đủ thời gian và không có đủ lập trình viên thực hiện các project này. Vì hầu hết các dịch vụ hạt nhân sẵn có, các lập trình viên sẽ có thể tạo các ứng dụng nhanh hơn trước đây, do đó, cho phép nhiều project được hoàn tất.

Một môi trường hợp tác rộng, họ cũng có thể phải đối diện với sự thử thách về tích hợp nhiều hệ thống khác nhau. Việc sử dụng XML có thể giúp các hệ thống khác nhau này truyền thông qua

lại. Nếu các hệ thống này truyền trên Internet, thì việc vượt khỏi bức tường lửa của môi trường hợp tác với giao thức tư hữu hoặc dữ liệu nhị phân có thể gặp khó khăn nếu xảy ra. Việc sử dụng XML trong tất cả các phiên truyền trong .NET framework sẽ bảo đảm bạn có thể truyền thông giữa các hệ thống này bằng một chuẩn mở, như HTTP.

Khi các công ty phát triển, nhu cầu mở rộng một ứng dụng trở thành vấn đề. Khả năng mở rộng của .NET được lập sẵn trong tất cả các dịch vụ cốt lõi. Việc lập mạch qua tất cả các ngôn ngữ cũng sẽ trợ giúp khả năng mở rộng này.

Quá trình đào tạo luôn luôn là một vấn đề khi các người phát triển cấp cao sử dụng các công nghệ mới hoặc tư hữu, sau đó các người phát triển mới đưa vào để duy trì các hệ thống này. Cách tiếp cận OOP (Lập trình hướng đối tượng) của .NET tạo cho việc tìm hiểu các đối tượng này dễ dàng hơn nhiều.

Bảo mật là một lãnh vực khác có thể sẽ giúp các nhà quản lý an tâm. Có nhiều kiểu bảo mật khác nhau sẽ thỏa đáng hầu hết các nhu cầu của mọi người dùng.

Cách di trú

Dù bạn không muốn di trú các ứng dụng cũ, nhưng họ cố tạo cho điều đó dễ dàng. Thí dụ, các DLL dựa trên COM có thể được gọi từ các ứng dụng .NET của bạn. Nếu bạn đã phát triển bằng cách dùng một kiến trúc DNA, bạn sẽ thấy tiến trình di trú khá đơn giản.

Đối với các ứng dụng Windows của bạn, các control ActiveX vẫn được hỗ trợ, nhưng không phải tất cả chúng đều sẽ hoạt

động trong .NET framework . Các vấn đề này phải được xử lý từng trường hợp.

Đối với các người phát triển Visual Basic, có một Migration Wizard sẽ cố chuyển đổi mã cũ của bạn thành Visual Basic.NET. Tuy nhiên, bạn vẫn phải sửa đổi nhiều mã thủ công. Wizard sẽ chỉ xử lý một tác vụ phổ biến hơn. Thực ra, bạn cần viết lại các đoạn mã nào đó để tận dụng các công cụ mới.

Có cần viết lại ứng dụng không?

Trong hầu hết các trường hợp, bạn nên để nguyên các ứng dụng cũ ở ngôn ngữ chúng đã được viết ban đầu. Tuy nhiên, nếu bạn thấy một ứng dụng đã bắt đầu tiến hành như một ứng dụng mức khởi đầu, nay cần được mở rộng, bạn nên viết lại kiểu ứng dụng này.

Như chúng ta biết Internet đã làm thay đổi xã hội và hoạt động kinh doanh. Nay bạn cần quan tâm tới tất cả các chương trình "toàn cầu" và "phân phối". Việc thực hiện này với các công cụ chúng ta có hiện nay không dễ dàng, vì vậy trong các trường hợp này, bạn nên di trú các ứng dụng cũ của bạn. Trong hầu hết các trường hợp, bạn thấy có thể xóa bỏ nhiều đoạn mã của bạn và thay thế chúng bằng một vài đối tượng trong .NET framework mới.

.NET framework và IDE của Visual Studio.NET cả hai đều có thể tải xuống từ web site của Microsoft ở www.microsoft.com.

Tóm tắt

Nền .NET sẽ thay đổi cách bạn phát triển các ứng dụng trong một thời gian dài. Đây là một sự biến đổi lớn cho hầu hết các

người phát triển. Nếu bạn đã phát triển bằng cách dùng các thành phần COM và rất thuận tiện với kĩ thuật OOP, sau khi bạn chuyển tiếp sang nền mới này sẽ tương đối tiện lợi.

Câu hỏi ôn tập

1. Điểm chủ yếu của nền .NET là gì?
2. Tất cả các ngôn ngữ .NET được biên dịch thành ngôn ngữ gì?
3. Đúng hoặc Sai: .NET có thể truy xuất COM+ Services trong Windows 2000?
4. Các kiểu dữ liệu được định nghĩa theo từng ngôn ngữ hoặc theo ngôn ngữ chung vào thời gian chạy?
5. Dịch vụ web là gì?

TRẢ LỜI CÂU HỎI ÔN TẬP

1. Microsoft Intermediate Language.
2. Đúng.
3. Theo CLR.
4. Một loạt các phương thức trình bày giao diện HTTP.

Chương 2

Visual Studio .NET

- Tìm hiểu để khởi động một project mới với Visual Studio .NET.
- Tìm hiểu cách cấu hình môi trường phát triển của bạn.

Visual Studio .NET

Để khai thác Visual Studio .NET, bạn cần biến đổi nó thích hợp với kiểu làm việc của bạn. Với nhiều tùy chọn cấu hình khác nhau, vừa quen thuộc lẫn mới lạ, bạn nên khảo sát một số tùy chọn khác nhau bạn có thể thiết lập.

Trong tài liệu này, nhiều cấu hình khác nhau được đề cập và bạn sẽ tìm hiểu về các kiểu thiết lập khác nhau trong Visual Studio.NET. Bạn cũng sẽ tìm hiểu về các kiểu cửa sổ khác nhau,

bao gồm các cửa sổ Tool, có thể được chốt giữ vào môi trường hoặc thả trôi và tìm hiểu về các cửa sổ Document.

Kiểu Project	Mô tả
Windows Application	Template để tạo ứng dụng desktop Windows thông thường.
Class Library	Template để tạo các lớp (class) sẽ được dùng trong các ứng dụng khác. Giống như DLL.
Control Library	Template để tạo các control sẽ được dùng trên WinForms.
ASP.NET Web Application	Template để tạo Web site với trang HTML tĩnh hoặc trang động như giao diện người dùng.
ASP.NET Web Service	Template để tạo Web Services có thể được gọi bằng các giao diện XML SOAP.
Web Control Library	Template để tạo các control riêng của bạn sẽ được gọi từ các ứng dụng Web.
Console Application	Template tạo các ứng dụng dòng lệnh.
Windows Service	Template để tạo dịch vụ riêng của bạn chạy trong môi trường Windows.
Empty Project/Empty Web Project	Tạo project rỗng mà bạn có thể dùng để xây dựng bất kỳ khi nào bạn muốn. Các template sẽ không được cung cấp.

Bảng 1: *Các kiểu project bạn có thể tạo với Visual Studio.NET.*

Các template Project

Có các kiểu project khác nhau bạn có thể tạo với Microsoft .NET. Visual Studio.NET cung cấp cho bạn với template khởi đầu sẽ làm cho công việc tạo một Project mới rất dễ dàng. Bảng 2 liệt kê các kiểu project khác nhau mà bạn có thể thực hiện với VS.NET.

Bảng 1 cung cấp danh sách về một số tùy chọn trên màn hình My Profile:

Trường	Mô tả
Profile	Xác lập trường này là kiểu phát triển bạn sẽ thường làm việc nhất. Bạn được tùy chọn trong số Visual Studio, Visual Basic, Visual C++, Visual InterDev hoặc Visual C#.
Keyboard Scheme	Thay đổi các phím mặc định mà bạn dùng để thực hiện các tác vụ nào đó bằng cách xác lập tùy chọn này là Visual Studio Default, Visual Basic 6, VC++ 2, VC++ 6 hoặc Visual Studio 6.
Window Layout	Xác lập bố trí cửa sổ mặc định là Visual Studio Default, Visual Basic 6, VC++6, Minimal Window Layout hoặc No Tools Window Layout.
Help Filter	Xác lập ngôn ngữ lập trình mặc định bạn muốn thấy ở các thí dụ trong Help.
Show Help	Chọn xem Help trong một cửa sổ tách riêng hoặc được tích hợp vào IDE.
At Startup Show	Khi bạn bắt đầu một thể nghiệm Visual Studio.NET mới, bạn có tùy chọn trình bày Visual Studio Home Page, Most Recent Solution, hộp thoại Open Project hoặc hộp thoại New Project. Bạn cũng có thể để nó mặc định là một project IDE rỗng.

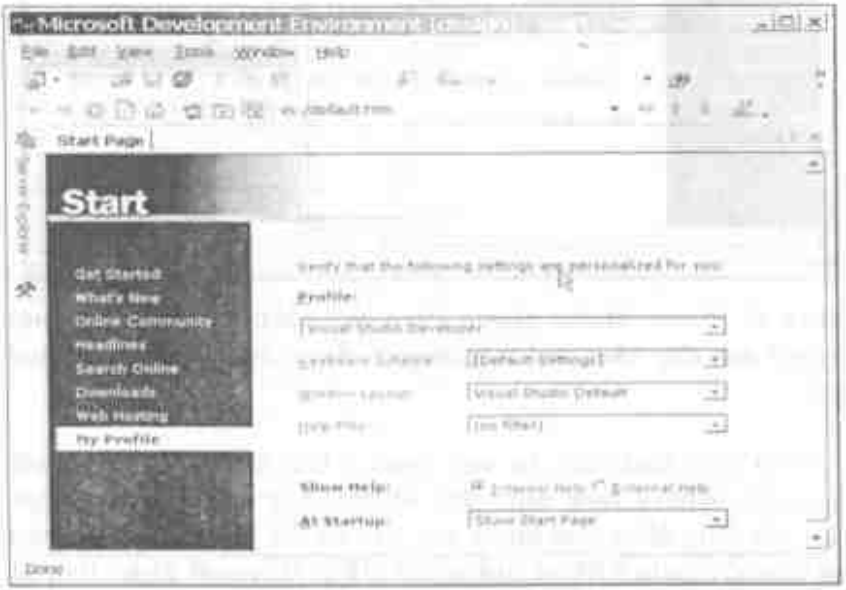
Bảng 1: *Các trường trên màn hình My Profile.*

Bắt đầu với VS.NET

Lần đầu tiên bạn sử dụng Visual Studio.NET, bạn sẽ được nhắc về một số thông tin cấu hình mà bạn sẽ thường dùng môi trường này nhất. Hình 1 trình bày một thí dụ về màn hình My Profile.

MÀN HÌNH MY PROFILE

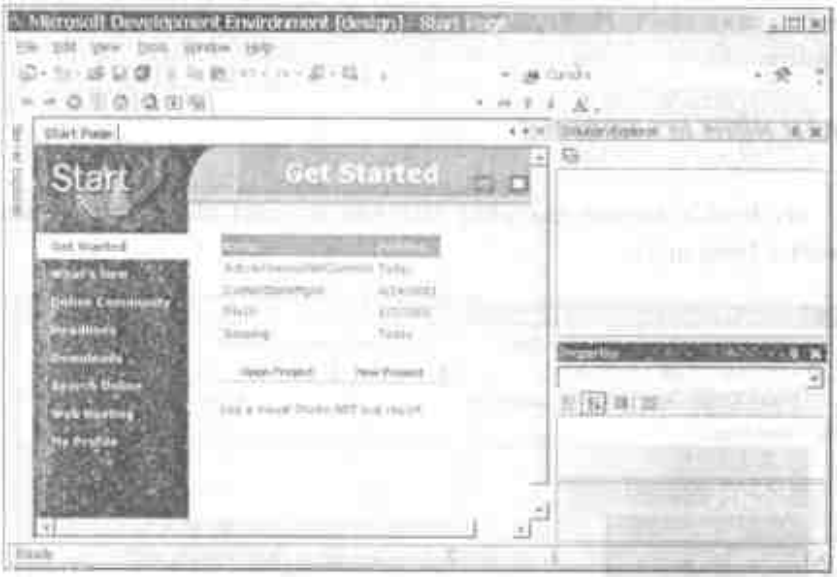
My Profile Screen cho phép bạn xác lập một số mặc định môi trường tổng quát.



Hình 1: *Xác lập cấu hình trên màn hình My Profile.*

VISUAL STUDIO HOME PAGE

Nếu bạn chọn bắt đầu trang là Visual Studio Home Page, bạn sẽ thấy màn hình như hình 2.



Hình 2: *Visual Studio Home Page cho phép bạn bắt đầu một project gần đây nhất, mở một project sẵn có hoặc tạo một project mới.*

Trên màn hình này, có một menu ở bên trái để bạn liên kết với menu What's New help. Bạn có thể thấy một danh sách các liên kết cộng đồng trực tuyến, nơi bạn có thể nhận được trợ giúp với Visual Studio.NET và nhiều sản phẩm Microsoft khác. Bạn có thể nhận các thông tin Headlines for MSDN và bạn có khả năng tìm MSDN site về những thông tin liên quan tới Visual Studio. Bạn cũng có thể xác lập profile của bạn.

Ở thân của form này, bạn có thể lựa một project mới đây, tạo một project mới, mở một project sẵn có hoặc ghi một báo cáo lỗi của Visual Studio.NET.

Tạo một Project mới

Nếu bạn chọn File ➤ New ➤ Project từ menu Visual Studio.NET, bạn sẽ thấy một hộp thoại như ở hình 3. Khi kết hợp một ứng dụng vào Visual Studio.NET, bạn có thể có nhiều project. Tập các project kết hợp với nhau tạo thành những gì được gọi là Solution.

Hình 3: *Hộp thoại New Project cho phép bạn tạo một file Solution mới của một kiểu project riêng biệt.*

Ở bên trái màn hình này, bạn có thể chọn kiểu project bạn sẽ tạo. Tùy thuộc vào các tùy chọn bạn đã lựa khi cài đặt môi trường Visual Studio, bạn có thể chọn từ Visual Basic.NET, C#,

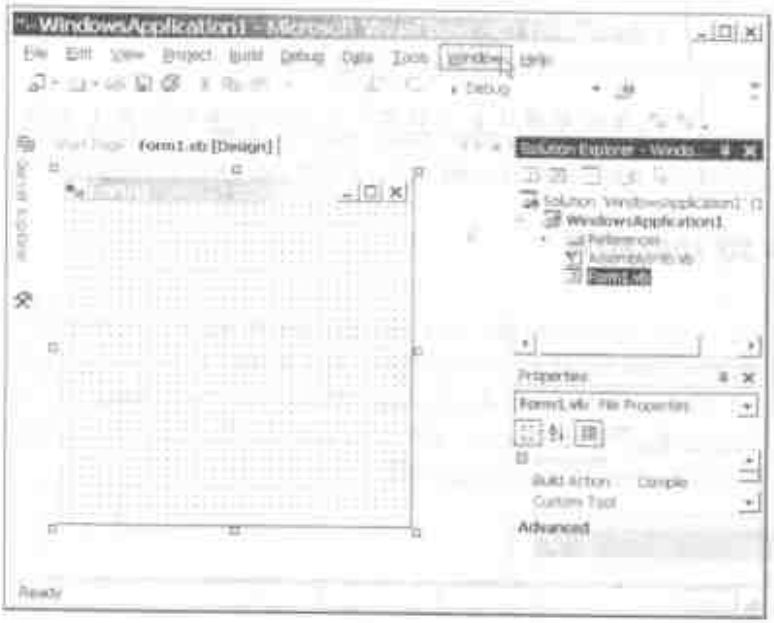
C++ và có thể có các ngôn ngữ lập trình khác. Không phải tất cả các ngôn ngữ này dành cho Visual Studio đến từ Microsoft; còn có các công ty khác phát triển các ứng dụng sẽ dùng .NET Framework.

Ở bên phải màn hình này, có nhiều template khác nhau mà bạn có thể chọn template mặc định cho kiểu project bạn sẽ tạo. Bảng 1 cung cấp danh sách một số kiểu template của project này.

Trước khi thêm một project mới vào solution này, bạn cần xác lập Name và Path, nơi project này sẽ cư trú trên ổ đĩa cứng của bạn. Điền vào đường dẫn, nơi bạn muốn project này cư trú vào hộp văn bản Location. VS.NET tạo đường dẫn cần thiết và sẽ tạo tên folder cùng tên với project. Thí dụ, nếu bạn điền vào Name là **LoginTest** và xác lập Path là D:\MySamples, thì solution này sẽ được tạo trong D:\MySamples\LoginTest\LoginTest.sln.

Môi trường phát triển tích hợp VS.NET (IDE)

Khi bạn bắt đầu một project mới trong Visual Studio.NET, bạn sẽ thấy một nhóm các cửa sổ được mở bên trong môi trường (xem hình 4). Các phần kế tiếp ở tài liệu này sẽ mô tả từng cửa sổ.



Hình 4: *Sẽ có một số cửa sổ mở khi bạn bắt đầu một project Visual Studio.NET mới.*

Trên màn hình chính này, bạn sẽ thấy một danh sách các menu nằm ngang trên đầu môi trường. Từ các menu này bạn có thể thực hiện các hoạt động sẽ ảnh hưởng tới project của bạn. Ngay dưới các menu là một loạt các thanh công cụ cho bạn các shortcut tới các mục menu. Ở bên trái của môi trường Visual Studio bạn thấy Server Explorer và một đồ họa của một số công cụ (cửa sổ Toolbox). Khi bạn đưa con trỏ chuột vào vùng này, các cửa sổ khác trở thành hiển thị. Bạn sẽ tìm hiểu các cửa sổ này sau.

Ở bên phải màn hình, bạn thấy cửa sổ Solution Explorer. Đây là nơi bạn sẽ thấy một hoặc nhiều project tạo thành ứng dụng bạn đang làm việc. Tất cả các file cho mỗi ứng dụng được liệt kê trong cửa sổ này. Ngay dưới cửa sổ Solution Explorer là cửa sổ

Properties. Cửa sổ này chứa một danh sách các thuộc tính cho đối tượng trong môi trường hiện đang được tô sáng.

Chúng ta hãy khám phá từng cửa sổ được mô tả bên dưới, cũng như các cửa sổ có thể sử dụng bằng một số mục menu.

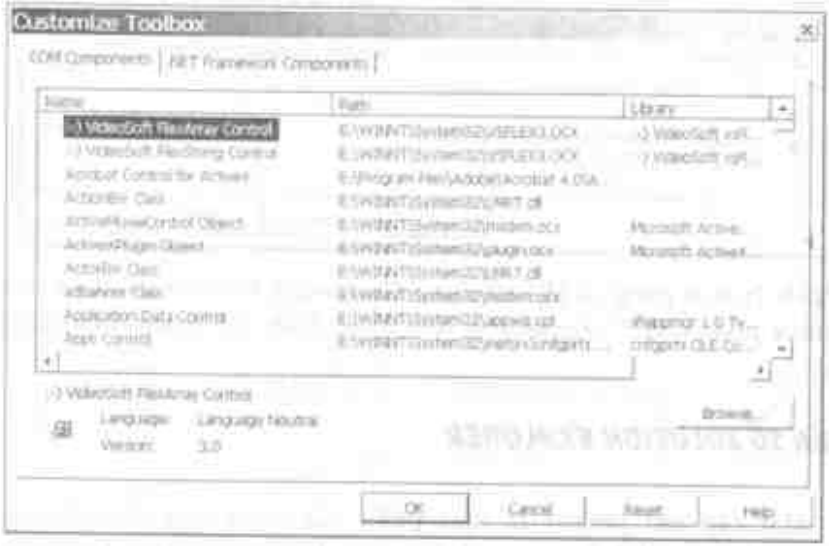
CỬA SỔ TOOLBOX

Cửa sổ có thể chốt giữ này lưu giữ danh sách các control mà bạn có thể kéo và thả vào các form của mình. Còn có một số tập control khác có thể sử dụng, tùy thuộc vào bạn đang sử dụng WinForms hay Web Forms. Để xem cửa sổ Toolbox, lựa View ➤ Toolbox từ menu Visual Studio.NET.

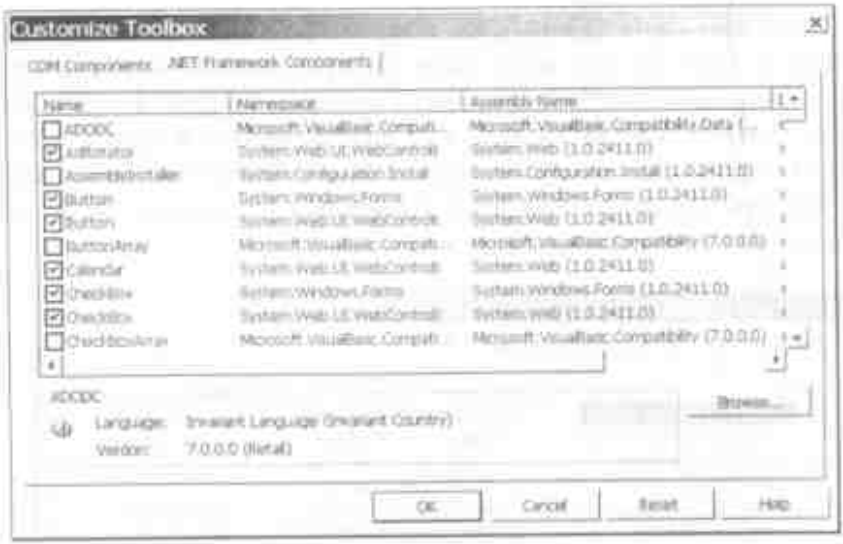
***Hình 5:** Toolbox lưu giữ danh sách các control.*

Tùy biến Toolbox

Nếu bạn muốn tùy biến các công cụ được hiển thị trong hộp công cụ hoặc nếu bạn muốn thêm bất kỳ control ActiveX nào vào hộp công cụ, chọn Tools ➤ menu Customize Toolbox để hiển thị hộp thoại Customize Toolbox, như được trình bày ở hình 6 và 7. Hình 6 trình bày các thành phần ActiveX (COM) mà bạn có thể thêm vào hộp công cụ và hình 7 trình bày các control .NET mà bạn có thể thêm vào hộp công cụ.



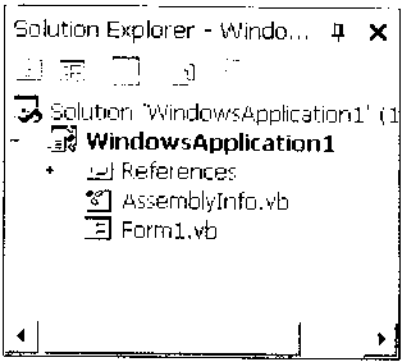
Hình 6: *Tùy biến Toolbox để bạn thêm các control COM / ActiveX vào hộp công cụ.*



Hình 7: *Bạn cũng có thể sử dụng Customize Toolbox để chọn các thành phần .NET framework nào được hiển thị trong hộp công cụ.*

CỬA SỔ SOLUTION EXPLORER

Một tập các project là một phần của cùng ứng dụng trong Visual Studio.NET được gọi là *Solution*. Cửa sổ Solution Explorer trình bày với bạn một danh sách hiển thị cây của mỗi project, các tham chiếu project và các thành phần của mỗi project. Nếu cửa sổ này được đóng, bạn có thể mở nó bằng cách lựa View ➤ Solution Explorer (Ctrl+R) từ menu. Các thành phần có thể được tạo thành gồm các form, các lớp, các module và các file bất kỳ khác mà nó dùng để tạo ứng dụng. Nhấp đôi vào một mục để xem chế độ thiết kế của nó trong IDE.



Hình 8: *Solution Explorer trình bày biểu tượng của tất cả các file tạo thành (các) project của bạn.*

Nằm ngang ở đầu cửa sổ này, còn có một loạt các nút. Bảng 3 mô tả từng nút này, bắt đầu từ trái sang phải.

Nút	Mô tả
View Code	Trình bày mã của file có tiêu điểm ở Solution Explorer
View Designer	Trình bày thiết kế cho một file riêng biệt có tiêu điểm ở Solution Explorer
Refresh	Làm tươi Solution Explorer
Show All Files	Trình bày tất cả các file, bao gồm bất kỳ các file form mã đàng sau.
Properties	Trình bày các đặc tính của file được lựa.

Bảng 3: *Các nút trên cửa sổ Solution Explorer.*

CỬA SỔ CLASS VIEW

Khi bạn bắt đầu tạo một số lớn các lớp, bạn nên xem một danh sách tất cả các đặc tính và các phương thức có thể dùng trong các lớp đó. Bạn có thể sử dụng cửa sổ Class View, như được

trình bày ở hình 9 để có khái niệm rõ về các mục này. Bạn có thể hiển thị cửa sổ này, bằng cách dùng View ➤ menu Class View. Khi cửa sổ được hiển thị, bạn có thể mở rộng từng mức trong danh sách để xem các đặc tính và các phương thức. Nếu bạn chọn một trong các đặc tính hoặc phương thức này, bạn có thể nhấp nút chuột phải và sẽ thấy một menu các hành động áp dụng trực tiếp vào định nghĩa của phương thức hoặc đặc tính đó.

***Hình 9:** Class View là cách tốt nhất để xem tất cả các đặc tính và các phương thức của các lớp.*

CỬA SỔ SERVER EXPLORER

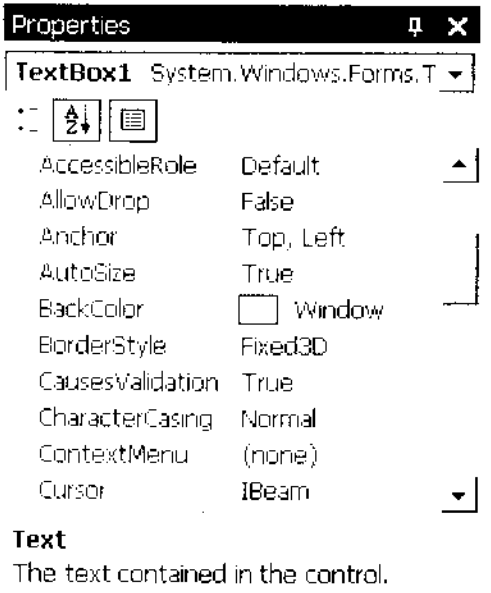
Vào lúc nào đó, bạn có thể chọn xây dựng một ứng dụng CSDL bằng cách dùng Visual Studio.NET. Trong trường hợp này, bạn sẽ cần xem các bảng CSDL, các thủ tục lưu trữ, các khung nhìn và các đối tượng khác. Bạn không phải giữ nguyên môi trường VS.NET; bạn có thể duyệt các CSDL, xem danh sách các bảng, xem dữ liệu trong các bảng, thay đổi thiết kế của bảng, vân vân.

Tất cả các điều này được thực hiện bằng cửa sổ Server Explorer. Lựa cửa sổ này bằng cách dùng View ➢ menu Server Explorer.

Hình 10: *Cửa sổ Server Explorer trình bày danh sách các server, các CSDL, các bảng, các khung nhìn và các đối tượng CSDL khác.*

CỬA SỔ PROPERTIES

Khi làm việc với các lớp, chẳng hạn như các hộp văn bản và các form, bạn có thể phải thay đổi các thuộc tính nào đó về các lớp đó. Bạn có thể hiển thị Properties Window bằng cách dùng View ➢ mục menu Properties Window (F4). Khi cửa sổ này hiển thị, bạn có thể xem danh sách theo thứ tự chữ cái hoặc được phân loại theo thuộc tính. Các đặc tính trong cửa sổ này có thể được lựa từ một danh sách hoặc bằng cách nhấp vào nút để hiển thị hộp thoại. Có thể có các đặc tính khác mà bạn có thể gõ văn bản vào, như đặc tính Text được dùng để thay đổi tiêu đề của form.



Hình 11: *Properties Window là nơi bạn sẽ làm việc nhiều.*

CỬA SỔ OBJECT BROWSER

Tương tự cửa sổ Class View, hộp thoại Object Browser trình bày danh sách các lớp, các đặc tính và các phương thức tương ứng của chúng. Một tính năng tốt của Object Browser là còn trình bày với bạn khai báo đầy đủ về phương thức hoặc đặc tính. Hiển thị Object Browser bằng cách dùng View ➢ Other Windows ➢ menu Object Browser (Ctrl+Alt+B).

Hình 12: *Cửa sổ Object Browser trình bày một danh sách đầy đủ về tất cả các lớp, các đặc tính và các phương thức trong project của bạn.*

CỬA SỔ TASK LIST

Còn có cửa sổ Task List để bạn định nghĩa các tác vụ bạn cần thực hiện để hoàn tất project của mình. Hiển thị cửa sổ Task List bằng cách dùng View ➢ Other Windows ➢ menu Task List. Sau đó bạn thấy một cửa sổ xuất hiện trong môi trường thiết kế của bạn như được trình bày ở hình 13. Để thêm một tác vụ mới, bạn có thể nhấp vị trí cho biết "Click here to add a new task" hoặc

bạn có thể thêm các chú thích ở định dạng riêng ngay vào mã của project. Chú thích mặc định thêm vào danh sách Task như sau:

```
'TODO: Finish calculations in this routine.
```

Sử dụng **TODO:** sau chú thích và danh sách tác vụ (task list) sẽ chọn nó. Bạn có thể nhấp đôi tác vụ để đưa bạn tới ngay chú thích đó trong mã.

Hình 13: *Các tác vụ có thể giúp bạn tổ chức ý tưởng về những gì cần thực hiện để hoàn tất project.*

CÁC KIỂU CỬA SỔ

Có hai kiểu cửa sổ trong VS.NET IDE: các cửa sổ Tool và các cửa sổ Document. Các cửa sổ Tool được liệt kê ở menu View và sự thay đổi dựa vào ứng dụng hiện hành và các ứng dụng bổ sung khác nhau mà bạn có thể cài đặt. Các cửa sổ Document là các cửa sổ bạn mở để soạn thảo văn bản hoặc cho các cửa sổ Help của Visual Studio.

