



Phuong Lan (Chủ biên)

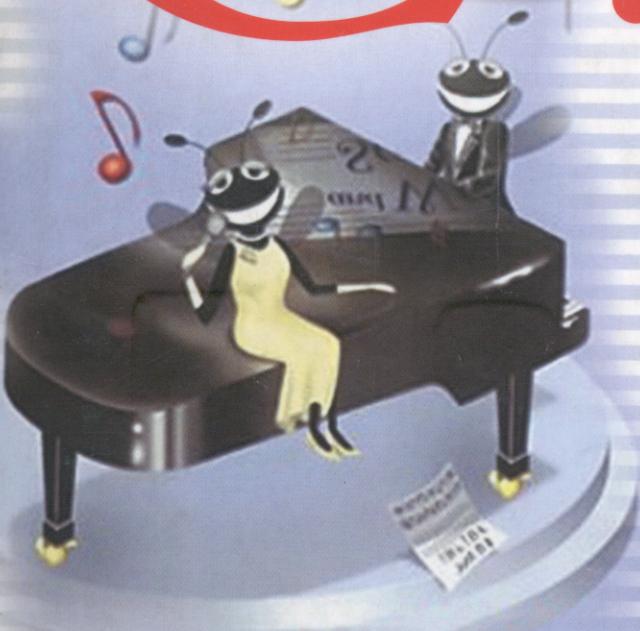


LẬP TRÌNH WINDOWS

với



.net



- Tìm hiểu kiến trúc .NET qua C#**
- Tạo ứng dụng Windows Forms, ADO.NET, Web Service, ASP.NET**
- GDI+, COM, Assembly trong .NET**



NHÀ XUẤT BẢN LAO ĐỘNG - XÃ HỘI



Phương Lan (Chủ biên)
Hoàng Đức Hải

vnmt 4722

LẬP TRÌNH WINDOWS

C# với .net

NHÀ XUẤT BẢN LAO ĐỘNG - XÃ HỘI

LỜI NGỎ

Kính thưa quý Bạn đọc gần xa, Ban xuất bản MK.PUB trước hết xin bày tỏ lòng biết ơn và niềm vinh hạnh trước nhiệt tình của đồng đảo Bạn đọc đối với tủ sách MK.PUB trong thời gian qua.

Khẩu hiệu của chúng tôi là:

- * Lao động khoa học nghiêm túc.
- * Chất lượng và ngày càng chất lượng hơn.
- * Tất cả vì Bạn đọc.

Rất nhiều Bạn đọc đã gửi mail cho chúng tôi đóng góp nhiều ý kiến quý báu cho tủ sách.

Ban xuất bản MK.PUB xin được kính mời quý Bạn đọc tham gia cùng nâng cao chất lượng tủ sách của chúng ta:

Trong quá trình đọc, xin các Bạn ghi chú lại các sai sót (dù nhỏ, lớn) của cuốn sách hoặc các nhận xét của riêng Bạn. Sau đó xin gửi về địa chỉ:

E-mail: mk.book@cinet.vnnews.com; mk.pub@cinet.vnnews.com

Hoặc gửi về: Nhà sách Minh Khai

249 Nguyễn Thị Minh Khai, Q.I, Tp. Hồ Chí Minh

Nếu Bạn ghi chú trực tiếp lên cuốn sách, rồi gửi cuốn sách đó cho chúng tôi thì chúng tôi sẽ xin hoàn lại cước phí bưu điện và gửi lại cho Bạn cuốn sách khác.

Chúng tôi xin gửi tặng một cuốn sách của tủ sách MK.PUB tùy chọn lựa của Bạn theo một danh mục thích hợp sẽ được gửi tới Bạn.

Với mục đích ngày càng nâng cao chất lượng của tủ sách MK.PUB, chúng tôi rất mong nhận được sự hợp tác của quý Bạn đọc gần xa.

"MK.PUB và Bạn đọc cùng làm!"

MK.PUB

MỤC LỤC

LỜI NGỎ	3
MỤC LỤC.....	5
Phần I MỞ ĐẦU	13
MỞ ĐẦU	15
Chương 1-1: Bộ khung .NET (.NET Famework)	15
1. Giới thiệu nhanh về .NET	15
2. Xoá bỏ huyền thoại về một máy ảo .net	15
3. Ngôn ngữ trung gian Microsoft (MSIL).....	17
4. Quản lý bộ nhớ .NET.....	18
5. Hệ thống kiểu dữ liệu của .NET	18
6. Các đối tượng trong hệ thống của khung .NET.....	19
7. C# - một ngôn ngữ lập trình mới.	19
8. Các đối tượng tự mô tả như thế nào?.....	20
9. Tương tác với thế giới COM	20
10. Windows Forms, Web Controls và GDI+.....	20
Chương 1.2 : Bộ thực thi ngôn ngữ tổng quát	21
1. Tổng quan	21
1.1. Đơn giản hóa việc phát triển	21
1.2. Công cụ hỗ trợ.....	21
1.3. Hỗ trợ đa ngôn ngữ	22
1.4. Việc triển khai được thực hiện dễ hơn	22
1.5. Sự tách biệt các phần mềm	22
1.6. Tính an toàn kiểu và sự kiểm tra	23
1.7. Tính bảo mật.....	23
2. Quan hệ của CLR với .NET	23
2.1. Chi tiết về CLR.....	23
2.2. CLR vào lúc thực thi	25
2.3. Các kiểu dữ liệu được cung cấp bởi CLR.....	27
2.4. Dữ liệu và mã lệnh của .NET	28
2.5. Mã lệnh của COM và việc truy xuất dữ liệu.....	29
2.6. Các thành phần mở rộng của .NET đối với C++	32
2.7. Các ngoại lệ (Exception)	35
2.8. Hỗ trợ việc gỡ rối.....	36
3. Kết chương	36

Chương 1.3 : Ngôn ngữ trung gian IL.....	37
1. Liên tác (INTER-OP) ngôn ngữ.....	38
2. HELLO IL.....	39
3. Hàm (FUNCTIONS)	41
4. Lớp (CLASSES).....	43
5. ILDASM.....	45
6. METADATA	48
7. REFLECTION API	49
8. Kết chương.....	52
Chương 1.4 : Làm việc với bộ quản lý C++.....	53
1. Các từ khoá mở rộng của C++.....	54
2. Sử dụng trình biên dịch C++ cho Managed C++.....	55
3. Các lớp thu gom rác (Garbage Collected)	56
4. Chỉ dẫn (Directive) #using	56
5. Chuỗi (String).....	58
6. Trộn các mã lệnh được quản lý (Managed) và không được quản lý (Unmanaged)	59
7. Chốt chặt (Pinning) các mã lệnh Managed	59
8. Các giao tiếp được quản lý (Managed Interface).....	60
9. Tạo ra các kiểu giá trị	62
10. Tạo và sử dụng các mô hình ủy thác (Delegate).....	65
11. Gọi các DLL .NET tùy biến từ mã lệnh Managed C++ của bạn.....	68
12. Sử dụng DLL C++ được quản lý và không được quản lý trong chương trình .NET của bạn.....	70
13. Dùng các thuộc tính (Properties) trong các lớp của bạn.....	74
14. Bảo đảm việc sắp xếp và đóng gói các cấu trúc C++ của bạn	76
15. Kết chương.....	84
Chương 1.5 : Giới thiệu VISUAL STUDIO .NET	85
1. Tích hợp môi trường phát triển (IDE)	85
2. Dự án (Projects).....	88
3. Nhiều dự án trong một Solution đơn	88
4. Các file phụ thuộc của dự án (Project Dependencies)	90
5. Class View	90
6. Resource View	91
7. Macro Explorer.....	91
8. Trình giúp đỡ (Help)	91
9. Server Explorer	93
10. Cửa sổ kết xuất thông tin (Output)	94

11. Tìm kiếm các ký hiệu đặc trưng	96
12. Chỉ mục và kết quả tìm kiếm	97
13. Cửa sổ gỡ lỗi (Debug).....	97
14. Trình trợ giúp động (Dynamic Help).....	100
15. Cửa sổ Favorite	101
16. Chương trình gỡ lỗi (Debugging).....	101
17. Cài đặt breakpoint nâng cao.....	102
17.1. Conditional Breakpoints	103
17.2. Hit Counts	104
18. Làm gì khi mã chương trình bị tạm ngừng	104
19. Gắn trình gỡ lỗi (Debugger) vào một tiến trình (Process)	104
20. JIT Debugging	105
21. Kết chương	106
Phần II : Ngôn Ngữ C#.....	107
Chương 2.1 : Các vấn đề cơ bản của C#	109
1. Hệ thống kiểu trong C#.....	110
1.1. Các kiểu giá trị trong lập trình.....	111
1.2. Các kiểu tham chiếu.....	114
1.3. Bao và không bao (Box và Unbox).....	116
2. Các khái niệm lập trình.....	117
2.1. Không gian tên (Namespace)	117
2.2. Các phát biểu lệnh (Statements).....	120
2.3. Toán tử điều kiện ?.....	123
2.4. Toán tử (Operators).....	134
2.5. Arrays (Mảng).....	137
2.6. Struct (Cấu trúc)	140
3. Lớp (Class).....	143
3.1. Đối tượng (Object).....	145
3.2. Phương thức (Method).....	145
3.3. Truyền tham số.....	147
3.4. Thuộc tính (Properties)	151
3.5. Toán tử (Operator).....	153
4. Kế thừa (Inheritance)	160
5. Đa hình (Polymorphism).....	162
6. Giao diện (Interfaces)	165
7. Mô hình chuyển giao (Delegates)	168
8. Kết chương	170

Chương 2.2 : C# nâng cao.....	171
1. Tập hợp (Collection) của .NET.....	171
1.1. Stack	171
1.2. Hàng đợi (Queue).....	172
1.3. Bảng băm Hashtable.....	173
2. Danh sách liên kết (Link List).....	174
3. Attributes (đặc tính).....	178
4. XML Serialization.....	179
5. C# hỗ trợ XML Serialization	180
6. Kết chương.....	192
Phần III : WINDOWS FORMS	193
Chương 3.1 : Windows Forms.....	195
1. Giới thiệu về Windows Forms của kiến trúc .NET	195
2. Windows Forms và ứng dụng Hello World	196
3. Tạo và sử dụng các bộ xử lý sự kiện (Event Handler).....	198
4. Định nghĩa các kiểu đường viền (Border Style) cho Forms	201
5. Tạo và thêm vào trình đơn menu.....	202
6. Tạo và thêm vào các mục chọn tắt (Shortcut Menu).....	204
7. Xử lý sự kiện phát sinh từ Menu.....	205
7.1. Sự kiện điều khiển giao diện người dùng của MenuItems	206
7.2. Định nghĩa MenuItem là thanh phân cách (Seperator).....	207
7.3. Xử lý sự kiện Select.....	209
7.4. Thiết kế khung Menu.....	212
7.5. Hiển thị Menu từ phải qua trái.....	214
7.6. Tạo và sử dụng các Menu ngữ cảnh (ContextMenu).....	214
7.7. Thay thế, sao chép lại và trộn các mục của Menu.....	215
7.8. Tạo và thêm vào các Menu con (sub-Menu)	222
8. Kết chương.....	224
Chương 3.2 : Xử lý giao diện đồ họa (GUI).....	225
1. Giới thiệu về thành phần giao diện đồ họa (GUI)	225
2. Hộp thoại (Dialog).....	225
2.1. Sử dụng các hộp thoại thông dụng (Common Dialog)	226
2.2. Hộp thoại Print và Print Preview.....	236
3. Tạo hộp thoại	244
3.1. Hộp thoại Model và Modeless	244
3.2. Chuyển dữ liệu giữa các thành phần trong hộp thoại	247
3.3. Kiểm tra hợp lệ (Validation)	249

4. Sử dụng các điều khiển.....	252
4.1. Checkbox và nút Radio	252
4.2. Ô soạn thảo Edit.....	257
4.3. ListBox	257
4.4. TreeView	257
4.5. Bảng điều khiển Tab	258
4.6. Quản lý các thành phần điều khiển động	258
5. Kết chương.....	284
Chương 3.3 : Ràng buộc dữ liệu	285
1. Chiến lược ràng buộc dữ liệu.....	285
2. Các nguồn dữ liệu ràng buộc	285
2.1. Giao tiếp IList.....	285
2.2. Các đối tượng .NET cài đặt IList	286
3. Ràng buộc giản đơn.....	286
4. Ràng buộc dữ liệu giản đơn	291
5. Ràng buộc dữ liệu phức hợp	297
6. Ràng buộc thành phần điều khiển với cơ sở dữ liệu sử dụng ADO.NET ..	301
7. Tạo ra một khung nhìn cơ sở dữ liệu (Database Viewer) với Visual Studio và ADO.NET	302
8. Kết chương.....	304
Chương 3.4 : Xây dựng ứng dụng Windows Forms (Scribble .NET).....	305
1. Các tài nguyên trong .NET	305
2. Địa phương hóa sản phẩm một cách dễ dàng	306
3. Các lớp quản lý tài nguyên của .Net	306
3.1. Tìm một văn hóa	307
4. Tạo lập các tài nguyên văn bản	308
4.1. Chuỗi tài nguyên văn bản.....	309
4.2. Một chương trình mẫu dựa trên tài nguyên	311
4.3. Tạo lập và sử dụng gói kết hợp	312
5. Dùng Visual Studio.NET để quốc tế hóa	315
6. Tài nguyên hình ảnh	317
7. Sử dụng danh sách hình ảnh	318
8. Phương cách lập trình truy xuất đến các tài nguyên	323
9. Đọc và viết tập tin XML RESX	329
10. Kết chương	338
Chương 3.5 : GDI+: Giao diện đồ họa của .NET	339
1. Các nguyên lý cơ bản của GDI+.....	339



2. Đối tượng Graphics	340
3. Các hệ tọa độ đồ họa	340
4. Vẽ đường thẳng và các hình đơn giản.....	341
5. Sử dụng bút vẽ (Pen) và cọ vẽ (Brush) chuyển màu gam màu	344
6. Bút vẽ và cọ vẽ có chất liệu.....	346
7. Làm trơn các đầu mút của đường thẳng.....	347
8. Đường cong và các đường đồ thị	349
9. Đối tượng GraphicsPath	357
9.1. Thêm văn bản (Text) và các đường đồ thị khác	359
10. Kỹ thuật cắt bằng đường đường đồ thị và vùng (Region).....	360
11. Các phép biến đổi (Transformations).....	365
12. Sự phối màu với giá trị Alpha (Alpha Blending).....	374
13. Sự phối màu với giá trị Alpha cho hình ảnh.....	376
14. Những thao tác xử lý khác trong không gian màu	380
15. Kết chương.....	384
Chương 3.6 : Thực hành ứng dụng Windows Forms.....	385
1. Sử dụng thuộc tính	385
2. Nâng cấp thuộc tính Experience	390
3. Giải thích ứng dụng : Formpaint.exe	393
4. Kết chương.....	432
Phần IV : Kỹ thuật Web.....	433
Chương 4.1 : ASP.NET	435
1. Web một hơi thở mới	435
2. Yêu cầu cần thiết cho ASP.NET	435
3. Hello ASP.NET	436
4. Thêm chút hương vị	441
4.1. Kiểm tra ý tưởng	441
5. Kết chương.....	450
Chương 4.2 : Truy xuất dữ liệu trên .NET.....	451
1. Tầng dữ liệu (DataLayer)	451
2. Lớp Employee và lớp Department.....	462
3. Stored Procedures	462
4. Cài đặt lớp	463
5. Kiểm tra (Testing)	470
6. Hỗ trợ tìm kiếm giản đơn	472
7. Kết chương.....	478
Chương 4.3 : WebForms.....	479

1. Giới thiệu ASP.NET WebForms	479
1.1. Các UserControl	479
1.2. HeaderControl	479
1.3. EmployeeViewPagelet: Một cách tiếp cận khác.....	485
2. Các trang ASPX	492
2.1. EmployeeEdit.....	492
2.2. EmployeeListing.....	498
2.3. Trang tìm kiếm: Nơi bắt đầu tìm thông tin.....	507
3. Kết chương.....	514
Chương 4.4 : Các dịch vụ Web (Webservices).....	515
1. Dịch vụ Echo (Echo Service)	515
2. Xây dựng lớp Proxy.....	520
3. Proxyfactory.....	525
4. Các chương trình Windows forms sử dụng WebService	527
5. Trả về kiểu do người dùng định nghĩa	530
6. Tạo Service	530
7. Tạo ra các ràng buộc cho Client	533
8. Các thuộc tính XML.....	536
9. Kết chương.....	542
Phần V : GÓI KẾT HỢP	543
Chương 5.1 : Các gói kết hợp (Assembly)	545
1. Gói kết hợp (Assembly) là gì?.....	545
1.1. Nội dung gói kết hợp.....	545
1.2. Định vị gói kết hợp	546
2. Gói kết hợp lưu trong một tập tin đơn	547
3. Gói kết hợp lưu trong nhiều tập tin	547
4. Các thuộc tính của gói kết hợp	549
5. Tải nạp gói kết hợp khi chương trình đang thực thi.....	551
5.1. Dự án FormHost.....	551
5.2. Trình duyệt gói (Assembly Viewer) đơn giản	560
6. Kết chương.....	568
Chương 5.2 : Chữ ký số và phiên bản.....	569
1. Dll hell	569
2. Bộ đệm (Cache) toàn cục dành cho gói kết hợp (Global Assembly Cache)	569
3. Phiên bản (Version) của component.....	570
4. Các gói Assembly sử dụng nhiều phiên bản	573

5. Ràng buộc tùy biến: Cấu hình ứng dụng	574
6. Kết chương	578
Chương 5.3 : Tương tác với thế giới COM.....	579
1. Thế giới của com	579
2. .NET hỗ trợ com	579
2.1. Trình Tlbimp.exe	581
2.2. Ràng buộc sớm (Early Binding)	585
2.3. Ràng buộc muộn (Late Binding).....	585
2.4. IConnectionPoint.....	587
2.5. Những vấn đề về tiêu trình (Thread)	589
2.6. So sánh kiểm .NET và COM	589
3. Xem các thành phần .NET như là các đối tượng COM	590
3.1. Trình RegAsm.exe	591
3.2. COM Callable Wrapper.....	591
4. Các vấn đề khi dùng chung .net và com với nhau	591
5. Kết chương	592
Chương 5.4 : Các tiêu trình (Threads).....	593
1. Đa tiêu trình	593
1.1. Tiêu trình ứng dụng (Application Thread)	593
1.2. Tiêu trình làm việc hay tiêu trình thợ (Worker Thread).....	594
1.3. Tạo một tiêu trình thợ	594
1.4. Thuộc tính ThreadStatic	596
1.5. Liên kết – Đem các tiêu trình lại bên nhau	598
2. Đồng bộ tiêu trình (Thread Syncronization)	599
2.1. Từ khoá Lock	599
2.2. Mutex	601
2.3. AutoResetEvent	605
2.4. ManualResetEvent	607
2.5. Các tiêu trình dùng chung (Thread Pools)	607
2.6. QueueUserWorkItem	607
3. Mở rộng các tiêu trình .NET	609
3.1. Lớp WorkerThread	609
3.2. Bữa tối của các nhà triết học	611
4. Kết chương	614

Phân I

MỞ ĐẦU

Trong phần này:

- ◆ Bộ khung .NET (.NET Framework)
- ◆ Bộ thực thi ngôn ngữ tổng quát
- ◆ Ngôn ngữ trung gian IL
- ◆ Làm việc với bộ quản lý C++
- ◆ Giới thiệu Visual Studio .NET

MỞ ĐẦU

CHƯƠNG 1-1: BỘ KHUNG .NET (.NET FAMEWORK)

1. GIỚI THIỆU NHANH VỀ .NET

Trong công nghiệp máy tính, có thể nói rằng Microsoft đã mang lại những bước đi quan trọng nhất. Chính hành động can đảm của Bill Gates và Steve Ballmer khi họ nói với ông lớn IBM rằng họ có một hệ điều hành cần bán đã đem lại thành công vang dội cho DOS. Họ cũng đã làm như thế khi tạo ra một hệ điều hành Windows phổ biến hầu như khắp toàn cầu khi “cầm nhầm” cái phong cách “nhìn và cảm nhận” (Look and Feel) của Apple Macintosh.

Và bây giờ, có thể sẽ là một chuyển biến lớn khi Microsoft phát minh lại một cách tổng thể phương thức mà chúng ta sẽ làm việc, sẽ lập trình cho chiếc máy tính thân yêu. Nếu bạn đang là một nhà lập trình C++, hay bạn đang sống nhờ vào hệ điều hành Windows hay bộ thư viện MFC, không còn nghi ngờ chắc chắn bạn cũng sẽ bị đắm chìm trong thế giới .NET.

Trên thực tế ngày nay, Internet đã trở thành phương tiện cho chúng ta làm việc, thăm bạn bè, sử dụng tài khoản ngân hàng, chơi, tán ngẫu, rất thực tế và hiệu quả nhờ vào các phần mềm được viết để thực hiện các kết nối IP (Internet Protocol) cho WWW (World Wide Web). Các phần mềm được yêu cầu thực hiện hàng triệu cuộc giao dịch dữ liệu giữa các máy tính với nhau, và số lượng các cuộc giao dịch này ngày càng tăng theo sự tăng nhanh của khả năng và sự phức tạp của Web. Việc sử dụng thư điện tử, lướt trên các trang Web, trao đổi với các cơ sở dữ liệu và sử dụng các ứng dụng phân tán ngày càng trở nên phức tạp hơn, vì thế trình độ và kỹ thuật lập trình để tạo và bảo dưỡng các phần mềm cũng ngày càng được yêu cầu cao hơn, tinh vi hơn. Các nhà lập trình chúng ta hàng ngày phải đổi mới với COM, COM+, DCOM, SP, SOAP, XML, XSL ...cũng như phải chiến đấu với hàng tá các sách hướng dẫn, các bộ công cụ SDK (Software Developer Kit) phức tạp.

Microsoft, cũng như đa số trong chúng ta, đã tạo nên ngày càng nhiều các công cụ chuyên dùng. Hệ điều hành hay các ứng dụng Windows đầy đủ tính năng hơn, theo yêu cầu của kỹ thuật, công nghệ, và do đó đã thêm vào không ít các SDK hay các tiêu chuẩn ở chỗ này, chỗ nọ. Kết quả là hệ điều hành, thư viện, công cụ trả nền không còn khá chuyển, quá nặng nề với biết bao các mở rộng (extensions), add-in (phần phụ thêm). Thế là bộ khung .NET ra đời. .NET thực sự thay đổi điều kể trên. Đó không chỉ là một bước tiến, một nâng cấp hay một hợp nhất các công việc. Mà đó chính là một điều vĩ đại, một sự dung cảm, một cuộc bứt phá ngoạn mục. Nó định nghĩa lại tất cả mọi điều bạn đã biết về lập trình cho máy PC về tất cả mọi hình thù lẫn kích thước, kể cả về ngôn ngữ lập trình trong đó.

2. XÓA BỎ HUYỀN THOẠI VỀ MỘT MÁY ẢO .NET

Một trong những thực tế về .NET (mà có khả năng bị trích dẫn sai nhiều nhất) là bộ khung này không có cài đặt kỹ thuật máy ảo (tương tự như Java đã làm với JVM). Phần mềm chạy trên nền .NET thật sự được biên dịch và chạy ở

mức mã máy tương tự như các mã đã biên dịch của các chương trình C. Quan niệm phổ biến nhưng sai lầm là .NET đã dùng một ngôn ngữ trung gian, mà thường được gọi là P-Code (pseudo-code). Thật ra, .NET cài đặt một chế độ biên dịch nhiều tầng (multistage compilation): trước tiên, chương trình sẽ được biên dịch sang một dạng trung gian khả chuyển và đến lúc nó được thực thi, nó sẽ được biên dịch tức thời (just-in-time Compile hay còn gọi là JIT Compile) sang dạng thực thi cuối cùng (final executable form).

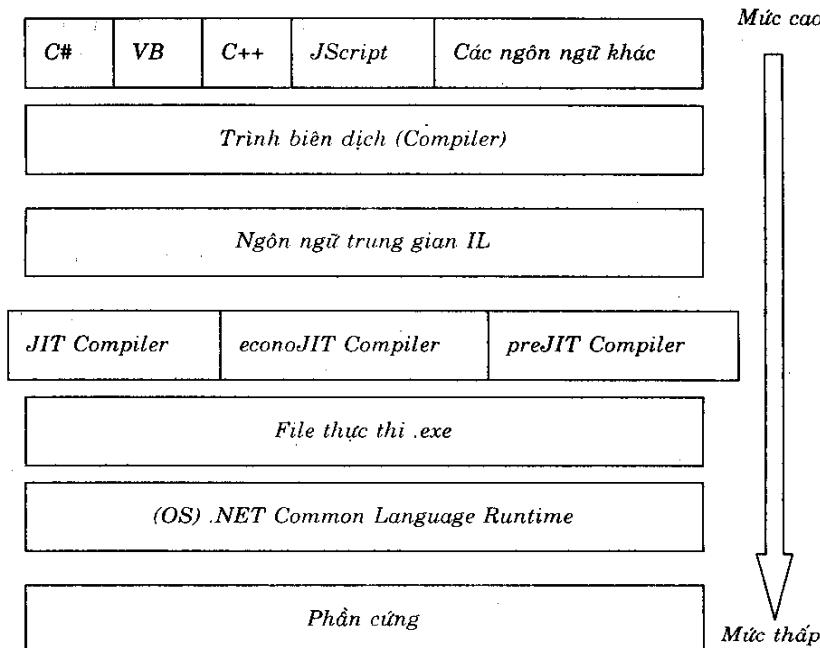
Ghi chú: Theo chúng tôi có lẽ đây là một cách nói cho khác với Java vì ta cũng có thể hiểu dạng trung gian khả chuyển gần như là binary-code của Java và JIT cũng gần giống với Java Runtime.

Mặc dù vẫn còn khá phức tạp, hệ thống này cũng đã đem lại cho các nhà lập trình vô số thuận tiện. Tất cả mọi ngôn ngữ đều có một mức chung sau khi biên dịch sơ khởi ở dạng trung gian. Nhờ thế, các módul đã được viết từ các ngôn ngữ khác nhau như C#, Visual Basic, Eiffel, FORTRAN, COBOL hay bất kỳ ngôn ngữ nào sẽ được hỗ trợ trong tương lai có thể được tích hợp vào trong một ứng dụng duy nhất. Lý do của sự thể này là một módul một khi được biên dịch ở dạng ngôn ngữ trung gian (Intermediate Language hay gọi tắt là IL) sẽ được đóng gói với tất cả mô tả của mình. Các giao diện, thuộc tính, phương thức và các lớp của nó được mô tả bằng siêu dữ liệu (metadata). Tất cả những thông tin này sẽ dễ dàng đọc được bởi các môi trường lập trình nhanh RAD (rapid application development environment) như Visual Studio .NET.

Khi trình thực thi, .NET được gọi để chạy một chương trình đã được chuyển đổi sang dạng IL, nó sẽ gọi một trình biên dịch tức thời (JITer) để biên dịch đoạn IL đó sang mã máy hoàn toàn để thực thi trên bộ vi xử lý của máy. Trình JITer làm việc rất nhanh và hiệu quả, cả ở việc không biên dịch các phần không được dùng đến của mã lệnh IL. Hiện nhiên là việc biên dịch cũng cần đến thời gian và phụ thuộc vào việc chúng ta xác định thời gian nạp (load time) chương trình hay tính khả chuyển (portability) cái nào sẽ cần hơn. Và khi ta quyết định tính khả chuyển không quan trọng, hệ thống sẽ cung cấp một trình biên dịch pre-JIT cho phép ta chuyển đổi IL thành mã máy thường trực, coi như tạo nên các tập tin thực thi .EXE hay .COM quen thuộc.

Minh họa dưới cho thấy các tầng của bộ khung .NET, được xem xét bởi một lập trình viên. Các phần in xám là các phần được .NET thêm vào. Bộ khung .NET hiện nay cung cấp 4 ngôn ngữ riêng: C#, Visual Basic, C++ với bộ *tự quản mở rộng* (managed extension) và JScript. Một vài ngôn ngữ khác được phát triển bởi các công ty khác như Fujitsu với COBOL,...

Đã có một nhóm các trình biên dịch chuyển mã lệnh của các ngôn ngữ hỗ trợ .NET thành các módul ngôn ngữ trung gian. Bên dưới sẽ là các trình biên dịch tức thời chuyển ngôn ngữ trung gian sang mã máy (IL-to-machine-code JITer). Có một trình JITer chuẩn được hệ thống dùng với nguồn lực và bộ nhớ dồi dào, nó sẽ biên dịch dạng chuẩn của IL thành dạng được tối ưu hóa của mã máy trên máy hiện hành.



Hình 1-1

Kế đó là trình biên dịch JIT kinh tế (econoJIT compiler), nó biên dịch rất nhanh nhưng không tính đến (hay tính đến rất ít) sự tối ưu của lệnh và phụ thuộc vào một tập con của ngôn ngữ trung gian tiền tối ưu (Pre-optimized IL).

Trình biên dịch cuối cùng là trình tiền biên dịch (preJIT compiler), có nhiệm vụ là tạo ra các mã thực thi được, các mã mà ta có thể phân phối thẳng đến người dùng. Phương thức này chỉ được sử dụng đối với những chương trình không cần thay đổi gì trên những nền phần cứng khác nhau.

Cuối cùng là CLR (.NET Common Language Runtime - tạm dịch là trình thực thi ngôn ngữ phổ quát). Khối chức năng này cung ứng tất cả các dịch vụ mà các thành tố của chúng sẽ giao tiếp với máy hay với hệ điều hành riêng trên máy đó. Rất thú vị nếu chúng ta chú ý rằng máy mà ta vừa nói ở trên không chỉ là máy PC và hệ điều hành cũng không chỉ là DOS hay Windows. Microsoft đã có một chiến lược “gom thâu thiên hạ” bằng cách xây dựng bộ khung .NET trên mọi nền, từ các dụng cụ nhỏ nhất, các thiết bị cầm tay, điện thoại, các PDA cho đến các hệ UNIX hay LINUX. Mặc dù đến nay .NET vẫn chưa thể đáp ứng ngay với các hệ thống khác hơn là Windows 2000 (I386), nhưng tính khả chuyển của nó cũng đang trở nên đủ dùng như là một môi trường lập trình viết-một-lần-chạy-khắp-nơi.

3. NGÔN NGỮ TRUNG GIAN MICROSOFT (MSIL)

Điều đang nói là IL của Microsoft không ẩn dangle sau các tầng của máy. Nó là một ngôn ngữ/mã đủ rõ, dựa trên stack, có một chút tương tự như mã assembly.

mà bạn có thể tự viết láy nếu bạn đủ can đảm. Dĩ nhiên cũng có những công cụ để bạn có thể dịch ngược các bản mã IL lại để xem các dòng lệnh của mình và các đối tượng hệ thống. Trong một chương sau chúng ta sẽ cùng phân tích chi tiết về IL và thử viết trực tiếp các ứng dụng nhỏ bằng IL.

4. QUẢN LÝ BỘ NHỚ .NET

Một đặc điểm đáng bàn cãi của .NET là trình quản lý bộ nhớ thực thi của nó hoạt động như một hệ thống thu gom rác (garbage-collected system viết tắt là GC system). Có người thì thích thú, người lại lo sợ hay chê tẻ lARENT di vì nó. Có thể bạn đã là các nhà lập trình từ lâu lắm, nếu thế, chắc bạn khó thể mà quên được những ác mộng trong những ngày của Lisp, lúc đợi chờ người dọn rác đến, mà ông ta chỉ đến vào thứ ba: cả núi giấy loại ở chung quanh bạn. Còn bạn là lập trình viên C++ thì hẳn bạn sẽ thấy trình quản lý bộ nhớ nhảy loạn xạ và có khi dẫn đến treo máy khi bạn quên đi lời nguyền về sự quản lý việc định vị con trỏ và loại bỏ các vùng nhớ bạn đã cấp.

Trình quản lý bộ nhớ của .NET đã tiếp cận việc định vị bộ nhớ theo một cách khác. Một khối bộ nhớ được tự động ghi nhớ thông tin khi nó được yêu cầu cấp phát cho đối tượng và có những tham chiếu đến nó. Chỉ khi tất cả các tham chiếu đến khối nhớ được xoá thì đối tượng hay vùng nhớ mới bị hủy thật sự. Cách giải quyết như thế thật sự đã giảm nhẹ gánh nặng quản lý bộ nhớ cho lập trình viên. Bạn không còn cần phải nhớ là đã xoá vùng nhớ đã cấp hay chưa; đơn giản là từ nay bạn chỉ cần biết là bạn không còn dùng nó nữa. Một lớp sẽ không còn phải chú ý theo dõi lượng tham chiếu đến nó nữa, nó đã biết được khi nào nó sẽ tự mình huỷ bỏ.

Để giảm sự phân mảnh của vùng nhớ động (heap), trình GC cũng đã hướng mục tiêu vào việc cung cấp các vùng bộ nhớ đang sử dụng hay các vùng nhớ tự do chưa được dùng đến. Điều này ngăn ngừa bộ nhớ bị mất vô lý và nâng cao hiệu quả của một hệ máy chủ. Heap được quản lý cũng chắc chắn rằng tất cả các truy xuất bộ nhớ không chuẩn, như tràn bộ nhớ hay treo máy, sẽ không thể tác động đến dữ liệu của chương trình khác đang chạy trên cùng máy này. Và như thế nó làm cho hệ thống trở lên tin cậy và bảo mật.

Bộ GC thường bị coi là chậm và không hiệu quả nhưng Microsoft đã quyết định rằng GC của .NET sẽ khác, nó thật sự hoạt động theo thời gian thực. GC của .NET sẽ hoạt động tức thời và không tác động đáng kể lên bộ vi xử lý (CPU). Tổng quát, trên một máy Windows 2000, GC chỉ cần khoảng 1/1000 tổng thời gian xử lý của CPU.

Cuối cùng, có cả một hệ thống dự phòng cho cái gọi là vùng nhớ hay mã lệnh không được quản lý (hay không an toàn) mà các lập trình viên cần dùng con trỏ để định vị vùng nhớ thường gặp phải. Như thế bạn có thể dùng các cấu trúc cổ xưa của C++ ngay trong bộ khung .NET.

5. HỆ THỐNG KIỂU DỮ LIỆU CỦA .NET

Các lập trình viên C++ có lẽ sẽ có nhiều ngạc nhiên khi biết về cách quản lý các kiểu dữ liệu của .NET. Một kiểu nguyên int sẽ không chỉ có nghĩa như

là một vùng nhớ 2-bytes nào đó. Thật ra, dữ liệu vẫn được lưu vào bộ nhớ với một kích cỡ nào đấy nhưng bây giờ bạn đã có thể xem chúng như là những đối tượng thực sự và truy cập đến chúng theo các phương thức mà đối tượng cho phép.

Có 2 phân biệt nhỏ về nhóm dữ liệu: kiểu giá trị (Value type) như chars, int, doubles và một số nữa. Kiểu tham chiếu (Reference type) như array, interface, class hay string.

Một điều thú vị là mọi ngôn ngữ trong Visual Studio .NET như C#, VB, C++ đều sử dụng các kiểu dữ liệu này giống như cách mà .NET đã tự định nghĩa. Nghĩa là, integer của VB sẽ dùng hoàn toàn như integer của C# và việc chuyển đổi giữa kiểu thật (real type) và các kiểu như variant là không cần thiết.

6. CÁC ĐỐI TƯỢNG TRONG HỆ THỐNG CỦA KHUNG .NET

Mô hình các đối tượng của hệ thống của .NET được lưu giữ trong một tập các tập tin DLL. Vùng không gian hệ thống (System namespace) đã chứa một hệ thống các lớp (classes) cho các tập hợp, bảo mật, truy xuất tập tin, truy cập các hàm Windows API, XML,... và hiển nhiên là tất cả hệ thống .NET đều đã sẵn sàng cho C# và VB cũng như các ngôn ngữ khác.

Lưu ý: .NET đưa ra một khái niệm mới là Namespace hay không gian tên. Nó tương đương với khái niệm tổng quát thư viện (library).

7. C# - MỘT NGÔN NGỮ LẬP TRÌNH MỚI.

Khởi đầu với tên gọi là Cool vào khoảng năm 2000, C# đã tạo ra vô số các cuộc tranh luận lớn trên nền .NET. Từ các huyền thoại, các đồn đại và các ước đoán, C# ngày nay đã thật sự trở thành một ngôn ngữ đơn giản, hoàn toàn hướng đối tượng và rất tuyệt vời. Các đặc trưng này của C# cũng gần giống với những tuyên bố của Java? Dù có hay không thì rõ ràng các nhà thiết kế C# đã rất muốn tạo ra một ngôn ngữ có thể làm vừa lòng giới lập trình C++. C# rất dễ đáp ứng với họ và có không ít các cấu trúc gần giống như C++. Dĩ nhiên ta cũng không loại trừ sẽ có vài thiếu sót nhỏ nào đó mà đối với một số người thì chúng trở nên bất cập và không tiện lợi.

C# không hỗ trợ đa thừa kế (multiple inheritance): Một lớp (class) của C# có thể được khai báo là nó có thể dựng nên 1 hay nhiều các interface (giao diện hay giao tiếp), nhưng tính đa kế thừa của các lớp (classes) cơ bản lại không được chấp nhận. Đối với một số lập trình viên, đặc biệt là những ai yêu thích ALT, thì tính chất này được xem là một bất cập.

Hệ thống C# cũng không cho phép sử dụng bất kỳ các kiểu mẫu template nào cả. Dù vậy, chúng ta sẽ bàn thêm về vấn đề này ở các chương sau. Đa số các cuộc thảo luận về C# hiện nay đều từ phản ứng của những ai đã gạt bỏ C# như là một cố gắng hiển nhiên nhằm vào việc xoá bỏ Java. Tuy nhiên, thật ra C# và Java có thể được xem như những con ngựa trên những đường đua khác nhau, chúng có thể cùng tồn tại và chung sống hoà bình. C# và toàn bộ hệ thống .NET sẽ là công cụ tốt hơn cho việc kết hợp các ứng dụng khách và các ứng dụng phía máy chủ, còn Java có những khuyết điểm rõ ràng cho phần máy khách.

Chúng ta có thể nói thêm về C# là một ngôn ngữ hoàn toàn hướng đối tượng (totally object-oriented). C++ bắt nguồn từ C: trước hết đó là một sự bổ sung phức hợp cho C, sau đó là một cố gắng thực sự của định hướng đối tượng. Để tốt hơn, C++ phải giữ lại một số đặc điểm của ngôn ngữ không-hướng-dối-tượng: Đó là

những hàm độc lập và các biến toàn cục. C# thì khác, nó tuyệt giao hoàn toàn với các đặc tính như thế. Mọi kiểu biến, kể cả kiểu giá trị đơn giản, đều được xem xét như là một đối tượng. Mọi hàm đều phải thuộc một lớp (class). Không còn được dùng các biến toàn cục. Tuy thế, xin đừng lo lắng, vì các lớp chứa các hàm tĩnh hay biến tĩnh có thể được xem và hành động như các hàm độc lập và các biến toàn cục. Ta hãy xem xét một ví dụ vô cùng nổi tiếng, vô cùng được ưa dùng và cũng rất bị ghét: chương trình nhỏ gọn "Hello World" trong ngôn ngữ C#.

```
using System;
class helloworld;
{
    public static int Main()
    {
        Console.WriteLine("Hello Word");
        return 0;
    }
}
```

Như bạn thấy, các khai báo và câu lệnh khá giống C, và bạn có thể hiểu được dễ dàng nếu bạn đã biết lập trình C trước đây. Một chương trình đơn giản của C# là như vậy. Chúng ta sẽ tìm hiểu nó trong chương sau

8. CÁC ĐỐI TƯỢNG TỰ MÔ TẢ NHƯ THẾ NÀO?

Siêu dữ liệu (metadata) là thành phần chính yếu trong kiến trúc đóng gói của .NET Framework. Metadata dùng để mô tả các cấu trúc dữ liệu, bạn có thể xem metadata thuộc loại dữ liệu dùng mô tả dữ liệu. Nó dùng mô tả các lớp, phương thức, danh sách tham số, các thành phần dữ liệu trong file .EXE hoặc .DLL. Bằng mô tả metadata bất kỳ đối tượng nào cũng có thể đọc và hiểu được cấu trúc của đối tượng viết bằng ngôn ngữ khác. Ví dụ đối tượng C# hoàn toàn có thể đọc và sử dụng được các đối tượng viết bằng Visual Basic hay C++ .NET.

9. TƯƠNG TÁC VỚI THẾ GIỚI COM

Tuy .NET là mô hình mới thay thế cho thế giới COM đang có trong kiến trúc của Windows nhưng việc thay thế không diễn ra đến nỗi bạn mất hết khả năng sử dụng đối tượng COM. Nên .NET cho phép bạn sử dụng và gọi các đối tượng COM ngay từ các đối tượng .NET và tạo ra các lớp vỏ bọc cho phép đối tượng .NET có thể biến thành đối tượng COM.

10. WINDOWS FORMS, WEB CONTROLS VÀ GDI+

Nếu bạn đã từng viết chương trình Windows trước đây bạn hẳn đã quen với lớp thư viện đồ họa cực kỳ hấp dẫn. Trong .NET bạn có rất nhiều lựa chọn để lập trình giao diện đồ họa, sử dụng lớp Windows Forms, thư viện GDI+ hoặc các điều khiển Web Control để tiếp cận với giao diện dành cho ứng dụng Web. Bạn không còn phải mệt nhọc với các lớp MFC, thay vào đó là những lớp thư viện .NET đơn giản dễ hiểu và hiệu quả hơn rất nhiều. Hàng loạt công cụ lập trình hiệu quả được thêm vào môi trường Visual Studio .NET mà bạn không hề tìm thấy trong những môi trường lập trình trước đây.

Mọi thứ đã sẵn sàng, có lẽ đã đến lúc chúng ta đi sâu và tìm hiểu ngôn ngữ C# và đó cũng là mục đích của giáo trình này.

Chương 1.2

BỘ THỰC THI NGÔN NGỮ TỔNG QUÁT

(COMMON LANGUAGE RUNTIME)

Các vấn đề chính sẽ được đề cập đến

- ✓ Tổng quan về kiến trúc .NET và ngôn ngữ CLR
- ✓ Quan hệ của CLR trong .NET

1. TỔNG QUAN

Bộ thực thi ngôn ngữ tổng quát (Common Language Runtime – CLR) giữ nhiều vai trò trong hệ thống .NET. Nó là phần quan trọng trong hệ thống thực thi, nhưng nó cũng giữ nhiệm vụ tích cực trong việc phát triển và triển khai phần mềm chạy trên kiến trúc này. Nó cung cấp sự hỗ trợ đa ngôn ngữ bằng cách quản lý các trình biên dịch (compiler) được dùng để biến đổi mã nguồn thành ngôn ngữ trung gian (intermediate language – IL) và từ IL đến mã gốc (native code), và nó buộc chương trình phải thoả tính an toàn và bảo mật.

Chúng ta hãy xem xét những vai trò này chi tiết hơn.

1.1. Đơn giản hóa việc phát triển

Vai trò phát triển của CLR là quan trọng trong việc giảm gánh nặng mà chúng ta phải mang trong công việc của chúng ta từ ngày nay sang ngày khác. Dùng .NET, bạn không cần phải lo đến chi tiết thực chất vấn đề của GUIDS, IUnknown, IDispatch, hay các thư viện kiểu. CLR sẽ thay bạn chăm lo tất cả những thứ này.

Mã sẽ đáng tin cậy hơn vì việc quản lý bộ nhớ cũng là trách nhiệm của CLR. Các đối tượng được thể hiện dưới CLR không cần tính đến tham chiếu tường minh; dữ liệu mà bạn cấp phát được thu hồi lại bởi bộ dọn rác (Garbage Collector) khi mã lệnh của bạn không sử dụng nữa. Metadata (siêu dữ liệu – dữ liệu cơ sở) cũng cho phép liên kết động các chương trình khả thi. Điều này muốn nói rằng các trực trặc trong việc kết hợp các DLL quá hạn hoàn toàn không còn nữa. Độ tin cậy cũng được nâng cao qua tính an toàn kiểu. Bạn không cần xem xét đến những điều chưa biết như kích thước cấu trúc hay tổ chức các thành phần trong một đối tượng nào đó, nhờ thế bạn cũng không cần lo lắng gì về các sắp xếp hay hiệu chỉnh. Hơn thế, không bao giờ bạn cần lo về việc kiểm tra giới hạn của mảng (array) hay bộ đệm (buffer).

1.2. Công cụ hỗ trợ

CLR cộng tác chặt chẽ với các công cụ như Visual Studio, các trình biên dịch, các trình gỡ rối, và các trình tạo **profiles** để làm cho công việc của người phát triển dễ dàng hơn nhiều. Ngay cả bạn không cần Visual Studio .NET mà chỉ cần đến bộ SDK cho có được sự tích lũy kinh nghiệm. Một chương trình gỡ rối

được cung cấp bởi SDK cũng tốt như là trình đã được tích hợp vào VS.NET. Các công cụ khác, chẳng hạn như bộ dịch ngược ngôn ngữ trung gian (Intermediate Language Disassembler – ILDASM), tận dụng tiện lợi của các dịch vụ được cung cấp bởi CLR.

1.3. Hỗ trợ đa ngôn ngữ

Vấn đề cơ bản về hỗ trợ đa ngôn ngữ là hệ thống kiểu chung và metadata (siêu dữ liệu - dữ liệu cơ sở). Các kiểu dữ liệu cơ bản được sử dụng bởi CLR là phổ biến đối với tất cả các ngôn ngữ. Vì vậy không có các vấn đề chuyển đổi ở các kiểu số nguyên (integer), số chấm động (floating-point), và chuỗi (string). Tất cả các ngôn ngữ đều giải quyết tất cả các kiểu dữ liệu theo cùng một cách. Cũng có một cơ chế để định nghĩa và quản lý các kiểu mới.

Tất cả các ngôn ngữ cấp cao biên dịch thành IL. Một khi đã được biên dịch và metadata được tạo cho đối tượng, có thể truy xuất dễ dàng từ các ngôn ngữ khác. Điều này muốn nói rằng các đối tượng được viết bằng một ngôn ngữ có thể được kế thừa trong một ngôn ngữ thứ hai. Có thể viết một lớp bằng VB và kế thừa từ nó trong C#.

Đến nay, có khoảng 15 ngôn ngữ được hỗ trợ bởi kiến trúc .NET, kể cả C++, Visual Basic, C#, Perl, Python, Jscript, Pascal, COBOL, và Smalltalk.

1.4. Việc triển khai được thực hiện dễ hơn

Với kiến trúc .NET, không bao giờ cần phải đăng ký các component (thành phần) trong hệ thống. Đơn giản bạn có thể chép một component vào một thư mục, và CLR sẽ bảo đảm rằng nó được biên dịch và chạy đúng. Về phần mã được viết cho ASP.NET, bạn chỉ cần chép mã nguồn C# cơ bản, Jscript, hay VB, và trình biên dịch IL sẽ được thực thi trên mã nguồn đúng đó.

Các component có thể triển khai hay cài đặt vào 2 nơi. Nơi thứ nhất là thư mục mà ứng dụng cư trú. Điều này cho phép các phiên bản khác nhau của ứng dụng cùng tồn tại trên cùng một máy. Nơi thứ hai là Global Assembly Cache. Đây là kho lưu trữ trung tâm cho các hợp ngữ (assembly) mà tất cả các ứng dụng của .NET có thể truy xuất đến.

Cho dù các component được lưu trữ bất cứ nơi đâu, thủ tục cập nhật rất đơn giản. Bạn chỉ chép hợp ngữ (assembly) vào đúng thư mục, dù là đối tượng đó đang chạy (bạn không thể làm điều này với một DLL của Windows), và hệ thống sẽ khiến nó hoạt động.

1.5. Sự tách biệt các phần mềm

Các ứng dụng bị cô lập với một ứng dụng khác bởi CLR để chúng không thể ảnh hưởng lẫn nhau. Các tài nguyên có thể được dùng chung, nhưng việc dùng chung được tạo ra một cách rõ ràng. Điều này muốn nói rằng một hệ thống .NET có thể có cách tiếp cận tính bảo mật của hệ thống nếu cần. Khi chế độ bảo mật được thả lỏng hơn, hay khi muốn truy xuất vào bất kỳ phần nào của tài nguyên hệ

thống, ta có thể điều chỉnh việc cho phép truy xuất các tập tin hay các tài nguyên hệ thống khác chỉ dựa trên cơ sở ứng dụng.

1.6. Tính an toàn kiểu và sự kiểm tra

CLR thực thi tính bảo mật và tin cậy qua an toàn kiểu, sự kiểm tra và ủy thác. Mã lệnh có tính an toàn kiểu được xây dựng phù hợp với các qui ước mà CLR đưa ra. Nó chỉ dùng các kiểu được nhận dạng, chỉ dùng bộ nhớ được cấp phát bởi trình quản lý bộ nhớ (memory manager) và không thể truy xuất vào dữ liệu cục bộ hay các phương thức của các ứng dụng khác hay các tiến trình (process) khác. Một chương trình an toàn kiểu không thể rơi vào tình trạng vi phạm không gian và ghi đè lên các khối bộ nhớ của các tiến trình khác. Thử nghiệm an toàn kiểu xảy ra khi mã lệnh được nạp lên và được thực thi bởi CLR. Trong suốt tiến trình thông dịch mỗi phương thức, CLR xem xét các đặc tả metadata (đặc tả dữ liệu) của phương thức và kiểm tra tính an toàn kiểu của nó. Có một vài tình huống mà việc kiểm tra này là không thể, chẳng hạn, khi CLR cần dùng mã lệnh bên ngoài .NET từ một đối tượng của .NET. Đây là trường hợp thông thường khi một lớp .NET gọi một Win32 DLL. Trong trường hợp này mã lệnh phải được ủy thác (must be trusted) bởi CLR.

1.7. Tính bảo mật

Vấn đề bảo mật của việc quan trắc các hệ thống mà có thể thực thi các nội dung hoạt động hay thực hiện các script của người dùng là rất cao. Với tư cách là một client bạn không cho phép một chương trình không rõ nguồn gốc chạy trên máy bạn, hủy các dữ liệu có giá trị. Với tư cách là một người cung cấp dịch vụ bạn không cho phép các cuộc tấn công hay các lỗi đơn giản mà người dùng mang lại cho toàn bộ hệ thống của bạn.

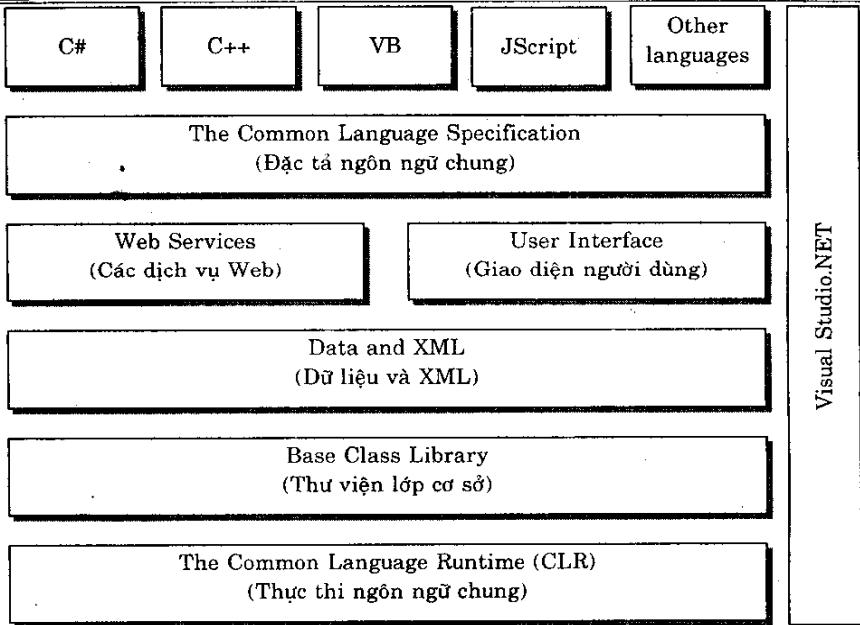
CLR kiểm soát tính bảo mật của hệ thống qua một cặp thông tin của người dùng là user và mã nhận dạng để kiểm tra. Việc nhận dạng mã, ví dụ như người tạo ra và nguồn gốc của mã lệnh, có thể được biết và việc cho phép sử dụng các tài nguyên sẽ được gán tương ứng. Kiểu bảo mật này được gọi là Evidence Based Security (Bảo mật dựa trên bằng chứng) và là một đặc tính chính của .NET. Kiến trúc .NET cũng cung cấp việc hỗ trợ cho việc bảo mật dựa trên vai trò (role-based) dùng trong các account và group của Windows NT.

2. QUAN HỆ CỦA CLR VỚI .NET

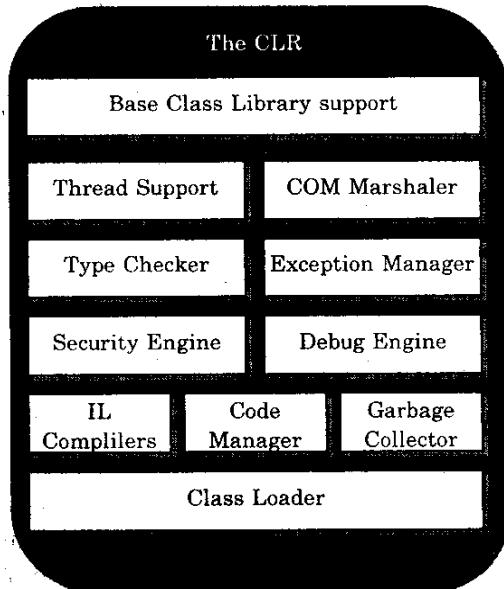
Sơ đồ khối được trình bày trong hình 1.2-1 cho thấy mối quan hệ của CLR với kiến trúc .NET.

2.1. Chi tiết về CLR

Như là một thành phần con của kiến trúc .NET, bản thân CLR được xây dựng từ các phần riêng biệt. Hình 1.2-2 cho thấy các phần này quan hệ với nhau như thế nào.



Hình 1.2-1 : CLR và kiến trúc .NET



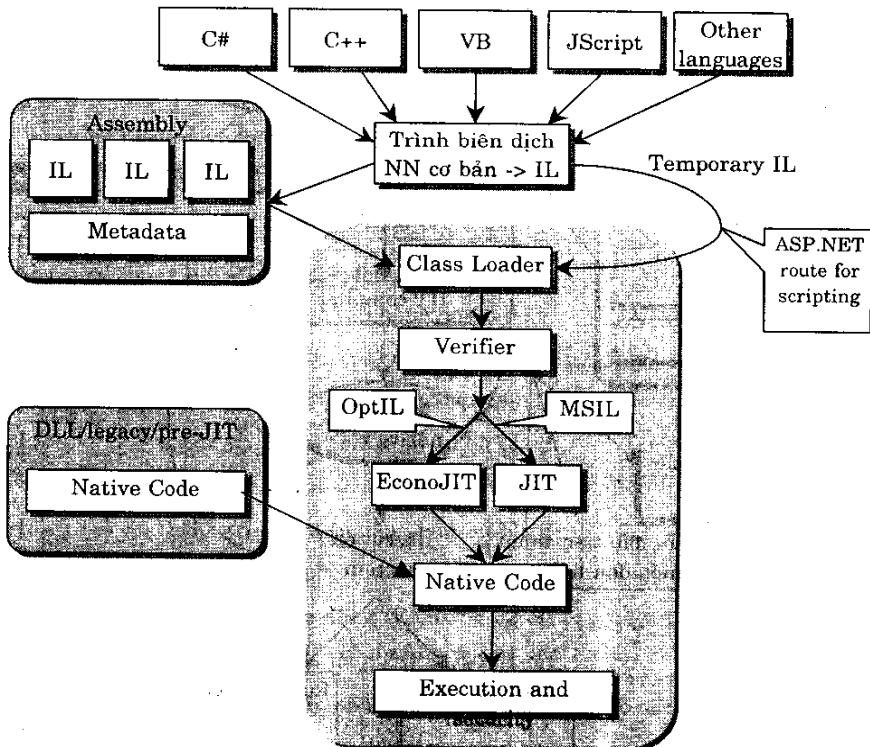
Hình 1.2-2 : Chi tiết các thành phần của CLR

Như bạn có thể thấy trong hình 1.2-2, chức năng chính của CLR như là một bộ nạp lớp (Class loader). Nó nạp các lớp mà bạn tạo hay các lớp mà được cung cấp như là bộ phận của thư viện các lớp cơ sở, chuẩn bị chúng cho việc sử dụng, và sau đó hoặc là thực thi chúng hoặc là tham gia vào lúc thiết kế (design-time).

Các lớp được nạp từ các hợp ngữ (assembly), .NET tương đương với DLL hay EXE. Đôi khi hợp ngữ chứa mã gốc nhưng hầu như chắc chắn là chứa các lớp đã được biên dịch thành IL và metadata liên quan đến chúng. Trong thời gian thiết kế, CLR giao tiếp với các công cụ như Visual Studio.NET để cung cấp kinh nghiệm về phát triển ứng dụng trong thời gian ngắn (Rapid Application Development – RAD) mà các nhà viết chương trình VB đã dùng trong thời gian quá lâu, mà còn cho tất cả các module trong tất cả các ngôn ngữ. Còn trong lúc thực thi, bộ nạp lớp sẽ tải các lớp.

2.2. CLR vào lúc thực thi

Khi bộ nạp lớp (class loader) mở một gói chương trình trong lúc thực thi, nó có một số bước quan trọng để thực hiện trước khi các lớp có thể thật sự được chạy. Hình 1.2-3 mô phỏng mô hình thực hiện cho CLR.

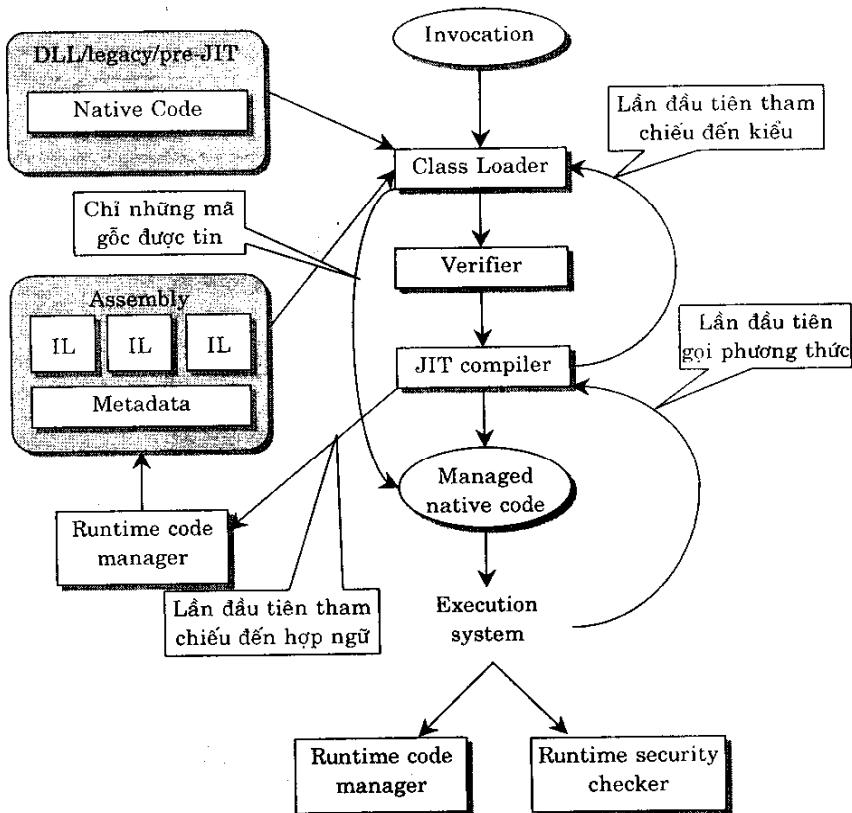


Hình 1.2-3 : Mô hình thực thi của CLR

Bộ nạp lớp (class loader) dùng bộ quản lý mã (Code Manager) để cấp bộ nhớ cho các đối tượng và dữ liệu. Bố trí các lớp trong bộ nhớ được tính, và mỗi phương thức đã được lưu trữ như ngôn ngữ trung gian sẽ được coi như là một stub dùng để gọi trình biên dịch (compiler) cho lần thực thi đầu tiên.

CHÚ Ý Bạn sẽ thấy mô hình trong hình 1.2-3 có hai kiểu ngôn ngữ trung gian khác biệt, MSIL và OptIL, được biên dịch bởi các trình biên dịch JIT của riêng chúng. OptIL là một tập con được tối ưu hóa cao của MSIL, được thiết kế đặc biệt để dùng bởi hệ thống chủ mà không có sự lãng phí bộ nhớ hay tốc độ tính toán. Vì công việc đặc biệt này mà trình biên dịch của ngôn ngữ cấp cao cố gắng tối ưu hóa, OptIL thì thích hợp hơn với các PDA, điện thoại thông minh, và những thứ tương tự. EconoJIT là trình biên dịch gọn nhẹ để dịch OptIL thành mã máy.

Bất cứ khi nào một hàm tham chiếu lần đầu một lớp ở trong một hợp ngữ khác hay một kiểu dữ liệu khác mà không được dùng trước đó, bộ nạp lớp (class loader) được gọi lại để nạp các đối tượng cần thiết. Bạn hãy nhìn vào lưu đồ để hiểu rõ tiến trình hơn. Hệ thống thực thi ảo của CLR (Virtual Execution System – VES) và các con đường điều khiển chính được trình bày trong hình 1.2-4.



Hình 1.2-4 : Hệ thống thực thi ảo (Virtual Execution System)

2.2.1. Bộ tạo mã lúc cài đặt

Một phần VES không được trình bày trong hình 1.2-4 là bộ tạo mã lúc cài đặt mà thỉnh thoảng xuất hiện khi các hợp ngữ được đặt lên máy chủ lần đầu tiên. Lúc này sẽ tạo ra mã gốc (native code) trong hợp ngữ để mà giảm thiểu thời gian khởi động một đối tượng cụ thể. Thông thường, việc thẩm định mã này được thực hiện lúc cài đặt, nhưng nếu một phiên bản thực thi hay việc kiểm tra tính bảo mật là cần thiết, thì mã lệnh sẽ được biên dịch lại bởi trình biên dịch JIT chuẩn theo cách bình thường.

2.3. Các kiểu dữ liệu được cung cấp bởi CLR

Như đã đề cập trước đó, kiến trúc .NET có một tập các kiểu dữ liệu cơ sở được dùng bởi tất cả các ngôn ngữ khác. Các kiểu này bao gồm các kiểu số nguyên như số nguyên 8-bit, 16-bit, 32-bit, và 64-bit và char (ký tự). Và kể cả các kiểu chấm động (floating – point) và các kiểu con trỏ đến cá vùng nhớ của .NET lẫn vùng nhớ bên ngoài .NET.

Tất cả các kiểu dữ liệu cơ sở có thể được sử dụng bởi mã lệnh viết bằng IL, và bạn có thể tự viết IL. IL rất giống với hợp ngữ (assembly language), nên nếu bạn đã từng làm việc và sử dụng Z80 hay 6502, bạn sẽ cảm thấy thoải mái với IL. Ở phần sau, chúng tôi sẽ trình bày với bạn một vài IL được viết và các công cụ đi với kiến trúc .NET mà có thể được dùng để biên dịch và dịch ngược lại.

Các kiểu cơ sở được định nghĩa trong bảng 1.2-1

Bảng 1.2-1 Các kiểu dữ liệu cơ sở của CLR

Tên kiểu	Mô tả
I1	Giá trị 8-bit có dấu phần bù của 2
U1	Giá trị nhị phân không dấu 8-bit
I2	Giá trị 16-bit có dấu phần bù của 2
U2	Giá trị nhị phân không dấu 16-bit
I4	Giá trị 32-bit có dấu phần bù của 2
U4	Giá trị nhị phân không dấu 32-bit
I8	Giá trị 64-bit có dấu phần bù của 2
U8	Giá trị nhị phân không dấu 64-bit
R4	Giá trị chấm động IEEE 754 32-bit
R8	Giá trị chấm động IEEE 754 64-bit
I	Giá trị có dấu phần bù của 2 có kích thước tự nhiên
U	Giá trị nhị phân không dấu có kích thước tự nhiên, cũng là con trỏ (pointer) của COM

R4Result	Kích thước tự nhiên cho kết quả tính toán chấm động 32-bit
R8Result	Kích thước tự nhiên cho kết quả tính toán chấm động 64-bit
Rprecise	Giá trị chấm động có độ chính xác cao
0	Đối tượng có kích thước tự nhiên tham chiếu đến vùng nhớ của .NET
&	Con trỏ có kích thước tự nhiên (có thể chỉ đến vùng nhớ của .NET)

Bảng 1.2-1 đề cập đến các kích thước “tự nhiên” cho các kiểu dữ liệu. Điều này nói đến các kích thước mà được ấn định sẵn bởi phần cứng. Chẳng hạn như, một máy có đường truyền 16-bit có thể sẽ có số nguyên tự nhiên là 16-bit, và một máy có đường truyền 32-bit sẽ có số nguyên lớn phù hợp. Điều này có thể là một vấn đề khi các chương trình phải nói chuyện với phần mềm ở đầu kia của một kết nối internet. Không có chỉ thị rõ ràng, thật là khó để biết một số nguyên ở đầu kia lớn như thế nào, và việc không tương thích về kích thước kiểu, và không liên kết về cấu trúc có thể xảy ra. Để biết điều này và các vấn đề tương tự khác, CLR chỉ dùng một vài kiểu dữ liệu dành riêng khi ước lượng và chạy phần mềm thực sự. Kiểu cơ sở mà không vừa với các kích thước ít được chọn thì được đóng gói và mở gói một cách cẩn thận thành những kiểu dữ liệu khả chuyển hơn. Điều này diễn ra ở bên dưới và bạn không phải lo lắng gì.

2.4. Dữ liệu và mã lệnh của .NET

CLR bao gồm bộ quản lý mã (Code Manager) và bộ dọn dẹp rác (Garbage Collector – GC). Các khôi phục năng này chịu trách nhiệm quản lý bộ nhớ của tất cả các đối tượng cơ sở của .NET, kể cả các kiểu dữ liệu được quản lý. Code Manager cấp phát không gian lưu trữ; Garbage Collector xoá nó và làm gọn heap để giảm thiểu các phân đoạn.

Vì GC chịu trách nhiệm làm gọn heap (vùng nhớ của chương trình), dữ liệu hay các đối tượng được sử dụng trong không gian được kiểm soát bởi .NET thường bị di chuyển từ nơi này sang nơi khác trong suốt thời gian sống của chúng. Đó là tại sao, trong các tình huống thông thường, các đối tượng luôn luôn được truy xuất bởi tham chiếu (reference). Trong thế giới C++, tham chiếu thì không khác với con trỏ (pointer). Cơ chế được dùng để tìm đối tượng được tham chiếu đến cũng giống như một con trỏ chỉ đến một khối bộ nhớ nào đó. Các tham chiếu C++ là cách nói của trình biên dịch mà nó chắc rằng bạn đang chỉ đến một kiểu cụ thể. Trong C++ có thể lưu một tham chiếu đến một đối tượng, hủy đối tượng đó, và sau đó sử dụng chính tham chiếu lỗi thời đó để truy xuất đến vùng đã dùng cho đối tượng. Điều này thường gây ra hư hỏng trầm trọng trong chương trình C++.

Tuy nhiên dưới hệ thống .NET, vấn đề này bị loại trừ. Bất cứ khi nào một tham chiếu đến một đối tượng được tạo, đối tượng đó biết nó đang được tham chiếu đến, và GC sẽ không bao giờ hủy nó trừ phi nó giải phóng gánh nặng. Hơn nữa, khi GC dọn dẹp heap (vùng nhớ của chương trình) và di chuyển các đối tượng

trong bộ nhớ, tất cả các tham chiếu đến đối tượng đó được cập nhật một cách tự động. Điều này khiến các component (thành phần) đang dùng đối tượng đó sẽ nhận được lấy tham chiếu đúng khi các thành phần này truy xuất đến đối tượng ở lần kế.

Qua Code Manager, CLR chịu trách nhiệm bố trí bộ nhớ thực của các đối tượng và các cấu trúc mà nó làm chủ. Thường tiến trình bố trí này là tự động, nhưng khi bạn cần làm thì bạn có thể chỉ ra thứ tự, sự sắp xếp, và định rõ việc bố trí bộ nhớ trong metadata. Rõ ràng là điều này có thể được thực hiện vì metadata thì luôn có sẵn đối với lập trình viên thông qua tập các interface (giao diện) đã được định nghĩa.

2.5. Mã lệnh của COM và việc truy xuất dữ liệu

Có lẽ không rõ ràng là tính an toàn kiểu, tính bảo mật, và việc kiểm tra của .NET cho phép bạn bảo vệ như thế nào sự đầu tư của mình trong mã C++ mà từ lâu nay bạn đã quá chăm chút cho nó.

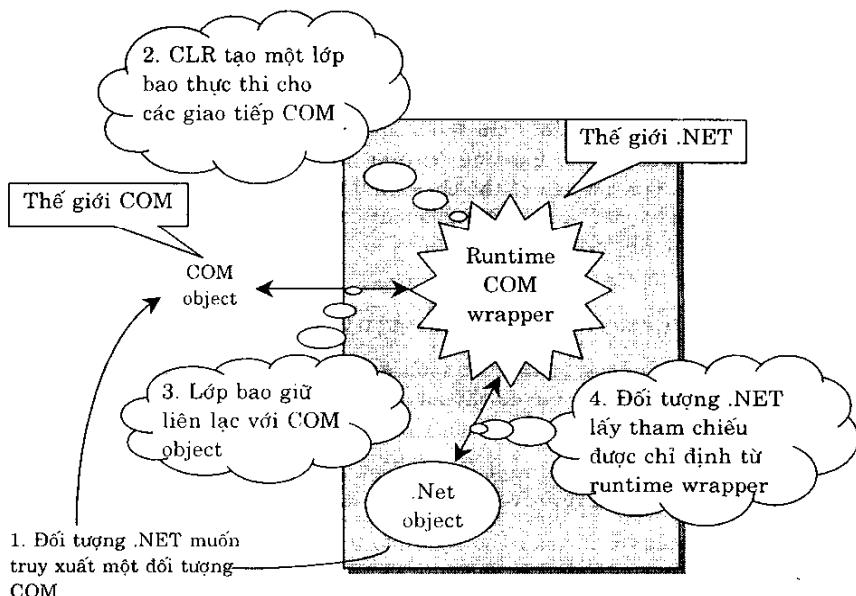
Như là một hệ quả, Microsoft ở trong cùng cảnh ngộ đúng như vậy và kiến trúc .NET có các khả năng tuyệt vời cho việc sử dụng lại mã có trước đây của bạn.

Có ba cơ chế cơ bản dành cho thao tác liên kết giữa .NET và COM trong .NET. COM Interop cho phép các đối tượng COM của bạn được sử dụng bởi kiến trúc .NET y như thế chúng là các đối tượng của .NET. Cơ chế gọi hệ thống (Platform Invoke - P/Invoke) để các đối tượng của .NET gọi các điểm vào tĩnh (static entry) trong các DLL giống như LoadLibrary và GetProcAddress. Cuối cùng, IJW (It Just Work – Làm việc ngay) trong hầu hết các hình thức cơ bản của nó cho phép bạn biên dịch lại mã của bạn và ... nó làm việc ngay. Một hình thức phức tạp hơn cho phép bạn nâng cấp mã lệnh của bạn trong việc sử dụng các phân mảng rộng của .NET đối với C++. Điều này cho phép bạn tạo đầy đủ các đối tượng GC trong vùng nhớ của .NET để mã nguồn C++ cũ của bạn sử dụng.

2.5.1. Cơ chế gọi các đối tượng COM (COM Interop) qua CLR

Bạn có thể thấy trong hình 1.2-2 rằng CLR chứa một bộ sắp đặt COM (COM marshaler). Khối chức năng này chịu trách nhiệm về việc gọi các đối tượng COM (COM Interop) trên .NET. Có hai kịch bản cần phải gọi các đối tượng COM trên kiến trúc .NET. Kịch bản thứ nhất là khi bạn muốn truy xuất các đối tượng COM cũ từ mã lệnh của C# hay VB mà bạn viết. Kịch bản thứ hai là khi bạn muốn thực hiện một giao diện (interface) thông dụng mà có các đối tượng COM của bạn truy xuất đến.

Đối với cả hai kịch bản này, CLR đóng vai trò rất năng động bằng cách tạo một lớp bao (wrapper) riêng biệt vào lúc thực thi và sau đó lắp đặt dữ liệu xuất nhập của đối tượng COM khi cần. Sơ đồ trong hình 1.2-5 cho thấy một đối tượng COM đang được truy xuất bởi một đối tượng client .NET.



Hình 1.2-5 : Mã được quản lý với tiến trình gọi đối tượng COM

Để sử dụng các đối tượng COM với các chương trình .NET của bạn, bạn cần phải đưa vào (import) định nghĩa thư viện kiểu (type library). VS.NET sẽ làm điều này cho bạn một cách tự động khi bạn thêm một tham chiếu vào đối tượng COM trong IDE, theo cùng một cách mà các lập trình viên VB đã làm trong nhiều năm. Bạn cũng có thể sử dụng công cụ TLBImp (Type Library Import) một cách tương tự. TLBImp sẽ chuyển các tên phương thức trong COM thành các tên phương thức trong .NET cho bạn. Ví dụ, tên phương thức trong COM

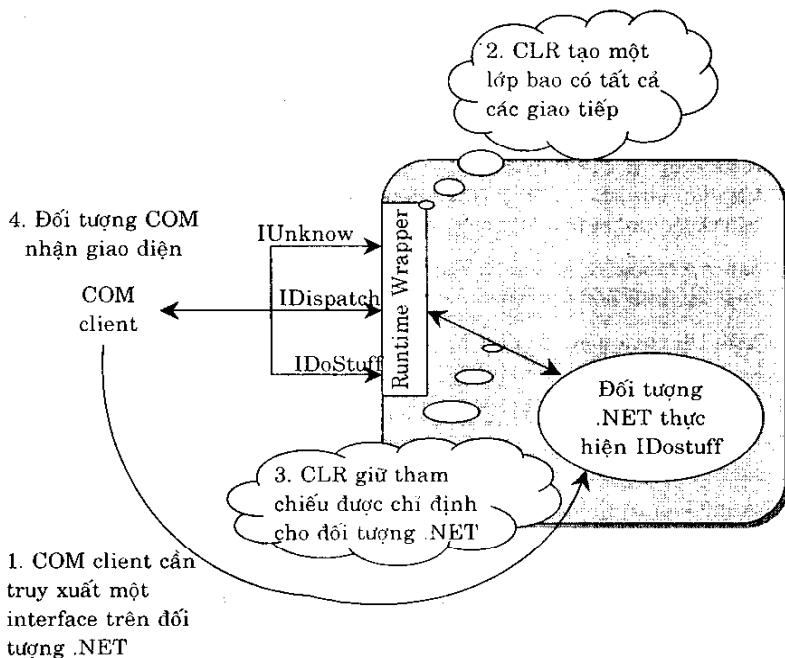
```
HRESULT MyMethod(BSTR b, int n, [out, retval]int *retval);
```

Được dịch thành

```
Int MyMethod(String b, int n);
```

Kết quả là, khi bạn gọi phương thức bạn không phải lo lắng về **HRESULT**, đơn giản bạn có thể gán giá trị nguyên trả về vào một biến trong mã lệnh của bạn. Thực tế, toàn bộ hình ảnh tiến trình .NET-to-COM là rất dễ sử dụng. Bạn không phải lo việc chuyển đổi dữ liệu. Như bạn có thể thấy trong ví dụ, các kiểu dữ liệu COM chẳng hạn như **BSTR** ánh xạ vào các kiểu tương đương dễ nhận thấy của .NET. Bạn không phải quản lý việc đếm tham chiếu và không có GUIDS. Nếu bạn nhận giá trị **HRESULT** thất bại, lớp wrapper đang thực thi sẽ tạo một exception (ngoại lệ) mà bạn có thể nắm bắt.

Khi bạn muốn gọi một đối tượng .NET từ các đối tượng COM, tiến trình mà CLR dùng để làm cho việc kết nối dễ dàng là tương tự với sơ đồ được đưa ra trong hình 1.2-5. Hình 1.2-6 mô phỏng việc kết nối COM-to-NET.



Hình 1.2-6 Truy xuất một đối tượng .NET từ COM

Các đối tượng COM chỉ có thể truy xuất các phương thức toàn cục của các đối tượng .NET của bạn, nhưng chúng có thể được sử dụng từ thế giới COM.

Đầu tiên, tạo một typelib từ đối tượng của bạn bằng cách sử dụng công cụ TlbExp của .NET. Kế đến, đăng ký hợp ngữ được tạo bởi TlbExp bằng công cụ RegAsm. Đây là lần duy nhất bạn sẽ cần đăng ký cái gì đó dưới .NET; COM đang chạy trong phần chưa được quản lý trên máy chủ cần phải kích hoạt đối tượng này theo cách chuẩn. Cuối cùng, từ COM bạn có thể sử dụng CoCreateInstance và QueryInterface, sử dụng AddRef và Release như thường, và sử dụng HRESULT mà lớp wrapper cung cấp.

Sau này, trong phần 5.3, chúng ta có một chương dành cho COM Interop, ta sẽ đi sâu hơn và có nhiều ví dụ hơn.

2.5.2. Lời gọi hệ thống (Platform Invoke – P/Invoke) từ CLR

Một trong những yêu cầu chung nhất đối với hoạt động liên kết sẽ là việc sử dụng một DLL mà đã tồn tại trên hệ thống cục bộ. Chẳng hạn như, bạn có thể cần phải tiếp tục sử dụng các công cụ mà bạn đã đầu tư vào và chỉ có sẵn ở dạng DLL.

Kiến trúc .NET cung cấp cơ chế P/Invoke cho mục đích này. Cơ chế nội tại thì rất giống với COM Interop, nhưng vì các DLL mà bạn dùng thì luôn luôn ở cục bộ (trên cùng một máy), hệ thống dễ sử dụng hơn.

Để sử dụng cơ chế P/Invoke, bạn cần một lớp wrapper để ủy quyền các lời gọi từ CLR đến hàm thực sự của DLL. Bạn có thể tạo các wrapper này bằng cách sử dụng các thuộc tính đặc biệt trong các định nghĩa phương thức của bạn.

Đoạn mã trong ví dụ sau sẽ cho phép bạn gọi một hàm trong một DLL tùy biến.

```
MyDll.Dll chứa một hàm int Myfunction(int x);
Public class MyDllWrapper
{
    [DllImport("MyDll.Dll", EntryPoint = "MyFunction")]
    public static extern int Myfunction(int x);
}
```

Bây giờ bạn có thể truy xuất DLL sử dụng các lệnh của C#:

```
Int r = MyDllWrapper.MyFunction(12);
```

Trong chương 1.4, “Làm việc với các phần mở rộng .NET và C++”, chúng ta sẽ khảo sát cơ chế P/Invoke chi tiết hơn khi chúng ta xem xét các phần mở rộng được quản lý đối với C++.

2.5.3. IJW (It just works – Nó hoạt động ngay)

Cách đơn giản nhất để đem các ứng dụng C++ của bạn vào kiến trúc .NET là phái tập hợp tất cả các mã nguồn EXE hay DLL của bạn. Biên dịch lại tất cả chúng với tuỳ chọn /CLR của trình biên dịch và chạy chúng. Microsoft nói “Nó hoạt động ngay.”. Có lẽ điều này đúng nếu bạn đã và đang lập trình theo phong cách hiện đại. Nếu mã của bạn vẫn còn chứa các khai báo kiểu K&R lâu đời, thì nó sẽ không hoạt động.

Có một vài giới hạn đối với tiến trình này. Bạn không thể biên dịch lại tất cả các thư viện lớp của bạn và sau đó kế thừa từ chúng trong nơi đã được kiểm soát. Bạn không thể truyền một cách dễ dàng các con trỏ của .NET vào các hàm và các ứng dụng của bạn.

Cả tiến trình IJW lẫn P/Invoke đều kéo dài cuộc sống các công cụ của bạn và cho bạn một cơ hội để đem vào kiến trúc .NET.

2.6. Các thành phần mở rộng của .NET đối với C++

Thường thì khi sử dụng lại, mã lệnh cũ của bạn là không đủ, và bạn cần một cách tiếp cận hợp nhất hơn từ viễn cảnh của lập trình viên C++, các thành

phần mở rộng của .NET đối với C++ là giải pháp lý tưởng hoặc là một con đường đi vào .NET.

Kiến trúc .NET thêm vào một vài khái niệm quan trọng để quản lý bộ nhớ và việc sử dụng dữ liệu. Tất cả các đặc tính này thì có sẵn trong ngôn ngữ C# và VB nhưng không có sẵn trong C++. Những đặc tính bổ sung này được thảo luận trong các phần sau.

2.6.1. Các lớp đơn dẹp rác (GC)

Các lớp GC là các đối tượng để mà tất cả việc quản lý bộ nhớ được thực hiện bởi CLR. Một khi bạn ngưng sử dụng một lớp GC, nó được đánh dấu một cách tự động để hủy và được thu hồi bởi heap của GC. Một lớp GC cũng có thể có phương thức hủy có thể được gọi một cách tường minh.

Bạn có thể tạo các lớp GC bằng cách sử dụng từ khoá mới `_gc` trên các lớp của bạn:

```
_gc class MyClass
{
    public:
    int m_x;
    int m_y;
    SetXY(int x, int y)
    {
        m_x = x;
        m_y = y;
    }
}
```

Bây giờ MyClass là một lớp GC.

CẢNH BÁO Nếu bạn sử dụng `asm` hay `setjmp` trong các phương thức C++ của bạn, trình biên dịch sẽ đưa ra các khuyến cáo; thử chạy mã lệnh này có thể dẫn đến hư hỏng nếu phương thức này sử dụng bất kỳ kiểu hay mã của .NET. Các lớp GC của bạn có thể chỉ dùng sự kế thừa duy nhất. Chúng không thể có một copy constructor (phương thức khởi dụng chồng) và không thể kế thừa chồng các toán tử (operator) & và `new`.

2.6.2. Các lớp giá trị

Từ khoá mới `_value` cho phép bạn tạo các kiểu giá trị trong mã lệnh C++. Chẳng hạn, bạn có thể cần tạo trên heap (vùng nhớ của chương trình) một cấu trúc `3D_Point` có các thành phần x,y,z được dùng như là một kiểu giá trị. Các kiểu này được tạo trên heap nhưng có thể được đóng hay được gói theo một kiểu bao đối tượng, sử dụng từ khoá `_box` để dùng chúng như các đối tượng của .NET. (Có một thảo luận bao quát về bao (boxing) các đối tượng và không bao (unboxing) các đối tượng trong Phần II của quyển sách này, "The C# Language") Khi chúng được bao, các kiểu giá trị này có thể được sử dụng bởi mã lệnh của .NET, được giữ trong các lớp tập hợp của .NET, được serialize thường là với XML, và vân vân.



Các lớp và các cấu trúc với từ khoá `_value` kế thừa từ lớp `System.ValueType` của kiến trúc .NET và có thể định nghĩa chồng bất kỳ phương thức nào mà đối tượng đó cần có. Đặt các phương thức virtual trên lớp giá trị của bạn là không được phép. Chẳng hạn, bạn có thể định nghĩa chồng phương thức `System.ValueType.ToString` để tạo một chuỗi để mô tả đối tượng. Ví dụ như, `int32.ToString()` trả về số ở dạng chuỗi. Như `itoa(...)`. Bạn cũng có thể muốn định nghĩa chồng phương thức `System.ValueType.GetHashCode` để sử dụng trong một tập hợp ánh xạ.

Các lớp giá trị này hoạt động như các lớp của C++ khi được sử dụng trong C++ nhưng cũng có thể được sử dụng bởi các chương trình thực thi của .NET. Chúng không được cấp phát trực tiếp từ heap (vùng nhớ của chương trình) của .NET, nhưng chúng có thể được sử dụng bởi các đối tượng của .NET khi được bao lì.

Có một vài qui tắc để áp dụng các lớp này. Bạn có thể nhận lớp giá trị từ một hay nhiều interfaces (giao tiếp) của .NET. Nhưng bạn không thể định nghĩa chồng một constructor (khởi động) cho nó. Một lớp giá trị không thể có từ khoá `_abstract` đi kèm. Các lớp giá trị được “niêm phong”, có nghĩa là chúng không thể được dẫn xuất. Bạn không thể khai báo một con trỏ đến một lớp giá trị.

Như với từ khoá `_gc`, bạn không nên dùng `_asm` hay `setjmp` nếu lớp đó sử dụng bất kỳ mã lệnh nào của .NET, chấp nhận các kiểu của .NET như các tham số, hay trả về một kiểu của .NET. Nó có thể biên dịch, nhưng nó sẽ có thể không hoạt động tốt.

2.6.3. Các thuộc tính (property)

Các lớp C# trong .NET có thể có các thuộc tính có các phương thức truy xuất get và set. Từ khoá `_property` cho phép bạn tạo các lớp GC để hỗ trợ các thuộc tính. Một thuộc tính giống như một thành phần dữ liệu trong lớp của bạn nhưng là một đoạn mã thật sự. Ví dụ dưới cho thấy một thuộc tính sẽ được tạo vì sử dụng như thế nào.

```
#using "mscorlib.dll"
_gc class My3Dpoint
{
    // mặc định các thành phần là cục bộ
    int m_x;
    int m_y;
    int m_z;
    public:
        _property int get_x(){return m_x;};
        _property void set_x(int value){m_x=value;};
        _property int get_y(){return m_y;};
        _property void set_y(int value){m_y=value;};
        _property int get_z(){return m_z;};
        _property void set_z(int value){m_z=value;};
    // các phương thức khác của 3D ở đây
};
```

```
void main(void)
```

```

{
    My3Dpoint *pP=new My3Dpoint();
    pP->x=10; // gọi set_x(10);
    int X = pP->x; // gọi get_x();
}

```

2.6.4. Ghim (pin) các giá trị lại

Heap của GC sẽ thường xuyên di chuyển các đối tượng từ chỗ này đến chỗ khác trong suốt quá trình hoạt động của nó. Bất cứ khi nào bạn cần truyền một con trỏ đến một đối tượng của .NET hay một hàm C++ của COM, bạn có thể ghim các đối tượng bằng việc sử dụng từ khoá `_pin`. Thao tác ghim (pin) này ngăn cấm GC di chuyển đối tượng đó cho đến khi nó không bị ghim nữa.

2.7. Các ngoại lệ (Exception)

Tất cả các ngoại lệ trong kiến trúc .NET được xử lý bởi CLR. CLR cung cấp một cơ chế mạnh và nhất quán trong việc theo vết và xử lý lỗi bất cứ khi nào chúng có thể xảy ra.

Các ngoại lệ trong CLR dùng các đối tượng exception, thường được dẫn xuất từ lớp `System.Exception` trong kiến trúc .NET và một trong bốn bộ xử lý ngoại lệ. Các bộ xử lý này là

- Bộ xử lý `finally` được thực hiện bất cứ khi nào một khối thoát ra. Bộ xử lý này được gọi trong lúc kết thúc bình thường một đối tượng cũng như khi một lỗi xảy ra.
- Bộ xử lý `fault` chạy khi một ngoại lệ đúng xảy ra.
- Bộ xử lý `type-filtered` phục vụ cho các ngoại lệ từ một lớp cụ thể hay các lớp dẫn xuất của nó. Ví dụ: `catch(MyExceptionType e)`
- Bộ xử lý `user-filtered` có thể phát hiện ngoại lệ đó có nên bỏ qua hay không, được xử lý bởi bộ xử lý được liên kết, hay được truyền đến ngoại lệ kế tiếp có hiệu lực.

Mỗi một phương thức của mỗi lớp trong các thư viện trong kiến trúc .NET hay trong mã lệnh của bạn có một bảng các bộ xử lý ngoại lệ kết hợp với nó. Các phần tử trong bảng mô tả một khối mã lệnh được bảo vệ và bộ xử lý ngoại lệ liên kết với nó. Có thể hoặc là không có bộ xử lý nào, bộ xử lý `catch`, bộ xử lý `finally`, hoặc là bộ xử lý `fault` cho mỗi phần tử của bảng. Khi một ngoại lệ xảy ra, CLR tìm một ngoại lệ trong bảng phương thức để đưa cho bộ xử lý. Nếu một ngoại lệ được tìm thấy, quyền điều khiển được truyền đến bộ xử lý như thường; nếu không, CLR tiếp tục qua ngăn xếp (stack) di đến phương thức đang gọi và vân vân, giữ lại chuỗi mốc xích các lời gọi cho đến khi một bộ xử lý được tìm thấy. Nếu một ngoại lệ không được tìm thấy, CLR sẽ kết thúc ứng dụng và tiến hành hiển thị ngăn xếp.

Khi chúng ta đi xuyên suốt quyển sách này, các ví dụ sẽ đưa nhiều thông tin chi tiết hơn về cách tạo các lớp ngoại lệ và các bộ xử lý.



2.8. Hỗ trợ việc gỡ rối

Kiến trúc .NET có trình gỡ rối được xây dựng trong nó ở mức rất thấp. Không như các hệ thống gỡ rối khác mà các lập trình viên C++ quen sử dụng trước kia, thường là xen vào, CLR thực hiện tất cả các mã lệnh, nó quản lý việc gỡ rối và gửi các sự kiện đến bộ gỡ rối được kết nối bất cứ khi nào cần.

SDK của .NET có các trình gỡ rối của riêng nó, thêm vào đó là trình được tích hợp với Visual Studio.NET. Tất cả các trình gỡ rối hoạt động qua các API chung được cung cấp bởi kiến trúc.

3. KẾT CHƯƠNG

CLR là trái tim của hệ thống .NET. Nó nạp và quản lý các mã lệnh, tương tác với các công cụ phát triển và các trình gỡ rối cho bạn, tuân thủ tính an toàn và bảo mật của hệ thống của bạn, và cho phép phần mềm của bạn tích hợp với các hệ thống có sẵn trước đó như COM và các DLL của người dùng. Là một thành phần của .NET CLR không chắc rằng bạn nghĩ nhiều về nó chút nào.

Chương 1.3

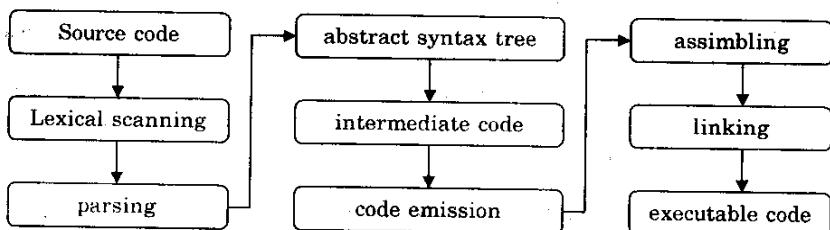
NGÔN NGỮ TRUNG GIAN IL

Các vấn đề chính sẽ được đề cập đến

- ✓ *Liên kết (Inter-Op) Ngôn ngữ*
- ✓ *Hello IL*
- ✓ *Hàm (functions)*
- ✓ *Lớp (classes)*
- ✓ *ILDASM*
- ✓ *Metadata*
- ✓ *Reflection API*

Vào thời đại của sự phát triển phần mềm ngày nay, những người phát triển gần như chỉ sử dụng ngôn ngữ cấp cao như C++, Visual Basic, hay Java để viết các ứng dụng và các thành phần (components). Có lẽ còn rất ít người hiểu được công việc những người lập trình xuyên phiêu (binary developers) và cũng có lẽ chỉ còn dăm ba người lập trình hợp ngữ.

Vậy tại sao IL, Intermediate language, quan trọng đối với .NET đến như vậy? Để trả lời câu hỏi này cần đi lướt qua những điều cơ bản (mà ta thường gọi là 101 điều) của lý thuyết về trình biên dịch. Công việc cơ bản của trình biên dịch (compiler) là phải dịch mã nguồn đọc được thành mã gốc có thể thực thi cho một hệ thống (platform) đã định. Quá trình biên dịch bao gồm vài giai đoạn, bao gồm mà không bị giới hạn về việc quét từ vựng, phân tích, tạo cây cú pháp trừu tượng (abstract syntax tree), tạo mã trung gian (intermediate code), phát sinh mã (code emission), trình dịch hợp ngữ (assembler), và nối kết (linking). Hình 1.3-1 trình bày một sơ đồ cơ bản của quá trình này.



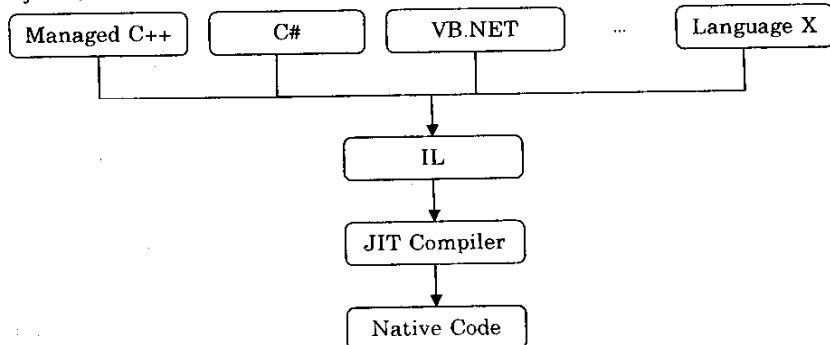
Hình 1.3-1 Tiến trình sự biên dịch.

Sự tạo thành mã lệnh trung gian (intermediate code) đặt ra một câu hỏi thú vị: Nếu có thể dịch một tập các ngôn ngữ đã định thành mã trung gian chuẩn nào

đó thì sao? Đây là chỗ mà IL tỏ ra thích ứng với vấn đề đặt ra. Khả năng dịch các cấu trúc ngôn ngữ cấp cao thành một dạng chuẩn, như IL, có chú ý đến trình biên dịch cơ bản đơn (single underlying compiler) hay trình biên dịch just-in-time (JIT). Đây là cách thức được thực hiện do .NET.

Với lời giới thiệu về Visual Studio .NET, Microsoft cung cấp Managed C++, C#, Visual Basic, và ASP.NET, mỗi ngôn ngữ này tạo ra IL từ mã nguồn tương ứng của chúng. Cũng có một vài nhà cung cấp ngôn ngữ thứ ba đưa ra các phiên bản .NET từ những ngôn ngữ tương ứng của họ. Hình 1.3-2 trình bày tiến trình biên dịch .NET cơ bản.

Điều quan trọng để chú ý đến là IL không được thông dịch. Đúng hơn là, hệ thống .NET (.NET platform) sử dụng các kiểu khác nhau của các trình biên dịch just-in-time để đưa ra mã gốc (native code) từ mã IL. Mã gốc chạy trực tiếp trên phần cứng và không cần phải có một máy ảo, như ở trường hợp Java và những phiên bản ban đầu của Visual Basic. Bằng cách biên dịch IL thành mã gốc (native code), hiệu năng lúc thực thi được tăng cao so với các ngôn ngữ thông dịch chạy trên một máy ảo. Chúng ta sẽ khảo sát các kiểu khác nhau của các trình biên dịch just-in-time sau. (Từ đây trở đi chúng ta sẽ gọi các trình biên dịch just-in-time như là jitter).



Hình 1.3-2 Sự biên dịch .NET.

IL là ngôn ngữ trọn vẹn, dựa vào ngăn xếp (full, stack-based language) mà bạn có thể sử dụng để cài đặt các thành phần .NET. Mặc dù nó không chắc chắn được đưa ra vì mục đích về hiệu suất, bạn có thể sử dụng IL để thực hiện những vấn đề mà bạn không thể thực hiện với C#. Ví dụ, trong IL, bạn có thể tạo các hàm toàn cục và gọi chúng từ bên trong IL.

1. LIÊN TÁC (INTER-OP) NGÔN NGỮ

Khả năng liên tác (inter-op) một cách liền lạc giữa các ngôn ngữ đã được bàn đến từ lâu. Trên các nền Windows khác nhau, tiêu chuẩn để phát triển ngôn ngữ chéo nhau là COM (Component Object Model). Một công nghệ hoàn hảo đã ra đời mà mục đích duy nhất là phát triển các COM component and ActiveX control.

Bởi vì COM là một chuẩn nhị phân, bất cứ ngôn ngữ nào có khả năng sử dụng (of consuming) những thành phần COM là có thể sử dụng chúng không kể đến ngôn ngữ nào đã tạo nên thành phần COM.

Mặc dù COM đã cho khả năng sử dụng lại các thành phần, nhưng nó còn lâu mới được hoàn hảo. Các ngôn ngữ nào đó, như các ngôn ngữ script, chỉ có thể sử dụng giao diện mặc định được cung cấp bởi đối tượng COM. Ngoài ra, nó không thể kế thừa (inherit) một thành phần COM để mở rộng chức năng cơ bản của nó. Để mở rộng một đối tượng nhị phân, những người phát triển đã phải bao thành phần COM lại, không là một phương pháp hướng đối tượng thật sự.

Với sự xuất hiện của .NET và IL, những trở ngại như vậy không tồn tại lâu hơn nữa. Bởi vì mỗi ngôn ngữ khi đã nhảy vào nền tảng .NET đều hiểu rõ bản chất IL và siêu dữ liệu (metadata) của thành phần .NET, một cấp độ mới về khả năng hoạt động liên kết (interoperability) ra đời. Siêu dữ liệu (metadata) cho phép duyệt một thành phần .NET để tìm ra các lớp (class), các phương thức (method), các trường (field), và các giao diện mà nó chứa. Đề tài của siêu dữ liệu (metadata) sẽ được trình bày ở cuối phần này.

2. HELLO IL

Việc sử dụng IL để phát triển các thành phần và các ứng dụng .NET có khả năng là bạn sẽ không thực hiện được nhiều việc. Tuy nhiên, để nghiên cứu thật sự về kiến trúc .NET (.NET framework), việc nắm được cơ bản về cấu trúc IL là quan trọng. NET SDK không chuyển giao mã nguồn cơ sở. Điều này không có nghĩa là bạn không thể nghiên cứu việc thực hiện các thành phần khác. Tuy nhiên, cần phải có một số kiến thức về IL.

Thông thường, hầu như cần thiết, bước đầu tiên trong việc học một ngôn ngữ mới là hiển thị thông báo "Hello World" mà ai cũng biết trên màn hình. Thực hiện công việc này chỉ với IL thì không mất nhiều công sức như khi làm cho nó phát ra âm thanh. IL không là một ngôn ngữ cấp thấp thực sự, tựa assembly. Tương tự như hầu hết các ngôn ngữ hiện đại khác, đã có một lượng đáng kể các khái niệm trừu tượng ngôn ngữ IL. Ví dụ 1.3-1 trình bày chương trình nổi tiếng "Hello World" viết bằng MSIL.

Ví dụ 1.3-1 Hello World

```

1: //File           :Hello.cil
2: //Tác giả       :Richard L. Weeks
3: //
4: //
5:
6: //định nghĩa một vài thông tin assembly cơ bản
7: .assembly hello 8: {
8:
9:   .ver    1:0:0:0
10: }
11:

```

```

12:
13: //tạo một điểm nhập cho exe
14: .method public static void main( ) il managed
15: {
16:
17:     .entrypoint
18:     .maxstack 1
19:
20:
21:     //tải chuỗi lên để hiển thị
22:     ldstr "Hello World IL style\n"
23:
24:     //hiển thị ra màn hình
25:     call void [mscorlib]System.Console::WriteLine(
26:         class System.String )
27:
28: }
29:

```

Để biên dịch ví dụ 1.3-1, đưa ra dòng lệnh sau:

ilasm hello.cil

Trình dịch hợp ngữ IL (IL assembler) tạo ra tập tin .NET hello.exe.

Mã lệnh IL hiển thị thông báo “Hello World IL style” bằng cách sử dụng phương thức static Write của đối tượng System.Console. Lớp Console là một phần của kiến trúc .NET (.NET framework) và, như cách sử dụng của nó cho biết, mục đích của nó là ghi ra kết xuất chuẩn (standard output).

Chú ý rằng không cần tạo một lớp. IL không là một ngôn ngữ hướng đối tượng (object-oriented language), nhưng nó cung cấp các cấu trúc cần thiết để mô tả các đối tượng và sử dụng chúng. IL được thiết kế để tạo sự tương thích giữa các cấu trúc ngôn ngữ và các kiểu lập trình. Như vậy tính đa dạng và rộng rãi trong thiết kế của nó sẽ tính đến nhiều ngôn ngữ khác nhau để nhắm vào thời gian thực thi của .NET.

Có một số chỉ thị (directive) có ý nghĩa đặc biệt trong IL. Các chỉ thị (directive) này được trình bày ở bảng 1.3-1.

Bảng 1.3-1 Các chỉ thị của IL được dùng trong “Hello World”

Chỉ thị	Ý nghĩa
.assembly	Tên kết xuất của assembly
.entrypoint	Bắt đầu sự thực hiện cho một .exe assembly
.maxstack	Số các khe ngăn xếp để dành riêng cho các phương thức hay hàm hiện hành

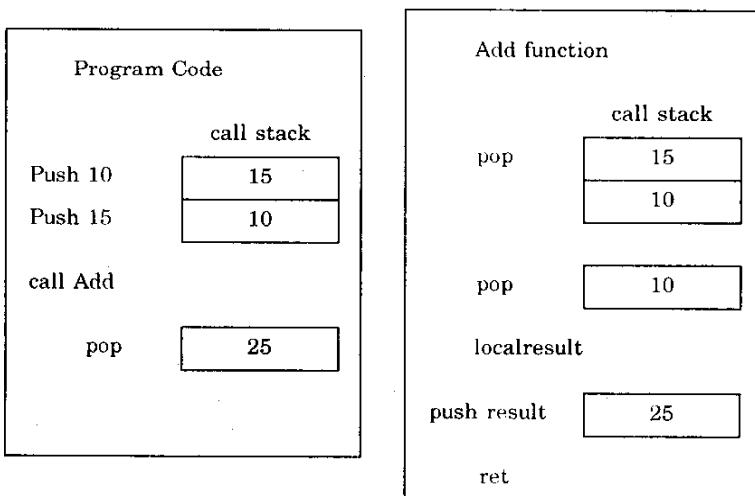
Chú ý rằng chỉ thị .entrypoint xác định điểm bắt đầu việc thực hiện cho bất cứ .exe assembly nào. Dù cho hàm của chúng ta có tên main, nó có thể có bất cứ tên nào mà chúng ta muốn. Không giống như C, C++, và các ngôn ngữ cấp cao khác, IL không định nghĩa một hàm được dùng như điểm khởi đầu của việc thực thi. Đúng hơn, IL dùng chỉ thị .entrypoint để đáp ứng mục đích này.

Chi thị khác cần chú ý là chỉ thị .assembly. IL sử dụng chỉ thị này để đặt tên kết xuất của assembly. Chỉ thị .assembly cũng chứa số thứ tự phiên bản của assembly.

3. HÀM (FUNCTIONS)

Ví dụ Hello World của IL chỉ bao hàm một phương thức. Giả mà tất cả những nhu cầu phát triển phần mềm đơn giản như thế này! Bởi vì IL là một ngôn ngữ dựa vào ngăn xếp (stack-based language), các tham số của hàm được đẩy lên ngăn xếp (stack) và hàm được gọi. Hàm được gọi có trách nhiệm xoá bất cứ tham số nào khỏi ngăn xếp (stack) và đẩy bất cứ giá trị trả về nào trở lại ngăn xếp (stack) cho người gọi hàm.

Quá trình đẩy các giá trị vào và lấy các giá trị ra từ ngăn xếp gọi (call stack) có thể được hình dung như đã trình bày ở hình 1.3-3.



Hình 1.3-3 Ngăn xếp gọi (the call stack).

Hình 1.3-3 mô tả một kết xuất rất đơn giản của tiến trình thật sự, nhưng cung cấp đủ thông tin để hiểu toàn bộ các bước liên quan. Từ hình 1.3-3 chúng ta dùng IL tạo ra mã nguồn trong ví dụ 1.3-2

Ví dụ 1.3-2 IL Function Calls

```
1: //File           :function.cil
2:
3: .assembly function_call 4: {
4:
5:   .ver 1:0:0:0
6: }
7:
8:
9: //Thêm vào phương thức
10: .method public static int32 'add'( int32 a, int32 b)
11:   {
12:
13:     .maxstack 2
14:     .locals    (int32 V_0, int V_1)
15:
16:     //lấy các tham số từ ngăn xếp gọi
17:     ldarg.0
18:     ldarg.1
19:
20:     add
21:
22:     //đẩy kết quả trả lên ngăn xếp gọi
23:     stloc.0
24:     ldloc
25:
26:     ret
27:   }
28:
29: .method public static void main( ) il managed
30:   {
31:     .entrypoint
32:     .maxstack 2
33:     .locals (int32 V_0)
34:
35:     //đẩy hai tham số lên ngăn xếp gọi (call stack)
36:     ldc.i4.s 10
37:     ldc.i4.s 5
38:
39:     call int32 'add'(int32, int32)
40:
41:     stloc.0          //lấy kết quả ở biến cục bộ V_0
42:
43:     //hiển thị kết quả
44:     listr "10 + 5 = {0}"
45:     ldloc V_0
```

```

46:     box           [mscorlib]System.Int32
47:     call void [mscorlib]System.Console::WirteLine(
        class System.String, class System.Object)
48:
49:     ret
50: }
51:

```

Mã lệnh của hàm .cil trong ví dụ 1.3-2 đưa ra một chỉ thị mới: .locals. Chỉ thị này được dùng để dành trước một vùng cho các biến cục bộ trong hàm hiện hành. Phương thức add được định nghĩa ở dòng 10 khai báo một vùng cho hai biến cục bộ. Các tham số được truyền đến phương thức add sẽ được nạp vào vùng dành trước qua lệnh ldarg. Sau khi lệnh add được gọi, bước kế tiếp là lưu trữ kết quả và sau đó đẩy kết quả trở lên ngăn xếp gọi.

Phương thức main chỉ cấp phát không gian cho một biến cục bộ đơn lẻ. Biến cục bộ này sẽ được dùng để giữ kết quả được trả về từ phương thức add do người dùng định nghĩa. Quá trình đẩy các giá trị 10 và 5 lên ngăn xếp (stack), gọi phương thức add, và lấy kết quả tương ứng được mô tả ở hình 1.3-3.

Bạn rất có thể đã chú ý lệnh box ở dòng 46. Boxing là một cái gì đó mới mẻ đối với .NET. Boxing làm cho các kiểu giá trị, chẳng hạn như các kiểu cơ sở, được xem như các đối tượng.

4. LỚP (CLASSES)

Như đã được phát biểu trước đây, IL tự nó không là một ngôn ngữ hướng đối tượng (OO language). Mặc dù IL hỗ trợ nhiều ngôn ngữ nguồn (source language) và các cấu trúc, IL cung cấp khái niệm về việc tạo các lớp (class), và tạo các thể hiện (instance) của lớp. Xem xét IL về việc khai báo và tạo các lớp (class) sẽ cho bạn hiểu nhiều hơn về Common Language Specification (CLS). Ví dụ 1.3-3 trình bày mã IL cho việc tạo và sử dụng một lớp (class).

Ví dụ 1.3-3 Tạo lớp trong IL

```

1:.assembly class_test as "class_test"
2:{ 
3:   .ver 1:0:0:0
4: }
5:
6:
7:.module class_test.exe
8:
9:
10://Tạo một lớp Dog với hai trường toàn cục (public
 //fields)

```

```
11:
12: .class public auto ansi Dog extends [mscorlib]
          System.Object {
13:
14:     //các trường dữ liệu chung (public data fields)
15:     .field public class System.String m_Name
16:     .field public int32 m_Age
17:
18:
19:     //Tạo một phương thức để hiển thị các trường
20:     .method public hidebysig instance void Print()
                  il managed
21:     {
22:         .maxstack  8
23:         ldstr      "Name : {0}\nAge :{1}"
24:         ldarg.0
25:         ldfld      class System.String Dog::m_Name
26:         ldarg.0
27:         ldfld      int 32 Dog::m_Age
28:         box        [mscorlib]System.Int32
29:         call       void
[mscorlib]System.Console::
                           WriteLine(class System.String,
30:                                         class
System.Object,
31:                                         class System.Object)
32:         ret
33:     }
34:
35:     // Phương thức khởi dựng (Constructor)
36:     .method public hidebysig specialname
                  rtspecialname instance void .ctor()
                  il managed
37:     {
38:         .maxstack  8
39:         ldarg.0
40:         call       instance void [mscorlib]System.
                           Object::ctor()
41:         ret
42:     }
43:
44: }
45:
46: .method public static void main( ) il managed
47: {
48:     .entrypoint
49:
50:     .maxstack  2
```



```

51:
52:     .locals (class Dog V_0)
53:
54:     newobj      instance void Dog:::ctor()
55:     stloc.0
56:     ldloc.0
57:     ldstr       "Daisy"
58:     stfld       class System.String Dog:::m_Name
59:     ldloc.0
60:     ldc.i4.3
61:     stfld       int32 Dog:::m_Age
62:     ldloc.0
63:     call        instance void Dog:::Print()
64:     ret
65: }
66:

```

Tạo một lớp trong IL là một quá trình khá dễ. Lớp Dog được khai báo trong ví dụ 1.3-3 được dẫn xuất từ lớp System.Object, lớp đó là lớp cơ sở của tất cả các lớp .NET. Trong .NET, các thành phần dữ liệu được hiểu như các trường (field) và các trường như vậy được nhận biết với từ khoá .field

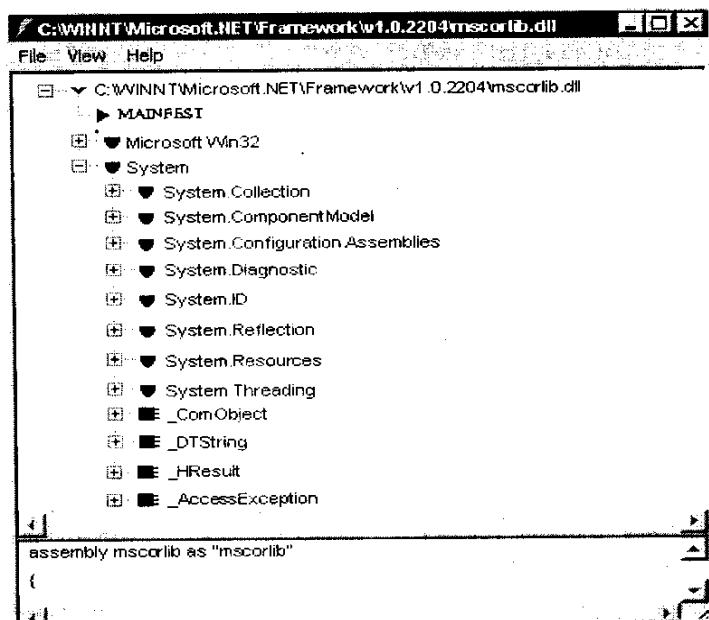
Để định nghĩa một phương thức của một lớp, tất cả những điều cần thiết là phải định nghĩa phương thức bên trong phạm vi lớp với cái mà phương thức đó thuộc về. Phương thức Print được khai báo ở dòng 20 trong ví dụ 1.3-3. Chú ý đến từ khoá instance áp dụng cho phương thức Print. Vì phương thức Print được gọi, một thể hiện (instance) của lớp Dog cần đến.

Cùng một điều ấy diễn ra đúng đắn với phương thức khởi dựng (constructor) của lớp Dog. IL cung cấp hai modifier mà phải được áp dụng vào các phương thức khởi dựng của thể hiện. Modifier rtspecialname được dùng vào lúc thời gian thực hiện (runtime), nhưng ngược lại specialname được hỗ trợ cho các công cụ .NET khác nhau, như Visual Studio .NET.

5. ILDASM

.NET đi kèm với một vài công cụ, từ các trình biên dịch (compiler) cho đến những người thiết kế form. Tuy nhiên, công cụ quan trọng nhất là ILDASM, Intermediate Language Disassembler. ILDASM chú ý đến việc quan sát các assembly khác nhau của .NET. Không ngôn ngữ được dùng để cài đặt assembly, ILDASM có khả năng hiển thị IL đã được biên dịch.

Cửa sổ chính của ILDASM tạo một sơ đồ cây (treeview) trình bày chi tiết nội dung của assembly trong .NET. Sơ đồ cây (treeview) được nhóm theo không gian tên (namespace) với mỗi không gian tên (namespace) chứa một hay nhiều không gian tên (namespace), cấu trúc (struct), lớp (class), giao tiếp (interfaces), hay bản liệt kê (enumerations) được lồng nhau (xem hình 1.3-4).



Hình 1.3-4 ILDASM với mscorelib.dll được tải lên.

Hãy tìm một assembly có tên mscorelib.dll trong .NET và dùng ILDASM để xem nội dung. Tùy theo cài đặt của .NET SDK, the .NET assemblies sẽ được định vị ở <rootdrive>:\<windows directory>\Microsoft.Net\Framework\vn.N.NNN.

Thực hiện những sự thay thế thích hợp. Hình 1.3-5 trình bày các ký hiệu khác nhau và các ý nghĩa của chúng trong ILDASM.

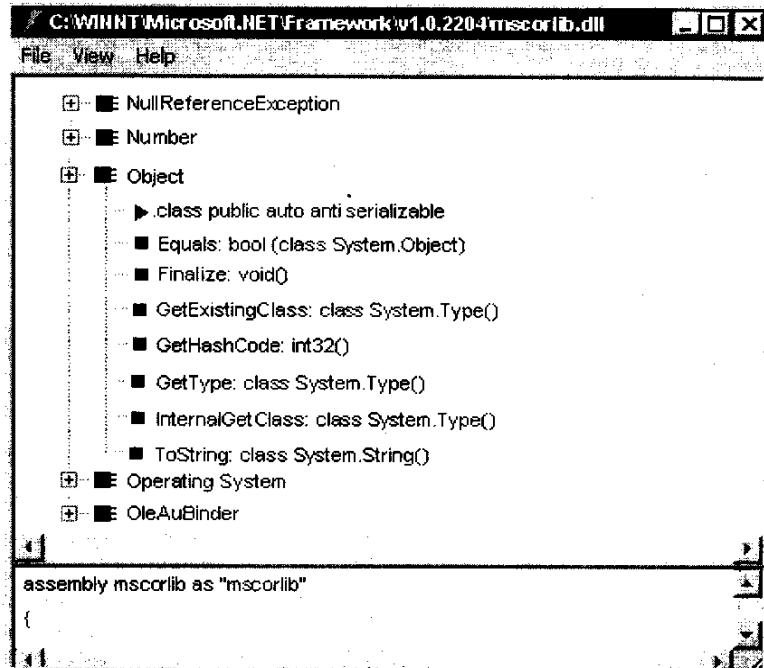
ILDASM Symbol Tables	
Symbol	Symbol Meaning
►	More Information
▼	NameSpace
■	Class
_F	Value Type (static)
IE	Interface
■	Instance Method
S	Static Method
◆	Field (data member)
◆	static Field (data member)
▼	Event
▲	Property

Hình 1.3-5 Bảng ký tự ILDASM.



Trái tim của kiến trúc .NET (.NET framework) là msclib.dll. Assembly này chứa hầu hết các giao diện và các lớp chuẩn là thành phần của .NET. Chắc chắn có các assembly khác cho truy cập cơ sở dữ liệu (database access), nối mạng (networking), tính an toàn và vân vân, nhưng nguồn gốc của tất cả đều ở msclib.dll.

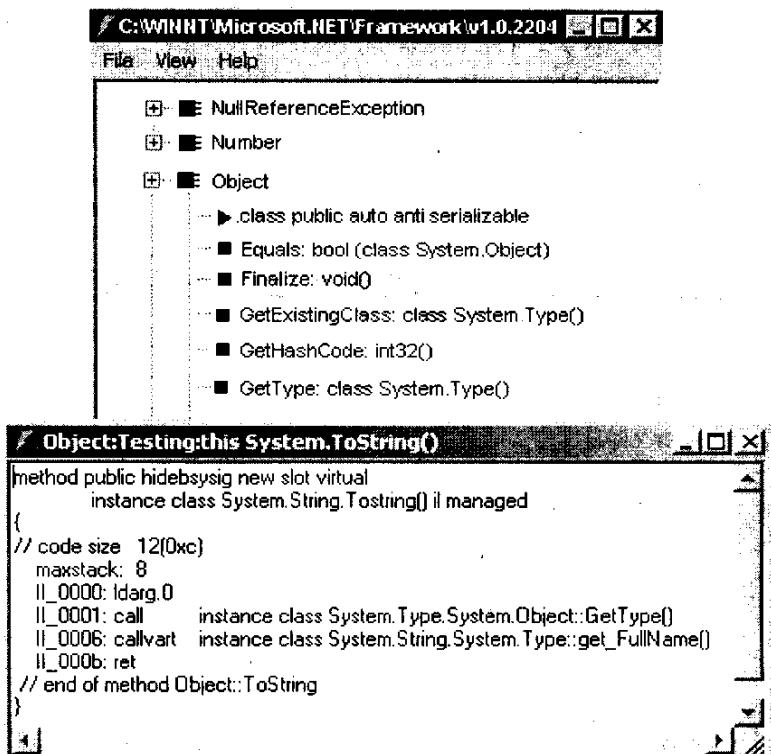
Sử dụng bảng ký tự ở hình 1.3-5, bạn có thể xác định một cách trực quan các kiểu thực thể căn cứ trong msclib.dll. Tìm System.Object và nhấp đúp vào nút đó để mở rộng nó (xem hình 1.3-6).



Hình 1.3-6 System.Object trong ILDASM.

Lớp System.Object là lớp cơ sở của tất cả các lớp trong .NET. Lớp cơ sở System.Object cung cấp chức năng chia sẻ được dùng suốt .NET. Tìm phương thức ToString và nhấp đúp nó để xem mã IL cơ sở, bao hàm cả nó (xem hình 1.3-7).

Những người phát triển MFC đã được làm quen với việc có mã nguồn Visual C++ đối với kiến trúc MFC (MFC framework) sẵn sàng dùng cho việc nghiên cứu của họ. Điều này không giống như kiến trúc .NET (.NET framework). Hiện nay, cách duy nhất để được quan sát ở bên dưới lớp vỏ bọc là phải sử dụng một công cụ như ILDASM để xem cách cài đặt của IL. Mặc dù IL cũng không dễ đọc bằng C# hay VB.NET, nhưng nó sẽ giúp bạn hiểu được các hoạt động bên trong các lớp .NET.



Hình 1.3-7 Mã IL cho *System.Object.ToString()*.

ILDASM cung cấp khả năng tạo ra một tập tin văn bản (text file) với mã nguồn IL hoàn chỉnh cho assembly đã định trong .NET. Chạy ILDASM từ dòng lệnh và sử dụng tham số /OUT=<filename>, ILDASM sẽ trực tiếp xuất ra tập tin đã định thay vì hiển thị GUI của ILDASM.

6. METADATA

Mỗi thành phần trong .NET chứa metadata. Metadata mô tả các kiểu runtime, lớp (lớp), giao tiếp (interface), cấu trúc (struct), và tất cả các thực thể bên trong một thành phần .NET. Metadata cũng cung cấp các thông tin về cài đặt và bố trí mà .NET runtime sử dụng để dùng cho JIT biên dịch mã IL.

Metadata có thể dùng được tất cả các công cụ qua hai tập API khác nhau. Ở đó tồn tại một tập ở mức thấp (low-level), các unmanaged API và một tập các managed API mà có thể được sử dụng để tập hợp thông tin về một thành phần .NET. Ở đó cũng tồn tại một tập các API để tạo Metadata. API này được sử dụng bởi các trình biên dịch trong suốt quá trình biên dịch một ngôn ngữ nguồn đã quy

định thành IL cho .NET runtime. Sự mô tả đầy đủ về các API của Metadata và cơ sở hạ tầng sẽ cần phải có một quyền sách khác, nhưng quyền sách này sẽ trình bày một phần nhỏ về cách sử dụng Reflection API để xem xét một thành phần .NET.

7. REFLECTION API

Reflection mô tả một cơ chế cực kỳ mạnh để làm việc với các thành phần .NET. Do sử dụng reflection, các thành phần .NET có thể được nạp động và nội dung của chúng được tìm ra lúc đang thực hiện. Metadata là điểm chính yếu của chức năng này và như vậy, Metadata có thể được xem đến như một typelib cũ có trong kiến trúc COM.

CLR .NET cung cấp thông tin kiểu runtime rõ ràng và cung cấp một lớp System.Type. Sau khi một thành phần .NET được nạp, lúc đó có thể nhận được kiểu hay các kiểu được chứa trong gói assembly đã định. Để mô phỏng sức mạnh của Metadata và Reflection API, bạn tạo một tập tin văn bản đặt tên employee.cs và chép ví dụ 1.3-4 vào đó. Biên dịch ví dụ, dùng câu lệnh sau:

```
csc /t:library employee.cs
```

Câu lệnh này sẽ xây dựng gói assembly employee.dll.

Ví dụ 1.3-4 Employee Class

```

1://File      :employee.cs
2://Tác giả   :Richard L. Weeks
3://Mục đích  :Tạo một thành phần .NET và sử dụng
4://           reflection API để xem thành phần
5:
6:using System;
7:
8:namespace Stingray {
9:
10:
11:    //Tạo một lớp cơ bản
12:    public class Employee {
13:
14:        //các thành phần dữ liệu riêng
15:        private string m_FirstName;
16:        private string m_LastName;
17:        private string m_Title;//tên tác vụ
18:
19:        //Các đặc tính chung (Public Properties)
20:        public string FirstName {
21:            get { return m_FirstName; }
22:            set { m_FirstName = value; }
23:        }
24:

```



```
25:     public string LastName {
26:         get { return m_LastName; }
27:         set { m_LastName = value; }
28:     }
29:
30:     public string Title {
31:         get { return m_Title; }
32:         set { m_Title = value; }
33:     }
34:
35:     //Các phương thức chung (Public Methods)
36:     public void ShowData() {
37:         object[] args = { m_LastName, m_FirstName,
38:                           m_Title }
39:
40:         Console.WriteLine("*****\n")
41:         Console.WriteLine("Employee : {0}, {1}\n"
42:                           nTitle : {2}\n", args)
43:         Console.WriteLine("*****\n")
44:     }
45:     //Các phương thức riêng (private methods)
46:     private void Update() {
47:         //THỰC HIỆN: Cập nhật thông tin Employee
48:     }
49: }
50:
```

Lớp Employee thuộc về không gian tên stringray và chứa ba thành phần dữ liệu riêng (private), tương ứng với các thuộc tính chung (public), và hai phương thức. Sử dụng Reflection API, nó có thể nạp một cách nhanh chóng employee.dll lúc đang thực hiện. Sau khi assembly được nạp, tất cả các kiểu được chứa trong assembly có thể được cẩn đến và Metadata của chúng được truy cập. Ví dụ 1.3-5 sử dụng Reflection API để quan sát một thành phần .NET.

Ví dụ 1.3-5 Metaview Source Listing

```
1://File      :Metaview.cs
2://Tác giả  :Richard L. Weeks
3://Mục đích  :Dùng Managed reflection API để duyệt
4://          một thành phần .NET
5:
6:
7:using System;
8:using System.Reflection;
9:
10:
11:
```

```
12: public class ClsView {
13:
14:
15:     public static void Main( String[] args ) {
16:
17:         try {
18:
19:             //Nạp assembly
20:             Assembly asm = Assembly.LoadForm(args[0]);
21:
22:             //Lấy các kiểu (Types) được chứa trong
23:             //Assembly
24:             System.Type[] Types = asm.GetTypes();
25:
26:             //Hiển thị thông tin cơ bản về mỗi kiểu
27:             foreach( Type T in Types ) {
28:                 Console.WriteLine("Type {0}",
29:                                 T.FullName);
30:                 DisplayFields( T );
31:                 DisplayProperties( T );
32:                 DisplayMethods( T );
33:                 Console.WriteLine("");
34:             }
35:         } catch( Exception ex ) {
36:             Console.WriteLine( ex );
37:         }
38:     }
39:
40:
41:     public static void DisplayFields( Type T ) {
42:         Console.WriteLine("*****[DisplayFields]
43:                           *****");
44:         FieldInfo[] Fields = T.GetFields(
45:             System.Reflection.BindingFlags.Instance |
46:             System.Reflection.BindingFlags.Public |
47:             System.Reflection.BindingFlags.NonPublic );
48:
49:         foreach( FieldInfo F in Fields )
50:             Console.WriteLine( "FieldName = {0}",
51:                               F.Name );
52:
53:     public static void DisplayProperties( Type T ) {
54:         Console.WriteLine("*****[Display Properties]*****");
55:         System.Reflection.PropertyInfo[] Properties
```



```
= T.GetProperties (
    System.Reflection.BindingFlags.Instance |
    System.Reflection.BindingFlags.Public |
    System.Reflection.BindingFlags.NonPublic );
54:
55:     foreach( PropertyInfo pi in Properties )
56:         Console.WriteLine( pi.Name );
57:     }
58:
59:     public static void DisplayMethods( Type T ) {
60:         Console.WriteLine("*****");
61:         System.Reflection.MethodInfo[] Methods
62:         = T.GetMethods(
63:             System.Reflection.BindingFlags.Instance |
64:             System.Reflection.BindingFlags.Public |
65:             System.Reflection.BindingFlags.NonPublic );
66:     }
```

Ví dụ 1.3-5 sử dụng các Reflection API cơ bản nhất để mô tả chi tiết thông tin về mỗi trường (field), thuộc tính (property), và phương thức (method) với employee assembly được lên. Mặc dù ngôn ngữ C# còn chưa được khảo sát đến phần lớn mã lệnh sẽ nói rõ bản thân điều đó. Bạn hãy thử thực hiện một phiên bản thiết thực về Metaview, hoàn toàn với Windows Forms GUI. Một dự án (project) cụ thể sẽ rất có thể làm tăng thêm sự hiểu biết của bạn về .NET và tận dụng phong phú các đặc tính mà nó cung cấp.

8. KẾT CHƯƠNG

Chúng ta đã đi qua nhiều vấn đề cơ bản trong chương này. Các vấn đề đã bao gồm IL, ILDASM, metadata, và kiến trúc Reflection.NET (Reflection.NET framework). Chắc chắn kiến trúc .NET đem lại một lời khuyên về tương lai của sự phát triển phần mềm: Không còn nữa một rào cản ngôn ngữ; hệ thống.NET hỗ trợ các ngôn ngữ nhằm vào khả năng thực hiện liên kết tốt hơn, cho đến bây giờ điều này không dễ dàng sử dụng hay mở rộng.

Đã qua những ngày sử dụng các tập tin IDL và DEF để mô tả các giao diện được đưa ra bởi các đối tượng. Đúng hơn, tất cả các đối tượng, các giao tiếp, và các thành phần được mô tả đầy đủ với metadata. Những nhà phát triển quen thuộc với Java sẽ thấy rõ quyết định gắn Reflection vào .NET bởi vì nó hỗ trợ một cơ chế mạnh mẽ để kéo dài sự sống của các ứng dụng và sử dụng các thành phần mới một cách nồng động.

Chương 1.4

LÀM VIỆC VỚI BỘ QUẢN LÝ C++

Các vấn đề chính sẽ được đề cập đến:

- ✓ Các từ khoá mở rộng của C++
- ✓ Sử dụng Bộ biên dịch (Compiler) C++ cho trình C++ được quản lý
- ✓ Các lớp thu gom rác (Garbage Collected)
- ✓ Chỉ dẫn (directive) #using
- ✓ Chuỗi (String)
- ✓ Trộn lẫn các mã lệnh được quản lý (Managed) và không được quản lý (Unmanaged)
- ✓ Chốt chặt (pinning) các mã lệnh được quản lý
- ✓ Các Giao diện được quản lý (Managed Interface)
- ✓ Tạo lập các kiểu giá trị (Value Type)
- ✓ Tạo và sử dụng Mô hình uỷ thác (Delegate)
- ✓ Gọi các DLL .NET tùy biến từ các mã lệnh C++ được quản lý (managed C++ code) của bạn
- ✓ Sử dụng các DLL managed C++ và Unmanaged C++ trong chương trình .NET của bạn
- ✓ Dùng các thuộc tính (Properties) trong các lớp của bạn
- ✓ Bảo đảm việc sắp xếp và đóng gói các cấu trúc C++ của bạn

Trong chương trước chúng ta đã xem sơ về bộ mở rộng được quản lý (managed extension) của C++ như một cách để sử dụng các kỹ năng về C++ của bạn, cũng như môi trường được quản lý (managed environment) của .NET hay như một cách để bạn dùng lại (reuse) những đoạn mã lệnh đã có của mình. Nếu bạn là một lập trình viên C++ kinh điển và đang tìm cách chuyển sang .NET, câu hỏi thông dụng nhất trong đầu bạn sẽ là “Tôi có thể bảo vệ tất cả những gì đã đầu tư vào mã lệnh C++ không?” và “Liệu tôi có thể dễ dàng dùng chung C++ và C# không?”. May thay, dựa vào khả năng của các kỹ sư Microsoft, cả hai câu hỏi trên đều nhận được những câu trả lời tích cực. Bạn hãy tham quan vùng đất của bộ mở rộng được quản lý của C++ và cảm nhận xem nơi nào bạn có thể đặt những mã lệnh cũ vào trong thế giới mới của .NET.

Bộ mở rộng được quản lý thêm vài từ khoá mới vào ngôn ngữ C++. Rõ ràng không lấy làm thích thú gì để tranh cãi về ngôn ngữ, thế nhưng hãy nhớ rằng trong thực tế, Managed C++ không định sửa đổi các đặc tả ANSI mà nó hứa hẹn

nhiều lợi ích từ bộ thực thi quản lý được (managed runtime) lại không phải gánh lấy những vất và của việc phải học một ngôn ngữ hoàn toàn mới. Ở một mặt khác, trình biên dịch VS .NET C++ thực ra tương thích với ANSI nhiều hơn các vị tiền bối khi sử dụng ở chế độ C++ gốc (native C++). Khi được biên dịch theo kiểu C++ thông thường, trình biên dịch sinh ra các mã đối tượng x86 để liên kết (link) theo kiểu thông thường. Khi được dùng để biên dịch các Managed C++, thông số /CLR của dòng lệnh được sử dụng để tạo ra các IL.

1. CÁC TỪ KHOÁ MỞ RỘNG CỦA C++

<u>abstract</u>	Chỉ ra rằng lớp này là lớp trừu tượng và cần cài đặt một số hay tất cả các phương thức của nó.
<u>box</u>	Tạo một đối tượng được quản lý từ một kiểu giá trị như cấu trúc hay biến. Đối tượng box thì được cấp phát trên heap của Bộ Thu gom Rác (GC - Garbage Collected) và sau đó được GC quản lý.
<u>delegate</u>	Là câu trả lời của .NET cho callback. Nó là một khai báo cho ẩn chỉ của một hàm nhằm cung cấp thông tin về kiểu trả về và các tham số của hàm đó. Khai báo của Mô hình uỷ thác (delegate) cho phép không chỉ các thành viên static mà cả các thành viên thể hiện của một lớp được gọi. Nó cũng được dùng để ánh xạ một bộ xử lý sự kiện (event handler) ứng với một tác nhân kích thích (stimulus) nào đó. Có hai kiểu của Mô hình uỷ thác: Single Cast và MultiCast. Chúng cho phép gọi hoặc là một xử lý sự kiện đơn giản hoặc là một chuỗi các xử lý cho một tác động.
<u>gc</u>	Làm cho các lớp hay cấu trúc của bạn có thể được quản lý bởi bộ thực thi GC.
<u>identifier</u>	Cho phép bạn dùng các từ khóa của C++ như các định danh, ví dụ như class hay for. Tài liệu của Microsoft nói rằng việc sử dụng chúng chẳng qua là do phong cách.
<u>nogc</u>	Khai báo rằng thể hiện của cấu trúc hay lớp không nên được quản lý bởi bộ thực thi GC.
<u>pin</u>	Được dùng để ngăn cản cơ chế nén của GC không di chuyển các lớp hay kiểu dữ liệu được quản lý. Mỗi khi một thể hiện của một đối tượng được chốt chặt (pin), nó phải được tham khảo đến bằng một con trỏ. Một đối tượng không bị chốt chặt thì giống như “trôi đi dưới chân bạn” do các hoạt động của GC sẽ di chuyển các đối tượng để dọn sạch heap. Việc này có thể có một kết quả tai hại!
<u>property</u>	Các lớp của ngôn ngữ C# có thể đưa ra các thuộc tính. Chúng giống như các thành phần dữ liệu cho các đối tượng bên ngoài.

và được thiết lập quyền đọc hay viết bằng cách gọi hàm `get_` hay `set_`. Trong thực tế, các thuộc tính có thể thực hiện một số tính toán và truy nhập vào các hàm khác, khi sử dụng các dữ liệu được cung cấp cho bộ truy cập (accessor) set hay trước khi trả về giá trị cho bộ truy cập get.

_sealed

Có hai cách dùng. Nó có thể được dùng như một điều chỉnh lớp để cho biết rằng lớp này không được phép dẫn xuất. Nó cũng còn có thể dùng như một chỉnh sửa trên cài đặt cuối cùng của một phương thức áo nào đó để ngăn không cho chúng bị viết chồng (override).

_try_cast

Cho phép bạn thử chuyển một lớp sang một kiểu đặc biệt nào đó nhưng sẽ đưa ra (throw) một ngoại lệ (exception) nếu chuyển không được trong thời gian thực thi. Cú pháp của nó như sau:

_try_cast <type_id> (expression)

Nếu việc chuyển kiểu không thành công, `System::InvalidOperationException` sẽ được gọi. `_try_cast` nên được dùng trong thời gian xây dựng và thử chương trình và sau đó được thay thế bởi `static_cast` hay `dynamic_cast` nếu cần.

_typeof

trả về `System::Type` của đối tượng đã được khai báo. Tương tự như `[object]->GetType()` cần một thể hiện của một đối tượng, từ khoá `_typeof` có thể trả về kiểu của một khai báo trùu tượng, ví dụ:

```
MyObject o = new MyObject;
Type *pt = o->GetType(); // cần một thể hiện của một MyObject
Type *pt2 = __typeof(MyObject); // không cần một thể hiện của MyObject
```

_value

Cấu trúc và các lớp đơn giản có thể được chuyển sang kiểu giá trị được quản lý bằng cách dùng điều chỉnh này. Một kiểu giá trị có thể được cấp phát trên heap GC và có thể đòi lại trong thời gian thực thi bởi GC khi chúng không còn được cần nữa. Nhớ rằng, kiểu giá trị nên được giữ đơn giản nhất bởi vì có thể sẽ xảy ra bùng nổ (overhead) khi chúng được sử dụng.

2. SỬ DỤNG TRÌNH BIÊN DỊCH C++ CHO MANAGED C++

Trình biên dịch có thêm ngắt chuyển (switch) dòng lệnh /CLR có thể nhận biết rằng mã lệnh nên được biên dịch cho ngôn ngữ thực thi chung (Common Language Runtime).

Sử dụng ngắt chuyển /LD sẽ tạo ra một DLL thay vì EXE.

Khi sử dụng trình biên ở chế độ này, trình liên kết (linker) sẽ tự động tham gia vào việc tạo ra một tập tin có khả năng thực thi, tập tin này cần có .NET CLR



khi chạy. Điều này cũng có nghĩa là bạn không thể tạo ra các tập tin có thể thực thi và chạy chúng trên hệ thống không hỗ trợ hệ thống thực thi .NET. Bạn có thể tắt bước tạo liên kết bằng cách dùng ngắt chuyển dòng lệnh /c. Trong trường hợp này, một tập tin .obj được tạo ra theo kiểu bình thường.

Bạn có thể dùng tùy chọn /link của trình biên dịch để chuyển phần còn lại của dòng lệnh trực tiếp đến trình liên kết (linker). Chỉ dẫn /link là lưu ý cuối cùng về trình biên dịch. Tất cả các chỉ dẫn khác đều sẽ được chuyển đến dòng lệnh của trình liên kết (linker).

3. CÁC LỚP THU GOM RÁC (GARBAGE COLLECTED)

Các lớp C++ thường được cấp phát trên heap hay stack khi cần thiết. Các lớp được cấp phát từ khóa new thì phải được xoá theo thứ tự để phòng tránh lõi hổng trong vùng nhớ. Các lớp thu gom rác được cấp phát trên heap của GC, lệ thuộc vào việc quản lý vòng đời và sự dịch chuyển của GC, và sẽ bị xóa (delete) khỏi hệ thống chỉ khi tham khảo cuối cùng đến chúng được giải phóng.

Khai báo của các lớp thu gom rác dưới Managed C++ thì như sau:

```
_gc class SomeClass
{
};
```

Dưới .NET, tất cả các lớp không có lớp cơ sở rõ ràng sẽ được dẫn xuất từ System::Object. Các lớp Managed C++ được khai báo với _gc cũng còn được thêm vào bằng cách này nhờ trình biên dịch. Các lớp cơ sở có các phương thức mà bạn có thể override để tăng cường sự tương tác giữa các lớp GC của bạn và hệ thống. Xem các phương thức của System::Object trong tài liệu của .NET SDK để biết thêm chi tiết.

4. CHỈ DẪN (DIRECTIVE) #using

#using có thể được xem như tương đương lệnh import và nó được dùng để tham khảo đến các thư viện của .NET. Ví dụ, để dùng các lớp của kiến trúc .NET trong mã lệnh C++ của bạn, bạn có thể dùng như sau:

```
#using <mscorlib.dll>
```

Khai báo của namespace mà bạn cần nên được đưa ra ngay sau chỉ dẫn này.

Ví dụ 1.4.1 trình bày cách tạo lập một ứng dụng đơn giản chứa lớp GC.

Ví dụ 1.4.1 managed.cpp: Một cái nhìn ban đầu về Cơ chế thu gom rác

```
#using <mscorlib.dll> // import thư viện
using namespace System; // khai báo namespace

_gc class MCPPDemo
```

```

{
public:

    void SayHello()
    {
        Console::WriteLine("Hello Managed C++");
    }

};

void main()
{
    MCPPDemo *pD=new MCPPDemo;

    pD->SayHello();
}

```

Ở đây có vài điều phải giải thích thêm. Thứ nhất là, có một sự khác biệt rõ rệt giữa "#using" và "using namespace". Chỉ dẫn #using chỉ đến một DLL sẽ được nhập vào (import). Còn using namespace giữ ý nghĩa truyền thống của nó. Tiếp theo là, hàm main tạo ra một con trỏ và gán một lớp mới vào nó, dùng nó nhưng lại không xóa nó khi thoát ra. Lạ thay, điều này lại đúng bởi vì từ khóa __gc đã được áp dụng vào định nghĩa của lớp, cho phép nó được quản lý bởi trình thực thi (runtime) .NET. Cơ chế thu gom rác tự nó sẽ quan tâm đến vòng đời của những đối tượng như vậy và sẽ xóa (delete) chúng khi bạn không dùng chúng nữa. Kết quả là sẽ không có lỗi hổng trong vùng nhớ (memory leak).

Bởi vì các lớp __gc thì được đặc chế trên heap được quản lý, bạn không thể tạo một thể hiện trên stack theo kiểu:

```
MCCPDemo d;
```

Nếu bạn thử dùng, bạn sẽ nhận được một lỗi. Bạn phải luôn luôn tạo ra các lớp GC bằng toán tử new và dùng chuyển “->” khi bạn truy xuất đến các phương thức hay dữ liệu của nó. Đây là một điều khác lạ của Managed C++ bởi vì C# và các ngôn ngữ được quản lý khác dùng dấu “.” để tham khảo đến các thành viên của lớp.

Nếu bạn muốn chạy chương trình trong ví dụ 1.4.1, hãy gõ nó trong Notepad, lưu lại nó trong tập tin managed.cpp và sau đó dùng dòng lệnh sau để biên dịch chúng:

```
CL /CLR managed.cpp
```

Nó sẽ tạo ra managed.exe, và bạn có thể chạy chúng từ dòng lệnh. Lưu ý về sự thiếu hụt rõ ràng của bước liên kết. Trình liên kết sẽ tự động tham gia vào quá trình đó.

5. CHUỖI (STRING)

Kiến trúc .NET hỗ trợ mạnh mẽ cho chuỗi. Namespace System có lớp System.String có thể được dùng để lưu giữ văn bản (text).

Bộ mở rộng được quản lý của C++ còn cho phép bạn sử dụng System.String thông qua bộ mở rộng String.

Bạn có thể định nghĩa một chuỗi dưới Managed C++ như sau:

```
String *pS = S"Hello World";
```

Các chuỗi này được tạo ra trên GC heap và chịu sự quản lý dòng đời.

Lưu ý tiền tố S trước chuỗi ký tự. Tất cả thể hiện của các chuỗi có tiền tố S là cần thiết cho cùng một chuỗi. Một chuỗi còn có thể có tiền tố L. Dấu hiệu này cho biết nó là một chuỗi ký tự rộng (wide character string):

```
String *s = L"This is a wide string";
```

Mỗi chuỗi là biến. Mỗi khi bạn đã tạo ra nó thì bạn sẽ không thể thay đổi nó. Tuy nhiên, có những phương thức của đối tượng String cho phép bạn tạo chuỗi mới, thay đổi chuỗi từ nguyên mẫu. Ngoài ra, còn có lớp StringBuilder có thể giúp cho quá trình đó.

Ví dụ 1.4.2 trình bày một minh họa.

Ví dụ 1.4.2: strings.cpp: Sử dụng Managed String của C++

```
#using <mscorlib.dll>
using namespace System;
void stringcompare(String *a, String *b)
{
Console::WriteLine("a=\"{0}\" b=\"{1}\" same string?
{2}\n",a,b,a==b ? S"true":S"false");
}
void main()
{
    String* a=S"Hello";
    String* b=S"Hello";
    StringCompare(a,b);
    a=a->Concat(S" World");
    StringCompare(a,b);
    a=a->Remove(5,6);
    sStringCompare(a,b);
}
```

Thêm vào chuỗi String đã sẵn sàng như một kiểu gốc, còn có một số lớp được thiết kế đặc biệt để thao tác các chuỗi. Lớp StringBuilder có sẵn ở namespace System::Text, và bộ ghi luồng (stream) cho chuỗi được gọi là StringWriter namespace System::IO.

Lớp `StringBuilder` cung cấp một tập các chức năng thao tác như thêm các ký tự vào một chuỗi hoặc cắt một chuỗi con từ giữa một chuỗi. `StringWriter` cung cấp một giao tiếp kiểu luồng (stream-style) cho chuỗi và cho phép bạn dùng các chức năng định dạng chuỗi kiểu .NET thay vì luôn phải phụ thuộc vào `sprintf`. Ngoài ra còn có lớp `StringReader` để đọc chuỗi theo luồng khi nó nằm ở một tập tin nào đó.

Các ví dụ về `StringBuilder` và `StringWriter` nằm trong một số ví dụ trong phần còn lại của chương này, đặc biệt là ví dụ 1.4.10.

6. TRỘN CÁC MÃ LỆNH ĐƯỢC QUẢN LÝ (MANAGED) VÀ KHÔNG ĐƯỢC QUẢN LÝ (UNMANAGED)

Trộn các mã lệnh được quản lý và không được quản lý khá là đơn giản, nhưng nó phải theo các hướng dẫn.

Để tạo (hay bảo trì) một khối các mã lệnh không được quản lý, bạn cần giới hạn mã lệnh bằng các chỉ dẫn `#pragma_unmanaged` và `#pragma_managed` khi cần. Ví dụ 1.4.3 trình bày một hàm bằng mã lệnh không được quản lý.

Ví dụ 1.4.3 Sử dụng `#pragma_unmanaged`

```
#pragma unmanaged

void timestwo (int * pToMultiply)
{
    *pToMultiply *= 2;
}

#pragma managed
```

Như bạn có thể thấy, hàm `timestwo` nhân một số nguyên đang gán vào con trỏ `pToMultiply` và được coi như mã lệnh không được quản lý bởi việc sử dụng chỉ dẫn `#pragma_unmanaged`.

7. CHỐT CHẶT (PINNING) CÁC MÃ LỆNH MANAGED

Đoạn mã lệnh trong ví dụ 1.4.3 sẽ có vấn đề nếu như bạn muốn truyền một con trỏ đến một số nguyên thuộc về kiểu được quản lý. Bạn sẽ nhớ rằng các đối tượng được quản lý có thể được dịch chuyển trong bộ nhớ của hệ thống nên GC, vì thế con trỏ truyền đến mã lệnh không được quản lý sẽ phải được bảo đảm rằng không bị dịch chuyển. Nó được thực hiện bằng việc chốt chặt đối tượng ở một nơi bằng từ khóa `_pin`.

Ví dụ 1.4.4 minh họa cách một đối tượng mã lệnh được quản lý có thể bị chốt lại để cho phép dùng một con trỏ

Ví dụ 1.4.4 pinned.cpp: Giữ chặt con trỏ bằng cách chốt chặt

```
#using <mscorlib.dll>
```



```
using namespace System;

__value struct ManagedStruct
{
    int p;
    int q;
};

#pragma unmanaged

void timestwo(int* pToMultiply)
{
    *pToMultiply *= 2;
}

#pragma managed

void main()
{
    ManagedStruct* pStruct=new ManagedStruct;
    pStruct->p=1;
    pStruct->q=2;

    int __pin* pinnedp=&pStruct->p;
    int __pin* pinnedq=&pStruct->q;

    timestwo(pinnedp);
    timestwo(pinnedq);

    Console::WriteLine("p={0}, q={1}", __box(pStruct->p),
    __box(pStruct->q));
}
```

8. CÁC GIAO TIẾP ĐƯỢC QUẢN LÝ (MANAGED INTERFACE)

Từ khóa `_interface` có thể được dùng để tạo ra giao tiếp kiểu COM với ít khó khăn nhất và không cần IDL. Từ khóa này không được liệt kê ở phần đầu của chương bởi vì nó áp dụng cho công nghệ khác.

Tuy nhiên, khi được sử dụng chung với phần mở rộng được quản lý `_gc`, nó tạo ra một giao tiếp được quản lý. Một giao tiếp đơn giản sẽ được giới thiệu ở ví dụ 1.4.5.

Ví dụ 1.4.5 managedif.cpp: Một giao tiếp được quản lý đơn giản

```
#using <mscorlib.dll>
```

```

using namespace System;

__gc __interface IDoSomething
{
    String * SayHello();
};

__gc class CDоСomething : public IDoSomething
{
    String * IDoSomething :: SayHello()
    {
        return new String("Hello");
    }
};

void main()
{
    CDоСomething* pD=new CDоСomething();
    IDoSomething *pI=static_cast<IDoSomething*>(pD);
    String* s=pI->SayHello();
    Console::WriteLine(s);
}

```

Một thuận lợi lớn của việc kết hợp các đối tượng được quản lý và giao tiếp là việc trình biên dịch có thể hỗ trợ cho các tên nhập nhằng của các phương thức. Nếu bạn có hai giao tiếp với các phương thức có cùng tên, bạn có thể chỉ ra những cái đặt nào sẽ đi cùng với giao tiếp nào. Ví dụ 1.4.6 chỉ ra các giải quyết các giao tiếp nhập nhằng.

Ví dụ 1.4.6 Ambigous.cpp: Giải quyết các giao tiếp nhập nhằng

```

#using <mscorlib.dll>

using namespace System;

__gc __interface IFist
{
    void TheMethod();
};

__gc __interface ISecond
{
    void TheMethod();
};

```

```

gc class AClass : public IFirst, public ISecond
{
    void IFirst::TheMethod()
    {
        Console::WriteLine("Invoked IFirst::TheMethod");
    }

    void ISecond::TheMethod()
    {
        Console::WriteLine("Invoked ISecond::TheMethod");
    }
};

void main()
{
    AClass* c=new AClass();

    IFirst *pF=static_cast<IFirst*>(c);

    ISecond *pS=static_cast<ISecond*>(c);

    pF->TheMethod();

    pS->TheMethod();
}

```

Lớp AClass phân biệt giữa hai định nghĩa của giao tiếp bằng cách chỉ rõ phương thức TheMethod được cài đặt.

9. TẠO RA CÁC KIỂU GIÁ TRỊ

Kiến trúc .NET nhận ra hai họ kiểu: các kiểu giá trị và các kiểu tham chiếu.

Các kiểu giá trị là các cấu trúc nhỏ tương đối đơn giản, được tạo ra và dùng trực tiếp trong stack. Kiểu tham chiếu là các đối tượng cần thiết trong quyền hạn của chúng, được tạo ra trên heap được quản lý, và được bộ thu gom rác quản lý vòng đời.

Một kiểu giá trị phải là một kiểu lưu trữ đơn giản như int hay float. Nó cũng có thể là một cấu trúc hay lớp nhỏ như time_t hay CPoint. Một kiểu tham chiếu là một lớp, giao tiếp, mảng ...

