





Giải pháp lập trình C#

tổng hợp và biên dịch
Nguyễn Ngọc Bình Phương - Thái Thanh Phong

cùng sự cộng tác của
Nguyễn Thanh Nhân - Trần Lê Vĩnh Phong
Nguyễn Quang Nam - Đinh Phan Chí Tâm
Bùi Minh Khoa - Lê Ngọc Sơn
Thái Kim Phụng - Lê Trần Nhật Quỳnh

Chịu trách nhiệm xuất bản: TS. Nguyễn Xuân Thủy

Biên tập: Hồ Nguyễn Thị Thanh Thủy

Trình bày bìa: Nguyễn Thị Thanh Thủy

Chế bản & Sửa bản in: Nguyễn Ngọc Bình Phương

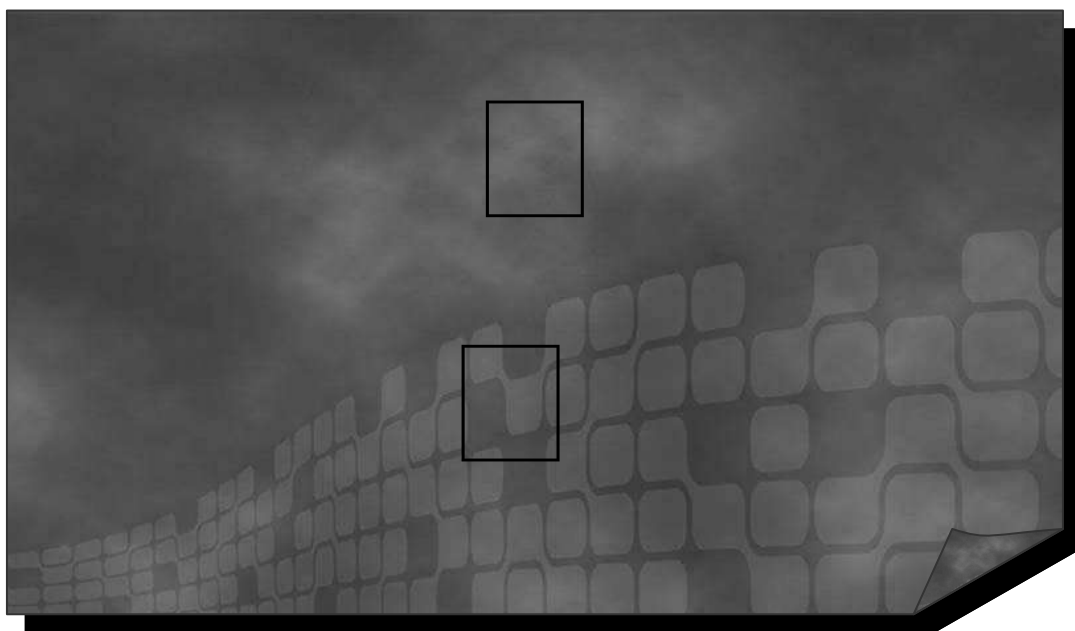
Nhà sách Đất Việt

Địa chỉ: 225 Nguyễn Tri Phương, Q.5, TP. Hồ Chí Minh

Điện thoại: (08) 2 652 039

E-mail: datviet@dvpub.com.vn

Website: www.dvpub.com.vn



Nguyễn Ngọc Bình Phương - Thái Thanh Phong

tổng hợp & biên dịch

<http://www.dvpub.com.vn/dv/details.aspx?itemid=243>

NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI



Các giải pháp lập trình C# khảo sát chiều rộng của thư viện lớp .NET Framework và cung cấp giải pháp cụ thể cho các vấn đề thường gặp. Mỗi giải pháp được trình bày theo dạng “vấn đề/giải pháp” một cách ngắn gọn và kèm theo là các ví dụ mẫu.

Các giải pháp lập trình C# không nhằm mục đích hướng dẫn bạn cách lập trình C#. Tuy vậy, ngay cả khi mới làm quen với lập trình ứng dụng được xây dựng trên .NET Framework với C#, bạn cũng sẽ nhận thấy quyển sách này là một tài nguyên vô giá.