CÁC CỬA SỔ TOOL

Cửa sổ Tool là tất cả các cửa sổ bạn vừa tìm hiểu ở tài liệu này, bao gồm Toolbox, Solution Explorer, Properties và Server Explorer. Bạn có thể thao tác và sắp xếp Tool Windows trong

IDE bằng nhiều cách. Bạn có thể làm cho các cửa sổ này ẩn hiện tự động. Bạn có thể cho một loạt cửa sổ hiển thị ở định dạng băng. Bạn có thể chốt giữ chúng ở các cạnh của IDE hoặc thả trôi chúng, bằng cách lựa hoặc xóa tùy chọn Dockable trên menu Window. Thậm chí bạn có thể hiển thị các cửa sổ này trên monitor thứ hai nếu bạn có khả năng dùng monitor kép. Để đặt Tool Windows vào các monitor khác nhau, sử dụng các xác lập Display trong Control Panel để thiết lập nhiều cấu hình monitor. Sau đó bạn có thể kéo Tool Window vào monitor khác.

Mẹo:

Bạn có thể di chuyển một cửa sổ chốt giữ mà không cần nhấp kéo nó tới một vị trí nào đó, bằng cách ấn phím CTRL khi kéo nó trong IDE.

Tính năng của cửa sổ Tool

Bạn có thể có nhiều thể nghiệm về một số cửa sổ Tool. Thí dụ, bạn có thể có hơn một cửa sổ Web Browser mở cùng một lúc. Sử dụng Window ➢ menu New Window để tạo các thể nghiệm cửa sổ mới.

Các cửa sổ Tool có thể được xác lập tự động ẩn khi bạn đưa vệt sáng vào một cửa sổ khác. Khi ẩn tự động, tự chúng thu nhỏ vào các cạnh của IDE ở dạng băng. Bạn chỉ cần nhấp chúng để hiển thị. Mỗi khi một cửa sổ mở, bạn có thể nhấp biểu tượng đinh ghim trên thanh tiêu đề của cửa sổ để chốt giữ nó trên IDE.

CÁC CỬA SỔ DOCUMENT

Các cửa sổ Document thấy được khi bạn soạn thảo mã cho một form hoặc soạn thảo form ở chế độ thiết kế. Cửa sổ Document là cửa sổ không chốt giữ được và chỉ dành riêng cho một project nhất định. Cách các cửa sổ Document xuất hiện trong IDE sẽ phụ

thuộc rất nhiều vào chế độ giao diện bạn đã xác lập trong Visual Studio.NET.

CÁC CHẾ ĐỘ GIAO DIỆN

VS.NET hỗ trợ hai chế độ giao diện khác nhau cho các cửa sổ tài liệu: Multiple Document Interface (MDI) và Tabs. Bạn có thể thay đổi các chế độ bằng cách dùng ô cửa General dưới các tùy chọn Environment trong hộp thoại Options. Bạn có thể nhận được hộp thoại này bằng cách chọn Tools ➢ Options, rồi lựa General dưới cây Environment. Ở chế độ MDI, IDE cung cấp một cửa sổ cha đáp ứng như một bộ chứa logic và trực quan cho tất cả các cửa sổ Tool và Document.

Tóm tắt

Việc sử dụng IDE của Visual Studio.NET có nhiều hiệu quả hơn NotePad. Có nhiều tùy chọn bạn có thể xác lập để tùy biến và tìm hiểu về IDE của Visual Studio.NET để tăng tốc độ phát triển và xây dựng các giao diện của bạn tốt hơn.

Câu hỏi ôn tập

1. Bạn bắt đầu một project mới trong Visual Studio.NET bằng cách nào?
2. Template nào bạn sẽ dùng để bắt đầu một ứng dụng kiểu desktop mới?
3. Cửa sổ nào trong Visual Studio.NET trình bày danh sách tất cả các file tạo thành ứng dụng của bạn?
4. Cửa sổ nào sẽ trình bày danh sách các mục To Do?

5. Bạn thêm các thành phần COM vào hộp công cụ bằng cách nào?

Bài tập

- Thiết lập profile của bạn cho trình phát triển Visual Basic.
- Tạo một project ứng dụng Windows mới và đặt tên nó là MyFirstApp.

TRẢ LỜI CÂU HỎI ÔN TẬP

1. Lựa File | New Project hoặc nhấp vào nút New Project trên trang đầu tiên.
2. Template Windows Application
3. Solution Explorer
4. Task List
5. Tools | menu Customize Toolbox

Chương 3

WinForms

- Tìm hiểu các WinForm là gì và cách sử dụng chúng.

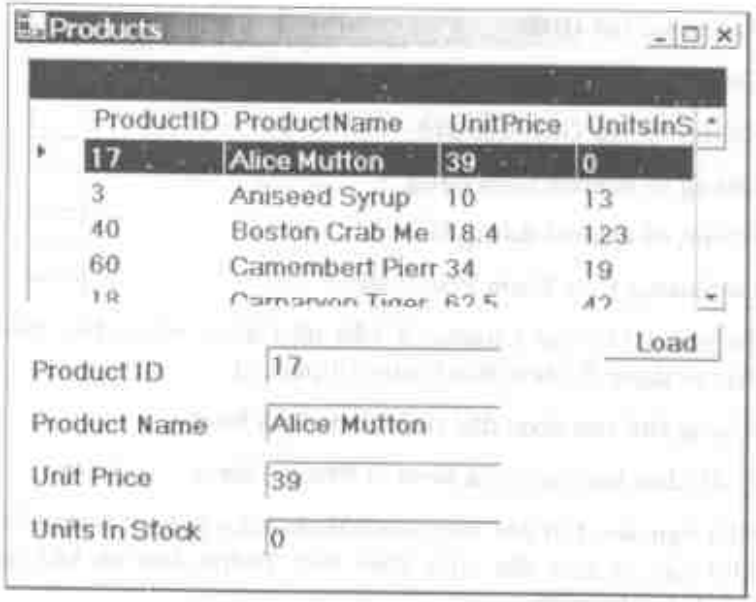
Window Forms (WinForms)

Microsoft .NET đã giới thiệu Windows Forms (WinForms) là một tập các lớp .NET cơ sở được dùng để xây dựng giao diện người dùng cho các ứng dụng desktop của bạn. Cùng các lớp cơ sở có thể được dùng từ một ngôn ngữ .NET bất kỳ. Vì vậy, dù bạn đang dùng VB.NET hoặc C# sẽ không thành vấn đề, các lớp WinForm đều được sử dụng.

Các phiên bản trước đây của các form Visual Basic được tạo hầu hết đằng sau các nền của chính VB. Hiện nay mã đều được trình bày, nếu bạn chọn khảo sát nó và thực ra bạn có thể viết giao diện người dùng bằng cách viết mã VB.NET. Bạn không phải dùng Visual Designer, nhưng nó dễ thực hiện hơn nhiều.

WinForms là các lớp giống như chúng ở trong VB, nhưng hiện nay chúng hoàn toàn trình bày với lập trình viên và có thêm một số tính năng, như tính thừa kế để tạo và duy trì các form cho các ứng dụng của bạn mạnh hơn và dễ dàng hơn nhiều.

WinForms được thiết lập bằng chính Form và một tập các control chuẩn đi cùng WinForms. Xét form được trình bày ở hình 1. Form này được thiết lập bằng control DataGrid, một số Label, một số Textbox và một control Button. Mỗi control này được engine WinForms cung cấp đi cùng với .NET



Hình 1: Form này được thiết lập bằng các control WinForm chuẩn.

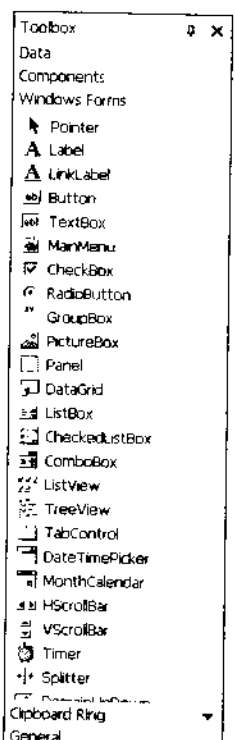
Các thay đổi của WinForm

Sử dụng các form có nhiều thay đổi trong Visual Basic.NET. Bên dưới là danh sách về một số các thay đổi này.

- Có thể xác lập TabIndex bằng 0 trên tất cả các control
- Zorder được dùng để đi qua các control vào thời gian chạy
- Control LinkLabel mới cho các khả năng siêu liên kết (hyperlink)
- Control GroupBox thay thế control Frame
- Control Splitter mới cho bạn giao diện giống Explorer
- Drag & Drop hỗ trợ hầu như khác nhau
- Không hỗ trợ DDE
- Control Time không thể xác lập Interval bằng 0
- Không hỗ trợ OLE Control
- Không có control hình dạng
- Không có control đường kẻ
- Bỏ phương thức Form.PrintForm
- Sự tương tác của Clipboard hầu như khác nhau, bây giờ bạn sử dụng System.WinForms.Clipboard
- Không thể tìm được đặc tính Name của form
- Phải khai báo biến của form là Show a form
- Nếu bạn xác lập đặc tính ScaleMode trên bất kỳ form của VB6 nào là một đặc tính khác hơn Twips, bạn sẽ không thể di trú form đó.
- Không thể dùng dữ liệu liên kết với DAO hoặc RDO. Bạn vẫn có thể dùng các đối tượng, nhưng chúng được coi là các công nghệ thừa kế và cuối cùng sẽ mất.

Các control WinForm chuẩn

Có nhiều control bạn sẽ tìm thấy chuẩn trong cửa sổ Toolbox. Tất cả các control này sẽ giúp bạn xây dựng giao diện người dùng cho ứng dụng của bạn. Hình 2 trình bày cửa sổ Toolbox của bạn sẽ trông như thế nào khi bạn hiển thị nó trong IDE của Visual Studio.NET.



Hình 2: *Toolbox lưu giữ tất cả các control chuẩn mà bạn sẽ dùng để xây dựng giao diện người dùng.*

Bảng 1 cho biết các tên của từng control chuẩn và mô tả của nó.

Control	Mô tả
Label	Được dùng để trình bày văn bản tĩnh với người dùng.
LinkLabel	Cho phép bạn đặt một siêu liên kết vào WinForm của bạn. Nó tác động giống như một siêu liên kết bạn thường dùng trong một trình duyệt.
Button	Người dùng nhấp vào control này khiến cho một hoạt động nào đó xảy ra trên form.
TextBox	Vùng nhập liệu của người dùng chính trên form.
MainMenu	Trợ giúp bạn xây dựng menu trên form.
CheckBox	Control có thể được chọn (kiểm) hoặc thôi chọn (không kiểm) hay thuộc một trạng thái không xác định.
RadioButton	Control được dùng với các control RadioButton khác để cho phép người dùng lựa một nút và chỉ một nút trong nhóm.
GroupBox	Control được dùng để lập nhóm control khác nhau.
PictureBox	Một control được dùng để trình bày hình.
Panel	Control được dùng để lập nhóm các control khác nhau.
DataGrid	Trình bày dữ liệu từ một dataset của ADO.NET ở định dạng bảng. Nó rất giống control FlexGrid trong VB6,

	nhưng có nhiều tính năng mới.
ListBox	Danh sách một số mục mà người dùng có thể lựa một mục hoặc một số mục tùy thuộc vào các đặc tính được xác lập trên control ListBox.
CheckedListBox	ListBox tăng cường cho phép người dùng chọn một hoặc nhiều mục trong danh sách bằng cách dùng hộp kiểm được định vị bên mỗi mục. Trong VB6, điều này được thực hiện bằng cách xác lập đặc tính Style property là 1 -Checked.
ComboBox	Danh sách thả xuống của các mục mà người dùng có thể lựa một mục.
ListView	Danh sách các mục có thể được trình bày trong các cột ở ListView.
TreeView	Danh sách các mục và các mục con. Giống như công cụ Windows Explorer.
TabControl	Kết hợp các tính năng của cả control TabStrip lẫn SSTab từ VB6
DateTimePicker	Cho phép người dùng nhập liệu hoặc lựa ngày tháng và/hoặc thời gian. Lịch thả xuống có thể được trình bày trên control này mà người dùng có thể lựa ngày tháng.
MonthCalendar	Lịch tháng mà người dùng có thể lựa ngày tháng.
HScrollBar	Thanh cuộn ngang.
VScrollBar	Thanh cuộn dọc.
Timer	Control sẽ kích hoạt một sự kiện trong mỗi mili giây.
Splitter	Cho phép bạn tạo một giao diện giống Explorer với ô cửa kép.

DomainUpDown	Kết hợp Text Box với một mũi tên lên xuống. Hoạt động giống như Combo Box mà bạn được phép đặt bất kỳ kiểu dữ liệu nào vào danh sách này và người dùng có thể cuộn qua dữ liệu đó bằng cách nhấp vào mũi tên lên và mũi tên xuống.
NumericUpDown	Kết hợp control TextBox với một control mũi tên lên xuống và cho phép người dùng tăng hoặc giảm giá trị số trong control đó.
TrackBar	Control sẽ trình bày tỉ lệ và cho phép người dùng di chuyển con trỏ trên tỉ lệ đó, do đó tăng hoặc giảm giá trị trong TrackBar.
ProgressBar	Control sẽ hiển thị các vạch trên control. Bạn thường dùng control này khi muốn báo với người dùng một tiến trình nào đó sắp xảy ra.
RichTextBox	Control cho phép người dùng nhập văn bản và áp dụng các tính chất font vào văn bản đó.
ImageList	Control lưu giữ một hoặc nhiều ảnh. Sau đó các ảnh này được các control khác sử dụng, chẳng hạn như TreeViews.
HelpProvider 	Khi bạn thêm control này vào Tray của form, nó sẽ thêm đặc tính HelpString, HelpTopic và ShowHelp vào mỗi control trên form.
ToolTip	Khi bạn thêm control này vào Tray của form, nó sẽ thêm đặc tính ToolTip vào mỗi control trên form.
ContextMenu	Thêm một menu khác vào form của bạn, sau đó nó có thể được gán vào

	một control bất kỳ trên form của bạn.
ToolBar	Control sẽ tạo thanh công cụ nằm ngang trên đầu form. Bạn có thể dùng các ảnh này cho các nút trên thanh công cụ từ control ImageList.
StatusBar	Control trong đó bạn có thể thiết lập một hoặc nhiều ô cửa sổ để trình bày thông tin bằng văn bản.
NotifyIcon	Thường chỉ được dùng khi bạn đang tạo một Windows Service, cho phép bạn thiết lập Icon được hiển thị trong System Tray trên thanh tác vụ. Khi được nhấp, bạn có thể hiển thị một menu hoặc trình bày một form bất kỳ trong chương trình đó.
OpenFileDialog	Hiển thị hộp thoại OpenFile. Control này được dùng như phương thức của control CommonDialog cũ trong VB6.
SaveFileDialog	Hiển thị hộp thoại SaveFile. Control này được dùng như phương thức của control CommonDialog cũ trong VB6.
FontDialog	Hiển thị hộp thoại Font. Control này được dùng như phương thức của control CommonDialog cũ trong VB6.
ColorDialog	Hiển thị hộp thoại Color. Control này được dùng như phương thức của control CommonDialog cũ trong VB6.
PrintDialog	Hiển thị hộp thoại Print. Control này dùng như phương thức của control CommonDialog cũ trong

	VB6.
PrintPreviewDialog	Được dùng kết hợp với PrintPreviewControl
PrintPreviewControl	Cho phép bạn in một cửa sổ xem trước thay vì in ra bằng máy in.
ErrorProvider	Khi bạn thêm control này vào Tray của form, nó sẽ thêm đặc tính Error, IconAlignment và IconPadding vào từng control trên form.
PrintDocument	Cùng với các lớp khác, cho phép bạn gửi tài liệu vào máy in.
PageSetupDialog	Hiển thị hộp thoại, nơi người dùng có thể thao tác các xác lập, chẳng hạn như các lề và hướng giấy.
CrystalReportViewer	Cho phép bạn hiển thị Crystal Report như hộp thoại bật lên từ ứng dụng của bạn.

Bảng 1: *Có nhiều control để chọn khi xây dựng giao diện người dùng.*

Các đặc tính thông dụng trên các control

Thường bạn sẽ cần một tập các đặc tính thông dụng trên mỗi control. Giống như bạn đã dùng trong VB6, bạn cũng có một tập các đặc tính trên mỗi control trong .NET. Bảng 2 trình bày danh sách và mô tả về mỗi đặc tính làm gì.

Đặc tính mới	Mô tả
AccessibleDescription	Mô tả sẽ được Narrator đọc. Narrator là một công cụ truyền thông đọc các từ trên màn hình máy tính ra loa. Nó có thể dùng được trên hầu hết các hệ thống Windows và bạn có thể gọi ra bằng cách dùng Start>Programs>Accessories>Accessibility>Narrator. Công cụ này đọc các từ khi bạn di chuyển chuột trên màn hình. Khi bạn điền vào mô tả này và chạy công cụ Narrator, nó đọc mô tả này với người dùng.
AccessibleName	Tên của control sẽ được phụ kiện Narrator báo.
AccessibleRole	Vai trò sẽ được báo cho người dùng bởi phụ kiện Narrator.
AllowDrop	Xác định dù control có nhận các thông báo kéo và thả hay không.
Anchor	Xác định nơi control sẽ được neo vào form. Bạn có thể chọn hầu như bất kỳ kiểu neo nào bạn muốn.
ContextMenu	Xác lập đặc tính này là tên của control ContextMenu và khi nhấp nút chuột phải vào control này, menu đó sẽ tự động được hiển thị.
Dock	Xác định nơi control này sẽ chốt giữ trên form. Khi bạn chốt giữ một control, control tăng giảm theo form, nhưng vẫn được chốt giữ ở cùng vị trí.
Locked	Nếu xác lập là True, control này sẽ không di chuyển được vào thời gian thiết kế.
Modifiers	Xác định phạm vi của control. Theo mặc định, trong .NET tất cả các control đều là Private. Trong các phiên bản VB trước đây, tất cả các control đều là Public. Bây giờ bạn có thể chọn Private, Protected hoặc Public.
Location	Cấu trúc Point cho các tọa độ X và Y của vị trí control này được định vị. Trong các phiên bản VB trước đây, nó được xác lập bằng cách dùng đặc tính Left và Top.
Size	Cấu trúc Size cho chiều rộng và chiều cao của control. Trong các phiên bản VB trước đây, nó được xác lập bằng cách dùng đặc tính Width và Height.

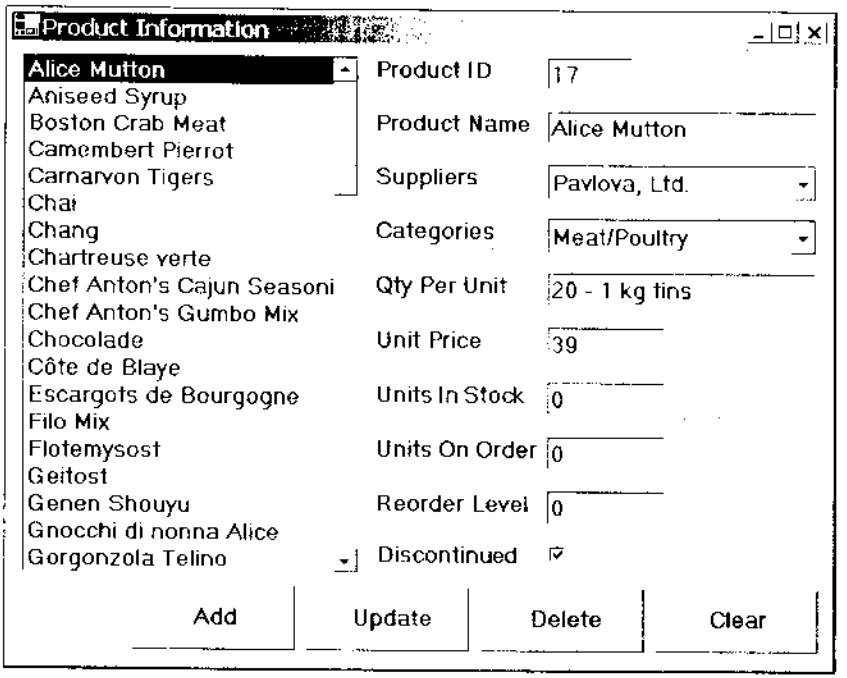
Bảng 2: *Có nhiều đặc tính mới trên mỗi control trong .NET.*

Neo và Chốt giữ

Một điển hình mạnh đã được thêm vào các control WinForm, bây giờ bạn sẽ tìm hiểu về hai đặc tính được gọi là Anchor (neo) và Dock (chốt giữ). Hai đặc tính này sẽ cho bạn nhiều khả năng mà trước đây bạn đã phải viết mã thủ công.

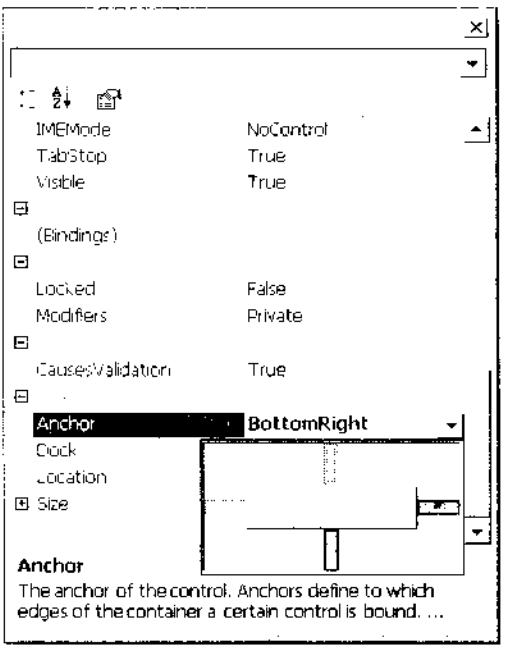
Neo

Nếu bạn giống như nhiều người phát triển VB, bạn đã viết mã để neo các control vào cạnh phải dưới của form nhiều lần. Bạn muốn người dùng có thể thay đổi kích thước form, nhưng bạn cũng muốn các control nào đó trên form luôn luôn duy trì vị trí của chúng so với cạnh dưới và/hoặc cạnh phải của form. Không còn dùng mã nữa! VS.NET cung cấp một giải pháp đơn giản: đặc tính Anchor. Đặc tính này cho phép bạn neo control vào một hoặc tất cả các cạnh của form. Việc neo control vào cạnh dưới và/hoặc cạnh phải khiến cho nó trôi khi bạn thay đổi kích thước form. Việc neo control vào cạnh trên và/hoặc cạnh trái khiến cho nó thay đổi kích thước khi bạn thay đổi kích thước form. Bạn nên thực nghiệm với đặc tính mới này để biết được ý nghĩa hiệu ứng của nó trên các control.



Hình 3: *Một màn hình nhập dữ liệu tiêu biểu với các nút được neo vào cạnh trái dưới.*

Trên form nhập dữ liệu, giống như được trình bày ở hình 3, có lẽ bạn cần có các nút Add, Update, Delete và Clear nằm ở góc phải dưới của form. Để thực hiện điều này, đưa vệt sáng vào các nút và xác lập đặc tính Anchor là BottomRight, bằng cách tìm đặc tính Anchor trong cửa sổ Properties và nhấp mũi tên xuống. Thôi chọn các vạch ở trên và dưới, rồi sau đó lựa các vạch ở dưới và bên phải, như được trình bày ở hình 4.



Hình 4: *Hộp thoại Properties với Anchor thả xuống hoạt động.*

Đặc tính Anchor cho phép bạn neo các control vào các cạnh của form mà không phải viết một dòng mã nào. Bằng cách lựa các tùy chọn từ các menu đồ họa, bạn có thể lựa ứng xử bạn cần. Trong mã, bạn có thể dùng các hằng, chẳng hạn như TopLeft, BottomRight, Left, Right, Top, Bottom, TopLeftBottom và All, để xác lập giá trị của đặc tính này.

Chốt giữ

Đặc tính Dock giống như đặc tính Anchor, nó cho phép bạn chốt giữ một control vào bất kỳ cạnh nào của form. Khi bạn chốt giữ một control, control ấy co hoặc dãn theo form, nhưng cạnh nó được chốt giữ vẫn không thay đổi. Nếu bạn chốt giữ một control

vào tất cả các cạnh của form, thì control ấy sẽ phủ đầy toàn bộ form.

Sự thay đổi trong .NET

Một trong các thay đổi lớn nhất trong .NET so với VB6 là thấy được mã thể nghiệm các control trên form. Trước đây, tất cả mã này bị ẩn trong control ActiveX và trong VB vào thời gian chạy. Bây giờ bạn có thể thấy nó bằng cách mở rộng đoạn #Region trong mã đằng sau form. Sau đây là một form mẫu có một nhãn, một hộp văn bản và một control nút.

```
#Region " Windows Form Designer generated code "

Public Sub New()
    MyBase.New()

    'This call is required by the Windows Form Designer.
    InitializeComponent()

    'Add any initialization after the
InitializeComponent() call

End Sub

'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing
As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub
Friend WithEvents btnLOad As System.Windows.Forms.Button
Friend WithEvents grdProducts As
System.Windows.Forms.DataGrid
Friend WithEvents txtProductID As
System.Windows.Forms.TextBox
Friend WithEvents txtProduct As
System.Windows.Forms.TextBox
```

```
Friend          WithEvents          txtUnitPrice          As
System.Windows.Forms.TextBox
Friend          WithEvents          txtUnitsInStock          As
System.Windows.Forms.TextBox

'Required by the Windows Form Designer
Private components As System.ComponentModel.Container

'NOTE: The following procedure is required by the Windows
Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
<System.Diagnostics.DebuggerStepThrough()>    Private    Sub
InitializeComponent()
Me.btnLOad = New System.Windows.Forms.Button()
Me.txtUnitPrice = New System.Windows.Forms.TextBox()
Me.Label4 = New System.Windows.Forms.Label()
Me.txtUnitsInStock = New System.Windows.Forms.TextBox()
Me.Label3 = New System.Windows.Forms.Label()
Me.Label1 = New System.Windows.Forms.Label()
Me.Label2 = New System.Windows.Forms.Label()
Me.grdProducts = New System.Windows.Forms.DataGrid()
Me.txtProduct = New System.Windows.Forms.TextBox()
Me.txtProductID = New System.Windows.Forms.TextBox()
CType(Me.grdProducts,
System.ComponentModel.ISupportInitialize).BeginInit()
Me.SuspendLayout()
'
'btnLOad
'
Me.btnLOad.Location = New System.Drawing.Point(376, 176)
Me.btnLOad.Name = "btnLOad"
Me.btnLOad.TabIndex = 0
Me.btnLOad.Text = "Load"
'
'txtUnitPrice
'
Me.txtUnitPrice.Location  =  New  System.Drawing.Point(160,
266)
Me.txtUnitPrice.Name = "txtUnitPrice"
Me.txtUnitPrice.Size = New System.Drawing.Size(152, 26)
Me.txtUnitPrice.TabIndex = 4
Me.txtUnitPrice.Text = ""
'
'Label4
'
Me.Label4.Location = New System.Drawing.Point(8, 302)
```

```
Me.Label4.Name = "Label4"
Me.Label4.Size = New System.Drawing.Size(136, 24)
Me.Label4.TabIndex = 6
Me.Label4.Text = "Units In Stock"
'
'txtUnitsInStock
'
Me.txtUnitsInStock.Location = New
System.Drawing.Point(160, 303)
Me.txtUnitsInStock.Name = "txtUnitsInStock"
Me.txtUnitsInStock.Size = New System.Drawing.Size(152, 26)
Me.txtUnitsInStock.TabIndex = 5
Me.txtUnitsInStock.Text = ""
'
'Label3
'
Me.Label3.Location = New System.Drawing.Point(8, 268)
Me.Label3.Name = "Label3"
Me.Label3.Size = New System.Drawing.Size(136, 24)
Me.Label3.TabIndex = 6
Me.Label3.Text = "Unit Price"
'
'Label1
'
Me.Label1.Location = New System.Drawing.Point(8, 200)
Me.Label1.Name = "Label1"
Me.Label1.Size = New System.Drawing.Size(136, 24)
Me.Label1.TabIndex = 6
Me.Label1.Text = "Product ID"
'
'Label2
'
Me.Label2.Location = New System.Drawing.Point(8, 234)
Me.Label2.Name = "Label2"
Me.Label2.Size = New System.Drawing.Size(136, 24)
Me.Label2.TabIndex = 6
Me.Label2.Text = "Product Name"
'
'grdProducts
'
Me.grdProducts.DataMember = ""
Me.grdProducts.Location = New System.Drawing.Point(8, 8)
Me.grdProducts.Name = "grdProducts"
Me.grdProducts.Size = New System.Drawing.Size(448, 160)
Me.grdProducts.TabIndex = 1
'
'txtProduct
```

```
'
Me.txtProduct.Location = New System.Drawing.Point(160,
229)
Me.txtProduct.Name = "txtProduct"
Me.txtProduct.Size = New System.Drawing.Size(152, 26)
Me.txtProduct.TabIndex = 3
Me.txtProduct.Text = ""
'
'txtProductID
'
Me.txtProductID.Location = New System.Drawing.Point(160,
192)
Me.txtProductID.Name = "txtProductID"
Me.txtProductID.Size = New System.Drawing.Size(152, 26)
Me.txtProductID.TabIndex = 2
Me.txtProductID.Text = ""
'
'frmProducts
'
Me.AutoScaleBaseSize = New System.Drawing.Size(8, 19)
Me.ClientSize = New System.Drawing.Size(464, 346)
Me.Controls.AddRange(New System.Windows.Forms.Control()
{Me.Label4, Me.Label3, Me.Label2, Me.Label1,
Me.txtUnitsInStock, Me.txtUnitPrice, Me.txtProduct,
Me.txtProductID, Me.grdProducts, Me.btnLOad})
Me.Font = New System.Drawing.Font("Microsoft Sans Serif",
12!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
Me.Name = "frmProducts"
Me.Text = "Products"
CType(Me.grdProducts,
System.ComponentModel.ISupportInitialize).EndInit()
Me.ResumeLayout(False)

End Sub

#End Region
```

Như bạn thấy, mỗi control được tạo và sau đó các đặc tính được khởi hoạt, chẳng hạn như Location và Size. Các đặc tính bất kỳ khác bạn xác lập trong Property Window bằng trình thiết kế Visual Studio cũng được thêm bằng mã vào đoạn này. Nếu bạn thực muốn thiết lập, bạn có thể thiết kế giao diện người dùng của mình chỉ bằng Notepad! Dù bạn không thể hiểu tất cả mã được

trình bày ở trên, nhưng nó cho bạn khái niệm về mỗi control được tạo trên form như thế nào.