Các kiểu dữ liệu được tạo ra và sử dụng trên stack thì không thuộc sự quản lý của GC bởi vì chúng được xóa khá hiệu quả bằng cách dịch chuyển con trỏ stack khi nó di ra khỏi phạm vi. Nó ngũ ý rằng chúng được tạo ra khá đơn giản và tồn rất ít thời gian để quản lý. Có thể sẽ cần thiết để tạo ra một cấu trúc trên stack

và truyền chúng đến các hàm khác. Với lý do này, bộ thực thi được quản lý dùng kỹ thuật đóng hộp (boxing) cấu trúc đơn giản trong một đối tượng bao bọc đang giữ bản sao của dữ liệu gốc nhưng đã bị gán lại trên heap được quản lý.

Nhằm chuẩn bị cho cấu trúc đơn giản của chính bạn dễ dàng như một kiểu dữ liệu, bạn có thể dùng từ khóa `_value`:

```
_value struct Birthday
{
    int day;
    int month;
    int year;
}
```

Cũng lưu ý rằng từ khóa `_value` không tác động đến từ khóa `_gc`, bởi vậy nếu bạn muốn gán được kiểu giá trị của mình trên vùng heap được quản lý, bạn phải dùng cả hai điều chỉnh như sau:

```
_value _gc struct Birthday
{
    int day;
    int month;
    int year;
}
```

Lưu ý

Thứ tự của các điều chỉnh là quan trọng. Nếu bạn dùng `_gc _value`, bạn sẽ bị lỗi..

Ví dụ 1.4.7 trình bày một kỹ thuật có thể làm cho bất kỳ lập trình viên C++ nào cũng phải nản lòng. Nó tạo các giá trị đơn giản trên stack nội bộ và truyền con trỏ đến chúng để gói gọn chúng trong một tập hợp. Với C++ không được quản lý, nó có thể tạo ra một lỗi tai hại khi tập hợp đó cố gắng để tham khảo đến các item mà hiện đã ở ngoài phạm vi. Dưới Managed C++, bộ thu gom rác sẽ quản lý dòng đời của các giá trị được tham khảo đến và sẽ không cho phép chúng bị xóa cho đến khi tất cả mọi thứ đều di ra khỏi chúng.

Ví dụ 1.4.7 refdemo.cpp: Kiểu giá trị dưới sự điều khiển của GC

```
#using <mscorlib.dll>
using namespace System;
using namespace System::Collections;

_value struct simple
{
    int i;
    float f;
};

_gc class refdemo
```

```
{  
    ArrayList* l;  
  
public:  
  
    refdemo(){l=new ArrayList;}  
  
    void Add(Object* o)  
    {  
        l->Add(o);  
    }  
  
    void create(int n, Object **pO)  
    {  
        simple s;  
        s.i=n;  
        s.f=n*3.1415926;  
        *pO=__box(s);  
    }  
  
    void init()  
    {  
        for(int x=0;x<10;x++)  
        {  
            Object *pO;  
            create(x,&pO);  
            Add(pO);  
        }  
    }  
  
    void show()  
    {  
        for(int x=0; x<l->get_Count(); x++)  
        {  
            try  
            {  
                simple* pS=__try_cast<simple*>(l->get_Item(x));  
                Console::WriteLine("{0} * PI = {1}", __box(pS->i),  
__box(pS->f));  
            }  
            catch(System::InvalidOperationException*)  
            {  
                Console::WriteLine("Bad __try_cast...");  
            }  
        }  
    }  
};
```



```

void main()
{
    refdemo *pD=new refdemo;
    pD->init(); // tạo ra danh sách giá trị
    pD->show(); // lặp hết danh sách và hiển thị giá trị.
}

```

Việc tạo ra một kiểu giá trị với từ khóa `_value` sẽ thay đổi cấu trúc đơn giản của bạn thành một lớp được dẫn xuất từ `System::ValueType`. Nó có nghĩa rằng tự kiểu giá trị có thể override một số phương thức của lớp cơ sở để cho phép chúng góp phần đầy đủ hơn nữa vào việc thao tác với mã lệnh được quản lý. Một trong những cách sử dụng thông dụng của lớp dữ liệu là trích xuất (extract) ra một chuỗi. Cơ chế này cho phép bạn gửi một integer hay float được đóng gói đến tham số của hàm `Console::WriteLine` và nhận về kiểu của bạn để tự in chúng ra. Với cách này, bạn sẽ không phải lo lắng về định dạng phù hợp của mã lệnh `printf` cho một integer, float hay string. Đối tượng biết cách để trả về chuỗi mô tả chính nó.

10. TẠO VÀ SỬ DỤNG CÁC MÔ HÌNH ỦY THÁC (DELEGATE)

Mô hình uỷ thác là phiên bản .NET của callback và con trả hàm. Chúng chủ yếu được sử dụng trong việc định hướng các sự kiện đến bộ xử lý ngay khi chúng xuất hiện.

Một khai báo Mô hình uỷ thác mô tả một dấu hiệu của hàm (function signature) với một mẫu đặc biệt của các tham số và kiểu trả về. Mỗi khi mẫu này được thiết lập, nó có thể được dùng để gọi bất cứ hàm nào có cùng dấu hiệu. Một sự khác nhau giữa Mô hình uỷ thác và hàm callback được cài đặt trong các lớp C++ là, trong khi callback trong C++ phải luôn luôn là các thành viên static của lớp, Mô hình uỷ thác cũng có thể được dùng để gọi các thành viên thể hiện (instance).

Một delegate được khai báo như sau:

```
_delegate <kiểu trả về> MyMCDelegate(<danh sách tham số>)
```

Lưu ý:

Mô hình uỷ thác có thể có nhiều tham số. Mô hình uỷ thác phù hợp với mẫu dựa trên kiểu trả về và danh sách tham số. Vì thế,

```
_delegate char MyDelegate(int x, int y, double d)
```

có thể được dùng trong cả hai khai báo phương thức sau:
`char AFunction (int a, int b, double dData)`
`static char Foo (int moo, int hoo, double doo)`

Ví dụ 1.4.8 trình bày cách cài đặt Mô hình uỷ thác. Chúng minh họa một Mô hình uỷ thác đơn giản, một Mô hình uỷ thác sẽ nhận nhập các tham số, các Mô hình uỷ thác trả về các giá trị và số khác trả về dữ liệu thông qua tham số.

Ví dụ 1.4.8 delegate.cpp: Sử dụng các Mô hình uỷ thác

```
#include <stdio.h>

// Đầu tiên khai báo các dấu hiệu delegate sẽ dùng.

_delegate void SimpleDelegate(void);

_delegate void DelegateWithParameter(int);

_delegate int DelegateWithReturn(int);

_delegate void DelegateWithOutParameter(int, int *);

_gc class AClass
{
public:
    static void StaticHandler(int n)
    {
        printf("I am a static handler and my parameter is
%d\n",n);
    }

    void InstanceHandler(int n)
    {
        printf("I am an instance handler and my parameter is
%d\n",n);
    }

    void HandlerA()
    {
        printf("I am handler (a)\n");
    }

    void HandlerB()
    {
        printf("I am handler (b)\n");
    }

    void HandlerC()
    {
        printf("I am handler (c)\n");
    }

    void HandlerD()
    {
        printf("I am handler (d)\n");
    }
}
```



```

    }

    void WithReturn(int x)
    {
        printf("Returning %d*5\n", x);
    }

    void WithOutParam(int x, int* refval)
    {
        printf("Returning %d*14 through the out
parameter\n", x);
        *refval = x*14;
    }

};

void main()
{
    // khởi tạo lớp handler..
    AClass* pC=new AClass();

    // tạo ra mô hình uỷ thác
    DelegateWithParameter* sh=new
    DelegateWithParameter(NULL,&AClass::StaticHandler);
    DelegateWithParameter* ih=new
    DelegateWithParameter(pC,&AClass::InstanceHandler);

    DelegateWithReturn* wr=new
    DelegateWithReturn(pC,&AClass::WithReturn);
    DelegateWithOutParam* wo=new
    DelegateWithOutParam(pC,&AClass::WithOutParam);

    //Gọi static...
    sh->Invoke(10);
    // và những instance handlers.
    ih->Invoke(100);

    //Bây giờ là một multicast của tất cả handlers trong một
    danh sách

    SimpleDelegate* mc=new
    SimpleDelegate(pC,AClass::HandlerA);
    mc=static_cast<SimpleDelegate*>(Delegate::Combine(mc,
new SimpleDelegate(pC,&AClass::HandlerB)));
    mc=static_cast<SimpleDelegate*>(Delegate::Combine(mc,
new SimpleDelegate(pC,&AClass::HandlerC)));
}

```

```

mc=static_cast<SimpleDelegate*>(Delegate::Combine(mc
new SimpleDelegate(pC,&AClass::HandlerD)));
mc->Invoke(); // thực thi 4 lệnh gọi

//Bây giờ một mô hình uỷ thác với một tham số xuất
printf("Invoking delegate with value 5\n");
int r = wr->Invoke(5);
printf("WithReturn send back %d\n", r);

//Kết thúc một mô hình uỷ thác với một tham số xuất
printf("Invoking delegate with value 5\n");
wo->Invoke(11,&r);
printf("WithOutParam send back %d\n", r);

}

```

11. GỌI CÁC DLL .NET TÙY BIẾN TỪ MÃ LỆNH MANAGE C++ CỦA BẠN

Managed C++ dùng các .NET DLL bất cứ khi nào nó import các thành phần như MSCORLIB.DLL bằng chỉ thị #using. Nó còn hữu ích để có thể đặt một phần mã lệnh của bạn bằng C++ và một phần bằng C# hay VB. Nghiêm là bạn sẽ import và dùng những DLL tùy biến của chính bạn. Quy trình đơn giản và được mô tả trong phần này.

Đầu tiên, chúng ta sẽ chỉ tạo ra một DLL C# mà bạn có thể import và sử dụng. Ví dụ 1.4.9 trình bày một DLL rất đơn giản có một lớp và một phương thức.

Ví dụ 1.4.9 mydll.cs: Một DLL C# đơn giản

```

using System;

namespace mydll {
    public class myclass
    {
        public void MyMethod()
        {
            Console.WriteLine("MyMethod invoked!!!!");
        }
    }
}

```

Bạn không cần phải là một chuyên gia C# để biết cách chúng làm việc. Lưu ý rằng lớp được khai báo với điều chỉnh public. Điều này sẽ cho phép nó và các phương thức của nó được export từ thư viện. Gõ tập tin này bằng Notepad trong một thư mục trống, lưu nó thành mydll.cs và dịch nó bằng trình biên dịch C# như sau:

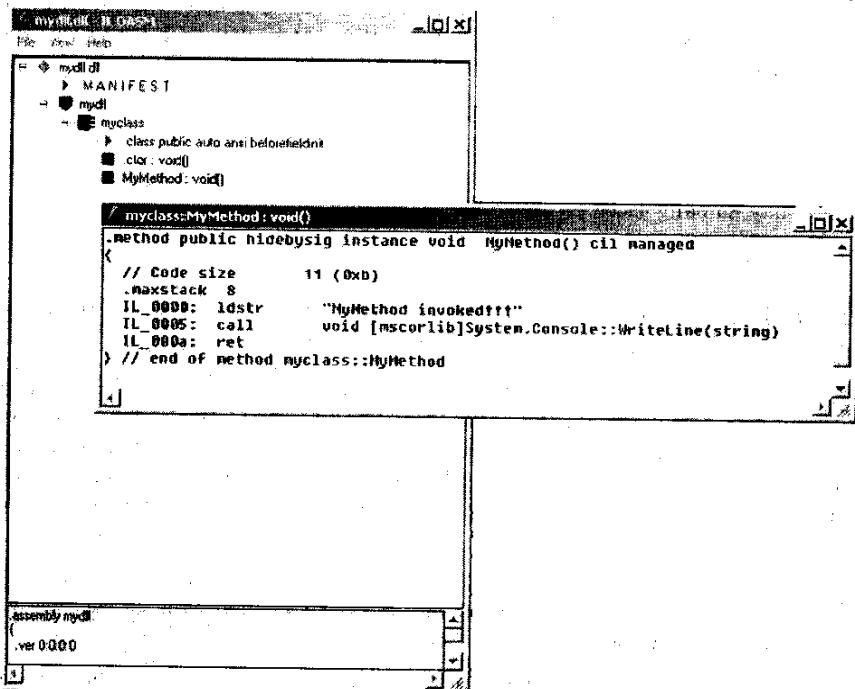
```
csc /t:library mydll.cs
```

Tùy chọn dòng lệnh /t: sẽ báo cho trình biên dịch biết để tạo ra thư viện DLL. Bạn có thể thấy rằng không có lỗi hay cảnh báo.

Không có điều gì phải nói với hàm khi viết hay dùng DLL nhưng chỉ là một sở thích, bạn có thể sẽ muốn gõ dòng lệnh này:

```
Ildasm mydll.dll
```

Điều này sẽ gọi trình biên dịch ngược Intermediate Language Disassembler (ILDASM), cho phép bạn khảo sát DLL bạn vừa tạo ra. Hình 1.4-1 sẽ cho ta thấy.



Hình 1.4-1 ILDASM, cho thấy nội dung của DLL thử.

Bạn cũng có thể dùng ILDASM để khảo cứu một số tập tin khá thi khác bạn đã tạo ra trong chương này.



Ví dụ 1.4.10 cho thấy một lớp C++ đơn giản sẽ sử dụng phương thức MyMethod gọi từ namespace mydll

Ví dụ 1.4.10 useCSdll.cpp: Một chương trình C++ sử dụng DLL C# tùy biến

```
#using <mscorlib.dll>
#using <mydll.dll>

using namespace System;
using namespace mydll;

void main(void)
{
    myclass *pM=new myclass();
    pM->MyMethod();
}
```

Gõ tập tin này vào Notepad và lưu nó thành useCSdll.cpp. Nhớ rằng cả các lệnh #using và chỉ dẫn using namespace đều tham khảo đến DLL C# ta vừa tạo ra ở ví dụ 1.4.9.

Dịch tập tin bằng dòng lệnh:

```
cl /CLR useCSdll.cpp
```

Chạy nó bằng cách gõ useCSdll ở dòng lệnh.

Bạn sẽ thấy DLL C# bạn đã tạo ra được nạp và gọi tự động.

12. SỬ DỤNG DLL C++ ĐƯỢC QUẢN LÝ VÀ KHÔNG ĐƯỢC QUẢN LÝ TRONG CHƯƠNG TRÌNH .NET CỦA BẠN

Việc liên kết và sử dụng thư viện động chỉ đơn giản là dùng bộ mở rộng Managed C++ dưới .NET. Các DLL C++ được dịch thành IL, giống như phần ngôn ngữ tự nhiên .NET của chúng, được đánh dấu bởi siêu dữ liệu (megadata) cung cấp thông tin chương trình về các lớp và các phương thức lưu trong DLL. Điều này có nghĩa là bạn có thể dùng chúng như các đối tượng .NET được viết bằng C# hoặc VB.

Các DLL khác mà bạn có thể dùng, như là các DLL hệ thống hay những DLL do bạn tạo, cần một chút công sức nhưng rõ ràng vẫn phải cảm ơn các dịch vụ Interop được cung cấp bởi Common Language Runtime.

Ví dụ 1.4.11 chỉ ra một thành phần managed C++ sẽ được biên dịch như một DLL

Ví dụ 1.4.11 cppdll.cpp: Một DLL Managed C++ đơn giản

```
#using <mscorlib.dll>
```

```

using namespace System;
using namespace System::Text;
using namespace System::IO;

// Đây là một managed C++ DLL dùng .Net framework classes
// dịch bằng...
// cl /CLR /LD cppdll.cpp

namespace cppdll {

// minh họa việc gọi một class method....
public __gc class AClass
{
public:
    void FirstMethod(int n)
    {
        String* pS=S"Hi from Firstmethod ";
        StringBuilder* pB=new StringBuilder(pS);

        StringWriter* pW=new StringWriter(pB);

        for(int x=n;x>=0;x--)
        {
            pW->Write(__box(x));
            if(x)
                pW->Write(S", ");
            else
                pW->Write(S"...");
        }
        pS=pW->ToString();
        Console::WriteLine(pS);
    }
};

}

```

Gõ mã lệnh này và lưu vào cppdll.cpp. Sau đó dịch chúng bằng dòng lệnh:
 cl /CLR /LD cppdll.cpp

DLL này sẽ được gọi như là một đối tượng .NET được quản lý đơn giản.

Ví dụ 1.4.12 trình bày một DLL không được quản lý tiêu biểu có một hàm tương ứng với DLL chúng ta vừa tạo ở ví dụ 1.4.9. Như bạn có thể thấy, nó có hàm DllMain chuẩn và một DllFunction có thể được gọi để thao tác một số thứ và in ra một thông điệp.

Ví dụ 1.4.12 straingtdll.cpp: Một DLL x86 đơn giản

```
// Đây là một C++ DLL thông thường sẽ được gọi bằng
// system interop services của .NET

#include <windows.h>
#include <stdio.h>
#include <string.h>

static char *buffer;

char* __stdcall DllFunction(int n)
{
    sprintf(buffer, "Hi From DllFunction! ");
    for(int x=n;x>=0;x--)
    {
        char temp[100];
        sprintf(temp, "%d%s", x, (x==0 ? "..." : ", "));
        strcat(buffer,temp);
    }
    return &buffer[0];
}

BOOL __stdcall DllMain(HINSTANCE hInst, DWORD dwReason,
LPVOID resvd)
{
    switch(dwReason)
    {
        case DLL_PROCESS_ATTACH:
            buffer = (char *)malloc(1000);
            memset(buffer, 0, 1000);
            printf("Straingtdll loaded\n");
            return TRUE;
        case DLL_PROCESS_DETACH:
            free(buffer);
            printf("Straingtdll unloaded\n");
            return TRUE;
        default:
            return FALSE;
    }
}
```

Theo sau là tập tin DEF kiểu cũ được dùng để export các hàm của DLL.
EXPORTS

DllFunction @1

DllMain @2

Gõ chúng vào và lưu thành tập tin straingtdll.def, sau đó dịch thành DLL với dòng lệnh sau:

```
cl /LD strainghtdll.cpp /link /DEF: strainghtdll.def
```

Các DLL này có thể được dùng bởi chương trình C# hay VB mà bạn viết. Khả năng gọi DLL kiểu cũ với mã lệnh hay ngôn ngữ kiểu mới này sẽ giữ các đầu tư của bạn ở các mã lệnh, ứng dụng, và công cụ cũ khi bạn muốn chuyển sang kiến trúc .NET. Để minh họa, chúng ta sẽ viết một chương trình cực kỳ đơn giản C# cho thấy cách dùng lớp được quản lý C++ và DLL không được quản lý. Ví dụ 1.4.13 cho thấy mã lệnh của tập tin thực thi cppdllusr.exe.

Ví dụ 1.4.13 cppdllusr.cs: Một chương trình C# đơn giản cho thấy cách dùng DLL

```

1: using System;
2: using System.Text;
3: using System.IO;
4: using System.Runtime.InteropServices;
5:
6: using cppdll;
7:
8: class dlltester
9: {
10:
11:     [DllImport("strainghtdll.dll")]
12:     [MarshalAs(UnmanagedType.LPStr)]
13:     public static extern String DllFunction(int n);
14:
15:     static void Main()
16:     {
17:
18:         // Đầu tiên method trong C++ dll "cppdll.dll" được
19:         // gọi...
20:         AClass pC=new AClass();
21:         pC.FirstMethod(10);
22:
23:         // Bây giờ gọi hàm bao bọc cho DLL không được quản lý
24:         // Console.WriteLine(DllFunction(10));
25:     }
26: }
```

Sử dụng DLL của managed C++ khá rõ ràng, nhưng cái không được quản lý thì phức tạp hơn. Chúng ta khảo sát phần làm cho sự tương tác với mã lệnh không được quản lý là có thể.

Dòng mã lệnh 11, 12, và 13 ở ví dụ 1.4.13 cho ta thấy cách tạo ra một tấm phủ (wrapper) cho một hàm trong một DLL không được quản lý. Cách này dùng các thuộc tính (attribute) để tạo ra một proxy cho các mã lệnh không được quản lý

và sắp đặt các tham số và kiểu trả về từ và đến các phần không được khai báo của hệ thống.

Thuộc tính `[DllImport(...)]` chỉ rõ rằng dấu hiệu của hàm sau là một thứ được xuất khẩu bởi DLL được đặt tên, trong trường hợp này là `strainghtdll.dll` mà chúng ta đã tạo ở ví dụ 1.4.11. Hàm này trả về một `char *` đến một vùng đệm mà nó được cấp phát. Tuy nhiên, bộ thực thi được quản lý sẽ cần dữ liệu trả về để được đóng gói trong một đối tượng thân thiện hơn với .NET, do đó, thay vì dùng các `char *` như là kiểu trả về của hàm của chúng ta, chúng ta dùng `String` và bảo hệ thống rằng nó cần sắp đặt (`marshal`) dữ liệu khi nó được trả về từ hàm. Điều này được thực hiện bởi thuộc tính `[return:MarshalAs(...)]`. Vị trí của nó đứng trước khai báo hàm cho thấy nó là kiểu trả về sẽ được sắp xếp (`marshal`). Thuộc tính như thế có thể dùng trên một tham số để chuyển dữ liệu từ một `string` thành một con trỏ `char *`. Tham số của thuộc tính `MarshalAs` là một trong danh sách các kiểu có thể nhận biết từ namespace `System::Runtime::InteropServices`.

Việc gọi `DllFunction` ở dòng 23. Đó là một tham số đơn giản cho phương thức `Console.WriteLine`. Nó chỉ chấp nhận các item dựa trên `System.Object`, bởi vậy bạn có thể thấy rằng việc sắp xếp các ký tự được trả về `char *` vào đối tượng `String` như thế nào.

Gõ ví dụ 1.4.13 vào tập tin `cppdllusr.cpp` và dịch chúng bằng câu lệnh:
`csc /r:cppdll.dll cppdllusr.cs`

Chương trình sẽ biên dịch, và bạn có thể chạy `cppdllusr` từ dòng lệnh.

13. DÙNG CÁC THUỘC TÍNH (PROPERTIES) TRONG CÁC LỚP CỦA BẠN

Các thuộc tính là các thành viên giá-dữ liệu của các lớp của bạn. Nó được truy xuất như các thành viên dữ liệu public nhưng trong thực tế chúng là các hàm bên trong lớp.

Việc thêm vào các thuộc tính cho các lớp thu gom rác C++ được thực hiện bằng cách dùng từ khoá `_property`. Ví dụ 1.4.14 trình bày một lớp GC đơn giản với các thuộc tính.

Ví dụ 1.4.14 props.cpp: Các thuộc tính đơn giản trong một lớp GC

```
#using <mscorlib.dll>

using namespace System;
using namespace System::IO;

__gc class PropClass
{
    int m_x;
    int m_y;
```

```
int m_z;

public:
    PropClass() : m_y(19762), m_z(3860)
    {
    }

    // properties đọc/viết cho một data member
    __property int get_x(){return m_x;}
    __property void set_x(int v){m_x=v;}

    // properties chỉ đọc cho một data member
    __property int get_y(){return m_y;}

    // properties chỉ ghi cho một data member
    __property void set_z(int v){m_z=v;}

    // properties không tác động trên dữ liệu
    __property int get_bogus()
    {
        Console::WriteLine("Read the bogus property!");
        return 10;
    }
    __property void set_bogus(int v)
    {
        Console::WriteLine("Wrote {0} to the bogus
property!", __box(v));
    }

    String* ToString()
    {
        StringWriter *w=new StringWriter();
        w->Write("PropClass; m_x = {0}, m_y = {1}, m_z =
{2}", __box(m_x), __box(m_y), __box(m_z));
        return w->ToString();
    }
};

void main()
{
    PropClass *pP=new PropClass();
    Console::WriteLine("At Creation...{0}", pP);
```

```

pP->x=50;
Console::WriteLine("Wrote to x...{0}",pP);

pP->z=100;
Console::WriteLine("Wrote to z...{0}",pP);

pP->bogus = 150;
Console::WriteLine(pP);

int x=pP->x;
Console::WriteLine("Read x, Return ={0},
{1}",__box(x),pP);

int y=pP->y;
Console::WriteLine("Read y. Return ={0},
{1}",__box(y),pP);

int z=pP->bogus;
Console::WriteLine("Returned value = {0},
{1}",__box(z),pP);

}

```

Chương trình props.cpp cho thấy các thuộc tính truy xuất trực tiếp và thành viên dữ liệu trong một lớp và các thuộc tính sẽ thực hiện các tính toán không cần liên hệ đến các thuộc tính của lớp.

Chương trình cũng chỉ ra kỹ thuật tạo lập các thuộc tính chỉ đọc hay chỉ ghi. Để tạo ra một thuộc tính chỉ đọc, chỉ cần đơn giản thêm vào accessor get. Thuộc tính chỉ ghi chỉ cần accessor set.

Cuối cùng, chương trình trình diễn một kỹ thuật hấp dẫn của việc override phương thức ToString(). Nhớ rằng các lớp được khai báo với __gc thì đều được xuất từ System::Object. Lớp này định nghĩa phương thức ToString, cái sẽ dùng để xuất một chuỗi của lớp ra luồng (stream) văn bản.

14. BẢO ĐÁM VIỆC SẮP XẾP VÀ ĐÓNG GÓI CÁC CẤU TRÚC C++ CỦA BẠN

Bởi vì bộ thu gom rác làm khá nhiều việc sắp xếp và bố trí mã lệnh cho nên đôi khi rất khó hiểu một cấu trúc nào đây được bố trí như thế nào trong

nhớ. Nó có thể là rất quan trọng nếu bạn đang đem đến một cấu trúc theo thứ tự byte từ chương trình hay DLL C++ trước đây và dùng chúng trong ngữ cảnh của .NET. Các dịch vụ Platform Invoke có các thuộc tính mà bạn có thể dùng để chỉ ra chính xác cấu trúc của bạn sẽ được bố trí như thế nào, nhờ vậy bạn có thể đọc chúng vào các mã lệnh được quản lý và vẫn chắc chắn rằng thứ tự và sắp xếp của các biến bộ nhớ là đã được hiểu rõ.

Thuộc tính StructLayout có thể được dùng để bố trí một cấu trúc liên tục, đặt các biến thành phần vào một offset xác định bên trong cấu trúc hay để khai báo một union. Bố trí liên tục là dạng đơn giản nhất, và bạn có thể chỉ rõ thêm về việc đóng gói các thành phần của cấu trúc.

Kịch bản thường gặp nhất là bạn đến nơi mà đã có một cấu trúc C++ được định nghĩa trước trong một ứng dụng và bạn cần dùng cấu trúc như vậy để quản lý thế giới, có thể là từ một chương trình Managed C++ hoặc có vài đoạn mã lệnh được viết bằng C#. Điều này sẽ tác động đến tất cả. Hãy nhìn vào cấu trúc C++ đơn giản ở ví dụ 1.4.15.

Ví dụ 1.4.15 simplestruct.h: Một cấu trúc C++ đơn giản

```
#pragma pack(8)

typedef struct {
    char name[17],
    int age,
    double salary,
    short ssh[3]
} simple;
```

Khai báo ở ví dụ 1.4.15 định nghĩa một cấu trúc với một dạng đóng gói đặc biệt. Để biết xem cấu trúc ấy ở trong bộ nhớ sẽ như thế nào, chúng ta hãy điền vào chúng những giá trị đặc biệt và sau đó ghi vào các biến trong cấu trúc. Ví dụ 1.4.16 cho thấy quy trình và bảng dump nội dung của cấu trúc ấy lên màn hình.

Ví dụ 1.4.16 simpletest.h: Một chương trình để thử cấu trúc đơn giản

```
1: #include <stdio.h>
2: #include <memory.h>
3: #include "simple.h"
4:
5: void dump(unsigned char * pC, int count)
6: {
7:     printf ("Dumping %d bytes\n", count);
8:     int n=0;
9:     while (n<count)
10:    {
11:        int i;
12:        int toDump = count-n >= 16 ? 16 : count-n;
13:        for(i=0; i<toDump ; i++)
```

```

14:     printf ("%02x", pC[n+i]);
15:     while (i++<16)
16:         printf ("    ");
17:     printf ("    ");
18:     for(i=0; i<toDump ; i++)
19:         printf ("%c", pC[n+i]>0x1f?pC[n+i]:'.');
20:     printf ("\n");
21:     n += toDump;
22: }
23:
24:
25:
26: void main (void)
27: {
28:     simple s;
29:
30:     memset (&s, 0xff, sizeof(simple));
31:     memcpy (&s.name[0], "abcdefghijklmnopq", 17);
32:     s.age = 0x11111111;
33:     s.salary = 100.0;
34:     s.ssn[0] = 0x2222;
35:     s.ssn[1] = 0x3333;
36:     s.ssn[2] = 0x4444;
37:
38:     dump((unsigned char*)&s, sizeof(simple));
39: }
```

Dịch mã lệnh này với dòng lệnh sau:

C1 simpletest.cpp

Bây giờ hãy chạy nó từ dòng lệnh.
Dumping 40 bytes

61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70	abcdefgijklmnop
71 ff ff ff 11 11 11 11 00 00 00 00 00 00 00 59 40	qY@
22 22 33 33 44 44 ff ff	"33DD

Mã lệnh ở ví dụ 1.4.16 cho thấy rằng cấu trúc simple đã được diền đầy dữ liệu. Đầu tiên, ở dòng 30, nó được diền vào số hexa ff để phản ánh dữ liệu thực với những khoảng trống của việc đóng gói cấu trúc. Dòng 31 chỉ ra thành phần age đang được diền bằng 17 ký tự, và dòng 32 đến dòng 36 cho thấy tất cả các biến số đều được gán trị. Mã lệnh nằm từ dòng 5 đến 32 chỉ đơn giản dump cấu trúc ra màn hình.

Kết xuất thì khá rõ ràng, nơi mà ff được nhìn thấy là phần của cấu trúc. Một dấu mới khác là $17 + 4 + 8 + 2 + 2 + 2$ không bằng 40, kích cỡ được khai báo của cấu trúc bởi sizeof.

**Lưu ý:**

Khi truyền cấu trúc này đến mã lệnh cũ của bạn từ managed C++ hay C# thì có thể sẽ có lỗi bởi vì bộ nhớ được quản lý theo cách khác và CLR không cho phép bạn tạo ra các cấu trúc được quản lý với một mảng kích cỡ cố định. Dữ liệu sẽ cần để vừa bố trí và sắp xếp (marshal) giữa hai thế giới được quản lý và không được quản lý.

Ví dụ 1.4.7 minh họa cách một cấu trúc có thể được tạo ra với vị trí chính xác của các thành viên và cách chuyển một dữ liệu mảng.

Ví dụ 1.4.7 managedsimple.txt: Phiên bản được quản lý của cấu trúc đơn giản.

```
[StructLayout(LayoutKind::Explicit, Size=40)]
public __gc struct managedsimple {
    [FieldOffset(0)]
    [MarshalAs(UnmanagedType::ByValTStr, SizeConst=17)]
    System::String *name;
    [FieldOffset(20)]
    int age;
    [FieldOffset(24)]
    double salary;
    [FieldOffset(32)]
    [MarshalAs(UnmanagedType::ByValArray, SizeConst=3)]
    short ssn __gc();
};
```

Lưu ý

Nhớ rằng bất cứ khi nào dùng những thuộc tính này trong phần mềm của mình, bạn nên khai báo rằng bạn đang dùng namespace `System::Runtime::InteropServices`.

Ở đây bạn thấy rằng không chỉ bố trí vật lý của cấu trúc được điều khiển cẩn thận bởi việc dùng các thuộc tính `StructLayout` và `FieldOffset` mà các mảng C có chiều dài cố định `char` và `short` còn được thay thế bởi các kiểu `System::String` và `System::Array` thân thiện với bộ thu gom rác.

Bởi vậy, chương trình biết làm thế nào để các mảng được quản lý sẽ được chuyển về các mã lệnh cũ, (thuộc tính `MarshalAs`, cái mà chúng ta đã thấy trong phần kết hợp với kiểu trả về), đã được mượn để chỉ ra rằng mảng cần được sắp xếp như là một mảng chuỗi 17 ký tự và một mảng của 3 số nguyên `short`.

Ví dụ 1.4.18 là một thay đổi của mã lệnh ở ví dụ 1.4.16. Ở đây, chúng ta định nghĩa một DLL C++ không được quản lý sẽ làm nhiệm vụ dump giống như trước cho chúng ta. DLL này sẽ được chuyển về chương trình managed C++, cái mà giống như là các module C# hay VB.NET. Kết quả sẽ được hiện lên màn hình lần nữa. Đầu tiên là DLL không được quản lý.

Ví dụ 1.4.18 structuredll.cpp: DLL không được quản lý

```
1: #include <stdio.h>
2: #include <memory.h>
3: #include "simple.h"
4:
5: void dump(unsigned char * pC, int count)
6: {
7:     printf ("Dumping %d bytes\n", count);
8:     int n=0;
9:     while (n<count)
10:    {
11:        int i;
12:        int toDump = count-n >= 16 ? 16 : count-n;
13:        for(i=0; i<toDump ; i++)
14:            printf ("%02x", pC[n+i]);
15:        while (i++<16)
16:            printf ("      ");
17:        printf ("      ");
18:        for(i=0; i<toDump ; i++)
19:            printf ("%c", pC[n+i]>0x1f?pC[n+i]:'.');
20:        printf ("\n");
21:        n += toDump;
22:    }
23: }
24:
25: void __stdcall ShowSimple(simple *s)
26: {
27:     printf ("Unmanaged DLL\n");
28:     dump((unsigned char *)s, sizeof(simple));
29:     printf ("Name=%17s\n", &s->name[0]);
30:     printf ("Age=%d\n", &s->age);
31:     printf ("salary=%f\n", &s->salary);
32:     printf ("SSN=%d-%d-%d\n\n", &s->ssm[0], &s->ssm[1], &s->ssm[2]);
33:     printf ("Changing the salary...\\n");
34:     s->salary=999999.0;
35:     printf ("Changing the name...\\n");
36:     strcpy(s->name, "Mavis    ");
```

```

37:     printf ("Changing the ssn... \n");
38:     s->ssm[0]=99;
39:     s->ssm[1]=999;
40:     s->ssm[2]=9999;
41: }
42:
43: BOOL __stdcall DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID)
44: {
45:     switch (dReason)
46:     {
47:         case DLL_PROCESS_ATTACH:
48:         case DLL_PROCESS_ATTACH:
49:             return TRUE;
50:         default:
51:             return FALSE;
52:     }
53: }

```

Tập tin DEF xuất khẩu các phương thức mà bạn sẽ dùng, lưu chúng trong structuredll.def.

EXPORTS

```

DllMain @1
ShowSimple @2

```

Dịch DLL này với dòng lệnh:

```
C1 /LD structuredll.cpp /link /DEF: structuredll.def
```

Ví dụ 1.4.19 cho thấy rằng một ứng dụng managed C++ sẽ dùng DLL ở ví dụ 1.4.18 thông qua lời gọi hệ thống, truyền các cấu trúc được định nghĩa đặc biệt và được quản lý việc sắp xếp.

Ví dụ 1.4.19 simpledump .cpp: Chương trình thử managed C++

```

1: // Đây là managed C++ test
2: // để simpledump một dll
3:
4: #using <mscorlib.dll>
5:
6: using namespace System;
7: using namespace System::Runtime::InteropServices;
8:
9: // Đây là khai báo của cấu trúc

```

```
10: [StructLayout(LayoutKind::Explicit, Size=40)]
11: public __gc struct managedsimple {
12:     [FieldOffset(0)]
13:     [MarshalAs(UnmanagedType::ByValTStr, SizeConst=17)]
14:     System::String *name;
15:     [FieldOffset(20)]
16:     int age;
17:     [FieldOffset(24)]
18:     double salary;
19:     [FieldOffset(32)]
20:     [MarshalAs(UnmanagedType::ByValArray, SizeConst=3)]
21:     short ssn __gc();
22: };
23:
24:
25: // Đây là lớp sẽ imports các DLL function ta cần
26: __gc class stDll {
27: public:
28:     [DllImport("structuredll.dll")]
29:     static void ShowSimple([In, Out,
30:     MarshalAs(UnmanagedType::LPStruct,
31:     SizeConst=40)]managedsimple);
32:     // nếu bạn muốn import các functions khác của DLL
33:     // có thể thêm vào ở đây
34: };
35:
36: void main (void)
37: {
38:     managedsimple *m = new managedsimple;
39:
40:     m->age = 24;
41:     m->salary = 180000.0;
42:     m->name = new System::String("Beavis");
43:     m->ssn = new short __gc(3);
44:     m->ssn[0] = 10;
45:     m->ssn[1] = 100;
46:     m->ssn[2] = 1000;
47: }
```

```

48:     managedsimple __pin* pS = m;
49:
50:     StD1l::ShowSimple(pS);
51:
52:     pS = 0;
53:
54:     Console::WriteLine("Back in the managed code");
55:     Console::WriteLine(m->salary);
56:     Console::WriteLine(m->name);
57:     Console::WriteLine("{0}-{1}-{2}", __box(m-
>ssn[0]), __box(m->ssn[1]),
58:         __box(m->ssn[2]));
59: }

```

Biên dịch tập tin này bằng dòng lệnh:

C1 /clr simpledump .cpp

Và bây giờ hãy chạy nó từ dòng lệnh. Bạn sẽ thấy kết xuất sau:
Unmanaged DLL

Dumping 40 bytes

42 65 61 76 69 73 00	Beavis.....
00 00 00 00 18 00 00 00 00 00 00 00 00 00 80 66 40 Cf@
0a 00 64 00 e8 03 48 00	..d..H.

Name= Beavis
Age=24
Salary=180.000000
SSN=10-100-1000

Changing the salary...
Changing the name...
Changing the ssn...
Back in managed code
999999
Malvis
99-999-9999

Bắt đầu với hàm main ở dòng 36, chúng ta tạo ra một cấu trúc ở dòng 38 và sinh chúng ở dòng 40 đến 46. Dòng 50 gọi hàm của DLL, và chúng ta nhảy sang dòng 25 ví dụ 1.4.18.

Phương thức này sẽ dump một nội dung của cấu trúc bằng hệ hexa, cho thấy cách cấu trúc được quản lý của chúng ta, với cả chuỗi và mảng các số nguyên ngắn (short) được sắp xếp vào thế giới không được quản lý của DLL. Ở dòng 29 đến 40 của việc liệt kê cấu trúc, dữ liệu trong cấu trúc không được quản lý được



thay đổi bởi mã lệnh C++ cũ không được quản lý nhưng tốt. Khi tiến trình kết thúc, trình sắp xếp (marshal) .NET lại đem các cấu trúc không được quản lý vào thế giới được quản lý nơi dòng 54 đến 58 của ví dụ 1.4.19 xác nhận các thay đổi bằng cách in các giá trị mới ra ngoài màn hình.

Như bạn có thể thấy từ các đoạn mã lệnh trong phần này, khả năng siêu - thao tác giữa các thế giới được quản lý và không được quản lý được cung cấp tốt và chúng giúp bạn giữ các giá trị trí tuệ đã có vào những năm của .NET.

15. KẾT CHƯƠNG

Các lập trình viên C++ đang làm việc với .NET được chăm sóc khá kỹ. Các mã lệnh đã có của bạn có thể vượt qua hàng rào .NET, bạn cũng không phải quẳng hết mọi thứ và làm lại từ đầu. Có thể sẽ có nguyên một cuốn sách riêng về đề tài dịch chuyển (migration) mã lệnh, chúng ta tin rằng một ai đó sẽ làm điều này, tuy nhiên chương này sẽ giúp bạn thấy được con đường và dựng bạn dậy để chạy.

Chương 1.5

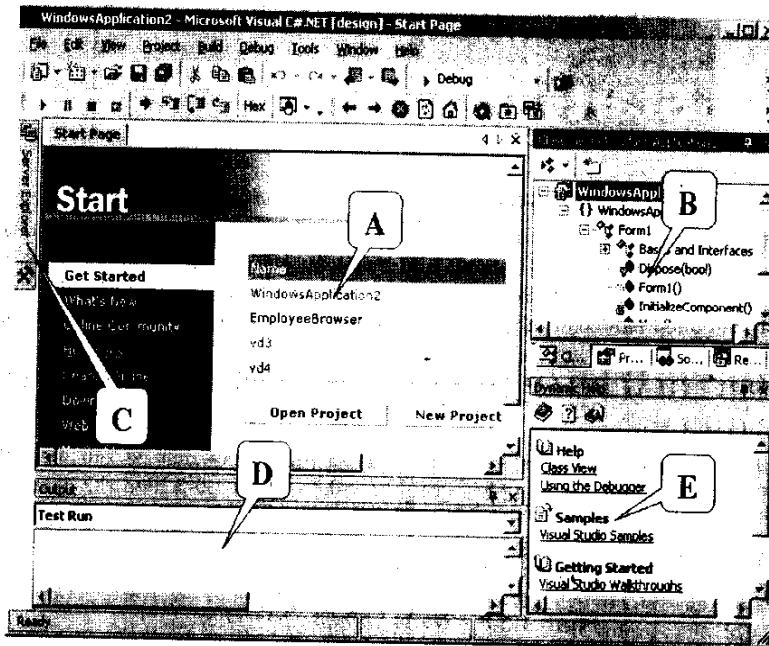
GIỚI THIỆU VISUAL STUDIO .NET

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Tích hợp môi trường phát triển (IDE).*
- ✓ *Khu vực soạn thảo chính.*
- ✓ *Giải pháp (Solution), Lớp (Class), Mục (Item), và trình giúp đỡ (Help).*
- ✓ *Hộp công cụ (Toolbox) và Server Explorer.*
- ✓ *Những tác vụ (Task), kết xuất thông tin (Output), kết quả tìm kiếm (Search Result), và Watch.*
- ✓ *Thuộc tính, Trình giúp đỡ năng động, và những Favorite.*
- ✓ *Chương trình gỡ lỗi.*

1. TÍCH HỢP MÔI TRƯỜNG PHÁT TRIỂN (IDE).

Nếu bạn đã quen thuộc với Visual Studio 5.x hoặc 6.x, thì giao diện của Visual Studio .NET cũng không khác mấy. Tuy nhiên, có một vài sự khác biệt bạn nên tìm hiểu để dễ dàng phát triển sản phẩm với bộ công cụ mới này.



Hình 1.5-1 – Visual Studio .NET

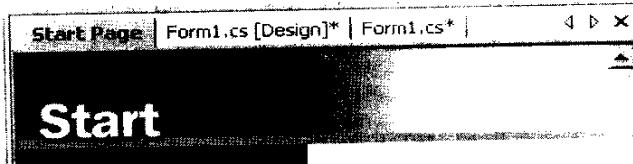
Visual Studio .NET khá tiên tiến và lớn hơn so với họ Visual Studio cũ. Trong quá khứ, việc phát triển của C++ và Visual Basic thường tách biệt nhau. Tuy nhiên giờ đây, bạn được cung cấp sự chọn lựa của C++, C#, JavaScript, Visual Basic, và rất nhiều ngôn ngữ khác đang chạy cạnh nhau trên kiến trúc .NET framework. Mỗi trường phát triển tinh vi và có thể xử lý tất cả những phần của ứng dụng của bạn theo cách hợp nhất.

Riêng bộ Visual Studio muốn đề cập hết có lẽ chiếm rất nhiều trang của giáo trình, nhưng trong chương này, chúng tôi chỉ giới thiệu khái quát đầy đủ để bạn có thể làm quen và làm việc hiệu quả ngay với bộ công cụ phần mềm này.

Hình 1.5-1 hiển thị ảnh ban đầu của Visual Studio .NET. Những thành phần chính của Visual Studio được chú thích từ A qua E.

PHẦN A : KHU VỰC SOẠN THẢO CHÍNH

Đầu tiên bạn nhìn thấy trong khu vực này là trang Start Page. Giống những cửa sổ khác của VS.NET, nó là một cửa sổ HTML Browser, cửa sổ này hiển thị những dự án gần nhất của bạn và đưa ra những thông tin hữu dụng khác về VS.NET. Khu vực này cũng là nơi soạn thảo của những tập tin chương trình (source file) của bạn. Ở trên cửa sổ này là một thanh tab cho phép bạn dễ dàng chọn những tập tin đang mở hiện hành để soạn thảo (hình 1.5-2).



Hình 1.5-2 – Chọn những tập tin đang mở

Ở đây bạn có thể nhìn thấy rằng Form1.cs được mở cho việc soạn thảo trong chế độ thiết kế (Design Mode) (giống như một biểu mẫu trong Visual Basic), và tập tin chương trình nguồn C#. Bên phải thanh này là những thành phần điều khiển (control) cho phép bạn cuộn tab bên trái hoặc bên phải và đóng cửa sổ hiện hành.

PHẦN B : GIẢI PHÁP (SOLUTIONS), LỚP (CLASSES), TRỢ GIÚP (HELP).

Trong phần này của cửa sổ IDE, bạn sẽ nhìn thấy một số tab điều khiển (tabbed control) với nhiều khung hiển thị khác nhau. Hình 1.5-3 hiển thị các tab này.



Hình 1.5-3 – Những ô nhân bên phải.

Mặc định, khung này chứa đựng các cửa sổ Solution Explorer, Class View, Help Contents, và Help Search.

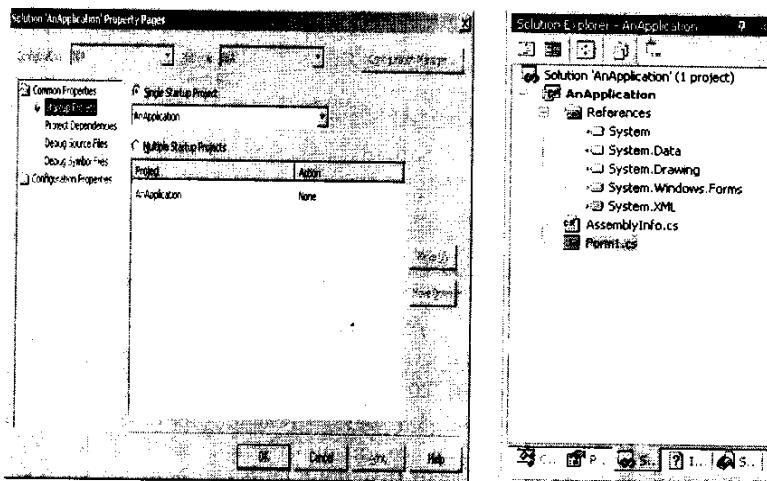
1.1. Solution và Solution Explorer

Solution (giải pháp) là một khối công việc nền tảng trong .NET. Một solution có thể chứa đựng nhiều kiểu dự án (project) khác nhau. Ví dụ, bạn muốn tạo một ứng dụng khách chủ (client/server) đầy đủ với các dự án con (sub-project) riêng lẻ từ chương trình khách (client) được viết mã trên MFC và chương trình chủ (server) được viết mã trên ASP.NET. Solution là nơi kết hợp tất cả những dự án con trong ứng dụng. Một Solution có thể biên dịch một lần chung với những dự án con.

Khi bạn chọn tab Solution Explorer, bạn sẽ thấy một cấu trúc cây bao gồm những mục khác nhau. Cây này chứa đựng những dự án, thư mục, tập tin, tài nguyên (resource), và những dữ liệu linh tinh khác định nghĩa phạm vi công việc hiện hành của bạn.

Một solution có những thuộc tính ảnh hưởng đến tất cả những dự án bên trong nó. Như với hầu hết những mục khác trong Visual Studio .NET, khi nhấp chuột phải lên tên Solution và chọn Properties sẽ cho phép bạn điều chỉnh những đặc tính của Solution.

Hình 1.5-4 - hiển thị khung cửa sổ Solution và hộp hội thoại thuộc tính dành cho Solution đó.



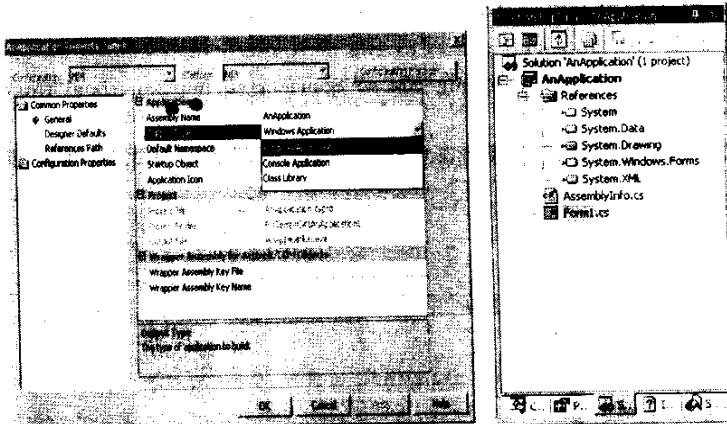
Hình 1.5-4 – Solution và những thuộc tính của nó.

Ở đây khung cửa sổ Solution Explorer nằm bên phải chứa một dự án đơn tên là AnApplication. Hộp hội thoại thuộc tính Solution (nằm bên trái) cho phép bạn chọn dự án nào là dự án khởi động, mỗi dự án phụ thuộc vào dự án khác như thế nào, nơi IDE sẽ tìm kiếm những tập tin tài nguyên khi gỡ lỗi, và nơi những tập tin ký hiệu gỡ lỗi được lưu (debug symbol files). Những thiết lập này sẽ dùng chung cho tất cả những dự án bên trong một Solution.

2. DỰ ÁN (PROJECTS)

Solution chứa các dự án, còn dự án chứa những mục thông tin. Những mục này có thể là những thư mục dùng cho việc tổ chức mã nguồn của bạn, những tập tin định nghĩa (header file) và tập tin tài nguyên (resource file), những tham chiếu tới những dự án khác, những dịch vụ web (web service) và những tập tin hệ thống (system files), cùng tất cả những tập tin mã nguồn của dự án bên trong solution. Trong hình 1.5-4, bạn có thể nhìn thấy dự án đang mở có 2 tập tin là Form1.cs và AssemblyInfo.cs, cùng tập hợp những tham chiếu (reference). Các tham chiếu reference này tương tự như chỉ dẫn #include trong tập tin C++; chúng cho phép truy xuất tới các siêu dữ liệu metadata trong các không gian tên (namespaces). Những tham chiếu này cũng có thể tham chiếu tới mô tả Web Service Description Language (WSDL) của một dịch vụ Web. Kiểu tham chiếu đến dịch vụ Web gọi là Web reference.

Mỗi dự án bên trong một Solution có những thuộc tính. Hình 1.5-5 hiển thị thuộc tính của dự án AnApplication.



Hình 1.5-5 – Hộp thoại hiển thị thuộc tính của dự án.

Trong hộp thoại hiển thị ở hình 1.5-5 như bạn thấy, chúng đơn giản, dễ sử dụng và khá ngắn gọn hơn so với VS 6.0.

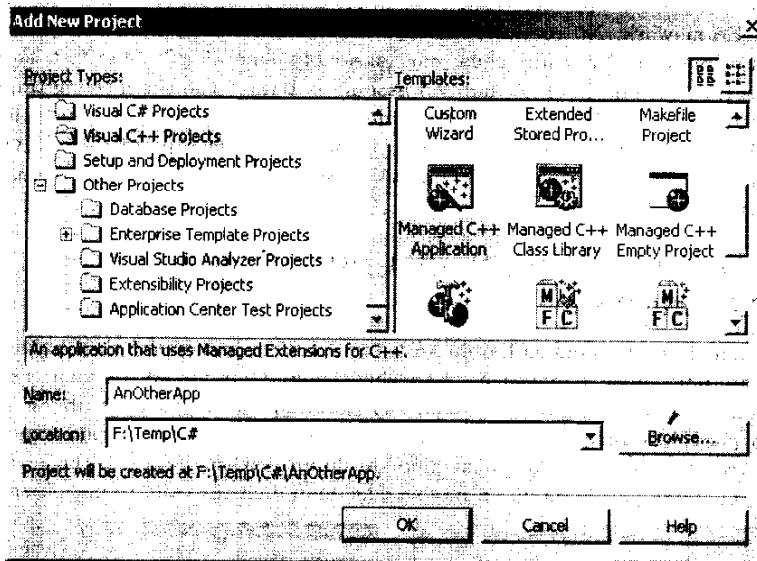
Các thuộc tính của dự án bao gồm những tùy chọn (options) quy định cách biên dịch chương trình như mức tối ưu hóa (optimizations), cách khởi động và hiển thị form (start options), kể cả cách soạn thảo dòng lệnh và những tùy chọn gỡ lỗi.

Nhấp chuột phải trên dự án sẽ hiển thị trình đơn ngữ cảnh (cho phép bạn chọn cách biên dịch dự án một cách riêng biệt, cùng với thêm vào những mục cần thiết cho dự án, như tham chiếu, file tài nguyên, mã nguồn mới ...).

3. NHIỀU DỰ ÁN TRONG MỘT SOLUTION ĐƠN

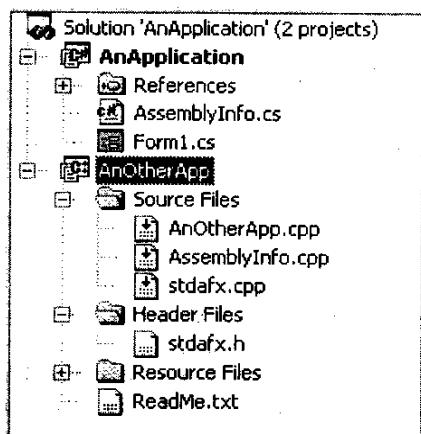
Việc thêm một dự án vào một solution khá đơn giản. Bạn nhấp phải chuột trên Solution và chọn Add. Visual Studio sẽ hiển thị danh sách các chọn lựa dành cho việc tạo mới đối tượng, sử dụng đối tượng hiện có trên máy của bạn, hoặc lấy một đối tượng về từ Web.

Hình 1.5-6 hiển thị hộp thoại New Project Wizard của lệnh Add, chúng ta sẽ thêm một dự án Managed C++ vào Solution.



Hình 1.5-6 – Thêm một dự án vào solution.

Sau khi chấp nhận hộp thoại (nhấn OK), Solution giờ đây sẽ chứa 2 dự án (hình 1.5-7). Dự án thứ hai có cấu trúc khác với dự án đầu. Vì là một dự án Managed C++, nên nó chứa những thư mục cho dành cho các tập tin mã nguồn, tập tin định nghĩa và tập tin tài nguyên.

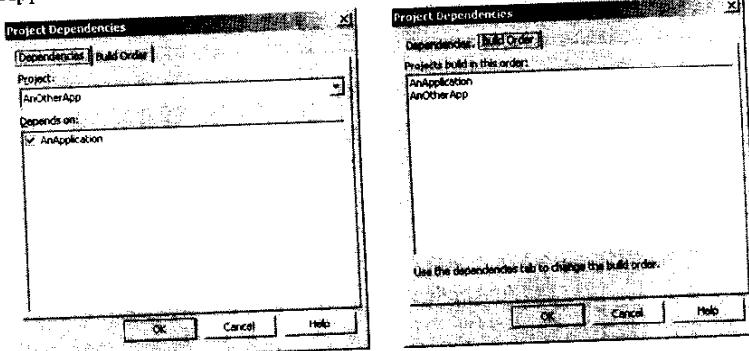


Hình 1.5-7 – Một solution với nhiều dự án.

Trong cả hai dự án này, có một tập tin gọi là Assembly. Tập tin này sẽ có trong tất cả những dự án .NET Framework. Nó tương tự như tài nguyên VS_VERSION_INFO sử dụng trong Visual C++. Một Assembly là một đơn vị cơ bản của Framework. Nó có thể chứa một hoặc nhiều những module thực thi và những tài nguyên. Nó cũng chứa một assembly “manifest”. Manifest là một metadata (hay siêu dữ liệu) kết hợp với assembly. Bạn có thể thêm thông tin, như số phiên bản của mã chương trình, nhãn hiệu đăng ký, các mô tả, và những thứ dữ liệu khác vào manifest bằng cách điền vào nội dung của tập tin AssemblyInfo. Trong phần 5 của cuốn sách này, chúng tôi sẽ giải thích cách sử dụng và quản lý Assembly.

4. CÁC FILE PHỤ THUỘC CỦA DỰ ÁN (PROJECT DEPENDENCIES)

Hiện chúng ta đang có hai dự án trong Solution, bạn có thể thấy những file mà dự án phụ thuộc bằng cách nhấp chuột phải trên dự án và chọn Project Dependencies từ menu ngữ cảnh. Menu này cho phép bạn quyết định những dự án nào cần phải được biên dịch trước. Hộp thoại xác định tính phụ thuộc được hiển thị như trong hình 1.5-8. Hộp thoại hiển thị dự án AnOtherApp phụ thuộc vào dự án AnApplication và những dự án này sẽ được biên dịch theo thứ tự được hiển thị.



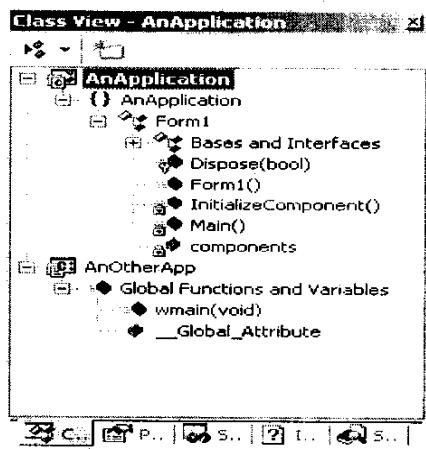
Hình 1.5-8 – Chỉ định phụ thuộc trong dự án

5. CLASS VIEW

Class View

Nếu đã quen thuộc với Visual C++ 6.0, thì ắt hẳn bạn sẽ cảm thấy quen thuộc với tab Class View. Class View là nơi quan sát các lớp của dự án cũng cho phép xem những dự án phân loại theo lớp (class), thành viên (member), và hàm (function). Nhấp chuột trên một mục sẽ đưa bạn vào môi trường soạn thảo. Nhấp chuột phải trên một lớp sẽ đưa ra một trình đơn ngữ cảnh bao gồm những mục dành cho việc thêm vào phương thức (methods), thuộc tính (properties), trường (fields), và các chỉ mục (index). Những lớp này cũng thể hiện

lớp cơ sở mà chúng kế thừa cùng với các giao tiếp (interfaces). Hình 1.5-9 hiển thị nội dung của khung Class View cho Solution của chúng ta.



Hình 1.5-9 Khung Class View.

6. RESOURCE VIEW

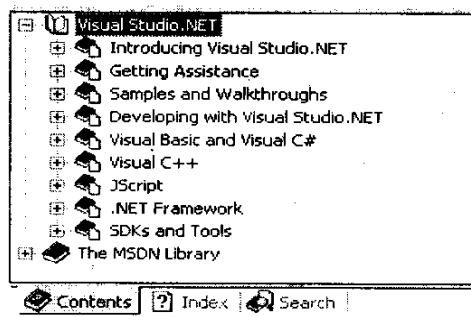
 **Resource View** Tab điều khiển chứa Class View và Solution Explorer cũng có thể dùng để chứa khung cửa sổ Resource Review. Khi đang tạo ra một ứng dụng Windows Forms, bạn sẽ không nhìn thấy tab này, nhưng bạn có thể yêu cầu hiển thị nó một cách thủ công bằng cách chọn View | Resource View từ menu chính hoặc nhấn phím tắt Ctrl+Shift+E. Khung cửa sổ này sẽ cho phép soạn thảo những tập tin tài nguyên theo kiểu cũ mà bạn đã từng biết từ C++ và MFC. Chú ý rằng Visual Studio .NET sẽ cho phép bạn tạo và soạn thảo những ứng dụng MFC và ATL bên cạnh những dự án theo kiến trúc .NET mới.

7. MACRO EXPLORER

 **Macro Explorer** Có thể mở rộng việc sử dụng macro trong công việc phát triển hàng ngày của bạn. Visual Studio .NET có một kiểu đối tượng thông minh, đối tượng này cho phép bạn viết kịch bản (scripting) về mọi chức năng mà bạn có thể tưởng tượng. Macro Explorer là nơi mà bạn sẽ tìm thấy tất cả những macro kết hợp với phiên bản VS.NET của chính bạn.

8. TRÌNH GIÚP ĐỠ(HELP)

Hệ thống Help hỗ trợ bạn rất nhiều thứ nó bao gồm nội dung trợ giúp, chỉ mục, tìm kiếm. Hình 1.5-10 là một hệ thống trợ giúp Help đầy đủ.



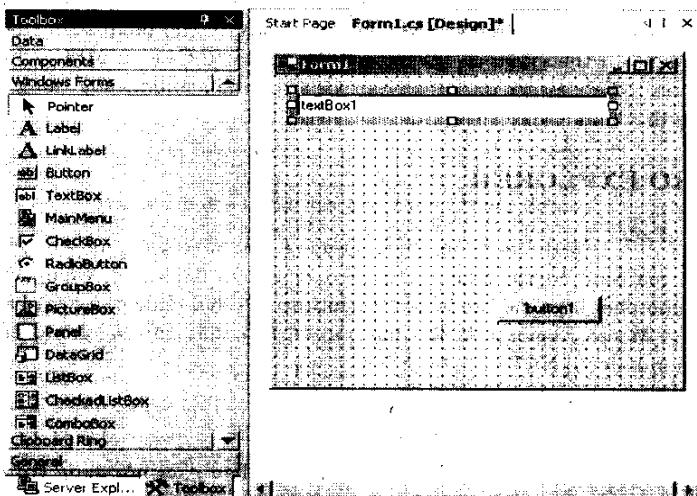
Hình 1.5-10 – Khung cửa sổ Help

Tab Index cho phép bạn đánh vào một nhóm từ đơn giản và chỉ mục để tìm kiếm nhóm từ gần giống nhất ngay tức khắc. Nhãn Search cho phép bạn thực hiện những câu truy vấn phức tạp. Những mục tài liệu có thể được tìm kiếm bằng cách sử dụng biểu thức tìm kiếm, ví dụ, "Thread and blocking" sẽ tìm kiếm tất cả những tài liệu chứa những từ "Thread and blocking" hoặc "Thread" và "Blocking".

PHẦN C : HỘP CÔNG CỤ (TOOLBOX) VÀ SERVER EXPLORER

8.1. Toolbox

Hộp công cụ (toolbox) chứa nhiều phần, phần chính là component toolbox. Hình 1.5-11 hiển thị Toolbox trong trạng thái mở rộng. Giao diện của Toolbox tương tự như Outlook Bar của Microsoft Outlook nhưng khá cố định. Ở đây, bạn có thể nhìn thấy Windows Forms đang bị chỉnh sửa. Các thành phần giao diện có thể được kéo từ Toolbox đưa vào form.

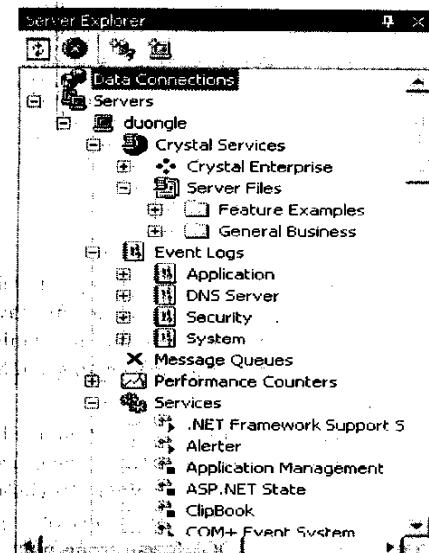


Hình 1.5-11 Hộp công cụ.

9. SERVER EXPLORER

Làm bạn với Toolbox trong thanh công cụ này là Server Explorer. Đây là một tính năng mới trong VS.NET cho phép lập trình và truy xuất nhanh cơ sở dữ liệu của các máy chủ (server), những hàng đợi thông điệp, những dịch vụ hệ thống, và những sự kiện hệ thống đều được ghi lại trong Server Explorer.

Server Explorer hiển thị tập hợp các nút theo cấu trúc cây cho tất cả những máy chủ mà nó được kết nối tới. Bạn có thể chọn những loại nút khác nhau và kéo chúng lên Windows Forms nơi mà VS.NET tạo một tập hợp những bộ điều phối dữ liệu (adapter) hộ bạn. Các bộ điều phối adapter này cho phép chương trình của bạn truy xuất nút dữ liệu được chọn. Hình 1.5-12 là giao diện của Server Explorer



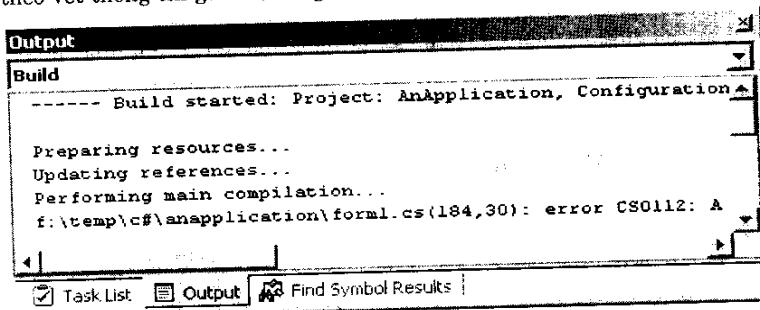
Hình 1.5-12 – Cửa sổ Server Explorer

Bạn có thể kéo một cơ sở dữ liệu Access từ gốc Data Connections lên trên một form, form này tạo một OleDbConnection tới cơ sở dữ liệu. Mở rộng nút cơ sở dữ liệu trong Server Explorer cho phép bạn kéo các bảng dữ liệu (tables), các thủ tục stored procedure, và khung nhìn dữ liệu (view) đặt lên form, đồng thời hệ thống cũng sẽ tạo ra các bộ điều phối dữ liệu tương ứng chính xác cho chúng.

Bạn cũng có thể thêm một nút server vào cửa sổ Server Explorer bằng cách sử dụng Tools | Connection to Server menu của thanh công cụ chính. Sau khi thực hiện kết nối vào server, bạn có thể truy xuất những dữ liệu mà server cung cấp ngay trong Server Explorer.

PHẦN D : TÁC VỤ (TASKS), KẾT XUẤT THÔNG TIN (OUTPUT), TÌM KIẾM KẾT QUẢ, VÀ WATCH.

Ngay bên dưới môi trường phát triển bạn có thể tìm thấy một tab cửa sổ với nhiều khung dùng trình bày các tác vụ (Task), và kết xuất (Output) từ tiến trình biên dịch, theo vết thông tin gỡ lỗi (debug), các kết quả tìm kiếm... (Hình 1.5-13)



Hình 1.5-13 Khung cửa sổ Task, Output, và kết quả tìm kiếm.

9.1. Tasks

Các lập trình làm việc với Visual C++ trước đây sẽ sử dụng những lời chú thích //TODO để cài đặt thêm mã tùy biến của mình. Các chú thích (comment) này xuất hiện trong quá trình Wizard phát sinh mã. Bạn chỉ cần tìm những nơi dấu dấu TODO với trình tìm kiếm tập tin và bạn có thể nhìn thấy mã chương trình đầy đủ như thế nào.

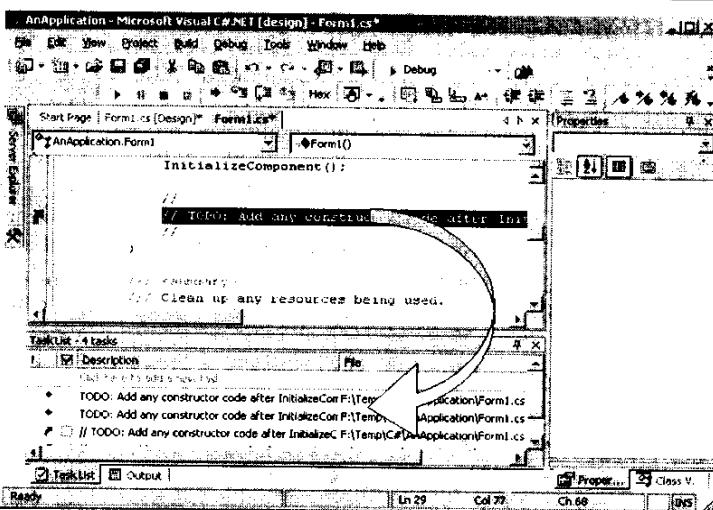
Visual Studio .NET sẽ tự động hóa các tác vụ (task) theo vết tiến trình (process) giúp bạn. Bất cứ khi nào bạn nhập vào lời chú thích //TODO trong chương trình của bạn, IDE sẽ đưa một tác vụ vào danh sách tác vụ. Nhấp đúp chuột trên tác vụ task mà bạn muốn, bạn sẽ được dẫn đến lời chú thích trong mã chương trình. Bạn có thể gắn thêm các thẻ như TODO, HACK, UNDONE vào trong những thẻ tùy biến của bạn, những thẻ này cũng được hiển thị trong task list.

Hình 1.5-14 hiển thị danh sách Task List trong mối quan hệ với một TODO Task trong mã chương trình.

Để thêm thẻ vào trong chính danh sách Task List của bạn, click Tools | Options từ menu chính, và sau đó chọn Environment, Task List từ danh sách cây của những cài đặt tùy chọn trong khung bên trái hộp thoại. Mặc định, những tác vụ được dấu trong cửa sổ Task. Để nhìn thấy tất cả các tác vụ, bạn nhấp ch phải trên khung Task, chọn Show Tasks, và chọn All.

10. CỦA SỔ KẾT XUẤT THÔNG TIN (OUTPUT)

Chọn tab Output sẽ hiển thị cho bạn rất nhiều thông tin. Những thông tin biên dịch (bao gồm những khuyến cáo và thông báo lỗi) sẽ được hiển thị tại đây. Những thông điệp từ quá trình theo vết gỡ lỗi cũng được hiển thị.



Hình 1.5-14 Danh sách tác vụ (Task List).

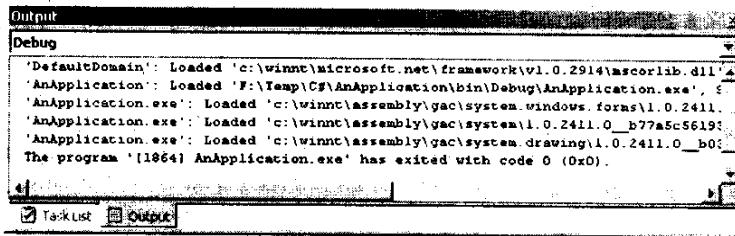
Bạn có thể đưa những thông điệp của riêng mình vào cửa sổ này (cụ thể là khi gỡ lỗi chương trình) bằng cách sử dụng đối tượng System.Diagnostics.Trace. Ví dụ như khi xử lý một sự kiện của bộ định giờ (timer) trong chương trình C#, mã chương trình được hiển thị trong ví dụ 1.5-1

Ví dụ 1.5-1 : Sử dụng đối tượng TRACE để kết xuất thông tin.

```
private void OnTick(
    object sender, System.EventArgs e)
{
    System.Diagnostics.Trace.WriteLine("The timer
        ran !!\n", "Timer");
}
```

Đoạn mã này sinh ra kết quả như hình 1.5-15.

Như bạn có thể nhìn thấy, thông điệp “The timer ran !! ” được hiển thị lặp lại một số lần do kết quả của chương trình đang chạy.

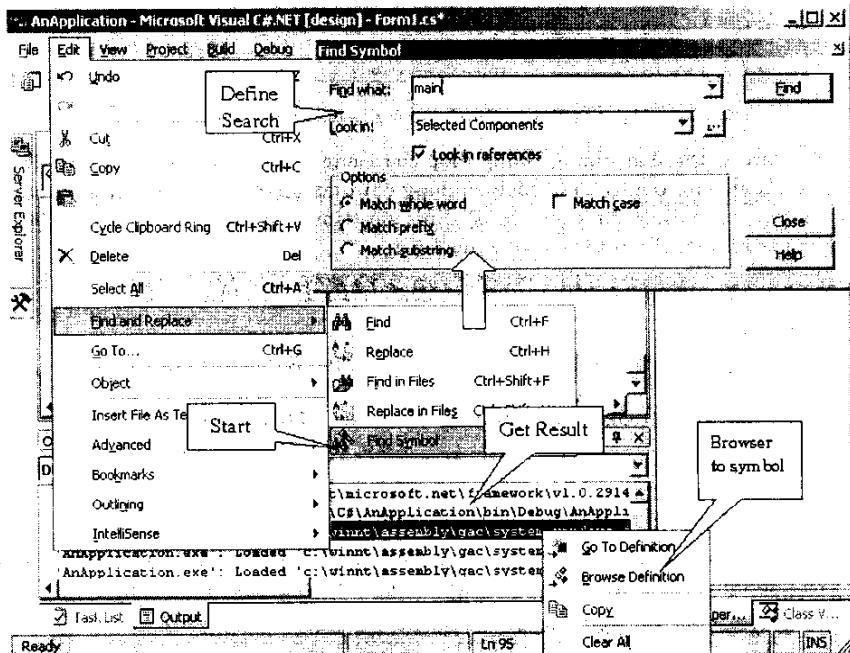


Hình 1.5-15 – Hoạt động theo vết trong cửa sổ Output.

11. TÌM KIẾM CÁC KÝ HIỆU ĐẶC TRƯNG

Nhấn Alt+F12 hoặc chọn Edit | Find and Replace | Find Symbol sẽ hiển thị hộp thoại Find Symbol. Với hộp thoại này bạn có thể tìm kiếm ký hiệu (symbol) trong chính những tập tin ứng dụng của dự án, các tham chiếu (references), hoặc những component mà chương trình sử dụng. Kết quả được hiển thị trong ô Find Symbol Results. Nhấp chuột phải trên một mục (entry) trong ô đó sẽ cho phép bạn đi tới nơi định nghĩa trống khu vực soạn thảo hoặc xem lướt qua những nơi định nghĩa các tham chiếu cho ký hiệu đó. Chọn Browse Definition sẽ hiển thị ô Object Browser trong cửa sổ chính nơi mà bạn có thể nhìn thấy toàn bộ các kế thừa đối tượng cho đối tượng được chọn và những đối tượng khác trong chính nhóm đó.

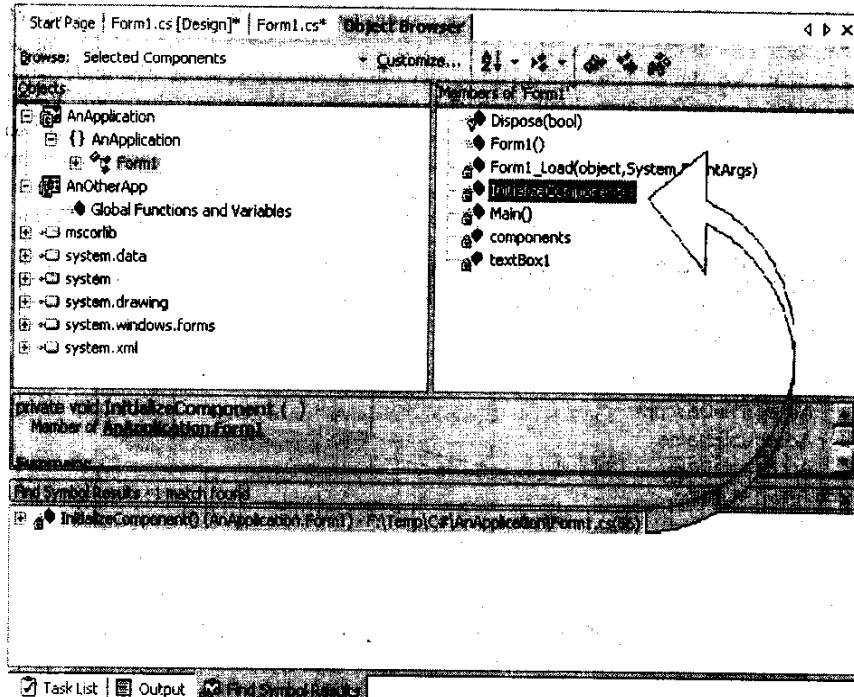
Hình 1.5-16 hiển thị những bước cần làm để tìm một ký hiệu (symbol) trong VS.NET.



Hình 1.5-16 – Sử dụng hộp thoại Find Symbol.

Giao diện của Object Browser hơi khác so với Browser Info trong VS 6.0. c sử dụng Browser Information trong phiên bản trước, dự án cần phải được biên dịch một cách rõ ràng. Object Browser trong VS.NET luôn sẵn sàng, ngay trước khi bạn biên dịch dự án. Nó cũng hữu dụng cho những tác vụ phát triển hàng ngày của bạn. Browser gồm 3 khung – khung Object bên trái, khung Members bên phải, và khung Description ở bên dưới. Hơn nữa, thanh công

(toolbar) ở trên cho phép bạn chọn những tùy chọn (options) khác. Sử dụng Object Browser, bạn có thể tìm hiểu không gian tên, lớp, kiểu, các giao tiếp, kiểu enum, và các cấu trúc struture. Bạn có thể tùy ý sắp xếp các đối tượng trong khung Browser bằng cách sử dụng thanh công cụ của Object Browser để sắp xếp thành viên theo thứ tự abc, theo kiểu dữ liệu, theo đối tượng truy xuất (object access), hoặc nhóm theo kiểu đối tượng. Hình 1.5-17 hiển thị kết quả tìm kiếm phương thức InitializeComponent và kết quả trạng thái của Object Browser.



Hình 1.5-17 – Object Browser.

12. CHỈ MỤC VÀ KẾT QUẢ TÌM KIẾM

Cả hai khung này hiển thị kết quả của việc tìm kiếm theo nội dung, hoặc bởi yêu cầu một chỉ mục đơn giản, hoặc thông qua sự tìm kiếm khá phức tạp. Nhấp chuột trên nút Search Result sẽ đưa bạn đến mục trợ giúp.

13. CỦA SỔ GỠ LỖI (DEBUG)

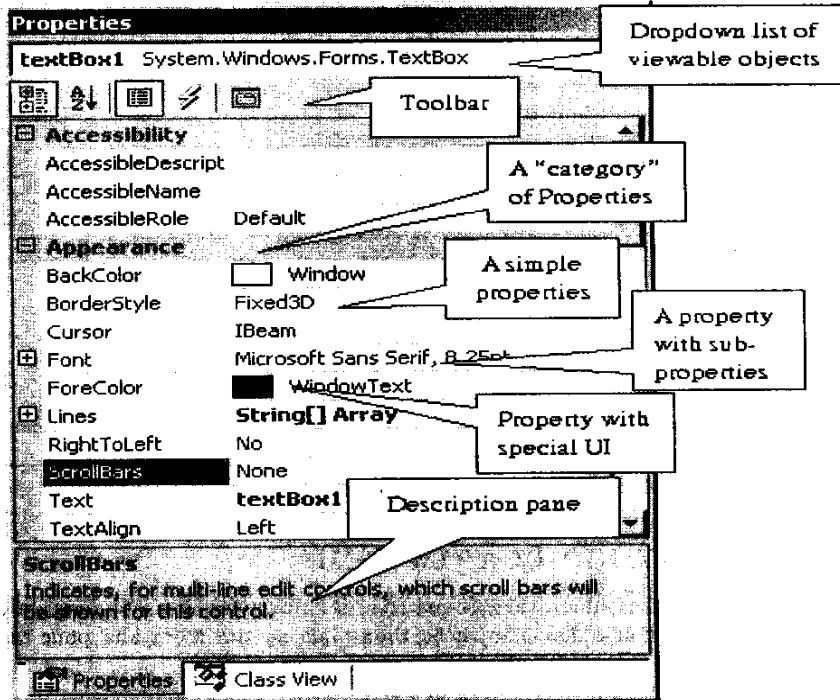
Bạn có thể thấy một số khung cửa sổ khác. Chúng là các khung Breakpoint và cửa sổ dòng lệnh trực tiếp (immediate command window). Chúng tôi sẽ giải thích những khung cửa sổ này trong phần “Gỡ lỗi chương trình”.

PHẦN E : CỬA SỔ THUỘC TÍNH, DYNAMIC HELP, VÀ FAVORITES

Bây giờ chúng ta hãy tìm hiểu những cửa sổ thường xuyên sử dụng nhất trong môi trường soạn thảo IDE.

13.1. Property Browser

Hầu hết các thành phần của .NET được cấu hình, điều khiển, và xem xét thông qua các thuộc tính hệ thống. Các đối tượng bên trong .NET có khả năng hiển thị những đối số của chúng thông qua những đặc tính riêng, những đặc tính này được gắn vào những thuộc tính của chúng. Nếu bạn quen thuộc với Visual Basic, thì thuộc tính lưới (property grid) sẽ làm cho bạn cảm thấy dễ chịu. Nếu bạn là người lập trình C++ kỳ cựu, thì phân hệ thống thuộc tính (property system) có thể tốn một ít thời gian để làm quen với nó. Hình 1.5-18 hiển thị khung cửa sổ Properties được dùng để chỉnh sửa các thành phần điều khiển của Window Forms.

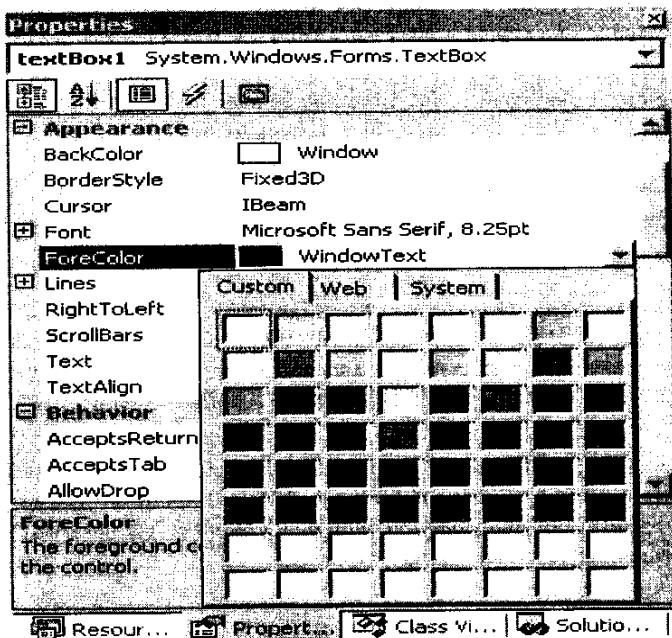


Hình 1.5-18 Property Browser.

Khung Property Browser hiển thị một ô lưới gồm tên và giá trị của những mục có thể nhóm lại tùy ý thành những phân loại. Một số mục có thêm thuộc tính con. Những thuộc tính này có thể được hiển thị nhiều thông tin hơn bằng cách

nhấn vào biểu tượng dấu + bên trái. Tương tự, toàn bộ những phân loại (categories) có thể mở và thu lại bằng cách nhấn vào biểu tượng + và - thích hợp. Một cách tổng quát, thuộc tính lưới có quan hệ với những giá trị đơn giản riêng biệt, ví dụ, một con số hoặc một chuỗi giá trị. Property Browser còn có khả năng biến đổi các kiểu thông tin (ví dụ một giá trị màu dạng số RGB) thành phần tử giao diện (UI – user interface), chẳng hạn như những mẫu màu trong hình 1.5-18 hiển thị một thuộc tính Cursor và giá trị hiện hành được chọn là Ibeam.

Property Browser không chỉ hiển thị thông tin ở dạng thân thiện người dùng mà còn hiển thị theo dạng gợi nhớ. Nó cho phép bạn chỉnh sửa theo cách riêng của mình. Những mẫu màu hiển thị trong hình 1.5-18 có thể được đổi lại sử dụng bộ chọn màu (color picker) chuyên dụng xuất hiện khi bạn muốn thay đổi giá trị màu tương ứng. Hình 1.5-19 hiển thị Property Browser đang thay đổi một trong những giá trị màu.



Hình 1.5-19 – Thay đổi giá trị màu với Property Browser.

Còn một số trình soạn thảo chuyên dụng dành cho những thuộc tính khác của đối tượng. Ví dụ thuộc tính Anchor và Dock của các phần tử điều khiển Windows Form có những trình soạn thảo rất trực giác, theo giao diện đồ họa.

Thanh công cụ của Property Browser (trên đầu cửa sổ Property Browser) cho phép thay đổi cách nhìn về danh sách thuộc tính. Ví dụ bạn có thể thấy những nút nhấn sau:



Phân loại (Categorized) – Nhóm các thuộc tính theo phân loại như Appearance hoặc Behavior.



Sắp xếp danh sách tên thuộc tính theo thứ tự chữ cái (Aphabetic) thay vì nhóm theo phân loại.



Sự kiện (Events) – Liệt kê tất cả các sự kiện của một component và cho phép bạn tạo ra bộ xử lý sự kiện (event handler) cho chúng. Để tạo hoặc hiệu chỉnh nội dung bộ xử lý cho một sự kiện cụ thể, bạn nhấp chuột trên tên sự kiện trong danh sách. IDE sẽ cho phép bạn đánh vào tên của bộ xử lý hoặc chọn bộ xử lý đã tồn tại từ một danh sách thả xuống. Sau khi đã tạo xong bộ xử lý, bạn có thể nhấp đúp chuột trên nó để di chuyển đến mã chương trình. Nếu bạn muốn tạo bộ xử lý với một tên được chọn trước, đơn giản nhấp đúp chuột trên tên sự kiện (event), IDE sẽ tự tạo một bộ xử lý tương ứng cho sự kiện.



Thông điệp (Messages) – Nút nhấn này sẽ xuất hiện trên thanh công cụ khi khung cửa sổ Class View đang hiển thị ứng dụng Windows C++. Bạn có thể xem, chỉnh sửa, và thêm vào những bộ xử lý thông điệp cho các lớp C++ thông qua Property Browser. Cách thức của việc thêm những bộ xử lý thông điệp vào các lớp C++ tương tự như phần thêm và hiệu chỉnh các bộ xử lý sự kiện đã đề cập trước đây.



Định nghĩa chồng (Overrides) – Khi Class View đang hiển thị các lớp C++, bạn có thể soạn thảo, thêm vào, và xem các thành phần định nghĩa chồng (gồm phương thức, thuộc tính) cho một lớp được chọn trong Class View bằng cách nhấp chuột vào nút nhấn này trên thanh công cụ của Property Browser.



Trang thuộc tính (Property pages) – Khi bạn sử dụng Solution Explorer để xem dự án (project) hoặc những cấu hình của Solution, thanh công cụ của Property Browser sẽ hiển thị nút nhấn này. Sử dụng nó để hiển thị hộp thoại với những thuộc tính sẵn sàng cho cấu hình bạn cần.



Thuộc tính (Properties) – Nếu bạn đã sử dụng Property Browser để hiển thị những sự kiện hoặc thông tin khác, thì bạn có thể chuyển ngược lại tới cách xem thuộc tính bình thường bằng nút nhấn này.

14. TRÌNH TRỢ GIÚP ĐỘNG (DYNAMIC HELP)

Hệ trợ giúp động (Dynamic Help) mới trong Visual Studio .NET tạo ra sự khác biệt lớn đối với người lập trình. Bây giờ bạn không cần tìm kiếm cơ sở dữ liệu trợ giúp khi đang lập trình, bởi vì Dynamic Help luôn luôn giữ một danh sách thông tin được cập nhật liên quan tới những hành động mà bạn đang thao tác. Thậm chí nó còn giữ thông tin về những cửa sổ mà bạn đang sử dụng trong IDE.

Ví dụ, nếu bạn chọn tab Dynamic Help trong phần E và sau đó chọn Class View, một mục trợ giúp Help về việc sử dụng Class View sẽ sẵn sàng bất kỳ lúc nào để bạn tham khảo.

Kinh nghiệm cho thấy Dynamic Help yêu cầu nhiều công sức xử lý, bởi vì nó tìm kiếm và đổi chiếu dữ liệu hầu như với mọi thao tác bấm phím, di chuyển chuột... Bạn có thể tùy chọn tắt Dynamic Help.

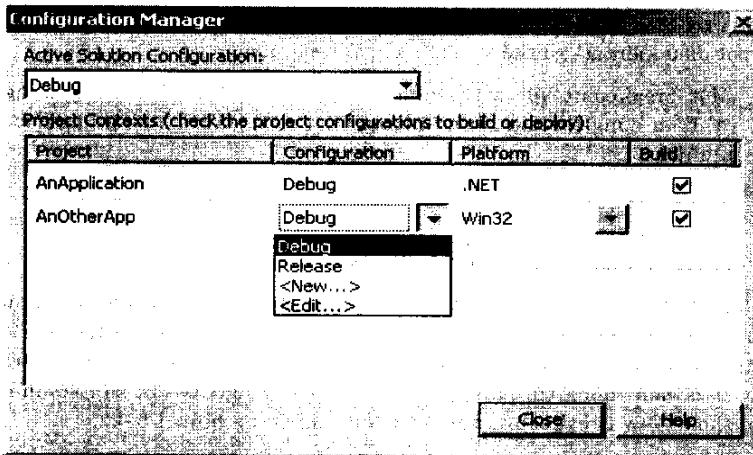
15. CỦA SỔ FAVORITE

Đây là nơi chứa danh sách các địa chỉ Internet ưa thích (gọi là Favorite) của bạn. Nó cho phép bạn chọn một liên kết thường sử dụng nhất và nội dung của trang tài liệu do liên kết trỏ đến sẽ hiển thị trong cửa sổ soạn thảo chính tương tự như cửa sổ trình duyệt Internet Explorer.

16. CHƯƠNG TRÌNH GỠ LỖI (DEBUGGING).

Ngay cả người lập trình tuyệt vời cũng có thể tạo ra lỗi, vì vậy có một chương trình gỡ lỗi tốt là điều rất sức quan trọng. Việc gỡ lỗi với Visual Studio được thực hiện khá dễ dàng, và bạn có khá nhiều tùy chọn cho việc ngừng chương trình và xem xét mã chương trình. Cách đơn giản nhất để gỡ lỗi (debug) mã chương trình là khi bắt đầu quá trình biên dịch. Chương trình gỡ lỗi cũng có thể bẫy một điều kiện lỗi, hoặc bạn có thể gắn chương trình gỡ lỗi (debugger) vào ngay tiến trình đang chạy. Chúng ta hãy xem xét trường hợp đơn giản đầu tiên.

Để gỡ lỗi, chương trình cần phải được dịch ở chế độ debug. Để chọn kiểu debug, bạn sử dụng Debug | Configuration Manager từ menu chính. Hình 1.5-21 là cửa sổ Configuration Manager



Hình 1.5-21 – Hộp thoại Configuration Manager.



Sau khi cấu hình debug được chọn, bước kế tiếp là chỉ ra nơi bạn muốn chương trình ngừng (stop) để kiểm soát. Điều này được thực hiện bằng cách đặt điểm dừng (breakpoint) ngay trong mã nguồn. Bạn cũng có thể làm điều này bằng cách chọn dòng mã nơi mà bạn muốn ngừng và nhấn F9. Một dấu đố sẽ xuất hiện ở mép của trình soạn thảo mã (code editor), lúc đó breakpoint đang ở trạng thái kích hoạt (active). Bạn cũng có thể đặt breakpoint bằng cách nhấp chuột vào mép xám bên trái cửa sổ soạn thảo, hoặc nhấn Ctrl+B.

Để kích hoạt khung cửa sổ Debug, chọn Debug | Windows | Breakpoint hoặc nhấn Ctrl+Alt+B.

Sau khi breakpoint được đặt thành công, chương trình có thể chạy với mong đợi của việc đánh trúng điểm dừng (break). Nhấn F5 để thực thi mã chương trình trong kiểu debug.

ĐIỂM DỪNG BREAKPOINT

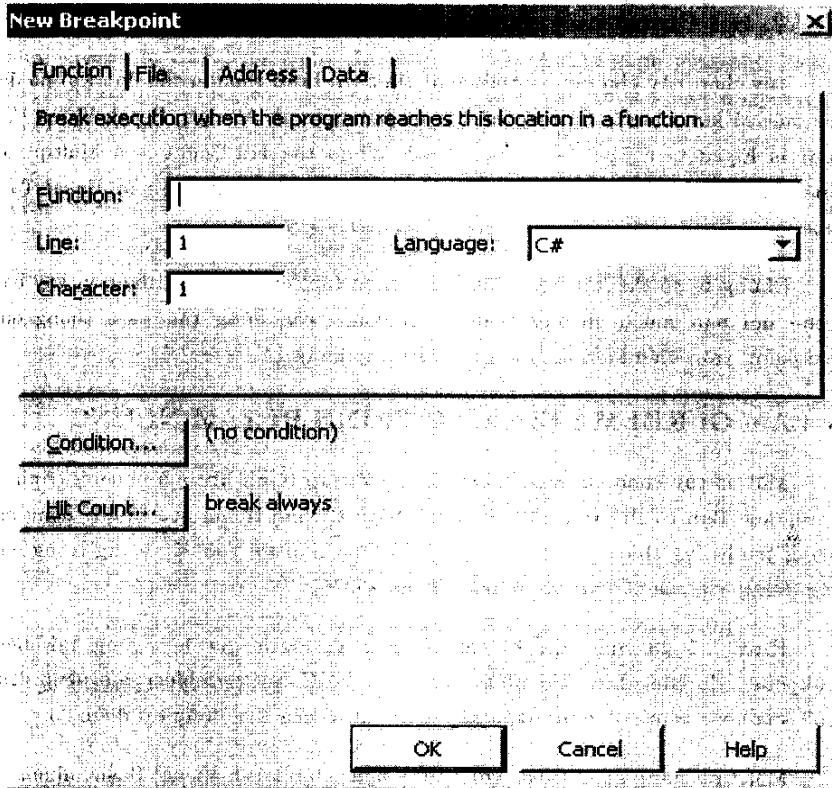
Một vòng tròn đố rỗng là một breakpoint không kích hoạt. Breakpoint có thể được kích hoạt (active) hoặc ở dạng không kích hoạt (deactive). Chọn Debug | Windows | Breakpoints hoặc nhấn Ctrl+Alt+B. Một điểm dừng breakpoint với một dấu hỏi bên trong cho biết nó có thể không có hiệu lực, bởi vì tiến trình mà nó hợp tác chưa chạy. Ví dụ, điều này xảy ra khi bạn đặt breakpoint trong ứng dụng A và sau đó chạy ứng dụng B từ cùng Solution. Ngay khi tiến trình chứa breakpoint được tải vào, điểm dừng breakpoint sẽ chuyển sang trạng thái kích hoạt.

Breakpoint với một dấu chấm than không thể đặt, bởi vì có một lỗi. Có lẽ vị trí breakpoint thì không hợp lệ hoặc breakpoint có thể bị tùy thuộc vào một điều kiện mà có thể không bao giờ thoả mãn.

Một breakpoint với một chấm trong nó thì được đặt trong trang ASP, và điểm dừng (break) được ánh xạ tới trang HTML tương ứng được phát sinh bởi ASP.

17. CÀI ĐẶT BREAKPOINT NÂNG CAO.

Thỉnh thoảng bạn cần nhiều hơn một nơi để ngừng và kiểm tra mã nguồn. Để tạo ra một breakpoint mới, chọn dòng mã và nhấn F9 hoặc Ctrl+B. F9 sẽ t động diền vào hàm và dòng mã (line of code) nơi mà bạn muốn ngừng. Ctrl+B sẽ yêu cầu bạn diền thông tin bằng tay. Khi bạn sử dụng Ctrl+B, bạn sẽ nhìn thấy hộp thoại New Breakpoint, như được hiển thị trong hình 1.5-22.



Hình 1.5-22 – Hộp hội thoại New Breakpoint.

Có 2 nút nhấn trên hộp hội thoại này là Condition và Hit Count. Để chỉnh sửa breakpoint đã được đặt, tìm breakpoint trong ô Breakpoints và nhấp chuột phải trên nó. Hành động này để bạn chọn những Properties từ menu ngữ cảnh, một hộp chỉnh sửa Breakpoint xuất hiện, hộp này cũng sẽ có những tùy chọn tương tự.

17.1. Conditional Breakpoints

Chọn tùy chọn này cho phép bạn quyết định khi nào bạn muốn chương trình treo (halt). Ví dụ, bạn muốn có một chương trình được ngừng khi một biến đạt tới một giá trị cụ thể nếu nó được thay đổi lần cuối thông qua breakpoint.

17.2. Hit Counts.

Tùy chọn này cho phép breakpoint thực thi bình thường cho đến khi nó được một số lần nào đó. Bạn có thể đặt nó bằng Break Always, Break When Hit Count is Equal to a Specific Value, Break When the Hit-Count is a Multiple Some Number, hoặc Break When the Hit-Count is Greater Than or Equal To Value.

Chúng ta có thể kết hợp những tùy chọn (option) này với nhau, vì vậy bạn có thể nói bạn muốn chương trình gián đoạn trên dòng thứ năm thông qua breakpoint, trên điều kiện mà biến lớn hơn hay bằng 14.

18. LÀM GÌ KHI MÃ CHƯƠNG TRÌNH BỊ TẠM NGỪNG

Một số tùy chọn cũ quen thuộc với người lập trình khi mã chương trình tạm ngừng. Bạn có thể xem những biến (thỉnh thoảng thông qua những dòng ra chương trình), và thậm chí gửi những lệnh trực tiếp tới các đối tượng trong ứng dụng thông qua cửa sổ lệnh (command window).

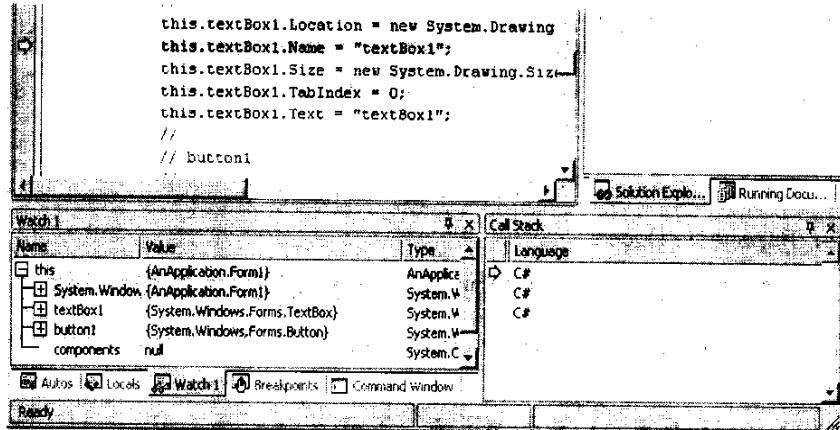
Hình 1.5-23 hiển thị bộ xử lý dịch vụ định giờ bị dừng tại điểm breakpoint. Nó cũng hiển thị những biến cục bộ (local variables) cho phép điều chỉnh sửa hoặc xem nội dung ngăn xếp (call stack) của ứng dụng tại điểm đó.

Một tính năng quen thuộc với những người lập trình Visual Basic, nhưng lẽ không quen thuộc cho những người lập trình C++, là cửa sổ dòng lệnh (command window). Tab này (được định vị ở đáy góc phải ở đáy màn hình) cho phép bạn viết những lệnh trực tiếp tác động lên các đối tượng bị treo trong trình gỡ lỗi (debugger). Trong trường hợp mã chương trình của bộ định giờ bạn có thể `this.timer1.Enabled = false <enter>` và bộ định giờ sẽ bị vô hiệu hóa. Việc sử dụng cửa sổ dòng lệnh, bạn có thể dễ dàng thay đổi nội dung hoặc cấu hình những đối tượng trong mã chương trình để thử những kịch bản giúp bạn dễ dàng sửa những lỗi hiện có trong chương trình.

19. GẮN TRÌNH GỠ LỖI (DEBUGGER) VÀO MỘT TIỀN TRÌNH (PROCESS)

Đây là tính năng hữu dụng nếu bạn đang gỡ lỗi một tập hợp những ứng dụng từ cùng một Solution hoặc có lẽ đang gỡ lỗi một DLL. Bạn có thể chạy ứng dụng, chạy trình gỡ lỗi, và chọn để sửa chữa bên trong chương trình bằng cá

ngừng nó lại. Bạn sẽ được hỏi loại ứng dụng mà bạn muốn gỡ lỗi. Những chọn lựa thông thường là CLR-base App, Microsoft T-SQL, Native code program, Possibly C++, hoặc script. Việc gắn tới một tiến trình khác cũng có thể được làm trên máy từ xa (remote machine) nếu bạn được phép truy xuất nó.



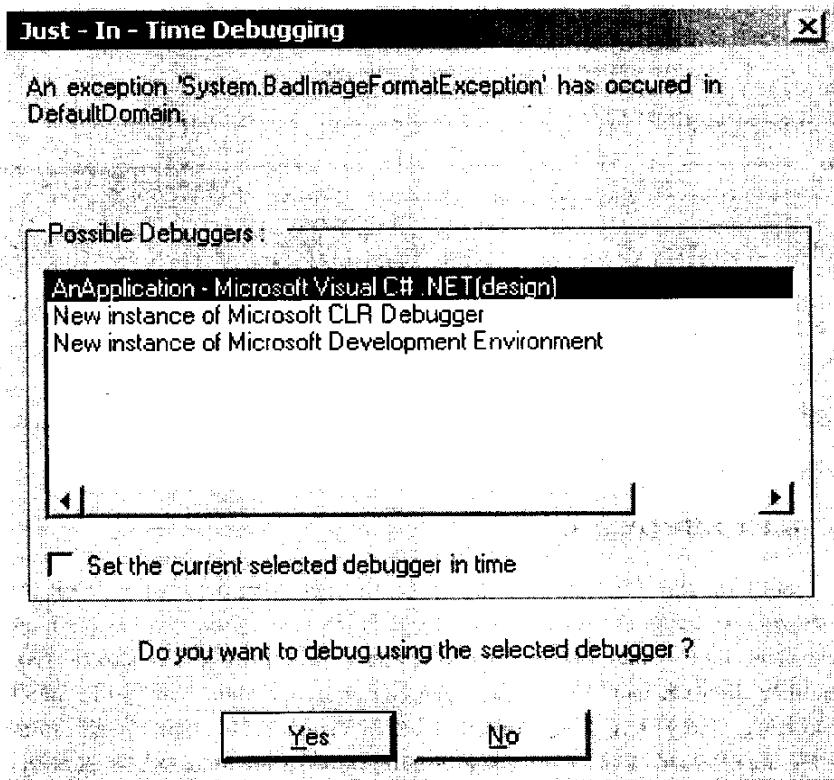
Hình 1.5-23 – Mã chương trình ngừng tại điểm breakpoint.

20. JIT DEBUGGING

Just in time (JIT) debugging xảy ra khi một đoạn mã chương trình (code) được quăng ra ngoại lệ (exception) không xử lý và không được gắn với trình gỡ lỗi (debugger). Bạn có một vài tùy chọn cho việc gỡ lỗi khi điều này xảy ra. Bạn sẽ được hiện diện với một hộp hội thoại mà cho phép bạn chọn trình gỡ lỗi nào bạn muốn dùng. Chú ý rằng dù là bạn không có cài Visual Studio, bạn vẫn sẽ có những tùy chọn gỡ lỗi. Nếu bạn có cài Visual Studio, thì cách tốt nhất là bạn chọn Visual Studio làm trình gỡ lỗi cho bạn, bởi vì những tính năng của VS.NET nâng cao hơn những chọn lựa khác. Hình 1.5-24 hiển thị hộp hội thoại JIT debug sau khi một ngoại lệ phát sinh.

Ở đây, bạn nhìn thấy những tùy chọn được dùng cho trình gỡ lỗi đang chạy (đây là trình gỡ lỗi đang được dùng để sửa chữa ứng dụng AnApplication được sử dụng trong chương này), để sử dụng trình gỡ lỗi CLR, hoặc một thể hiện mới của Visual Studio .NET. Chọn trình gỡ lỗi và nhấp chuột trên nút nhấn Yes sẽ chạy trình gỡ lỗi được chọn, trình gỡ lỗi này sẽ dừng lại tại điểm mà lỗi bị bẫy. Nếu chương trình được biên dịch với cấu hình gỡ lỗi (debug configuration) gây ra ngoại

lệ (exception), bạn nên xem mã chương trình và đi từng bước qua mã để tìm lỗi. Nếu ứng dụng bị lỗi được biên dịch với chế độ ứng dụng hoàn chỉnh (release), JIT có thể không hữu dụng lắm bởi vì có lẽ bạn sẽ được hiện diện với mã máy X86 (assembly code) để gỡ lỗi.



Hình 1.5-24 – Hộp hội thoại Just – In – Time Debugging.

21. KẾT CHƯƠNG.

Việc giới thiệu nhanh về Visual Studio .NET này sẽ cho phép bạn khai phá IDE và sử dụng tất cả những tính năng chủ yếu. Chương này chúng ta chỉ giới thiệu tổng quan về .NET. Phần 2 của cuốn sách này chúng ta sẽ bắt đầu học ngôn ngữ C#.

Phân II

NGÔN NGỮ C#

Trong phân này:

- ♦ Các vấn đề cơ bản của C#
- ♦ C# Nâng cao

Chương 2.1

CÁC VẤN ĐỀ CƠ BẢN CỦA C#

Các vấn đề chính sẽ được đề cập đến

- ✓ Hệ thống kiểu trong C#
- ✓ Các khái niệm lập trình
- ✓ Mảng (Array)
- ✓ Cấu trúc (Struct)
- ✓ Lớp (Class)
- ✓ Giao diện (Interface)
- ✓ Mô hình chuyển giao (Delegate)

C# mô tả một ngôn ngữ hiện đại hướng đối tượng. Nó được thiết kế để chú ý đến việc dien đạt C++ theo kiểu lập trình và phát triển nhanh ứng dụng RAD (Rapid Application Development) chẳng hạn như Microsoft Visual Basic, Delphi, C++ Builder. C# được xây dựng và kiến trúc bởi Anders Hejlsberg, người đã viết nên trình biên dịch Pascal và đã có rất nhiều đóng góp cho ngôn ngữ Delphi cũng như Java. Và do đó sự tiến triển của C# chịu ảnh hưởng bởi các ngôn ngữ như C++, SmallTalk, Java, và các ngôn ngữ khác.

Phần này sẽ khảo sát ngôn ngữ C# cùng với ngữ nghĩa, và văn phạm là các thành phần chính của ngôn ngữ. Thiết kế chu đáo của C# cho phép bạn với tư cách là người phát triển tập trung vào các tác vụ xây dựng các ứng dụng theo nhu cầu công việc của bạn hơn là vào những kỹ thuật lắt léo của ngôn ngữ lập trình.

Điều quan trọng trước tiên đó là chúng ta cần tìm hiểu những khái niệm và vấn đề cơ bản trước khi đi vào chính ngôn ngữ. Trước hết, mọi thứ trong C# đều là đối tượng. Không như các ngôn ngữ thủ tục, ngôn ngữ C# không quan tâm đến dữ liệu toàn cục hay các hàm toàn cục. Tất cả dữ liệu và phương thức được chứa trong hoặc là khai báo cấu trúc struct hoặc là class. Đây là khái niệm chính trong bất kỳ ngôn ngữ hướng đối tượng nào. Tất cả dữ liệu và các phương thức thao tác trên dữ liệu cần phải được đóng gói như một đơn vị chức năng. Các đơn vị chức năng này là những đối tượng có thể sử dụng lại, chúng độc lập và có thể tự hoạt động.

Mặc dù mục tiêu của phần này không phải là để khảo sát các mẫu thiết kế (design pattern) hướng đối tượng và công nghệ phần mềm, nhưng bạn cũng cần hiểu những khái niệm chính về thiết kế và cài đặt hướng đối tượng. C# sẽ cho phép cả những người lập trình có trình độ cao và cho cả những người mới vào nghề viết một chương trình có thể tin cậy được với công sức tối thiểu.



1. HỆ THỐNG KIẾU TRONG C#

Trong kiến trúc .NET, có một hệ thống kiểu chung được sử dụng để cho phép tất cả các ngôn ngữ nhầm vào môi trường .NET thao tác liên kết lẫn nhau. C# sử dụng hệ thống kiểu cơ sở này. Chúng ta đã nói rằng mọi thứ trong C# là đối tượng. Điều này hầu như đúng cho cả kiểu dữ liệu. Các kiểu dữ liệu đơn nguyên như int, char không được xem là đối tượng, lý do để bảo đảm tính hiệu quả. Bởi vì các đối tượng được cấp phát trên heap và được quản lý bởi GC (bộ dọn rác), điều này sẽ đưa ra một ý nghĩa quan trọng về toàn bộ mối quan hệ của các kiểu cơ sở chẳng hạn như int và char cùng với những kiểu đối tượng mới. Vì lý do này, C# cài đặt các kiểu đơn nguyên (primitive type) ở dạng struct, được coi như là các kiểu giá trị (value type). Trong C#, kiểu giá trị được cấp phát trên stack (ngăn xếp), thời gian sống của chúng bị giới hạn trong phạm vi mà chúng khai báo.

Bảng 2.1-1 trình bày một danh sách các kiểu có sẵn trong C#

Trừ kiểu chuỗi String ra, tất cả các kiểu được trình bày trong bảng 2.1-1 được cài đặt như một cấu trúc struct. Kiểu String trong thực tế được cài đặt như một lớp đóng dấu (sealed). Một lớp sealed là một lớp không cho phép kế thừa và vì thế nó là đầu cuối của mảng xích kế thừa.

Bảng 2.1-1 Các kiểu của C#

Kiểu	Mô tả
object	Lớp cơ sở của tất cả các đối tượng trong C#
string	Dãy các ký tự ở dạng Unicode
sbyte	Nguyên có dấu 8-bit
short	Nguyên có dấu 16-bit
Int	Nguyên có dấu 32-bit
Long	Nguyên có dấu 64-bit
Byte	Nguyên không dấu 8-bit
Ushort	Nguyên không dấu 16-bit
Uint	Nguyên không dấu 32-bit
Ulong	Nguyên không dấu 64-bit
Float	Số chấm động có độ chính xác đơn
Double	Số chấm động có độ chính xác đôi
Bool	Kiểu logic – true hay false
Char	Ký tự Unicode
decimal	Số thập phân có 28 chữ số có nghĩa

1.1. Các kiểu giá trị trong lập trình

Khi một kiểu đơn nguyên (int, char) được khai báo, C# yêu cầu biến đó phải được khởi tạo trước khi biến được sử dụng. Trong C++, giá trị của một biến không khai báo là không xác định; trong C# cũng áp dụng nguyên tắc giống như vậy. Sự khác biệt trong C++, biến có thể được sử dụng với các kết quả không được biết.

Khai báo một biến trong C# có cú pháp như sau:

Type variable-name [=initialization]

Ở đây *type* là kiểu biến, và *variable-name* có thể chứa chữ cái, số, và dấu gạch dưới. Tuy nhiên, tên biến phải bắt đầu bằng hoặc là dấu gạch dưới hoặc là chữ cái và không được bắt đầu bằng số.

Dưới đây là các ví dụ về khai báo biến:

```
int _999; // hợp lệ
int a_dog; // hợp lệ
int 123_go; // không hợp lệ
```

Chú ý: C# có cả các ký hiệu ghi chú // và /*_*/. Dấu // biểu thị tất cả các văn bản đứng sau nó trên dòng hiện hành là ghi chú. Dấu /*_*/ được sử dụng cho ghi chú có nhiều dòng.

C# cũng tuân thủ nghiêm ngặt việc kiểm tra kiểu và phép gán. Trong C++, bạn có thể khai báo một kiểu số nguyên không dấu và sau đó gán vào biến một giá trị là -1. Trình biên dịch C# sẽ lập tức nhận ra phép gán đó và tạo ra một lỗi biên dịch chỉ ra phép gán không hợp lệ.

```
unsigned int cpp_fudge = -1 // hợp lệ trong C++
uint csharp_fudge = -1 // có lỗi trong C#
```

Lỗi được sinh ra bởi trình biên dịch của C# sẽ cho biết rằng giá trị -1 không thể được chuyển đổi thành một số nguyên không dấu. Đó chính là tính nghiêm ngặt!

1.1.1. Struct (cấu trúc)

Trong các ngôn ngữ chẵng hạn như C hay C++, không thể tạo một kiểu đơn nguyên được. Các kiểu đơn nguyên như là các thực thể mà việc cài đặt chúng trước đây được xem là một bí mật. C# cài đặt các kiểu đơn nguyên không khác gì các struct đơn giản. Điều này muốn nói rằng các lập trình viên có thể tạo ra các kiểu riêng của mình và xử lý chúng theo cùng một cách như các kiểu đơn nguyên của C#.

Khi tạo một struct, điều quan trọng là phải cài đặt những cái thật cần thiết. Xét cho cùng, nếu chức năng thêm vào là cần thiết trong việc cài đặt, thì việc cài đặt một thực thể như là một đối tượng đầy đủ sẽ tốt hơn. Struct thường được dùng

để biểu diễn các mẫu dữ liệu nhỏ mà thường có thời gian sống bị hạn chế và thường không cần thiết để giữ lại lâu trong bộ nhớ.

Ngược lại với C++, các struct và class (lớp) là hai thực thể khác nhau hoàn toàn. Trong C++, điều khác nhau duy nhất giữa struct với class là thành phần mặc định có thể thấy được. Trong C#, các struct và class dùng chung các điều tương tự ở mặt ngoài, nhưng chúng không có quan hệ và thật sự có nhiều khác biệt.

Một struct trong C# không thể kế thừa từ một struct khác hay class, tuy nhiên một struct có thể cài đặt một hay nhiều interface (giao tiếp). Các struct có thể chứa các thành phần dữ liệu và các phương thức. Một struct không thể có một phương thức khởi dụng (constructor) không có đối số. Tất cả các struct đều chứa một constructor ẩn mà constructor đó chịu trách nhiệm khởi tạo tất cả các thành phần dữ liệu với các giá trị mặc định của chúng. Tuy nhiên, một struct có thể định nghĩa một construct nhận đối số. Một construct trong C# thì y như một constructor trong C++ hay Java. Điểm khác biệt lớn giữa struct và class là các struct đó là các kiểu giá trị được cấp phát trên stack và không được coi là tham chiếu.

Dưới đây là một cú pháp khai báo struct:

```
Struct name {
    [access-modifier] member;
}
```

Cú pháp này tương tự với cả C++ lẫn Java về khai báo và cấu trúc. Tín thấy được của biến thành phần **member** phụ thuộc vào từ khoá hay bối cảnh **private** trong C#. Các thành phần trong struct có thể có các access-modifier (các bối cảnh truy cập) được gắn vào chúng như: public, private, internal. Vì một struct không thể cho một struct khác kế thừa nó, nên từ khoá **protected** không sử dụng trong struct; nếu được sử dụng, trình biên dịch C# sẽ lập tức chỉ ra lỗi.

Ví dụ 2.1-1 cho thấy khai báo một struct có tên Fraction. Chú ý rằng các truy xuất một thành phần là sử dụng toán tử chấm, điều này có thể được thấy dòng 13 và 14. Trong C#, tất cả các việc truy xuất thành phần đều thông qua toán tử chấm bất kể đó là struct hay là class.

Ví dụ 2.1-1 Một struct đơn giản

```
1://File      : part02_01.cs
2://Author   : Richard L. Weeks
3://Purpose  : Khai báo một cấu trúc đơn giản
4:
5:struct Fraction {
6:    public int numerator;
7:    public int denominator;
8:}
9:
```

```

10: public class StructTest {
11:     public static void main() {
12:         Fraction f;
13:         f.numerator      = 5;
14:         f.denominator= 10;
15:     }
16: }
```

Các kiểu giá trị cũng có một toán tử gán được xây dựng bên trong để thực hiện việc sao chép tất cả các thành phần dữ liệu. Trong C++, chức năng này có được bằng cách cài đặt toán tử gán và viết các mã lệnh cần thiết để sao chép các thành phần dữ liệu. Ví dụ 2.1-2 sử dụng các toán tử gán được xây dựng bên trong C# để sao chép giá trị của một struct này vào một struct khác có cùng một kiểu.

Ví dụ 2.1-2 Phép gán struct

```

1://File   : part02_02.cs
2://Author : Richard L. Weeks
3://Purpose: Khai báo một cấu trúc đơn giản
4:
5:using System;
6:
7:struct Fraction {
8:    public int numerator;
9:    public int denominator;
10:
11:    public void Print() {
12:        Console.WriteLine("{0}/{1}",
13:                           numerator,denominator);
14:    }
15:
16:
17:    public class StructTest {
18:        public static void Main() {
19:
20:            Fraction f;
21:            f.numerator      = 5;
22:            f.denominator= 10;
23:            f.Print();
24:
25:            Fraction f2 = f;
26:            f2.Print();
27:
28:            //sửa đổi thể hiện f2 của struct
29:            f2.numerator = 1;
30:
31:            f.Print();
32:
```

```

32:         f2.Print();
33:     }
34: }

```

Ví dụ 2.1-2 mở rộng việc cài đặt struct Fraction bằng cách thêm vào phương thức Print ở dòng 11. Phương thức Print sử dụng phương thức Console.WriteLine để hiển thị giá trị hiện hành của Fraction.

Để minh họa toán tử gán được cung cấp bởi C# cho các struct, hai thể hiện của Fraction được khai báo. Dòng 25 khai báo một biến f2 và khởi tạo nó với biến f có kiểu Fraction. Khi f2.Print() được gọi, kết xuất 5/10 sẽ được hiển thị y như gọi f.Print().

Điều quan trọng bạn cần nhận thấy rằng một sự sao chép đã xảy ra và không phải là một phép gán tham chiếu của f vào f2. Khi f2 được sửa đổi trên dòng 29, lời gọi các phương thức Print kế tiếp hiển thị hai giá trị khác nhau. Gọi f.Print() vẫn cho ra kết xuất 5/10, còn gọi f2.Print() bây giờ kết xuất sẽ là 1/10.

1.2. Các kiểu tham chiếu

Một cấu trúc class (lớp) là một ví dụ về kiểu tham chiếu trong C#. Các class (lớp) có thể được xem như đàn anh đối với các struct (cấu trúc) của C#. Việc đếm tham chiếu có nghĩa là bất cứ kiểu tham chiếu nào muốn tồn tại chỉ cần ở đó còn lại một vài tham chiếu còn hoạt động (active reference) đối với thực thể. Trong mô hình COM chuẩn, việc đếm tham chiếu (reference counting) có thể thấy được ở các phương thức AddRef và Release của một đối tượng COM. Khi tham chiếu cuối cùng được giải phóng trên thể hiện của đối tượng, đối tượng thực hiện các bước cần thiết để dọn dẹp mọi thứ.

Thật may mắn, C# đã được tritu tượng hóa hết tất cả các chi tiết hóc búa về việc đếm tham chiếu (reference counting). GC chịu trách nhiệm thu dọn bộ nhớ đang được sử dụng bởi các lớp và các giao tiếp interface không được tham chiếu (unreferenced). Khi một tham chiếu của đối tượng đếm đến số không, GC sẽ gọi phương thức Finalize của đối tượng, phục hồi lại bộ nhớ, và trả nó về vùng heap ứng dụng chung. Phương thức Finalize tương tự với các khái niệm destructor trong C++.

Một tham chiếu có được theo hai cách: khi một thể hiện của một kiểu tham chiếu được tạo ra và khi thực hiện một phép gán. Nhớ rằng khi toán tử gán đã được sử dụng chung với các kiểu giá trị, một bản sao của kiểu giá trị đã được tạo. Điều này không ở trường hợp khi toán tử gán được sử dụng với các kiểu tham chiếu. Việc biến đổi cấu trúc (struct) Fraction thành một lớp là thay đổi hành vi của toán tử gán, như được trình bày ở ví dụ 2.1-3.

Ví dụ 2.1-3 Các kiểu tham chiếu

```

1://File      :part02_03.cs
2://Tác giả   :Richard L. Weeks

```

```

3: //Mục đích : Các kiểu tham chiếu
4:
5: using System;
6:
7: //Lớp mô tả một kiểu tham chiếu trong C#
8: class Fraction {
9:     public int numerator;
10:    public int denominator
11:
12:    public void Print() {
13:        Console.WriteLine("{0}/{1}", numerator,
14:                           denominator);
15:    }
16:
17:
18:    public class ReferenceTest {
19:        public static void Main() {
20:
21:            Fraction f = new Fraction();
22:            f.numerator = 5;
23:            f.denominator = 10;
24:            f.Print();
25:
26:            Fraction f2 = f; //f2 là một tham chiếu đến
27:                         //f và không là một bản sao!!!
28:
29:            //sửa đổi thể hiện f2. Chú ý rằng f cũng được
30:            //thực hiện.
31:            f2.numerator = 1;
32:            f.Print();
33:            f2.Print();
34:        }
35:    }

```

Chỉ có hai thay đổi được thực hiện ở ví dụ 2.1-2 để tạo ví dụ 2.1-3. Thay đổi đầu tiên là việc khai báo Fraction ở một lớp để thay cho một struct. Thay đổi nhó này có nghĩa là để thay cho Fraction đang được cấp phát ở ngăn xếp (stack), bây giờ nó sẽ được tạo ở vùng heap và tham chiếu được đếm.

Thay đổi kế tiếp liên quan đến cách thức tạo một thể hiện Fraction. Chú ý rằng dòng 21 đã được sửa đổi. Để tạo một thể hiện có kiểu tham chiếu, từ khóa new phải được sử dụng. Bây giờ, chúng ta phải tạo một thể hiện của lớp Fraction để khai báo biến f. Trừ phi bạn đang dùng khai báo riêng, biến f sẽ được xem như một biến không được khởi tạo.

Những thay đổi ở ví dụ 2.1-2 làm ảnh hưởng đến toàn bộ ngũ nghĩa của đoạn mã ở ví dụ 2.1-3. Chú ý rằng sự khai báo của biến f2 ở dòng 26 lúc này được xem như một tham chiếu đến biến f. Điều này có nghĩa là biến f2 như biến f – f2 không là một bản sao. Đây là nguyên tắc phân biệt giữa các kiểu giá trị và các kiểu tham chiếu. Khi f2 được sửa đổi ở dòng 30, sự sửa đổi giống như vậy là hiển nhiên ở biến f.

Lời gọi f.Print() và f2.Print() sẽ luôn đưa ra cùng một kết xuất. Khi một thay đổi được thực hiện ở f2.numerator, điều đó cũng như việc thay đổi f.numerator. Trong C++, có thể đã định nghĩa một toán tử gán và để điều khiển hành vi của toán tử. Khả năng này không tồn tại ở C#. Toán tử gán không thể được định nghĩa chồng hay được cài đặt lại bởi người lập trình.

1.3. Bao và không bao (box và unbox)

Khái niệm về bao (box) cho phép xử lý kiểu dữ liệu như là kiểu tham chiếu. Thời gian tồn tại là điều cần thiết để một kiểu giá trị được xử lý như một đối tượng, cũng như việc lưu trữ các giá trị vào một mảng hay một vài tập hợp khác.

Khi một kiểu giá trị được bao (box), một thể hiện của đối tượng được tạo trên vùng heap và giá trị của kiểu được chép vào đối tượng đó. Khi điều này xảy ra, đối tượng được bao (boxed object) và kiểu giá trị là hai thực thể khác nhau. Đối tượng không là tham chiếu đến kiểu giá trị gốc. Bất cứ sự thay đổi nào đến kiểu giá trị không được phản ánh ở đối tượng và ngược lại.

Việc bao (box) một kiểu giá trị có thể được thực hiện với một phép gán không tường minh. Một phép gán không tường minh là một phép gán không yêu cầu một sự ép kiểu (type cast), như được trình bày sau đây:

```
int i = 10;
object o = i;
```

Kiểu giá trị biến i bị ép kiểu một cách không tường minh thành kiểu object. Khi kiểu giá trị i được bao (box) thành đối tượng o, một thể hiện của kiểu object được tạo trên vùng heap còn thông tin về kiểu và giá trị ở bên phải của biểu thức được chép vào đối tượng.

Để không bao (unbox) một đối tượng, việc chuyển đổi tường minh là cần thiết, như được trình bày sau đây:

```
int i = 10;
object o = i;
int j = (int)o; //sự chuyển đổi tường minh từ object thành int
```

Biến nguyên (int) j lúc này chứa giá trị mà đã được chứa bởi đối tượng o. Điều quan trọng để biết rằng tất cả các biến i, o, và j là độc lập với nhau và không đơn thuần tham chiếu đến cùng vùng nhớ.



2. CÁC KHÁI NIỆM LẬP TRÌNH

Nếu bạn đã quen với C++ hoặc Java, bạn sẽ không cần tốn công nhiều để học về C#. Cú pháp và những sự trình bày được nhận thấy ở ngôn ngữ này là rất giống với những điều được thấy ở C++ hay Java. Sự thật của vấn đề là vì C# đã được thiết kế đơn giản để học và dù khả năng để diễn đạt rõ ràng.

2.1. Không gian tên (namespace)

Trong những năm gần đây, sự thịnh hành của không gian tên (namespace) tham gia một vai trò trọng đại vào công nghệ phần mềm và phát triển thành phần. Không gian tên (namespace) hỗ trợ cho sự cô lập và việc đóng gói các lớp (class), các giao diện, và các struct có liên quan thành một đơn vị (unit).

Kiến trúc .NET sử dụng các không gian tên (namespace) lồng nhau xuất phát từ không gian tên (namespace) System. Microsoft cũng đã cung cấp các lớp (class), và các giao diện được đặt dưới không gian tên (namespace) Microsoft dành cho chức năng riêng biệt của Windows. Khi việc phát triển các thành phần cho nền tảng .NET, bạn sẽ sử dụng tên công ty của bạn như không gian tên (namespace) ở phía ngoài ở nơi mà chứa tất cả các lớp (class), các giao diện, và mã thành phần của bạn.

Một khai báo không gian tên (namespace) sẽ đứng trước bất cứ mã lệnh nào mà bạn phát triển, mặc dù điều đó là không cần thiết. Cú pháp cho việc khai báo một không gian tên (namespace) là như sau:

```
namespace some-namespace-name {
    //các lớp (class), các giao diện, các
    //struct, v.v
}
```

Sử dụng các thực thể bên trong một không gian tên (namespace) có thể được thực hiện hai cách khác nhau. Cách dễ nhất để truy cập các thực thể bên trong một không gian tên (namespace) là sử dụng chỉ thị using. Xem xét không gian tên (namespace) System, cái mà đã được dùng trong mỗi ví dụ được trình bày cho đến nay. Dòng đầu tiên của đoạn mã trong các ví dụ trước là dòng

```
using System;
```

Chỉ thị này chỉ thị cho trình biên dịch sử dụng không gian tên (namespace) System để xác định tên của các lớp được dùng trong thân của đoạn mã (code).

Sự lựa chọn thứ hai là sử dụng tên đầy đủ của một thực thể riêng biệt. Ví dụ, lớp Console tồn tại bên trong không gian tên System. Thay vì sử dụng chỉ thị using, nó có thể dùng tên đầy đủ để thay vào, như được trình bày sau đây:

```
System.Console.WriteLine("Fully Qualified
Name Access");
```



Chúng ta hãy xem cách sử dụng không gian tên (namespace) trong khi lập trình. Đoạn mã mẫu ở ví dụ 2.1-4 thực hiện hai không gian tên (namespace). Mỗi tên vùng chứa một lớp Money. Điều đó quan trọng để biết rằng hai lớp Money không ở cùng phần của C#. Dù là lớp Money đồng nhất trong khai báo và cài đặt, C# xét hai lớp rõ rệt được phân biệt bởi không gian tên (namespace) khác nhau.

Ví dụ 2.1-4 Không gian tên

```
1://File           :part02_04.cs
2://Ngày viết     :12.28.2000
3://Tác giả       :Richard L. Weeks
4://Mục đích      :làm thấy rõ hơn về không gian tên
                     (Namespace)

5:
6:
7:using System;
8:
9:
10:
11: namespace Foo {
12:
13:     public class Money {
14:
15:         private double m_Amount;
16:
17:         public Money( ) {
18:             Init(0.0);
19:         }
20:
21:         public Money( double Amount ) {
22:             Init( Amount );
23:         }
24:
25:
26:         public void Print( ) {
27:             Console.WriteLine("Foo.Money.Print(0)", 
28:                               m_Amount);
29:
30:
31:         public void Init( double Amount ) {
32:             m_Amount = Amount;
33:         }
34:     }
35: }
36:
37: namespace Bar {
38:
```

```

39:     public class Money {
40:
41:         private double m_Amount;
42:
43:         public Money( ) {
44:             Init(0.0);
45:         }
46:
47:         public Money( double Amount ) {
48:             Init( Amount );
49:         }
50:
51:
52:         public void Print( ) {
53:             Console.WriteLine("Bar.Money.Print {0}",
54:                               m_Amount );
55:
56:
57:         private void Init( double Amount ) {
58:             m_Amount = Amount;
59:         }
60:     }
61: }
62:
63:
64: public class NamespaceTest {
65:
66:     public static void Main( ) {
67:
68:         Foo.Money fm = new Foo.Money(5.00);
69:         Bar.Money bm = new Bar.Money(5.00);
70:
71:
72:         fm.Print( );
73:         bm.Print( );
74:
75:
76:     }
77: }
```

Đoạn mã ở ví dụ 2.1-4 khai báo hai không gian tên (namespace) – Foo và Bar. Một lớp Money tồn tại bên trong mỗi không gian tên (namespace). Để sử dụng mỗi lớp Money, phương thức Main tạo một thể hiện của mỗi lớp Money bằng cách sử dụng tên đầy đủ. Một điều kiện được lưu ý là câu lệnh Using, ở đó không tồn tại một từ khoá không sử dụng một không gian tên (namespace). Cho nên, nếu bạn có hai đối tượng có cùng tên thuộc về không gian tên (namespace) khác nhau, chắc chắn sử dụng tên đầy đủ thì hay hơn là chỉ thị using.

2.2. Các phát biểu lệnh (statements)

Để điều khiển trình tự thực hiện đối với một đoạn mã chương trình, cần phải có một hay nhiều khái niệm về điều khiển lưu đồ thực hiện chương trình. Đó là nội dung chính của phần này.

2.2.1. Phát biểu if

Một trong những câu lệnh điều khiển cơ bản nhất là phát biểu if. Rất đơn giản, phát biểu if sẽ xác định giá trị của một biểu thức luận lý. Kết quả của biểu thức luận lý sẽ cho biết là chương trình có thực hiện một dòng mã hay một đoạn mã hay không. Khi kết quả nhận được từ biểu thức là true, các phần mã bên trong của lệnh if sẽ được thực hiện. Khi kết quả của biểu thức điều kiện là false, đoạn mã bên trong lệnh if sẽ không được thực hiện.

Cú pháp của phát biểu if như sau:

```
if ( conditional-expression )
    phát biểu lệnh đơn;
```

hoặc:

```
if (conditional-expression) {
    một hay nhiều câu lệnh
}
```

Biểu thức điều kiện có thể là một biểu thức đơn giản hoặc là một biểu thức phức tạp, nhưng điều quan trọng là giá trị kết quả sau cùng phải là một giá trị Boolean. Trong ngôn ngữ C và C++, bất cứ biểu thức nào có giá trị khác 0 đều xem như là true và giá trị 0 thì được xem là false. C# sẽ thực hiện việc kiểm soát kiểu nghiêm ngặt và yêu cầu biểu thức điều kiện phải luôn là một trong hai giá trị là true hoặc false (xem ví dụ 2.1-5).

Ví dụ 2.1-5 Phát biểu if

```
1: //File      :part02_07.cs
2: //Date     :12.29.2000
3: //Author   :Richard L. Weeks
4: //Purpose  :C# if statement usage
5:
6:
7: using System;
8:
9:
10:
11: public class Statements {
12:     public static void Main() {
13:         // ...
14:     }
15: }
```

```

16:     bool bSimple_1 = true;
17:     bool bSimple_2 = false;
18:
19:
20:     if( bSimple_1 )
21:         Console.WriteLine("You should see this");
22:
23:     if( bSimple_2 )
24:         Console.WriteLine("You should not see
this");
25:
26:     //sử dụng if với một phát biểu đơn
27:     if( bSimple_1 && !bSimple_2 )
28:         Console.WriteLine("Will you see this?");
29:
30:
31:     //Sử dụng if với nhiều phát biểu lệnh
32:     if( bSimple_1 ) {
33:         Console.WriteLine("Statement 1");
34:         Console.WriteLine("Statement 2");
35:     }
36: }
37: }
```

Ví dụ 2.1-5 thể hiện ý nghĩa cơ bản của phát biểu if. Việc xác định giá trị của biểu thức được thực hiện theo cách tính ước lượng rút gọn (short-circuit). Có nghĩa là biểu thức sẽ được duyệt từ trái sang phải, ngay khi biểu thức có giá trị là false thì phép tính dừng lại ngay với kết quả của cả biểu thức là false. Cách tính này cũng được áp dụng trong C và C++, nhưng Visual Basic thì không sử dụng cách tính short-circuit này cho các toán tử luân lý như And và Or. Trong VB.NET, hai toán tử AndAlso vàOrElse được thêm vào để thể hiện cách tính short-circuit này.

2.2.2. Phát biểu if...else

Cấu trúc if...else đơn giản chỉ là phần mở rộng của phát biểu chuẩn if. Mệnh đề if chỉ cho phép thực hiện phần mã bên trong khi biểu thức điều kiện có kết quả là true, mệnh đề else cho phép thực hiện phần mã bên trong khi kết quả của biểu thức điều kiện là false. Ví dụ 2.1-6 trình bày việc sử dụng cấu trúc if-else để xác định giá trị nguyên integer do người dùng nhập vào là chẵn hay lẻ

Ví dụ 2.1-6 Cách sử dụng mệnh đề if...else

```

1: //File    :part02_06.cs
2: //Author  :Richard L. Weeks
3: //Purpose :C# if-else statement
4:
```



```
5: using System;
6:
7: public class StatementTest {
8:
9:     public static void Main() {
10:        Console.WriteLine("Enter a number between 1 and 10: ");
11:        int i = Int32.Parse(Console.ReadLine());
12:
13:        //Kiểm tra số chẵn hoặc lẻ
14:        if( (i >= 1) && (i <= 10) ) {
15:
16:            //Kiểm tra số chẵn hoặc lẻ
17:            if( (i % 2) == 0 )
18:                Console.WriteLine("The number {0} is even",
19:                                i );
20:            else
21:                Console.WriteLine("The number {0} is odd", i );
22:        } else
23:            Console.WriteLine("You must enter a
24:                            number between 1 and 10");
25:    }
26: }
```

Chúng ta hãy dành ít phút để xem qua các dòng lệnh trong ví dụ 2.1-6. Ở dòng 10 chương trình nhắc người sử dụng nhập một giá trị từ 1 đến 10. Dòng kế tiếp sử dụng phương thức ReadLine của đối tượng Console để đọc giá trị nhập bởi người sử dụng. Vì phương thức ReadLine trả về kiểu chuỗi string, nên cần phải đổi từ kiểu string sang một giá trị số nguyên. Việc này được hoàn thành bởi việc sử dụng phương thức tĩnh (static) Parse trả về giá trị kiểu Int32.

Bước kế tiếp là kiểm tra giá trị nhập để xem giá trị có nằm trong khoảng được yêu cầu hay không. Nếu biểu thức điều kiện là true, bước kế tiếp sẽ tiến hành kiểm tra số được nhập là chẵn hay lẻ. Để kiểm tra việc chẵn lẻ, chương trình sử dụng toán tử chia lấy phần dư modulo. Toán tử modulo chia giá trị bên trái cho giá trị bên phải và trả về số dư của phép chia. Vì số chia là 2 nên chỉ có các số chẵn mới có số dư là 0.

Như đã nói ở phần trên, vì C# thực hiện kiểm tra kiểu chặt chẽ nên ở dòng 17, kết quả phải có kiểu Boolean. Trong C và C++, dòng lệnh có thể được viết như sau:

~~if((i % 2) == 0)~~

Tuy nhiên, biểu thức $i \% 2$ sẽ trả về một số integer, không phải là một kết quả kiểu Boolean. Chính vì vậy, C# sẽ không chấp nhận mệnh đề này và sẽ báo lỗi.

2.3. Toán tử điều kiện ?:

Toán tử điều kiện (condition operator) có thể được xem như là một hình thức khác của mệnh đề if...else. Nó rất tiện lợi khi muốn gán giá trị cho biến dựa trên một vài biểu thức điều kiện. C, C++ và cả Visual Basic đều hỗ trợ toán tử điều kiện. Visual Basic sử dụng phát biểu IIf(...) để thể hiện toán tử điều kiện. Không giống như VB, C# dùng cách tính short-circuit. Trong cách tính này chỉ những kết quả cần thiết mới được tính. Trong VB, cả điều kiện true và false đều được tính. Điều này không hiệu quả và có thể dẫn đến một số lỗi tiềm tàng.

Toán tử điều kiện có cú pháp như sau:

```
result = conditional-expression ? true expression : false expression
```

Luôn nhớ là C# yêu cầu tất cả các biểu thức điều kiện có kết quả là kiểu Boolean. Điều này phải được đảm bảo do tính kiểm tra kiểu nghiêm ngặt trong C#. Ví dụ 2.1-7 là một phiên bản cải thiện của ví dụ 2.1-6, sử dụng toán tử điều kiện để kiểm tra số được nhập vào là chẵn hay lẻ.

Ví dụ 2.1-7 Toán tử điều kiện

```

1: //File      :part02_07.cs
2: //Author    :Richard L. Weeks
3: //Purpose   :conditional-operator
4:
5:
6:
7: using System;
8:
9:
10: public class StatementTest {
11:
12:
13:     public static void Main( ) {
14:
15:         Console.WriteLine("Enter an number between 1
16:                           and 10: ");
17:         int i = Int32.Parse(Console.ReadLine( ) );
18:
19:
20:         if( (i >= 1) && (i <= 10) ) {
21:
22:             //kiểm tra số chẵn lẻ
23:             string odd_even = (i % 2) == 0 ? "even" : "odd";
24:
25:             Console.WriteLine( "The number {0} is {1}",
26:                               i, odd_even );
27:         }
28:     }
29: }
```

```
26:  
27:     } else  
28:         Console.WriteLine("You must enter a number  
                           between 1 and 10");  
29:  
30:     }  
31: }
```

Dòng 23 trong ví dụ 2.1-7 cho thấy cách dùng của toán tử điều kiện. Ở đây, biến kiểu string odd_even được gán giá trị dựa trên điều kiện $(i \% 2) == 0$. Khi kết quả của điều kiện là true, biến odd_even được gán bằng số dạng chữ là "even". Khi biến thức là false, biến odd_even được gán giá trị là "odd".

Trong ví dụ 2.1-7, toán tử điều kiện được sử dụng để thay thế cho bốn dòng lệnh từ dòng 17 đến dòng 20 của ví dụ 2.1-6.

2.3.1. Phát biểu goto

Vâng, phát biểu goto vẫn còn được sử dụng. Tôi biết rằng các bạn sẽ thắc mắc vì sao một ngôn ngữ hiện đại như C# lại vẫn sử dụng phát biểu goto? Phát biểu goto cho phép tiến trình thực hiện ngắt ngang và lập tức chuyển đến thực hiện tiếp chương trình tại một nhãn (label) chỉ định.

Đôi khi việc sử dụng phát biểu goto một cách đúng đắn có thể làm cho đoạn mã chương trình chặt chẽ và hiệu quả. Mặt khác, sử dụng phát biểu goto sẽ làm cho đoạn mã chương trình dễ đọc hơn.

Có một ít điều ràng buộc khi sử dụng phát biểu goto. Phát biểu goto không được sử dụng để nhảy vào đoạn xử lý của các phương thức khác hay thoát ra khỏi phương thức hiện hành. Phát biểu goto chỉ được sử dụng để chuyển điều khiển đến một nhãn ngay trong vùng mã hiện tại.

Cú pháp của phát biểu goto cũng tương tự như C, C++ hay VB. Nhãn đích là chuỗi các ký tự chữ cái và ký số, trong đó vị trí đầu tiên phải là một chữ cái. Tên của nhãn được kết thúc bởi ký tự hai chấm (:) để chỉ rõ rằng, đây chỉ là nhãn chứ không phải là câu lệnh, biểu thức hay biến.

Phát biểu goto có cú pháp như sau:

```
goto label-name;
```

Ví dụ 2.1-8 minh họa cách dùng cơ bản của phát biểu goto.

Ví dụ 2.1-8 Phát biểu goto

```
1: //File  :02_10.cs  
2: //Author :Richard L. Weeks  
3: //Purpose :Demonstrate the use of the goto statement  
4:  
5:
```

```

6: using System;
7:
8:
9: public class GotoTest {
10:
11:     public static void Main() {
12:
13:
14:         Console.WriteLine("about to goto label1");
15:         goto label1;
16:
17:         Console.WriteLine("This will not print");
18:
19:     label1:
20:         Console.WriteLine("label1 goto effective");
21:     }
22: }
23:

```

Ví dụ 2.1-8 trình bày cách dùng cơ bản của phát biểu goto. Ở dòng 19, một nhãn có tên label1 được khai báo. Phát biểu goto ở dòng 15 chuyển điều khiển chương trình ngay lập tức tới nhãn được chỉ định. Kết quả là đoạn mã từ dòng 15 tới dòng 19 sẽ bị bỏ qua. Chú ý là câu lệnh WriteLine ở dòng 17 chỉ ra rằng phần chữ này sẽ không được in ra.

Khi biên dịch ví dụ, trình biên dịch sẽ đưa ra một cảnh báo là đã dò ra một đoạn mã thừa (Unreachable Code). Đoạn mã thừa ở ví dụ này là dòng 17. Bởi vì điều khiển thực hiện chương trình bỏ qua đoạn mã này cho nên không có lý do nào để nó tồn tại. Bạn phải luôn chú ý vào những cảnh báo của trình biên dịch và xem như đó cũng chính là các lỗi biên dịch.

2.3.2. Phát biểu switch

Nhiều khi cần phải xem xét một vài điều kiện và chọn lựa thực hiện theo nhiều chọn lựa hay trường hợp. Đây chính là vai trò của phát biểu switch. Với một vài biểu thức cần định giá trị, phát biểu switch có thể được sử dụng để chọn ra đúng trường hợp để thực hiện. Hãy xem mệnh đề switch như là cách thể hiện việc thực hiện dựa trên nhiều chọn lựa thay vì phải sử dụng nhiều câu lệnh if lồng nhau.

Phát biểu switch bao gồm một biểu thức để xác định giá trị và một hay nhiều hàng số giá trị, đại diện cho các trường hợp có thể xảy ra.

```

switch( expression ) {
    case constant-expression:
        statement(s)
        jump statement

    [default: statement; jump-statement]
}

```



Biểu thức điều khiển cho phát biểu switch có thể là biểu thức kiểu số hay kiểu chuỗi. Các biểu thức hằng của case phải cùng kiểu dữ liệu với biểu thức điều khiển và không thể có 2 case có cùng một biểu thức hằng.

Không như trong C và C++, C# không cho phép từ case này thực hiện tiếp sang case khác. Để thực hiện điều này, có thể sử dụng phát biểu goto để chuyển điều khiển chương trình nhảy tới đoạn mã của case khác. Mỗi case phải có phát biểu nhảy, kể cả case cuối cùng hay case default.

Ví dụ 2.1-9 trình bày cách sử dụng cơ bản của phát biểu switch.

Ví dụ 2.1-9 Phát biểu switch

```
1: //File  :part02_09.cs
2: //Author :Richard L. Weeks
   //Purpose :The switch statement
3: using System;
4: public class SwitchTest {
5:     public static void Main( ) {
6:         Console.WriteLine("Please make your selection");
7:         Console.WriteLine("1 Hamburger");
8:         Console.WriteLine("2 Cheese Burger");
9:         Console.WriteLine("3 Fish");
10:        int Selection = int.Parse( Console.ReadLine( ) );
11:        switch( Selection ) {
12:            case 1:
13:                Console.WriteLine("Hamburger");
14:                break;
15:            case 2:
16:                Console.WriteLine("Cheese Burger");
17:                break;
18:            case 3:
19:                Console.WriteLine("Fish");
20:                break;
21:            default:
22:                Console.WriteLine("Unknown choice");
23:                break;
24:        }
25:    }
```

Ví dụ 2.1-9 trình bày cách sử dụng phát biểu switch. Mỗi biểu thức của case đại diện bằng một hằng số. Chú ý là lệnh nhảy được thực hiện cho mỗi case.

Vì C# không cho phép chương trình thực hiện từ case này sang case khác, nên phát biểu goto có thể được sử dụng để chuyển điều khiển sang case khác hay

case default. Ví dụ 2.1-10 sử dụng phát biểu goto để chuyển điều khiển từ một mệnh đề case sang mệnh đề case khác.

Ví dụ 2.1-10 Sử dụng goto bên trong mệnh đề case

```
1: //File :02_12.cs
2: //Author :Richard L. Weeks
3: //Purpose :The switch statement
4:
5: using System;
6:
7: public class SwitchTest {
8:     public static void Main() {
9:         Console.WriteLine("Please make your selection");
10:        Console.WriteLine("1 Hamburger");
11:        Console.WriteLine("2 Cheese Burger");
12:        Console.WriteLine("3 Fish");

13:        int Selection = int.Parse(Console.ReadLine());
14:        switch(Selection) {
15:            case 1:
16:                Console.WriteLine("Hamburger");
17:                goto case 4;
18:            case 2:
19:                Console.WriteLine("Cheese Burger");
20:                goto case 4;
21:            case 3:
22:                Console.WriteLine("Fish");
23:                break;
24:            case 4:
25:                Console.WriteLine("Transferred to case 4");
26:                Console.WriteLine("Transferring to
27:                                default case");
28:                goto default;
29:            default:
30:                Console.WriteLine("Unknown choice");
31:                break;
32:        }
33:    }
34: }
```

Ví dụ 2.1-10 là đoạn chương trình được sửa lại từ ví dụ 2.1-9. Các case 1,2 và 4 sử dụng phát biểu goto để chuyển hướng thực hiện chương trình từ case hiện tại tới case khác. Như vậy một lần nữa chúng ta thấy, phát biểu goto được dùng để chuyển điều khiển của chương trình tới nhãn case khác hay nhãn default.

2.3.3. Mệnh đề for

Mệnh đề for là một trong những mệnh đề lặp cơ bản nhất. Những trường hợp số lần lặp được xác định trước, sử dụng mệnh đề for là rất thuận tiện. Tương tự như C và C++, mệnh đề for bao gồm giá trị khởi tạo, một biểu thức điều kiện và mệnh đề điều khiển lặp. Cú pháp của for như sau:

```
for( initialization; conditional-expression; iteration(s) )
    statement;
```

initialization là giá trị khởi tạo, có thể là một danh sách các khai báo và gán giá trị cho biến, được phân cách bởi dấu phẩy. conditional-expression là biểu thức điều khiển phải có giá trị là kiểu Boolean và được sử dụng để xác định khi nào vòng lặp for có thể kết thúc. iteration là mệnh đề điều khiển việc lặp nó thường dùng để tăng hay giảm các biến điều khiển được sử dụng bởi mệnh đề for (Xem ví dụ 2.1-11).

Ví dụ 2.1-11 Mệnh đề for

```
1: using system;
2:
3: public class ForStatement{
4:
5:     public static void Main( ) {
6:
7:         for(int i = 0; i < 10; i++)
8:             Console.WriteLine( "i = {0}", i );
9:     }
10: }
```

Ví dụ 2.1.10 trình bày hoạt động của vòng lặp for. Biến kiểu integer i được sử dụng để điều khiển số lần lặp được thực hiện. Kết quả của ví dụ 2.1.10 có dạng như sau:

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
```

Chú ý là miền giá trị của i là từ 0 tới 9 chứ không phải từ 0 tới 10. Điều này tương ứng với điều kiện $i < 10$. Chú ý là biểu thức kiểm tra giá trị của i là i phải nhỏ hơn 10. Khi giá trị của i bằng 10, vòng lặp for sẽ kết thúc và chương trình sẽ tiếp tục thực hiện các câu lệnh kế tiếp ngay sau cấu trúc for.

Mệnh đề for chỉ tác động đến câu lệnh hay một đoạn câu lệnh (khối) kế tiếp ngay sau phát biểu for. Để thực hiện nhiều lệnh ứng với for, các câu lệnh phải được đặt trong một khối (khối). Khối là một đoạn các câu lệnh bên trong dấu mở và đóng mốc { }. Ví dụ 2.1-12 trình bày mệnh đề for để điều khiển một khối các câu lệnh.

Ví dụ 2.1-12 Khối các câu lệnh

```

1: using System;
2:
3: public class ForTest {
4:
5:     public static void Main( ) {
6:         for(int i = 0; i < 10; i++) {
7:             Console.WriteLine("value of i to follow");
8:             Console.WriteLine("i = {0}", i);
9:         }
10:    }
11:
12: }
```

Với việc thêm vào mệnh đề khối, mệnh đề for bây giờ sẽ điều khiển việc thực hiện các câu lệnh bên trong khối. Kết quả của ví dụ 2.1.12 có dạng như sau:

```

1: value of i to follow
2: i = 0
3: value of i to follow
4: i = 1
5: value of i to follow
6: i = 2
7: value of i to follow
8: i = 3
9: value of i to follow
10: i = 4
11: value of i to follow
12: i = 5
13: value of i to follow
14: i = 6
15: value of i to follow
16: i = 7
17: value of i to follow
18: i = 8
19: value of i to follow
20: i = 9
```

Cần để ý tầm vực xác định của các biến. Trong cả 2 ví dụ về cách sử dụng mệnh đề for, các biến điều khiển đều được khai báo và khởi tạo giá trị bên trong

chính mệnh đề for. Chính vì vậy, biến i chỉ tồn tại bên trong cấu trúc for, và mọi nỗ lực truy xuất tới biến sau cấu trúc for sẽ dẫn đến lỗi biên dịch (compile error).

2.3.4. Mệnh đề while

Mệnh đề while cho phép thực hiện lặp câu lệnh ở phần thân khi biểu thức điều kiện cho giá trị là true.