Lưu ý:
Đừng thay đổi bất kỳ mã nào trong vùng #Region này. Mã này được tạo và hủy tự động bằng môi trường Visual Studio bất kỳ khi nào bạn thêm hoặc xóa các control khỏi form.

Hiển thị WinForm

Vì các form không khác gì hơn các lớp (class), bạn phải tạo thể nghiệm của lớp để có thể hiển thị nó. Ở các phiên bản VB trước đây, thì nó không nhất thiết phải như VB sẽ tạo một đối tượng cho bạn tự động. Bây giờ, bạn phải khai báo một biến đối tượng của form trước khi bạn có thể hiển thị form. Bên dưới là mã hiển thị form như được trình bày ở hình 1.

```
Dim frm As frmProducts

frm = New frmProducts()

frm.Show()
```

Tóm tắt

WinForms được cải tiến rất nhiều so với các phiên bản Visual Basic trước đây. Có một tập các control phong phú mà bạn có thể thấy mã được dùng để xây dựng các control và với các tính năng mới như neo và chốt giữ, việc tạo các giao diện người dùng đầy tính năng có thể được thực hiện nhanh mà không phải viết mã.

Câu hỏi ôn tập

1. Đúng hay sai: Trong VB.NET, tất cả các form đều được tạo bằng cách dùng mã.

2. Control nào bạn sẽ dùng để hiển thị siêu liên kết trên một WinForm?

3. Control nào bạn sẽ dùng để cho phép người dùng nhập liệu ngày tháng và tùy chọn ngày tháng từ một lịch thả xuống?

4. Đặc tính nào bạn xác lập trên một control bất kỳ khiến cho nó không được di chuyển vào thời gian thiết kế?

5. Đặc tính nào sẽ khiến cho control di chuyển khi một cạnh của form được di chuyển?

Bài tập

1. Mở Visual Studio.NET

2. Tạo một project ứng dụng Windows mới với tên ProductInfo.

3. Đặt lại tên form mặc định trong cửa sổ Solution Explorer là frmMain.vb.

4. Xác lập đặc tính Caption là "Main Form".

5. Thêm một control nút vào form này và đặt tên của nút là btnProducts.

6. Xác lập đặc tính Text của nút này là "Products". Form của bạn bây giờ trông như sau:

7. Thêm một form mới vào project này bằng cách lựa Project ➢ Add Windows Form.

8. Đặt tên là frmProducts.vb.

9. Tạo control DataGrid, bốn control nhãn và bốn control hộp văn bản trên form này. Bên dưới là những gì form này thể hiện.

10. Bây giờ mở lại form chính.

11. Nhấp đôi vào control nút và thêm mã sau.

```
Private Sub btnProducts_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnProducts.Click
    Dim frm As frmProducts

    frm = New frmProducts()

    frm.Show()
End Sub
```

12. Chạy project để hiển thị Main Form.

13. Nhấp vào nút để hiển thị form Products của bạn.

Trả lời câu hỏi ôn tập

1. Đúng.
2. LinkLabel
3. DateTimePicker
4. Locked
5. Anchor

Chương 4

Có những gì mới trong VB.NET

- Tìm hiểu những gì đã thay đổi kể từ Visual Basic 6.0

VB.NET

Có nhiều đổi mới trong ngôn ngữ Visual Basic dưới nền .NET. Chương này sẽ đề cập một số cấu trúc thông dụng hơn đã làm thay đổi ngôn ngữ này.

Chuyển đổi sang .NET

Dù sự chuyển đổi các chương trình cũ sẽ không phải không khó khăn, nhưng Microsoft đã cung cấp cho chúng ta Migration Wizard. Wizard này sẽ có thể chuyển đổi một số cấu trúc ngôn ngữ của bạn, nhưng nó sẽ không thể thực hiện được tất cả. Nơi nó không thể chuyển đổi điều gì đó là vị trí có chú thích "TO

DO:" bên mã nó không thể chuyển đổi. Sau đó "TO DO" này sẽ xuất hiện trong cửa sổ Task List. Khi đó bạn có thể tìm và nâng cấp các mục này khá dễ dàng.

Xử lý mảng

Câu lệnh Option Base đã được loại khỏi ngôn ngữ này. Điều này có nghĩa là tất cả các mảng bắt đầu với zêrô (0). Chiều dài Dimensioned là số phần tử bạn chỉ định cộng 1 giống như trong VB6. Thí dụ:

```
Dim astrValues(20) As String
```

Mã ở trên sẽ tạo mảng phần tử 21 bắt đầu với 0 và kết thúc với 20. Bạn không thể tạo Dim astrValues(1 to 4) được nữa. Kiểu viết mã này không hợp lệ.

Các mảng có kích thước cố định cũng là một vấn đề trong quá khứ. Một mảng bất kỳ có thể được thay đổi kích thước bằng cách dùng câu lệnh Redim. Bạn không thể thay đổi số của các kích thước, nhưng bạn có thể thay đổi số phần tử bên trong kích thước. Bạn không thể dùng Redim để khai báo mảng, bạn phải khai báo mảng trước, sau đó dùng Redim vào mảng đó.

Bạn cũng có thể khai báo và khởi tạo một mảng với tất cả các giá trị ở trong một câu lệnh.

```
Dim aintValues(4) = {10, 20, 30, 40, 50}
```

Xử lý chuỗi

Bạn không được phép tạo các chuỗi có chiều dài cố định nữa. Cú pháp sau không hợp lệ.

```
Dim strName As String * 30
```

Thay vào đó bạn phải dùng kiểu mã sau:

```
Dim strName As String
strName = " ".PadRight(30)
```

Mã ở trên tạo một chuỗi thông thường, sau đó gán nó vào 30 kí tự trắng, bằng cách dùng kĩ thuật gọi là *boxing*. Boxing có nghĩa là bạn dùng nguyên nghĩa để chuyển đổi nó thành một đối tượng kiểu dữ liệu riêng biệt, sau đó áp dụng một trong các phương thức có kiểu dữ liệu lập sẵn. Trong trường hợp này, bạn sẽ áp dụng phương thức PadRight() vào chuỗi để thêm 30 kí tự trắng.

Kiểu dữ liệu Integer

Một số kiểu dữ liệu integer đã thay đổi để nhất quán với các ngôn ngữ khác và tương ứng từng kiểu dữ liệu một với các engine CSDL. Bên dưới là bảng các dữ liệu Integer mới này.

Kiểu dữ liệu	Phạm vi các số
Byte	0 tới 255
Short	-32,678 tới 32,767
Integer	-2,147,483,648 tới 2,147,483,647
Long	-9,223,372,036,854,775,808 tới 9,223,372,036,854,775,807

Như bạn thấy từ bảng này, kiểu dữ liệu Short là kiểu dữ liệu bạn đã biết trước đây là Integer. Bây giờ Integer là kiểu dữ liệu trước đây bạn đã biết là Long. Long là integer 64 bit mới và có thể xử lý các số mà chúng ta không thể xử lý trong các phiên bản cũ của Visual Basic.

Các kiểu dữ liệu Decimal

Kiểu dữ liệu Currency đã được loại khỏi ngôn ngữ. Thay vào đó có một kiểu dữ liệu Decimal mới. Một lần nữa, nó tạo cho các thứ nhất quán hơn với các ngôn ngữ khác và khá nhất quán với các engine CSDL. Kiểu dữ liệu Decimal là tiện ích lưu trữ 12 byte. Nó có thể xử lý từ 0 tới 28 vị trí ở bên phải của dấu thập phân.

Kiểu dữ liệu Char

Kiểu dữ liệu Char chính là kí tự đơn. Nó là một số 2 byte, vì vậy nó có thể xử lý các kí tự UNICODE.

Loại bỏ Variant

Kiểu dữ liệu Variant đã được thay bằng kiểu dữ liệu Object. Object có tính năng giống Variant, trong đó nó có thể lưu trữ bất kỳ một trong các kiểu dữ liệu khác.

Kiểu dữ liệu Date

Kiểu dữ liệu Date bây giờ được lưu trữ như một giá trị Integer 8 byte ngược lại với Double như ở trong các phiên bản trước đây của Visual Basic. Bạn không còn có thể chuyển đổi giữa Date và Double. Hàm Date() và Now() đã được thay bằng hàm Today(). Hàm Time() đã được thay bằng hàm TimeOfDay().

DefType

Một số lập trình viên sẽ khai báo các câu lệnh DefType ở trên đầu các module của họ để khai báo tất cả các biến bắt đầu với một kí tự nào đó được coi là một số kiểu nào đó, chẳng hạn như

DefInt, DefLng, vân vân. Tất cả các câu lệnh này đã được loại khỏi ngôn ngữ. Bây giờ bạn buộc phải khai báo tất cả các biến.

Các toán tử Boolean

Các toán tử "And", "Or" và "XOR" đã được dùng cho cả Boolean lẫn các phép toán bitwise. Bây giờ bạn phải dùng BitAnd, BitOr, BitXOR và BitNot mới. Bạn có thể dùng Boolean hoặc các kiểu dữ liệu numeric với các toán tử này.

Ngắn mạch

.NET đưa vào ngôn ngữ VB.NET hai từ khóa mới có thể được dùng để ngắn mạch một số phép toán logic. **AndAlso** và **OrElse** được dùng để bỏ qua các biểu thức còn lại sau khi điều kiện đúng hoặc sai khiến cho phần còn lại của câu lệnh If không thành công.

Visual Basic.NET không nhất thiết sẽ hữu hiệu hóa tất cả các biểu thức trong câu lệnh If nếu bạn đang dùng điều kiện "AndAlso". Nếu toán hạng đầu tiên trong biểu thức "AndAlso" hữu hiệu hóa là False, thì thậm chí không cần hữu hiệu hóa các biểu thức khác. Kiểu ứng xử thấy ở VB6 này có thể gây ra các vấn đề. Thí dụ, trường hợp VB6 sau:

```
' In VB6 FlagTestTrue is called twice
If FlagTestTrue() And FlagTestTrue() Then
   MessageBox.Show("Made it here")
End If

Private Function FlagTestTrue() As Boolean
   MessageBox.Show("FlagTestTrue()")
   Return True
End Function
```

Ở mã VB6 trên, hàm được gọi hai lần, vì biểu thức đầu tiên (gọi FlagTestTrue) cho trở lại giá trị True. Tuy nhiên, nếu bạn xét mã VB.NET sau:

```
' VB.NET Code
' FlagTestFalse() is called once
' FlagTestTrue() is NOT called
If FlagTestFalse() AndAlso FlagTestTrue() Then
   ' We never get here
   MessageBox.Show("Made it here")
End If

Private Function FlagTestFalse() As Boolean
   MessageBox.Show("FlagTestFalse()")
   Return False
End Function
```

Trong trường hợp trên, hàm FlagTestTrue không được gọi vì biểu thức đàng trước điều kiện AndAlso cho trở lại False. Với điều kiện một trong các biểu thức trong khi kiểm tra AndAlso cho trở lại giá trị False, thì toàn bộ biểu thức đều là False. Do đó, nếu biểu thức đầu tiên cho trở lại giá trị False, thì thậm chí không cần thực hiện biểu thức thứ hai.

Bây giờ xét mã sau, trong đó điều kiện "OrElse" được dùng trong biểu thức If.

```
' FlagTestTrue() is called once
If FlagTestTrue() OrElse FlagTestTrue() Then
   MessageBox.Show("Made it here")
End If

' FlagTestFalse() is called once
' FlagTestTrue() is called once
If FlagTestFalse() OrElse FlagTestTrue() Then
   MessageBox.Show("Made it here")
End If
```

Ở câu lệnh If đầu tiên, hàm FlagTestTrue được gọi chỉ một lần. Nếu một biểu thức bất kỳ trong khi kiểm tra OrElse cho trở

lại True, thì toàn bộ câu lệnh được coi là True. Do đó, nếu biểu thức đầu tiên cho trở lại giá trị True, thì thậm chí không cần hữu hiệu hóa biểu thức thứ hai.

Ở câu lệnh If thứ hai, hàm FlagTestFalse được gọi một lần và vì nó cho trở lại giá trị False, FlagTestTrue (phần thứ hai của biểu thức) được gọi.

Loại bỏ các đặc tính mặc định

Khả năng sử dụng các đặc tính mặc định không còn tồn tại trong Visual Basic.NET. Xét thí dụ sau:

```
Dim t1 As TextBox
Dim t2 As TextBox

t1.Text = "Hi There"
t2 = "Text for the text box"   ' Ambiguous

t1 = t2   ' assigns text property to
          text property
SET t1 = t2   ' Assigns a pointer
```

Có một số lý do tại sao chúng không đạt:

- Chúng quá khó đọc.
- Khó xác định nếu một đối tượng có đặc tính mặc định.
- Khó xác định đặc tính mặc định của một đối tượng là gì.
- Cho bạn viết mã khác nhau để gán một đối tượng vào một biến tham chiếu so với việc chỉ sử dụng đặc tính mặc định.

- Vì vậy, bạn không còn dùng các đặc tính mặc định nữa, trừ trường hợp khi đặc tính mặc định là một chỉ mục vào một tập hợp trong đối tượng.

```
Dim t1 As TextBox
Dim t2 As TextBox

t1.Text = "Hi There"
t2.Text = "Text for the text box"

t1 = t2   ' assigns a pointer
```

Câu lệnh **Set** không còn hỗ trợ trong Visual Basic.NET nữa.

Câu lệnh Return

Câu lệnh Return bây giờ có thể được dùng để cho trở lại một giá trị từ một hàm, để tách từ Sub hoặc cho trở lại giá trị từ đặc tính Get.

Các khai báo đặc tính lớp (Class)

Việc tạo các đặc tính cho các lớp hầu như đều khác trong Visual Basic.NET so với các phiên bản trước đây của Visual Basic. Cú pháp Property Get/Let/Set không còn được dùng nữa. Bây giờ bạn sẽ khai báo tên đặc tính và kiểu dữ liệu một lần, sau đó dùng khối Get và Set như được trình bày ở mã bên dưới.

```
Private mstrFirstName As String
Private mintCode As Integer = -1

Property FirstName() As String
   Get
      Return mstrFirstName
   End Get
   Set(ByVal Value As String)
      mstrFirstName = Value
   End Set
End Property
```

```
Property Code() As Integer
   Get
      Code = mintCode
   End Get
   Set(ByVal Value As Integer)
      If Value = 0 Or Value = 1 Then
         mintCode = Value
      Else
         mintCode = -1
      End If
   End Set
End Property
```

Khối Get...End Get rất giống Property Get mà bạn đã dùng trong các phiên bản trước đây của Visual Basic. Bạn thường cho trở lại một giá trị từ hàm. Bạn có thể dùng câu lệnh Return hoặc có thể gán giá trị để cho trở lại tên của đặc tính.

Câu lệnh Set giống câu lệnh Property Let cũ mà bạn đã dùng để tạo trong VB6. Visual Studio.NET sẽ tự động tạo cú pháp Set và cung cấp cho bạn với tham số *Value* cho câu lệnh Set.

Các kiểu người dùng xác định

Type...End Type không còn được dùng để tạo các kiểu do người dùng xác định nữa. Thay vào đó bạn sẽ dùng Structure...End Structure. Tất cả các biến được khai báo bên trong cấu trúc phải có dấu định tính truy xuất, chẳng hạn như Public, Private, vân vân. Về cơ bản chúng giống module lớp bây giờ.

```
Structure Customer
   Public CustID As Integer
   Dim CustomerName As String
   Private TotalSalesAmount As Decimal
End Structure
```

Các thay đổi khai báo biến

Bây giờ có thể khai báo nhiều biến trên cùng dòng trong Visual Basic.NET và tất cả chúng sẽ thuộc cùng kiểu dữ liệu.

```
Dim intLoop, intValue As Integer
Dim strFirst, strLast As String
```

Trong VB6, intLoop và strFirst sẽ là kiểu dữ liệu Variant. Trong Visual Basic.NET cả intLoop lẫn intValue đều là kiểu dữ liệu Integer và strFirst và strLast đều là kiểu dữ liệu String.

Bây giờ bạn có thể khai báo một biến và khởi tạo biến ấy tất cả trên cùng dòng.

```
Dim intLoop As Integer = 1
Dim strFirst As String = "Unknown"
```

Từ khóa As Any không còn được hỗ trợ cho các khai báo bên ngoài, chẳng hạn như các lời gọi của Windows API. Bây giờ bạn phải quá tải (overload) khai báo cho các kiểu dữ liệu khác nhau bạn muốn dùng.

Phạm vi của các biến

Bây giờ các biến có thể được khai báo bên trong một khối mã. Khi điều này xảy ra, chúng không thể được dùng bên ngoài khối đó. Ở thí dụ sau, biến intLoop được khai báo bên trong câu lệnh If. Do đó, sau câu lệnh End If, biến này không thể được dùng.

```
If boolPerform Then
   Dim intLoop As Integer
   For intLoop = 0 To 10
      Debug.WriteLine(intLoop.ToString())     Next
End If
' intLoop is undeclared in the next line
Debug.WriteLine(intLoop.ToString())
```

Tạo đối tượng

Các đối tượng bây giờ được tạo khi được khai báo vì câu lệnh Dim giờ là một câu lệnh khả thi và bạn có ba phương thức có thể dùng để tạo một đối tượng.

Phương thức 1:

```
Dim oCust As New Customer()
```

Phương thức 2:

```
Dim oCust As Customer = New Customer()
```

Phương thức 3:

```
Dim oCust As Customer
oCust = New Customer()
```

Khi bạn tạo một lớp, bạn được phép tạo cấu tử tùy biến riêng.

```
Dim oCust As New Customer(5, "Bill Jones")
Dim oCust As Customer =
 New Customer(10, "Sally James")
```

Bạn sẽ tìm hiểu cách xây dựng các cấu tử tùy biến sau. Nó rất thuận tiện để có thể khởi tạo các đặc tính nào đó ngay khi bạn tạo một đối tượng mới.

Chuyển tham số

Tất cả các tham số bây giờ được chuyển bởi giá trị (ByVal) thay vì bởi các tham chiếu (ByRef) như ở các phiên bản trước đây của Visual Basic. Bất kỳ các tham số tùy chọn nào bạn tạo đều

phải có giá trị mặc định được gán cho chúng. Hàm IsMissing và IsEmpty không còn được hỗ trợ trong Visual Basic.NET.

Khai báo thủ tục

Bạn không còn có thể chỉ định Static Sub hoặc Static Function để khai báo tất cả các biến bên trong các thủ tục đó là static. Bây giờ bạn phải khai báo tất cả các biến là Static mà bạn muốn có.

Các thủ tục gọi

Trong VB6 bạn phải có các dấu ngoặc khi sử dụng câu lệnh Call. Tuy nhiên, khi bạn không dùng câu lệnh Call, bạn không thể gộp chúng. Bây giờ việc ứng xử không nhất quán này được sửa đổi trong Visual Basic.NET và bạn luôn luôn sử dụng các dấu ngoặc.

```
CustomerDisplay(1)
Call CustomerDisplay(1)
```

Mảng Param

Mảng Param được chuyển tới thủ tục phải được chuyển bằng ByVal. Kiểu dữ liệu duy nhất của các phần tử trong mảng Param phải thuộc kiểu Object.

Các thay đổi control luồng

Có một số câu lệnh control luồng không còn dùng được trong ngôn ngữ Visual Basic.NET.

```
GoSub
On <Exp> GoTo
On <Exp> GoSub
```

While...Wend đã đổi thành While...End While
Tuy nhiên, câu lệnh On Error GoTo vẫn còn được hỗ trợ.

Các hàm được thay thế bằng các phương thức

Một số hàm bạn có thể đã dùng trước đây bây giờ đã đổi thành các phương thức là một phần của thư viện lớp (class library) của .NET. Bảng sau liệt kê một số hàm này.

Hàm	Được thay bằng
Circle()	System.Drawing.DrawEllipse
Line()	System.Drawing.DrawLine
Atn()	System.Math.Atan
Sgn()	System.Math.Sign
Sqr()	System.Math.Sqrt
Rnd()	System.Math.Rnd
Round()	System.Math.Round
IsNull()	IsDBNull()
IsObject()	IsReference()
VarType()	System.Object.GetType()
MsgBox	MessageBox.Show()
LSet	System.String.PadRight
RSet	System.String.PadLeft
DoEvents	System.WinForms.Application.DoEvents

Các mục được xóa bỏ/thay đổi

Có một số thay đổi ngôn ngữ hỗn hợp khác không thích hợp với bất kỳ phân loại nào. Sau là một số:

- Pset và Scale được loại bỏ.

- Empty và Null được loại bỏ.
- IsEmpty() được loại bỏ.
- Bất kỳ một hàm cũ đã cho trở lại Null trong các phiên bản trước đây của Visual Basic không còn nữa.
- Thêm Null vào một biểu thức bất kỳ sẽ tạo ra lỗi.
- Tiền tố "vb" đã được xóa khỏi tất cả các hằng.
- Bây giờ các hằng là một phần của lớp ControlChars.
- Không cần sử dụng $ sau các hàm, thực ra, dù sao trình soạn thảo cũng sẽ xóa chúng.
- Let a=b không còn hoạt động.
- Class_Initialize và Class_Terminate được loại bỏ.
- VarPtr, VarPtrArray, VarPtrStringArray, ObjPtr, StrPtr được loại bỏ.

Các thay đổi của WinForm

Có nhiều thay đổi về sử dụng các form trong Visual Basic.NET. Bên dưới là một số các thay đổi này:

- Có thể xác lập TabIndex là 0 trên tất cả các control.
- Zorder được dùng để di chuyển qua các control vào thời gian chạy.
- Control LinkLabel mới cho các khả năng siêu liên kết.

- Control GroupBox thay thế control Frame.
- Control Splitter cho bạn giao diện giống Explorer.
- Hỗ trợ Drag & Drop khác nhau nhiều.
- Không có hỗ trợ DDE.
- Control Time không thể xác lập Interval bằng 0.
- Không hỗ trợ OLE Control.
- Không có control hình dạng.
- Không có control đường kẻ.
- Phương thức Form.PrintForm được loại bỏ.
- Sự tương tác của Clipboard hầu như khác nhau, bây giờ bạn sử dụng System.WinForms.Clipboard.
- Không thể đạt tới đặc tính Name của form.
- Phải khai báo biến form là Show a form.
- Nếu bạn đã xác lập đặc tính ScaleMode trên bất kỳ các form của VB6 là một đặc tính nào khác hơn Twips, bạn sẽ không thể di trú form đó.
- Không thể dùng dữ liệu liên kết với DAO hoặc RDO. Bạn vẫn có thể dùng các đối tượng này, nhưng chúng được coi là các công nghệ thừa kế và cuối cùng cũng sẽ mất.

Gỡ rối

Debug.Print đã được thay thế bằng Debug.Write và Debug.WriteLine.

Các công cụ CSDL

Cửa sổ DataView được loại bỏ và bây giờ thay thế bằng Server Explorer.

Tất cả các công cụ CSDL đã được nâng cấp để làm việc với SQL Server 2000, bao gồm Cascade Update/Delete và Indexed Views.

Các ứng dụng Web

Việc xây dựng các ứng dụng Web trong Visual Basic.NET hoàn toàn khác. Do đó, nếu bạn đã dùng một số kĩ thuật xây dựng Web trong VB6, bạn sẽ phải viết lại chúng từ vùng gom tin.

- Các ứng dụng IIS không còn được hỗ trợ. Thay vào đó, bạn sẽ dùng ASP.NET và WebForms.
- Các ứng dụng DHTML không còn được hỗ trợ. Hãy duy trì các ứng dụng này trong VB6.
- Các tài liệu ActiveX không còn được hỗ trợ. Hãy duy trì các ứng dụng này trong VB6.

Add-Ins

Có một kiểu đối tượng hoàn toàn mới cho Add-Ins. Có nhiều móc nối (hook) hơn để làm việc với IDE mới của Visual Studio.NET.

Chuẩn bị cho Visual Basic.NET

Để chuẩn bị nâng cấp lên Visual Basic.NET bạn cần phải loại bỏ càng nhiều thứ càng tốt khỏi các ứng dụng hiện hành đã được thảo luận ở chương này. Tốt nhất nếu bạn đang viết các ứng dụng bằng cách dùng các Class và các kĩ thuật phát triển bậc *n* càng nhiều càng tốt. Các kiểu ứng dụng này sẽ di trú dễ dàng hơn nhiều so với các kiểu ứng dụng khác.

Nếu nâng cấp

Rất có thể bạn cần duy trì các ứng dụng VB6 sẵn có trong VB6. Không cần nâng cấp chúng lên .NET trừ khi bạn cần khai thác WebServices hoặc một số tính năng khác mà .NET cung cấp cho bạn. Visual Basic.NET và VB6 có thể chạy bên nhau trên cùng một máy, vì vậy bạn không cần phát triển trên hai máy.

Nâng cấp ứng dụng VB6

Bạn sẽ thấy có thể nâng cấp các ứng dụng VB6 dễ dàng hơn nhiều nếu bạn thực hiện các điều sau ngay bây giờ.

- Sử dụng các khái niệm bậc *n*.
- Không sử dụng DefInt, DefLng, vân vân.
- Không sử dụng các mảng khác zêrô.
- Lập trình bằng cách dùng các kĩ thuật có cấu trúc.
- Không thường sử dụng Variant quá.

- Sử dụng các Constant, chẳng hạn như vbMaximized và True.

- Sử dụng ADO.

Tóm tắt

Mặc dù có nhiều tính năng mới và tính năng đã thay đổi trong ngôn ngữ Visual Basic.NET, nhưng ngôn ngữ lõi vẫn giống nhau. Tuy nhiên, việc nâng cấp các ứng dụng sẵn có lên .NET sẽ phức tạp và mất thời gian, còn .NET Framework cung cấp một môi trường phát triển hiệu quả hơn. Không cần phải nâng cấp các ứng dụng sẵn có như VB6, vì nó vẫn là một ngôn ngữ rất tốt. Đối với các ứng dụng mới, Visual Basic.NET sẽ đem lại nhiều hiệu quả hơn sau khi bạn bước vào tìm hiểu cách làm việc với các lớp .NET framework mới.

Câu hỏi ôn tập

1. Kiểu dữ liệu nào đã thay thế kiểu dữ liệu Currency?
2. Đúng hay sai: Các chuỗi có chiều dài cố định bây giờ được thực thi bằng cách khác trong .NET?
3. Kiểu dữ liệu nào thay thế Variant?
4. Kiểu dữ liệu Date bây giờ lưu trữ như kiểu dữ liệu nào bên trong?
5. Hai toán tử nào được đưa vào để xử lý ngắn mạch trong các câu lệnh If?

Trả lời câu hỏi ôn tập

1. Decimal.
2. Đúng.
3. Object.
4. Integer.
5. AndAlso và OrElse.

Chương 5

Lớp DataTable

- Tìm hiểu cách nạp dữ liệu vào DataGrid
- Tìm hiểu cách dùng DataTable trong các lớp DataRow

Lớp DataTable

DataTable là một lớp của .NET, nó giống bảng CSDL quan hệ. Nó được tạo thành bởi các dòng dữ liệu với mỗi dòng có một loạt các cột. Mỗi cột có thể có tên, kiểu dữ liệu và các thuộc tính khác được gán cho nó, chẳng hạn như ReadOnly và DefaultValue. DataTable được tạo thành bởi một tập hợp các đối tượng DataRow và mỗi đối tượng DataRow được tạo thành bởi một tập hợp các đối tượng DataColumn.

Tại sao sử dụng DataTable

DataTable có khả năng lưu giữ dữ liệu khi bạn cần lưu trữ nhiều dòng và nhiều giá trị trong mỗi dòng. Có nhiều cách bạn có thể thực hiện cùng tác vụ này, nhưng DataTable rất dễ dùng. Một số cơ chế khác có thể bạn đã dùng trước đây, gồm:

- Mảng các kiểu người dùng xác định.
- Tập hợp các lớp.
- ADO Recordset.

Mặc dù tất cả các cơ chế này đều hoạt động, nhưng chúng đòi hỏi bạn phải viết một ít mã để thực hiện những gì bạn đang cố làm. Bạn sẽ thấy là đối tượng DataTable dễ xử lý hơn nhiều.

Tạo DataTable

Vào form bạn đã tạo ở chương trước đây, bây giờ bạn nạp dữ liệu nào đó vào DataTable, sau đó được DataGrid sử dụng DataTable để hiển thị dữ liệu trong khung lưới. Đầu tiên bạn sẽ tạo một biến bên trong lớp form, nhưng bên ngoài bất kỳ thủ tục nào. Nó được coi là biến thành viên. Thêm mã được trình bày ở đoạn mã bên dưới.

```
Public Class frmProductsSimple
    Inherits System.Windows.Forms.Form

    Private mdt As DataTable
```

Bạn đã thêm biến Private có tên *mdt*. Biến này được khai báo là lớp DataTable và sẽ được dùng để lưu giữ dữ liệu của bạn. Mã bên dưới trình bày cách tạo DataTable này và đưa dữ liệu vào nó.

```
Private Sub btnLoad_Click(ByVal sender As
System.Object, _
 ByVal e As System.EventArgs) Handles btnLOad.Click
    Dim dr As DataRow
    Dim intLoop As Integer

    Me.Cursor = Cursors.WaitCursor

    ' Create New DataTable object
    mdt = New DataTable

    ' Create Columns in DataTable
    mdt.Columns.Add("ProductID")
    mdt.Columns.Add("ProductName")
    mdt.Columns.Add("UnitPrice")
    mdt.Columns.Add("UnitsInStock")

    ' Load data into DataTable
    For intLoop = 1 To 10
        ' Create a New Row
        dr = mdt.NewRow

        ' Load columns in new row with Data
        dr("ProductID") = intLoop
        dr("ProductName") = "Product " & _
         intLoop.ToString()
        dr("UnitPrice") = intLoop * 10
        dr("UnitsInStock") = intLoop * 20

        ' Add New Row to DataTable
        mdt.Rows.Add(dr)
    Next

    ' Tell Grid to AutoSize Columns
    grdProducts.PreferredColumnWidth = _
        grdProducts.AutoColumnSize
    ' Load Grid with Data
    grdProducts.DataSource = mdt

    Me.Cursor = Cursors.Default
End Sub
```

Ở mã trên bạn khai báo đối tượng DataRow có tên *dr*. Sau đó bạn tạo một thể nghiệm mới của DataTable bằng cách dùng từ

khóa New để tạo một thể nghiệm mới của kiểu này. Kế tiếp, bạn định nghĩa các cột bạn sẽ dùng với đối tượng DataTable này. Bạn làm điều này bằng cách dùng phương thức Add trên tập hợp các cột và đặt tên cột mới.

Tiếp theo bạn xây dựng một vòng lặp để phát sinh dữ liệu sản phẩm nào đó. Bạn sẽ lặp qua 10 lần và mỗi lần qua vòng lặp bạn sẽ tạo một đối tượng DataRow mới. Đối tượng DataRow được xây dựng với các định nghĩa cột bạn đã định nghĩa ban đầu trên DataTable. Vào các cột trong DataRow này bạn sẽ đặt bộ đếm vòng lặp vào cột ProductID, chuỗi "Product 1", "Product 2", vân vân, vào cột ProductName và thực hiện một số phép tính để xác lập dữ liệu cột UnitPrice và UnitsInStock. Cuối cùng bạn thêm đối tượng DataRow mới này vào tập hợp Rows của đối tượng DataTable bằng cách dùng phương thức Add.

Khi DataTable được xây dựng thì bạn có thể xác lập đặc tính DataSource của control DataGrid là đối tượng DataTable này. Trước khi thực hiện điều này, bạn phải xác lập đặc tính PreferredColumnWidth là đặc tính AutoColumnSize. Nó sẽ thay đổi kích thước các cột dựa trên giá trị dữ liệu lớn nhất trong mỗi cột khi nó nạp dữ liệu.

Truy tìm DataRow đơn

Sau khi khung lưới được nạp, bạn nên lựa một dòng đơn để có thể trình bày dữ liệu trong các hộp văn bản trên form này. Để thực hiện điều này, bạn cần biết số dòng người dùng đã nhấp. Bạn có thể tìm được nó bằng cách kiểm tra đặc tính RowNumber trên đặc tính CurrentCell của khung lưới khi bạn ở trong thủ tục sự kiện Click của DataGrid. Khi bạn có số dòng, bạn có thể dùng nó như chỉ mục vào tập hợp Rows để truy tìm DataRow bạn đã nhấp.

```
Private Sub grdProducts_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles grdProducts.Click
    Dim dr As DataRow
    Dim intRow As Integer

    ' Get current row number
    intRow = grdProducts.CurrentCell.RowNumber
    ' Get DataRow from DataTable
    dr = mdt.Rows(intRow)

    txtProductID.Text = dr("ProductID")
    txtProductName.Text = dr("ProductName")
    txtUnitPrice.Text = dr("UnitPrice")
    txtUnitsInStock.Text = dr("UnitsInStock")
End Sub
```

Khi bạn truy tìm DataRow, bạn có thể truy xuất từng cột bằng tên của nó để truy tìm dữ liệu và đặt dữ liệu đó vào từng hộp văn bản trên form này.

Tóm tắt

Ở chương này, bạn đã được giới thiệu về lớp DataTable. Lớp này cho phép bạn xây dựng một tập dữ liệu và truy tìm từng cột từ tập dữ liệu này. Bạn sẽ thấy DataTable là một sự thay thế rất tốt cho các mảng hai chiều, các tập hợp và cho ADO Disconnected Recordset.

Câu hỏi ôn tập

1. Hai đối tượng nào làm thành DataTable?

2. Bạn dùng phương thức nào để tạo một đối tượng DataColumn mới?

3. Bạn dùng phương thức nào để tạo một đối tượng DataRow mới?

4. Bạn gán đặc tính nào vào DataTable để có khung lưới sử dụng DataTable này như nguồn dữ liệu của nó?

Bài tập

- Nạp DataGrid trên form bạn đã tạo ở chương trước bằng cách gõ vào mã được trình bày ở chương này.

Trả lời câu hỏi ôn tập

1. DataRow và DataColumn.

2. Add.

3. NewRow.

4. DataSource.

Chương 6

Làm việc với lớp MessageBox

- Tìm hiểu cách dùng lớp MessageBox.

MessageBox

Nếu bạn đã từng sử dụng Windows, bạn chắc chắn đã gặp cơ chế cảnh báo được tất cả các ứng dụng Windows sử dụng. Hộp thoại *modal* này xuất hiện trong nhiều trường hợp, đôi khi với một biểu tượng, đôi khi với nhiều nút. Bạn có thể điều khiển ứng xử của hộp thoại lập sẵn này và cũng có thể xác định nút người dùng đã nhấp, vì vậy bạn có thể thao tác thích hợp để đáp ứng hộp thoại đó. Ở chương này, bạn sẽ thấy cách dùng lớp MessageBox được .NET framework cung cấp.

Thuật ngữ cần lưu ý:

Modal: Khi thuật ngữ này được áp dụng vào một form, có nghĩa là form "nắm giữ" tiêu điểm và không cho phép bạn làm việc với bất kỳ một form nào khác trong ứng dụng của bạn cho tới khi bạn đóng form ấy.

Dialog box: Một form bất kỳ chiếm giữ tiêu điểm nhập liệu và không trao lại tiêu điểm ấy cho tới khi bạn đóng hộp thoại này. Khi bạn mở một form nắm giữ tiêu điểm là bạn đang tạo một hộp thoại.

Tại sao cần làm điều này?

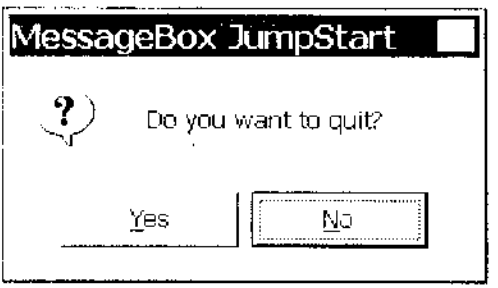
Lớp MessageBox cung cấp một cách thức được chuẩn hóa để tạo các form hộp thoại để yêu cầu thông tin đơn giản từ người dùng. Mặc dù bạn có thể tạo các form hộp thoại riêng, khi bạn tìm hiểu ở phần sau ở tài liệu này, nhưng nếu bạn chỉ cần trả lời kiểu Yes/No, tốt hơn bạn nên dùng lợi thế của lớp MessageBox lập sẵn.

Mẹo:
Nếu bạn chuyển từ VB6 sang VB.NET, có lẽ bạn đã dùng hàm MsgBox. Lớp MessageBox thay thế hàm này có tính năng tương tự.

Sử dụng lớp MessageBox

Hãy tạo một form chứa một nút. Sự tác động trở lại khi nhấp nút ấy sẽ hiển thị một cảnh báo với lời nhắc "Do you want to quit?" (Bạn có muốn thoát không?) và biểu tượng dấu chấm hỏi,

nút No được chọn mặc định. Tiêu đề của hộp thoại là "MessageBox Chapter". Nếu bạn nhấp Yes, đóng form. Nếu bạn nhấp No, giữ nguyên form. Khi bạn chạy form của mình, cảnh báo sẽ giống như hình 1.



Hình 1: *Nhấp Yes để đóng form và No để nó mở.*

.NET framework cung cấp lớp MessageBox để tạo cho người phát triển dễ hiển thị các hộp thoại đơn giản và yêu cầu thông tin vào thời gian chạy. Lớp này cho phép bạn hiển thị hộp thoại được định dạng theo cách riêng biệt, nó có thể trình bày một hoặc nhiều nút, trong số đó một nút có thể được mặc định. (Nếu bạn ấn **ENTER** thay vì nhấp một nút, bạn sẽ có hiệu ứng của nút mặc định).

Các bước

Lớp MessageBox làm việc hầu như giống lớp Clipboard cũ trong VB6. Bạn không cần thuyết minh bất kỳ đối tượng nào của lớp này mà bạn chỉ cần dùng các phương thức và các đặc tính của chính tên lớp. Thực hiện các bước sau chạy thử nó:

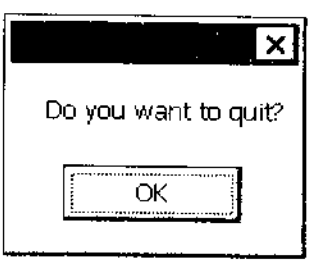
1. Tạo một project Visual Basic mới (một project Windows chuẩn) có tên **MessageBoxTest**.

2. Thêm một nút vào Form1 và xác lập đặc tính **Text** của nó là "Test MessageBox".

3. Nhấp đôi vào nút ấy để nhảy tới handler sự kiện Click của nó và thêm mã như được trình bày sau:

```
Private Sub btnSimple_Click( _
 ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnSimple.Click
    MessageBox.Show("Do you want to quit?")
End Sub
```

4. Chạy project, nhấp nút trên form và bạn sẽ thấy hộp thoại như được trình bày ở hình 2. (Lưu ý, mất tiêu đề và biểu tượng — xảy ra vì bạn đã không chỉ định đầy đủ thông tin khi gọi phương thức Show).



Hình 2: *Khi bạn gọi phương thức Show chỉ với một tham số, hộp thoại của bạn không có tiêu đề hoặc biểu tượng. Nó chỉ trình bày một nút OK.*

Ở lần tạo thử hộp thoại cảnh báo đầu tiên này, bạn đã đặt văn bản chính xác vào hộp thoại, nhưng các nút không đúng, không có tiêu đề và biểu tượng. Nếu bạn chỉ chỉ định văn bản khi gọi phương thức Show của lớp MessageBox, thì bạn chỉ có văn bản.

Thêm tiêu đề

Phương thức Show cung cấp ba phiên bản (bạn đã dùng một phiên bản cho phép bạn chỉ định văn bản được hiển thị trong hộp thoại với một nút OK). Phiên bản thứ hai cho phép bạn chỉ định hai chuỗi: văn bản được hiển thị và tiêu đề cho hộp thoại. Thực hiện theo các bước sau để thêm tiêu đề:

- Sửa đổi mã bạn đã tạo thành mã sau:

```
Private Sub btnTitle_Click( _
 ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnTitle.Click
    MessageBox.Show("Do you want to quit?", _
     "MessageBox Sample")
End Sub
```

- Chạy lại project, lưu ý, bây giờ bạn thấy tiêu đề trên hộp thoại.

Xử lý các nút và biểu tượng

Phiên bản thứ ba của phương thức Show cho phép bạn chỉ định ba tính năng của hộp thoại, được kết hợp vào một tham số:

- Biểu tượng
- Các nút
- Nút mặc định

Để chỉ định biểu tượng nào, các nút nào và nút mặc định nào bạn muốn có, lựa các giá trị từ bảng 1. Bảng này có ba phần (một cho các nút, một cho các biểu tượng và một để lựa nút mặc định). Bạn có thể chọn một mục của mỗi phần (hoặc không chọn mục nào từ mỗi phần), chỉ định các tham chiếu của bạn. Cộng các chọn lựa của bạn, bằng cách dùng toán tử +. (Về cơ bản, mỗi mục trong bảng 1 biểu thị hằng numeric và việc sử dụng toán tử +,

bạn sẽ kết hợp các giá trị được cung cấp ở mỗi hằng vào một giá trị mà phương thức Show có thể sử dụng). Như bạn thấy, nếu bạn không lựa biểu tượng, bạn sẽ không có nó. Nếu bạn không lựa các nút, bạn chỉ có nút OK. Thực hiện các bước sau, để hoàn tất mã:

- Sửa đổi thủ tục của bạn thành mã sau:

```
Private Sub btnButtons_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnButtons.Click
    MessageBox.Show("Do you want to quit?", _
     "MessageBox Sample", _
     MessageBoxButtons.YesNo, _
     MessageBoxIcon.Hand, _
     MessageBoxDefaultButton.Button2)
End Sub
```

- Chạy project, kiểm tra chắc chắn bạn thấy hộp thoại được trình bày như ở hình 1.

	Đặc tính	Mô tả
Các nút	AbortRetryIgnore	Trình bày các nút Abort, Retry và Ignore.
	OK	Trình bày nút OK.
	OKCancel	Trình bày nút OK và Cancel.
	RetryCancel	Trình bày nút Retry và Cancel.
	YesNo	Trình bày nút Yes và No.
	YesNoCancel	Trình bày nút Yes, No và Cancel.
Các biểu tượng	IconAsterisk	Trình bày biểu tượng dấu sao.
	IconError	Trình bày biểu tượng bàn tay.
	IconExclamation	Trình bày biểu tượng dấu chấm than.
	IconHand	Trình bày biểu tượng bàn tay.
	IconInformation	Trình bày biểu tượng dấu sao.

	IconQuestion	Trình bày biểu tượng dấu chấm hỏi.
	IconStop	Trình bày biểu tượng bàn tay.
	IconWarning	Trình bày biểu tượng dấu chấm than.
Nút mặc định	DefaultButton1	Nút đầu tiên là nút mặc định.
	DefaultButton2	Nút thứ hai là nút mặc định.
	DefaultButton3	Nút thứ ba là nút mặc định.

Bảng 1: *Chọn các kiểu giá trị từ bảng này: Zêrô hoặc thêm các mục ở mỗi phần cộng lại với nhau.*

Bây giờ làm gì?

Bây giờ bạn đã gọi phương thức Show và định dạng hộp thoại theo yêu cầu, làm thế nào bạn có thể cho biết bạn đã nhấp nút nào? Để cung cấp thông tin này, phương thức Show cho trở lại giá trị kiểu DialogResult, chỉ định nút nào được nhấp khiến cho hộp thoại biến mất. Xem bảng 1, bạn có thể cho biết có khả năng có 7 giá trị trở lại: Abort, Retry, Ignore, OK, Cancel, Yes và No. Sự liệt kê của DialogResult có bảy giá trị này — bạn có thể sử dụng câu lệnh Select Case để xác định hoạt động nào xảy ra để tác động trở lại phương thức MessageBox Show.

Để đóng form nếu bạn nhấp OK trên hộp thoại, thực hiện các bước sau:

1. Sửa đổi mã trong handler sự kiện thành mã sau:

```
Private Sub btnResult_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnResult.Click
    Dim dr As DialogResult

    dr = MessageBox.Show("Do you want to quit?", _
     "MessageBox Sample", _
```

```
            MessageBoxButtons.YesNo, _
            MessageBoxIcon.Hand, _
            MessageBoxDefaultButton.Button2)

        Select Case dr
            Case DialogResult.Yes
                Me.Close()

            Case DialogResult.No
                ' Do Nothing at all

        End Select
    End Sub
```

2. Chạy project một lần nữa và thử lựa Yes và sau đó lựa No. Xác định là form gốc đóng nếu bạn nhấp Yes trên hộp thoại, nhưng vẫn mở nếu bạn nhấp No.

Lớp MessageBox còn cung cấp thêm một vài đặc tính nữa, hầu như để xử lý các vấn đề quốc tế và làm việc với các cảnh báo từ trong các dịch vụ Windows. Ngoài ra, phương thức Show của lớp MessageBox có thêm ba phiên bản, một tương ứng với phiên bản được trình bày ở chương này với tham số đầu tiên cho phép bạn chỉ định đối tượng IWin32Window là chủ nhân của cảnh báo. Nếu bạn cần chỉ định một form Windows là chủ nhân của đối tượng MessageBox bạn tạo, bạn có thể gọi phương thức Show như sau:

```
MessageBox.Show(YourForm, "Text", "Title")
```

Bạn cũng nên tạo các form hộp thoại riêng — đối tượng MessageBox có thể quá giới hạn các nhu cầu của bạn. Bạn có thể tạo các form hộp thoại riêng và có một chương riêng đề cập cách bạn có thể làm điều đó.

Tóm tắt

Lớp MessageBox có tính năng giống hàm MsgBox cũ trong VB6. Bạn chuyển vào các tham số khác nhau, nhưng kết quả cuối cùng sẽ giống nhau. Nó có thể được dùng giống như một câu lệnh để hiển thị hộp thoại modal hoặc có thể được dùng như một hàm để cho trở lại kết quả.

Câu hỏi ôn tập

1. Tham số đầu tiên được chuyển tới MessageBox.Show sẽ thực hiện điều gì?
2. Nếu bạn muốn trình bày nút Abort, Retry và Ignore, bạn chuyển hằng nào vào tham số thứ ba?
3. Chuyển hằng nào để có biểu tượng Error xuất hiện trong MessageBox?
4. Kiểu dữ liệu nào được cho trở lại từ phương thức MessageBox.Show?

Trả lời câu hỏi ôn tập

1. Hiển thị chuỗi đó trong MessageBox
2. AbortRetryIgnore.
3. IconError.
4. DialogResult.

Chương 7

MDI và các menu

- Tìm hiểu cách tạo các form cha MDI.
- Tìm hiểu cách tạo các form con MDI.
- Tìm hiểu cách tạo và sử dụng các menu.

MDI

Chương này sẽ trình bày với bạn khái niệm về MDI (Multiple Document Interface – Giao diện đa tài liệu). Bạn sẽ tìm hiểu cách tạo ứng dụng MDI trong Visual Basic và tìm hiểu tại sao bạn nên sử dụng giao diện này. Bạn cũng tìm hiểu về các form con được nằm trong ứng dụng MDI. Ngoài ra, bạn còn tìm hiểu cách tạo các menu nhạy ngữ cảnh (context) và bật lên.

MDI là một giao diện phổ biến khi bạn có nhiều tài liệu/form được mở trong một ứng dụng. Các thí dụ về ứng dụng MDI gồm

Microsoft Word, Microsoft Excel, Microsoft PowerPoint và thậm chí cả môi trường phát triển tích hợp Visual Basic. Mỗi ứng dụng này cho phép bạn mở nhiều tài liệu và tất cả chúng đều được chứa trong một form container. Form container cũng được coi là cửa sổ cha. Bạn được phép có một hoặc nhiều cửa sổ cha. Mỗi form con có một cửa sổ riêng, nhưng được giới hạn bởi các biên của cửa sổ cha. Điều này có nghĩa là bạn không thể kéo form ấy ra ngoài cửa sổ cha.

***Hình 1:** MDI được dùng để mở nhiều cửa sổ và để tất cả chúng ở trong vùng của cửa sổ cha.*

SDI

MDI là một cơ chế duy nhất để tạo ứng dụng. Có thể có ứng dụng chỉ có một form. Đây là trường hợp của ứng dụng SDI (Single Document Interface – Giao diện tài liệu đơn). Notepad là

một thí dụ về SDI. Khi bạn ở trong Notepad bạn chỉ có thể soạn thảo và xem mỗi lần một file.

Bạn không phải tạo các ứng dụng theo mô hình MDI. Dù bạn có nhiều form trong project, nhưng bạn chỉ có thể có một form là form độc lập tách riêng không nằm trong form cha nào. Điều trở ngại duy nhất của điều này là nếu bạn muốn đóng ứng dụng của bạn, chúng phải đóng từng form riêng hoặc bạn phải viết mã để đóng tất cả các cửa sổ khi bạn muốn đóng toàn bộ ứng dụng.

Sử dụng MDI

MDI được dùng phổ biến nhất trong các ứng dụng, nơi người dùng có thể có nhiều form để làm việc đồng thời. Các ứng dụng xử lý từ (như Word), các ứng dụng bảng tính (như Excel) và các ứng dụng trình quản lý project (như Project) đều thích hợp tốt với các ứng dụng MDI. MDI cũng thuận tiện khi bạn có một ứng dụng lớn và muốn cung cấp một cơ chế để đóng tất cả các form khi người dùng thoát khỏi ứng dụng.

Tạo form MDI

Để tạo form MDI bạn có thể chỉ dùng một trong các form sẵn có và xác lập đặc tính IsMDIContainer là True. Bây giờ form này sẽ có thể chứa các form khác như các form con. Bạn có thể có một hoặc nhiều form container trong ứng dụng của bạn.

Bạn có thể có bao nhiêu form con tùy theo bạn muốn trong project. Các form con là các form nằm trong container của MDI. Một form Child không có gì khác hơn một form thông thường mà bạn xác lập đặc tính MdiParent là tham chiếu của form container của MDI. Mã bên dưới được dùng để phát sinh hình 1.

```
Dim frmParent As frmParent
Dim frm As frmProducts

frmParent = New frmParent()
frmParent.Show()

frm = New frmProducts()
frm.MdiParent = frmParent

frm.Show()

' Open 2nd instance
frm = New frmProducts()
frm.MdiParent = frmParent

frm.Show()
```

Ở mã trên, bạn tạo hai tham chiếu form. Form *frmParent* xác lập đặc tính IsMdiContainer là True vào thời gian thiết kế. Sau khi thuyết minh một form Product mới, bạn có thể xác lập form MdiParent của nó là tham chiếu với form cha. Bây giờ form *frmProducts* là form con và ứng xử như một form con dưới form cha này. Đương nhiên, nếu bạn không xác lập đặc tính MdiParent trên form *frmProducts*, thì *frmProducts* sẽ đóng vai trò như một form bình thường. Các tính năng của form con MDI được mô tả ở phần kế tiếp.

Các tính năng vào thời gian chạy của các form con MDI

Vào thời gian chạy, form MDI và các form con MDI đảm nhận các tính năng đặc biệt. Bên dưới là danh sách các tính năng này:

- Tất cả các form con được hiển thị trong vùng *client* của form cha MDI. Vùng client là vùng bên dưới thanh tiêu đề

của form cha MDI, các menu bất kỳ và các thanh công cụ bất kỳ.

- Các form con có thể được di chuyển và lập kích thước chỉ trong vùng client của form cha MDI.
- Các form con có thể thu nhỏ và biểu tượng của chúng sẽ được hiển thị trong vùng client của form cha.
- Các form con có thể được phóng to trong vùng client của form cha và caption (chú giải) được gắn với caption của form MDI.
- Windows tự động cho các form con có đặc tính BorderStyle được xác lập thành đường viền lớn và kích thước mặc định. Kích thước này dựa vào kích thước vùng client của form cha MDI. Bạn có thể ghi đè kích thước này bằng cách xác lập đặc tính BorderStyle của form con là một trong các kiểu đường viền cố định.
- Các form con không thể được hiển thị là một form có tiêu điểm.
- Form MDI có thể được thu nhỏ và một biểu tượng duy nhất sẽ được hiển thị trên desktop biểu thị form MDI và tất cả các form con của nó.
- Nếu form MDI không được nạp, tất cả các form con đã nạp cũng sẽ không được nạp.

Lưu ý:

Vùng client là một vùng bất kỳ có thể dùng trên form MDI, trừ các thanh công cụ hoặc các thanh trạng thái có thể ở trên form MDI.

Tạo project MDI

Bây giờ bạn sẽ tìm hiểu cách tạo ứng dụng MDI bằng cách dùng VB.NET. Để thực hiện điều này, bạn sẽ tạo một form mới là form cha MDI. Bạn sẽ thêm một số menu vào form mới này, sau đó bạn gọi form Product từ một menu là một form con.

Tạo form cha MDI

Bước đầu tiên bạn sẽ tạo form cha MDI.

1. Mở Visual Studio.NET.
2. Tạo một project Windows Application mới.
3. Đặt tên của project là **MDI.sln**.
4. Đặt lại tên form được tạo tự động là **frmMain.vb**.
5. Với form frmMain được hiển thị, xác lập đặc tính IsMdiContainer là **True**.
6. Xác lập đặc tính WindowState là **Maximized**.

Hoàn tất các bước trên bạn sẽ có một form cha MDI.

Tạo các menu

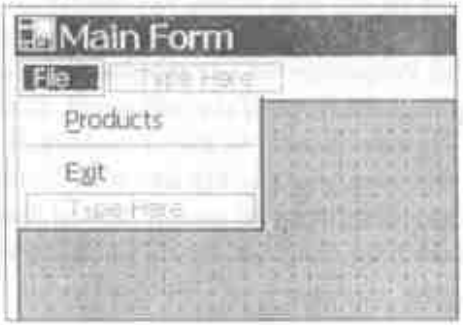
Bây giờ bạn cần thêm một tập các menu vào form chính. Từ các mục menu này, là nơi bạn sẽ thực hiện các thao tác, chẳng hạn như mở các form con, chép dữ liệu, dán dữ liệu và sắp xếp các cửa sổ. Có một trình thiết kế menu mới trong Visual Studio.NET đảm trách tạo và sửa đổi các menu nhanh chóng. Các menu được xử lý như một kiểu control khác trong .NET framework.

Nhấp đôi công cụ MainMenu trong cửa sổ Toolbox để thêm một đối tượng mới có tên MainMenu1 vào khung form.

Ở trên đầu form cha MDI bạn sẽ thấy một hộp nhỏ với văn bản Type Here ở trong nó. Nhấp vào vùng này và gõ **&File**.

1. Ấn phím Enter để di chuyển tới mục menu kế tiếp. Nhập văn bản **&Products**.
2. Ấn phím Enter để di chuyển tới mục menu kế tiếp và nhập kí tự gạch nối (-).
3. Ấn phím Enter một lần nữa và nhập "E&xit".

Bây giờ bạn đã tạo menu thả xuống đầu tiên của mình trên form chính. Bạn sẽ có một menu như ở hình 2.



Hình 2: *Trình thiết kế menu cho phép bạn gõ vào cấu trúc menu theo kiểu WYSIWYG.*

Ở cùng cấp và ở bên phải menu File, bạn sẽ thấy một hộp nhỏ với văn bản Type Here. Nhấp vào vùng menu và gõ vào các mục menu bên dưới bằng cách sau mỗi mục ấn phím Enter.

&Edit
Cu&t
&Copy
&Paste

Một lần nữa ở cùng cấp và ở bên phải menu Edit, thêm các mục menu sau vào như cách ở trên.

&Window
&Cascade
Tile &Horizontal
Tile &Vertical
&Arrange Icons

Tạo các tên cho mỗi menu

Sau khi tạo tất cả các mục menu, xác lập đặc tính Name cho mỗi mục. Thay vì mỗi lần nhấp vào từng mục menu và sau đó chuyển tới cửa sổ Properties để xác lập đặc tính Name, Visual Studio cung cấp một shortcut: Nhấp nút chuột phải vào một mục trên menu, rồi lựa Edit Names từ menu context. Bây giờ bạn có thể nhấp từng mục menu và xác lập đặc tính Name ở bên phải mỗi mục. Điều này chắc chắn sẽ nhanh hơn sử dụng cửa sổ Properties để thực hiện cùng tác vụ. Sử dụng các tên sau cho các mục menu của bạn:

mnuFile
mnuFProducts
mnuFExit
mnuEdit
mnuECut
mnuECopy
mnuEPaste
mnuWindow

mnuWCasade
mnuWHorizontal
mnuWVertical
mnuWArrange

Ở điểm này, bạn có thể chạy ứng dụng của mình bằng cách ấn F5 và sẽ thấy cửa sổ MDI chính của bạn xuất hiện với hệ thống menu.

Gọi form con

Bạn sẽ tạo mã để trình bày form như một form con bằng cách nhấp đôi vào File ➢ menu Products. Nó sẽ đưa ra thủ tục sự kiện Click cho menu mnuProducts. Gõ mã bên dưới vào.

```
Private Sub mnuFProducts_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles
mnuFProducts.Click
    Dim frm As frmProducts

    frm = New frmProducts()
    frm.MdiParent = Me

    frm.Show()
End Sub
```

Ở thủ tục sự kiện Click của mnuFProducts, bạn sẽ khai báo một biến form mới, *frm*, như kiểu frmProducts. Sau đó bạn tạo một thể nghiệm mới của form này bằng cách dùng từ khóa New. Để tạo form này là form con của form frmMain bạn sẽ gán Me vào đặc tính MdiParent của biến *frm*. Me là form hiện hành, là form cha MDI trong thí dụ của bạn. Bây giờ khi bạn gọi ra phương thức Show, nó sẽ hiển thị form này như form con với tất cả các tính năng được mô tả ban đầu.

Các menu trong các ứng dụng MDI

Không giống các phiên bản VB trước đây, bạn có thể chỉ định cách ứng xử của các menu ở trên các form con khi bạn hiển thị form con. Ở các phiên bản trước đây, các menu trên form con đã thay thế tất cả các menu trên form MDI chính. Trong VB.NET, bây giờ bạn có thể chỉ định những gì xảy ra với một menu bằng cách dùng đặc tính MergeOrder và MergeType.

Đặc tính MergeOrder được xác lập là zêrô theo mặc định. MergeType được xác lập là Add theo mặc định. Điều này có nghĩa là nếu bạn tạo một form con với một menu trên nó, thì menu ấy sẽ được thêm vào cuối menu chính. Xét ở hình 3, một form con được gọi từ menu chính. Vì form này có menu Maintenance trên nó và tất cả các đặc tính MergeOrder trên menu chính đều được xác lập là 0 và đặc tính MergeOrder của menu này được xác lập là 0, menu này sẽ được thêm vào cuối menu chính trên form MDI.

Hình 3: *Form con có các menu theo mặc định sẽ được thêm vào cuối menu trên menu chính.*

Để tạo form ở hình 3, thực hiện các bước sau:

1. Lựa Project > Add Windows Form từ menu VS.NET.
2. Đặt tên là frmChildWithMenus.vb.
3. Thêm control MainMenu vào form này.
4. Xác lập đặc tính Name là mnuMainMaint
5. Thêm các menu sau:

Menu	Tên
&Maintenance	mnuMaint
&Suppliers	mnuMSuppliers
&Catagories	mnuMCategories

Nếu bạn gọi form chính xác như bạn đã thực hiện form Products ở phần trước, bạn sẽ thấy form chính của bạn trông giống như hình 4. Bạn có thể thực hiện điều đó, theo mặc định menu được thêm vào cuối form này. Gọi form này bằng cách thêm một mục menu mới dưới menu file.