```
while( condition-expression )
      statement
```

Mệnh đề while hoạt động như sau:

1. Biểu thức điều kiện được kiểm tra
2. Nếu biểu thức điều kiện cho giá trị true, câu lệnh phần thân (body) sẽ được thực hiện
3. Việc này sẽ tiếp diễn cho đến khi điều kiện trở thành false.
4. Khi điều kiện có giá trị false, mệnh đề được kết thúc và chương trình chuyển đến thực hiện câu lệnh kế tiếp.

Vì lý do này, phần thân của mệnh đề có khả năng sẽ không được thực hiện lần nào. Nếu điều kiện là false, phần thân của mệnh đề sẽ không được thực hiện.

Mệnh đề while rất phù hợp trong trường hợp số lần lặp không được biết trước, không giống như mệnh đề for là đòi hỏi phải biết trước số lần lặp.

Hãy xem xét cách thực hiện phương pháp của Newton trong việc tìm căn bậc hai. Số lần lặp không được biết trước vì phương pháp sẽ tiếp tục chạy khi phần chênh lệch giữa giá trị trước và giá trị hiện tại lớn hơn một giá trị epsilon định trước. Hãy xem phần mã giả của phương pháp Newton và sau đó là cách thể hiện bằng C#.

```
Epsilon := 1.0e-9
BaseGuess := N // Có thể là giá trị bất kỳ
Value := M // giá trị do người dùng định nghĩa
NewGuess := ((Value / BaseGuess) + BaseGuess) / 2;
While abs_delta( NewGuess, BaseGuess ) > Epsilon
Begin
    BaseGuess := NewGuess;
    NewGuess := ((Value / BaseGuess) + BaseGuess) / 2;
End
```

Lưu ý rằng không có cách để xác định số lần lặp của mệnh đề while. Biểu thức điều kiện không dựa trên một biến đếm (counter) mà là dựa trên giá trị Boolean, có thể trở thành true bất cứ lúc nào. Vậy đối với số lần lặp không thể xác định trước, mệnh đề while sẽ phù hợp hơn mệnh đề for (cần biết trước số lần lặp). Phần mã của phương pháp Newton ở trên có thể diễn tả một cách dễ dàng trong C# như trong ví dụ 2.1-13 sau đây.



Ví dụ 2.1-13 Mệnh đề while

```
1: //File :part02_13.cs
2: //Author :Richard L. Weeks
3: //Purpose :Make use of the while statement to
4: //           implement Newton's method for finding
5: //           the square root of a number.
6:
7: using System;
8:
9:
10:
11: public class Newton {
12:
13:     public static void Main( ) {
14:
15:
16:         const double epsilon = 1.0e-9;
17:         double dGuess = 11.0;
18:
19:         //yêu cầu người dùng nhập vào thông tin
20:         Console.WriteLine("Enter a positive number: ");
21:         double dValue = double.Parse(
22:                                     Console.ReadLine( ) );
23:
24:         double dResult = ((dValue / dGuess)
25:                           + dGuess) / 2;
26:
27:         Console.WriteLine( "Guess Value = {0}", dGuess );
28:         Console.WriteLine( "Result Value = {0}", dResult );
29:
30:         while( Math.Abs(dResult - dGuess) > epsilon ) {
31:
32:             dGuess = dResult;
33:             dResult = ((dValue / dGuess) + dGuess) / 2;
34:             Console.WriteLine( "Guess Value = {0}",
35:                               dGuess );
36:             Console.WriteLine( "Result Value = {0}",
37:                               dResult );
38:         }
39:         Console.WriteLine( "\n*****\nThe approx sqrt of
40:                           {0} is {1}\n*****", dValue, dResult );
41:     }
42: }
```

Việc thể hiện phương pháp Newton trong ví dụ trên chỉ đơn thuần là chuyển thẳng từ phân mã đã liệt kê trước. Có một ít điểm cần phải hiểu rõ. Dòng 21 trình bày một ví dụ của kiểu cơ sở double và phương thức Parse. Nên nhớ rằng C# thể hiện các kiểu cơ sở như các struct và struct thì có thể chứa các method (phương thức). Trường hợp này cũng có thể xảy ra đối với tất cả các kiểu cơ sở trong C#.

Điểm lưu ý kế tiếp là mệnh đề while và biểu thức điều kiện điều khiển ở dòng 30. Hãy chú ý phương thức Abs, là phương thức tĩnh (static) của lớp Math. Phương thức Abs trả về giá trị tuyệt đối của một biểu thức số nguyên cho trước.

2.3.5. Vòng lặp do...while

Có cùng cấu trúc như mệnh đề while là mệnh đề do...while. Không giống như mệnh đề while, phần thân của mệnh đề do...while sẽ được lặp ít nhất là một lần. Điều này tương ứng với việc biểu thức điều kiện được kiểm tra ở cuối của mỗi lần lặp chứ không phải là ở phần bắt đầu. Ví dụ 2.1-14 được sửa lại từ ví dụ 2.1.13, thay thế cách sử dụng vòng lặp while bằng sử dụng vòng lặp do...while. Kết quả của cả hai cách là như nhau, chỉ có cách thức thực hiện là khác nhau.

Ví dụ 2.1-14 Mệnh đề do...while

```

1: //File  :
2: //Author :Richard L. Weeks
3: //Purpose :Make use of the do while statement to
4: //      :implement Newton's method for finding
5: //      :the square root of a number.
6:
7: using System;
8:
9: public class Newton {
10:
11:     public static void Main() {
12:
13:
14:         const double epsilon = 1.0e-9;
15:         double dGuess = 11.0;
16:         double dResult = 0.0;
17:
18:
19:         Console.Write("Enter a positive number: ");
20:         double dValue = double.Parse(Console.ReadLine());
21:
22:
23:         dResult = ((dValue / dGuess) + dGuess) / 2;
24:
25:
26:         do {

```

```

27:     Console.WriteLine( "Guess Value = {0}", dGuess );
28:     Console.WriteLine( "Result Value = {0}", dResult
    );
29:
30:     dGuess = dResult;
31:     dResult = ((dValue / dGuess) + dGuess) / 2;
32:
33: } while( Math.Abs(dResult - dGuess) > epsilon );
34:
35:
36: Console.WriteLine( "\n*****\nThe approx sqrt of
            {0} is {1}\n*****", dValue, dResult );
37: }
38: }
```

2.3.6. Mệnh đề foreach

Mệnh đề foreach sẽ rất quen thuộc nếu bạn đã từng phát triển ứng dụng với Visual Basic. Về cơ bản mệnh đề foreach cho phép việc lặp dựa theo các thành phần bên trong của một mảng array hay một collection (tập hợp) nào đó sử dụng giao tiếp (interface) `IEnumerable`. Chúng ta sẽ đề cập interface này ở phần sau.

Mỗi tập hợp collection cung cấp một số phương thức cho việc thực hiện lặp thông qua nội dung của collection. Trong thế giới C++ khái niệm iterator (bộ lặp) trùng khớp với khái niệm enumerator (bộ đếm) trong thế giới COM. Những người đã hài lòng với interface `IEnumVARIANT` nhận thấy rõ việc dễ dàng sử dụng interface `IEnumerable` cùng với việc hỗ trợ một interface `IEnumerator` trong kiến trúc .NET.

```
foreach(element_type variable in expression)
    statement;
```

Hãy xem xét cú pháp của mệnh đề foreach, `element_type` là kiểu dữ liệu được sử dụng trong việc khai báo biến. `expression` có thể là một mảng array hay một lớp tập hợp (collection class) chứa các phần tử có kiểu là `element_type`. Ví dụ 2.1-15 sử dụng foreach statement để lặp thông qua nội dung của một mảng integer.

Ví dụ 2.1-15 Mệnh đề foreach

```

1: //File :part02_15.cs
2: //Author :Richard L. Weeks
3: //Purpose:Using the foreach statement
4:
5:
6: using System;
7: public class ArrayListing {
8:
```

```

9:     public static void Main( ) {
10:         int[] whole_numbers = {1,2,3,4,5,6,7,8,9,10};
12:
13:
14:         //Hiển thị toàn bộ nội dung của các phần tử trong mảng
15:         foreach( int value in whole_numbers )
16:             Console.WriteLine( "value = {0}", value );
17:     }
18: }
```

Không có quá nhiều mã trong ví dụ 2.1-14 vì tôi muốn đặt trọng tâm là làm sao để foreach hoạt động. Khi bạn nhìn toàn bộ mệnh đề foreach, bạn sẽ thấy dòng 15 và 16 của ví dụ 2.1-15 là tương đương với ví dụ 2.1-16

Ví dụ 2.1-16 Mở rộng mệnh đề foreach

```

1: IEnumarator iterator = whole_number.GetEnumerator( );
2: while( iterator.MoveNext( ) ) {
3:     int value = (int)iterator.Current;
4:     Console.WriteLine( "value - {0}", value );
5: }
```

Như bạn thấy , phần mã mở rộng trên sử dụng interface IEnumarator và phương thức GetEnumerator() cung cấp bởi kiểu System.Array. Kế tiếp, một mệnh đề while được sử dụng để liệt kê số phần tử của mảng. Phần tử hiện tại được truy xuất và ép sang kiểu dữ liệu phù hợp.

2.4. Toán tử (operators)

Toán tử là phép toán được áp dụng cho các kiểu cơ sở hay kiểu đối tượng object. Toán tử cung cấp việc thực hiện các phép tính. Toán tử một ngôi (unary operator) là toán tử thực hiện trên một toán hạng duy nhất. Toán tử hai ngôi (Binary operator) cần phải có hai toán hạng bên phải và trái. Toán tử quan hệ (relational operator) có dạng là biểu thức Boolean có thể được dùng để tạo những biểu thức điều kiện. Bảng 2.1-2 chứa tất cả các toán tử của C#.

Bảng 2.1-2 Các toán tử của C#

Toán tử	Ý nghĩa
Toán học	+,-,*,/,%
Logical {Boolean và Bitwise}	&, , ^, !, ~, &&,
Cộng một ngôi,	++, --
Trừ một ngôi	
Shift (dịch chuyển bit)	>>, <<

Quan hệ so sánh

==, !=, <, >, <=, >=

Gán

=, +=, -=, *=, /=, %=, !=, ^=, <<=, >>=

Thông tin kiểu

is

Chuyển kiểu

(Type) Variable, as

Các toán tử trong bảng 2.1-2 rất quen thuộc với các lập trình viên trong phần lớn các ngôn ngữ với ngoại trừ toán tử is và as.

2.4.1. Toán tử is

Toán tử is dùng để kiểm tra xem kiểu của một thực thể là kiểu gì. C# hỗ trợ rất mạnh thông tin về kiểu trong quá trình chạy (runtime) và toán tử is sử dụng thông tin kiểu này để xác định xem thực thể đã cho có phải là kiểu đã yêu cầu hay không. Cú pháp toán tử is như sau:

expression is type

Toán tử is có kết quả là Boolean và có thể được sử dụng trong biểu thức điều kiện. Toán tử is trả về giá trị true nếu các điều kiện sau thỏa mãn:

- Biểu thức có giá trị khác null
- Biểu thức có thể ép kiểu (casting) an toàn. Ép kiểu ở đây được xem là ép kiểu tường minh (explicit) có dạng: (type) (expression)

Nếu cả hai điều kiện trên được thỏa mãn, toán tử is sẽ trả về giá trị true, ngược lại, giá trị trả về là false. Ví dụ 2.1-17 trình bày cách sử dụng toán tử is.

Ví dụ 2.1-17 Toán tử is

```

1: //File      :part02_16.cs
2: //Author :Richard L. Weeks
3: //Purpose  :Demonstrate the 'is' operator
4:
5:
6: using System;
7:
8:
9: //Tạo lớp rỗng
10:
11: class Dog {

```



```
12: }
13:
14: class Cat {
15: }
16:
17:
18: public class IsDemo {
19:
20:     public static void Main( ) {
21:
22:         Dog myDog = new Dog( );
23:         Cat myCat = new Cat( );
24:         int i = 10;
25:
26:         WhatIsIt(myDog);
27:         WhatIsIt(myCat);
28:         WhatIsIt(i);
29:     }
30:
31:     public static void WhatIsIt( object o ) {
32:
33:         if( o is Dog )
34:             Console.WriteLine("It is a dog");
35:         else if( o is Cat )
36:             Console.WriteLine("It is a cat");
37:         else
38:             Console.WriteLine("I don't know what it is");
39:     }
40: }
```

Ví dụ 2.1-17 trình bày hai khái niệm chưa được giới thiệu. Đầu tiên là việc tạo phương thức tĩnh (static) WhatIsIt trong lớp IsDemo. Một phương thức tĩnh có thể sử dụng như là một hàm độc lập bởi vì không cần một thể hiện (instance) của đối tượng. Phương thức tĩnh sẽ được trình bày chi tiết ở phần sau.

Điều thú vị kế tiếp là tham số được truyền cho phương thức tĩnh WhatIsIt. Để ý rằng dạng của tham số là object. Trong C#, tất cả các class đều ngầm định là thừa kế từ lớp cơ sở object và tất cả các kiểu giá trị có thể được gói lại như object. Chính vì điều này, object có thể được sử dụng như là một kiểu tổng hợp có thể chấp nhận mọi kiểu.

Cách sử dụng thực tế của toán tử is rất rõ ràng. Với mỗi phát biểu if, toán tử is được sử dụng để định ra thông tin về kiểu lúc thực thi của đối tượng object được truyền vào hàm. Tuỳ thuộc vào kết quả của biểu thức is, phần trả lời tương ứng sẽ được trình bày lên màn hình.

Sau đây là kết quả của ví dụ 2.1-16

It is a dog

It is a cat
I don't know what it is

2.4.2. Toán tử as

Cũng sử dụng thông tin về kiểu trong quá trình thực thi như toán tử is, toán tử as sẽ cố gắng ép kiểu một biểu thức sang kiểu được yêu cầu. Cách ép kiểu thông thường có dạng (*T*) *e*, trong đó *T* là kiểu và *e* là biểu thức. Phép ép kiểu sẽ tạo nên một InvalidCastException khi có sự ép kiểu không hợp lệ. Toán tử as không phát sinh ngoại lệ exception, thay vì vậy, kết quả trả về là giá trị null.

Toán tử as sử dụng cú pháp tương tự như toán tử is:

expression as type

Cú pháp của toán tử as có thể được mở rộng như sau:

expression is type ? (type)expression : (type)null

Vì vậy, toán tử as đơn giản chỉ là cách viết rút gọn của toán tử is và toán tử điều kiện. Ưu điểm của toán tử as là dễ sử dụng và điều rõ ràng là không phát sinh ra ngoại lệ exception ngay cả khi việc ép kiểu dữ liệu không thành công.

2.5. Arrays (Mảng)

Trong C#, Array (mảng) là đối tượng khá đặc biệt. Tất cả các đối tượng kiểu mảng đều được ngầm định là kế thừa từ kiểu dữ liệu System.Array. Lớp cơ sở System.Array cung cấp rất nhiều phương thức được sử dụng trong quá trình thao tác trên mảng. Array cũng cho phép kiểm tra chỉ mục (index). Có nghĩa là khi có sự truy suất đến một chỉ mục không hợp lệ, ví dụ như là chỉ mục ngoài khoảng xác định (out of range), sẽ phát sinh một ngoại lệ exception. Đối với C và C++, một mảng đơn giản sẽ không bị kiểm tra phạm vi và có thể dẫn đến việc ghi đè (overwrite) lên vùng stack hay vùng nhớ động (heap). C# thì không như vậy.

Việc khai báo một biến mảng thoát tiền có vẻ hơi kỳ cục, nhưng khi phân tích lại cú pháp, bạn sẽ thực sự nhận thấy khai báo như vậy sẽ rõ ràng hơn cách khai báo mảng trong C và C++.

Cú pháp sau đây được sử dụng để cấp phát một mảng cấp 1

array-type[] var = new array-type[size]

Chú ý là dấu vuông ở ngay kế kiểu của mảng (array-type) chứ không phải ngay sau tên biến. Cú pháp này làm cho việc khai báo rõ ràng hơn là đặt dấu mốc vuông ngay sau tên biến.

Mảng trong C# sử dụng chỉ mục đếm từ 0. Những ngôn ngữ như COBOL và Visual Basic sử dụng chỉ mục đếm từ 1, tuy nhiên Visual Basic cho phép bạn định nghĩa lại giá trị đầu tiên của chỉ mục. Ví dụ 2.1-18 sử dụng mảng một chiều và một vài phương thức do kiểu System.Array cung cấp.

Ví dụ 2.1-18 Mảng một chiều

```
1: //File  :part02_17.cs
2: //Author :Richard L. Weeks
3:
4:
5: using System;
6:
7: public class ArrayTest {
8:
9:     public static void Main( ) {
10:
11:         //Khai báo mảng đơn có kiểu int
12:         int[] array_1 = new int[5];
13:
14:         //Điền đầy mảng
15:         for( int i = array_1.GetLowerBound(0) ;
16:              i <= array_1.GetUpperBound(0) ; i++ )
17:             array_1[i] = i+1;
18:
19:         //Hiển thị nội dung mảng
20:         for( int j = array_1.GetLowerBound(0) ;
21:              j <= array_1.GetUpperBound(0) ; j++ )
22:             Console.WriteLine("array_1[{0}] = {1}" ,
23:                               j , array_1[j]);
24:
25:         //Khai báo khởi tạo mảng
26:         int[] array_2 = new int[] { 25, 10, 4, 7, 15,
27:                                   2, 1 };
28:
29:         //Sắp xếp mảng
30:         System.Array.Sort( array_2 );
31:
32:         //Hiển thị nội dung mảng
33:         for( int k = array_2.GetLowerBound(0) ;
34:              k <= array_2.GetUpperBound(0) ; k++ )
35:             Console.WriteLine("array_2[{0}] = {1}" ,
36:                               k , array_2[k] );
37:
```

Ví dụ 2.1-18 ở trên cho thấy cách sử dụng phương thức GetLowerBound và GetUpperBound của System.Array. Phương thức GetLowerBound nhận một tham số kiểu nguyên integer chỉ định giá trị chặn dưới của chỉ mục. Tương tự như vậy GetUpperBound nhận một tham số integer để chỉ định giá trị chặn trên của chỉ mục. Cách sử dụng các phương thức này được thể hiện trong ví dụ ở các dòng 15, 19 và 31.

Dòng 12 đến dòng 16 khai báo một mảng và dùng vòng lặp for để khởi tạo giá trị. Ở dòng 25 việc khai báo và khởi tạo giá trị được thực hiện trên một dòng duy nhất.

Lớp System.Array cũng cung cấp phương thức Sort. Phương thức Sort dùng để sắp xếp các kiểu dữ liệu nội tại hay các kiểu dữ liệu được phát triển theo giao diện (interface) IComparable. Cách phát triển interface sẽ được trình bày trong chương 2.2 (C# nâng cao).

Mảng Array trong C# không nhất thiết là một chiều. Khai báo mảng nhiều chiều đơn giản là chỉ định chiều dài cho mỗi chiều. Điều khá quan trọng là bạn phải chú ý rằng mảng không nhất thiết là phải vuông vức. Các chiều có thể có những chặn trên khác nhau. Chúng ta hãy xem ví dụ 2.1-19 sau về cách sử dụng mảng nhiều chiều.

Ví dụ 2.1-19 Mảng đa chiều.

```

1: //File  :part02_18.cs
2: //Author :Richard L. Weeks
3: //Purpose :Arrays with a Rank greater than 1
4:
5:
6: using System;
7:
8: public class ArrayTest {
9:     public static void Main() {
10:
11:         int[,] grid = new int[3,3] { {1,2,3},
12:                                     {4,5,6}, {7,8,9} };
13:
14:         //Hiển thị nội dung của ma trận
15:         for( int i = 0; i < 3; i++ ) {
16:             for( int j = 0; j < 3; j++ ) {
17:                 Console.Write("{0,3}", grid[i,j] );
18:             }
19:             Console.WriteLine("");
20:         }
21:     }

```

Khai báo cho biến mảng hai chiều ở dòng 11 tương tự như cú pháp khai báo và khởi tạo của mảng cấp 1. Điều khác biệt duy nhất là số chiều bây giờ là 2; đây



gọi là mảng cấp 2. Chú ý là các số chỉ mục cho mảng được phân cách bởi dấu phẩy. Cú pháp này rất quen thuộc đối với những lập trình viên Visual Basic, nhưng nếu bạn là những người phát triển trên C và C++, cú pháp này khác hẳn những gì bạn thường thấy.

Ví dụ 2.1-19 cũng trình bày cách khởi tạo danh sách khi khai báo biến mảng hai chiều.

2.6. Struct (cấu trúc)

Phần đầu của chương, chúng ta đã bàn về kiểu tham trị (value type) và kiểu tham chiếu (reference type). Trong C#, một cấu trúc struct được xem là một kiểu tham trị và như vậy, được quản lý ở trong vùng nhớ ngăn xếp stack chứ không phải ở trong vùng nhớ heap. C# phát triển các kiểu int và char như là các struct.

Struct dùng rất thích hợp với các kiểu dữ liệu đơn giản hoặc lưu giữ cơ chế tuân tự hóa serialize và trạng thái của đối tượng. Struct có thể chứa các thành phần dữ liệu, phương thức (method), thuộc tính (property) và constructor (bộ khởi tạo hay còn gọi là phương thức khởi dựng). Trong hầu hết các trường hợp struct hoàn toàn tương tự như lớp (class). Tuy nhiên struct không thể kế thừa (inherit) từ một class hay là một struct khác. Struct không thể tạo các constructor với tham số bên ngoài truyền vào, và nếu bạn cố gắng làm điều này trình biên dịch sẽ phát sinh lỗi.

Protection (mức độ bảo vệ) mặc nhiên của các thành phần trong struct là riêng tư (private). Trong C++, sự khác biệt duy nhất giữa struct và class là cách bảo vệ biến mặc định. C# thì không như vậy. Điều quan trọng cần phải biết là trong C#, struct và class không thể dùng để hoán chuyển lẫn nhau được. Nên nhớ rằng một struct là một kiểu dữ liệu giá trị, còn một class là một kiểu dữ liệu tham chiếu. Ví dụ 2.1-20 dưới đây sẽ phát triển một struct đơn giản đại diện cho một tọa độ điểm (point).

Ví dụ 2.1-20 Khai báo và sử dụng struct

```
1: using System;
2:
3: public struct Point {
4:     public int x;
5:     public int y;
6: }
7:
8: public class StructTest {
9:
10:    public static void Main( ) {
11:        Point p;
12:        p.x = 5;
13:        p.y = 10;
14:        Point p2 = p;
15:        PrintPoint( p );
```



```

16:     PrintPoint( p2 );
17: }
18:
19: public static void PrintPoint( Point p ) {
20:     Console.WriteLine( "x = {0}, y = {1}", p.x, p.y );
21: }
22: }
```

Ví dụ 2.1-20 cho bạn thấy làm thế nào định nghĩa và sử dụng struct của C#. Struct của cấu trúc Point chứa hai thành phần dữ liệu public (cho phép truy cập toàn cục) – x và y. Chú ý ở dòng 14 biến p2 được gán giá trị là biến p. Khi xem xét kiểu giá trị, phép gán mặc định sẽ chép nội dung của biến bên phải cho biến phía bên trái. Điều này phản ánh rõ nét sự khác biệt với cách thực hiện của kiểu tham chiếu. Nên nhớ rằng đối với kiểu tham chiếu, phép gán sẽ thực hiện việc gán tham chiếu chứ không chép nội dung như là kiểu giá trị.

Có một số điểm thú vị cần phải ghi nhận khi làm việc với struct. Một struct không thể được sử dụng cho đến khi tất cả các giá trị ở bên trong struct đã được khởi tạo. Mọi lớp class đều có phương thức khởi động constructor mặc định, khởi tạo các thành phần dữ liệu với những giá trị mặc định tương ứng. Tuy nhiên, khi một struct chứa một thành phần dữ liệu dạng riêng tư private, phương thức khởi động mặc định sẽ không khởi tạo giá trị của nó nếu không có yêu cầu rõ ràng. Để làm được điều này, phải tạo cho struct toán tử new. Điều này gây nhầm lẫn một chút bởi vì struct vẫn được tạo trong vùng stack, không phải trong vùng nhớ heap như chúng ta thường hiểu đối với toán tử new. Ví dụ 2.1-21 trình bày cảnh báo do trình biên dịch phát sinh khi cố gắng sử dụng một struct có chứa những thành phần chưa được khởi tạo.

Ví dụ 2.1-21 Khởi tạo các thành phần của struct

```

1: using System;
2:
3: public struct Simple {
4:
5:     public int i;
6:     private string s;
7:
8:     public void init( ) {
9:         i = 10;
10:        s = "Hello";
11:    }
12:
13: }
14:
15:
16:
17: public class T {
18: }
```



```
19:     public static void Main() {
20:
21:         Simple simple;
22:
23:         simple.init();
24:
25:     }
26: }
```

Khi biên dịch ví dụ 2.1-21, trình biên dịch C# sẽ phát sinh lỗi sau:

```
Microsoft (R) Visual C# Compiler Version 7.00.9030
[CLR version 1.00.2204.21]
```

```
Copyright (C) Microsoft Corp 2000. All rights reserved.
```

```
struct_02.cs(25,3): error CS0165: Use of unassigned
local variable 'simple'.
```

Lỗi phát sinh bởi trình biên dịch C# có vẻ hơi mập mờ vì nó than phiền về cách sử dụng biến cục bộ simple khi chưa được khởi tạo. Lỗi này phát sinh là do dữ liệu thành phần dạng private s chưa được khởi tạo giá trị. Để khởi tạo giá trị cho thành phần private, một thể hiện của struct phải được khai báo sử dụng toán tử new (xem ví dụ 2.1-22).

Ví dụ 2.1-21 Khởi tạo các thành phần của struct

```
1: using System;
2:
3: public struct Simple {
4:
5:     public int i;
6:     private string s;
7:
8:     public void init() {
9:         i = 10;
10:        s = "Hello";
11:    }
12:
13:    public void show() {
14:
15:        Console.WriteLine("i = {0}", i);
16:        Console.WriteLine("s = {0}", s);
17:    }
18:
19: }
20:
21: public class T {
22:
23:     public static void Main() {
24: }
```

```

25:     Simple simple = new Simple( );
26:
27:     simple.init( );
28:
29: }
30: }

```

Xem lại ví dụ trước và thực hiện hai thay đổi, bây giờ chương trình có thể biên dịch không có lỗi và hoạt động như mong đợi. Thay đổi thứ nhất là thêm vào phương thức show, trong đó thể hiện việc sử dụng dữ liệu thành phần s. Thay đổi này thỏa mãn được sự nghiêm ngặt của trình biên dịch về việc không cho phép truy xuất các dữ liệu thành phần dạng private.

Chú ý là khai báo dòng 25 có sử dụng toán tử new. Nhắc lại lần nữa, điều hết sức quan trọng là phải hiểu một struct là một kiểu giá trị. Và như vậy, nó vẫn thuộc về stack. Việc sử dụng toán tử new làm cho phương thức khởi dụng constructor mặc định thực hiện và khởi tạo giá trị của dữ liệu thành phần private.

Bởi vì C# không cho phép constructor có tham số nên mọi cố gắng phát triển constructor dạng như vậy cho kiểu giá trị sẽ phát sinh lỗi biên dịch.

3. LỚP (CLASS)

Lớp hay còn gọi là class là sự gói gọn các dữ liệu và các phương thức (method) hoạt động trên dữ liệu đó. Trong C#, lớp được xem là kiểu dữ liệu tham chiếu (reference), và như vậy, các thể hiện (instance) của lớp sẽ được chứa tại vùng nhớ heap và quản lý bởi GC (bộ thu gom rác). Khi một thể hiện của lớp được tạo, bộ nhớ sẽ phân phối vùng nhớ tương ứng trên heap và các tham chiếu đến đối tượng sẽ bắt đầu được đếm. Khi số đếm tham chiếu bằng 0, GC sẽ phục hồi lại vùng nhớ bị chiếm bởi đối tượng và trả bộ nhớ sang trạng thái sẵn sàng để sử dụng.

Lớp có thể chứa các fields (trường), methods (phương thức), events (sự kiện), properties (thuộc tính) và các lớp lồng nhau. Lớp cũng có thể kế thừa từ những lớp khác và phát triển đa giao diện (multiple interfaces).

Cũng giống như struct, mức độ bảo vệ mặc định của lớp là private. Lớp có thể khai báo các thành phần là public, protected, private, internal hay protected internal.

Khai báo một lớp bao gồm các điểm sau:

```

[attributes] [access modifier] class class-name [: [base-
class], [interface]*]
{
    body
}

```

Dấu ngoặc vuông cho biết đây là phần các tùy chọn , không bắt buộc phải khai báo. Ví dụ 2.1-23 trình bày một lớp đơn giản định nghĩa cho kiểu dữ liệu là Dog.

Ví dụ 2.1-23 Lớp đơn giản

```
1: using System;
2:
3: public class Dog {
4:
5:     //Trường
6:     private string Name;
7:
8:     //Phương thức khởi dựng còn gọi là Constructor
9:     public Dog( string name ) {
10:         Name = name;
11:     }
12:
13:     // Phương thức
14:     public void Speak( ) {
15:         Console.WriteLine( "Hello. My name is {0}", Name
16:     }
17: }
18:
19: public class Simple {
20:
21:     public static void Main( ) {
22:
23:         Dog spot = new Dog( "spot" );
24:         spot.Speak( );
25:     }
26: }
```

Lớp Dog trong ví dụ 2.1-23 giải thích cách cơ bản để định nghĩa và phát triển một lớp trong C#. Lớp chứa một trường riêng tư private, một constructor theo kiểu có tham số và duy nhất một phương thức. Không giống như struct , toán tử new phải được sử dụng để tạo một thể hiện của lớp. Dòng 23 trình bày việc tạo một thể hiện của lớp Dog và sử dụng constructor với tham số để khởi tạo cho đối tượng.

Bảng 2.1-3 dưới đây trình bày các bổ từ truy xuất qui định tầm ảnh hưởng của chúng đối với các biến trong lớp

Bảng 2.1-3 Truy xuất cập nhật các thành phần của lớp

Bổ từ

Định nghĩa

public

Tất cả các mã đều thấy được

protected

Chỉ có thể thấy được ở lớp hiện tại và lớp kế thừa

private	Chỉ thấy được ở lớp hiện tại
internal	Chỉ thấy được ở gói (assembly) hiện tại
protected internal	Chỉ thấy được ở gói hiện tại và của kiểu dẫn xuất từ lớp

3.1. Đối tượng (object)

Mỗi lớp trong C# đều được dẫn xuất từ lớp System.Object. Bởi vì tất cả các lớp đều dẫn xuất từ một lớp cơ sở chung, khả năng tạo một lớp tập hợp như collection tổng hợp của các lớp trở thành điều không cần thiết. Mỗi thể hiện của một lớp có thể được xem như là một thể hiện của System.Object.

Lớp System.Object cũng cung cấp một số dịch vụ cơ bản được các lớp khác trong kiến trúc .NET sử dụng. Ví dụ, phương thức Console.WriteLine sẽ dùng phương thức ToString của lớp để trình bày nội dung lớp ra màn hình. Do đó bất cứ lớp nào trong C# cũng có thể định nghĩa chồng lên phương thức ToString và cung cấp phát triển tùy chỉnh đặc biệt đối với lớp. Bảng 2.1-4 liệt kê một số phương thức cơ bản của lớp Object.

Bảng 2.1-4 Lớp System.Object

Phương thức	Mục đích
Equal(Object)	Phép so sánh kiểu Boolean
Finalize	Tương tự như destructor của C++
ToString	Chuyển đổi lớp sang dạng string.

Lớp cơ sở System.Object còn cung cấp thêm một số phương thức về thông tin kiểu (type information), phản chiếu nội dung lớp (reflection), và sao chép (cloning). Những phương thức này nằm ngoài nội dung trình bày của phần này, nhưng việc tìm hiểu chúng cũng rất hữu ích.

3.2. Phương thức (Method)

Trong thuật ngữ của lập trình hướng đối tượng, một phương thức (method) là một hành động của đối tượng. Phương thức có thể được tạo ra và gọi từ thể hiện của đối tượng hoặc là phương thức tĩnh (static) có thể được gọi trực tiếp từ một lớp. Một phương thức gọi bằng thể hiện của đối tượng cần phải tạo ra đối tượng trước khi có thể sử dụng phương thức. Tuy phương thức tĩnh không cần đến việc khởi tạo đối tượng, nhưng nó không thể truy xuất đến các dữ liệu thành phần của lớp.

Phương thức cũng giống như dữ liệu thành phần, nó cũng có thể có những bổ túc truy cập áp dụng. Ví dụ phương thức public cho phép user gọi trực tiếp, nói cách khác, đó là phương thức dùng chung. Phương thức protected chỉ được sử dụng bởi chính đối tượng hay những đối tượng dẫn xuất từ đối tượng đó. Phương thức private chỉ có thể truy xuất bởi lớp đã khai báo phương thức. Các lớp dẫn xuất không thể sử dụng phương thức private của lớp cơ sở.

Ví dụ 2.1-24 dưới đây sẽ giải thích cách sử dụng phương thức thể hiện (instance method) tức là phương thức được gọi chỉ khi hình thành đối tượng và phương thức tĩnh (static method) là phương thức gọi thông qua tên lớp không cần kiến tạo đối tượng.

Ví dụ 2.1-24 Phương thức Instance và Static

```

1: using System;
2:
3: public class MyMath {
4:
5:     //instance method
6:     public long Factorial( long l ) {
7:         return l <= 0 ? 1 : l * Factorial( l - 1 );
8:     }
9:
10:    //static method
11:    public static long SFactorial( long l ) {
12:        return l <= 0 ? 1 : l * SFactorial( l - 1 );
13:    }
14: }
15:
16: public class Methods {
17:
18:     public static void Main() {
19:
20:         //Sử dụng phương thức static
21:         Console.WriteLine("5 Factorial = {0}",
22:                           MyMath.SFactorial( 5 ) );
23:
24:         //Sử dụng phương thức instance
25:         MyMath m = new MyMath();
26:         Console.WriteLine("5 Factorial = {0}",
27:                           m.Factorial( 5 ) );
}

```

Để truy xuất phương thức tĩnh static, tên của phương thức phải đi kèm sau tên của lớp. Dòng 21 của ví dụ 2.1-24 gọi phương thức SFatoial của lớp MyMath bằng cách sử dụng toán tử dấu chấm. C# sử dụng toán tử dấu chấm để truy xuất các thành phần và phương thức. C# thống nhất cách truy xuất các thành phần và phương thức theo toán tử dấu chấm . Trong C++ , việc truy xuất các thành phần có thể thông qua con trỏ, dấu chấm hay truy xuất theo phạm vi (scope) là tùy thuộc vào từng tình huống. C# thì khác hẳn, chỉ toán tử dấu chấm được sử dụng cho mọi truy xuất.

Phương thức instance cần có một đối tượng để gọi. Phương thức Factorial là phương thức loại này bởi vì nó không được khai báo là static. Dòng 24 tạo ra một

thể hiện hay instance của lớp MyMath để gọi nó. Chú ý là trong mọi trường hợp, dấu chấm đều được sử dụng để gọi phương thức tĩnh static cũng như phương thức thể hiện instance.

3.3. Truyền tham số

Mỗi ngôn ngữ đều có cú pháp định nghĩa là một tham số được truyền theo tham trị - một bản sao của tham số được đặt vào stack khi gọi- hay tham biến - một bí danh (alias) của biến được đặt vào stack khi gọi. Trong C và C++, bạn có thể truyền một tham số bằng tham trị, tham biến, con trỏ (pointer). Lập trình viên Visual Basic biết rằng các tham số được truyền mặc định là tham biến và khi muốn tham số được truyền là tham trị thì phải chỉ định rõ ràng.

C# không hỗ trợ việc truyền tham số bằng tham trị hay tham biến mà còn cho phép sử dụng thêm chỉ dẫn như in và out. Những chỉ dẫn này cũng tương tự như in và out dùng trong COM.

Trong C#, các kiểu giá trị như kiểu đơn nguyên (primitive type), struct được truyền bằng tham trị, trừ khi bạn chỉ định khác đi. Khi tham số được truyền bằng tham trị , một bản sao cùng kiểu giá trị sẽ được tạo ra. Phương thức sẽ nhận tham số được truyền và sử dụng để sao chép hay thay đổi giá trị. Tuy nhiên, tham số không hề có sự liên hệ với thế giới bên ngoài. Nếu một phương thức thay đổi giá trị của tham trị, ảnh hưởng chỉ có tác dụng trong bản thân phương thức. Những kiểu tham chiếu như lớp hay giao diện interface, được truyền bằng tham biến, không thể truyền bằng tham trị. Để truyền một kiểu giá trị như là tham biến, chúng ta sử dụng từ khoá ref. Khi một tham số được truyền như tham biến cho một phương thức, phương thức có thể thay đổi giá trị và việc thay đổi này có ảnh hưởng thực sự đến tham số, điều này phát sinh ra hiệu ứng lề (side effect). Ví dụ 2.1-25 giải thích cách truyền tham số trong C#.

Ví dụ 2.1-25 Truyền tham số

```
using System;

//Tạo kiểu cấu trúc mang tên Point
public struct Point {

    public int x;
    public int y;
}

//Tạo kiểu tham chiếu lớp
public class MyObject {
    public int i;
}
```

```
public class Pass {  
    public static void Main() {  
  
        int i = 100;  
  
        Console.WriteLine("Value of i before  
                          PassByValue Method is {0}", i);  
        PassByValue(i);  
        Console.WriteLine("Value of i after  
                          PassByValue Method is {0}", i);  
  
        Console.WriteLine("");  
  
        Console.WriteLine("Value of i before  
                          PassByRef Method is {0}", i);  
        PassByRef(ref i);  
        Console.WriteLine("Value of i before  
                          PassByRef Method is {0}", i);  
  
        Console.WriteLine("");  
  
        //Tạo thể hiện của Point  
        Point p; p.x = 10; p.y = 15;  
        Console.WriteLine("Value of p before  
                          PassByValue is x={0}, y={1}", p.x, p.y);  
        PassByValue(p);  
        Console.WriteLine("Value of p after PassByValue  
                          is x={0}, y={1}", p.x, p.y);  
  
        Console.WriteLine("");  
  
        Console.WriteLine("Value of p before PassByRef  
                          is x={0}, y={1}", p.x, p.y);  
        PassByRef(ref p);  
        Console.WriteLine("Value of p after PassByRef  
                          is x={0}, y={1}", p.x, p.y);  
  
        Console.WriteLine("");  
  
        //Tạo thể hiện của đối tượng  
        MyObject o = new MyObject();  
        o.i = 10;  
        Console.WriteLine("Value of o.i  
                          before PassReferenceType is {0}", o.i);  
        PassReferenceType(o);  
    }  
}
```



```
Console.WriteLine("Value of o.i after
                  PassReferenceType is {0}", o.i );
}

public static void PassByValue( Point p ) {
    Console.WriteLine( "Entering public static void
                      PassByvalue( Point p )" );

    Console.WriteLine( "Value of Point.x = {0} :
                      Point.y = {1}" , p.x, p.y );
    p.x++;
    p.y++;
    Console.WriteLine( "New Value of
                      Point.x = {0} : Point.y = {1}" , p.x, p.y );

    Console.WriteLine( "Exiting public static void
                      PassByvalue( Point p )" );
}

public static void PassByValue( int i ) {
    Console.WriteLine( "Entering public static
                      void PassByValue( int i )" );

    Console.WriteLine("Value of i = {0}" , i );
    i++;
    Console.WriteLine("New Value of i = {0}" , i );

    Console.WriteLine( "Exiting public static void
                      PassByValue( int i )" );
}

public static void PassByRef( ref Point p ) {
    Console.WriteLine( "Entering public static
                      void PassByRef( ref Point p )" );

    Console.WriteLine("Value of Point.x = {0} :
                      Point.y = {1}" , p.x, p.y );
    p.x++; p.y++;
    Console.WriteLine("New Value of Point.x = {0} :
                      Point.y = {1}" , p.x, p.y );

    Console.WriteLine( "Exiting public static
                      void PassByRef( ref Point p )" );
}

public static void PassByRef( ref int i ) {
    Console.WriteLine( "Entering public static
```



```
        void PassByRef( ref int i ) ;

    Console.WriteLine("Value of i = {0}", i );
    i++;
    Console.WriteLine("New Value of i = {0}", i );

    Console.WriteLine( "Exiting public static
                      void PassByRef( ref int i )" );
}

public static void PassReferenceType( MyObject o ) {
    Console.WriteLine( "Entering public static void
                      PassReferenceType( MyObject o )" );

    Console.WriteLine("Value of MyObject.i = {0}",
                      o.i);
    o.i++;
    Console.WriteLine("New Value of MyObject.i = {0}",
                      o.i);

    Console.WriteLine( "Exiting public static void
                      PassReferenceType( MyObject o )" );
}
}
```

Việc truyền tham số trong ví dụ 2.1-25 thể hiện trường hợp truyền tham số kiểu đơn nguyên, struct và kiểu tham chiếu cho phương thức bằng giá trị và bằng tham biến.

Kết quả của ví dụ 2.1-25 như sau:

```
Value of i before PassByValue Method is 100
Entering public static void PassByValue( int i )
Value of i = 100
New Value of i = 101
Exiting public static void PassByValue( int i )
Value if i after PassByValue Method is 100
```

```
Value of i before PassByRef Method is 100
Entering public static void PassByRef ( int i )
Value of i = 100
New Value of i = 101
Exiting public static void PassByRef ( int i )
Value if i after PassByRef Method is 101
```

```
Value of p before PassByValue is x=10, y=15
Entering public static void PassByValue( Point p )
Value of Point.x = 10 : Point.y = 15
NewValue of Point.x = 11 : Point.y = 16
```

```
Exiting public static void PassByValue( Point p )
Value of p after PassByValue is x=10, y = 15
```

```
Value of p before PassByRef is x=10, y=15
Entering public static void PassByRef ( Point p )
Value of Point.x = 10 : Point.y = 15
NewValue of Point.x = 11 : Point. PassByRef = 16
Exiting public static void PassByRef ( Point p )
Value of p after PassByRef is x=11, y = 16
```

```
Value of o.i before PassReferenceType is 10
Entering public static void PassReferenceType(MyObject o )
Value of MyObject.i = 10
NewValue of MyObject.i = 11
Exiting public static void PassReferenceType(MyObject o )
Value of o.i after PassReferenceType is 11
```

3.4. Thuộc tính (Properties)

Trong thế giới C++ và COM, Property có ý nghĩa đơn giản là những phương thức assessor (lấy giá trị) và setter (đặt giá trị). Trong COM, những phương thức này sẽ là `put_T` và `get_T` trong đó `T` là tên thuộc tính. Lập trình viên Visual Basic ngay lập tức sẽ thấy quen thuộc với khái niệm thuộc tính này bởi vì cũng có những phương thức tương tự.

C# cho phép thuộc tính có thể chỉ đọc (read only), chỉ ghi (write only) hay vừa đọc vừa ghi (read/write), trong đó thuộc tính chỉ ghi ít khi được dùng.

Cấu trúc khai báo một thuộc tính property có cú pháp như sau:

```
access-modifier return-typePropertyName {
    [get { statement; }]
    [set { statement; }]
}
```

Thuộc tính cung cấp cú pháp đơn giản để truy cập các yếu tố trong một lớp trong khi vẫn quan tâm đến việc cài đặt ở bên trong. Ví dụ 2.1-26 xem lại phương pháp Newton để tính căn bậc hai và phát triển hai thuộc tính là Value và Result.

Ví dụ 2.1-26 Cách sử dụng Property

```
1: using System;
2:
3:
4: public class Newton {
5:
6:     private double m_dblValue;
7:     private double m_dblResult;
8:
```

```
9:
10:    public Newton( ) {
11:        m_dblValue = 1.0;
12:        m_dblResult = 0.0;
13:    }
14:
15:    //Properties
16:
17:    //Value is set/get
18:    public double Value {
19:        set {
20:            if( value <= 0 ) {
21:                Console.WriteLine("Value must be greater
22:                                than Zero");
23:            }
24:            m_dblValue = value;
25:        }
26:
27:        get { return m_dblValue; }
28:    }
29:
30:
31:    public double Result {
32:        get { return m_dblResult; }
33:    }
34:
35:
36:    public void FindSqrt( ) {
37:        const double Epsilon = 1.0e-9;
38:        double Guess = 11;
39:
40:        m_dblResult = ((m_dblValue / Guess) + Guess) / 2;
41:
42:        while( Math.Abs( m_dblResult - Guess ) > Epsilon
43:              ) {
44:            Guess = m_dblResult;
45:            m_dblResult = ((m_dblValue / Guess)
46:                           + Guess) / 2;
47:        }
48:
49:
50:
51:
52:    public class PropertyTest {
53:
```



```

54: public static void Main() {
55:
56:     Newton n = new Newton();
57:
58:
59:     n.Value = 100;
60:
61:     n.FindSqrt();
62:
63:
64:     Console.WriteLine("The Sqrt of {0} is {1}",
65:                         n.Value, n.Result);
66: }

```

Trở lại ví dụ của Newton, Ví dụ 2.1-26 trình bày cách cài đặt phương pháp Newton để tính căn bậc hai của một số. Lớp Newton cài đặt hai thuộc tính là Value và Result. Thuộc tính Value phát triển cả hai phương thức get và set. Điều này cho phép thuộc tính có thể đọc và ghi. thuộc tính Result chỉ phát triển phương thức get, kết quả là thuộc tính chỉ có thể đọc (read only).

Mục đích của thuộc tính là cho phép truy cập đến các dữ liệu thành phần theo ngôn ngữ tự nhiên mà không cần để ý xử lý ở bên trong. Khi phát triển thuộc tính get và set, bạn có thể thoải mái thực hiện kiểm tra, chuyển đổi và những xử lý logic khác nếu cần thiết. Ở khía cạnh người dùng, tất cả những phát triển bên trong là trùu tượng và việc truy xuất dữ liệu của các thành phần đối với họ là hết sức tự nhiên, rành mạch.

3.5. Toán tử (Operator)

C# cung cấp khả năng cho phép người dùng định nghĩa các toán tử (operators). Bất cứ struct hay lớp đều có thể cung cấp những phát triển của các toán tử cho trước, như phép cộng, trừ hay ép kiểu từ kiểu này sang kiểu khác. Khả năng tạo ra những kiểu giá trị mới cộng với việc định nghĩa những toán tử mới tạo ra khả năng phát triển những kiểu giá trị cũng như các kiểu tham chiếu mới.

Như đã bàn bạc ở phần đầu về toán tử gán và sự khác biệt khi sử dụng nó đối với kiểu giá trị (value type) và kiểu tham chiếu (reference type), C# không cho phép phát triển toán tử gán. Những người phát triển trên C++ bây giờ sẽ có vẻ thắc mắc. Sự hạn chế này là bởi việc đếm các tham chiếu đến biến. Một nguyên nhân nữa là .NET được xây dựng nhằm để các ngôn ngữ khác nhau có thể giao tiếp với nhau. Để sao chép, hầu hết các kiểu tham chiếu (reference type) đều cung cấp một phương thức Copy. Khi xây dựng một lớp, bạn phải theo đúng thiết kế như vậy.

C# yêu cầu tất cả các toán tử (operator) phải là static phương thức. Điều này làm cho ngôn ngữ thêm dễ hiểu hơn. Toán tử gắn liền với một kiểu chứ không phải là một thể hiện của đối tượng.

Để hiểu thêm về việc mở rộng toán tử (operator), lớp Fraction trong ví dụ 2.1-27 dưới đây sẽ giải thích cách phát triển hai phép tính toán học + và -.



Ví dụ 2.1-27 Định nghĩa chồng toán tử (Operation Overloading)

```
1: //File      :part02_26.cs
2: //Author :Richard L. Weeks
3: //Purpose  :Demonstrate operator overloading
4:
5: using System;
6:
7: public class Fraction {
8:
9:     //data members
10:    private int m_numerator;
11:    private int m_denominator;
12:
13:    //Properties
14:    public int Numerator {
15:        get { return m_numerator; }
16:        set { m_numerator = value; }
17:    }
18:    public int Denominator {
19:        get { return m_denominator; }
20:        set { m_denominator = value; }
21:    }
22:
23:
24:    //Constructors
25:    public Fraction() {
26:        m_numerator = 0;
27:        m_denominator = 0;
28:    }
29:
30:    public Fraction( int iNumerator,
31:                      int iDenominator ) {
32:        m_numerator = iNumerator;
33:        m_denominator = iDenominator;
34:    }
35:
36:
37:    //Arithmetic operators +,-,/,*
38:    public static Fraction operator+(Fraction f1,
39:                                     Fraction f2) {
40:        Fraction Result = new Fraction();
41:
42:        if( f1.Denominator != f2.Denominator ) {
43:            Result.Denominator = f1.Denominator *
44:                f2.Denominator;
45:            Result.Numerator = f1.Numerator *
46:                f2.Denominator + f2.Numerator *
47:                f1.Denominator;
48:        }
49:        else
50:            Result.Numerator = f1.Numerator +
51:                f2.Numerator;
52:            Result.Denominator = f1.Denominator;
53:    }
54:
55:
```





```
79:  
80:     f3 = f3 - f2;  
81:     Console.WriteLine("f3 - f2 = {0}/{1}",  
                           f3.Numerator, f3.Denominator );  
82: }  
83: }
```

Lớp Fraction trình bày trong ví dụ 2.1-27 trên phát triển cả hai toán tử + và -. Toán tử cộng được phát triển ở dòng 34, và toán tử trừ ở dòng 49. Hình thức mở rộng chung của một toán tử có thể biểu diễn như sau:

```
public static return-type operator T(param p [,param p1])
```

Trong đó *return-type* chỉ định kết quả của toán tử, *T* là toán tử thực sự sẽ bị chồng lên, và số lượng các tham số (*parameter*) tùy thuộc vào toán tử đang được định nghĩa chồng lên.

Để mở rộng các toán tử toán học chuẩn, C# cung cấp khả năng định nghĩa chồng các toán tử quan hệ (*relational*) và các toán tử ép kiểu (*casting*). Toán tử quan hệ thường đi theo từng cặp. Ví dụ, khi định nghĩa chồng lên toán tử quan hệ ==, toán tử tương đương là != cũng phải được định nghĩa luôn. Toán tử quan hệ khi định nghĩa chồng cũng được sử dụng tương tự như những toán tử khác.

Toán tử ép kiểu (*casting*) có ngữ nghĩa khác chút ít so với các toán tử thông thường. Khi phát triển một toán tử ép kiểu, bạn phải quyết định ngầm định (*implicit*) hay tường minh (*explicit*). Cần nhớ là một ép kiểu dạng ngầm định thì không cần chỉ định kiểu, trong khi ép kiểu dạng tường minh đòi hỏi phải có.

```
Fraction f = new Fraction(1, 5);  
double d = f;           //ép kiểu ngầm định  
double dd = (double)f; //ép kiểu tường minh
```

Cú pháp để định nghĩa chồng một toán tử ép kiểu như sau:

```
public static [implicit|explicit] operator Return-Type (  
                                         Type T)
```

Xin nhắc lại, *Return-Type* chỉ rõ kiểu mà *Type T* sẽ được ép hay chuyển sang. Mở rộng ví dụ trước, lớp Fraction ở ví dụ 2.1-28 được mở rộng để phát triển một toán tử ép kiểu dạng tường minh sang kiểu double và toán tử quan hệ == cũng như !=. Khi biên dịch phần mã của ví dụ 2.1-28, trình biên dịch sẽ phát sinh hai cảnh báo CS660 và CS661. Cảnh báo này sinh từ việc phát triển các toán tử == và != và yêu cầu ở đây là bất cứ lớp nào phát triển những toán tử này phải thực hiện phát triển thêm những phương thức *Object.Equals* và *Object.GetHashCode*. Bây giờ, bạn tạm thời bỏ qua những cảnh báo này. Tuy nhiên khi tạo mã sản phẩm, nên chắc chắn là phải phát triển những phương thức cần thiết là *Equals* và *GetHashCode* nhằm đảm bảo luật có thể khẳng định rằng hai đối tượng nào đó được xem là bằng nhau thông qua phương thức *Equals*, hay bằng những toán tử == và != có cùng mã băm (hash code).

**Ví dụ 2.1-28 Mở rộng lớp Fraction**

```
1: //File      :part02_27.cs
2: //Author:Richard L. Weeks
3: //Purpose  :Demonstrate operator overloading
4:
5:
6: using System;
7:
8:
9:
10: public class Fraction {
11:
12:     //data members
13:     private int m_numerator;
14:     private int m_denominator;
15:
16:
17:     //Properties
18:     public int Numerator {
19:         get { return m_numerator; }
20:         set { m_numerator = value; }
21:     }
22:     public int Denominator {
23:         get { return m_denominator; }
24:         set { m_denominator = value; }
25:     }
26:
27:
28:     //Constructors
29:     public Fraction() {
30:         m_numerator = 0;
31:         m_denominator = 0; }
32:
33:     public Fraction( int iNumerator,
34:                      int iDenominator ) {
35:         m_numerator = iNumerator;
36:         m_denominator = iDenominator;
37:     }
38:
```



```
35: 
36: //Toán tử +,-,/,*
37: 
38: public static Fraction operator+(Fraction f1,
39:                                     Fraction f2) {
40: 
41:     Fraction Result = new Fraction();
42: 
43:     if( f1.Denominator != f2.Denominator ) {
44:         Result.Denominator = f1.Denominator *
45:                               f2.Denominator;
46:         Result.Numerator = (f1.Numerator *
47:                               f2.Denominator) + (f2.Numerator *
48:                               f1.Denominator);
49:     } else {
50:         Result.Denominator = f1.Denominator;
51:         Result.Numerator = f1.Numerator +
52:                               f2.Numerator;
53:     }
54:     return Result;
55: }
56: 
57: public static Fraction operator-(Fraction f1,
58:                                     Fraction f2) {
59: 
60:     Fraction Result = new Fraction();
61: 
62:     if( f1.Denominator != f2.Denominator ) {
63:         Result.Denominator = f1.Denominator *
64:                               f2.Denominator;
```

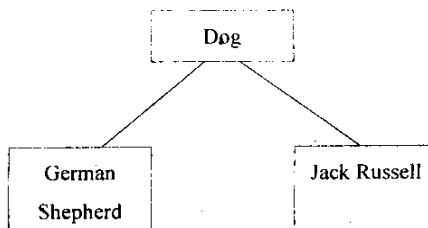



```
99:  
100:    //Subtract f2 from f3 should get f1  
101:    f3 = f3 - f2;  
102:    Console.WriteLine("f3 - f2 = {0}/{1}",  
103:        f3.Numerator, f3.Denominator );  
104:    //Print f3 as a double  
105:    Console.WriteLine("f3 as a double = {0}",  
106:        (double)f3);  
107: }
```

4. KẾ THỪA (INHERITANCE)

Kế thừa là khái niệm then chốt của thiết kế trong các ngôn ngữ hướng đối tượng. Kế thừa cho phép những chức năng (functionality) và thuộc tính (attribute) dùng chung ở trong lớp cơ sở (base class), và những lớp chỉ định có thể kế thừa những chức năng của lớp cơ sở. C# chỉ hỗ trợ kế thừa đơn (single inherit). C++ có phép đa kế thừa (multiple inheritance) và nếu được sử dụng đúng cách, đây là sự lấp đầy rất mạnh. Tuy nhiên, phải thừa nhận là đa kế thừa rất khó quản lý như vậy, cũng rất khó áp dụng. Đây là một trong những lý do C# chỉ phát triển kế thừa đơn.

Hình 2.1-1 Trình bày trường hợp chung của kế thừa



Hình 2.1-1 Kế thừa cơ bản

Lớp cơ sở Dog sẽ chứa thuộc tính và phương thức chung cho tất cả giống chó. Mỗi lớp dẫn xuất có thể cài đặt cá biệt hóa thêm nếu thấy cần thiết. Điều quan trọng nên chú ý rằng C# chỉ cung cấp kế thừa dạng public (xem ví dụ 2.1-2). Có lẽ lúc này những người phái triển trên C++ sẽ lại than phiền.

Ví dụ 2.1-29 Kế thừa

```
1: //File      :part02_28.cs  
2: //Purpose   :Demonstrate basic inheritance  
3: //
```

```
4:
5: using System;
6:
7: public class Dog {
8:
9:     //Common attributes
10:    public string Name;
11:    public int Weight;
12:
13:
14:    //Common methods
15:    public void Speak() {
16:        Console.WriteLine("ruff!");
17:    }
18:    public void DrinkWater() {
19:        Console.WriteLine("Gulp");
20:    }
21:    //Lớp kế thừa
22:    public class GermanShepard : Dog {
23:        public void OnGuard() {
24:            Console.WriteLine("In Guard Mode");
25:        }
26:
27:        public class JackRussell : Dog {
28:            public void Chew() {
29:                Console.WriteLine("I'm chewing your
30:                                favorite shoes!");
31:            }
32:
33:            public static void Main() {
34:
```

```

35:     GermanShepard Simon = new GermanShepard();
36:     JackRussell Daisy = new JackRussell();
37:
38:     //Both Simon and Daisy have a name and weight
39:     Simon.Name = "Simon"; Simon.Weight = 85;
40:     Daisy.Name = "Daisy"; Daisy.Weight = 25;
41:
42:
43:     Simon.Speak(); Simon.DrinkWater();
44:     Daisy.Speak(); Daisy.DrinkWater();
45:
46:
47:     //Only Simon has the OnGuard Method
48:     Simon.OnGuard();
49:
50:     //Only Daisy has the Chew method
51:     Daisy.Cheat();
52: }
53: }
```

Hình 2.1-1 được phát triển trong C# và được thể hiện trong ví dụ 2.1-29. Nên nhớ rằng C# chỉ hỗ trợ kế thừa đơn, vì vậy không thể tạo lớp dẫn xuất từ hai lớp cơ sở trở lên. Cú pháp để kế thừa từ một lớp cơ sở là đặt tên của lớp cơ sở ở bên phải dấu hai chấm theo sau tên của lớp dẫn xuất như trình bày sau đây:

```
class Bar : Foo ()
```

Trong trường hợp này, Bar là lớp dẫn xuất và Foo là lớp cơ sở.

5. ĐA HÌNH (POLYMORPHISM)

Trong thuật ngữ lập trình hướng đối tượng, khả năng ghi đè phương thức của lớp cơ sở và cung cấp một phát triển khác ở trong lớp dẫn xuất là một hình thức cơ bản của khái niệm đa hình.

Hãy xem xét ví dụ trước, trong đó GermanShepard và JackRussell được dẫn xuất từ lớp cơ sở chung Dog. Lớp cơ sở Dog cung cấp phương thức Speak đưọc các lớp dẫn xuất thừa kế. Tuy nhiên, nếu bạn đã từng nghe thì giống German Shepard sủa sẽ khác với giống Jack Russel, chúng sủa không giống nhau. Vì vậy lớp cơ sở Dog phải cho phép các lớp dẫn xuất cung cấp phát triển phương thức Speak của riêng chúng.

C# cung cấp từ khóa virtual để chỉ rõ là phương thức có thể bị các lớp dẫn xuất ghi đè (overridden). Một lớp dẫn xuất có thể ghi đè phương thức virtual bằng cách sử dụng từ khóa override.

```
Class Dog{
    public virtual void Speak() {...}
}
Class JackRussell : Dog{
    public override void Speak() {...}
}
```

Sức mạnh thực sự của đa hình (polymorphism) thể hiện khi kiểu của đối tượng (object type) chưa xác định ở thời điểm biên dịch chương trình. Điều này cho phép thông tin kiểu lúc thực thi (runtime type) được sử dụng để xác định phương thức thực sự được gọi. Ví dụ 2.1-30 cập nhật lớp cơ sở Dog và cung cấp phương thức ảo speak.

Ví dụ 2.1-30 Phương thức ảo (virtual)

```
1: //File      :part02_28.cs
2: //Purpose   :Demonstrate basic inheritance
3: //
4:
5: using System;
6:
7: public class Dog {
8:
9:     //Common attributes
10:    public string Name;
11:    public int Weight;
12:
13:
14:     //Common methods
15:    public virtual void Speak() {
16:        Console.WriteLine("ruff!");
17:    }
18:    public void DrinkWater() {
19:        Console.WriteLine("Gulp");
20:    }
21:
22: }
23:
24: public class GermanShepard : Dog {
```



```
21:
22:     public void OnGuard() {
23:         Console.WriteLine("In Guard Mode");
24:
25:     public override void Speak() {
26:         Console.WriteLine("RUFF,RUFF,RUFF");
27:     }
28:
29: public class JackRussell : Dog {
30:     public void Chew() {
31:         Console.WriteLine("I'm chewing your
32:                         favorite shoes!");
33:     }
34:
35: }
36:
37: public class Inherit {
38:
39:     public static void Main() {
40:
41:         GermanShepard Simon = new GermanShepard();
42:         JackRussell Daisy = new JackRussell();
43:
44:         //Both Simon and Daisy have a name and weight
45:         Simon.Name = "Simon"; Simon.Weight = 85;
46:         Daisy.Name = "Daisy"; Daisy.Weight = 25;
47:
48:         Dog d = Simon;
49:         d.Speak(); //calls GermanShepard.Speak();
50:
51:         d = Daisy;
52:         d.Speak(); //calls JackRussell.Speak();
```

```

53:
54:    }
55:    }

```

Bây giờ ví dụ về kế thừa sau khi sửa lại đã sử dụng phương thức ảo Speak. Cả hai lớp dẫn xuất GermanShepard và JackRussel đều phát triển đặc biệt hoá phương thức Speak của lớp cơ sở Dog.

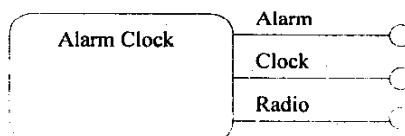
Chú ý việc khai báo biến kiểu Dog và khởi tạo của nó tham chiếu đến Simon. Đây mới là điểm thú vị. Khi phương thức Speak được gọi, kiểu của đối tượng được định nghĩa và dựa trên thông tin này, phát triển phù hợp với phương thức Speak được gọi. Trong trường hợp này là GermanShepard.Speak(). Theo sau lời gọi d.Speak(), biến d được gán tham chiếu đến Daisy. Một lần nữa, d.Speak() lại được gọi và thông tin kiểu được truy xuất để định ra phương thức phù hợp.

Đa hình (polymorphism) thể hiện một mô hình mạnh mẽ trong thế giới hướng đối tượng. Khi được sử dụng đúng cách, các thành phần có thể tương tác lẫn nhau mà không cần phải có những kiến trúc riêng biệt.

6. GIAO DIỆN (INTERFACES) HOẶC GIAO TIẾP

Giao diện (Interfaces) cung cấp hình thức trừu tượng hoá rất thuận tiện cho việc phát triển theo phong cách xây dựng theo thành phần cơ sở (component-based). Interfaces cung cấp những thỏa thuận chung cho phép các thành phần làm việc với nhau. Trong một trường hợp cụ thể hơn, đồng hồ báo thức được đặt bên cạnh giường của bạn, phát triển rất nhiều interfaces – alarm interface, Clock interface, và còn có thể là Radio interface.

Hình 2.1-2 vẽ một thành phần AlarmClock trong đó có hỗ trợ một Alarm interface, một Clock interface và một Radio interface. Hình 2.1-2 được coi như là mô hình hộp muỗng, trong đó các muỗng là những đường thẳng với những vòng tròn ở đầu và phần đuôi muỗng đính vào phần ngoài của hộp.



Hình 2.1-2 *AlarmClock component và interfaces*

Một interface không phải là một lớp, và do đó, cũng không có bất cứ mã cài đặt nào. Một interface chứa các khai báo của các thành phần (members) và các phương thức (methods), trong đó, tất cả đều phải khai báo public. C# cho phép cả struct và lớp cài đặt một hay nhiều interfaces. Điều này khác hẳn việc kế thừa (inheritance). Kế thừa là việc cài đặt một lớp từ lớp cơ sở, còn cài đặt interface thì chỉ xác định ra rằng lớp sẽ phát triển một interface đặc biệt, và cam đoan sẽ hoàn thành interface như đã đề ra.



C# sử dụng từ khoá **interface** để thể hiện việc khai báo của một giao diện.

```
interface name {  
    body;  
}
```

Một lớp hay struct có thể cài đặt một interface bằng việc khai báo interface trong quá trình định nghĩa. Ví dụ:

```
public class AlarmClock : IAlarm, IClick, IRadio {  
    // cài đặt.  
}
```

Chú ý: Một quy ước tiêu chuẩn áp dụng khi khai báo interface là tên của interface nên được bắt đầu bằng ký tự chữ I hoa. Quy ước cách đặt tên này thể hiện tinh thần quan và giúp dễ dàng phân biệt đâu là interface , tránh sự nhầm lẫn với định nghĩa của lớp và struct.

Kiến trúc .NET sử dụng thường xuyên các interface. Một điều chắc chắn là, để tận dụng ưu điểm của mọi service được cung cấp bởi .NET, bạn bắt buộc phải hiểu rõ cách phát triển và cài đặt đối với interface. Ví dụ 2.1-31 sẽ cài đặt lớp AlarmClock thể hiện trọn vẹn interface được trình bày trong hình 2.1-2.

Ví dụ 2.1-31 Lớp AlarmClock

```
1: //File :part02_30.cs  
2: //Author:Richard L. Weeks  
3: //Purpose :Interfaces  
4:  
5: using System;  
6:  
7: //định nghĩa IAlarm interface  
8: interface IAlarm {  
9:     bool On { get; set; }  
10:    void Snooze();  
11: }  
12:  
13: //định nghĩa IClock interface  
14: interface IClock {  
15:     void SetTime();  
16: }  
17:  
18:  
19: //định nghĩa the IRadio interface  
20: interface IRadio {  
21:     void SetStation( double station_id );  
22: }
```



```
24:  
25:  
26: //Tạo lớp AlarmClock cài đặt IAlarm, IClock và IRadio  
27: public class AlarmClock : IAlarm, IClock, IRadio {  
28:  
29:     //Data members  
30:     private bool m_bOnOff;  
31:  
32:  
33:     //The IAlarm interface implementation  
34:     public bool On {  
35:         get { return m_bOnOff; }  
36:         set { m_bOnOff = value; }  
37:     }  
38:     public void Snooze() {  
39:         Console.WriteLine("IAlarm.Snooze");  
40:     }  
41:     //IClock Interface  
42:     public void SetTime() {  
43:         Console.WriteLine("IClock.SetTime");  
44:     }  
45:     //IRadio interface  
46:     public void SetStation( double station_id ) {  
47:         Console.WriteLine("IRadio.SetStation( {0} )", station_id );  
48:     }  
49:  
50:  
51:     public class InterfaceTest {  
52:         public static void Main() {  
53:             AlarmClock a = new AlarmClock();  
54:  
55:             //Get the IAlarm Interface  
56:             IAlarm ialarm = (IAlarm)a;  
57:             ialarm.On = false;  
58:             ialarm.Snooze();  
59:  
60:             //Get the IClock interface  
61:             IClock iclock = (IClock)a;  
62:             iclock.SetTime();  
63:  
64:             //Get the IRadio interface  
65:             IRadio iradio = (IRadio)a;  
66:             iradio.SetStation( 98.1 );
```



For Evaluation Only.

63: }

64: }

Hình 2.1-2 được triển khai trong ví dụ 2.1-31. Toán tử ép kiểu được sử dụng để nhận được interface yêu cầu (xem dòng 52 của ví dụ 2.1-31). Interface IAlarm được sử dụng bởi thể hiện của AlarmClock. Trong trường hợp lớp AlarmClock không hỗ trợ IAlarm interface, ngoại lệ InvalidCastException sẽ được phát sinh. Nếu interface tồn tại, một con trỏ đến biến AlarmClock sẽ được trả về. Như vậy, bây giờ ta có hai tham chiếu đến AlarmClock, một là tham chiếu của biến AlarmClock, hai là tham chiếu đến IAlarm interface. Cả hai tham chiếu này đều được giải phóng trước khi GC thu gom thể hiện của AlarmClock.

7. MÔ HÌNH CHUYỂN GIAO (DELEGATES)

Delegate là con trỏ hàm cơ bản. Lập trình viên C và C++ rất quen thuộc với con trỏ hàm (function pointer) nhưng không có ý niệm về thể hiện (instance) của một con trỏ hàm. Một delegate có thể hiểu như là một cơ chế callback, có nghĩa là “Khi đúng thời điểm, hãy gọi phương thức này cho tôi”.

Hãy xem xét kịch bản sau: Bộ phận làm việc của bạn vừa mới thuê một nhân viên mới. Bộ phận Nhân sự cần được báo khi có nhân viên mới được thuê để họ có thể thu xếp tiến hành hàng loạt những công việc sổ sách giấy tờ rườm rà cũng như ra sức huấn luyện nhân viên mới về quy chế của công ty cũng như công tác huấn luyện nghiệp vụ. Những công việc cực nhọc này của bộ phận nhân sự cứ lặp đi lặp lại. Đây chính là ví dụ hoàn hảo cho một Delegate. Cơ bản là, Bộ phận nhân sự đòi hỏi được thông báo khi có nhân viên mới được thuê và sẽ cung cấp lời gọi để thực hiện một phương thức. Ví dụ 2.1-32 trình bày cách sử dụng cơ bản của delegate.

Ví dụ 2.1-31 Lớp AlarmClock

```

1: //File      :part02_31.cs
2: //Author :Richard L. Weeks
3: //Purpose  :Demonstrate the use of delegates
4:
5: using System;
6:
7: //Define a person struct
8: public struct Person {
9:   public string FName;
10:  public string LName;
11: }
12:
13: //định nghĩa Delegate

```



```
14: public delegate void OnNewHire( Person person );
15:
16: //Lớp HR
17: public class HR {
18:
19:
20: public void OnNewHire( Person person ) {
21:     Console.WriteLine("HR is in the process
22:         of putting {0} to sleep", person.FName );
23: }
24:
25: //Create a department
26: public class Department {
27:
28:     //Who to notify
29:     private OnNewHire m_OnNewHireDelegate = null;
30:
31:     //set the OnNewHire delegate
32:     public void AddOnNewHireDelegate( OnNewHire onh ) {
33:         m_OnNewHireDelegate = onh;
34:     }
35:
36:
37:     public void HirePerson( Person p ) {
38:         //do we need to notify someone?
39:         if( m_OnNewHireDelegate != null )
40:             m_OnNewHireDelegate( p );
41:     }
42: }
43:
44:
45: public class DelegateTest {
46:
47:     public static void Main( ) {
48:
49:         HR hr = new HR( );
50:         Department dept = new Department( );
```

51: For Evaluation Only.

52: //Register the OnNewHire Delegate

53: dept.AddOnNewHireDelegate(new OnNewHire(

 hr.OnNewHire));

54:

55: //Create a person

56: Person me; me.FName = "Richard";

 me.LName = "Weeks";

57:

58: //Hire ME!!!

59: dept.HirePerson(me);

60: }

61: }

Ví dụ 2.1-32 cài đặt kịch bản làm việc của bộ phận nhân sự và sử dụng Delegate để thông báo cho bộ phận nhân sự khi có người mới được thuê. Delegate OnNewHire được định nghĩa ở dòng 14. Hãy chú ý cách sử dụng của từ khoá **delegate** là để chỉ rõ Delegate được khai báo. Nên nhớ rằng, C# không cho phép khai báo phương thức toàn cục, vì vậy C# sẽ phát sinh lỗi nếu bạn thiếu từ khoá delegate.

Lớp HR cung cấp một trình điều khiển cho Delegate. Tên của phương thức không nhất thiết phải đặt trùng tên với Delegate, việc đặt trùng tên ở đây nhằm cho các bạn dễ theo dõi hơn. Lớp Department cung cấp phương thức AddOnNewHireDelegate để điều khiển việc kết nối với trình điều khiển của Delegate. Lời gọi hàm ở dòng 53 mới thực sự ghép trình điều khiển delegate của HR cho Department. Một delegate là một kiểu trong C# và đòi hỏi phải là một thể hiện, do đó phải sử dụng từ khoá new để tạo một Delegate mới. Chúng ta sẽ gặp lại Delegate thật chi tiết ở phần sau bởi vì nó được sử dụng thường xuyên trong .NET, đặc biệt là trong việc phát triển Window Forms.

8. KẾT CHƯƠNG

Bây giờ có lẽ bạn đã biết là C# rất mạnh mẽ và cũng rất dễ dùng. Phát triển hướng giao diện là đề tài nổi bật trong những năm gần đây, và C# được thiết kế theo hướng này với phong cách phát triển đơn giản và nhất quán. Đặc biệt với sự bổ sung thêm khái niệm chuyển giao Delegate, C# càng trở nên một ngôn ngữ mạnh mẽ. Chương kế tiếp sẽ bàn về những chủ đề như tập hợp Collection, thuộc tính Attributes, và sự hỗ trợ XML.

Chương 2.2

C# NÂNG CAO

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Tập hợp (Collection) của .NET.*
- ✓ *Đặc tính Attribute.*
- ✓ *Tuần tự hóa XML Serialization*

1. TẬP HỢP (COLLECTION) CỦA .NET

Tập hợp là nơi chứa một danh sách các đối tượng tương tự mảng. Không bao giờ có một tập hợp cụ thể dành riêng cho các đối tượng như nhân viên, xe hơi, nhà cửa, xe cộNET cung cấp cho bạn một lớp đối tượng tổng quát Collections có khả năng chứa tất cả các đối tượng.

Các lớp đối tượng tập hợp trong C# phải cài đặt giao tiếp IEnumerable. Giao tiếp IEnumerable cho phép duyệt qua những phần tử chứa trong tập hợp. Trong chương này chúng ta sẽ xem qua những tập hợp sau: Stack, Queue, và Hashtable. Cùng với những tập hợp cơ bản này, chúng ta sẽ cài đặt danh sách liên kết theo giao tiếp IEnumerable.

Để biết những thông tin chi tiết về tập hợp (collection), cấu trúc dữ liệu (data structure), và những thuật toán (algorithm). Bạn nên đọc những cuốn sách sau : Algorithms C++ by Sedgewick, The Art of computer programming, Volume 3 by Donald knuth.

1.1. Stack

Stack đại diện cho cơ chế FILO (First in last out – vào trước ra sau), và lớp chứa đựng (container). Giả thiết cơ bản của stack là đặt những phần tử trên đỉnh của stack và lấy những phần tử ra từ đỉnh của stack. Trong C#, System.Collections.Stack (nhìn ví dụ 2.2-1) cũng cài đặt giao tiếp IEnumerable, giao tiếp IEnumerable quan tâm đến việc liệt kê nội dung của Stack.

Ví dụ 2.2-1 : System.Collection.Stack

```
1: using System;
2: using System.Collections;
3:
4: public class StackTest {
5:
6:     public static void main() {
7:
8:         Stack myStack = new Stack();
```

```
9:  
10:    for (int i = 0; i < 10; i++)  
11:        myStack.Push(i);  
12:  
13:    for (int i = 0; i < 10; i++)  
14:        Console.WriteLine("{0}", myStack.Pop());  
15:  
16:  
17:  
18:    for (int i = 0; i < 10; i++)  
19:        myStack.Push(i);  
20:  
21:    foreach (int i in myStack)  
22:        Console.WriteLine("{0}", i);  
23:    }  
24: }
```

Mã chương trình trong ví dụ 2.2.1 cho thấy cách sử dụng cơ bản về stack. Những phần tử được đặt vào stack và được lấy ra theo thứ tự ngược lại. Stack hữu dụng cho những thuật toán đệ qui và có thể phục vụ như những nơi tiếp nhận dữ liệu tạm.

1.2. Hàng đợi (Queue)

Hàng đợi đại diện cho tập hợp có tính năng FIFO (First in First out – vào trước ra trước). Hàng đợi cung cấp những phương thức Enqueue và Dequeue để đưa những phần tử vào hàng đợi và lấy những phần tử ra khỏi hàng đợi. Cũng như với tất cả những tập hợp collection khác của .NET, tập hợp queue cũng cung cấp giao tiếp IEnumerator. Ví dụ 2.2.2 sử dụng Queue được cung cấp bởi .NET để giải thích việc sử dụng chức năng cơ bản của hàng đợi.

Ví dụ 2.2-2 : Hàng đợi

```
1: using System;  
2: using System.Collections;  
3:  
4: public class QueueTest {  
5:  
6:     public static void Main() {  
7:         Queue myQueue = new Queue();  
8:  
9:         // đặt các phần tử vào hàng đợi  
10:        for (int i = 0; i < 10; i++)  
11:            myQueue.Enqueue(i);  
12:  
13:  
14:         // Empty the queue  
15:         for (int i = 0; i < 10; i++)
```

```

16:         Console.WriteLine("{0}",
17:                         myQueue.Dequeue());
18:
19:         // diền đầy dữ liệu vào hàng đợi một lần nữa
20:         for (int i = 0; i < 10; i++)
21:             myQueue.Enqueue(i);
22:
23:         foreach (int i in myQueue)
24:             Console.WriteLine("{0}", i);
25:     }
26: }
```

1.3. Bảng băm hashtable

Bảng băm Hashable hữu dụng khi bạn cần lưu trữ đối tượng và truy xuất chúng theo khóa. Thời gian tìm kiếm cho hashtable là rất nhanh. Mỗi khóa được dùng để phát sinh 1 giá trị băm (hash value), hash value này hoạt động như một chỉ mục cho đối tượng trong tập hợp. Khi có yêu cầu truy xuất đến 1 phần tử xác định, thì giá trị băm của khóa được tính toán và phần tử truy xuất được định vị. Nói chung, bảng băm hashtable không tiết kiệm bộ nhớ để đổi lấy tốc độ, và còn tùy thuộc vào cách sử dụng. ví dụ 2.2-3 sẽ minh họa cách sử dụng bảng băm

ví dụ 2.2-3 : Hashtable

```

1:  using System;
2:  using System.Collections;
3:
4:  struct Person {
5:      public Person ( string f, string l ) {
6:          FName = f; LName = l; }
7:      public string LName;
8:      public string FName;
9:  }
10: public class HashtableTest {
11:
12:     public static void Main() {
13:
14:         Hashtable People = new Hashtable();
15:
16:         // Thêm vào đối tượng Person
17:         People.Add( "Smith", new Person("Jim",
18:                                         "Smith"));
19:         People.Add( "Jones", new Person("Dawn",
20:                                         "Jones"));
```



```
19:         People.Add( "Powell", new Person("Bob",
                                         "Powell"));
20:
21:         // Tìm kiếm Jim Smith
22:         Person p = (Person)People["Smith"];
23:         Console.WriteLine("{0}", p.FName,
                           p.LName);
24:     }
25: }
```

Một chú ý quan trọng là bảng băm hashtable sẽ không cho phép nhiều hơn 1 mục dùng chung giá trị khóa. Nếu làm như vậy bảng băm hashtable sẽ ném ra 1 ngoại lệ ArgumentException.

2. DANH SÁCH LIÊN KẾT (LINK LIST)

Để nhấn mạnh tầm quan trọng của những giao tiếp và tính mở rộng của kiến trúc .NET, một tập hợp danh sách liên kết được đưa ra cài đặt giao tiếp IEnumerable và cung cấp đối tượng tiếp cận IEnumerator. Bạn có thể sử dụng câu lệnh foreach để duyệt nội dung của danh sách liên kết.

Danh sách liên kết cài đặt trong ví dụ 2.2-4 vẫn chưa đầy đủ. Như là một bài tập, bạn nên mở rộng chức năng của danh sách liên kết. Ví dụ mở rộng này gồm khả năng cung cấp 1 phương thức để thêm những mục tại nút cuối của danh sách hoặc tại bất kỳ vị trí nào trong danh sách. Ví dụ này cũng có thể xác định 1 đối tượng cụ thể bên trong danh sách. (lưu ý : giao tiếp IComparable sẽ được sử dụng vào khi cần thiết).

Ví dụ 2.2-4 : Danh sách liên kết.

```
1://File    : part02_35.cs
2://Author  : Richard L. Weeks
3://Purpose : Implement a linked list that supports the
              IEnumerable interface
4:
5:
6:
7:using System;
8:using System.Collections;
9:
```

```
10:  
11:  
12: public class _node {  
13:  
14:     public _node next;  
15:     public _node prev;  
16:     public object value;  
17:  
18:     public _node() { next = prev = null; value = null; }  
19:     public _node(object o) {  
20:         value = o;  
21:         next = prev = null;  
22:     }  
23:  
24:     public _node(_node n) {  
25:         next = n.next;  
26:         prev = n.prev;  
27:         value = n.value;  
28:     }  
29: }  
30:  
31:  
32: // Linked list enumerator  
33: public class LinkedListEnumerator : IEnumerator {  
34:  
35:     private _node m_current = null;  
36:     private _node m_begin = null;  
37:     private m_last = false;  
38:  
39:     public LinkedListEnumerator(_node n) { m_current  
        = m_begin = n; }  
40:
```

```
41:  
42: //Implement the IEnumator interface  
43:     public object Current {  
44:         get { return m_current.value; }  
45:         set { m_current.value = value; }  
46:     }  
47:  
48:     public bool MoveNext() {  
49:         if(m_last)  
50:             return false;  
51:         else {  
52:             m_curretn = m_current.next;  
53:             m_last = m_current == m_begin ? true : false;  
54:             return true;  
55:         }  
56:     }  
57:  
58:     public void Reset() {  
59:         m_current = m_begin ;  
60:         m_last = false;  
61:     }  
62: }  
63:  
64: public class LinkedList : IEnumerable {  
65:  
66:     private _node m_root = null;  
67:  
68:  
69:     //Implement the IEnumerable interface method  
GetEnumerator()  
70:     public IEnumerator Getenumerator() {
```

```
71:             return (IEnumerator) new
    LinkedListEnumerator(m_root.prev);
72:         }
73:
74:
75:     //Implement some basic methods
76:     public void AddHead( object o) {
77:         _node newNode = new _node(o);
78:         if(m_root == null) {
79:             m_root = newNode;
80:             m_root.next = m_root;
81:             m_root.prev = m_root;
82:         }else {
83:             newNode.next = m_root;
84:             newNode.prev = m_root.prev;
85:             m_root.prev.next = newNode;
86:             m_root.prev = newNode;
87:             m_root = newNode;
88:         }
89:
90:     }
91:
92:
93:
94:
95:     public class LinkedListTest {
96:
97:         public static void Main() {
98:
99:
100:            LinkedList l = new LinkedList();
101:
```

```
102:         for(int i = 0; i < 10;i++)
103:             l.AddHead(i);
104:
105:         foreach (int i in l)
106:             Console.WriteLine(i);
107:     }
108: }
```

Ví dụ danh sách liên kết giải thích những khái niệm cơ bản của việc cài đặt những giao tiếp và C# có thể sử dụng những giao tiếp này như thế nào. Bởi vì lớp LinkedList thực thi giao tiếp IEnumerable và cung cấp thực thể IEnumerator, câu lệnh foreach có thể được sử dụng để lập thông qua những nội dung của danh sách liên kết.

3. ATTRIBUTES (ĐẶC TÍNH)

Attribute là một khái niệm mới của .NET. Attribute là những thẻ (tag) được công bố có thể được áp dụng cho những thành viên và những phương thức khác nhau. Thông tin đó có thể được xem xét sử dụng bởi System.Reflection API. Chi tiết và cách sử dụng Reflection nằm ngoài phạm vi thảo luận của giáo trình này tuy nhiên tầm quan trọng của những Attribute cũng cần phải được xem qua. Ví dụ 2.2-5 giải thích sử dụng đặc tính Conditional được cung cấp bởi những lớp của kiến trúc .NET.

Ví dụ 2.2-5 : Conditional Attribute

```
1: //File      : part02_36.cs
2: //Author : Richard L. Weeks
3: //Purpose : The conditional attribute
4: //
5: //Compile instructions :
6: //csc /define:DEBUG part02_36.cs
7: //csc part02_36.cs
8:
9:     using System;
10:    using System.Diagnostics;//the ConditionalAttribut
   lives here
11:
```

```

12: public class Foo {
13:     ...
14:     [Conditional("DEBUG")]
15:     public void OnlyWhereDebugIsDefined() {
16:         Console.WriteLine("DEBUG is defined");
17:     }
18: }
19:
20: public class AttributeTest {
21:     public static void Main() {
22:         Foo f = new Foo();
23:         f.OnlyWhereDebugIsDefined();
24:     }
25: }

```

Cách tốt nhất để hiểu nhanh về những attribute là nhìn vào ví dụ ở trên. Ví dụ 2.2.5 sử dụng Conditional attribute. Như tên của nó gợi ý, Conditional attribute được dùng trong quá trình biên dịch có điều kiện. Nhìn vào dòng 14. Conditional attribute được áp dụng cho phương thức OnlyWhenDebugIsDefined. Phương thức OnlyWhenDebugIsDefined chỉ hợp lệ khi DEBUG được định nghĩa trong quá trình biên dịch. Chú ý dòng 23 tạo ra 1 lời gọi phương thức OnlyWhenDebugIsDefined và không có bất kỳ điều kiện nào. Đây là một lợi ích được thêm vào Conditional attribute. Không cần bọc mã chương trình ở trên trong điều kiện #ifdef types của những câu lệnh tiền xử lý.

Một điểm đáng chú ý, những phương thức được đánh dấu với Conditional attribute vẫn được tính khi biên dịch mã máy. Tuy nhiên, tất cả những lời gọi tới phương thức đều bị bỏ đi. Trong bản dịch cuối cùng, tất cả mã điều kiện nên được lấy ra từ quá trình biên dịch bằng cách sử dụng những câu lệnh tiền xử lý #if.

.NET sử dụng những attribute không chỉ cho việc biên dịch có điều kiện, mà còn cho WebServices, XML và Window Services. Những đặc tính Attribute không chỉ giới hạn trong C#, chúng cũng có thể được tìm thấy trong ATL Server, thế hệ mới của ATL từ Microsoft.

4. XML SERIALIZATION

Đối tượng tuần tự hóa (serialization) là một chủ đề chính trong thế giới hướng đối tượng (OO – Object Oriented) trong nhiều năm. Khả năng nhất quán 1 đối tượng và những đối tượng kết hợp luôn được quan tâm và thường được yêu cầu về việc phát triển của mã chương trình đặc biệt để xử lý những trường hợp cụ thể.



Khả năng cung cấp một kiến trúc tổng quát cho việc tuân tự hóa (serialization) thường yêu cầu những đối tượng đóng vai trò tích cực trong tính nhất quán của chúng.

XML đã trở nên 1 phần quan trọng của việc kinh doanh ngày nay. Với chuẩn giao tiếp mở B2B, sự cần thiết của XML đóng vai trò quan trọng trong việc phát triển phần mềm hiện đại. Bởi vì XML định nghĩa không chỉ dữ liệu mà còn metadata.

5. C# HỖ TRỢ XML SERIALIZATION

C# sử dụng những đặc tính Attribute để hỗ trợ XML Serialization. Các đặc tính Attribute tồn tại cho việc định nghĩa những phần tử gốc, những đặc tính Attribute con, và những phần tử khác bên trong tài liệu XML. Dĩ nhiên, có một số những yêu cầu cơ bản để hỗ trợ tuân tự hóa. Các thành viên của lớp hoặc cấu trúc struct muốn tuân tự hóa đều phải là public hoặc có một thuộc tính truy cập (accessor). Nếu một thành viên muốn khôi phục trở về trạng thái trước khi tuân tự hóa (deserialized), nó phải cài đặt thuộc tính setter tương ứng. Cũng có sự hạn chế về tuân tự hóa (serialization) của những lớp collection. Hiện nay, chỉ có kiểu mảng (array) có thể được tuân tự hóa và chuyển về trạng thái phi tuân tự (deserialized). Dĩ nhiên, điều này yêu cầu người lập trình cung cấp 1 thuộc tính cho phép biến đổi lớp Collection đang được sử dụng thành 1 mảng, và từ một mảng thành Collection thích hợp.

Để định nghĩa những phần tử gốc, chúng ta có thể sử dụng XmlRootAttribute áp dụng cho lớp hoặc cấu trúc struct. Phương thức khởi tạo (constructor) của XmlRootAttribute nhận một chuỗi tên của phần tử gốc bên trong tài liệu XML. Chỉ có phần tử gốc (phần tử trên đỉnh) yêu cầu XmlRoot attribute. Tất cả những phần tử con chỉ cần định nghĩa những thành viên được nhất quán.

```
[XmlRoot("purchase-order")]
public class PurchaseOrder {
    // PurchaseOrder members
}
```

Lớp PurchaseOrder đại diện phần tử gốc cho tài liệu XML. Mỗi thành viên bên trong lớp PurchaseOrder sẽ cần được qui cho là XmlElement attribute hoặc XmlAttribute, còn phụ thuộc vào chính phần tử được nhất quán như thế nào.

Ví dụ 2.2-6 : Sử dụng XML Attributes cho tuân tự hóa.

```
1: //File : PurchaseOrder.cs
2: //Author: Richard L. Weeks
3: //Purpose : Demonstrate the basics of XML serialization
4: //
```



```
5: //Compilation instructions
6: //csc PurchaseOrder.cs /r:System.dll,System.Xml.dll
7:
8:
9: using System;
10: using System.Xml;
11: using System.Xml.Serialization;
12: using System.Collections;
13: using System.IO;
14:
15:
16: ///////////////
17: Define the Purchase Order
18: [XmlRoot("purchase-order")]
19: public class PurchaseOrder {
20:
21:     //private
22:     private ArrayList m_Items;
23:
24:     public PurchaseOrder() {
25:         m_Items = new ArrayList();
26:     }
27:
28:     //Properties
29:     [XmlElement("item")]
30:     public Item[] Items {
31:         get {
32:             Item[] items = new Item[m_Items.Count];
33:             m_Items.CopyTo(items);
34:             return items;
35:         }
36:         set {
37:             if(value == null) return;
38:             Item[] items = new (Item[])value;
```

```
39:         m_Items.Clear();
40:         foreach(Item i in items)
41:             m_Items.Add(i);
42:     }
43: }
44:
45: //methods
46: public void AddItem(Item item) {
47:     m_Items.Add(item);
48: }
49:
50: //indexer
51: public Item this[string sku] {
52:     get {
53:         //locate the item by sku
54:         foreach(Item i in m_Items)
55:             if(i.sku == sku)
56:                 return i;
57:         throw(new Exception("Item not found"));
58:     }
59: }
60:
61: public void DisplayItems() {
62:     foreach(Item i in m_items)
63:         Console.WriteLine(i);
64: }
65:
66: }
67:
68: ///////////////
69: //Define an item entity
70: public class Item {
71:
72:     //item data
```

```
73:     [XmlAttribute("sku")] public string sku;
74:     [XmlAttribute("desc")] public string desc;
75:     [XmlAttribute("price")] public double price;
76:     [XmlAttribute("qty")] public int qty;
77:
78:
79://Default constructor required for XML serialization
80:     public Item() {}
81:
82:     public Item(string sku, string Desc, double Price, int
83: Qty) {
84:         sku = Sku;
85:         desc = Desc;
86:         price = Price;
87:         qty = Qty;
88:     }
89:
90:     public override string ToString() {
91:         object[] o = new object[] {sku,desc,price,qty};
92:         return string.Format("{0,-5} {1,-10}
93: ${2,5:#,###.00} {3,3}",o);
94:     }
95:
96:
97:
98:
99:/// 
100://Test the XML Serialization and Deserialization
101://
102:public class POExample {
103:
104:
```

```
105:     public static void Main() {
106:
107:     PurchaseOrder po = new PurchaseOrder();
108:
109:     po.AddItem( new Item("123", "pencial", 15, 100));
110:     po.AddItem( new Item("321", "copy
111:     paper", 7.50, 25));
112:
113:     po.DisplayItems();
114:     Console.WriteLine("Serialization in progress");
115: //Serialize the Current Purchase Order
116:     XmlSerializer s = new
117:         XmlSerializer(typeof(PurchaseOrder));
118:     TextWriter w = new StreamWriter("po.xml");
119:     s.Serialize(w,po);
120:     w.Close();
121:     Console.WriteLine("Serialization complete \n\n");
122:     Console.WriteLine("Deserialization in progress");
123: //Deserialize to a new PO
124:     PurchaseOrder po2;// = new PurchaseOrder();
125:     TextReader r = new StreamReader("po.xml");
126:     po2 = (PurchaseOrder)s.Deserialize(r);
127:     r.Close();
128:     Console.WriteLine("Deserialization complete");
129:     po2.DisplayItems();
130: }
131:
132: }
```

Ví dụ 2.2-6 minh họa cách C# hỗ trợ XML Serialization. Một lần nữa, lớp PurchaseOrder đại diện cho phần tử trên đỉnh trong tài liệu XML. Lớp PurchaseOrder chứa đựng một ArrayList, ArrayList này chứa những phần tử được đưa vào. Bởi vì hiện nay .NET chưa hỗ trợ việc tuẩn tự hóa những lớp chứa đựng (container), việc cung cấp 1 thuộc tính cho phép chuyển đổi ArrayList thành một

mảng và ngược lại là rất cần thiết. Dòng 27 định nghĩa 1 phân tử XElement cho mục hàng. Thuộc tính này được cài đặt như phép biến đổi ArrayList thành mảng kiểu Item[].

Để phi tuần tự (deserialized) 1 đối tượng, lớp hoặc cấu trúc struct bạn cần cung cấp 1 phương thức khởi tạo (constructor) mặc định. Trong trường hợp của lớp Item (bởi vì chúng ta đã định nghĩa 1 đối số cho phương thức khởi tạo cơ sở) cần một phương thức khởi tạo mặc định mà không có đối số, vì vậy đối tượng được tạo 1 cách năng động tại thời điểm chạy. Việc định nghĩa chồng (override) phương thức ToString của lớp Item không đóng vai trò trong XML Serialization. Sự tồn tại của nó chỉ cho phép kết xuất dòng dữ liệu (stream) ra màn hình (console).

Để tuần tự (serialize) PurchaseOrder, ta cần tạo một thể hiện của nó dựa vào các mục item vào đó. Tiếp theo là việc tạo 1 thể hiện của đối tượng XmlSerializer. Đối tượng XmlSerializer hỗ trợ một vài những phương thức khởi tạo tính đến thông tin phần tử phụ – không gian tên mặc định và tên phần tử gốc. Ở ví dụ này, chỉ có kiểu đối tượng được truyền vào.

Đối với những người quen thuộc COM Serialization trước đây, thì khái niệm dòng hay luồng (stream) không phải là khái niệm xa lạ nó đơn thuần chỉ là một luồng dữ liệu cho phép kết nối tới cơ sở dữ liệu, tập tin hệ thống (File system), hoặc thậm chí là 1 đầu cuối kết nối mạng (Network socket).

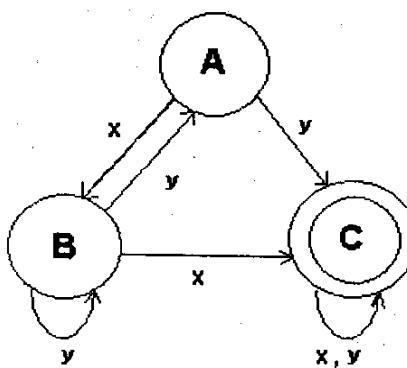
Khi PurchaseOrder được thực thi, đoạn mã XML sau là kết quả của việc tuần tự hóa.

```
<?xml version="1.0" encoding="utf-8"?>
<purchase-order
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <item sku="123" desc="pencial" price="0.15" qty="100" />
  <item sku="321" desc="copy paper" price="7.5" qty="25" />
  <item sku="111" desc="white out" price="1.35" qty="10" />
</purchase-order>
```

Tiến trình của việc khôi phục sau khi tuần tự hóa (deserialization) cơ bản thực hiện theo cùng cách. XmlSerializer sử dụng Reflection API để xây dựng những đối tượng cần thiết và gán các Attribute cùng những phần tử cho mục hàng item.

Bây giờ để làm một điều gì đó thú vị, thay vì tạo ra chuẩn tuần tự hóa tẻ nhạt của các đối tượng, chúng ta sẽ tạo ra một chương trình Finite State Machine sử dụng XML. Dựa trên trạng thái và những sản phẩm được định nghĩa cho cổ máy, chúng ta đưa vào một chuỗi ký hiệu dùng cho sự xử lý. Hình 2.2-1 mô tả

trạng thái của chương trình FSM nhỏ, FSM sẽ được hiện diện trong mã chương trình sử dụng XML serialization được cung cấp bởi .NET.



Hình 2.2-1 Ba trạng thái của FSM.

XML mô tả trạng thái máy trong hình 2.2-1 như sau :

```

<?xml version="1.0"?>
<fsm
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema" state_count
  = "3">

  <state name ="a" is_start="true" is_final="false"
  production_count="2">
    <production token="x" state="b"/>
    <production token="y" state="c"/>
  </state>

  <state name ="b" is_start="false" is_final="false"
  production_count="2">
    <production token="x" state="c"/>
    <production token="y" state="b"/>
  </state>

  <state name ="c" is_start="false" is_final="true"
  production_count="2">
    <production token="x" state="c"/>
    <production token="y" state="c"/>
  </state>

```

```
</state>  
</fsm>
```

Phần tử gốc fsm định nghĩa Finite State Machine. fsm chứa đựng những trạng thái khác nhau, và mỗi trạng thái chứa đựng những sản phẩm mà nó hỗ trợ.

Đoạn mã chương trình xử lý của State Machine được viết rất cô đọng nhằm giải thích khả năng của XML Serialization. Khi đoạn mã đơn giản này được cài đặt, nó sẽ nhắc nhở người sử dụng cho vào một chuỗi ký hiệu để xử lý. Ví dụ, chuỗi ký hiệu xyxyxxy được xem là một chuỗi hợp lệ, trong khi chuỗi xyxy thì không hợp lệ. Chỉ những chuỗi dấu hiệu kết thúc ở trạng thái cuối cùng mới được xem xét hợp lệ. Ví dụ 2.2-7 sẽ cài đặt FSM mô tả trong lưu đồ 2.2-1.

Ví dụ 2.2-7 : Cài đặt FSM với XML Serialization.

```
1: //File : part02_38.cs  
2: //Author : Richard L. Weeks  
3: //////////////////////////////////////////////////////////////////  
4: //Purpose : Make use of XML Serialization to implement  
5: // a finite state machine  
6: //  
7: //  
8: //  
9:  
10: using System;  
11: using System.Xml;  
12: using System.Xml.Serialization;  
13: using System.IO;  
14:  
15:  
16: //////////////////////////////////////////////////////////////////  
17: //The FSM class  
18: [XmlRoot("fsm")]  
19: public class FSM {  
20:  
21:     //data members  
22:     private int m_StateCount;
```

```
23:  
24:     [XmlAttribute("state_count")]  
25:     public int StateCount {  
26:         get { return m_StateCount; }  
27:         set {  
28:             m_StateCount = value;  
29:             State = new State[m_StateCount];  
30:         }  
31:     }  
32:  
33:     [XmlElement("state")]  
34:     public State[] States = null;  
35:  
36:  
37:     public bool ProcessString(string s) {  
38:  
39:         State CurrentState = GetStartState();  
40:         Console.WriteLine("Start state is {0}",  
        CurrentState.Name);  
41:         //Process the token string  
42:         for(int i = 0;i<s.Length;i++) {  
43:             string next_state =  
CurrentState.ProcessToken(string.Format("{0}",s[i]));  
44:             if(next_state != "")  
45:                 CurrentState = GetState(next_state);  
46:             else {  
47:                 Console.WriteLine(  
"No production from {0} with token {1}",  
CurrentState.Name,s[i]);  
48:                 return false;  
49:             }  
50:             Console.WriteLine(  
"Current State => {0}",CurrentState.Name);  
51:         }
```

```
52:  
53:     return CurrentState.IsFinal;  
54: }  
55:  
56: private State GetState (string state_name) {  
57:     //Locate the state name  
58:     for(int i=0;i<States.Length;i++)  
59:         if(States[i].Name == state_name)  
60:             return State[i];  
61:     return null;  
62: }  
63:  
64: private State GetStartState() {  
65:     for(int i = 0;i<State.Length;i++)  
66:         if(State[i].IsStart)  
67:             return State[i];  
68:     return null;  
69: }  
70: }  
71:  
72: //////////////////////////////////////////////////////////////////  
73: //State class  
74: [XmlRoot("state")]  
75: public class State {  
76:  
77:     private string m_Name;  
78:     private bool m_IsStart;  
79:     private bool m_IsFinal;  
80:     private int m_ProductionCount;  
81:  
82:  
83:     [XmlAttribute("name")]  
84:     public string Name {  
85:         get { return m_Name; }
```

```
86:         set { m_Name = value; }
87:     }
88:
89:     [XmlAttribute("is_start")]
90:     public bool IsStart {
91:         get { return m_IsStart; }
92:         set { m_IsStart = value; }
93:     }
94:
95:     [XmlAttribute("is_final")]
96:     public bool IsFinal {
97:         get { return m_IsFinal; }
98:         set { m_IsFinal = value; }
99:     }
100:
101:    [XmlAttribute("production_count")]
102:    public int ProductionCount {
103:        get { return m_ProductionCount; }
104:        set {
105:            Productions = new Production[value];
106:            m_ProductionCount = value;
107:        }
108:    }
109:
110:   [XmlElement("production")]
111:   public Production[] Productions = null;
112:
113:
114:   public string ProcessToken(string token) {
115:       //loop through the productions and
return the name of the next state
116:       for(int i=0;
i < Productions.Length; i++) {
```



```
117:         Console.WriteLine("State {0} is evaluating
      token {1}",
m_Name, token);
118:         Console.WriteLine("Testing Production {0} :
      {1}",
Productions[i].token, Productions[i].state);
119:         if(Productions[i].token == token)
120:             return Productions[i].state;
121:         }
122:         return "";
123:     }
124:
125: }
126:
127: ///////////////////////////////////////////////////
128: // Production struct
129: [XmlRoot("production")]
130: public struct Production {
131:
132:     [XmlAttribute("token")]
133:     public string token;
134:
135:     [XmlAttribute("state")]
136:     public string state;
137: }
138:
139:
140:
141:
142: public class FiniteStateMachine {
143:
144:     public static void Main() {
145:
146:         //Deserialize the FSM from the xml file
```

```
147:     XmlSerializer s = new
148:         XmlSerializer(typeof(FSM));
149:     FSM fsm = (FSM)s.Deserialize(tr);
150:     tr.Close();
151:
152:     //Get the token string to process
153:     Console.WriteLine("Enter token string to
process: ");
154:     string tokens = Console.ReadLine();
155:     string result = fsm.ProcessString(token) ?
"valid" : "invalid";
156:
157:     Console.WriteLine("The token string {0} is {1}",
tokens,result);
158:
159: }
160: }
```

6. KẾT CHƯƠNG

Trong phần này, chúng ta đã tìm hiểu những khía cạnh khác nhau của C# và kiến trúc .NET. Microsoft đã đặt những tài nguyên đáng kể và nỗ lực vào kiến trúc .NET. Còn rất nhiều thứ của .NET mà chúng ta cần khai phá, trong chương sau bạn sẽ bắt đầu cảm thấy .NET thực tế hơn với phong cách lập trình giao diện Windows Forms, và có lẽ đây cũng là chương bạn cảm thấy thú vị và hữu ích nhất.

Phân III

WINDOWS FORMS

Trong phân này:

- ◆ Windows Forms
- ◆ Xử lý giao diện đồ họa (GUI)
- ◆ Ràng buộc dữ liệu
- ◆ Xây dựng ứng dụng Windows Forms (ScRibble .NET)
- ◆ GDI+: Giao diện đồ họa của .NET
- ◆ Thực hành ứng dụng Windows Forms

Chương 3.1

WINDOWS FORMS

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Chương trình ứng dụng Windows Forms Hello*
- ✓ *Tạo và sử dụng bộ quản lý sự kiện*
- ✓ *Định nghĩa Border Style cho form*
- ✓ *Thêm menu vào cửa sổ form*
- ✓ *Tạo menu tắt shortcut*
- ✓ *Xử lý sự kiện của mục chọn menu*

1. GIỚI THIỆU VỀ WINDOWS FORMS CỦA KIẾN TRÚC .NET

Windows Forms là sự thay thế của .NET cho MFC trong Visual C++. Không như thư viện MFC dùng đóng gói cho tập các hàm Win32 API. Windows Forms hoàn toàn là các lớp hướng đối tượng, mang tính kế thừa dành cho các nhà phát triển ứng dụng trong môi trường .NET.

Mặc dù mang thuật ngữ “Forms” nhưng việc thiết kế các thành phần giao diện trên Form lại không dựa vào các file tài nguyên resource như khi thiết kế các biểu mẫu Dialog trong MFC theo cách truyền thống của Windows. Mỗi thành phần giao diện còn gọi là component đặt trên Form của Windows Forms là một thể hiện (instance) cụ thể của một lớp nào đó. Component được định vị trên Form và thay đổi giao diện thông qua các phương thức (method), thuộc tính (property) của lớp đối tượng. Các công cụ phát triển trực quan sẽ cho phép kéo thả (drag-drop) những thành phần component lên Form và quản lý chắc chắn việc khởi tạo mã nguồn cũng như tương tác giữa các thành phần component này với nhau.

Windows Forms cũng dùng các file tài nguyên (resource file) cho các tác vụ đặt một ảnh lên Form, lưu các dữ liệu text cục bộ tương tự những file resource của Windows có từ những năm 1980s, tuy nhiên khuôn dạng của các file này đã thay đổi, chúng được viết theo ngôn ngữ và cấu trúc XML chuẩn. Chúng ta sẽ nghiên cứu chi tiết về các file tài nguyên này trong chương sau “Xây dựng ứng dụng với Windows Forms (Scribble .NET)”

Để xây dựng và kết nối các component trên một Form bạn có thể sử dụng các bảng công cụ trực quan trong những môi trường phát triển như VS.NET, chẳng hạn C#, VB.NET hay C++ hoặc một ngôn ngữ .NET nào đó. Tuy nhiên, nhằm giúp bạn hiểu rõ hơn về Windows Form, chúng tôi hạn chế sử dụng những công cụ phát triển trực quan này. Ở đây ta chỉ sử dụng trình soạn thảo đơn giản notepad viết mã và xây dựng Form cho ứng dụng Windows Forms hoàn toàn dựa trên phương thức, thuộc tính của các lớp. Một khi đã hiểu rõ cách Windows Forms liên kết mã,

bạn có thể chuyển sang sử dụng các công cụ trực quan mà không gặp bất kỳ khó khăn nào.

Tương tác của người dùng với các thành phần component trong Windows Form thực hiện thông qua sự kiện (event). Mỗi thành phần component cung cấp một tập các sự kiện nguồn (event source) như sự kiện di chuyển chuột, nút nhấn được kích hoạt, con trỏ thay đổi vị trí ... Nhiệm vụ của lập trình viên là viết các bộ xử lý (handler) cho những sự kiện này. Các hàm xử lý sự kiện được gọi thông qua mô hình chuyển giao (delegate) mà không cần dựa vào bất kỳ thông điệp nào trong hệ thống. Bạn không cần phải ánh xạ (mapping) các thông điệp sự kiện như trong MFC của C++.

2. WINDOWS FORMS VÀ ỨNG DỤNG HELLO WORLD

Ví dụ 3.1-1 dưới đây là một ứng dụng Windows Forms đơn giản hiển thị vè chuỗi Hello World ra giữa cửa sổ Form.

Ví dụ 3.1-1 HWF.cs

```
// HWF.cs
namespace HelloWindowsFormsNamespace {

    using System;
    using System.Drawing;
    using System.ComponentModel;
    using System.Windows.Forms;

    public class HelloWindowsForms :
        System.Windows.Forms.Form
    {
        //Label 1 is the component that displays the text
        //messace "Hello Windows Forms!"
        private System.Windows.Forms.Label label1;

        //The constructor is where all initialization happens.
        //For forms created with the designer there is
        //an InitializeComponent() method.
        public HelloWindowsForms()
        {

            this.label1 = new System.Windows.Forms.Label();

            label1.Location = new System.Drawing.Point(8, 8);
            label1.Text = "Hello Windows Forms!";
            label1.Size = new System.Drawing.Size(408, 48);
            label1.Font = new System.Drawing.Font(
                "Microsoft Sans Serif", 24f);
        }
    }
}
```

```

label1.TabIndex = 0;
label1.TextAlign = ContentAlignment.MiddleCenter;

this.Text = "Hello World";
this.MaximizeBox = false;
this.AutoScaleBaseSize =
    new System.Drawing.Size(5, 13);
this.FormBorderStyle =
    FormBorderStyle.FixedSingle;
this.MinimizeBox = false;
this.ClientSize = new System.Drawing.Size(426, 55);
this.Controls.Add(label1);
}

// This main function instantiates a new form and runs
it.
public static void Main(string[] args)
{
    Application.Run(new HelloWindowsForms());
}
}

} // end of namespace

```

Như bạn thấy, lớp HelloWindowsForms chỉ chứa một thành phần component đơn giản đó là nhãn (lớp Label) mang tên label1 cùng với hai phương thức : phương thức tĩnh Main dùng để tạo và chạy thể hiện của lớp HelloWindowsForms. Phương thức khởi động HelloWindowsForms() dùng tạo nhãn và gán các thuộc tính ban đầu cho các thành phần component trên Form.

Ví dụ trên cho thấy những chức năng và sự phức tạp tối thiểu của một chương trình Windows Forms. Các chương trình được thiết kế và quản lý bởi VS.NET hay WinDes.exe sẽ có thêm một vài tính năng và những thành viên dữ liệu (data member) khác, tuy nhiên chúng chỉ dùng để các môi trường phát triển ứng dụng trực quan (RAD) dễ dàng theo dõi và quản lý việc thiết kế Form mà thôi.

Để tạo file thực thi của chương trình, chúng ta chuẩn bị một file batch (.BAT) liên kết những thư viện DLL cần thiết cho việc biên dịch mã nguồn của Windows Forms. Sử dụng file batch có ưu điểm là bạn không cần dùng đến môi trường phát triển Visual Studio nặng nề công kênh và thường che dấu hầu những kỹ thuật biên dịch quan trọng. Nếu bạn muốn nghiên cứu sâu về C# và Windows Forms, cách tốt nhất là hãy bắt đầu từ những tham số biên dịch dòng lệnh, tuy đơn giản không thân thiện nhưng chúng tỏ ra cực kỳ hiệu quả và mạnh mẽ. Ví dụ 3.1-2 là nội dung file build.bat dùng biên dịch mã nguồn các file *.cs sử dụng thư viện Windows Forms.

Ví dụ 3.1-2 build.bat

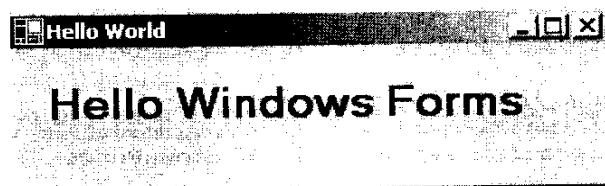
```
csc /t:winexe /r:Microsoft.win.Interop.dll, system.dll
```

```
system.configuration.dll, system.data.dll,  
system.diagnostics.dll, system.drawing.dll  
system.winforms.dll /out: %1.exe %1.cs %2 %3 %4 %5 %6 %7 %8  
%9
```

Lưu ý: khi gõ lệnh cho file build.bat tất cả các lệnh phải ở trên một dòng và không được có khoảng trắng giữa tên thư viện trong chỉ dẫn /r:. Bạn gọi file build.bat từ dòng lệnh để biên dịch các file .cs như sau

Build hws.cs

Kết quả, trình biên dịch sẽ tạo ra file hwf.exe. Chạy chương trình hwf.exe bạn sẽ được cửa sổ Forms hiển thị như hình 3.1-1



Hình 3.1-1 Cửa sổ chương trình HelloWindowsForms.

3. TẠO VÀ SỬ DỤNG CÁC BỘ XỬ LÝ SỰ KIỆN (EVENT HANDLER)

Nếu bạn đã quen với việc phát triển các ứng dụng Windows trên C, bạn sẽ không cảm thấy lạ đối với khái niệm xử lý thông điệp trong hàm WinProc với câu lệnh switch truyền thống lựa chọn các thông điệp. Nếu đã từng sử dụng MFC, bạn cũng đã biết rằng thông điệp được truyền đến các đối tượng thông qua bảng ánh xạ thông điệp (message map). Trong .NET thông điệp được gửi đi theo hình thức chuyển giao (delegate). Ứng dụng của bạn trước hết định nghĩa một bộ xử lý sự kiện (event handler). Bộ xử lý này sẽ được hệ thống triệu gọi bất kỳ khi nào thông điệp thích hợp mà bộ xử lý này đăng ký phát sinh. Ví dụ như khi bạn kích chuột vào một nút nhấn, sự kiện này thường được chuyển giao cho bộ xử lý **EventHandler** đảm trách. Những kiểu sự kiện khác, như sự kiện chuột hoặc sự kiện do bộ định thời (timer) phát sinh sẽ được chuyển giao cho những Handler tương ứng của nó xử lý. Nói chung, bạn không cần phải quan tâm đến thông điệp, cái mà chương trình ứng dụng của bạn cần quan tâm là những bộ xử lý sự kiện.

Thêm một nút nhấn (button) vào cửa sổ Form của một ứng dụng Windows Forms rất đơn giản. Bạn có thể thêm một thành phần nút nhấn vào lớp chương trình chính, khởi tạo các giá trị ban đầu cho nó trong phương thức khởi dựng (constructor) của lớp, sau cùng là đặt nó vào tập hợp (collection) của những thành phần điều khiển (controls) trong ứng dụng.

Khi những công việc trên hoàn tất, bạn có thể bắt đầu xây dựng các bộ xử lý kết nối đến sự kiện. Bộ xử lý này sẽ bẫy trạng thái kích chuột của nút nhấn.



Các sự kiện gửi từ thông qua hệ thống sẽ được móc nối vào một chuỗi các handler đã đăng ký trước đó. Mỗi bộ xử lý handler phải có một dấu hiệu nhận dạng để triệu hồi như sau:

```
void EventMethodName(Object sender, EventArgs e);
```

Bộ xử lý sự kiện được đặt vào tập nguồn các sự kiện của thành phần điều khiển bằng toán tử `+=` và loại bỏ bằng toán tử `=`, vì vậy các bộ xử lý sự kiện có thể thay đổi động trong quá trình thực thi của chương trình. Cùng một sự kiện bạn có thể dùng một hoặc nhiều bộ xử lý sự kiện handler khác nhau. Tính năng này được gọi là multicast.

Chương trình HWFButton.cs dưới đây sẽ cho thấy cách tạo vài cài đặt bộ xử lý sự kiện cho nút nhấn.

Ví dụ 3.1-3 HWFButton.cs

```
// HWFButton.cs
namespace HelloWindowsFormsNamespace {

    using System;
    using System.Drawing;
    using System.ComponentModel;
    using System.Windows.Forms;
    using System.Runtime.InteropServices;

    public class HelloWindowsForms :
        System.Windows.Forms.Form
    {

        //Label 1 is the component that displays the text message
        "Hello Windows Forms!"
        private System.Windows.Forms.Label label1;

        //Adding a button member allows us to place a button on
        the panel.
        private System.Windows.Forms.Button button1;

        //The constructor is where all initialization happens.
        //For forms created with the designer there is an
        InitializeComponent() method.
        public HelloWindowsForms()
        {

            this.label1 = new System.Windows.Forms.Label();
            label1.Location = new System.Drawing.Point(8, 8);
            label1.Text = "Hello Windows Forms!";
        }
}
```



200

Edited by Foxit Reader

PHẦN III : Windows Forms

Copyright(C) by Foxit Software Company, 2005-2008

For Evaluation Only

```
label1.Size = new System.Drawing.Size(408, 48);
label1.Font = new System.Drawing.Font("Microsoft
Sans Serif", 24f);
label1.TabIndex = 0;
label1.TextAlign = ContentAlignment.MiddleCenter;

this.button1=new System.Windows.Forms.Button();

button1.Location=new System.Drawing.Point(8,58);
button1.Size = new System.Drawing.Size(408,25);
button1.Text = "Click Me!";
button1.TabIndex = 1;
button1.Click += new EventHandler(OnButton1Clicked);

    this.Text = "Hello World";
    this.MaximizeBox = false;
    this.AutoScaleBaseSize = new System.Drawing.Size(5,
13);
    this.FormBorderStyle = FormBorderStyle.FixedSingle;
    this.MinimizeBox = false;
    this.ClientSize = new System.Drawing.Size(426, 85);

    this.Controls.Add(label1);
this.Controls.Add(button1);
}

void OnButton1Clicked(Object sender,EventArgs e)
{
if(this.label1.Text == "Hello Windows Forms!")
    this.label1.Text = "You Clicked?";
else
    this.label1.Text = "Hello Windows Forms!";
}

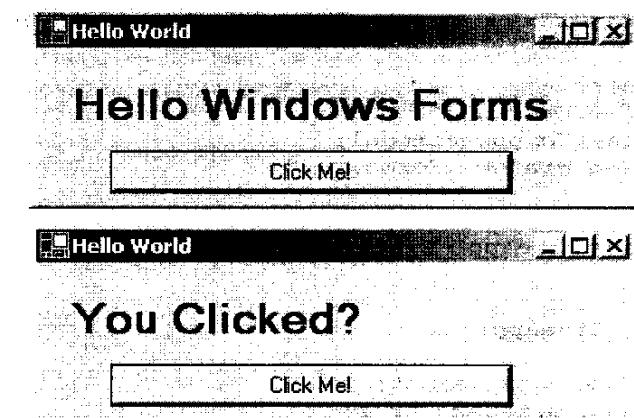
// This main function instantiates a new form and runs
it.
public static void Main(string[] args)
{
    Application.Run(new HelloWindowsForms());
}
}

} // end of namespace
```

Sử dụng file build.bat bạn biên dịch chương trình HWFButton.cs như sau

Build HWFButton.cs

Bạn chạy chương trình HWFButton.exe thu được sau khi biên dịch. Kết quả chương trình sẽ hiển thị cửa sổ như hình 3.1-2.



Hình 3.1-2 Kết quả chương trình HWFButton

Khi bạn kích chuột vào nút nhấn, bộ xử lý sự kiện sẽ được gọi thay thế nội dung chữ trong nhãn label1. Lưu ý, trong ví dụ trên, chúng ta sử dụng cách triệu gọi phương thức theo không gian tên đầy đủ của lớp đối tượng

```
button1.size = new System.Drawing.Size(408, 25);
```

Bạn có thể gọi theo cách ngắn hơn

```
button1.size = new Size(408, 25);
```

Các trình sinh mã (code generate) trong những công cụ phát triển trực quan thường sinh mã theo dạng tên đầy đủ bởi chúng làm điều này một cách tự động. Mặc khác, khi triệu gọi phương thức với đầy đủ không gian tên, chúng ta luôn bão dảm rằng phương thức được gọi tham chiếu đúng tuyệt đối. Khi viết mã bằng tay bạn có thể sẽ mệt mỏi vì phải sử dụng tên khai báo dài, khi đó ta sử dụng cách viết tắt. Hầu như những đối tượng được tạo ra bên trong không gian tên mà bạn khai báo nó bạn đều có thể triệu gọi tên phương thức của đối tượng mà không cần dùng đến không gian tên.

4. ĐỊNH NGHĨA CÁC KIỂU ĐƯỜNG VIỀN (BORDER STYLE) CHO FORMS

Các ví dụ trên đây khá đơn giản, cửa sổ Form tạo ra có kích thước cố định, không có nút nhấn phóng to hay thu nhỏ bên góc phải. Bạn có thể thêm vào những thuộc tính này thông qua thay đổi kiểu đường viền (border) của Form. Khi thay đổi kiểu đường viền, cửa sổ Form của bạn có thể thay đổi kích thước, phóng



to, thu nhỏ như những ứng dụng Windows thuần túy. Chương trình resize.cs trong ví dụ 3.1-4 dưới đây cho sẽ minh họa tính năng cơ bản của một cửa sổ Form.

Ví dụ 3.1-4 resize.cs

```
using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;

public class SizeApp : System.Windows.Forms.Form
{
    public SizeApp()
    {
        this.Text = "SizeApp";
        this.MaximizeBox = true;
        this.FormBorderStyle = FormBorderStyle.Sizable;
    }

    static void Main()
    {
        Application.Run(new SizeApp());
    }
}
```

Biên dịch resize.cs bằng chương trình build.bat. Chạy chương trình bạn có thể phóng to cửa sổ Form toàn màn hình hay thu nhỏ xuống thành icon trên thanh taskbar của Windows.

5. TẠO VÀ THÊM VÀO TRÌNH ĐƠN MENU

Hầu hết các chương trình ứng dụng trong Windows đều có một trình đơn menu. Chương trình Windows Forms cũng không là ngoại lệ. Tương tự như thời phần nút nhấn Button và nhãn Label bạn đã học qua trong phần trên, thành phần trình đơn có thể được tạo ra và thêm vào thông qua biến thành viên **MenuItem** của ứng dụng chính. Sự kiện phát sinh từ một mục chọn của trình đơn cũng được nối vào một bộ xử lý sự kiện qui định dành cho menu.

Menu trong kiến trúc .NET có hai dạng. **MainMenuItem** là trình đơn chỉ được áp dụng cho cửa sổ Form, nó cung cấp giao diện kiểu thực đơn ngang (menubar) truyền thống trong Windows. Dạng thứ hai là trình đơn ngữ cảnh (**ContextMenu**) được dùng để hỗ trợ các tính năng nhanh của ứng dụng. **ContextMenu** thường được hiển thị khi người dùng kích chuột phải lên một vùng nào đó của cửa sổ Form. Trong cả hai dạng nêu trên, từng mục chọn trong menu được biểu diễn là kiểu đối tượng **MenuItem**. Trình đơn menu thường đư

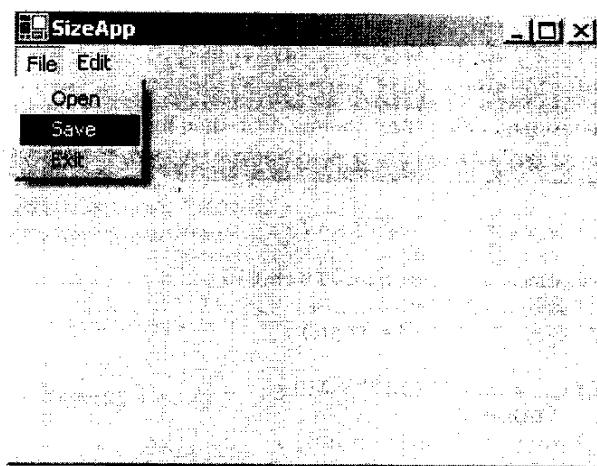
xây dựng theo quan hệ kế thừa cha con. MainMenu quản lý các popup hay dropdown menu, tiếp đó những thành phần này lại quản lý các mục chọn MenuItem của riêng chúng.

Trình tự tạo menu điển hình thường được thực hiện như sau

Ví dụ 3.1-5 Xây dựng menu

```
MainMenu menu = new MainMenu();  
  
MenuItem filemenu = new MenuItem();  
filemenu.Text = "&File";  
menu.MenuItems.Add(filemenu);  
  
MenuItem open = new MenuItem();  
open.Text = "&Open";  
filemenu.MenuItems.Add(open);  
  
MenuItem save= new MenuItem();  
save.Text = "&Save";  
filemenu.MenuItems.Add(save);  
  
MenuItem exit= new MenuItem();  
exit.Text = "E&xit";  
filemenu.MenuItems.Add(exit);  
  
MenuItem editmenu= new MenuItem();  
editmenu.Text = "&Edit";  
  
menu.MenuItems.Add(editmenu);  
  
MenuItem cut= new MenuItem();  
cut.Text = "&Cut";  
editmenu.MenuItems.Add(cut);  
  
MenuItem copy= new MenuItem();  
copy.Text = "Copy";  
editmenu.MenuItems.Add(copy);  
  
MenuItem paste= new MenuItem();  
paste.Text = "&Paste";  
editmenu.MenuItems.Add(paste);  
  
this.Menu = menu;
```

Kết quả menu xây dựng trong bước trên khi hiển thị sẽ có giao diện như hình 3.1-3



Hình 3.1-3 Chương trình với menu

6. TẠO VÀ THÊM VÀO CÁC MỤC CHỌN TẮT (SHORTCUT MENU)

Khi đặt dấu & trước ký tự thể hiện tên của mục chọn trên menu, Windows sẽ gạch chân ký tự này thể hiện đây là ký tự tắt. Bạn có thể kết hợp phím Alt và ký tự tắt để triệu hồi mục chọn trên trình đơn. Ví dụ:

```
filemenu.Text = "&File";
...
open.Text = "&Open";
```

để gọi mục chọn “Open” bạn có thể nhấn phím tắt Alt+F để mở menu popup File tiếp đến nhấn phím O để gọi mục Open. Chúng ta có thể chỉ định tương ứng phím tắt cho một mục chọn, ví dụ Ctrl+O sẽ gọi mục Open như sau:

Ví dụ 3.1-6 Thêm phím tắt vào mục chọn File

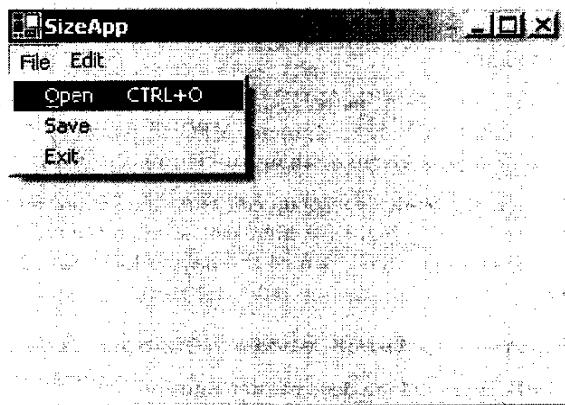
```
MainMenu menu = new MainMenu();
MenuItem filemenu = new MenuItem();
filemenu.Text = "&File";
menu.MenuItems.Add(filemenu);

MenuItem open = new MenuItem();
open.Text = "&Open";
filemenu.MenuItems.Add(open);

open.Shortcut=CtrlO;
```

```
open.ShowShortcut=true;
```

Đoạn mã trên sẽ gán phím tắt Ctrl+O cho mục chọn Open. Bạn hãy nhấn Alt+F để hiển thị menu con File tiếp đến nhấn O để đến mục chọn Open (hình 3.1-4). Lưu ý phím Ctrl+O hiển thị kế bên tiêu đề Open. Nếu bạn nhấn trực tiếp Ctrl+O thì bộ xử lý sự kiện dành cho mục chọn Open sẽ phát sinh. Chúng ta sẽ học cách cài đặt bộ xử lý sự kiện cho mục chọn của menu trong phần sau.



Hình 3.1-4 Thể hiện menu tắt

7. XỬ LÝ SỰ KIỆN PHÁT SINH TỪ MENU

Có rất nhiều sự kiện bạn cần phải xử lý đối với mục chọn MenuItem. Một bộ xử lý sự kiện cơ bản là một phương thức tuân theo khai báo

```
void_function(Object, EventArgs);
```

Bạn khai báo phương thức theo dạng này tiếp đến đăng ký nó với mục chọn trên trình đơn thông qua lớp EventHandler. Ví dụ, khi người dùng kích chuột vào mục chọn Open, ta hiển thị một hộp thoại thông báo sự kiện đã diễn ra, để thực hiện điều này, trước hết bộ xử lý sự kiện được cài đặt như sau:

Ví dụ 3.1-7 Cài đặt bộ xử lý sự kiện cho mục chọn

```
public class SizeApp : System.Windows.Forms.Form
{
    public void OnFileOpen(Object sender, EventArgs e)
    {
        MessageBox.Show("You selected File-open");
    }
}
```

Ở đây OnFileOpen() là bộ xử lý sự kiện, bạn có thể đặt cho phương thức một tên khác. Tuy nhiên để bộ xử lý này móc nối được với sự kiện Click của mục



chọn MenuItem mang tên Open, bạn chuyển giao bộ xử lý cho đối tượng (hay nói cách khác là đăng ký phương thức xử lý sự kiện Click với đối tượng) như sau:

```
open.Click+=new EventHandler(OnFileOpen);
```

Bây giờ bất kỳ khi nào bạn nhấn Alt+F+O hoặc Ctrl+O thì mục chọn Open sẽ kích hoạt sự kiện Click và phương thức OnFileOpen() được gọi thực thi.

7.1. Sự kiện điều khiển giao diện người dùng của MenuItems

Một số sự kiện khác phát sinh bởi MenuItems cho phép bạn phản hồi hay tùy biến cho người dùng nhiều kinh nghiệm. MFC sử dụng lớp **CCmdUI** cho mục đích này trong khi Windows Forms sử dụng sự kiện nguồn mang tên **Popup**.

Trước khi hiển thị trình đơn popup con chứa mục chọn, sự kiện **Popup** sẽ phát sinh. Đây là cơ hội giúp bạn thực hiện một số thao tác kiểm tra để hiển thị đúng đắn trạng thái của mục chọn. Bạn có thể bẫy sự kiện này bằng cách đưa vào bộ xử lý như sau:

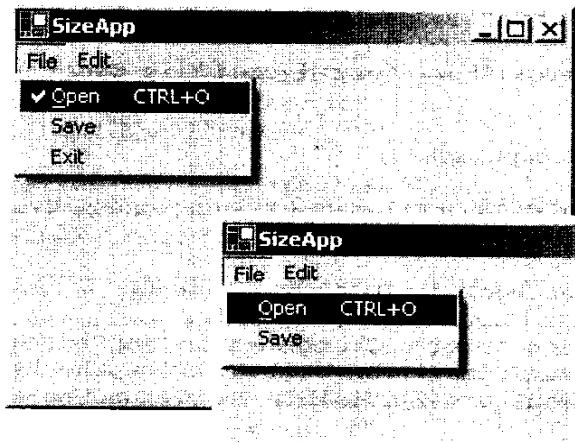
```
filemenu.Popup+=new EventHandler(OnPopupFileMenu);
```

Bộ xử lý **OnPopupFileMenu** được cài đặt như sau:

Ví dụ 3.1-8 Cài đặt bộ xử lý **OnPopupFileMenu**

```
bool m_bPopupChecked;
public void OnPopupFilemenu(Object Sender, EventArgs e)
{
    m_bPopupChecked = !m_bPopupChecked;
    MenuItem item = (MenuItem)Sender;
    item.MenuItems[0].Checked = m_bPopupChecked;
    item.MenuItems[1].Enabled = m_bPopupChecked;
    item.MenuItems[2].Visible = m_bPopupChecked;
}
```

Phương thức **OnPopupFilemenu()** trên đây cài đặt một số công việc thường làm của sự kiện **Popup** đó là kiểm tra, đặt dấu checkmark, ẩn hoặc vô hiệu hóa mục chọn **MenuItems** theo một điều kiện nào đó. Lớp chương trình chính chứa một biến Boolean mang tên **m_bPopupChecked**. Mỗi khi menu File được bung ra, cờ **m_bPopupChecked** được bật trị true hoặc false tùy vào trạng thái trước đó. Đối tượng **Sender** khi này chính là thể hiện của lớp **MenuItems**, cho nên việc chuyển kiểu **Sender** từ **Object** về **MenuItem** là hoàn toàn hợp lệ. Ba mục chọn trong trình đơn File được đánh dấu, cho phép hiệu lực hoặc ẩn đi tùy thuộc vào trạng thái của cờ **m_bPopupChecked**. Hình 3.1-5 cho thấy mục chọn trên menu hiển thị ở hai trạng thái



Hình 3.1-5 Kết quả xử lý menu

7.2. Định nghĩa MenuItems là thanh phân cách (seperator)

Các mục chọn trên menu thường được nhóm lại theo một chức năng nào đó. Giữa những nhóm này thường có một thanh phân cách (seperator) giúp người dùng phân biệt rõ các nhóm mục chọn. Trong Windows Forms, thanh phân cách trên menu không thực hiện bất kỳ chức năng gì ngoài việc thể hiện một đường thẳng phân ranh giới giữa các mục chọn. Việc tạo thanh phân cách rất đơn giản. Bạn chỉ việc gán thuộc tính Text của mục chọn là ký tự "-". Ví dụ:

```
MenuItem dummysmenu = new MenuItem();
dummysmenu.Text = "Separator";
menu.MenuItems.Add(dummysmenu);

dummysmenu.MenuItems.Add(new MenuItem("Above"));
dummysmenu.MenuItems.Add(new MenuItem("-"));
dummysmenu.MenuItems.Add(new MenuItem("Below"));
```

Dưới đây là toàn bộ mã nguồn của chương trình resize2.cs bổ sung các mục chọn menu vào chương trình resize.cs

```
using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;
```

```
public class SizeApp : System.Windows.Forms.Form
{
```

```
public void OnFileOpen(Object sender, EventArgs e)
{
    MessageBox.Show("You selected File-Open!");
}

bool m_bPopupChecked;

public void OnPopupFilemenu(Object Sender, EventArgs e)
{

    m_bPopupChecked = !m_bPopupChecked;
    MenuItem item = (MenuItem)Sender;
    item.MenuItems[0].Checked = m_bPopupChecked;
    item.MenuItems[1].Enabled = m_bPopupChecked;
    item.MenuItems[2].Visible = m_bPopupChecked;
}

public    SizeApp()
{
    this.Text = "SizeApp";
    this.MaximizeBox = true;
    this.FormBorderStyle = FormBorderStyle.Sizable;

    MainMenu menu = new MainMenu();
    menu.RightToLeft = RightToLeft.Yes;
    MenuItem filemenu = new MenuItem();
    filemenu.Text = "&File";
    menu.MenuItems.Add(filemenu);

    MenuItem open = new MenuItem();
    open.Text = "&Open";

    filemenu.MenuItems.Add(open);

    open.Shortcut = Shortcut.CtrlO;
    open.ShowShortcut = true;
    open.Click += new EventHandler(OnFileOpen);

    MenuItem save= new MenuItem();
    save.Text = "Save";
    filemenu.MenuItems.Add(save);

    MenuItem exit= new MenuItem();
    exit.Text = "Exit";
    filemenu.MenuItems.Add(exit);

    MenuItem editmenu = new MenuItem();
    editmenu.Text = "Edit";
```

```

        menu.MenuItems.Add(editmenu);

        MenuItem cut = new MenuItem();
        cut.Text = "Cut";
        editmenu.MenuItems.Add(cut);

        MenuItem copy = new MenuItem();
        copy.Text = "Copy";
        editmenu.MenuItems.Add(copy);

        MenuItem paste = new MenuItem();
        paste.Text = "Paste";
        editmenu.MenuItems.Add(paste);

        MenuItem dummmymenu = new MenuItem();
        dummmymenu.Text = "Separator";
        menu.MenuItems.Add(dummmymenu);

        dummmymenu.MenuItems.Add(new MenuItem("Above"));
        dummmymenu.MenuItems.Add(new MenuItem("-"));
        dummmymenu.MenuItems.Add(new MenuItem("Below"));

        filemenu.Popup += new EventHandler(OnPopupFilemenu);

        this.Menu = menu;
    }

    static void Main()
    {
        Application.Run(new SizeApp());
    }
}

```

7.3. Xử lý sự kiện Select

Khi các mục chọn trên menu popup đã hiển thị, người dùng di chuyển chuột đến từng mục chọn hoặc dùng các phím mũi tên di chuyển vệt sáng đến các mục chọn khác nhau của trình đơn. Khi vệt sáng đậu vào một mục chọn, sự kiện Select sẽ phát sinh. Chương trình có thể dùng sự kiện này để cập nhật thông tin hướng dẫn về mục chọn trên thanh trạng thái statusbar của cửa sổ Form hoặc thực hiện một thao tác tùy biến giao diện người dùng nào đó. Dưới đây là ví dụ cho thấy cách xử lý sự kiện Select

Ví dụ 3.1-9 menus.cs

```
using System;
```

```
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;

//This application shows off som of the menu features
//Lines below have been added but then commented out
//please feel free to remove the comment slashes to
//see the effect of the commands.
public class menuapp : System.Windows.Forms.Form
{
    Label label;

    void ShowInfo(Object Sender, EventArgs e)
    {
        MenuItem item=(MenuItem)Sender;
        switch(item.Text)
        {
            case "&Open":
                label.Text = "Open a file from disk";
                break;
            case "&Save":
                label.Text = "Save a file onto disk";
                break;
            case "E&xit":
                label.Text = "Exit MenuApp";
                break;
        }
    }

    public menuapp()
    {
        this.Text = "MenuApp";
        this.MaximizeBox = true;
        this.FormBorderStyle = FormBorderStyle.Sizable;

        this.label = new Label();
        label.Location = new Point(8,100);
```



```
label.Size = new Size(200,25);

this.Controls.Add(label);

MainMenu menu = new MainMenu();

MenuItem filemenu = new MenuItem();
filemenu.Text = "&File";
menu.MenuItems.Add(filemenu);

MenuItem open = new MenuItem();
open.Text = "&Open";
open.Select += new EventHandler>ShowInfo);
filemenu.MenuItems.Add(open);

MenuItem save= new MenuItem();
save.Text = "&Save";
save.Select += new EventHandler>ShowInfo);
filemenu.MenuItems.Add(save);

MenuItem exit= new MenuItem();
exit.Text = "E&xit";
exit.Select += new EventHandler>ShowInfo);
filemenu.MenuItems.Add(exit);

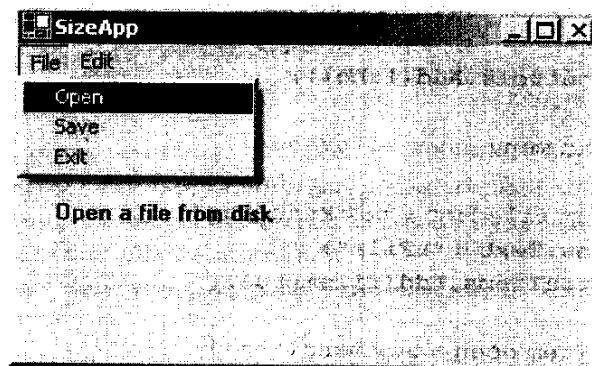
this.Menu = menu;
```

}

```
static void Main()
{
    Application.Run(new menuapp());
}
```

}

Biên dịch và chạy chương trình. Khi bạn di chuyển chuột trên menu thì nội dung nhân label sẽ thay đổi phản ánh mục chọn đang chiếu sáng (hình 3.1-6).



Hình 3.1-6 Kết quả xử lý sự kiện Select

7.4. Thiết kế khung Menu

Trình đơn được xây dựng từ tập hợp các thành phần MenuItem. Những mục chọn menu thông thường được dàn ngang trên thanh menubar của cửa sổ Form và đặt từ trái qua phải. Bạn có thể thay đổi kiểu thiết lập mặc định này.

Phương thức Break và BarBreak được dùng để tạo trình đơn xếp ngang hay vì xếp dọc. Thiết lập thuộc tính BarBreak trong MenuItem sẽ làm cho mục chọn được vẽ sang một cột mới. BarBreak sẽ thêm vào thanh phân cách dọc tạo thành các cột trong trình đơn. Break chỉ tạo một cột mới nhưng không vẽ thanh dọc phân cách. Bạn có thể thay đổi menus.cs đôi chút để được một khung menu mới như sau:

```

MenuItem filemenu = new MenuItem();
filemenu.Text = "&File";
menu.MenuItems.Add(filemenu);

MenuItem open = new MenuItem();
open.Text = "&Open";
open.Select += new EventHandler>ShowInfo);
filemenu.MenuItems.Add(open);

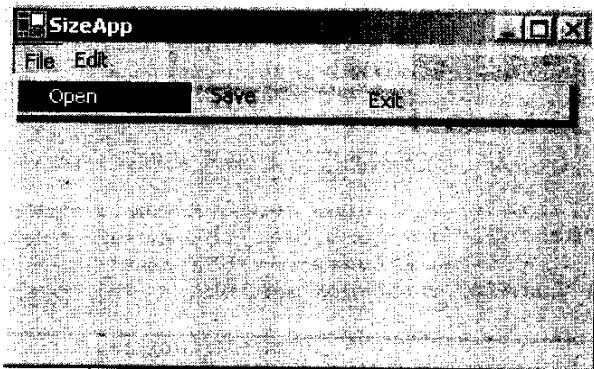
MenuItem save= new MenuItem();
save.Text = "&Save";
save.Select += new EventHandler>ShowInfo);
filemenu.MenuItems.Add(save);
save.BarBreak=true;

MenuItem exit= new MenuItem();
exit.Text = "E&xit";
exit.Select += new EventHandler>ShowInfo);
filemenu.MenuItems.Add(exit);

```

```
exit.BarBreak=true;
exit.Break=true;
```

Kết quả khi đó sẽ hiển thị như hình 3.1-7



Hình 3.1-7 Tác động của thuộc tính BarBreak

Các cột không bắt buộc phải cân đối số mục chọn, bạn có thể đặt mục chọn Exit riêng rẽ trong một cột tách biệt như sau:

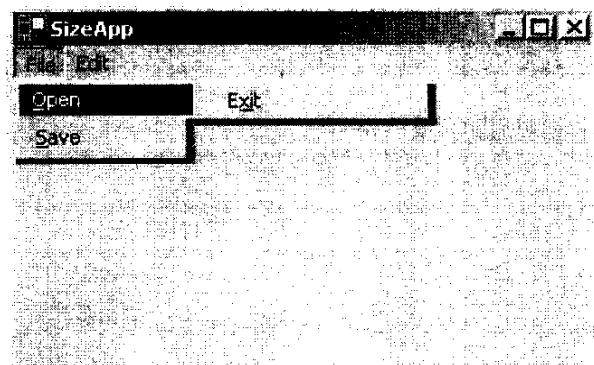
```
MenuItem filemenu = new MenuItem();
filemenu.Text = "&File";
menu.MenuItems.Add(filemenu);

MenuItem open = new MenuItem();
open.Text = "&Open";
open.Select += new EventHandler>ShowInfo);
filemenu.MenuItems.Add(open);

MenuItem save= new MenuItem();
save.Text = "&Save";
save.Select += new EventHandler>ShowInfo);
filemenu.MenuItems.Add(save);

MenuItem exit= new MenuItem();
exit.Text = "E&xit";
exit.Select += new EventHandler>ShowInfo);
filemenu.MenuItems.Add(exit);
exit.Break=true;
```

Mỗi khi Break mang trị true, mục chọn sẽ được thêm qua cột mới. Kết hiển thị như hình 3.1-8



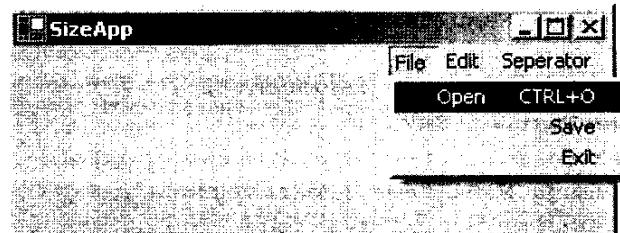
Hình 3.1-8 Một dạng thể hiện khác của BarBreak

7.5. Hiển thị Menu từ phải qua trái

Bạn có thể đặt menu không theo qui ước chuẩn. Thông thường các menu của menubar bắt đầu từ trái qua phải. Chúng ta có thể thay đổi thứ tự từ phải qua trái như sau

```
MainMenu menu = new MainMenu();
MenuItem filemenu = new MenuItem();
filemenu.Text = "&File";
menu.MenuItems.Add(filemenu);
menu.RightToLeft=RightToLeft.Yes;
```

kết quả bạn sẽ được menu hiển thị như hình 3.1-9.



Hình 3.1-9 Menu cạnh phải

7.6. Tạo và sử dụng các Menu ngữ cảnh (ContextMenu)

Menu ngữ cảnh còn gọi là các menu popup nổi. Nó xuất hiện và thay đổi các mục chọn tùy theo trạng thái của ngữ cảnh người dùng. Khi người dùng di chuột phải lên vùng soạn thảo, lên đối tượng ảnh, lên cửa sổ Form ta

thể cung cấp menu ngữ cảnh để người dùng triệu gọi. Menu ngữ cảnh làm tăng tốc truy xuất chức năng của ứng dụng hơn là các menu cố định trên menubar.

Để tạo menu ngữ cảnh trong các ứng dụng Windows Forms, bạn sử dụng lớp thành phần **ContextMenu**.

Thêm vào, hiển thị, thay đổi, cài đặt bộ xử lý sự kiện cho các mục chọn trong menu ngữ cảnh hoàn toàn tương tự như trong **MainMenu**. Đoạn mã sau sẽ tạo ra menu ngữ cảnh với 3 mục chọn

```
ContextMenu cmenu = new ContextMenu();

cmenu.MenuItems.Add(new MenuItem("&First"));
cmenu.MenuItems.Add(new MenuItem("&Second"));
cmenu.MenuItems.Add(new MenuItem("-"));
cmenu.MenuItems.Add(new MenuItem("&Third"));

this.ContextMenu=cmenu;
```

Hãy kích chuột phải vào cửa sổ Form của ứng dụng, menu ngữ cảnh trên sẽ xuất hiện.

7.7. Thay thế, sao chép lại và trộn các mục của Menu

Một tập quán thông thường của các ứng dụng Windows đó là thay đổi menu tùy theo nội dung dữ liệu hiển thị trong vùng cửa sổ. Nhằm giúp bạn tiết kiệm thời gian trong việc tạo ra và hủy các menu động .NET cho phép bạn hoán chuyển hoặc trộn lẫn các mục chọn của menu này với menu đã định nghĩa trước đó.

Ví dụ dưới đây sẽ minh họa cách tạo nhiều menu sau đó thực hiện việc kết hợp trộn chúng lại theo các cách khác nhau tùy theo lựa chọn của người dùng.

Ví dụ 3.1-10 menuswop.cs

```
using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;

namespace Sams {

    class menuswop : System.Windows.Forms.Form
    {
        MainMenu m_menu;
        MenuItem m_editmenu,m_menumenu,m_switchitem,
                  m_showsecond,m_merge;
```

```
MenuItem m_playmenu;

bool m_bswop;
bool m_bshowsecond;
bool m_bmerge;

void addmenuItem(MenuItem menu, string s)
{
    MenuItem temp=new MenuItem(s);
    temp.Enabled=false;
    menu.MenuItems.Add(temp);
}

void BuildMenu()
{
    m_menu=new MainMenu();
    m_menu.MenuItems.Add(m_menumenu.CloneMenu());

    if(m_bmerge) // trộn các mục chọn
    {
        MenuItem temp=new MenuItem();

        if(!m_bswop)
        {
            addmenuItem(temp,"Edit");
            temp.MergeMenu(m_editmenu.CloneMenu());
        }
        else
        {
            addmenuItem(temp,"Play");
            temp.MergeMenu(m_playmenu.CloneMenu());
        }

        temp.MenuItems.Add(new MenuItem("-"));

        if(m_bshowsecond)
        {

```

```
if(!m_bswap)
{
    addmenuitem(temp,"Play");
    temp.MergeMenu(m_playmenu.CloneMenu());
}
else
{
    addmenuitem(temp,"Edit");
    temp.MergeMenu(m_editmenu.CloneMenu());
}
}

temp.Text = "&Merged";
m_menu.MenuItems.Add(temp);

}

else // khi không trộn các mục chọn.
{
    if(!m_bswap)
    {
        if(m_bshowsecond)
        {
            m_menu.MenuItems.Add(
                m_editmenu.CloneMenu());
        }
        m_menu.MenuItems.Add(m_playmenu.CloneMenu());
    }
    else
    {
        if(m_bshowsecond)
        {
            m_menu.MenuItems.Add(
                m_playmenu.CloneMenu());
        }
        m_menu.MenuItems.Add(m_editmenu.CloneMenu());
    }
}
```

```
    this.Menu = m_menu;
}

//This method sets or resets the checks on menu items

void PopupMenuMenu(Object sender, EventArgs e)
{
    m_menu.MenuItems[0].MenuItems[0].Checked = m_bswop;
    m_menu.MenuItems[0].MenuItems[1].Checked =
m_bshowsecond;
    m_menu.MenuItems[0].MenuItems[2].Checked =
m_bmerge;
}

// The event handler for the switch menu entry
void OnSwitchMenu(Object sender, EventArgs e)
{
    m_bswop = !m_bswop;
    BuildMenu();
}

//The event handler for the show menu entry
void Onshowsecond(Object sender, EventArgs e)
{
    m_bshowsecond = !m_bshowsecond;
    BuildMenu();
}

//The event handler for the merge menu entry
void OnMerge(Object sender, EventArgs e)
{
    m_bmerge = !m_bmerge;
    BuildMenu();
}

public menuswop()
{
    // setup a main menu
    m_menuitem = new MenuItem("&Menu");
}
```

```
m_menumenu.Popup += new EventHandler(PopupMenuMenu);

//Create the switch item.
m_switchitem=new MenuItem("&Switch");
m_switchitem.Click+=new
    EventHandler(OnSwitchMenu);
m_menumenu.MenuItems.Add(m_switchitem);

m_showsecond = new MenuItem("&Show");
m_showsecond.Click+= new
    EventHandler(Onshowsecond);
m_menumenu.MenuItems.Add(m_showsecond);

m_merge = new MenuItem("&Merge");
m_merge.Click += new EventHandler(OnMerge);
m_menumenu.MenuItems.Add(m_merge);

// create a second menu
m_editmenu=new MenuItem("&Edit");
m_editmenu.MenuItems.Add(new MenuItem("Cut"));
m_editmenu.MenuItems.Add(new MenuItem("Copy"));
m_editmenu.MenuItems.Add(new MenuItem("Paste"));

// and an alternative.
m_playmenu=new MenuItem("&Play");
m_playmenu.MenuItems.Add(new MenuItem("Normal"));
m_playmenu.MenuItems.Add(new MenuItem(
    "Fast Forward"));
m_playmenu.MenuItems.Add(new MenuItem("Reverse"));

m_bshowsecond=true;

//Now build the menu from its parts..
BuildMenu();

}

public static void Main()
```

```
{  
    Application.Run(new menuSwop());  
}  
}  
  
}// end of Sams namespace
```

Trong ví dụ trên, bạn có thể thấy việc khởi tạo menu được thực hiện theo cách thông thường, tuy nhiên thực sự không có menu nào được thêm vào danh sách MenuItems cha. Mục Edit và Play trên menu được giữ tách biệt.

Hàm BuildMenu chịu trách nhiệm tạo menu và gán nó cho menu chính trong chương trình. Điều này sẽ khiến GC xóa tất cả các mục chọn và menu con trong menu chính. BuildMenu sau đó tiếp tục tạo ra menu được sắp xếp hoặc trộn hai menu con lại thành một tùy theo thiết lập của các biến thành viên bên trong lớp.

Kỹ thuật này cho phép bạn khởi tạo các mục chọn menu cùng với những bộ xử lý mục chọn ngay một chỗ. Sau đó bạn có thể dùng chúng theo nhiều cách kết hợp khác nhau mà không cần phải theo dõi hoặc đặt chúng về giá trị mặc định ban đầu.

Việc trộn các mục menu trong ví dụ trên rất đơn giản. Thực tế, bạn có thể kết hợp các mục MenuItems theo cách phức tạp hơn. Mỗi mục chọn MenuItems có một thuộc tính qui định trật tự sao cho khi được trộn vào menu khác chúng sẽ được sắp xếp theo một trình tự nhất định. Ví dụ, bạn có thể vừa muốn bạn muốn trong menu File có các mục được nhóm lại tùy theo kiểu hoặc loại file tài nguyên nào đó. Với việc đặt thứ tự cho mục chọn bạn có thể đảm bảo rằng những mục chọn thường sử dụng sẽ được đặt ở đầu. Những mục chọn ít sử dụng sẽ được sắp xếp ở phần sau. Chương trình menuorder.cs dưới đây sẽ minh họa chức năng trộn menu theo thứ tự tùy theo việc lựa chọn thường xuyên của người sử dụng.

Ví dụ 3.1.11 menuorder.cs

```
using System;  
using System.Drawing;  
using System.ComponentModel;  
using System.Windows.Forms;  
  
namespace Sams {  
    class menuorder : System.Windows.Forms.Form  
    {  
        MainMenu m_menu;  
        MenuItem m_workingmenu;  
        MenuItem m_menuentry;
```

```
// this event handler is called whenever an item is used
void OnUsed(Object sender, EventArgs e)
{
    for(int i=0;i<m_menuentry.MenuItems.Count;i++)
        m_menuentry.MenuItems[i].MergeOrder++;

    MenuItem m=(MenuItem)sender;

    m.MergeOrder--;

    if(m.MergeOrder < 0)
        m.MergeOrder = 0;

}

// this event handler is also invoked. You could have
// many event handlers attached to the Click sources
void OnClicked(Object sender, EventArgs e)
{
    MenuItem m=(MenuItem)sender;
    MessageBox.Show("You clicked "+m.Text);
}

//As a menu is popped up it is constructed on the fly.
void OnPopup(Object sender, EventArgs e)
{
    m_menu.MenuItems.Clear();
    m_workingmenu.MenuItems.Clear();
    m_workingmenu.MergeMenu(m_menuentry);
    m_menu.MenuItems.Add(m_workingmenu);
}

// Sets up the initial menu text and orders.
public menuorder()
{
```

```
string[] s=new
string[8]{"Cats","Dogs","Elephants","Geese","Moosees"
Rats","Giunea-pigs","Horses"};
```

```
m_menu = new MainMenu();
m_menuentry = new MenuItem("&Menu");
m_menuentry.Popup += new EventHandler(OnPopup);

m_workingmenu = new MenuItem();

for(int i=0;i<8;i++)
{
    MenuItem temp = new MenuItem(s[i]);
    temp.MergeOrder=i;
    temp.Click+=new EventHandler(OnUsed);
    temp.Click+=new EventHandler(OnClicked);
    m_menuentry.MenuItems.Add(temp);
}

m_workingmenu.MergeMenu(m_menuentry);
m_menu.MenuItems.Add(m_workingmenu);
this.Menu = m_menu;
}

// instantiates and runs the application.
public static void Main()
{
    Application.Run(new menuorder());
}
}
```

Ví dụ trên cũng minh họa cho thấy cách thêm vào hai bộ xử lý cho cùn **kiêm Click, Select hay Popup** của Windows Forms.

7.8. Tao và thêm vào các Menu con (sub-Menu)

Đối tượng **MenuItems** trong Windows Forms chứa một tập hợp các đối tượng **MenuItem** (collection). Điều này cho phép bạn tạo ra một cấu trúc kế thừa

của menu. Cấu trúc kế thừa này bao gồm những menu cha và menu con lồng nhau với số cấp tùy ý. Ví dụ để tạo menu con chúng ta thực hiện như sau:

Ví dụ 3.1-12 submenus.cs

```
using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;

public class submenuapp : System.Windows.Forms.Form
{
    public submenuapp()
    {
        this.Text = "SubMenuApp";
        this.MaximizeBox = true;
        this.BorderStyle = FormBorderStyle.Sizable;

        MainMenu menu = new MainMenu();

        MenuItem filemenu = new MenuItem();
        filemenu.Text = "&File";
        menu.MenuItems.Add(filemenu);

        MenuItem open = new MenuItem();
        open.Text = "&Open";
        filemenu.MenuItems.Add(open);

        MenuItem print= new MenuItem();
        print.Text = "Print...";
        filemenu.MenuItems.Add(print);

        MenuItem temp= new MenuItem();
        temp.Text = "Pre&view";
        print.MenuItems.Add(temp);

        temp= new MenuItem();
        temp.Text = "To &File";
        print.MenuItems.Add(temp);

        MenuItem exit= new MenuItem();
        exit.Text = "E&xit";
        filemenu.MenuItems.Add(exit);

        this.Menu = menu;
    }
}
```

```

ContextMenu cmenu = new ContextMenu();
cmenu.MenuItems.Add(new MenuItem("&First"));
cmenu.MenuItems.Add(new MenuItem("&Second"));
cmenu.MenuItems.Add(new MenuItem("-"));
cmenu.MenuItems.Add(new MenuItem("&Third"));

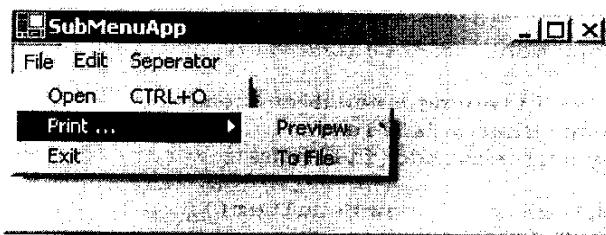
this.ContextMenu=cmenu;

}

static void Main()
{
    Application.Run(new submenuapp());
}

}

```



Hình 3.1-10 Menu con

8. KẾT CHƯƠNG

Trong chương này, chúng ta đã xem qua cách xây dựng ứng dụng Windows Form từ mức đơn giản cho đến xây dựng hệ thống menu phức hợp xử lý các thao tác lựa chọn của người dùng thông qua các bộ xử lý sự kiện (event handler). Chúng ta cũng đã học qua mô hình chuyển giao sự kiện trong hệ thống (Delegate system) thay cho cách xử lý thông điệp truyền thống trước đây trong MFC hay hàm WinProc kinh điển của ứng dụng Windows viết trong C. Song song đó chúng ta đã xem qua một số thành phần giao diện (GUI component) cơ bản của môi trường .NET. Tiếp theo trong chương sau bạn sẽ học chi tiết hơn về những thành phần giao diện đồ họa này.

Chương 3.2

XỬ LÝ GIAO DIỆN ĐỒ HỌA (GUI)

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Hộp thoại (Dialogs)*
- ✓ *Tạo và xử lý hộp thoại*
- ✓ *Sử dụng các thành phần điều khiển*

1. GIỚI THIỆU VỀ THÀNH PHẦN GIAO DIỆN ĐỒ HỌA (GUI)

Chương này chúng ta sẽ tìm hiểu qua các thành phần giao diện của Windows Forms. Bạn sẽ học cách tạo và kết nối những thành phần giao diện này cùng với bộ xử lý sự kiện tương ứng vào cửa sổ Form hay hộp thoại (Dialog) của ứng dụng. Thành phần giao diện đồ họa sẽ giúp bạn tạo nên các ứng dụng Windows rất bắt mắt.

Thư viện của Windows Form chứa rất nhiều thành phần giao diện người dùng chuẩn có thể sử dụng cho ứng dụng của bạn. Rất nhiều thành phần giao diện có những tính năng cao cấp do việc cải tiến hơn hẳn những thành phần điều khiển anh em của chúng trong họ thư viện Win32 truyền thống của Windows. Đó là các hộp thoại, danh sách listbox, điều khiển cây Tree, khung chứa (panel), nhãn (label), thanh công cụ (toolbar)... Một tập hợp phong phú các hộp thoại thông dụng như hộp thoại chọn file, chọn màu, chọn font, hộp thoại in ấn ... Sau giao diện menu, hộp thoại hay dialog là thành phần giao diện được sử dụng nhiều nhất đối với người dùng. Nào chúng ta hãy bắt đầu tìm hiểu chi tiết về những thành phần này.

2. HỘP THOẠI (DIALOG)

Có hai kiểu hộp thoại: kiểu modal (bắt buộc nhập liệu) và kiểu modeless (không bắt buộc nhập liệu). Hộp thoại kiểu modal yêu cầu người dùng phải hoàn tất thao tác nhập liệu hoặc những yêu cầu chọn lựa trước khi quay trở về dòng hoạt động chính của chương trình (ví dụ như các hộp thoại Options trong MS Word). Hộp thoại kiểu modeless có thể được dùng để thực hiện thao tác nào đó nhưng vẫn cho phép người dùng tương tác với phần còn lại của chương trình chính (ví dụ như hộp thoại tìm kiếm Search Dialog).

Hộp thoại rất hữu dụng nếu nó được thiết kế đơn giản. Nó giúp chương trình tăng khả năng tương tác với người sử dụng. Tuy nhiên sử dụng quá nhiều hộp thoại nhỏ chứa ít thông tin có thể gây bức bối cho người dùng. Ngược lại, các hộp thoại với quá nhiều mục có thể làm người dùng bối rối và không biết phải chọn những mục nào trước.

Hộp thoại trong thư viện MFC của Visual C++ thường là sự pha trộn giữa các lời gọi thuần Win32 API và các lớp thư viện MFC hướng đối tượng nhất là trong việc trao đổi dữ liệu sử dụng hệ thống DDX (Dialog Data eXchange) giữa các thành phần điều khiển trong hộp thoại. Hệ thống trao đổi này xem dữ liệu trong hộp thoại như là các biến dữ liệu thành viên (data member) của lớp Dialog nhưng lại dùng những biến này như là vùng tạm để chứa các thông tin thật sự hiệu chỉnh bởi các thành phần điều khiển trên bề mặt của hộp thoại. Điều này có nghĩa là không có cơ chế cập nhật diễn ra tức thời cho hộp thoại bởi vì dữ liệu cần phải được chuyển đến hoặc đi khỏi các thành phần điều khiển hay cửa sổ con bên trong hộp thoại bởi những thông điệp Windows. Hộp thoại trong Windows Form hoàn toàn khác. Chúng tương tự như bản thân ứng dụng đã tạo ra nó, các thành phần điều khiển và cơ chế xử lý hoàn toàn mang tính hướng đối tượng cũng như làm việc theo một logic hợp lý hơn.

Trong môi trường .NET, hộp thoại là thành phần thiết yếu của Windows Form. Các điều khiển được đặt trên hộp thoại theo cùng cách mà bạn dùng để đặt chúng trên các cửa sổ form thông thường của ứng dụng – bằng công cụ trực quan hoặc bằng việc viết mã.

2.1. Sử dụng các hộp thoại thông dụng (common dialog)

Các hộp thoại thông dụng được dùng với mục đích đơn giản hóa và tạo sự tương tác thống nhất cho người dùng sử dụng hệ điều hành Windows. Hộp thoại được hiển thị bằng cách gọi hàm ShowDialog. Tất cả các hộp thoại thông dụng đều ra đời và kế thừa từ lớp `System.Windows.Forms.CommonDialog`. Bạn cũng có thể dùng lớp cơ sở này để tạo ra các hộp thoại tùy biến theo ý mình.

Hộp thoại thường sử dụng nhất – `FileDialog`

Hộp thoại thường được sử dụng nhất trong các hộp thoại đó là hộp thoại mở, đọc và ghi file (file dialog). Windows Forms định nghĩa hai loại hộp thoại dạng file đó là hộp thoại mở file (open file dialog) và hộp thoại lưu file (save file dialog). Để dùng các hộp thoại này trong ứng dụng, bạn cần phải khai báo biến thành viên có kiểu `OpenFileDialog` hoặc `SaveFileDialog` trong lớp ứng dụng chính. Và cũng như các hộp thoại cơ bản khác, hai lớp này cũng bắt nguồn từ lớp `System.Windows.Forms.Dialog`.

Ví dụ 3.2-1 dưới đây là một ứng dụng đơn giản cho phép hiển thị hộp thoại mở file.

Ví dụ 3.2-1 `fileopen.cs`

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Sams
{

```



```
class fileopenapp : System.Windows.Forms.Form
{
    MainMenuItem m_menu;
    MainMenuItem m_filemenu;

    void OnOpenFile(Object sender, EventArgs e)
    {
        OpenFileDialog dlg=new OpenFileDialog();
        if(dlg.ShowDialog() == DialogResult.OK)
        {
            MessageBox.Show("You selected
                            the file "+dlg.FileName);
        }
    }

    void OnExit(Object sender, EventArgs e)
    {
        Dispose();
    }

    fileopenapp()
    {
        m_menu = new MainMenuItem();
        m_filemenu=new MainMenuItem("&File");
        m_menu.MenuItems.Add(m_filemenu);
        MainMenuItem t;

        t=new MainMenuItem("&Open");
        t.Click += new EventHandler(OnOpenFile);
        m_filemenu.MenuItems.Add(t);

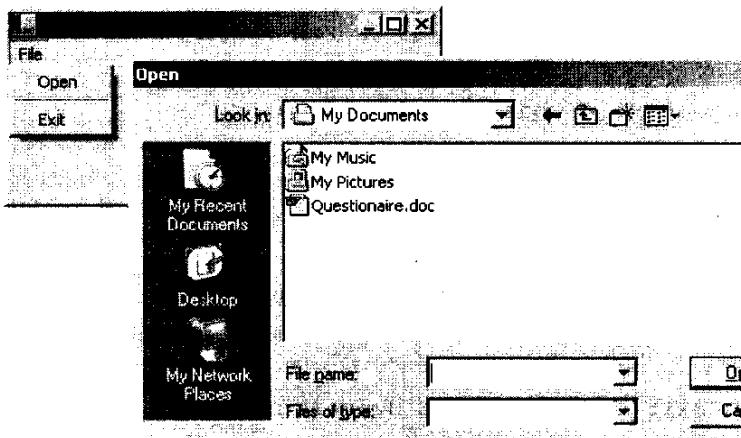
        t=new MainMenuItem("-");
        m_filemenu.MenuItems.Add(t);

        t=new MainMenuItem("E&xit");
        t.Click += new EventHandler(OnExit);
        m_filemenu.MenuItems.Add(t);

        this.Menu = m_menu;
    }

    public static void Main()
    {
        Application.Run(new fileopenapp());
    }
}
```

```
} // end of Sams namespace
```



Hình 3.2-1 Kết quả chương trình fileopen.cs

Trên đây là ví dụ đơn giản nhất về cách sử dụng hộp thoại. Trong các hộp thoại thường cung cấp nhiều chức năng và hoạt động phức tạp hơn.