1. Mở frmMain.vb ở chế độ thiết kế.
2. Nhấp vào đường phân tách sau mục menu Products và ấn phím Insert để thêm mục menu mới.
3. Gõ vào **Child form with Menus** như Text của mục menu mới này.
4. Xác lập đặc tính Name trên mục menu mới này là **mnuFChild**.
5. Nhấp đôi mục menu mới này và thêm mã sau vào thủ tục sự kiện Click của nó.

```
Private Sub mnuFChildMenus_Click( _
 ByVal sender As System.Object, _
```

```
    ByVal e As System.EventArgs) Handles
mnuPChildMenus.Click
    Dim frm As frmChildWithMenus

    frm = New frmChildWithMenus()
    frm.MdiParent = Me

    frm.Show()
End Sub
```

Hình 4: *Theo mặc định, các menu được thêm vào cuối menu chính.*

Nếu bạn muốn thêm một menu vào giữa menu Edit và menu Window, bạn sẽ xác lập đặc tính MergeOrder trên menu Edit là 1, đặc tính MergeOrder trên menu Window là 2. Sau đó trên menu Maintenance trên form con, xác lập đặc tính MergeOrder là 1 và để nguyên MergeType là Add. Nó sẽ thêm menu Maintenance sau menu trên form chính với cùng số MergeOrder nó có.

Đặc tính MergeOrder xác định vị trí trong menu chính các mục menu khác của các form con được đặt vào khi form con được hiển thị. Đặc tính MergeType xác định những gì xảy ra với các mục menu đó. Bạn có bốn giá trị có thể xác lập MergeType như được trình bày ở bảng 1.

Giá trị	Mô tả
Add	Thêm menu này sau mục menu có cùng số MergeOrder được xác lập.
MergeItems	Kết hợp menu với menu có cùng số MergeOrder được xác lập.
Remove	Xóa menu với cùng số MergeOrder được xác lập.
Replace	Thay thế menu với cùng số MergeOrder được xác lập.

Bảng 1: *MergeType điều khiển nơi menu xuất hiện khi menu con được thêm vào cửa sổ MDI.*

Sắp xếp các form con

Nếu bạn có nhiều form con mở, bạn nên để chúng tự sắp xếp nhưng bạn có thể làm trong Word hoặc Excel dưới menu Window. Bốn tùy chọn bạn có với VB.NET là Tile Horizontal, Tile Vertical, Cascade và Arrange Icons. Thêm một số menu vào form chính cho mỗi tùy chọn này.

- Mở frmMain.vb ở chế độ thiết kế.
- Nhắp đôi mục menu Cascade dưới menu Window.
- Thêm mã như được trình bày bên dưới vào thủ tục sự kiện Click cho mục menu Cascade.

```
Private Sub mnuWCascade_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles mnuWCascade.Click
    Me.LayoutMdi(MdiLayout.Cascade)
End Sub
```

Phương thức LayoutMDI thay thế phương thức Arrange mà bạn có thể đã dùng trong VB6. Nhấp đôi mỗi mục menu khác dưới menu Window và thêm mã thích hợp vào mỗi mục.

Định vị các form con

Chạy chương trình và hiển thị form. Lưu ý, nó xuất hiện chỉ ở một điểm trên MDI. Nếu bạn muốn định vị tự động form này, bạn có thể xác lập đặc tính X và Y dưới đặc tính Location của form Product. Tuy nhiên, các vị trí này liên quan tới vùng client của form MDI. Vùng client của form MDI ở ngay dưới menu và ngay trên đáy form. Vì vậy các tọa độ 0,0 liên quan tới vị trí ngay dưới menu của form MDI.

Canh giữa các form

Đôi khi bạn có thể muốn canh giữa các form con bên trong form cha. Đặc tính StartPosition trên các form có thể được dùng để xác lập form là một trong năm giá trị.

Giá trị	Mô tả
Manual	Đặc tính vị trí và kích thước sẽ xác định vị trí của form này.
CenterScreen	Form sẽ được canh giữa bên trong các biên của vùng màn hình.
CenterOwner	Form sẽ được canh giữa bên trong vùng client của form cha.
WindowsDefaultLocation	Form được định vị bởi Windows ở vị trí mặc định, nhưng kích thước sẽ không thay đổi so với đặc tính kích thước của form.
WindowsDefaultBounds	Form được định vị bởi Windows ở vị trí mặc định và sẽ đổi kích thước

	thành kích thước được xác lập mặc định bởi Windows.

Ở Beta 2, đặc tính CenterOwner không hoạt động. Nếu bạn muốn canh giữa màn hình bạn có thể viết một ít mã. Đầu tiên bạn sẽ phải xác lập StartPosition của form con là Manual. Sau đó bạn chuyển form chính và form con tới thủ tục bạn sẽ viết gọi là CenterChildForm như phương thức của form frmMain.

```
Public Sub CenterChildForm(ByVal frmChild As Form)
    Dim ptLoc As Point = New Point()

    ptLoc.X = (Me.ClientRectangle.Width / 2) _
     - (frmChild.Size.Width / 2)
    ptLoc.Y = (Me.ClientRectangle.Height / 2) _
     - (frmChild.Size.Height / 2)

    frmChild.Location = ptLoc
End Sub
```

Để có thể canh giữa một form bạn cần biết chiều rộng và chiều cao của vùng client của form cha. Bạn có thể có được các giá trị này từ đặc tính ClientRectangle của form cha. Sau đó bạn chia các giá trị này cho hai. Kế tiếp bạn trừ số này với chiều rộng của form được chia cho hai. Bạn có thể xác lập đặc tính X và Y của form con riêng hoặc bạn có thể tạo đối tượng Point và gán nó vào đặc tính Location của form. Để gọi thủ tục này từ menu, nơi bạn đã tạo form Products, bạn phải thêm mã sau ngay sau thuyết minh của form Products mới.

```
Call CenterChildForm(frm)
```

Truy tìm các cửa sổ con

VB.NET sẽ truy tìm tất cả các form con bạn tạo. Nếu bạn muốn xem danh sách tất cả các form con và có thể cho một form

con nhất định tiêu điểm, bạn có thể thực hiện điều này rất dễ dàng.

1. Đưa form chính vào chế độ thiết kế trong VS.NET.
2. Nhấp menu Window.
3. Vào cửa sổ Properties, nhấp đặc tính MdiList và xác lập giá trị này là **True**.
4. Chạy project, mở hai form Products, sau đó thả xuống menu Window. Bạn sẽ thấy từng thể nghiệm của form Product bạn đã mở.

Tạo các menu bật lên

Trong nhiều ứng dụng, bạn có thể ấn nút chuột phải vào các vị trí nào đó và một menu nhạy ngữ cảnh sẽ xuất hiện. Các menu này cho bạn khả năng thực hiện các thao tác dựa trên vị trí bạn đã nhấp.

Để tạo một menu bật lên (pop-up), bạn sẽ thấy control ContextMenu bên trong hộp công cụ và thả nó vào form của mình. Thực ra, control này sẽ xuất hiện trong Tray của form. Sau đó bạn có thể nhấp menu context này và tạo một menu giống như bạn tạo một menu chính. Các menu này sẽ không thể hiện trên form của bạn, trừ khi bạn nhấp control ContextMenu vào thời gian thiết kế. Chúng sẽ không thể hiện vào thời gian chạy cho tới khi bạn gán tên control vào một control nào đó, sau đó nhấp nút chuột phải vào control đó.

1. Mở form Products.
2. Tìm control ContextMenu trên Toolbox.
3. Nhấp đôi control này để thêm nó vào Tray của form.

4. Nhấp control ContextMenu để cho nó tiêu điểm.
5. Đổi đặc tính Name thành **cmnuProdID**.
6. Thêm các mục menu sau vào control này và xác lập các đặc tính tên như được trình bày. Khi bạn thêm một menu context, bạn bắt đầu với mục menu đầu tiên dưới menu cấp trên cùng.

Menu	Tên
&Lookup	mnuPLoopUp
&Copy	mnuPCopy
&Paste	mnuPPaste

7. Nhấp control hộp văn bản Product ID.
8. Đi tới cửa sổ Properties, nhấp ContextMenu và thả xuống danh sách các control của menu context trên form này. Xác lập đặc tính ấy là **cmnuProdID**.
9. Chạy project, mở form sản phẩm và nhấp nút chuột phải vào hộp văn bản Product ID để xem menu context xuất hiện.
10. Bây giờ bạn có thể thêm mã vào để đáp ứng các sự kiện nhấp chuột trên từng menu này.

Thao tác các menu vào thời gian chạy

Về mặt lập trình bạn có thể thêm hoặc xóa các menu vào thời gian chạy. Bằng cách dùng phương thức Add và Remove của tập hợp MenuItems mà bạn có thể tạo các menu mới vào thời gian chạy, sau đó xóa một trong các menu bất kỳ bên trong menu, bất kể chúng được tạo vào thời gian chạy hay vào thời gian thiết kế.

Để thiết lập, thực hiện các bước sau để bạn thêm ba mục menu mới dưới menu File trên form MDI của bạn.

- Mở frmMain.vb ở chế độ thiết kế.
- Nhấp menu File để xem tất cả các mục menu.

Thêm các menu

Để thêm một menu bạn phải thêm mã như ở dưới vào thủ tục sự kiện nhấp chuột của Add Menus.

```
Private Sub mnuFAddMenus_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles
mnuFAddMenus.Click
    ' Adds these menus to the end of the file menu
    mnuFile.MenuItems.Add("New Menu 1")
    mnuFile.MenuItems.Add("New Menu 2")
End Sub
```

Một trong các menu dưới control MainMenu là đối tượng MenuItem. Bạn có thể tham chiếu từng mục này bằng cách dùng tên của đối tượng bạn đã gán vào mục đó vào thời gian thiết kế. Ở thí dụ trên, bạn sẽ thêm hai mục dưới menu &File bằng cách gọi ra phương thức Add trên tập hợp MenuItems của menu mnuFile. Nhớ là mnuFile là tên bạn đã gán vào mục menu &File.

Xóa các menu

Để xóa các mục menu, bạn tham chiếu các mục để xóa bằng tên mục menu hoặc bằng số chỉ mục trong tập hợp MenuItems dưới menu cấp trên cùng. Vì bạn vừa thêm hai mục vào cuối menu mnuFile, bạn có thể xóa hai mục này bằng cách dùng phương thức RemoveAt và và chuyển vào Count của các mục menu trong đối tượng mnuFile trừ đi 1, vì tất cả các tập hợp đều có cơ số bằng 0 trong .NET.

```
Private Sub mnuFRemoveMenus_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles mnuFRemoveMenus.Click

    With mnuFile.MenuItems
        ' Remove the last two items
        .RemoveAt(.Count - 1)
        .RemoveAt(.Count - 1)
    End With

End Sub
```

Tạo và kết hợp các menu vào thời gian chạy

Như đã được thảo luận trước đây ở chương này, bạn có thể dùng MergeOrder và MergeType để xác định nơi các menu được đặt vào khi thêm hai menu cùng với nhau. Ở mã kế tiếp, bạn sẽ tạo một mục MainMenu mới, sau đó tạo MenuItem mới dưới mục MainMenu mới này. Khi bạn tạo một MenuItem mới, bạn sẽ xác lập MergeOrder là 1, vì nó sẽ được đặt ngay sau menu Edit cũng có MergeOrder bằng 1.

Bạn có thể thêm các mục menu con dưới mục menu mới này, bằng cách gọi ra phương thức Add trên tập hợp MenuItems. Cuối cùng, bạn sẽ dùng phương thức MergeMenu trên đối tượng mnuMain để kết hợp hai đối tượng MainMenu lại. Chuyển đối tượng menu mới bạn đã tạo tới phương thức MergeMenu và dựa trên đặc tính MergeOrder nó sẽ đặt menu mới này vào vị trí chính xác trong menu chính.

```
Private Sub mnuFAddNew_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles mnuFAddNew.Click
    Dim mnuNew As MainMenu
    Dim mnuMaint As MenuItem

    ' Create New Menu
    mnuNew = New MainMenu()
```

```
    mnuMaint = mnuNew.MenuItems.Add("Maintenance")
    ' Place it after the Edit menu
    mnuMaint.MergeOrder = 1
    mnuMaint.MergeType = MenuMerge.Add
    ' Add Sub Items
    mnuMaint.MenuItems.Add("Suppliers")
    mnuMaint.MenuItems.Add("Categories")

    ' Merge to the top level menu
    mnuMain.MergeMenu(mnuNew)
End Sub
```

Sử dụng các menu chọn bằng dấu kiểm

Bạn có thể đặt một dấu kiểm bên mỗi mục menu để cho người dùng biết menu này vừa được lựa. Để đặt dấu kiểm bên một menu, chỉ cần xác lập đặc tính Checked là True như được trình bày ở thí dụ sau:

```
mnuFAddMenus.Checked = True
```

Sử dụng các menu chọn bằng nút radio

Thay vì đặt dấu kiểm bên một menu, bạn còn có thể hiển thị nút radio bên một menu trong một nhóm các nút radio loại trừ lẫn nhau (chỉ chọn một). Để có RadioButton như dấu kiểm ở bên một menu, bạn xác lập đặc tính RadioChecked là True. Sau đó khi bạn xác lập đặc tính Checked, một bullet sẽ thể hiện bên mục menu. Tên RadioChecked hơi dễ lầm, vì việc xác lập đặc tính này trên một nhóm các mục menu không làm cho chúng hoạt động giống như các nút radio. Nếu bạn muốn có một mục được thôi chọn khi một mục khác được chọn, bạn phải tự lập trình điều này. Bên dưới là mã mẫu trình bày cách bạn có thể làm điều này trên một nhóm menu.

```
Private Sub mnuFFont_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles mnuFFont.Click
    mnuFFont.Checked = True
    mnuFColor.Checked = False
    mnuFSize.Checked = False
End Sub

Private Sub mnuFColor_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles mnuFColor.Click
    mnuFFont.Checked = False
    mnuFColor.Checked = True
    mnuFSize.Checked = False
End Sub

Private Sub mnuFSize_Click(
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles mnuFSize.Click
    mnuFFont.Checked = False
    mnuFColor.Checked = False
    mnuFSize.Checked = True
End Sub
```

Ngưng nạp một ứng dụng MDI

Để ngưng nạp một ứng dụng MDI, bạn chỉ cần ngưng nạp form MDI. Ở File ➢ menu Exit bạn có thể đặt mã sau.

```
Private Sub mnuFExit_Click()
   Me.Close
End Sub
```

Điều này sẽ khiến cho form MDI ngưng nạp. Khi form cha ngưng nạp, bất kỳ các form con mở cũng phải được ngưng nạp. Vì vậy một dòng mã này sẽ khiến cho mọi thứ được ngưng nạp và ứng dụng của bạn sẽ đóng.

Dừng ngưng nạp

Nếu người dùng bắt đầu soạn thảo thông tin trên form con, thì việc đóng form cha MDI sắp xảy ra. form con có khả năng ngăn chặn form cha ngưng nạp khi nó ở chế độ soạn thảo này. Đương nhiên, điều đó còn tùy thuộc form con báo cho form cha là nó không thể đóng ngay bây giờ.

Để thực hiện điều này, bạn truy tìm "state" của form con. Nếu form con được coi là "dirty", nghĩa là một trong các giá trị control của nó đã thay đổi, thì form sẽ nhắc người dùng phải làm gì với các thay đổi trước khi đóng cửa sổ. Bạn có thể thực hiện điều này với một biến thành viên đơn giản được khai báo trong lớp form Product. Thêm biến thành viên sau, *mboolDirty*, như được trình bày ở mã sau.

```
Public Class frmProducts
    Inherits System.Windows.Forms.Form

    Private mboolDirty As Boolean
```

Thêm mã thích hợp vào sự kiện Closing của form để hỏi người dùng xem họ muốn làm gì khi việc ngưng nạp sắp xảy ra trên form và biến Boolean là True.

```
Private Sub frmProducts_Closing(ByVal sender As Object,
_
 ByVal e As System.ComponentModel.CancelEventArgs) _
 Handles MyBase.Closing
    If mboolDirty Then
        If MessageBox.Show("Close Window", "Closing", _
         MessageBoxButtons.YesNo, _
         MessageBoxIcon.Question) = DialogResult.Yes
Then
            e.Cancel = False
        Else
            e.Cancel = True
        End If
    End If
```

```
End Sub
```

Khi một form sắp ngưng nạp, nó sẽ gọi thủ tục sự kiện Closing của form. VB.NET sẽ chuyển vào tham số *e* như CancelEventArgs. Một trong các đặc tính của đối tượng *e* là Cancel. Nếu bạn xác lập Cancel bằng True, thì form này sẽ không đóng. Nó là True dù có yêu cầu đóng cửa sổ con hay không hoặc yêu cầu đóng cửa sổ cha và cửa sổ cha cố đóng cửa sổ con.

Tóm tắt

Ở chương này bạn đã tìm hiểu cách xây dựng một ứng dụng MDI bằng cách dùng một số kĩ thuật mà bạn thấy trong các ứng dụng Windows chuyên nghiệp. Bạn đã tìm hiểu cách tạo các form nằm ở trong các đường biên của form MDI. Bạn cũng đã tìm hiểu cách tạo các menu. Dù bạn có chọn dùng mô hình MDI hay không, nó vẫn tùy thuộc vào tính phức tạp của ứng dụng của bạn và bao nhiêu form sẽ được hiển thị đồng thời.

Câu hỏi ôn tập

1. Bạn được cho phép có bao nhiêu form cha MDI trong chương trình VB.NET?
2. Bạn tạo một form con như thế nào?
3. Bạn thêm một mục menu mới vào form như thế nào?
4. Phương thức nào cho bạn sắp xếp các cửa sổ con bên trong form cha MDI?
5. Bạn xác định đặc tính nào để ngăn chặn form con ngưng nạp?

Bài tập

- Tạo ứng dụng của chương này.

Trả lời câu hỏi ôn tập

1. Tùy theo bạn muốn.
2. Xác lập đặc tính MdiParent của form con là tham chiếu với form cha.
3. Sử dụng phương thức Add của tập hợp MenuItems của menu.
4. LayoutMDI.
5. Đặc tính Cancel trên tham số CancelEventArgs trong thủ tục sự kiện Closing.

Chương 8

Tạo form hộp thoại

- Tìm hiểu cách tạo các form hộp thoại của riêng bạn.

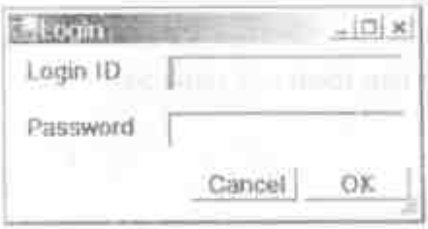
Tạo các form hộp thoại

Trong nhiều ứng dụng, rất thường có nhu cầu cho xuất hiện một form modal đóng vai trò một hộp thoại. Điều này làm cho ngưng thi hành mã và chờ người dùng loại bỏ form, truy tìm các giá trị từ form hộp thoại và sau đó đóng form hộp thoại. Điều này có thể xảy ra trong các phiên bản trước đây của Visual Basic, nhưng kĩ thuật chính xác chưa cụ thể. Trong Visual Studio.NET, có một cách chính xác để tạo các form hộp thoại.

Trong ứng dụng MDI của bạn đang xây dựng ở giáo trình này, hầu như bạn muốn người dùng đăng nhập vào ứng dụng. Thông thường bạn sẽ cần bật lên một form hộp thoại modal để yêu cầu họ cho user name và password. Nếu người dùng nhấp nút Cancel, bạn muốn thoát khỏi ứng dụng. Nếu người dùng nhấp nút OK và

user name và password khớp với những gì trong CSDL, thì bạn sẽ cho phép họ tiếp tục vào ứng dụng.

Ở chương này, bạn sẽ tạo một form login (đăng nhập), như được trình bày ở hình 1. Bạn sẽ gọi form này bằng thủ tục sự kiện Load trên form chính. Sau khi điền vào Login ID và password, bạn sẽ nhấp nút OK. Sau đó nó sẽ đóng form login và cho lại kết quả OK trở lại form chính. Nó báo cho form chính là nó cần truy tìm dữ liệu từ form login và kiểm tra dữ liệu đó dựa vào nguồn dữ liệu nào đó hữu hiệu hóa để người dùng truy xuất được ứng dụng này. Nếu bạn nhấp nút Cancel, thì kết quả Cancel được cho trở lại form chính. Điều này sẽ cho form chính biết tự ngưng nạp và kết thúc ứng dụng.



Hình 1: Form login là chuẩn trong hầu hết các ứng dụng.

Các bước để tạo Form

Các bước sau để tạo một project mới và hai form của nó.

1. Mở project bạn đang xây dựng trong giáo trình này.
2. Lựa Project | Add Windows Form từ menu.
3. Đặt tên form mới này là frmLogin.vb.
4. Tạo form như được trình bày ở hình 1 bằng cách dùng các thông tin ở bảng 1.

Control	Đặc tính	Giá trị
Label	Name	lblLoginID
	Text	Login ID
Label	Name	lblPassword
	Text	Password
TextBox	Name	txtLoginID
	TabIndex	0
TextBox	Name	TxtPassword
	PasswordChar	*
	TabIndex	0
Button	Name	BtnOK
	Text	OK
	TabIndex	1
	DialogResult	OK
Button	Name	BtnCancel
	Text	Cancel
	TabIndex	2
	DialogResult	Cancel

Bảng 1: *Xác lập các đặc tính của các control trên frmLogin bằng cách dùng hướng dẫn này.*

Trên chính form frmLogin, bạn sẽ xác lập các đặc tính được trình bày ở bảng 2.

Đặc tính	Giá trị
BorderStyle	FixedDialog
ControlBox	False
MaximizeBox	False
MinimizeBox	False
Text	Login

Bảng 2: *Xác lập các đặc tính của frmPassword bằng cách dùng hướng dẫn này.*

Hầu hết các form đều có đường viền khá dày và các form này không thay đổi kích thước được. Đó là lý do xác lập BorderStyle là FixedDialog. Điều này sẽ tạo kiểu đường viền đó.

Tự động đóng hộp thoại

Trên form frmLogin bạn sẽ muốn có thể ấn phím **ENTER** và form hoạt động như là bạn đã nhấp nút OK và ngưng nạp form. Ngoài ra, bạn còn cần có thể ấn phím **ESCAPE** và form hoạt động như là bạn đã nhấp nút Cancel và ngưng nạp form. Để thêm ứng xử này, xác lập hai đặc tính form như được trình bày ở danh sách sau:

Đặc tính	Giá trị
AcceptButton	btnOK
CancelButton	btnCancel

Khi hai giá trị này được xác lập, nếu bạn ấn phím **ENTER** hoặc phím **ESCAPE**, form sẽ ngưng nạp. Bạn không cần viết bất kỳ mã nào để điều này xảy ra, chỉ bằng cách xác lập hai đặc tính này, nó sẽ thực hiện điều này. Điều này tốt hơn nhiều so với trong VB6 mà bạn phải viết mã dưới mỗi nút này để ngưng nạp form.

Trả lại kết quả từ form hộp thoại

Bạn cần có cách nào đó để chỉ định thủ tục gọi nút mà bạn đã lựa trên form logic khi bạn đã thoát khỏi form logic. Visual Studio cung cấp đặc tính DialogResult trong control nút để cho phép bạn xác lập hộp thoại chuẩn trả lại giá trị. Trong trường

hợp này, xác lập đặc tính **DialogResult** của **btnOK** là **OK**. Xác lập đặc tính **DialogResult** của **btnCancel** là **Cancel**.

Mẹo:

Việc xác lập đặc tính DialogResult của một nút hoạt động có sức lôi cuốn bạn — khi bạn mở form, sử dụng phương thức ShowDialog của nó, nhấp nút có đặc tính DialogResult đã được xác lập để đặt giá trị đặc tính DialogResult của nó vào đặc tính DialogResult của form và sau đó đóng form cho bạn. Nếu bạn định chuyển giá trị của đặc tính DialogResult trở lại đối tượng gọi, bạn không cần viết mã để đóng form — form đảm trách điều này cho bạn.

Tạo Sub Main

Trước khi bạn gọi form Login, đầu tiên bạn sẽ tạo thủ tục Main sẽ là điểm khởi đầu cho toàn bộ ứng dụng của bạn. Để khởi đầu với Sub Main như đối tượng khởi động của bạn thay vì một form luôn luôn là ý tưởng hay. Điều này cho bạn tính mềm dẻo để thực hiện bất kỳ hoạt động nào đàng sau các nền khi ứng dụng của bạn nạp. Thí dụ, bạn muốn hiển thị một màn hình lóe lên khi bạn nạp một số biến khởi tạo từ một file cấu hình hoặc từ file Registry. Bên dưới là một thí dụ về Sub Main sẽ nạp form cha MDI chính của ứng dụng, sau đó hiển thị form Login bạn đã tạo ở chương này.

```
Module modMain
    Sub Main()
        Dim frmMain As frmMain
        Dim frmLogin As frmLogin

        ' Display the Main form first
        frmMain = New frmMain()

        frmMain.Show()
```

```
        frmMain.Refresh()

        ' Display the Login form
        frmLogin = New frmLogin()

        If frmLogin.ShowDialog() = DialogResult.OK Then
            ' Check user name and password
            MessageBox.Show("Valid    Login    ID    and
Password")
        Else
            frmMain.Close()
        End If
    End Sub
End Module
```

Ở mã trên bạn khai báo hai biến form, một cho form chính và một cho form logic. Sau khi tạo một thể nghiệm mới của form chính, bạn sẽ hiển thị nó, sau đó làm tươi nó để chắc chắn nó được hiển thị hoàn toàn trên màn hình trước khi tạo form login. Phương thức ShowDialog sẽ hiển thị form login nắm giữ tiêu điểm. Tùy theo nút người dùng ấn hoặc phím (**ENTER** hoặc **ESCAPE**) họ ấn, phương thức ShowDialog sẽ trả lại giá trị theo đặc tính DialogResult của control tương ứng. Sau đó bạn có thể so sánh giá trị đó với DialogResult.OK và nếu chúng khớp với nhau thì bạn biết người dùng đã ấn nút OK. Nếu giá trị trả lại không bằng DialogResult.OK thì bạn biết người dùng đã ấn nút Cancel.

Khi bạn gọi phương thức ShowDialog của form, frmLogin, bạn đang chỉ định form là bạn muốn xử lý nó như một hộp thoại. Vì bạn xác lập đặc tính DialogResult cho hai nút trên frmLogin, bạn có thể truy tìm đặc tính DialogResult của form, sau khi đóng nó. (Và vì việc xác lập đặc tính DialogResult của nút khiến form tự đóng khi bạn lựa nút ấy, bạn không cần cung cấp bất kỳ mã nào để đóng form). Bạn có thể dùng đặc tính DialogResult cho các giá trị khác hơn OK và Cancel. Các giá trị sau có thể xảy ra đối với đặc tính DialogResult:

- None

- OK
- Cancel
- Abort
- Retry
- Ignore
- Yes
- No

Với tất cả các tổ hợp này của các kết quả hộp thoại, bạn có thể tạo một kiểu hộp thoại bất kỳ bạn muốn. Đương nhiên, nếu bạn cần một hộp thoại đặc biệt với một loại kết quả riêng, bạn sẽ phải tạo đặc tính riêng trên form hộp thoại, sau đó xác lập giá trị đó dựa vào nút được ấn trên form. Điều này tương tự như cách bạn đã dùng để tạo các hộp thoại trong VB6.

Tóm tắt

Ở chương này bạn đã tìm hiểu cách tạo các form hộp thoại bằng cách dùng đặc tính DialogResult của control nút. Bạn đã tìm hiểu cách gọi một form nắm giữ tiêu điểm bằng cách dùng ShowDialog và cách nó trả lại giá trị từ đặc tính DialogResult của control nút được dùng để đóng form modal.

Câu hỏi ôn tập

1. Bạn dùng tên đặc tính nào để xác lập kết quả trả lại từ form hộp thoại?
2. Bạn sẽ xác lập giá trị BorderStyle nào cho form hộp thoại?

3. Bạn xác lập đặc tính nào trên form để có phím **ENTER** đóng form ấy?
4. Bạn xác lập đặc tính nào trên form để có phím **ESCAPE** đóng form ấy?

Bài tập

- Tạo form Login và Sub Main như được mô tả ở chương này.

Trả lời câu hỏi ôn tập

1. DialogResult.
2. FixedDialog.
3. AcceptButton.
4. CancelButton.

Chương 9

Sử dụng trình gỡ rối

- Tìm hiểu ba chế độ của Visual Basic.
- Tìm hiểu cách gọi trình gỡ rối.
- Tìm hiểu cách xem và xác lập các điểm ngắt.
- Tìm hiểu cách điều khiển và xem đường dẫn thi hành của mã nguồn trong trình gỡ rối.
- Tìm hiểu cách sử dụng cửa sổ Immediate.

Gỡ rối

Chương này khảo sát các khái niệm cơ bản về trình gỡ rối (debugger) mã nguồn được xây dựng vào môi trường Visual Basic. Bạn sẽ tìm hiểu về các chế độ khác nhau của Visual Basic, cách bước xuyên qua (step through) mã và cách xác lập các điểm ngắt (breakpoint). Ngoài ra, bạn còn tìm hiểu về các cửa sổ khác nhau

trong môi trường Visual Basic cho bạn nhiều thông tin về chương trình bạn sẽ chạy.

Ba chế độ của Visual Basic

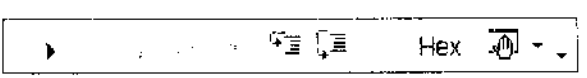
Có ba chế độ khi bạn làm việc trong Visual Basic; thiết kế, chạy và ngắt. Mỗi chế độ có một hàm riêng và biết được bạn đang ở chế độ nào rất quan trọng.

Chúng ta hãy xét ba chế độ khác nhau bạn sẽ làm việc. Đầu tiên, là chế độ Design (thiết kế), nơi bạn xây dựng giao diện người dùng và viết mã nguồn. Kế tiếp là chế độ Run (chạy), nơi bạn sẽ chạy ứng dụng của mình. Cuối cùng là chế độ Break (ngắt), nơi bạn có thể hiệu chỉnh mã, kiểm tra các biến, xác lập lại các giá trị và bước xuyên qua mã nguồn từng dòng một.

Bạn có thể biết bạn đang ở chế độ nào bằng cách xem thanh tiêu đề Visual Basic. Sau tên của project, bạn sẽ thấy Microsoft Visual Basic tiếp theo là [design], [run] hoặc [break]. Thường xem thanh tiêu đề về thông tin chế độ hiện hành của bạn. Còn có một số thứ bạn có thể hoặc không thể thực hiện trong mỗi chế độ.

Thanh công cụ Debug

Nếu bạn nhấp nút chuột phải vào vùng thanh công cụ của Visual Basic IDE, bạn có thể lựa thanh công cụ Debug. Khi được lựa, bạn sẽ thấy thanh công cụ xuất hiện như ở hình 1.



Hình 1: *Thanh công cụ gỡ rối cho bạn hầu hết các tính năng của menu Debug.*

Từ thanh công cụ này, bạn sẽ có thể chạy, dừng và tạm ngưng ứng dụng. Bạn cũng sẽ có thể thực hiện các chức năng khác liên quan đến việc gỡ rối ứng dụng của bạn. Từ bên trái, các công cụ như sau:

- Run the project
- Pause the project
- Stop the project
- Restart
- Show Next Statement
- Quy trình Step into
- Quy trình Step over
- Quy trình Step out
- Chốt chọn hiển thị thập lục phân – thập phân về tất cả các giá trị trong cửa sổ Quick Watch
- Chốt chọn giữa cửa sổ Breakpoints và Immediate

Bạn sẽ tìm hiểu thêm về các công cụ này và các công cụ khác ở chương này.

Gọi ra trình gỡ rối

Để bắt đầu ứng dụng của bạn bằng cách dùng trình gỡ rối. Lựa Debug|Step Into hoặc các tùy chọn menu Debug|Step Over.

Các phím nóng của mỗi tùy chọn này có thể khác nhau tùy theo cách bạn đã tùy biến profile của bạn khi lần đầu bạn chạy Visual Studio. Thí dụ, nếu bạn thiết lập như Visual Studio Developer, các phím nóng cho hai lệnh này sẽ là **F11** và **F10** tương ứng. Nếu bạn thiết lập như Visual Basic bạn sẽ dùng **F8** và **SHIFT-F8** tương ứng.

Bước vào mã

Tùy chọn Step Into trình bày từng dòng mã nguồn khi nó chạy. Tùy chọn này rất tốt khi bạn cần kiểm tra từng dòng mã khi nó thi hành và có thể kiểm tra các biến khác nhau hoặc thậm chí thay đổi các dòng mã. Bạn có thể ấn giữ F11 để bước xuyên qua từng dòng trong ứng dụng của bạn. Khi một form được hiển thị trên màn hình, bạn cần nhấp vào thứ gì đó hoặc thực hiện một thao tác khác để làm cho chương trình của bạn đáp ứng trước khi bạn được đưa trở lại chế độ bước.

Bước qua mã

Tùy chọn Step Over cho phép bạn chạy toàn bộ quy trình mà không bước vào quy trình đó. Điều này hữu dụng khi bạn biết một hàm hoạt động và không muốn mất thời gian bước qua quy trình đó. Khi bạn sử dụng Step Over (bước qua), quy trình sẽ thi hành và bạn sẽ trở về chế độ Break với con trỏ của bạn ở dòng ngay sau lời gọi quy trình đó.

Khi ứng dụng của bạn đang chạy bạn có thể gọi ra trình gỡ rối và vào chế độ Break bằng một số cách.

- Bằng cách ấn <Ctrl-Break> vào thời gian chạy.
- Bằng cách đặt câu lệnh Stop vào mã của bạn.

- Bằng cách xác lập điểm ngắt.
- Ấn **F11/F8** để chạy chương trình.

Các công cụ để gỡ rối

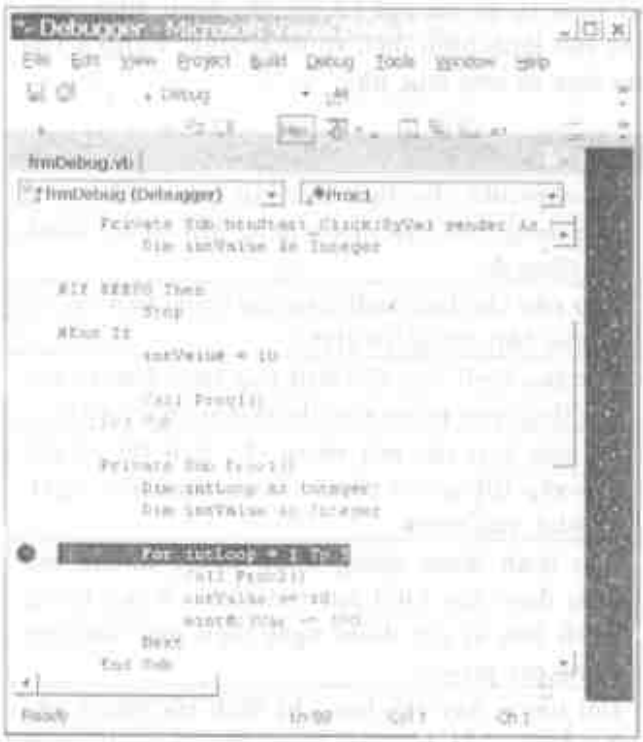
Bạn có thể dùng nhiều công cụ tùy chọn khi cần gỡ rối ứng dụng. Ngoài ra, các công cụ gỡ rối còn có các câu lệnh, như Stop, Trace, Debug và các câu lệnh biên dịch có điều kiện. Bảng 1 cho bạn định nghĩa tóm lược về mỗi mục này.

Công cụ	Mô tả
Breakpoints	Breakpoints cho bạn xác lập dòng mã trong ứng dụng, nơi bạn muốn dừng khi sự thi hành tới dòng đó.
Debug Class	Lớp này cho bạn xuất liệu các chuỗi vào cửa sổ output bên trong VS IDE.
Conditional Statements	Các câu lệnh này cho bạn gộp hoặc loại ra các câu lệnh bên trong ứng dụng được biên dịch.
Various Windows	Có một loạt các cửa sổ sẽ cho bạn tất cả các sắp xếp thông tin về các biến, các điểm ngắt, bộ nhớ, call stack ...
Stop Statement	Câu lệnh dừng giống điểm ngắt, nhưng mã phải được xóa khỏi mã của bạn. Chúng tương thích hơn vì các điểm ngắt hiện nay vẫn còn trong các project.
Trace Class	Đối tượng này cho bạn chỉ định các chuỗi nào đó được xuất liệu vào một file, console, một bản ghi sự kiện hoặc vị trí khác. Nó sẽ giúp bạn giám sát nơi và khi các hoạt động xảy ra trong ứng dụng.

Bảng 1: *Các công cụ bạn có thể dùng để gỡ rối ứng dụng.*

Các phần kế tiếp trong chương này sẽ giới thiệu với bạn các công cụ khác nhau mà bạn có thể dùng để gỡ rối các ứng dụng của bạn.

BreakPoints



***Hình 1:** Xác lập điểm ngắt có thể thực hiện bằng cách nhấp vào một dòng và ấn F9.*

Thông thường khi bạn đang kiểm tra một ứng dụng, bạn không muốn phải bước xuyên qua từng dòng mã trong chương trình để tới một nơi bạn nghĩ là bị lỗi. Vì vậy, thay vào đó bạn

có thể xác lập điểm ngắt trong mã nguồn để báo Visual Basic vị trí đứng thi hành khi nó tới dòng mã này.

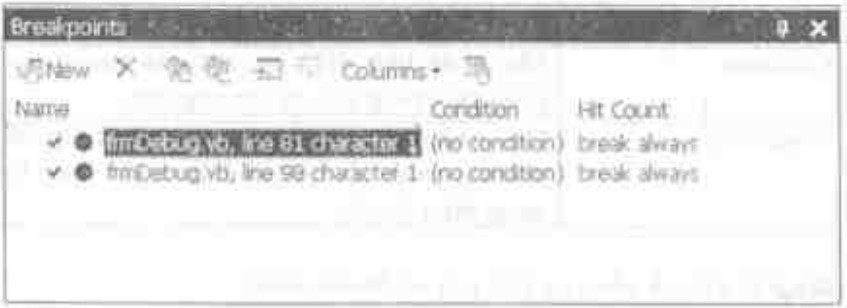
Bạn cũng có thể xác lập điểm ngắt bằng cách lựa Toggle Breakpoint từ menu Debug, nhấp biểu tượng "bàn tay" trên thanh công cụ debug.

Sau khi điểm ngắt được xác lập, bạn có thể chạy ứng dụng bình thường. Khi Visual Basic đụng dòng đó, nó tự động đưa bạn vào chế độ Break. Ở điểm này bạn có thể thay đổi mã, kiểm tra các biến hoặc cần làm gì đó để tạo cho ứng dụng của bạn chạy chính xác.

Các điểm ngắt được lưu với project của bạn, vì vậy sau khi đóng IDE và khởi động lại nó, các điểm ngắt này vẫn còn ở các vị trí của chúng.

Cửa sổ BreakPoints

Nếu bạn lựa Debug|Windows|Breakpoints từ menu IDE, bạn sẽ thấy một cửa sổ trông giống hình 2.



Hình 2: *Cửa sổ Breakpoints trình bày tất cả các điểm ngắt trong ứng dụng của bạn.*

Cửa sổ này sẽ cho bạn một danh sách tất cả các điểm ngắt bạn đã xác lập trong ứng dụng. Nếu bạn nhấp đôi một điểm ngắt, nó sẽ đưa bạn tới ngay vị trí đó trong mã nguồn của bạn.

Thanh công cụ Breakpoints

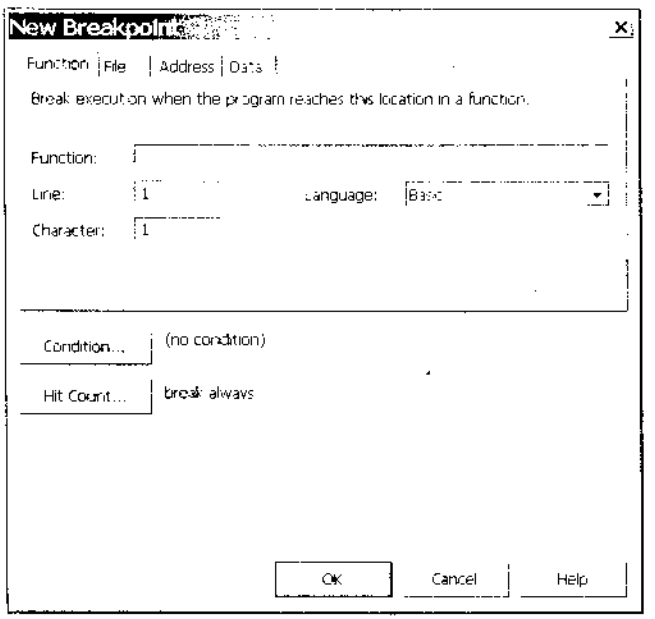
Có một thanh công cụ trên cửa sổ này, nó sẽ cho bạn khả năng làm việc với các điểm ngắt.

Công cụ	Mô tả
New	Tạo một điểm ngắt mới và cho phép bạn cấu hình các điều kiện chương trình sẽ dừng trên điểm ngắt này.
Delete	Xóa điểm ngắt được tô sáng hiện hành.
Clear All Breakpoints	Xóa tất cả các điểm ngắt khỏi project hiện hành.
Disable All Breakpoints	Vô hiệu hóa tất cả các điểm ngắt cho tới khi bạn hữu hiệu hóa chúng trở lại.
Go To Source Code	Di chuyển con trỏ tới một dòng trong mã nguồn của bạn, nơi điểm ngắt được tô sáng hiện hành định vị.
Go To Disassembly	Mở cửa sổ Disassembly ở dòng, nơi điểm ngắt được tô sáng hiện hành định vị.
Columns	Cho phép bạn chọn các cột nào bạn muốn hiển thị trong cửa sổ Breakpoint.
Properties	Cho phép bạn cấu hình các điều kiện mà chương trình sẽ dừng ở điểm ngắt được tô sáng hiện hành.

Bảng 2: *Thanh công cụ của cửa sổ Breakpoints.*

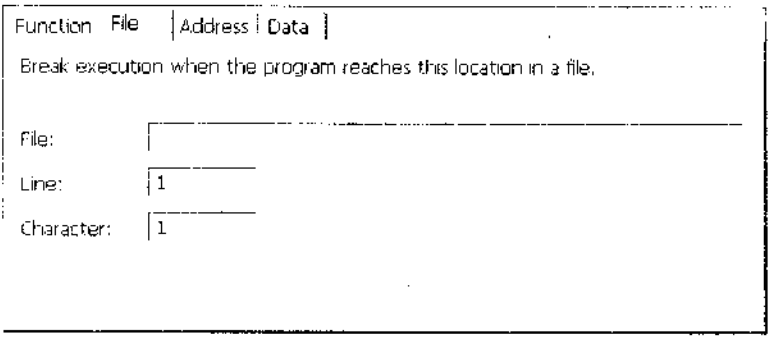
Các điều kiện trên các điểm ngắt

Nếu bạn nhắp nút chuột phải một điểm ngắt bất kỳ trong cửa sổ Breakpoints, bạn có thể xác lập một số thuộc tính trên điểm ngắt mới hoặc đọc một số thuộc tính về điểm ngắt hiện hành.



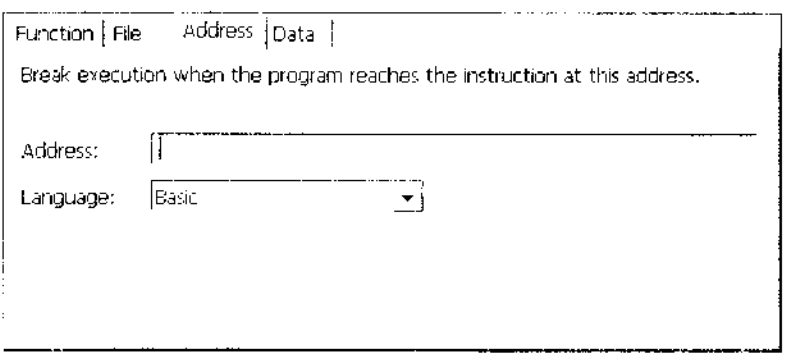
Hình 3: *Hộp thoại New Breakpoint trình bày các thông tin về điểm ngắt hoặc cho phép bạn tạo một điểm ngắt mới.*

Trên tab Function, bạn được phép xác lập hàm nào bạn muốn tạo một điểm ngắt. Bạn có thể chỉ định số dòng và ngay cả số kí tự, nơi bạn muốn có điểm ngắt. Lý do cần số kí tự trong trường hợp bạn có nhiều câu lệnh trên một dòng.



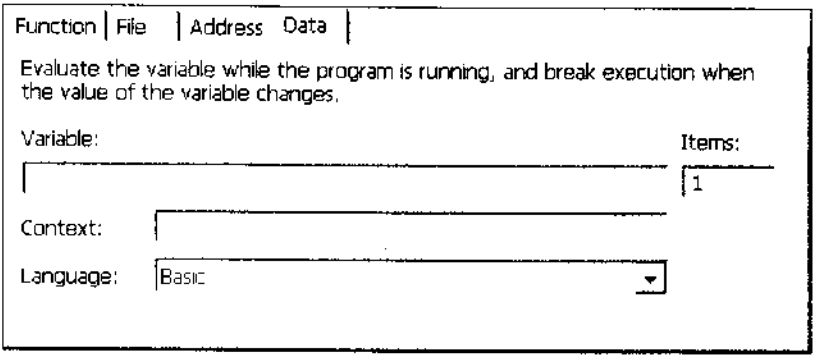
Hình 4: *Tab File của hộp thoại Breakpoint cho bạn xác lập file nào và số dòng bạn muốn đặt điểm ngắt.*

Tab File cho bạn đặt điểm ngắt bên trong một file nhất định trong ứng dụng của bạn theo số dòng và số kí tự.



Hình 5: *Tab Address của hộp thoại Breakpoint cho bạn xác lập địa chỉ chỉ lệnh của vị trí ngắt.*

Tab Address của hộp thoại Breakpoint cho bạn xác lập địa chỉ thực của chỉ lệnh nơi bạn muốn đặt điểm ngắt.

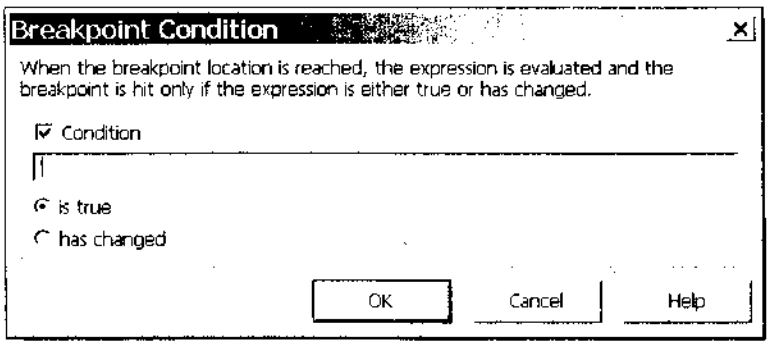


Hình 6: *Tab Data của hộp thoại Breakpoint cho bạn ngắt khi một biến đã chỉ định thay đổi giá trị của nó.*

Tab Data của hộp thoại Breakpoint cho bạn vào chế độ ngắt khi một tên biến đã chỉ định thay đổi giá trị của nó. Điều này có thể rất hữu dụng khi truy tìm trong chương trình của bạn một biến toàn cục thay đổi.

Xác lập điều kiện

Khi bạn nhấp nút Condition trên hộp thoại New Breakpoint, bạn sẽ thấy một form giống hình 7.

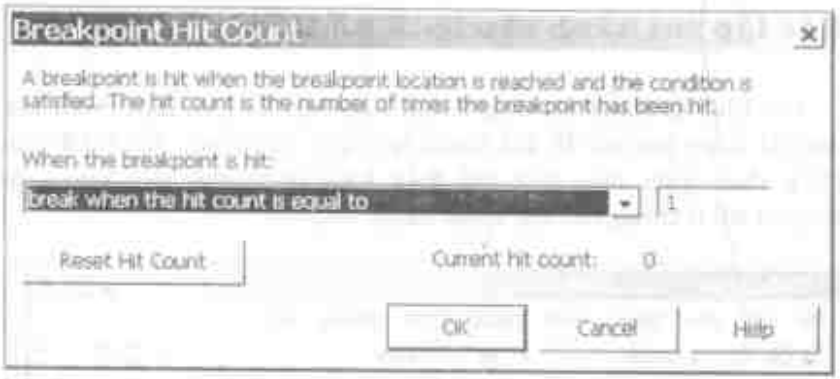


Hình 7: *Hộp thoại Breakpoint Condition cho phép bạn chỉ định một điều kiện, chẳng hạn như intNumber = 5 hoặc boolValue = True.*

Trên hộp thoại Breakpoint Condition, bạn có thể nhập các biểu thức và buộc chương trình vào chế độ ngắt khi biểu thức đó là True hoặc khi biểu thức thay đổi. Một biểu thức là nơi bạn kiểm tra điều kiện True có thể là điều gì đó, chẳng hạn như intNumber = 10 hoặc boolFlag = True. Hoặc có thể bạn chỉ nhập tên biến và lựa "has changed". Bất kỳ khi nào giá trị của biến đó thay đổi, bạn sẽ được đưa vào chế độ Break.

Chỉ định Hit Count

Đôi khi bạn không muốn phải bước xuyên qua một vòng lặp 50 lần cốt để tìm lỗi vào lần thứ 50. Trong trường hợp này, bạn có thể nhấp nút "Hit Count..." trên form hộp thoại New Breakpoint. Khi bạn nhấp nút này, bạn sẽ thấy một form như được trình bày ở hình 8.



Hình 8: Hộp thoại Breakpoint Hit Count cho bạn vào chế độ ngắt chỉ sau khi số lần chỉ định mà điểm ngắt sẽ gặp.

Bạn có bốn tùy chọn có thể lựa từ danh sách thả xuống trên hộp thoại này. Bảng sau trình bày từng tùy chọn này.

Tùy chọn	Mô tả
Break Always	Nó giống một điểm ngắt bình thường, nó sẽ dừng việc thi hành mỗi lần tới điểm ngắt.
break when the hit count is equal to	Khi bạn lựa tùy chọn này, bạn sẽ được yêu cầu nhập một số. Khi số lần bạn đụng điểm ngắt này bằng số chương trình sẽ vào chế độ ngắt.
break when the hit count is a multiple of	Khi bạn lựa tùy chọn này, bạn sẽ được yêu cầu nhập một số. Khi số lần bạn đụng điểm ngắt chia cho nó bằng 0, thì chương trình sẽ vào chế độ ngắt. Nó giống hàm MOD.
break when the hit count is greater than or equal to	Khi bạn lựa tùy chọn này, bạn sẽ được yêu cầu nhập một số. Khi số lần bạn đụng điểm ngắt này lớn hơn hoặc bằng số mà chương trình sẽ vào chế độ ngắt.

Bảng 3: *Các tùy chọn xác lập trên hộp thoại Hit Count thả xuống.*

Xác lập thi hành câu lệnh kế tiếp

Đôi khi thi hành chương trình, bạn cần bỏ qua một đoạn mã nào đó hoặc trở lại để thi hành lại cùng đoạn mã. VB.NET cho phép thực hiện điều này với điều kiện dòng mã bạn muốn di chuyển tới ở trong thủ tục hiện hành.

Hình 9: *Mũi tên ở bên tay trái của IDE cho phép bạn di chuyển tới một dòng mã khác để thi hành.*

Các cửa sổ hữu dụng cho việc gỡ rối

Có nhiều cửa sổ bạn có thể dùng khi ở chế độ Break của ứng dụng.

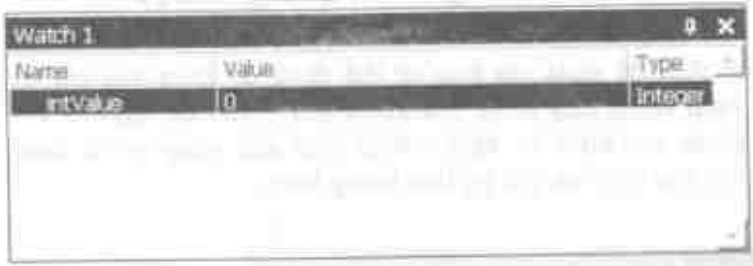
Kiểm tra các biến bằng cách dùng chuột

Phương thức dễ nhất mà bạn có thể dùng để kiểm tra giá trị của một biến trong ứng dụng của mình khi ở chế độ ngắt là chỉ cần đưa chuột vào biến đó. Một tooltip (chú giải công cụ) sẽ được hiển thị với tên biến và giá trị bên trong biến.

Hình 10: Đưa con trỏ vào biến trong chế độ ngắt để xem giá trị của biến đó.

Cửa sổ Watch

Nếu bạn hiển thị cửa sổ Watch bằng cách lựa Debug|Windows|Watch, bạn sẽ thấy một form giống hình 11. Bạn có thể thêm các biến mới bằng cách gõ vào trong cột Name. Bạn cũng có thể sửa đổi các giá trị bằng cách gõ vào bên trong cột Value.



Hình 11: *Cửa sổ Watch sẽ hiển thị bất kỳ các biến nào.*

Quick Watch

Nếu bạn ở chế độ Break và muốn xem ngay nội dung của một biến trong thủ tục hiện hành, đưa con trỏ vào biến đó và lựa Debug|QuickWatch từ menu VS.NET. Bạn cũng có thể dùng các phím Ctrl-Alt-Q hoặc Shift-F9 tùy thuộc xác lập profile VS.NET của bạn. Sau đó bạn sẽ thấy một hộp thoại được hiển thị giống hình 12.

Hình 12: *Quick watch truy tìm giá trị của một biến rất tốt.*

Giá trị của *intTemp* là 0. Cửa sổ Quick Watch để xem nội dung các biến rất nhanh.

Cửa sổ Call Stack

Khi ở chế độ Break, Visual Basic cho phép bạn hiển thị danh sách các thủ tục mà bạn đã thi hành để có vị trí hiện hành. Điều này rất thuận tiện khi bạn không chắc chắn cách tới vị trí hiện hành của mình. Bạn có thể hiển thị cửa sổ Call Stack bằng cách lựa Debug | Windows | Call Stack.

Hình 13: *Cửa sổ Call Stack cho bạn xem vị trí phát xuất của bạn khi bạn đi qua một ứng dụng.*

Sử dụng cửa sổ Command

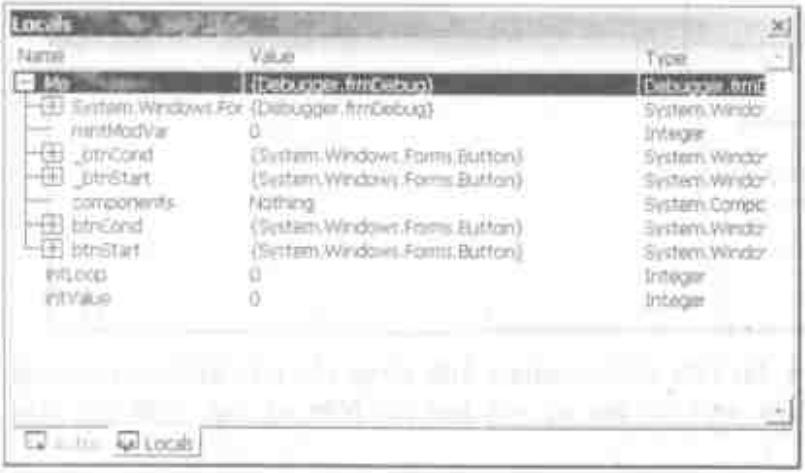
Cửa sổ Command hoặc Immediate cho phép bạn thực hiện các câu lệnh đơn giản, chẳng hạn như thuộc số học, hiển thị giá trị của biến hoặc xác lập giá trị biến. Các câu lệnh này có thể là một biểu thức Visual Basic hợp lệ bất kỳ. Thí dụ, bạn có thể in các kết quả của một phép toán, in nội dung của biến hoặc đặc tính hay thậm chí xác lập giá trị biến hoặc đặc tính. Bên dưới là một cửa sổ Immediate mẫu.

Hình 14: *Cửa sổ Immediate hữu dụng cho việc kiểm tra các giá trị biến, gán các giá trị mới vào các biến và thực hiện các biểu thức.*

Bạn có thể sử dụng dấu chấm hỏi được theo sau bằng biểu thức để hiển thị kết quả. Không giống các phiên bản VB trước đây, bạn không thể gọi bất kỳ hàm nào hoặc thủ tục nào từ cửa sổ Command này.

Cửa sổ Locals

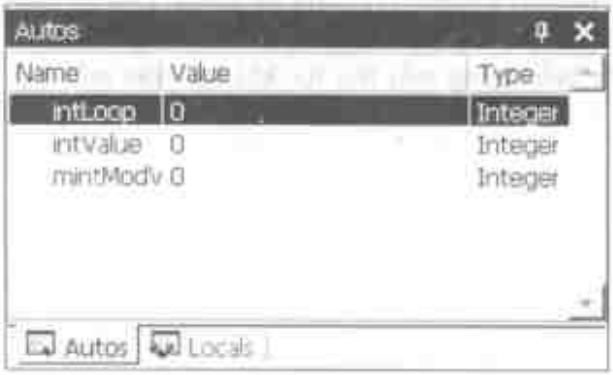
Nếu bạn muốn theo dõi tất cả các biến cục bộ trong một thủ tục, bạn có thể lựa Debug | Windows | Locals từ menu. Cửa sổ này sẽ xuất hiện được chốt giữ ở phần dưới của môi trường thiết kế và sẽ trình bày từng biến trong mỗi thủ tục khi bạn vào mỗi thủ tục.



***Hình 15:** Cửa sổ Locals trình bày bất kỳ các biến cục bộ nào và bất kỳ các control form nào của cửa sổ.*

Cửa sổ Autos

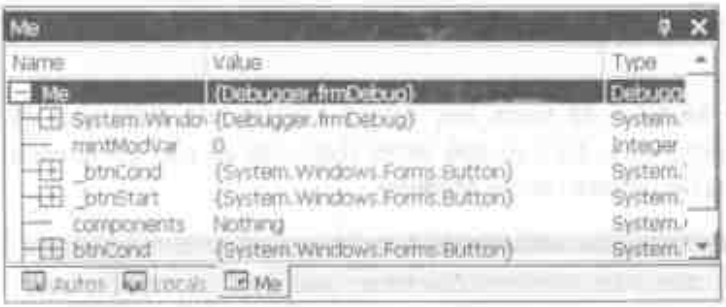
Cửa sổ Autos được hiển thị bằng cách dùng Debug|Windows|menu Autos, hiển thị tất cả các biến cục bộ bạn đã khai báo trong thủ tục.



Hình 16: *Autos trình bày tất cả các biến thành viên và cục bộ tự động.*

Cửa sổ Me

Cửa sổ Me được hiển thị bằng cách dùng Debug|Windows|menu Me, sẽ hiển thị tất cả các đối tượng bên trong form hoặc lớp hiện hành.



Hình 17: *Cửa sổ Me trình bày tất cả các đối tượng bên trong lớp hoặc form hiện hành.*

Cửa sổ Threads

Cửa sổ Threads, được hiển thị bằng cách dùng Debug|Windows| menu Threads sẽ hiển thị danh sách tất cả các mạch khả thi trong chương trình của bạn.