Một thành phần điều khiển của hộp thoại được sử dụng theo cách biệt khi ứng dụng được kiến tạo bởi Visual Studio .NET. Trong ví dụ dưới đây ta sẽ tạo ra một hộp thoại khi chương trình chạy, đồng thời kết nối sự kiện (event source) mang tên FileOK vào bộ xử lý tinh huống của nút nhấn trên Form. Việc chuyển giao cho bộ xử lý sự kiện trong nút khác so với sự kiện của menu. Dấu hiệu nhận dạng cho bộ xử lý CancelEventArgs là hàm mang tên

```
void FN(object sender,  
       System.ComponentModel.CancelEventArgs
```

Ví dụ dưới đây sẽ cài đặt cơ chế xử lý sự kiện khi nút nhấn trên được kích:

Ví dụ 3.2-2 fileopenevent.cs

```
using System;  
using System.Drawing;  
using System.ComponentModel;  
using System.Windows.Forms;  
using System.IO;
```

```
namespace Sams  
{
```



```

class fileopenapp : System.Windows.Forms.Form
{
    MainMenuItem m_menu;
    MenuItem m_filemenu;
    OpenFileDialog openfiledlg1;

    void OnOpenFile(Object sender, EventArgs e)
    {
        openfiledlg1.Filter = "C# files (*.cs)|*.
này là | *.cs|Bitmap files (*.bmp)|*.bmp";
        openfiledlg1.FilterIndex = 1;
        openfiledlg1.ShowDialog();
    }

    void OnExit(Object sender, EventArgs e)
    {
        Dispose();
        exit();
    }

    fileopenapp()
    {
        m_menu = new MainMenu();
        m_filemenu=new MenuItem("&File");
        m_menu.MenuItems.Add(m_filemenu);
        MenuItem t;
        t=new MenuItem("&Open");
        t.Click += new EventHandler(OnOpenFile);
        m_filemenu.MenuItems.Add(t);

        t=new MenuItem("-");
        m_filemenu.MenuItems.Add(t);

        t=new MenuItem("E&xit");
        t.Click += new EventHandler(OnExit);
        m_filemenu.MenuItems.Add(t);
    }

    this.Menu = m_menu;
    openfiledlg1 = new OpenFileDialog();
    openfiledlg1.FileOk += new
        CancelEventHandler(OnFileOpenOK);
}

void OnFileOpenOK(Object sender, CancelEventArgs e)
{
}

```

```

    OpenFileDialog dlg = (OpenFileDialog)sender;

    Stream FileStream;
    if((FileStream = dlg.OpenFile())!=null)
    {
        //Thực hiện xử lý dữ liệu ở đây
        FileStream.Close(); //Cuối cùng là đóng luồng
    }
}

public static void Main()
{
    Application.Run(new fileopenapp());
}
}

// end of Sams namespace

```

Sự kiện trong ví dụ trên đây chỉ phát sinh nếu nút nhấn OK của hộp thoại được kích. Ở đây cũng có thể phát sinh sự kiện HelpRequested nếu nút nhấn Help trong hộp thoại được chọn.

Hộp thoại file có thể được cấu hình sao cho chỉ hiện thị danh sách một file đã qua điều kiện lọc hoặc thậm chí có thể dùng mở file và trả về cho bạn luồng (stream) đọc ghi tên file. Ví dụ, để lọc ra các file dựa trên kiểu file ví dụ đã gọi các lệnh sau:

```

openfiledlg1.Filter = "C# files (*.cs)|>"+
    "*.cs|Bitmap files (*.bmp)|*.bmp";
openfiledlg1.FilterIndex = 1;
openfiledlg1.ShowDialog();

```

Lớp SaveFileDialog cũng hoạt động theo cách tương tự bởi vì tương tự như OpenFileDialog nó thừa hưởng và kế thừa các tính năng từ lớp OpenFileDialog. Nếu như bạn sử dụng hàm OpenFile(), file được chọn sẽ được tạo ra hoặc mở cho mục đích ghi file.

Lưu ý: OpenFileDialog, SaveFileDialog, OpenFileDialog hỗ trợ thuộc tính phép bạn thay đổi cách file được hiển thị, đánh dấu hoặc chọn. Hãy tham khảo chi tiết trong tài liệu .NET SDK.

Hộp thoại chọn màu – ColorDialog

Chọn màu là công việc rất thường hay làm trong môi trường đồ họa Windows. Rất nhiều chương trình đã cố gắng viết riêng cho mình một bộ màu vì lý do tạo tính cách riêng và nổi bật cho chương trình. Tuy nhiên, đơn



và hiệu quả nhất là bạn hay sử dụng lớp ColorDialog. Ngoài ưu điểm dễ dùng ra, hộp thoại chọn màu là hộp thoại chuẩn dễ dùng và quen thuộc đối với hầu hết người dùng ứng dụng Windows.

Gọi hiển thị hộp thoại ColorDialog theo cùng cách bạn đã gọi OpenFileDialog. Trước hết tạo thể hiện của đối tượng từ lớp ColorDialog, sau đó thì gọi phương thức ShowDialog để hiển thị.

Hình 3.2-2 là kết quả hiển thị của hộp thoại chọn màu ColorDialog theo nhiều chế độ khác nhau.

Lớp ColorDialog trả về thông tin màu ở dạng đối tượng Color. Thông tin này bao gồm trị alpha (hay còn gọi là độ trong suốt), trị đỏ (Red), xanh lá cây (Green) và xanh da trời (Blue) đây là các giá trị màu của hệ RGB. bạn cũng có thể chuyển đổi chúng sang hệ màu HUE diễu tả bằng độ màu, độ sáng và độ bão hòa.

ColorDialog còn cung cấp một thuộc tính mang tên AllowFullOpen cho phép hiển thị nút nhấn tùy biến định nghĩa màu. Nếu bạn sử dụng thuộc tính FullOpen, hộp thoại chọn màu sẽ hiển thị ở chế độ phức tạp và đầy đủ nhất để bạn tự định nghĩa giá trị màu cần chọn. Thuộc tính CustomColors là một mảng chứa các giá trị màu có thể được chọn ra để điền vào ô màu tùy biến.

Dưới đây là chương trình ví dụ cho thấy cách sử dụng hộp thoại chọn màu ColorDialog:

Ví dụ 3.2-3 colordlgs.cs

```
using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;

namespace Sams {
    class ColorStretcher : System.Windows.Forms.Form
    {
        void OnFull(Object sender, EventArgs e)
        {
            ColorDialog dlg=new ColorDialog();
            dlg.FullOpen = true;
            dlg.ShowDialog();
        }
    }
}
```

```
}

void OnNoCustom(Object sender, EventArgs e)
{
    ColorDialog dlg=new ColorDialog();
    dlg.AllowFullOpen = false;

    dlg.ShowDialog();
}

void OnNormal(Object sender, EventArgs e)
{
    ColorDialog dlg=new ColorDialog();
    dlg.Color = Color.PaleGoldenrod;
    dlg.ShowDialog();
}

void OnWithColours(Object sender, EventArgs e)
{
    ColorDialog dlg=new ColorDialog();
    dlg.FullOpen = true;
    //Câu lệnh này định nghĩa 5 màu tùy biến đầu tiên thiết lập
    //như là các giá trị int. ví dụ 0xAARRGGBB trong đó AA là
    //độ trong suốt, RR là trị số GG là trị xanh dương còn BB là
    //trị xanh lá cây. Các trị này biểu diễn ở dạng số hex.

    dlg.CustomColors = new int[5]{
        0x00ff8040, 0x00c256fe,
        0x00aa2005, 0x0004f002, 0x002194b5};

    dlg.ShowDialog();
}

ColorStretcher()
{
    // Trước tiên là menu để lựa chọn

    MainMenu m=new MainMenu();
}
```

```

MenuItem top,temp;

top=new MenuItem("ColorDialog");
m.MenuItems.Add(top);

temp=new MenuItem("Full");
temp.Click+=new EventHandler(OnFull);
top.MenuItems.Add(temp);

temp=new MenuItem("No custom");
temp.Click+=new EventHandler(OnNoCustom);
top.MenuItems.Add(temp);

temp=new MenuItem("With Colours");
temp.Click+=new EventHandler(OnWithColours);
top.MenuItems.Add(temp);

temp=new MenuItem("Normal");
temp.Click+=new EventHandler(OnNormal);
top.MenuItems.Add(temp);

this.Menu = m;

}

public static void Main()
{
    Application.Run(new ColorStretcher());
}
}
}

```

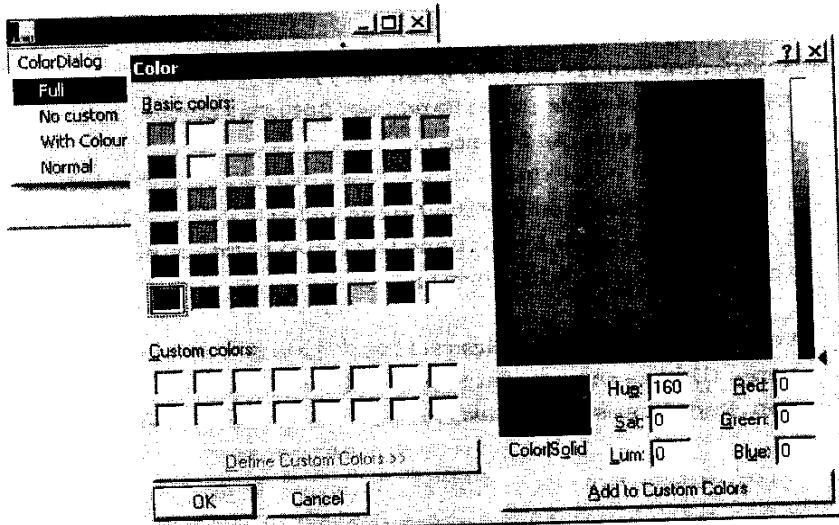
Lưu ý các giá trị màu tùy biến được định nghĩa như là trị nguyên integer trong bộ xử lý sự kiện OnWithColours

```

ColorDialog dlg=new ColorDialog();
dlg.FullOpen = true;
dlg.CustomColors = new int[5]{
    0x00ff8040, 0x00c256fe,
    0x00aa2005, 0x0004f002, 0x002194b5};

```

Để lấy về giá trị màu được chọn từ hộp thoại ColorDialog, đơn giản bạn chỉ cần đọc nội dung của thuộc tính Color. Thiết lập giá trị cho thuộc tính này trước khi gọi phương thức ShowDialog() sẽ áp đặt màu được chọn vào lúc hộp thoại hiển thị. Điều này sẽ rất thuận lợi cho việc thay đổi màu đã được thiết lập sẵn trong ứng dụng trước đó. Chúng ta thực hiện điều này trong bộ xử lý sự kiện OnNormal. Để thấy được màu đã chọn, bạn cần kích chuột vào nút nhấn Define Custom Color. Bảng màu sẽ xuất hiện bên tay phải còn màu chọn sẽ là màu tương ứng với giá trị mà bạn chỉ định.



Hình 3.2-2 Kết quả hiển thị của chương trình Colordlgs.cs

Hộp thoại chọn font – FontDialog

Trong Windows chọn Font chữ cho văn bản (text) rất quan trọng và thường xuyên được sử dụng trong rất nhiều chương trình. Windows cho phép bạn chọn tên font (Times Roman, Arial, Tahoma ...), cỡ chữ, kiểu chữ (dậm, nghiêng, gạch chéo). Lớp FontDialog một lần nữa giúp lập trình viên trong môi trường .NET đơn giản hóa thao tác xử lý font.

Kỹ thuật để gọi và hiển thị hộp thoại font hoàn toàn giống với các hộp thoại trước đây. Thuộc tính của hộp thoại và dữ liệu trả về sẽ duy nhất và chỉ định rõ loại font cũng như những thông tin liên quan đến font chữ mà người dùng đã chọn.

Ví dụ dưới đây là một ứng dụng đơn giản với một nút nhấn và một nút nhấn cho phép chọn hiển thị hộp thoại font. Hộp thoại font bao gồm nó chứa rất nhiều tùy chọn, bạn có thể thay đổi và thiết lập những tùy chọn này thông qua thuộc tính. Các tùy chọn này còn qui định cách làm việc của hộp thoại font. Trong ví dụ



của ta, nút nhấn Apply trong hộp thoại được thiết lập và gắn với sự kiện cập nhật font chữ cho nhãn khi nút được kích hoạt.

Ví dụ 3.2-4 fontdlg.cs

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace Sams {

    class FontPicker : System.Windows.Forms.Form
    {

        Button b;
        Label l;

        void OnApply(Object sender, System.EventArgs e)
        {
            FontDialog dlg = (FontDialog)sender;
            l.Font=dlg.Font;
        }

        void OnClickedb(Object sender, EventArgs e)
        {
            // Tạo hộp thoại Font
            FontDialog dlg=new FontDialog();

            //Khởi tạo hộp thoại với thông tin font có sẵn trong đối tượng nhãn
            dlg.Font = l.Font;

            //Đặt thuộc tính cho phép hiển thị nút "Apply"
            dlg.ShowApply = true;

            //Gắn phương thức xử lý sự kiện OnApply cho nút nhấn
            dlg.Apply += new EventHandler(OnApply);

            if(dlg.ShowDialog() != DialogResult.Cancel)
            {
                l.Font = dlg.Font;
            }
        }

        public FontPicker()
        {
```

```

        this.Size=new Size(416,320);

        l=new Label();
        l.Location = new Point(8,8);
        l.Size = new Size(400,200);
        l.Text = "0 1 2 3 4 5 6 7 8 9
                  \nabcdefghijklmnopqrstuvwxyz"+
                  "\nABCDEFGHIJKLMNOPQRSTUVWXYZ";
        l.Font = new Font("Microsoft Sans Serif", 18f);
        this.Controls.Add(l);

        b=new Button();
        b.Text = "Choose Font";
        b.Click += new EventHandler(OnClickedb);
        b.Location = new Point(8,250);
        b.Size = new Size(400,32);
        this.Controls.Add(b);

    }

    static void Main()
    {
        Application.Run(new FontPicker());
    }
}
}

```

Bạn biên dịch chương trình trên từ dòng lệnh như sau

```
csc /t:winexe fontdlg.cs
```

2.2. Hộp thoại Print và Print Preview

Vẽ nội dung của một tài liệu lên màn hình hoặc ra máy in đòi hỏi bạn phải có kiến thức về đồ họa. Kỹ thuật đồ họa sẽ được chúng ta nghiên cứu chi tiết trong chương sau. Chúng ta chưa cần bàn sâu đến vấn đề đồ họa trong chương này. Cần bạn hiểu được cơ chế in và xem trước khi in (print-preview) được thực hiện như thế nào qua ứng dụng minh họa sau đây:

Ví dụ 3.2-5 printpreview.cs

```

namespace Sams
{
    using System;
    using System.Drawing;
    using System.Drawing.Printing;
    using System.Collections;
    using System.ComponentModel;
    using System.Windows.Forms;

```

```
// <summary>
// ppView là thành phần điều khiển đơn giản của Windows cung cấp
// những hàm vẽ thông thường cho phép vẽ ra màn hình và máy in
// hoặc thực hiện chức năng print-preview
// </summary>
public class ppView : System.Windows.Forms.Panel
{
    private ArrayList points;

    Point mousePoint;

    void OnClick(Object sender, MouseEventArgs e)
    {
        if(points == null)
        {
            points=new ArrayList();
        }

        points.Add(mousePoint);
        Invalidate();
    }

    void OnMouseMove(Object sender, MouseEventArgs e)
    {
        mousePoint = new Point(e.X,e.Y);
    }

    public ppView()
    {
        this.MouseDown+=new MouseEventHandler(OnClick);
        this.MouseMove+=new
                        MouseEventHandler(OnMouseMove);
        this.BackColor=Color.White;
    }

    private void DrawSmiley(Point pt, Graphics g)
    {
        g.FillEllipse(Brushes.Black,pt.X-52,
                      pt.Y-52,104,104);
        g.FillEllipse(Brushes.Yellow,pt.X-50,
                      pt.Y-50,100,100);
        g.FillEllipse(Brushes.Black,pt.X-30,
                      pt.Y-10,60,40);
        g.FillEllipse(Brushes.Yellow,pt.X-35,
```

```

        pt.Y-10,70,35);
g.FillEllipse(Brushes.Black,pt.X-25,
    pt.Y-15,10,10);
g.FillEllipse(Brushes.Black,pt.X+10,
    pt.Y-15,10,10);
}

private void DoDraw(Graphics g)
{
    if(points == null)
    {
        return;
    }
    foreach(Point p in points)
    {
        DrawSmiley(p,g);
    }
}
protected override void OnPaint(PaintEventArgs e)
{
    if(points == null)
    {
        return;
    }
    DoDraw(e.Graphics);
}
public void OnPrintPage(Object sender,
    PrintPageEventArgs e)
{
    if(points == null)
    {
        return;
    }
    DoDraw(e.Graphics);
}
}

public class MDIChildForm : System.Windows.Forms.Form
{
    private System.ComponentModel.Container
        components;
    private ppView vw;
}

```

```
// Phương thức cho phép truy cập biến của lớp
public ppView View {
    get
    {
        return vw;
    }
}

public MDIChildForm()
{
    InitializeComponent();

    vw=new ppView();
    vw.Location = new Point(3,3);
    vw.Size = new Size(this.Size.Width-
                        6,this.Size.Height-6);
    vw.Anchor=AnchorStyles.Left|
                AnchorStyles.Top|
                AnchorStyles.Right|
                AnchorStyles.Bottom;

    this.Controls.Add(vw);

    //
    // TODO: Thêm vào bất kỳ đoạn mã nào của bạn sau phần
    //      InitializeComponent
    //
}

/// <summary>
/// Giải phóng tài nguyên
/// </summary>
public override void Dispose()
{
    base.Dispose();
    components.Dispose();
}

/// <summary>
/// Phương thức hỗ trợ cho Designer – Đừng thay đổi phương thức
/// này trong các cửa sổ soạn thảo code của môi trường phát triển
/// </summary>
```



```
private void InitializeComponent()
{
    this.components = new
        System.ComponentModel.Container();
    this.Text = "MDIChildForm";
    this.AutoScaleBaseSize =
        new System.Drawing.Size(5, 13);
    this.ClientSize = new System.Drawing.Size(328,
                                              277);
}

/// <summary>
/// Chương trình chính
/// </summary>
public class MainApp : System.Windows.Forms.Form
{

    private System.ComponentModel.Container
        components;
    private System.Windows.Forms.MenuItem FileExit;
    private System.Windows.Forms.MenuItem
        FilePrintPreview;
    private System.Windows.Forms.MenuItem FilePrint;
    private System.Windows.Forms.MenuItem FileNew;
    private System.Windows.Forms.MenuItem menuItem1;
    private System.Windows.Forms.MainMenu mainMenu1;

    public MainApp()
    {
        //
        // Hỗ trợ cho Windows Form Designer
        //
        InitializeComponent();

        //
        //Bạn đặt thêm các đoạn mã khởi tạo của mình ở đây
        //
    }

    /// <summary>
    /// Xóa bỏ tài nguyên đã sử dụng
    /// </summary>
    public override void Dispose()
```



```
{  
    base.Dispose();  
    components.Dispose();  
}  
  
private void InitializeComponent()  
{  
    this.components = new  
        System.ComponentModel.Container();  
    this.mainMenu1 = new  
        System.Windows.Forms.MainMenu();  
    this.FilePrintPreview = new  
        System.Windows.Forms.MenuItem();  
    this.FileExit = new  
        System.Windows.Forms.MenuItem();  
    this.menuItem1 = new  
        System.Windows.Forms.MenuItem();  
    this.FileNew = new  
        System.Windows.Forms.MenuItem();  
    this.FilePrint = new  
        System.Windows.Forms.MenuItem();  
    mainMenu1.MenuItems.AddRange(  
        new System.Windows.Forms.MenuItem[1]  
            {this.menuItem1});  
    FilePrintPreview.Text = "P&review...";  
    FilePrintPreview.Index = 2;  
    FilePrintPreview.Click +=  
        new System.EventHandler  
            (this.FilePrintPreview_Click);  
    FileExit.Text = "&Exit";  
    FileExit.Index = 3;  
    FileExit.Click += new  
        System.EventHandler (this.FileExit_Click);  
    menuItem1.Text = "&File";  
    menuItem1.Index = 0;  
    menuItem1.MenuItems.AddRange(new  
        System.Windows.Forms.MenuItem[4]  
    {this.FileNew,  
        this.FilePrint,  
        this.FilePrintPreview,  
        this.FileExit});  
    FileNew.Text = "&New";  
    FileNew.Index = 0;  
    FileNew.Click += new  
        System.EventHandler (this.FileNew_Click);  
    FilePrint.Text = "&Print...";  
    FilePrint.Index = 1;
```

```
FilePrint.Click += new
    System.EventHandler (this.FilePrint_Click);
this.Text = "MainApp";
this.AutoScaleBaseSize = new
    System.Drawing.Size (5, 13);
this.IsMdiContainer = true;
this.Menu = this.mainMenu1;
this.ClientSize = new
    System.Drawing.Size (368, 289);
}

protected void FileExit_Click (object sender,
                               System.EventArgs e)
{
    Application.Exit();
}

protected void FilePrintPreview_Click (object
                                       sender, System.EventArgs e)
{
    PrintPreviewDialog d=new PrintPreviewDialog();
    PrintDocument doc = new PrintDocument();

    MDIChildForm f =
        (MDIChildForm)this.ActiveMdiChild;
    ppView vw = f.View;
    doc.PrintPage += new
        PrintPageEventHandler (vw.OnPrintPage);

    d.Document=doc;

    d.ShowDialog();

    doc.PrintPage -= new
        PrintPageEventHandler (vw.OnPrintPage);
}

protected void FilePrint_Click (object sender,
                               System.EventArgs e)
{
    PrintDialog dlg = new PrintDialog();
    PrintDocument doc = new PrintDocument();

    MDIChildForm f =
        (MDIChildForm)this.ActiveMdiChild;
    ppView vw = f.View;
    doc.PrintPage += new
        PrintPageEventHandler (vw.OnPrintPage);
```

```

        dlg.Document=doc;

        dlg.ShowDialog();

        doc.PrintPage -= new
            PrintPageEventHandler(vw.OnPrintPage);
    }

protected void FileNew_Click (object sender,
                            System.EventArgs e)
{
    MDIChildForm f = new MDIChildForm();
    f.MdiParent=this;
    f.Show();
}

/// <summary>
/// Chương trình chính
/// </summary>
public static void Main(string[] args)
{
    Application.Run(new MainApp());
}
}
}

```

Chương trình của ta có 3 thành phần. Thành phần form, gọi là MainApp. Đây là loại form theo giao diện đa tài liệu (MDI – hay Multiple Document Interface) truyền thống của Windows. Cửa sổ MDI bao gồm nhiều cửa sổ con thuộc lớp đối tượng MDIChildForm. Những đối tượng cửa sổ con này lại chứa trong nó một thành phần đối tượng khác gọi là ppView.

Đối tượng ppView thực hiện tất cả các công việc như theo dõi trạng thái chuột, thêm các điểm và danh sách mảng, vẽ một khuôn mặt cười tại mỗi điểm lưu trữ. ppView cũng sử dụng một hàm vẽ đơn giản nhận đối tượng Graphics làm đối số.

Cơ chế in (print) và xem trước khi in (print-preview) sử dụng PrintDocument để xử lý sự kiện phát sinh cho mỗi trang cần xem hoặc in. Để có thể sử dụng được các tiện ích print và print-preview, OnPaint và OnPrintPage đã gọi một thủ tục vẽ đơn giản đó là DoDraw(). Thủ tục vẽ này sẽ chịu trách nhiệm vẽ ra màn hình hay thiết bị máy in tùy theo ngữ cảnh được gọi. DoDraw() nhận đối số là kiểu đối tượng Graphics cho biết ngữ cảnh vẽ của thiết bị. Graphics đủ thông minh để nhận ra độ phân giải cần thiết cần sử dụng để cung cấp hình ảnh phù hợp trên thiết bị vẽ ra.

Nếu bạn thích thú với đồ họa thì những hàm sau trong chương trình sẽ giúp bạn vẽ ra khuôn mặt cười xinh xắn:

```
private void DrawSmiley(Point pt, Graphics g)
{
    g.FillEllipse(Brushes.Black, pt.X-52,
                  pt.Y-52, 104, 104);
    g.FillEllipse(Brushes.Yellow, pt.X-50,
                  pt.Y-50, 100, 100);
    g.FillEllipse(Brushes.Black, pt.X-30,
                  pt.Y-10, 60, 40);
    g.FillEllipse(Brushes.Yellow, pt.X-35,
                  pt.Y-10, 70, 35);
    g.FillEllipse(Brushes.Black, pt.X-25,
                  pt.Y-15, 10, 10);
    g.FillEllipse(Brushes.Black, pt.X+10,
                  pt.Y-15, 10, 10);
}
```

3. TẠO HỘP THOẠI

Trong môi trường .NET, việc diễn dịch và xem khi nào form nên là hộp thoại khi nào không còn tùy thuộc vào mục đích mà lập trình viên lựa chọn. Nên đã dễ dàng trước đây, chúng ta có hai loại hộp thoại: hộp thoại modal yêu cầu người dùng phải kết thúc nhập liệu trước khi quay trở về chương trình chính. Và hộp thoại modeless cho phép tương tác song song với chương trình. Giữa form và hộp thoại bạn nên quyết định cần tập trung phần nào dành cho form và phần nào dành cho hộp thoại. Nếu hộp thoại quá phức tạp nó sẽ nên không hữu dụng dành mất chức năng vốn có của hộp thoại.

3.1. Hộp thoại modal và modeless

Hộp thoại được tạo ra theo cùng cách tạo form. Lớp Form chứa một thuộc tính mang tên Modal, nếu bạn đặt thuộc tính này là true thì Form tạo ra sẽ là hộp thoại dạng modal. Phương thức ShowDialog() của Form sẽ đặt giá trị cho thuộc tính này hộ bạn và cho phép bạn hiển thị form ở dạng modal. Ví dụ dưới đây chương trình cho phép hiển thị cửa sổ form ở dạng hộp thoại ở cả hai chia sẻ modal và modeless.

Ví dụ 3.2-6 Models.cs

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.Collections;
```



```
class ADialog : Form
{
    private Button ok, can;

    public bool Modeless
    {
        set
        {
            if(value)
            {
                ok.Click += new EventHandler(OnCloseModeless);
                can.Click += new
                    EventHandler(OnCloseModeless);
            }
        }
    }

    void OnCloseModeless(Object sender, EventArgs e)
    {
        this.Close();
    }

    public ADialog()
    {
        this.Location = new Point(100,100);
        this.Size=new Size(200,100);
        this.FormBorderStyle =
            FormBorderStyle.FixedDialog;
        ok = new Button();
        ok.Text = "OK";
        ok.DialogResult = DialogResult.OK;
        ok.Location = new Point(20,10);
        ok.Size = new Size(80,25);
        this.Controls.Add(ok);

        can = new Button();
        can.Text = "Cancel";
        can.DialogResult = DialogResult.OK;
        can.Location = new Point(110,10);
        can.Size = new Size(80,25);
        this.Controls.Add(can);
    }
}
```

```
class AnApp : Form
{
    void OnModeless(Object sender, EventArgs e)
    {
        ADialog dlg = new ADialog();
        dlg.Text = "Modeless";
        dlg.Modeless = true;
        dlg.Show();
    }

    void OnModal(Object sender, EventArgs e)
    {
        ADialog dlg = new ADialog();
        dlg.Text = "Modal";
        dlg.ShowDialog();
    }

    public AnApp()
    {
        this.Size=new Size(400,100);
        this.FormBorderStyle =
            FormBorderStyle.FixedSingle;
        this.Text = "Dialog Mode Tester";

        Button modal = new Button();
        modal.Text = "New Modal dialog";
        modal.Location = new Point(10,10);
        modal.Size = new Size(180,25);
        modal.Click += new EventHandler(OnModal);
        this.Controls.Add(modal);

        Button modeless = new Button();
        modeless.Text = "New Modeless dialog";
        modeless.Location = new Point(210,10);
        modeless.Size = new Size(180,25);
        modeless.Click += new EventHandler(OnModeless);
        this.Controls.Add(modeless);
    }

    static void Main()
    {
        Application.Run( new AnApp());
    }
}
```

Trong ví dụ trên, chương trình chính là một form với hai nút nhấn. Nút nhấn dùng gọi và hiển thị hộp thoại dạng modeless. Nút nhấn còn lại gọi



thị hộp thoại dạng modal. Khi kích nút nhấn hiển thị hộp thoại dạng modal bạn chỉ có thể hiển thị được một. Trong khi đó, bạn có thể kích chuột vào nút nhấn hiển thị nhiều lần cửa sổ hộp thoại modeless khác nhau. Đóng cửa sổ của chương trình chính cũng khiến cho tất cả các cửa sổ modeless đang mở đóng theo.

3.2. Chuyển dữ liệu giữa các thành phần trong hộp thoại

Đơn giản nhất hộp thoại có thể chỉ cần chuyển dữ liệu về sự kiện cho biết nút nhấn nào đã được kích hoạt để đóng và thoát khỏi hộp thoại. Các hộp thoại phức tạp các thể tạo sẵn những thuộc tính cho phép lập trình viên dùng nó để lưu hoặc truy xuất các thông tin liên quan đến các thành phần điều khiển trên hộp thoại. Ví dụ form hiển thị bởi phương thức ShowDialog có thể được đóng lại bằng cách thiết lập giá trị cho thuộc tính DialogResult. Giá trị này sẽ được trả về cho đoạn mã đã gọi hiển thị hộp thoại.

Dưới đây là ví dụ minh họa cho thấy các trao đổi dữ liệu của hộp thoại với chương trình hay cửa sổ form chính:

Ví dụ 3.2-7 simpledialog.cs

```
using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;

namespace Sams {
    class SimpleDialog : Form
    {
        public SimpleDialog()
        {
            // Tạo hai nút nhấn
            Button OkButton=new Button();
            OkButton.Text = "Ok";
            OkButton.DialogResult = DialogResult.OK;
            OkButton.Location = new Point(8,20);
            OkButton.Size = new Size(50,24);
            this.Controls.Add(OkButton);

            Button CancelButton=new Button();
            CancelButton.Text = "Cancel";
            CancelButton.DialogResult = DialogResult.Cancel;
            CancelButton.Location = new Point(64,20);
            CancelButton.Size = new Size(50,24);
            this.Controls.Add(CancelButton);

            this.Text="Dialog";
            this.Size = new Size(130,90);
        }
    }
}
```

```
        this.FormBorderStyle =
            FormBorderStyle.FixedSingle;
        this.StartPosition =
            FormStartPosition.CenterParent;
        this.ControlBox = false;
    }
}

class AnApp : Form
{
    void OnTest(Object sender, EventArgs e)
    {
        SimpleDialog dlg = new SimpleDialog();
        dlg.Owner = this;

        if(dlg.ShowDialog() == DialogResult.OK)
            MessageBox.Show("You clicked Ok");
        else
            MessageBox.Show("You clicked Cancel");
    }

    public AnApp()
    {
        this.Menu = new MainMenu();
        this.Menu.MenuItems.Add(new MenuItem("Dialog"));
        this.Menu.MenuItems[0].MenuItems.Add(new
            MenuItem("Test"));
        this.Menu.MenuItems[0].MenuItems[0].Click += new
            EventHandler(OnTest);
    }

    static void Main()
    {
        Application.Run(new AnApp());
    }
}
}
```

Hộp thoại cũng như form có thể chứa mọi thành phần điều khiển của Windows Forms. Hộp thoại có thể dùng để trả về các giá trị đơn giản chứa thông tin của checkbox, nút radiolo, ô textbox ... Để thực hiện điều này, bạn cần có khả năng trả về những thông tin phức tạp hơn là giá trị đơn giản như trong ví dụ trên.

Dữ liệu trong hộp thoại các thể được truy xuất thông qua các biến thành viên chung (public member) hoặc thông qua các phương thức và thuộc tính thêm vào để dùng cho công việc đặc biệt của bạn. Nếu bạn là tín đồ lập trình MFC trên



C++ bạn cần chuyển thói quen truy xuất dữ liệu bằng các biến thành viên trở thành truy xuất thông qua thuộc tính. Truy xuất thuộc tính hay phương thức đóng vai trò rất lớn trong tương tác của các đối tượng .NET.

Tương tác giữa người dùng và hộp thoại có thể rất phức tạp. Một nút nhấn được kích hoạt, ô check box được chọn hoặc các hành động khác có thể sinh ra sự kiện được bẫy bởi đoạn mã do hộp thoại của bạn cài đặt và thực thi trước cả khi nút nhấn OK được nhấn. Thiết kế một hộp thoại có thể hoàn toàn giống với cách thiết kế form nơi sẽ gọi hộp thoại.

3.3. Kiểm tra hợp lệ (validation)

Một vài thành phần điều khiển, ví dụ như điều khiển NumericUpDown, có những ứng xử đặc thù cho việc kiểm tra hợp lệ. Điều khiển này chứa thuộc tính Minimum và Maximum có thể được dùng để định nghĩa phạm vi của số thể hiện trong điều khiển. Thành phần điều khiển cũng sẽ phát sinh một sự kiện khi nó đang kiểm tra chính dữ liệu của chính bản thân mình và sau khi dữ liệu đã kiểm tra xong. Bạn có thể bẫy các sự kiện này và chúng thực sự dễ dàng hơn trong Windows Forms so với MFC. Sự kiện kiểm tra dữ liệu được kế thừa từ lớp System.Windows.Forms.Control và sẽ phát sinh bất kỳ Khi nào điều khiển mất quyền kiểm soát (lost focus). Điều khiển sẽ mất quyền kiểm soát khi người dùng chuyển qua sử dụng điều khiển khác (chẳng hạn như khi nhấn phím Tab di chuyển giữa các thành phần điều khiển), hoặc khi bạn kích chuột vào nút nhấn để đặt giá trị cho thuộc tính DialogResult và đóng hộp thoại. Nếu điều kiện kiểm tra dữ liệu nhập vào không hợp lệ, hộp thoại sẽ bị ngăn không cho đóng lại.

Trong chương trình ví dụ dưới đây, hộp thoại sẽ không đóng lại trừ khi việc kiểm tra giá trị số chưa trong thành phần điều khiển là hợp lệ

Ví dụ 3.2-8 dialogValid.cs

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.IO;
using System.Text;

namespace Sams {
    class DialogValid : System.Windows.Forms.Form
    {
        private Button okButton;
        private Button cancelButton;
        private NumericUpDown num;
        public decimal Num {
```

```
get { return num.Value; }
set { num.Value = value; }
}

void OnValidating(Object sender, CancelEventArgs e)
{
    MessageBox.Show("NumericUpDown is validating");
}

void OnValid(Object sender, EventArgs e)
{
    MessageBox.Show("NumericUpDown is valid");
}

public DialogValid()
{
    Size = new Size(400,100);
    FormBorderStyle = FormBorderStyle.FixedSingle;
    Text = "Dialog test";

    //đặt các nút nhấn lên Form
    okButton = new Button();
    okButton.DialogResult = DialogResult.OK;
    okButton.Location = new Point(20,28);
    okButton.Size = new Size(80,25);
    okButton.Text = "OK";
    Controls.Add(okButton);

    cancelButton = new Button();
    cancelButton.Location = new Point(300,28);
    cancelButton.Size = new Size(80,25);
    cancelButton.Text = "Cancel";
    cancelButton.DialogResult = DialogResult.Cancel;
    Controls.Add(cancelButton);

    // đặt nhãn lên form
    Label l = new Label();
    l.Text = "NumericUpDown";
    l.Location = new Point(20,5);
    l.Size = new Size(120,25);
    Controls.Add(l);

    // đặt thành phần Numeric lên form
    num = new NumericUpDown();
    num.Location = new Point(140,5);
    num.Size = new Size(80,25);
```

```
num.Minimum = (decimal)10.0;
num.Maximum = (decimal)100.0;
num.Value = (decimal)10.0;

//Bộ xử lý sự kiện cho thấy quá trình kiểm tra dữ liệu
num.Validating+=new
    CancelEventHandler(OnValidating);
num.Validated+=new EventHandler(OnValid);

Controls.Add(num);

}

}

class ddApp : System.Windows.Forms.Form
{

void OnExit(Object sender, EventArgs e)
{
    Application.Exit();
}

void OnDialogTest(Object sender, EventArgs e)
{
    DialogValid dlg = new DialogValid();

    DialogResult r=dlg.ShowDialog();

    StringWriter sw=new StringWriter(new
        StringBuilder()));

    sw.WriteLine("Dialog return value = {0}"+
"\nNumericUpDown = {1}",r,dlg.Num);

    MessageBox.Show(sw.ToString());
}

public ddApp()
{
    MainMenu mm=new MainMenu();
    mm.MenuItems.Add(new MenuItem("&Dialog"));
    mm.MenuItems[0].MenuItems.Add(new
        MenuItem("&Test",
            new EventHandler(OnDialogTest)));
    mm.MenuItems[0].MenuItems.Add(new MenuItem("-"));
}
```

```

        mm.MenuItems[0].MenuItems.Add(new
            MenuItem("E&xit",
            new EventHandler(OnExit)));
    Menu = mm;
}

static void Main()
{
    Application.Run(new ddApp());
}
}
}

```

Khi bạn tự xây dựng một thành phần điều khiển, bạn có thể sử dụng thuộc tính CauseValidation để yêu cầu sự kiện Validating và Validated phát sinh khi điều khiển của bạn mất quyền kiểm soát.

4. SỬ DỤNG CÁC ĐIỀU KHIỂN

Cho đến thời điểm này chúng ta đã sử dụng và làm quen với hai thành phần điều khiển đó là nút nhấn (Button) và NumericUpdown. Windows Forms cung cấp rất nhiều điều khiển hữu ích khác, chúng sẽ được chúng ta xem qua chi tiết sau đây.

4.1. Checkbox và nút Radio

Các ứng xử của Checkbox và nút radio cơ bản rất giống với người anh em của nó trong MFC. Mặc dù vậy Windows Forms không sử dụng cơ chế DDX trao đổi dữ liệu giữa các nút điều khiển như trong MFC.

Checkbox có thể được đặt trực tiếp trên form hoặc trong một khung container hay một nhóm bao gồm nhiều nút. Tuy nhiên, bởi vì checkbox nói chung là phụ thuộc vào trạng thái của các nút khác trong nhóm cho nên xử lý check/toggle đối đơn giản.

Nút nhấn radio thường được kết hợp với những nút khác và thường chung trong một nhóm các nút quan hệ với nhau. Chọn nút này sẽ dẫn đến chọn các nút khác trong cùng nhóm.

Lấy về trạng thái của các nút này đơn giản chỉ là vấn đề đọc trực tiếp giá trị hiện hành của chúng hoặc kiểm tra các sự kiện kích chuột lên nút để theo dõi sự lựa chọn của người dùng.

Ví dụ sau đây bao gồm một số nút checkbox và radio minh họa những khác nhau để đọc trạng thái và bẫy sự kiện phát sinh từ chúng.

Ví dụ 3.2-9 dialogtest.cs

```
using System;
```

```
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.IO;
using System.Text;

namespace Sams {

    class DialogTest : System.Windows.Forms.Form
    {

        private Button okButton;
        private Button cancelButton;
        private CheckBox checkbox;
        private GroupBox radiogroup;
        private RadioButton radio1, radio2, radio3;

        public int Radio;

        public bool Check {
            get { return checkbox.Checked; }
            set { checkbox.Checked = value; }
        }

        void OnRadio(Object sender, EventArgs e)
        {
            int n=0;
            foreach(Object o in radiogroup.Controls)
            {
                if(o is RadioButton)
                {
                    RadioButton r=(RadioButton)o;
                    if(r.Checked)
                        Radio=n;
                    n++;
                }
            }
        }

        public DialogTest()
        {
            Size = new Size(400,300);
            FormBorderStyle = FormBorderStyle.FixedDialog;
            Text = "Dialog test";

            //đặt các nút nhấn lên form
            okButton = new Button();
        }
    }
}
```

```
okButton.DialogResult = DialogResult.OK;
okButton.Location = new Point(20, 230);
okButton.Size = new Size(80, 25);
okButton.Text = "OK";
Controls.Add(okButton);

cancelButton = new Button();
cancelButton.Location = new Point(300, 230);
cancelButton.Size = new Size(80, 25);
cancelButton.Text = "Cancel";
cancelButton.DialogResult = DialogResult.Cancel;
Controls.Add(cancelButton);

//đặt các nút nhấn check box lên form
checkbox = new CheckBox();
checkbox.Location = new Point(20, 30);
checkbox.Size = new Size(300, 25);
checkbox.Text = "CheckBox";
Controls.Add(checkbox);

//đặt các nút radio lên forms
//Các nút này được gom vào một nhóm GroupBox
radiogroup = new GroupBox();
radiogroup.Text = "Radio Buttons";
radiogroup.Location = new Point(10, 60);
radiogroup.Size = new Size(380, 110);
Controls.Add(radiogroup);

radio1 = new RadioButton();
radio1.Location = new Point(10, 15);

// Tạo độ tương đối trong nhóm
radio1.Size = new Size(360, 25);
radio1.Click += new EventHandler(OnRadio);
radio1.Text = "Radio Button #1";
radiogroup.Controls.Add(radio1);

radio2 = new RadioButton();
radio2.Location = new Point(10, 40);

radio2.Size = new Size(360, 25);
radio2.Click += new EventHandler(OnRadio);
radio2.Text = "Radio Button #2";
radiogroup.Controls.Add(radio2);
```



```
radio3 = new RadioButton();
radio3.Location = new Point(10,70);

radio3.Size = new Size(360,25);
radio3.Click += new EventHandler(OnRadio);
radio3.Text = "Radio Button #3";
radiogroup.Controls.Add(radio3);

}

}

class dtApp : System.Windows.Forms.Form
{

    void OnExit(Object sender, EventArgs e)
    {
        Application.Exit();
    }

    void OnDialogTest(Object sender, EventArgs e)
    {
        DialogTest dlg = new DialogTest();

        DialogResult r=dlg.ShowDialog();

        StringWriter sw=new StringWriter(new
                                         StringBuilder());

        sw.WriteLine("Dialog return value = {0}" +
                    "\nRadio Buttons = {1}\nCheck box = {2}",
                    r,dlg.Radio,dlg.Check);

        MessageBox.Show(sw.ToString());
    }

    public dtApp()
    {
        MainMenu mm=new MainMenu();
        mm.MenuItems.Add(new MenuItem("&Dialog"));
        mm.MenuItems[0].MenuItems.Add(new
                                      MenuItem("&Test",
                                              new EventHandler(OnDialogTest)));
        mm.MenuItems[0].MenuItems.Add(new MenuItem("-"))
        mm.MenuItems[0].MenuItems.Add(new
```

```

        MenuItem("E&xit",
        new EventHandler(OnExit)));
    Menu = mm;
}

static void Main()
{
    Application.Run(new dtApp());
}
}
}

```

Đọc kỹ đoạn mã trong ví dụ trên bạn sẽ thấy các nút radio thật ra là của thành phần điều khiển nhóm (groupbox) Nhóm được tạo ra sau đó là các radio thêm vào như sau:

```

radiogroup = new GroupBox();
radiogroup.Text = "Radio Buttons";
radiogroup.Location = new Point(10, 60);
radiogroup.Size = new Size(380, 110);
Controls.Add(radiogroup);

```

Tất cả các nút radio trong nhóm đều sử dụng cùng phương thức xử lý kiện kích chuột là OnRadio . Trong phương thức xử lý này, chúng ta duyệt qua cả các nút có trong nhóm và kiểm tra xem trạng thái nút nào đang được c. Tương ứng với nút được chọn chúng ta sẽ thiết lập các giá trị thích hợp cho biến thành viên.

```

void OnRadio(Object sender, EventArgs e)
{
    int n=0;
    foreach(Object o in radiogroup.Controls)
    {
        if(o is RadioButton)
        {
            RadioButton r=(RadioButton)o;
            if(r.Checked)
                Radio=n;
            n++;
        }
    }
}

```

Đây là một trong số những kỹ thuật mà bạn có thể sử dụng để đọc g của nút radio. Nếu muốn, bạn có thể thiết kế mỗi nút nhấn một phương thức riêng biệt và khi sự kiện nút được chọn phát sinh bạn thiết lập giá trị ch biến thành viên tương ứng với nút đó.



Trường hợp của nút radio, hộp thoại lưu dữ liệu trong các biến thành viên public có thể truy xuất trực tiếp từ bên ngoài bởi các đối tượng khác. Đối với checkbox, dữ liệu thể hiện lại là các thuộc tính truy xuất trực tiếp đến giá trị của checkbox.

4.2. Ô soạn thảo Edit

Windows Forms cung cấp cho bạn thành phần điều khiển là ô soạn thảo edit có khả năng dùng nhập liệu một dòng hoặc nhiều dòng dữ liệu. Bạn có thể dùng thành phần edit cho nhiều mục đích khác nhau. Ví dụ đối tượng edit có thể là như một cửa sổ soạn thảo tài liệu tựa notepad của Windows. Hoặc đơn giản đây là một khung nhỏ cho phép nhập vào mật khẩu (password) với các ký tự ẩn đại diện. Trong ứng dụng Windows, đối tượng edit hay textbox được sử dụng rỗng rãi cho mục đích nhập liệu. Ví dụ về đối tượng này được thể hiện trong ví dụ 3.2-10 ở phần sau.

4.3. ListBox

Đối tượng ListBox (hộp danh sách) trong Windows form cho phép hiển thị các mục chọn ở dạng danh sách dọc hoặc ngang. Bạn có thể chọn cùng lúc một hoặc nhiều mục chọn trong danh sách. Ví dụ về ListBox cũng sẽ được trình bày trong ví dụ 3.2-10.

4.4. TreeView

Đối tượng TreeView đã trở thành chuẩn dùng để hiển thị hoặc trình bày các cấu trúc dữ liệu cây và những dữ liệu dạng kế thừa. Ví dụ, cấu trúc hệ thống tập tin, thư mục trên đĩa, cấu trúc của các phần tử trong tài liệu XML ... là một cấu trúc cây thể hiện rất hiệu quả bằng TreeView. Windows Form hỗ trợ khả năng thực vụ và xử lý đối tượng TreeView rất phong phú. Trình Explorer chính là một TreeView hoàn chỉnh trong ứng dụng Windows.

TreeView bao gồm các nút (node). Mỗi nút có thể có một nhãn (label) kết hợp với nó cùng với các thuộc tính khác như hình ảnh biểu diễn trạng thái của nút. Nút có thể chứa các nút con lưu trong tập hợp nút biểu diễn bởi lớp Collection. Nút con bắn thân nó lại có thể chứa tập hợp các nút con khác và có thể mở rộng ra. Đơn giản bạn có thể hình dung cấu trúc cây tương tự như menu có các mục bao gồm mỗi mục chọn trong menu lại có các menu con (sub-menu) ...

TreeView hỗ trợ rất nhiều sự kiện kiểm tra và nhận biết sự thay đổi trạng thái của nút. Các sự kiện mang tên Beforexxxxx sẽ cho phép bạn bắn các tình trạng thay đổi trước khi diễn ra và được phép hủy bỏ hay cho phép sự kiện tiếp tục phát sinh. Các sự kiện mang tên Afterxxxxx dùng thông báo cho bạn những thay đổi lên nút đã được thực hiện hoàn tất.

Các thao tác xử lý nút trong TreeView và những sự kiện kết hợp với đối tượng này sẽ được trình bày chi tiết trong ví dụ 3.2-10 phần sau.

4.5. Bảng điều khiển Tab

Bảng điều khiển (Tab) có thể dùng để phân hộp thoại thành nhiều trang. Bảng điều khiển bao gồm thanh Tab bar cho phép người dùng chọn tab và một hộp các đối tượng TabPage dùng để chứa các thành phần điều khiển tương tự như và phân trang cho hộp thoại. Ví dụ 3.2-10 trong chương trình MungoTabApp cũng sẽ minh họa cách sử dụng thành phần điều khiển này.

4.6. Quản lý các thành phần điều khiển động

Như bạn đã thấy trong những ví dụ minh họa ở các chương trước, việc tìm vào, định vị và lập trình cho những thành phần điều khiển đều được thực hiện lúc chương trình thực thi (runtime) và không cần tham chiếu đến bất kỳ thành phần tài nguyên nào từ bên ngoài. Điều này cũng có nghĩa là sử dụng các thành phần điều khiển của Windows Forms rất dễ dàng trong việc tùy biến theo chọn của người dùng khi cần co giãn, định vị lại các thành phần điều khiển trên form hay hộp thoại. Ứng dụng MungoTabApp.cs trong ví dụ 3.2-10 dưới đây sẽ thấy một số đặc điểm thú vị trong việc điều khiển các thành phần giao diện đan xen với minh họa cách sử dụng những thành phần điều khiển đã nêu.

Lưu ý do mã nguồn chương trình khá dài nên chúng sẽ được cắt ra và giải thành từ đoạn. Khi viết mã bạn có thể gộp chúng lại thành một mã nguồn nhất.

Ví dụ 3.2-10 MungoTabApp.cs

```
namespace Sams
{
    using System;
    using System.ComponentModel;
    using System.Drawing;
    using System.Windows.Forms;
    using System.Text;

    /// <summary>
    /// Ứng dụng này sẽ trình diễn khả năng của các thành phần
    /// điều khiển trong Windows Form
    /// </summary>
    public class MungoTabApp : Form
    {
```

Tất cả các biến thành viên trong ứng dụng này đều là cục bộ (private) không truy xuất được từ bên ngoài. Đây cũng là thói quen tốt cho tính năng gói (encapsulation) của lớp và đối tượng. Trước hết chúng ta bắt đầu khai báo các đối tượng là thành phần điều khiển giao diện như sau:

// Phần cơ sở



```
private Timer timer1;
private MainMenu mainMenu1;
private TabControl MainTabControl;
privateTabPage WelcomeTabPage;
privateTabPage SimpleTabPage;
privateTabPage DynamicTabPage;
privateTabPage ListBoxTabPage;
privateTabPage MouseTabPage;
privateTabPage TreeTabPage;

// Welcome page
private RichTextBox WelcomeTextBox;

// Các điều khiển dành cho Simple Control Page
private Label label1;
private Label label2;
private LinkLabel linkLabel1;
private TextBox ClearTextBox;
private TextBox PasswordTextBox;
private GroupBox groupBox1;
private RadioButton radioButton1;
private RadioButton radioButton2;
private Panel panel1;
private RadioButton radioButton3;
private RadioButton radioButton4;
private Button button1;
private CheckBox checkBox1;
private CheckBox checkBox2;

// Listbox page
private ListBox listBox1;
private CheckedListBox checkedListBox1;
private Label label3;
private Label label4;
private Label PickAWord;
private ComboBox comboBox1;
private ListView listView1;
private DateTimePicker dateTimePicker1;
private Label label6;
private Label label7;
private MonthCalendar monthCalendar1;
private Label label10;
private TrackBar trackBar1;
private ProgressBar progressBar1;
private Label label8;
private DomainUpDown domainUpDown1;
```



```
private NumericUpDown numericUpDown1;
private Label label9;
private Label label11;

//Di chuyển và kiểm soát chuột
private Button ClickMeButton;

// Các thành phần điều khiển động
private CheckBox ShowDynamic;
private CheckBox UseAlternates;
private CheckBox HideChecks;
private GroupBox DynGroup;
private RadioButton DynRadioButtn1;
private RadioButton DynRadioButtn2;
private RadioButton DynRadioButtn3;
private RadioButton DynRadioButtn4;
private ListBox EventList1;
private ListBox EventList2;
private Button ClearEvents1;
private Button ClearEvents2;

//TreeView tab
private TreeView treeView1;
private ListBox tvlistBox;
private Button button4;
private Button button5;

private bool ShowingRadioGroup;
```

Phần kế tiếp sẽ khởi tạo trang chào mừng (Welcome page).

```
private void InitWelcome()
{
    WelcomeTextBox=new RichTextBox();
    WelcomeTabPage = new TabPage();
    WelcomeTabPage.Text = "Welcome";
    WelcomeTabPage.Size = new
System.Drawing.Size(576, 422);
    WelcomeTabPage.TabIndex = 0;
    WelcomeTextBox.Text = "Welcome to the Mungo Tab
App.\n"+
        "This Windows Forms demonstration" +
        " application accompanies the Sams C# and the
.NET framework"+
```



```

    " book by Bob Powell and Richard Weeks.\n\nThis
tab hosts a"+
    " RichTextBox. You can edit this text if you
wish."+
    "\n\nThe tabs in this form will show you"+
    " some of the more complex controls that you can
use in your "+
    "Windows Forms application.\n\nPlease examine
the source code"+
    " for this application carefully.\n\nBob
Powell.\n";
    WelcomeTextBox.Size = new
System.Drawing.Size(576, 424);
    WelcomeTextBox.TabIndex = 0;
    WelcomeTextBox.Anchor = AnchorStyles.Top |
    AnchorStyles.Left |
    AnchorStyles.Right |
    AnchorStyles.Bottom;
    WelcomeTextBox.Visible = true;
    WelcomeTabPage.Controls.Add(WelcomeTextBox);
    MainTabControl.Controls.Add(WelcomeTabPage);
}

```

Tiếp theo là cài đặt các bộ xử lý sự kiện cho mỗi trang

```

// Các bộ xử lý sự kiện

private void OnClickedSimple1(Object sender,
                           EventArgs e)
{
    // đây là một trong hai bộ xử lý sự kiện được gắn với nút nhấn
    string message = "You clicked the big button";
    if(this.checkBox1.Checked)
    {
        message = "And the password
is...."+this.PasswordTextBox.Text;
    }

    MessageBox.Show(message);
}

private void OnCheckColorEdit(Object
                           sender, EventArgs e)
{
    // Bộ xử lý này sẽ thêm vào hoặc loại bỏ bộ xử lý thứ hai áp đặt
    // cho nút nhấn
}

```

```
if(this.checkBox2.Checked)
{
    this.button1.Click += new
        EventHandler(OnColorEdit);
}
else
{
    this.button1.Click -= new
        EventHandler(OnColorEdit);
}
}

private void OnColorEdit(Object sender,
                        EventArgs e)
{
    // Bộ xử lý sự kiện thứ hai được đưa vào sự kiện click của nút
    // nhấp khi checkbox được đánh dấu chọn
    ColorDialog dlg = new ColorDialog();
    if(dlg.ShowDialog() == DialogResult.OK)
    {
        this.panel1.BackColor=dlg.Color;
    }
}

// Bộ xử lý này sẽ được gọi khi bạn nhấp vào nhän. Nó sẽ đưa bạn
// đến một Web site.

private void LinkClick(Object sender, EventArgs e)
{
    this.linkLabel1.LinkVisited=true;
    if(this.ClearTextBox.Text==
        "This is an editable text box")
    {
        System.Diagnostics.Process.Start(
            "IExplore.exe",
            "http://www.bobpowell.net/");
    }
    else
    {
        try
        {
            System.Diagnostics.Process.Start(
                ClearTextBox.Text);
        
```



```
        }
        catch(Exception)
        {
            MessageBox.Show(
                "Cannot start process "+ClearTextBox.Text);
        }
    }
    this.linkLabel1.Text = "Been there, Done that!";
}

private voidTextChanged(Object sender,
    EventArgs e)
{
    if(linkLabel1.LinkVisited)
    {
        linkLabel1.Text = ClearTextBox.Text;
    }
}

// Khởi tạo simple page.
private voidInitSimple()
{
    SimpleTabPage = new TabPage();
    SimpleTabPage.Size = new
        System.Drawing.Size(576, 422);
    SimpleTabPage.TabIndex = 1;
    SimpleTabPage.Text = "Simple controls";

    button1 = new Button();
    button1.Location = new System.Drawing.Point(32,
240);
    button1.Size = new System.Drawing.Size(520, 32);
    button1.TabIndex = 7;
    button1.Text = "Buttons can be clicked...";
    button1.Click+=new
EventHandler(OnClickedSimple1);
    checkBox1 = new CheckBox();
    checkBox1.Location = new System.Drawing.Point(32,
288);
    checkBox1.Size = new System.Drawing.Size(520, 16);
    checkBox1.TabIndex = 8;
    checkBox1.Text =
        "Checking this box will make the button "+
        "above say whats in the password box";
}
```

```
checkBox2 = new CheckBox();
checkBox2.Location = new System.Drawing.Point(327);
checkBox2.Size = new System.Drawing.Size(520, 16);
checkBox2.TabIndex = 9;
checkBox2.Text = "Checking this box will make the button" +
    " above edit the colour of the text panel";
checkBox2.Click += new
EventHandler(OnCheckColorEdit);

ClearTextBox = new TextBox();
ClearTextBox.Location = new
System.Drawing.Point(344, 8);
ClearTextBox.Size = new System.Drawing.Size(216,
20);
ClearTextBox.TabIndex = 2;
ClearTextBox.Text = "This is an editable text box"

domainUpDown1 = new DomainUpDown();
domainUpDown1.AccessibleName = "DomainUpDown";
domainUpDown1.AccessibleRole =
AccessibleRole.ComboBox;
domainUpDown1.Location = new
System.Drawing.Point(128, 368);
domainUpDown1.Size = new System.Drawing.Size(144
20);
domainUpDown1.TabIndex = 10;
domainUpDown1.Text = "domainUpDown1";
domainUpDown1.Items.AddRange(new object
[] {"England",
    "Africa",
    "Mongolia",
    "Japan"}));

groupBox1 = new GroupBox();
groupBox1.Location = new System.Drawing.Point(8,
80);
groupBox1.Size = new System.Drawing.Size(560, 80);
groupBox1.TabIndex = 5;
groupBox1.TabStop = false;
groupBox1.Text = "A GroupBox";

label1 = new Label();
label1.Location = new System.Drawing.Point(8, 8);
label1.Size = new System.Drawing.Size(144, 24);
label1.TabIndex = 0;
label1.Text = "This is a label control";
```



```
label12 = new Label();
label12.Location = new System.Drawing.Point(8, 41);
label12.Size = new System.Drawing.Size(328, 24);
label12.TabIndex = 4;
label12.Text = "The edit box to the right has a
password character";

label19 = new Label();
label19.Location = new System.Drawing.Point(16,
368);
label19.Size = new System.Drawing.Size(104, 24);
label19.TabIndex = 12;
label19.Text = "DomainUpDown";

label10 = new Label();
label10.Location = new System.Drawing.Point(276,
370);
label10.Size = new System.Drawing.Size(104, 24);
label10.TabIndex = 13;
label10.Text = "NumericUpDown";

linkLabel1 = new LinkLabel();
linkLabel1.Location = new
System.Drawing.Point(152, 8);
linkLabel1.Size = new System.Drawing.Size(176,
24);
linkLabel1.TabIndex = 1;
linkLabel1.TabStop = true;
linkLabel1.Text = "Link labels are like hypertext
links";
linkLabel1.Click += new EventHandler(LinkClick);

numericUpDown1 = new NumericUpDown();
numericUpDown1.BeginInit();
numericUpDown1.EndInit();
numericUpDown1.Location = new
System.Drawing.Point(392, 368);
numericUpDown1.Size = new
System.Drawing.Size(176, 20);
numericUpDown1.TabIndex = 11;

panel1 = new Panel();
panel1.BackColor =
(System.Drawing.Color)System.Drawing.Color.FromArgb
((byte)255,
(byte)255,
```

```
(byte)128);
panel1.BorderStyle =
    (BorderStyle)FormBorderStyle.FixedSingle;
panel1.Location = new System.Drawing.Point(8,
    168);
panel1.Size = new System.Drawing.Size(560, 64);
panel1.TabIndex = 6;

radioButton1 = new RadioButton();
radioButton1.Location = new
    System.Drawing.Point(16, 24);
radioButton1.Size = new System.Drawing.Size(504,
    16);
radioButton1.TabIndex = 0;
radioButton1.Text = "RadioButtons";

radioButton2 = new RadioButton();
radioButton2.Location = new
    System.Drawing.Point(16, 48);
radioButton2.Size = new System.Drawing.Size(504,
    16);
radioButton2.TabIndex = 1;
radioButton2.Text = "In a groupBox are used to
isolate";

radioButton3 = new RadioButton();
radioButton3.Location = new
    System.Drawing.Point(16, 8);
radioButton3.Size = new System.Drawing.Size(536,
    16);
radioButton3.TabIndex = 0;
radioButton3.Text = "Other radio buttons";

radioButton4 = new RadioButton();
radioButton4.Location = new
    System.Drawing.Point(16, 32);
radioButton4.Size = new System.Drawing.Size(536,
    16);
radioButton4.TabIndex = 1;
radioButton4.Text = "in other GroupBoxes,
or in this case, Panels.";

panel1.Controls.Add(radioButton3);
panel1.Controls.Add(radioButton4);

groupBox1.Controls.Add(radioButton1);
groupBox1.Controls.Add(radioButton2);
```



```

    PasswordTextBox = new TextBox();
    PasswordTextBox.Location = new
        System.Drawing.Point(344, 40);
    PasswordTextBox.PasswordChar = '*';
    PasswordTextBox.Size = new
        System.Drawing.Size(216, 20);
    PasswordTextBox.TabIndex = 3;
    PasswordTextBox.Text = "Password";

    ClearTextBox.TextChanged += new
Event Handler(TextChanged);

    SimpleTabPage.Controls.Add(button1);
    SimpleTabPage.Controls.Add(checkBox1);
    SimpleTabPage.Controls.Add(checkBox2);
    SimpleTabPage.Controls.Add(ClearTextBox);
    SimpleTabPage.Controls.Add(domainUpDown1);
    SimpleTabPage.Controls.Add(groupBox1);
    SimpleTabPage.Controls.Add(label1);
    SimpleTabPage.Controls.Add(label10);
    SimpleTabPage.Controls.Add(label2);
    SimpleTabPage.Controls.Add(label9);
    SimpleTabPage.Controls.Add(linkLabel1);
    SimpleTabPage.Controls.Add(numericUpDown1);
    SimpleTabPage.Controls.Add(panel1);
    SimpleTabPage.Controls.Add(PasswordTextBox);
}

// Các bộ xử lý dành cho Listbox

// Bộ xử lý này chuyển giá trị trong trackbar cho progress bar.
private void OnTrack(Object sender, EventArgs e)
{
    TrackBar b=(TrackBar)sender;
    this.progressBar1.Value = b.Value;
}

// Bộ xử lý này sẽ xây dựng câu từ các mục checkbox trong danh sách
// sau đó hiển thị câu bên trong nhãn.
private void CheckedListHandler(Object
                                sender, ItemCheckEventArgs e)
{
    StringBuilder sb=new StringBuilder();
    int ni=-1;
    if(e.NewValue==CheckState.Checked)
        ni=e.Index;
    for(int i=0;i<checkedListBox1.Items.Count;i++)

```

```
{  
    if(i==ni || (i!=e.Index &&  
        checkedListBox1.GetItemChecked(i)))  
  
    sb.Append(checkedListBox1.Items[i].ToString() + " ");  
}  
    PickAWord.Text = sb.ToString();  
}  
  
// this handler gets the items from the list box and  
changes thier case  
// as the mouse passes over them.  
private void ListBoxMouseOver(Object sender,  
                               MouseEventArgs e)  
{  
    string s;  
    int i=0;  
    // first we reset the case of all the strings  
    foreach(object o in listBox1.Items)  
    {  
        s=(string)o;  
        listBox1.Items[i++]=s.ToLower();  
    }  
    i = listBox1.IndexFromPoint(e.X,e.Y);  
    if(i>-1)  
    {  
        s=(string)listBox1.Items[i];  
        listBox1.Items[i]=s.ToUpper();  
    }  
}  
  
// Right clicking the combo box invokes this handler  
// it sorts the contents of the dropdown.  
private void SortComboboxHandler(Object sender,  
                                 EventArgs e)  
{  
    this.comboBox1.Sorted=true;  
}  
  
// List box tab initialization.  
private void InitLists()  
{  
    ListBoxTabPage = new TabPage();  
    ListBoxTabPage.Size = new  
        System.Drawing.Size(576, 422);  
    ListBoxTabPage.TabIndex = 2;  
    ListBoxTabPage.Text = "List boxes";
```

```
checkedListBox1 = new CheckedListBox();
checkedListBox1.Items.AddRange(new object[]
    {"The",
     "These", "All", "Words", "Men", "Are", "Can", "Be",
     "Might", "Not", "Be", "Made", "As", "Happy", "Equal",
     "Stupid", "Lost"});
checkedListBox1.Location = new
    System.Drawing.Point(216, 8);
checkedListBox1.Size = new
    System.Drawing.Size(192, 94);
checkedListBox1.TabIndex = 1;
checkedListBox1.CheckOnClick=true;
checkedListBox1.ItemCheck += new
    ItemCheckEventHandler(CheckedListHandler);

comboBox1 = new ComboBox();
comboBox1.Items.AddRange(new object[] {"A",
    "Little", "aardvark", "Never", "Hurt",
    "Anyone", "5 ", "9 ", "7 ", "1", "0 ",
    "2", "4", "3", "6", "8"});
comboBox1.Location = new
    System.Drawing.Point(8, 144);
comboBox1.Size = new
    System.Drawing.Size(184, 21);
comboBox1.TabIndex = 5;
comboBox1.Text = "Context menu sorts";
ContextMenu m=new ContextMenu();
MenuItem t=new MenuItem("Sort",
    new EventHandler(SortComboboxHandler));
m.MenuItems.Add(t);
comboBox1.ContextMenu = m;

dateTimePicker1 = new DateTimePicker();
dateTimePicker1.Location =
    new System.Drawing.Point(216, 272);
dateTimePicker1.Size =
    new System.Drawing.Size(344, 20);
dateTimePicker1.TabIndex = 7;

label3 = new Label();
label3.Location = new
    System.Drawing.Point(8, 112);
label3.Size = new
    System.Drawing.Size(184, 16);
label3.TabIndex = 2;
label3.Text = "A Simple list box";
label4 = new Label();
label4.Location = new
```

```
        System.Drawing.Point(224, 112);
label4.Size = new
        System.Drawing.Size(184, 16);
label4.TabIndex = 3;
label4.Text = "A Checked list box";
label6 = new Label();
label6.Location = new
        System.Drawing.Point(216, 248);
label6.Size = new System.Drawing.Size(184, 16);
label6.TabIndex = 8;
label6.Text = "A list view";
label7 = new Label();
label7.Location = new
        System.Drawing.Point(214, 303);
label7.Size = new System.Drawing.Size(184, 16);
label7.TabIndex = 9;
label7.Text = "A DateTimePicker";
label8 = new Label();
label8.Location = new
        System.Drawing.Point(7, 341);
label8.Size = new System.Drawing.Size(184, 16);
label8.TabIndex = 11;
label8.Text = "The MonthCalender control";

label11 = new Label();
label11.Location = new
        System.Drawing.Point(7, 384);
label11.Size = new System.Drawing.Size(184, 16);
label11.TabIndex = 14;
label11.Text = "Trackbar
        and progress bar (Right)";

listBox1 = new ListBox();
listBox1.Items.AddRange( new object[] {"Fish",
    "Chips", "Vinegar", "Marmite",
    "Cream Crackers",
    "Marmalade", "Stilton", "Mushy Peas",
    "Sherbert Lemons",
    "Wellie boots", "Spanners"});
listBox1.Location = new
        System.Drawing.Point(8, 8);
listBox1.Size = new System.Drawing.Size(184, 9);
listBox1.TabIndex = 0;
listBox1.MouseMove += new
        MouseEventHandler(ListBoxMouseOver);
listView1 = new ListView();
listView1.ForeColor =
        System.Drawing.SystemColors.WindowTe
```

```
listView1.Items.Add(new
    ListViewItem("knives"));
listView1.Items.Add(new ListViewItem("forks"));
listView1.Items.Add(new
    ListViewItem("spoons"));
listView1.Items.Add(new ListViewItem("dogs"));
listView1.Items.Add(new ListViewItem("fish"));
listView1.Location = new
    System.Drawing.Point(216, 136);
listView1.Size = new
    System.Drawing.Size(352, 96);
listView1.TabIndex = 6;

monthCalendar1 = new MonthCalendar();
monthCalendar1.Location = new
System.Drawing.Point(8, 184);
monthCalendar1.TabIndex = 10;
monthCalendar1.TabStop = true;

progressBar1 = new ProgressBar();
progressBar1.Location = new
System.Drawing.Point(216, 344);
progressBar1.Size = new System.Drawing.Size(336,
24);
progressBar1.TabIndex = 13;

PickAWord = new Label();
PickAWord.Location = new
System.Drawing.Point(416, 8);
PickAWord.Size = new System.Drawing.Size(152, 96);
PickAWord.TabIndex = 4;

trackBar1 = new TrackBar();
trackBar1.BeginInit();
trackBar1.Location = new
System.Drawing.Point(272, 376);
trackBar1.Maximum = 100;
trackBar1.Size = new System.Drawing.Size(184, 42);
trackBar1.TabIndex = 12;
trackBar1.ValueChanged += new
EventHandler(OnTrack);
trackBar1.EndInit();

ListBoxTabPage.Controls.Add(checkedListBox1);
ListBoxTabPage.Controls.Add(comboBox1);
ListBoxTabPage.Controls.Add(dateTimePicker1);
ListBoxTabPage.Controls.Add(label11);
ListBoxTabPage.Controls.Add(label3);
```

```
    ListBoxTabPage.Controls.Add(label4);
    ListBoxTabPage.Controls.Add(label6);
    ListBoxTabPage.Controls.Add(label7);
    ListBoxTabPage.Controls.Add(label8);
    ListBoxTabPage.Controls.Add(listBox1);
    ListBoxTabPage.Controls.Add(listView1);
    ListBoxTabPage.Controls.Add(monthCalendar1);
    ListBoxTabPage.Controls.Add(PickAWord);
    ListBoxTabPage.Controls.Add(progressBar1);
    ListBoxTabPage.Controls.Add(trackBar1);
}

// This is the first of two possible events fired by
the
// dynamic radio buttons. It adds to a list box.
private void RadioEvent1(Object sender, EventArgs e)
{
    RadioButton r = (RadioButton)sender;
    this.EventList1.Items.Add("Event #1 from:
"+r.Text);
}

// This is the second of two possible events fired by
the
// dynamic radio buttons. It adds to a list box.
private void RadioEvent2(Object sender, EventArgs e)
{
    RadioButton r = (RadioButton)sender;
    this.EventList2.Items.Add("Event #2 from:
"+r.Text);
}

// This handler clears the first list box out
private void Clear1(Object sender, EventArgs e)
{
    this.EventList1.Items.Clear();
}

// This handler clears the second list box out
private void Clear2(Object sender, EventArgs e)
{
    this.EventList2.Items.Clear();
}

// This routine removes all events from all radio
buttons
// in the dynamic page.
private void RemoveEvents()
```



```
{  
    DynRadioButtn4.Click -= new  
EventHandler(RadioEvent1);  
    DynRadioButtn3.Click -= new  
EventHandler(RadioEvent1);  
    DynRadioButtn2.Click -= new  
EventHandler(RadioEvent1);  
    DynRadioButtn1.Click -= new  
EventHandler(RadioEvent1);  
    DynRadioButtn4.Click -= new  
EventHandler(RadioEvent2);  
    DynRadioButtn3.Click -= new  
EventHandler(RadioEvent2);  
    DynRadioButtn2.Click -= new  
EventHandler(RadioEvent2);  
    DynRadioButtn1.Click -= new  
EventHandler(RadioEvent2);  
}  
  
// This method add the correct event handler  
alternative  
// to the radiobuttons on the dynamic page  
private void AddEvents()  
{  
  
    if(!this.UseAlternates.Checked)  
    {  
        DynRadioButtn4.Click += new  
EventHandler(RadioEvent1);  
        DynRadioButtn3.Click += new  
EventHandler(RadioEvent1);  
        DynRadioButtn2.Click += new  
EventHandler(RadioEvent1);  
        DynRadioButtn1.Click += new  
EventHandler(RadioEvent1);  
    }  
    else  
    {  
        DynRadioButtn4.Click += new  
EventHandler(RadioEvent2);  
        DynRadioButtn3.Click += new  
EventHandler(RadioEvent2);  
        DynRadioButtn2.Click += new  
EventHandler(RadioEvent2);  
        DynRadioButtn1.Click += new  
EventHandler(RadioEvent2);  
    }  
}
```

```
// This event handler swaps the dynamic radio button
event handlers
public void OnUseAlternates(Object sender, EventArgs
e)
{
    if(ShowingRadioGroup)
    {
        RemoveEvents();
        AddEvents();
    }
}

// This method removes the whole dynamic radiobutton
// panel, clears the event list and destroys the items
public void RemoveRadio()
{
    if(ShowingRadioGroup)
    {
        DynGroup.Controls.Remove(DynRadioButtn4);
        DynGroup.Controls.Remove(DynRadioButtn3);
        DynGroup.Controls.Remove(DynRadioButtn2);
        DynGroup.Controls.Remove(DynRadioButtn1);
        DynamicTabPage.Controls.Remove(DynGroup);

        RemoveEvents();
        DynamicTabPage.Controls.Remove(DynGroup);

        DynRadioButtn4.Dispose();
        DynRadioButtn3.Dispose();
        DynRadioButtn2.Dispose();
        DynRadioButtn1.Dispose();
        DynGroup.Dispose();

        ShowingRadioGroup = false;
    }
}

//This method adds the dynamic radio button group and
//wires up the event handlers.
private void AddRadio()
{
    if(!ShowingRadioGroup)
    {
        DynGroup = new GroupBox();
        DynGroup.Location = new
System.Drawing.Point(240, 16);
```

```
DynGroup.Size = new System.Drawing.Size(312,
120);
    DynGroup.TabIndex = 3;
    DynGroup.TabStop = false;
    DynGroup.Text = "Dynamic radiobuttons";

    DynRadioButtn1 = new RadioButton();
    DynRadioButtn1.Location = new
System.Drawing.Point(8, 24);
    DynRadioButtn1.Size = new
System.Drawing.Size(296, 16);
    DynRadioButtn1.TabIndex = 0;
    DynRadioButtn1.Text = "Choice 1";
    DynRadioButtn2 = new RadioButton();
    DynRadioButtn2.Location = new
System.Drawing.Point(8, 41);
    DynRadioButtn2.Size = new
System.Drawing.Size(296, 16);
    DynRadioButtn2.TabIndex = 1;
    DynRadioButtn2.Text = "Choice 2";

    DynRadioButtn3 = new RadioButton();
    DynRadioButtn3.Location = new
System.Drawing.Point(8, 64);
    DynRadioButtn3.Size = new
System.Drawing.Size(296, 16);
    DynRadioButtn3.TabIndex = 2;
    DynRadioButtn3.Text = "Choice 3";

    DynRadioButtn4 = new RadioButton();
    DynRadioButtn4.Location = new
System.Drawing.Point(8, 88);
    DynRadioButtn4.Size = new
System.Drawing.Size(296, 16);
    DynRadioButtn4.TabIndex = 3;
    DynRadioButtn4.Text = "Choice 4";

    AddEvents();

    DynGroup.Controls.Add(DynRadioButtn4);
    DynGroup.Controls.Add(DynRadioButtn3);
    DynGroup.Controls.Add(DynRadioButtn2);
    DynGroup.Controls.Add(DynRadioButtn1);
    DynamicTabPage.Controls.Add(DynGroup);

    ShowingRadioGroup = true;
}
```

```
}

//This event handler uses helper methods to manage the
presence of the
//dynamic radiobutton group and the handlers that
they use
private void ShowDynamicEvent(Object sender,
EventArgs e)
{
    CheckBox b = (CheckBox) sender;

    if(b.Checked)
    {
        AddRadio();
    }
    else
    {
        RemoveRadio();
    }

}

// This event handler adds or removes the dynamic
check buttons and
// repositions the control to make it look neat and
tidy.
private void AddChecks(Object sender, EventArgs e)
{
    CheckBox c = (CheckBox) sender;

    if(this.HideChecks.Checked)
    {
        RemoveRadio();
        c.Location = new Point(8,16);

        DynamicTabPage.Controls.Remove(UseAlternates);
        DynamicTabPage.Controls.Remove>ShowDynamic.Click -= new
EventHandler(ShowDynamicEvent);
        UseAlternates.Dispose();
        ShowDynamic.Dispose();
    }
    else
    {
        c.Location = new Point(8,64);
        ShowDynamic = new CheckBox();
    }
}
```

```
        ShowDynamic.Location = new
System.Drawing.Point(8, 16);
        ShowDynamic.Size = new System.Drawing.Size(168,
16);
        ShowDynamic.TabIndex = 0;
        ShowDynamic.Text = "Show dynamic RadioButtons";
        ShowDynamic.Click += new
EventHandler(ShowDynamicEvent);

        UseAlternates = new CheckBox();
        UseAlternates.Location = new
System.Drawing.Point(8, 40);
        UseAlternates.Size = new
System.Drawing.Size(168, 16);
        UseAlternates.TabIndex = 1;
        UseAlternates.Text = "Use alternate handlers";
        UseAlternates.Click += new
EventHandler(OnUseAlternates);

        DynamicTabPage.Controls.Add(UseAlternates);
        DynamicTabPage.Controls.Add>ShowDynamic);
    }
}

// This method initializes the dynamic buttons tab
private void InitDynamic()
{
    DynamicTabPage = new TabPage();
    DynamicTabPage.Size = new
System.Drawing.Size(576, 422);
    DynamicTabPage.TabIndex = 3;
    DynamicTabPage.Text = "Dynamic controls";

    ClearEvents1 = new Button();
    ClearEvents1.Location = new
System.Drawing.Point(48, 328);
    ClearEvents1.Size = new System.Drawing.Size(128,
24);
    ClearEvents1.TabIndex = 6;
    ClearEvents1.Text = "Clear the events";
    ClearEvents1.Click += new EventHandler(Clear1);

    ClearEvents2 = new Button();
    ClearEvents2.Location = new
System.Drawing.Point(340, 330);
    ClearEvents2.Size = new System.Drawing.Size(128,
24);
```

```
ClearEvents2.TabIndex = 7;
ClearEvents2.Text = "Clear the events";
ClearEvents2.Click += new EventHandler(Clear2);

EventList1 = new ListBox();
EventList1.Location = new
System.Drawing.Point(16, 176);
EventList1.Size = new System.Drawing.Size(200,
121);
EventList1.TabIndex = 4;
EventList2 = new ListBox();
EventList2.Location = new
System.Drawing.Point(308, 180);
EventList2.Size = new System.Drawing.Size(200,
121);
EventList2.TabIndex = 5;

HideChecks = new CheckBox();
HideChecks.Location = new System.Drawing.Point(
64);
HideChecks.Size = new System.Drawing.Size(168,
16);
HideChecks.TabIndex = 2;
HideChecks.Text = "Hide checkboxes";
HideChecks.Click += new EventHandler(AddChecks);
AddChecks(HideChecks, new EventArgs());

DynamicTabPage.Controls.Add(ClearEvents1);
DynamicTabPage.Controls.Add(ClearEvents2);
DynamicTabPage.Controls.Add(EventList2);
DynamicTabPage.Controls.Add(EventList1);
DynamicTabPage.Controls.Add(HideChecks);
}

// 
// Handlers for the mouse tab
//

// This handler moves the button to the opposite side
of the tab-page
// from the mouse cursor
private void OnMouseMoved(Object sender,
MouseEventArgs e)
{
    Point center =
        new
        Point(MouseTabPage.Width/2, MouseTabPage.Height/2);
```



```
ClickMeButton.Location =
    new Point(center.X-
(ClickMeButton.Size.Width/2)-(e.X-center.X),
    center.Y-(ClickMeButton.Size.Height/2)-(e.Y-
center.Y));
}

//This handler shows when the button is caught and
clicked.
private void OnClickedClickme(Object sender,
EventArgs e)
{
    MessageBox.Show("Caught me!!");
}

// This method initializes the mouse page.
private void InitMouse()
{
    MouseTabPage = new TabPage();
    MouseTabPage.Controls.Add(ClickMeButton);
    MouseTabPage.Size = new System.Drawing.Size(576,
422);
    MouseTabPage.TabIndex = 4;
    MouseTabPage.Text = "Mouse interaction";

    ClickMeButton = new Button();
    ClickMeButton.Location = new
System.Drawing.Point(200, 128);
    ClickMeButton.Size = new System.Drawing.Size(184,
112);
    ClickMeButton.TabIndex = 0;
    ClickMeButton.Text = "Click me!";
    ClickMeButton.Click += new
EventHandler(OnClickedClickme);

    MouseTabPage.Controls.Add(ClickMeButton);

    MouseTabPage.MouseMove += new
MouseEventHandler(OnMouseMoved);
}

//Handlers for the tree tab.....

//The overloaded list function shows the tree view
events in a list

private void List(string s, TreeNode n)
```

```
{  
    string o=s+" "+n.Text;  
    tvlistBox.Items.Add(o);  
}  
  
private void List(string s, string l, TreeNode n)  
{  
    string o=s+" (new = "+l+") current = "+n.Text;  
    tvlistBox.Items.Add(o);  
}  
  
// These handlers simply reflect the event type and  
little bit of  
// node data to the list box on the right of the  
treeView1 control.  
private void OnAfterCheck(Object  
sender,TreeViewEventArgs e)  
{  
    List("AfterCheck", e.Node);  
}  
  
private void OnAfterCollapse(Object  
sender,TreeViewEventArgs e)  
{  
    List("AfterCollapse", e.Node);  
}  
  
private void OnAfterExpand(Object  
sender,TreeViewEventArgs e)  
{  
    List("AfterExpand", e.Node);  
}  
  
private void OnAfterSelect(Object  
sender,TreeViewEventArgs e)  
{  
    List("AfterSelect", e.Node);  
}  
private void OnBeforeCheck(Object  
sender,TreeViewCancelEventArgs e)  
{  
    List("AfterCollapse", e.Node);  
}  
private void OnBeforeCollapse(Object  
sender,TreeViewCancelEventArgs e)  
{  
    List("BeforeCollapse", e.Node);  
}
```



```
    private void OnBeforeExpand(Object
sender,TreeViewCancelEventArgs e)
{
    List("BeforeExpand", e.Node);
}
private void OnBeforeLabelEdit(Object
sender,NodeLabelEditEventArgs e)
{
    List("BeforeEdit", e.Label, e.Node);
}

private void OnAfterLabelEdit(Object
sender,NodeLabelEditEventArgs e)
{
    List("AfterEdit", e.Label, e.Node);
}

private void OnBeforeSelect(Object
sender,TreeViewCancelEventArgs e)
{
    List("BeforeSelect", e.Node);
}

private void OnAddRoot(Object sender, EventArgs e)
{
    button5.Enabled=true;

    if(treeView1.Nodes.Count==0)
    {
        treeView1.Nodes.Add(new TreeNode("Root
node"));
    }
    else
    {
        treeView1.Nodes.Add(new TreeNode("Sibling
node"));
    }
}

private void OnAddChild(Object sender, EventArgs e)
{
    if(treeView1.SelectedNode==null)
        return;
    treeView1.SelectedNode.Nodes.Add(new
TreeNode("Child"));
}
```

```
//Initializes the tree control.
private void InitTree()
{
    TreeTabPage = new TabPage();
    TreeTabPage.Text = "TreeView";
    TreeTabPage.Size = new System.Drawing.Size(576,
422);
    TreeTabPage.TabIndex = 5;

    treeView1 = new TreeView();
    treeView1.Anchor = AnchorStyles.Left |
        AnchorStyles.Top |
        AnchorStyles.Right |
        AnchorStyles.Bottom;
    treeView1.Size = new System.Drawing.Size(264,
360);
    treeView1.TabIndex = 0;
    treeView1.ShowLines=true;
    treeView1.ShowPlusMinus=true;
    treeView1.ShowRootLines=true;
    treeView1.LabelEdit=true;

    treeView1.AfterCheck +=
        new TreeViewEventHandler(OnAfterCheck);
    treeView1.AfterCollapse +=
        new TreeViewEventHandler(OnAfterCollapse);
    treeView1.AfterExpand +=
        new TreeViewEventHandler(OnAfterExpand);
    treeView1.AfterSelect +=
        new TreeViewEventHandler(OnAfterSelect);
    treeView1.AfterLabelEdit +=
        new
NodeLabelEditEventHandler(OnAfterLabelEdit);
    treeView1.BeforeCheck +=
        new
TreeViewCancelEventHandler(OnBeforeCheck);
    treeView1.BeforeCollapse +=
        new
TreeViewCancelEventHandler(OnBeforeCollapse);
    treeView1.BeforeExpand +=
        new
TreeViewCancelEventHandler(OnBeforeExpand);
    treeView1.BeforeSelect +=
        new
TreeViewCancelEventHandler(OnBeforeSelect);
    treeView1.BeforeLabelEdit +=
        new
NodeLabelEditEventHandler(OnBeforeLabelEdit);
```

```
tvlistBox = new ListBox();
tvlistBox.Location = new
System.Drawing.Point(272, 0);
tvlistBox.Size = new System.Drawing.Size(304,
424);
tvlistBox.ForeColor =
System.Drawing.SystemColors.WindowText;
tvlistBox.TabIndex = 1;

button4 = new Button();
button4.Location = new System.Drawing.Point(16,
376);
button4.Size = new System.Drawing.Size(96, 24);
button4.TabIndex = 2;
button4.Text = "Add Root";
button4.Click += new EventHandler(OnAddRoot);

button5 = new Button();
button5.Location = new System.Drawing.Point(138,
376);
button5.Size = new System.Drawing.Size(96, 24);
button5.TabIndex = 3;
button5.Text = "Add Child";
button5.Click += new EventHandler(OnAddChild);
button5.Enabled=false;

TreeTabPage.Controls.Add(button4);
TreeTabPage.Controls.Add(button5);
TreeTabPage.Controls.Add(tvlistBox);
TreeTabPage.Controls.Add(treeView1);
}

public MungoTabApp()
{
    // components = new
System.ComponentModel.Container();
    AutoScaleBaseSize = new System.Drawing.Size(5,
13);
    ClientSize = new System.Drawing.Size(600, 477);
    MainTabControl = new TabControl();
```

```
MainTabControl.Location = new
System.Drawing.Point(8, 8);
MainTabControl.SelectedIndex = 0;
MainTabControl.Size = new
System.Drawing.Size(584, 448);
MainTabControl.TabIndex = 0;

InitWelcome();
InitSimple();
InitLists();
InitDynamic();
InitMouse();
InitTree();

mainMenu1 = new MainMenu();
Menu = mainMenu1;
Text = "Mungo Tab App";
timer1 = new Timer();

MainTabControl.Controls.Add(SimpleTabPage);
MainTabControl.Controls.Add(ListBoxTabPage);
MainTabControl.Controls.Add(DynamicTabPage);
MainTabControl.Controls.Add(MouseTabPage);
MainTabControl.Controls.Add(TreeTabPage);

Controls.Add(MainTabControl);

}

public static int Main(string[] args)
{
    Application.Run(new MungoTabApp());
    return 0;
}
```

5. KẾT CHƯƠNG

Chương này chúng ta đã xem qua toàn bộ hệ thống Windows Forms, chương trình đơn giản Windows Forms Hello World cho đến ứng dụng phức tạp bao gồm đầy đủ các thành phần giao diện như MungoTabApp.cs. Bạn đã hiểu được cách sử dụng và điều khiển những thành phần giao diện Windows Forms thông qua sự kiện, thuộc tính. Để chuyển các ứng dụng đồ họa của môi trường Windows cũ sang .NET thì Windows Forms là tất cả những gì bạn cần phải quan tâm đến. Tạm dừng để tài Windows Forms ở đây, chúng ta sẽ bắt đầu chuyển sang học về cách xử lý dữ liệu của .NET trong chương sau.

Chương 3.3

RÀNG BUỘC DỮ LIỆU

Các vấn đề chính sẽ được đề cập đến:

- ✓ Chiến lược ràng buộc dữ liệu
- ✓ Nguồn dữ liệu ràng buộc
- ✓ Ràng buộc đơn giản với DataSet
- ✓ Ràng buộc phức hợp các điều khiển vào dữ liệu
- ✓ Ràng buộc điều khiển vào cơ sở dữ liệu bằng ADO.NET
- ✓ Tạo khung nhìn Database Viewer với Visual Studio và ADO.NET

1. CHIẾN LƯỢC RÀNG BUỘC DỮ LIỆU

Trong chương trước, chúng ta đã tìm hiểu qua các thành phần giao diện chuẩn của Windows Forms. Một số thành phần điều khiển sẽ trở nên dễ dàng kiểm soát và lập trình hơn nếu nó được ràng buộc (binding) vào một nguồn dữ liệu nào đó ví dụ như một bảng trong cơ sở dữ liệu.

Dữ liệu nói chung được lưu giữ trong bảng (table) bao gồm các dòng và cột. Ràng buộc dữ liệu (Data Binding) là kỹ thuật kết nối giao diện của thành phần điều khiển trực tiếp với một hoặc nhiều dòng, cột dữ liệu trong bảng. Thậm chí có thể toàn bộ bảng dữ liệu nhằm mục đích cho phép người dùng xem, xử lý và hiệu chỉnh dữ liệu trực tiếp.

Có hai kiểu ràng buộc dữ liệu, ràng buộc đơn (Simple bind) và ràng buộc phức (Complex bind). Ràng buộc đơn là kết nối dữ liệu của một dòng hay cột trong bảng vào các điều khiển đơn giản như textbox, checkbox ... Ràng buộc phức là kết nối toàn bộ bảng hoặc nhiều bảng vào thành phần điều khiển để hiển thị hay xử lý dữ liệu.

2. CÁC NGUỒN DỮ LIỆU RÀNG BUỘC

Nguồn dữ liệu thích hợp cho việc ràng buộc nói chung bao gồm bất kỳ đối tượng nào có cài đặt giao tiếp hay interface IList. Hầu hết các tập hợp trong thư viện .NET đều cài đặt giao tiếp này. Đối tượng dữ liệu cung cấp bởi ADO.NET cũng cài đặt IList, bản thân bạn cũng có thể tự xây dựng các đối tượng làm nguồn dữ liệu thông qua việc cài đặt IList interface.

2.1. Giao tiếp IList

Giao tiếp IList chỉ yêu cầu một số phương thức mà bạn cần cài đặt như sau

- **Add** – Thêm một mục vào danh sách
- **Clear** – Xóa tất cả các mục khỏi danh sách
- **Contains** – Tìm một mục dữ liệu trong danh sách
- **IndexOf** – Xác định chỉ số của một giá trị nào đó trong danh sách
- **Insert** – Đặt một giá trị mới vào danh sách
- **Remove** – Loại một giá trị ra khỏi danh sách
- **RemoveAt** – Loại một mục khỏi danh sách dựa trên chỉ số của nó

Truy cập một mục dữ liệu trong danh sách được thực hiện thông qua chỉ số Ví dụ, `Item x= myList[n]`. Các mục chứa trong danh sách được xem là có kiểu đối tượng .NET.

2.2. Các đối tượng .NET cài đặt IList

Có rất nhiều lớp trong thư viện .NET cài đặt IList. Một số chỉ dùng cho mục đích truy xuất dữ liệu. Các lớp thường được sử dụng cụ thể có ArrayList và StringCollection. Các lớp phức tạp hơn như DataSetView hay DataView làm việc chung với ADO cung cấp cách truy xuất toàn bộ dữ liệu của bảng hoặc khung nhìn (View).

Các lớp phụ còn lại ít được dùng hơn, chủ yếu được dùng để mở rộng cho công cụ của các nhà phát triển .NET truy tìm các siêu dữ liệu kết hợp bên trong một lớp. Chúng bao gồm các lớp như CodeClassCollection và CodeStatementCollection cùng những đối tượng khác sử dụng mã của mô hình tài liệu đối tượng (Document Object Model) hay còn gọi là DOM.

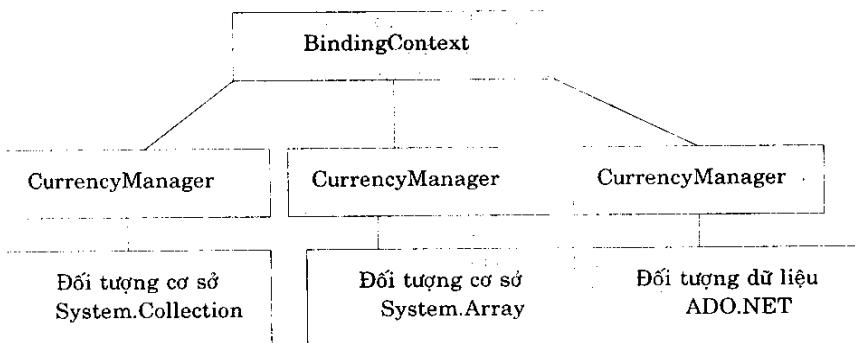
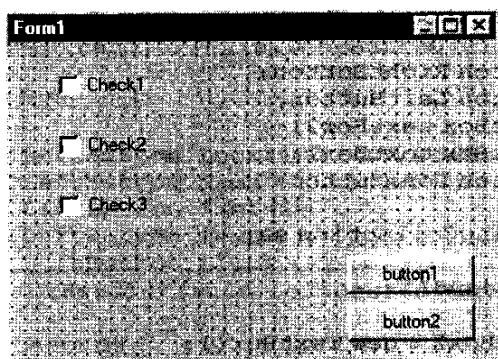
3. RÀNG BUỘC GIẢN ĐƠN

Như đã đề cập trước đây, ràng buộc giản đơn là công việc lấy các mẫu dữ liệu đơn giản và buộc chúng vào các thành phần điều khiển. Khi dữ liệu thay đổi điều khiển sẽ phản ánh sự thay đổi này. Khi thành phần điều khiển bị hiệu chỉnh dữ liệu sẽ được ghi lại nếu quyền ghi trên dữ liệu được phép.

Mỗi điều khiển Windows Forms quản lý một đối tượng BindingContext, đối tượng này có một tập hợp các đối tượng CurrencyManager. Dù sao th CurrencyManager không giúp gì cho công việc tài chính. Nó đơn giản chỉ quản lý vị trí hiện hành bên trong một đối tượng dữ liệu cụ thể nào đó giúp bạn.

Các điều khiển con, như GroupBox, cũng có thể có đối tượng BindingContexts và CurrencyManager độc lập.

Hình 3.3-1 minh họa quan hệ giữa Form, Điều khiển và đối tượng BindingContext, CurrencyManager.



Hình 3.3-1 Quan hệ giữa Form và BindingManager, CurrencyManager

Chúng ta hãy xem các sử dụng BindingManager và CurrencyManager với một ứng dụng đơn giản ràng buộc thuộc tính Text của ô textBox với một tập hợp chuỗi. Ví dụ 3.3-1 là mã nguồn của ứng dụng DataBound

Ví dụ 3.3-1 databound.cs

```

namespace Sams
{
    using System;
    using System.Drawing;
    using System.Collections;
    using System.Collections.Specialized;
    using System.ComponentModel;
    using System.Windows.Forms;

    /// <summary>

```

```
/// Summary description for databound.
/// </summary>
public class databound : System.Windows.Forms.Form
{
    private Button RightButton;
    private Button LeftButton;
    private TextBox textBox2;
    private TextBox textBox1;
    private Button DoneButton;

    private StringCollection sa;

    public databound()
    {
        this.textBox2 = new TextBox();
        this.RightButton = new Button();
        this.textBox1 = new TextBox();
        this.DoneButton = new Button();
        this.LeftButton = new Button();
        this.SuspendLayout();
        //
        // textBox2
        //
        this.textBox2.Location = new Point(184, 16);
        this.textBox2.Name = "textBox2";
        this.textBox2.TabIndex = 2;
        this.textBox2.Text = "textBox2";
        //
        // RightButton
        //
        this.RightButton.Location = new Point(192, 64);
        this.RightButton.Name = "RightButton";
        this.RightButton.TabIndex = 4;
        this.RightButton.Text = ">>";
        this.RightButton.Click += new
            EventHandler(this.RightButton_Click);
        //
        // textBox1
        //
        this.textBox1.Location = new Point(8, 16);
        this.textBox1.Name = "textBox1";
        this.textBox1.TabIndex = 1;
        this.textBox1.Text = "textBox1";
        //
        // DoneButton
        //
        this.DoneButton.Location = new Point(104, 64);
        this.DoneButton.Name = "DoneButton";
```

```
this.DoneButton.TabIndex = 0;
this.DoneButton.Text = "Done";
this.DoneButton.Click += new
    EventHandler(this.DoneButton_Click);
//
// LeftButton
//
this.LeftButton.Location = new Point(16, 64);
this.LeftButton.Name = "LeftButton";
this.LeftButton.TabIndex = 3;
this.LeftButton.Text = "<<";
this.LeftButton.Click += new
    EventHandler(this.LeftButton_Click);
//
// databound
//
this.AutoScaleBaseSize = new
    System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(294,
    111);
this.ControlBox = false;
this.Controls.AddRange(new Control[] {
    this.RightButton,
    this.LeftButton,
    this.textBox2,
    this.textBox1,
    this.DoneButton});
this.FormBorderStyle =
    FormBorderStyle.FixedDialog;
this.Name = "databound";
this.ShowInTaskbar = false;
this.Text = "databound";
this.ResumeLayout(false);

// Đoạn mã này dùng khởi tạo dữ liệu cho ràng buộc giản đơn
```

```
// Chúng ta ràng buộc hai điều khiển textbox vào cùng một
// StringCollection
sa=new StringCollection();
sa.Add("Hello databinding");
sa.Add("Sams publishing");
sa.Add("C# example");
sa.Add("By Bob Powell");
```

```
// Ràng buộc điều khiển vào cơ sở dữ liệu
```

```
this.textBox1.DataBindings.Add("Text", sa, "");
this.textBox2.DataBindings.Add("Text", sa, "");
}

public override void Dispose()
{
    base.Dispose();
}

//Đơn giản chỉ tăng trị của thuộc tính Position trong CurrencyManager
protected void RightButton_Click (object sender,
                                  System.EventArgs e)
{
    this.BindingContext[sa].Position++;
}

//Đơn giản chỉ giảm trị của thuộc tính Position trong CurrencyManager
protected void LeftButton_Click (object sender,
                                  System.EventArgs e)
{
    this.BindingContext[sa].Position--;
}

protected void DoneButton_Click (object sender,
                                 System.EventArgs e)
{
    Application.Exit();
}

public static void Main()
{
    Application.Run(new databound());
}
}
```

Bạn biên dịch chương trình bằng lệnh sau:

csc t:/winexe databound.cs

Bạn lưu ý đến đoạn mã tạo và đưa StringCollection vào sử dụng

```
sa=new StringCollection();
sa.Add("Hello databinding");
sa.Add("Sams publishing");
sa.Add("C# example");
```

```
sa.Add("By Bob Powell");
```

Nhớ rằng các tập hợp Collection đều cài đặt giao tiếp `IList` hỗ trợ cho việc ràng buộc dữ liệu. Đoạn mã sau sẽ thực hiện ràng buộc dữ liệu vào ô `TextBox`

```
//Ràng buộc điều khiển vào cơ sở dữ liệu
this.textBox1.DataBindings.Add("Text", sa, "");
this.textBox2.DataBindings.Add("Text", sa, "");
```

Cuối cùng thủ tục `RightButtonClick` và `LeftButtonClick` sẽ xử lý cho phép di chuyển con trỏ xem các vị trí dữ liệu khác trong tập hợp.

4. RÀNG BUỘC DỮ LIỆU GIẢN ĐƠN

Ràng buộc dữ liệu là một tính năng quan trọng trong .NET, các điều khiển ràng buộc dữ liệu có thể sử dụng trong Windows Forms, Web Forms, ASP.NET hoặc bất kỳ ứng dụng Windows nào sử dụng nền .NET. Dữ liệu ràng buộc được thực hiện thông qua đối tượng `Dataset`. Bạn có thể hình dung `Dataset` là vùng đệm tạm thời lưu giữ phần dữ liệu đang cần xử lý. Cách đơn giản để hiểu về `Dataset` là hãy tập sử dụng nó. Ví dụ 3.3-2 dưới đây sẽ cho thấy cách xử lý `Dataset` bên trong ứng dụng sử dụng cơ chế ràng buộc giản đơn

Ví dụ 3.3-2 datasetapp.cs

```
1:  namespace Sams
2:  {
3:      using System;
4:      using System.Drawing;
5:      using System.Collections;
6:      using System.Data;
7:      using System.ComponentModel;
8:      using System.Windows.Forms;
9:
10:     //This application shows simple data binding to a
11:     //programmatically created dataset.
12:     public class datasetapp :
13:         System.Windows.Forms.Form
14:     {
15:         private System.ComponentModel.Container
16:             components;
17:
18:         //Component declarations
19:         private Label
20:             lbl_first, lbl_name, lbl_title, lbl_company, lbl_phon
21:             e;
22:         private TextBox
23:             FirstName, SurName, Title, Company, Phone;
24:         private Button btnNext, btnPrev, btnNew, btnEnd;
```



```
18: //The dataset used to store the table
19: private DataSet dataset;

20: //Button handler to navigate backwards through
   the table records
21: private void OnPrev(Object sender, EventArgs e)
22: {
23:   this.BindingContext[dataset.Tables["Contacts"
   ]].Position--;
24: }

25: //Button handler to navigate forward through the
   table records
26: private void OnNext(Object sender, EventArgs e)
27: {
28:   this.BindingContext[dataset.Tables["Contacts"
   ]].Position++;
29: }

30: //Button handler to create a new row
31: private void OnNew(Object sender, EventArgs e)
32: {
33:   NewEntry();
34: }

35: //Button handler to exit the application
36: private void OnEnd(Object sender, EventArgs e)
37: {
38:   Application.Exit();
39: }

40: //Method to move to the last record. Used when
   adding a row.
41: private void MoveToEnd()
42: {
43:   this.BindingContext[dataset.Tables["Contacts"
   ]].Position=
44:   dataset.Tables["Contacts"].Rows.Count-1;
45: }

46: //Method to add a new row to the table. Called at
   initialization
47: //and by the "New" button handler.
48: private void NewEntry()
49: {
50:   DataRow row =
      dataset.Tables["Contacts"].NewRow();
51:   //set up row data with new entries of your choice
```



```
52:     row["First"] = "Blank";
53:     row["Name"] = "";
54:     row["Company"] = "";
55:     row["Title"] = "";
56:     row["Phone"] = "";
57:     dataset.Tables[0].Rows.Add(row);
58:     dataset.AcceptChanges();
59:     MoveToEnd();
60: }

61: //Called at creation to initialize the
62: //dataset and create an empty table
63: private void InitDataSet()
64: {
65:     dataset = new DataSet("ContactData");

66:     DataTable t = new DataTable("Contacts");

67:     t.Columns.Add("First", typeof(System.String));
68:     t.Columns.Add("Name", typeof(System.String));
69:     t.Columns.Add("Company", typeof(System.String))
    );
70:     t.Columns.Add("Title", typeof(System.String));
71:     t.Columns.Add("Phone", typeof(System.String));

72:     t.MinimumCapacity = 100;

73:     dataset.Tables.Add(t);
74: }

75: //Called at initialization to do simple binding
  of the edit
76: //controls on the form to the dataset's
  "Contacts" table entries
77: private void BindControls()
78: {
79:     FirstName.DataBindings.Add("Text", dataset.Tables["Contacts"], "First");
80:     SurName.DataBindings.Add("Text", dataset.Tables["Contacts"], "Name");
81:     Title.DataBindings.Add("Text", dataset.Tables["Contacts"], "Title");
82:     Company.DataBindings.Add("Text", dataset.Tables["Contacts"], "Company");
83:     Phone.DataBindings.Add("Text", dataset.Tables["Contacts"], "Phone");
84: }
```

```
85:     //Constructor. Positions the form controls,
86:     //Initializes the dataset, binds the controls
87:     and
88:     //wires up the handlers.
89:     public datasetapp()
90:     {
91:         this.components = new
92:             System.ComponentModel.Container();
93:         this.Text = "datasetapp";
94:         this.AutoScaleBaseSize = new
95:             System.Drawing.Size (5, 13);
96:         this.ClientSize = new System.Drawing.Size (250,
97:             200);
98:         this.FormBorderStyle =
99:             FormBorderStyle.Fixed3D;
100:
101:        lbl_first = new Label();
102:        lbl_first.Text="First name";
103:        lbl_first.Location = new Point(5,5);
104:        lbl_first.Size = new Size(120,28);
105:        lbl_first.Anchor = AnchorStyles.Left |
106:            AnchorStyles.Right;
107:        Controls.Add(lbl_first);
108:
109:        FirstName = new TextBox();
110:        FirstName.Location = new Point(125,5);
111:        FirstName.Size = new Size(120,28);
112:        FirstName.Anchor =
113:            AnchorStyles.Left|AnchorStyles.Right;
114:        Controls.Add(FirstName);
115:
116:        lbl_name = new Label();
117:        lbl_name.Text="Surname";
118:        lbl_name.Location = new Point(5,35);
119:        lbl_name.Size = new Size(120,28);
120:        lbl_name.Anchor =
121:            AnchorStyles.Left|AnchorStyles.Right;
122:        Controls.Add(lbl_name);
123:
124:        SurName = new TextBox();
125:        SurName.Location = new Point(125,35);
126:        SurName.Size = new Size(120,28);
127:        SurName.Anchor = AnchorStyles.Left |
128:            AnchorStyles.Right;
129:        Controls.Add(SurName);
130:
131:        lbl_company = new Label();
132:        lbl_company.Text="Company";
133:
```



```
119:     lbl_company.Location = new Point(5,65);
120:     lbl_company.Size = new Size(120,28);
121:     Controls.Add(lbl_company);

122:     Company = new TextBox();
123:     Company.Location = new Point(125,65);
124:     Company.Size = new Size(120,28);
125:     Controls.Add(Company);

126:     lbl_title = new Label();
127:     lbl_title.Text="Title";
128:     lbl_title.Location = new Point(5,95);
129:     lbl_title.Size = new Size(120,28);
130:     Controls.Add(lbl_title);

131:     Title = new TextBox();
132:     Title.Location = new Point(125,95);
133:     Title.Size = new Size(120,28);
134:     Controls.Add(Title);

135:     lbl_phone = new Label();
136:     lbl_phone.Text="Telephone";
137:     lbl_phone.Location = new Point(5,125);
138:     lbl_phone.Size = new Size(120,28);
139:     Controls.Add(lbl_phone);

140:     Phone = new TextBox();
141:     Phone.Location = new Point(125,125);
142:     Phone.Size = new Size(120,28);
143:     Controls.Add(Phone);

144:     btnNew = new Button();
145:     btnNew.Location = new Point(5,155);
146:     btnNew.Size = new Size(70,28);
147:     btnNew.Text="New";
148:     btnNew.Click+=new EventHandler(OnNew);
149:     Controls.Add(btnNew);

150:     btnPrev = new Button();
151:     btnPrev.Location = new Point(80,155);
152:     btnPrev.Size = new Size(35,28);
153:     btnPrev.Text="<<";
154:     btnPrev.Click += new EventHandler(OnPrev);
155:     Controls.Add(btnPrev);

156:     btnEnd = new Button();
157:     btnEnd.Location = new Point(120,155);
158:     btnEnd.Size = new Size(70,28);
```

```
159:     btnEnd.Text = "End";
160:     btnEnd.Click += new EventHandler(OnEnd);
161:     Controls.Add(btnEnd);

162:     btnNext = new Button();
163:     btnNext.Location = new Point(200, 155);
164:     btnNext.Size = new Size(35, 28);
165:     btnNext.Text = ">>";
166:     btnNext.Click += new EventHandler(OnNext);
167:     Controls.Add(btnNext);

168:     InitDataSet();

169:     NewEntry();

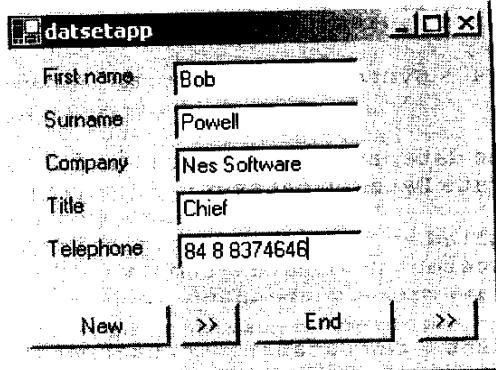
170:     BindControls();

171: }

172: //Cleans up the Form
173: public override void Dispose()
174: {
175:     base.Dispose();
176:     components.Dispose();
177: }

178: //Main method to instantiate and run the
application.
179: static void Main()
180: {
181:     Application.Run(new datasetapp());
182: }
183: }
184: }
```

Ví dụ 3.3-2 minh họa khá chi tiết. Đối tượng DataSet được khởi tạo c^{yếu} ở dòng 66. Ở đây, DataSet được tạo ra và một bảng mới được đặt vào tập h^{àng}. Bảng được tạo ra với một tập hợp các cột xác định bởi chuỗi "First", "Name" ... Việc thêm dữ liệu vào một bảng được thực hiện từng hàng tại một thời điểm. Phương thức ở dòng 48 thêm một hàng mới vào bảng và điền đầy nó với nội dung tùy ý. Bạn cũng có thể tạo ra một bảng với giá trị số hoặc những kiểu khác. Dòng 79-85 thực hiện ràng buộc dữ liệu trong bảng với thuộc tính Text của những ô khiền xác định. Giờ đây nó có thể dùng để hiệu chỉnh thông tin trong bảng, chúng sẽ hiển thị mỗi mục tự động khi bạn duyệt qua cơ sở dữ liệu. Các bộ xử lý event handler của điều khiển nút nhấn sẽ thực hiện việc duyệt dữ liệu. Chúng sử dụng thuộc tính ContextManager.Position để đánh chỉ mục các hàng trong bảng. H^{ình} 3.3-2 cho thấy màn hình của ứng dụng.



Hình 3.3-2 Ứng dụng ràng buộc Dataset

5. RÀNG BUÔC DỮ LIỆU PHỨC HỢP

Ràng buộc giản đơn là quá trình kết nối một điều khiển đơn tới một mục dữ liệu đơn. Ràng buộc phức hợp là ràng buộc một điều khiển đơn tới nhiều mục dữ liệu. Những điều khiển thường sử dụng cho dữ liệu phức tạp gồm có DataGridView, đối tượng này xuất hiện như một bảng biểu đơn giản. Điều khiển này có thể hiển thị, xóa các bản ghi từ một bảng. Thao tác ràng buộc dữ liệu đơn được dùng để gắn một điều khiển với DataTable.

DataGrid chỉ có thể sử dụng để hiển thị một bảng tại một thời điểm, nhưng nó cũng có thể dùng để hiển thị quan hệ giữa các bảng. Ví dụ 3.3-3 cho thấy phiên bản sửa đổi của ví dụ về ràng buộc giản đơn được trình bày trong 3.3-2. Trong ví dụ này ta loại bỏ tất cả các điều khiển có liên hệ với DataSet và thay thế chúng bằng một điều khiển lưới phức tạp hơn. Ràng buộc dữ liệu phức hợp chỉ thực sự phức tạp bên trong chính chức năng của nó, không phải trong cách sử dụng.

Ví dụ 3.3-3 gridbind.cs

```
1: namespace Sams
2: {
3:     using System;
4:     using System.Drawing;
5:     using System.Collections;
6:     using System.Data;
7:     using System.ComponentModel;
8:     using System.Windows.Forms;
9:     //This application shows simple data binding to a
10:    //programmatically created dataset.
11:    public class GridBind :
12:        System.Windows.Forms.Form
```

```
13:     private System.ComponentModel.Container
14:         components;
15:     private System.Windows.Forms.DataGrid
16:         dataGridView1;
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
```

private System.ComponentModel.Container
components;
private System.Windows.Forms.DataGrid
dataGridView1;

//The dataset used to store the table
private DataSet dataset;

//Called at creation to initialize the
//dataset and create an empty table
private void InitDataSet()
{
 dataset = new DataSet("ContactData");

 DataTable t=new DataTable("Contacts");

 t.Columns.Add("First",typeof(System.String));
 t.Columns.Add("Name",typeof(System.String));
 t.Columns.Add("Company",typeof(System.String))
);
 t.Columns.Add("Title",typeof(System.String));
 t.Columns.Add("Phone",typeof(System.String));

 t.MinimumCapacity=100;

 dataset.Tables.Add(t);
}

//Called at initialization to do complex binding
of the DataGridView
//to the dataset's "Contacts" table entries
private void BindGrid()
{
 this.dataGridView1.SetDataBinding(dataset.Tables[
 "Contacts"], "");
}

//Constructor. Positions the form controls,
//Initializes the dataset, binds the controls
and
//wires up the handlers.
public GridBind()
{
 InitializeComponent();

 InitDataSet();

 BindGrid();



```
45:      }
46:      //Cleans up the Form
47:      public override void Dispose()
48:      {
49:          base.Dispose();
50:          components.Dispose();
51:      }

52:      //Method added by the form designer
53:      private void InitializeComponent()
54:      {
55:          this.components = new
56:              System.ComponentModel.Container ();
56:          this.dataGrid1 = new
57:              System.Windows.Forms.DataGrid ();
58:          dataGrid1.BeginInit ();
58:          dataGrid1.Location = new System.Drawing.Point
59:              (8, 16);
59:          dataGrid1.Size = new System.Drawing.Size (472,
60:              224);
60:          dataGrid1.DataMember = "";
61:          dataGrid1.TabIndex = 0;
62:          this.Text = "GridBind";
63:          this.AutoScaleBaseSize = new
64:              System.Drawing.Size (5, 13);
64:          this.FormBorderStyle =
65:              System.Windows.Forms.FormBorderStyle.Fixed3D;
65:          this.ClientSize = new System.Drawing.Size (486,
66:              251);
66:          this.Controls.Add (this.dataGrid1);
67:          dataGrid1.EndInit ();
68:      }

69:      //Main method to instantiate and run the
70:      //application.
70:      static void Main()
71:      {
72:          Application.Run(new GridBind());
73:      }
74:  }

75: }
```

Dịch chương trình này với dòng lệnh sau:
csc /t:winexe gridbind.cs

Xem thông tin trong Ví dụ 3.3-3, bạn có thể thấy rằng mã để duyệt tập hợp dữ liệu không còn cần đến nữa. Phương thức để thêm một hàng dữ liệu mới cũng không cần. Điều khiển DataGridView làm tất cả điều này hộ bạn. Ràng buộc dữ liệu với khung lưới diễn ra ở dòng 35. Việc khởi tạo tập hợp dữ liệu hoàn toàn giống với ví dụ trước đây; nó được minh họa ở dòng 19-30. Hình 3.3-3 cho thấy ứng dụng GridBind khi hoạt động. Chú ý rằng khung lưới cung cấp cột và các dòng tiêu đề cho phép bạn chọn toàn bộ dòng hoặc sắp đặt bảng thứ tự theo alphabe bởi các cột.

Điều khiển DataGridView cũng có thể sử dụng để chỉ ra quan hệ phân cấp giữa nhiều bảng dữ liệu.

Chẳng hạn như một ứng dụng dạng Quản lý Tài nguyên Khách hàng (Customer Resource Management) sẽ cần đến mối quan hệ giữa một khách hàng với nhiều đơn đặt hàng của. Khi đó, ràng buộc dữ liệu phức hợp làm cho quá trình này trở nên dễ hơn.

Last	First	Title
Davolio	Nancy	Sales Repre
Fuller	Andrew	Vice Preside
Leverling	Janet	Sales Repre
Peacock	Margaret	Sales Repre
Buchanan	Steven	Sales Mana
Suyama	Michael	Sales Repre
King	Robert	Sales Repre

Hình 3.3-3 Kết quả ràng buộc dữ liệu phức hợp

Cho đến lúc này, bạn đã nhìn thấy cách ràng buộc dữ liệu với DataSets và DataTables được tạo ra trong đoạn mã trên. Chúng ta hãy xem qua một ví dụ thực tế cho thấy cách ràng buộc phức hợp trên cơ sở dữ liệu hiện hành. Các đối tượng mà chúng ta dùng để chứa và xử lý dữ liệu là thành quả do ADO.NET mang lại. Hệ thống này cho phép tương tác ở mức cao với các bảng cơ sở dữ liệu từ nhiều nguồn cung cấp dữ liệu khác nhau. ADO.NET có thể được sử dụng kết nối với OLEDB,, SQL, và những trình điều khiển dữ liệu khác.

Giờ chúng ta hãy xem qua một ví dụ về cách sử dụng kiểu ràng buộc cơ sở dữ liệu phức hợp.

6. RÀNG BUỘC THÀNH PHẦN ĐIỀU KHIỂN VỚI CƠ SỞ DỮ LIỆU SỬ DỤNG ADO.NET

ADO.NET sử dụng khái niệm mô hình ngắt kết nối với cơ sở dữ liệu, mô hình này sử dụng các bảng dữ liệu được đặt vào vùng đệm (cache) sẵn trên máy khách khi hiển thị và xử lý.

Thậm chí ngay cả khi cơ sở dữ liệu được lưu trên cùng một máy với chương trình hiệu chỉnh. ADO.NET sẽ sao chép một ảnh (snapshot) của dữ liệu vào đối tượng DataSet và chuyển những thay đổi ngược về cơ sở dữ liệu sau khi xử lý xong. Mô hình này được sử dụng nhằm cho phép các ứng dụng cơ sở dữ liệu được uyển chuyển và dễ thay đổi qua mạn hơn vì nó đáp ứng nhu cầu cho mở nhiều kết nối đồng thời truy vấn cơ sở dữ liệu. Các bảng dữ liệu rất có cấu trúc và có thứ bậc. Điều này làm cho XML trở nên tưởng cho việc di chuyển và lưu trữ dữ liệu.

Thật vậy, khi ADO tạo kết nối đến cơ sở dữ liệu, thông tin được chuyển sang dạng XML. Có nghĩa là phần mềm của bạn có thể tận dụng ưu điểm của mạng cục bộ Intranet và dữ liệu trên Internet được lưu trữ dễ dàng tương tự như cách nó lưu trữ trong cơ sở dữ liệu của máy cục bộ. Điều quan trọng cần nhớ đó là việc di chuyển một số lượng lớn dữ liệu đến và đi khỏi cơ sở dữ liệu trở thành nhu cầu cũng cần như việc cơ sở dữ liệu ngắt kết nối với máy khách. Mô hình hệ thống nhiều tầng thường tiếp xúc sử dụng với rất nhiều cơ sở dữ liệu nằm trên những vùng địa lý khác nhau và các dịch vụ Web thường hướng dữ liệu theo yêu cầu của trình khách. Điều này có nghĩa là các ứng dụng khách cần thử dùng những khối dữ liệu nhỏ có thể được cập nhật và làm mới lại nhanh chóng trên một bảng thông làm việc nhỏ hẹp. Vì lý do này, trách nhiệm của lập trình viên thường là phải phân chia và quản lý các lược đồ dữ liệu thực hiện trong giao tác. Một khả năng khác của ADO.NET đó là che dấu những công việc có thể làm bạn mệt nhọc và giúp bạn làm việc hiệu quả hơn với dữ liệu.

VisualStudioData được sao chép đến hoặc di từ DataSet bởi DataAdapter. ADO.NET phân chia hai kiểu DataAdapter chính là OleDbDataAdapter dùng kết nối DataSet với những nguồn cung cấp dữ liệu dạng OLE và SqlDataAdapter. Nhắm đến mục tiêu là các kết nối SQL server. Cả hai lớp tiếp hợp (adaptor) này sử dụng một tập hợp các đối tượng Command để thực hiện lựa chọn, cập nhật, chèn, và xoá nội dung cơ sở dữ liệu mà chúng nối kết đến. Lệnh Select lấy dữ liệu từ nguồn dữ liệu. Các lệnh Update, Insert, Delete thực hiện thay đổi dữ liệu dựa trên những thay đổi tác động vào DataSet. Điền vào Dataset những thông tin lấy từ cơ sở dữ liệu rất đơn giản. Các bước thực hiện được minh họa trong đoạn mã dưới đây:

```

1:  System.Data.OleDb.OleDbConnection Connection=
2:  new System.Data.OleDb.OleDbConnection();
3:  Connection.ConnectionString =
4:  @"Provider=Microsoft.Jet.OLEDB.4.0;" +
5:  @"Data Source=northwin.mdb;";
```



```
6:     System.Data.OleDb.OleDbCommand SelectCommand= new
      System.Data.OleDbCommand("Select * from Customers");
7:     SelectCommand.Connection=connection;
8:     System.Data.Dataset ds=new
      System.Data.Dataset("Customer Table");
9:     ds.Tables.Add("Customers");
10:    System.Data.OleDb.OleDbDataAdapter adapter=
11:    new
      System.Data.OleDb.OleDbDataAdapter(SelectCommand);
12:    adapter.Fill(ds);
13:    Console.WriteLine(ds.GetXml());
```

Đoạn mã này giả thiết rằng cơ sở dữ liệu NorthWind mang tên northwind.mdb có sẵn trong thư mục cục bộ. Các dòng 1 -5 tạo ra một OleDbConnection và tập hợp các chuỗi kết nối. Dòng 6 và 7 tạo ra SelectCommand và cho phép sử dụng chuỗi được chọn với câu truy vấn hợp lệ. Dòng 7 phối hợp kết nối cơ sở dữ liệu với lệnh select, trong khi dòng 8 tạo ra đối tượng DataSet và thêm vào đó một bảng.

Dòng 10 tạo ra một bộ tiếp hợp dữ liệu và thêm vào lệnh select. Dòng 1 diễn đầy dữ liệu thiết lập từ bộ tiếp hợp và cuối cùng, để chứng minh rằng nó thực sự làm được cái gì đó, dòng cuối cùng đổ toàn bộ bảng dữ liệu khách hàng của northwind từ DataSet thành dữ liệu XML.

Theo cách nhìn nhận về năng suất, bạn có lẽ sẽ thực hiện các thao tác như vậy thông qua sử dụng môi trường IDE của Visual Studio.NET để tạo ra ứng dụng. Ràng buộc dữ liệu sử dụng Visual Studio sinh ra một khối khổng lồ mã và làm cản công việc kết nối cơ sở dữ liệu bạn trở nên rất toàn diện. Nói chung cho đến bây giờ, chúng ta vẫn tránh sử dụng Visual Studio bởi vì bộ khung của những ứng dụng tạo ra theo cách này không phải là dễ đọc và đòi hỏi có tác dụng ngược, gây khó khăn cho bạn trong việc bổ sung và soạn thảo mã do Visual Studio phát sinh. Tuy nhiên, trong trường hợp này, cách học tốt nhất đó là duyệt qua toàn bộ khung nhìn của cơ sở dữ liệu sử dụng IDE và sau đó kiểm tra kết quả phản hồi của ứng dụng để xem nó làm cái gì.

7. TẠO RA MỘT KHUNG NHÌN CƠ SỞ DỮ LIỆU (DATABROWSER) VỚI VISUAL STUDIO VÀ ADO.NET

Bạn cần thực hiện cẩn thận những bước sau để tạo ra ứng dụng:

1. Tạo ra một dự án Windows Forms mới, đặt tên cho nó là DBBind
2. Kéo một DataGridView từ toolbox và định vị nó trên form. Chọn thuộc tính DataGridView và thiết lập thuộc tính Dock là Fill bằng cách kích chuột vào giữa nút nhấn trong cửa sổ soạn thảo dành cho thuộc tính Dock.
3. Từ thanh công cụ, chọn OleDbDataAdapter từ bảng Data. Khi đặt nó lên form, bạn sẽ nhìn thấy hộp thoại Data Adapter Configuration Wizard

4. Trên trang đầu tiên của màn hình trợ giúp Wizard, chọn cơ sở dữ liệu NorthWind. Có thể bạn cần nhấp vào mục New Connection, chọn Microsoft Jet 4.0 OLE DB Provider trong tab Provider. Nhấn Next, và duyệt tìm northwind.mdb. Khi bạn tìm thấy northwind.mdb, hoặc cơ sở dữ liệu nào đó mà bạn quen thuộc, nhấn nút Connection để bảo đảm rằng cơ sở dữ liệu có thể sử dụng được. Nhấn Ok trong hộp thoại New Connection và nhấn Next trên màn hình Wizard.
5. Chọn nút radio đánh dấu Use SQL Statements. Nhấn Next.
6. Nhập câu truy vấn mà bạn biết hoặc chọn bộ xây dựng truy vấn để tạo ra câu truy vấn từ các bảng trong cơ sở dữ liệu kết nối nội. Trong ví dụ trên, chúng ta sử dụng bộ xây dựng truy vấn, thêm vào bảng Customers từ NorthWind, chọn ô checkbox đánh dấu All Columns trên bảng hiện ra, và nhấn nút Ok trong hộp thoại.
7. Nhấn Next sẽ cho bạn thấy việc tạo bộ tiếp hợp và kết nối thành công, bạn có thể nhấn nút Finish lúc này, bạn sẽ có một form với hai đối tượng, một là OleDbDataAdapter và một là OleDbConnection trên khay biểu tượng (icon Tray).
8. Chọn OleDbDataAdapter trong khay biểu tượng. Dưới thuộc tính browser, bạn sẽ thấy một liên kết Generate DataSet. Nhấn vào liên kết này để tạo ra dataset.

Đơn giản chỉ cần kích OK trong hộp thoại theo sau; Tốt nhất là sử dụng các giá trị mặc định hiện có. Bây giờ bạn sẽ có ba biểu tượng trong khay.

9. Chọn DataGrid trên form lần nữa và hiệu chỉnh thuộc tính DataSource thông qua danh sách combo box. Bạn sẽ nhìn thấy một số nguồn dữ liệu. Nếu bạn tiếp tục làm theo và sử dụng cơ sở dữ liệu NorthWind, chọn DataSet1.Customers. Khi làm điều này, bạn sẽ thấy DataGrid trên form bắt đầu hiển thị tất cả cột tiêu đề cơ sở dữ liệu.
10. Bước cuối cùng được thực hiện thủ công. Bạn cần sửa đổi phương thức constructor của lớp đôi chút và thêm vào lệnh Fill. Thêm vào dòng mã sau lời gọi InitializeComponent trong phương thức khởi động constructor:

```
this.oleDbDataAdapter1.Fill(this.dataSet1);
```

Bây giờ bạn có thể biên dịch và chạy chương trình bằng cách nhấn F5.

Bạn sẽ thấy màn hình ứng dụng hiển thị như hình 3.3-4 nó cho phép bạn duyệt xem, sắp xếp và hiệu chỉnh, nói chung là mọi thao tác với bảng dữ liệu Customers của NorthWind.

Do để học tập, hãy trở lại bộ soạn thảo Visual Studio một lần nữa và tìm phương thức InitializeComponent. Nó có thể được đặt nằm ở nhánh folder nào đó và bạn phải bung rộng chúng ra. Xem kỹ mã lệnh thêm vào bởi DataAdapter Wizard để thiết lập kết nối OleDbAdapter cùng với bốn đối tượng lệnh khác. Khi

thực hiện qua các bước, bạn sẽ nhận được mã phát sinh, chúng tôi không liệt kê chúng ở đây.

Bạn có thể rõ ràng nhìn thấy cách thức trình Wizard và những bộ trợ giúp Helper hướng dẫn bạn sử dụng ADO.NET ngay trên đầu ngón tay.

Address	City	CompanyName	ContactName	Country	CustomerID
Obere Str. 57	Berlin	Alfreds Futterkiste	Sales Repres	Germany	ALFKI
Avda. de la Constitución 9101	México D.F.	Ana Trujillo Emparedados y helados	Owner	Mexico	ANATR
Mataderos 2	México D.F.	Antonio Moreno Trujillo Emparedados y helados	Owner	Mexico	ANTOIN
120 Hanover	London	Around the Horn	Sales Repres	UK	AROU
Berguvsvägen 22	Luleå	Berglunds snabbköp	Order Admini	Sweden	BERGS
Forsterstr. 57	Mannheim	Blauer See Delicatessen	Sales Repres	Germany	BLAUS
24, place Kléber	Strasbourg	Blondes Dsслы	Marketing Mgr	France	BLONDI
C/ Araquil, 67	Madrid	Bólido Comidas Preparadas	Owner	Spain	BOLID
12, rue des Bouchers	Marseille	Bon app'	Owner	France	BONAI
23 Tsawassen	Tsawassen	Bottom-Dollar Markets	Accounting Mgr	Canada	BOTTI
Fauntleroy Ci	London	B's Beverages	Sales Repres	UK	BSBEV

Hình 3.3-4 Kết quả ràng buộc phức hợp trong khung lưới

8. KẾT CHƯƠNG

Trong chương này, chúng ta đã xem qua những nguyên tắc cơ bản của ràng buộc dữ liệu trong ADO.NET. Bạn cũng đã thấy sức mạnh của các điều khiển .NET có thể giúp chúng ta làm việc để tạo ra một môi trường có hiệu suất cao. Tiếp đến, bạn cũng đã nhìn thấy cách ràng buộc DataGridView làm việc ở cả mô hình cột lặp các bảng cục bộ và với cơ sở dữ liệu đầy đủ thông qua ADO.NET. Chúng ta sẽ tiếp tục quan tâm đến Windows Form trong chương 3.4 khi xây dựng ứng dụng thực tế mang tên "Scribble.NET" nơi bạn sẽ có điều kiện học cách sử dụng các thành phần đồ họa để trang trí giao diện cho ứng dụng.

Chương 3.4

XÂY DỰNG ỨNG DỤNG WINDOWS FORMS (SCRIBBLE .NET)

Các vấn đề chính sẽ được đề cập đến:

- ✓ Các tài nguyên trong .NET
- ✓ Địa phương hóa (Localization) một cách dễ dàng
- ✓ Các lớp (class) quản lý tài nguyên của .NET
- ✓ Tạo lập tài nguyên văn bản (text)
- ✓ Sử dụng Visual Studio.NET để Quốc tế hóa (Internationalization)
- ✓ Các tài nguyên hình ảnh (image)
- ✓ Sử dụng các danh sách hình ảnh (Image List)
- ✓ Phương cách lập trình truy xuất đến các tài nguyên
- ✓ Đọc và viết tập tin XML RESX

1. CÁC TÀI NGUYÊN TRONG .NET

Không giống như Win32 và MFC, các ứng dụng .NET không phụ thuộc nặng nề vào các tài nguyên cho hộp thoại (dialog) và trình bày khuôn mẫu (form layout). Tuy nhiên, các tài nguyên cần thiết cho việc địa phương hóa ứng dụng, việc tích hợp các hình ảnh và biểu tượng, và cho việc sử dụng các dữ liệu của người dùng lại dễ dàng được thay đổi mà không cần đến việc biên dịch lại một ứng dụng.

Các mẫu tài nguyên (resource model) trong .NET là các tài nguyên mặc định và một tập hợp tùy chọn các tài nguyên bổ sung (additional resource) cung cấp khả năng địa phương hóa, hoặc quốc tế hóa chính xác hơn, hoặc các dữ liệu cho một nền văn hóa xác định nào đó. Ví dụ: một ứng dụng nên có một tài nguyên mặc định về văn hóa Anh – theo kiểu Mỹ (US – English) nhưng cũng cần địa phương hóa Anh – theo kiểu truyền thống (UK - English) bởi sự khác nhau về biểu tượng của tiền tệ, hoặc Pháp, nơi mà các chuỗi ký tự trên thanh thực đơn (menu) phải sử dụng ngôn ngữ phù hợp (tiếng Pháp, chứ không phải tiếng Anh).

Các tài nguyên mặc định thì thường được đóng gói (assembly) trong các đoạn mã chương trình. Các gói kết hợp assembly chứa các tài nguyên có thể được tải lên và sử dụng khi cần. Phương cách này tương tự như việc sử dụng các tài nguyên DLL trong Win32 và MFC.

Một ứng dụng tốt cần luôn được địa phương hóa và nên luôn cung cấp các tài nguyên cho bất kỳ nền văn hóa nào mà phần mềm ấy có thể được sử dụng. Nó cũng có nghĩa là không phải tất cả các ứng dụng đều cần phải là “công dân của



toàn cầu”, do đó bạn nên lựa chọn mức độ hỗ trợ ngôn ngữ, mức độ này tuỳ thuộc vào kênh phân phối sản phẩm và nhu cầu của người dùng mà bạn nhắm đến.

2. ĐỊA PHƯƠNG HOÁ SẢN PHẨM MỘT CÁCH DỄ DÀNG

Cách tốt nhất để hiểu về tính địa phương hóa là thực hành xây dựng một ứng dụng. Điều quan trọng nhất mà bạn có thể làm để trước tiên đó là nhớ rằng đừng nhúng bất kỳ chuỗi nào vào trong các đoạn mã chương trình của bạn. Ví dụ, hãy dùng bao giờ gõ:

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

Bởi vì khi đó bạn đã nhúng chuỗi “Hello World!” vào trong mã chương trình và nó sẽ làm cho chương trình cứng nhắc. Để làm cho chương trình trở thành “công dân của toàn cầu”, bạn có thể tạo lập một tài nguyên mặc định với một chuỗi được gán giá trị. Ví dụ:

```
MessageBox.Show((String)resources.GetObject(  
    "MyHelloWorldResourceString"));
```

Trong dòng trên, các ký tự thực sự của thông điệp không được đặt ở trong các dòng mã và được tải lên từ một tài nguyên chuỗi ký tự có định danh mang tên “MyHelloWorldResourceString”.

Nếu nền văn hoá mặc định của bạn là Anh – kiểu Mỹ, chuỗi sẽ mang nội dung “Hello World!”. Nếu ứng dụng cần được hỗ trợ trong nước Pháp, một gói nguồn tài nguyên giành cho ngôn ngữ tiếng Pháp sẽ có chuỗi tương ứng với định danh “MyHelloWorldResourceString” nhưng lại có giá trị khác là “Bonjour Le Monde!” chẳng hạn.

3. CÁC LỚP QUẢN LÝ TÀI NGUYÊN CỦA .NET

Bộ khung Framework .Net cung cấp một họ các lớp được thiết kế để làm cho việc quản lý các tài nguyên trở nên dễ dàng và chắc chắn. Các lớp đó là:

ResourceManager Dùng để quản lý tất cả các tài nguyên của ứng dụng.

ResourceSet Trình bày một tập hợp các tài nguyên của một nền văn hoá xác định.

ResourceReader Là lớp thi hành cho giao tiếp IResourceReader. Nó được dùng để đọc tài nguyên từ một luồng dữ liệu (stream).

ResourceWriter Là lớp thi hành cho giao tiếp IResourceWriter. Nó được dùng để ghi tài nguyên vào một luồng dữ liệu (stream).

Ứng dụng của bạn phải dùng một ResourceManager để tải các gói tài nguyên và làm cho chúng sẵn sàng cho phần còn lại của chương trình. ResourceManager sẽ trả về ResourceSet cho mỗi văn hoá tương ứng được yêu cầu.

3.1. Tìm một văn hoá

Các nền văn hoá và ngôn ngữ khác nhau được định danh thông qua lớp CultureInfo. Lớp này chứa những thông tin đơn giản về nền văn hoá mà người dùng lựa chọn khi anh ta hay cô ta cài đặt hệ điều hành. Nó bao gồm thông tin về ngôn ngữ, lịch, tiền tệ, và các thông số khác có thể được lựa chọn. Phần mềm của bạn sẽ truy xuất các thông tin này và biết được nó có khả năng cung cấp các tài nguyên đặc trưng về văn hoá đó hay phải dùng các giá trị mặc định.

Những đặc trưng về văn hoá là cực kỳ quan trọng đối với thông điệp mà bạn cố gắng để truyền tải. Những điều dưới đây sẽ minh họa tại sao.

Một chút về văn hoá các nước

Một nhà máy xà bông lớn in quảng cáo về một đồng quần áo bẩn đang được bỏ vào máy giặt. Bức hình thứ hai cho thấy bột giặt đang được đổ vào máy giặt. Và bức thứ ba cho thấy quần áo sạch sẽ được lấy ra khỏi máy. Thế nhưng, nhà máy không hiểu được vì sao số lượng sản phẩm bị sứt giảm nghiêm trọng ở vài nước cho đến khi biết rằng số lượng bán được thấp nhất là ở những nước có văn hoá đọc từ phải sang-trái. Mẫu quảng cáo ấy, được làm bởi một người Mỹ, cho thấy quần áo sạch, sau khi giặt với bột giặt, thì trở nên bẩn thỉu và đang được lấy ra khỏi máy. Đừng bao giờ giả sử điều gì mà bạn chưa khảo sát kỹ.

Các thông tin văn hoá về ngôn ngữ được cung cấp bởi một mã 2-ký-tự và một tuỳ chọn định danh vùng đi cùng với mã đó. Ví dụ, tiếng Anh nói chung được biểu thị bằng hai ký tự “en”. Thế nhưng, bởi vì có những sự khác nhau trong việc dùng tiếng Anh – tiếng Anh của nước Anh, Canada, Mỹ hay những nơi khác – được đi kèm bởi những mã phụ (subcode) như GB, CA, US và những mã khác.

Ví dụ 3.4-1 dưới đây chỉ ra cách mà một chương trình sẽ lấy CultureInfo trên máy tính của bạn và nhờ đó dùng đúng ngôn ngữ và vùng được áp dụng cho máy của bạn. Nó sẽ hiển thị tên đầy đủ của văn hoá và lịch mà nền văn hoá của bạn đang sử dụng.

Ví dụ 3.4-1 culture.cs: Hiển thị Thông tin Văn hoá Hiện tại

```
//culture.cs
1: namespace Sams
2: {
3:     using System;
4:     using System.Drawing;
5:     using System.Collections;
6:     using System.ComponentModel;
7:     using System.Globalization;
```



```
8:     using System.Windows.Forms;
9:     using System.Data;
10:
11:    class CultureInfoTest : Form
12:    {
13:
14:        public CultureInfoTest()
15:        {
16:            this.Size = new Size(300, 200);
17:
18:            Label l = new Label();
19:            l.Location = new Point(3, 5);
20:            l.Size = new Size(294, 190);
21:
22:            InputLanguage inL =
InputLanguage.CurrentInputLanguage;
23:
24:            CultureInfo info = inL.Culture;
25:
26:            l.Text = String.Format("Culture identifier =
{0}\n"+
27:                "Display name = {1}\n"+
28:                "Calender = {2}\n",
29:                info.ToString(),
30:                info.DisplayName,
31:                info.Calendar.ToString());
32:            this.Controls.Add(l);
33:
34:        }
35:        static void Main()
36:        {
37:            Application.Run(new CultureInfoTest());
38:        }
39:    }
40:
41: }
```

4. TẠO LẬP CÁC TÀI NGUYÊN VĂN BẢN

Có nhiều cách để bạn có thể tạo ra tài nguyên văn bản cho chương trình của bạn. Cách đơn giản nhất là tạo một tập tin văn bản có một tập hợp các tên – giá trị (name – value). Bằng cách này chúng ta sẽ dễ dàng chọn ra giá trị của tài nguyên văn bản liên kết với tên mà bạn dùng trong mã chương trình của bạn.

Khi hoàn tất việc tạo tập tin, bạn phải chuyển nó thành một tập tin tài nguyên .resx hay resource bằng cách dùng công cụ RESGEN.EXE của .NET. dù tiếp sau sẽ chỉ rõ cách làm.

4.1. Chuỗi tài nguyên văn bản

Tập tin văn bản đơn giản sau đây cho thấy rõ một tập các cặp tên – giá trị cho một tài nguyên sẽ được dùng làm các chuỗi mặc định cho một ứng dụng. Lưu ý rằng dấu nháy không cần thiết để biểu diễn chuỗi. Nếu dùng dấu nháy, nó sẽ bị hiểu là một thành phần của chuỗi.