Hình 18: Cửa sổ Threads sẽ hiển thị tất cả các mạch khả thi.

Cửa sổ Modules

Cửa sổ Modules sẽ trình bày tất cả các module được nạp để chạy ứng dụng này. Bạn có thể nhận được cửa sổ này bằng cách dùng Debug | Windows | menu Modules.

Hình 19: Cửa sổ Modules trình bày tất cả các module được nạp.

Cửa sổ Disassembly

Nếu bạn muốn có hợp ngữ mức rất thấp, hãy xét cửa sổ Disassembly bằng cách lựa Debug | Windows | Disassembly từ menu.

Hình 20: *Cửa sổ Disassembly trình bày hợp ngữ cần thiết để chạy ứng dụng.*

Cửa sổ Registers

Cửa sổ Registers sẽ trình bày trong các register (thanh ghi) của máy sau mỗi chỉ lệnh quy định trong ứng dụng .NET của bạn.

Hình 21: *Cửa sổ Registers trình bày mã hex trong mỗi thanh ghi của máy tính.*

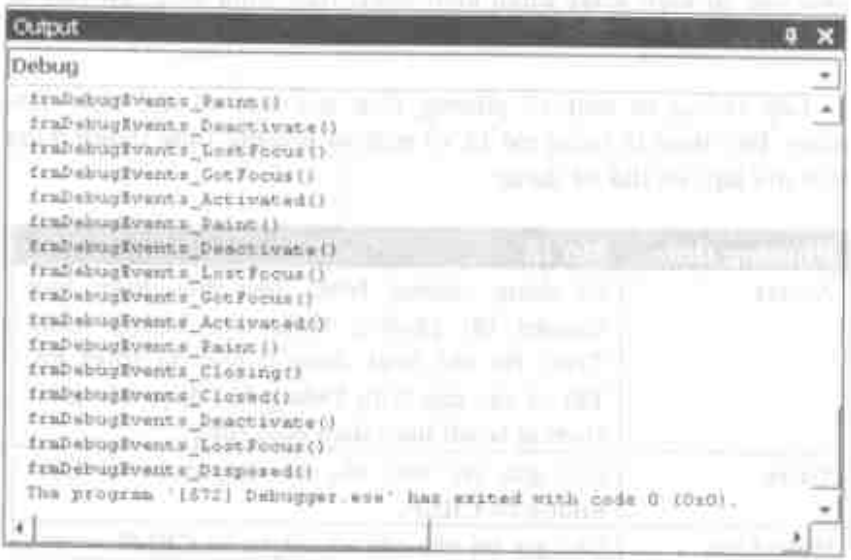
Sử dụng lớp Debug

Thông thường trong khi kiểm tra ứng dụng, bạn có thể thấy là khi bạn đang bước xuyên qua mã, nó có thể làm việc tốt, nhưng khi bạn chạy chương trình trực tiếp, nó không hoạt động. Lý do xảy ra điều này là có thể bạn thay đổi cách chương trình chạy bằng cách dùng trình gỡ rối. Mỗi lần bạn chuyển từ chương trình sang trình gỡ rối, bạn vô tình kích hoạt các sự kiện gây trở ngại cho chương trình của bạn. Vì các sự kiện khác nhau kích hoạt trong tiến trình này, nó thay đổi cách chương trình chạy bình thường. Vì vậy, thay vì sử dụng trình gỡ rối để kiểm tra các giá trị, bạn có thể thấy là ghi các giá trị đó ra cửa sổ Immediate sẽ tiện hơn.

Bạn có thể in trực tiếp ra cửa sổ Command hoặc Immediate từ bên trong ứng dụng bằng cách tiến hành phương thức Debug.Write hoặc Debug.WriteLine. Các giá trị được đưa vào cửa sổ Immediate bằng cách dùng một trong hai phương thức này sẽ vẫn dùng được ở chế độ thiết kế. Điều này có thể hữu dụng khi xem các giá trị trung gian đó mà không phải vào chế độ Break và thay đổi sự thi hành của chương trình. Bên dưới là một thí dụ về sử dụng phương thức Debug.WriteLine.

```
Debug.WriteLine("strMyString = " & strMyString)
Debug.WriteLine("Now executing btnStart_Click()")
```

Để minh họa các điểm này, bạn có thể nhấp vào nút ở cửa sổ mẫu ở chương này, "Debug Class". Nút này sẽ nạp một form được gọi là frmDebugEvents. Form này có một số lời gọi phương thức Debug.WriteLine trong nhiều thủ tục sự kiện khác nhau trên chính form ấy. Điều này minh họa việc dùng phương thức Debug.WriteLine và cho một khái niệm về các sự kiện được kích hoạt khi một form nạp, kích hoạt, ngưng kích hoạt và ngưng nạp. Hình 21 trình bày một số xuất liệu mẫu từ các lời gọi phương thức Debug.WriteLine này.



Hình 22: *Xuất liệu mẫu của form frmDebugEvents.*

Mã bên dưới là một mẫu về mã của form frmDebugEvents.

```
Private Sub frmDebugEvents_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load
```

```
    Debug.WriteLine("frmDebugEvents_Load()")
End Sub

Private Sub frmDebugEvents_Activated( _
 ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles MyBase.Activated
    Debug.WriteLine("frmDebugEvents_Activated()")
End Sub

Private Sub frmDebugEvents_Click( _
 ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles MyBase.Click
    Debug.WriteLine("frmDebugEvents_Click()")
End Sub
```

Khi bạn chạy mẫu này, bạn cố chuyển tới lui giữa hai form để xem các sự kiện khác nhau kích hoạt. Bạn cũng sẽ nhấp vào để thay đổi kích thước form.

Lớp Debug có một số phương thức mà bạn có thể thấy hữu dụng. Bên dưới là bảng mô tả về một số phương thức thông dụng hơn mà bạn có thể sử dụng.

Phương thức	Mô tả
Assert	Sẽ dừng chương trình nếu điều kiện bạn chuyển tới phương thức này không bằng True. Nó chỉ hoạt động ở chế độ thiết kế. Tất cả các câu lệnh Debug đều được xóa khỏi chương trình biên dịch cuối cùng.
Write	Ghi giá trị vào cửa sổ output (xuất liệu) không có CRLF.
WriteLine	Ghi giá trị vào cửa sổ output có CRLF.
WriteIf	Ghi giá trị vào cửa sổ output nếu một điều kiện nhất định là True không có CRLF.
WriteLineIf	Ghi giá trị vào cửa sổ output nếu điều kiện nhất định là True có CRLF.

Bảng 4: *Các phương thức của lớp Debug.*

Phương thức Assert

Phương thức Assert là một công cụ gỡ rối rất mạnh trong đó nó cho phép bạn chắc chắn là các tham số chính xác luôn luôn được chuyển vào thủ tục con hoặc một biến luôn luôn có một giá trị hoặc một phạm vi các giá trị riêng. Một số lập trình viên khẳng định là bạn sẽ không bao giờ lập trình một ứng dụng mà không sử dụng phương thức Debug.Assert trong toàn bộ ứng dụng của bạn. Bên dưới là mã mẫu mà bạn có thể viết để kiểm tra xem một số được gõ vào hộp văn bản có chính xác hay không.

```
Private Sub btnAssert_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnAssert.Click
    Dim intNum As Integer

    intNum = CInt(txtNumber.Text)
    Debug.Assert(intNum >= 0 And intNum <= 5, _
     "Number must be between 0 and 5")
End Sub
```

Nếu bạn chạy mã trên và gõ số 6 vào hộp văn bản trên form mẫu này, thì sự khẳng định sẽ cho trở lại False. .NET vào thời gian chạy sẽ hiển thị một thông điệp như ở hình 22. Sau đó bạn được chọn lựa Abort (hủy bỏ) toàn bộ chương trình, Retry (thử lại) sự khẳng định bằng cách vào chế độ Break hoặc Ignore (bỏ qua) nó để tiếp tục.

Hình 23: *Assert sẽ hiển thị hộp thoại này khi sự khẳng định không thành công.*

Phương thức WriteLineIf

Nếu bạn chỉ muốn viết một dòng vào cửa sổ output khi một điều kiện nhất định là True, bạn sẽ sử dụng phương thức WriteLineIf của lớp Debug.

```
Private Sub btnWriteLineIf_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles
btnWriteLineIf.Click
    Dim intNum As Integer

    intNum = CInt(txtNumber.Text)
    Debug.WriteLineIf(intNum >= 0 And intNum <= 5, _
     "You input a correct number")
End Sub
```

Biên dịch có điều kiện

Các chỉ thị của trình biên dịch được dùng khi bạn cần thay đổi cách chương trình được biên dịch. Các chỉ thị của trình biên dịch là các câu lệnh #If để chỉ thị trình biên dịch giữ nguyên và lấy ra mã nào. Thí dụ, bạn có thể muốn đưa mã gỡ rối vào ứng dụng của bạn, nhưng khi bạn biên dịch lần cuối, bạn không muốn còn mã gỡ rối ở đó. Hoặc có thể bạn muốn lấy mã ra cho các tính năng nào đó nếu bạn đang tạo một phiên bản demo của sản phẩm.

Để sử dụng biên dịch có điều kiện, đầu tiên bạn phải khai báo hằng của trình biên dịch. Sau đó bạn khai báo bằng cách dùng câu lệnh #Const. Mỗi hằng được quy định một tên duy nhất và được gắn một giá trị. Khi các hằng này được khai báo, bạn có thể dùng chúng trong khối #If..#End If.

Khai báo hằng của trình biên dịch mức file

Bạn có thể khai báo các hằng của trình biên dịch bất kỳ nơi nào bên trong một file của chương trình. Sau đó hằng của trình biên dịch đó có thể dùng bất kỳ nơi nào bên trong file đó. Sẽ không thành vấn đề dù bạn đặt khai báo hằng ở trên đầu file, bên trong một lớp đã định nghĩa hoặc bên trong một thủ tục, vẫn có thể sử dụng nó trong toàn bộ file.

```
Public Class frmDebug
    Inherits System.Windows.Forms.Form

#Const conDebugMode = 1
```

#If...Then...#Else...#End If

Để tạo chỉ thị trình biên dịch, bạn sử dụng dấu # trước câu lệnh If. Điều này cho trình biên dịch biết là câu lệnh #If này

dành cho nó, chứ không phải để dùng vào thời gian chạy. Một thí dụ có thể trình bày các ngôn ngữ khác nhau với người dùng tùy thuộc vào phiên bản đã biên dịch mã của bạn.

```
Private Sub btnCond_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnCond.Click
#If conLanguage = 1 Then
    MessageBox.Show("Good Morning, Mr. Gates")
#Else
    MessageBox.Show("Guten Morgen, Herr Gates")
#End If
End Sub
```

Khi bạn khai báo hằng của trình biên dịch có tên *conLANGUAGE* và xác lập giá trị đó là một (1), bản tiếng Anh của phương thức MessageBox.Show được gộp trong file .EXE. Nếu giá trị được xác lập là một giá trị bất kỳ khác, phiên bản tiếng Đức được gộp.

Không giống một câu lệnh If bình thường. Thực ra, câu lệnh #If xóa bỏ mã trước khi nó biên dịch. Điều này dẫn tới kích thước của .EXE giảm và ngăn chặn mã ngoài ý muốn hoặc không cần thiết (chẳng hạn như các hộp thông điệp) đang được triển khai với người dùng. Đối với thí dụ trên, thực chất trình biên dịch chỉ thấy mã sau.

```
Private Sub btnCond_Click(
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnCond.Click
    MessageBox.Show("Good Morning, Mr. Gates")
End Sub
```

Dù chưa được nói tới ở Visual Basic, các toán tử logic có thể được dùng trong câu lệnh #If. Thí dụ, mã sau là hoàn toàn hợp lệ:

```
#If conDEBUGCODE = 1 Or conDEBUGCODE = 2 Then
   MessageBox.Show("Debugging")
#End If
```

Các hằng lập sẵn

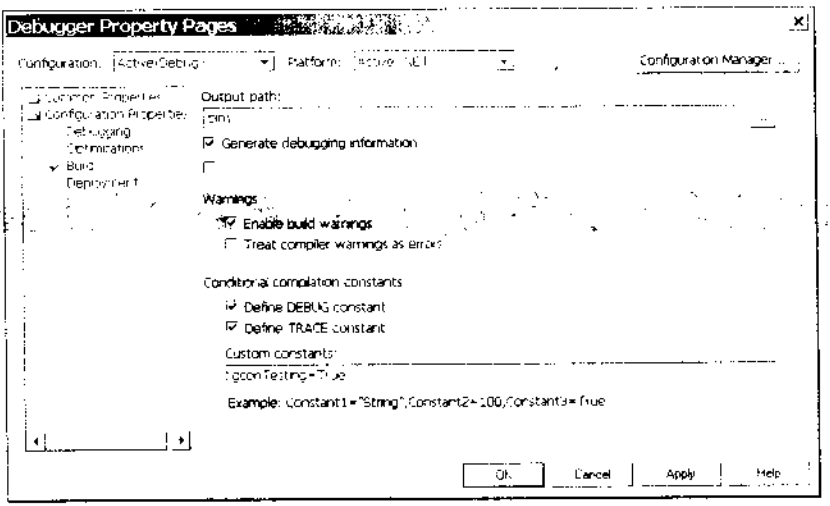
Có ba hằng lập sẵn trong .NET mà bạn có thể khai thác.

Hằng	Mô tả
Config	Không rõ.
Debug	Hằng này được định nghĩa là giá trị True khi bạn biên dịch một build Debug của chương trình. Bạn có thể chọn kiểu build bạn đang thực hiện bằng cách lựa Build|Configuration Manager từ menu, sau đó chọn phiên bản Debug hoặc Release. Khi bạn xác lập ứng dụng để xác lập là Release, hằng Debug được xác lập là False.
Trace	Hằng này được định nghĩa là giá trị True khi bạn đang biên dịch build Debug của chương trình. Khi xác lập là True, các phương thức của đối tượng Trace sẽ xuất liệu.

Bảng 5: *Các hằng lập sẵn của trình biên dịch.*

Khai báo hằng của trình biên dịch toàn cục

Để khai báo một hằng của trình biên dịch có thể được dùng trong toàn bộ ứng dụng, bạn lựa Project|menu Properties và lựa Configuration Properties|hộp thoại Build.



Hình 24: *Property Page để xác lập các hằng.*

Trong hộp văn bản Custom Constants, nhập tên hằng, một dấu bằng và giá trị gán. Khi bạn tạo tên hằng, nó phải giống các nguyên tắc đặt tên cho biến Visual Basic. Giá trị bạn gán vào hằng chỉ có thể là một giá trị numeric. Bạn có thể đặt nhiều giá trị vào hộp văn bản này bằng cách tách chúng với dấu hai chấm (:) như được trình bày bên dưới.

```
gconLANGUAGE = 0 : gconDEBUGCODE = -1
```

Ngoài ra, bên trong hộp thoại này là nơi bạn có thể định nghĩa cả hằng DEBUG lẫn TRACE.

Câu lệnh Stop

Câu lệnh Stop có thể được đặt vào bất kỳ nơi nào trong mã của bạn, nơi bạn muốn dừng thi hành và muốn vào chế độ gỡ rối. Stop là một câu lệnh khả thi và được lưu với mã của bạn. Ưu

điểm chính của việc dùng câu lệnh Stop là nó được lưu khi bạn thoát khỏi Visual Basic.

Tuy nhiên, bạn phải chắc chắn xóa các câu lệnh này trước khi tạo một file .EXE. Chúng hoạt động giống câu lệnh End trong file .EXE, trong đó chúng sẽ dừng chương trình của bạn ngay.

```
Private Sub btnStart_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnStart.Click
    Dim intValue As Integer

    Stop

    intValue = 10

    Call Proc1()
End Sub
```

Mẹo:

Phải xóa các câu lệnh Stop khỏi mã của bạn trước khi tạo file .EXE hoặc đưa câu lệnh Stop vào các câu lệnh biên dịch có điều kiện (#if/#endif) để chúng không còn trong chương trình đã biên dịch.

Tóm tắt

Chương này trình bày các tính năng khác nhau của môi trường gỡ rối tích hợp trong Visual Basic. Công cụ mạnh mẽ này giúp bạn truy tìm lỗi rất dễ dàng. Bằng cách xác lập các điểm ngắt, kiểm tra giá trị của các biến và bước xuyên qua từng dòng mã của bạn, bạn có thể gỡ rối ứng dụng của bạn hiệu quả hơn nhiều.

Câu hỏi ôn tập

1. Ba chế độ của Visual Basic.NET là gì?

2. Bạn dùng phím nào để bước xuyên qua mã nếu bạn đã cấu hình là Visual Studio developer?

3. Điểm ngắt có tác dụng gì?

4. Đúng hay sai: Câu lệnh Stop tự động xóa khỏi mã biên dịch cuối cùng của bạn?

5. Cửa sổ nào trình bày tất cả các biến của bạn bên trong một thủ tục?

Trả lời câu hỏi ôn tập

1. Design, Break và Run.

2. F11.

3. Dừng chương trình trên dòng mã, nơi nó được đặt.

4. Sai.

5. Cửa sổ Locals.

Chương 10

Xử lý lỗi trong VB.NET

- So sánh xử lý lỗi trong VB.NET với xử lý lỗi trong VB6.
- Tìm hiểu cách sử dụng khối Try/Catch để xử lý các lỗi thời gian chạy.
- Sử dụng các đối tượng Exception để xác định lỗi nào đã xảy ra.
- Đưa các ngoại lệ trở lại đối tượng gọi thủ tục.
- Tạo các lớp Exception của riêng bạn.

Xử lý lỗi

.NET framework cung cấp xử lý ngoại lệ có cấu trúc bằng cách dùng từ khóa Try, Catch, Finally và Throw trong VB.NET. Kiểu xử lý lỗi này đã có sẵn trong C++. Với bản phát hành .NET CLR,

kiểu xử lý lỗi này có thể dùng cho tất cả các ngôn ngữ .NET, gồm cả VB.NET.

Mặc dù Visual Basic đã hỗ trợ cơ chế riêng của nó để xử lý lỗi với điều kiện "Visual" được gắn với tiêu đề, nhưng các kĩ thuật mà các người phát triển VB có thể dùng hầu như chưa bổ sung đầy đủ. Còn có một số vấn đề liên quan đến việc xử lý lỗi của VB (xem Listing 1) đã khiến cho các người phát triển VB chưa được thỏa đáng, kể cả người đã có kinh nghiệm và những người mới tiếp cận:

- VB6 đòi hỏi bạn vào các thủ tục để xử lý các lỗi. Các câu lệnh On Error Goto, Resume và Resume Next đều liên quan tới việc nhảy tới lui trong mã. Các kĩ thuật xử lý lỗi của VB6 chuẩn ít nhất liên quan tới hai bước nhảy bên trong thủ tục (bước một tới khối xử lý lỗi và bước hai trở lại điểm thoát thủ tục chung).

- Nếu bạn thực hiện tốt việc lập trình trong VB6, bao gồm việc bảo đảm các thủ tục có duy nhất một điểm thoát, vị trí thuận tiện nhất đặt điểm thoát đó là ở giữa các thủ tục (trước khối xử lý lỗi). Điều đó rất dễ quên "Exit Sub" hoặc "Exit Function".

- Không có cách nào để "đẩy" và "bật ra" các handler (trình xử lý) lỗi trong VB6. Nếu bạn muốn duy trì bẫy lỗi (error trap) hiện hành, thiết lập một bẫy khác và sau đó trở lại bẫy đầu tiên, bạn phải nhớ câu lệnh On Error Goto... chính xác mỗi lần bạn muốn thay đổi các handler.

- VB6 chỉ có một đối tượng Err đơn giản. Nếu lỗi xảy ra và bạn không xử lý lỗi đó ngay, bạn có thể mất thông tin lỗi vĩnh viễn trước khi bạn có cơ hội xử lý lỗi.

- Tài liệu của VB6 hầu như không đề cập tới các kiểu lỗi (tức là các số hiệu lỗi) mà bạn có thể nhận vì một hoạt động bạn đã sử dụng trong mã. Sự trông cậy duy nhất của bạn là thực nghiệm, xem các số hiệu lỗi nào bạn có thể tạo ra bằng cách kích hoạt các lỗi khi kiểm tra và bẫy các lỗi riêng biệt đó trong mã của bạn.

```
Sub TestVB6()
    On Error GoTo HandleErrors

    ' Do something in here that
    ' might raise an error.

ExitHere:
    ' Perform cleanup code here.
    ' Disregard errors in this
    ' cleanup code.
    On Error Resume Next
    ' Perform cleanup code.
    Exit Sub

HandleErrors:
    Select Case Err.Number
        ' Add cases for each
        ' error number you want to trap.
        Case Else
            ' Add "last-ditch" error handler.
            MsgBox "Error: " & Err.Description
    End Select
    Resume ExitHere
End Sub
```

Listing 1: *Xử lý lỗi trong VB6 ít nhất phải cần một bước nhảy và thường dùng nhiều bước nhảy hơn.*

Ngoài ra, mặc dù các người phát triển VB có khả năng dùng phương thức Err.Raise rất tốt để đưa các lỗi trở lại các thủ tục gọi, nhưng kĩ thuật này không trở thành chuẩn. Nhiều người phát triển tạo mã được gọi bởi các phương thức khác chỉ cho trở lại giá trị lỗi để chỉ định thành công hay thất bại, thay vì đưa ra lỗi khiến khó bị sai sót. Có thể chỉ vì coi thường các giá trị lỗi trả lại từ các thủ tục bạn gọi, trong nhiều trường hợp, mà bị lỗi vì

một lý do nào đó vào thời gian chạy sẽ không đưa các lỗi thích hợp trở lại các đối tượng gọi của nó.

Với sự bổ sung xử lý ngoại lệ có cấu trúc, các người phát triển dễ dàng xử lý các thông báo lỗi, đưa ra các lỗi và xác định nguyên nhân xảy ra lỗi thời gian chạy. Xử lý ngoại lệ có cấu trúc cung cấp một số tính năng làm việc khác hơn trong các phiên bản trước đây của VB:

- Xử lý lỗi trong .NET dựa vào lớp Exception, chứa thông tin không những về lỗi hiện hành mà còn cả một danh sách liên kết của các lỗi có thể kích hoạt lỗi hiện hành này.

- Bạn có thể thừa kế từ lớp Exception, tạo các ngoại lệ riêng của bạn có cùng tính năng như lớp cơ sở hoặc tính năng được mở rộng khi cần. Vì mã của bạn có thể bẫy các ngoại lệ riêng biệt, tạo lớp ngoại lệ riêng cho bạn nhiều khả năng mềm dẻo.

- Vì mỗi lớp trong .NET framework đưa ra các ngoại lệ khi nó gặp các lỗi thời gian chạy, các người phát triển thường có thói quen bẫy các ngoại lệ và xử lý chúng. Điều này rất có thể tạo cho các ngoại lệ bạn đưa ra từ trong các thành phần cũng sẽ được xử lý thành công.

- Bạn có thể lồng các khối Try/Catch bên trong khối Try, Catch hoặc Finally. Điều này giúp các người phát triển khả năng quản lý xử lý ngoại lệ ở một mức tinh vi mà chúng đòi hỏi.

Listing 2 trình bày sự bố trí của một handler ngoại lệ đơn giản trong VB.NET. Các phần sau mô tả chi tiết cách dùng từng từ khóa được trình bày ở listing 2 và cách khai thác lớp Exception để truy tìm và đưa ra các lỗi.

```
Sub TestVBNET()
  Try
    ' Do something in here that
    ' might raise an error.
  Catch
    ' Handle exceptions that occur within
    ' the Try block, here.
  Finally
    ' Perform cleanup code in here.
  End Try
End Sub
```

Listing 2: *Xử lý lỗi trong VB.NET không đòi hỏi các bước nhảy.*

Mẹo:

Bạn có thể pha trộn kiểu xử lý lỗi cũ của VB6 với xử lý ngoại lệ có cấu trúc của .NET trong cùng project, nhưng không ở bên trong cùng thủ tục. On Error và Try không thể tồn tại bên trong cùng thủ tục.

Thí dụ

Tất cả các thí dụ ở đây sử dụng cùng giả thuyết cơ bản: mục đích của bạn là mở một file, truy tìm chiều dài của nó và sau đó đóng file ấy. Mỗi thí dụ sử dụng mã sau để thực hiện tác vụ của nó, truy tìm tên file từ một hộp văn bản trên form mẫu txtFileName:

```
Dim lngSize As Long
Dim s As FileStream

s = File.Open(txtFileName.Text, FileMode.Open)
lngSize = s.Length
s.Close()
```

Đương nhiên, có thể (vì nhiều nguyên nhân) mã có thể không thành công. Thí dụ, mã sẽ phát sinh một ngoại lệ nếu:

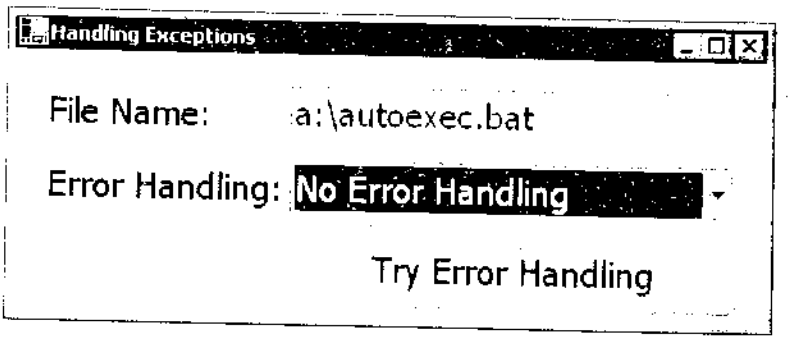
- Không tìm thấy file.
- Đường dẫn không tồn tại.
- Ổ đĩa chứa file không sẵn sàng (có thể do bạn yêu cầu kích thước của một file trên ổ đĩa mềm không có đĩa).
- Bạn không được phép truy xuất file hoặc folder.
- Bạn đã chỉ định một tên file không hợp lệ.

Tất cả các thí dụ tiếp theo sử dụng mã này có phần thay đổi để minh họa các tính năng của xử lý ngoại lệ có cấu trúc.

Sử dụng tính năng xử lý lỗi

Các phần tiếp theo tiến hành qua một loạt các thí dụ, thêm các tính năng xử lý lỗi càng phức tạp hơn vào mã mẫu bạn đã thấy. Bắt đầu với trường hợp bạn không thêm mã xử lý ngoại lệ nào, các thí dụ này đưa ra các khái niệm về bẫy và nhận dạng các ngoại lệ.

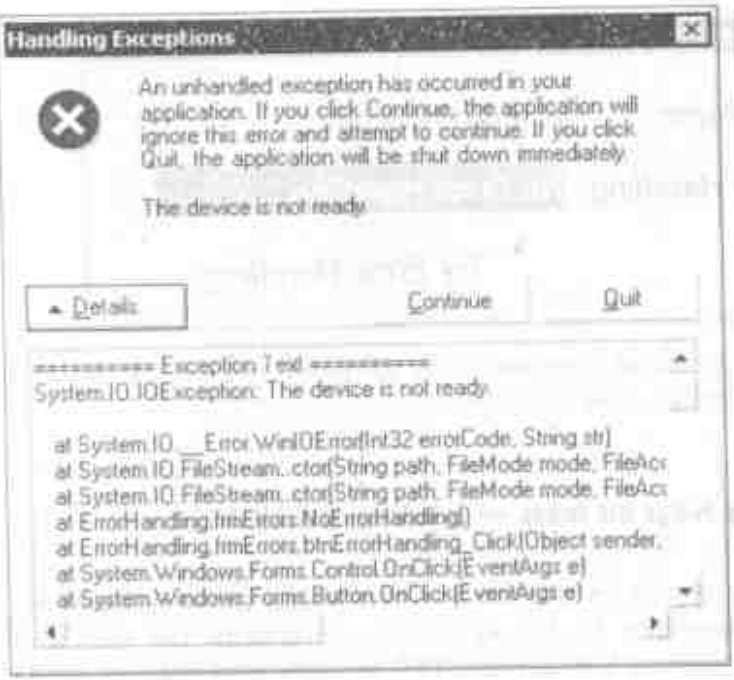
Ứng dụng mẫu tương ứng với tài liệu này, ErrorHandling.sln, bao gồm một form, frmErrors để cho phép bạn thử các kĩ thuật khác nhau được mô tả ở đây (xem hình 1). Đối với mỗi trường hợp, thử nhập đường dẫn tới một file không tồn tại hoặc một ổ đĩa không hiện hữu hay một ổ đĩa không có đĩa hoặc một đường dẫn bất kỳ khác có thể kích hoạt lỗi hệ thống file.



Hình 1: *Sử dụng form mẫu này để minh họa tất cả các tính năng khác nhau được thảo luận ở đây.*

Trường hợp cơ bản — không xử lý lỗi

Điều gì sẽ xảy ra nếu mã của bạn không có xử lý ngoại lệ? Trong trường hợp đó, bất kỳ các lỗi nào xảy ra vào thời gian chạy sẽ được xuất hiện trở lại .NET vào thời gian chạy và lúc đó nó hiển thị với người dùng một hộp thoại cảnh báo và khó hiểu như ở hình 2. Để tránh gặp hộp thoại này, nếu lỗi thời gian chạy xảy ra, ít nhất bạn phải thêm xử lý ngoại lệ vào các thủ tục mức trên cùng và vào các thủ tục mức thấp khi cần.



Hình 2: *Sử dụng nút "Continue" làm cho handler xử lý lỗi mặc định của .NET hơi nguy hiểm. Nút Details cũng không hữu dụng gì.*

Mẹo:

Giống như trong VB6, nếu bạn không thêm xử lý ngoại lệ vào thủ tục và lỗi xảy ra bên trong thủ tục đó, .NET vào thời gian chạy sẽ bỏ ngay thủ tục hiện hành khỏi *call stack* (ngăn xếp gọi) và sẽ trở lại thủ tục trước đó. Nếu thủ tục đó có xử lý lỗi, .NET vào thời gian chạy sẽ sử dụng mã đó. Nếu không, .NET vào thời gian chạy tiếp tục bỏ ngay các thủ tục khỏi ngăn xếp cho tới khi nó trở lại một thủ tục có xử lý lỗi. Nếu không có thủ tục nào có xử lý lỗi, thì trong khi trở lại thủ tục gọi đầu tiên, .NET vào thời gian chạy tự xử lý lỗi, như bạn thấy ở hình 2.

Thêm khối Try/Catch đơn giản

Để xử lý các lỗi thời gian chạy tốt, thêm khối Try/Catch/End Try vào bên mã bất kỳ bạn muốn bảo vệ. Nếu lỗi thời gian chạy xảy ra ở mã ấy bên trong khối Try, thì sự thi hành sẽ tiếp tục ngay với mã đó bên trong khối Catch:

```
Try
  s = File.Open(txtFileName.Text, FileMode.Open)
  lngSize = s.Length
  s.Close()
Catch
  MessageBox.Show("Error occurred!")
End Try
```

Khi mã này chạy, thay vì ứng dụng hiển thị cảnh báo và dừng, bạn sẽ thấy một cảnh báo "Error occurred" và ứng dụng có thể tiếp tục. Để bạn tự kiểm tra điều này, chọn tùy chọn "Simple Catch" trong hộp combo Error Handling trên form mẫu.

Mẹo:

Nếu bạn thêm khối Try/End Try vào thủ tục của bạn, bạn cần có ít nhất một khối Catch (bạn sẽ thấy có nhiều khối Catch sau). Nếu bạn không quan tâm đến các lỗi xảy ra, không cần đưa gì vào khối Catch. Điều này không thích hợp lắm, nhưng chỉ bỏ qua bất kỳ các lỗi nào xảy ra.

Xác định lỗi đã xảy ra

Khi lỗi thời gian chạy xảy ra, làm cách nào bạn có thể xác định lỗi gì và bạn xử lý nó bằng cách nào? Bạn có thể tạo một biến, đã khai báo là As Exception để truy tìm thông tin lỗi cho bạn. Lớp Exception cung cấp thông tin về lỗi thời gian chạy, như ở bảng 1.

Thành viên	Mô tả
HelpLink	Liên kết với file trợ giúp kết hợp với ngoại lệ này.
InnerException	Một tham chiếu với ngoại lệ bên trong — ngoại lệ đã xảy ra ban đầu, nếu ngoại lệ này dựa vào ngoại lệ trước đó. Các ngoại lệ có thể được lồng nhau. Tức là khi một thủ tục đưa ra một ngoại lệ, nó có thể lồng một ngoại lệ khác bên trong ngoại lệ đang phát sinh, chuyển cả hai ngoại lệ tới đối tượng gọi. Đặc tính InnerException cho truy xuất ngoại lệ bên trong.
Message	Văn bản thông điệp lỗi.
StackTrace	Dấu vết ngăn xếp, như một chuỗi ở điểm lỗi xảy ra.
TargetSite	Tên của phương thức đã phát sinh ngoại lệ.
ToString	Chuyển đổi tên ngoại lệ, mô tả và xuất liệu ngăn xếp hiện hành thành một chuỗi đơn giản.
Message	Cho trở lại mô tả của lỗi đã xảy ra.

Bảng 1: *Các thành viên hữu dụng của lớp Exception.*

Khối Catch có tham chiếu với biến, như sau:

```
Try
    ' Code that might trigger an exception.
Catch e As Exception
    ' Handle the exception, using e, in here.
End Try
```

Bạn cũng có thể khai báo biến Exception bên ngoài khối Catch:

```
Dim e As Exception
Try
```

```
    ' Code that might trigger an exception.
Catch e
    ' Handle the exception, using e, in here.
End Try
```

Bạn có thể dùng mã sau để bẫy một ngoại lệ và hiển thị văn bản chỉ định vấn đề đã xảy ra:

```
' Simple Exception option on the sample form.
Private Sub SimpleException()
  Dim lngSize As Long
  Dim s As FileStream

  ' Display the entire contents of the Exception
object.
  Try
    s = File.Open(txtFileName.Text, FileMode.Open)
    lngSize = s.Length
    s.Close()
  Catch e As Exception
    MessageBox.Show(e.ToString)
  End Try
End Sub
```

Mẹo:

Tên của đối tượng Exception không quan trọng. Mã mẫu sử dụng "e" như tên biến, nhưng điều này được tùy chọn. Nếu bạn thấy tên đó bất lợi trong các thủ tục, bạn có thể chọn một tên khác.

Nếu bạn chỉ muốn hiển thị thông báo lỗi, chỉ định lỗi cá biệt bạn bẫy, bạn có thể dùng đặc tính Message của lớp Exception, như sau:

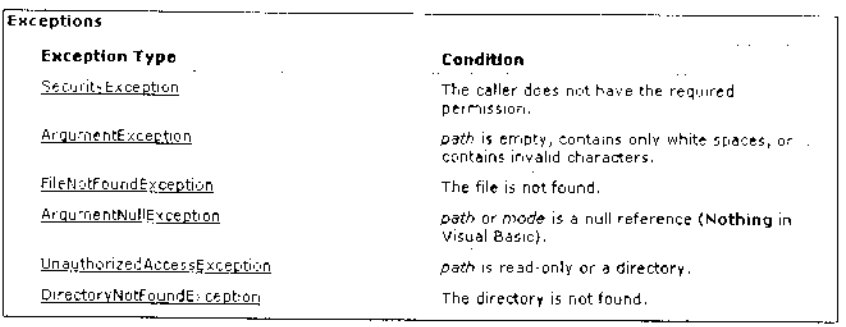
```
' Which Exception option on the sample form.
Private Sub WhichException()
  Dim lngSize As Long
  Dim s As FileStream

  ' Now you can at least tell what went wrong!
  Try
```

```
        s = File.Open(txtFileName.Text, FileMode.Open)
        lngSize = s.Length
        s.Close()
    Catch e As Exception
        MessageBox.Show("Error occurred: " & e.Message)
    End Try
End Sub
```

Đến đây bạn đã biết cách bẫy một ngoại lệ khi nó xảy ra và cách cho người dùng biết lỗi nào xảy ra. Phần lớn thời gian bạn cần để có thể dùng một hoạt động nhất định tùy thuộc vào lỗi cụ thể đã xảy ra. Trong VB6, điều này có nghĩa là việc thêm khối Select Case dựa vào số hiệu lỗi hoạt động. Trong VB.NET, điều này liên quan tới việc cộng thêm các khối Catch cho mỗi lỗi bạn muốn bẫy riêng. Phần kế tiếp tìm hiểu cách bạn có thể thêm tính năng này vào các thủ tục.

Làm việc với các ngoại lệ riêng biệt

.NET framework cung cấp một số lớp ngoại lệ riêng biệt đáng kể, tất cả thừa kế từ lớp Exception cơ sở. Trong tài liệu framework, bạn sẽ thấy các bảng liệt kê tất cả các ngoại lệ có thể xảy ra khi bạn gọi một phương thức bất kỳ. Thí dụ, hình 3, đã thu giữa từ tài liệu .NET framework, tạo cho nó dễ xác định những gì có thể sai khi gọi phương thức File.Open.



Exceptions

Exception Type	Condition
SecurityException	The caller does not have the required permission.
ArgumentException	*path* is empty, contains only white spaces, or contains invalid characters.
FileNotFoundException	The file is not found.
ArgumentNullException	*path* or *mode* is a null reference (**Nothing** in Visual Basic).
UnauthorizedAccessException	*path* is read-only or a directory.
DirectoryNotFoundException	The directory is not found.

Hình 3: *Tài liệu .NET liệt kê tất cả các ngoại lệ có thể xảy ra khi gọi phương thức File.Open.*

Các thủ tục của bạn có thể có tối đa các khối Catch khi cần thiết, để bạn xử lý từng các ngoại lệ khác nhau. Thủ tục sau từ project mẫu, kiểm tra một số ngoại lệ khác nhau và xử lý từng ngoại lệ. Để kiểm tra thủ tục này, thử một số ngoại lệ riêng biệt. Thí dụ, thay đổi tên file:

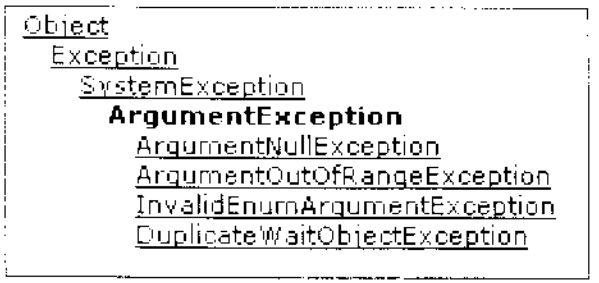
1. Ở một đường dẫn tốt, nhưng lựa một file không tồn tại.
2. Trên một ổ đĩa không tồn tại.
3. Ở một đường dẫn không tồn tại.
4. Trên một ổ đĩa không sẵn sàng.

```
' Multiple Exceptions option on the sample form.
Private Sub MultipleExceptions()
  Dim lngSize As Long
  Dim s As FileStream

  Try
    s = File.Open(txtFileName.Text, FileMode.Open)
    lngSize = s.Length
    s.Close()
  Catch e As ArgumentException
    MessageBox.Show( _
```

```
      "You specified an invalid filename. " &
      "Make sure you enter something besides spaces.")
    Catch e As FileNotFoundException
      MessageBox.Show( _
       "The file you specified can't be found. " & _
       "Please try again.")
    Catch e As ArgumentNullException
      MessageBox.Show("You passed in a Null argument.")
    Catch e As UnauthorizedAccessException
      MessageBox.Show( _
       "You specified a folder name, not a file name.")
    Catch e As DirectoryNotFoundException
      MessageBox.Show( _
       "You specified a folder that doesn't exist " &`_
       "or can't be found.")
    Catch e As SecurityException
      MessageBox.Show( _
       "You don't have sufficient rights " & _
       "to open the selected file.")
    Catch e As IOException
      ' A generic exception handler, for any IO error
      ' that hasn't been caught yet. Here, it ought
      ' to just be that the drive isn't ready.
      MessageBox.Show( _
       "The drive you selected is not ready. " & _
       "Make sure the drive contains valid media.")
    Catch e As Exception
      MessageBox.Show("An unknown error occurred.")
    End Try
  End Sub
```

Xác định phân cấp ngoại lệ

Theo bất kỳ một trong các liên kết trong bảng Exceptions được trình bày ở hình 3, để đưa bạn tới tài liệu về một đối tượng ngoại lệ cá biệt. Tài liệu này có một phân cấp ngoại lệ như được trình bày ở hình 4. Bạn cần hiểu phân cấp các đối tượng này khi thêm nhiều khối Catch.



Hình 4: *Phân cấp thừa kế cho phép bạn xác định quan hệ "thuộc về" của các đối tượng.*

Sử dụng phân cấp thừa kế ngoại lệ

Trong phân cấp ngoại lệ được trình bày ở hình 4, bạn có thể thấy ArgumentNullException thừa kế từ ArgumentException, ArgumentException thừa kế từ SystemException, SystemException thừa kế từ Exception. Mỗi cấp trong phân cấp chỉ định tăng thêm một mức đặc trưng riêng — tức là, bạn càng xuống thấp, ngoại lệ càng chi tiết.

Vì mỗi mức thừa kế từ lớp được định nghĩa trên nó, mỗi mức thấp hơn là một thể nghiệm của kiểu được chỉ định trên nó. ArgumentNullException "thuộc về" ArgumentException, ArgumentException "thuộc về" SystemException, SystemException "thuộc về" Exception. "thuộc về" xuất hiện trong các dấu ngoặc kép, vì nó là một tác tử có nghĩa — khi bạn có nhiều khối Catch, các khối đó so khớp dựa vào ngoại lệ hiện hành bằng cách dùng nguyên tắc "thuộc về". Tức là khi xác lập nhiều khối Catch, đầu tiên vào thời gian chạy khi tìm thấy một so khớp nơi ngoại lệ hiện hành đáp ứng nguyên tắc "thuộc về" đối với ngoại lệ được bẫy bởi khối Catch, vào thời gian chạy sử dụng khối Catch để xử lý ngoại lệ và không tìm thêm nữa. Nói cách khác, thứ tự của các khối Catch có ý nghĩa dựa vào quan hệ "thuộc về" này. Tất cả các ngoại lệ thừa kế từ lớp Exception cơ

sở, vì vậy bạn luôn luôn cần có khối Catch xử lý lớp Exception cơ sở cuối cùng, nếu bạn luôn luôn có nó.

Đưa ra các ngoại lệ

Bạn có thể cần đưa ra các lỗi của các thủ tục của mình, cho các đối tượng gọi biết một ngoại lệ nào đó đã xảy ra. Bạn có thể đơn giản chuyển trở lại một ngoại lệ thời gian chạy chuẩn được cung cấp bởi .NET framework hoặc bạn cần tạo một điều kiện ngoại lệ riêng. Trong cả hai trường hợp, bạn sẽ dùng từ khóa Throw để đưa ra ngoại lệ của khối hiện hành.

Lưu ý: Từ khóa Throw hoạt động hầu như giống cách phương thức Err.Raise làm việc trong VB6.

Sử dụng từ khóa Throw

Bạn có thể sử dụng từ khóa Throw bằng hai cách. Bạn có thể:

- Đưa ra một lỗi trở lại từ bên trong khối Catch:

```
Catch e As Exception
  Throw
```

- Đưa ra một lỗi từ bên trong mã bất kỳ, bao gồm khối Try:

```
Throw New FileNotFoundException()
```

Lưu ý: Kĩ thuật đầu tiên, đưa ra một ngoại lệ trở lại, chỉ hoạt động từ bên trong khối Catch. Kĩ thuật thứ hai, đưa ra một lỗi mới, hoạt động bất kỳ nơi đâu.

Tìm các handler

Khi bạn đưa ra một ngoại lệ, .NET thời gian chạy tiến hành hoạt động của nó tới call stack (ngăn xếp gọi) thủ tục để tìm handler ngoại lệ thích hợp. (Nếu bạn ở khối Try khi bạn đưa ra ngoại lệ, ngoại lệ thời gian chạy sẽ dùng các khối Catch cục bộ, nếu có, để xử lý ngoại lệ ấy đầu tiên). Ngay khi ngoại lệ thời gian chạy tìm thấy khối Catch cho ngoại lệ bạn đưa ra, nó thi hành mã nó tìm thấy ở đó. Nếu nó không tìm thấy khối Catch thích hợp suốt đường tới ngăn xếp gọi, ngoại lệ thời gian chạy xử lý chính ngoại lệ ấy (như được trình bày trước đây ở hình 2).

Các tùy chọn xử lý lỗi

Bạn có thể xác định các ngoại lệ nào bạn muốn xử lý và các ngoại lệ nào bạn muốn đưa ra trở lại các đối tượng gọi của mình. Khi một ngoại lệ xảy ra, bạn có các tùy chọn sau:

- **Do nothing at all.** Trong trường hợp đó, .NET thời gian chạy sẽ tự động đưa ra ngoại lệ trở lại thủ tục đã gọi mã.

- **Catch specific errors.** Trong trường hợp này, các ngoại lệ bạn xử lý sẽ không chuyển trở lại, nhưng các ngoại lệ bạn không xử lý sẽ đưa trở lại thủ tục gọi.

- **Handle all errors.** Thêm khối "Catch e as Exception" vào tập các khối Catch và không có lỗi nào chuyển qua xử lý ngoại lệ của bạn, trừ khi chính bạn đưa ra một lỗi riêng biệt.

- **Throw errors**. Bạn dùng tùy chọn này để đưa ra một lỗi bất kỳ trở lại đối tượng gọi cụ thể. Bằng cách dùng câu

lệnh Throw, bạn có thể đưa ra lỗi hiện hành hoặc bất kỳ lỗi nào khác tới handler ngoại lệ của đối tượng gọi.

Chuyển thông tin lỗi

Nếu bạn cần ngăn chặn các ngoại lệ khác nhau và đưa tất cả chúng trở lại đối tượng gọi như một kiểu ngoại lệ đơn giản, Throw thực hiện điều này dễ dàng. Ở thí dụ kế tiếp, mã bắt tất cả các ngoại lệ và bất kể những gì gây ra ngoại lệ, nó đưa ra đối tượng FileNotFoundException trở lại đối tượng gọi. Trong một số trường hợp, như trường hợp này, thủ tục gọi không thể chú ý tới chính xác những gì đã xảy ra hoặc tại sao không thể tìm thấy file. Đối tượng gọi chỉ có thể chú ý tới file chưa có và cần phân biệt ngoại lệ đặc biệt đó với các ngoại lệ khác nhau khác.

Cấu tử đối tượng Exception

Cấu tử đối tượng Exception quá tải (overload) bằng một số cách. Bạn có thể không chuyển vào tham số nào (bạn sẽ có một đối tượng Exception generic (chung) với các giá trị mặc định cho các đặc tính của nó); một chuỗi chỉ định thông báo lỗi bạn muốn gửi trở lại đối tượng gọi hoặc một chuỗi và một đối tượng Exception để chỉ định thông báo lỗi ấy và ngoại lệ gốc đã xảy ra (điền vào đặc tính InnerException của ngoại lệ bạn chuyển trở lại đối tượng gọi). Thí dụ, ở đây sử dụng cấu tử cuối cùng để chuyển trở lại ngoại lệ bên trong.

Bạn cũng nên làm cho đối tượng gọi sử dụng được thông tin ngoại lệ gốc, ngoài ngoại lệ mã của bạn phát sinh. Trong trường hợp đó, bạn sẽ thấy cấu tử của lớp Exception cung cấp một phiên bản quá tải để cho phép bạn chỉ định ngoại lệ bên trong. Tức là

bạn có thể chuyển đối tượng ngoại lệ đã đưa ra lỗi ban đầu. Đối tượng gọi có thể kiểm tra ngoại lệ này, nếu nó cần.

Mẹo:

Đặc tính InnerException của một ngoại lệ chính là đối tượng Exception và nó cũng có thể có đặc tính InnerException không phải là Nothing. Do đó bạn có thể thôi đi theo một danh sách các ngoại lệ liên kết khi bạn bắt đầu đi sâu vào đặc tính InnerException. Bạn nên tiếp tục truy tìm đặc tính InnerException lặp đi lặp lại cho tới khi đặc tính cho trở lại Nothing, để tìm xuyên qua tất cả các lỗi đã xảy ra.

Ở thí dụ sau, thủ tục TestThrow đưa FileNotFoundException trở lại đối tượng gọi của nó, bất kể nó nhận lỗi nào. Ngoài ra, nó điền vào đặc tính InnerException của ngoại lệ với đối tượng ngoại lệ gốc. Thí dụ này hiển thị thông báo lỗi được đặt ra cùng với văn bản kết hợp với ngoại lệ gốc:

```
' Throw Exception option on the sample form.
Private Sub ThrowException()
  Dim lngSize As Long
  Dim s As FileStream

  ' Catch an exception thrown by the called procedure.
  Try
    TestThrow()
Catch e As FileNotFoundException
    MessageBox.Show("Error occurred: " & e.Message)
    ' Use e.InnerException to get to error
    ' that triggered this one.
    MessageBox.Show(e.InnerException.Message)
  End Try
End Sub

Private Sub TestThrow()
  Dim lngSize As Long
  Dim s As FileStream
```

```
    ' No matter what happens, throw back
    ' a File Not Found exception.
    Try
      s = File.Open(txtFileName.Text, FileMode.Open)
      lngSize = s.Length
      s.Close()
    Catch e As Exception
Throw (New FileNotFoundException( _
        "Unable to open the specified file.", e))
    End Try
End Sub
```

Chạy mã không điều kiện

Bạn có thể thấy là ngoài mã trong khối Try và Catch, bạn cần thêm mã để chạy dù có xảy ra lỗi hay không. Bạn nên giải phóng tài nguyên, đóng file hoặc xử lý các vấn đề khác cần xảy ra dưới bất kỳ các tình huống nào. Để chạy mã không điều kiện, bạn cần phải dùng khối Finally.

Khối Finally

Để chạy mã không điều kiện, thêm khối Finally sau bất kỳ các khối Catch nào. Mã trong khối này sẽ chạy dù mã của bạn đưa ra ngoại lệ và dù là bạn thêm một câu lệnh Exit Function (hoặc Exit Sub) cụ thể bên trong khối Catch.

Thí dụ, bạn có thể quyết định mã của mình cần xác lập biến đối tượng FileStream là Nothing, dù một lỗi bất kỳ có xảy ra hay không khi làm việc với file ấy. Bạn có thể sửa đổi thủ tục như được trình bày ở dưới để gọi mã hoàn tất dù có lỗi xảy ra hay không:

```
' Test Finally option on the sample form.
Private Sub TestFinally()
```

```
    Dim lngSize As Long
    Dim s As FileStream

    Try
      s = File.Open(txtFileName.Text, FileMode.Open)
      lngSize = s.Length
      s.Close()
    Catch e As Exception
      MessageBox.Show(e.Message)
    Finally
      ' Run this code no matter what happens.
      s = Nothing
    End Try
  End Sub
```

Mẹo:

Mặc dù khối Try/End Try của bạn phải chứa một hay nhiều khối Catch hoặc khối Finally, nhưng nó không cần chứa cả hai. Tức là khối Finally không có các khối Catch chính xác. Tại sao gộp khối Finally nếu bạn không có khối Catch? Nếu một ngoại lệ xảy ra bên trong thủ tục của bạn, .NET vào thời gian chạy sẽ tìm handler ngoại lệ thích hợp và điều đó có thể có nghĩa là nó thoát khỏi thủ tục của bạn (nếu không có khối Catch, điều này chắc chắn sẽ xảy ra), tìm call stack cho handler ngoại lệ. Nếu bạn muốn chạy mã trước khi .NET thời gian chạy thoát khỏi thủ tục, bạn cần có khối Finally.

Tạo các lớp ngoại lệ

Bạn có thể thấy là .NET framework không cung cấp cho bạn Exception đáp ứng các nhu cầu riêng của bạn. Thí dụ, có thể bạn muốn đưa ra một ngoại lệ, nếu người dùng lựa một file lớn hơn 100 byte. Mặc dù nó thường được coi là một điều kiện ngoại lệ, nhưng nó có thể là điều kiện lỗi bên trong ứng dụng của bạn.

Để tạo lớp ngoại lệ riêng, thực hiện các bước sau:

1. Tạo một lớp mới.

2. Thừa kế từ lớp cơ sở Exception.

Mẹo:

Thực ra bạn có thể thừa kế từ một lớp bất kỳ mà chính nó thừa kế từ lớp Exception. Thí dụ, có thể bạn cần thừa kế từ lớp IOException hoặc FileNotFoundException. Một trong hai lớp này sẽ làm như lớp cơ sở cho ngoại lệ của riêng bạn.

3. Cung cấp phương thức New của riêng bạn (thêm các overload thích hợp khi cần). Gọi lại MyBase.New để có lời gọi cấu tử của lớp cơ sở.

4. Thêm tính năng bổ sung bạn cần.

Lớp FileTooLargeException

Project mẫu có hai định nghĩa lớp sau (bên trong module frmErrors.vb) để cung cấp định nghĩa cho FileTooLargeException:

```
Public Class FileTooLargeException
  Inherits Exception
Private mlngFileSize As Long

  Public Sub New(ByVal Message As String)
    MyBase.New(Message)
  End Sub

  Public Sub New(ByVal Message As String, _
   ByVal Inner As Exception)
    MyBase.New(Message, Inner)
  End Sub

Public Sub New(ByVal Message As String, _
   ByVal Inner As Exception, ByVal FileSize As Long)
```

```
    MyBase.New(Message, Inner)
    mlngFileSize = FileSize
  End Sub

  Public ReadOnly Property FileSize() As Long
    Get
      Return mlngFileSize
    End Get
  End Property
End Class
```

Lớp này cung cấp các đặc tính lớp Exception chuẩn (vì nó thừa kế từ Exception), nhưng thêm một "wrinkle" mới — nó thêm một cấu tử mới để cho bạn chuyển vào kích thước file đã kích hoạt ngoại lệ. Ngoài ra, nó cung cấp đặc tính FileSize, vì vậy các đối tượng gọi của các thủ tục của bạn có thể xác định kích thước file đã kích hoạt ngoại lệ.

Hàm GetSize function được trình bày ở đây cố mở một file. Nếu file bạn yêu cầu quá lớn, GetSize đưa FileTooLargeException trở lại đối tượng gọi của bạn, chuyển thông báo lỗi của riêng nó và kích thước file bạn đã yêu cầu:

```
Private Function GetSize( _
 ByVal strFileName As String) As Long
  Dim lngSize As Long
  Dim s As FileStream

  ' Return the file size. If it's larger than 100 bytes
  ' (an arbitrary size), throw a FileTooLargeException
  ' (a user-defined exception) to the caller.

  Try
    s = File.Open(txtFileName.Text, FileMode.Open)
    lngSize = s.Length
    s.Close()
    If lngSize > 100 Then
      ' Pass back the new exception. There's no
      ' inner exception to pass back, so pass Nothing.
      Throw (New FileTooLargeException( _
       "The file you selected is too large.", _
       Nothing, lngSize))
```

```
    End If
    Return lngSize
  Catch
    ' Throw the exception right back to the caller.
    Throw
  Finally
    ' Run this code no matter what happens.
    s = Nothing
  End Try
End Function
```

Thủ tục kiểm tra được chuyển vào file bạn chỉ định trên form mẫu và bẫy FileTooLargeException. Trong khối Catch riêng biệt đó, mã truy tìm đặc tính FileSize của ngoại lệ và mã biên dịch và chạy tốt (dù cho đối tượng Exception bình thường không cung cấp đặc tính FileSize) vì thực chất ngoại lệ riêng biệt FileTooLargeException này không cung cấp đặc tính này:

```
' User-Defined Exception option on the sample form.
Private Sub UserDefinedException()
  Dim lngSize As Long

  ' Test a user-defined exception.

  Try
    lngSize = GetSize(txtFileName.Text)
  Catch e As FileTooLargeException
    MessageBox.Show( _
     String.Format( _
     "Please select a smaller file! " & _
     "The file you selected was {0} bytes.", _
     e.FileSize))
  Catch e As Exception
    MessageBox.Show(e.Message)
  End Try
End Sub
```

Mẹo:

Bạn sẽ thấy có thể tạo các lớp ngoại lệ của riêng bạn sẽ hữu dụng, thừa kế từ lớp Exception cơ sở, bất kỳ khi nào bạn cần thêm những thông tin riêng của bạn vào thông báo lỗi chuẩn.

Bạn nên tạo một lớp ngoại lệ để cung cấp toàn bộ thông tin khung ngăn xếp (tức là một cấu trúc dữ liệu nào đó chứa tất cả ngăn xếp gọi), thay vì một chuỗi đơn giản mà .NET framework cung cấp cho bạn trong đặc tính StackFrame của nó. Bạn có thể dễ dàng thực hiện điều này, bằng cách dùng lớp StackTrace và các thành viên của nó. Xem tài liệu framework để có thêm thông tin về lớp StackTrace và StackFrame.

Tóm tắt

- Xử lý ngoại lệ có cấu trúc mạnh hơn xử lý lỗi được VB6 cung cấp.
- Sử dụng khối Try để thêm xử lý ngoại lệ vào một khối mã.
- Thêm các khối Catch khi cần để bẫy các ngoại lệ cá biệt.
- .NET thời gian chạy xử lý các khối Catch theo thứ tự, tìm so khớp "thuộc về" dựa trên ngoại lệ hiện hành. Nó sử dụng khối đầu tiên nó tìm thấy để so khớp.
- Bạn có thể lồng các khối Try, làm cho nó dễ "đẩy" và "bật" các trạng thái xử lý ngoại lệ hiệu quả.
- Thêm khối Finally vào khối Try để chạy mã không có điều kiện, dù lỗi có xảy ra hay không.
- Bạn có thể tạo các lớp ngoại lệ để thừa kế từ lớp Exception cơ sở (hoặc một lớp bất kỳ thừa kế từ lớp đó) để thêm tính năng của riêng bạn.

Câu hỏi ôn tập

1. Những gì xảy ra nếu bạn không xử lý một ngoại lệ trong thủ tục của bạn?

2. Bạn có thể bẫy lỗi và dùng mã như thế nào vào .NET thời gian chạy bất kể lỗi đó?

3. Lớp nào cho phép bạn xác định thông báo lỗi từ bên trong khối Catch?

4. Bạn đã thêm mã vào khối Catch để xử lý lỗi và sau đó thoát khỏi thủ tục. Bạn cũng có thể thêm mã để chạy trước khi thoát khỏi thủ tục ấy dù lỗi có xảy ra hay không bằng cách nào?

5. Bạn cần cho đối tượng gọi thủ tục biết còn lại bao nhiêu không gian trống trên ổ đĩa khi ổ đĩa đầy quá không ghi được dữ liệu chỉ định. Hiện bạn đang đưa ra IOException, nhưng ngoại lệ này không cung cấp thông tin trên. Làm cách nào bạn có thể thêm đặc tính FreeSpace vào ngoại lệ lập sẵn?

Trả lời câu hỏi ôn tập

1. .NET thời gian chạy tiến hành hoạt động của nó tới ngăn xếp gọi cho tới khi nó tìm thấy một handler ngoại lệ thích hợp. Nếu bạn không xử lý lỗi bất kỳ nơi nào cho tới ngăn xếp gọi, handler lỗi mặc định được cung cấp bởi .NET thời gian chạy đảm nhiệm và bạn sẽ thấy một cảnh báo như ở hình 2.

2. Thêm khối Try vào bên mã của bạn và bên trong mã đó thêm khối Catch. Trong khối Catch không thêm mã nào.

3. Lớp Exception và đặc tính Message của nó.

4. Thêm khối Finally và đặt mã vào đó. Khối này phải xảy ra sau bất kỳ các khối Catch.

5. Tạo lớp của riêng bạn thừa kế từ IOException. Thêm đặc tính FreeSpace và cấu tử cho phép bạn chỉ định giá trị này khi bạn tạo thể nghiệm của lớp mới.