```
#Default culture resources
WindowText = Internationalization example
LabelText = Hello World!!!
```

Tập tin văn bản có thể được chuyển thành một trong hai dạng: dạng cơ sở XML (XML-based) được lưu trong tập tin .resx hoặc trực tiếp thành dạng đã được biên dịch .Resources. Tập tin .resx là một tập dữ liệu XML được dùng để nối tiếp các thông tin tài nguyên trong quá trình phát triển. Sự chuyển đổi dạng của tài nguyên được thực hiện nhờ công cụ Resgen. Gõ những dòng trên và lưu thành tập tin firstresource.txt. Sau đó bạn tạo thành tập tin .resx bằng cách dùng lệnh sau:

```
Resgen firstresource.txt firstresource.resx
```

Ứng dụng sẽ biên dịch tập tin trên và báo cho bạn biết rằng hai tài nguyên đã được chuyển đổi. Ví dụ 3.4-2 sẽ giới thiệu tập tin XML.

Ví dụ 3.4-2 firstresource.resx: Tập tin Tài nguyên đã được chuyển sang dạng XML

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <xsd:schema id="root" targetNamespace="" xmlns="" xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:element name="data">
      <xsd:complexType mixed="true">
        <xsd:all>
          <xsd:element name="value" minOccurs="0" type="xsd:string"/>
        </xsd:all>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="type" type="xsd:string"/>
        <xsd:attribute name="mimetype" type="xsd:string"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="resheader">
      <xsd:complexType mixed="true">
        <xsd:all>
          <xsd:element name="value" minOccurs="0" type="xsd:string"/>
        </xsd:all>
```

```
<xsd:attribute name="name" use="required">
  type="xsd:string"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="root" msdata:IsDataSet="true">
  <xsd:complexType>
    <xsd:choice maxOccurs="unbounded">
      <xsd:element ref="data"/>
      <xsd:element ref="resheader"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
<data name="WindowText">
  <value>Internationalization example</value>
</data>
<data name="LabelText">
  <value>Hello World!!!</value>
</data>
<resheader name="ResMimeType">
  <value>text/microsoft-resx</value>
</resheader>
<resheader name="Version">
  <value>1.0.0.0</value>
</resheader>
<resheader name="Reader">
  <value>System.Resources.ResXResourceReader</value>
</resheader>
<resheader name="Writer">
  <value>System.Resources.ResXResourceWriter</value>
</resheader>
</root>
```

Bạn có thể thấy trong ví dụ 3.4-2 rằng có thể đây không phải là tập tin bạn muốn gõ bằng tay. Phần thứ nhất của tập tin là là một lược đồ (schema) nghĩa nội dung của tập tin XML. Mỗi thành tố (element) được chứa trong một (node) XML. Ví dụ:

```
<data name="some-name">
```

Nếu bạn nhìn một lược hết tập tin, bạn sẽ thấy hai tài nguyên được định trong tập tin văn bản và giá trị của nó. Đây là dạng .resx của tài nguyên Visual Studio duy trì khi bạn tạo lập tài nguyên trong ứng dụng của mình.

Để dùng tập tin tài nguyên trong chương trình mẫu, bạn cần chuyển sang dạng đã được biên dịch. Bạn có thể tạo ra nó từ tập tin văn bản hay tệp XML bằng cách dùng một trong các dòng lệnh sau:

```
Resgen firstresource.txt firstresource.resources
```

Hay:

Resgen firstresource.resx firstresource.resources

4.2. Một chương trình mẫu dựa trên tài nguyên

Để thử tập tin tài nguyên trong ví dụ 3.4-2, bạn có thể tạo một ứng dụng "Hello World" đơn giản như ví dụ 3.4-3. Chương trình này dựa trên tập tin firstresource.resources đã được tạo ra trước đây để cung cấp văn bản cho nhãn (label) và thanh tiêu đề của cửa sổ:

Ví dụ 3.4-3 hellores.cs: Sử dụng chuỗi tài nguyên đơn giản

```
1: namespace Sams
2: {
3:     using System;
4:     using System.Drawing;
5:     using System.Collections;
6:     using System.ComponentModel;
7:     using System.Globalization;
8:     using System.Windows.Forms;
9:     using System.Data;
10:    using System.Resources;
11:
12:    class hellores : Form
13:    {
14:
15:        public hellores()
16:        {
17:            ResourceManager rm=new
18:                ResourceManager("firstresource",
19:                    this.GetType().Assembly);
20:
21:            this.Size = new Size(400,100);
22:            this.Text=rm.GetString("WindowText");
23:
24:            Label l = new Label();
25:            l.Location = new Point(3,5);
26:            l.Size = new Size(394,90);
27:            l.Font = new
28:                Font("Tahoma",36F,FontStyle.Bold);
29:            l.Text=rm.GetString("LabelText");
30:            this.Controls.Add(l);
31:
32:        }
33:        static void Main()
34:        {
35:            Application.Run(new hellores());
36:        }
37:    }
38: }
```



```
34:     }
35: }
36:
37: }
```

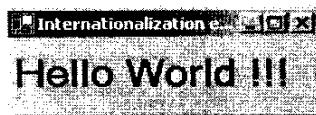
Đoạn mã trong ví dụ 3.4-3 là một ứng dụng “công dân toàn cầu” tốt. Nó không nhúng bất kỳ chuỗi nào mà vẽ tất cả văn bản từ tài nguyên được cung cấp. Dòng thứ 17 cho ta thấy một bộ quản lý tài nguyên đã được tạo ra như thế nào. Dòng 20 và 26 chỉ ra cách mà một chuỗi ứng với một tên được tải lên từ tài nguyên của bộ gói assembly được đưa vào các thành tố của Windows Forms như thế nào.

Để dịch tập tin này và chạy nó, bạn sẽ cần gõ vào dòng lệnh sau từ dấu nhắc lệnh:

```
Csc /out:heliores.exe /t:winexe
/res:firstresource.resources heliores.cs
```

Lưu ý rằng tham số /res: của dòng lệnh dùng để nhúng dạng đã dịch rồi của tài nguyên đã được tạo ra trước đó.

Chạy chương trình sẽ cho ta kết quả như hình 3.4-1.



Hình 3.4-1 Chương trình đang chạy

4.3. Tạo lập và sử dụng gói kết hợp

Ví dụ toàn cầu trên chưa sử dụng bộ tài nguyên thay thế. Trong ví dụ tiếp sau, ta sẽ thêm vào việc dùng gói tài nguyên chứa tập hợp tài nguyên liên quan đến ngôn ngữ tiếng Pháp.

Ghi chú: Gói kết hợp là một file thư viện *DLL* (*Dynamic Link Library*), chứa mã chương trình hoặc dữ liệu, được dùng bởi một gói chính của chương trình.

Một gói kết hợp, trong trường hợp này, là một DLL chỉ chứa dữ liệu. Các gói kết hợp dành cho việc địa phương hóa thường là của riêng mỗi ứng dụng. Chúng có thể được lưu trong những thư mục con nào đó của thư mục chứa chương trình chính. Trong trường hợp này, bạn sẽ cần tạo ra một thư mục gọi là “fr” để bộ quản lý tài nguyên có thể tìm thấy nó.

Đầu tiên, tạo ra một tài nguyên văn bản dùng cùng tên nhưng có các chuỗi giá trị khác. Nhớ rằng, ở đây chỉ có văn bản của nhãn (label) được thay đổi. Điều này sẽ được giải thích ngắn gọn.

#Version Francaise.

LabelText = Bonjour le monde!!!

Gọi tập tin này là firstresource.fr.txt. Sau khi hoàn tất tập tin, ta có thể tạo ra một gói kết hợp bằng cách gọi dòng lệnh sau:

```
Resgen firstresource.fr.txt firstresource.fr.resource
Al /out:fr/hellores2.Resources.DLL /c:fr /embed:
firstresource.fr.resources
```

Chương trình ứng dụng được dùng ở dòng thứ 2 là Bộ liên kết Gói (AL - Assembly Linker). AL có thể được dùng để tạo ra gói cho tất cả các dạng, nhưng ở đây chúng ta chỉ quan tâm đến khả năng đóng gói tài nguyên của nó. Lưu ý rằng tham số /out: đặt gói hay file .DLL vào thư mục con "fr".

Bây giờ ta có thể dịch ra chương trình hellores2.exe. Nó hoàn toàn tương tự như nguyên bản ngoại trừ có tùy chọn dòng lệnh cho phép ta chọn nền văn hoá bằng cách gõ vào định danh của nó. Ví dụ 3.4-4 chỉ rõ điều này.

Ví dụ 3.4-4 hellores2.cs: Địa phương hóa ví dụ Hello World

```
1: namespace Sams
2: {
3:     using System;
4:     using System.Drawing;
5:     using System.Collections;
6:     using System.ComponentModel;
7:     using System.Globalization;
8:     using System.Windows.Forms;
9:     using System.Data;
10:    using System.Resources;
11:    using System.Threading;
12:
13:    class hellores : Form
14:    {
15:
16:        private Label l;
17:        private ResourceManager rm;
18:
19:
20:        public hellores(string culture)
21:        {
22:            Thread.CurrentThread.CurrentCulture = new
23:                CultureInfo(culture);
24:
25:            rm=new ResourceManager("firstresource",
26:                this.GetType().Assembly);
27:
28:            this.Size = new Size(400,100);
29:            this.Text=rm.GetString("WindowText");
```



```
28:         l = new Label();
29:         l.Location = new Point(3, 5);
30:         l.Size = new Size(394, 90);
31:         l.Font = new
32:             Font("Tahoma", 36F, FontStyle.Bold);
33:             l.Text=rm.GetString("LabelText");
34:             l.Anchor = (AnchorStyles.Top |
35:                 AnchorStyles.Left |
36:                 AnchorStyles.Bottom |
37:                 AnchorStyles.Right);
38:             this.Controls.Add(l);
39:
40:     }
41:
42:     static void Main(string[] args)
43:     {
44:         string culture = "";
45:         if(args.Length == 1)
46:             culture = args[0];
47:             Application.Run(new hellores(culture));
48:     }
49: }
50:
51: }
```

Có một vài thay đổi nhỏ trong ví dụ 3.4-4. Dòng 22 thiết lập giao diện văn hoá hiện hành dựa trên các giá trị được nhập từ dòng lệnh. Các thuộc tính của nhãn được sửa đổi trên dòng 34-37 do đó kích cỡ của nhãn sẽ được thay đổi cùng với form. Trong ví dụ này, bạn sẽ cần thay đổi kích thước form để thấy được toàn bộ nội dung của nhãn. Điều này là một điểm của việc quốc tế hoá mà bạn phải làm quen: kích cỡ vật lý của tài nguyên có thể khác nhau giữa các nền văn hoá. Bạn cũng có thể lưu lại thông tin về kích cỡ và vị trí của tài nguyên ở trong gói các tài nguyên. Visual Studio sẽ làm tất cả những điều đó cho bạn, và chúng ta sẽ thấy ngay sau đây. Mặt khác, việc dùng tài nguyên chuỗi cũng sẽ tương tự.

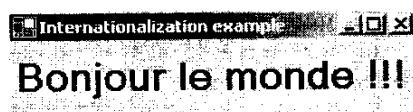
Ứng dụng có thể được biên dịch dựa vào lệnh sau:

```
csc /out:hellores2.exe /t:winexec
/res:firstresource.resources hellores2.cs
```

Bây giờ ta có thể chạy chương trình với chọn lựa văn hoá "fr" như sau:

```
hellores2 fr
```

Trong trường hợp này, bạn sẽ thấy rằng ứng dụng đưa ra màn hình chuỗi tài nguyên tiếng Pháp như hình 3.4-2.

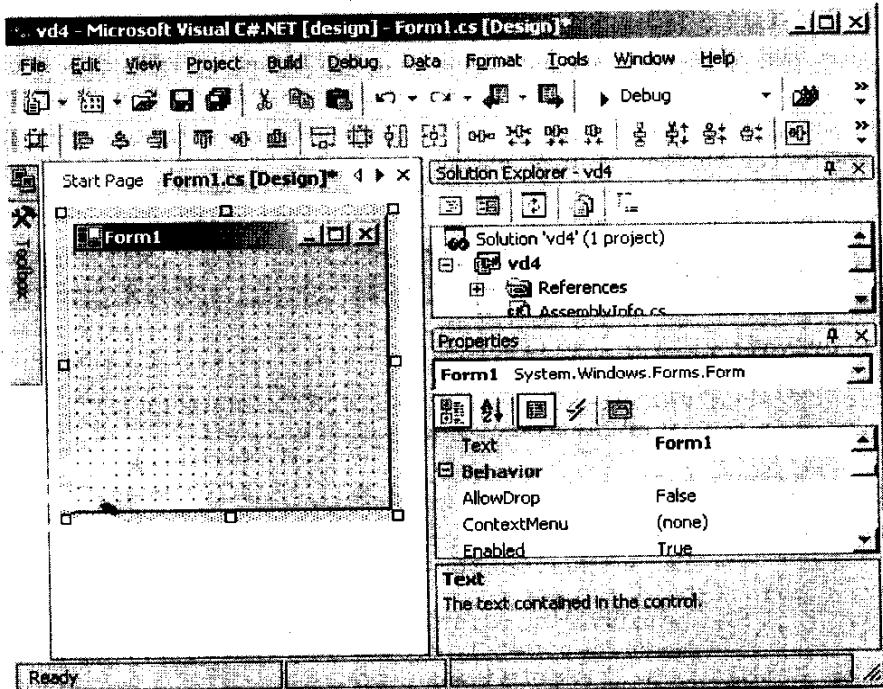


Hình 3.4-2 Liên kết với văn hóa ngôn ngữ Pháp

Đó là lý do để chỉ chỉnh sửa một tài nguyên trong gói kết hợp. Nó minh hoạ rằng bộ quản lý tài nguyên dùng tài nguyên văn bản và các tài nguyên khác phù hợp với địa phương và văn hoá nếu có thể. Còn không, nó sẽ trở lại dùng các thông tin mặc định. Chuỗi "WindowText" không có trong gói kết hợp. Bởi vậy, mặc định đã được dùng.

5. DÙNG VISUAL STUDIO.NET ĐỂ QUỐC TẾ HÓA

Visual Studio.NET sẽ quản lý tài nguyên cho bạn và giúp bạn tạo các form và thành phần (components) được địa phương hoá. Hình 3.4-3 minh họa thuộc tính Localizable được thiết lập cho một form. Thiết lập thuộc tính này thành True sẽ làm cho môi trường thiết kế lưu lại các thông tin cần thiết trong các tập tin tài nguyên cho bạn.



Hình 3.4-3 Thiết lập các thông số địa phương hóa



Phương thức (method) InitializeComponent của Form1 sẽ chứa các mã khởi tạo một bộ quản lý tài nguyên và tải bất kỳ một tài nguyên nào cần khi ứng dụng được chuyển giữa các địa phương từ tài nguyên.

Như bạn đã thấy trong ví dụ ở hình 3.4-2, việc dịch chuỗi văn bản làm cho chiều dài vật lý của chuỗi rộng ra, bởi vậy ta sẽ cần lưu thông tin về vị trí và kích cỡ trong tài nguyên. Phương thức InitializeComponent của Form1 được minh họa ở ví dụ 3.4-5.

Ví dụ 3.4-5 Form1.cs: Phương thức InitializeComponent

```
1: private void InitializeComponent()
2: {
3:     System.Resources.ResourceManager resources =
4:         new System.Resources.ResourceManager(
5:             typeof(Form1));
6:     this.AutoScaleBaseSize = new
7:         System.Drawing.Size(5, 13);
8:     this.ClientSize = ((System.Drawing.Size)(
9:         resources.GetObject(": this.ClientSize")));
10:    this.Text = resources.GetString(": this.Text");
11: }
```

Bạn có thể thấy rằng môi trường soạn thảo IDE đã tạo ra bộ quản lý tài nguyên mới ở dòng 3, và tải văn bản cùng kích thước tài nguyên ở dòng 6 và 7.

Khi một form trở thành có khả năng địa phương hóa, tất cả các thành phần (component) bạn đặt trên form đó sẽ cũng được lưu giá trị văn bản, thông tin kích cỡ và vị trí vào một tập tin tài nguyên. Ví dụ 3.4-4 minh họa phương thức InitializeComponent sau khi được gắn thêm một nhãn vào Form1.

Ví dụ 3.4-4 Form1.cs: Phương thức InitializeComponent sau khi gắn thêm một nhãn

```
1: private void InitializeComponent()
2: {
3:     System.Resources.ResourceManager resources =
4:         new System.Resources.ResourceManager(
5:             typeof(Form1));
6:     this.label1 = new System.Windows.Forms.Label();
7:     this.label1.Location =
8:         ((System.Drawing.Point)
9:             (resources.GetObject("label1.Location")));
10:    this.label1.Size =
11:        ((System.Drawing.Size)
12:            (resources.GetObject("label1.Size")));
13:    this.label1.TabIndex = ((int)
14:        (resources.GetObject("label1.TabIndex")));
15:    this.label1.Text =
```

```

        resources.GetString("label1.Text"));
13:     this.AutoScaleBaseSize = new
           System.Drawing.Size(5,13);
14:     this.ClientSize =
           (System.Drawing.Size)(
               resources.GetObject(": this.ClientSize")));
15:     this.Controls.AddRange(new
           System.Windows.Forms.Control[] {
16:         this.Text = resources.GetString(": this.Text");
17:     });
18: }

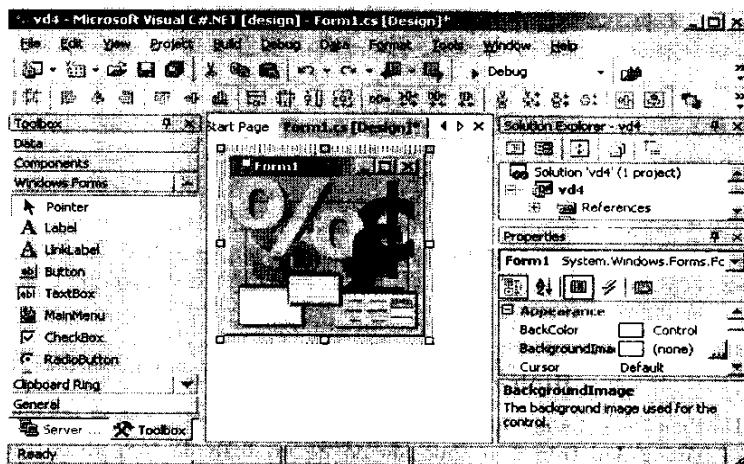
```

Bây giờ, thành phần nhãn của form đã lưu vị trí, kích cỡ, và văn bản vào tài nguyên.

6. TÀI NGUYÊN HÌNH ẢNH

Vậy là chúng ta đã làm việc với các tài nguyên văn bản, thế nhưng, cũng như Win32, các ứng dụng .NET cho phép lưu hình ảnh như bitmap, biểu tượng, hình nền, ... vào trong tài nguyên. Sau đó, chúng ta biên dịch các tài nguyên này ở dạng .resource. Tuy nhiên, nếu bạn tạo chúng bằng VisualStudio hoặc bằng tay, nó được lưu ở dạng XML .resx. Dĩ nhiên, việc chỉnh sửa tập tin hình ảnh như là văn bản trong XML thì không phải là việc vui vẻ gì. Sự lựa chọn nên là đặt hình ảnh trong các tài nguyên thông qua các công cụ soạn thảo ảnh như của VS.NET, cung cấp bởi Microsoft.

Một thành phần component, ví dụ, sẽ có hình và nền. Bạn có thể đặt thành phần component lên form và chỉnh sửa ảnh nền của thành phần dễ dàng. Trình soạn thảo sẽ cho phép bạn lựa chọn một hình ảnh mà sau này sẽ được tự động đặt trong tài nguyên cho bạn. Hình 3.4-4 sẽ minh họa vấn đề này.



Hình 3.4-4 Đặt một ảnh nền trên form

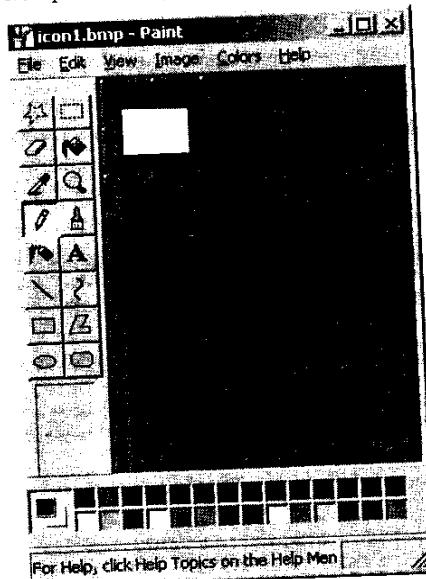
7. SỬ DỤNG DANH SÁCH HÌNH ẢNH

Rất nhiều thành phần component cần hình ảnh cho giao diện của mình. Các lớp ListView và TreeView có thể dùng hình ảnh để nâng cấp diện mạo của nó. Danh sách hình ảnh là một tập hợp của các ảnh bitmap được giữ trong một tập tin đơn. Những hình ảnh này có thể có một nền trong suốt (transparent background) đối với một màu nào đó. Khi đó, chỉ có một phần chính của hình ảnh được nhìn thấy.

Chúng ta sẽ cùng xem đoạn mã minh họa cho phần này. Sử dụng Visual Studio, bạn tạo một ứng dụng C# và đặt tên là imagelistdemo. Lần lượt làm theo từng bước cho đến cuối chương để hoàn tất công việc này.

Để tạo một danh sách hình ảnh với Visual Studio, bạn nên dùng chuột kéo (drag) một thành phần ImageList từ hộp công cụ (toolbox) lên form. Bây giờ hãy tạo ra một ít hình ảnh. Nên dùng Ms Paint, chỉ cần nhớ lưu hình ảnh trong một thư mục chung nào đó. Hình 3.4-5 hiển thị một biểu tượng 16x16 pixel đang được chỉnh sửa trong Ms Paint.

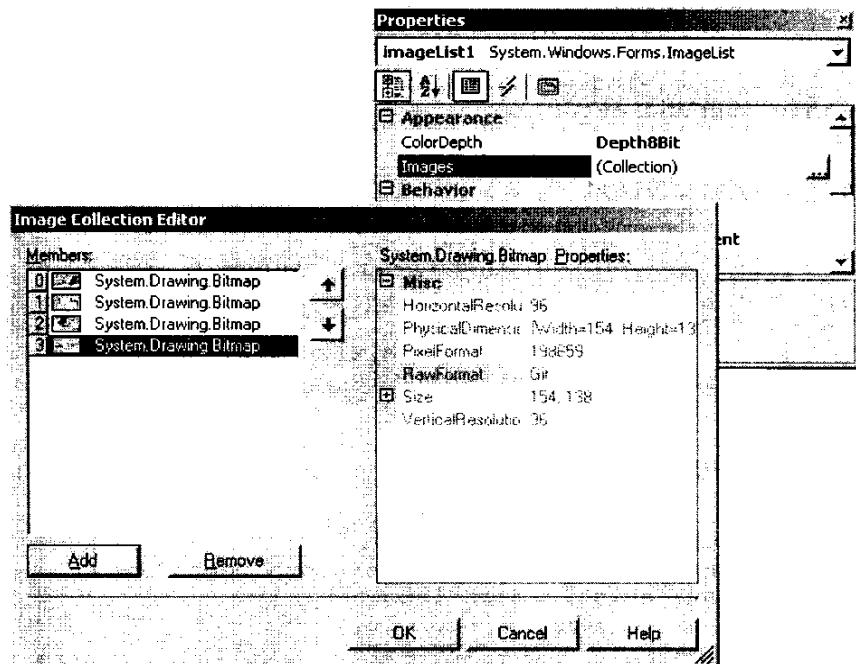
Lưu ý: Để chuẩn bị cho ví dụ này, chúng tôi đã chuẩn bị bốn biểu tượng nhỏ, từ icon1.bmp đến icon4.bmp



Hình 3.4-5 Chỉnh sửa một hình ảnh

Nền của biểu tượng được tô bởi màu hồng (magenta). Màu này sẽ không hiển thị bất cứ đâu trên ảnh thực, do đó nó an toàn để làm khoá trong suốt (transparent key). Để thêm những hình ảnh này vào trong đối tượng ImageList, (transparent key). Để thêm những hình ảnh này vào trong đối tượng ImageList, bạn có thể chọn tập hợp Images trong bảng thuộc tính của đối tượng và click vào

nút Browse. Một hộp thoại sẽ hiện ra như hình 3.4-6. Bạn có thể dùng nó để thêm vào những hình ảnh mà bạn cần. Lưu ý rằng, phải nhớ giữ cho tất cả hình ảnh trong cùng một danh sách hình ảnh có cùng kích thước, ví dụ như 16x16.



Hình 3.4-6 Thêm ảnh vào danh sách hình ảnh

Bạn có thể thấy rằng những tấm hình trên danh sách hình ảnh được đánh số từ 1 đến 3, và thuộc tính (properties) của chúng có thể được thấy bên cạnh.

Bây giờ, để dùng những hình ảnh trong danh sách hình ảnh, từ thanh công cụ, kéo (drag) đối tượng ListView vào form. Chỉ trong chốc lát, chúng ta sẽ dễ dàng có được ListView để trình bày những hình ảnh nhỏ này. Chọn đối tượng listView1 và xem các thuộc tính của nó. Thuộc tính SmallImageList bây giờ có thể được dùng với imageList1 vừa được tạo ra.

Hãy nhìn đoạn mã chương trình của InitializeComponent (trong ví dụ 3.4-5) để xem những việc trên đã tác động như thế nào đến các đoạn mã chương trình.

Ví dụ 3.4-5 Form1.cs: InitializeComponent

```
1: private void InitializeComponent()
2: {
3:     this.components = new
System.ComponentModel.Container();
```

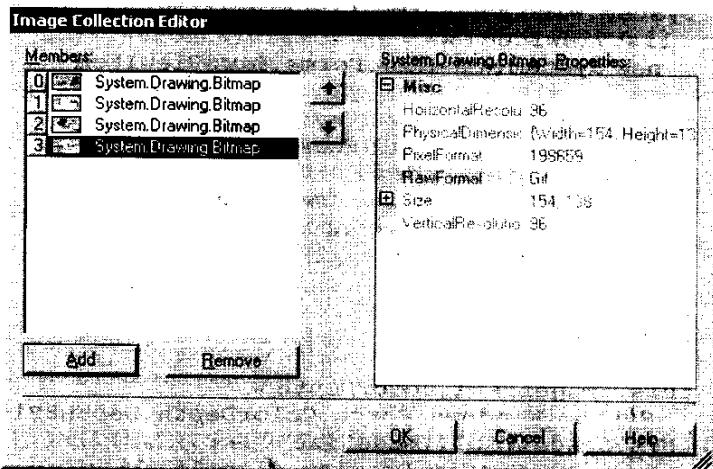
```
4:     System.Resources.ResourceManager resources =
5:         new System.Resources.ResourceManager(
6:             typeof(Form1));
7: this.listView1 = new
8:     System.Windows.Forms.ListView();
9: this.imageList1 = new
10:    System.Windows.Forms.ImageList(this.component);
11: this.SuspendLayout();
12: this.listView1.Location = new
13:     System.Drawing.Point (8, 8);
14: this.listView1.Name = "listView1";
15: this.listView1.Size = new
16:     System.Drawing.Size(272, 200);
17: this.listView1.SmallImageList = this.imageList1;
18: this.listView1.TabIndex = 0;
19: //  
// imageList1  
//  
  
20: this.imageList1.ColorDepth =
21:     System.Windows.Forms.ColorDepth.Depth8Bit;
22: this.imageList1.ImageSize = new
23:     System.Drawing.Size(16, 16);
24: this.imageList1.ImageStream
25: ((System.Windows.Forms.ImageListStreamer)
26: (resource.GetObject("imageList1.ImageStream")));
27: this.imageList1.TransparentColor =
28:     System.Drawing.Color.Magenta;
29: //  
// Form 1  
//  
30: this.AutoScaleBaseSize = new
31:     System.Drawing.Size(5,13);
32: this.ClientSize = new
33:     System.Drawing.Size(292, 273);
34: this.Controls.AddRange(new
35:     System.Windows.Forms.Control[] {
36:         this.listView1});
37: this.Name = "Form1";
38: this.Text = "Form1";
39: this.ResumeLayout(false);
```



Dòng 4 và 5 tạo ra ResourceManager cho form. Dòng 12-16 tạo ra ListView và thiết lập thông số SmallImageList để dùng thành phần imageView1. Dòng 20-24 khởi tạo một danh sách hình ảnh bằng cách lấy từ bộ định dòng (streamer) danh sách hình ảnh, thiết lập màu trong suốt và kích cỡ cho hình ảnh.

Để dùng những tài nguyên trong ListView, chọn bảng thuộc tính của đối tượng ListView, chọn tập thuộc tính Items, click vào dấu ba chấm (...). Một hộp thoại sẽ hiện lên như hình 3.4-7 cho phép bạn thêm và chỉnh sửa các mục trong danh sách.

Lưu ý: chương trình của bạn cũng có thể thêm các mục vào thành phần một cách nhanh chóng. Visual Studio cung cấp một cách để làm việc đó.



Hình 3.4-7 Sử dụng tài nguyên hình ảnh trong đối tượng ListViewItem

Mỗi mục được thêm vào trong cây sẽ có một số ký tự và hình ảnh để hiển thị. ListView cũng cung cấp các sự kiện (event) tương tự như minh họa trong chương 3.2 “Thành phần Giao diện Người dùng” khi bạn chọn, chỉnh sửa, ... xử lý những sự kiện này cho phép bạn tác động qua lại với các mục trong danh sách.

Nào, chúng ta hãy xem qua ví dụ 3.4-6 dưới đây:

Ví dụ 3.4-6 Form1.cs: InitializeComponent

```

1: private void InitializeComponent()
2: {
3:     this.components = new
        System.ComponentModel.Container();
4:     System.Resource.ResourceManager resources =
        new System.Resource.ResourceManager(
            typeof(Form1));
5:

```

```
6:     System.Windows.Forms.ListViewItem
7:         listViewItem1 = new
8:             System.Windows.Forms.ListViewItem (
9:                 "Folder", 0);
10:    System.Windows.Forms.ListViewItem
11:        listViewItem2 =
12:            new System.Windows.Forms.ListViewItem (
13:                "Open Folder", 1);
14:    System.Windows.Forms.ListViewItem
15:        listViewItem3 =
16:            new System.Windows.Forms.ListViewItem
17:                ("Book", 2);
18:    System.Windows.Forms.ListViewItem
19:        listViewItem4 =
20:            new System.Windows.Forms.ListViewItem
21:                ("Wrench", 3);
22: this.listView1 = new
23:     System.Windows.Forms.ListView();
24: this.imageList1 = new
25:     System.Windows.Forms.ImageList(
26:         this.component);
27: this.SuspendLayout();
28: // imageList1
29: //
30: this.imageList1.ColorDepth =
31:     System.Windows.Forms.ColorDepth.Depth8Bit;
32: this.imageList1.ImageSize = new
33:     System.Drawing.Size(16, 16);
34: this.imageList1.ImageStream
35:     ((System.Windows.Forms.ImageListStreamer)
```



```

36:     listViewItem4);
37:     this.listView1.Location = new
            System.Drawing.Point (8, 8);
38:     this.listView1.Name = "listView1";
39:     this.listView1.Size = new
            System.Drawing.Size(272, 200);
40:     this.listView1.SmallImageList = this.imageList1;
41:     this.listView1.TabIndex = 0;
42:     this.listView1.View =
            System.Windows.Forms.View.SmallIcon;
43:     //
44:     // Form 1
45:     //
46:     this.AutoScaleBaseSize = new
            System.Drawing.Size(5,13);
47:     this.ClientSize = new
            System.Drawing.Size(292, 273);
48:     this.Controls.AddRange(new
            System.Windows.Forms.Control[] {
49:         this.listView1});
50:     this.Name = "Form1";
51:     this.Text = "Form1";
52:     this.ResumeLayout(false);
53: }

```

Các dòng 6-13 tạo ra đối tượng ListViewItem, thiết lập nhãn và hình ảnh trong ImageList cho các mục. Dòng 32-36 thêm các mục vào ListView.

8. PHƯƠNG CÁCH LẬP TRÌNH TRUY XUẤT ĐẾN CÁC TÀI NGUYÊN

Ngoài việc tạo tài nguyên thông qua Visual Studio hoặc các công cụ phát triển khác, bạn còn có thể tạo, quản lý, và sử dụng tài nguyên thông qua mã chương trình. Một ví dụ về tài nguyên kiểu này là lưu một vài dữ liệu cá nhân cho ứng dụng của bạn (ví dụ, kích cỡ cửa sổ và vị trí) để tải lên lần nữa khi chương trình chạy lại.

Đọc và ghi tài nguyên được thực hiện nhờ vào các lớp ResourceReader và ResourceWriter. Các đối tượng này cho phép ta làm việc với tài nguyên lưu trong các luồng (stream) hay trong các tập tin.

Trong ví dụ dưới đây, chúng ta sẽ chuẩn bị một ứng dụng Windows Forms đơn giản để hiển thị hai quả bóng: một đỏ và một vàng trên bề mặt của form. Bạn có thể nhặt một quả bóng bằng chuột và di chuyển nó. Khi chương trình đóng lại, nó ghi các thông tin hiện hành trên form vào file tài nguyên gọi là ball_locations.resources. Tài nguyên này lưu lại vị trí trên màn hình của hai quả bóng, nhờ thế lần sau khi được tải lên, ứng dụng vẽ nó lại ở nơi mà trước đây bạn đã thấy.

Thêm vào đó, ứng dụng này cũng minh họa một vài cách xử lý chuột đơn giản cũng như việc sử dụng ImageList để vẽ hình ảnh trên bề mặt form.

Ví dụ 3.4.7 Resourcerw.cs: Ứng dụng đọc/ghi tài nguyên

```
1: using System;
2: using System.Drawing;
3: using System.Collections;
4: using System.ComponentModel;
5: using System.Windows.Forms;
6: using System.Data;
7: using System.Resources;
8: using System.Globalization;
9:
10:
11:
12: namespace resourcerw
13: {
14:     /// <summary>
15:     /// Summary description for DataForm.
16:     /// </summary>
17:     public class Form1 : System.Windows.Forms.Form
18:     {
19:         private System.Windows.Forms.ImageList
20:                         imageList1;
21:         private System.ComponentModel.IContainer
22:                         components;
23:         private System.Drawing.Point[]
24:                         BallAllocations;
25:
26:         public Form1()
27:         {
28:             //
29:             // Required for Windows Form Designer support
30:             //
31:             InitializeComponent();
32:
33:             //
34:             // TODO: Add any constructor code after
35:             //InitializeComponent call
36:
37:             //
38:             this.BallLocations = new
39:                         System.Drawing.Point[2];
```

```
39:         }
40:
41:     /// <summary>
42:     /// Clean up any resources being used.
43:     /// </summary>
44:     protected override void Dispose(bool disposing)
45:     {
46:         if (disposing)
47:         {
48:             if (components != null)
49:             {
50:                 components.Dispose();
51:             }
52:         }
53:         base.Dispose(disposing);
54:     }
55:
56:     #region Web Form Designer generated code
57:     /// <summary>
58:     /// Required method for Designer support do not
59:     /// modify
60:     /// the contents of this method with the code
61:     /// editor.
62:     /// </summary>
63:     private void InitializeComponent()
64:     {
65:         this.components = new
66:             System.ComponentModel.Container();
67:         System.Resource.ResourceManager resources =
68:             new System.Resource.ResourceManager(
69:                 typeof(Form1));
70:         this.imageList1 =
71:             new System.Windows.Forms.ImageList(
72:                 this.component);
73:         //
74:         // imageList1
75:         //
76:         this.imageList1.ColorDepth =
77:             System.Windows.Forms.ColorDepth.Depth8Bit;
78:         this.imageList1.ImageSize = new
    System.Drawing.Size(64, 64);
79:         this.imageList1.ImageStream =
80:             (System.Windows.Forms.ImageListStreamer)
81:                 (resource.GetObject(
82:                     "imageList1.ImageStream")));
83:         this.imageList1.TransparentColor =
84:             System.Drawing.Color.Magenta;
85:         //
86:     }
```



```
79:         // Form 1
80:         //
81:         this.AutoScaleBaseSize = new
82:             System.Drawing.Size(5,13);
83:         this.BackColor =
84:             System.Drawing.Color.Green;
85:         this.ClientSize = new
86:             System.Drawing.Size(376, 301);
87:         this.Name = "Form1";
88:         this.Text = "Resource read-write";
89:         this.MouseDown += new
90:             System.Windows.Forms.MouseEventHandler(
91:                 this.Form1_MouseDown);
92:         this.Closing += new
93:             System.ComponentModel.CancelEventHandler(
94:                 this.Form1_Closing);
95:         this.Load +=
96:             new System.EventHandler(this.Form1_Load);
97:         this.MouseUp += new
98:             System.Windows.Forms.MouseEventHandler(
99:                 this.Form1_MouseUp);
100:        this.Paint += new
101:            System.Windows.Forms.PaintEventHandler(
102:                this.Form1_Paint);
103:        thisMouseMove += new
104:            System.Windows.Forms.MouseEventHandler(
105:                this.Form1_MouseMove);
106:        }
107:    #endregion
108:    /// <summary>
109:    /// The main entry point for the application.
110:    /// </summary>
111:    [STAThread]
112:    static void Main()
113:    {
114:        Application.Run(new Form1());
115:    }
116:
117:    private void Form1_MouseMove(object sender,
118:        System.Windows.Forms.MouseEventArgs e)
119:    {
120:        if (_mousedown && (_grabbed != -1))
121:        {
122:            if (e.X>31 && e.Y>31 &&
123:                e.X<(this.Size.Width-32)) &&
```



```
118:             e.Y<(this.Size.Height-32)
119:         {
120:             BallLocations[_grabbed].X=e.X;
121:             BallLocations[_grabbed].Y=e.Y;
122:
123:             this.Invalidate();
124:         }
125:     }
126: }
127:
128: private void Form1_MouseUp(object sender,
129:     System.Windows.Forms.MouseEventArgs e)
130: {
131:     _mousedown = false;
132: }
133:
134: private void Form1_MouseDown(object sender,
135:     System.Windows.Forms.MouseEventArgs e)
136: {
137:     _mousedown = true;
138:     int index = 0;
139:     _grabbed = -1
140:     foreach (Point p in this.BallLocations)
141:     {
142:         if(Math.Abs(e.X-p.X)<32 && Math.Abs(
143:             e.Y-p.Y)<32 )
144:             _grabbed = index;
145:             index++;
146:     }
147:
148: private void Form1_Paint(object sender,
149:     System.Windows.Forms.PaintEventArgs e)
150: {
151:     int index = 0;
152:     foreach (Point p in this.BallLocations)
153:     {
154:         this.imageList1.Draw(e.Graphics,
155:             p.X-32, p.Y-32, 64, 64, index++);
156:     }
157:
158: private void Form1_Load(object sender,
159:     System.EventArgs e)
160:
161:     ResourceSet rs;
```



```
163:     try
164:     {
165:         rs = new
166:             ResourceSet("ball_locations.resources");
167:             BallLocations[0] = (Point)
168:                 rs.GetObject("RedBall", false);
169:                 BallLocations[1] = (Point)
170:                     rs.GetObject("YellowBall", false);
171:                     rs.Close();
172:     }
173:     catch (System.Exception)
174:     {
175:     }
176:     }
177:     }
178:
179:     private void Form1_Closing(object sender,
180:         System.ComponentModel.CancelEventArgs e)
181:     {
182:         // Lưu vị trí quả bóng vào file tài nguyên
183:         // ta chỉ cần ghi lại toàn bộ các đối tượng là đủ
184:         ResourceWriter rw = new ResourceWriter(
185:             "ball_locations.resources");
186:             rw.AddResource("RedBall",
187:                 this.BallLocations[0]);
188:                 rw.AddResource("YellowBall",
189:                     this.BallLocations[1]);
190:             rw.Generate();
191:             rw.Close();
192:     }
193: }
```

Dòng 61-99 cho thấy phương thức InitializeComponent cần cho Visual Studio. Dòng 63-65 mở một tài nguyên hình ảnh bằng đối tượng ImageList, và dòng 66-77 khởi tạo ImageList bằng cách tải vào luồng dữ liệu ảnh (image stream) và thiết lập các thông số về kích cỡ và màu trong suốt. Dòng 86-97 thêm các xử lý chuột và vẽ ảnh.

Đối với các bộ xử lý sự kiện (event handler), hai sự kiện quan trọng nhất là nạp và đóng form. Dòng 158-175 là bộ xử lý sự kiện tải form. Nó sẽ mở tài nguyên và lấy ra vị trí của các quả bóng. Nếu nó thất bại vì bất cứ lý do gì, một bộ xử lý ngoại lệ (exception handler) sẽ đơn giản cho rằng tập tin không tồn tại và khởi



tạo hai quả bóng ở vị trí mặc nhiên của nó. Dòng 179-188 được gọi khi form đã sẵn sàng để đóng lại. Ở đây, mã chương trình sẽ tạo ra một tập tin tài nguyên và lưu tất cả vị trí của các quả bóng vào đó. Lần kế tiếp chương trình chạy, tập tin đã tồn tại, và bộ xử lý tải form sẽ tìm được vị trí đã được lưu.

Bộ xử lý MouseUp (dòng 128-132) đơn giản khởi tạo lại giá trị của cờ để biết rằng không có việc kéo bóng đến vị trí khác. Bộ xử lý MouseDown (dòng 134-146) quyết định xem quả bóng nào trong 2 quả, nếu có, đang ở vị trí của chuột, và đánh dấu cờ chọn nó.

Bộ xử lýMouseMove kiểm tra xem có quả bóng nào được chọn và kiểm tra xem chuột đang ở trong vùng màn hình xử lý không để ngăn không cho người dùng di chuyển bóng ra ngoài. Nếu tất cả đều đúng, nó sẽ thay đổi vị trí hiện thời của quả bóng. Phương thức này có thể tìm thấy từ dòng 112 đến 126.

Cuối cùng, ở dòng 148-156, bộ xử lý Paint sẽ vẽ quả bóng trên bề mặt của form.

9. ĐỌC VÀ VIẾT TẬP TIN XML RESX

Các tập tin tài nguyên, như chúng tôi đã nhắc ngay từ đầu chương, có nhiều loại: tập tin .resources và tập tin .resx. Tập tin resx là tập tin XML thô và chưa được dịch, được dùng như dạng đơn giản của tài nguyên. Ở ví dụ trước, chúng ta đã nói về các lớp ResourceReader và ResourceWriter. Chúng ta cũng có các lớp RexsResourceReader và RexsResourceWriter được dùng để xử lý dạng tài nguyên XML.

Trong ví dụ cuối của chương này, chúng ta sẽ tạo ra một ứng dụng đơn giản cho phép bạn tải các tập tin hình ảnh, văn bản, biểu tượng và chuyển đổi chúng sang thành tập tin tài nguyên kiểu .REXS (Xem ví dụ 3.4-8). Sự phức tạp của nhiệm vụ này và sự đơn giản của chương trình sẽ minh họa một số sức mạnh tiềm tàng của kiến trúc Windows Form.

Ví dụ 3.4-8 ResxUtilForm.cs: Tiện ích Ghi RESX

```

1: using System;
2: using System.Drawing;
3: using System.Collections;
4: using System.ComponentModel;
5: using System.Windows.Forms;
6: using System.Data;
7: using System.IO;
8:
9: /*
10:  * Lưu ý: một lỗi trên các phiên bản trước của .NET có thể làm
11:  * chương trình gặp một lỗi ngoại lệ khi chỉnh sửa
12:  * các tài nguyên. Trong tập tin này có 2 đoạn được lược chú
13:  * với mục đích cho hỗ trợ giải quyết lỗi đó.

```



```
14: * Nếu đoạn mã này không chạy được, hãy bỏ các dấu chú thích của
15: * những đoạn được đánh dấu BUGFIXER và dịch lại
16: *
17: */
18:
19: namespace ResxUtil
20: {
21:     /// <summary>
22:     /// Summary description for Form1.
23:     /// </summary>
24:     public class ResxUtilForm :
25:         System.Windows.Forms.Form
26:     {
27:         private System.Windows.Forms.ListView
28:             listView1;
29:         private System.Windows.Forms.Button button1;
30:         private System.Windows.Forms.Button button2;
31:         private System.Windows.Forms.ImageList
32:             imageList1;
33:         private System.Windows.Forms.ColumnHeader
34:             columnHeader2;
35:         private System.Windows.Forms.ColumnHeader
36:             columnHeader1;
37:         private System.ComponentModel.IContainer
38:             components;
39:         public ResxUtilForm()
40:         {
41:             //
42:             // Required for Windows Form Designer support
43:             //
44:             InitializeComponent();
45:             //
46:             // BUGFIXER uncomment the line below
47:             //this.listView1.LabelEdit = false;
48:         }
49:         /// <summary>
50:         /// Clean up any resources being used.
51:         /// </summary>
52:         protected override void Dispose(bool disposing)
53:         {
54:             if (disposing)
55:             {
```

```
56:         }
57:         base.Dispose(disposing);
58:     }
59:
60:     #region Web Form Designer generated code
61:     /// <summary>
62:     /// Required method for Designer support do not
63:     /// modify
64:     /// the contents of this method with the code
65:     /// editor.
66:     /// </summary>
67:     private void InitializeComponent()
68:     {
69:         this.components = new
70:             System.ComponentModel.Container();
71:         System.Resource.ResourceManager resources =
72:             new System.Resource.ResourceManager(
73:                 typeof(ResxUtilForm));
74:         this.columnHeader2 = new
75:             System.Windows.Forms.ColumnHeader();
76:         this.columnHeader1 = new
77:             System.Windows.Forms.ColumnHeader();
78:         this.imageList1 = new
79:             System.Windows.Forms.ImageList(
80:                 this.component);
81:         this.listView1 = new
82:             System.Windows.Forms.ListView();
83:         this.button1 = new
84:             System.Windows.Forms.Button();
85:         this.button2 = new
86:             System.Windows.Forms.Button();
87:         this.SuspendLayout();
88:
89:         // columnHeader2
90:         //
91:         this.columnHeader2.Text = "File Name";
92:         this.columnHeader2.Width = 545;
93:
94:         // columnHeader1
95:         //
96:         this.columnHeader1.Text = "Resource Name";
97:         this.columnHeader1.Width = 220;
98:
99:         // imageList1
100:
101:        this.imageList1.ColorDepth =
102:            System.Windows.Forms.ColorDepth.Depth8Bit;
103:        this.imageList1.ImageSize = new
```

```
        System.Drawing.Size(16, 16);
91:    this.imageList1.ImageStream =
         ((System.Windows.Forms.ImageListStreamer)
             (resource.GetObject(
                 "imageList1.ImageStream"))));
92:    this.imageList1.TransparentColor =
         System.Drawing.Color.Magenta;
93:    //
94:    // listView1
95:    //
96:    this.listView1.Anchor =
         ((System.Windows.Forms.AnchorStyles.Top |
             System.Windows.Forms.AnchorStyles.Left) |
             System.Windows.Forms.AnchorStyles.Right);
97:    this.listView1.Columns.AddRange (new
98:        System.Windows.Forms.ColumnHeader[] {
99:            this.columnHeader1,
100:            this.columnHeader1});
101:           this.listView1.FullRowSelect = true;
102:           this.listView1.Location = new
         System.Drawing.Point (8, 8);
103:          this.listView1.Name = "listView1";
104:          this.listView1.Size = new
         System.Drawing.Size(792, 304);
105:          this.listView1.SmallImageList =
         this.imageList1;
106:          this.listView1.TabIndex = 0;
107:          this.listView1.View =
         System.Windows.Forms.View.Details;
108:          this.listView1.KeyDown += new
         System.Windows.Forms.KeyEventHandler
             (this.listView1_KeyDown);
109:          this.listView1.LabelEdit = true;
110:          //
111:          // button1
112:          //
113:          this.button1.Location = new
         System.Drawing.Point (696, 328);
114:          this.button1.Name = "button1";
115:          this.button1.Size = new
         System.Drawing.Size(104, 32);
116:          this.button1.TabIndex = 1;
117:          this.button1.Text = "Save";
118:          this.button1.Click += new
         System.EventHandler (
             this.button1_Click);
119:          //
120:          // button2
```

```
121:      //  
122:      this.button2.Location = new  
123:          System.Drawing.Point (8, 328);  
124:      this.button2.Name = "button2";  
125:      this.button2.Size = new  
126:          System.Drawing.Size(88,32);  
127:      this.button2.TabIndex = 1;  
128:      this.button2.Text = "Add...";  
129:      this.button2.Click += new  
130:          System.EventHandler (  
131:              this.button2_Click);  
132:      //  
133:      // ResxUtilForm  
134:      //  
135:      this.AllowDrop = true;  
136:      this.AutoScaleBaseSize = new  
137:          System.Drawing.Size(5,13);  
138:      this.ClientSize = new  
139:          System.Drawing.Size(816, 373);  
140:      this.Controls.AddRange(new  
141:          System.Windows.Forms.Control[] {  
142:              this.button1,  
143:              this.button2,  
144:              this.listView1});  
145:      this.FormBorderStyle =  
146:          System.Windows.Forms.  
147:              FormBorderStyle.FixedSingle;  
148:      this.Name = "ResxUtilForm";  
149:      this.Text = "Resx Utility";  
150:      this.ResumeLayout(false);  
151:  }  
152: #endregion  
153: /// <summary>  
154: /// The main entry point for the application.  
155: /// </summary>  
156: [STAThread]  
157: static void Main()  
158: {  
159:     Application.Run(new ResxUtilForm());  
160: }  
161: private void button2_Click (object sender,  
162:                         System.EventArgs e)  
163: {  
164:     int icon = -1;  
165:     OpenFileDialog dlg = new OpenFileDialog();
```



```
159:         if(dlg.ShowDialog() == DialogResult.OK)
160:         {
161:             string s = dlg.FileName;
162:             string[] d = s.Split(new char[] { '.' });
163:             string resname = "Unidentified";
164:
165:             if (d.Length < 2)
166:                 return;
167:
168:             switch (d[d.Length - 1].ToLower())
169:             {
170:                 case "bmp":
171:                     icon=0;
172:                     break;
173:                 case "txt":
174:                     icon=1;
175:                     break;
176:                 case "ico":
177:                     icon=2;
178:                     break;
179:             }
180:
181:
182:             // BUGFIXER
183:             // code below.
184:             /* Form f = new Form();
185:             f.Size = new Size(350, 110);
186:             f.Text = "Resource name";
187:             f.FormBorderStyle =
188:                 FormBorderStyle.FixedDialog;
189:             Button ok = new Button();
190:             ok.Dialog.Result = DialogResult.OK;
191:             ok.Location = new Point(5, 32);
192:             ok.Size = new Size(75, 22);
193:             ok.Text = "OK";
194:             f.Controls.Add(ok);
195:             Button can = new Button();
196:             can.Dialog.Result = DialogResult.Cancel;
197:             can.Location = new Point(260, 32);
198:             can.Size = new Size(75, 22);
199:             can.Text = "Cancel";
200:             f.Controls.Add(can);
201:             TextBox tb = new TextBox();
202:             tb.Location = new Point(5, 5);
203:             tb.Size = new Size(330, 25);
204:             tb.Text = "Resource" +
((object)(this.listView1.Items.Count
+1)).ToString();
```

```
205:         f.Controls.Add(tb);
206:         if(f.ShowDialog() == DialogResult.Cancel
207:             || tb.Text.Trim() == "")
208:             return;
209:         restname = tb.Text.Trim();
210:         */
211:
212:         if (icon>0)
213:         {
214:             ListViewItem item = new
215:                 ListViewItem(resname, ico);
216:             item.SubItems.Add(s);
217:             this.listView1.Items.Add(item);
218:         }
219:     }
220:
221:     private void button1_Click (object sender,
222:                             System.EventArgs e)
223:     {
224:         if (this.listView1.Items.Count > 0)
225:         {
226:             // Kiểm tra tên của tất cả resource
227:             foreach(ListViewItem item in
228:                     this.listView1.Items)
229:             {
230:                 if (item.Text == "Undefined")
231:                 {
232:                     MessageBox.Show("All resources
233:                                     must have names");
234:                     return;
235:                 }
236:             }
237:             //Lấy tên file resx
238:             SaveFileDialog dlg = new SaveFileDialog();
239:             dlg.Title = "Save .resx file";
240:             dlg.Filter = "XML resource files|.resx";
241:             if (dlg.ShowDialog() ==
242:                 DialogResult.Cancel)
243:                 return;
244:             //Tạo lớp ghi tài nguyên
245:             System.Resources.ResXResourceWriter rw =
new System.Resources.
ResXResourceWriter (dlg.FileName);
```

```
246:         //Duyệt qua các mục trong danh sách
247:         foreach (ListViewItem item
248:             in this.listView1.Items)
249:         {
250:             switch (item.ImageIndex)
251:             {
252:                 case 0: //bitmap images
253:                     goto case 2;
254:                 case 1: //text files
255:                     System.IO.StreamReader sr =
256:                         new System.IO.StreamReader(
257:                             item.SubItems[1].Text);
258:                     string inputline;
259:                     do
260:                     {
261:                         inputline = sr.ReadLine();
262:                         if (inputline != null)
263:                         {
264:                             string[] splitline =
265:                                 inputline.Split(new char[] {'='}
266:                                         , 2);
267:                             if (splitline.Length == 2)
268:                             {
269:                                 rw.AddResource(
270:                                     tem.Text.TrimEnd().TrimStart()
271:                                     +"." + splitline[0].
272:                                         TrimEnd().TrimStart(),
273:                                         splitline[1]);
274:                             }
275:                         while (inputline != null);
276:                         break;
277:                         case 2: //icons
278:                             System.Drawing.Image im =
279:                                 System.Drawing.Image.FromFile(
280:                                     item.SubItems[1].Text);
281:                             rw.AddResource(item.Text, im);
282:                             break;
283:                         }
284:                     }
285:                 rw.Generate();
286:                 rw.Close();
287:             }
288:         }
```

```

290:
291:
292:     private void listView1_KeyDown(object sender,
293:         System.Windows.Forms.KeyEventArgs e)
294:     {
295:         if (this.listView1.SelectedIndices.Count
296:             == 0)
297:             return;
298:         if (e.KeyCode.Equals(Keys.Delete))
299:             this.listView1.Items.RemoveAt(
300:                 this.listView1.SelectedIndices[0]);
301:     }
302:
303: }

```

Ghi chú: *Chú ý rằng kiểu của bố trí (layout) ở đây và ở các danh sách khác đã được thay đổi để phù hợp với định dạng 80 cột.*

Bây giờ, chúng ta nên đi thẳng vào phương thức InitializeComponent và tìm hiểu xem chuyện gì đang diễn ra, chúng ta sẽ nắm được những điểm chính yếu của chương trình.

Dòng 79-80 tạo ra một tập các tiêu đề của cột (column header) để dùng cho ListView khi ở chế độ xem chi tiết (gọi là Detail View). Chúng ta đã chọn để hiển thị biểu tượng của một mục trên ImageList cùng với tên của tài nguyên và tên của tập tin chứa tài nguyên.

Dòng 89-92 thiết lập ImageList, đem đến luồng dữ liệu hình ảnh (image stream), thiết lập kích cỡ, màu trong suốt cho ảnh.

Dòng 98-100 thêm tiêu đề cột (column header) vào điều khiển ListView. Dòng 93-100 thiết lập các thuộc tính của ListView cho phép sử dụng lựa chọn các mục (nhờ vậy bạn có thể xoá một mục trong danh sách), thay đổi nhãn, và cuối cùng là bộ điều khiển sự kiện KeyDown (chỉ trả lời cho phím Delete).

Dòng 110-128 là các nút nhấn (button). Một nút được dùng để mở lại tài nguyên, còn nút khác lưu tài nguyên vào tập tin RESX trở lại.

Dòng 155-219 xử lý nút nhấn Add. Nó cho phép bạn duyệt các tập tin (dòng 154 - 155) bitmap, văn bản và biểu tượng. Sau khi bạn tìm được tài nguyên cần thiết, dòng 168-179 quyết định chọn đúng biểu tượng cho tập tin đó. Cuối cùng, dòng 212-217 sử dụng biểu tượng mà bạn chọn để tạo ra ListViewItem có một nhãn (thoạt đầu là “Undefined”) và các mục phụ (sub-item), các mục này sẽ dùng cột thứ hai của điều khiển danh sách (list control) để chứa tên tập tin.

Nút Save trên dòng 221-184 sẽ kiểm tra nhanh xem tất cả các tài nguyên đều đã được đặt tên hay chưa (226-233) và lấy ra tên tập tin của tập tin đích (dòng 236-240). Dòng 243-244 sẽ tạo ra một đối tượng ResxResourceWriter và gửi nó đến vòng lặp foreach để duyệt tất cả các mục và tạo ra tài nguyên.



Câu lệnh switch (dòng 249) có 3 trường hợp (case). Trường hợp 0 và 2 được xử lý tương tự bởi vì, theo cấu trúc định dạng của XML, không có sự khác nhau giữa bitmap và icon. Trường hợp 1 (dòng 254-276) đọc tập tin văn bản một dòng mỗi lần và chia dòng ấy ra thành các cặp tên – giá trị và lưu vào như tài nguyên văn bản. Tên được đặt với tiền tố (prefix) là tên bạn gắn cho tập tin văn bản trên hộp danh sách (listbox). Lưu ý rằng chương trình minh họa này chỉ hiểu các tập tin văn bản có cấu trúc “<tên chuỗi>=<giá trị chuỗi>”. Các tập tin khác sẽ sinh ra lỗi khi xử lý. Sau khi chạy xong, dòng 286-287 sẽ ghi và đóng tập tin RESX.

Cuối cùng, dòng 292-300 xử lý khi phím Delete được nhấn và xoá một mục (item) ra khỏi danh sách.

Lưu ý về lỗi: Dòng 42 và khối mã từ dòng 184-208 xử lý những vấn đề tiềm tàng có thể gặp trên các phiên bản trước của kiến trúc .NET. Nếu bạn chỉnh sửa tên của tài nguyên chương trình có thể sụp đổ hãy bỏ dấu ghi chú (comment) của hai đoạn mã chương trình với những dòng được đánh dấu “BUGFIXER” và dịch lại chương trình. Nó sẽ hoạt động, và bạn sẽ thấy hơi khác một chút, nhưng nó sẽ không bị lỗi.

Cũng hãy lưu ý cách đoạn mã bugfix tạo ra một hộp thoại khi chương trình đang chạy (on-the-fly) cho bạn.

10. KẾT CHƯƠNG

Trong chương này, bạn đã học cách làm thế nào để tạo một tài nguyên văn bản bằng tay, làm thế nào để quốc tế hoá một ứng dụng dùng gói kết hợp, biết làm thế nào để xây dựng một ứng dụng có hình ảnh và các tài nguyên khác, năng động sáng tạo và dùng các tập tin tài nguyên trong chương trình của bạn. Trong chương tới, chúng ta sẽ cùng tìm hiểu về các chức năng đồ họa mạnh mẽ của GDI+.

Chương 3.5

GDI+: GIAO DIỆN ĐỒ HỌA CỦA .NET

Các vấn đề chính sẽ được đề cập đến

- ✓ Các nguyên lý cơ bản của GDI+
- ✓ Đối tượng Graphics
- ✓ Các hệ tọa độ đồ họa
- ✓ Vẽ đường thẳng và các hình đơn giản
- ✓ Sử dụng bút vẽ và cọ vẽ chuyển màu
- ✓ Bút vẽ và cọ vẽ có chất liệu
- ✓ Làm tròn các đầu mứt của đường thẳng
- ✓ Đường cong và các đường đồ thị
- ✓ Đối tượng GraphicsPath
- ✓ Kỹ thuật cắt bằng Path và vùng
- ✓ Các phép biến đổi
- ✓ Sự phối màu với giá trị alpha
- ✓ Sự phối màu với giá trị alpha cho ảnh
- ✓ Những thao tác khác trong không gian màu

Trong chương này, chúng ta sẽ xem xét giao diện thiết bị đồ họa (GDI – Graphic Device Independence). Windows Forms sử dụng nó để tạo đồ họa. Chú ý tên là GDI+ không là GDI.NET. Đó là vì GDI+ không là tập API duy nhất của kiến trúc Framework .NET. Nó cũng có thể được sử dụng bởi mã không được quản lý. Tuy nhiên với mục đích của chương này, chúng ta sẽ sử dụng nó riêng biệt với Windows Forms.

1. CÁC NGUYÊN LÝ CƠ BẢN CỦA GDI+

Giống như GDI, GDI+ là chế độ đồ họa kiểu tức thì. Điều này có nghĩa là khi chúng ta gửi những mệnh lệnh đến giao diện đồ họa, những tác động đó ngay lập tức được thấy trên bề mặt thiết bị hay trong bộ nhớ hiển thị. Một kiểu chế độ đồ họa khác là duy trì, kiểu này thường duy trì tính kế thừa của các đối tượng để chúng biết cách tự vẽ.

GDI+ dùng cọ vẽ (brush) và bút vẽ (pen) để vẽ lên bề mặt của đồ họa, như chính GDI đã làm. Mặc dù không như GDI, bạn có thể sử dụng bất cứ đối tượng đồ



họa nào trên bất kỳ bề mặt nào vào bất cứ lúc nào. GDI yêu cầu bạn tạo một đối tượng đồ họa, chọn nó để sử dụng trong ngữ cảnh thiết bị (Device Context hay DC), dùng đối tượng này, bỏ chọn đối tượng này ra khỏi DC, và sau đó hủy đối tượng này. Nếu bạn không làm theo thứ tự đó, bạn có thể bị những lỗi hổng trong tài nguyên của hệ thống mà cuối cùng là sẽ làm cho tất cả bộ nhớ được cấp phát trong hệ thống của bạn mất đi và hệ thống cần phải khởi động lại. Nói chung sử dụng GDI+ thì tiện lợi hơn vì bạn không phải lo lắng về việc chọn hay hủy bỏ việc chọn tài nguyên theo thứ tự cần phải có.

Như bạn mong đợi, sự phát triển của GDI+ phản ánh nhiều sự thay đổi gần đây theo tiềm năng của card đồ họa. Một số tính năng mới được thêm vào bao gồm phối màu với giá trị alpha (Alpha Blending) hay sự trong suốt đối với ảnh (image transparency), làm tròn (antialiasing), những phép biến đổi (transformation) như phép quay (rotation) và phép co dãn (scaling), thêm vào nhiều phương pháp mới trong việc hiệu chỉnh màu và thao tác vùng.

2. ĐỐI TƯỢNG GRAPHICS

Với .NET, đối tượng Graphics là nơi phát sinh của tất cả các khả năng đồ họa của GDI+. Điều này tương tự với Win32 DC và chúng ta có thể lấy đối tượng này một cách tường minh từ bộ quản handle của cửa sổ hoặc nó được truyền đến ứng dụng từ sự kiện OnPaint. Chẳng hạn để có được đối tượng Graphics trong C++, chúng ta sẽ phải viết đoạn mã sau:

```
HDC dc;  
PAINTSTRUCT ps;  
Graphics *pGraphics;  
  
Dc=BeginPaint(hWnd,&ps);  
PGraphics=new Graphics(dc);  
  
// sử dụng Graphics...  
EndPaint(hWnd,&ps);
```

Với những mục đích của chương này, chúng ta sẽ giả sử rằng những thao tác của GDI+ sẽ xảy ra trong phạm vi của ứng dụng Windows Forms và chúng ta có thể lấy đối tượng Graphics trong quá trình chuyển giao (delegate) sự kiện. Ví dụ, chúng ta có được đối tượng Graphics trong sự kiện PaintEventArgs như sau:

```
public void OnPaint(object sender,PaintEventArgs e)  
{  
    // sử dụng e.Graphics.SomeMethod(..) ở đây  
}
```

3. CÁC HỆ TỌA ĐỘ ĐỒ HỌA

Có không ít hơn ba hệ thống tọa độ được sử dụng để vẽ các đối tượng lên trên màn hình. Đó là hệ thống tọa độ thế giới thực (World Coordinate system), đây là khái niệm lý thuyết về không gian trong hệ thống đồ họa, hệ thống tọa độ trên giấy (Page Coordinate system) là khái niệm lý thuyết khác về chiều vật lý

mang tính chất giả thuyết, và hệ thống tọa độ thiết bị (Device Coordinate system) gắn liền với thuộc tính vật lý của màn hình hay máy in mà trên đó đồ họa được tạo ra. Để hiển thị bất cứ điều gì trên màn hình, dữ liệu đồ họa mà bạn ghi vào đối tượng đồ họa trong GDI+ phải được biến đổi một vài lần. Lý do về điều này là cho phép bạn xử lý đồ họa độc lập với thiết bị (device-independent). Chẳng hạn, nếu bạn đặt một hình hay một văn bản trong tọa độ thế giới thực, chắc có lẽ bạn muốn rằng kết quả đồ họa được hiển thị như là trên một tờ giấy A4 chuẩn. Hình ảnh phải trông như thế trên A4 cả trên màn hình lẫn trên máy in có độ phân giải nào đó. Cho nên tọa độ thế giới thực được ánh xạ vào tọa độ trên giấy (Page Coordinate) mà nó tương ứng với kích thước $8\frac{1}{2}'' \times 11''$. Khi hệ thống hiển thị trang A4, màn hình nên có độ phân giải 72 điểm cho mỗi inch (DPI) – máy in A có độ phân giải 300 DPI và máy in B có độ phân giải 400 DPI. Kế đó không gian trang giấy cần được ánh xạ vào không gian thiết bị, một cách chính xác, màn hình ánh xạ 8.5 inch trên trang giấy thành 612 điểm ảnh (pixel) trên màn hình – 2550 điểm ảnh trên máy in A và 10200 điểm ảnh trên máy in B.

GDI+ cho phép bạn cài đặt cách mà các đối tượng được biến đổi từ thế giới thực vào trang giấy sử dụng những thuộc tính `PageUnit`, `Pagescale`. Bạn không có quyền điều khiển phép biến đổi trên không gian thiết bị. Tất cả những điều đó xảy ra trong các drivers thiết bị của chúng. Tuy nhiên bạn có thể đọc DPI của thiết bị bằng các thuộc tính `DpiX` và `DpiY`.

Chúng ta sẽ khảo sát các phép biến đổi tọa độ một cách chi tiết trong chương này.

4. VẼ ĐƯỜNG THẲNG VÀ CÁC HÌNH ĐƠN GIẢN

Chúng ta sử dụng bút vẽ (pen) để vẽ các đường thẳng hay đường viền của một hình, và cọ vẽ (brush) để tô một vùng nằm bên trong của một hình. Ví dụ 3.5-1 dưới đây sẽ cho thấy những thao tác này.

Chú ý: Về việc quản lý các đối tượng đồ họa trong GDI+. Như bạn đã biết trong .NET, bộ thu gom rác GC (Garbage Collection) thực hiện việc quản lý bộ nhớ cho bạn. Về mặt kỹ thuật, bạn không cần phải thu hồi các đối tượng bằng cách hủy chúng bởi vì cuối cùng thì bộ đọn dẹp rác (garbage collector) sẽ làm điều này.

Điều này cũng đúng cho các thao tác tạo bút vẽ (pen) và cọ vẽ (brush). Tuy nhiên việc gởi các hình vẽ với tần suất cao có thể tạo ra hàng ngàn bút vẽ (pen) và cọ vẽ (brush) trong suốt mỗi chu kỳ vẽ, và trong những tình huống như vậy bộ đọn dẹp rác GC không thể thu hồi tất cả các đối tượng một cách nhanh chóng, và kết quả là một số lượng lớn các thành phần đã chết còn ở lại trên heap trong một khoảng thời gian đáng kể.

Để giảm thiểu số lượng công việc của bộ đọn dẹp rác GC, bất cứ khi nào có thể bạn nên gọi phương thức `Dispose()` một cách tường minh đối với các bút vẽ (pen), các cọ vẽ (brushes), và những đối tượng đồ họa mang tính chất tương tự. Điều này sẽ thu hồi bộ nhớ của đối tượng và thông báo đến bộ đọn dẹp rác GC không cần dành thời gian cho chúng.



Trong một số ví dụ, bạn sẽ thấy đối tượng được khai báo inline, chẳng hạn: `DrawRectangle(new Pen(Color.Rec), ...)`. Điều này sẽ không gây ra lỗi hổng trong bộ nhớ của đối tượng pen, đơn giản chỉ là trì hoãn việc thu hồi đối tượng khi nó không còn trong phạm vi nữa. Trong một số ví dụ khác, bạn sẽ thấy rõ ràng việc sử dụng phương thức này, chẳng hạn: `myPen.Dispose()`. Để bộ nhớ đạt hiệu suất cao nhất, chúng ta nên sử dụng phương thức này.

Ví dụ 3.5-1 DrawLines.cs: Vẽ Đường thẳng và tô Hình Chữ Nhật

```
1: using System;
2: using System.Drawing;
3: using System.Windows.Forms;
4:
5: class drawlines : System.Windows.Forms.Form
6: {
7:     Timer t;
8:     bool BackgroundDirty;
9:
10:    public void TickEventHandler(object sender,
11:                                EventArgs e)
12:    {
13:        Invalidate();
14:    }
15:    public void OnPaint(object sender,
16:                        PaintEventArgs e)
17:    {
18:        // đối tượng graphics hiện hành cho
19:        // window này ở trong PaintEventArgs
20:        Random r=new Random();
21:
22:        Color c = Color.FromArgb(r.Next(255),
23:                               r.Next(255),
24:                               r.Next(255));
25:        Pen p=new Pen(c,(float)r.NextDouble()*10);
26:        e.Graphics.DrawLine(p,
27:                            r.Next(this.ClientSize.Width),
28:                            r.Next(this.ClientSize.Height),
29:                            r.Next(this.ClientSize.Width),
30:                            r.Next(this.ClientSize.Height));
31:        p.Dispose();
32:    }
33:    protected override
34:    void OnPaintBackground(PaintEventArgs e)
```

```

32: {
33:     // Khi chúng ta thay đổi kích thước
34:     // hay chạy lần
35:     // đầu tiên. Chúng ta sẽ vẽ màu nền, ngược lại
36:     // nó sẽ làm mất các đường thẳng đã vẽ
37:     if(BackgroundDirty)
38:     {
39:         BackgroundDirty = false;
40:         // Chúng ta có thể gọi
41:         // base.OnPaintBackground(e);
42:         // nhưng điều đó không đưa ra
43:         // hình chữ nhật đang tô
44:         e.Graphics.FillRectangle(
45:             new SolidBrush(this.BackColor),
46:             this.ClientRectangle);
47:     }
48: }
49: public void OnSized(object sender, EventArgs e)
50: {
51:     BackgroundDirty=true;
52:     Invalidate();
53: }
54: public drawlines()
55: {
56:     t=new Timer();
57:     t.Interval=300;
58:     t.Tick+=new System.EventHandler(
59:         TickEventHandler);
60:     t.Enabled=true;
61:     this.Paint+=new PaintEventHandler(OnPaint);
62:     this.SizeChanged+=
63:         new EventHandler(OnSized);
64:     this.Text="Lines and lines and lines";
65:     BackgroundDirty = true;
66: }
67: static void Main()
68: {
69:     Application.Run(new drawlines());
}

```

Biên dịch ví dụ 3.5-1 bằng cách sử dụng dòng lệnh sau:
csc /t:winexe drawlines.cs

Dòng 53-62 tạo ra một bộ định thời (timer) và thêm bộ xử lý (handler) OnPaint vào danh sách các sự kiện của đối tượng Paint. Nó cũng thêm một bộ xử lý sự kiện cho sự kiện SizeChanged, để mà chúng ta có thể biết được khi nào một nền (background) mới cần phải vẽ lại.

Những dòng 10-13 là một bộ xử lý sự kiện Tick đơn giản chỉ thực hiện thao tác cập nhật nội dung cửa sổ. Các dòng 15-29 là bộ xử lý vẽ dùng tạo ra bút vẽ (pen) và vẽ một đường thẳng giữa hai điểm ngẫu nhiên trên màn hình.

Những dòng 31-43 vẽ màu nền cho chúng ta. Lý do có những dòng này là để cho những đường thẳng đã được vẽ không bị mất đi bởi việc cập nhật màn hình. Nếu bạn muốn thấy điều này xảy ra thì bạn cứ xóa đoạn mã này. Nó mô tả cách một cọ vẽ (brush) tô hình chữ nhật.

5. SỬ DỤNG BÚT VẼ (PEN) VÀ CỌ VẼ (BRUSH) CHUYỂN MÀU GAM MÀU

Bút vẽ (pen) và cọ vẽ (brush) đã tiến xa trong một thời gian ngắn. GDI+ cho phép bạn vẽ các đường thẳng và tô các vùng bằng chuyển đổi của màu sắc gam màu (gradient) hay một dãy nhiều màu. Sửa đổi đoạn mã trong ví dụ 3.5-1 sẽ cho phép chúng ta sử dụng các bút vẽ chuyển màu (gradient pens) thay cho những đường liền hay sử dụng những tính chất tô chuyển màu (gradient fills) thay cho những màu tô đồng nhất (solid colors).

Tô màu nền của cửa sổ khá dễ, chúng ta chỉ cần xác định cọ tô chuyển màu. Đối tượng LinearGradientBrush là một thành viên trong không gian tên (namespace) System.Drawing.Drawing2D. Ví dụ 3.5-2 là hai đoạn mã được chọn ra từ ví dụ 3.5-1 mà bạn cần thay đổi. Nó chỉ chứa hai phương thức OnPaint và OnPaintBackground đã được sửa đổi. Tuy nhiên, bạn cũng nhớ thêm khai báo sau vào mã nguồn của bạn.

```
Using System.Drawing.Drawing2D;
```

Ví dụ 3.5.2 DrawLines2.cs: Mã được sửa đổi từ ví dụ 3.5.1

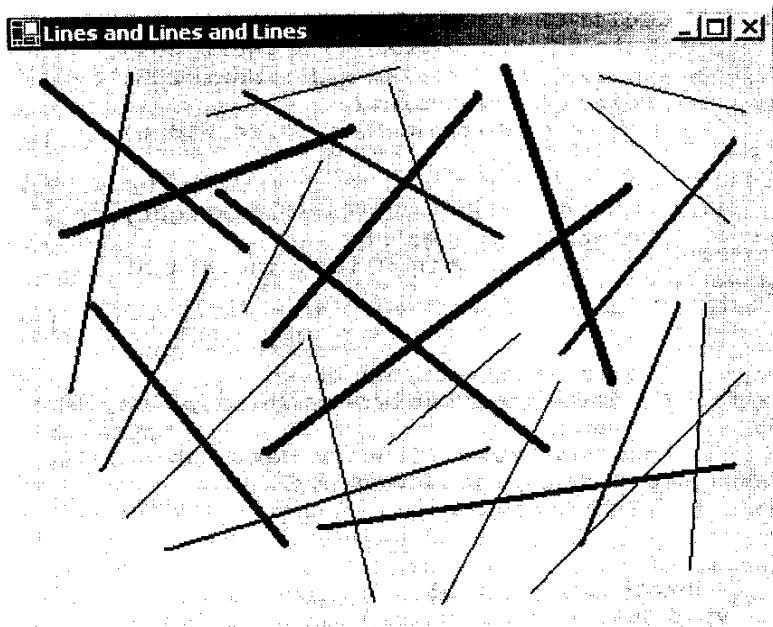
```
1: 
2:     public void OnPaint(object sender,
3:                           PaintEventArgs e)
4:     {
5:         // dổi tượng graphics hiện hành cho
6:         // window này ở trong PaintEventArgs
7:         Random r=new Random();
8:         Color cA = Color.FromArgb(r.Next(255),
9:                               r.Next(255),
10:                              r.Next(255));
```

```

10:     Color cB = Color.FromArgb(r.Next(255),
                               r.Next(255),
                               r.Next(255));
11:     Point pA = new Point(
                           r.Next(this.ClientSize.Width),
12:                               r.Next(this.ClientSize.Height));
13:     Point pBB= new Point(
                           r.Next(this.ClientSize.Width),
14:                               r.Next(this.ClientSize.Height));
15:     LinearGradientBrush brush =
           new LinearGradientBrush(pA,pB,cA,cB);
16:     Pen p=
           new Pen(brush,(float)r.NextDouble()*10);
17:     e.Graphics.DrawLine(p,pA,pB);
18:     p.Dispose();
19: }
20:
21: protected override
void OnPaintBackground(PaintEventArgs e)
22: {
23:     // Khi chúng ta thay đổi kích thước hay chạy lần
24:     // đầu tiên. Chúng ta sẽ vẽ màu nền, ngược lại
25:     // nó sẽ làm mất các đường thẳng đã vẽ
26:     if(BackgroundDirty)
27:     {
28:         BackgroundDirty = false;
29:         LinearGradientBrush gb =
30:             new LinearGradientBrush(
                           this.ClientRectangle,
                           Color.Navy,
                           Color.Aquamarine,
                           90);
31:         e.Graphics.FillRectangle(gb,
                           this.ClientRectangle);
32:         gb.Dispose();
33:     }
34: }

```

Hình 3.5-1 cho thấy ứng dụng đang chạy trong hào quang chói lọi



Hình 3.5-1 Chương trình DrawLines đã được sửa

Chú ý: Bạn nên chú ý rằng đối tượng pen tự bản thân nó sử dụng một cọ vẽ (brush) để tô màu nội tại của chính nó. Bạn cũng có thể sử dụng cọ vẽ theo một mẫu pattern để vẽ một đường thẳng. Điều này thật sự rất mạnh.

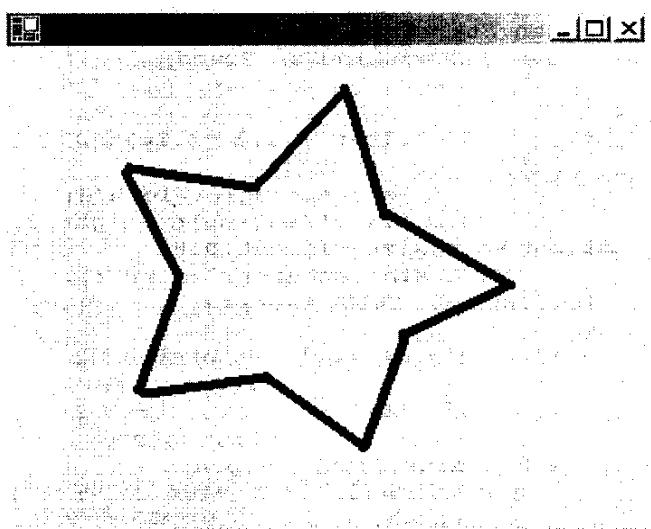
6. BÚT VẼ VÀ CỌ VẼ CÓ CHẤT LIỆU

Như bạn đã thấy trong ví dụ trước, một bút vẽ có thể được đưa cho một cọ vẽ để vẽ đường thẳng. Cọ vẽ có thể là nét liền (solid), chuyển màu (gradient), chất liệu (texture), hay mẫu tô. Một cọ vẽ có chất liệu (textured brush) có thể được sử dụng để vẽ những đường thẳng mà những đường thẳng này được vẽ bằng ảnh bitmap. Để thực hiện điều này, bạn có thể tạo một đối tượng bằng cách nạp một bitmap từ tập tin hay từ một tài nguyên, kế đó bạn truyền đối tượng này vào đối tượng pen, và cuối cùng vẽ các đường thẳng bằng bút vẽ này. Đoạn mã sau trình bày tiến trình này:

```
Image i = Image.FromFile("stone.bmp");
TextureBrush b = new TextureBrush(i);
Pen p = new Pen(p, 10);
e.Graphics.DrawLine(p, 0, 0, 100, 100);
p.Dispose();
```

7. LÀM TRÒN CÁC ĐẦU MÚT CỦA ĐƯỜNG THẲNG

Những đoạn thẳng được vẽ trong hình 3.5-1 có những đầu mút vuông vắn và có lẽ không đẹp đối với việc vẽ các đa giác. Hình 3.5-2 cho thấy một ngôi sao đơn giản được vẽ bằng những đoạn thẳng có độ rộng với các đầu mứt lởm chởm.



Hình 3.5-2 Những đầu mứt lởm chởm trong đa giác

Như bạn có thể thấy, các đỉnh đầu mứt thẳng làm cho điểm nối của các đường thẳng lởm chởm. Để làm gọn, chúng ta có thể chỉ ra đỉnh đầu mứt được làm tròn với các đường thẳng. Lớp bút vẽ (Pen) có hai thuộc tính: StartCap và EndCap. Hai thuộc tính này có thể được gán để vẽ các đầu mứt của đường thẳng theo một kiểu cụ thể. Trong ví dụ 3.5-3, chúng ta có chương trình vẽ ngôi sao và chương trình này cho phép chúng ta bật hoặc tắt các đỉnh đầu mứt bằng một nút (button) ở trên form.

Ví dụ 3.5-3 DrawStar.cs: Ví dụ về đầu mứt của đường thẳng

```

1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Windows.Forms;
5:
6: class drawstar : System.Windows.Forms.Form
7: {
8:     Button button1;
9:
10:    bool EndCap;
11:

```



```
12:     public void OnPaint(object sender,
13:                           PaintEventArgs e)
14:     {
15:         Pen pen=new Pen(Color.Black,(float)20.0);
16:         if(EndCap)
17:         {
18:             pen.StartCap=LineCap.Round;
19:             pen.EndCap=LineCap.Round;
20:         }
21:         float r36 = (float)(36.0 * 3.1415926 / 180);
22:         Point Center=new Point(
23:             this.ClientRectangle.Width/2,
24:             this.ClientRectangle.Height/2);
25:         float Pointsiz = (float)0.8
26:             *Math.Min(Center.X,Center.Y);
27:         for(int i=0; i<10; i++)
28:         {
29:             float d1=(i&1)==1 ? Pointsiz / 2
30:                               : Pointsiz;
31:             float d2=(i&1)==0 ? Pointsiz / 2
32:                               : Pointsiz;
33:             e.Graphics.DrawLine(pen,
34:                 new Point((int)(d1*Math.Cos(r36*i))
35:                           +Center.X,
36:                           (int)(d1*Math.Sin(r36*i))+Center.Y),
37:                 new Point(
38:                     (int)(d2*Math.Cos(r36*(i+1)))
39:                     +Center.X,
40:                     (int)(d2*Math.Sin(r36*(I+1))
41:                         +Center.Y)));
42:         }
43:         pen.Dispose();
44:     }
45:     public void OnSizeChanged(object sender,
46:                               EventArgs e)
47:     {
48:         EndCap = !EndCap;
49:         Invalidate();
50:     }
51: }
```



```

49:     public drawstar()
50:     {
51:         this.Paint+=new PaintEventHandler(OnPaint);
52:         this.Text="Star";
53:         this.SizeChanged+=
54:             new EventHandler(OnSizeChanged);
54:
55:         button1 = new Button();
56:         button1.Location=new Point(5,5);
57:         button1.Size=new Size(50,20);
58:         button1.Text = "Caps";
59:         button1.Click+=
60:             new EventHandler(OnClickedButton1);
61:         this.Controls.Add(button1);
61:
62:     }
63:
64:     static void Main()
65:     {
66:         Application.Run(new drawstar());
67:     }
68: };

```

Biên dịch chương trình này với dòng lệnh sau:

csc /t:winexe drawstart.cs

Khi bạn chạy chương trình, bạn sẽ thấy một nút nhấn ở góc trên bên trái của cửa sổ, bấm lên nút này để thay đổi đỉnh đầu mút thành tròn. Có tất cả 11 kiểu đỉnh đường thẳng khác nhau, kể cả kiểu tùy thích.

Bạn đã có thể thấy rằng GDI+ là một sự cải tiến tuyệt vời hơn cả GDI đáng tin cậy xưa kia, và chúng ta chỉ mới thật sự xem xét đường thẳng và hình chữ nhật. Bây giờ chúng ta hãy xem xét đường cong, đường đồ thị

8. ĐƯỜNG CONG VÀ CÁC ĐƯỜNG ĐỒ THỊ

Vào lúc có GDI đã có đường cong Bezier và các đường đồ thị. GDI+ mở rộng những khả năng này bằng cách cung cấp một tập chức năng phong phú cho các đường cong và đối tượng đường đồ thị.

Vẽ đường cong từ một mảng (array) các điểm là hình thức đơn giản nhất. Đường cong được vẽ như thế sẽ đi qua mỗi điểm của mảng. Độ căng của đường cong đối với điểm cụ thể nào đó được kiểm soát bởi vị trí của những điểm ở hai bên cạnh nó trong mảng. Đường cong Bezier đúng chuẩn có hai điểm liên kết với mỗi nút (node) – một điểm cho vị trí và một điểm để kiểm soát. Độ căng của đường cong được xác định bởi mối quan hệ của điểm kiểm soát với hướng của đường thẳng.



Ví dụ 3.5-4 minh họa cách dùng phương thức Graphics.DrawCurve bằng cách tạo một tập các vị trí trong phạm vi của hình chữ nhật trên màn hình. Một đường cong cơ sở được vẽ dọc theo mảng các điểm này, và bạn có thể chọn độ căng của đường cong bằng một thanh trượt (slider).

Ví dụ 3.5-4 DrawCurve.cs: Sử dụng phương thức DrawCurve

```
1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Collections;
5: using System.ComponentModel;
6: using System.Windows.Forms;
7: using System.Data;
8:
9: namespace Curves
10: {
11:     /// <summary>
12:     /// This simple object bounces points
13:     /// around a rectangular area.
14:     /// </summary>
15:     class Bouncer
16:     {
17:         int dirx;
18:         int diry;
19:         int X;
20:         int Y;
21:
22:         Size size;
23:
24:         public Bouncer()
25:         {
26:             dirx=diry=1;
27:         }
28:
29:         public void Move()
30:         {
31:             X+=dirx;
32:             Y+=diry;
33:
34:             if(X<=0 || X>=size.Width)
35:                 dirx*=-1;
36:
37:             if(Y<=0 || Y>=size.Height)
38:                 diry*=-1;
39:         }
40:
41:         public Point Position
```

```
42:         {
43:             get{return new Point(X,Y);}
44:             set{X=value.X; Y=value.Y;}
45:         }
46:
47:         public Size Size
48:         {
49:             get{ return size;}
50:             set{size = value;}
51:         }
52:     }
53:
54: /// <summary>
55: /// Summary description for Form1.
56: /// </summary>
57: public class Curves : System.Windows.Forms.Form
58: {
59:
60:     Timer bounce,paint;
61:     TrackBar trk;
62:
63:     Bouncer[] bouncers;
64:
65:     void OnTickBounce(object sender, EventArgs e)
66:     {
67:         foreach(Bouncer b in bouncers)
68:             b.Move();
69:     }
70:
71:     void OnTickPaint(object sender, EventArgs e)
72:     {
73:         Invalidate();
74:     }
75:
76:     public void OnPaint(object sender,
77:                         PaintEventArgs e)
78:     {
79:         Pen p=new Pen(Color.Red,10);
80:         SolidBrush br=new SolidBrush(Color.Blue);
81:         Point [] points = new Point[bouncers.Length];
82:         int i=0;
83:         // cần chuyển bouncers thành một mảng các điểm
84:         foreach(Bouncer b in bouncers)
85:             points[i++]=b.Position;
86:         // Vẽ đường cong
e.Graphics.DrawCurve(p,points,0,
                    points.Length-1,
```

```
        (float)trk.Value);
87:    //Bây giờ vẽ các nút trong đường cong.
88:    foreach(Point pn in points)
89:        e.Graphics.FillEllipse(br,pn.X-5,
90:                               pn.Y-5,10,10);
91:    p.Dispose();
92:    br.Dispose();
93:
94:    public Curves()
95:    {
96:        this.Paint+=new PaintEventHandler(OnPaint);
97:        // Bộ định thời để quản lý bouncing.
98:        bounce=new Timer();
99:        bounce.Interval=5;
100:       bounce.Tick+=
101:           new EventHandler(OnTickBounce);
102:           // Bộ định thời để quản lý việc vẽ lại.
103:           paint=new Timer();
104:           paint.Interval=100;
105:           paint.Tick+=new EventHandler(OnTickPaint);
106:           // Bộ tạo số ngẫu nhiên cho những vị trí ban đầu
107:           Random r=new Random();
108:           // Kích thước ban đầu của Form
109:           this.Size=new Size(800,600);
110:           //khởi tạo mảng chứa các điểm bouncing
111:           bouncers = new Bouncer[6];
112:           for(int i=0;i<6;i++)
113:           {
114:               bouncers[i]=new Bouncer();
115:               bouncers[i].Position=
116:                   new Point(r.Next(800),r.Next(600));
117:               bouncers[i].Size=new Size(800,600);
118:           }
119:           // Bật các bộ định thời
120:           bounce.Enabled=true;
121:           paint.Enabled=true;
122:           //tạo thanh trượt để điều khiển
123:           //độ cong của đường
124:           trk = new TrackBar();
125:           trk.Location=new Point(5,25);
126:           trk.Size=new Size(100,20);
127:           trk.Minimum=1;
128:           trk.Maximum=10;
129:           trk.Value=2;
           this.Controls.Add(trk);
           //và dán nhãn cho người sử dụng
           Label lb=new Label();
```

```

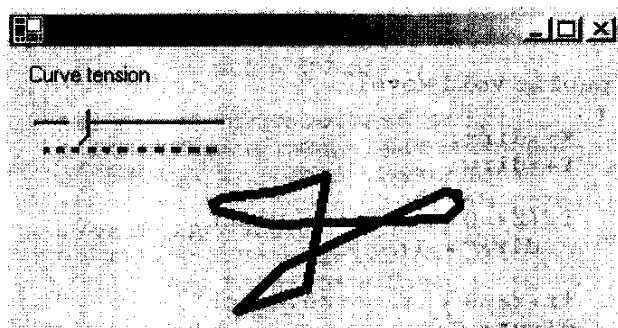
130:     lb.Location=new Point(5,5);
131:     lb.Size=new Size(100,20);
132:     lb.Text="Curve tension";
133:     this.Controls.Add(lb);
134: }
135:
136: static void Main()
137: {
138:     Application.Run(new Curves());
139: }
140: }
141: }

```

Biên dịch ví dụ 3.5-4 với dòng lệnh sau:
csc /t:winexe drawcurve.cs

Công việc này được làm bởi hai bộ định thời (timer) và một bộ xử lý paint. Bộ định thời số 1 (dòng 98 và 65–69) tạo ra vị trí của những nút (node) trên đường cong trong phạm vi màn hình. Bộ định thời số 2 (dòng 102 và 71–74) cập nhật màn hình để cho việc vẽ không xảy ra quá thường xuyên. Bộ xử lý (handler) paint (dòng 76–90) tạo một mảng các điểm từ đối tượng bounce; tuy nhiên, Point là một lớp bị “đóng kín” nên bạn không thể kế thừa nó được. Sau đó dòng 86 vẽ đường cong, độ căng của đường cong phụ thuộc vào vị trí của điều khiển Trackbar. Ví dụ này cho thấy một cách rõ ràng hiệu chỉnh độ căng của đường đường cong đơn giản.

Hình 3.5-3 cho thấy ứng dụng DrawCurves đang chạy



Hình 3.5-3 Vẽ đường cong với độ căng

Đường cong Bezier thì hơi khó. Đối với mỗi đoạn thẳng, ngoài hai điểm xác định đầu và cuối của đường thẳng, còn có hai điểm kiểm soát khác để tăng thêm

độ căng và độ cong của đường thẳng. Ví dụ 3.5-5 cho thấy đường cong Bezier được dùng như thế nào.

Ví dụ 3.5-5 Beziercurves.cs: Sử dụng đường cong Bezier

```
1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Collections;
5: using System.ComponentModel;
6: using System.Windows.Forms;
7: using System.Data;
8:
9: namespace Curves
10: {
11:     /// <summary>
12:     /// This simple object bounces points
13:     /// around a rectangular area.
14:     /// </summary>
15:     class Bouncer
16:     {
17:         int dirx;
18:         int diry;
19:         public int X;
20:         public int Y;
21:
22:         Size size;
23:
24:         public Bouncer()
25:         {
26:             dirx=diry=1;
27:         }
28:
29:         public void Move()
30:         {
31:             X+=dirx;
32:             Y+=diry;
33:
34:             if(X<=0 || X>=size.Width)
35:                 dirx*=-1;
36:
37:             if(Y<=0 || Y>=size.Height)
38:                 diry*=-1;
39:         }
40:
41:         public Point Position
42:         {
43:             get{return new Point(X,Y);}
```

```
44:     set{X=value.X; Y=value.Y; }
45:   }
46:
47:   public Size Size
48:   {
49:     get{ return size; }
50:     set{size = value; }
51:   }
52: }
53:
54: /// <summary>
55: /// Summary description for Form1.
56: /// </summary>
57: public class BezierCurves
58:   : System.Windows.Forms.Form
59: {
60:   Timer bounce,paint;
61:   Bouncer[] bouncers;
62:
63:   void OnTickBounce(object sender, EventArgs e)
64:   {
65:     foreach(Bouncer b in bouncers)
66:       b.Move();
67:   }
68:
69:   void OnTickPaint(object sender, EventArgs e)
70:   {
71:     Invalidate();
72:   }
73:
74:   public void OnPaint(object sender,
75:                         PaintEventArgs e)
76:   {
77:     Pen p=new Pen(Color.Red,10);
78:     p.StartCap=LineCap.DiamondAnchor;
79:     p.EndCap=LineCap.ArrowAnchor;
80:     SolidBrush br=new SolidBrush(Color.Blue);
81:     //Vẽ đường cong
82:     e.Graphics.DrawBezier(p,
83:                           new Point(550,300),
84:                           bouncers[0].Position,
85:                           bouncers[1].Position,
86:                           new Point(50,300));
87:     //bây giờ vẽ các nút trên đường cong.
88:     foreach(Bouncer b in bouncers)
89:       e.Graphics.FillEllipse(br,b.X-5,
90:                             b.Y-5,10,10);
```



```
88:      //và cho thấy mối quan hệ giữa nút bouncing
89:      //và điểm đầu mút của đường cong bezier.
90:      p=new Pen(Color.Black,1);
91:      p.DashStyle=DashStyle.DashDotDot;
92:      e.Graphics.DrawLine(p,
93:          bouncers[0].Position,
94:          new Point(550,300));
95:      e.Graphics.DrawLine(p,
96:          bouncers[1].Position,
97:          new Point(50,300));
98:      p.Dispose();
99:  }
100: public BezierCurves()
101: {
102:     this.Paint+=new PaintEventHandler(OnPaint);
103:     // A timer to manage the bouncing
104:     bounce=new Timer();
105:     bounce.Interval=5;
106:     bounce.Tick+=
107:         new EventHandler(OnTickBounce);
108:     // Bộ định thời để quản lý việc vẽ lại
109:     paint=new Timer();
110:     paint.Interval=100;
111:     paint.Tick+=new EventHandler(OnTickPaint);
112:     // Bộ tạo số ngẫu nhiên cho những vị trí
113:     //ban đầu
114:     Random r=new Random();
115:     //kích thước ban đầu của form
116:     this.Size=new Size(800,600);
117:     //khởi tạo một mảng chứa các điểm bouncing.
118:     bouncers = new Bouncer[2];
119:     for(int i=0;i<2;i++)
120:     {
121:         bouncers[i]=new Bouncer();
122:         bouncers[i].Position=
123:             new Point(r.Next(800),r.Next(600));
124:         bouncers[i].Size=new Size(800,600);
125:     }
126:     static void Main()
127:     {
128:         Application.Run(new BezierCurves());
129:     }
```

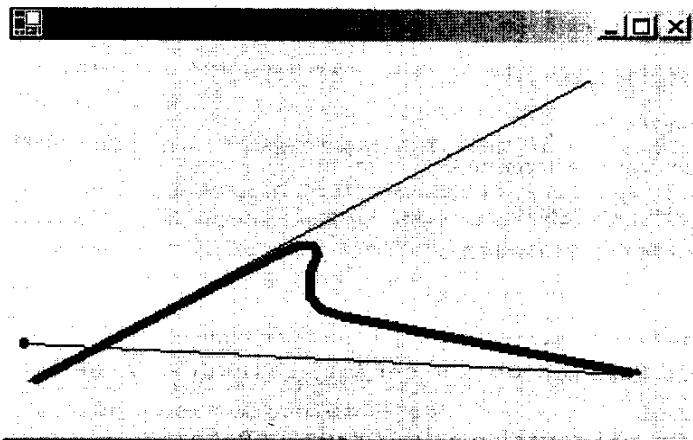
```

    it bouncing } : }
    bezier. ) : }

Dot;

```

Phần chủ yếu của ứng dụng này thì thật sự tương tự với cái đã được trình ng ví dụ 3.5-4. Những điểm đáng chú ý là những dòng 83-86 vẽ đường cong và những dòng 92-96 trình bày cho bạn cách vẽ những đường thẳng nét nh trong hình 3.5-4 là ảnh của một màn hình của ứng dụng và nó cho thấy điểm kiểm soát của đường cong Bezier được sử dụng như thế nào để tăng ường và độ căng đối với một nút cụ thể.



Hình 3.5-4 Đường cong Bezier đang hoạt động

Tập mảng những điểm cũng có thể được sử dụng để tạo một đường cong nhiều đoạn. Mảng này phải có một bộ bốn điểm và được sắp xếp như

oint[0] = Điểm đầu của đoạn thẳng 1

oint[1] = Điểm kiểm soát cho điểm đầu

.Next(600)) point[2] = Điểm kiểm soát cho điểm cuối

(800, 600); point[3] = Điểm cuối của đoạn thẳng 1 và là điểm đầu của đoạn thẳng 2

oint[4] = Điểm kiểm soát cho điểm đầu của đoạn thẳng 2

ho 4 điểm mở đầu cộng thêm 3 điểm cho mỗi đoạn thẳng theo sau.

TƯỢNG GRAPHICSPATH

Đường đồ thị đồ họa trong GDI+ là một cách thích hợp để tập hợp các họa lại, hoặc tối thiểu cũng là các đường biên của chúng, vào một đơn vị. Khi một đường đồ thị đã được tạo, chúng ta có thể vận dụng nó tô màu,

Ví dụ 3.5-4 minh họa cách dùng phương thức Graphics.DrawCurve để cách tạo một tập các vị trí trong phạm vi của hình chữ nhật trên màn hình. M đường cong cơ sở được vẽ theo mảng các điểm này, và bạn có thể chọn độ c của đường cong bằng một thanh trượt (slider).

Ví dụ 3.5-4 DrawCurve.cs: Sử dụng phương thức DrawCurve

```
1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Collections;
5: using System.ComponentModel;
6: using System.Windows.Forms;
7: using System.Data;
8:
9: namespace Curves
10: {
11:     /// <summary>
12:     /// This simple object bounces points
13:     /// around a rectangular area.
14:     /// </summary>
15:     class Bouncer
16:     {
17:         int dirx;
18:         int diry;
19:         int X;
20:         int Y;
21:
22:         Size size;
23:
24:         public Bouncer()
25:         {
26:             dirx=diry=1;
27:         }
28:
29:         public void Move()
30:         {
31:             X+=dirx;
32:             Y+=diry;
33:
34:             if(X<=0 || X>=size.Width)
35:                 dirx*=-1;
36:
37:             if(Y<=0 || Y>=size.Height)
38:                 diry*=-1;
39:         }
40:
41:         public Point Position
```



```
42:         {
43:             get{return new Point(X,Y);}
44:             set{X=value.X; Y=value.Y;}
45:         }
46:
47:         public Size Size
48:         {
49:             get{ return size;}
50:             set{size = value;}
51:         }
52:     }
53:
54: /// <summary>
55: /// Summary description for Form1.
56: /// </summary>
57: public class Curves : System.Windows.Forms.Form
58: {
59:
60:     Timer bounce,paint;
61:     TrackBar trk;
62:
63:     Bouncer[] bouncers;
64:
65:     void OnTickBounce(object sender, EventArgs e)
66:     {
67:         foreach(Bouncer b in bouncers)
68:             b.Move();
69:     }
70:
71:     void OnTickPaint(object sender, EventArgs e)
72:     {
73:         Invalidate();
74:     }
75:
76:     public void OnPaint(object sender,
77:                         PaintEventArgs e)
78:     {
79:         Pen p=new Pen(Color.Red,10);
80:         SolidBrush br=new SolidBrush(Color.Blue);
81:         Point[] points = new Point[bouncers.Length];
82:         int i=0;
83:         // cần chuyển bouncers thành một mảng các điểm
84:         foreach(Bouncer b in bouncers)
85:             points[i++]=b.Position;
86:         //Vẽ đường cong
e.Graphics.DrawCurve(p,points,0,
                    points.Length-1,
```



```

        (float)trk.Value);
87:     //Bây giờ vẽ các nút trong đường cong.
88:     foreach(Point pn in points)
89:         e.Graphics.FillEllipse(br,pn.X-5,
90:             pn.Y-5,10,10);
91:         p.Dispose();
92:     }
93:
94:     public Curves()
95:     {
96:         this.Paint+=new PaintEventHandler(OnPaint);
97:         // Bộ định thời để quản lý bouncing.
98:         bounce=new Timer();
99:         bounce.Interval=5;
100:        bounce.Tick+=
101:            new EventHandler(OnTickBounce);
102:        // Bộ định thời để quản lý việc vẽ lại.
103:        paint=new Timer();
104:        paint.Interval=100;
105:        paint.Tick+=new EventHandler(OnTickPaint);
106:        // Bộ tạo số ngẫu nhiên cho những vị trí ban đầu
107:        Random r=new Random();
108:        // Kích thước ban đầu của Form
109:        this.Size=new Size(800,600);
110:        //khởi tạo máng chứa các điểm bouncing
111:        bouncers = new Bouncer[6];
112:        for(int i=0;i<6;i++)
113:        {
114:            bouncers[i]=new Bouncer();
115:            bouncers[i].Position=
116:                new Point(r.Next(800),r.Next(600));
117:            bouncers[i].Size=new Size(800,600);
118:        }
119:        // Bật các bộ định thời
120:        bounce.Enabled=true;
121:        paint.Enabled=true;
122:        //tạo thanh trượt để điều khiển
123:        //độ cong của đường
124:        trk = new TrackBar();
125:        trk.Location=new Point(5,25);
126:        trk.Size=new Size(100,20);
127:        trk.Minimum=1;
128:        trk.Maximum=10;
129:        trk.Value=2;
        this.Controls.Add(trk);
        //và dán nhãn cho người sử dụng
        Label lb=new Label();
    }
}

```

```

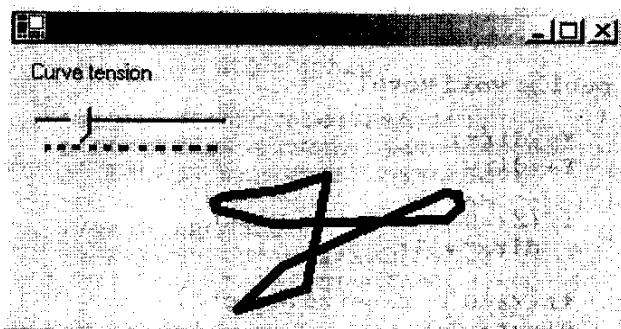
130:     lb.Location=new Point(5,5);
131:     lb.Size=new Size(100,20);
132:     lb.Text="Curve tension";
133:     this.Controls.Add(lb);
134: }
135:
136: static void Main()
137: {
138:     Application.Run(new Curves());
139: }
140: }
141: }

```

Biên dịch ví dụ 3.5-4 với dòng lệnh sau:
csc /t:winexe drawcurve.cs

Công việc này được làm bởi hai bộ định thời (timer) và một bộ xử lý paint. Bộ định thời số 1 (dòng 98 và 65–69) tạo ra vị trí của những nút (node) trên đường cong trong phạm vi màn hình. Bộ định thời số 2 (dòng 102 và 71–74) cập nhật màn hình để cho việc vẽ không xảy ra quá thường xuyên. Bộ xử lý (handler) paint (dòng 76–90) tạo một mảng các điểm từ đối tượng bounce; tuy nhiên, Point là một lớp bị “đóng kín” nên bạn không thể kế thừa nó được. Sau đó dòng 86 vẽ đường cong, độ căng của đường cong phụ thuộc vào vị trí của điều khiển Trackbar. Ví dụ này cho thấy một cách rõ ràng hiệu chỉnh độ căng của đường cong đơn giản.

Hình 3.5-3 cho thấy ứng dụng DrawCurves đang chạy



Hình 3.5-3 Vẽ đường cong với độ căng

Đường cong Bezier thì hơi khó. Đối với mỗi đoạn thẳng, ngoài hai điểm xác định đầu và cuối của đường thẳng, còn có hai điểm kiểm soát khác để tăng thêm



độ căng và độ cong của đường thẳng. Ví dụ 3.5-5 cho thấy đường cong Bezier được dùng như thế nào.

Ví dụ 3.5-5 Beziers.cs: Sử dụng đường cong Bezier

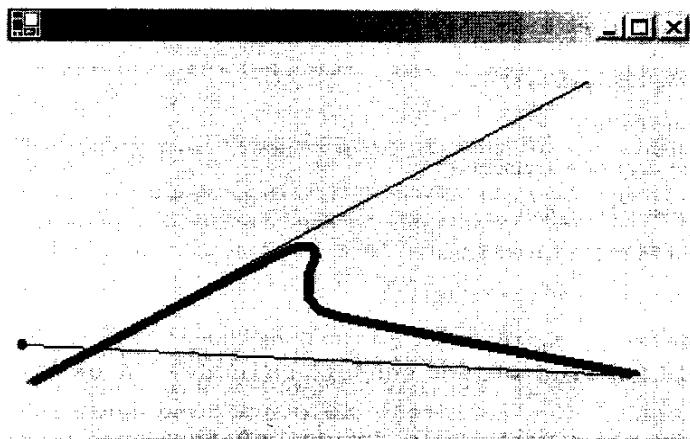
```
1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Collections;
5: using System.ComponentModel;
6: using System.Windows.Forms;
7: using System.Data;
8:
9: namespace Curves
10: {
11:     /// <summary>
12:     /// This simple object bounces points
13:     /// around a rectangular area.
14:     /// </summary>
15:     class Bouncer
16:     {
17:         int dirx;
18:         int diry;
19:         public int X;
20:         public int Y;
21:
22:         Size size;
23:
24:         public Bouncer()
25:         {
26:             dirx=diry=1;
27:         }
28:
29:         public void Move()
30:         {
31:             X+=dirx;
32:             Y+=diry;
33:
34:             if(X<=0 || X>=size.Width)
35:                 dirx*=-1;
36:
37:             if(Y<=0 || Y>=size.Height)
38:                 diry*=-1;
39:         }
40:
41:         public Point Position
42:         {
43:             get{return new Point(X,Y);}
```

```
44:         set{X=value.X; Y=value.Y; }
45:     }
46:
47:     public Size Size
48:     {
49:         get{ return size; }
50:         set{size = value; }
51:     }
52: }
53:
54: /// <summary>
55: /// Summary description for Form1.
56: /// </summary>
57: public class BezierCurves
58: : System.Windows.Forms.Form
59: {
60:     Timer bounce,paint;
61:     Bouncer[] bouncers;
62:
63:     void OnTickBounce(object sender, EventArgs e)
64:     {
65:         foreach(Bouncer b in bouncers)
66:             b.Move();
67:     }
68:
69:     void OnTickPaint(object sender, EventArgs e)
70:     {
71:         Invalidate();
72:     }
73:
74:     public void OnPaint(object sender,
75:                         PaintEventArgs e)
76:     {
77:         Pen p=new Pen(Color.Red,10);
78:         p.StartCap=LineCap.DiamondAnchor;
79:         p.EndCap=LineCap.ArrowAnchor;
80:         SolidBrush br=new SolidBrush(Color.Blue);
81:         //Vẽ đường cong
82:         e.Graphics.DrawBezier(p,
83:                               new Point(550,300),
84:                               bouncers[0].Position,
85:                               bouncers[1].Position,
86:                               new Point(50,300));
87:         //bây giờ vẽ các nút trên đường cong.
88:         foreach(Bouncer b in bouncers)
89:             e.Graphics.FillEllipse(br,b.X-5,
90:                                   b.Y-5,10,10);
```

```
88:     //và cho thấy mối quan hệ giữa nút bouncing
89:     //và điểm đầu mút của đường cong bezier.
90:     p=new Pen(Color.Black,1);
91:     p.DashStyle=DashStyle.DashDotDot;
92:     e.Graphics.DrawLine(p,
93:         bouncers[0].Position,
94:         new Point(550,300));
95:     e.Graphics.DrawLine(p,
96:         bouncers[1].Position,
97:         new Point(50,300));
98:     p.Dispose();
99: }
100:
101: public BezierCurves()
102: {
103:     this.Paint+=new PaintEventHandler(OnPaint);
104:     // A timer to manage the bouncing
105:     bounce=new Timer();
106:     bounce.Interval=5;
107:     bounce.Tick+=
108:         new EventHandler(OnTickBounce);
109:     // Bộ định thời để quản lý việc vẽ lại
110:     paint=new Timer();
111:     paint.Interval=100;
112:     paint.Tick+=new EventHandler(OnTickPaint);
113:     // Bộ tạo số ngẫu nhiên cho những vị trí
114:     //ban đầu
115:     Random r=new Random();
116:     //kích thước ban đầu của form
117:     this.Size=new Size(800,600);
118:     //khởi tạo một mảng chứa các điểm bouncing.
119:     bouncers = new Bouncer[2];
120:     for(int i=0;i<2;i++)
121:     {
122:         bouncers[i]=new Bouncer();
123:         bouncers[i].Position=
124:             new Point(r.Next(800),r.Next(600));
125:         bouncers[i].Size=new Size(800,600);
126:     }
127:     static void Main()
128:     {
129:         Application.Run(new BezierCurves());
130:     }
```

```
129 : }
130 : }
```

Phần chủ yếu của ứng dụng này thì thật sự tương tự với cái đã được trình bày trong ví dụ 3.5-4. Những điểm đáng chú ý là những dòng 83–86 vẽ đường cong Bezier, và những dòng 92–96 trình bày cho bạn cách vẽ những đường thẳng nét dứt. Ảnh trong hình 3.5-4 là ảnh của một màn hình của ứng dụng và nó cho thấy những điểm kiểm soát của đường cong Bezier được sử dụng như thế nào để tăng thêm hướng và độ căng đối với một nút cụ thể.



Hình 3.5-4 Đường cong Bezier đang hoạt động

Một mảng những điểm cũng có thể được sử dụng để tạo một đường cong Bezier có nhiều đoạn. Mảng này phải có một bộ bốn điểm và được sắp xếp như sau:

Point[0] = Điểm đầu của đoạn thẳng 1

Point[1] = Điểm kiểm soát cho điểm đầu

Point[2] = Điểm kiểm soát cho điểm cuối

Point[3] = Điểm cuối của đoạn thẳng 1 và là điểm đầu của đoạn thẳng 2

Point[4] = Điểm kiểm soát cho điểm đầu của đoạn thẳng 2

...

Cho 4 điểm mở đầu cộng thêm 3 điểm cho mỗi đoạn thẳng sau.

9. ĐỐI TƯỢNG GRAPHICSPATH

Các đường đồ thị đồ họa trong GDI+ là một cách thích hợp để tập hợp các hình đồ họa lại, hoặc tối thiểu cũng là các đường biên của chúng, vào một đơn vị duy nhất. Khi một đường đồ thị đã được tạo, chúng ta có thể vận dụng nó tô màu,

vẽ nét, hoặc có thể sử dụng để thực hiện những thao tác đồ họa khác, chẳng hạn như sử dụng để tạo một vùng cắt.

Bất cứ tổ hợp nào của các thành phần đồ họa cơ bản đều có thể đặt trong đối tượng đường đồ thị Path:

Cung (Arcs)

Đường cong Bezier

Đường cong cơ bản

Ellipses

Đường thẳng (Lines)

Đường đồ thị

Hình quạt (Pie segments)

Đa giác (Polygons)

Hình chữ nhật (Rectangles)

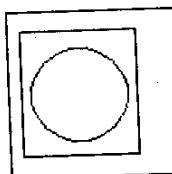
Hình ảnh của ký tự từ chuỗi

Đoạn mã dưới đây cho ra kết quả như trong hình 3.5-5.

```
void OnPaint(Object sender, PaintEventArgs e)
```

```
{
```

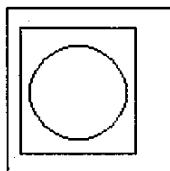
```
    GraphicsPath p = new GraphicsPath();
    p.AddEllipse(10,10,100,100);
    p.AddRectangle(new Rectangle(0,0,120,120));
    Pen pen = new Pen(Color.Black,1);
    e.Graphics.DrawPath(pen,p);
    pen.Dispose();
}
```



Hình 3.5-5 Hai hình trong đối tượng GraphicsPath

Tô đường path bằng cách sử dụng một cọ vẽ thích hợp và phương thức cho ra kết quả như trong hình 3.5-6.

```
SolidBrush brush = new SolidBrush(Color.Blue);
e.Graphics.FillPath(brush,p);
```

**Hình 3.5-6 Path được tô**

Phần hình vuông của đường đồ thị đã được tô và đường tròn thì không. Đó là vì thuộc tính FillMode của đối tượng Path được gán mặc định là Alternate.

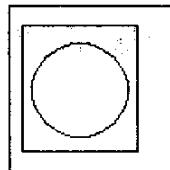
Để tô hoàn toàn bên trong đường biên xa nhất của hình, ta gán cho FillMode giá trị Winding.

9.1. Thêm văn bản (Text) và các đường đồ thị khác

Ta có thể thêm một đường đồ thị khác vào một đường đường đồ thị có sẵn khá dễ dàng bằng cách gọi phương thức AddPath. Đoạn mã dưới đây tạo ra đường đồ thị thứ hai và thêm nó vào đường đường đồ thị đã được trình bày trong hình 3.5-5

```
void OnPaint(object sender, PaintEventArgs e)
{
    GraphicsPath p = new GraphicsPath();
    p.AddEllipse(10, 10, 100, 100);
    p.AddRectangle(new Rectangle(0, 0, 120, 120));
    Pen pen = new Pen(Color.Black, 1);
    GraphicsPath p2 = new GraphicsPath();
    Point[] tripoint = new Point[3];
    tripoint[0] = new Point(80, 10);
    tripoint[1] = new Point(80, 110);
    tripoint[2] = new Point(150, 60);
    p2.AddClosedCurve(tripoint, (float)0);
    p2.AddPath(p, true);
    SolidBrush brush = new SolidBrush(Color.Blue);
    e.Graphics.FillPath(brush, p2);
    e.Graphics.DrawPath(pen, p2);
}
```

Hình ảnh hiển thị lúc này gồm cả hai đường đồ thị. Hình 3.5-7 cho thấy kết xuất từ việc chúng ta đã sửa đổi bộ xử lý OnPaint.

**Hình 3.5-7 Kết hợp hai đường đồ thị**



Chuyển các ký tự trong văn bản (text) thành dạng hình ảnh đồ họa có các đường đồ thị. Đây là cách tạo các hiệu ứng văn bản (text effects) tuyệt vời. Một đường đồ thị của văn bản (text path) chỉ chứa các đường viền hình ảnh của các ký tự được sử dụng. Chúng ta sử dụng những đường viền này để tạo các đường cắt (clip paths), để tô với các mẫu tô hoặc màu, để co dãn theo tỷ lệ, để quay, hay dùng phép biến đổi để tạo nên một vài hiệu ứng khá ấn tượng.

Sửa đổi trong bộ xử lý OnPaint của chúng ta sẽ minh họa lại điều này được hoàn thành như thế nào.

```
void OnPaint(object sender, PaintEventArgs e)
{
    GraphicsPath p = new GraphicsPath();
    p.AddString("AYBABTU",
                FontFamily.GenericSanserif,
                0, (float)72, new Point(0,0),
                StringFormat.GenericDefault);
    SolidBrush brush = new SolidBrush(Color.Blue);
    e.Graphics.FillPath(brush,p);
    brush.Dispose();
}
```

Chúng ta sẽ tiếp tục thảo luận điều này trong phần kế bởi vì nó dẫn đến một vấn đề thú vị.

10. KỸ THUẬT CẮT BẰNG ĐƯỜNG ĐỒ THỊ VÀ VÙNG (REGION)

Một vùng (region) là một hình khép kín mà có thể được sử dụng như một mặt nạ để thực hiện các thao tác đồ họa. Vùng có thể là những hình đều, như hình chữ nhật hay elip; và cũng có thể là những hình không đều, những hình này có thể được tạo từ những đường cong hoặc các ánh của văn bản (text glyphs). Vì vùng có thể được tạo từ các đường, các đường này có thể rất phức tạp, hay các hình đã bị cắt. Trở lại ví dụ về text path trước đó, chúng ta có thể tạo một vùng cắt (clipping region) từ text path này và sử dụng nó cho những điều thú vị khác.

Trong ví dụ 3.5-6, chúng ta sử dụng một đường đồ thị, được tô bằng một chuỗi văn bản.

Ví dụ 3.5-6 ClipToPath.cs: Sử dụng path để cắt vùng đang vẽ

```
1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Collections;
5: using System.ComponentModel;
```

```
6:using System.Windows.Forms;
7:using System.Data;
8:
9:namespace Clipping
10: {
11:     public class ClipToPath
12:         :System.Windows.Forms.Form
13:     {
14:         Timer timer;
15:         GraphicsPath p;
16:         bool dirty;
17:
18:         void OnPaint(Object sender,PaintEventArgs e)
19:         {
20:             Random r = new Random();
21:             SolidBrush brush = new SolidBrush(
22:                 Color.FromArgb(r.Next(255),
23:                                 r.Next(255),
24:                                 r.Next(255)));
25:             e.Graphics.SetClip(p);
26:             e.Graphics.FillEllipse(brush,
27:                                 r.Next(500),
28:                                 r.Next(200),
29:                                 r.Next(20),
30:                                 r.Next(20));
31:             brush.Dispose();
32:         }
33:
34:         void OnTick(object sender, EventArgs e)
35:         {
36:             Invalidate();
37:         }
38:
39:         protected override
void OnPaintBackground(PaintEventArgs e)
40:         {
41:             if(dirty)
42:             {
43:                 e.Graphics.ResetClip();
44:                 SolidBrush b =
45:                     new SolidBrush(this.BackColor);
46:                 e.Graphics.FillRectangle(b,
47:                                         this.ClientRectangle);
48:                 dirty = false;
49:                 b.Dispose();
50:             }
51:         }
52:
```



```
50:         void OnSized(object sender, EventArgs e)
51:     {
52:         dirty = true;
53:         Invalidate();
54:     }
55:
56:
57:     public ClipToPath()
58:     {
59:
60:         p=new GraphicsPath();
61:         p.AddString("AYBABTU",
62:             FontFamily.GenericSansSerif,
63:             0,(float)72,new Point(0,0),
64:             StringFormat.GenericDefault());
65:         dirty=true;
66:         this.Paint+=
67:             new PaintEventHandler(OnPaint);
68:         this.SizeChanged+=
69:             new EventHandler(OnSized);
70:         timer = new Timer();
71:         timer.Interval = 10;
72:         timer.Tick+=new EventHandler(OnTick);
73:         timer.Enabled=true;
74:     }
75:     static void Main()
76:     {
77:         Application.Run(new ClipToPath());
78:     }
}
```

Dường đồ thị được tạo một lần trên những dòng 59–62 và được giữ lại để sử dụng sau đó. Kế đến bộ định thời (timer) được khởi tạo để có thể vẽ lại theo một chu kỳ nào đó.

Những dòng 39–48 xử lý việc vẽ lại nền nếu chúng ta thay đổi kích thước (resize) window hoặc đó là lần vẽ đầu tiên.

Cuối cùng, dòng 25 chọn đường cắt (clip-path) cho đối tượng Graphics. Những dòng kế đó đặt những giọt nước màu trên cửa sổ form; bạn có thể thấy hoặc không thấy những giọt nước này phụ thuộc vào sự ngẫu nhiên của nó với đường cắt (clipping path).

Trong hình 3.5-8 là kết xuất của ví dụ 3.5-6.



Hình 3.5-8 *Bức tranh bị cắt bởi đường đồ thị*

Những thao tác trên vùng (region) khác với trên đường đồ thị. Một đường đồ thị chỉ ra một tập các đường biên, một vùng (region) chỉ ra một khu vực hay một nhóm các khu vực được sử dụng như một mặt nạ (mask). Các vùng có thể được kết hợp theo nhiều cách. Những thao tác này được quy định bởi những giá trị của `CombineMode`. Ví dụ 3.5-7 minh họa cách sử dụng vùng trong bốn cách khác nhau.

Ví dụ 3.5-7 Regions.cs: Kết hợp vùng theo nhiều cách khác

```

1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Collections;
5: using System.ComponentModel;
6: using System.Windows.Forms;
7: using System.Data;
8:
9: namespace regions
10: {
11:     /// <summary>
12:     /// Summary description for Form1.
13:     /// </summary>
14:     public class RegionsTest
15:         : System.Windows.Forms.Form
16:     {
17:         void PaintRegions(Graphics g,
18:             CombineMode m, Point offset, string text)
19:         {
20:             Region ra, rb;
21:             GraphicsPath p=new GraphicsPath();
22:             p.AddRectangle(
23:                 new Rectangle(60,60,120,120));
24:             ra=new Region(p);
25:             p=new GraphicsPath();
26:             p.AddEllipse(0,0,120,120);
27:             rb=new Region(p);
28:             ra.Translate(offset.X,offset.Y );
29:             rb.Translate(offset.X,offset.Y );
30:         }
31:     }
32: }
```

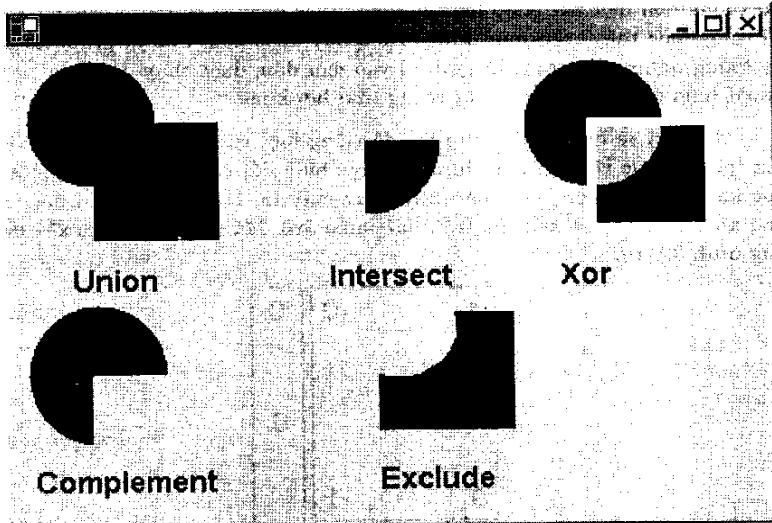


```
27:         g.SetClip(ra, CombineMode.Replace);
28:         g.SetClip(rb,m);
29:         SolidBrush brush=
30:             new SolidBrush(Color.Black);
31:         g.FillRectangle(brush,
32:             this.ClientRectangle);
33:         g.ResetClip();
34:         g.DrawString(text,
35:             new Font("Ariel", (float)16),
36:             brush, (float)offset.X+60,
37:             (float)offset.Y+220);
38:     }
39: 
40:     void OnPaint(object sender,
41:                 PaintEventArgs e)
42:     {
43:         // hợp của hai vùng...
44:         PaintRegions(e.Graphics,
45:                     CombineMode.Union,
46:                     new Point(0,0), "Union");
47:         // giao của hai vùng.
48:         PaintRegions(e.Graphics,
49:                     CombineMode.Intersect,
50:                     new Point(250,0), "Intersect");
51:         // vùng riêng biệt
52:         PaintRegions(e.Graphics, CombineMode.Xor,
53:                     new Point(500,0), "Xor");
54:         // Phần bù của hai vùng
55:         PaintRegions(e.Graphics,
56:                     CombineMode.Complement,
57:                     new Point(0,250), "Complement");
58:         // Hiệu.
59:         PaintRegions(e.Graphics,
60:                     CombineMode.Exclude,
61:                     new Point(250,250), "Exclude");
62:     }
63: 
64:     public RegionsTest()
65:     {
```

```
66: }
67: }
```

OnPaint gọi liên tục phương thức PaintRegions với các giá trị riêng biệt của CombineMode, thông tin về vị trí, và một chuỗi tên của kiểu kết hợp. Phương thức PaintRegions (dòng 16-35) bắt đầu bằng hai khai báo biến kiểu Region, `ra` và `rb`, và tô chúng bằng một đường tròn và một hình chữ nhật. Dòng 25 và 26 bao đảm rằng hai vùng này đã được di chuyển vào đúng vị trí cho việc hiển thị, kế tiếp dòng 27 chọn vùng cắt (clip region) thứ nhất, thay thế cái có sẵn trong đối tượng Graphics. Ở dòng 28, vùng thứ hai được kết hợp một tham số CombineMode, và toàn bộ khu vực client được tô bằng màu đen. Chú ý rằng chỉ khu vực bên trong vùng bị cắt là được tô. Kế đó ở dòng 31 chương trình di chuyển vùng cắt (clipping region) để vẽ văn bản ở những dòng 32-34.

Trong hình 3.5-9 là kết xuất của ví dụ



Hình 3.5-9 Khảo sát các kiểu kết hợp vùng

11. CÁC PHÉP BIẾN ĐỔI (TRANSFORMATIONS)

Có lẽ trong ví dụ trước, bạn đã nhận ra vài dòng lệnh áp dụng phép biến đổi để di chuyển các vùng (region) đến một vị trí khác. Có hai cách cơ bản trong việc áp dụng các phép biến đổi cho các đối tượng đồ họa trong kiến trúc (Framework). Bạn có thể sử dụng các phương thức sẵn có, chẳng hạn như Translate hay Rotate, hay bạn có thể định rõ một ma trận (matrix) của phép biến đổi sẽ được sử dụng.

Cách tốt nhất hơn hẳn mọi cách khác để xử lý các phép biến đổi này là sử dụng các phương thức mà nó bao gồm những thao tác ma trận ở phía dưới cho bạn.



Những phép biến hình này cho phép bạn làm các điều sau:

- Phép tịnh tiến (Translate) – Tịnh tiến một đối tượng.
- Phép quay (Rotate) – Xoay một đối tượng xung quanh một gốc tọa độ.
- Phép quay theo tọa độ (RotateAt) – Xoay một đối tượng xung quanh một điểm khác với gốc tọa độ.
- Phép co dãn (Scale) – Phóng to hay thu nhỏ đối tượng.
- Phép biến dạng (Shear) – Bóp méo một đối tượng bằng cách thay đổi hình chữ nhật bao của nó thành một hình bình hành bao.

Một ma trận biểu diễn các phép biến đổi này bằng cách thực hiện phép toán trên ma trận, chẳng hạn cộng, trừ, hay nhân với phép biến đổi đồ họa hiện hành. Chúng ta đã đề cập ngay ở đầu chương này rằng những lệnh đồ họa cần phải thông qua một vài phép biến đổi giữa việc được vẽ trong hệ thống đồ họa cuối cùng được thấy trên màn hình. Điều này được xem như là graphics pipeline. Hãy tưởng tượng những mệnh lệnh đi vào một đầu, được thay đổi theo đường ống và xuất hiện ở đầu kia trong một trạng thái hơi khác.

Ma trận sẽ biến đổi những tọa độ riêng biệt của đối tượng đồ họa. Mỗi m điểm (pixel) được vẽ trên màn hình sẽ được biến đổi trong đường ống một vài l trước khi nó xuất hiện trên màn hình hay máy in. Những ma trận được sử dụng trong kiến trúc này là một ma trận hai chiều 3×3 . Ma trận này được xây dựng như trong hình 3.5-10.

1	0	0
0	1	0
0	0	1

Linear part
Translation part
Always 0,0,1

Hình 3.5-10 Ma trận đơn vị 3×3

Ma trận mà bạn thấy trong hình được gọi là ma trận đơn vị (identity matrix). Một ma trận đơn vị có thể được áp dụng như một phép biến đổi lên n đối tượng và nó không bị ảnh hưởng. Ma trận đơn vị thường là điểm bắt đầu của tất cả các phép biến đổi. Ma trận được sử dụng trong GDI+ có một vài giá trị trong cột phải thứ nhất là cố định. Phần mà luôn luôn là 0,0,1 trong ma trận là

thì được dùng để cho tiếp theo phép biến đổi phức hợp (chẳng hạn như tính toán tuyến tính, quay, co dãn) có phép tịnh tiến trong cùng phép nhân. Điều này được biết như là một phép tính của ma trận affine. Để thực hiện một phép tính affine trên ma trận n chiều, ma trận nhân phải là $(n+1) \times (n+1)$. Vì vậy một phép toán hai chiều cần một ma trận 3×3 . Vì lý do này, cột thứ ba của ma trận luôn luôn được gán là 0,0,1, và bạn không thể thay đổi nó.

Có nhiều sách về đồ họa nhưng, vì mục đích của tính toán vẹn, các phép biến đổi được thực hiện bởi ma trận trên hệ tọa độ hoạt động như sau. Ví dụ sau thực hiện một phép tịnh tiến đơn giản với một giá trị X và Y.

Một tọa độ được biến thành một vector bằng cách thêm vào thành phần thứ ba, đó là cột z. Ví dụ,

[10,5] trở thành [10,5,1]

Giá trị tịnh tiến dx, dy của ma trận là 10,30

Ma trận sẽ nhân với vector này theo cách sau:

[10, 5, 1]

nhân...

1 0 0

0 1 0

dX dY 1

bằng...

1*x 0*x 0*x

+ + +

0*y 1*y 0*y

+ + +

1*dX 1*dY 1*1

bằng...

dx + x dy + y 1

bằng...

20 35 1

Lấy thành phần thứ ba ra, và bạn còn lại [20,35]. Đó là kết quả mà [10,5] được tịnh tiến bởi [10,30].

Phép quay quanh gốc tọa độ cũng được thực hiện theo một cách tương tự. Chúng ta sẽ sử dụng 90° , bởi vì sin và cos 90° dễ tính. Ma trận cho phép quay được khởi tạo như sau:



$\cos\theta \sin\theta 0$

$-\sin\theta \cos\theta 0$

$0 0 1$

Do $\cos(90) = 0$ và $\sin(90) = 1$, nên phép tính trên ma trận giống như sau:

[10, 5 1]

nhân...

$0 1 0$

$-1 0 0$

$0 0 1$

bằng...

$0*x 1*x 0*x$

+ + +

$-1*y 0*y 0*y$

+ + +

$1*0 1*0 1*1$

bằng...

[-5, 10, 1]

Bỏ phần mở rộng, bạn được [-5, 10]

Lấy hai ma trận cộng hay nhân chúng với nhau tạo ra một ma trận kết qí Cộng hay nhân liên tiếp tạo ra một ma trận mà nó là tổng hợp của tất cả các ph biến đổi được thực hiện. Điều này muốn nói rằng bạn có thể thực hiện vài ph biến đổi trên ma trận và tất cả chúng tích lũy lại, và sau đó phép biến đổi tro ma trận được áp dụng cho mỗi điểm (pixel) mà lệnh vẽ tạo ra để có được những trí của chúng trong kết xuất cuối cùng.

Thứ tự các phép biến đổi cũng rất quan trọng. Ví dụ, Phép quay – Phép dãn – Phép tịnh tiến không cùng một ý nghĩa với Phép co dãn – Phép tịnh tiến Phép quay. Điều này ngũ ý rằng bạn cũng cần phải suy nghĩ về những lệnh API áp dụng vào những ma trận mà bạn trao cho chúng như thế nào. Bạn có nh ma trận hiện hành với một ma trận mà bạn vừa mới đưa ra hay một ma trận bạn có? Thật may mắn, những lời gọi Rotate, Scale, và vân vân, có một cờ (fl mà bạn có thể sử dụng để điều khiển cách mà những ma trận làm việc. Mặc dù MatrixOrder.Prepend áp dụng vào ma trận mà bạn truyền vào đầu tiên, và sau là ma trận hiện hành. MatrixOrder.Append áp dụng vào phép biến đổi được cầu sau khi ma trận hiện hành được áp dụng.

Đoạn mã sau minh họa nội dung của một ma trận khi nó tiến triển nhiều phép biến đổi.

Matrix m = new Matrix() // Tạo một ma trận đơn vị

$1 0 0$

$0 1 0$

```

0 0 1
m.Rotate(30,MatrixOrder.Append); // quay 30°
0.8660254 0.5 0
-0.5 0.8660254 0
0 0 1
// Kéo dài 3 lần theo trục Y
m.Scale((float)1,(float)3,MatrixOrder.Append);
0.8660254 1.5 0
-0.5 2.598076 0
0 0 1
// di chuyển 100 theo trục X và 130 theo trục Y
m.Translate((float)100,(float)130,MatrixOrder.Append);
0.8660254 1.5 0
-0.5 2.598076 0
100 130 1

```

Với sự biến đổi liên tiếp này, bạn có thể thấy ma trận tích lũy những phép biến đổi với mỗi lời gọi kế tiếp như thế nào.

Chú ý rằng việc theo dõi ma trận đồ họa là rất quan trọng, đặc biệt nếu bạn muốn đặt nhiều đối tượng khác nhau lên màn hình, mỗi đối tượng có phép biến đổi của chính nó. Đôi khi dùng ma trận hiện hành vào một trạng thái đã biết thì thật là hữu ích, có thể với sự bắt đầu trong một nơi khác hay phóng to hay thu nhỏ bằng phép co dãn, và sau đó thực hiện các phép biến đổi khác. Mỗi phép biến đổi tự hành xử và để lại ma trận hiện hành như bạn đã thấy. Việc lưu trạng thái của đối tượng Graphics trong đối tượng GraphicsState có thể làm điều này. Đối tượng GraphicsState được ghi thông tin vào bởi phương thức Graphics.Save() và được phục hồi bằng phương thức Graphics.Restore(state). Ví dụ như:

```

GraphicsState gs = theGraphics.Save();
//thực hiện thao tác ở đây
theGraphics.Restore(gs);
//Đồ họa trở về trạng thái gốc của chúng

```

Ví dụ 3.5-8 minh họa một dây các phép biến đổi trước đây, cũng như sử dụng GraphicsState và những thao tác ma trận khác trong ngữ cảnh của một vài hình đồ họa đơn giản.

Ví dụ 3.5-8 MatrixElements.cs

```

1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Drawing.Text;

```

```
5:using System.Collections;
6:using System.ComponentModel;
7:using System.Windows.Forms;
8:using System.Data;
9:using System.Text;
10:
11: namespace matrixelements
12: {
13:     public class MatrixElements
14:         : System.Windows.Forms.Form
15:     {
16:         void DumpMatrix(Graphics g, Matrix m,
17:                           Point p)
18:         {
19:             StringBuilder sb=new StringBuilder();
20:             sb.AppendFormat("{0},{1},0\n{2},
21:                             {3},0\n{4},{5},1",
22:                             m.Elements[0],
23:                             m.Elements[1],
24:                             m.Elements[2],
25:                             m.Elements[3],
26:                             m.Elements[4],
27:                             m.Elements[5]);
28:             GraphicsState s=g.Save();
29:             g.ResetTransform();
30:             g.DrawString(sb.ToString(),
31:                         new Font("Ariel",(float)16),
32:                         new SolidBrush(Color.Black),p,
33:                         StringFormat.GenericDefault);
34:             g.Restore(s);
35:         }
36:     }
37:
38:     void OnSize(object sender,EventArgs e)
39:     {
40:         Invalidate();
41:     }
42:
43:     void OnPaint(object sender,
```

```
        PaintEventArgs e)
34:    {
35:        GraphicsState gs;
36:        Matrix m=new Matrix();
37:
38:        //xác định vị trí và vẽ các trục bằng cách
//tịnh tiến cửa sổ để mà gốc tọa độ ở giữa
39:        //màn hình.
40:        e.Graphics.TranslateTransform(
41:            (float)this.ClientRectangle.Width/2,
42:            (float)this.ClientRectangle.Height/2);
43:        e.Graphics.DrawLine(
44:            new Pen(Color.Black,(float)1),
45:            0,-1000,0,1000);
46:        e.Graphics.DrawLine(
47:            new Pen(Color.Black,(float)1),
48:            -100,0,1000,0);
49:
50:        //Vẽ một hình vuông.
51:        e.Graphics.DrawRectangle(
52:            new Pen(Color.Black,(float)3),
53:            -50,-50,100,100);
54:        DumpMatrix(e.Graphics,m,new Point(0,0));
55:
56:        m.Rotate(30,MatrixOrder.Append);
57:        DumpMatrix(e.Graphics,m,
58:            new Point(0,100));
59:
60:        gs=e.Graphics.Save();
61:        e.Graphics.MultiplyTransform(m);
62:        e.Graphics.DrawRectangle(
63:            new Pen(Color.Red,3),
64:            -50,-50,100,100);
65:        e.Graphics.Restore(gs);
66:
67:        m.Scale(1,3,MatrixOrder.Append);
68:        DumpMatrix(e.Graphics,m,
69:            new Point(0,200));
```

```
61:
62:     gs=e.Graphics.Save();
63:     e.Graphics.MultiplyTransform(m);
64:     e.Graphics.DrawRectangle(
65:         new Pen(Color.Green,3),
66:         -50,-50,100,100);
67:     e.Graphics.Restore(gs);
68:     m.Translate(100,130,MatrixOrder.Append);
69:     DumpMatrix(e.Graphics,m,
70:                 new Point(0,300));
71:     gs=e.Graphics.Save();
72:     e.Graphics.MultiplyTransform(m);
73:     e.Graphics.DrawRectangle(
74:         new Pen(Color.Blue,3),
75:         -50,-50,100,100);
76:     e.Graphics.Restore(gs);
77:
78:     public MatrixElements()
79:     {
80:         this.Paint+=
81:             new PaintEventHandler(OnPaint);
82:         this.SizeChanged+=
83:             new EventHandler(OnSize);
84:     }
85:
86:     static void Main()
87:     {
88:         Application.Run(new MatrixElements());
89:     }
```

Dòng 40 và 41 tạo một ma trận thay đổi gốc tọa độ thành tâm của cù
Dòng 42 và 43 vẽ các trục và đưa ra những con số để tham khảo.

Dòng 46 và 47 vẽ hình vuông. Điều này giống như lệnh vẽ được sử dụng cho tất cả những đối tượng khác trên màn hình, và sau đó dòng 48 đưa ra ngoài ma trận đơn vị.

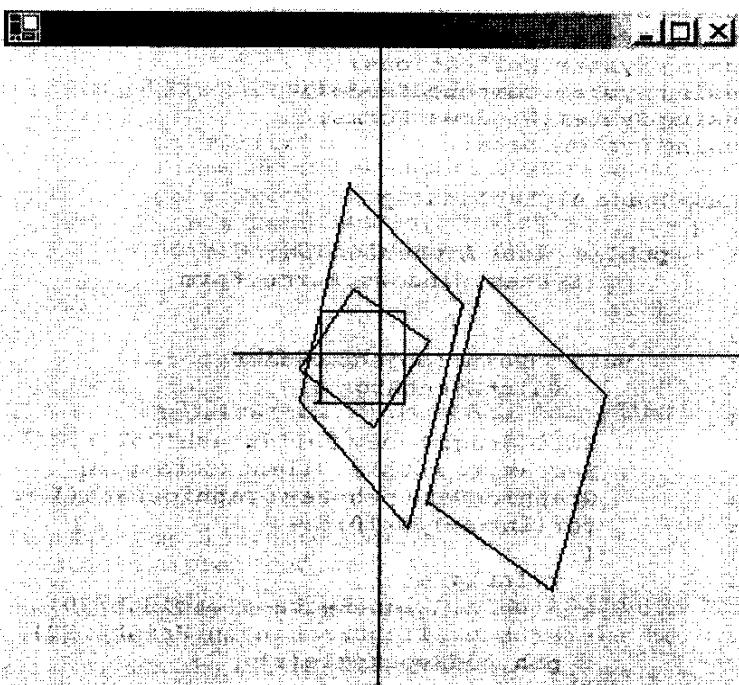
Dòng 50 quay ma trận đó, nó lại được đưa ra ngoài với văn bản, và sau đó dòng 53 lưu trạng thái đồ họa để cho không làm hư gốc tọa độ. Dòng 54 dùng ma trận, dòng 55 và 56 vẽ hình vuông mới được quay, và dòng 57 phục hồi lại phép biến đổi của thế giới thực cho trạng thái gốc tọa độ nằm ở giữa.

Dòng 59 phóng to ma trận, làm cho bề cao hơn ba lần bề rộng, và kéo dài kết xuất theo trục Y. Dòng 62-66 dùng phép biến đổi, vẽ hình vuông, và trả về phép biến đổi thế giới thực được chọn mặc định.

Kế đó dòng 68 tịnh tiến ma trận đi qua phải và xuống dưới. Dòng 69 đưa ra những cài đặt của ma trận và trên các dòng 71-75 chúng ta làm tròn việc vẽ kéo dài, quay, phóng to, và tịnh tiến hình vuông.

Để ma trận dữ liệu tự vẽ, các dòng 15-26 lưu trữ trạng thái đồ họa và đặt lại phép biến đổi, đưa văn bản lên màn hình, và sau đó trả lại phép biến đổi gốc của nó trước khi trở về.

Form kết xuất cuối cùng được đưa ra trong hình 3.5-11



Hình 3.5-11 Các phép biến đổi ma trận cho hình vuông đơn giản



12. SỰ PHỐI MÀU VỚI GIÁ TRỊ ALPHA (ALPHA BLENDING)

Tất cả các màu trong GDI+ có một thành phần thứ tư được thêm vào các giá trị màu thông thường red, green và blue. Đó là giá trị "alpha" và nó điều chỉnh giá trị thực mà màu nền (background) thể hiện qua đối tượng vừa được thiết lập trên màn hình.

Alpha blending có thể được áp dụng cho tất cả các hình dạng đồ họa, như đường thẳng và văn bản. Nó cũng có thể được ứng dụng cho hình ảnh, hoặc là một giá trị toàn cục ánh hưởng đến diện mạo của toàn bộ ánh, hoặc là một giá trị cho mỗi pixel riêng lẻ, nó tạo ra phần hình ánh trong suốt hơn những cái khác.

Để tạo một màu với một thành phần alpha, bạn có thể dùng phương thức `Color.FromArgb(...)` và cung cấp alpha là một giá trị từ 0, hoàn toàn trong suốt (transparency) tới 255 hoàn toàn mờ đục (opaque).

Ví dụ 3.5-9 minh họa tác động này bằng cách tạo một dãy hình ellipse mà sự thể hiện màu nền (background) hoàn toàn tuỳ theo giá trị alpha.

Ví dụ 3.5-9 AlphaBlend.cs: Sử dụng cọ vẽ với độ trong suốt Alpha

```
1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Collections;
5: using System.ComponentModel;
6: using System.Windows.Forms;
7: using System.Data;
8:
9: namespace Alphablending
10: {
11:     public class Alphablending
12:         :System.Windows.Forms.Form
13:     {
14:         void OnPaint(object sender,
15:                     PaintEventArgs e)
16:         {
17:             SolidBrush b=new SolidBrush(Color.Red);
18:             Rectangle r=this.ClientRectangle;
19:             GraphicsPath pth=new GraphicsPath();
20:             for(int c=1;c<10;c++)
21:             {
22:                 r.Inflate(
23:                     -(this.ClientRectangle.Width/20),
24:                     -(this.ClientRectangle.Height/20));
25:                 pth.AddRectangle(r);
e.Graphics.FillPath(b, pth);
```

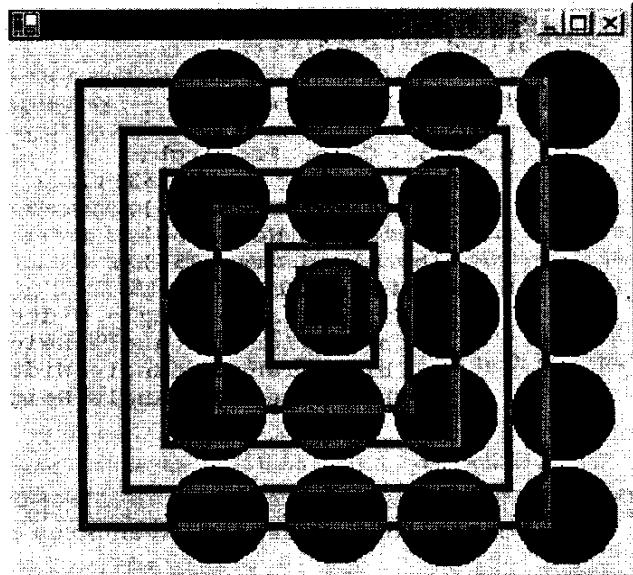
```

26:         Random rnd=new Random();
27:         for(int y=0;y<5;y++)
28:         {
29:             for(int x=0;x<5;x++)
30:             {
31:                 b.Color=Color.FromArgb(
32:                     (int)((((5*x)+y)*10.63)),
33:                     (byte)rnd.Next(255),
34:                     (byte)rnd.Next(255),
35:                     (byte)rnd.Next(255));
36:                 e.Graphics.FillEllipse(b,
37:                     this.ClientRectangle.Width/5*x,
38:                     this.ClientRectangle.Height/5*y,
39:                     this.ClientRectangle.Width/5,
40:                     this.ClientRectangle.Height/5);
41:             }
42:         }
43:     void OnSize(object sender, EventArgs e)
44:     {
45:         Invalidate();
46:     }
47:
48:     public Alphablending()
49:     {
50:         this.Paint+=
51:             new PaintEventHandler(OnPaint);
52:         this.SizeChanged+=
53:             new EventHandler(OnSize);
54:         this.BackColor=Color.White;
55:     }
56:     static void Main()
57:     {
58:         Application.Run(new Alphablending());
59:     }
60: }
61: }
```

Biên dịch ví dụ 3.5-9 bằng dòng lệnh sau:

csc /t:winexe alphablend.cs

Kết xuất của ví dụ 3.5-9 được trình bày ở hình 3.5-12 và thể hiện phông nền một cách rõ ràng, những hình chữ nhật đồng tâm màu sẫm, được làm mờ dần bởi alpha blends được tăng dần của những hình ellipses.



Hình 3.5-12 Phép phối màu với giá trị alpha đang có tác dụng

13. SỰ PHỐI MÀU VỚI GIÁ TRỊ ALPHA CHO HÌNH ẢNH

Công việc này cũng dễ dàng thực hiện thông qua việc sử dụng đối tượng ColorMatrix. Chúng ta đã xem ở phần thao tác ma trận mà ma trận có thể dùng để thực hiện thao tác trong không gian hai chiều cho những đối tượng đồ họa sử dụng ma trận 3×3 . Một ColorMatrix liên quan đến khái niệm bốn chiều, không gian màu, các chiều của không gian màu là R,G,B và A tương ứng với Red, Green, Blue và Alpha. Thực hiện những thao tác ma trận tùy ý trên đối tượng bốn chiều yêu cầu một ma trận 5×5 . Giống ví dụ ma trận 3×3 cho không gian ba chiều (3 ColorMatrix duy trì một cột giá trị là bộ $0,0,0,1$).

Ảnh ở hình 3.5-13 là ma trận đơn vị cho đối tượng ColorMatrix

1 0 0 0	0
0 1 0 0	0
0 0 1 0	0
0 0 0 1	0
0 0 0 0	1

Linear part Translation part
Always 0,0,0,1

Hình 3.5-13 Ma trận đơn vị của không gian màu

Lớp ColorMatrix, giống ma trận dành cho thao tác 2D, có các thuộc tính cho những phần tử của nó có thể được truy cập riêng biệt. Để thiết lập alpha cho một hình ảnh, bạn chỉ cần đặt giá trị alpha vào thuộc tính Matrix33 như sau:

```
//Tạo một identity matrix
ColorMatrix cm=new ColorMatrix ();
m.Matrix33=(float)AlphaValue;
```

Ma trận màu này sau đó được chuyển đến lớp ImageAttributes:
`ImageAttributes ia = new ImageAttributes();
iaSetColorMatrix(m);`

Đối tượng ImageAttributes được khởi tạo sau đó được sử dụng bởi phương thức Graphics.DrawImage() để tô hình với một alpha blend cụ thể.

Ví dụ 3.5-10 trình bày cách dùng những đặc tính alpha blend của GDI+ với bất kỳ ảnh bitmap nào từ máy của bạn. Chương trình có một nút nhấn cho phép bạn chọn lựa một file ảnh và một thanh trượt (track bar) cho phép bạn chọn độ trong suốt với hình ảnh được hiển thị.

Ví dụ 3.5-10 ImageAlpha.cs: Alpha Blending cho ảnh

```
1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Drawing.Imaging;
5: using System.Collections;
6: using System.ComponentModel;
7: using System.Windows.Forms;
8: using System.Data;
9:
10: namespace ImageAlpha
11: {
12:     class Form1 : Form
13:     {
14:
15:         Button b;
16:         TrackBar t;
17:         Image i;
18:
19:         void OnPaint(object Sender, PaintEventArgs e)
20:         {
21:             SolidBrush b=new SolidBrush(Color.Red);
22:             Rectangle r=this.ClientRectangle;
```

```
23:         GraphicsPath pth=new GraphicsPath();
24:         for(int c=1;c<10;c++)
25:         {
26:             r.Inflate(
27:                 -(this.ClientRectangle.Width/20),
28:                 -(this.ClientRectangle.Height/20));
29:             pth.AddRectangle(r);
30:         }
31:         e.Graphics.FillPath(b,pth);
32:         if(i!=null)
33:         {
34:             ColorMatrix m=new ColorMatrix();
35:             m.Matrix33=(float)(1.0/256*t.Value);
36:             ImageAttributes ia=
37:                 new ImageAttributes();
38:             iaSetColorMatrix(m);
39:             e.Graphics.DrawImage(i,
40:                 this.ClientRectangle,
41:                 0,0,i.Width,i.Height,
42:                 GraphicsUnit.Pixel,ia);
43:         }
44:     }
45:     void OnClickB(object sender, EventArgs e)
46:     {
47:         OpenFileDialog dlg=new OpenFileDialog();
48:         dlg.Filter="Bitmap files (*.bmp)|*.bmp";
49:         if(dlg.ShowDialog()==DialogResult.OK)
50:         {
51:             i=Image.FromFile(dlg.FileName);
52:             Invalidate();
53:         }
54:     }
55:     void OnTrack(object sender, EventArgs e)
56:     {
57:         Invalidate();
58:     }
```

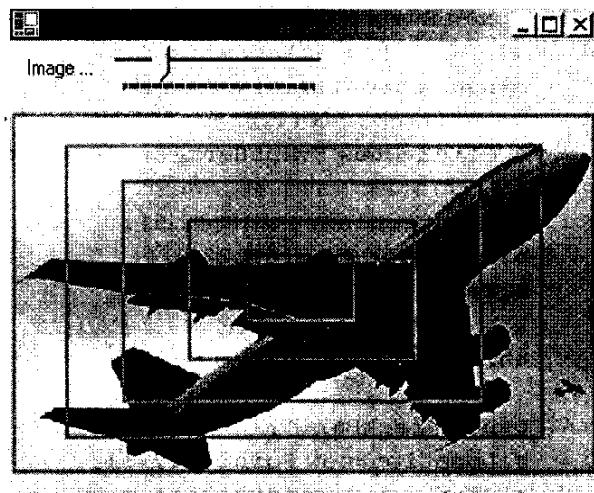
```
56:     }
57:
58:     void OnSize(object sender, EventArgs e)
59:     {
60:         Invalidate();
61:     }
62:
63:     public Form1()
64:     {
65:         this.Paint+=
66:             new PaintEventHandler(OnPaint);
67:         this.SizeChanged+=
68:             new EventHandler(OnSize);
69:
70:         b=new Button();
71:
72:         b.Location=new Point(5,5);
73:         b.Size=new Size(60,22);
74:         b.Text="Image...";
75:
76:         this.Controls.Add(b);
77:
78:         t=new TrackBar();
79:         t.Location=new Point(100,5);
80:         t.Size=new Size(200,22);
81:         t.Maximum=255;
82:         t.Minimum=0;
83:         t.ValueChanged+=
84:             new EventHandler(OnTrack);
85:
86:         this.Controls.Add(t);
87:
88:         static void Main()
89:         {
90:             Application.Run(new Form1());
```

```
91:      }
92:      }
93:  }
```

Biên dịch file này bằng dòng lệnh sau:
csc /t:winexe imagealpha.cs

Những dòng quan trọng là 34-38. Dòng 34 tạo một ma trận đơn vị. Dòng 36 dùng giá trị ở thanh trượt (track bar) để cập nhật thành phần alpha của ảnh trong ma trận. Dòng 36 tạo lớp ImageAttributes mà nó sử dụng ColorMatrix ở dòng 5. Sau cùng dòng 38 vẽ ảnh lên màn hình, cho phép ảnh nền của chúng ta ở giây thấy xuyên qua hay không, tùy thuộc vào giá trị của thanh trượt (track bar).

Hình 3.5-14 thể hiện kết quả này với thanh trượt (track bar) được gán khoảng 70%. Bạn sẽ ngạc nhiên, Đó là một chiếc Triumph Bonneville 750.



Hình 3.5-14 Phối màu với giá trị alpha cho ảnh bitmap

14. NHỮNG THAO TÁC XỬ LÝ KHÁC TRONG KHÔNG GIAN MÀU

Có một ma trận toàn vẹn riêng biệt cho alpha blending đường như là cursive, cho đến khi bạn nhận thấy rằng thao tác nào đó trong không gian màu là thể thực hiện được với một công cụ. Kỹ thuật thao tác một ma trận nhằm nêu đích thay đổi không gian màu được gọi là *recoloring* (tái tạo màu) và cũng dễ dàng thực hiện với GDI+. Một lần nữa, công cụ được dùng là ColorMatrix. Đáng tiếc không có những phương thức trong ColorMatrix để thực hiện việc luân phiên vì màu, nhưng chúng có thể đã đưa ra giá trị đúng trong phần tuyến tính của ma trận. Dưới đây là ví dụ:

**Ví dụ 3.5-11 ColorSpace1.cs: Những thao tác khác trong không gian màu**

```
1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Drawing.Imaging;
5: using System.Collections;
6: using System.ComponentModel;
7: using System.Windows.Forms;
8: using System.Data;
9:
10: namespace ImageAlpha
11: {
12:     class Form1 : Form
13:     {
14:
15:         Button b;
16:         TrackBar tr,tg,tb;
17:         Image i;
18:
19:         void OnPaint(object Sender,PaintEventArgs e)
20:         {
21:             SolidBrush b=new SolidBrush(Color.Red);
22:             Rectangle r=this.ClientRectangle;
23:             GraphicsPath pth=new GraphicsPath();
24:             if(i!=null)
25:             {
26:                 ColorMatrix m=new ColorMatrix();
27:                 m.Matrix00=(float)(1.0/256*tr.Value);
28:                 m.Matrix11=(float)(1.0/256*tg.Value);
29:                 m.Matrix22=(float)(1.0/256*tb.Value);
30:                 ImageAttributes ia=
31:                     new ImageAttributes();
32:                 ia.SetColorMatrix(m);
33:                 e.Graphics.DrawImage(i,
this.ClientRectangle,
0,0,i.Width,i.Height,
GraphicsUnit.Pixel,ia);
```

```
34:         }
35:     }
36:
37:     void OnClickB(object sender, EventArgs e)
38:     {
39:         OpenFileDialog dlg=new OpenFileDialog();
40:         dlg.Filter="Bitmap files(*.bmp)|*.bmp";
41:         if(dlg.ShowDialog()==DialogResult.OK)
42:         {
43:             i=Image.FromFile(dlg.FileName);
44:             Invalidate();
45:         }
46:     }
47:
48:     void OnTrack(object sender, EventArgs e)
49:     {
50:         Invalidate();
51:     }
52:
53:     void OnSize(object sender, EventArgs e)
54:     {
55:         Invalidate();
56:     }
57:
58:     public Form1()
59:     {
60:         this.Paint+=
61:             new PaintEventHandler(OnPaint);
62:         this.SizeChanged+=
63:             new EventHandler(OnSize);
64:
65:         b=new Button();
66:
67:         b.Click+=new EventHandler(OnClickB);
68:
69:         b.Location=new Point(5,5);
70:         b.Size=new Size(60,22);
71:         b.Text="Image...";
```

```
70:  
71:        this.Controls.Add(b);  
72:  
73:        tr=new TrackBar();  
74:        tr.Location=new Point(100,5);  
75:        tr.Size=new Size(200,22);  
76:        tr.Maximum=255;  
77:        tr.Minimum=0;  
78:        tr.ValueChanged+=  
             new EventHandler(OnTrack);  
79:  
80:  
81:        this.Controls.Add(tr);  
82:  
83:        tg=new TrackBar();  
84:        tg.Location=new Point(100,55);  
85:        tg.Size=new Size(200,22);  
86:        tg.Maximum=255;  
87:        tg.Minimum=0;  
88:        tg.ValueChanged+=  
             new EventHandler(OnTrack);  
89:  
90:  
91:        this.Controls.Add(tg);  
92:  
93:        tb=new TrackBar();  
94:        tb.Location=new Point(100,105);  
95:        tb.Size=new Size(200,22);  
96:        tb.Maximum=255;  
97:        tb.Minimum=0;  
98:        tb.ValueChanged+=  
             new EventHandler(OnTrack);  
99:  
100:  
101:       this.Controls.Add(tb);  
102:    }  
103:  
104:   static void Main()
```



```
105:      {  
106:          Application.Run(new Form1());  
107:      }  
108:  }  
109: }
```

Biên dịch file này bằng dòng lệnh sau:

```
csc /t:winexe colorspace1.cs
```

File này về cơ bản giống với ví dụ 3.5-10 trừ thanh trượt (track bars) cho red, green, và blue đã sử dụng thay cho alpha. Ma trận được thiết lập ở dòng 27-29 để điều chỉnh cường độ mỗi kênh màu R, G và B riêng biệt.

15. KẾT CHƯƠNG

Có nhiều vấn đề trong GDI+ cần khảo sát mà chương này có thể lên đến hàng trăm trang nữa, chúng tôi nghĩ những gì ở đây sẽ đem lại cho bạn sự tin cậy để thử nghiệm sâu hơn và sẽ cho bạn hiểu hầu hết những quy ước được yêu cầu chuyển tiếp từ GDI cũ mà bạn biết và yêu thích có khả năng tồn tại ở GDI+. Chúng ta sẽ học tiếp để khám phá nhiều hơn về ứng dụng Windows Forms trong chương sau “Những ứng dụng cụ thể trên Windows Forms”

Chương 3.6

THỰC HÀNH ỨNG DỤNG WINDOWS FORMS

Các vấn đề chính sẽ được đề cập đến:

- ✓ Sử dụng thuộc tính và đặc tính Attribute
- ✓ Giải thích ứng dụng FormPaint.exe

Phần này của giáo trình chúng ta sẽ học qua các nền tảng, từ những yếu tố cơ bản của Windows Forms tới việc sử dụng GDI+ trong những ứng dụng của bạn. Trong chương cuối, chúng ta sẽ khảo sát tí mỉ khía cạnh tạo những ứng dụng với Windows Forms. Phần quan trọng của kiến trúc .NET là khả năng của mỗi đối tượng có thể mô tả chính nó và cho bạn truy suất những thuộc tính của nó. Khả năng này có thể được sử dụng trong chính những chương trình của bạn để giúp đỡ người sử dụng và cung cấp một giao diện rõ ràng nhất quán.

Trong những năm gần đây, các hệ điều hành Windows đã gia tăng việc sử dụng thuộc tính (property). Theo cách thông thường thì bạn có thể nhấp chuột phải trên mục chọn và chọn những thuộc tính từ menu ngữ cảnh. Thông thường người lập trình bố trí lại thuộc tính hệ thống cho mọi ứng dụng thông qua hộp thoại hay cửa sổ. Giờ đây, .NET cung cấp một phương pháp cơ bản cho việc tạo và sử dụng những thuộc tính giao tiếp được chuẩn hóa.

1. SỬ DỤNG THUỘC TÍNH

Lớp đơn giản được hiển thị trong ví dụ 3.6-1

Ví dụ 3.6-1 Lớp SimpleObject

```
1: Public class SimpleObject
2: {
3:     private int _int;
4:     private string _string;
5:     private Color _color;
6:
7:     public SimpleObject()
8:     {
9:         //
10:        // Khởi động biến ở đây.
11:        //
12:    }
13:
```



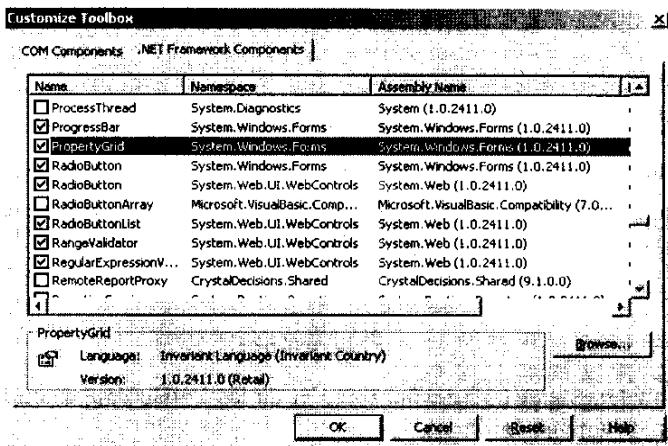
```
14: public int TheInteger
15: {
16:     get
17:     {
18:         return _int;
19:     }
20:     set
21:     {
22:         _int = value;
23:     }
24: }
25:
26: public string TheString
27: {
28:     get
29:     {
30:         return _string;
31:     }
32:     set
33:     {
34:         _string = value;
35:     }
36: }
37:
38: public Color TheColor
39: {
40:     get
41:     {
42:         return _color;
43:     }
44:
45:     set
46:     {
47:         _color = value;
48:     }
49: }
```

Bạn có thể thấy rằng lớp trên có 3 thành viên dữ liệu cục bộ - _int, _string, _color. Có 3 phương thức toàn cục truy xuất những thuộc tính – TheInteger, TheString, và TheColor. Đây là một kế hoạch thiết kế tốt bởi vì nó cô lập việc đóng gói dữ liệu và nó đưa ra một giao tiếp rõ ràng và không mơ hồ.

Việc sử dụng đối tượng (object) của chúng ta trong ứng dụng thì khá đơn giản. Sau đây là những bước tuân tự tạo ứng dụng C# thông qua Visual Studio .NET.

Chú ý: Một trong những thành phần giao diện (component) hữu dụng nhất trong hộp công cụ (toolbox) là đối tượng khung lưới thuộc tính (grid) – Mặc định thì thuộc tính này không hiển thị, bạn phải thêm nó vào bằng tay. Để cập nhật hộp công cụ, nhấp phải chuột trên nó và chọn Customize Toolbox, bạn sẽ thấy hộp thoại hiển thị như hình 3.6-1.

Dưới .NET Framework Components tab, chọn PropertyGrid

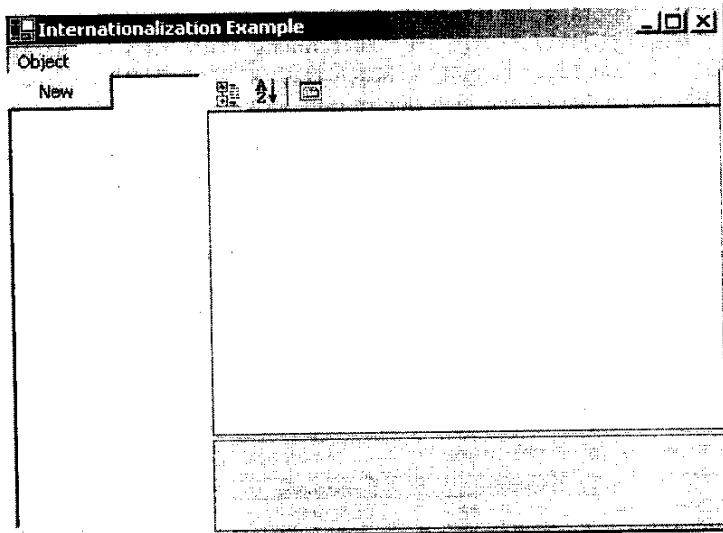


Hình 3.6-1 Hộp công cụ Customize (Customize Toolbox)

Để đưa một thành phần điều khiển vào ứng dụng, ta phải làm những bước sau :

- Kéo ListBox từ hộp công cụ thả lên trên form. Đặt thuộc tính Dock của nó thành Left.
- Kéo Splitter từ hộp công cụ thả lên trên form, nó sẽ tự động gắn vào cạnh bên phải nhất của ListBox.
- Kéo điều khiển (control) PropertyGrid vào form và đặt nó vào bên phải của thanh Splitter. Đặt thuộc tính Dock của nó thành Fill.
- Kéo MainMenu lên trên form và đặt nó trên thanh công cụ (toolbar), đặt tên nó là Object và đặt tên cho mục menu đầu tiên là New.

Tại thời điểm này, bạn nên có một form giống như hình 3.6-2



Hình 3.6-2 Form Object mới vừa tạo với *ListBox*, *Splitter*, và *PropertyGrid*

Chọn dự án trong cửa sổ Solution Explorer. Nhấp chuột phải trên nó và chọn Add New Item. Chọn C# class – được chỉ ra bởi biểu tượng sau :



Class

Tạo ra lớp SimpleObject, như trong ví dụ 3.6-1. Bảo đảm sao lại một cách chính xác nội dung mã nguồn.

Bây giờ, thêm vào một bộ xử lý (handler) cho Object, mục menu New, nhấp chuột 2 lần trên mục menu New trong form thiết kế. Visual Studio .NET sẽ đưa vào bộ xử lý (handler) cho bạn. Điện mã vào bộ xử lý như được hiển thị trong ví dụ 3.6-2

Ví dụ 3.6-2 Bộ xử lý Object Menu

```
1: private void menuItem2_Click(object sender,
                               System.EventArgs e)
2: {
3:     SimpleObject o = new SimpleObject();
4:     this.listBox1.Items.Add(o);
5: }
```

Cuối cùng, tạo ra bộ xử lý cho ListBox.

Chọn ListBox trong form thiết kế, sau đó chọn biểu tượng sự kiện (Event) trong Property Browser, và nhấp chuột 2 lần trên mục selectedIndexChanged.

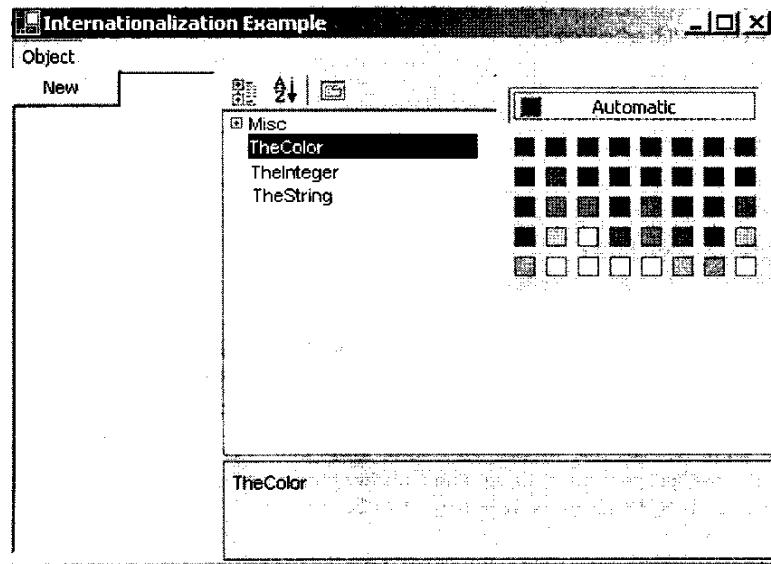
Một lần nữa bạn sẽ thấy Visual Studio .NET sẽ thêm vào bộ xử lý cho bạn. Đưa mã vào bộ xử lý như ví dụ 3.6-3.

Ví dụ 3.6-3 Bộ xử lý SelectedIndexChanged

```
1: private void listBox1_SelectedIndexChanged(
          object sender, System.EventArgs e)
2: {
3:     this.propertyGrid1.SelectedObject =
        this.listBox1.SelectedItem;
4: }
```

Bây giờ, chúng ta sẵn sàng chạy chương trình. Nhấn phím F5 và đợi cho việc biên dịch hoàn tất.

Bạn có thể thêm nhiều SimpleObject vào Listbox bằng việc sử dụng mục menu. Sau khi thêm vào, chọn mục menu trong listbox sẽ hiển thị nhiều thuộc tính trong điều khiển PropertyGrid. Chú ý PropertyGrid sẽ giúp bạn trong việc soạn thảo những thuộc tính như thế nào. Thuộc tính màu được chúng ta đặc biệt quan tâm đến, bởi vì nó sẽ cho phép bạn chọn từ bảng màu, vào một tên màu đã biết (ví dụ, màu đỏ (Red)) hoặc đánh vào những giá trị riêng biệt cho màu đỏ (90), màu xanh lá cây(180), màu xanh(240) như hình 3.6-3



Hình 3.6-3 Ứng dụng Property Browser.

Đối với 1 số lượng công việc nhỏ như vậy, quả thực đây là một ứng dụng mạnh.



2. NÂNG CẤP THUỘC TÍNH EXPERIENCE

Nếu bạn nhìn cẩn thận trong hình 3.6-3 Bạn sẽ nhìn thấy 2 điều lưu ý được cung cấp bởi Framework và PropertyGrid. Đầu tiên, hộp feedback tại đây của PropertyGrid có văn bản trong nó. Đây là thông tin phản hồi giúp đỡ người sử dụng hiểu việc họ đang làm. Thứ hai, danh sách của những thuộc tính trong khung được nhóm vào một phân loại gọi là Misc. Nếu bạn nhìn vào VS.NET hoặc những ứng dụng Windows Forms khác, bạn sẽ thấy những loại khác như Behavior hoặc Appearance.

Bạn có thể sắp xếp các thuộc tính của bạn theo phân loại và làm cho chúng cung cấp những thông tin phản hồi bằng cách sử dụng những thông tin có sẵn được cung cấp bởi Framework. Những lớp đặc thù sử dụng là Category Attribute và Description Attribute. Khi sử dụng chúng trong đoạn mã của bạn, tên được viết ngắn là Category hay Description. Trình biên dịch (compiler) sẽ thêm phần Attribute cho bạn.

Chúng ta hãy xem lại lớp simpleObject 1 lần nữa. Đoạn mã sau hiển thị những đặc tính (attribute) được thêm vào 1 trong những thuộc tính (Property).

```
[Category("Appearance")]
[Description("controls the color of your moods")]
public Color TheColor
{
    get
    {
        return _color;
    }
    set
    {
        _color = value;
    }
}
```

Khi thuộc tính này được xem trong PropertyGrid, nó sẽ thuộc về phân loại Appearance, và việc mô tả sẽ được hiển thị trong ô feedback tại đây của ô PropertyGrid.

Lớp cũng có thể có những đặc tính giúp bạn xây dựng PropertyGrid. [DefaultProperty("propertyname")] sẽ chọn những thuộc tính nào hiển thị đầu tiên trong Property Grid. Ví dụ : hãy xem dòng 12 của ví dụ 3.6-4

Ví dụ 3.6-4 Thêm những đặc tính vào SimpleObject

```
3:using System.Drawing.Drawing2D;
4:using System.Windows.Forms.Design;
5:using System.ComponentModel;
6:
7:namespace simpleprops
8:{  
9:    /// <summary>
10:   /// summary description for SimpleObject
11:   /// </summary>
12:   [DefaultProperty("TheInteger")]
13:   public class SimpleObject
14:   {
15:       private int _int;
16:       private string _string;
17:       private Color _color;
18:
19:       public SimpleObject()
20:       {
21:           //
22:           //Khởi động biến ở đây.
23:           //
24:       }
25:
26:
27:       [Category("Nothing Special")]
28:       [Description("Yes guy's 'n gal's, its an
integer")]
29:       public int TheInteger
30:       {
31:           get
32:           {
33:               return _int;
34:           }
35:           set
36:           {
```



```
37:           _int = value;
38:       }
39:   }
40:
41:   [Category("A peice of string")]
42:   [Description("This string does absolutely
nothing but tie up your time")]
43:   public int TheString
44:   {
45:       get
46:       {
47:           return _string;
48:       }
49:       set
50:       {
51:           _string = value;
52:       }
53:   }
54:
55:   [Category("Appearance")]
56:   [Description("controls the color of
your moods")]
57:   public int TheString
58:   {
59:       get
60:       {
61:           return _color;
62:       }
63:       set
64:       {
65:           _color = value;
66:       }
67:   }
68: }
69: }
```

Những đặc tính attribute cũng có thể ngăn cản khả năng nhìn thấy của thuộc tính trong khung. Điều này khá hữu dụng nếu bạn khai báo thuộc tính toàn cục (public) cho người lập trình nhưng giữ nó từ người sử dụng. Để làm điều này, ta sử dụng đặc tính [Browsable(false)].

Những đặc tính attribute khác cho các thuộc tính chỉ hữu dụng nếu bạn đang viết những thành phần điều khiển (control) được dùng trong môi trường thiết kế, như là VS.NET.

3. GIẢI THÍCH ỨNG DỤNG : FORMPAINT.EXE

Ứng dụng này kết hợp nhiều yếu tố cơ bản lại với nhau, chúng tôi sẽ hướng dẫn bạn xây dựng nó và sẽ trình bày một vài tính năng của Windows Forms trong 1 ứng dụng ngữ cảnh đầy đủ.

Form Paint là một ứng dụng giao tiếp đa tài liệu (Multi Document Interface) MDI cho phép bạn vẽ những hình ảnh với cọ vẽ (brush) và các dạng hình học (shape). Nó có những menu tùy biến (customize) và cũng minh họa khả năng làm việc của GDI+.

Phần 1 : Khung Framework cơ bản

Bắt đầu, tạo một ứng dụng trong VS.NET và ngay đặt thuộc tính IsMDIContainer của form bằng true. Thao tác này khiến cho form làm cửa sổ cha cho những form cơ sở được giữ trong nó.

Tên Form1 phải được đổi thành MainForm và tiêu đề của cửa sổ đổi thành FormPaint. Nhớ rằng khi tên của MainForm thay đổi, thì bạn cũng phải thay đổi bằng tay phương thức tĩnh Main như sau:

```
[STAThread]
static void Main()
{
    Application.Run(new MainForm()); // đổi tên ứng dụng
}
```

Kéo MainMenu từ hộp công cụ lên trên MainForm, Dánh vào từ &File như tên menu và thêm vào 3 mục menu là &New, &Open và &Exit.

Kéo OpenFileDialog và SaveFileDialog từ hộp công cụ thả lên trên MainForm. OpenFileDialog và SaveFileDialog sẽ xuất hiện trong khay biểu tượng (icon tray) dưới form.

Nhấp đúp chuột lên mục menu Open trong menu chính và đánh dấu mă sau vào bộ xử lý :

```
This.openFileDialog1.Filter = "Bitmap
files (*.BMP)|*.bmp";
```

```
This.openFileDialog1.ShowDialog();
```

Tạo ra một menu thứ hai gọi là Windows. Menu này dùng để theo dõi những form con MDI trong ứng dụng. Điều này được làm tự động cho bạn nếu bạn đã thuộc tính MDIList của menu bằng true.

Bước kế tiếp, thêm vào 1 cửa sổ form mới vào dự án (project), gọi nó ChildForm. Đặt BackColor bằng green.

Thêm một biến cục bộ gọi là myImage kiểu Image vào form con và một phương thức truy xuất toàn cục như sau :

```
private Image myImage;
public Image Image
{
    set {myImage = value;}
    get {return myImage;}
}
```

Bây giờ, nhấp chuột đôi lên OpenFileDialog trong khay biểu tượng bên dưới form. Hành động này sẽ tạo ra bộ xử lý FileOK cho bạn. Dánh đoạn mã sau vào bộ xử lý :

```
ChildForm c = new ChildForm();
c.MdiParent = this;
c.Image =
Image.FromFile(this.openFileDialog1.Filename);
c.Text = this.openFileDialog1.FileName;
c.Show();
```

Những thao tác trong đoạn mã trên sẽ mở tập tin, tạo ra một form con, tải ảnh vào form con. Form con bây giờ cần hiển thị ảnh. Chọn ChildForm trong thiết kế, nhấp chuột trên nút nhấn Events trong Property Browser, và nhấp chuột lên sự kiện Paint. Thao tác này tạo ra mô hình chuyển giao (delegate) PaintEventHandler cho ChildForm và mở trình soạn thảo Code Editor. Dánh đoạn mã sau vào bộ xử lý.

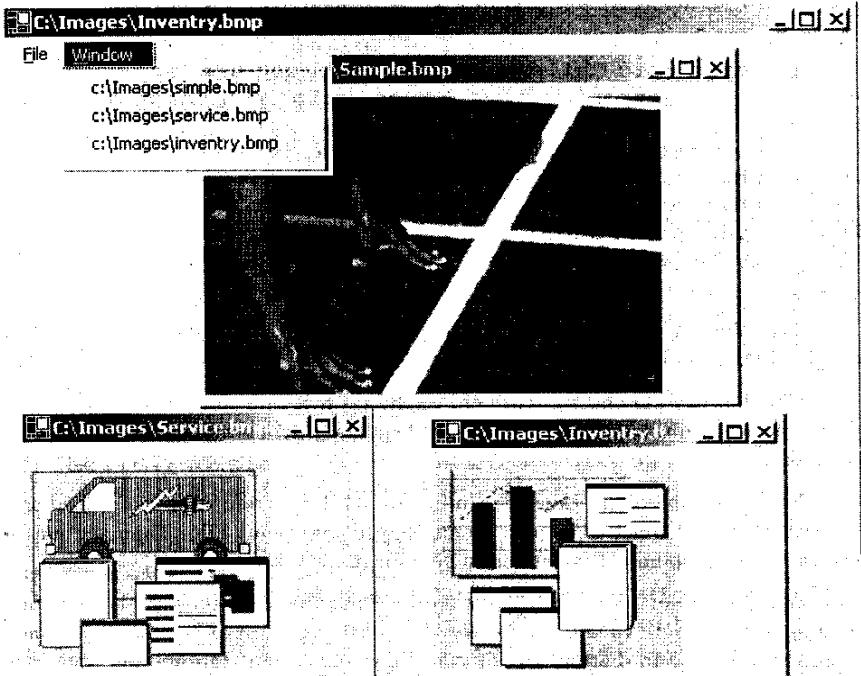
```
e.Graphics.DrawImage(myImage, 0, 0,
myImage.Width, myImage.Height);
```

Đây là nơi tốt nhất để ngừng và đánh giá dự án. Dịch và chạy chương trình bằng cách nhấn phím F5. Bạn sẽ thấy ứng dụng tải và hiển thị hình ảnh trên nhiều cửa sổ. Hình 3.6-4 hiển thị ứng dụng này.

Phần 2 : Cuộn cửa sổ và tạo ra những hình mới

Nếu bạn chạy chương trình FormPaint và tải lên vài hình ảnh, bạn chú ý những cửa sổ sẽ thay đổi kích thước. Nhưng, khi những cửa sổ này trở nên quá nhỏ để hiển thị hình ảnh, đơn giản những cửa sổ sẽ cắt đứt hình ảnh mà không

đưa cho bạn cơ hội để nhìn thấy những phần còn lại bị che từ cạnh cửa sổ chính trở đi. Chúng ta sẽ sửa vấn đề này một cách nhanh chóng bằng cách thêm đoạn mã đơn giản sau vào ChildForm.



Hình 3.6-4 *Ứng dụng cơ bản MDI.*

Đầu tiên, trong thuộc tính Image đặt bộ truy xuất (accessor) cho ChildForm, chúng ta sẽ thêm một lệnh để đặt thuộc tính AutoScrollMinSize bằng kích thước của hình trong biến myImage. Đoạn mã sau chỉ cách thêm vào chức năng này (dòng 6).

```

1: public Image Image
2: {
3:     set
4:     {
5:         myImage = value;
6:         this.AutoScrollMinSize = myImage.Size;
7:     }
8:     get{ return myImage; }
9: }
```

Bây giờ, bất cứ khi nào kích thước hình chữ nhật của vùng client bị giảm dưới kích thước nhỏ nhất trong mỗi hướng, thì thanh cuộn tương ứng xuất hiện.

Thứ hai, việc thay đổi phải tạo cho bộ xử lý OnPaint xác định vị trí của những thanh cuộn. Ở đây, chúng ta cần xác định vị trí tương đối của hình băng giá trị của vị trí thanh cuộn, vì vậy hình sẽ được vẽ ra một cách chính xác. Form cũng cung cấp thuộc tính vị trí thanh cuộn được hiển thị trong đoạn mã sau :

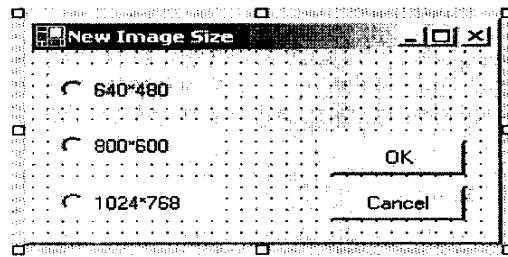
```
1:  private void ChildForm_Paint(object sender,
2:                                System.Windows.Forms.PaintEventArgs e)
3:  {
4:      e.Graphics.DrawImage(myImage,
5:                           this.AutoScrollPosition.x,
6:                           this.AutoScrollPosition.y,
7:                           myImage.Width,
8:                           myImage.Height);
9:  }
```

Trên dòng 4 và 5, bạn có thể nhìn thấy những giá trị X và Y của thuộc tính AutoScrollPosition của form là vị trí bắt đầu của ảnh.

Cho đến bây giờ chương trình chỉ có thể tải một ảnh tồn tại trên đia. Chúng ta cần tạo một ảnh, bằng kích thước ảnh được chọn, và cho phép ảnh đó được lùi xuống đia.

Việc tạo ảnh có thể thực hiện như sau: sử dụng trang thiết kế MainForm chọn và nhấp đôi chuột trên mục menu New trong menu File. Thao tác này sẽ sinh ra bộ xử lý (handler) mà chúng ta sẽ điền mã vào sau.

Bây giờ, nhấn phím chuột trên dự án FormPaint và chọn Add New Item. Chọn một form mới và gọi nó là NewImageDialog.cs. Kéo 3 nút chọn (Radio button) và 2 nút nhấn (button) từ hộp công cụ (toolbox) lên trên hộp hội thoại. Sau đó sắp xếp chúng như hình 3.6-5.



Hình 3.6-5 Nút nhấn sắp xếp trên hộp hội thoại

Thuộc tính DialogResult của nút OK và Cancel phải đặt bằng DialogResult.OK và DialogResult.Cancel riêng từng cái. Để lấy thông tin cài đặt từ những nút chọn, chúng ta có thể thêm vào một thuộc tính đơn giản trả về kí tự. Soạn thảo bằng tay đoạn mã của tập tin NewImageDialog.cs và thêm vào thuộc tính toàn cục sau :

```

public Size ImageSize
{
    get
    {
        if(this.radioButton2.Checked)
            return new Size(800, 600);
        if (this.radioButton3.Checked)
            return new Size(1024, 768);
        return new Size(640, 480);
    }
}

```

Bây giờ chúng ta có thể quay lại bộ xử lý đã tạo ra trước đây và điền mã vào nó để tạo ra ảnh rỗng. Tìm bộ xử lý trong tập tin MainForm.cs và thêm vào những mã cần thiết sau:

```

1: private void menuItem2_Click(object sender,
                               System.EventArgs e)
2: {
3:     // Tạo ra tập tin mới...
4:     NewImageDialog dlg = new NewImageDialog();
5:     if(dlg.ShowDialog() == DialogResult.OK)
6:     {
7:         ChildForm c = new ChildForm();
8:         c.MdiParent = this;
9:         c.Image = new Bitmap(dlg.ImageSize.Width,
10:                               dlg.ImageSize.Height);
11:         Graphics g = Graphics.FromImage(c.Image);
12:         g.FillRectangle(new
13:                         SolidBrush(Color.White), 0, 0,
14:                         c.Image.Width, c.Image.Height);
12:         c.Show();
13:     }
14: }

```

Thêm vào chức năng lưu tập tin

Việc lưu tập tin (Save) hay là lưu như tập tin khác (Save As) sẽ được thực hiện thông qua menu chính File. Bạn thêm vào mục chọn menu Save và Save As. Nhớ sử dụng ký hiệu "&" trước những ký tự chính – Ví dụ: &Save và Save &As. Đối với những lý do thẩm mỹ, bạn có thể muốn dời những mục chọn menu vào nơi dễ nhận thấy trên menu và đưa một thanh phân cách giữa những chức năng của File và Exit.

Khi bạn tạo ra những mục này, tiếp tục lần lượt nhấp đôi chuột cho mỗi mục để tạo ra những bộ xử lý cho chúng. Đoạn mã sau hiển thị 2 bộ xử lý lưu tập tin.



```
1: private void menuItem6_Click(object sender,
                               System.EventArgs e)
2: {
3:     // Lưu tập tin ảnh.
4:     ChildForm child = (ChildForm)this.ActiveMdiChild;
5:     child.Image.Save(child.Text);
6: }
7:
8: private void menuItem7_Click(object sender,
                               System.EventArgs e)
9: {
10:    // Lưu tập tin ảnh ra một tập tin khác
11:    ChildForm child =
12:        (ChildForm)this.ActiveMdiChild;
13:    this.saveFileDialog1.FileName = child.Text;
14:    if(this.saveFileDialog1.ShowDialog() ==
15:        DialogResult.OK)
16:    {
17:        child.Image.Save(
18:            this.saveFileDialog1.FileName);
19:        child.Text = this.saveFileDialog1.FileName;
20:    }
21: }
```

Từ dòng 1 tới dòng 6 là bộ xử lý lưu tập tin đơn giản. Tên tập tin đã được biết, vì vậy việc lưu tập tin được thực hiện bằng cách sử dụng văn bản của thanh tiêu đề ChildForm.

Từ dòng 13 tới dòng 18 ta sử dụng tên tập tin trả về từ lời triết gọi SaveFileDialog (hộp thoại SaveFileDialog cho phép người sử dụng chọn tên tập tin mới để lưu file).

Hành vi này tốt nếu bạn có thể bảo đảm rằng ChildForm luôn luôn mở. Nếu bạn sử dụng xử lý trên một form chính rỗng, thì một ngoại lệ (exception) sẽ xảy ra bởi không có cửa sổ con MDI hoạt động (active). Vấn đề này có thể khắc phục bằng một trong hai cách. Hoặc là viết mã xử lý ngoại lệ, sử dụng khối try ... catch vào trong 2 bộ xử lý hoặc, không bao giờ cho phép bộ xử lý được gọi nếu không có cửa sổ con MDI. Phương pháp thứ 2 khá tốt cho trường hợp của ta bởi nó cho phép chúng ta sử dụng những sự kiện menu popup 1 cách chính xác.

Ở chương 3.1 “Giới Thiệu Windows Forms”, bạn đã học sự kiện menu popup được kích hoạt khi người sử dụng chọn menu ở mức cao trên thanh công cụ (toolbar) trước khi menu được vẽ. Điều này đưa cho chúng ta cơ hội sửa đổi hành vi menu phù hợp với những hoàn cảnh hiện hành. Đoạn mã sau hiển thị bộ xử lý Popup cho mục menu File.

```
1: private void menuItem1_Popup(object sender,
                               System.EventArgs e)
```

```

2: {
3:     if(this.MdiChildren.Length != 0)
4:     {
5:         menuItem6.Enabled = true;
6:         menuItem7.Enabled = true;
7:     }
8: else
9: {
10:    menuItem6.Enabled = false;
11:    menuItem7.Enabled = false;
12: }
13: }
```

Tại dòng 3, bộ xử lý kiểm tra xem form chính có cửa sổ con nào không. Nếu có, thì nó sẽ cho phép sử dụng mục menu Save (tại dòng 5 và 6). Nếu không có bất kỳ cửa sổ con nào, nó không cho phép hay vô hiệu hóa (disable) những mục menu này (dòng 10 và 11).

Phần 3 : Giao diện (User Interface).

Với việc tái ánh, hiển thị, và lưu ánh vào một nơi, chúng ta có thể xúc tiến công việc thêm vào phần còn lại những mục giao diện mà sẽ được sử dụng để chạy chương trình. Chúng ta cần một bảng công cụ (palette tool); trong trường hợp này, bảng công cụ đơn giản sẽ đủ giải thích những yếu tố cơ bản, thanh trạng thái (status bar), xử lý chuột, và lớp nút nhấn tùy biến (custom control) được dẫn xuất trên UserControl.

Tạo ra một Custom User Control

Một thuận tiện quan trọng của những user control trong .NET là tính dễ sử dụng lại của chúng. Những control này được chạy trong môi trường thiết kế và có thể được bán như những bộ phận lắp ráp nếu bạn muốn.

Để thêm vào dự án thư viện một control C# mới. Bạn chọn nút Add to Current Solution trên hộp hội thoại wizard. Đặt tên dự án là FormPaintControl. Bạn sẽ được đưa ra với một wizard để chọn kiểu control thêm vào. Chọn UserControl mới và đặt tên nó là CustomImageButton.cs.

Ví dụ 3.6-5 hiển thị đầy đủ mã nguồn của CustomImageButton control.

Ví dụ 3.6-5 CustomImageButton.cs : Lớp CustomImageButton

```

1: using System;
2: using System.Collections;
3: using System.ComponentModel;
4: using System.Drawing;
5: using System.Drawing.Drawing2D;
6: using System.Data;
```



```
7: using System.Windows.Forms;
8: using System.Drawing.Imaging;
9:
10: namespace FormPaintControl
11: {
12: /// <summary>
13: /// Summary description for CustomImageButton.
14: /// </summary>
15: public class CustomImageButton :
16:     System.Windows.Forms.UserControl
17: {
18:     private Image image;
19:     private Color transparentColor;
20:     private bool ownerDraw;
21:     private bool down;
22:     [
23:     Category("Behavior"),
24:     Description("Allows external paint delegates
25:                 to draw the control")
26:     public bool OwnerDraw
27:     {
28:         get
29:         {
30:             return ownerDraw;
31:         }
32:         set
33:         {
34:             ownerDraw = value;
35:         }
36:     }
37:
38:     [
39:     Category("Appearance"),
40:     Description("The image displayed on the control")
41:     ]
42:     public bool Image
43:     {
```

```
44:     get
45:     {
46:         return image;
47:     }
48:     set
49:     {
50:         image = value;
51:     }
52: }
53:
54: [
55:     Category("Appearance"),
56:     Description("The transparent color used
57:                 in the image")
58: ]
59: public bool TransparentColor
60: {
61:     get
62:     {
63:         return transparentColor;
64:     }
65:     set
66:     {
67:         transparentColor = value;
68:     }
69:
70: protected override void OnSizeChanged(EventArgs e)
71: {
72:     if(DesignMode)
73:         Invalidate();
74:     base.OnSizeChanged(e);
75: }
76:

77: public void DrawFocusRect(Graphics g)
78: {
79:     Pen p = new Pen(Color.Black,1);
```



```
80:     p.DashStyle = DashStyle.Dash;
81:     Rectangle rc = ClientRectangle;
82:     rc.Inflate(-1,-1);
83:     g.DrawRectangle(p,rc);
84: }
85:
86: protected override void OnLostFocus(EventArgs e)
87: {
88:     Invalidate();
89:     Base.OnLostFocus(e);
90: }
91:
92: protected override void OnPaint(PaintEventArgs e)
93: {
94:     Rectangle rc = ClientRectangle;
95:     rc.Offset(4,4);
96:     e.Graphics.SetClip(ClientRectangle);
97:     if(!ownerDraw)
98:     {
99:         if(image == null || !(image is Bitmap))
100:        {
101:            Pen p = new Pen(Color.Red,2);
102:            e.Graphics.DrawRectangle(p,
103:                                     ClientRectangle);
104:            e.Graphics.DrawLine(p,
105:                               ClientRectangle.Location.X,
106:                               ClientRectangle.Location.Y,
107:                               ClientRectangle.Location.X +
108:                               ClientRectangle.Width,
109:                               ClientRectangle.Location.Y,
110:                               ClientRectangle.Location.X,
111:                               ClientRectangle.Location.Y +
112:                               ClientRectangle.Height);
112:     return;
```



```
113:     }
114:     Bimap bm = (Bitmap)image;
115:     bm.MakeTransparent(transparentColor);
116:     if(down || Focused)
117:     {
118:         e.Graphics.DrawImage(bm,
119:             rc.Location.X,
120:             rc.Location.Y,
121:             bm.Width,bm.Height);
122:         if(Focused)
123:             DrawFocusRect(e.Graphics);
124:     }
125:     else
126:     {
127:         ControlPaint.DrawImageDisabled(
128:             e.Graphics, bm,
129:             rc.Location.X,rc.Location.Y,BackColor);
130:         e.Graphics.DrawImage(bm,
131:             ClientRectangle.Location.X,
132:             ClientRectangle.Location.Y,
133:             bm.Width, bm.Height);
134:     }
135:     base.OnPaint(e);
136: }
137:
138: /// <summary>
139: /// Required designer variable
140: /// </summary>
141: private System.ComponentModel.Container
142:         components = null;
143: public CustomImageButton()
144: {
145:
146:     InitializeComponent();
147:
148:     //Thêm các khởi động khác sau khi gọi InitForm
```

```
149:  
150: }  
151:  
152: protected override void OnMouseEnter(EventArgs e)  
153:{  
154:     base.OnMouseEnter(e);  
155:     down = true;  
156:     Invalidate();  
157: }  
158:  
159: protected override void OnMouseLeave(EventArgs e)  
160: {  
161:     base.OnMouseLeave(e);  
162:     down = false;  
163:     Invalidate();  
164: }  
165:  
166: [  
167: Browser(true);  
168: DesignerSerializationVisibility(  
        DesignerSerializationVisibility.Visible)  
169: ]  
170: public override string Text  
171: {  
172:     get  
173:     {  
174:         return base.Text;  
175:     }  
176:     set  
177:     {  
178:         base.Text = value;  
179:     }  
180: }  
181:  
182: /// <summary>  
183: /// Clean up any resource being used.  
184: /// </summary>  
185: protected override void Dispose(bool disposing)
```

```
186: {
187:     if(disposing)
188:     {
189:         if(components != null)
190:             components.Dispose();
191:     }
192:     base.Dispose(disposing);
193: }
194:
195: #region Component Designer generated code
196: /// <summary>
197: /// Required method for Designer support - do not
   modify
198: /// the contents of this method of the code editor.
199: /// </summary>
200:     private void InitializeComponent()
201:     {
202:         //
203:         // CustomImageButton
204:         //
205:         this.Name = "CustomImageButton";
206:         this.Size = new System.Drawing.Size(64, 56);
207:
208:     }
209: #endregion
210:
211: }
212: }
```

Bắt đầu với dòng 1, chúng ta thêm vào việc sử dụng những không gian tên (namespace) cho các điều khiển (mặc định chỉ có không gian tên System được sử dụng).

Dòng 10 định nghĩa không gian tên FormPaintControl, dòng 15 bắt đầu khai báo lớp CustomImageButton.

Từ dòng 17 tới dòng 20 định nghĩa những biến dữ liệu thành viên cục bộ để lưu thông tin sử dụng trong việc vẽ điều khiển control. Có một biến dữ liệu thành viên image kiểu Image, một biến dữ liệu thành viên transparentColor kiểu Color, hai cờ kiểu bool cho thuộc tính ownerDraw và trạng thái của nút nhấn là chìm xuống hay nổi lên (down hay up).



Phương thức truy xuất toàn cục những thuộc tính cho những thành viên liên quan bảo đảm control sẽ tích hợp với môi trường thiết kế và đưa ra thông tin phản hồi cho người sử dụng. Chú ý những đặc tính Category và Description có trước thuộc tính định nghĩa trên những dòng 26, 42, và 58.

Dòng 70 định nghĩa phương thức OnSizeChanged. Khi control này được sử dụng trong thiết kế, nó cần vẽ lại chính nó bất cứ khi nào kích thước của nó bị thay đổi. Nó cũng gọi phương thức từ lớp cơ sở để bảo đảm rằng bất kỳ những mô hình chuyển giao (delegate) khác được thêm vào sự kiện SizeChanged cũng sẽ được gọi.

Dòng 77 định nghĩa phương thức mới, phương thức này vẽ một đường nhiều chấm (dotted line) xung quanh control khi nó nhận được tác động của phím hay chuột (còn gọi là focus). Phương thức này được gọi từ phương thức OnPaint.

Phương thức trên dòng 86 bảo đảm control được vẽ lại khi trạng thái focus bị mất.

Phương thức OnPaint, bắt đầu tại dòng 92 phải tranh giành với việc vẽ bình thường của control và việc vẽ control khi nó được sử dụng trong môi trường thiết kế. Điều này có nghĩa là trong thể hiện (instance) ban đầu, control sẽ được phát minh bởi người thiết kế với thuộc tính image rỗng. Vì vậy những ngoại lệ (exception) sẽ không xảy ra, control sẽ vẽ hình chữ nhật màu đỏ với hình chữ thập màu đỏ trong nó nếu không có hình để vẽ. Điều này được thực hiện trên những dòng từ 102 tới 122. Nói cách khác, việc vẽ diễn ra bình thường trên dòng 118 tới 123 nếu nút nhấn bị lún xuống hay nhận được focus.

Dòng từ 127 tới 132 vẽ nút nhấn trong vị trí lồi lên của nó. Chú ý việc sử dụng phương thức ControlPaint trên những dòng 127 và 128 để vẽ một hình bị mở đăng sau ảnh bitmap chính như là ảnh mờ. Phương thức này cũng gọi phương thức cơ sở (base) OnPaint để bảo đảm việc vẽ chính xác khi bộ xử lý vẽ khác được thêm vào sự kiện Paint.

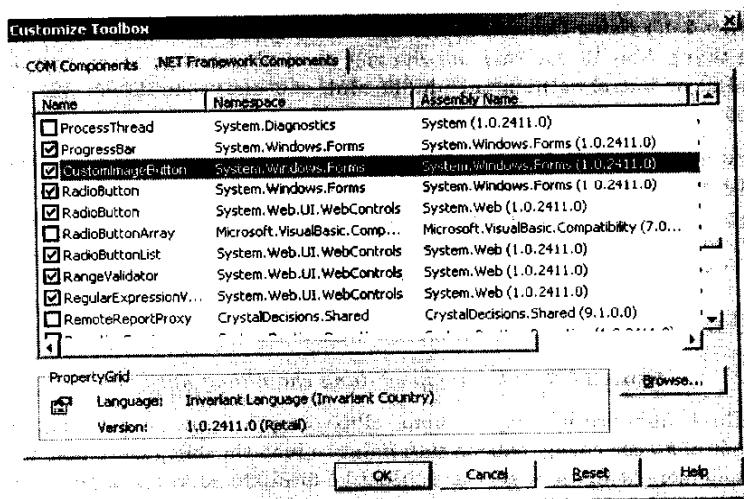
Control chuẩn được thêm vào tất cả những dòng từ 138 tới 150, và chúng ta tiếp tục thêm những bộ xử lý MouseEnter và MouseLeave trên những dòng 152 và 159. Những dòng này làm cho nút nhấn ấn xuống bắt cứ khi nào chuột lướt đi trên nó.

Trên dòng 167, có một đặc tính attribute cho phép thuộc tính Text được nhìn thấy trong Property Browser. UserControl từ lớp này được dẫn xuất ẩn thuộc tính, vì vậy chúng ta cần định nghĩa chồng (override) lên nó (dòng 170 - 180) và làm cho nó có thể nhìn thấy được trong khung nhìn. Trên dòng 169, đặt tính DesignerSerializationVisibility bảo đảm văn bản được lưu trong phương thức InitializeComponent.

Phần còn lại của tập tin là rác được đưa vào bởi control wizard và bao gồm phương thức Dispose và phương thức InitializeComponent.



Dịch control và sau đó nhấn phải chuột trên hộp công cụ (toolbox) chọn CustomizeToolbox và và bạn sẽ nhìn thấy hộp hội thoại được hiển thị trong hình 3.6-6.



Hình 3.6-6 Thêm control vào toolbox.

Để nhìn thấy CustomImageButton, đầu tiên bạn phải tìm tập tin DLL được dịch. Tập tin này ở trong :

<Tên dự án của bạn>\
Formpaint\FormPaintControl\Debug\Bin\FormpaintControl.dll

Bảo đảm hộp kiểm tra (check box) phải được đánh dấu và control sẽ xuất hiện trong hộp công cụ (toolbox) của bạn.

Chú ý :

Bạn có thể làm control này như một công cụ lâu dài để sử dụng trong chương trình của bạn nếu bạn chuyển sang kiểu Realease, dịch lại control và sau đó chọn phiên bản release của FormPaintControl.dll. Bạn cũng có thể chép phiên bản release này tới một nơi trên đĩa cứng của bạn, ở nơi này bạn chắc chắn tập tin dll không bị xóa.

Sử dụng CustomImageButton

Thêm một ô Panel vào MainForm và gắn nó vào cạnh bên phải của form. Khu vực này dùng để giữ bảng công cụ đơn giản của chúng ta. Kéo thanh trạng thái (status bar) từ hộp công cụ lên trên form, chọn các khung Panel trong Property Browser, và nhấp chuột trên nút nhấn khi combo box xuất hiện.

Bạn sẽ thấy một hộp thoại cho phép thêm vào một hay nhiều đối tượng StatusBarPanel; cho đến bây giờ chúng ta chỉ cần thêm một. Nhớ đặt những thuộc tính ShowPanel của đối tượng StatusBar bằng true.

Kéo 4 đối tượng CustomImageControl lên trên Panel và sắp xếp chúng theo phương thẳng đứng để tạo ra 1 nơi cho việc chọn những công cụ vẽ. Ban đầu, 1 CustomImageControl sẽ hiển thị như là hình chữ nhật màu đỏ với dấu chữ tì trong nó.

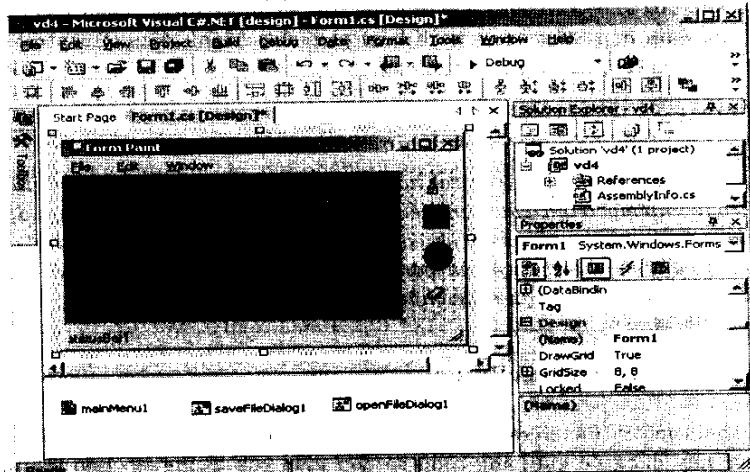
Sử dụng Property Browser, thêm vào các hình cho thuộc tính Image của nhấp. Chúng ta chọn hình bút vẽ (paintbrush), hình chữ nhật, hình Ellip, hình gồm cho bảng công cụ của chúng ta (tool palette). Hình 3.6-7 hiển thị 4 hình nà



Hình 3.6-7 Ảnh bitmap sử dụng cho những nút nhấp.

Khi 4 ảnh brush.bmp, rect.bmp, ellipse.bmp, và eraser.bmp được đưa vào những nút nhấp của chúng, phương thức Paint sẽ hiển thị chúng cho chúng ta, chèn trong môi trường thiết kế. Cửa sổ thiết kế form khi đó sẽ trông như hình 3.

CustomImageControl có thể giữ văn bản (text) cho chúng ta, vì vậy chúng có thể sử dụng control này để tạo ra giao diện (UI) tích hợp đơn giản và cơ phản hồi thông tin (feedback). Chúng ta sẽ lưu thông tin trong văn bản (text) chú giải nút nhấp với tooltips và thông tin phản hồi trên thanh trạng thái (status bar). Sau đó, chúng ta sử dụng cùng cơ chế phản hồi thông tin để nâng cấp menu.



Hình 3.6-8 Bảng bố trí thiết kế cuối cùng của MainForm.

Thuộc tính Text của 4 nút nhấn sẽ chứa đựng caption, tooltip, và status bar. Chúng được tách rời bởi một thanh đứng (ký tự 124). Ví dụ : dòng sau đây hiển thị định dạng của thuộc tính Text.

Caption Text | Status Bar feedback | Tooltip text .

Lần lượt mỗi thuộc tính Text, thêm vào dòng thích hợp sử dụng Property Browser :

```
CustomImageButton1.Text "|Apply color with a brush tool  
| Paintbrush"
```

```
CustomImageButton1.Text "|Draw a rectangle| Rectangle"
```

```
CustomImageButton1.Text "|Draw a ellipse | Ellipse"
```

```
CustomImageButton1.Text "|Eraser an area| Eraser"
```

Chú ý : không có thuộc tính Text nào có phần caption, bởi vì text caption không cần thiết.

Status và Tooltips

Tooltip đưa cho người sử dụng thông tin phản hồi hữu dụng và những mục trong bảng công cụ (tool palette) của chúng ta có một văn bản (text) được hiển thị khi chuột di chuyển trên chúng, vì vậy chương trình FormPaint sử dụng một phương thức đơn giản để hiển thị tooltip và văn bản trong thanh trạng thái (status bar). Lợi dụng khả năng của lớp String chúng ta tách chuỗi tại những vị trí có ký tự '|', chúng ta có thể tạo ra một lớp đơn giản, lớp này sẽ lấy tooltip, status bar và caption text cho chúng ta. Thêm một lớp C# mới vào dự án và đặt tên nó là CSTSplit.

Lớp CSTSplit là lớp đơn giản và được hiển thị trong ví dụ 3.6-6

Ví dụ 3.6.6 Lớp CSTSplit.cs : Lớp SCTSsplit

```
1: using System;  
2:  
3: namespace FormatPaint  
4: {  
5:     /// <summary>  
6:     /// CSTSplit divides up a given string into  
7:     /// peices at a "|" delimiter to provide  
8:     /// Caption , status, and Tooltip text.  
9:     /// </summary>  
10:    public class CSTSplit  
11:    {  
12:        string[] splitString;  
13:        public CSTSplit(string toSplit)  
14:        {  
15:            splitStrings = toSplit.Split(new char[] {'|'});  
16:        }
```

```
17: }
18:
19: public string Caption
20: {
21:     get
22:     {
23:         if(splitStrings.Length > 0)
24:             return splitStrings[0];
25:         return "";
26:     }
27: }
28:
29: public string Status
30: {
31:     get
32:     {
33:         if(splitStrings.Length > 1)
34:             return splitStrings[1];
35:         return "";
36:     }
37: }
38:
39: public string Tooltip
40: {
41:     get
42:     {
43:         if(splitStrings.Length > 2)
44:             return splitStrings[2];
45:         return "";
46:     }
47: }
48: }
49: }
```

Phương thức khởi dụng trên dòng 10 tách một chuỗi (string) được cung cấp thành một mảng 3 chuỗi được chia tại những thanh phân cách.

Những thuộc tính Caption, Status, Tooltip (tại những dòng 19, 20 và 39) lấy một chuỗi chính xác hoặc cấp một chuỗi rỗng nếu không có văn bản (text) cho phần đó của chuỗi được đưa vào.

Lúc đầu sử dụng lớp này sẽ thêm những tooltip vào các nút nhấn trong tool palette. Điều này được thực hiện trong phương thức khởi dụng của MainForm, sau khi phương thức InitializeComponent được gọi. Để hiển thị tooltip, kéo đối tượng tooltip từ hộp công cụ lên trên trang thiết kế của MainForm. Tooltip sẽ xuất hiện trong khay biểu tượng (icon tray) dưới cửa sổ chính. Sau đó đưa đoạn mã sau vào phương thức khởi tạo của MainForm.

```
CSTSsplit splitter = new CSTSplit(this.customImageButton1.Text);
this.toolTip1.SetToolTip(this.customImageButton1, splitter.ToolTip);
splitter = new CSTSplit(this.customImageButton2.Text);
this.toolTip1.SetToolTip(this.customImageButton2, splitter.ToolTip);
splitter = new CSTSplit(this.customImageButton3.Text);
this.toolTip1.SetToolTip(this.customImageButton3, splitter.ToolTip);
splitter = new CSTSplit(this.customImageButton4.Text);
this.toolTip1.SetToolTip(this.customImageButton4, splitter.ToolTip);
```

Bạn có thể thấy rằng lần lượt mỗi CustomImageButton bị truy xuất thuộc tính text và thuộc tính ToolTip từ đối tượng CSTSplit. CSTSplit trả về text chính xác được xử lý bởi phương thức ShowTip của ToolTip.

Bây giờ, khi chuột ngừng trên nó khoảng nửa giây hoặc hơn, 1 tooltip sẽ xuất hiện chứa đựng nội dung văn bản chính xác.

Mỗi control này cũng chứa đựng văn bản (text) cho thanh trạng thái (status bar). Để tạo cho text này hiển thị chính nó, đầu tiên chọn control CustomImageButton trong vùng thiết kế và đánh tên phương thức ShowStatus vào trong sự kiện MouseEvent trong Properties Browser. Bộ xử lý sẽ được tạo và diễn mã vào như sau :

```
private void ShowStatus(object sender,
                        System.EventArgs e)
{
    Control c = (Control)sender;
    CSTSplit splitter = new CSTSplit(c.Text);
    this.statusBarPanel1.Text = splitter.Status;
}
```

Dịch và chạy chương trình sẽ hiển thị những thông điệp của tooltip và thanh trạng thái (status bar) một cách chính xác.

Phần 4 : Tool properties và ứng dụng

Bây giờ chương trình đã ở trong trạng thái mà chúng ta có thể bắt đầu làm việc thực sự. Chúng ta cần chọn những thuộc tính riêng lẻ cho những tool và áp dụng chúng vào ảnh bitmap.

Đối tượng Paintbrush yêu cầu một vài thuộc tính, như là hình bút lông, màu sắc, và kích thước. Công cụ hình Ellip và hình chữ nhật (Rectangle) chỉ cần định nghĩa màu của đường nét, màu tö, và độ dày của nét. Cục gôm (eraser) sẽ có kích thước và hình ảnh, giống như bút vẽ (paintbrush), nhưng nó sẽ luôn luôn xóa bằng màu trắng.



Bạn có thể sử dụng PropertyGrid để chọn những thuộc tính này. Chúng ta sẽ tạo ra một lớp cho mỗi công cụ, lớp này sẽ giữ lại những chọn lựa của người sử dụng.

Định nghĩa Tool Properties

Ba lớp công cụ mô tả các đối tượng được trình bày trong ví dụ 3.6-7,3.6-8 và 3.6-9 sau đây

Ví dụ 3.6-7 Tập tin PaintBrushProperties.cs

```
1: using System;
2: using System.ComponentModel;
3: using System.Drawing;
4: using System.Globalization;
5:
6: namespace FormPaint
7: {
8:
9:     public enum Shape
10:    {
11:         Round,
12:         Square,
13:         Triangle
14:    };
15:
16:
17:     public class PaintBrushProperties
18:    {
19:         private Shape shape;
20:         private int size;
21:         private Color color;
22:         private int transparency;
23:
24:         public object Clone()
25:        {
26:             return new PaintBrushProperties(shape, size,
27:                 color, transparency);
28:         }
29:
30:         protected PaintBrushProperties(Shape _shape,
31:             int _size, Color _color, int _transparency)
32:        {
33:             shape = _shape;
34:             size = _size;
35:             color = _color;
36:             transparency = _transparency;
37:         }
38:
```

```
37:
38: [
39: Category("Brush"),
40: Description("The brush shape")
41: ]
42: public Shape Shape
43: {
44:     get{return shape;}
45:     set{shape = value;}
46: }
47:
48: [
49: Category("Brush"),
50: Description("The brush color")
51: ]
52: public Color Color
53: {
54:     get{return color;}
55:     set{color = value;}
56: }
57:
58: [
59: Category("Brush"),
60: Description("The size of the brush in pixels")
61: ]
62: public int Size
63: {
64:     get{return size;}
65:     set{size = value;}
66: }
67:
68: [
69: Category("Brush"),
70: Description("Percentage of transparency
for the brush")
71: ]
72: public int Transparency
73: {
74:     get{return transparency;}
75:     set{transparency = value;}
76: }
77:
78: public PaintBrushProperties()
79: {
80:     color = Color.Black;
81:     size = 5;
82:     shape = Shape.Round;
```



```
83:     transparency = 0;
84: }
85: }
86: }
```

Ví dụ 3.6-8 ShapeProperties.cs : Lớp ShapeProperties

```
1: using System.Drawing;
2:
3: namespace FormPaint
4: {
5:     public class ShapeProperties
6:     {
7:         private Color fillColor;
8:         private Color lineColor;
9:         private bool line;
10:        private bool fill;
11:        private int lineWidth;
12:
13:        public object Clone()
14:        {
15:            return new ShapeProperties(fillColor, lineColor,
16:                                      line, fill, lineWidth);
17:        }
18:
19:        protected ShapeProperties(Color _fillColor,
20:                                  Color _lineColor, bool _line, bool _fill,
21:                                  int _lineWidth)
22:        {
23:            fillColor = _fillColor;
24:            lineColor = _lineColor;
25:            fill = _fill;
26:            line = _line;
27:            lineWidth = _lineWidth;
28:
29:
30:        [
31:            Category("Geometric Shape"),
32:            Description("The color to fill the shape with")
33:        ]
34:        public Color FillColor
35:        {
36:            get{return fillColor;}
37:            set{fillColor = value;}
38:        }
39:
40:    [
```

```
41: Category("Geometric Shape"),
42: Description("The color to outline the shape with")
43: ]
44: public Color LineColor
45: {
46:     get{return lineColor;}
47:     set{lineColor = value;}
48: }
49:
50: [
51: Category("Geometric Shape"),
52: Description("The width of the line"),
53: ]
54: public int LineWidth
55: {
56:     get{return lineWidth;}
57:     set{lineWidth = value;}
58: }
59:
60: [
61: Category("Geometric Shape"),
62: Description("Draw the outline")
63: ]
64: public bool Line
65: {
66:     get{return line;}
67:     set{line = value;}
68: }
69:
70: [
71: Category("Geometric Shape"),
72: Description("Fill the shape")
73: ]
74: public bool Fill
75: {
76:     get{return fill;}
77:     set{fill = value;}
78: }
79:
80:     public ShapeProperties()
81:     {
82:     }
83: }
84: }
```

Ví dụ 3.6-9 EraserProperties.cs : Lớp EraserProperties

```
1: using System;
```



```
2: using System.ComponentModel;
3:
4: namespace FormPaint
5: {
6:     public class EraserProperties
7:     {
8:         private Shape shape;
9:         private int size;
10:
11:        public object Clone()
12:        {
13:            return new EraserProperties(shape, size);
14:        }
15:
16:        protected EraserProperties(Shape _shape, int
17:        _size)
18:        {
19:            shape = _shape;
20:            size = _size;
21:        }
22:
23:        [
24:            Category("Eraser"),
25:            Description("The Eraser shape")
26:        ]
27:        public Shape Shape
28:        {
29:            get{return shape;}
30:            set{shape = value;}
31:        }
32:
33:        [
34:            Category("Eraser"),
35:            Description("The size of the Eraser in pixels")
36:        ]
37:        public int Size
38:        {
39:            get{return size;}
40:            set{size = value;}
41:        }
42:        public EraserProperties()
43:        {
44:            shape = Shape.Square;
45:            size = 10;
46:        }
47:    }
48:}
```

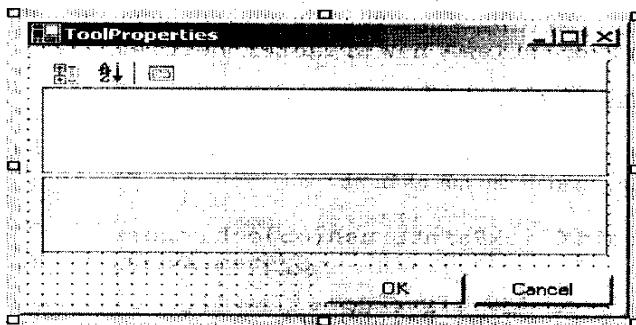
Chú ý : Tất cả những đối tượng đều có phương thức `Clone()`. Phương thức này sử dụng khi thuộc tính được chỉnh sửa trong hộp thoại, và người sử dụng có sự lựa chọn để nhấp chuột lên nút nhấn `Cancel`. Nếu hộp hội thoại này bị hủy, thì việc sao chép lại đối tượng bằng phương thức `Clone()` sẽ bị hủy bỏ.

Soạn thảo Tool Properties

Việc soạn thảo cho tất cả thuộc tính của công cụ được thực hiện với cùng một hộp thoại. Hộp hội thoại `ToolProperties` được tạo bằng môi trường thiết kế IDE như sau:

Kéo 2 nút nhấn từ thanh công cụ (toolbar) vào hộp thoại, ghi nhãn lên chúng là `OK` và `Cancel`, và sau đó đặt thuộc tính `DialogResult` của chúng bằng giá trị `tương ứng`.

Kéo và xác định vị trí đối tượng `PropertyGrid` lên trên bề mặt của bản thiết kế. Kết quả cuối cùng nên giống như hình 3.6-9.



Hình 3.6-9 Hộp hội thoại `ToolProperties`

Hộp hội thoại này cần đưa vào một vài hành vi, vì vậy chúng ta sẽ thêm một thuộc tính để đặt và lấy đối tượng mà `PropertyGrid` soạn thảo. Đoạn mã sau hiển thị phương thức truy xuất đơn giản :

```
public object Object
{
    get{return this.propertyGrid1.SelectedObject;}
    set{this.propertyGrid1.SelectedObject = value;}
}
```

Sau phương thức truy xuất này, chúng ta có thể thêm vào chức năng cho việc soạn thảo và sử dụng các thuộc tính.

MainForm của ta phải sửa đổi để giữ thuộc tính và tool hiện hành đang được sử dụng. Đoạn mã sau hiển thị những chức năng thêm vào.

Cấu trúc Tool được thêm vào không gian tên `FormPaint` và dùng để định nghĩa tool nào đang được sử dụng.



```
public enum Tool
{
    Paintbrush,
    Rectangle,
    Ellipse,
    Eraser
}
```

Sau đây là những biến thành viên cục bộ được thêm vào lớp FormPaint:

```
private Tool currentTool;
private PaintBrushProperties paintbrushProperties;
private ShapeProperties shapeProperties;
private EraseProperties eraserProperties;
```

Các lệnh sau đây được đưa vào phương thức khởi động MainForm.

```
paintbrushProperties = new PaintBrushProperties();
shapeProperties = new ShapeProperties();
eraserProperties = new EraseProperties();
```

Thêm vào các bộ xử lý cho Tool Properties

Để sử dụng công cụ, người dùng sẽ nhấp một lần chuột lên nó. Chúng ta cài đặt thủ tục sự kiện cho nút nhấn như sau

```
private void ClickPaintbrush(object sender,
                             System.EventArgs e)
{
    currentTool = Tool.Paintbrush;
}

private void ClickRectangle(object sender,
                           System.EventArgs e)
{
    currentTool = Tool.Rectangle;
}

private void ClickEllipse(object sender,
                         System.EventArgs e)
{
    currentTool = Tool.Ellipse;
}

private void ClickEraser(object sender,
                        System.EventArgs e)
{
    currentTool = Tool.Eraser;
}
```

Bạn có thể thấy rằng những bộ xử lý này đơn giản chỉ thay đổi những giá trị của biến thành viên currentTool trong MainForm.

Bây giờ tất cả bốn nút nhấn cần phải gắn vào cùng bộ xử lý DoubleClick. Bắt đầu bằng cách tạo bộ xử lý gọi là OnToolProperties cho mỗi nút nhấn và điền mã cho chúng như sau :

```
private void OnToolProperties(object sender,
                               System.EventArgs e)
{
    ToolProperties dlg = new ToolProperties();

    switch(currentTool)
    {
        case Tool.Paintbrush:
            dlg.Object = paintbrushProperties.Clone();
            break;
        case Tool.Rectangle:
            dlg.Object = shapeProperties.Clone();
            break;
        case Tool.Ellipse:
            dlg.Object = shapeProperties.Clone();
            break;
        case Tool.Eraser:
            dlg.Object = eraserProperties.Clone();
            break;
    }
    if(dlg.ShowDialog() == DialogResult.OK)
    {
        switch(currentTool)
        {
            case Tool.Paintbrush:
                paintbrushProperties =
                    (PaintBrushProperties)dlg.Object;
                break;
            case Tool.Rectangle:
                shapeProperties = (ShapeProperties)dlg.Object;
                break;
            case Tool.Ellipse:
                shapeProperties = (ShapeProperties)dlg.Object;
                break;
            case Tool.Eraser:
                eraserProperties = (EraserProperties)dlg.Object;
                break;
        }
    }
}
```

Bộ xử lý này quyết định thuộc tính của những lớp nào được sử dụng, dù một lớp chính xác vào trình soạn thảo, triệu gọi trình soạn thảo, bỏ hoặc thay thế thuộc tính bị soạn thảo tất cả phụ thuộc vào giá trị của DialogResult.

Để cho phép ChildForm truy xuất những biến của MainForm, như là tool và hiện hành hoặc những thuộc tính của cọ vẽ (brush), hình ảnh (shape), chúng ta thêm vào một vài phương thức truy xuất thuộc tính. Có một phương thức truy xuất thuộc tính ChildForm đưa văn bản text vào trong thanh trạng thái (status bar) của MainForm. Điều này rất có ích cho người sử dụng. Đoạn mã sau chứa đựng những phương thức truy xuất thuộc tính lớp

```
public Tool CurrentTool
{
    get{return currentTool;}
}

public ShapeProperties ShapeProperties
{
    get{return shapeProperties;}
}

public EraserProperties EraserProperties
{
    get{return eraserProperties;}
}

public PaintBrushProperties PaintBrushProperties
{
    get{return paintbrushProperties;}
}

public string StatusText
{
    get{return this.statusStrip1.Text = value;}
}
```

Chạy ứng dụng sẽ cho bạn kiểm tra các nút công cụ tool được chọn và hiển thị các thuộc tính.

Bắt đầu vẽ ảnh

Nào, chúng ta sẽ bắt đầu thao tác vẽ ảnh. Tất cả điều này được thực hiện bởi bộ xử lý trong lớp ChildForm.

Lớp này thực hiện hai thao tác cơ bản. Thao tác đầu tiên là đưa một hìn đốm màu xuất hiện bất cứ khi nào nút chuột bị nhấn. Thao tác còn lại là cho phép người sử dụng đưa vào dạng hình học cần vẽ và chỉnh lại kích thước của nó.

Ví dụ 3.6-10 hiển thị đầy đủ tài nguyên của lớp ChildForm.

**Ví dụ 3.6-10 Lớp ChildForm**

```
1: using System;
2: using System.Drawing;
3: using System.Drawing.Drawing2D;
4: using System.Collections;
5: using System.ComponentModel;
6: using System.Windows.Forms;
7:
8: namespace FormPaint
9: {
10:    /// <summary>
11:    /// Summary description for ChildForm.
12:    /// </summary>
13:    public class ChildForm : System.Windows.Forms.Form
14:    {
15:        private Image myImage;
16:        private Bitmap tempBM;
17:
18:        private Pen pen;
19:        private SolidBrush brush;
20:        private GraphicsPath path;
21:        private Point firstPoint;
22:        private Point lastPoint;
23:        private Size blitBounds;
24:        private Point blitPos;
25:
26:        private bool Drawing;
27:        private bool dirty;
28:        private bool fill;
29:        private bool stroke;
30:
31:        private Graphics myGraphics;
32:        private Graphics imageGraphics;
33:
34:
35:        public Image Image
36:        {
37:            set
38:            {
39:                myImage = value;
40:                this.AutoScrollMinSize = myImage.Size;
41:                tempBM = new Bitmap(myImage.Size.Width,
42:                                    myImage.Size.Height);
42:            }
43:            get{return myImage;}
44:        }
}
```



Những biến được khai báo trên dòng từ 15 đến 32, theo sau là phương thức truy xuất Image, phương thức này gán một ảnh vào form con và khởi động lại giá trị AutoScroll để vừa với kích thước ảnh. Phương thức truy xuất này cũng cho phép những phương thức khác lấy ảnh chứa trong form.

Ví dụ 3.6-10 Tiếp theo

```
45:  
46: public bool Dirty  
47: {  
48:     get{return dirty;}  
49:     set{dirty = value;}  
50: }  
51:  
52: /// <summary>  
53: /// Required designer variable.  
54: /// </summary>  
55: private System.ComponentModel.Container  
      components = null;  
56:  
57:  
  
58: public ChildForm()  
59: {  
60:     //  
61:     // Yêu cầu cho Windows Forms  
62:     //  
63:     InitializeComponent được gọi  
64:     //  
65:     // Khởi động biến  
66:     //  
67:     Drawing = false ;  
68:     Dirty = false;  
69:     Text = "Untitled.bmp";  
70: }  
71:  
72:
```

Phương thức khởi động trên dòng từ 58 tới 71 gọi phương thức quan trọng InitializeComponent đặt những giá trị khởi động cho các cờ dùng để xác định nếu việc vẽ bị thay đổi hoặc hành động vẽ đang xảy ra, cùng với đặt thuộc tính Text của form bằng untitled.bmp cho trong trường hợp tạo một hình mới không cần tải từ đĩa.

```
73: /// <summary>  
74: /// Clean up any resource being used.  
75: /// </summary>  
76: protected override void Dispose(bool disposing)  
77: {
```

```
78:     if(disposing)
79:     {
80:         tempBM.Dispose();
81:         if(components != null)
82:         {
83:             components.Dispose();
84:         }
85:     }
86:     base.Dispose(disposing);
87: }
88:
89: #region Windows Form Designer generated code
90: /// <summary>
91: /// Required method for Designer support - do not
92: /// modify
93: /// the contents of this method with the code
94: /// editor.
95: ///
96: ///
97: //ChildForm
98: ///
99: this.AutoScaleBaseSize = new
100: System.Drawing.Size(5,13);
101: this.BackColor = System.Drawing.Color.Green;
102: this.ClientSize = new
103: System.Drawing.Size(808,565);
104: this.Name = "ChildForm";
105: this.Text = "ChildForm";
106: this.MouseDown += new
107: System.Windows.Forms.MouseEventHandler(
108:     this.OnMouseDown);
109: this.Closing += new
110: System.ComponentModel.CancelEventHandler(
111:     this.ChildForm_Closing);
112: this.MouseUp += new
113: System.Windows.Forms.MouseEventHandler(
114:     this.OnMouseUp);
115: this.Paint += new
116: System.Windows.Forms.PaintEventHandler(
117:     this.ChildForm_Paint);
118: this.MouseMove += new
119: System.Windows.Forms.MouseEventHandler(
120:     this.OnMouseMove);
121: }
122: #endregion
```



Phương thức Dispose và phương thức InitializeComponent được thêm vào bởi IDE. Phương thức Dispose phải dọn dẹp ảnh bitmap trung gian. Bộ xử lý được thêm vào ChildForm trên dòng 104 tới 108. Trên dòng 113 tới 116 sau thủ tục ToRadians đơn giản dùng đổi từ độ sang radian sử dụng cho những thủ tục toán học.

```
112:  
113: private double ToRadians(double angle)  
114: {  
115:     return angle/180*Math.PI;  
116: }  
117:  
118: private void CreateBrushPath(Shape s)  
119: {  
120:  
121:     path = new GraphicsPath();  
122:  
123:     MainForm form = (MainForm)this.ParentForm;  
124:     int Toolsize = 3;  
  
125:     switch(form.CurrentTool)  
126:     {  
127:         case Tool.Paintbrush:  
128:             Toolsize = form.PaintBrushProperties.Size;  
129:             break;  
130:         case Tool.Eraser:  
131:             Toolsize = form.EraserProperties.Size;  
132:             break;  
133:     }  
134:  
135:     if(Toolsize<3)  
136:         Toolsize = 3;  
137:     if(Toolsize>100)  
138:         Toolsize = 100;  
139:  
140:     switch(s)  
141:     {  
142:         case Shape.Round:  
143:             path.AddEllipse(-Toolsize/2, -Toolsize/2,  
144:                             Toolsize,Toolsize);  
145:             break;  
146:         case Shape.Square:  
147:             path.AddRectangle(new Rectangle(Toolsize/2,  
148:                             -Toolsize/2, Toolsize,Toolsize));  
149:             break;  
150:         case Shape.Triangle:  
151:             Point[] points = new Point[3];
```

```
152:     points[0] = new  
153:Point((int)(Math.Cos(ToRadians(90))*Toolsize),  
        (int)(Math.Sin(ToRadians(90))*Toolsize));  
154:     points[1] = new  
155:Point((int)(Math.Cos(ToRadians(210))*Toolsize),  
        (int)(Math.Sin(ToRadians(210))*Toolsize));  
156:     points[2] = new  
157:Point((int)(Math.Cos(ToRadians(330))*Toolsize),  
        (int)(Math.Sin(ToRadians(330))*Toolsize));  
158:     path.AddEllipse(points);  
159:     break;  
160: }  
161:}  
162:
```

Phương thức CreateBrushPath phát sinh ra đối tượng Path dùng để tô màu vùng ảnh. Trong ví dụ đơn giản này, màu sắc có thể áp dụng cho hình vuông, hình tam giác. Câu lệnh switch và những câu lệnh case của trên những dòng 140 – 161 quản lý việc tạo ra đường đồ thị Path. Path được lưu lại cho mỗi lần sử dụng khi nút nhấn chuột được kích hoạt. Trên dòng từ 163 tới 171, thủ tục Paint không làm gì hết ngoài việc chép ảnh nền vào màn hình chính.



```
187:     MainForm form = (MainForm)this.MdiParent;
188:     switch(form.CurrentTool)
189:     {
190:         case Tool.Paintbrush:
191:             brushColor = Color.FromArgb(
192:                 255-(int)(255.0/
193:                     100 *
194:                     form.PaintBrushProperties.Transparency) ,
195:                     form.PaintBrushProperties.Color);
196:             brush = new SolidBrush(brushColor);
197:             CreateBrushPath(
198:                 form.PaintBrushProperties.Shape);
199:             break;
200:         case Tool.Eraser:
201:             brush = new SolidBrush(Color.white);
202:             CreateBrushPath(
203:                 form.PaintBrushProperties.Shape);
204:             break;
205:         case Tool.Rectangle:
206:             goto case Tool.Ellipse;
207:         case Tool.Ellipse:
208:             fill = form.ShapeProperties.Fill;
209:             stroke = form.ShapeProperties.Line;
210:             firstPoint = new Point(e.X,e.Y);
211:             lastPoint = new firstPoint;
212:             pen = new
213:                 Pen(form.ShapeProperties.LineColor,
214:                     form.ShapeProperties.LineWidth);
215:             brush = new
216:                 SolidBrush(form.ShapeProperties.FillColor);
217:             break;
218:         }
219:     }
```

Bộ xử lý OnMouseDown là hành động đầu tiên trong tiến trình vẽ. Nó xử lý 2 trường hợp chính. Nếu nút chuột được nhấn và công cụ vẽ (paint tool) hay công cụ gôm (eraser tool) được chọn, nó tạo ra đường cọ vẽ mới chính xác với kích thước và dạng hình vẽ, sau đó đặt màu cho cọ vẽ và độ trong suốt (transparency). Trong trường hợp của công cụ hình (tool shape) nó lưu điểm đầu tiên trong hình, thường là điểm ở góc trái cao nhất, và sau đó định nghĩa màu cọ và bề rộng cọ. Cuối cùng, Phương thức này gọi bộ xử lý di chuyển chuột (mouse move) một lần nữa để đảm bảo việc vẽ lần đầu tiên được áp dụng vào vùng vẽ.

Trên các dòng từ 221 tới 257, bộ xử lý chuột thả ra (mouse up) chỉ có liên quan với việc vẽ ra hình ảnh mong đợi cuối cùng – hoặc là hình chữ nhật hoặc là hình Elip – trên bức vẽ. Bút vẽ hay gồm màu được áp dụng trên mọi sự kiện di chuyển chuột. Việc vẽ này không những thực hiện trên màn hình mà còn trên hình ảnh được giữ trong bộ nhớ bởi form. Hình này không bao giờ bị cuộn, vì vậy vị trí của chuột và thanh cuộn phải bù cho nhau trên những dòng 229 – 232.

```
220:
221: private void OnMouseUp(object sender,
222:     System.Windows.Forms.MouseEventArgs e)
223: {
224:
225:     if(this.MdiParent.ActiveMdiChild == this)
226:     {
227:         Point topLeft;
228:         Size bounds;
229:         topLeft = new
230:             Point(Math.Min(this.firstPoint.X,e.X),
231:                   Math.Min(this.firstPoint.Y,e.Y));
232:         bounds = new Size(Math.Abs(e.X-firstPoint.X),
233:                           Math.Abs(e.Y-firstPoint.Y));
234:         topLeft.Offset(-AutoScrollPosition.X,
235:                         -AutoScrollPosition.Y);
236:         MainForm form = (MainForm)this.MdiParent;
237:         switch(form.CurrentTool)
238:         {
239:             case Tool.Rectangle:
240:                 if(fill)
241:                     imageGraphics.FillRectangle(brush,
242:                                                 topLeft.X,topLeft.Y,
243:                                                 bounds.Width,bounds.Height);
244:                 if(stroke)
245:                     imageGraphics.DrawRectangle(pen,
246:                                                 topLeft.X,topLeft.Y,
247:                                                 bounds.Width,bounds.Height);
248:                 break;
249:             case Tool.Ellipse:
250:                 if(fill)
251:                     imageGraphics.FillEllipse(brush,
252:                                                 topLeft.X,topLeft.Y,
253:                                                 bounds.Width,bounds.Height);
254:                 if(stroke)
255:                     imageGraphics.DrawEllipse(pen,topLeft.X,
```

```
256: Invalidate();
257: }
258:
259: private void OnMouseMove(object sender,
260:                         System.Windows.Forms.MouseEventArgs e)
261: {
262:     if(!Drawing)
263:         return;
264:     if(this.MdiParent.ActiveMdiChild == this)
265:     {
266:         Graphics gTemp = Graphics.FromImage(tempBM);
267:         MainForm form = (MainForm)this.MdiParent;
268:         if(form.CurrentTool == Tool.Ellipse ||
269:             form.CurrentTool == Tool.Rectangle)
270:         {
271:             blitPos = new Point(
272:                 Math.Min(Math.Min(e.X,
273:                                 firstPoint.X),lastPoint.X),
274:                 Math.Min(Math.Min(e.Y,
275:                                 firstPoint.Y),lastPoint.Y));
276:             blitBounds = new Size(Math.Max(e.X,
277:                                         Math.Max(firstPoint.X,lastPoint.X))
278:                                     -blitPos.X,
279:                                     Math.Max(e.Y,Math.Max(firstPoint.Y,
280:                                                 lastPoint.Y))-blitPos.Y);
281:             blitPos.Offset(-(int)(1+pen.Width/2),
282:                             -(int)(1+pen.Width/2));
283:             blitBounds.Width += (int)(2+pen.Width);
284:             blitBounds.Height += (int)(2+pen.Width);
285:             form.StatusText = blitPos.ToString() + " " +
286:             blitBounds.ToString();
287:         }
288:         switch(form.CurrentTool)
289:         {
290:             case Tool.Paintbrush:
291:                 GraphicsContainer ctr =
292:                     myGraphics.BeginContainer();
293:                 myGraphics.Transform.Reset();
294:                 myGraphics.TranslateTransform(e.X,e.Y);
295:                 myGraphics.FillPath(this.brush,this.path);
296:                 myGraphics.EndContainer(ctr);
297:                 ctr = imageGraphics.BeginContainer();
298:                 imageGraphics.Transform.Reset();
299:                 imageGraphics.TranslateTransform(e.X-
300:                     AutoScrollPosition.X, e.Y-
301:                     AutoScrollPosition.Y);
302:                 imageGraphics.FillPath(this.brush,
303:                     this.path);
304:         }
305:     }
306: }
```

```
296:         imageGraphics.EndContainer(ctr);
297:
298:         break;
299:     case Tool.Eraser:
300:         goto case Tool.Paintbrush;
301:     case Tool.Rectangle:
302:         gTemp.DrawImage(myImage,
303:             new Rectangle(blitPos,blitBounds),
304:             blitPos.X - AutoScrollposition.X,
305:             blitPos.Y - AutoScrollposition.Y,
306:             blitBounds.Width,blitBounds.Height,
307:             GraphicsUnit.Pixel);
308:     if(fill)
309:         gTemp.FillRectangle(brush,
310:             Math.Min(this.firstPoint.X,e.X),
311:             Math.Min(this.firstPoint.Y,e.Y),
312:             Math.Abs(e.X-firstPoint.X),
313:             Math.Abs(e.Y-firstPoint.Y));
314:     if(stroke)
315:         gTemp.DrawRectangle(pen,
316:             Math.Min(this.firstPoint.X,e.X),
317:             Math.Min(this.firstPoint.Y,e.Y),
318:             Math.Abs(e.X-firstPoint.X),
319:             Math.Abs(e.Y-firstPoint.Y));
320:     myGraphics.DrawImage(tempBM,
321:             new Rectangle(blitPos,blitBounds),
322:             blitPos.X,blitPos.Y,
323:             blitBounds.Width,
324:             blitBounds.Height,
325:             GraphicsUnit.Pixel);
326:     break;
327:     case Tool.Ellipse:
328:         gTemp.DrawImage(myImage,
329:             new Rectangle(blitPos,blitBounds),
330:             blitPos.X - AutoScrollposition.X,
331:             blitPos.Y - AutoScrollposition.Y,
332:             blitBounds.Width,
333:             blitBounds.Height,
334:             GraphicsUnit.Pixel);
335:     if(fill)
336:         gTemp.FillEllipse(brush,
337:             Math.Min(this.firstPoint.X,e.X),
338:             Math.Min(this.firstPoint.Y,e.Y),
339:             Math.Abs(e.X-firstPoint.X),
340:             Math.Abs(e.Y-firstPoint.Y));
341:     if(stroke)
342:         gTemp.DrawEllipse(pen,
343:             Math.Min(this.firstPoint.X,e.X),
```

```
344:         Math.Min(this.firstPoint.Y, e.Y),
345:         Math.Abs(e.X-firstPoint.X),
346:         Math.Abs(e.Y-firstPoint.Y));
347:     myGraphics.DrawImage(tempBM,
348:         new Rectangle(blitPos,blitBounds),
349:         blitPos.X,blitPos.Y,
350:         blitBounds.Width,
351:         blitBounds.Height,
352:         GraphicsUnit.Pixel);
353:     break;
354: }
355: lastPoint.X = e.X;
356: lastPoint.Y = e.Y;
357: }
358: }
```

Phương thức di chuyển chuột OnMouseMove trên những dòng 259 tới 358 là một thủ tục làm nhiều việc trong chương trình. Nó làm việc trong hai chế độ (mode), sự khác nhau đó là giữa bút vẽ (paintbrush), những thao tác tẩy xóa và thao tác vẽ hình. Trong chế độ thứ nhất, việc vẽ được áp dụng vào ảnh bitmap sử dụng GraphicsPath tạo ra trong sự kiện OnMouseDown. Màu sắc được đưa vào cả bộ nhớ màn hình và trên ảnh nội bộ. Có nghĩa là thao tác vẽ sẽ xảy ra và tác động đến ảnh (image) ngay tức khắc. Điều này diễn ra trên dòng 285 tới 300 với dòng 289 thực hiện vẽ trên màn hình và dòng 295 đặt màu trên ảnh nền.

Những thao tác diễn ra trong quá trình vẽ hình khá phức tạp. Khi một hình được vẽ, người sử dụng muốn kéo giãn hình xung quanh để lấy kích thước chính xác. Điều đó có nghĩa là việc vẽ không thể đưa vào một cách trực tiếp trên ảnh nền. Kỹ thuật vùng đệm kép (double buffer) sử dụng ảnh trung gian được dùng để ghép một vùng từ màn hình với ảnh đồ họa. Ảnh này sau đó được chép vào màn hình, và tiến trình được lập lại cho đến khi nút nhấn chuột được thả ra. Kỹ thuật này sẽ tránh được nguyên nhân chớp giật bởi việc làm tươi ảnh và vẽ lại ảnh trực tiếp trên màn hình.

Dòng 269 kiểm tra xem thao tác vẽ hình chữ nhật Rectangle hay thao tác vẽ hình Ellipse đang thực hiện. Nếu điều đó đúng, thì kích thước lớn nhất của vùng màn hình bị ảnh hưởng, lưu ý tới vị trí bắt đầu, vị trí hiện hành, và vị trí trước đó của chuột trên những dòng 271 và 274. Vùng vẽ được tăng lên bằng phân nửa bề rộng đường vẽ xung quanh để bù vào bề dày của bút vẽ hiện hành trên những dòng 278 tới 280 và sau đó kỹ thuật bộ đệm kép sẽ vẽ những hình chữ nhật (trên các dòng 301 tới 326) và hình Ellipse (trên những dòng 327 tới 353).

Bạn cũng chú ý dòng 281, cập nhật thanh trạng thái với vị trí và kích thước ảnh.

```
362: {
363:     if(Dirty)
364:     {
365:         DialogResult result = MessageBox.Show(this,
366:             "This file has changed, do you wish to save",
367:             "Save file",
368:             MessageBoxButtons.YesNoCancel,
369:             MessageBoxIcon.Question);
370:         switch(result)
371:         {
372:             case DialogResult.Cancel:
373:                 e.Cancel = true;
374:                 break;
375:             case DialogResult.No:
376:                 break;
377:             case DialogResult.Yes:
378:                 this.Image.Save(this.Text);
379:                 break;
380:         }
381:     }
382: }
383:
384: protected override void OnPaintBackground(
PaintEventArgs e)
385: {
386:     Region r = new Region(new
Rectangle(0,0,Image.Width,Image.Height));
387:     Region w = new Region(new
388: Rectangle(AutoScrollPosition.X,
AutoScrollPosition.Y,
389: ClientRectangle.Width,
390: ClientRectangle.Height));
391:     r.Complement(w);
392:     e.Graphics.FillRegion(new
SolidBrush(this.BackColor),r);
393: }
394:
395: protected override void OnSizeChanged(EventArgs e)
396: {
397:     Invalidate();
398:     base.OnSizeChanged(e);
399: }
400: }
401: }
```

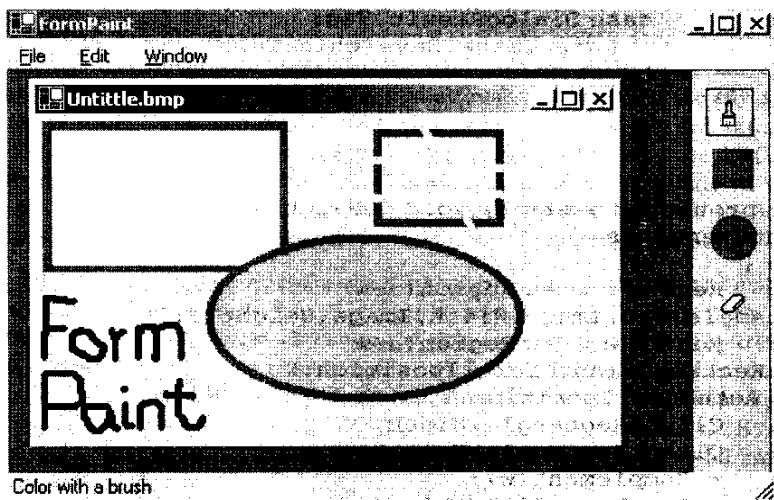
Cuối cùng là các phương thức liên quan đến việc lưu tập tin nếu tập tin này bị sửa đổi (dòng 360 tới 381). Bộ xử lý này được gọi trước khi form đóng lại. Hộp thông điệp được dùng ở dòng từ 365 tới 369 xác định nếu người sử dụng muốn lưu

hoặc bỏ hình. Trong trường hợp hủy bỏ (cancel) CancelEventArgs được cung cấp thông qua lời gọi delegate cần phải được cập nhật (dòng 373).

Phương thức định nghĩa chồng (override) OnPaintBackground được cung cấp để loại bỏ việc chớp giật. Với màu nền xanh lá cây và các thiết lập mặc định, toàn bộ ảnh bitmap được vẽ trên màu nền xanh lá cây trước khi ảnh được vẽ lại. Điều này sẽ khiến màn hình chớp giật khó chịu trừ phi bạn sử dụng phương pháp tính toán vùng màn hình bên ngoài của ảnh bằng cách sử dụng phần bù của khu vực này với khu vực được bao phủ bởi ảnh (dòng từ 386 tới 391) và chỉ vẽ lại mẫu yêu cầu.

Phương thức OnSizeChanged định nghĩa chồng phương thức lớp cơ sở trong các dòng từ 395 tới 399 bảo đảm màu nền được vẽ lại một cách chính xác.

Ảnh trong hình 3.6-10 hiển thị kết quả thực hiện của FormPaint.



Hình 3.6-10 Ứng dụng FormPaint

4. KẾT CHƯƠNG

Trong chương này, bạn đã sử dụng thuộc tính lưới (PropertyGrid). Học cách tạo ra ứng dụng Window Forms. Học cách nâng cao kinh nghiệm người dùng bởi những phần tử điều khiển tùy biến. Hy vọng rằng chương này sẽ là điểm bắt đầu để bạn phát triển hệ thống Window Forms, và bắt đầu viết ra những ứng dụng mạnh mẽ trên nền .NET cho riêng mình.

Phân IV

KỸ THUẬT WEB

Trong phần này:

- ◆ ASP.NET
- ◆ Truy xuất dữ liệu .NET
- ◆ Web forms
- ◆ Dịch vụ Web

Chương 4.1

ASP.NET

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Web một hơi thở mới*
- ✓ *Những điều cần thiết cho ASP.NET*
- ✓ *Hello ASP.NET*
- ✓ *Thêm vào chút hương vị*

1. WEB MỘT HƠI THỞ MỚI

Suốt hai năm vừa rồi, việc chuyển các ứng dụng truyền thống desktop sang các ứng dụng Web đang trở thành một xu hướng nóng bỏng. Khả năng tạo lập và triển khai các ứng dụng Web đủ sức cung cấp cho người dùng những tiện nghi và công cụ như các ứng dụng desktop không phải là một chuyện đơn giản. Khi có một số kỹ thuật mới nổi lên, ví dụ như kỹ thuật viết kịch bản script cao cấp kết hợp với DHTML, khả năng tạo lập một giao diện ứng dụng Web mạnh mẽ bắt đầu trở thành hiện thực. Người dùng không chỉ muốn dùng các chức năng của phần mềm mà họ còn muốn có một sản phẩm có tính mỹ thuật.

Lý do chính yếu nhất để chuyển dần các ứng dụng sang giao diện Web là việc dễ dàng trong triển khai ứng dụng. Hãy hình dung đến một công ty có hơn 10.000 nhân viên khắp thế giới. Bạn có muốn cài đặt chương trình phần mềm mới nhất trên hơn 10.000 cái máy tính? Không! Bằng cách dịch chuyển ứng dụng sang hướng phát triển và triển khai theo mô hình intranet/extranet, không còn một lý do nào để bạn phải cài đặt phần mềm trên máy tính của người dùng cuối. Thay đổi trên ứng dụng ngay lập tức có hiệu lực trên tất cả các máy tính của người dùng cuối.

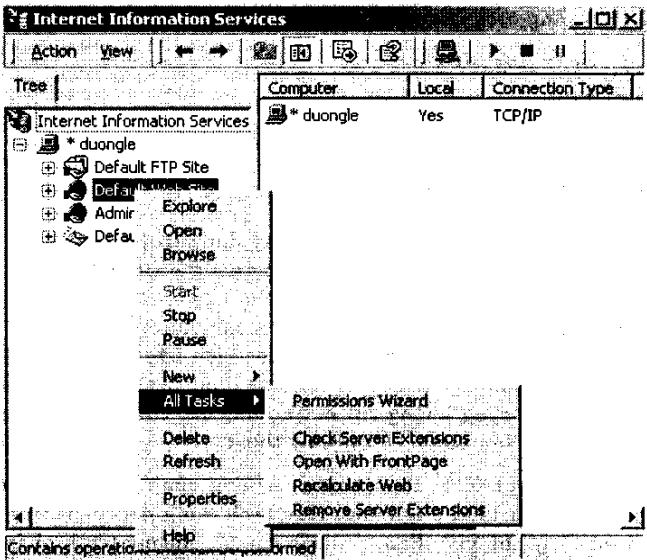
Hạn chế lớn nhất của mô hình phát triển ứng dụng Web điển hình là hiệu suất (performance). Bởi vì các trang ASP (Active Server Page) chưa bao giờ được biên dịch nên các Web server luôn phải thông dịch từng dòng mã ASP. Với sự phát triển của ASP.NET, các trang và các thành phần ASP.NET đã được biên dịch, do đó có thể chạy ở tốc độ rất nhanh. ASP.NET cho phép lập trình viên sử dụng bất cứ ngôn ngữ .NET nào như: C#, VB.NET, Managed C++ và thậm chí cả Cobol.

2. YÊU CẦU CẦN THIẾT CHO ASP.NET

Để chạy các trang ASP.NET, IIS cần phải được cấu hình hợp lý với các phần mở rộng (extension) của FrontPage. Suốt quá trình cài đặt .NET SDK hay Visual Studio .NET, các phiên bản cần thiết của phần FrontPage mở rộng cũng sẽ

được cài đặt. Để kiểm tra lại, bạn mở trình quản lý IIS Manager, nhấn chuột trên Website, thông thường là Website mặc định, chọn Check Server Extens trên menu All Task. Menu All Task sẽ được hiển thị mỗi khi ta nhấp chuột trên server cục bộ (local). Nó sẽ hiện ra một menu ngữ cảnh. Hình 4.1-1 sẽ thiệu menu ngữ cảnh.

Với IIS đã được cấu hình đúng, phần tiếp theo phải làm là tạo ra chương trình thử “Hello ASP.NET”.

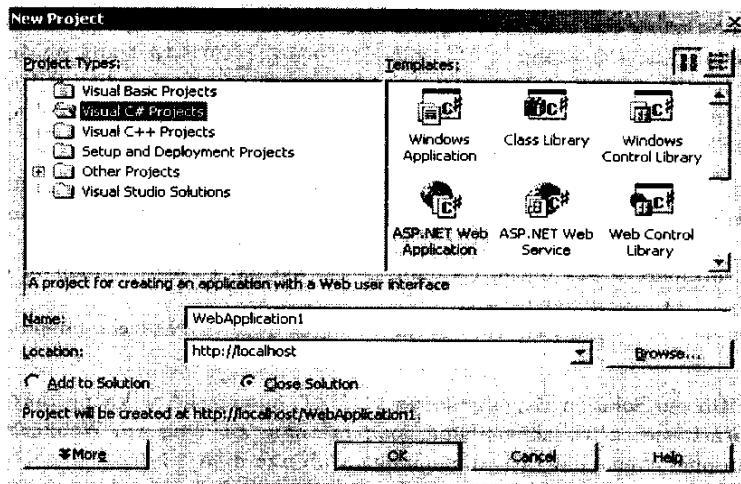


Hình 4.1-1 IIS Configuration Manager

3. HELLO ASP.NET

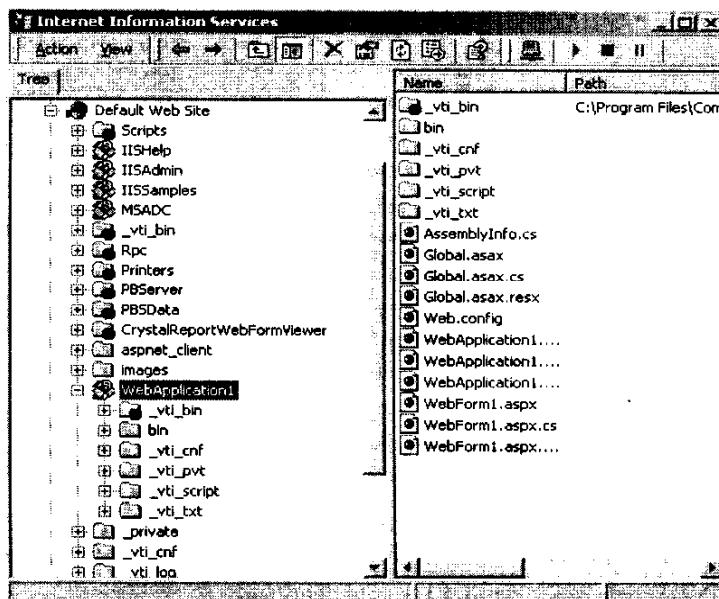
Khi xây dựng một ứng dụng ASP.NET, ta sẽ thấy Visual Studio .NET (VS.NET) là đáng đồng tiền bát gạo. Với VS.NET, tất cả các lập trình viên đều chung một IDE thân thiện và đơn giản. Tất cả những gì cần cho việc phát triển VB.NET, C#, VC++, FoxPro, và dĩ nhiên là cả ASP.NET, đều được tích hợp trợ trên một IDE duy nhất. Thêm nữa, Intellisense bây giờ đã có sẵn cho người dùng ASP.NET.

Để tạo ra một ví dụ Hello ASP.NET, hãy khởi động và bắt đầu xây dựng một ứng dụng Web mới. Hình 4.1-2 minh họa hộp thoại New Project với tên tương của “ASP.NET Web Application” được chọn. Hộp thoại cung cấp tên định danh của dự án “WebApplication”. Hãy giữ nguyên tên này hiện tại ta chưa cần đổi tên. Khi được phát sinh (generate), tên của dự án sẽ là WebApplication1.



Hình 4.1-2 Hộp thoại VS.NET New Project.

Sau khi VS.NET tạo ra một dự án Web mới, bạn nên mở trình quản lý IIS để xem cấu trúc và các tập tin được tạo ra bởi VS.NET. Hình 4.1-3 sẽ cho thấy IIS Manager Explorer.

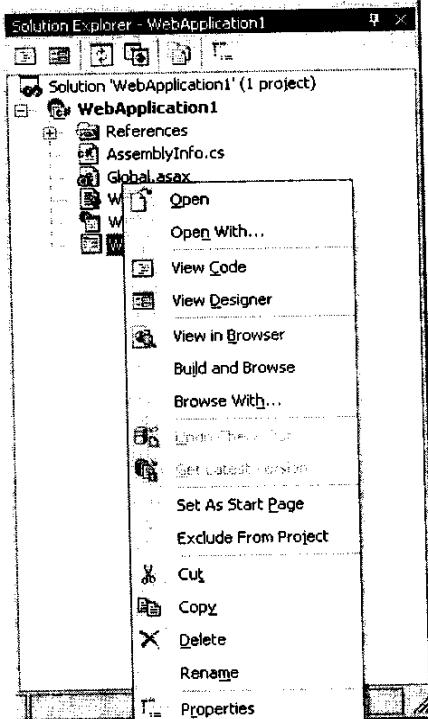


Hình 4.1-3 IIS Manager Explorer.

Bên trong thư mục WebApplication1, bạn sẽ thấy có tập tin *.csproj cùng với *.aspx, *.cs. Các trang ASPX chứa hai phần, HTML và trang nguồn, trong trường hợp của chúng ta, là tập tin .cs. Sự phân cách giữa giao diện (UI) và logic nghiệp vụ là cần thiết cho sự phát triển của tất cả các ứng dụng và đội ngũ phát triển ASP.NET.

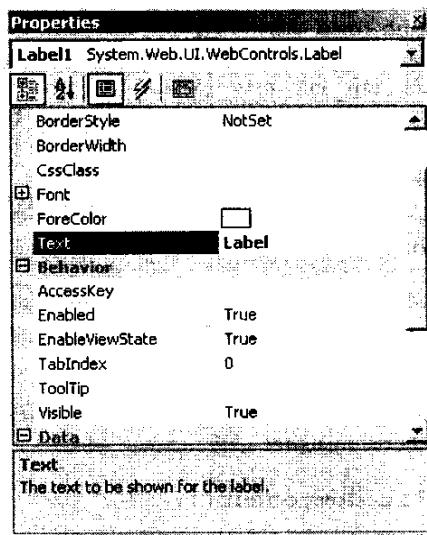
- Đối với các tập tin được tạo ra cho mỗi ứng dụng ASP.NET, nên chú ý:
- WebForm1.aspx Trang ASP sẽ được yêu cầu bởi trình duyệt của phía máy khách.
 - WebForm1.aspx.cs Mã nguồn C# dùng để cài đặt bất cứ logic nghiệp vụ nào cho trang aspx.
 - Web.config Một tập tin cấu hình bằng XML. ASP.NET cho phép bạn quản trị dễ dàng bằng XML.

Khi mở Solution Explorer trong VS.NET, bạn có thể sẽ không thấy tập tin WebForm1.aspx.cs. Để thấy những đoạn mã nằm ẩn dưới sau WebForm1.aspx, bạn cần phải chọn View Code ở menu ngữ cảnh như hình 4.1-4.



Hình 4.1-4 Solution Explorer

Để tiếp tục, hãy đặt một điều khiển nhãn vào cửa sổ thiết kế WebForm1.aspx và thiết lập các thuộc tính như hình 4.1-5.



Hình 4.1-5 Thuộc tính nhãn

Để xem các mã HTML, chọn View HTML Source từ menu ngữ cảnh.

```

1: <%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
AutoEventWireup="false"
Inherits="WebApplication.WebForm1" %>
2: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
3: <html>
4: <head>
5:   <meta name="GENERATOR" Content="Microsoft Visual
Studio 7.0">
6:   <meta name="CODE_LANGUAGE" Content="C#">
7:   <meta name="vs_defaultClientScript"
Content="JavaScript (ECMAScript)">
8:   <meta name="vs_targetSkelma"
Content="http://schemas.microsoft.com/intellisense/ie5
">
9: </head>
10:  <body MS_POSITIONING="GridLayout">
11:    <form id="Form1" method="post" runat="server">
12:      <asp:Label id=Label1 style="Z-INDEX: 101; LEFT:
33px; POSITION: absolute; TOP: 108px" runat="server">
Hello ASP.NET</asp:Label>
13:    </form>
14:  </body>
15: </html>

```

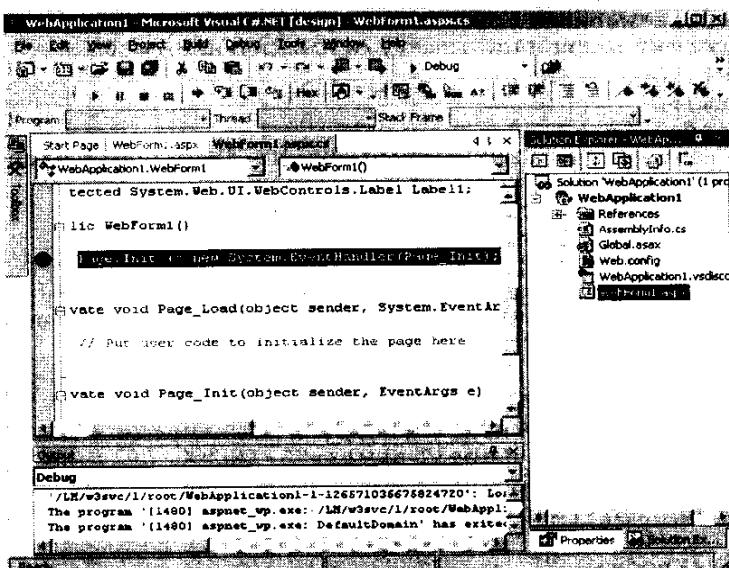


Với các thuộc tính nhẫn được thiết lập, hãy dịch và chạy thử dự án này bằng cách nhấn F5. Lần đầu tiên khi trang aspx được yêu cầu triệu gọi từ IIS, nó sẽ được dịch ra một tập mã máy (assembly), trong trường hợp này là WebApplication1.dll, nó được tạo ra và đặt vào thư mục bin. Trong những lần được yêu cầu sau, IIS sẽ kiểm tra xem dll này đã được cập nhật hay chưa và sẽ tự biên dịch lại nếu mã nguồn chương trình có sự thay đổi.

Lợi ích lớn nhất của ASP.NET và VS.NET là khả năng gỡ rối (debug) một ứng dụng Web bằng trình gỡ rối VS.NET. Hãy xem đoạn mã sau:

```
1: protected void Page_Init(object sender,
                           EventArgs e)
2: {
3:     // 
4:     // CODEGEN: This call is required by the ASP+
// Windows Form Designer.
5:     // 
6:     InitializeComponent();
7: }
```

Đặt một điểm dừng (break point) tại InitializeComponent. Một điểm dừng có thể đặt đặt bằng cách nhấn phím F9. Với một điểm dừng được đặt, như ở hình 4.1-6, nhấn F5 để chạy dự án ứng dụng Web này và bạn sẽ thấy IE khởi động. Quyền điều khiển sau đó sẽ được chuyển cho trình gỡ rối VS.NET khi phương thức Page_Init được gọi và chạy đến điểm dừng ở dòng 6.



Hình 4.1-6 Gỡ rối ASP.NET

Lập trình viên đã từng quen thuộc với các phiên bản trước của Microsoft Visual Studio có thể thấy “tự nhiên như ở nhà” khi xây dựng ứng dụng ASP.NET. Trước khi có .NET và VS.NET, việc gỡ rối chương trình ASP giống như là “mò kim trong bóng tối”.

4. THÊM CHÚT HƯƠNG VỊ

Mục đích của ASP.NET là phát triển ứng dụng Web thật nhanh, Microsoft đã phát triển hơn 45 điều khiển (control) cho các nhà lập trình và phát triển ứng dụng Web. Các điều khiển này chạy ở phía-server (server-side) và tự động trả lời cho các yêu cầu từ phía client. Hầu hết các Website ngày nay đều hỗ trợ IE và Netscape, do đó các lập trình viên Web phải duy trì cùng lúc hai bộ mã tương thích với chúng. Với sự tiến bộ của các điều khiển phía-server, những khó khăn này sẽ không còn nữa.

ASP.NET hỗ trợ các điều khiển HTML chuẩn cũng như các điều khiển ASP.NET. Thực chất, chúng có thể sống chung và thậm chí có thể lồng vào nhau.

Một trong những mục đích chính của bất kỳ một ứng dụng nào là lấy dữ liệu và trình diễn dữ liệu một cách có ý nghĩa. Hãy thử nghĩ về một ứng dụng cục bộ được dùng để lấy thông tin về tuyển dụng. Các thông tin này sẽ được nhập vào bởi các quy trình HR và lưu trong cơ sở dữ liệu.

Để bắt đầu quy trình, một kiến thức nhỏ về WebForm là cần thiết. Các ứng dụng ASP bây giờ dùng kiểu Request/Response, ASP.NET cũng vậy. Hình thức này thông qua kiểu Form Post truyền thống.

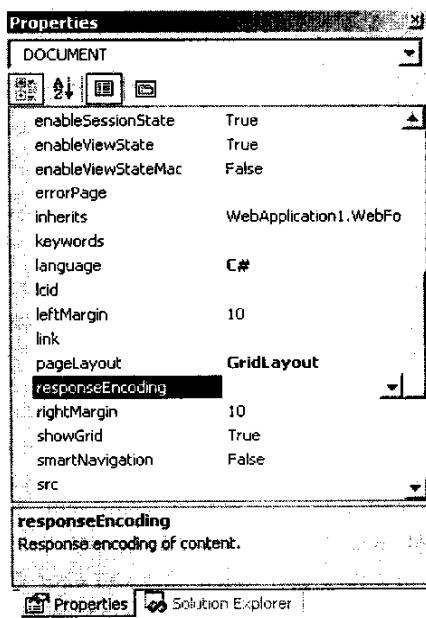
4.1. Kiểm tra ý tưởng

Cách hay nhất để học một ngôn ngữ/kiến trúc mới là thử xây dựng một ứng dụng có ích nào đó dùng càng nhiều kiến thức về ngôn ngữ càng tốt. Trong trường hợp của ASP.NET và kiến trúc .NET, nhiệm vụ này có thể thay đổi.

Để khám phá về ASP.NET, một cơ sở dữ liệu về nhân viên sẽ được dùng để minh họa. Không quan tâm các vấn đề về kỹ thuật ở đây. Chúng ta sẽ chỉ bàn về các điều khiển ASP, ADO.NET, các điều khiển Web, việc tải các tập tin lên server và những gì liên quan đến chúng.

Để bắt đầu, hãy tạo ra một ứng dụng có tên DataForm. Thuộc tính DOCUMENT .pageLayout nên để mặc định là GridLayout, nó sẽ cho phép ta định

vị trí tuyệt đối toạ độ x, y của các điều khiển. Ngoài ra, thuộc tính targetSchema nên thiết lập thành IE 5.



Hình 4.1-7 Các thuộc tính Document

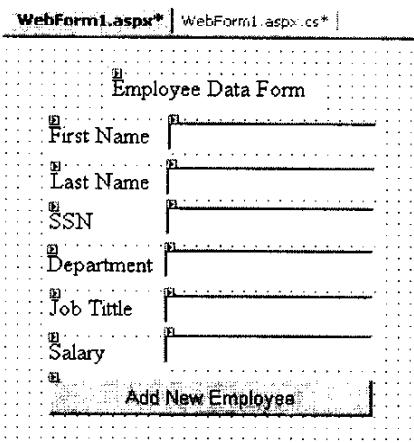
Sau đó, tạo khuôn mẫu như trong hình 4.1-8 và dùng các biến như tảng 4.1-1 sau:

Bảng 4.1-1 Biến cho trang DataForm.aspx

Kiểu Điều khiển	Tên biến	Nhận
asp:label	FirstNameLabel	First Name
asp:label	LastNameLabel	Last Name
asp:label	SSNLabel	SSN
asp:label	DepartmentLabel	Department
asp:label	JobTitleTable	Job Title
asp:label	SalaryLabel	Salary
Asp:textbox	FirstName	
Asp:textbox	LastName	
Asp:textbox	SSN	

Asp:textbox	Department	
Asp:textbox	JobTitle	
Asp:textbox	Salary	
Asp:textbox	AddEmployee	Add New Employee

Giao diện của Form sẽ như hình 4.1-8



Hình 4.1-8 Khuôn mẫu của DataForm

Khi đã có khuôn mẫu, hãy thử xem các đoạn mã HTML được phát sinh như thế nào.

Ví dụ 4.1-1 Mã của DataForm.aspx

```

1:<%@ Page language="c#" Codebehind="DataForm.cs"
AutoEventWireup="false"
Inherits="DataForm.EmployeeForm" %>
2:<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
3:<html>
4:<head>
5:<meta name="GENERATOR" Content="Microsoft Visual
Studio 7.0">
6:<meta name="CODE_LANGUAGE" Content="C#">
7:<meta name="vs_defaultClientScript"
Content="JavaScript (ECMAScript)">
8:<meta name="vs_targetSchema"
Content="http://schemas.microsoft.com/intellisense/ie5
">
9:</head>
```

```
10: <body ms_positioning="GridLayout">
11:   <form id="Form1" method="post" runat="server">
12:     <asp:Label id=FirstNameLabel style="Z-INDEX: 102; LEFT: 80px; POSITION: absolute; TOP: 100px" runat="server" Width="100" Height="25">First Name</asp:Label>
13:     <asp:Label id=LastNameLabel style="Z-INDEX: 103; LEFT: 80px; POSITION: absolute; TOP: 130px" runat="server" Width="100" Height="25">Last Name</asp:Label>
14:     <asp:Label id=SSNLabel style="Z-INDEX: 104; LEFT: 80px; POSITION: absolute; TOP: 160px" runat="server" Width="100" Height="25">SSN</asp:Label>
15:     <asp:Label id=DepartmentLabel style="Z-INDEX: 105; LEFT: 80px; POSITION: absolute; TOP: 190px" runat="server" Width="100" Height="25">Department</asp:Label>
16:     <asp:Label id=JobTitleLabel style="Z-INDEX: 106; LEFT: 80px; POSITION: absolute; TOP: 220px" runat="server" Width="100" Height="25">Job Title</asp:Label>
17:     <asp:Label id=SalaryLabel style="Z-INDEX: 107; LEFT: 80px; POSITION: absolute; TOP: 250px" runat="server" Width="100" Height="25">Salary</asp:Label>
18:     <asp:textbox id=Salary style="Z-INDEX: 113; LEFT: 190px; POSITION: absolute; TOP: 250px" runat="Server" width="100" height="25"></asp:textbox>
19:     <asp:textbox id=JobTitle style="Z-INDEX: 112; LEFT: 190px; POSITION: absolute; TOP: 220px" runat="Server" width="100" height="25"></asp:textbox>
20:     <asp:textbox id=Dept style="Z-INDEX: 111; LEFT: 190px; POSITION: absolute; TOP: 190px" runat="Server" width="100" height="25"></asp:textbox>
21:     <asp:textbox id=SSN style="Z-INDEX: 110; LEFT: 190px; POSITION: absolute; TOP: 160px" runat="Server" width="100" height="25"></asp:textbox>
22:     <asp:textbox id=LastName style="Z-INDEX: 109; LEFT: 190px; POSITION: absolute; TOP: 130px" runat="Server" width="100" height="25"></asp:textbox>
23:     <asp:textbox id=FirstName style="Z-INDEX: 108; LEFT: 190px; POSITION: absolute; TOP: 100px" runat="Server" width="100" height="25"></asp:textbox>
24:     <asp:button id=AddEmployee text="Add New Employee" style="Z-INDEX: 114; LEFT: 80px; POSITION: absolute; TOP: 280px" runat="Server" width="200" height="30" />
25:
26:   <!-- Dành cho asp:Repeater -->
27: </body>
```



```
28: </form>
29: </html>
```

Windows Form Designer sinh ra những đoạn mã quái quỉ cực kỳ khó đọc đối với người thường, và WebForms cũng vậy. Chúng tôi đã phải cắt dán và gọt giũa các đoạn chương trình cho dễ đọc. Dạng HTML cho DataForm.aspx thực chất là XHTML. Lưu ý rằng ở dòng 26 chúng tôi đã thêm vào một ghi chú rằng điều khiển asp:repeater sẽ được thêm vào sau. Nó sẽ được dùng để hiển thị các nhân viên mới.

Đối tượng WebForm được dẫn xuất từ System.Web.UI.Page và trang DataForm.aspx được dẫn xuất từ trang DataForm.DataBehind, nó được đặt tên lại trong lớp WebForm1. Điều quan trọng bạn phải lưu ý rằng ASP.NET dường như có vấn đề khi làm việc với các lớp có cùng tên với không gian tên (namespace). Bởi vậy, thường thì người ta không đặt tên trùng với tên tập tin mà nó nằm trong đó. VS.NET thường phát sinh ra những tên chung chung thay vì nhắc và yêu cầu lập trình viên nhập vào. Thường phải tốn thời gian để đổi tên chúng lại!

Bên cạnh việc khai báo các biến trong trang aspx, trang DataForm.aspx.cs cũng chứa các khai báo biến. Nó cho phép ta truy nhập vào các thuộc tính và phương thức của các điều khiển trong quá trình được tải lên và trong quá trình xử lý các sự kiện của DataFormBehide. Nếu bạn đang thắc mắc xem chuyện gì sẽ xảy ra như thế nào thì bạn nên biết rằng ASP.NET đã phát sinh ra các đoạn mã chương trình khi được truy cập đến lần đầu tiên. Trong thời điểm đó, các dòng mã keo (glue-code) đã được sinh ra để kết nối các điều khiển aspx với các biến được khai báo trong trang CodeBehind. Xem lớp DataFormBehide ở ví dụ 4.1-2 để thấy rõ hơn cách thức chúng liên kết cấu trúc trang, ràng buộc dữ liệu (databinding), tạo khuôn mẫu, kiểm tra giá trị đầu vào, thực hiện các logic nghiệp vụ (business logic).

Ví dụ 4.1-2 Lớp DataFormBehide

```
1: using System;
2: using System.Collections;
3: using System.ComponentModel;
4: using System.Data;
5: using System.Drawing;
6: using System.Web;
7: using System.Web.SessionState;
8: using System.Web.UI;
9: using System.Web.UI.WebControls;
10: using System.Web.UI.HtmlControls;
11:
12: namespace DataForm
13: {
14:     /// <summary>
15:     /// Summary description for DataForm.
16:     /// </summary>
```

```
17:     public class DataFormBehind : System.Web.UI.Page
18:     {
19:         protected System.Web.UI.WebControls.Label
20:             FirstNameLabel;
21:         protected System.Web.UI.WebControls.Label
22:             LastNameLabel;
23:         protected System.Web.UI.WebControls.Label
24:             SSNLabel;
25:         protected System.Web.UI.WebControls.Label
26:             DepartmentLabel;
27:         protected System.Web.UI.WebControls.Label
28:             JobTitleLabel;
29:         protected System.Web.UI.WebControls.Label
30:             SalaryLabel;
31:         protected System.Web.UI.WebControls.Button
32:             AddEmployee;
33:         protected System.Web.UI.WebControls.TextBox
34:             FirstName;
35:         protected System.Web.UI.WebControls.TextBox
36:             LastName;
37:         protected System.Web.UI.WebControls.TextBox
38:             SSN;
39:         protected System.Web.UI.WebControls.TextBox
40:             Dept;
41:         protected System.Web.UI.WebControls.TextBox
42:             JobTitle;
43:         protected System.Web.UI.WebControls.TextBox
44:             Salary;
45:         protected System.Web.UI.WebControls.Repeater
46:             EmployeeRepeater;
47:     }
48:     public DataFormBehind()
49:     {
50:         Page.Init += new
51:             System.EventHandler(Page_Init);
52:     }
53:     private void Page_Load(object sender,
54:             System.EventArgs e)
55:     {
56:         if (!this.IsPostBack)
57:         {
58:             //Create an ArrayList to hold Employees
59:             ArrayList employees = new ArrayList();
60:             this.Session["employees"] = employees;
61:         }
62:     }
63:     else {
```

```
49:         //Since only one button exists on the form
50:         //it must have been clicked so add the
51:         //new employee and update the databinding
52:         Employee emp = new Employee();
53:         emp.FirstName = FirstName.Text;
54:         emp.FirstName = FirstName.Text;
55:         emp.SSN      = SSN.Text;
56:         emp.Dept     = Dept.Text;
57:         emp.JobTitle = JobTitle.Text;
58:         emp.Salary    = Double.Parse(Salary.Text);
59:
60:         ((ArrayList)this.Session["employees"]).Add(emp);
61:         EmployeeRepeater.DataSource
62: =((ArrayList)this.Session["employees"]);
63:         EmployeesRepeater.DataBind();
64:     }
65: }
66:
67: private void Page_Init(object sender,
68: {
69:     //
70:     // CODEGEN: This call is required by the
71:     //          ASP.NET Web Form Designer
72:     // InitializeComponent();
73: }
74:
75: #region Web Form Designer generated code
76: /// <summary>
77: /// Required method for Designer support do not
78: /// the contents of this method with the code
79: ///          modify
80: ///          editor.
81: /// </summary>
82: private void InitializeComponent()
83: {
84:     this.Load += new
85:             System.EventHandler(this.Page_Load);
86: }
87: }
```

Ở đây chúng ta đã thêm vài thay đổi vào lớp DataFormBehind. Đầu tiên khai báo biến EmployeeRepeater. Một Repeater cho phép mở rộng mẫu HTML (template) trong quá trình trang được xử lý. Trong ví dụ này, các dòng sẽ được thêm vào bảng HTML ứng với mỗi nhân viên. Lớp Repeater sẽ kết nối dữ liệu với các thuộc tính chung của đối tượng. Thay đổi DataForm.cs để phù hợp với nhữn thay đổi ở Ví dụ 4.1-2

Tiếp theo, ta dùng một lớp để trình diễn thông tin nhân viên. Từ bây giờ mỗi khi một nhân viên được thêm vào, anh ta sẽ được lưu trong một mảng database. Mảng này sẽ được sử dụng như nguồn dữ liệu (data source) cho điều khiển Repeater. Một lớp rất đơn giản sẽ được xây dựng như trong ví dụ 4.1-3 sau

Ví dụ 4.1-3 Lớp Employee

```

1: namespace DataForm
2: {
3:     using System;
4:
5:     /// <summary>
6:     /// Summary description for Employee.
7:     /// NOTE: Since the employee class will be used in
8:     /// conjunction with DataBinding, public properties
9:     /// are required. DataBinding makes use of the
10:    /// System.Reflection API to access the get/set Properties.
11:    public class Employee
12:    {
13:        private string _FirstName;
14:        private string _LastName;
15:        private string _SSN;
16:        private string _Dept;
17:        private string _JobTitle;
18:        private double _Salary;
19:
20:        public string FirstName {
21:            get { return _FirstName; }
22:            set { _FirstName = value; }
23:        }
24:        public string LastName {
25:            get { return _LastName; }
26:            set { _LastName = value; }
27:        }
28:        public string SSN {
29:            get { return _SSN; }
30:            set { _SSN = value; }
31:        }
32:        public string Dept {

```

```

33:         get { return _Dept; }
34:         set { _Dept = value; }
35:     }
36:     public string JobTitle {
37:         get { return _JobTitle; }
38:         set { _JobTitle = value; }
39:     }
40:     public double Salary {
41:         get { return _Salary; }
42:         set { _Salary = value; }
43:     }
44: }
45: }
46:

```

Mặc dù lớp này không thực sự làm gì với dữ liệu, nhưng nó minh họa cho ta thấy cách mà điều khiển Repeater làm việc. Bất cứ điều khiển nào sử dụng cơ chế ràng buộc dữ liệu (data binding) đều cần các thuộc tính của đối tượng bound.

Để sử dụng điều khiển Repeater, bạn thêm các dòng code ở ví dụ 4.1-4 vào tập tin DataForm.aspx

Ví dụ 4.1-4 Mã mẫu EmployeeRepeater

```

1: <asp:repeater id=EmployeeRepeater runat="Server">
2:   <HeaderTemplate>
3:     <table style="BORDER-RIGHT: lightgrey thin ridge;
   BORDER-TOP: lightgrey thin ridge; LEFT: 300px; BORDER-
   LEFT: lightgrey thin ridge; BORDER-BOTTOM: lightgrey
   thin ridge; POSITION: absolute; TOP: 100px">
4:       <tr style="FONT-SIZE: large; COLOR: purple;
   BACKGROUND-COLOR: lightgrey">
5:         <td>Name</td>
6:         <td>Department</td>
7:         <td>Job Title</td>
8:       </tr>
9:   </HeaderTemplate >
10:  <ItemTemplate>
11:    <tr>
12:      <td>
13:        <%# DataBinder.Eval( Container.DataItem,
   "LastName" ) %>,
14:        <%# DataBinder.Eval( Container.DataItem,
   "FirstName" ) %>
15:      </td>
16:      <td>
17:        <%# DataBinder.Eval( Container.DataItem,
   "Dept" ) %>

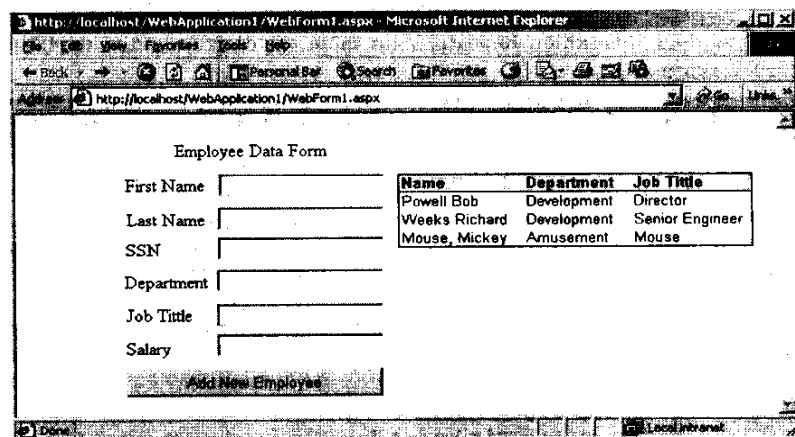
```

```

18:           </td>
19:           <td>
20:               <%# DataBinder.Eval( Container.DataItem,
21:                               "JobTitle") %>
22:           </td>
23:           </tr>
24:       </ItemTemplate>
25:       <FooterTemplate>
26:           </tbody> </table>
27:       </FooterTemplate>
28:   </asp:repeater>

```

Với sự bổ sung của mã `asp:repeater`, một dòng mới sẽ được thêm vào bản mỗi khi một nhân viên mới được tạo ra. Các lập trình viên C++ chắc hẳn đã quen thuộc với ý tưởng dùng mẫu template trong quá trình phát triển. Mẫu cho phép một kiến trúc tổng thể có thể được mở rộng khi cần. Bạn nên làm quen với `asp:repeater` bởi vì nó sẽ tiết kiệm cho bạn hàng giờ lập trình. Sản phẩm cuối sẽ nhìn giống như hình 4.1-9.



Hình 4.1-9 Trang minh họa ý tưởng cơ bản

Những tiền đề cơ bản của ứng dụng đã được thiết lập. Cảm giác ban đầu, việc phát triển ASP.NET sẽ để lại trong bạn những cảm xúc dễ chịu và sẵn sàng để tham gia vào một ứng dụng Web lớn hơn nữa.

5. KẾT CHƯƠNG

Chương này đã bao quát những điều cơ bản về phát triển Web sử dụng ASP.NET. Trong các chương tới, một cái nhìn sâu hơn về hầu hết các lĩnh vực Web sẽ được đề cập đến. Mỗi chủ đề chúng ta sẽ cố gắng xây dựng và tìm hiểu gắn liền với ứng dụng EmployeeBrowser.

Chương 4.2

TRUY XUẤT DỮ LIỆU TRÊN .NET

Các vấn đề chính sẽ được đề cập đến:

Chương này chúng ta sẽ học và bàn về cách truy xuất dữ liệu sử dụng mô hình .NET

✓ DataLayer

✓ Lớp Employee và lớp Department

Khi học một ngôn ngữ mới, một hệ điều hành hay một kiến trúc mới, cách tiếp cận tốt nhất là phát triển một ứng dụng và sử dụng càng nhiều tính năng có thể được. Khi thiết kế một ứng dụng, tiến trình tổng quát bao gồm việc tạo ra tài liệu chi tiết cùng với những những trường dữ liệu sử dụng. Thiết nghĩ một danh sách ngắn gọn sẽ đủ cho ví dụ này. Cơ sở dữ liệu của ứng dụng nhân sự (employee) của ta sẽ hỗ trợ những tính năng sau :

1. Tìm kiếm (Search).
 - a. Bảng phòng ban (Department.)
 - b. Bảng tên (Name)
 - c. Bảng tên bên trong Department.
2. Soạn thảo (Edit).
3. Thêm (Add).
4. Tải hình ảnh lên (Upload picture).

Với những yêu cầu trên chúng ta hoàn toàn có thể tiếp cận và thực hành với ASP.NET. Chúng ta sẽ tìm hiểu về thuộc tính, khả năng phản chiếu mã (reflection), các đoạn mã trang pagelet, và những đối tượng điều khiển (control) của ASP. Đây là toàn bộ mục tiêu mà chúng ta sẽ nghiên cứu ASP và .NET bên trong ngữ cảnh của một ứng dụng thực thụ.

1. TẦNG DỮ LIỆU (DATALAYER)

Cấu trúc dữ liệu cần thiết cho ứng dụng bao gồm hai bảng Department và Employee có quan hệ như hình 4.2-1.

The screenshot shows two tables defined in a database:

- DEPARTMENT** table:

Column Name	Data Type	Nullable
DEPT_ID	int	NOT NULL
NAME	varchar	NOT NULL
- EMPLOYEE** table:

Column Name	Data Type	Nullable
EMP_ID	char	NOT NULL
DEPT_ID	int	NOT NULL
FIRST_NAME	varchar	NOT NULL
LAST_NAME	varchar	NOT NULL
PIC_ID	uniqueidentifier	NULL

Hình 4.2-1 – Các bảng của cơ sở dữ liệu Stingray

Các tùy biến thuộc tính được thảo luận ngắn gọn trước đây giờ là thời điểm tốt để sử dụng chúng. Tầng dữ liệu sẽ định nghĩa hai thuộc tính là DBTableAttribute và DBFieldAttribute. Những thuộc tính này sẽ chứa đựng thông tin cần thiết để triệu hồi thủ tục nội tại stored procedure của cơ sở dữ liệu. Tập các hàm Reflective API có thể được sử dụng để xác định nội dung và cấu trúc thuộc tính của đối tượng.

Một lớp sẽ làm mô hình cho mọi thực thể trong cơ sở dữ liệu. Những lớp này sẽ được dẫn xuất từ lớp cơ sở trùu tượng chung là DBEntity. DBEntity định nghĩa 2 thuộc tính là IsDirty và IsNew cùng với phương thức trùu tượng FromDataRow. Thuộc tính IsDirty sẽ được sử dụng để quyết định nếu DBEntity cần được cập nhật trong cơ sở dữ liệu. Thuộc tính IsNew được sử dụng để quyết định xem là dữ liệu dùng để cập nhật hoặc chèn mới vào. Phương thức trùu tượng FromDataRow được dùng để xây dựng DBEntity từ phần tử DataRow của đối tượng DataSet. Ví dụ 4.2.1 cho thấy cách xây dựng lớp DBEntity.

Ví dụ 4.2.1 : Lớp cơ sở trùu tượng DBEntity.

```
1.  ///<summary>
2.  ///  DBEntity defines a basic database entity and
3.  ///  serves as the
4.  ///</summary>
5.
6.
7.  namespace Stingray.Data
8.  {
9.      using System;
10.     using System.Data;
11.
12.
13.
14.  /// <summary>
15.  /// The DBEntity abstract base class
16.  /// </summary>
17.  public abstract class DBEntity  {
18.
19.      private bool m_bIsDirty = false;
```

```

20.    private bool m_bIsNew = true;
21.
22.    public virtual bool IsDirty {
23.        get { return m_bIsDirty; }
24.        set { m_bIsDirty = value; }
25.    }
26.
27.    public virtual bool IsNew {
28.        get { return m_bIsNew; }
29.        set { m_bIsNew = value; }
30.    }
31.
32.    public abstract bool FromDataRow(
System.Data.DataRow data );
33. }
34.
35. }
```

Công việc tiếp theo là phải tạo ra và tùy biến thuộc tính của các lớp DBTableAttribute và DBFieldAttribute. DBTableAttribute sẽ chứa tên bảng, chèn tên của thủ tục nội stored procedure, và cập nhật tên của stored procedure. DBFieldAttribute định nghĩa tên đối số, kiểu dữ liệu, và chiều dài của dữ liệu nếu cần thiết. Những thuộc tính này sẽ được áp dụng cho từng đối tượng đại diện một thực thể bên trong cơ sở dữ liệu. Bằng cách sử dụng Reflection API, những thuộc tính này sẽ được dùng để xử lý những tác vụ như chèn và cập nhật các thực thể. Ví dụ 4.2-2 cho thấy cách cài đặt hai lớp DBTableAttribute và DBFieldAttribute.

Ví dụ 4.2.2 : Các lớp tùy biến thuộc tính

```

1: namespace Stingray.Data
2: {
3:     using System;
4:     using System.Data;
5:     using System.Data.SqlClient;
6:     using System.Data.SqlTypes;
7:
```

```
8:  
9:  
10: ///<summary>  
11: ///The class DBTableAttribute defines the table  
    within which  
12: ///a particular entity reside. In addition, this thuộc  
    tính  
13: ///also contains properties for the SQL Insert and  
    Update stored  
14: ///procedures used by the entity  
15: ///</summary>  
16: [attributeusage(AttributeTargets.Class)]  
17: public class DBTableAttribute : System.Thuộc tinh {  
18:  
19: /*****[Fields ]*****/  
20:  
21:     private string m_TableName;  
22:     private string m_SPIinsertCommand;  
23:     private string m_SPUupdateCommand;  
24:  
25:  
26: /*****[Properties ]*****/  
27:  
28:     public string TableName {  
29:         get { return m_TableName; }  
30:         set { m_TableName = value; }  
31:     }  
32:  
33:     public string InsertCommand {  
34:         get { return m_SPIinsertCommand; }  
35:         set { m_SPIinsertCommand = value; }  
36:     }  
37:
```

```
38:     public string UpdateCommand {
39:         get { return m_SPUpdateCommand; }
40:         set { m_SPUpdateCommand = value; }
41:     }
42:
43: /******[Constructor(s)]*****/
44:     public DBTableAttribute( string TableName ) {
45:         m_TableName = TableName;
46:     }
47: }
48:
49:
50:
51: ///<summary>
52: ///The DBFieldAttribute is used to map class
      properties
53: ///onto SQL stored procedure parameters.
54: ///</summary>
55: [attributeusage(AttributeTargets.Property)]
56:     public class DBFieldAttribute : System.Thuộc tính
      {
57:
58: /******[Fields]*****/
59:
60:     private string    m_ParamName;
61:     private SqlDbType m_DataType;
62:     private int       m_Length = 0;
63:
64:
65: /******[Properties]*****/
66:     public string ParameterName {
67:         get { return m_ParamName; }
```

```
68:         set { m_ParamName = value; }
69:     }
70:
71:     public SqlDbType DataType {
72:         get { return m_DataType; }
73:         set { m_DataType = value; }
74:     }
75:
76:     public int Length {
77:         get { return m_Length; }
78:         set { m_Length = value; }
79:     }
80:
81: /******[Constructor(s)]*****/
82:     public DBFieldAttribute( string ParameterName
83: ) {
84:     m_ParamName = ParameterName;
85: }
86:
87: }
```

DBTableAttribute được quy bằng AttributeUsage. Do DBTableAttribute chỉ sử dụng trong một lớp, AttributeTarget enum được sử dụng để chỉ ra điều này. AttributeUsage cho phép kiểm soát cách thuộc tính được sử dụng. AttributeTarget enum chứa những giá trị được định nghĩa sau:

```
public enum AttributeTargets {
    All,
    Assembly,
    Class,
    Constructor,
    Delegate,
    Enum,
```

```
Event,  
Field,  
Interface,  
Method,  
Module,  
Parameter,  
Property,  
ReturnValue,  
Struct  
}
```

Thực sự một thuộc tính không quan trọng lắm cho đến khi nó được cài đặt. **System.Attribute** phục vụ như lớp cơ sở, từ lớp cơ sở này có thể dẫn xuất tạo những lớp thuộc tính con do người dùng định nghĩa. Cũng như các lớp do người dùng tự định nghĩa khác, chi tiết, cách dùng và việc cài đặt nó sẽ tùy thuộc vào mục đích mà thuộc tính mới phục vụ.

Với lớp DBEntity, DBTableAttribute, và DBFieldAttribute, chúng ta có thể xây dựng lớp DBAccess. DBAccess sẽ sử dụng các hàm Reflection API để chèn hoặc cập nhật lớp dẫn xuất DBEntity. DBAccess chưa đựng 3 phương thức tĩnh (static). Phương thức AcquireConnection là mã chương trình cố định (hard-code) dùng trả về một kết nối tới cơ sở dữ liệu ở máy cục bộ (localhost), cùng với thông tin đăng nhập máy cần thiết. Bạn có thể cho phép lấy chuỗi kết nối từ tập tin cấu hình và sau đó lưu trữ cho lần truy cập gần nhất. Tuy nhiên, thỉnh thoảng đơn giản lại tốt hơn.

Lớp DBAccess cũng cung cấp phương thức Save chung. Phương thức này dùng để lưu các lớp dẫn xuất DBEntity. Ví dụ 4.2.3 chứa đựng danh sách tài nguyên cho lớp DBEntity.

Ví dụ 4.2.3 : DBAccess

```
1:  namespace Stingray.Data  
2:  {  
3:      using System;  
4:      using System.Reflection;  
5:      using System.Data;  
6:      using System.Data.SqlClient;  
7:
```

```
8:
9: public class DBAccess
10: {
11:
12: /// <summary>
13: /// Get an active connection to the stingray database
14: /// </summary>
15:     public static SqlConnection AquireConnection()
16:     {
17:         SqlConnection dbCon =
18:             new SqlConnection(
19:                 "user id=sa;password=;initial catalog=Stingray;
20:                  data source=.; Connect Timeout=30"
21:             );
22:         try {
23:             dbCon.Open();
24:         } catch( Exception ) {
25:             dbCon.Dispose();
26:             dbCon = null;
27:         }
28:     }
29:     /// <summary>
30:     /// Save a DBEntity into the Stingray Database
31:     /// </summary>
32:     /// <param name="ActiveConnection">
33:     /// Active Database Connection</param>
34:     /// <param name="entity"> The Entity to be
35:     /// saved</param>
```

```
36:     public static bool Save( SqlConnection
  ActiveConnection, DBEntity entity ) {
37:
38: //Is there anything to save?
39:     if( !entity.IsDirty )
40:         return true;
41:
42:     object[] Attributes = entity.GetType(
  ).GetCustomAttributes(
43:     typeof(Stingray.Data.DBTableAttribute),
44:     false
45: );
46:
47:     if(Attributes.Length != 1 )
48:         return false;
49:
50:     System.Data.ADO.SqlClient.SqlCommand Sqlcmd
  = new SqlCommand( );
51:     Sqlcmd.ActiveConnection = ActiveConnection;
52:     Sqlcmd.CommandType =
  System.Data.CommandType.StoredProcedure;
53:
54: /**
55: //Do we insert or Update?
56: /**
57:     DBTableAttribute TableAttribute =
  (DBTableAttribute)Attributes[0];
58:
59:     if( entity.IsNew ) {
60:         Sqlcmd.CommandText =
  TableAttribute.InsertCommand;
61:     } else {
62:         Sqlcmd.CommandText =
  TableAttribute.UpdateCommand;
```

```
63:         }
64:
65:         AddCmdParam( Sqlcmd, entity );
66:
67:         try {
68:
69:             Sqlcmd.ExecuteNonQuery();
70:             return true;
71:
72:         } catch( Exception ) {
73: //Do something for Heavens sake!!
74:     }
75:
76: //Should not make it to here unless it go BOOM!
77:     return false;
78: }
79:
80: /// <summary>
81: /// Create the ADOParameter(s) for the DBEntity object
82: /// </summary>
83: /// <param name="cmd"> The SqlCommand object to add
     the Parameters to</param>
84: /// <param name="entity">The DBEntity to scrape
     </param>
85:     protected static void AddCmdParam( SqlCommand
     cmd, DBEntity entity ) {
86:
87: /**
88: ///Get the Public properties and create the SQL
     Parameters
89: /**
90: Type T = entity.GetType();
91: PropertyInfo[] Properties =
```

```
92:             T.GetProperties(
93:                 BindingFlags.DeclaredOnly |
94:                 BindingFlags.Instance | BindingFlags.Public
95:             );
96:             foreach( PropertyInfo pi in Properties ) {
97:                 object[] Attributes =
98:                     pi.GetCustomAttributes(
99:                         typeof(Stingray.Data.DBFieldAttribute),
100:                         false
101:                     );
102:                     if(Attributes.Length == 1) {
103:                         DBFieldAttribute Field =
104:                             (DBFieldAttribute)Attributes[0];
105:                         MethodInfo mi = pi.GetGetMethod( false );
106:
107:                         SqlCommand.Parameters.Add(Field.ParameterName,
108:                         Field.DataType);
109:                         SqlCommand.Parameters[Field.ParameterName].Value =
110:                           result;
111:                           if( Field.Length > 0 ) {
112:                               SqlCommand.Parameters[Field.ParameterName].Size =
113:                                 Field.Length;
114:                           }
115:
116:                           }
117:                           }
118: }
```

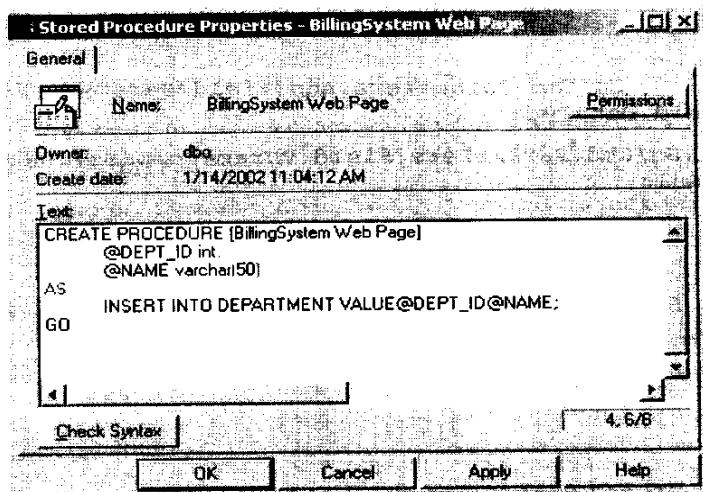
DBAccess hoàn toàn có thể tự giải thích điều mà nó đang làm. Nó bắt đầu bằng việc tìm kiếm thuộc tính tùy biến DBTableAttribute. Nếu tìm thấy, thì tiến trình của việc truy xuất các thuộc tính cần thiết cho việc tuân tự hóa (serialization) bắt đầu. Phương thức AddCmdParam túm lấy những thể hiện của thuộc tính hiện hành và tìm kiếm thuộc tính tùy biến DBFieldAttribute. Nhớ rằng DBFieldAttribute chưa đựng những thuộc tính dành cho tên đối số, kiểu dữ liệu, và chiều dài của đối số nếu có thể áp dụng được.

2. LỚP EMPLOYEE VÀ LỚP DEPARTMENT

Với cấu trúc cơ sở dữ liệu đã có, bước tiếp theo là tạo ra các lớp Employee và Department cùng những thủ tục nội (stored procedure) riêng biệt cho các bảng dữ liệu. Để dễ hiểu, stored procedure sẽ không cài đặt các tác vụ kiểm tra lỗi. Thay vào đó, các ràng buộc của SQLServer sẽ bảo đảm rằng không có xung đột về khóa chính hoặc khoá ngoại.

3. STORED PROCEDURES

Employee Browser sẽ chỉ sử dụng bốn stored procedures đơn giản. SQL Server Enterprise Manager cho phép xem và tạo những stored procedure từ cửa sổ Management Console. Hình 4.2-2 hiển thị trình soạn thảo stored procedure chứa bên trong SQL Server.



Hình 4.2-2 – Trình soạn thảo stored procedure của SQL Server.

Sử dụng trình soạn thảo của SQL Enterprise Manager bạn tạo các stored procedure sau:

```
CREATE PROCEDURE sp_InsertDept
```

```

@DEPT_ID int,
@NAME varchar(50)
AS
    INSERT INTO DEPARTMENT VALUES (@DEPT_ID, @NAME)
GO

CREATE PROCEDURE sp_UpdateDept
    @DEPT_ID int,
    @NAME varchar(50)
AS
    UPDATE DEPARTMENT SET NAME = @NAME WHERE DEPT_ID =
    @DEPT_ID
GO

CREATE PROCEDURE sp_InsertDept
    @EMP_ID char(9),
    @DEPT_ID int,
    @FIRST_NAME varchar(25)
    @LAST_NAME varchar(25)
    @PIC_ID uniqueidentifier
AS
    INSERT INTO EMPLOYEES
VALUES (@EMP_ID, @DEPT_ID, @FIRST_NAME, @LAST_NAME, @PIC_ID)
)
GO

CREATE PROCEDURE sp_UpdateEmployee
    @EMP_ID char(9),
    @DEPT_ID int,
    @FIRST_NAME varchar(25)
    @LAST_NAME varchar(25)
    @PIC_ID uniqueidentifier
AS
    UPDATE EMPLOYEE SET DEPT_ID = @DEPT_ID
        FIRST_NAME = @FIRST_NAME
        LAST_NAME = @LAST_NAME
        PIC_ID = @PIC_ID
    WHERE EMP_ID = @EMP_ID
GO

```

Những câu lệnh trên khá dễ hiểu; chúng chỉ là những câu lệnh tạo bảng TSQL thuần túy trong SQL Server. Nếu bạn cần nhiều thông tin hơn về TSQL, hãy sử dụng trình trợ giúp (help) bên trong SQL Server.

4. CÀI ĐẶT LỚP

Các lớp Employee và Department không có gì khác hơn ngoài việc ánh xạ mô hình thông tin trong cơ sở dữ liệu. Ví dụ, lớp cung cấp các thuộc tính dùng để

chèn vào hoặc việc cập nhật một mẫu tin cụ thể của bảng. Mỗi lớp kế thừa từ lớp cơ sở trừu tượng DBEntity và sử dụng những thuộc tính tùy biến đã được phát triển trước đây.

Bởi vì lớp Department đơn giản nhất, chúng ta có thể bắt đầu cài đặt nó như trong ví dụ 4.2-4 sau:

Ví dụ 4.2.4 : Lớp Department

```
1: namespace Stingray.Data
2: {
3:     using System;
4:     using System.Data;
5:     using System.Data.SqlClient;
6:
7:     /// <summary>
8:     /// Simple class for a Department
9:     /// </summary>
10:    [
11:        Stingray.Data.DBTableAttribute(
12:            "DEPARTMENT",
13:            InsertCommand="sp_InsertDept",
14:            UpdateCommand="sp_UpdateDept"
15:        )
16:    ]
17: public class Department : Stingray.Data.DBEntity {
18:
19:
20: /*****[Department Implementation]*****/
21:     private int    m_Id;
22:     private string m_Name;
23:
24:     /// <summary>
25:     /// Department Id Property
```

```
26: /// </summary>
27: [Stingray.Data.DBFieldAttribute("@DEPT_ID", DataType=SqlDbType.Int)]
28:     public int Id {
29:         get { return m_Id; }
30:         set {
31:             if( m_Id != (int)value) {
32:                 m_Id = value;
33:                 IsDirty = true;
34:             }
35:         }
36:     }
37:
38: /// <summary>
39: /// Department Name property
40: /// </summary>
41: [Stingray.Data.DBFieldAttribute("@NAME", DataType=SqlDbType.VarChar)]
42:     public string Name {
43:         get { return m_Name; }
44:         set {
45:             if( m_Name != (string)value) {
46:                 m_Name = value;
47:                 IsDirty = true;
48:             }
49:         }
50:     }
51:
52: /// <summary>
53: ///
54: /// </summary>
55: /// <param name="data"> </param>
```

```
56: public override bool FromDataRow( DataRow data ) {  
57:     this.m_Id = (int)data["DEPT_ID"];  
58:     this.m_Name = (string)data["NAME"];  
59:     return true;  
60: }  
61:  
62: }  
63: }
```

Lớp Department cho thấy việc ánh xạ dữ liệu rất đơn giản từ bảng DEPARTMENT vào trong cấu trúc lớp. Những thuộc tính tùy biến được sử dụng bởi lớp DBAccess nhằm chèn vào hoặc cập nhật mẫu tin DEPARTMENT khi cần. Ta sử dụng cùng kiểu cài đặt này được sử dụng để tạo lớp Employee trong ví dụ 4.2-5 sau:

Ví dụ 4.2-5 : Lớp Employee

```
1: namespace Stingray.Data  
2: {  
3:  
4:     using System;  
5:     using System.Data;  
6:     using System.Data.SqlClient;  
7:  
8: /// <summary>  
9: /// The Employee Class  
10: /// </summary>  
11: ///  
12: [  
13:     Stingray.Data.DBTableAttribute(  
14:         "EMPLOYEE",  
15:         InsertCommand="sp_InsertEmployee",  
16:         UpdateCommand="sp_UpdateEmployee")  
17: ]  
18: public class Employee : DBEntity
```

```
19: {
20:
21: //*****[Data Members]*****/
22:     private string m_EmpId;    //SSN
23:     private int    m_DeptId;
24:     private string m_FirstName;
25:     private string m_LastName;
26:     private Guid   m_PicId;
27:
28: //*****[Properties]*****/
29: [
30:     Stingray.Data.DBFieldAttribute("@EMP_ID",
31:                                     DataType=SqlDbType.Char, Length=9)
32: ]
33:     public string Id {
34:         get { return m_EmpId; }
35:         set {
36:             if(m_EmpId != (string)value) {
37:                 m_EmpId = (string)value;
38:                 IsDirty = true;
39:             }
40:         }
41:
42:
43: [
44:     Stingray.Data.DBFieldAttribute("@DEPT_ID",
45:                                     DataType=SqlDbType.Int)
45: ]
46:     public int DeptId {
47:         get { return m_DeptId; }
48:         set {
```

```
49:             if( m_DeptId != (int)value ) {
50:                 m_DeptId = (int)value;
51:                 IsDirty = true;
52:             }
53:         }
54:     }
55:
56:     [
57:         Stingray.Data.DBFieldAttribute("@FIRST_NAME",
58:                                         DataType=SqlDbType.VarChar)
59:     ]
60:     public string FirstName {
61:         get { return m_FirstName; }
62:         set {
63:             if(m_FirstName != (string)value) {
64:                 m_FirstName = (string)value;
65:                 IsDirty = true;
66:             }
67:         }
68:
69:
70:     [
71:         Stingray.Data.DBFieldAttribute("@LAST_NAME",
72:                                         DataType=SqlDbType.VarChar)
73:     ]
74:     public string LastName {
75:         get { return m_LastName; }
76:         set {
77:             if( m_LastName != (string)value) {
78:                 m_LastName = (string)value;
IsDirty = true;
```



```
79:         }
80:     }
81: }
82:
83: [
84:     Stingray.Data.DBFieldAttribute("@PIC_ID", DataType=
SqlDbType.UniqueIdentifier)
85: ]
86:     public Guid PictureId {
87:         get { return m_PicId; }
88:         set {
89:             if( m_PicId != (Guid)value ) {
90:                 m_PicId = new Guid(
((Guid)value).ToString());
91: IsDirty = true;
92:             }
93:         }
94:     }
95:
96: /// <summary>
97: /// Create from a row of data
98: /// </summary>
99: /// <param name="data"> </param>
100:    public override bool FromDataRow( DataRow
data ) {
101:        this.Id = (string)data["EMP_ID"];
102:        this.DeptId = (int)data["DEPT_ID"];
103:        this.FirstName =
(string)data["FIRST_NAME"];
104:        this.LastName =
(string)data["LAST_NAME"];
105:        if( data["PIC_ID"].ToString() != "" )
106:            this.m_PicId = new Guid(
data["PIC_ID"].ToString());
```

```
107:  
108:         this.IsNew = false;  
109:         return true;  
110:     }  
111:  
112:  
113: //*****[Constructor(s)]*****/  
114:     public Employee() {  
115:         //What to do??  
116:         IsNew = true;  
117:         IsDirty = false;  
118:     }  
119:  
120: }  
121: }
```

5. KIỂM TRA (TESTING)

Những thực thể cơ bản đã được cài đặt, chúng ta sẽ sử dụng một chương trình để kiểm tra để bảo đảm rằng mã chương trình làm việc như mong đợi Chương trình kiểm tra trong ví dụ 4.2-6 được dùng cho mục đích này.

Ví dụ 4.2-6 : Chương trình kiểm tra các lớp

```
1: namespace TestBed.  
2: {  
3:     using System;  
4:     using System.Data;  
5:     using System.Data.SqlClient;  
6:     using Stingray.Data;  
7:     using System.Reflection;  
8:  
9:  
10:  
11:    public class DBTest  
12:    {
```

```
13:
14:     public static int Main(string[] args)
15:     {
16:
17:         SqlConnection dbCon =
18:             new SqlConnection(
19: "user id=sa;password=;initial catalog=Stingray;
20:             data source=.;connect Timeout=30"
21:             );
22:
23:         Department[] Departments = new Department[5];
24:         string[] names = { "Development", "Tech Support",
25: "Sales", "Consulting", "Marketing" };
26:
27:         for(int i = 0; i < 5; i++) {
28:             Departments[i] = new Department();
29:             Departments[i].IsNew = true;
30:             Departments[i].Id = (i+1);
31:             Departments[i].Name = names[i];
32:         }
33:
34:         //Save the Departments
35:         foreach( Department dept in Departments ) {
36:             DBAccess.Save( dbCon, dept );
37:         }
38:
39: //Do a select and display the results
40:         System.Data.SqlClient.SqlDataAdapter dsCmd =
41: new SqlDataAdapter("SELECT * FROM DEPARTMENT", dbCon);
42:         System.Data.DataSet dataSet = new
43:             System.Data.DataSet();
44:         dsCmd.Fill( dataSet, "DEPARTMENT" );
45:
46: //display the records
47:         foreach( System.Data.DataRow row in
48:             dataSet.Tables["DEPARTMENT"].Rows )
49:             Console.WriteLine("{0} : {1}",
```

```
47:  
48: row[DataSet.Tables["DEPARTMENT"].Columns["DEPT_ID"]]  
    ],row[DataSet.Tables["DEPARTMENT"].Columns["NAME"]]  
    ]);  
49:  
50:  
51:         return 0;  
52:     }  
53:  
54: }  
55: }  
56:
```

Nếu tất cả làm việc tốt, chương trình kiểm tra bed sẽ hiển thị nội dung bảng DEPARTMENT và những kết quả được chọn. Bạn có thể sử dụng SQL Server Query Analyzer, đưa ra lệnh SELECT thực thi sẽ trả về tất cả dòng của bảng DEPARTMENT, như hiển thị trong hình 4.2-3.

The screenshot shows the SQL Server Query Analyzer interface. On the left, the Object Browser pane displays the database structure under 'MERLIN(ss)'. In the center, the 'Query' pane contains the SQL command: 'select * from department'. To the right, the results pane displays the data from the DEPARTMENT table:

DEPT_ID	NAME
1	Development
2	Teach Support
3	Sales
4	Consulting
5	Maketing

Hình 4.2-3 – Những mẫu tin mới trong bảng DEPARTMENT

6. HỖ TRỢ TÌM KIẾM GIẢN ĐƠN

Một trong những yêu cầu của cơ sở dữ liệu Employee là khả năng tìm kiếm một nhân viên cụ thể. Việc tìm kiếm nên chú ý đến tìm theo tên, bộ phận phò

ban (department), hoặc liệt kê tất cả những nhân viên trong phòng ban chỉ định. Thay vì sử dụng thuộc tính tùy biến, lớp tìm kiếm sẽ được tạo để trả về kết quả mong đợi, chẳng hạn như lớp System.Data.Dataset.

Lớp Search chỉ cung cấp các phương thức tĩnh trong cùng cách như lớp DBAccess. Một lý do để làm như vậy là do không lớp nào yêu cầu bất kỳ thông tin trạng thái, do đó có thể tránh được toàn bộ phần định vị đối tượng đã thêm vào. Lớp Search của chúng ta sẽ cung cấp 2 phương thức: phương thức Find với một phương thức định nghĩa chồng và phương thức Retrieve để trích rút thông tin. Phương thức Find sẽ tìm một nhân viên riêng lẻ bằng cách sử dụng phát biểu SQL LIKE. Như vậy, bất kỳ ký tự đại diện nào so khớp thích hợp sẽ được trả về. Phương thức định nghĩa chồng Find quan tâm đến mã phòng ban (Department ID) chỉ định và sử dụng cùng mệnh đề LIKE cho tên tương thích. Phương thức cuối cùng, Retrieve, lấy mã phòng ban Department ID và trả về tất cả những nhân viên từ bộ phận phòng ban tương ứng với mã đó. Ví dụ 4.2-7 là mã chương trình cho lớp Search.

Ví dụ 4.2-7 : Lớp Search

```
1: namespace Stingray.Data
2: {
3:     using System;
4:     using System.Data;
5:     using System.Data.SqlClient;
6:
7: /// <summary>
8: /// Basic search class
9: /// </summary>
10: public class Search
11: {
12:
13: /// <summary>
14: /// Try and locate an employee
15: /// </summary>
16: /// <param name="ActiveConnection"> </param>
17: /// <param name="FirstName"> </param>
```

```
18: /// <param name="LastName" > </param>
19:     public static DataSet Find( SqlConnection
   ActiveConnection,
20:             string FirstName,
21:             string LastName ) {
22:             string SelectStmt =
23:             string.Format(
24: "SELECT * FROM EMPLOYEE WHERE FIRST_NAME LIKE '{0}%'
   and LAST_NAME LIKE '{1}%'",
25:             FirstName, LastName);
26:
27:             return Execute( ActiveConnection,
   SelectStmt, "EMPLOYEE" );
28:
29: }
30:
31: /// <summary>
32: /// Try and locate an employee within a department
33: /// </summary>
34: /// <param name="ActiveConnection" > </param>
35: /// <param name="DepartmentId" > </param>
36: /// <param name="FirstName" > </param>
37: /// <param name="LastName" > </param>
38:     public static DataSet Find( ADOConnection
   ActiveConnection,
39:             int DepartmentId,
40:             string FirstName,
41:             string LastName ) {
42:
43:             object[] args = { FirstName, LastName,
   DepartmentId };
44:
45:             string SelectStmt =
```

```
46:             string.Format(
47: "SELECT * FROM EMPLOYEE WHERE FIRST_NAME LIKE '{0}%'
   and LAST_NAME LIKE '{1}%' and DEPT_ID = {2}",
48: args);
49:
50:         return Execute( ActiveConnection,
   SelectStmt, "EMPLOYEE" );
51:     }
52:
53: /// <summary>
54: /// Retrieve a list of employees for a given
   department
55: /// </summary>
56: /// <param name="ActiveConnection"> </param>
57: /// <param name="DepartmentId"> </param>
58:     public static DataSet Retrieve(
   SqlConnection ActiveConnection,
59:             int DepartmentId ) {
60:             string SelectStmt =
61:             string.Format(
62: "SELECT * FROM EMPLOYEE WHERE DEPT_ID = {0}",
63: DepartmentId );
64:
65:             return Execute( ActiveConnection,
   SelectStmt, "EMPLOYEE" );
66:         }
67:
68: /// <summary>
69: /// Nice and tidy. Do the grunt work in one place
70: /// </summary>
71: /// <param name="ActiveConnection"> </param>
72: /// <param name="Stmt"> </param>
73: /// <param name="TableName"> </param>
```

```
74: private static DataSet Execute( SqlConnection  
    ActiveConnection,  
75:                     string SelectStmt,  
76:                     string TableName ) {  
77:  
78:     SqlDataAdapter dataAdapter = new SqlDataAdapter(   
        SelectStmt, ActiveConnection );  
79:     DataSet dsResult = new DataSet( );  
80:  
81:     try {  
82:         dataAdapter.Fill( dsResult,  
        TableName );  
83:     } catch( Exception ) {  
84: //Do some magic here  
85:     }  
86:  
87:     return dsResult;  
88: }  
89: }  
90: }
```

Việc cài đặt cho lớp Search khá dễ hiểu và không thực sự đòi hỏi cơ sở dữ liệu làm quá nhiều việc. Mỗi phương thức yêu cầu một đối tượng SqlConnection giả sử đối tượng này được kết nối tới cơ sở dữ liệu. Kiểu trả về từ mỗi phương thức là đối tượng DataSet. Trong .NET, một đối tượng DataSet chứa đựng rất nhiều dữ liệu chung và có thể được sử dụng trong những lớp dữ liệu như là lớp SqlDataAdapter. Thật ra không quan trọng để so sánh 2 đối tượng DataSet và RecordSet. Quá thực DataSet có thể được dùng để giữ nhiều bảng (table) nơi mà RecordSet chứa đựng những mẩu tin từ kết quả đơn (single result set).

Như với những lớp cơ sở dữ liệu trước, một ứng dụng kiểm tra nhỏ sẽ khiến bạn thấy rõ hơn về việc sử dụng lớp Search. Hãy xem ví dụ 4.2-8 sau.

Ví dụ 4.2-8 : Test2

```
1: namespace TestBed  
2: {  
3:     using System;  
4:     using System.Data;
```

```
5:         using System.Data.SqlClient;
6:         using Stingray.Data;
7:
8:         public class TestBed2
9:         {
10:             public static void Main( )
11:             {
12:                 SqlConnection dbCon =
13:                     new SqlConnection(
14:                         "user id=sa;password=;initial
catalog=Stingray;data source=.; Connect Timeout=30"
15: );
16:
17:                 dbCon.Open( );
18:
19: //Find me
20:             DataSet dsResult = Search.Find( dbCon,
"Richard", "Weeks" );
21:             if( dsResult != null &&
dsResult.Tables["EMPLOYEE"] != null) {
22:                 foreach( DataColumn c in
dsResult.Tables["EMPLOYEE"].Columns) {
23:                     Console.WriteLine(
dsResult.Tables["EMPLOYEE"].Rows[0][c]);
24:                 }
25:             }
26:
27: //Get Development
28:             dsResult = Search.Retrieve( dbCon, 1 );
29:             if( dsResult != null) {
30:                 foreach( DataRow row in
dsResult.Tables["EMPLOYEE"].Rows) {
31:                     Console.WriteLine("*****");
32:                     foreach( DataColumn col in
dsResult.Tables["EMPLOYEE"].Columns ) {
33:                         Console.WriteLine( row[col]);
34:                     }
35:                 }
36:             }
```

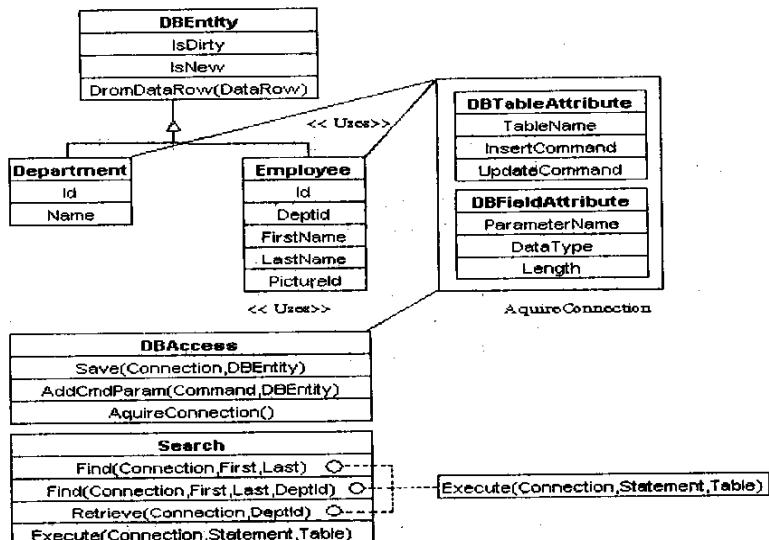
```

37:         }
38:
39:     }
40: }
41:

```

Bằng việc cài đặt và kiểm tra lớp Search, tầng dữ liệu của chúng ta đã hoàn tất. Hình 4.2-4 cho thấy tất cả kết quả những gì chúng ta đã làm được cho đến thời điểm này bằng kiểu biểu đồ UML.

Bước tiếp theo trong việc phát triển cơ sở dữ liệu Employee là phải tạo những đoạn chương trình ASP.NET gọi là pagelet và những trang aspx dùng cho việc hiển thị dữ liệu. Ngoài ra chúng ta cũng phải xây dựng các trang cho phép soạn thảo thông tin nhân viên cùng với trang tìm kiếm.



Hình 4.2-4 – Mô hình đối tượng hiện hành.

7. KẾT CHƯƠNG

Đối với hầu hết các ứng dụng, khả năng để làm việc với cơ sở dữ liệu là yêu cầu chung. Kiến trúc .NET cung cấp rất nhiều tập hợp cùng những hàm API để truy xuất cơ sở dữ liệu. Trong các chương sau chúng ta sẽ sử dụng tầng dữ liệu đơn giản này để phát triển ứng dụng Employee Browser.

Chương 4.3

WEBFORMS

Các vấn đề chính sẽ được đề cập đến

- ✓ Giới thiệu ASP.NET WebForms
- ✓ Các UserControl
- ✓ HeaderControl
- ✓ Các trang ASPX

1. GIỚI THIỆU ASP.NET WEBFORMS

Với việc thêm vào kiến trúc WebForms, ASP.NET cung cấp một môi trường thật sự hướng vào người phát triển. Khả năng tách biệt mã ra khỏi phần trình bày của HTML thể hiện tính mềm dẻo và dễ sử dụng lại cao hơn về phía những người phát triển Web. ASP.NET chú ý đến việc tạo các điều khiển của người dùng gọi là UserControls, cùng với khả năng kết hợp những đoạn mã nhỏ gọi là pagelets. Những UserControl này giống như các thành phần điều khiển (control) tùy biến của Windows Forms. Những người phát triển Web bây giờ có thể tạo các control tùy biến và dễ dàng sử dụng lại chúng thông qua các ứng dụng Web khác nhau. Chương này sẽ khảo sát một số phương pháp tạo các UserControl và cung cấp mã cùng với các trang ASPX sử dụng những control này.

1.1. Các UserControl

ASP.NET chú ý đến việc tạo UserControl có thể nhúng vào bên trong các UserControl hay những trang aspx khác. Hãy nghĩ đến một UserControl như là một thành phần nhỏ có thể sử dụng lại, được hình thành bởi mã ASP.NET và HTML. Bên trong một trang Web điển hình, có nhiều thứ thường được lặp lại, chẳng hạn như vùng thể hiện nội dung cho trang Web, các control phụ giúp cho việc tìm kiếm, các tiêu đề trang (page header). Những thực thể như thế nên trở thành những ứng cử viên lý tưởng cho việc tạo các UserControl.

1.2. HeaderControl

Vì việc phát triển UserControl là khá dễ dàng, nên một thành phần header tùy biến với hai thuộc tính là đủ để làm được. Thành phần header được phát triển để tính đến sự tương tác với các thành phần khác trên trang ASP. Sau này, một cách tiếp cận khác sẽ được đưa ra, chứng tỏ rằng có hơn một cách để hoàn thành việc này.

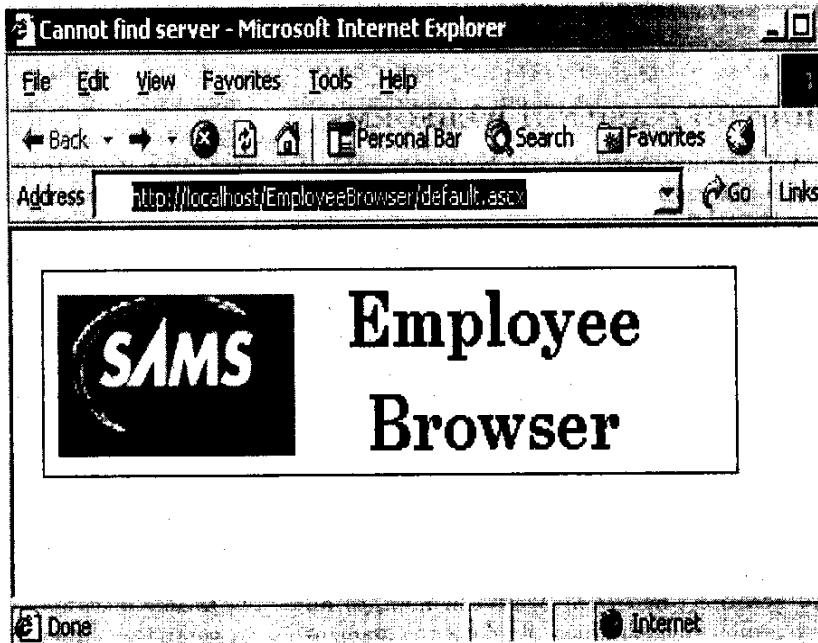
Thành phần header, được minh họa trong hình 4.3-1, sẽ được phát triển bằng cách sử dụng các WebControls sau:

- *asp:table* Tương tự với tag *table* trong HTML

- asp:image Tương tự với tag img trong HTML
- asp:label Tạo một span trong HTML

Một trong những tiện lợi của việc sử dụng các thành phần của asp:* là năng tạo mã HTML tự động.

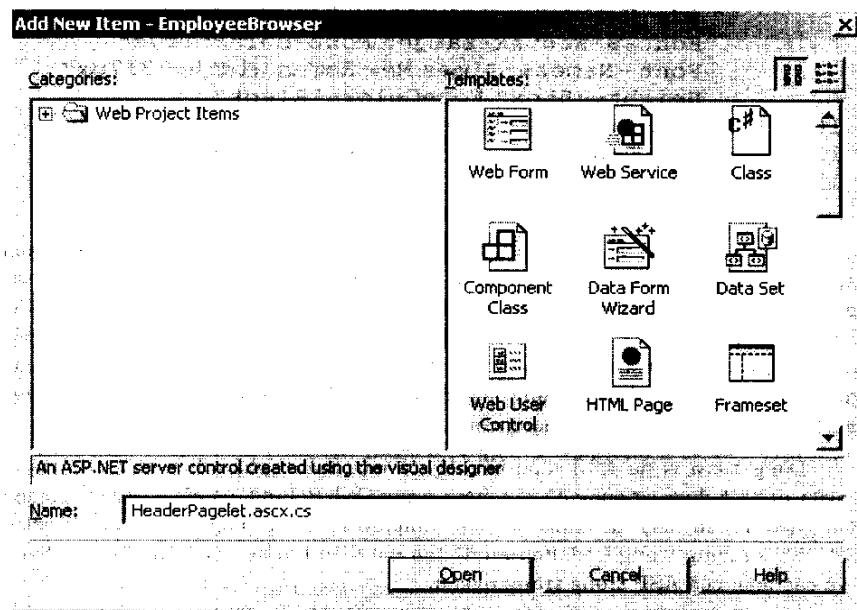
Thành phần header gồm có hai tập tin – HeaderControl.aspx, HeaderControl.ascx.cs. Tập tin có phần mở rộng .aspx biểu thị một thực thể dẫn xuất từ UserControl. Đây là một trong những tiện lợi được đưa ra ASP.NET.



Hình 4.3-1 Thành phần header

Để xây dựng thành phần header, mà thực tế là toàn bộ dự án về Web được thiết kế, bạn hãy tạo mới một ứng dụng Web ASP.NET trong Visual Studio. Dự án được đặt tên là EmployeeBrowser.

Thành phần control chỉ là một bảng nhỏ có hai ô – một thành phần (image control) và một cái nhãn (label). Mã để tạo control này khá nhỏ. Đầu ta tạo một thư mục bên trong giải pháp (Solution) EmployeeBrowser có tên Pagelets; Thư mục này dành cho việc tổ chức mã bên trong dự án. Nhấp chuột vào thư mục đó và chọn Add/Add Web User Control từ trình đơn thả xuống. Nó sẽ hiện ra hộp thoại Add New Item, như hình 4.3-2.



Hình 4.3-2 Hộp thoại New Item

Gán tên HeaderPagelet vào UserControl mới. Mở tập tin HeaderPagelet.ascx để soạn thảo dạng HTML và nhập vào đoạn mã trong ví dụ 4.3-1 sau:

Ví dụ 4.3-1 Mã nguồn ASPX cho HeaderPagelet

```

1: <%@ Control Language="c#"
   codebehind="HeaderPagelet.ascx.cs"
   Inherits="EmployeeBrowser.Pagelets.
   HeaderPagelet" %>
2: <asp:table id="HeaderTable" runat="server"
   Width="100%" Height="25%" BorderColor="Purple"
   BackColor="LightGrey" BorderStyle="Groove"
   BorderWidth="2px" HorizontalAlign="Center">
3:   <asp:tablerow>
4:     <asp:tablecell Width="1"
        VerticalAlign="Middle"
        HorizontalAlign="Center">
5:       <!--The Image for the control -->
6:       <asp:Image runat="Server"
          id="headerImage" />
7:     </asp:TableCell>
8:     <asp:TableCell VerticalAlign="Middle"
        HorizontAlign="Center">
9:       <asp:Label runat="server"

```

```
id="HeaderLabel" runat="server"
Font-Size="XX-Large" Font-Bold="True"
Font-Names="Times New Roman Width="317px"
Height="55px" ForeColor="Black" />
10:    </asp:TableCell>
11:  </asp:TableRow>
12: </asp:Table>
```

Trước khi đi sâu vào mã của C# ở phía sau trang ascx này, chúng ta nên dành ít thời gian để xem chi tiết việc tạo nhãn. Trong trường hợp này, ta đã tạo tất cả các lớp dẫn xuất từ UserControl trong không gian tên EmployeeBrowser.Pagelets. Nhờ rằng không gian tên là cách hữu ích để tổ chức cây mã nguồn (source tree) của bạn. Bằng cách giữ tất cả các pagelets trong không gian tên riêng biệt, những người phát triển khác khi sử dụng mã của bạn có thể nhanh chóng liên kết không gian tên với các mục hay đối tượng chứa trong nó.

Dòng 1 của ví dụ 4.3-1 khai báo đây là một thành phần điều khiển và chỉ rõ là ngôn ngữ C# được sử dụng để xây dựng trang CodeBehind, và cũng cho biết thành phần điều khiển này kế thừa từ lớp EmployeeBrowser.Pagelets.HeaderPagelet. ASP.NET sử dụng trang CodeBehind để tạo mã điều khiển việc liên kết các thành tố và các biến của ASP trong trang nguồn.

Các thẻ asp:table, asp:image, và asp:label sử dụng thuộc tính runat="Server". Điều này cho phép việc xử lý các thành phần và dữ liệu kết nối giữa các thành phần và mã diễn ra ở server. Chỉ có các thẻ asp:image và asp:label sử dụng id=X, X là tên biến được định nghĩa bên trong mã nguồn của CodeBehind. Ví dụ 4.3-2 chứa mã nguồn CodeBehind cho HeaderControl.

Ví dụ 4.3-2 HeaderControl.aspx.cs

```
1: namespace EmployeeBrowser.Pagelets
2: {
3:   using System;
4:   using System.Data;
5:   using System.Drawing;
6:   using System.Web;
7:   using System.Web.UI.WebControls;
8:   using System.Web.UI.HtmlControls;
9:
10:  /// <summary>
11:  /// Tóm tắt sự mô tả HeaderPagelet.
12:  /// </summary>
13:  public abstract class HeaderPagelet:
System.Web.UI.UserControl
14:  {
15:    protected System.Web.UI.WebControls.Table
HeaderTable;
16:    protected System.Web.UI.WebControls.Image
```

```
17:     HeaderImage;
18:     protected System.Web.UI.WebControls.Label
19:     HeaderLabel;
20:
21:     public string ImageUrl
22:     {
23:         set
24:         {
25:             HeaderImage.ImageUrl = value;
26:         }
27:     }
28:
29:     public string Text
30:     {
31:         set
32:         {
33:             HeaderLabel.Text = value;
34:         }
35:
36:     /// <summary>
37:     public HeaderPagelet()
38:     {
39:         this.Init +=
40:             new System.EventHandler(Page_Init);
41:     }
42:
43:     protected void Page_Load( object sender,
44:                               System.EventArgs e)
45:     {
46:         //Người dùng đặt code khởi động trang ở đây
47:
48:         private void Page_Init( object sender,
49:                               EventArgs e)
50:         {
51:             //
52:             //CODEGEN: Lời gọi này được yêu cầu
53:             //bởi the ASP.NET Web Form Designer.
54:             //
55:             InitializeComponent();
56:         }
57:
58:         #region Web Form Designer Generated code
59:         //Phương thức yêu cầu người thiết kế hỗ trợ -
60:         ///đừng sửa đổi nội dung của phương thức này
61:         ///đổi với việc soạn thảo code.
```

```

58:      ///<summary>
59:      private void InitializeComponent()
60:      {
61:          this.Load +=
62:              new System.EventHandler(this.Page_Load);
63:      }
64:      #endregion
65:  }
66: }

```

Mã của HeaderControl khá ngắn gọn. Ở đây ta định nghĩa hai thuộc tính - Text và ImageUrl. Những thuộc tính này có thể được thao tác bên trong thành phần ascx hay trang aspx khác. Thực tế, HeaderControl của chúng ta chỉ xuất hiện như thể nó là một phần của ASP.NET.

Bất cứ một lớp dẫn xuất nào từ UserControl cũng đều có thể phản hồi lại các sự kiện (event) của ASP.NET giống như các trang aspx. Khi EmployeeViewPagelet được phát triển, chúng ta sẽ thấy điều này trong cách làm.

Để chạy thử HeaderControl, bạn thêm một Web Form mới vào dự án với tên là default.aspx. IIS luôn tìm một trang bắt đầu mang tên index.aspx hoặc default.aspx; đối với trường hợp của ASP.NET, ta sẽ sử dụng default.aspx. Ví dụ 4.3-3 chứa mã của HTML cho tập tin default.aspx.

Ví dụ 4.3-3 default.aspx

```

1: <%@ Register   TagPrefix="EmployeeBrowser"
2:           TagName="EmployeeView"
3:           Src="Pagelets/HeaderPagelet.ascx"
%>
4:
5: <%@ Page language="c#"
   Codebehind="default.aspx.cs"
6:     AutoEventWireup="false"
7:     Inherits="EmployeeBrowser.MainPage" %>
8:
9: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
   Transitional//EN">
10: <HTML>
11:   <HEAD>
12:     <meta name="GENERATOR" Content="Microsoft
       Visual Studio 7.0">
13:     <meta name="CODE_LANGUAGE" Content="C#">
14:     <meta name="vs_defaultClientScript"
       content="JavaScript (ECMAScript)">
15:     <meta name="vs_targetSchema" content="http:
       //chemas.microsoft.com/intellisense/ie5">
16:   </HEAD>

```

```

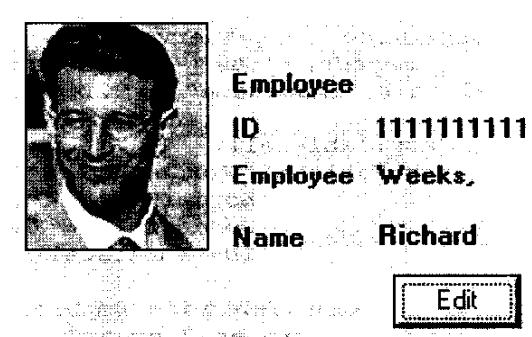
17: <body MS_POSITIONING="GridLayout">
18:   <form id="Form1" method="post"
19:     runat="server">
20:       <!--
21:           Show off the newly created header control
22:           -->
23:         <EmployeeBrowser:HeaderControl
24:           runat="server"
25:           ImageUrl="-/images/logo.jpg" Text="
26:           Employee Browser" />
27:       </form>
28:     </body>
29:   </HTML>

```

Dòng 1 giới thiệu khái niệm về việc đăng ký một thẻ. HeaderControl có thể được gán vào TagPrefix hay TagName nếu bạn muốn. Cuối cùng dòng 22 là khai báo của HeaderControl. Chú ý sự kết hợp TagPrefix:TagName được sử dụng để nhận biết một thẻ hiện của thành phần header. Cú pháp này có vẻ lạ – hơi giống với asp:label hay asp:image phải không? Vâng, asp là TagPrefix và label hay image là TagName.

1.3. EmployeeViewPagelet: Một cách tiếp cận khác

Thay cho việc thao tác các control của ASP một cách trực tiếp vào mã nguồn của CodeBehind, EmployeeViewPagelet sẽ sử dụng tiến trình xử lý ở phía server để thiết lập các thuộc tính khác nhau cho các thành tố chứa bên trong nó. Hình 4.3-3 cho thấy kết quả hiển thị hoàn chỉnh của EmployeeViewPagelet.



Hình 4.3-3 Thành phần EmployeeViewPagelet

Thành phần EmployeeViewPagelet được sử dụng để hiển thị mã số nhân viên, tên, hình, và nút nhấn Edit liên kết với trang Edit. Việc hình thành mã của ASP.NET và HTML khá đơn giản, nhưng có một sự khác biệt khá tinh vi giữa việc thực hiện của thành phần này với cái hiện có trong HeaderControl. Ví dụ 4.3-4 trình bày mã HTML cho EmployeeViewPagelet..



Ví dụ 4.3-4 EmployeeViewPagelet.ascx

```
1: <%@ Control Language="c#"
2:     AutoEventWireup="false"
3:     Codebehind=
4:         "EmployeeViewPagelet.ascx.cs"
5:     Inherits=
6:         "EmployeeBrowser.Pagelets.
7:             EmployeeViewPagelet"%>
8:
9: <asp:Table id="EmpViewTable"
10:    runat="server"
11:    BackColor="LightGray"
12:    BorderColor="Purple"
13:    BorderStyle="Ridge"
14:    BorderWidth="2px"
15:    Height="100px"
16:    Width="25%">
17:    <asp:TableRow>
18:        <asp:TableCell BorderStyle="Ridge"
19:            BorderWidth="2px"
20:            VerticalAlign="Middle"
21:            BackColor="Black"
22:            HorizontalAlign="Center"
23:            BorderColor="Gray"
24:            ID="EmpicCell">
25:            <asp:Image runat="server"
26:                ImageUrl ='<%# "-/images/employees/" +
27: DataBinder.Eval(this,"ImageName")%>' ID="Image1"/>
28:        </asp:TableCell>
29:        <asp:TableCell BorderStyle="Solid"
30:            BorderWidth="2px"
31:            BorderColor="Red"
32:            ID="EmDataCell">
33:            <asp:Table id="EmpDataTbl"
34:                runat="server"
35:                width="100%">
36:
37:                <asp:TableRow>
38:                    <asp:TableCell
39:                        VerticalAlign="Middle"
40:                        HorizontalAlign="Center"
41:                        style="FONT-WEIGHT:bold;
42:                             FONT-SIZE:medium;
```

```
        COLOR:purple"
41:      BorderStyle="Ridge"
        BorderWidth="2px"
        Text="Employee ID"/>
42:
43:      <asp:TableCell
        VerticalAlign="Middle"
44:
        HorizontalAlign="Center"
45:      style="FONT-
WEIGHT:bold;
46:
        FONT-SIZE:medium;
        COLOR:purple"
        BorderStyle="Ridge"
        BorderWidth="2px"
47:      <%#
DataBinder.Eval(this,
        "EmployeeId")%>
48:      </asp:TableCell>
49:    </asp:TableRow>
50:
51:    <asp:TableRow>
52:      <asp:TableCell
        VerticalAlign="Middle"
        HorizontalAlign="Center"
        style="FONT-WEIGHT:bold;
        FONT-SIZE:medium;
        COLOR:purple"
        BorderStyle="Ridge"
        BorderWidth="2px"
        Text="Employee Name"/>
53:
54:
55:      <asp:TableCell
        VerticalAlign="Middle"
        HorizontalAlign="Center"
        style="FONT-WEIGHT:bold;
        FONT-SIZE:medium;
        COLOR:purple"
        BorderStyle="Ridge"
        BorderWidth="2px"
        Text="Employee Name"/>
56:
57:      <asp:TableCell
        VerticalAlign="Middle"
58:
        HorizontalAlign="Center"
59:      style="FONT-
WEIGHT:bold;
60:
        FONT-SIZE:medium;
        COLOR:purple"
        BorderStyle="Ridge"
        BorderWidth="2px"
61:      <%#
DataBinder.Eval(this,
        "FullName")%>
62:      </asp:TableCell>
63:
64:    </asp:TableRow>
```

```
65:             <asp:TableRow>
66:                 <asp:TableCell/>
67:                 <asp:TableCell
68:                     HorizontalAlign="Right">
69:                     <asp:Button id="btnEdit"
70:                         runat="server"
71:                         onclick=
72:                             "OnEdit_Click"
73:                         Text="Edit"
74:                         CommandArgument=
75:                             '<%# DataBinder.Eval(this, "EmployeeId")%>' />
76:                     </asp:TableCell>
77:                 </asp:TableRow>
78:             </asp:Table>
```

Những dòng thật sự cần quan tâm ở đây là 25, 47, 61, và 72. Chú ý <%# DataBinder.Eval(this, "Employee") %> trên dòng 47. Dẫn hướng này tương tự với một khối script được biểu thị bởi cặp <% %>. Tuy nhiên, khối nhỏ này tạo nên sự kết nối dữ liệu giữa thuộc tính EmployeeId với cái được định nghĩa bên trong EmployeeView.ascx.cs. Kiểu kết nối dữ liệu này có thể được sử dụng thay cho việc phải thao tác từng thuộc tính của control. Lớp EmployeeView định nghĩa những thuộc tính có thể ràng buộc với các thành tố của asp. Trong suốt quá trình nạp trang chưa thành phần điều khiển EmployeeViewPagelet, tất cả các dẫn hướng kết nối dữ liệu sẽ được ước lượng. Kỹ thuật này làm cho việc phát triển dễ dàng hơn vì nó không cần quan tâm đến tất cả công việc ở bên trong.

Một trong những dòng thú vị nhất trong ví dụ 4.3-4 là dòng 72:

```
72: <asp:Button onclick="OnEdit_Click" runat="Server"
    Text="Edit"
    Commandargument="<%#DataBinder.Eval(this, "EmployeeId")%>"
```

Ở đây, ta khai báo một asp:Button cùng với sự kiện onclick được liên kết với phương thức OnEdit_Click. Bất cứ khi nào một control ở phía server được kích hoạt, một sự kiện PostBack được gửi đến server cùng với thông tin trạng thái chưa trong các trường ẩn trên form. asp:Button có thuộc tính commandargument có thể dùng để cung cấp thông tin phụ trợ, chẳng hạn ngữ cảnh của nút nhấn (button) được sử dụng như thế nào. Trong trường hợp này, EmployeeId đang được sử dụng như commandargument. Ví dụ 4.3-5 chứa mã cho EmployeePagelet dưới hình thức của một trang CodeBehind mang tên EmployeeView.cs.

Ví dụ 4.3-5 EmployeeViewPagelet.ascx.cs

```
1: namespace EmployeeBrowser.Pagelets
2: {
```

```
3:  using System;
4:  using System.Data;
5:  using System.Drawing;
6:  using System.Web;
7:  using System.Web.UI.WebControls;
8:  using System.Web.UI.HtmlControls;
9:
10: /// <summary>
11: /// Tóm tắt sự mô tả EmployeeListing.
12: /// </summary>
13: public abstract class EmployeeViewPagelet :
14:     System.Web.UI.UserControl
15: {
16:     protected System.Web.UI.WebControls.Button
17:         btnEdit;
18:
19:     private string employeeId;
20:     protected System.Web.UI.WebControls.Table
21:         EmpViewTable;
22:
23:
24:     public string EmployeeId
25:     {
26:         get
27:         {
28:             return employeeId;
29:         }
30:         set
31:         {
32:             employeeId = value;
33:         }
34:     }
35:
36:     public string FirstName
37:     {
38:         get
39:         {
40:             return firstName;
41:         }
42:         set
43:         {
44:             firstName = value;
45:         }
46:     }
47:
```

```
48:     public string LastName
49:     {
50:         get
51:         {
52:             return lastName;
53:         }
54:         set
55:         {
56:             lastName = value;
57:         }
58:     }
59:
60:     public string FullName
61:     {
62:         get
63:         {
64:             return string.Format("{0}, {1}",
65:                                 lastName, firstName );
66:         }
67:     }
68:     public Guid PictureId
69:     {
70:         get
71:         {
72:             return pictureId;
73:         }
74:         set
75:         {
76:             pictureId = new Guid( value.ToString()
77:                                   );
78:         }
79:     }
80:     public string ImageName
81:     {
82:         get
83:         {
84:             return string.Format("{0}.jpg",
85:                                 pictureId.ToString() );
86:         }
87:     }
88:     ///<summary>
89:     public EmployeeViewPagelet()
90:     {
91:         this.Init += new System.EventHandler(
92:             Page_Init);
```



```
92:     }
93:
94:     private void Page_Load(object sender,
95:                           EventArgs e)
96:     {
97:         //Người dùng đặt code khởi động trang ở đây
98:     }
99:     private void Page_Init(object sender,
100:                            EventArgs e)
101:    {
102:        //
103:        //CODEGEN: Cuộc gọi này được yêu cầu
104:        //bởi the ASP.NET Web Form Designer.
105:        //
106:        InitializeComponent();
107:    }
108:    ///<summary>
109:    ///Bộ xử lý sự kiện cho nút nhấn Edit. Phương
110:    ///thức này dẫn đến một trang soạn thảo và
111:    ///truyền EmployeeId như một đối số URL
112:    ///đến trang
113:    protected void OnEdit_Click(object sender,
114:                                EventArgs e)
115:    {
116:        Response.Redirect(
117:            string.Format("~/EmployeeEdit.
118:                           aspx?EmpId={0}",
119:                           btnEdit.CommandArgument));
120:
121:        #region Web Form Designer Generated code
122:        ///Phương thức yêu cầu người thiết kế hỗ trợ
123:        ///-dùng sửa đổi nội dung của phương thức
124:        ///này đối với việc soạn thảo code.
125:        ///</summary>
126:        private void InitializeComponent()
127:        {
128:            this.Load += new System.EventHandler(
129:                this.Page_Load);
130:        }
131:
```

```

130:      #endregion
131:  }
132: }
```

Lớp EmployeeViewPagelet thật sự chỉ là một tập hợp các thuộc tính k nhau cùng với bộ xử lý cho sự kiện click của asp:Button. Khi sự kiện click p sinh. EmployeeId được lấy từ thuộc tính CommandArgument. Kế đến, phương t Response.Redirect được sử dụng để chuyển sang trang Edit và EmployeeId c truyền theo kiểu đặc thù của phương thức GET trong HTTP.

Đến lúc này, chúng ta đã tạo được hai control. Mỗi cái sử dụng một cách đặt khác của ASP.NET để thực hiện chức năng theo yêu cầu.

2. CÁC TRANG ASPX

Bây giờ chúng ta có hai UserControl, đã đến lúc để tạo vài trang aspx c các control này. Trang đầu tiên là EmployeeEdit. Trang này cho phép thành p EmployeeViewPagelet tương tác qua lại với trang soạn thảo. Trang soạn thảo j cho phép thay đổi tên nhân viên, phòng ban, và tài hình.

2.1. EmployeeEdit

Một trong những yêu cầu đối với site EmployeeBrowser là khả năng s tháo thông tin của nhân viên và gắn hình vào. Thay cho việc xây d UserControl khác, chức năng này sẽ thực hiện bên trong ngữ cảnh của một tr aspx.

Hình 4.3-4 cho thấy trang EmployeeEdit với một mẫu tin (record) đã c soạn thảo.

	Employee Id: 1111111111
First Name:	Richard
Last Name:	Weeks
Department:	Development
<input type="button" value="Browse..."/>	
<input type="button" value="Update"/>	

Hình 4.3-4 Trang EmployeeEdit

Control đầu tiên để xem là danh sách sổ xuống (drop-down list) đượ dụng để giữ tên của các phòng ban khác nhau. Gán các giá trị cố định vào danh sách (list box) hay trình đơn sổ xuống (drop-down menu) là cần thiết. V 4.3-6 cho thấy khai báo của asp:dropdownList sẽ được kết nối vào một biến trong mã nguồn của EmployeeEdit.cs.

**Ví dụ 4.3-6 Đoạn trích dropdownlist từ trong EmployeeEdit.aspx**

```

1:<tr>
2:  <td>Department</td>
3:  <td>
4:    <asp:dropdownlist runat="Server"
      id="Departments" />
5:  </td>
6:</tr>

```

Thay vì khai báo các mục danh sách listitems trong trang aspx, mỗi WebControl trong ASP.NET cung cấp cách ràng buộc dữ liệu tương tự với các control của Windows Form. Trong đoạn mã cung cấp cho trang EmployeeEdit.aspx, ở đó sẽ tồn tại đoạn mã để thiết lập dữ liệu ràng buộc cho control. Các phòng ban được giữ trong một mảng danh sách như sau:

```

ArrayList Depts = new ArrayList();
FillDepts (Depts);
Departments.DataSource = Depts;
Departments.DataBind();

```

Thiết lập một thành phần ràng buộc dữ liệu không cần nhiều công sức của người phát triển. Dĩ nhiên, như bất kỳ WebControl nào khác, việc truy xuất thực tế luôn luôn là tùy chọn. Nếu việc ràng buộc dữ liệu không phù hợp yêu cầu, thì có thể viết mã bằng ngôn ngữ C# để điều khiển toàn bộ quá trình ràng buộc.

Control kế tiếp cần quan tâm là thành phần nhập liệu (input) có thuộc tính type được gán là "file". Nó cho phép người dùng duyệt chọn một file hình trên đĩa và tải lên Web server.

```
<input type="file" id="EmpImageFile"
       name="EmpImageFile" runat="server">
```

Đây không phải là thành phần (control) có kiểu asp.*, hãy xem nó có thể được truy xuất ở phía sau mã trang như thế nào? ASP.NET cung cấp việc ràng buộc các thành phần của HTML giống như các WebControl. Khi thẻ input được chỉ rõ với type = "file", bạn sẽ được sinh mã để tạo một biến có kiểu như sau:

```
System.Web.UI.HtmlControls.HtmlInputFile
```

Điều này cho phép trang mã truy xuất ngược lại các thành tố của HTML. Ví dụ 4.3-7 cho thấy chương trình C# cho EmployeeEdit.aspx.cs

Ví dụ 4.3-7 EmployeeEdit.aspx.cs

```

1:using System;
2:using System.Collections;
3:using System.ComponentModel;
4:using System.Data;
5:using System.Data.SqlClient;
6:using System.Drawing;

```

```
7: using System.Web;
8: using System.Web.SessionState;
9: using System.Web.UI;
10: using System.Web.UI.WebControls;
11: using System.Web.UI.HtmlControls;
12:
13: using Stingray.Data;
14:
15: namespace EmployeeBrowser
16: {
17:     /// <summary>
18:     /// Tóm tắt mô tả cho EmployeeEdit.
19:     /// </summary>
20:     public class EmployeeEdit : System.Web.UI.Page
21:     {
22:         protected System.Web.UI.WebControls.Image
23:             EmployeeImage;
24:         protected System.Web.UI.WebControls.Label
25:             EmployeeId;
26:         protected System.Web.UI.WebControls.TextBox
27:             FirstName;
28:         protected System.Web.UI.WebControls.TextBox
29:             LastName;
30:         Protected System.Web.UI.WebControls.
31:             DropDownList Department;
32:         protected System.Web.UI.WebControls.Button
33:             btnUpdate;
34:         protected System.Web.UI.HtmlControls.
35:             HtmlInputFile EmpImageFile;
36:
37:         public EmployeeEdit()
38:         {
39:             Page.Init += new System.EventHandler(
40:                 Page_Init);
41:
42:             //Người dùng đặt code khởi động trang ở đây
43:             //Cho phép kết nối dữ liệu
44:             DataBind();
45:
46:             if (!IsPostBack)
47:                 PopulateForm();
48:             else
49:                 UpdateEmployee();
50:
```

```
46:      }
47:
48:      private void PopulateForm()
49:      {
50:
51:          string EmpId = this.Request.QueryString[""
52:                                         EmpId"].ToString();
53:          SqlConnection dbCon = DBAccess.
54:                                         AcquireConnection();
55:          SqlDataAdapter cmd = new SqlDataAdapter(
56: string.Format("SELECT * FROM EMPLOYEE"
57: WHERE EMP_ID = {0}", EmpId), dbCon);
58:          DataSet dsResult = new DataSet();
59:          Employee emp = new Employee();
60:
61:          cmd.FillDataSet(dsResult, "EMPLOYEE");
62:          if( dsResult.Tables["EMPLOYEE"] != null)
63:              emp.FromDataRow(dsResult.Tables[""
64: EMPLOYEE"].Rows[0]);
65:
66:          //Gán giá trị vào các field
67:          this.EmployeeId.Text = emp.Id;
68:          this.FirstName.Text = emp.FirstName;
69:          this.LastName.Text = emp.LastName;
70:          this.EmployeeImage.ImageUrl =
71:             string.Format("~/images/employees/{0}."
72: jpg", emp.PictureId.ToString());
73:
74:          //Tạo danh sách phòng ban
75:          cmd.SelectCommand.CommandText =
76: "SELECT * FROM DEPARTMENT";
77:          cmd.FillDataSet(dsResult, "DEPARTMENT");
78:          Hashtable ht = new Hashtable();
79:
80:          foreach( DataRow row in dsResult.Tables[""
81: DEPARTMENT"].Rows)
82:          {
83:              int nValue = (int)row["DEPT_ID"];
84:              string Name = (string)row["NAME"];
85:              ht.Add( Name, nValue );
86:              this.lbDepartment.Items.Add( Name );
87:          }
88:
89:          this.Session["DEPT_MAPPING"] = ht;
90:          this.Session["CACHE:EMP"] = emp;
91:          //lưu nhân viên
92:      }
```




```
124:         }
125:
126:         //Cập nhật nhân viên
127:         DBAccess.Save( DBAccess.AquireConnection(
128:                         ),
129:                         string strUrl =
130:                         string.Format(
131:                             " EmployeeListing.aspx?DeptId={0}&
132:                             FirstName={1}&LastName={2},
133:                             emp.DeptId,emp.FirstName,
134:                             emp.LastName
135:                         );
136:
137:         Response.Redirect( strUrl );
138:
139:         private void Page_Init(object sender,
140:                               EventArgs e)
141:         {
142:             //
143:             //CODEGEN: Cuộc gọi này được yêu cầu
144:             //bởi the ASP.NET Web Form Designer.
145:             //
146:             InitializeComponent();
147:         }
148:         #region Web Form Designer Generated code
149:         ///<summary>
150:         ///
151:         ///dùng sửa đổi nội dung của phương thức
152:         ///này đối với việc soạn thảo code.
153:         ///
154:         private void InitializeComponent()
155:         {
156:             this.load += new
157:             System.EventHandler(this.Page_Load);
158:         }
159:     }
```

Trang EmployeeEdit được thiết kế để nhận một chuỗi truy vấn chứa Employee ID; Bạn hãy nhớ lại ví dụ về EmployeeViewPagelet và đoạn mã chưa bên trong phương thức OnEdit_Click trong ví dụ 4.3-5 có nội dung:



```
113:     protected void OnEdit_Click( object sender,
114:             EventArgs e )
115:     {
116:         Response.Redirect(
117:             string.Format( "-/EmployeeEdit.
118:                 aspx?EmpId={0}",
119:                 btnEdit.CommandArgument )
120:     }
121:
```

Khi nút nhấn Edit sẽ khiến phát sinh sự kiện nhấp chuột (click), thuộc tín CommandArgument chứa Employee ID. Employee ID này được dùng để tạo mì yêu cầu GET của HTTP. Dòng 115 cho thấy cú pháp để các tham số truyền 1 trang này sang trang khác.

Phương thức PopulateForm, ở ví dụ 4.3-7 truy xuất đến thành phần QueryString của đối tượng Request. Ở đây chúng ta bỏ qua việc kiểm tra lỗi dẫn hướng đến trang lỗi. Employee ID chứa tất cả các thông tin trạng thái cần thiết để đưa các điều khiển đã tạo vào bên trong trang.

Để lưu giữ thông tin, chẳng hạn như thông tin trạng thái hay các đối tượng có hai tùy chọn là dùng Session hoặc Cache. Trang EmployeeEdit sử dụng để lưu trữ mọi loại dữ liệu cần thiết. Một Session vẫn có thể tồn tại nếu IIS bị hỏng, tiến trình công việc khởi tạo lại, nhưng Cache thì không. Khi quyết định thông tin nên được lưu trữ ở đâu, bạn sẽ cần phải khảo sát mức độ cần thiết của dữ liệu bảo đảm rằng nó an toàn.

Cả hai đối tượng Session và Cache đều chứa một bảng băm (hash) để ái xạ một khoá vào đối tượng mà bạn muốn lưu trữ. Trang EmployeeEdit lưu trữ bảng băm (hash) lắn đối tượng Employee trong Session.

Khi nút nhấn Update phát sinh sự kiện click (nhấp chuột), trang được nạp lại cùng với việc gán cho IsPostBack là true. Page_Load kiểm tra điều kiện này gọi UpdateEmployee thay cho PopulateForm.

2.2. EmployeeListing

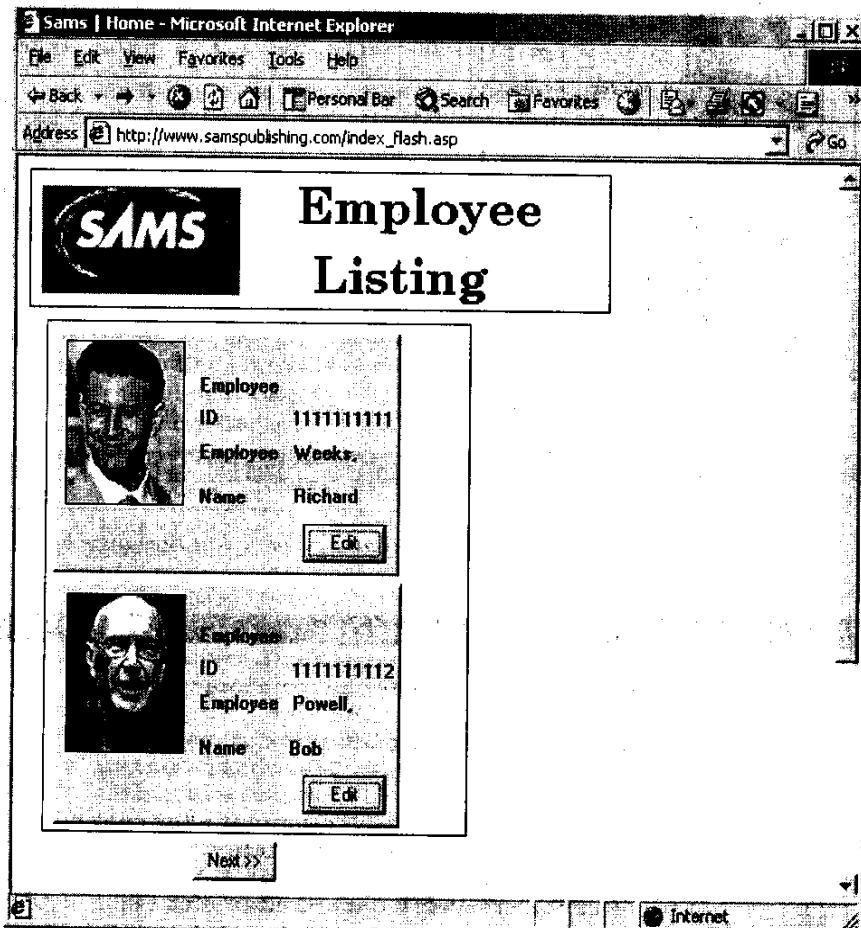
Chúng ta cũng đang bắt đầu với ứng dụng Web EmployeeBrowser. Sau khi hoàn thành HeaderControl, EmployeeView, và trang EmployeeEdit, đã đến lúc phải tạo trang Listings. Trang Listings hiển thị một hoặc nhiều thành phần điều khiển EmployeeView và kể cả việc phân trang tới lui duyệt qua danh sách nhân viên.

Hình 4.3-5 cho thấy thành phần điều khiển cơ bản của danh sách Listings hiển thị hai dòng cho mỗi trang màn hình. Bạn cũng nên chú ý đến nút nhấp Next>> ở cuối của form. Khi việc phân trang xảy ra, EmployeeListing.aspx sẽ định xem có các mẫu tin trước đó hay không, và hiển thị một nút nhấn <<Prev |

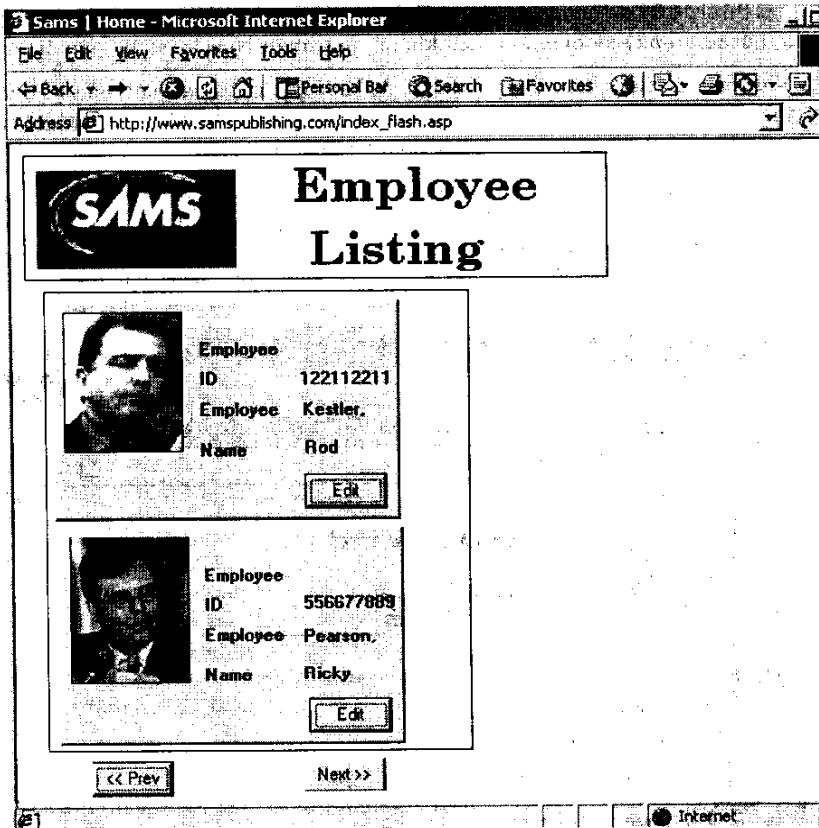
thích hợp (hình 4.3-5). Tương tự như vậy đối với nút nhấn Next>>. Nếu có thêm mẫu tin, nút nhấn này sẽ hiện lên; nếu không, nút nhấn Next>> sẽ bị giấu đi.

Như bạn có thể thấy, ở đây không đòi hỏi bất cứ ràng buộc nào về kích thước ảnh. Cách đơn giản nhất là xác định kích thước ảnh cho phép và loại bỏ bất kỳ ảnh nào không phù hợp với các tiêu chuẩn về kích thước.

Vậy trang EmployeeListing hiển thị EmployeeViewPagelets như thế nào? Khá đơn giản. Điều bí mật ở đây là phải sử dụng điều khiển asp:repeater. EmployeeListing sử dụng repeater để hiển thị danh sách của các nhân viên mỗi trang gồm một số người. Ví dụ 4.3-8 cho thấy việc ràng buộc với điều khiển EmployeeView.



Hình 4.3-5 EmployeeListing hiển thị hai nhân viên



Hình 4.3-6 EmployeeListing hiển thị hai nhân viên và nút Prev được bâ

Ví dụ 4.3-8 Đoạn trích từ EmployeeListing.aspx

```

1: <asp:repeater id=EmployeeRepeater runat="server">
2:   <template name="ItemTemplate">
3:     <tr>
4:       <td>
5:         <stringray:EmployeeView runat="server"
6:           EmployeeId='<%# DataBinder.Eval(
7:             Container.DataItem, "Id" ) %>'
8:             FirstName='<%# DataBinder.Eval(
9:               Container.DataItem, "FirstName" ) %>
10:             LastName='<%# DataBinder.Eval(
11:               Container.DataItem, "LastName" ) %>'>
12:       </td>
13:     </tr>
14:   </template>
15: </asp:repeater>

```

```

9:           Container.DataItem, "LastName" ) %>
10:          PictureId='<%# DataBinder.Eval(
11:            Container.DataItem,"PictureId" ) %>'
12:          </EmployeeBrowser:EmployeeViewControl>
13:        </td>
14:      </tr>
15:    </ItemTemplate>
16:  </asp:repeater>

```

EmployeeListing sử dụng điều khiển ràng buộc dữ liệu WebControl để thiết lập các thuộc tính khác nhau của thành phần EmployeeViewPagelet. Chú ý rằng mỗi thuộc tính của thành phần EmployeeViewPagelet được gán giá trị qua việc sử dụng

DataBinder.Eval(Container.DataItem, X)

Ở đây X là tên của một trường trong bảng dữ liệu nguồn (datasource). Mã nguồn ở phía sau trang sẽ tạo một mảng ArrayList chứa các đối tượng Employee ràng buộc danh sách này với thành phần điều khiển EmployeeRepeater. Sử dụng kỹ thuật này sẽ khiến cho việc ràng buộc các giá trị thuộc tính rất nhẹ nhàng! Trên thực tế, đoạn mã ở ví dụ 4.3-8 là phần quan trọng thực sự của trang aspx. Phần còn lại chỉ là các mã trang trí HTML (kể cả HeaderControl).

Cho đến lúc này cũng như với mọi trang aspx, trang EmployeeListing thực hiện việc phân trang trong file nguồn C#. Sau khi dữ liệu được liên kết vào repeater control, chỉ mục hiện hành được đẩy vào ngăn xếp và ngăn xếp đó được lưu vào Session. Khi nút *Next* hay *Prev* được nhấn, chỉ mục được tính cho mục kế tiếp hoặc cho chỉ mục trước đó được lấy ra khỏi ngăn xếp. Ví dụ 4.3-9 trình bày mã nguồn của trang EmployeeListing.

Ví dụ 4.3-9 EmployeeListing.aspx.cs

```

1: using System;
2: using System.Collections;
3: using System.ComponentModel;
4: using System.Data;
5: using System.Data.SqlClient;
6: using System.Drawing;
7: using System.Web;
8: using System.Web.SessionState;
9: using System.Web.UI;
10: using System.Web.UI.WebControls;
11: using System.Web.UI.HtmlControls;
12:
13: using Stingray.Data;
14:
15: namespace EmployeeBrowser
16: {
17:
18:     class ViewState

```



```
19:     {
20:         public static string VIEW_STATE_SESSION_KEY
21: = "EMPLOYEE_VIEWER_STATE";
22:         public DataSet m_EmployeeCache;
23:         public int    m_RowCount = 0;
24:         public int    m_MaxPerPage = 5;
25:         public int    m_CurrentIndex = 0;
26:         public Stack  m_IdxStack = new Stack();
27:     }
28:     /// <summary>
29:     /// Tóm tắt mô tả EmployeeListing.
30:     /// </summary>
31:     public class EmployeeListing :
System.Web.UI.Page
32:     {
33:         protected System.Web.UI.WebControls.Button
            btnNext;
34:         protected System.Web.UI.WebControls.Button
            btnPrev;
35:         protected System.Web.UI.WebControls.
            Repeater   EmployeeRepeater;
36:         private ArrayList   m_EmployeeList;
37:         private ViewState  m_VViewState;
38:
39:
40:
41:         public EmployeeListing()
42:         {
43:             Page.Init += new System.EventHandler(
                Page_Init);
44:         }
45:
46:         private void Page_Load(object sender,
            EventArgs e)
47:         {
48:             if (!IsPostBack)
49:             {
50:
51:                 int      DeptId = 0;
52:                 string   FirstName;
53:                 string   LastName;
54:
55:
56:                 m_VViewState = new ViewState();
57:                 this.Session.Add(
                    ViewState.VIEW_STATE_SESSION_KEY,
                    m_VViewState);
```

```
58:
59:         GetParams( out FirstName, out LastName,
60:                     ref DeptId,
61:                     ref m_ViewerState.m_MaxPerPage );
62:
63:
64:         //lưu chỉ mục hiện hành
65:         m_ViewerState.m_IdxStack.Push( 0 );
66:         int Index = this.CalcNextIndex( );
67:         m_ViewerState.m_CurrentIndex = Index;
68:         this.DisplayData( 0,
69:                           m_ViewerState.m_CurrentIndex );
70:         UpdateNavigationButtons( );
71:
72:     } else {
73:
74:         m_ViewerState =
75:             (ViewState)this.Session[ ViewState.VIEW_STATE_SESSION_KEY ];
76:     }
77:
78:     private void Page_Init(object sender,
79:                            EventArgs e)
80:     {
81:         //
82:         //CODEGEN: Lời gọi này được yêu cầu
83:         //bởi ASP.NET Web Form Designer.
84:         //
85:         InitializeComponent();
86:
87:         protected void GetParams(out string
88:                               FirstName, out string LastName,
89:                               ref int DeptId, ref int MaxPerPage )
90:
91:         {
92:             DeptId = this.Request.QueryString["
```

```
92:             "" : this.Request.QueryString["  
93:                 FirstName"].ToString();  
94:             LastName = this.Request.QueryString[  
95:                 "LastName"] == null ?  
96:                     "" : this.Request.QueryString["  
97:                         LastName"].ToString();  
98:             MaxPerPage = this.Request.QueryString[  
99:                 "MaxPerPage"] == null ?  
100:                  5 : Int32.FromString(  
101:                      this.Request.QueryString["  
102:                         MaxPerPage"].ToString());  
103:         }  
104:     protected void LoadDataSet(string FName,  
105:                                 string LName, int DeptId)  
106:     {  
107:         if( DeptId != -1 )  
108:         {  
109:             m_VViewState.m_EmployeeCache =  
110:                 Search.Find(  
111:                     DBAccess.AquireConnection(),  
112:                     DeptId, FName, LName);  
113:         } else {  
114:             m_VViewState.m_EmployeeCache =  
115:                 Search.Find(  
116:                     DBAccess.AquireConnection(),  
117:                     FName, LName);  
118:         }  
119:         m_VViewState.m_RowCount =  
120:             m_VViewState.m_EmployeeCache.  
121:             Tables["EMPLOYEE"].Rows.Count;  
122:     }  
123:     protected void DisplayData( int StartIndex,  
124:                                 int StopIndex )  
125:     {  
126:         m_EmployeeList = new ArrayList();  
127:         for( int i = StartIndex; i < StopIndex; i++ )  
128:         {  
129:             Employee e = new Employee();  
130:             e.FromDataRow( m_VViewState.
```

```
        m_EmployeeCache.Tables["  
EMPLOYEE"].Rows[i] );  
124:            m_EmployeeList.Add( e );  
125:        }  
126:  
127:            //Ràng buộc dữ liệu.  
128:            EmployeeRepeater.DataSource=  
m_EmployeeList;  
129:            EmployeeRepeater.DataBind( );  
130:  
131:  
132:        }  
133:  
134:        protected void OnNext_Click( object sender,  
EventArgs e )  
135:        {  
136:            //lưu chỉ mục hiện hành  
137:            int nLast = m_ViewerState.m_CurrentIndex;  
138:            m_ViewerState.m_IdxStack.Push( nLast );  
139:            m_ViewerState.m_CurrentIndex =  
CalcNextIndex( );  
140:            DisplayData( nLast,  
m_ViewerState.m_CurrentIndex );  
141:            UpdateNavigationButtons( );  
142:  
143:        }  
144:  
145:        protected void OnPrev_Click( object sender,  
EventArgs e )  
146:        {  
147:            int nStop =  
(int)m_ViewerState.m_IdxStack.Pop( );  
148:            int nStart =  
(int)m_ViewerState.m_IdxStack.Peek( );  
149:            DisplayData( nStart, nStop );  
150:            m_ViewerState.m_CurrentIndex = nStop;  
151:            UpdateNavigationButtons( );  
152:        }  
153:  
154:        protected int CalcNextIndex( )  
155:        {  
156:            return (m_ViewerState.m_CurrentIndex +  
m_ViewerState.m_MaxPerPage) >  
m_ViewerState.m_RowCount  
157:            ? m_ViewerState.m_RowCount :  
(m_ViewerState.m_CurrentIndex +  
m_ViewerState.m_MaxPerPage);  
158:
```



```
159:     }
160:
161:     protected void UpdateNavigationButtons()
162:     {
163:         this.btnAdd.Visible =
164:             m_ViewerState.m_CurrentIndex <
165:                 m_ViewerState.m_RowCount;
166:         this.btnDelete.Visible =
167:             (int)m_ViewerState.m_IdxStack.Peek() !=
168:                 0;
169:     }
170: 
171: #region Web Form Designer Generated code
172: //Phương thức cần thiết cho việc hỗ trợ
173: ///đứng sửa đổi nội dung của
174: ///phương thức này đổi với việc soạn thảo
175: 
176: ///////////////////////////////////////////////////////////////////
177: private void InitializeComponent()
178: {
179:     this.load +=
180:         new System.EventHandler(this.Page_Load);
```

Để xem việc ràng buộc dữ liệu được thiết lập như thế nào, hãy xem phương thức DisplayData ở dòng 115. Mảng ArrayList đã được khởi dụng và các đối tượng Employee được thêm vào dựa trên các đối số StartIndex và StopIndex truyền cho phương thức. Sau khi ArrayList được lấp đầy các đối tượng, thành phần EmployeeRepeater được ràng buộc vào ArrayList. Việc ràng buộc này được thực hiện qua lệnh:

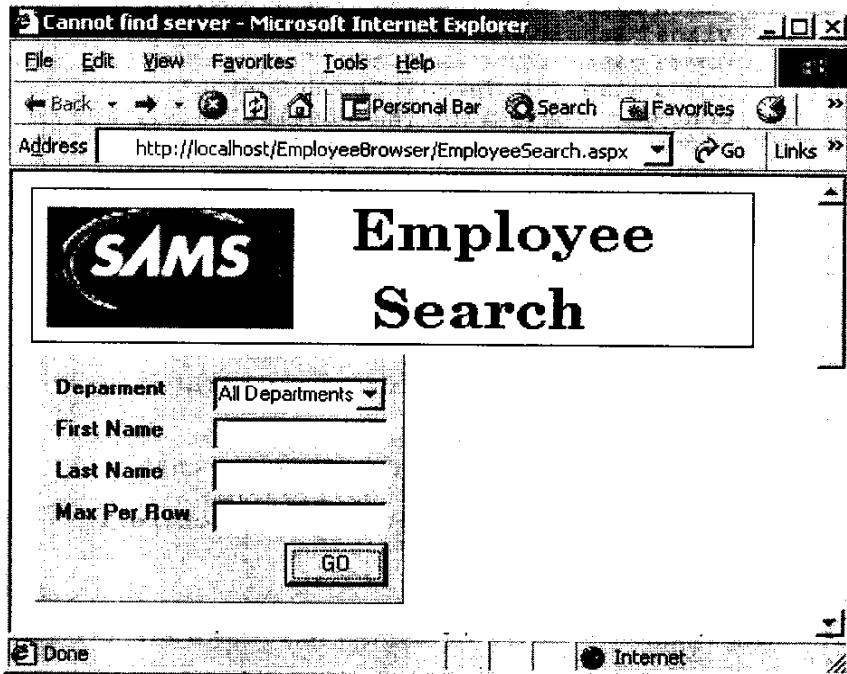
```
<%# DataBinder.Eval(Container.DataItem, X) %>
```

Để duy trì thông tin trạng thái của khung nhìn, chúng ta tạo ra một lớp nhỏ ViewState. ViewState duy trì một DataSet tương ứng với các tham số được truyền vào trang. Ngoài ra ViewState cũng duy trì chỉ mục hiện hành và các mục ngẫu nhiên để chứa các chỉ mục trước đó. Ngẫu nhiên này được sử dụng nhằm giúp cho việc đánh số trang được thuận tiện. Phần mã còn lại của EmployeeListing gồm có việc nạp dữ liệu, tính chỉ mục, và xử lý sự kiện OnClick cho các nút btnNext và btnPrev của WebControl.

2.3. Trang tìm kiếm: Nơi bắt đầu tìm thông tin

Bên trong ứng dụng Web EmployeeBrowser chúng ta cần một điểm tiếp cận và cung cấp thông tin. Một trang tìm kiếm đơn giản sẽ phù hợp với yêu cầu cho phép người dùng chọn phòng ban, họ tên, số trang của danh sách cần xem. Các chọn lựa này sau đó được truyền như tham số đến trang EmployeeListing.aspx xử lý.

Giao diện trang Search được trình bày như hình 4.3-7 sẽ được dùng để bắt đầu ứng dụng Web EmployeeBrowser.



Hình 4.3-7 Trang tìm kiếm (search).

Cho đến lúc này đây được xem là phần nhỏ nhất trong tất cả các ví dụ đã trình bày. Về cơ bản, trang này cho phép người dùng chọn phòng ban, họ tên, và số của dòng liệt kê trong danh sách ở mỗi trang. Một lần nữa, thay cho việc tạo một pagelet được dẫn xuất từ UserControl, trang aspx được dùng để tập hợp dữ liệu nhập đầu vào và tạo một lời yêu cầu ở dạng URL gọi trang EmployeeListing. Ví dụ 4.3-10 chứa mã nguồn HTML cho trang EmployeeSearch.aspx.

Ví dụ 4.3-10 EmployeeSearch.aspx

```
1: <%@ Register TagPrefix="EmployeeBrowser"
```

```
TagName="HeaderControl"
Src="-/Pagelets/HeaderPagelet.ascx" %>
2:<%@ Page language="c#"
Codebehind="EmployeeSearch.aspx.cs"
AutoEventWireup="false"
Inherits="EmployeeBrowser.EmployeeSearch" %>
3:<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
4:<HTML>
5:  <HEAD>
6:    <meta name="GENERATOR" Content="Microsoft
Visual Studio 7.0">
7:    <meta name="CODE_LANGUAGE" Content="C#">
8:    <meta name="vb_defaultClientScript"
content="JavaScript (ECMAScript)">
9:    <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/
intellisense">
10:   <HEAD>
11:   <body MS_POSITIONING="GridLayout">
12:     <form id="EmployeeSearch" method="post"
runat="server">
13:
14:   <!--
15:       Chèn một header control tùy biến
16:   -->
17:   <EmployeeBrowser:HeaderControl
runat="server" Text="Employee Search"
ImageUrl="-/images/logos/logo.jpg"
ID="Headercontrol1" NAME="Headercontrol1"
/>
18:
19:   <!--
20:       Bắt đầu phần giao diện của trang tìm kiếm
21:   -->
22:   <table style="BORDER-TOP-STYLE: outset;
BORDER-RIGHT-STYLE: outset;
BORDER-LEFT-STYLE: outset;
BACKGROUND-COLOR: lightgrey;
BORDER-BOTTOM-STYLE: outset" >
23:     <tbody>
24:       <tr>
25:         <td>Department</td>
26:         <td>
27:           <asp:dropdownlist id="
1bDepartments" runat="server" />
28:         </td>
29:       </tr>
```

```

30:
31:          <tr>
32:              <td>First Name</td>
33:              <td>
34:                  <asp:textbox runat="server"
35:                      id="txtFirstName" />
36:              </td>
37:          </tr>
38:          <tr>
39:              <td>Last Name</td>
40:              <td>
41:                  <asp:textbox runat="server"
42:                      id="txtLastName" />
43:              </td>
44:          </tr>
45:          <tr>
46:              <td>Max Per Row</td>
47:              <td>
48:                  <asp:textbox runat="server"
49:                      id="txtMaxPerRow" />
50:              </td>
51:          </tr>
52:          <tr>
53:              <td>!--Để một khoảng trắng để đặt nút
54:                  nhấn vào cột thứ hai --></td>
55:              <td align=right>
56:                  <asp:button id="btnGO"
57:                      onclick="OnSearch_Click"
58:                      text="GO!" runat="server" />
59:              </td>
60:          </tr>
61:      </tbody>
62:  </table>
63: </form>
64: </body>
65: </HTML>

```

Như bạn có thể thấy, việc tạo giao diện khá đơn giản. Để ý sự pha trộn của chuẩn HTML và ASP.NET. Nếu không có nhu cầu cần đến các control ở phía server (server-side), thì những thành phần HTML chuẩn hoàn toàn phù hợp với mục đích của chương trình. Sự thuận lợi của các thành phần ở phía server do ASP.NET cung cấp là khả năng điều khiển thậm chí ngay cả phía trình duyệt.

Cho đến lúc này mã của EmployeeSearch.cs (xem ví dụ 4.3-11) vẫn áp dụng các kỹ thuật xây dựng trang trước đây. Đối tượng Section được dùng để chứa thông

tin dùng cho việc ánh xạ, asp:dropdownbox sử dụng ràng buộc dữ liệu, và PostBack được tiến hành, dữ liệu thành phần được lấy ra từ lệnh HTTP GET trang EmployeeListing.aspx.

Ví dụ 4.3-11 EmployeeSearch.cs

```
1: using System;
2: using System.Collections;
3: using System.ComponentModel;
4: using System.Data;
5: using System.Data.SqlClient;
6: using System.Drawing;
7: using System.Web;
8: using System.Web.SessionState;
9: using System.Web.UI;
10: using System.Web.UI.WebControls;
11: using System.Web.UI.HtmlControls;
12:
13: using Stingray.Data;
14:
15: namespace EmployeeBrowser
16: {
17:     /// <summary>
18:     /// Tóm tắt mô tả EmployeeSearch.
19:     /// </summary>
20:     public class EmployeeSearch :
21:         System.Web.UI.Page
22:     {
23:         protected System.Web.UI.WebControls.
24:             DropDownList Departments;
25:         protected System.Web.UI.WebControls.
26:             TextBox FirstName;
27:         protected System.Web.UI.WebControls.
28:             TextBox LastName;
29:         protected System.Web.UI.WebControls.
30:             TextBox MaxPerRow;
31:         protected System.Web.UI.WebControls.
32:             Button btnGO;
33:     }
34: }
```



```
28:     protected ArrayList DepartmentList;
29:     protected Hashtable htDeptMapping;
30:
31:
32:     public EmployeeSearch()
33:     {
34:         Page.Init += new System.EventHandler(
35:             Page_Init);
36:
37:         protected void Page_Load(object sender,
38:                         System.EventArgs e)
39:         {
40:             if( !this.IsPostBack )
41:             {
42:                 htDeptMapping = new Hashtable();
43:                 LoadData();
44:
45:                 this.Session.Add( "SEARCH_HT_MAPPING",
46:                     htDeptMapping );
47:                 Departments.DataSource =
48:                     DepartmentList;
49:                 Departments.DataBind();
50:             }
51:             else
52:             {
53:                 htDeptMapping = (Hashtable)this.
54:                     Session["SEARCH_HT_MAPPING"];
55:             }
56:         }
57:         // 
58:         //CODEGEN: Lời gọi này được yêu cầu
//bởi ASP.NET Web Form Designer.
```

```
59:         //  
60:         InitializeComponent();  
61:     }  
62:  
63:     protected void OnSearch_Click( object  
             sender, EventArgs e )  
64:     {  
65:  
66:         //Lấy các đối số được chọn  
67:         if(this.htDeptMapping == null)  
68:         {  
69:             Response.Redirect( "EmployeeListing.  
aspx?MaxPerPage=5" );  
70:         }  
71:         else  
72:         {  
73:  
74:             int DeptId = (int)this.htDeptMapping[  
Departments.SelectedItem.Text];  
75:             string FirstName = this.FirstName.Text;  
76:             string LastName = this.LastName.Text;  
77:  
78:             int iMaxPerRow = 5;  
79:  
80:             if( MaxPerRow.Text != "" )  
81:             {  
82:                 iMaxPerRow = Int32.Parse(  
this.MaxPerRow.Text);  
83:             }  
84:  
85:  
86:             //xây dựng request URL  
87:             string request;  
88:  
89:             if( DeptId != 0)  
90:             {  
91:                 object[] args = { DeptId, FirstName,  
LastName, iMaxPerRow };
```

```
92:             request = string.Format("DeptId={0}&
93:             FirstName={1}&LastName={2}&
94:             MaxPerPage={3}", args);
95:         } else {
96:             object[] args = { FirstName, LastName,
97:             iMaxPerRow };
98:             request = string.Format(
99:             "FirstName={0}&LastName={1}&
100:             MaxPerPage={2}", args);
101:         }
102:     protected void LoadData()
103:     {
104:
105:         SqlConnection dbCon = DBAccess.
106:         AquireConnection();
107:         SqlDataAdapter cmd = new SqlDataAdapter(
108:         "SELECT * FROM DEPARTMENT", dbCon );
109:         DataSet dsResult = new DataSet();
110:         cmd.Fill( dsResult, "DEPARTMENT" );
111:
112:         DepartmentList = new ArrayList();
113:         DepartmentList.Add( "All Departments" );
114:         htDeptMapping.Add( "All Departments", 0 );
115:
116:         foreach( DataRow row in dsResult.
117:             Tables["DEPARTMENT"].Rows )
118:         {
119:             Department d = new Department();
120:             d.FromDataRow( row );
121:             DepartmentList.Add( d.Name );
122:             htDeptMapping.Add( d.Name, d.Id );
123:         }
```

```

121:
122:     dbCon.Close( );
123:     dbCon.Dispose( );
124:     dsResult.Dispose( );
125: }
126:
127:
128: #regoin Web Form Dsigner generated code
129: ///<summary>
130: ///
131: ///đừng sửa đổi nội dung của phương thức này
132: ///
133: ///<summary>
134: private void InitializeComponent()
135: {
136:     this.load +=
137:         new System.EventHandler(this.Page_Load);
138: }
139: }
140: }

```

Với việc hoàn thành trang EmployeeSearch, ứng dụng EmployeeBrowser sẵn sàng hoạt động. Chương trình không đi sâu vào thành phần giao diện như bạn vẫn có thể tự mình thực hiện công việc còn lại này.

3. KẾT CHƯƠNG

ASP.NET WebForms mang lại cho môi trường phát triển Web truyền thống một luồng sinh khí mới. WebFroms chú ý đến các ứng dụng Web động và mã dù diễn đạt hiệu quả, dễ phát triển và phù hợp với nhu cầu ngày càng cao của Internet hơn. Với trên 45 thành phần ASP chuyển giao cho Visual Studio. Net bạn có thể xây dựng các ứng dụng Web (Web Applications) với tất cả chức năng mà người dùng cuối có thể hài lòng tối đa cũng như thỏa mãn những người dùng ứng dụng desktop truyền thống trước đây. Khả năng tạo các thành phần có thể sử dụng lại cũng sẽ tăng năng suất của người phát triển và khiến ứng dụng dễ bảo trì.

Chương 4.4

CÁC DỊCH VỤ WEB (WEBSERVICES)

Các vấn đề chính sẽ được đề cập đến

- ✓ Dịch Vụ Echo
- ✓ Trả Về Kiểu Người Dùng Định Nghĩa
- ✓ Các Thuộc Tính XML

Có rất nhiều kỹ thuật khác nhau cho việc truy xuất dữ liệu bằng các ứng dụng phân bố (distributed application) từ nhiều năm nay. Từ cơ chế gọi hàm từ xa (Remote Procedure Protocol - RPC) như XML-RPC, đến những đối tượng phân bố đang phát triển mạnh như CORBA và DCOM. Mô hình dịch vụ Web (WebServices) dựa trên giả thuyết đơn giản của việc mở rộng cơ chế XML-RPC để cho phép các cuộc gọi hàm từ xa qua Internet dùng các nghi thức truyền thông mở như HTTP thông qua các kết nối TCP/IP.

.NET hỗ trợ việc gọi WebServices thông qua HTTP POST, HTTP GET, và nghi thức truy xuất đối tượng đơn giản (Simple Object Access Protocol - SOAP). Trước khi có sự hỗ trợ WebServices trong .NET, để phát triển ứng dụng liên kết một máy chủ (host), một proxy, và máy khách (client) thông qua dịch vụ của Web là một công việc khó khăn. Microsoft đã đưa ra bộ công cụ SOAP cho Visual Studio 6 cho phép người phát triển đưa một đối tượng COM vào Internet. Bộ công cụ này chưa đạt tới mức hỗ trợ tốt như trong .NET và C#. Việc phát triển một WebService đòi hỏi nhiều hơn việc định nghĩa một tập các phương thức dùng chung để đưa ra cho các máy khách client, cùng với các kiểu trả về thích hợp áp dụng WebMethodAttributes đơn giản cho các phương thức đó.

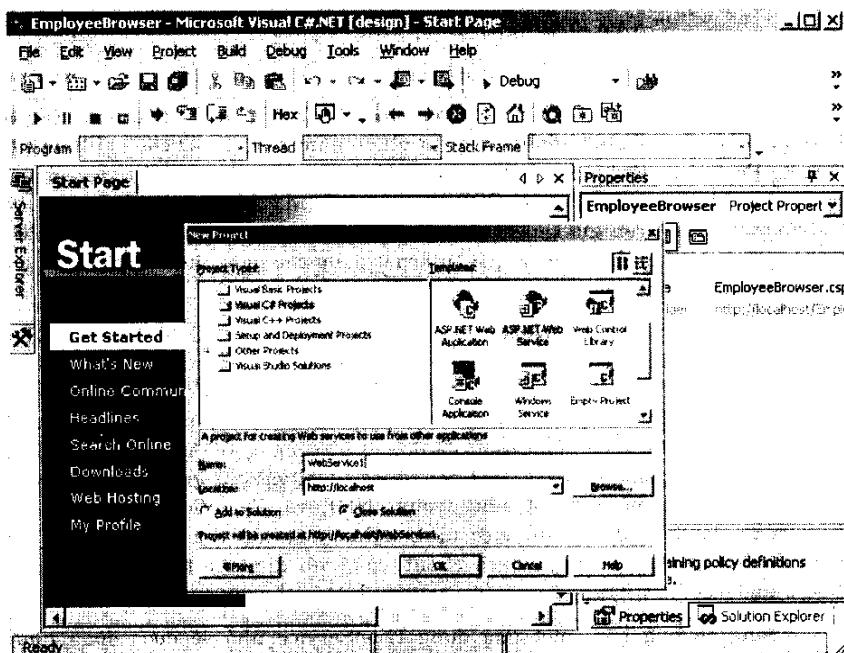
Trong chương này, một ứng dụng EchoService nhỏ sẽ được dùng như là điểm khởi đầu để giúp bạn làm quen với khái niệm WebServices và sau cùng chúng ta sẽ thêm một giao tiếp WebService vào EmployeeBrowser ASP WebApplication.

1. DỊCH VỤ ECHO (ECHO SERVICE)

Xây dựng các ứng dụng client/server giống như vấn đề con gà với quả trứng. Bạn cần một sever để chạy thử client và một client để chạy thử server. Một echo server sẽ phản hồi lại bất cứ thông tin mà nó nhận được từ client. Các Echo server cho phép một ứng dụng client chạy thử kết nối của nó và đảm bảo rằng dữ liệu đang gửi đi là đúng. Thực thi một EchoService dùng WebServices .NET sẽ là một sự khởi đầu tốt lành.

VS.NET hỗ trợ việc tạo WebServices trực tiếp từ IDE. VS.NET sẽ kết nối vào server IIS chuyên biệt dùng FrontPage Extension để mở một thư mục Web, giống như WebApplication được phát triển ở các chương trước. Hình 4.4-1 cho thấy một hộp thoại dự án (project) mới dùng để tạo một WebService.

WebServices, được tạo bởi VS.NET, gồm 2 tập tin – tập tin asmx và tập tin nguồn C#. Tuy nhiên, chỉ có tập tin asmx là cần thiết vì tất cả mã nguồn C# có thể ở trong tập tin asmx. Nhưng đây không phải là hướng đi được dùng để phát triển các ví dụ trong chương này. Ví dụ 4.4-1 cho thấy mã nguồn C# cho WebService EchoService. Ngoài ra, chúng tôi cũng đã đổi tên của lớp mặc định từ Service1 thành EchoServer cho dễ hiểu.



Hình 4.4-1 WebService VS.NET project

Ví dụ 4.4-1 Mã nguồn EchoService.asmx

```

1: using System;
2: using System.Collections;
3: using System.Data;
4: using System.ComponentModel;
5: using System.Diagnostics;
6: using System.Web;
7: using System.Web.Services;
8:
9: namespace EchoService
10: {

```



```
11: /// <summary>
12: /// Summary description for EchoServer.
13: /// </summary>
14: public class EchoServer :
15:     System.Web.Services.WebService
16: {
17:     public EchoServer ()
18:     {
19:         //CODEGEN: This call is required by the ASP.NET
20:         // Web Services Designer
21:         InitializeComponent();
22:     }
23: 
24:     #region Component Designer generated code
25:     /// <summary>
26:     /// Required method for Designer support - do not
27:     //modify the contents of this method with the code
28:     //editor.
29:     /// </summary>
30:     private void InitializeComponent()
31:     {
32:     }
33:     #endregion
34: 
35:     /// <summary>
36:     /// Clean up any resources being used.
37:     /// </summary>
38:     protected override void Dispose(bool disposing)
39:     {
40:         [WebMethod]
41:         public string Echo( string Message ) {
42:             return Message;
43:         }
44: 
45:         // WEB SERVICE EXAMPLE
46:         // The HelloWorld() example service returns the
47:         // string Hello World
48:         // To build, uncomment the following lines then save
49:         // and build the project
50:         //To test this WebService, press F5
51:         // [WebMethod]
52:         //public string helloWorld()
53:         //{
54:             // return "Hello World";
```



```
54:      //}
55:    }
56: }
```

EchoService.asmx trong ví dụ 4.4-1 minh họa một WebService cơ bản. Lớp EchoServer kế thừa từ lớp cơ sở System.Web.Services.WebService. Lớp cơ sở WebService không đòi hỏi lớp dẫn xuất phải ghi chồng bất kỳ phương thức nào; thay vào đó, lớp cơ sở WebService chỉ cung cấp những cài đặt cần thiết để hỗ trợ việc gọi phương thức.

Bản thân lớp EchoServer chỉ cung cấp một phương thức duy nhất cho phía trình khách client triệu gọi- đó là phương thức Echo. Lưu ý tới thuộc tính được dùng trong phương thức này. Thuộc tính này [WebMethod] được dùng vào lúc service thực thi. Mục đích là để xác định các phương thức mà lớp này cho phép các trình khách client được gọi. Nó tương tự như cách mà lớp DBAccess thực hiện trong ví dụ EmployeeBrowser, vào thời điểm service thực thi, ta dùng Reflection API để xác định phương thức có đặc tính WebMethod và gọi nó với những thông số thích hợp.

Để IIS có thể sử dụng được WebService, chúng ta cần có sẵn tập tin asmx trong thư mục Web ảo chứa EchoService. Tập tin asmx này được dùng để nối đoạn mã WebService vào tập tin .dll và lớp cung cấp WebService thực sự. Việc kết nối cơ bản của tập tin asmx này bằng mã C# được trình bày trong ví dụ 4.4-2 sau:

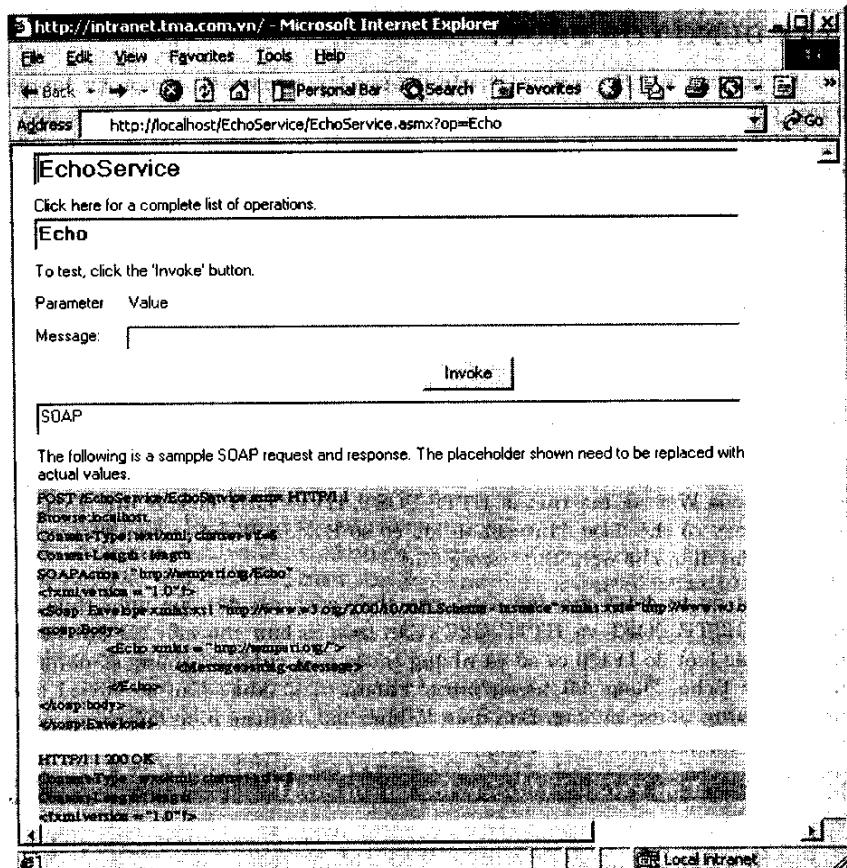
Ví dụ 4.4-2 EchoService.asmx

```
1: <%@ WebService Language="c#"
2:   Codebehind="EchoService.asmx.cs"
3:   Class="EchoService.EchoService" %>
4:
```

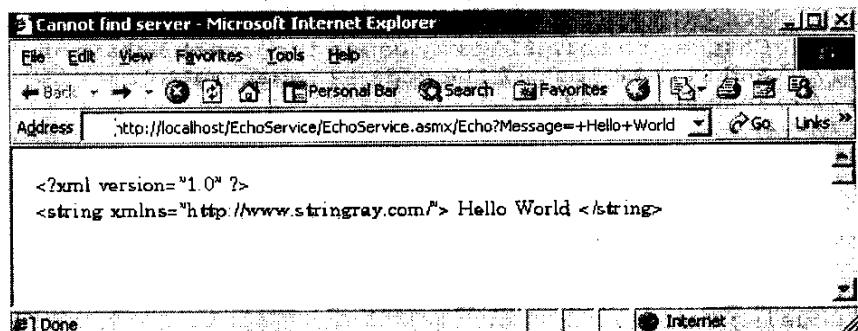
Bạn có thể thấy tập tin này rất đơn giản. Đối với các trang aspx, chúng ta đặt thêm chỉ dẫn ngôn ngữ dùng viết mã là C#, chỉ thị Codebehind cho biết trang asmx cài đặt mã, chỉ thị Class chỉ định tên chuẩn của lớp cung cấp việc cài đặt WebService.

Vì EchoService này được cài đặt như là tập tin có mã nguồn viết bằng C# và nó không phải là script nội (inline), tập tin EchoService.asmx.cs cần được biên dịch ra tập tin .dll sau đó copy vào thư mục bin của thư mục ảo hiện hành dùng cho dịch vụ EchoService. VS.NET sẽ tự động làm điều này trong quá trình biên dịch. Việc kiểm tra dịch vụ này chỉ đòi hỏi truy xuất vào tập tin asmx với trình duyệt Web Internet Explorer, như ở trong hình 4.4-2.

Hệ thống WebServices tự tạo ra một form HTML đơn giản, chúng ta sẽ dùng form này để kiểm tra các phương thức của WebService. Chọn ô nhập liệu và nhập một đoạn văn bản, sau đó nhấn nút Invoke. Nó sẽ gọi phương thức Echo và đoạn XML trả về sẽ được hiển thị trong trình duyệt (xem hình 4.4-3).



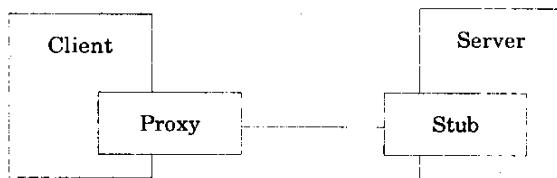
Hình 4.4-2 EchoService.asmx bên trong Internet Explorer.



Hình 4.4-3 Kết quả của Echo với "Hello World"

2. XÂY DỰNG LỚP PROXY

Để gọi một WebService từ C#, cần phải có lớp proxy. Một lớp proxy đại diện cho một cổng kết nối vào đoạn mã thực sự trên server. Trong phiên này, server này có một stub thay mặt server giao tiếp với proxy. Hình 4.4-4 là một minh họa đơn giản cho cơ chế proxy-stub.



Hình 4.4-4 Mối quan hệ proxy-stub

Để tạo một proxy, bạn cần xác định nghị thức giao tiếp. Các nghị thức giao tiếp hiện có của Web và Internet là HTTP POST, HTTP GET hoặc SOAP. Dựa trên điều này, proxy có thể được dẫn xuất từ lớp cơ sở System.Web.Services.Protocols, trong đó X đại diện cho nghị thức tương ứng.

Các ví dụ từ 4.4-3 cho đến 4.4-5 cho thấy cách cài đặt của nghị thức giao tiếp SOAP, HTTP POST và HTTP GET. Cấu trúc cơ bản cho mỗi cài đặt là nhau. Sự khác biệt đó là lớp cơ sở và những thuộc tính cùng với thông số dành cho phương thức Echo. Dùng đối tượng proxy không có gì khác lầm so với sử dụng những đối tượng thông thường. Đơn giản là khởi tạo một thể hiện đối tượng và giao phương thức thích hợp của đối tượng kèm theo tham số nếu có.

Ví dụ 4.4-3 Lớp Proxy EchoService cơ sở: SOAP

```

1: -----
2: // <autogenerated>
3: //   This code was generated by a tool.
4: //   Runtime Version: 1.0.2914.11
5: //
6: //   Changes to this file may cause incorrect behavior
7: //   and will be lost if the code is regenerated.
8: // </autogenerated>
9: -----
10:
11: //
12: // This source code was auto-generated by wsdl,
Version=1.0.2914.11.
13: //
14: using System.Xml.Serialization;
15: using System;
16: using System.Diagnostics;
17: using System.Web.Services.Protocols;
  
```



```
18: using System.Web.Services;
19:
20:
21: [System.Web.Services.WebServiceBindingAttribute(
Name="EchoServerSoap",
Namespace="http://tempuri.org/")]
22: public class EchoServiceSoapProxy :
System.Web.Services.Protocols.SoapHttpClientProtocol{
23:
24: [System.Diagnostics.DebuggerStepThroughAttribute(
)]
25:     public EchoServer() {
26:         this.Url =
"http://localhost/EchoService/EchoService.asmx";
27:     }
28:
29: [System.Diagnostics.DebuggerStepThroughAttribute(
)]
30: [System.Web.Services.Protocols.
SoapDocumentMethodAttribute("http://tempuri.org/Echo",
Use=System.Web.Services.Description.SoapBindingUse.Lit
eral,
ParameterStyle=System.Web.Services.Protocols.SoapParam
eterStyle.Wrapped)]
31:     public string Echo(string Message) {
32:         object[] results = this.Invoke("Echo", new
33:             object[] {Message});
34:         return ((string)(results[0]));
35:     }
36:
37: [System.Diagnostics.DebuggerStepThroughAttribute(
)]
38:     public System.IAsyncResult BeginEcho(string
Message, System.AsyncCallback callback,
object asyncState) {
39:         return this.BeginInvoke("Echo", new object[] {
40:             Message}, callback, asyncState);
41:     }
42:
43: [System.Diagnostics.DebuggerStepThroughAttribute(
)]
44:     public string EndEcho(System.IAsyncResult
asyncResult) {
45:         object[] results =this.EndInvoke(asyncResult);
46:         return ((string)(results[0]));
47:     }
48: }
```

Ví dụ 4.4.4 Lớp Proxy EchoService cơ sở: HTTP POST

```
1: //-
2: // <autogenerated>
3: // This code was generated by a tool.
4: // Runtime Version: 1.0.2914.11
5: //
6: // Changes to this file may cause incorrect behavior
7: // and will be lost if
8: // the code is regenerated.
9: // </autogenerated>
9: //-
10:
11: //
12: // This source code was auto-generated by wsdl,
13: Version=1.0.2914.11.
14: using System.Diagnostics;
15: using System.Xml.Serialization;
16: using System;
17: using System.Web.Services.Protocols;
18: using System.Web.Services;
19:
20:
21: public class EchoServer:
22:     System.Web.Services.Protocols.
23:     HttpPostClientProtocol{
24:         [System.Diagnostics.DebuggerStepThrough(
25:         )]
26:         public EchoServer () {
27:             this.Url =
28:                 "http://localhost/EchoService/EchoService.
29:                 asmx";
30:         }
31:         [System.Diagnostics.DebuggerStepThrough(
32:         )]
33:         [System.Web.Services.Protocols.
34:         HttpMethodAttribute(typeof(System.Web.Services.
35:         Protocols.XmlReturnReader),
36:         typeof(System.Web.Services.Protocols.
37:         HtmlFormParameterWriter))]
38:         [return: System.Xml.Serialization.
39:         XmlRootAttribute("string",
40:                         Namespace="http://tempuri.
41:                         org/", IsNullable=true)]
42:         public string Echo(string message) {
```



```
32:         return ((string)(this.Invoke("Echo", (this.Url
+ "/Echo"), new object[] {
33:             message})));
34:     }
35:
36: [System.Diagnostics.DebuggerStepThroughAttribute(
)]
37:     public System.IAsyncResult BeginEcho(string
message, System.AsyncCallback callback, object
asyncState) {
38:     return this.BeginInvoke("Echo", (this.Url +
"/Echo"), new object[] {
39:             message), callback, asyncState);
40:     }
41:
42: [System.Diagnostics.DebuggerStepThroughAttribute(
)]
43:     public string EndEcho(System.IAsyncResult
asyncResult) {
44:     return ((string)(this.EndInvoke
(asyncResult)));
45:     }
46: }
```

Ví dụ 4.4-5 Lớp Proxy EchoService cơ sở: HTTP GET

```
1://-----
2:// <autogenerated>
3://   This code was generated by a tool.
4://   Runtime Version: 1.0.2914.11
5://
6://   Changes to this file may cause incorrect behavior
and will be lost if
7://   the code is regenerated.
8:// </autogenerated>
9://-----
10:
11: //
12: // This source code was auto-generated by wsdl,
Version=1.0.2914.11.
13: //
14: using System.Diagnostics;
15: using System.Xml.Serialization;
16: using System;
17: using System.Web.Services.Protocols;
18: using System.Web.Services;
19:
20:
```

```
21: public class EchoServer :  
22:     System.Web.Services.Protocols.  
23:         HttpGetClientProtocol {  
24:  
25:     [System.Diagnostics.DebuggerStepThroughAttribute()  
26:     ]  
27:     public EchoServer() {  
28:         this.Url = "http://localhost/EchoService/  
29:             EchoService.asmx";  
30:     }  
31:     [System.Diagnostics.DebuggerStepThroughAttribute()  
32:     ]  
33:     [System.Web.Services.Protocols.  
34:         HttpMethodAttribute(typeof(System.Web.  
35:             Services.Protocols.XmlReturnReader),  
36:             typeof(System.Web.Services.Protocols.  
37:                 UrlParameterWriter))]  
38:     [return: System.Xml.Serialization.  
39:         XmlRootAttribute("string", Namespace=  
40:             "http://tempuri.org/", IsNullable=true)]  
41:     public string Echo(string message) {  
42:         return ((string)(this.Invoke("Echo", (this.Url  
43:             + "/Echo"), new object[] {  
44:                 message})));  
45:     }  
46:     [System.Diagnostics.DebuggerStepThroughAttribute()  
47:     ]  
48:     public System.IAsyncResult BeginEcho(string  
49:             message, System.AsyncCallback callback,  
50:             object asyncState) {  
51:         return this.BeginInvoke("Echo", (this.Url +  
52:             "/Echo"), new object[] {  
53:                 message}, callback, asyncState);  
54:     }  
55:     [System.Diagnostics.DebuggerStepThroughAttribute()  
56:     ]  
57:     public string EndEcho(System.IAsyncResult  
58:             asyncResult) {  
59:         return ((string)(this.EndInvoke  
60:             (asyncResult)));  
61:     }  
62: }
```

Tin vui cho bạn là không phải gõ đoạn mã này bằng tay. Kèm theo bộ .NET SDK là một công cụ hay chương trình WSDL.EXE. Dạng cơ bản của dòng lệnh gọi chương trình WSDL.EXE như sau:

```
WSDL <url or wsdl document> <language> <protocol>
```

WebServices có thể được gọi đồng bộ hoặc không đồng bộ. Trường hợp thông dụng là gọi phương thức đồng bộ. Việc gọi phương thức không đồng bộ chỉ hữu ích trong các trường hợp mà phương thức gọi này đòi hỏi một khoảng thời gian dài để xử lý, nó sẽ cho phép trình khía client tiếp tục làm việc trong khi chờ đợi những phản hồi không đồng bộ (async-response) trả về. Nó cho phép các công việc khác tiếp tục được thực hiện mà không bị bó buộc vào ứng dụng client.

Các liên kết khác - SOAP, HTTP POST và HTTP GET – đưa ra cơ hội để phát triển ứng dụng phía client dùng một cơ chế tổng quát hóa Factory để tạo và trả về ràng buộc được yêu cầu. Một WebService có khả năng đưa ra nhiều đối tượng, và mỗi đối tượng đó có thể được quan sát dưới dạng một giao tiếp chung (public interface). Đây là trường hợp một giao tiếp định nghĩa chung các phương thức của một WebService, và sau đó từng proxy sẽ cài đặt giao tiếp này.

Đối với EchoService, giao tiếp cho bản thân WebService có thể được diễn giải như sau:

```
Public interface IEchoService
{
    string Echo( string Message );
}
```

3. PROXYFACTORY

Mô hình Factory sẽ làm cô lập các lời gọi từ ứng dụng khách client khỏi ràng buộc về kiểu do Factory tạo ra. Factory trả về một kiểu thông dụng – trong trường hợp này, là giao tiếp IEchoService. Mỗi proxy, như trong các ví dụ 4.4-3, 4.4-4 và 4.4-5, sẽ phải cài đặt giao tiếp IEchoService.

Lớp ProxyFactory này sẽ nhận đối số enum để xác định đối tượng bên dưới khi tạo và trả về. Ví dụ 4.4-6 cho thấy việc cài đặt lớp này.

Ví dụ 4.4-6 Proxyfactory.cs

```
1: namespace ProxyFactory
2: {
3:     using System;
4:
5:
6:     ///<summary>
7:     /// Generalize the WebService as an interface to be
8:     /// implemented by each proxy
```

```
9:  public interface IEchoService
10: {
11:     string Echo( string Message );
12: }
13:
14:
15: ///<summary>
16: ///Enumerate the Various Bindings
17: ///</summary>
18: public enum ProxyProtocol
19: {
20:     SOAP,
21:    HttpGet,
22:    HttpPost
23: }
24:
25:
26: ///<summary>
27: ///The Factory pattern can be used to construct a
proxy for the requested binding.
28: ///The Factory will then return a well known
interface to the web service
29: ///</summary>
30: public class ProxyFactory
31: {
32:     public static IEchoService ConstructProxy(
ProxyProtocol protocol )
33:     {
34:
35:         switch( protocol )
36:         {
37:             case ProxyProtocol.SOAP:
38:                 return (IEchoService)new
EchoServiceSoapProxy();
39:             break;
40:
41:             case ProxyProtocol.HttpGet:
42:                 return (IEchoService)new
EchoServiceHttpGetProxy();
43:             break;
44:
45:             case ProxyProtocol.HttpPost:
46:                 return (IEchoService)new
EchoServiceHttpPostProxy();
47:             break;
48:
49:             default:
50:                 throw new System.Exception( "Invalid
```

```

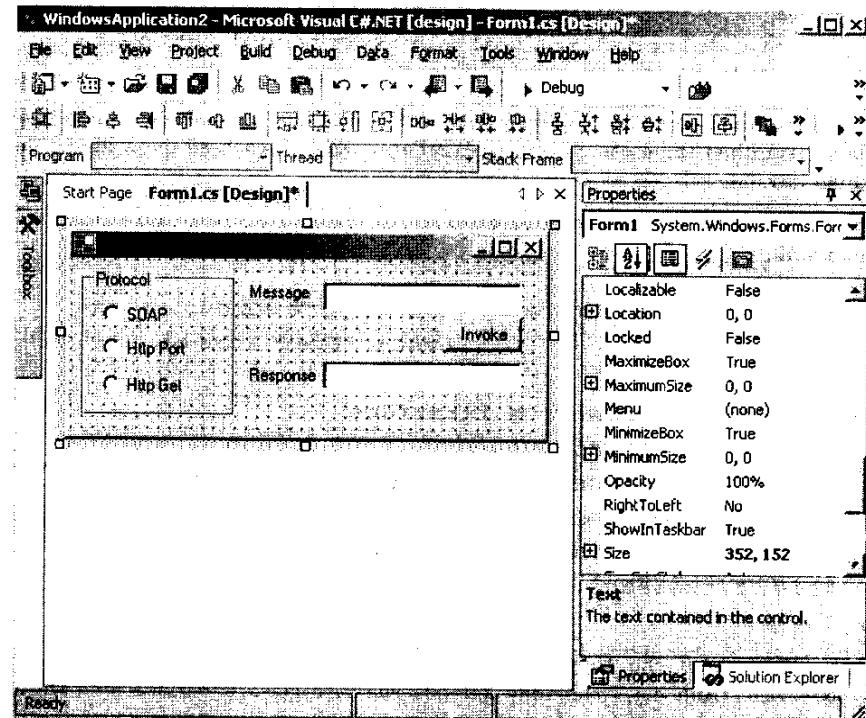
51:           break;
52:       }
53:   }
54: }
55: }
56:

```

Lớp ProxyFactory này bao gồm cả việc tạo ra đối tượng proxy dựa vào ràng buộc truyền cho phương thức ConstructProxy. Vì giao tiếp IEchoService được cài đặt nhờ vào cài đặt của từng proxy bên dưới, nên lớp ProxyFactory chỉ cần xây dựng lớp proxy thích hợp và trả về giao tiếp IEchoService là đủ.

4. CÁC CHƯƠNG TRÌNH WINDOWS FORMS SỬ DỤNG WEBSERVICE

Để kiểm tra WebService, Chương trình WinForms sau đây cung cấp một giao diện người dùng cho EchoService. Bạn sử dụng VS.NET để tạo một ứng dụng C# Windows Forms và cấu hình form này như hình vẽ 4.4-5.



Hình 4.4-5 Client WinForms

Do bộ thiết kế form đã phát sinh ra gần như toàn bộ mã chương trình nên ở đây chúng ta chỉ xem qua những đoạn mã quan trọng. Trước hết, bạn thêm vào biến được bảo vệ (protected) bằng khai báo sau:

```
Private ProxyProtocol  
SelectedProtocol=ProxyProtocol.SOAP
```

Nó cho phép ứng dụng theo dõi nghi thức giao tiếp (protocol) mà trình khách client muốn gọi đến phương thức của WebService. Hãy đặt thêm khai báo **using ProxyFactory** ở đầu chương trình. Sau đó ta kết nối mỗi nút chọn của điều khiển trong phương thức OnOpt_Click như sau:

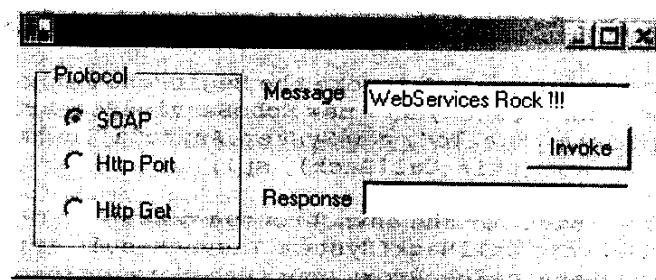
```
Protected void OnOpt_Click(object sender, EventArgs e)  
{  
    if(sender == optSOAP)  
        this.SelectedProtocol = ProxyProtocol.SOAP;  
    else if(sender == optHttpPost)  
        this.SelectedProtocol = ProxyProtocol.HttpPost;  
    else  
        this.SelectedProtocol = ProxyProtocol.HttpGet;  
}
```

Đoạn mã này cập nhập biến SelectedProtocol dựa trên nút chọn hiện hành. Bạn tham khảo thêm các chương về WinForms để biết cách sử dụng những điều khiển này.

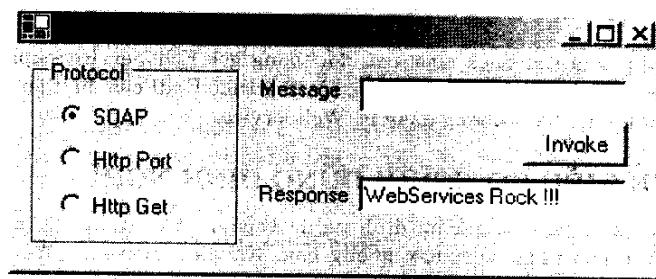
Sau cùng, WebService sẽ được gọi khi có sự kiện Click trên nút nhấn Invoke phát sinh.

```
protected void btnInvoke_Click (object sender,  
                               System.EventArgs e)  
{  
    // Get an IEchoService interface  
    IEchoService echo =  
    ProxyFactory.ConstructProxy(this.SelectedProtocol);  
    this.lblResponse.Text =  
        echo.Echo(this.txtMessage.Text);  
    this.txtMessage.Text = "";  
}
```

Hình 4.4-6 cho thấy ứng dụng client đã sẵn sàng cho việc gọi Echo WebService. Hình 4.4-7 cho thấy kết quả của việc gọi WebService hiển thị nội dung phản hồi do server trả về trong ô Response



Hình 4.4-6 Trước khi nhấn chọn nút Invoke WebService.



Hình 4.4-7 Sau khi nhấn chọn nút Invoke WebService.

Chúng ta vừa xem qua cách gọi phương thức đồng bộ, kế tiếp là gọi phương thức không đồng bộ. Cách gọi WebService không đồng bộ đòi hỏi phải có chỉ định ủy nhiệm bằng ký hiệu phương thức sau đây:

```
<access modifier> void <name>(System.IAsyncResult)
```

Quá trình cơ bản để gọi phương thức tương tự như ở mô hình đồng bộ, chúng ta chỉ tạo thêm một tuyến (thread) điều khiển quá trình gửi và nhận yêu cầu. Sau khi yêu cầu được hoàn thành, một sự kiện xuất hiện và phương thức ủy nhiệm kèm theo sẽ được gọi.

Hãy xem lại các proxy WebService được phát sinh trong các ví dụ 4.4-3, 4.4-4 và 4.4-5; mỗi proxy định nghĩa một phương thức có khai báo như sau:

```
Public System.IAsyncResult BeginEcho(
    string message,
    System.AsyncCallback callback,
    object asyncState)
```

Phương thức này yêu cầu tham số chuỗi như phương thức Echo. Ngoài ra, phương thức BeginEcho yêu cầu một phương thức ủy nhiệm phục vụ cho việc gọi lại và một đối tượng cho yêu cầu không đồng bộ phụ thuộc vào nó. Thay vì xây dựng một ứng dụng hoàn chỉnh để gọi phương thức này, đoạn mã nguồn trong ví dụ 4.4-7 sau sẽ minh họa cách gọi phương thức không đồng bộ.

**Ví dụ 4.4-7 Gọi một WebService không đồng bộ**

```
1: //Create the proxy and invoke the BeginEcho method
2: EchoServiceSoapProxy sp = new EchoServiceSoapProxy();
3: Sp.BeginEcho ("Hello", new System.AsyncCallback
               (this.CallBack), sp);
4:
5: // The callback for the asynchronous result
6: protected void CallBack(System.IAsyncResult ar) {
7:     EchoServiceSoapProxy sp = (EchoServiceSoapProxy)
                           ar.AsyncState;
8:     this.button1.Text = sp.EndEcho(ar) ;
9: }
```

Sau khi hàm CallBack được gọi, đối tượng gốc EchoServiceSoapProxy được lấy về từ giao tiếp IAsyncResult. Kế tiếp, phương thức EndEcho được gọi để lấy kết quả trả về từ lời gọi hàm không đồng bộ WebService.

5. TRẢ VỀ KIỂU DO NGƯỜI DÙNG ĐỊNH NGHĨA

Trả về kiểu do người dùng định nghĩa thường thấy trong các hàm, chẳng hạn kiểu dành cho nhân viên hay phòng ban. Khi làm điều này, WebService sẽ thực hiện cơ chế tuần tự hóa dữ liệu (serialization) tạo ra một đại diện chung của dữ liệu cần trả về cho client. Điều lưu ý là WebServices được dùng để trả về dữ liệu chứ không phải các đối tượng. Trong trường hợp đối tượng nhân viên, các thông tin cơ bản như ID, tên, phòng ban, và hình sẽ đại diện cho dữ liệu liên quan tới nhân viên này.

Quá trình tuần tự hóa sẽ sắp xếp các trường và thuộc tính chung (public) của đối tượng vào một gói dữ liệu phù hợp với cách truy xuất dữ liệu của lệnh HTTP GET. Để hỗ trợ quá trình ngược lại (không tuần tự), các thuộc tính tương ứng như set và get cũng phải cài đặt. Mỗi trường hoặc thuộc tính sẽ được ánh xạ thành một phần tử XML. Cách hành xử mặc định này có thể phục vụ đủ tốt cho các service đơn giản và cung cấp điểm khởi đầu cho việc dùng các kiểu do người dùng định nghĩa bên trong ngữ cảnh của một WebService.

6. TẠO SERVICE

Hãy bắt đầu bằng việc tạo ra một WebService tên là UDTWebService. Bạn dùng VS.NET tạo mới dự án Web và đổi tên mặc định Service1.asmx thành UDTService.asmx. WebService này sẽ cung cấp phương thức xác định một người nhờ vào họ tên của người đó; tên của phương thức này là LocatePerson. Trước khi thực hiện cài đặt phương thức này, bạn nên tạo một lớp Person dùng để trả về thông tin cho một WebService client như ví dụ 4.4-8 sau:

Ví dụ 4.4-8. Lớp UDTPerson

```
1: namespace UDTWebService
```

```

2: {
3: using System;
4:
5: /// <summary>
6: /// The Person Class represents a basic UDT( User
Defined Type )
7: /// </summary>
8: public class UDTPerson
9: {
10:     ///Private Fields
11:     private string m_FirstName;
12:     private string m_LastName;
13:
14:     //Public Properties
15:     public string FirstName
16:     {
17:         get { return m_FirstName; }
18:         set { m_FirstName = value; }
19:     }
20:
21:     public string LastName
22:     {
23:         get { return m_LastName; }
24:         set { m_LastName = value; }
25:     }
26:
27:
28:     //Constructor(s)
29:     public UDTPerson( )
30:     {
31:         //Default Constructor
32:     }
33:
34:     public UDTPerson( string FName, string LName )
35:     {
36:         m_FirstName = FName;
37:         m_LastName = LName;
38:     }
39: }
40: }
41:

```

Lớp này đại diện cho một lớp C# cơ bản và không sử dụng thuộc tính. Vậy làm cách nào để tuân tự hóa (serialization) được đối tượng? Hãy nhớ lại Reflection API. Đây là cơ chế được dùng để tuân tự hóa lớp này. WebService sẽ dùng Reflection để lấy thông tin từ lớp đối tượng và tuân tự hóa thông tin đó. Chỉ có các trường và thuộc tính chung mới được thực hiện tuân tự hóa.



Sau khi hoàn tất việc cài đặt UDTPerson, bước kế tiếp là thêm phương thức LocatePerson vào lớp UDTService. Để cập nhật lớp này, bạn thêm vào đoạn mã như trong ví dụ 4.4-9 sau:

Ví dụ 4.4-9 UDTService

```
1:  using System;
2:  using System.Collections;
3:  using System.ComponentModel;
4:  using System.Data;
5:  using System.Diagnostics;
6:  using System.Web;
7:  using System.Web.Services;
8:
9:  namespace UDTWebService
10: {
11:     /// <summary>
12:     /// Summary description for UDTService.
13:     /// </summary>
14:     public class UDTService :
15:         System.Web.Services.WebService
16:     {
17:         //Create a Hashtable to hold some People
18:         private Hashtable htPeople;
19:
20:         public UDTService()
21:         {
22:             //CODEGEN: This call is required by the
23:             //ASP.NET Web Services Designer
24:             InitializeComponent();
25:
26:             //Load the hashtable
27:             //Add some people to the hashtable
28:             htPeople = new Hashtable();
29:             htPeople.Add( "Powell", new UDTPerson(
30:                 "Bob", "Powell" ) );
31:             htPeople.Add( "Weeks", new UDTPerson(
32:                 "Richard", "Weeks" ) );
33:             htPeople.Add( "Martschenko", new UDTPerson(
34:                 "Bill", "Martschenko" ) );
35:             htPeople.Add( "Schumacher", new UDTPerson(
36:                 "Greg", "Schumacher" ) );
37:             htPeople.Add( "Pitzer", new UDTPerson(
38:                 "Jay", "Pitzer" ) );
39:
40:         }
41:
42:         #region Component Designer generated code
```

```

35:     /// <summary>
36:     /// Required method for Designer support - do
37:     /// the contents of this method with the code
38:     /// editor.
39:     private void InitializeComponent()
40:     {
41:     }
42:     #endregion
43:
44:     /// <summary>
45:     /// Clean up any resources being used.
46:     /// </summary>
47:     public override void Dispose()
48:     {
49:     }
50:
51:     [WebMethod]
52:     public UDTPerson LocatePerson( string LastName )
{
53:
54:     try {
55:         return (UDTPerson)htPeople[ LastName ];
56:     } catch( Exception ) {
57:         //something went wrong
58:     }
59:     return null;
60: }
61: }
```

Thay vì lấy về thông tin từ một cơ sở dữ liệu, ví dụ này tạo ra một bảng băm để lưu giữ thông tin của một số ít người được dùng cho mục đích tìm kiếm. Phương thức LocatePerson ở dòng 52 lấy tên LastName như là một đối số chuỗi. Cần lưu ý rằng phương thức này trông không có gì khác biệt so với WebMethod ở trong EchoService.

7. TẠO RA CÁC RÀNG BUỘC CHO CLIENT

Một lần nữa, công cụ WSDL.exe được dùng để tạo ra lớp proxy client cần thiết cho việc gọi phương thức LocatePerson. Cùng với lớp proxy, WSDL.exe sẽ phát sinh ra một lớp đơn giản đại diện cho kiểu trả về UDTPerson. Lý do là hầu hết người dùng WebService của bạn sẽ không truy xuất vào đoạn mã phía sau service này. Thay vào đó, họ sẽ nhờ vào mô tả WSDL của site để tạo ra đoạn mã client gọi đến bất kỳ service nào cho trước. Ví dụ 4.4-10 cho thấy đoạn mã tổng quát được tạo ra bởi công cụ WSDL.exe cho UDTService.

Ví dụ 4.4-10 UDTService Proxy

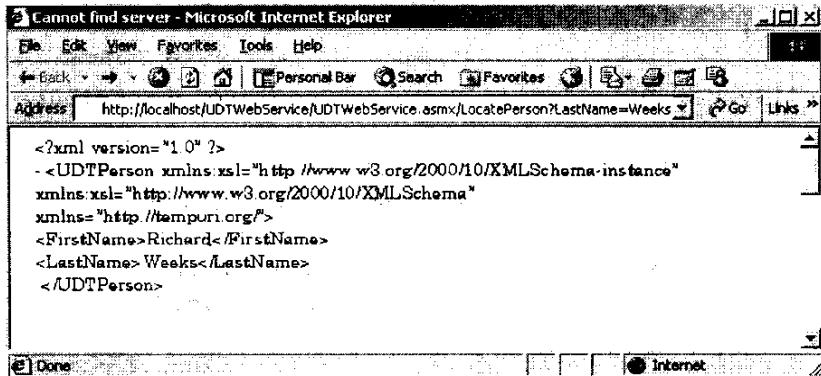
```
1: //-----
2: // <autogenerated>
3: //   This code was generated by a tool.
4: //   Runtime Version: 1.0.2914.11
5: //
6: //   Changes to this file may cause incorrect behavior
7: //   and will be lost if
8: //   the code is regenerated.
9: // </autogenerated>
9: //-----
10:
11: //
12: // This source code was auto-generated by wsdl,
13: // Version=1.0.2914.11.
14: //
15: using System.Diagnostics;
16: using System.Xml.Serialization;
17: using System;
18: using System.Web.Services.Protocols;
19:
20:
21: [System.Web.Services.WebServiceBindingAttribute
22: (Name="UDTServiceSoap",Namespace="http://tempuri.org/")]
22: public class UDTService :
23: System.Web.Services.Protocols.
24:     SoapHttpClientProtocol {
25:     public UDTService() {
26:         this.Url = "http://localhost/UDTWebService/
27:                         UDTService.asmx";
28:     }
29:     [System.Diagnostics.DebuggerStepThroughAttribute(
30:     [System.Web.Services.Protocols.
```



```
    SoapMethodAttribute("http://tempuri.org/
LocatePerson", Use=System.Web.Services.
Description.SoapBindingUse.Literal,
ParameterStyle= System.Web.Services.
Protocols.SoapParameterStyle.Wrapped))
31: public UDTPerson LocatePerson(string LastName) {
32:     object[] results = this.Invoke("LocatePerson",
new object[] {LastName});
33:     return ((UDTPerson)(results[0]));
34: }
35:
36: [System.Diagnostics.DebuggerStepThrough(
)]
37: public System.IAsyncResult BeginLocatePerson(
string LastName, System.AsyncCallback
callback, object asyncState) {
38:     return this.BeginInvoke("LocatePerson", new
39:     object[] {LastName}, callback, asyncState);
40: }
41:
42: [System.Diagnostics.DebuggerStepThrough(
)]
43: public UDTPerson EndLocatePerson(System.
IAsyncResult asyncResult) {
44:     object[] results = this.EndInvoke(
asyncResult);
45:     return ((UDTPerson)(results[0]));
46: }
47: }
48:
49:
50: public class UDTPerson {
51:
52:     public string FirstName;
53:
54:     public string LastName;
55: }
```

Về cơ bản, đoạn mã tạo bởi WSDL.EXE tương tự như những ví dụ mà bạn đã thấy trước đây; cái mới cần lưu ý là lớp UDTPerson tạo ra các thuộc tính FirstName và LastName như là những trường chung (public) thay vì riêng (private). WSDL.EXE không có cách nào biết việc cài đặt thực sự của UDTPerson vì nó nằm trên server; thay vào đó, WSDL.EXE phải dùng thông tin tìm thấy trong tài liệu WSDL để xây dựng các kiểu proxy và tham chiếu.

Quá trình gọi UDTService.LocatePerson từ Internet Browser sẽ tạo ra một gói dữ liệu XML và thể hiện nó trên trình duyệt, như trong hình 4.4-8. XML này đại diện cho thông điệp trả về từ WebService.



The screenshot shows a Microsoft Internet Explorer window with the title "Cannot find server - Microsoft Internet Explorer". The address bar contains "http://localhost/UDTWebService/UDTWebService.asmx/LocatePerson?LastName=Weeks". The main content area displays the following XML code:

```

<?xml version="1.0" ?>
<UDTPerson xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns="http://tempuri.org/">
  <FirstName>Richard</FirstName>
  <LastName>Weeks</LastName>
</UDTPerson>

```

Hình 4.4-8 Kết quả XML từ *LocatePerson*

8. CÁC THUỘC TÍNH XML

Cho tới giờ, chúng ta đã thấy các kiểu trả về bên trong và UDTs từ một WebService. Quá trình tuân tự hóa cơ bản được cung cấp bởi WebServices nó chung là đủ, nhưng đôi khi chúng ta cũng cần điều khiển và xử lý các phần tử XML được trả về này. Một tài liệu XML chỉ có thể có một nút gốc (root). Một XML root có thể được định nghĩa bằng cách dùng XMLRootAttribute. XMLRootAttribute được dùng để đặc tả các thuộc tính sau trong bảng 4.4-1.

Bảng 4.4-1 Các thuộc tính minh họa XMLRootAttribute

Thuộc tính	Nghĩa
DataType	Kiểu dữ liệu XML của phần tử gốc
ElementName	Tên của phần tử gốc
Form	Tên của phần tử gốc có đạt yêu cầu hay không
IsNullable	Xác định coi XmlSerializer nên được sắp xếp nếu thiết lập nó về null
Namespace	Namespace cho phần tử gốc

Một phần tử gốc XML root có thể chứa N số phần tử con, và mỗi con lại có khả năng chứa N phần tử con khác. Sự định nghĩa đệ quy này cho phép lồng các phần tử con vào nhau nhiều tùy ý theo nhiều cấp độ.

Cùng với phần tử gốc, một tài liệu XML còn chứa các phần tử và thuộc tính con. Một phần tử có thể được xem là 1 kiểu chứa đựng toàn thể văn bản. XMLElementAttribute được dùng để mô tả những phần tử này. Chúng có thể là một lớp hoặc thành viên của một lớp. Để bổ sung vào việc đặc tả XMLElementAttributes, các phần tử có thể có các thuộc tính áp dụng cho chúng bằng cách dùng lớp XMLAttributeAttribute.

Cho tới giờ, chỉ có các đối tượng đơn do WebServices tạo được trả về. XmlSerializer hoàn toàn có thể trả về các cấu trúc XML phức tạp với các lớp lồng vào nhau. Chúng ta cũng có thể dùng các thuộc tính XML khác nhau chung với các cấu trúc (struct) và trộn trong các thực thể không có thuộc tính XML.

Để biết khả năng của XmlSerializer, chúng ta sẽ tạo một WebService đơn giản dùng trả về một Employee chứa 2 đối tượng lồng nhau khác - một đối tượng Address và một cấu trúc Department. Ví dụ 4.4-11 cho thấy cách cài đặt của thực thể Employee, Address và Department.

Ví dụ 4.4-11 Entities.cs

```

1: namespace XML
2: {
3:     using System;
4:     using System.Xml;
5:     using System.Xml.Serialization;
6:
7:
8:     [XmlRoot("employee")]
9:     public class Employee
10:    {
11:        private string      m_FirstName;
12:        private string      m_LastName;
13:        private Address    m_Address;
14:        private Departmentm_Dept;
15:
16:        //Properties
17:        [XmlAttribute("first_name")]
18:        public string FirstName
19:        {
20:            get{ return m_FirstName; }
21:            set { m_FirstName = value; }
22:        }
23:
24:        [XmlAttribute("last_name")]
25:        public string LastName
26:        {

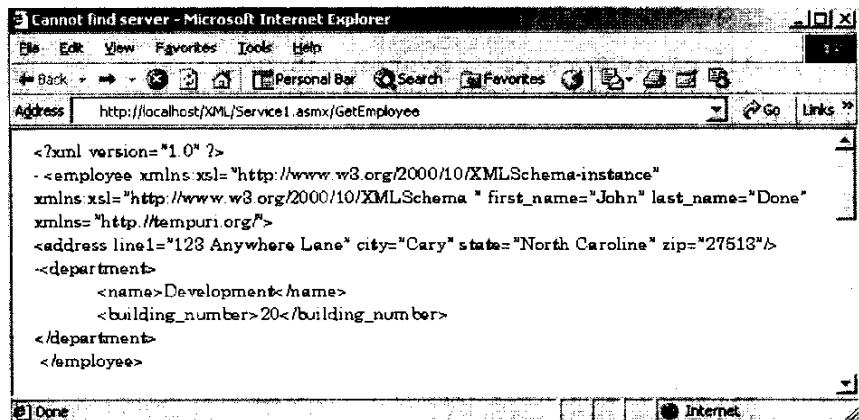
```

```
27:         get { return m_LastName; }
28:         set { m_LastName = value; }
29:     }
30:
31:
32:         [XmlElement("address")]
33:         public Address Address
34:         {
35:             get { return m_Address; }
36:             set { m_Address = value; }
37:         }
38:
39:         [XmlElement("department")]
40:         public Department Department
41:         {
42:             get { return m_Dept; }
43:             set { m_Dept = value; }
44:         }
45:     }
46:
47:
48:     public class Address
49:     {
50:
51:         //Use Public Fields with XmlAttributes
52:         [XmlAttribute("line1")]
53:         public string Line1;
54:
55:         [XmlAttribute("city")]
56:         public string City;
57:
58:         [XmlAttribute("state")]
59:         public string State;
60:
61:         [XmlAttribute("zip")]
62:         public string Zip;
63:     }
64:
65:     //Use a simple struct
66:     public struct Department
67:     {
68:         //Fields with no XML attributes
69:         public string name;
70:         public int building_number;
71:     }
72: }
73:
```



Các thực thể trong ví dụ 4.4-11 kết hợp các thuộc tính XML khác nhau và tạo ra lớp kết hợp. Cả Employee và Address đều dùng các thuộc tính XML, trong khi cấu trúc Department chỉ dùng các trường public. Mặc định, tất cả các trường public của một lớp hoặc struct sẽ được tuân tự hóa bởi đoạn mã WebService XML. Khả năng kết hợp này cung cấp một năng lực ẩn tượng cho phép bạn, người phát triển, có quyền điều khiển hoàn toàn trên quá trình tuân tự hóa.

Kết quả của lớp Employee kết hợp sau khi đã tuân tự hóa sẽ như hình 4.4-9.



Hình 4.4-9 Hiển thị dạng XML của lớp Employee

Lưu ý đến các thuộc tính XML khác nhau và cách chúng ảnh hưởng lên kết quả XML được tạo ra trong suốt quá trình tuân tự hóa thực thể Employee. Bạn lưu ý đến đoạn mã proxy tạo bởi WSDL.EXE trong ví dụ 4.4-12 sau.

Ví dụ 4.4-12 Đoạn mã phát sinh bởi Client Proxy

```

1: // <autogenerated>
2: // This code was generated by a tool.
3: // Runtime Version: 1.0.2615.1
4: //
5: // Changes to this file may cause incorrect behavior
6: // and will be lost if
7: // the code is regenerated.
8: //
9: //
10: // This source code was auto-generated by wsdl,
11: // Version=1.0.2615.1.
12: using System.Xml.Serialization;

```

```
13: using System;
14: using System.Web.Services.Protocols;
15: using System.Web.Services;
16:
17:
18: [
19:     System.Web.Services.WebServiceBindingAttribute(
20:         Name="Service1Soap",
21:         Namespace="http://tempuri.org/")
22: ]
23: public class Service1 : System.Web.Services.
24:     Protocols.SoapHttpClientProtocol {
25:     public Service1() {
26:         this.Url = "http://localhost/XML/Service1.
27:             asmx";
28:     }
29:     [
30:         System.Web.Services.Protocols.
31:             SoapMethodAttribute(
32:             "http://tempuri.org/
33:                 GetEmployee", MessageStyle=System.Web.
34:                     Services.Protocols.SoapMessageStyle.
35:                         ParametersInDocument)
36:     ]
37:     [return: System.Xml.Serialization.
38:         XmlElementAttribute("employee",
39:             IsNullable=false)]
40:     public Employee GetEmployee() {
41:         object[] results = this.Invoke("GetEmployee",
42:             new object[0]);
43:         return ((Employee)(results[0]));
44:     }
45:     public System.IAsyncResult BeginGetEmployee(
46:         System.AsyncCallback callback,
```



```
47:  
48: public class Employee {  
49:  
50:     [System.Xml.Serialization.XmlElementAttribute  
51:      (IsNullable=false)]  
52:     public Address address;  
53:     [System.Xml.Serialization.XmlElementAttribute  
54:      (IsNullable=false)]  
55:     public Department department;  
56:     [System.Xml.Serialization.XmlAttributeAttribute  
57:      ()]  
58:     public string first_name;  
59:     [System.Xml.Serialization.XmlAttributeAttribute  
60:      ()]  
61:     public string last_name;  
62:  
63:     public class Address {  
64:  
65:         [System.Xml.Serialization.XmlAttributeAttribute  
66:          ()]  
67:         public string line1;  
68:         [System.Xml.Serialization.XmlAttributeAttribute  
69:          ()]  
70:         public string city;  
71:         [System.Xml.Serialization.XmlAttributeAttribute  
72:          ()]  
73:         public string state;  
74:         [System.Xml.Serialization.XmlAttributeAttribute  
75:          ()]  
76:         public string zip;  
77:  
78:         [  
79:             System.Xml.Serialization.XmlTypeAttribute(  
80:                 Namespace="http://tempuri.org/")  
81:         ]  
82:         public class Department {  
83:  
84:             public string name;  
85:  
86:             public int building_number;
```



87 : }

88 :

Như thường lệ, WSDL.EXE tạo ra đoạn mã proxy cần thiết để gọi WebService theo yêu cầu. Cùng với mã proxy, WSDL.EXE cũng tạo ra các thư Employee, Address và Department như được mô tả bởi tài liệu wsdl WebServ. Lưu ý là sự khác biệt giữa Employee, Address và Department được phát triển đây và công cụ WSDL.EXE.

Dù mỗi class/struct dùng một hướng khác nhau để tạo ra dạng tuân tự thích hợp, cả 2 cách thực hiện này trực giao lẫn nhau. Về cơ bản, chỉ có thể thay đổi trong các trường và thuộc tính được bố trí lại. Để hiệu quả, WSDL dựa vào các thuộc tính XML khác nhau và công dụng của Reflection để tạo ra tuân tự thích hợp.

9. KẾT CHƯƠNG

Chương này chúng ta đã thử nghiệm và tìm hiểu về WebServices đại diện cho một mô hình phát triển mới trong việc tách dữ liệu cách thể hiện logic chương trình. WebServices là một đề tài khó hiểu nếu chưa từng tiếp cận với mô hình lập trình phân tán trước đây. Tuy nhiên, chúng ta tạm dừng đề tài WebServices ở đây để chuẩn bị tìm hiểu về kiến trúc bên trong các gói ứng dụng .NET trong chương sau.

Phân V

GÓI KẾT HỢP

Trong phần này:

- ◆ Các gói kết hợp(ASSEMBLY)
- ◆ Chữ ký số và phiên bản
- ◆ Tương tác với thế giới COM
- ◆ Các tiểu trình (THREADS)

Chương 5.1

CÁC GÓI KẾT HỢP(ASSEMBLY)

Các vấn đề chính sẽ được đề cập đến:

- ✓ Gói kết hợp (assembly) là gì?
- ✓ Gói kết hợp (assembly) trong một tập tin đơn
- ✓ Gói kết hợp (assembly) trong nhiều tập tin
- ✓ Các thuộc tính của gói kết hợp
- ✓ Tải nạp gói kết hợp (assembly) khi chương trình đang chạy

1. GÓI KẾT HỢP(ASSEMBLY) LÀ GÌ?

Khái niệm “gói kết hợp” ám chỉ một hay nhiều tập tin được nhóm lại một cách luận lý (logic) có thể sử dụng bên trong kiến trúc .NET. Mỗi gói kết hợp (assembly) có thể là một tập tin DLL/EXE đơn lẻ hay một nhóm DLL/EXE liên quan.

Trước thời của .NET, các đoạn mã được gói bên trong các DLL, EXE và nếu như có người trong các bạn còn nhớ: đó là những thư viện overlay. Bằng cách chia cắt các đoạn mã chương trình đã được biên dịch thành các đoạn nhỏ, ta có thể sử dụng lại những mã đã biên dịch rồi cho các ứng dụng khác. Dĩ nhiên, cách thức này làm cho các DLL quý quái gây ra biết bao nhiêu tranh cãi và giấy mực. Địa ngục DLL chỉ đơn thuần là những vấn đề sẽ nảy sinh khi thay một phiên bản của DLL bằng một phiên bản khác mới hơn, khi đó, kết quả là chương trình đột nhiên ngưng hoạt động. Lý do của việc này không phải lúc nào cũng dễ thấy, nhưng thực sự đó là bởi việc kiểm tra yếu kém về quản lý phiên bản của DLL.

Bằng việc sử dụng các gói kết hợp .NET, chúng ta có thể lưu giữ nhiều phiên bản của chúng trên mỗi gói riêng rẽ. Một ứng dụng có thể lựa chọn việc sử dụng phiên bản gói kết hợp (assembly) mà nó được biên dịch chung hay dùng gói kết hợp có phiên bản mới nhất. Sự kết nối động này được điều khiển bởi tập tin cấu hình của ứng dụng. Tập tin này nằm ở cùng thư mục với ứng dụng. Chúng ta sẽ xem xét về tập tin cấu hình này trong chương 5.2 “Chữ ký số và phiên bản hoá”.

1.1. Nội dung gói kết hợp

Mỗi gói kết hợp (assembly) chứa một bảng Manifest cùng với các đoạn mã đã được biên dịch. Một Manifest là một bảng các tập tin phụ trợ cho gói kết hợp gồm một hay nhiều tập tin, Manifest cũng dùng lưu giữ các thông tin về phiên bản, tên, và các thông tin khác về một gói kết hợp cụ thể. Sử dụng ILDASM, bạn có thể thấy rõ nội dung của bảng Manifest có trong gói kết hợp .NET. Hình 5.1-1 là nội dung Manifest của gói mscorlib.dll.



Các gói kết hợp (assembly) .NET đã tự giới thiệu đầy đủ về mình, đó là, tất cả các thông tin về nội dung của gói kết hợp (assembly) đều có thể được truy xuất đến trong khi chương trình đang thực thi hoặc từ các ứng dụng .NET khác. Giống như cách mà các chương trình COM client dò tìm những đối tượng phục vụ COM server về các giao tiếp được hỗ trợ, nó có thể lấy tất cả các thông tin của một gói kết hợp. Phần cuối của chương này sẽ tạo ra một trình đọc nội dung của gói kết hợp bao gồm những thông tin đơn giản.

The screenshot shows a code editor with a tab labeled "MANIFEST". The content of the manifest file is as follows:

```
module extern kernel32.dll
.module extern oleaut32.dll
.module extern mscoree.dll
.module extern ole32.dll
.module extern advapi32.dll
.module extern shfolder.dll
.module extern user32.dll
assembly mscorelib
{
    .custom instance void System.Resources.NeutralResourcesLanguageAttribute::ctor()
    .custom instance void System.Reflection.AssemblyDelayLoadAttribute::ctor()
    .custom instance void System.Reflection.AssemblyDescriptionAttribute::ctor()
    .custom instance void System.Resources.SatelliteContractVersionAttribute::ctor()
    .custom instance void System.Reflection.AssemblyKeyFileAttribute::ctor(int)
    .custom instance void System.Runtime.InteropServices.GuidAttribute::ctor()
    .custom instance void System.CLSCompliantAttribute::ctor(bool) = {01 00 00 00}
// --- The following custom attribute is added automatically, do not uncomplishmen
```

Hình 5.1-1 Manifest cho gói mscorelib.dll

1.2. Định vị gói kết hợp

Mục đích của các gói kết hợp (assembly) .NET là cho phép dễ dàng cài đặt các thành phần đối tượng (component). Có thể cài đặt một ứng dụng chỉ đơn giản bằng phong cách XCOPY ứng dụng đến một thư mục khác trên máy khách. Không cần phải đăng ký các thành phần, không cần các mẩu registry cho COM. Thế thì làm sao định vị được một gói kết hợp khi chương trình đang chạy? Hệ thống .NET sẽ định vị các gói kết hợp (assembly) cần thiết theo những tuỳ chọn sau:

- Đường dẫn của ứng dụng
- Thư mục bin dưới đường dẫn của ứng dụng
- Thư mục assembly dưới đường dẫn của ứng dụng (ví dụ: C:/program files\myapp\myassembly\myassembly.dll)
- Cache nơi lưu trữ tạm thời các gói kết hợp chung của hệ thống (xem chương 5.2)

Bạn có thể tìm thấy chi tiết cụ thể hơn về cách định vị gói kết hợp (assembly) trong tài liệu MSDN, kể cả các chủ đề về thuật toán dò tìm cùng những thứ linh tinh khác.



Với phần lớn các ứng dụng, ta dùng dùng các gói kết hợp riêng (private assembly). Một gói kết hợp (assembly) riêng là một gói kết hợp chỉ được dùng riêng cho một ứng dụng nào đó thôi. Bởi vì không ứng dụng nào khác cần biết về sự tồn tại của chúng, chúng có thể được đặt trong thư mục hệ thống hoặc thư mục con của ứng dụng. Chỉ khi công ty của bạn phát triển một bộ sản phẩm phải chia sẻ các gói kết hợp thì bạn mới nên nghĩ đến việc dùng cache bộ đệm chung nơi chứa các gói kết hợp thường dùng trong chương trình.

2. GÓI KẾT HỢP LUU TRONG MỘT TẬP TIN ĐƠN

Mặc dù có thể bạn không cần đến nó, các ví dụ ở đây đều xét đến gói kết hợp đơn. Hãy nhớ rằng gói kết hợp là các tập tin EXE hay DLL. Việc này có thể làm cho lý thuyết về gói kết hợp (assembly) có vẻ khó nuốt một chút. Một gói kết hợp đơn là kết quả của việc biên dịch mã chương trình, và các tài nguyên khác cho kết quả là một DLL hay một EXE/

3. GÓI KẾT HỢP LUU TRONG NHIỀU TẬP TIN

Quy trình tạo một gói kết hợp lưu trong nhiều tập tin được thực hiện thông qua các tiện ích dòng lệnh. Điểm chủ chốt cần phải giải thích là các gói kết hợp lưu trong nhiều tập tin chỉ có thể chứa một điểm vào. Đó là cách nói về việc chỉ một lớp (class) trong gói kết hợp được phép chứa phương thức Main. Dĩ nhiên, nếu gói kết hợp đa tập tin là một DLL thì việc này là không cần cù. Tuy nhiên, nếu gói kết hợp đa tập tin là một file EXE thì nó không được phép có nhiều hơn một điểm vào.

Để tạo ra gói kết hợp lưu trong nhiều tập tin, bạn cần tạo ra hai tập tin nguồn và dịch chúng thành các đoạn mã .netmodule. Ví dụ: 5.1-1 và 5.1-2 chứa mã chương trình cho một lớp rất đơn giản kết hợp trong nhiều tập tin.

Ví dụ 5.1-1 Lớp ClassOne - gói kết hợp lưu trong nhiều tập tin

```

1: using System;
2: using FileTwo;
3:
4: namespace FileOne {
5:
6:     public class ClassOne {
7:
8:         public static void Main( ) {
9:
10:             ClassTwo ct = new ClassTwo( );
11:             ct.SayHello( );
12:
13:
```

```
12:  
13:      }  
14:  }  
15: }
```

Ví dụ 5.1-2 ClassTwo của gói kết hợp lưu trong nhiều tập tin

```
1: using System;  
2:  
3:  
4: namespace FileTwo {  
5:  
6:  
7:     public class ClassTwo {  
8:  
9:         public void SayHello() {  
10:  
11:             Console.WriteLine("Hello From  
FileTwo.ClassTwo");  
12:         }  
13:     }  
14: }
```

Để tạo ra một gói kết hợp mới, trước hết cần dịch mỗi tập tin thành module. Lưu ý đến thứ tự các bước của quy trình:

1. csc /t:module two.cs
2. csc /addmodule:module two.netmodule /t:module one.cs
3. al one.netmodule two.netmodule /main:FileOne.ClassOne.Main /out:MultiAsm.
/target:exe

Bước đầu tiên của quá trình là tạo lập module từ tập tin mã nguồn thứ 1 Sở dĩ ta làm như vậy là vì tập tin thứ nhất cần dùng đến FileTwo.ClassTwo. T đó, tạo lập module từ tập tin mã nguồn thứ nhất có dùng đến các module mã được tạo lập ở bước trước. Bước cuối sử dụng đến Công cụ đóng gói kết hợp al để tạo ra gói kết hợp từ nhiều tập tin. Lưu ý rằng nếu bạn xoá các tập .netmodule và cố chạy MultiAsm.exe thì sẽ có một lỗi thực thi do các ràng buộc thiếu. Hình 5.1-2 cho thấy nội dung manifest của MultiAsm.exe; lưu ý rằng chỉ tham chiếu đến one.module và two.module.

Bằng cách đóng gói nhiều tập tin với nhau trong một gói kết hợp, chúng có thể chắc chắn được về sự chính xác của các tham chiếu và phiên bản của chương trình sẽ được thực thi.



```

# MANIFEST
.module extern one.netmodule
.assembly extern mscorlib
{
    .publickeytoken = {B7 7A 5C 56 19 34 E0 89} // .zv
    .hash = (C9 B3 D7 6A BE 8C 7E 48 1A A5 47 AC 39 C9 52 13 // ...j..~H..G.1
              2A 80 96 04) // *...
    .ver 1:0:2411:0
}
.assembly MultiAsm
{
    // --- The following custom attribute is added automatically, do not uncomment
    // .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute:
    //
    .hash algorithm 0x00008004
    .ver 0:0:0:0
}
.file one.netmodule
    .hash = (BA C9 1E 60 36 5C E7 7E 7C 18 90 D2 78 21 55 70 // ...`6\.^...
                  99 35 D8 AA) // .5..
.file two.netmodule
    .hash = (31 0D ED 8E 5E D2 B6 91 C1 A8 5F 84 65 A7 41 F5 // 1...^.....
                  9E E3 06 92)
.class extern public FileOne.ClassOne

```

Hình 5.1-2 Manifest của MultiAsm.exe

4. CÁC THUỘC TÍNH CỦA GÓI KẾT HỢP

Tất cả các gói kết hợp (assembly) đều chứa nhiều thuộc tính khác nhau, bao gồm tên của gói kết hợp, công ty, phiên bản, văn hoá, bản quyền, và các diễn giải khác. Ví dụ 5.1-3 sẽ cho thấy tập tin mặc định AssemblyInfo.cs được thêm vào từ bất cứ dự án C#.NET nào.

Ví dụ 5.1-3 AssemblyInfo.cs

```

1: using System.Reflection;
2: using System.Runtime.CompilerServices;
3:
4: //
5: // General Information about an assembly is controlled
6: // through the following
7: // set of attributes. Change these attribute values to
8: // modify the information
9: // associated with an assembly.
8: //
9: [assembly: AssemblyTitle("")]
10: [assembly: AssemblyDescription("")]
11: [assembly: AssemblyConfiguration("")]
12: [assembly: AssemblyCompany("")]
13: [assembly: AssemblyProduct("")]
14: [assembly: AssemblyCopyright("")]
15: [assembly: AssemblyTrademark("")]
16: [assembly: AssemblyCulture("")]

```

```
17:  
18://  
19:// Version information for an assembly consists of the  
following four values:  
20://  
21://    Major Version  
22://    Minor Version  
23://    Build Number  
24://    Revision  
25://  
26:// You can specify all the values or you can default the  
Revision and Build Numbers  
27:// by using the '*' as shown below:  
28:  
29:[assembly: AssemblyVersion("1.0.*")]  
30:  
31://  
32:// In order to sign your assembly you must specify a key  
to use. Refer to the  
33:// Microsoft .NET Framework documentation for more  
information on assembly signing.  
34://  
35:// Use the attributes below to control which key is used  
for signing.  
36://  
37:// Notes:  
38:// (*) If no key is specified, the assembly is not  
signed.  
39:// (*) KeyName refers to a key that has been installed  
in the Crypto Service  
40://     Provider (CSP) on your machine. KeyFile refers to  
a file which contains  
41:// (*) If the KeyFile and the KeyName values are both  
specified, the  
42://     following processing occurs:  
43://     (1) If the KeyName can be found in the CSP, that key  
is used.  
44://     (2) If the KeyName does not exist and the KeyFile  
does exist, the key  
45://     in the KeyFile is installed into the CSP and  
used.  
46:// (*) Delay Signing is an advanced option - see the  
Microsoft .NET Framework  
47://     documentation for more information on this.  
48://  
49:[assembly: AssemblyDelaySign(false)]  
50:[assembly: AssemblyKeyFile("")]  
51:[assembly: AssemblyKeyName("")]
```

Các thuộc tính đăng ký tên một gói kết hợp (AssemblyDelaySign, AssemblyKeyFile, và AssemblyKeyName) đã được trình bày ở chương 5.2. Các thuộc tính còn lại sẽ rất quen thuộc với các lập trình viên Win32. Mỗi gói kết hợp có thể có các thông tin về công ty cung cấp, kể cả các thông tin về bản quyền và phiên bản hiện tại.

5. TẢI NẠP GÓI KẾT HỢP KHI CHƯƠNG TRÌNH ĐANG THỰC THI

Hệ thống .NET sẽ tải tất cả các gói kết hợp khi cần thiết và tham chiếu đến những gói kết hợp khác dựa vào nội dung trong các Manifests. Ví dụ một ứng dụng Windows Forms thường tham chiếu đến gói System.Windows.Forms.dll. Tuy nhiên, đôi khi chúng ta cũng cần đến việc tải động (dynamic) các gói kết hợp (assembly) .NET khi đang thực thi. Cơ chế tải động sẽ cho chúng ta khả năng tải các thành phần .NET để dùng vào bất kỳ lúc nào chương trình có nhu cầu và với bất kỳ gói nào mà chương trình mong muốn sử dụng.

Tải các gói kết hợp trong khi chương trình đang chạy là một cơ chế mạnh để mở rộng một ứng dụng. Hãy nghĩ đến một ứng dụng cho phép các form được thêm vào ở một thời điểm bất kỳ. Ví dụ: Microsoft Management Console (MMC) cho phép phát triển các snap-in, chúng là đơn giản chỉ là các thành phần COM được gọi trong shell do MMC cung cấp.

5.1. Dự án FormHost

Nhằm minh họa những bước cơ bản để tải các gói kết hợp trong khi chương trình đang chạy với mục đích mở rộng một ứng dụng, chúng ta sẽ tạo ra một dự án minh họa nhỏ. Các ứng dụng minh họa sẽ tìm các gói kết hợp trong một thư mục con xác định. Mỗi gói kết hợp sẽ được tải lên, và kiểu của các gói kết hợp sẽ được theo dõi. Nếu một kiểu trong gói kết hợp nào đó dẫn xuất từ lớp System.Windows.Forms.Form và kiểu ấy hỗ trợ một giao tiếp mở rộng, trong ví dụ của chúng ta là IFormHostClient, thì form sẽ được tải lên và một mục chọn menu sẽ được thêm vào form chủ.

Để xây dựng ví dụ minh họa này, bạn hãy làm các bước sau:

1. Tạo một giải pháp (Solution) trống có tên DynamicAsmLoad.
2. Sau khi đã có một giải pháp trống, bắt đầu bằng cách thêm vào giải pháp một dự án thư viện lớp C#. Đặt tên thư viện ấy là FormHostSDK (Dự án FormHostSDK sẽ không làm gì ngoài việc cho phép một giao tiếp đơn dùng trong ví dụ minh họa).
3. Thêm một lớp mới vào FormHostSDK và đặt tên là IFormHostClient.

Ví dụ 5.1-4 dưới đây sẽ chỉ ra các định nghĩa giao tiếp được dùng cả trong dự án FormHost lẫn dự án FormOne.

Ví dụ 5.1-4 Giao tiếp IFormHostClient

```

1: ///Simple Interface for forms
2: ///
3: using System;
4:
5: namespace FormHostSDK
6: {
7:     public interface IFormHostClient {
8:
9:         string MenuText {
10:             get;
11:         }
12:     }
13: }
```

Giao thức IFormHostClient chỉ cần cài đặt một thuộc tính. Thuộc MenuText được dùng để thêm một mục menu cho FormHost (được tạo trong sau).

Bạn thêm một Windows Application C# vào dự án với tên FormHost. Windows Application này sẽ phục vụ như một ứng dụng chứa các form MDI chúng sẽ được đặt nằm trong các gói (assembly) và được tải động bởi ứng FormHost. Ví dụ 5.1-5 sẽ cho thấy các đoạn mã của ứng dụng FormHost.

Ví dụ 5.1-5 FormHost

```

1: using System;
2: using System.Drawing;
3: using System.Collections;
4: using System.ComponentModel;
5: using System.Windows.Forms;
6: using System.Data;
7: using System.Reflection;
8: using System.IO;
9: using FormHostSDK;
10:
11:namespace FormHost
12:{
13:    /// <summary>
14:    /// Summary description for MainForm.
15:    /// </summary>
16:    public class MainForm : System.Windows.Forms.Form
17:    {
18:        /// <summary>
19:        /// Required designer variable.
20:        /// </summary>
21:        private System.ComponentModel.Container
components = null;
```



```
22:     private System.Windows.Forms.MdiClient  
23:             mdiClient1;  
24:     private System.Windows.Forms.MainMenu mainMenu1;  
25:             private System.Windows.Forms.MenuItem  
26:                     FileMenuItem;  
27:             private System.Windows.Forms.MenuItem  
28:                     FormsMenuItem;  
29:             private System.Windows.Forms.MenuItem  
30:                     ExitMenuItem;  
31:     private Hashtable    forms = new Hashtable();  
32:     //  
33:     // Required for Windows Form Designer support  
34:     //  
35:     InitializeComponent();  
36:     //  
37:     // TODO: Add any constructor code after  
38:             InitializeComponent call  
39:     //  
40:     LoadForms();  
41: }  
42:  
43: /// <summary>  
44: /// Clean up any resources being used.  
45: /// </summary>  
46: protected override void Dispose( bool disposing )  
47: {  
48:     if( disposing )  
49:     {  
50:         if (components != null)  
51:         {  
52:             components.Dispose();  
53:         }  
54:     }  
55:     base.Dispose( disposing );  
56: }  
57:  
58: #region Windows Form Designer generated code  
59: /// <summary>  
60: /// Required method for Designer support - do not  
61: /// the contents of this method with the code  
62: /// </summary>  
63: private void InitializeComponent()
```

```
64:      {
65:          this.mdiClient1 = new
66:              System.Windows.Forms.MdiClient();
67:          this.mainMenu1 = new
68:              System.Windows.Forms.MainMenu();
69:          this.FileMenuItem = new
70:              System.Windows.Forms.MenuItem();
71:          this.FormsMenuItem = new
72:              System.Windows.Forms.MenuItem();
73:          this.ExitMenuItem = new
74:              System.Windows.Forms.MenuItem();
75:          this.SuspendLayout();
76:          //
77:          // mdiClient1
78:          //
79:          // this.mdiClient1.Dock =
80:              System.Windows.Forms.DockStyle.Fill;
81:          this.mdiClient1.Name = "mdiClient1";
82:          this.mdiClient1.TabIndex = 0;
83:          //
84:          // mainMenu1
85:          //
86:          this.mainMenu1.MenuItems.AddRange(
87:              new System.Windows.Forms.MenuItem[] {
88:                  this.FileMenuItem,
89:                  this.FormsMenuItem
90:              }
91:          );
92:          //
93:          // FormsMenuItem
94:          //
95:          this.FormsMenuItem.Index = 1;
96:          this.FormsMenuItem.Text = "F&orms";
97:          this.FileMenuItem.MenuItems.Add(
98:              this.ExitMenuItem );
99:          //
100:         // menuItemExit
101:         //
102:         this.ExitMenuItem.Index = 0;
103:         this.ExitMenuItem.Text = "E&xit";
104:         //
```

```
105:         // MainForm
106:         //
107:         this.AutoScaleBaseSize = new
108: System.Drawing.Size(5, 13);
109:         this.ClientSize = new
110: System.Drawing.Size(576, 429);
111:         this.Controls.AddRange(
112:             new System.Windows.Forms.Control[] {
113:                 this.mdiClient1
114:             }
115:         );
116:         this.IsMdiContainer = true;
117:         this.Menu = this.mainMenu1;
118:         this.Name = "MainForm";
119:         this.Text = "MainForm";
120:         this.ResumeLayout(false);
121:
122:     }
123: #endregion
124:
125:
126: /// <summary>
127: /// load any dynamic forms
128: /// </summary>
129: protected void LoadForms() {
130:     string FormsDir = string.Format(
131:         "{0}\\DynamicForms",
132:         Application.StartupPath
133:     );
134:
135:     //Locate all Assemblies (DLL's only)
136:     string[] Files = Directory.GetFiles(
137:         FormsDir, "*.dll" );
138:
139:     foreach( string strFile in Files ) {
140:         System.Reflection.Assembly curAsm =
141:             System.Reflection.Assembly.LoadFrom( strFile );
142:
143:         //Look at the exposed types
144:         System.Type[] types = curAsm.GetTypes();
145:         foreach( Type T in types ) {
146:             if( T.IsSubclassOf( typeof( Form ) ) ) {
147:                 //Create an instance and add to main menu
148:                 Form frm =
149:                     (Form)curAsm.CreateInstance( T.FullName );
150:                 this.forms.Add(
151:                     ((IFormHostClient)frm).MenuItemText, T );
152:                 MenuItem newItem = new MenuItem(
```

```
152:     ((IFormHostClient)frm)
153:     .MenuText,
154:     new EventHandler(
155:         this.OnFormSelectMenu ) );
156:     FormsMenuItem.MenuItems.Add( newItem
157: );
158: }
159: }
160: }
161:
162: /// <summary>
163: /// Create an instance of the requested form
164: /// </summary>
165: /// <param name="sender"></param>
166: /// <param name="e"></param>
167: protected void OnFormSelectMenu( object sender,
168:                                 EventArgs e ) {
169:     MenuItem mi = (MenuItem)sender;
170:     if( this.forms.ContainsKey( mi.Text ) ) {
171:         Type T = (Type)this.forms[ mi.Text ];
172:         Form frmChild =
173:             (Form)Activator.CreateInstance( T );
174:         frmChild.MdiParent = this;
175:         frmChild.Show();
176:     }
177:     /// <summary>
178:     /// The main entry point for the application.
179:     /// </summary>
180:     [STAThread]
181:     static void Main()
182:     {
183:         Application.Run(new MainForm());
184:     }
185: }
186: }
```

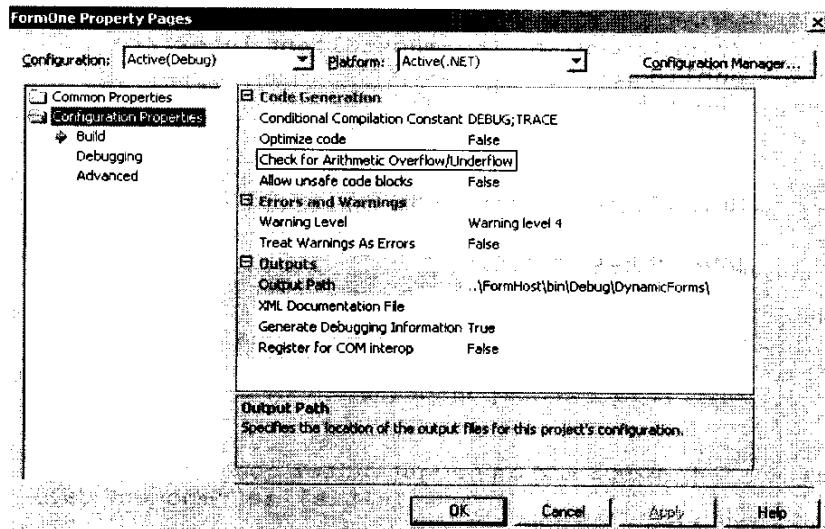
Bên trong các đoạn mã của FormHost, thực sự chỉ có hai phương thức đáng quan tâm. Phương thức thứ nhất, LoadForm, bắt đầu ở dòng 127, chịu trách nhiệm định vị các gói kết hợp (assembly) chứa các form có thể được gọi bên trại shell. Phương thức LoadForm bắt đầu bằng việc lấy một danh sách các gói kết hợp có thể có và duyệt qua chúng. Mỗi gói kết hợp sau đó sẽ được tải lên bằng phương thức tĩnh Assembly.LoadForm. Lớp Assembly nằm bên trong không gian System.Reflection.

Sau khi một gói kết hợp được tải lên, bước tiếp theo là trích rút các kiểu sau đó so sánh các kiểu ấy với hai bước kiểm tra đơn giản. Bước kiểm tra thứ n

là kiểm tra xem kiểu được tải lên có phải là dẫn xuất của kiểu cơ sở System.Windows.Forms.Form hay không. Nếu kiểu được tải lên là đúng, tên của nó sẽ được thêm vào trong menu của Form và các thông tin kiểu sẽ được lưu trong bảng băm.

Khi chọn form được tải lên trên menu, kiểu sẽ được tải từ bảng băm và sử dụng lớp Activator, một thể hiện mới sẽ được tạo ra và hiển thị trên cửa sổ MDI.

Để kết thúc phần minh họa, chúng ta sẽ thêm một dự án lớp thư viện C# vào dự án DynamicAsmLoad hiện tại và đặt tên nó là FormOne. Thư viện FormOne sẽ chứa một lớp đơn được dẫn xuất từ System.Windows.Forms và cũng sẽ được cài đặt giao tiếp cần thiết IFormHostClient. Bạn nhớ tham chiếu vào dự án FormHostSDK hoặc DLL. Thêm vào đó, ứng dụng FormHost tìm kiếm các gói kết hợp nằm trong thư mục con DynamicForms, vì thế phải chắc chắn rằng dự án FormOne sẽ được biên dịch trong thư mục này. Cách dễ nhất là thiết lập các thuộc tính biên dịch của dự án về thư mục này. Hình 5.1-3 là hộp thoại Project Properties và các thiết lập cho thông tin biên dịch.



Hình 5.1-3 Các thuộc tính của dự án FormOne

Ví dụ 5.1-6 là cài đặt của lớp FormOne

Ví dụ 5.1-6 FormOne

```
1: using System;
2: using System.Drawing;
3: using System.Collections;
4: using System.ComponentModel;
5: using System.Windows.Forms;
```

```
6:  
7: using FormHostSDK;  
8:  
9: namespace FormOne  
10:{  
11:     /// <summary>  
12:     /// Summary description for FormOne.  
13:     /// </summary>  
14:     public class FormOne : System.Windows.Forms.Form,  
           FormHostSDK.IFormHostClien  
15:     {  
16:  
17:         /// <summary>  
18:         /// Required designer variable.  
19:         /// </summary>  
20:         private System.ComponentModel.Container  
               components = null;  
21:  
22:  
23:         // Implement the IFormHostClient interface  
24:         public string MenuText {  
25:             get {  
26:                 return "Form One";  
27:             }  
28:         }  
29:  
30:  
31:         public FormOne()  
32:         {  
33:             //  
34:             // Required for Windows Form Designer support  
35:             //  
36:             InitializeComponent();  
37:             //  
38:             //  
39:             // TODO: Add any constructor code after  
                   InitializeComponent call  
40:             //  
41:         }  
42:  
43:         /// <summary>  
44:         /// Clean up any resources being used.  
45:         /// </summary>  
46:         protected override void Dispose( bool disposing )  
47:         {  
48:             if( disposing )  
49:             {  
50:                 if(components != null)  
51:                 {
```

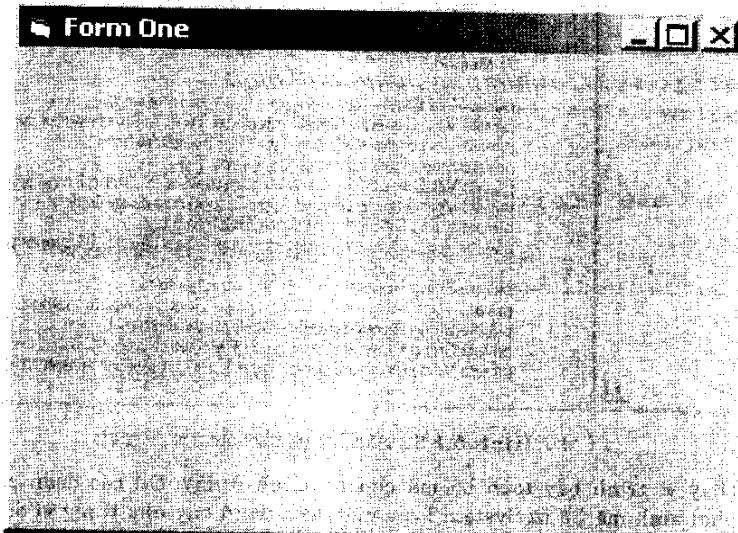


```

52:             components.Dispose();
53:         }
54:     }
55:     base.Dispose(disposing);
56: }
57:
58: #region Windows Form Designer generated code
59: /// <summary>
60: /// Required method for Designer support - do not
61: /// the contents of this method with the code
62: /// </summary>
63: private void InitializeComponent()
64: {
65:     this.components = new
66:         System.ComponentModel.Container();
67:     this.Size = new System.Drawing.Size(300, 300);
68:     this.Text = "FormOne";
69: }
70: }
71: }

```

Với sự hoàn tất lớp FormOne, chỉ cần biên dịch toàn bộ giải pháp và chạy thử ứng dụng FormHost. Nếu nó chạy tốt, một menu sẽ hiện ra trên form, và khi được chọn, cửa sổ FormOne sẽ xuất hiện như hình 5.1-4.



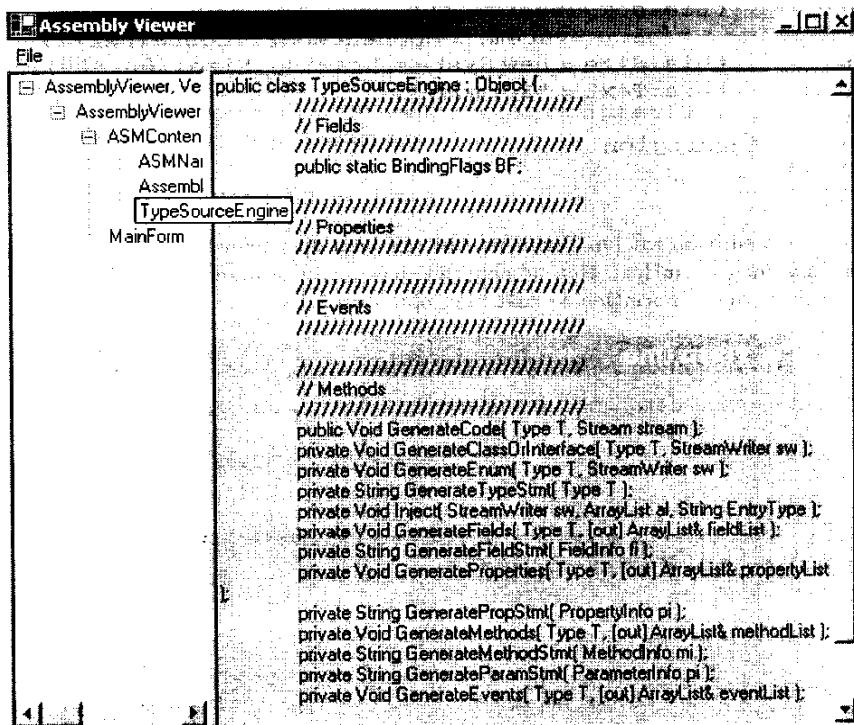
Hình 5.1-4 FormHost với FormOne

Mặc dù ví dụ này không ấn tượng lắm, nhưng nó đem đến cho bạn những cơ sở đủ để bắt đầu một ứng dụng lớn với nhiều tính năng hơn.

5.2. Trình duyệt gói (Assembly Viewer) đơn giản

Suốt cuốn sách này, các Reflection API đã được dùng để lượm lặt những thông tin về kiểu và các gói kết hợp. Các gói kết hợp .NET tự mô tả về mình khá đầy đủ đến nỗi bạn có thể dịch ngược một .NET assembly từ IL sang C#. Các qui trình để phát sinh C# từ IL sẽ mất một ít công sức, tuy nhiên, phát sinh các cấu trúc đơn giản của lớp .NET từ các mô tả được tìm thấy bên trong một gói thì lại đơn giản hơn rất nhiều.

Assembly Viewer, như trong hình 5.1-5, đã được xây dựng trong khoảng 30 phút và không tốn một công sức nào cả. Mặc dù Assembly Viewer không nhận vào IL và dịch ngược nó nhưng việc thêm vào tính năng này là hoàn toàn có thể.



The screenshot shows the Assembly Viewer application window. On the left, a tree view displays the assembly structure:

- AssemblyViewer, Version: 1.0.0.0, Culture: neutral, Public Key Token: null
 - AssemblyViewer
 - ASMContent
 - ASMNai
 - Assembl
 - TypeSourceEngine
 - MainForm

The main pane shows the generated C# code for the `TypeSourceEngine` class. The code includes sections for Fields, Properties, Events, Methods, and Events. The `TypeSourceEngine` class is derived from `Object`.

```

public class TypeSourceEngine : Object {
    // Fields
    public static BindingFlags BF;

    // Properties
    // Events
    // Methods
    public Void GenerateCode1( Type T, Stream stream );
    private Void GenerateClassOrInterface( Type T, StreamWriter sw );
    private Void GenerateEnum( Type T, StreamWriter sw );
    private String GenerateTypeSmi( Type T );
    private Void Inject( StreamWriter sw, ArrayList<String> list, String Entity );
    private Void GenerateFields( Type T, [out]ArrayList<FieldInfo> list );
    private String GenerateFieldSmi( FieldInfo fi );
    private Void GenerateProperties( Type T, [out]ArrayList<PropertyInfo> list );
    private String GeneratePropSmi( PropertyInfo pi );
    private Void GenerateMethods( Type T, [out]ArrayList<MethodInfo> list );
    private String GenerateMethodSmi( MethodInfo mi );
    private String GenerateParamSmi( ParameterInfo pi );
    private Void GenerateEvents( Type T, [out]ArrayList<EventInfo> list );
}

```

Hình 5.1-5 Assembly Viewer

Thay vì trình bày toàn bộ mã chương trình ở đây, chỉ các đoạn có trách nhiệm phát sinh mã C# từ `System.Type` mới được trình bày như trong ví dụ 5.1-7.

Ví dụ 5.1-7

```
1: using System;
2: using System.Collections;
3: using System.Reflection;
4:
5: namespace AssemblyViewer.ASMContent
6: {
7:     /// <summary>
8:     /// Summary description for TypeSourceEngine.
9:     /// </summary>
10:    public class TypeSourceEngine
11:    {
12:        public static BindingFlags BF =
13:            BindingFlags.Public |
14:            BindingFlags.NonPublic |
15:            BindingFlags.Instance |
16:            BindingFlags.Static |
17:            BindingFlags.DeclaredOnly;
18:
19:        /// <summary>
20:        /// Generate C# code from a given Type
21:        /// </summary>
22:        /// <param name="T">Type used for code
23:                               generation</param>
24:        /// <param name="stream">Stream object to write
25:                               to</param>
26:        public static void GenerateCode( Type T,
27:                                         System.IO.Stream stream ) {
28:
29:            System.IO.StreamWriter sw = new
30:                System.IO.StreamWriter( stream );
31:
32:            if( T.IsEnum )
33:                GenerateEnum( T, sw );
34:            else
35:                GenerateClassOrInterface( T, sw );
36:
37:        }
38:
39:        /// <summary>
```

```
36:     /// If the Type is a class or interface, generate the
           proper code
37:     /// </summary>
38:     /// <param name="T"></param>
39:     /// <param name="sw"></param>
40:     private static void GenerateClassOrInterface( Type
           T, System.IO.StreamWriter sw ) {
41:         ArrayList Fields;
42:         ArrayList Properties;
43:         ArrayList Methods;
44:         ArrayList Events;
45:
46:         GenerateFields( T, out Fields );
47:         GenerateProperties( T, out Properties );
48:         GenerateEvents( T, out Events );
49:         GenerateMethods( T, out Methods );
50:
51:         sw.WriteLine( GenerateTypeStmt( T ) );
52:         sw.WriteLine( "\r\n" );
53:         Inject( sw, Fields, "Fields" );
54:         Inject( sw, Properties, "Properties" );
55:         Inject( sw, Events, "Events" );
56:         Inject( sw, Methods, "Methods" );
57:         sw.Write("}");
58:         sw.Flush();
59:     }
60:
61:     /// <summary>
62:     /// Generate code for an Enum type
63:     /// </summary>
64:     /// <param name="T"></param>
65:     /// <param name="sw"></param>
66:     private static void GenerateEnum( Type T,
           System.IO.StreamWriter sw )
67:     {
68:         ArrayList Fields;
69:         GenerateFields( T, out Fields );
70:         sw.WriteLine( GenerateTypeStmt( T ) );
71:         sw.WriteLine( "\r\n" );
72:         Inject( sw, Fields, "Fields" );
73:         sw.Write("}");
74:         sw.Flush();
```

```
75:     }
76:
77:     /// <summary>
78:     /// Creates a type declaration statement:
79:     /// (public | private) [abstract|sealed]
[interface|class|struct] TypeName [: [Base Class],
[Interfaces] ]
80:     /// </summary>
81:     /// <param name="T"></param>
82:     /// <returns></returns>
83:     private static string GenerateTypeStmt( Type T ) {
84:         string stmt = "";
85:
86:         if( T.IsPublic )
87:             stmt = "public ";
88:         else if( T.IsNotPublic )
89:             stmt = "private ";
90:
91:         if( T.IsAbstract )
92:             stmt += "abstract ";
93:         else if( T.IsEnum )
94:             stmt += "enum ";
95:         else if( T.IsInterface )
96:             stmt += "interface ";
97:         else if( T.IsSealed )
98:             stmt += "sealed ";
99:
100:        if( T.IsClass )
101:            stmt += "class ";
102:        else if (T.IsInterface )
103:            stmt += "interface ";
104:        else
105:            stmt += "struct ";
106:
107:        bool bHasBase = false;
108:        stmt += string.Format( "{0} ", T.Name );
109:        if( T.BaseType != null ) {
110:            stmt += string.Format(": {0}",
111:                                T.BaseType.Name);
112:            bHasBase = true;
113:        }
```

```
114:     System.Type[] Interfaces = T.GetInterfaces();
115:     if( Interfaces.Length != 0 ) {
116:         if( !bHasBase )
117:             stmt += ":" ;
118:         foreach( Type tt in Interfaces ) {
119:             stmt += tt.Name;
120:             stmt += ", " ;
121:         }
122:         stmt = stmt.Substring(0, stmt.Length -
123: );
124:         stmt += " {";
125:         return stmt;
126:     }
127:
128:     /// <summary>
129:     /// Inject source into StreamWriter
130:     /// </summary>
131:     /// <param name="al"></param>
132:     /// <param name="EntryType"></param>
133:     private static void Inject(
134:         System.IO.StreamWriter s,
135:         ArrayList al,
136:         string EntryType ) {
137:         sw.WriteLine("\t/////////////////////////////");
138:         sw.WriteLine(string.Format("\t// {0}\r\n",
139:             EntryTyp
140:         );
141:         sw.WriteLine("\t");
142:         foreach( string s in al ) {
143:             sw.WriteLine(s);
144:             sw.WriteLine("\r\n");
145:         }
146:
147:
148:     /// <summary>
149:     /// Generate Field declarations
```

```
150:     /// </summary>
151:     /// <param name="T"></param>
152:     /// <param name="fieldList"></param>
153:     private static void GenerateFields( Type T, out
154:                                         ArrayList fieldList )
155:     {
156:         fieldList = new ArrayList();
157:         FieldInfo[] fields = T.GetFields( BF );
158:         foreach( FieldInfo fi in fields ) {
159:             fieldList.Add( GenerateFieldStmt( fi ) );
160:         }
161:
162:         /// <summary>
163:         /// Generate the actual field stmt
164:         /// ie: private int someFieldMember;
165:         /// </summary>
166:         /// <param name="fi"></param>
167:         /// <returns></returns>
168:         private static string GenerateFieldStmt(
169:                                         FieldInfo fi ) {
170:
171:             string stmt;
172:
173:             if( fi.IsPublic )
174:                 stmt = "public ";
175:             else if( fi.IsPrivate )
176:                 stmt = "private ";
177:             else
178:                 stmt = "protected ";
179:
180:             if( fi.IsStatic )
181:                 stmt += "static ";
182:
183:             stmt += fi.FieldType.Name;
184:             stmt += " ";
185:             stmt += fi.Name;
186:             stmt += ";";
187:             return stmt;
188:         }
189:         private static void GenerateProperties( Type T,
```

```
190:         out ArrayList propertyList ) {
191:
192:             PropertyInfo[] props = T.GetProperties( );
193:             propertyList = new ArrayList();
194:             foreach( PropertyInfo pi in props )
195:                 propertyList.Add( GeneratePropStmt( pi ) );
196:
197:     }
198:
199:     private static string GeneratePropStmt(
200:                                     PropertyInfo pi ) {
201:         string stmt = "public ";
202:         stmt += pi.PropertyType.Name;
203:         stmt += " ";
204:         stmt += pi.Name;
205:         stmt += " { ";
206:         if( pi.CanRead )
207:             stmt += "get; ";
208:         if( pi.CanWrite )
209:             stmt += "set; ";
210:         stmt += " } ; ";
211:         return stmt;
212:     }
213:
214:     private static void GenerateMethods( Type T,
215:                                         out ArrayList methodList ) {
216:         MethodInfo[] Methods = T.GetMethods( BF );
217:         methodList = new ArrayList();
218:         foreach( MethodInfo mi in Methods )
219:             methodList.Add( GenerateMethodStmt( mi ) );
220:
221:     }
222:
223:     private static string GenerateMethodStmt(
224:                                     ' MethodInfo mi ) {
225:         string stmt;
226:
227:         if( mi.IsPublic )
228:             stmt = "public ";
229:         else if( mi.IsPrivate )
230:             stmt = "private ";
```



```
230:         else if( mi.IsFamily )
231:             stmt = "protected ";
232:         else
233:             stmt = "protected internal ";
234:
235:         if( mi.IsVirtual )
236:             stmt += "virtual ";
237:         else if( mi.IsAbstract )
238:             stmt += "abstract ";
239:
240:         stmt += string.Format("{0} {1}({ ",
241:                         mi.ReturnType.Name, mi.Name );
242:         ParameterInfo[] Params = mi.GetParameters();
243:         if( Params.Length > 0 ) {
244:             int i;
245:             for( i = 0; i < Params.Length - 1; i++ )
246:                 stmt += string.Format("{0},",
247:                     GenerateParamStmt( Params[i] ) );
248:             stmt += string.Format("{0} ",
249:                     GenerateParamStmt( Params[ i ] ) );
250:         }
251:         stmt += ");";
252:         return stmt;
253:     }
254:
255:     private static string GenerateParamStmt(
256:                                         ParameterInfo pi ) {
257:
258:         string stmt = "";
259:         if( pi.IsIn )
260:             stmt = "[in] ";
261:         else if( pi.IsOut )
262:             stmt = "[out] ";
263:         else if( pi.IsRetval )
264:             stmt = "[ref] ";
265:         stmt += string.Format( "{0}"
266:                         ,pi.ParameterType.Name, pi.Name );
267:         return stmt;
268:     }
269:
270:     private static void GenerateEvents( Type T,
271:                                         out ArrayList eventList ) {
272:         EventInfo[] Events = T.GetEvents( BF );
```

```
267:         eventList= new ArrayList( );
268:         foreach( EventInfo ei in Events )
269:             eventList.Add( GenerateEventStmt( ei ) );
270:     }
271:
272:     private static string GenerateEventStmt(
273:                                         EventInfo ei ) {
274:         return string.Format("public {0} {1};",
275:                             ei.EventHandlerType.Name, ei.Name );
276:     }
277: }
```

Đoạn mã phát sinh C# trên đây khá đơn giản, chúng ta có thể cải tiến ví dụ 5.1-7. Cải tiến thứ nhất là loại bỏ tất cả các cấu trúc C# được gõ bằng tay vào và dùng lớp CodeDom, nằm ở System.CodeDom để thay thế. Sau đó, dĩ nhiên, là nạp một IL từ một gói kết hợp .NET và phát sinh các câu lệnh chính xác bằng ngôn ngữ đã được chọn.

6. KẾT CHƯƠNG

Các gói kết hợp .NET là cơ chế rất mạnh để đóng gói không chỉ các mã IL mà còn các thông tin về kiểu mà nhờ đó gói kết hợp có thể tự giới thiệu đầy đủ về nó. Trong chương tiếp theo, chủ đề về chữ ký số và phiên bản hoá các gói kết hợp sẽ được trình bày. Sử dụng các gói kết hợp .NET và các Reflection API sẽ chứng minh chúng là những tài nguyên vô giá để mở rộng đời sống của các ứng dụng và khiến chúng dễ thay đổi, bảo trì, nâng cấp hơn.

Chương 5.2

CHỮ KÝ SỐ VÀ PHIÊN BẢN

Các vấn đề chính sẽ được đề cập đến:

- ✓ *DLL hell*
- ✓ *Bộ đệm (cache) toàn cục dành cho gói kết hợp*
- ✓ *Quản lý phiên bản*
- ✓ *Cấu hình ứng dụng*

1. DLL HELL

Hầu như tất cả chúng ta đều biết, một trong những mục đích khi tạo ra Win32 DLLs là để dùng lại mã dễ dàng. Chúng ta cũng biết rằng các phiên bản (version) của những DLLs khác nhau có thể giúp nâng cấp một ứng dụng tốt hơn nhưng cũng có thể dễ dàng làm hỏng chương trình hiện hành đang chạy tốt. Các nguyên tắc của mô hình đối tượng COM đã cố gắng khắc phục vấn đề này thông qua những qui luật khắc khe, nhưng những qui luật này được áp dụng bởi các nhà phát triển và không bị bắt buộc trong bất cứ thư viện thực thi (runtime) nào. Vì vậy, ngay cả các ứng dụng dùng các thành phần COM (COM component) cũng chỉ ra rằng không phải tất cả các nhà phát triển COM đều theo các qui tắc này, và phần mềm sẽ sụp đổ khi sử dụng các version DLL khác nhau cho 1 component chung.

Với .NET, Microsoft đã chọn một hướng mới để giải quyết vấn đề đáng sợ về DLL này. .NET runtime tôn trọng version của các component và còn cho phép cài đặt các component theo những version khác nhau. Từ nay, các ứng dụng khách client hoàn toàn có thể chọn loại version DLL để dùng; theo mặc định, một ứng dụng sẽ nạp version của DLL mà nó đã ràng buộc vào lúc biên dịch ban đầu. Nếu có sẵn 1 version mới của một thành phần component dùng chung, ứng dụng này có thể được cấu hình lại để dùng version mới của DLL hoặc tiếp tục dùng DLL cũ hiện hành.

2. BỘ ĐỆM (CACHE) TOÀN CỤC DÀNH CHO GÓI KẾT HỢP (GLOBAL ASSEMBLY CACHE)

Global Assembly Cache (GAC) có nghĩa là sự thay thế cho các component dùng chung. Trong thế giới COM, mỗi khái niệm về COM như coclass, interface, progid, v.v... được đăng ký trong bộ Registry của hệ thống. Hệ thống phụ COM dùng thông tin này để xác định component COM và nạp đối tượng này. Trong .NET, GAC hoạt động như là chỗ ở cho tất cả component dùng chung. Bất cứ component .NET nào cũng được cài đặt trong GAC nhờ tiện ích gacutil.exe đi cùng với .NET runtime hoặc có thể dùng Windows Explorer và kéo component này bô vào trong thư

mục GAC. GAC nằm trong 1 thư mục con của thư mục system cơ sở; trong Windows 2000, tất cả component dùng chung nằm trong c:\WINNT\assembly nếu bạn chọn cài đặt mặc định. Hình 5.2-1 là nội dung của GAC.

Mỗi component .NET có 1 bảng mô tả Manifest như là 1 phần của tập tin gói; bảng mô tả này gồm thông tin về version được dùng trong bộ nạp CLR để buộc version component .NET phải tuân theo. Con số biểu diễn version gồm những thông tin sau:

- Major (thông tin chính)
- Minor (thông tin phụ)
- Revision (thông tin khi duyệt lại component này)
- Build Number (thông tin khi biên dịch component này)

Global Assembly Name	Type	Version	Culture	Public Key Token
Accessibility		1.0.2411.0		b03f5f7f11d50a3a
ADODB		2.7.0.0		b03f5f7f11d50a3a
CRVsPackageLib		1.0.0.0		4f3430cff154c24c
CrystalDecisions.CrystalReports.Engine		9.1.0.0		4f3430cff154c24c
CrystalDecisions.ReportSource		9.1.0.0		4f3430cff154c24c
CrystalDecisions.Shared		9.1.0.0		4f3430cff154c24c
CrystalDecisions.Web		9.1.0.0		4f3430cff154c24c
CrystalDecisions.Windows.Forms		9.1.0.0		4f3430cff154c24c
CrystalEnterpriseLib		1.0.0.0		4f3430cff154c24c
CrystalInfoStoreLib		1.0.0.0		4f3430cff154c24c
CrystalKeyCodeLib		1.0.0.0		4f3430cff154c24c
CrystalPluginMgrLib		1.0.0.0		4f3430cff154c24c
CrystalReportPluginLib		1.0.0.0		4f3430cff154c24c
65 object(s)			777 bytes	

Hình 5.2-1 Nội dung của thư mục GAC

Các thông tin về version ở trên được dùng để đảm bảo nạp đúng component .NET cho 1 ứng dụng client cho trước. Nếu ứng dụng client yêu cầu 1 version không có sẵn, bộ nạp CLR sẽ thất bại và nó sẽ báo cáo lại thông tin về lỗi này.

3. PHIÊN BẢN (VERSION) CỦA COMPONENT

Có 1 số bước phải tuân theo để tạo các component được phiên bản hóa. Vào thời điểm viết bài này, VS.NET chưa hỗ trợ khả năng tạo tự động các component đã được ký nhận bằng chữ ký số. Bước đầu tiên là phát sinh 1 khóa ký nhận số (signing key) nhờ tiện ích SN.exe đi cùng với bộ SDK .NET. Nói chung, 1 công ty sẽ tạo ra 1 khoá như trên và tất cả thành phần component .NET mà họ tự tạo,

đồng thời dùng khóa này để ký nhận cho component. Để tạo ra 1 cặp khóa public/private, ta dùng lệnh sau:

```
sn.exe -k sams.snk
```

Lệnh này tạo ra 1 tập tin khóa được dùng để ký nhận gói kết hợp chứa những thành phần component. Với tập tin khoá vừa tạo, ta tạo mới 1 dự án (project) VS.NET cho thư viện lớp C# và đặt tên dự án này là SharedComponent. Kế tiếp, ban thêm 1 lớp mới mang tên SharedServer, cho dự án này và nhập vào đoạn mã như trong ví dụ 5.2-1 dưới đây.

Ví dụ 5.2-1 Mã nguồn SharedServer

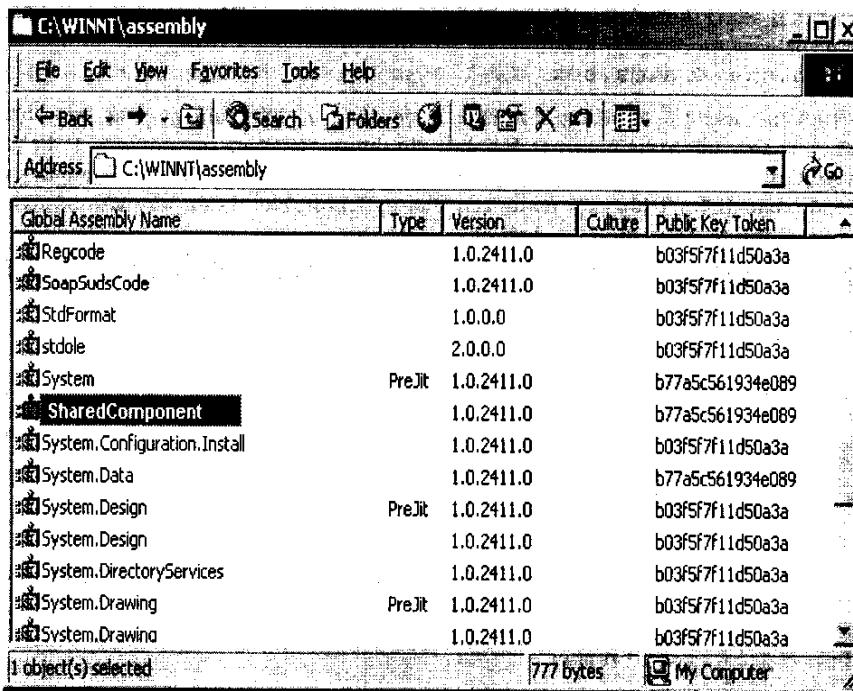
```
1: using System;
2:
3: namespace SharedComponent
4: {
5:     /// <summary>
6:     /// Summary description for Class1.
7:     /// </summary>
8:     public class SharedServer
9:     {
10:         //Single method to return version information
11:         public string WhoAmI( ) {
12:             string me = "Hello from version 1.1";
13:             return me;
14:         }
15:
16:     }
17: }
```

Sau đó lớp này sẽ được dùng để kiểm tra việc hỗ trợ version được cung cấp bởi bộ nạp CLR. Để tạo ra 1 dự án hỗ trợ việc ký nhận, ta sửa tập tin AssemblyInfo.cs và thêm thông tin khóa, như trong ví dụ 5.2-2 sau:

Ví dụ 5.2-2 AssemblyInfo.cs

```
1: [assembly: AssemblyDelaySign(false)]
2: [assembly: AssemblyKeyFile("D:\\SAMS\\SAMS.SNK")]
3: [assembly: AssemblyKeyName("")]
```

Lưu ý rằng thuộc tính [assembly: AssemblyKeyFile(...)] được dùng để chỉ định tập tin khóa cho việc ký nhận gói assembly này. Đó là các bước phải làm. Bây giờ, ta biên dịch gói vừa tạo và kéo DLL đã được biên dịch vào trong thư mục GAC mang tên c:\\WINNT\\assembly bằng Windows Explorer. Hình 5.2-2 cho thấy Windows Explorer của thư mục GAC và SharedComponent.dll mới được cài đặt.



Hình 5.2-2 SharedComponent.dll trong thư mục GAC

Sau khi đặt gói assembly chung đúng chỗ, ta có thể tạo ra một trình khác client sử dụng SharedComponent.dll này. Bạn tạo mới 1 dự án Windows Forms, thêm 1 tham chiếu tới gói SharedComponent bằng cách chọn dự án, nhấn phí phải chuột và chọn Add Reference từ menu tắt shortcut.

Để kiểm tra SharedComponent này hoạt động, bạn đặt một nhãn vào for chương trình sau đó thêm đoạn mã như ví dụ 5.2-3 vào trong phương thức kh^u dụng của form này.

Ví dụ 4.4.1 Kiểm tra SharedComponent

```

1: public Form1()
2: {
3:     //
4:     // Required for Windows Form Designer support
5:     //
6:     InitializeComponent();
7:
8:     //
9:     // TODO: Add any constructor code after
InitializeComponent call

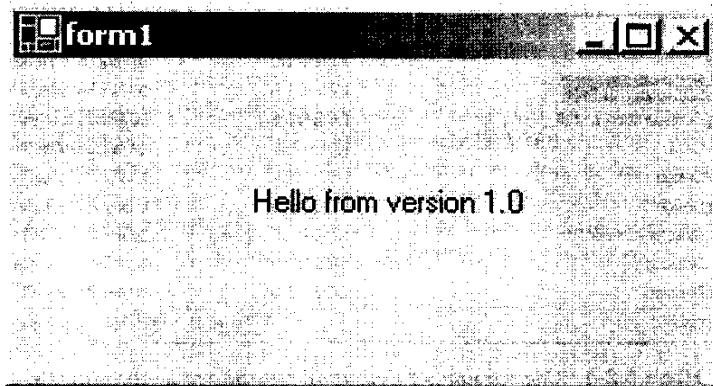
```

```

10:    //
11:    SharedComponent.SharedServer ss = new
12:        SharedComponent.SharedServer( );
13:    this.label1.Text = ss.WhoAmI( );

```

Các dòng 11, 12 tạo đối tượng SharedServer mới và gán thuộc tính Text của nhãn cho chuỗi trả về của hàm WhoAmI. Bây giờ, biên dịch và chạy ứng dụng client để kiểm tra sự kết nối của gói assembly này. Kết quả chạy sẽ hiển thị như hình 5.2-3.



Hình 5.2-3 Cửa sổ chương trình thử nghiệm

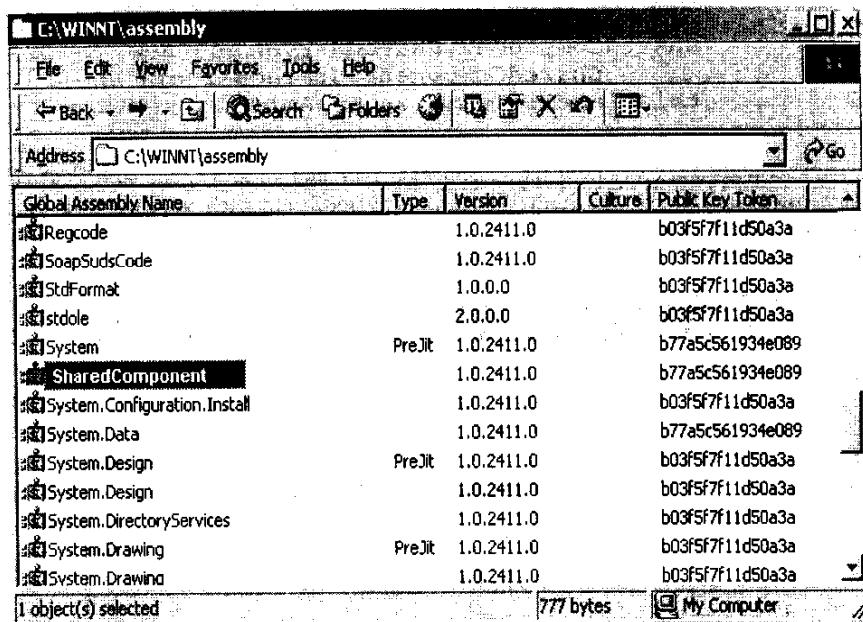
Ứng dụng client trên được ràng buộc với version hiện hành của gói assembly dùng chung nhờ vào thông tin version có sẵn khi biên dịch client. Trong phần tới, 1 version mới hơn của component này sẽ được tạo và cài đặt vào trong GAC. Sau khi làm xong điều này, ứng dụng client vẫn dùng version đầu tiên của DLL cho tới khi nó được cấu hình để dùng version mới hơn.

4. CÁC GÓI ASSEMBLY SỬ DỤNG NHIỀU PHIÊN BẢN

Để minh họa khả năng hỗ trợ các phiên bản liên tiếp nhau của các gói assembly .NET, bạn hãy mở lại dự án ShareComponent và thực hiện các thay đổi sau:

1. Cập nhật thuộc tính version trong AssemblyInfo.cs thành 1.1.
2. Thay đổi chuỗi trả về của hàm WhoAmI để phản ánh thông tin version mới.
3. Biên dịch và tạo gói assembly mới.
4. Cài đặt version mới của assembly này vào trong thư mục GAC.

Sau khi hoàn tất các bước trên, hai phiên bản version của ShareComponent.dll sẽ xuất hiện trong GAC như hình 5.2-4.



Hình 5.2-4 Các version kế tiếp nhau của SharedComponent.dll

Chạy lại ứng dụng client và lưu ý rằng thông báo trả về vẫn là “Hello from version 1.0.”. Điều quan trọng ở đây là không phải biên dịch lại ứng dụng client và sự tham chiếu giờ đây sẽ là version mới hơn của DLL này.

5. RÀNG BUỘC TÙY BIỂN: CẤU HÌNH ỨNG DỤNG

Để ứng dụng client dùng được gói assembly mới biên dịch này, bạn phải tệp tin dùng để cấu hình ứng dụng đặt nằm trong thư mục ứng dụng. Tập tin cấu hình sẽ có cùng tên với tên ứng dụng, bao gồm luôn cả phần mở rộng. Trong trường hợp của chương trình SharedClient.exe, tập tin cấu hình sẽ có tên là SharedClient.exe.config. Nội dung tập tin cấu hình này được thể hiện trong ví dụ 5.2-4, nó cho thấy các điểm cần thiết để hướng việc kết nối từ version 1.0 sang version 1.1 của SharedComponent.dll.

Ví dụ 5.2-4 SharedClient.exe.config

```

1: <?xml version="1.0" ?>
2: <configuration>
3:   <runtime>
4:     <assemblyBinding xmlns="urn:schemas-microsoft-
      com:asm.v1">
```

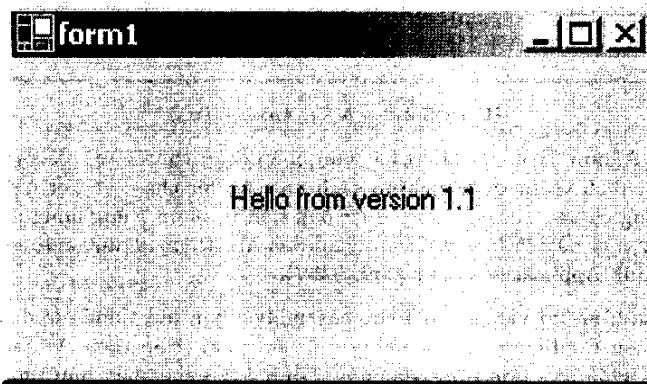
```

5:      <dependentAssembly>
6:          <assemblyIdentity name="SharedComponent"
7:              publicKeyToken="9ab72e2bd01f38e8" />
8:          <bindingRedirect oldVersion="1.0.547.38856"
9:              newVersion="1.1.547.38908" />
10:     </dependentAssembly>
11:   </assemblyBinding>
12: </runtime>
13: </configuration>

```

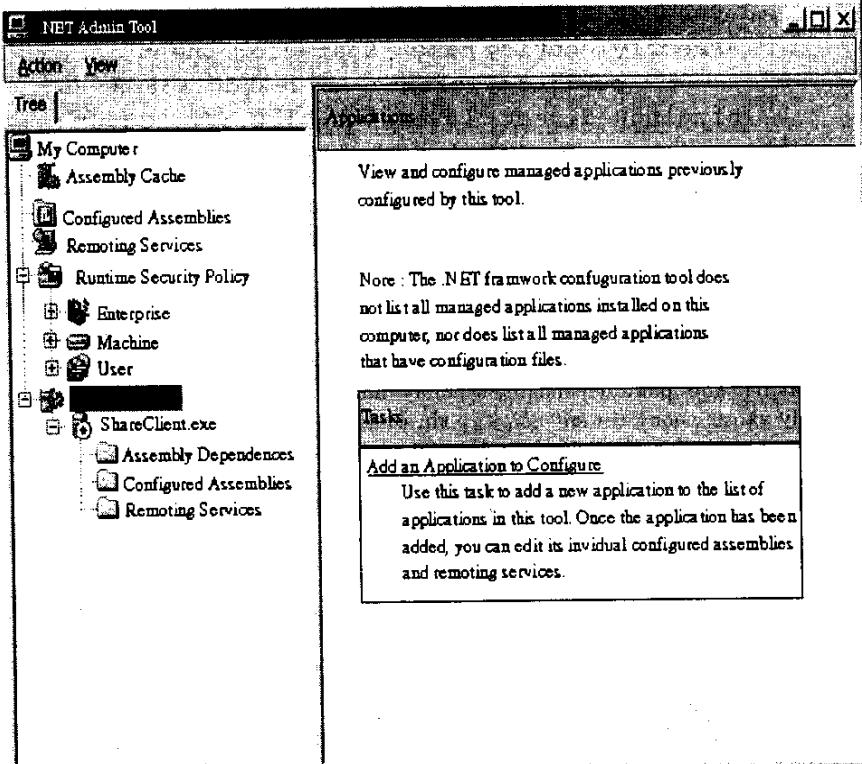
Trong phần assemblyBinding, tất cả gói assembly phụ thuộc được liệt kê ra để kết nối lại. Mỗi assembly được định hướng lại sẽ có 1 điểm nhập mới. Các điểm nhập này bao gồm phần dependentAssembly diễn đạt chi tiết sự đồng nhất của gói assembly và các thông tin kết nối. Tập tin config là 1 tập tin XML đơn giản được dùng trong thời gian chạy để thay đổi môi trường cho bất cứ ứng dụng nào. Bạn tạo 1 tập tin văn bản tên là SharedClient.exe.config và chép vào ví dụ 5.2-4 ở trên; nhớ kiểm tra lại các con số version vì chúng trùng tương tự nhau. Thông tin version được lấy từ việc duyệt thư mục GAC bằng Windows Explorer, như trong hình 5.2-4.

Sau khi để tập tin cấu hình đúng chỗ, chạy ứng dụng SharedClient.exe và lưu ý dòng chữ hiện trong nhãn bây giờ là “Hello from version 1.1” (hình 5.2-5). Có được điều này là nhờ việc kết nối lại các assembly lúc chạy chương trình.



Hình 5.2-5 SharedClient.exe với kết nối đã được định hướng lại

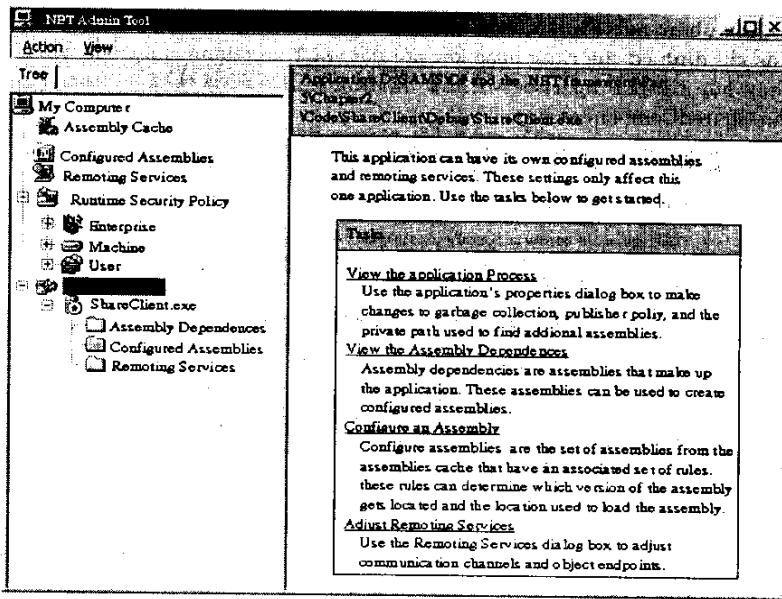
Thay vì tạo tập tin cấu hình bằng tay, ứng dụng framework .NET đi kèm với MMC snap-in cho phép bạn cấu hình các ứng dụng, bao gồm cả việc kết nối lại assembly. Thực sự ở đây chúng ta dùng MMC snap-in .NET Admin Tool (mscorcfg.msc) để tạo tập tin cấu hình cho chương trình SharedClient.exe. Hình 5.2-6 cho thấy .NET Admin Tool được nạp trong MMC.



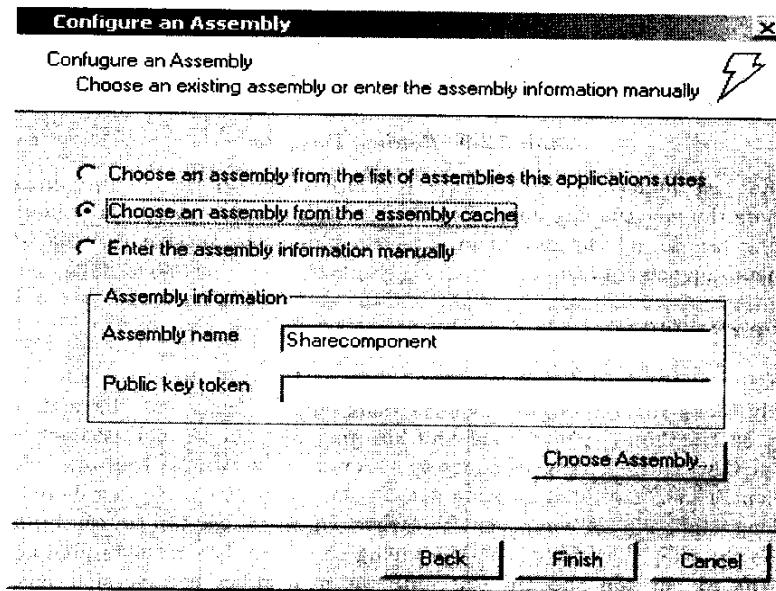
Hình 5.2-6 .NET Admin Tool

.NET Admin Tool của Microsoft cho phép soạn thảo GAC, các assembly đã cấu hình, các dịch vụ từ xa, các chính sách an toàn lúc chạy, và kết nối assembly. Bản thân công cụ này khá dễ dùng. Bước đầu tiên là thêm 1 ứng dụng để cấu hình trong trường hợp này là SharedClient.exe. Sau khi thêm, 1 nút mới sẽ xuất hiện bên dưới nhánh Applications (xem hình 5.2-7).

Để cấu hình lại việc kết nối cho assembly, chọn nút Configured Assembly và sau đó chọn Configure an Assembly ở ô bên phải. Hộp thoại Wizard Style sẽ xuất hiện để giúp bạn làm công việc này. Hình 5.2-8 cho thấy hộp thoại chọn gán assembly cùng với gói assembly SharedComponent được chọn trước. Để ứng dụng client dùng 1 assembly mới hơn, client phải được liên kết lại với assembly này hoặc phải có sẵn cấu hình kết nối cho client này. Trong COM nguyên thủy, các client thông thường dùng các đối tượng COM mới nhất vì COM yêu cầu tất cả giá tiếp phái bất biến. Trong .NET, 1 ứng dụng client sẽ cố gắng nạp assembly được ràng buộc kết nối lúc ban đầu trừ khi tập tin cấu hình ứng dụng chỉ định rắc buộc kết nối với 1 gói assembly khác.

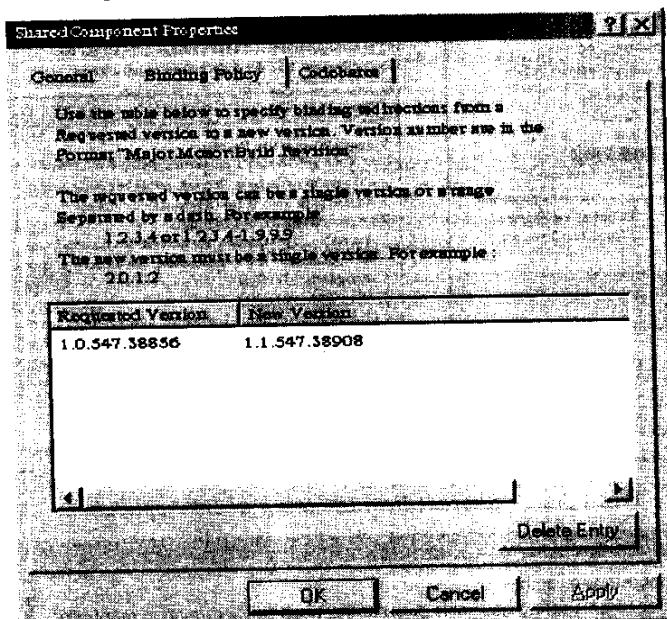


Hình 5.2-7 SharedClient.exe trong nhóm Applications



Hình 5.2-8 Hộp thoại chọn Assembly

Khi nhấn nút Finish, hộp thoại Properties sẽ xuất hiện. Hộp thoại này dùng để chỉ định cơ chế ràng buộc kết nối đối với 1 ứng dụng cho trước. Hình 5.2-9 cho thấy hộp thoại Properties với tab Binding Policy được chọn và thông tin kết nối cho SharedComponent này.



Hình 5.2-9 Binding Policy tab

Kết quả thu được từ .NET Admin Tool là lưu được tập tin cấu hình ứng dụng trong cùng thư mục của ứng dụng. Tập tin cấu hình này được dùng như 1 phần cung cấp nâng cấp để gửi cho các client cùng với phiên bản version mới của bất kỳ component dùng chung nào.

6. KẾT CHƯƠNG

Các chương 5.1 và 5.2 vừa minh họa cách .NET giao tiếp với các assembly thông thường và gói assembly dùng chung. Thuật ngữ “địa ngục DLL” giờ đây đã trở thành vấn đề quá khứ khi giao tiếp với các gói assembly dùng chung vì GAC có khả năng tổ chức các phiên bản version kế tiếp nhau để các component hoạt động không bị xung đột. Do đó, nó cho phép các ứng dụng client dùng đúng version của 1 component cho trước. Ngoài ra, các tập tin cấu hình ứng dụng còn cho phép thay đổi kết nối mà không phải biên dịch và phân phối lại ứng dụng. Tất cả chỉ cần tập tin cấu hình và một phiên bản version mới của component dùng chung, sau đó các client có thể tái ràng buộc kết nối để dùng assembly mới này.

Chương 5.3

TƯƠNG TÁC VỚI THẾ GIỚI COM

Các vấn đề chính sẽ được đề cập đến:

- ✓ Thế giới của COM
- ✓ .NET hỗ trợ COM
- ✓ Đưa ra những thành phần của .NET như những đối tượng COM.

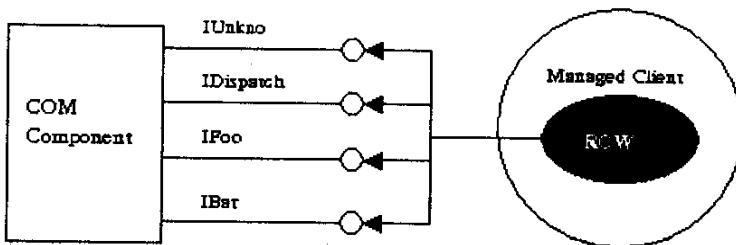
1. THẾ GIỚI CỦA COM

Có từ lâu đời về tầm nhìn rộng của nhiều ngôn ngữ trong việc chia sẻ những component dùng chung. Trong thế giới Windows, tầm nhìn rộng này được thực hiện bởi COM (Component Object Model). Mục đích duy nhất của COM là cho phép bất kỳ ngôn ngữ nào hiểu được chuẩn nhị phân của COM sẽ sử dụng được đối tượng component có trong hệ thống. Với sự xuất hiện của .NET, những vấn đề này không xuất hiện bởi vì tất cả ngôn ngữ .NET có thể thoải mái tương tác với ngôn ngữ khác dễ dàng. Sự phối hợp của CLS và CLR tạo ra tương tác như vậy.

Trước khi mạo hiểm vào thế giới của COM, chương này cho rằng bạn đã có chút ít quen thuộc với COM. Những chủ đề như giao tiếp (interface) (IUnknown, IDispatch, và IEnumVARIANT), coclass, Connection Points và COM threading đã phần nào biết qua. .NET cung cấp nhiều điều kiện thuận lợi cho trình ứng dụng khách hoạt động trong chế độ quản lý (Managed COM) dễ dàng tương tác với những thành phần COM không được quản lý (Unmanaged COM) và những đối tượng Managed COM được sử dụng như thể là chính chúng là những thành phần COM kinh điển.

2. .NET HỖ TRỢ COM

Vì các đoạn mã tự quản lý (Managed code) sử dụng những dịch vụ được cung cấp bởi thành phần COM, bạn sẽ không tìm thấy CoInitialize và CoCreateInstance bên trong kiến trúc .NET. Môi trường .NET giới thiệu khái niệm gọi COM thông qua lớp vỏ bọc - Runtime Callable Wrapper (RCW). Các đoạn mã của .NET sẽ gọi đối tượng COM thông qua RCW và Managed component. Hình 5.5-1 mô tả toàn cảnh nhìn chung của các ứng dụng được .NET quản lý (Managed client) về việc sử dụng RCW để tương tác với thành phần COM kinh điển.



Hình 5.3-1 – Tổng quát của mô hình RCW.



RWC có trách nhiệm bảo vệ người phát triển từ những tác vụ như tham chiếu bộ đếm (reference counting), tự động cấp phát hoặc giải phóng bộ nhớ.

Trách nhiệm của RCW bao gồm những công việc sau:

- Bảo đảm việc nhận diện đối tượng
- Duy trì thời gian sống của đối tượng
- Giao tiếp trung gian giữa đối tượng với COM và .NET
- Gọi các phương thức Marshaling.
- Cho phép gọi các giao tiếp COM hiện có

Mỗi đối tượng COM chỉ có một lớp bao bọc RCW. Điều này cho phép RCW bảo đảm việc nhận diện đối tượng COM bằng cách so sánh đối tượng với giao tiếp IUnknown. RCW được dùng để che giấu các lời gọi đến QueryInterface của COM. Khi ép kiểu (casting) cho một giao tiếp, RCW sẽ kiểm tra giao tiếp ẩn của kiểu yêu cầu và trả về nó nếu tìm thấy; ngược lại, khi một QI được triệu gọi, RCW sẽ triệu gọi QI trên đối tượng COM nằm dưới, giao tiếp trả về được đưa vào nơi vùng đệm trữ (cache) duy trì bởi RCW. Nếu giao tiếp yêu cầu không tìm thấy, một ngoại lệ chuẩn InvalidCastException sẽ được RCW ném ra.

Một thể hiện đơn của RCW cần thiết cho việc quản lý thời gian sống thích hợp của đối tượng COM nằm dưới. RCW xử lý những lời gọi cần thiết đến 2 phương thức AddRef() và Release(), và bảo đảm tham chiếu bộ đếm (reference counting) thích hợp dành cho đối tượng COM. Như với bất kỳ các đối tượng được quản lý Managed Object khác, RCW cũng có cơ chế dọn rác. Khi phương thức finalize của RCW được triệu gọi bởi GC, thì RCW sẽ gọi phương thức Release() trên tất cả những giao tiếp lưu trữ dùng cho đối tượng COM nằm dưới. Điều này làm giúp cho người phát triển thoải mái, không phải lo lắng về 2 phương thức AddRef() và Release(), bởi vì RCW chịu trách nhiệm toàn bộ về tham chiếu bộ đếm (reference counting).

Dù là RCW quản lý thời gian sống của đối tượng COM nằm dưới, đôi khi nó cũng cần ép buộc giải phóng đối tượng COM. Ví dụ như những đối tượng COM yêu cầu những tài nguyên quý giá như kết nối mạng (Socket connection), kết nối cơ sở dữ liệu (database connection), hoặc bộ nhớ dùng chung (shared memory). Khi GC không chủ động quyết định khi nào chấm dứt và hủy đối tượng bên trong .NET, RCW sẽ có trách nhiệm giải phóng đối tượng COM nằm dưới.

RCW cung cấp một số cơ chế cho phép những chương trình quản lý bởi .NET truy xuất vào các giao tiếp không được quản lý của COM. Như thế, RCW xuất hiện để thực thi tất cả những giao tiếp được cung cấp bởi đối tượng COM. Quả thực, điều này cho phép phương thức giao tiếp triệu gọi mà không cần phải ép kiểu tường minh cho giao tiếp đó. Xem xét ví dụ sau:

Đưa vào một đối tượng COM mô tả trong ví dụ 5.3-1, trình ứng dụng do .NET quản lý (Managed client) có thể triệu gọi bất kỳ phương thức giao tiếp nà

mà không phải ép kiểu tường minh cho giao tiếp đó. Ví dụ 5.3-2 giải thích cách tương tác với đối tượng COM.

Ví dụ 5.3-1 : Định nghĩa thành phần COM

```

1: [uuid(...)]
2: interface IFoo : IDispatch {
3:     [id(...)] HRESULT FooMethod(...);
4: }
5:
6: [uuid(...)]
7: interface IBar : IDispatch {
8:     [id(...)] HRESULT BarMethod(...);
9: }
10:
11: [uuid(...)]
12: coclass FooBar {
13:     [default] interface IFoo;
14:     interface IBar;
15: }
```

Ví dụ 5.3-2 : Managed Wrapper Code

```

1: FooBar fb = new FooBar();
2: //Invoke IFoo.FooMethod
3: //No need for explicit IFoo cast
4: fb.FooMethod;
5:
6: //Invoke IBar.BarMethod
7: //No need for explicit IBar cast
8: fb.BarMethod();
```

2.1. Trình TlbImp.exe

Trình TlbImp.exe được dùng để sinh ra lớp vỏ bọc cho đối tượng COM. Cú pháp cơ bản sử dụng TlbImp.exe như sau:

TlbImp.exe TypeLibName [/out : <Tên tập tin>]

Để hiểu tốt hơn về quan hệ giữa COM và .NET RCW, bạn hãy xem ví dụ về đối tượng COM sau:



Ví dụ 5.3-3 : SimpleObject.h

```
1: // SimpleObject.h : Declaration of the CSimpleObject
2:
3: #pragma once
4: #include "resource.h"      // main symbols
5:
6:
7: // ISimpleObject
8: [
9:     object,
10:    uuid("6D854C55-4549-44FB-9CDF-6079F56B232E"),
11:    dual, helpstring("ISimpleObject Interface"),
12:    pointer_default(unique)
13: ]
14: __interface ISimpleObject : IDispatch
15: {
16:
17:
18:     [id(1), helpstring("method SayHello")]
19:     HRESULT SayHello([in] BSTR Name, [out, retval] BSTR*
20:                         Message);
21:
22:
23: // CSimpleObject
24:
25: [
26:     coclass,
27:     threading("apartment"),
28:     aggregatable("never"),
29:     vi_progid("SimpleATL.SimpleObject"),
30:     progid("SimpleATL.SimpleObject.1"),
31:     version(1.0),
32:     uid("D82A38B8-5392-4D3D-ADEC-516C18E6A092"),
33:     helpstring("SimpleObject Class")
34: ]
```

```

35: class ATL_NO_VTABLE CSimpleObject :
36: public ISimpleObject
37: {
38: public:
39:     CSimpleObject()
40:     {
41:     }
42:
43:
44: DECLARE_PROTECT_FINAL_CONSTRUCT()
45:
46:     HRESULT FinalConstruct()
47:     {
48:         return S_OK;
49:     }
50:
51:     void FinalRelease()
52:     {
53:     }
54:
55: public:
56:
57:
58: STDMETHOD(SayHello)(BSTR Name, BSTR* Message);
59: };
60:
```

SimpleObject.h định nghĩa một giao tiếp đơn ISimpleObject và một lớp coclass CSimpleObject, CSimpleObject cài đặt giao tiếp ISimpleObject. ISimpleObject định nghĩa một phương thức đơn SayHello, phương thức này nhận tham số truyền vào kiểu BSTR và trả ra kiểu BSTR. RCW sẽ kiểm soát kiểu BSTR và chuyển đổi nó thành kiểu chuỗi CLR của .NET. Ví dụ 5.3-4 dưới đây cho thấy cách cài đặt phương thức SayHello.

Ví dụ 5.3-4 : SimpleObject.cpp

```

1: // SimpleObject.cpp : Implementation of
   CSimpleObject
2: #include "stdafx.h"
3: #include "SimpleObject.h"
4:
```

```

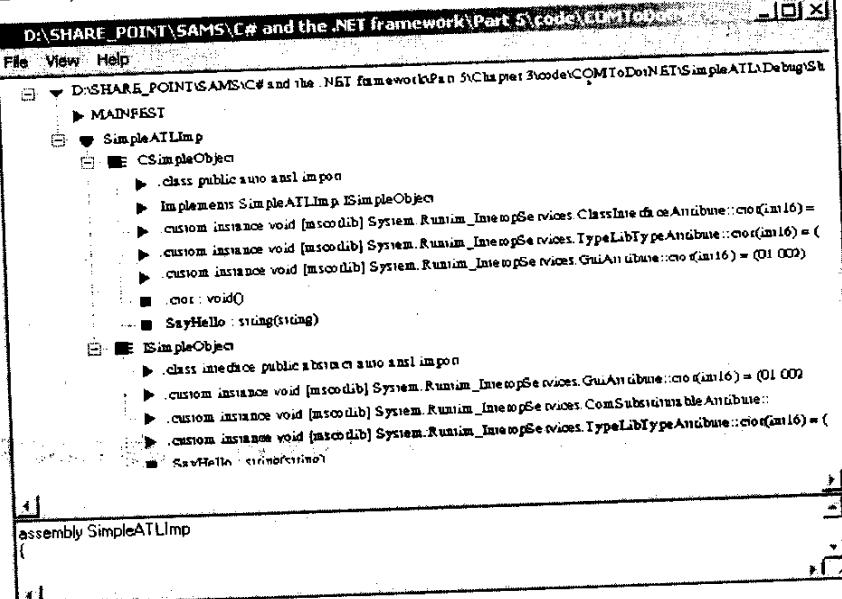
5: // CSimpleObject
6:
7: STDMETHODIMP CSimpleObject::SayHello(BSTR Name,
   BSTR* Message)
8: {
9: // TODO: Add your implementation code here
10: CComBSTR Msg( "Hello " ); Msg += Name;
11: *Message = ::SysAllocString( Msg.m_str );
12:
13: return S_OK;
14: }
15:

```

Phương thức SayHello đơn thuần là cài đặt phép kết nối chuỗi. Mẫu ATL do VS.NET tự động tạo ra. Để tạo RCW cho SimpleATL.dll, bạn sử dụng dòng lệnh sau :

```
TlbImp SimpleATL.dll /out:SimpleATLImp.dll
```

Lệnh trên sẽ tạo ra lớp RCW cần thiết cho việc sử dụng bởi các chương trình được quản lý của .NET. Bạn có thể kiểm tra lớp bọc này bằng trình ILDASM; như được mô tả trong hình 5.3-2.



Hình 5.3-2 – Managed Wrapper trong ILDASM.

2.2. Ràng buộc sớm (Early Binding)

Với RCW được tạo trên đây, việc sử dụng đối tượng COM là một vấn đề đơn giản của việc thêm vào một tham chiếu (reference) thích hợp tới dự án (project). Khi một reference được thêm vào dự án, thì đối tượng sẵn sàng cho cơ chế ràng buộc sớm. Ràng buộc sớm yêu cầu kiểm tra về kiểu khi biên dịch. Các đối tượng COM hợp lệ đối với cơ chế ràng buộc sớm đều phải hỗ trợ giao tiếp kép (dual interface). Dưới đây là ví dụ minh họa.

Ví dụ 5.3-5 : Ràng buộc sớm

```

1:  namespace EarlyBinding
2:  {
3:      using System;
4:
5:      class Early
6:      {
7:          static void Main(string[] args)
8:          {
9:              SimpleATLImp.CSimpleObject o =
new SimpleATLImp.CSimpleObject();
10:             Console.WriteLine(o.SayHello("Richard"));
11:         }
12:     }
13: }
14:
```

Với RCW được sinh ra, sử dụng đối tượng COM không đòi hỏi mã chương trình bổ sung cho tới khi trình ứng dụng của .NET quan tâm đến nó.

2.3. Ràng buộc muộn (Late binding)

Ràng buộc muộn liên quan đến việc sử dụng giao tiếp IDispath của các đối tượng COM, giao tiếp này được dùng để phát hiện những dịch vụ trong thời gian chạy do COM cung cấp. Mục đích của giao tiếp IDispath là cung cấp một giao tiếp phù hợp cho việc sử dụng với những scripting client (Scripting client là VBScript hoặc JScript). Scripting client không được phép sử dụng giao tiếp thô (raw interface) và yêu cầu ràng buộc muộn của IDispath.

.NET cũng cung cấp khả năng sử dụng ràng buộc muộn thông qua giao tiếp IDispath của đối tượng COM (ví dụ 5.3-6). Làm như vậy thì không chú ý đến kiểu kiểm tra thời gian dịch. Người phát triển phải sử dụng Reflection API để truy xuất những phương thức thể hiện và những thuộc tính hiện có của đối tượng COM.

Ví dụ 5.3-6 : Ràng buộc muộn

```
1:      namespace LateBinding
2:  {
3:      using System;
4:      using System.Reflection;
5:      using System.Runtime.InteropServices;
6:
7:      class Late
8:  {
9:      static void Main(string[] args)
10:     {
11:         try
12:         {
13:             Type SimpleObjectType =
Type.GetTypeFromProgID("SimpleATL.SimpleObject");
14:             object SimpleObjectInstance =
Activator.CreateInstance( SimpleObjectType );
15:
16:             Console.WriteLine("SimpleObjectType =
{0}",
SimpleObjectType.ToString( ) );
17:             Console.WriteLine("SimpleObjectInstance
Type = {0}",
SimpleObjectInstance.GetType( ).ToString( )
);
18:
19:
20: //Invoke the SayHello Instance Method
21:         string Message =
(string)SimpleObjectType.InvokeMember("SayHello",
22:             BindingFlags.Default |
BindingFlags.InvokeMethod,
23:             null,
24:             SimpleObjectInstance,
25:             new object[] { "Richard" } );
26: //Did it work?
27:         Console.WriteLine(Message);
28:     }
29:     catch( COMException e )
```

```
void OnSomeEvent([in] BSTR Message);  
12: };  
13:
```

Phương thức sự kiện OnSomeEvent, dòng 11, sẽ chuyển thành *cấp sự kiện/bộ chuyển giao sự kiện* (event/delegate) khi RCW được tạo ra. Bộ chuyển giao sự kiện có tên như sau:

```
_ISourceObjectEvent_OnSomeEventEventHandler
```

Bộ chuyển giao sự kiện này sau đó có thể được tạo ra và gắn vào sự kiện OnSomeEvent như ví dụ 5.3-9 sau:

Ví dụ 5.3-9 : Mã chương trình của SimpleSink

```
1:  namespace CSharpSink  
2:  {  
3:      using System;  
4:      using ATLSourceImp;  
5:  
6:      /// <summary>  
7:      /// Summary description for Class1.  
8:      /// </summary>  
9:      class SimpleSink  
10:     {  
11:         static void Main(string[] args)  
12:         {  
13:  
14:             CSourceObject source = new CSourceObject();  
15:             source.OnSomeEvent +=  
16:                 new  
_ISourceObjectEvents_OnSomeEventEventHandler(OnSomeE  
vent);  
17:  
18:             source.MakeEventFire("Hello");  
19:         }  
20:  
21:         public static void OnSomeEvent(string Message)  
22:         {  
23:             Console.WriteLine("Have Event: {0}", Message);  
24:         }  
25:     }
```

26: }

27:

Sự kiện ràng buộc không có gì khác so với cặp *sự kiện/bộ chuyển giao sự kiện* của .NET. Lớp CSimpleObject sẽ triệu gọi OnSomeEvent bất cứ khi nào phương thức của đối tượng bị triệu gọi.

2.5. Những vấn đề về tiêu trình (Thread)

COM đưa ra quá thừa thãi của những kiểu tiêu trình (threading). Các thành phần COM có thể là Apartment, Both, Free, MTA, Single, hoặc STA – quá nhiều sự chọn lựa, vì vậy cũng có thể có quá nhiều bất lợi. Nếu bạn không bao giờ đi sâu vào những kiểu threading của COM và không bao giờ hiểu những Apartment của COM, thì coi như đó là may mắn của bạn. Tuy nhiên, khi bạn cần làm việc với những đối tượng COM kinh điển, đây lại là điều quan trọng để hiểu mối quan hệ mật thiết của những kiểu threading khác nhau.

Các trình ứng dụng được .NET quản lý (Managed client) tạo ra những thread bên trong STA. Khi sử dụng đối tượng COM cũng là STA, không có sự bất lợi khi những phương thức đang triệu gọi trên những giao tiếp khác nhau. Tuy nhiên, Nếu đối tượng COM là MTA, thì đó là lúc đối tượng COM đứng ra chịu mọi tổn phí. May mắn, Managed client có thể thay đổi mô hình threading hiện hành, tạo ra đối tượng COM và tiết kiệm được một số bước.

Để thay đổi mô hình threading, bạn truy xuất CurrentThread và đặt trạng thái cho ApartmentState khi cần. Chú ý rằng ApartmentState chỉ có thể đặt một lần. Ví dụ 5.3-10 dưới đây sẽ thay đổi ApartmentState của thread hiện hành bằng với MTA để tương ứng với MTAObject đang được truy xuất.

Ví dụ 5.3-10 : Thay đổi Apartment của thread

```

1: Thread.Current.ApartmentState = ApartmentState.MTA;
2: //Create MTA Object
MTAObject o = new MTAObject();
3: o.Foo();

```

Sử dụng mô hình thread mới thay đổi tổng chi phí của các cuộc gọi COM liên tiếp thông qua những apartment khác nhau theo hướng giảm. Mã chương trình bên phần Server rất cần tốc độ, vì vậy việc để giành tổng chi phí là một việc hết sức quan trọng, và việc đổi tương COM sử dụng mô hình thread cho phép tối đa lời gọi đối tượng rất hiệu quả.

2.6. So sánh kiểm .NET và COM

Việc chuyển các kiểu từ COM sang các kiểu của .NET hoàn toàn dễ dàng. Bảng 5.3-1 hỗ trợ chuyển đổi những kiểu của COM thông dụng sang các kiểu trong .NET tương ứng.



26: }

27:

Sự kiện ràng buộc không có gì khác so với cặp *sự kiện/bộ chuyển giao sự kiện* của .NET. Lớp CSimpleObject sẽ triệu gọi OnSomeEvent bất cứ khi nào phương thức của đối tượng bị triệu gọi.

2.5. Những vấn đề về tiêu trình (Thread)

COM đưa ra quá thừa thãi của những kiểu tiêu trình (threading). Các thành phần COM có thể là Apartment, Both, Free, MTA, Single, hoặc STA – quá nhiều sự chọn lựa, vì vậy cũng có thể có quá nhiều bất lợi. Nếu bạn không bao giờ đi sâu vào những kiểu threading của COM và không bao giờ hiểu những Apartment của COM, thì coi như đó là may mắn của bạn. Tuy nhiên, khi bạn cần làm việc với những đối tượng COM kinh điển, đây lại là điều quan trọng để hiểu mối quan hệ mật thiết của những kiểu threading khác nhau.

Các trình ứng dụng được .NET quản lý (Managed client) tạo ra những thread bên trong STA. Khi sử dụng đối tượng COM cũng là STA, không có sự bất lợi khi những phương thức đang triệu gọi trên những giao tiếp khác nhau. Tuy nhiên, Nếu đối tượng COM là MTA, thì đó là lúc đối tượng COM đứng ra chịu mọi tổn phí. May mắn, Managed client có thể thay đổi mô hình threading hiện hành, tạo ra đối tượng COM và tiết kiệm được một số bước.

Để thay đổi mô hình threading, bạn truy xuất CurrentThread và đặt trạng thái cho ApartmentState khi cần. Chú ý rằng ApartmentState chỉ có thể đặt một lần. Ví dụ 5.3-10 dưới đây sẽ thay đổi ApartmentState của thread hiện hành bằng với MTA để tương ứng với MTAObject đang được truy xuất.

Ví dụ 5.3-10 : Thay đổi Apartment của thread

```
1: Thread.Current.ApartmentState = ApartmentState.MTA;
2: //Create MTA Object
MTAObject o = new MTAObject();
3: o.Foo();
```

Sử dụng mô hình thread mới thay đổi tổng chi phí của các cuộc gọi COM liên tiếp thông qua những apartment khác nhau theo hướng giảm. Mã chương trình bên phần Server rất cần tốc độ, vì vậy việc để giành tổng chi phí là một việc hết sức quan trọng, và việc đổi tượng COM sử dụng mô hình thread cho phép tối đa lời gọi đối tượng rất hiệu quả.

2.6. So sánh kiểm .NET và COM

Việc chuyển các kiểu từ COM sang các kiểu của .NET hoàn toàn dễ dàng. Bảng 5.3-1 hỗ trợ chuyển đổi những kiểu của COM thông dụng sang các kiểu trong .NET tương ứng.

Bảng 5.3-1 : Ánh xạ kiểu của COM và .NET

Kiểu COM	Kiểu .NET
BSTR	string
VARIANT	object
SAFEARRAY	array[]
I1	sbyte
I2	short
I4	int
I8	long
UI1	byte
UI2	ushort
CHAR	char
UI4	uint
R4	float
R8	double
IUnknown**	object
IDispatch**	object
I<SomeInterface>	I<SomeInterface>

Mặc dù bảng 5.3-1 chưa hoàn chỉnh, nó đã chuyển đổi các kiểu cơ bản và đảo ngược của chúng. Bạn tham khảo thêm MSDN để có danh sách hoàn chỉnh về các kiểu ngịch đảo khác nhau.

3. XEM CÁC THÀNH PHẦN .NET NHƯ LÀ CÁC ĐỐI TƯỢNG COM

Trong khi các Managed client có thể dùng nhiều lớp đối tượng COM, khả năng để tạo lớp bao bọc để COM có thể gọi được các đối tượng .NET cũng cần được xem xét đến - COM Callable Wrapper (CCW). Những cơ hội của việc tạo các thành phần .NET và thể hiện chúng như là các đối tượng COM có thể không theo kịp phát triển nói chung. Vì vậy, ở đây chúng ta chỉ xem qua các đăng ký 1 thành phần .NET như là đối tượng COM và các thủ tục liên quan để làm chuyện đó.

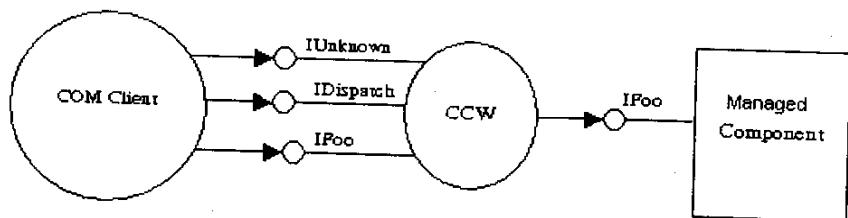


3.1. Trình RegAsm.exe

.NET SDK cung cấp một bộ công cụ RegAsm.exe được dùng để tạo các phần tử cần thiết cho hệ thống Registry. RegAsm cũng có thể tạo 1 thư viện kiểu, tập tin .tib dùng để mô tả các kiểu trong gói kết hợp assembly. Các phần tử Registry giống tương tự như bên đối tượng COM và cho phép tải 1 gói được .NET quản lý (Managed Assembly). Ngoài việc dùng RegAsm.exe để đăng ký Managed Assembly, nó còn dùng để thêm gói assembly này vào GAC, hoặc Global Assembly Cache. Bạn sử dụng chương trình gacutil.exe để thực hiện công việc này.

3.2. COM Callable Wrapper

Phản trái ngược với RCW là COM Callable Wrapper (CCW). Giống như RCW, CCW xử lý những vấn đề tương tự nhau, như duy trì nhận dạng đối tượng, quản lý chu kỳ sống, v.v... CCW cũng chịu trách nhiệm cung cấp các giao tiếp, như là IDispatch và IUnknown, bởi vì các giao tiếp này không thuộc Managed component. Giao tiếp IDispatch cho phép các client dạng kịch bản (scripting client) dùng những Managed component, trong khi chúng có thể dùng tất cả những đối tượng COM IDispatch khác. Hình 5.3-3 mô tả cơ chế làm việc và triệu gọi của CCW.



Hình 5.3-8 – Cơ chế làm việc của CCW.

CCW hoạt động như là một nhà môi giới trung gian (proxy) giữa những đối tượng được quản lý bởi .NET và các đối tượng COM thông thường.

4. CÁC VẤN ĐỀ KHI DÙNG CHUNG .NET VÀ COM VỚI NHAU

COM có những qui định nghiêm ngặt quản lý việc cài đặt đối tượng. Để COM sử dụng được các phương thức của đối tượng .NET một cách hiệu quả, chúng ta cần phải tuân theo các quy luật của COM. Ví dụ hãy xem xét lớp sau:

```

1: class Foo{
2:   public Foo(string s){...}
3:   public static void Bar(){...}
4: }
  
```

Việc chuyển lớp Foo thành COM sẽ không thành công vì 2 lý do sau:

- COM không hỗ trợ những phương thức khởi tạo (constructor) có thông số.
- COM không hỗ trợ xây dựng những phương thức phi thể hiện (no instance) kiểu phương thức static.

Lớp Foo được xem là không thân thiện với mô hình COM. Nên bỏ các hàm khởi tạo có thông số và sau đó cung cấp vài hàm khởi tạo thông thường. Chỉ tinh này phải được tuân theo để truyền các đối tượng Managed qua COM có hiệu quả.

COM hỗ trợ xây dựng kiểu giá trị, .NET cũng làm điều này. Tuy nhiên kiểu giá trị trong COM không có các thể hiện của phương thức như trong .NET ngay cả trong C++.

```
1: Public struct Point{  
2:     int x;  
3:     int y;  
4:     void SetXY(int x, int y){...}  
5: }
```

Khi kiểu giá trị Point được truyền cho đối tượng COM, phương thức thể hiện SetXY sẽ bị bỏ, bởi vì các biến thành viên x và y là private, kiểu giá trị Point trả về nên không có giá trị.

Trước khi bắt đầu vào 1 dự án để tạo các thành phần .NET với ý định truyền chúng qua các COM client, bạn phải đảm bảo việc cài đặt các đối tượng tuân theo các qui tắc của COM.

5. KẾT CHƯƠNG

Nhiều vấn đề về COM đã được bàn đến trong chương này. Ví dụ như các vấn đề về điều phối COM trong một Managed Client, mô hình thread COM mới, gọi đối tượng COM từ .NET và ngược lại. Mô hình COM cổ điển sẽ tồn tại trong nhiều năm nữa, vì vậy hầu hết các người phát triển đều yêu cầu hầu phải nắm vững điều cơ bản về cách COM tương tác với mã tự quản (Managed code) .NET. Với những dự án trong tương lai, bạn nên từ bỏ dần mô hình COM. Thay vào đó hãy tập trung tối đa vận dụng các tiện nghi được cung cấp bởi C# và tương ứng .NET.

Chương 5.4

CÁC TIỂU TRÌNH (THREADS)

Các vấn đề chính sẽ được đề cập đến:

- ✓ *Đa tiêu trình*
- ✓ *Đồng bộ hóa tiêu trình (Thread Synchronization)*
- ✓ *Mở rộng các tiêu trình .NET*

1. ĐA TIỂU TRÌNH

Mỗi ứng dụng sẽ có một tiêu trình thực thi chính. Tiêu trình chính này sẽ thực thi các mã ứng dụng theo một thứ tự cụ thể dựa trên các tương tác với người dùng hay trả lời các thông điệp (message) của Windows. Chuyện gì sẽ xảy ra khi nhiều hơn một tác vụ hoặc đoạn mã cần được thực thi. Thứ tự thực hiện phụ thuộc vào việc các ứng dụng đơn tiêu trình (single-thread application) ấy đã được xây dựng như thế nào, thế nhưng tại một thời điểm vẫn chỉ có một tác vụ cụ thể được thực thi.

Trong môi trường Windows, các ứng dụng có thể tạo sinh (spawn) nhiều tiêu trình và điều khiển việc giao tiếp giữa các tiêu trình ấy với nhau. Một ứng dụng có thể quản lý nhiều tác vụ (task) cùng lúc. Hãy thử nghĩ về trình duyệt Web của bạn. Khi download một tập tin, chúng sẽ tạo sinh ra một tiêu trình quản lý tác vụ download trong khi cùng lúc ấy bạn vẫn có thể duyệt Web. Nếu không có khả năng tạo sinh và điều khiển nhiều tiêu trình trong ứng dụng, việc download tập tin sẽ dùng hết tiêu trình chính của ứng dụng. Trong những trường hợp đó, trình duyệt Web sẽ không thể làm gì cho đến khi việc download tập tin hoàn tất.

Việc tạo ra các ứng dụng đa tiêu trình có thể cải thiện năng suất của một ứng dụng, nhưng việc có thêm các tiêu trình đã làm cho mức độ phức tạp của việc thiết kế, phát triển, và gỡ rối ứng dụng tăng lên. Thêm vào đó, không phải năng suất chỉ có nhờ vào việc thêm vào một tiêu trình mà còn nhờ vào việc sử dụng kiến trúc của ứng dụng.

1.1. Tiêu trình ứng dụng (Application Thread)

Kiến trúc .NET tạo ra một tiêu trình đơn trong vùng dành riêng để thực thi ứng dụng. Phương thức Main của một lớp biểu diễn một lối vào cho chương trình và cũng là lối vào cho tiêu trình. Mỗi tiêu trình được tạo ra từ một tiêu trình ứng dụng sẽ được gọi là tiêu trình con (child thread) của tiêu trình cha (parent thread). Khi ứng dụng cha hết hoạt động, tất cả các tiêu trình con cũng sẽ chấm dứt. Điều đặc biệt cần lưu ý là phải kiểm tra rằng tất cả các tiêu trình con được huỷ đúng cách và các tài nguyên được giải phóng trước khi huỷ tiến trình cha mẹ.



1.2. Tiểu trình làm việc hay tiểu trình thợ (Worker Thread)

Dạng thông dụng nhất của tiểu trình là tiểu trình thợ. Một tiểu trình t được dùng để làm một số kiểu thao tác nền (background), những thao tác có t chiếm giữ tiểu trình ứng dụng. Những ví dụ của các loại thao tác nền này bao g cả việc download tập tin, in nén, và tự động lưu. Khi được sử dụng hợp lý, ti trình thợ cho phép tiểu trình chính của ứng dụng trả lời các thông điệp của ứng dụng và khiến ứng dụng hoạt động suôn sẻ hơn.

1.3. Tạo một tiểu trình thợ

Giống như các tiểu trình ứng dụng, một tiểu trình thợ s̄e thực thi một đ m chương trình trong ứng dụng của bạn. Ví dụ 5.4-1 cho thấy cách xây dựng n cấu trúc của tiểu trình thợ.

Ví dụ 5.4-1. Tiểu trình thợ

```
1: using System;
2: using System.Threading;
3:
4: namespace BasicWorkerThread
5: {
6:
7:     class Class1
8:     {
9:
10:         static void WorkerThread( )
11:         {
12:             Console.WriteLine("Hello from WorkerThread");
13:         }
14:
15:         static void Main(string[] args) {
16:
17:             //Create the Worker Thread
18:             System.Threading.Thread WT = new Thread( new
19:             ThreadStart( WorkerThread ) );
20:
21:             //Start the Thread
22:             WT.Start( );
23:
24:             //end the application
25:             Console.WriteLine("Press enter to exit");
```

```

25:     Console.ReadLine();
26:
27: }
28: }
29: }
30:

```

Điều đầu tiên phải lưu ý trong ví dụ 5.4-1 là việc sử dụng **Using** để “nhập khẩu” thư viện hay không gian tên (namespace) System.Threading. Tất cả lớp liên quan đến tiểu trình đều ở trong namespace này. Để tạo ra một tiểu trình, cần phải có một phương thức làm việc như một đại diện chuyển giao (delegate) tiểu trình. Điều đó đúng: tiểu trình .NET dùng đại diện để định nghĩa phương thức mà tiểu trình mới sẽ dùng. Trong ví dụ trên, phương thức static WorkerThread đã đóng vai trò đại diện, định nghĩa cho tiểu trình. Ở dòng 18, một đối tượng Thread được tạo ra. Nhớ rằng, constructor cho một Thread sẽ là một thông số cho đại diện ThreadStart. Như đã nói trước, đại diện ThreadStart định nghĩa các phương thức mà tiểu trình mới được tạo thành sẽ thi hành. Đại diện của tiểu trình chỉ có duy nhất một chữ ký đơn (single signature) chứa một giá trị trả về void và danh sách tham số rỗng.

Nhiều người trong số các bạn quen thuộc với các tiểu trình Win32 truyền thống, hoặc các tiểu trình POSIX, có thể sẽ thắc mắc về việc tại sao không có tham số cho tiểu trình. Trong Win32, một hàm tiểu trình dùng một LPVOID (con trỏ kiểu LONG VOID) làm tham số. Điều này cho phép bất cứ tham số nào được chuyển đến phương thức static của một đối tượng đều có thể được dùng cho tiểu trình. Không như các tiểu trình Win32, các tiểu trình .NET có thể dựa vào thể hiện (instance) của đối tượng. Điều này có nghĩa là một phương thức không phải static cũng có thể được dùng cho tiểu trình. Khi chúng được sử dụng, phương thức và tiểu trình ấy sẽ có truy nhập đến tất cả các trường (field) và các thuộc tính của đối tượng thể hiện. Chúng cho thấy một khác biệt nhỏ so với các tiểu trình kiểu Win32. Ví dụ 5.4-2 dưới đây sẽ cho thấy việc tiểu trình truy xuất dữ liệu trong một lớp.

Ví dụ 5.4-2 Thể hiện của Tiểu trình

```

1: using System;
2: using System.Threading;
3:
4: namespace InstanceThread
5: {
6:     class Bank
7:     {
8:
9:         private string bankName = "First Bank";
10:

```

```

11:     public void WorkerThread( )
12:     {
13:         //Access instance member bankName
14:         Console.WriteLine("Bank Name = {0}",
this.bankName );
15:     }
16:
17:     public void Run( )
18:     {
19:         Thread WT = new Thread( new ThreadStart(
this.WorkerThread ) );
20:         WT.Start( );
21:     }
22:
23:
24:     static void Main(string[] args)
25:     {
26:         Bank b = new Bank( );
27:         b.Run( );
28:     }
29: }
30: }
```

Đoạn mã trong ví dụ 5.4-2 không khác nhiều với ví dụ 5.4-1. Trong ví dụ này, đại diện tiêu trình WorkerThread có một truy xuất (ở cấp độ thể hiện) trường private bankName. Những tiêu trình kiểu đối tượng như thế là một bất đồng từ các tiêu trình kiểu Win32 truyền thống. Nền .NET cung cấp một cách cận gân hơn với các quy ước về tiêu trình kiểu đối tượng so với trước đây, nhiên, nó không phải là hoàn hảo lắm. Nó không có sẵn một lớp cơ sở để các dân xuất từ nó có thể làm việc như các tiêu trình. Trong kiến trúc .NET, lớp System.Threading.Thread đã được phong kín lại, và như vậy, nó không thể phục vụ như một lớp cơ sở.

1.4. Thuộc tính ThreadStatic

Thuộc tính ThreadStatic cung cấp một cơ chế để tạo ra các trường static tiêu trình cần tới. Mỗi tiêu trình được tạo ra sẽ có bản sao chép riêng cho trường đã được chọn với thuộc tính ThreadStatic sẽ là 0 mà không quan tâm giá trị đã được khởi tạo trước đó. Các trường static sẽ chỉ giữ lại các giá trị được khởi tạo cho các tiêu trình chính của ứng dụng. Ví dụ 5.4-3 minh họa dùng thuộc tính ThreadStatic và các tác động lên các trường static.

Ví dụ 5.4-3 Thuộc tính ThreadStatic

```
1: using System;
2: using System.Threading;
3:
4:
5:
6: public class Task {
7:
8:     [ThreadStatic] static int m_nId = 10;
9:
10:    public string m_strThreadName;
11:
12:    public void Run( string ThreadName ) {
13:        this.m_strThreadName = ThreadName;
14:        Thread T = new Thread( new ThreadStart( threadProc ) );
15:        T.Start();
16:    }
17:
18:    public void threadProc( ) {
19:
20:        Console.WriteLine( "Thread {0} is running",
m_strThreadName );
21:
22:        //loop and increment the m_nId static field
23:        for( int i = 0; i < 10; i++ )
24:            Console.WriteLine("Thread {0} : m_nId = {1}",
this.m_strThreadName, m_nId++ );
25:
26:    }
27: }
28:
29:
30:public class ThreadApp {
31:
32:    public static void Main( ) {
33:
34:        //Create 2 worker Tasks
35:        Task t1 = new Task();
36:        Task t2 = new Task();
37:
38:        t1.Run( "Worker 1" );
39:        t2.Run( "Worker 2" );
40:
41:        //Execute on the Main thread
42:        Task t3 = new Task();
43:        t3.m_strThreadName = "Main Thread";
44:        t3.threadProc();
45:    }
46: }
```

Kết xuất của ví dụ 5.4-3 phụ thuộc vào thứ tự lập lịch thực thi các tiêu trì và bắt cứ tiêu trình nào cũng sẽ có thể nhảy vào chiếm chỗ. Tuy nhiên, bạn có thể thấy mỗi tiêu trình, khi chạy, đều có thành viên static m_nId với giá trị được khởi tạo là 0. Khi thành viên threadProc được thực thi từ ngữ cảnh của tiêu trình chỉ của ứng dụng (dòng 44), trường static giữ lại giá trị 10 của nó. Thuộc tính ThreadStatic cho phép mỗi tiêu trình giữ lấy giá trị dữ liệu thành phần của riêng nó. Nếu không có đủ hiệu quả như mong muốn, không nên dùng thuộc tính ThreadStatic, trong trường hợp này, tất cả thành viên static cũng đều sẵn sàng cho các tiêu trình và các giá trị của nó được chia sẻ lẫn nhau.

1.5. Liên kết – Đem các tiêu trình lại bên nhau

Thường thì chúng ta cần sinh ra một hay nhiều tiêu trình và có một tiêu trình cha chờ cho các tiêu trình con kết thúc trước khi tiếp tục thực thi. Việc này có thể hoàn thành một cách dễ dàng nhờ vào việc sử dụng phương thức Join để hiện tiêu trình. Khi phương thức Join được gọi, tiêu trình gọi sẽ tạm ngưng cho đến khi tiêu trình thợ hoàn tất. Sau khi tiêu trình thợ đã kết thúc, việc thi sẽ được tiếp tục với tiêu trình gọi (xem ví dụ 5.4-4).

Ví dụ 5.4-4 Sử dụng Join

```

1: using System;
2: using System.Threading;
3:
4:
5:
6: public class JoinTest {
7:
8:
9:     public static void ThreadProc() {
10:
11:     Console.WriteLine("Thread Active");
12:
13:     Thread.Sleep( 30 * 1000 );
14:
15:     Console.WriteLine("Thread Exiting");
16: }
17:
18:
19:     public static void Main() {
20:         Thread T = new Thread( new ThreadStart( ThreadProc ) );
21:         T.Start();
22:         T.Join();
23:         Console.WriteLine("After Join");
24:     }
25: }
```

2. ĐỒNG BỘ TIỂU TRÌNH (THREAD SYNCRONIZATION)

Như đã mô tả ở trên, các tiểu trình có thể được lập lịch để chạy bất cứ lúc nào và theo bất kỳ thứ tự nào. Thông thường chúng ta cần điều phối các tiểu trình để chia sẻ tài nguyên giữa chúng. Một điều phải lưu ý: tất cả các ứng dụng đa tiểu trình đều làm việc tốt ở chế độ gỡ lỗi (debug), nó chỉ thực sự bắt đầu gây rối rắm khi chạy ở chế độ phát hành (release). Việc gỡ rối các ứng dụng đa tiểu trình (multi-threaded) cần nhiều thời gian và kinh nghiệm; không có điều gì có thể làm cho cuộc sống dễ dàng hơn khi bạn phải đổi đầu với một con quái vật nhiều đầu kiểu như tiểu trình và ứng dụng. Vì những lý do không rõ hay không thể giải thích, việc lập lịch cho tiểu trình ở chế độ gỡ rối không giống chút gì với việc lập lịch tiểu trình ở chế độ ứng dụng hoàn chỉnh.

2.1. Từ khoá lock

C# dùng thêm từ khoá lock. Lock được sử dụng cho việc đồng bộ hoá các truy xuất cá thể hiện lẩn các trường static của một đối tượng. Để đồng bộ hoá truy xuất đến một trường ở cấp thể hiện, ứng dụng sẽ dùng lock (this) với this chỉ đến đối tượng hiện tại. Để đồng bộ hoá truy xuất đến các trường static, từ khoá static có thể dùng bên cạnh từ khoá typeof như sau:

```
lock ( typeof ( class ) )
```

Để minh họa cách dùng từ khoá lock và các ảnh hưởng có thể gặp khi không dùng nó, ví dụ 5.4-5 sẽ cho thấy một cách đơn giản để truy xuất mảng ở cấp thể hiện. Hai tiểu trình sẽ được tạo lập và mỗi tiểu trình đều cố gắng đặt các giá trị băm của nó vào một ô bên trong mảng. Khi câu lệnh lock đang hoạt động, chương trình không gây lỗi. Thế nhưng, nếu loại bỏ các câu lệnh lock, bạn sẽ gặp lỗi.

Ví dụ 5.4-5 Từ khoá lock

```

1: using System;
2: using System.Threading;
3:
4:
5: public class ThreadLockTest {
6:
7:     private int[] m_Array = new int[10];
8:     private int    m_CurrentIndex = 0;
9:
10:
11:    public void ThreadProc( ) {
12:
13:        int ThreadId = Thread.CurrentThread.GetHashCode(
14: );
```

```
15:     for( int i = 0; i < 10; i++ )  
16:         //comment out lock statement  
17:         //and watch the exception fly  
18:         lock( this ) {  
19:             if( m_CurrentIndex < 10 ) {  
20:                 Thread.Sleep( (new Random()).Next( 2 ) *  
1000 );  
21:                 m_Array[m_CurrentIndex++] = ThreadId;  
22:             }  
23:         }  
24:     }  
25:  
26:     public void PrintArray() {  
27:         for( int i = 0; i < 10; i++ )  
28:             Console.WriteLine( "m_Array[{0}] = {1}", i,  
m_Array[i] );  
29:     }  
30:  
31:  
32:  
33:     public static void Main() {  
34:  
35:         ThreadLockTest tlt = new ThreadLockTest();  
36:         Thread t1 = new Thread( new ThreadStart(  
tlt.ThreadProc ) );  
37:         Thread t2 = new Thread( new ThreadStart(  
tlt.ThreadProc ) );  
38:  
39:         t1.Start();  
40:         t2.Start();  
41:         t1.Join();  
42:         t2.Join();  
43:  
44:         tlt.PrintArray();  
45:     }  
46: }
```

Một lần nữa, với câu lệnh lock, chương trình sẽ chạy tốt và mỗi vị trí c
mảng đều chứa giá trị băm của tiểu trình đã gán chúng vào. Tuy nhiên, nếu loại
các câu lệnh lock, chúng ta sẽ gặp lỗi ngoại lệ IndexOutOfRangeException. Lý
năm ở dòng 19. Sau khi m_CurrentIndex đã được kiểm tra, tiểu trình hiện tại

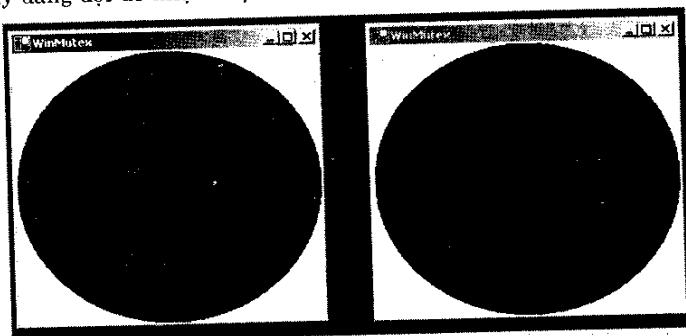
tạm ngủ (sleep) trong một khoảng thời gian ngẫu nhiên nào đó. Trong giai đoạn này, có thể có một tiểu trình khác chạy và không thay đổi m_CurrentIndex bằng cách tăng giá trị của nó lên trên 10.

2.2. Mutex

Tương tự như từ khoá lock, một mutex biểu diễn độc quyền truy xuất qua lại vào một hay nhiều tài nguyên. Không như miền găng, một mutex là một đối tượng cấp hệ thống kernel, và như vậy nó có thể chia sẻ giữa các tiến trình. Một trái của nó là thời gian cần để gọi mutex dài hơn so với việc sử dụng miền găng qua từ khoá lock.

Thay vì tạo một chương trình dòng lệnh nhỏ để minh họa Mutex, ở đây chúng ta sẽ tạo một ứng dụng Windows Forms nhỏ. Nó minh họa cách chia sẻ cùng một mutex giữa hai tiến trình. Để nhiều tiến trình truy xuất vào cùng một mutex, ta cần một số cơ chế. Trong thực tế, rất dễ để có chúng bằng cách tạo một mutex có cùng tên ở dạng hình thức trong một tham số string. Tham số string có thể được dùng cho nhiều tiến trình để nhận được mutex cấp kernel.

Hình 5.4-1 biểu diễn hai thể hiện của ứng dụng WinMutex. Khi một tiến trình nhận được mutex, một hình tròn xanh sẽ được vẽ. Khi mutex đã được giải phóng hay đang đợi để nhận được mutex, hình tròn sẽ có màu đỏ.



Hình 5.4-1 Ứng dụng WinMutex

Ví dụ 5.4-6 trình bày cài đặt của WinMutex. Lưu ý đặc biệt đến cấu trúc của đối tượng mutex sẽ được truy xuất đến bởi nhiều thể hiện của ứng dụng này.

Ví dụ 5.4-6 Ứng dụng WinMutex

```

1: using System;
2: using System.Drawing;
3: using System.Collections;
4: using System.ComponentModel;
5: using System.Windows.Forms;
6: using System.Data;

```

```
7: using System.Threading;
8:
9: namespace WinMutex
10: {
11:
12:     public class MainForm : System.Windows.Forms.Form
13:     {
14:
15:         private System.ComponentModel.IContainer
16:             components;
17:
18:         private Mutex m = new Mutex( false, "WinMutex" );
19:         private bool bHaveMutex = false;
20:
21:         public MainForm()
22:         {
23:             InitializeComponent();
24:
25:             // TODO: Add any constructor code after
26:             // InitializeComponent call
27:
28:             Thread T = new Thread( new ThreadStart(
29:                                         ThreadProc ) );
30:             T.Start();
31:
32:             }
33:
34:             /// <summary>
35:             /// Clean up any resources being used.
36:             /// </summary>
37:             public override void Dispose()
38:             {
39:                 if (components != null)
40:                 {
41:                     components.Dispose();
42:                 }
43:                 base.Dispose();
44:             }
```

```
43:  
44: #region Windows Form Designer generated code  
45: /// <summary>  
46: /// Required method for Designer support - do not  
     modify  
47: /// the contents of this method with the code  
     editor.  
48: /// </summary>  
49: private void InitializeComponent()  
50: {  
51:     this.components = new  
         System.ComponentModel.Container();  
52:     //  
53:     // MainForm  
54:     //  
55:     this.AutoScaleBaseSize = new  
         System.Drawing.Size(5, 13);  
56:     this.BackColor =  
         System.Drawing.SystemColors.Window;  
57:     this.ClientSize = new System.Drawing.Size(292,  
         273);  
58:     this.Name = "MainForm";  
59:     this.Text = "WinMutex";  
60:  
61: }  
62: #endregion  
63:  
64: protected override void OnPaint( PaintEventArgs e )  
{  
65:  
66:     if( this.bHaveMutex )  
         DrawCircle( System.Drawing.Color.Green );  
68:     else  
         DrawCircle( System.Drawing.Color.Red );  
70: }  
71:  
72: protected void DrawCircle( System.Drawing.Color  
     color ) {  
73:     Brush b = new SolidBrush( color );  
74:
```

```
75:         System.Drawing.Graphics g =
76:             this.CreateGraphics( );
77:         int x = this.Size.Width / 2;
78:         int y = this.Size.Height / 2;
79:         g.FillEllipse( b, 0, 0, this.ClientSize.Width,
80:                         this.ClientSize.Height );
81:         b.Dispose( );
82:         g.Dispose( );
83:
84:     }
85:
86:
87:     protected void ThreadProc( ) {
88:
89:         while( true ) {
90:             m.WaitOne( );
91:             bHaveMutex = true;
92:             Invalidate( );
93:             Update( );
94:             Thread.Sleep( 1000 );
95:             m.ReleaseMutex( );
96:             bHaveMutex = false;
97:             Invalidate( );
98:             Update( );
99:         }
100:    }
101:
102:    /// <summary>
103:    /// The main entry point for the application.
104:    /// </summary>
105:    [STAThread]
106:    static void Main()
107:    {
108:        Application.Run(new MainForm());
109:    }
110: }
111: }
```



Có vài điểm cần phải được chỉ ra trong ứng dụng trên bên cạnh việc chúng dùng chung mutex. Ở dòng 87 của ví dụ 5.4-6 là thủ tục tiêu trình ThreadProc. Phương thức này được dùng để lấy mutex và làm mất hiệu lực của form cha, đồng thời giúp cho vòng tròn vẽ ra có màu hợp lý. Tại sao không đơn thuần gọi phương thức DrawCircle trong phương thức ThreadProc? Câu trả lời nằm ở cơ chế quản lý bảng mã cửa sổ (handle map) của bản thân Windows đang được sử dụng bởi WindowsForms. Khi cố tạo ra một nội dung hình ảnh bởi một tiêu trình khác, một xử lý ngoại lệ sẽ được dùng vì sự vi phạm handle map. Về bản chất, không thể tạo ra một bản sao cho handle của một cửa sổ, đó là chính xác những gì xảy ra khi một tiêu trình con định tạo ra một nội dung hình ảnh. Để tránh ngoại lệ này, các tiêu trình thợ nên đơn thuần làm mất hiệu lực của form, cái mà khi đến lượt nó, sẽ sinh ra một message WM_PAINT và đặt trên hàng đợi thông điệp của ứng dụng. Tiêu trình chính của chương trình sẽ xử lý thông điệp đó sau này.

Chạy hai thẻ hiện của WinMutex cạnh nhau và bạn sẽ thấy rằng các hình tròn không bao giờ cùng xanh. Đó là kết quả của việc dùng Mutex và sử dụng đồng bộ hoá xuyên tiến trình.

2.3. AutoResetEvent

Một dạng khác của đồng bộ hoá xuyên tiêu trình là các sự kiện (event). Kiến trúc .NET cung cấp cả AutoResetEvents và ManualResetEvents. Một sự kiện cũng tương tự như một sự kiện trong Windows Forms, nơi mà ứng dụng chờ đợi một hành động nào đó của người dùng để sinh ra một sự kiện và sau đó trả lời sự kiện đó. Cũng như vậy, sự kiện có thể được dùng để đồng bộ hoá cơ chế xử lý da tiêu trình. Cái tên AutoResetEvent có được từ thực tế là sự kiện sẽ được khởi động (reset) lại tự động mỗi khi tiêu trình đang chạy được thông báo có sự kiện phát sinh.

Xem ví dụ sau: Một tiêu trình có trách nhiệm sản xuất vài thứ linh tinh và sau đó một tiêu trình khác tìm cách sử dụng hết chúng. Kiểu vấn đề này được gọi là bài toán sản xuất/tiêu thụ và là một ví dụ kinh điển về đồng bộ hoá tiêu trình. Trong ví dụ này, tiêu trình sản xuất sẽ phát sinh một sự kiện mỗi khi một vật được sản xuất. Để đáp ứng lại sự kiện đó, tiêu trình tiêu thụ sẽ làm gì đó với vật mới được tạo ra đó. Ví dụ 5.4-7 trình bày những điểm cơ bản của AutoResetEvent.

Ví dụ 5.4-7 AutoResetEvent

```

1: using System;
2: using System.Threading;
3:
4: namespace ProducerConsumer {
5:
6:     public class Factory {
7:
8:         private int[] Widgets = new int[100];
9:         private int WidgetIndex = 0;
10:        private AutoResetEvent NewWidgetEvent = new
11:            AutoResetEvent( false );

```

```
12:  
13:     protected void Producer() {  
14:  
15:         while( true ) { //run forever  
16:  
17:             lock( this ) {  
18:                 if( WidgetIndex < 100 ) {  
19:                     Widgets[ WidgetIndex ] = 1;  
20:                     Console.WriteLine("Widget {0} Produced",  
21:                                         WidgetIndex++ );  
22:                     NewWidgetEvent.Set();  
23:                 }  
24:             }  
25:             Thread.Sleep( (new Random()).Next( 5 ) *  
26:                             1000 );  
27:         }  
28:  
29:  
30:     protected void Consumer() {  
31:  
32:         while( true ) {  
33:             NewWidgetEvent.WaitOne();  
34:             int iWidgetIndex = 0;  
35:  
36:             lock( this ) {  
37:                 iWidgetIndex = --this.WidgetIndex;  
38:                 Console.WriteLine("Consuming widget {0}",  
39:                                     iWidgetIndex );  
40:                 Widgets[ iWidgetIndex-- ] = 0;  
41:             }  
42:         }  
43:  
44:     public void Run() {  
45:         //Create 3 producers  
46:         for( int i = 0; i < 3; i++ ) {  
47:             Thread producer = new ThreadStart(  
48:                 Producer );  
49:             producer.Start();  
50:         }  
51:         //Create 3 consumers  
52:         for( int i = 0; i < 3; i++ ) {  
53:             Thread consumer = new ThreadStart(  
54:                 Consumer );  
55:             consumer.Start();  
56:         }  
57:     }
```

```

56:     }
57:
58:     public static void Main( ) {
59:         Factory factory = new Factory( );
60:         factory.Run( );
61:     }
62: }
63: }
```

2.4. ManualResetEvent

Tương tự như AutoResetEvent, ManualResetEvent cũng có thể được dùng để thông báo rằng tiểu trình hiện tại đã phát sinh một sự kiện. Sự khác nhau giữa chúng là ManualResetEvent phải được khởi động lại nhờ vào việc gọi một phương thức khác thay vì có thể tự khởi động lấy. Đó là khác nhau duy nhất về mặt cơ chế. Cách sử dụng khác nhau giữa chúng thì thực sự phụ thuộc vào kiểu của đồng bộ hoá mà bạn cần. Với ManualResetEvent, ta có thể theo dõi tình trạng của một sự kiện để xác định được khi nào sự kiện sẽ cần khởi động lại.

2.5. Các tiểu trình dùng chung (Thread Pools)

Có nhiều ứng dụng dùng các tiểu trình để xử lý các yêu cầu của người dùng. Các ứng dụng như vậy gồm có: các máy chủ Web, các máy chủ Cơ sở dữ liệu, và các máy chủ ứng dụng. Thay vì phát sinh ra các tiểu trình mới mỗi khi nhận được yêu cầu của người dùng, Thread Pools cho phép các tiểu trình được dùng lại, các chi phí cho việc tạo lập và huỷ các tiểu trình được giảm thiểu. Mặc dù chi phí để tạo lập một tiểu trình thợ là nhỏ, chi phí để tạo lập và huỷ bỏ hàng trăm tiểu trình thì sẽ là một vấn đề ảnh hưởng đến hiệu năng.

Một tiểu trình dùng chung có thể được tạo ra trong ba tình huống: tình huống thứ nhất là dùng QueueUserWorkItem. Hàng đợi thời gian (timer queue) hay hàng đợi thao tác đăng ký hiện chờ là một hàm callback. Tình huống thứ hai là số lượng các tiểu trình sẽ được tạo ra và quản lý nhờ vào Thread Pool, chúng phụ thuộc vào số lượng bộ nhớ còn trống trên hệ thống và tình huống thứ ba là việc cài đặt một quản lý tiểu trình dùng chung.

2.6. QueueUserWorkItem

Để minh họa cách dùng QueueUserWorkItem, hãy nghĩ về nhu cầu một công ty bảo hiểm trong xử lý các yêu cầu bồi thường bảo hiểm. Sử dụng tiểu trình dùng chung, mỗi yêu cầu sẽ được coi là một việc cho một tiểu trình xử lý. Khi có một yêu cầu mới, ứng dụng có thể xếp nó trong hàng đợi như một mẩu việc dành cho tiểu trình dùng chung. Phương thức QueueUserWorkItem là một phương thức static của lớp ThreadPool và có dạng như sau:

`ThreadPool.QueueUserWorkItem (WaitCallback callback,
object state)`

Tham số đầu là đại diện callback của tiểu trình dùng để thực thi. Không giống như đại diện tiểu trình trong ví dụ trước, một đại diện WaitCallback cho

phép một tham số hình thức của kiểu object. Tham số object cho phép thông tin được gửi đến tiêu trình thư. Cho đến bây giờ, chúng ta vẫn chưa thử gửi thông tin đến tiêu trình thư. Ví dụ 5.4-8 dưới đây sẽ trình bày ví dụ về hằng bão hiềm sử dụng tiêu trình dùng chung để xử lý các yêu cầu.

Ví dụ 5.4-8 Thread Pools và QueueUserWorkItem

```

1: using System;
2: using System.Threading;
3:
4:
5: //The InsClaim represents
6: //a work item for a particular
7: //thread.
8: public struct InsClaim {
9:     public string ClaimId;
10:    public double Amount;
11: }
12:
13:
14:public class App {
15:
16:    public void ProcessInsuranceClaim( object state ) {
17:        //unbox the InsClaim passed in
18:        InsClaim theClaim = (InsClaim)state;
19:
20:        //Process the claim and sleep to simulate work
21:        Console.WriteLine("Processing Claim {0} for : {1}"
22:                         , theClaim.ClaimId, theClaim.Amount );
23:        Thread.Sleep( 5000 );
24:        Console.WriteLine("Claim {0} processed",
25:                           theClaim.ClaimId);
26:
27:    public static void Main( ) {
28:        App app = new App();
29:
30:        //Create 100 insurance claims
31:        Random r = new Random();
32:        for( int i = 0; i < 100; i++ ) {

```



```

33:         InsClaim claim;
34:         claim.ClaimId = string.Format("INS{0}", i);
35:         claim.Amount = r.NextDouble() * 5000;
36:         ThreadPool.QueueUserWorkItem( new
37:             WaitCallback( app.ProcessInsuranceClaim ), claim );
38:         System.Threading.ThreadStart
39:
40:         //allow threads in pool to run
41:         Console.ReadLine();
42:     }
43: }

```

Ví dụ bảo hiểm trên khá rõ ràng. Vòng lặp for được dùng để tạo ra 100 yêu cầu cần xử lý, bắt đầu ở dòng 33. Mỗi yêu cầu sau đó sẽ nằm trong hàng đợi và chờ đợi tiểu trình trong vùng lưu giữ tiểu trình dùng chung. Nhớ phải lưu ý các kết xuất được chen vào bởi ví dụ. Khi các yêu cầu đang bắt đầu được xử lý, các yêu cầu khác sẽ được xử lý bởi các tiểu trình đã kết thúc việc xử lý trước đó. Tiểu trình dùng chung là một công cụ hiệu quả cho các tác vụ nền, đặc biệt khi nó đòi hỏi thời gian chờ đợi.

3. MỞ RỘNG CÁC TIỂU TRÌNH .NET

Mặc dù .NET đã cung cấp một phương cách thống nhất cho các tiểu trình của tất cả các ngôn ngữ dùng kiến trúc .NET, nhưng thường như vẫn thiếu một sự kết nối và đóng gói các đối tượng tiểu trình. Bằng việc tạo ra một lớp cơ sở trừu tượng nhỏ (abstract) để biểu diễn tiểu trình thợ, khả năng mở rộng các hỗ trợ tiểu trình cơ bản sẽ chứng minh sự hữu ích khi phát triển các ứng dụng lớn hơn.

3.1. Lớp WorkerThread

Việc tạo một lớp trừu tượng đơn giản để biểu diễn các tiểu trình thợ thì không có gì khó khăn. Có một cặp lý thuyết cơ bản cần được cung cấp, đầu tiên là khả năng gắn kết dữ liệu với một tiểu trình xác định. Điều này cho phép tạo ra các tiểu trình thợ và tạo lập các dữ liệu cho chúng. Điều tiếp theo là khả năng dừng một tiểu trình thợ bằng một số cách nào đó. Cách dễ nhất để có được điều đó là cung cấp một phương thức Stop có thể tạo ra một ngoại lệ ThreadAbortException khi chạy phương thức Run của tiểu trình thợ. Ví dụ 5.4-9 mô tả cách cài đặt lớp tiểu trình thợ cơ sở.

Ví dụ 5.4-8 Lớp trừu tượng WorkerThread

```

1: using System;
2: using System.Threading;
3:

```

```
4: namespace SAMS.Threading
5: {
6:
7:
8:     /// <summary>
9:     /// Encapsulate a worker thread and data
10:    /// </summary>
11:   public abstract class WorkerThread {
12:
13:       private object ThreadData;
14:       private Thread thisThread;
15:
16:
17:       //Properties
18:       public object Data {
19:           get { return ThreadData; }
20:           set { ThreadData = value; }
21:       }
22:
23:       public object IsAlive {
24:           get { return thisThread == null ? false :
25:                           thisThread.IsAlive; }
26:
27:           /// <summary>
28:           /// Constructors
29:           /// </summary>
30:
31:           public WorkerThread( object data ) {
32:               this.ThreadData = data;
33:           }
34:
35:           public WorkerThread( ) {
36:               ThreadData = null;
37:           }
38:
39:           /// <summary>
40:           /// Public Methods
41:           /// </summary>
```

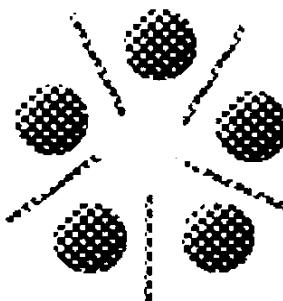
```

42:
43:     /// <summary>
44:     /// Start the worker thread
45:     /// </summary>
46:     public void Start() {
47:         thisThread = new Thread( new ThreadStart(
48:             this.Run ) );
49:         thisThread.Start();
50:     }
51:     /// <summary>
52:     /// Stop the current thread. Abort causes
53:     /// a ThreadAbortException to be raised within
54:     /// the thread
55:     /// </summary>
56:     public void Stop() {
57:         thisThread.Abort();
58:         while( thisThread.IsAlive );
59:         thisThread = null;
60:     }
61:
62:     /// <summary>
63:     /// To be implemented by derived threads
64:     /// </summary>
65:     protected abstract void Run();
66: }
67: }
```

Cài đặt của lớp WorkerThread cho thấy sự dễ dàng khi tạo ra một lớp bọc cho các hỗ trợ tiêu trình đã tồn tại. Một lớp bọc là một thuật ngữ chỉ một lớp có các tính năng cơ bản đã được bọc để mở rộng như là một kiểu chung hay lớp chung. Phương thức quan trọng nhất bên trong lớp WorkerThread là phương thức trùu tượng Run dùng để bắt đầu tiêu trình.

3.2. Bữa tối của các nhà triết học

Để làm việc với lớp WorkerThread, một vấn đề đồng bộ hoá kinh điển, được biết đến như là “Bữa tối của các nhà triết học”, sẽ được chúng ta tìm hiểu và cài đặt dưới đây. Trong bài toán này, một nhóm các nhà triết học ngồi quanh một chiếc bàn tròn. Họ vừa ăn vừa suy nghĩ. Để ăn, mỗi người phải lấy được một chiếc nĩa ở bên trái hay bên phải chỗ ngồi. Hình 5.4-2 cho thấy sự sắp xếp của các nhà triết học và những cái nĩa.



Hình 5.4-2 Bữa tối của các nhà triết học

Đi nhiên, vấn đề là khi mỗi nhà triết học có một cái nĩa thì phải ngồi đợi có được cái nĩa khác nữa. Tình huống này được gọi là deadlock. Trong tình huống deadlock, nhiều tiểu trình cùng chờ đợi các nguồn tài nguyên không thể được do bị khóa (lock) vòng tròn lẫn nhau. Vì vậy, mục đích của các nhà triết học là cầm cả hai nĩa hay không cầm gì cả chứ không bao giờ cầm giữ một nĩa. Vì 5.4-9 trình bày kỹ thuật và cách xử lý này với việc sử dụng lớp cơ WorkerThread.

Ví dụ 5.4-9 Bữa tối của các nhà triết học

```

1: using System;
2: using System.Threading;
3: using SAMS.Threading;
4:
5: namespace DinningPhilosophers
6: {
7:
8:     public struct PhilosopherData {
9:         public int             PhilosopherId;
10:        .public Mutex          RightChopStick;
11:        public Mutex          LeftChopStick;
12:        public int             AmountToEat;
13:        public int             TotalFood;
14:    }
15:
16:
17:
18:    public class Philosopher : WorkerThread
19:    {

```

```
20:     public Philosopher( object data ) : base( data ) { }
21:
22:     //Implement the abstract Run Method
23:     protected override void Run( ) {
24:
25:         PhilosopherData pd = (PhilosopherData)Data;
26:         Random r = new Random( pd.PhilosopherId );
27:         Console.WriteLine("Philosopher {0} ready",
28:                           pd.PhilosopherId );
29:         WaitHandle[] chopSticks = new WaitHandle[] {
30:             pd.LeftChopStick, pd.RightChopStick };
31:
32:         while( pd.TotalFood > 0 ) {
33:             //Get both chop sticks
34:             WaitHandle.WaitAll( chopSticks );
35:             Console.WriteLine("Philosopher {0} eating {1}
36:                               of {2} food", pd.PhilosopherId, pd.AmountToEat,
37:                               pd.TotalFood );
38:             pd.TotalFood -= pd.AmountToEat;
39:             Thread.Sleep( r.Next(1000,5000) );
40:
41:             //Release the chopsticks
42:             Console.WriteLine("Philosopher {0}
43:                               thinking", pd.PhilosopherId );
44:             pd.RightChopStick.ReleaseMutex();
45:             pd.LeftChopStick.ReleaseMutex();
46:
47:         }
48:
49:
50:     public class Restaurant {
51:
52:         public static void Main( ) {
53:             Mutex[] chopSticks = new Mutex[5];
```

```
54:  
55:     //init the chopSticks  
56:     for( int i = 0; i < 5; i++ )  
57:         chopSticks[i] = new Mutex( false );  
58:  
59:     //Create the Five Philosophers  
60:     for( int i = 0; i < 5; i++ ) {  
61:         PhilosopherData pd;  
62:         pd.PhilosopherId = i + 1;  
63:         pd.RightChopStick = chopSticks[ i - 1 >= 0 ? ( i  
64:             - 1 ) : 4 ];  
65:         pd.LeftChopStick = chopSticks[i];  
66:         pd.AmountToEat = 5;  
67:         pd.TotalFood = 35;  
68:         Philosopher p = new Philosopher( pd );  
69:         p.Start();  
70:  
71:         Console.ReadLine();  
72:     }  
73: }  
74: }
```

Ví dụ “Bữa ăn tối của các nhà triết học” cho thấy cách dùng phương thức static WaitHandle.WaitAll để có cả hai cái nĩa. Một cách hiệu quả, phương thức WaitAll sẽ không trả về cho đến khi nó có thể lấy được cả hai đũi tương ứng được dẫn xuất từ WaitHandle, trong trường hợp này là một cặp của mutex cho mỗi chiếc nĩa. Phương thức WaitAll còn cung cấp hai thể hiện chống cho phép thiết lập thời gian chờ đợi và một cờ Boolean để xác định các phạm vi. WaitAll chứng minh cách giải quyết khi sử dụng chung nguồn tài nguyên và tránh trường hợp deadlock do chờ đợi vòng tròn.

4. KẾT CHƯƠNG

Những điều thảo luận ở đây gói gọn trong khả năng hỗ trợ tiêu trình của kiến trúc .NET. Một lần nữa, mặc dù các lớp tiêu trình và các cơ chế được cung cấp không hẳn lúc nào cũng đủ để dùng, bạn phải luôn nghĩ trong đầu rằng kiến trúc .NET định hướng đến đa ngôn ngữ. Và như vậy, một cái đặt đầy đủ các chức năng mạnh là điều không phải lúc nào cũng có thể, và như thế, các lập trình viên cần tự mở rộng những cơ sở đã được cung cấp sẵn cho ngôn ngữ của mình khi cần. Sử dụng lớp trùu tượng cơ sở WorkerThread, bạn có thể mở rộng nó cho các nhu cầu đặc trưng và tiếp tục cải tiến và sàng lọc qua thời gian sau này.

SÁCH ĐÃ XUẤT BẢN

LẬP TRÌNH :

1. Bài tập ngôn ngữ C từ A đến Z
2. Giáo trình lý thuyết và bài tập ngôn ngữ C [tập 1 & 2]
3. Lập trình Windows (Bằng Visual C++)
4. Thiết kế đồ họa định hướng đối tượng với C++
5. Giáo trình lý thuyết và bài tập Pascal [tập 1 & 2]
6. Giáo trình lý thuyết và bài tập Foxpro [tập 1]
7. Sử dụng và khai thác Visual Foxpro 6.0
8. Access 2000 lập trình ứng dụng cơ sở dữ liệu [tập 1 & 2]
9. Giáo trình lý thuyết và bài tập Oracle
10. Giáo trình lý thuyết và bài tập Java
11. Java lập trình mạng
12. Giáo trình lý thuyết và bài tập visual J++6
13. Tự học lập trình cơ sở dữ liệu với VB6 trong 21 ngày [tập 1 & 2]
14. Visual Basic 6.0 – Lập trình cơ sở dữ liệu
15. Các kỹ xảo lập trình với VB6 và Delphi
16. Giáo trình lý thuyết và bài tập Delphi
17. Hợp ngữ và lập trình ứng dụng [tập 1&2]
18. Lập trình Linux [tập 1]
19. Đồ họa vi tính [tập 1 & 2]
20. Giáo trình trí tuệ nhân tạo – Mạng Nơron phương pháp và ứng dụng
21. Giáo trình trí tuệ nhân tạo – Cấu trúc dữ liệu + Thuật giải đì truyền = Lập trình tiến hóa
22. Giáo trình trí tuệ nhân tạo – Máy học
23. Tự học lập trình bằng ví dụ với VISUAL C++ MFC
24. Lập trình ứng dụng chuyên nghiệp SQL Server 2000 (tập 1 & 2)
25. Từng bước học lập trình với VB.NET
26. Lập trình với Windows C#.NET
27. Tự học lập trình chuyên sâu với VB.NET trong 21 ngày

INTERNET & VIỄN THÔNG

28. Internet Explorer toàn tập
29. Internet working với TCP/IP [tập 1&2]
30. Thực hành thiết kế trang web với Frontpage 2000
31. Frontpage 2000 toàn tập
32. Thiết kế hoạt hình cho web với Macromedia Flash
33. Hướng dẫn thiết kế trang web tương tác bằng Java Script

34. Sử dụng E-mail và tin học văn phòng trên mạng với Outlook 2000
35. Modem truyền số liệu
36. Cơ sở kỹ thuật chuyển mạch và tổng đài [tập 1 & 2]
37. XML – Nền tảng và ứng dụng
38. Xây dựng ứng dụng web với JSP, servlet, Javabeans.
39. ASP 3.0 / ASP.NET
40. Thiết kế trang WEB động với DHTML.
41. Lập trình ứng dụng WEB với JSP servlet
42. Kỹ thuật truyền số liệu.
43. Thiết kế web với Macromedia Dreamweaver 4.0

THIẾT KẾ ĐỒ HỌA

44. Vẽ minh họa với Corel Draw 9.0
45. Vẽ minh họa với Corel Draw 10.0 [tập 1, 2 & 3]
46. Autocad 2000 [tập 1 & 2]
47. Thiết kế 3 chiều với 3D Studio Max 3.0
48. Thiết kế 3 chiều với 3D Studio Max 4.0
49. Adobe Photoshop 5.5 và Imageready 2.0
50. Adobe Photoshop 6.0 và Imageready 3.0
51. Adobe Photoshop bài tập và kỹ xảo
52. Adobe Illustration 8.0
53. Adobe Illustration với các kỹ thuật thiết kế nâng cao
54. Adobe InDesign
55. Dàn trang với Quarkxpress
56. Thiết kế 3 chiều với 3DS MAX4

HỆ ĐIỀU HÀNH VÀ MẠNG

57. Vận hành và khai thác Windows 98
58. Làm chủ Windows 2000 Server [tập 1 & 2]
59. Giáo trình mạng Novell Netware 5.0
60. Giáo trình cấu trúc máy tính
61. Vi mạch và mạch tạo sóng
62. Họ vi xử lý 8051
63. Tìm hiểu cấu trúc và hướng dẫn sửa chữa, bảo trì máy PC [tập 1 & 2]
64. Giáo trình hệ thống mạng máy tính CCNA Semester I
65. Nâng vững Windows XP professional [tập 1]
66. Bảo mật và tối ưu hóa Linux
67. Giáo trình lý thuyết và thực hành Linux [tập 1]

VĂN PHÒNG

68. Microsoft Word 2000
69. Đồ họa và Multimedia trong văn phòng với MS Power Point 2000

LẬP TRÌNH WINDOWS VỚI C#.net

NHÀ XUẤT BẢN LAO ĐỘNG – XÃ HỘI

41B Lý Thái Tổ – Hà Nội

Tel: 8.241706 – Fax: 9.348283

* * *

Chịu trách nhiệm xuất bản :

NGUYỄN ĐÌNH THIỆM

Chịu trách nhiệm nội dung:

THANH DUY

Biên tập kỹ thuật:

DIỆU NGỌC

Sửa bản in :

NGỌC AN

Trình bày bìa :

NHŨ ĐÌNH NGOẠN

Tổng phát hành tại **NHÀ SÁCH MINH KHAI**

249 Nguyễn Thị Minh Khai – Quận 1

TP. Hồ Chí Minh

ĐT:(08) 8250590 – (08) 9250591 – Fax:(08) 8331124

E-mail(CLBbandoc) : mk.pub @cinet.vnnews.com

E-mail(thuongmai) : mk.book @cinet.vnnews.com

Web site : www.minhkhai.com.vn