Ở mức lý tưởng, khi bạn đối mặt với một vấn đề, quyển sách này sẽ cung cấp một giải pháp—hay ít nhất nó sẽ gợi cho bạn hướng đi đúng. Ngay cả nếu bạn chỉ muốn mở rộng kiến thức của mình về thư viện lớp .NET, **Các giải pháp lập trình C#** cũng là một tài liệu rất hữu ích.

Bạn không thể trở nên thành thạo C# và các lớp trong thư viện lớp .NET nếu chỉ đơn thuần đọc về chúng, bạn phải sử dụng và thử nghiệm chúng bằng cách viết thật nhiều chương trình. Cấu trúc và nội dung của quyển sách này cũng như tính khả thi trong thế giới thực của các giải pháp được đưa ra sẽ cung cấp điểm khởi đầu hoàn hảo, để từ đó làm bàn đạp cho việc thử nghiệm của chính bạn.

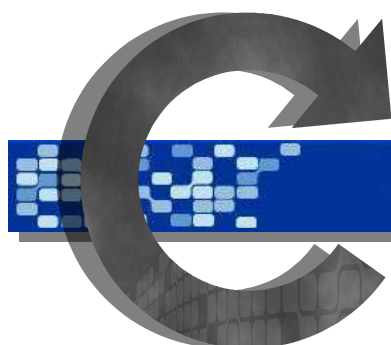
Phần mã lệnh trong quyển sách này đã được viết và chạy thử nghiệm trên phiên bản 1.1 của .NET Framework. Trong nhiều trường hợp, bạn sẽ nhận thấy ví dụ mẫu này cũng sẽ chạy trên phiên bản 1.0 hay 2.0 của .NET Framework, tuy nhiên điều này chưa được thử nghiệm.

Chúng tôi xin chân thành cảm ơn các bạn Nguyễn Thanh Nhân, Trần Lê Vĩnh Phong, Nguyễn Quang Nam, Đinh Phan Chí Tâm, Bùi Minh Khoa, Lê Ngọc Sơn, Thái Kim Phụng, và Lê Trần Nhật Quỳnh đã có những đóng góp

quý báu cho quyền sách; cảm ơn *Nhà xuất bản Giao thông Vận tải* và *Nhà sách Đất Việt* đã tạo điều kiện cho quyền sách này đến với bạn đọc.

Do lần đầu tiên xuất bản nên quyền sách này khó tránh khỏi những thiếu sót. Rất mong nhận được ý kiến đóng góp và nhận xét của các bạn để lần tái bản sau được hoàn thiện hơn.

Trân trọng cảm ơn



CẤU TRÚC CỦA SÁCH

Quyển sách này được chia thành 17 chương, mỗi chương tập trung vào một chủ đề cụ thể trong quá trình tạo các giải pháp *C#*.

Chương 1: **PHÁT TRIỂN ỨNG DỤNG**

Chương 2: **THAO TÁC DỮ LIỆU**

Chương 3: **MIỀN ỨNG DỤNG, CƠ CHẾ PHẢN CHIẾU,
VÀ SIÊU DỮ LIỆU**

Chương 4: **TIỂU TRÌNH, TIẾN TRÌNH, VÀ SỰ ĐỒNG BỘ**

Chương 5: **XML**

Chương 6: **WINDOWS FORM**

Chương 7: **ASP.NET VÀ WEB FORM**

Chương 8: **ĐỒ HỌA, ĐA PHƯƠNG TIỆN, VÀ IN ÁN**

Chương 9: **FILE, THƯ MỤC, VÀ I/O**

Chương 10: **CƠ SỞ DỮ LIỆU**

Chương 11: **LẬP TRÌNH MẠNG**

Chương 12: **DỊCH VỤ WEB XML VÀ REMOTING**

Chương 13: **BẢO MẬT**

Chương 14: **MẬT MÃ**

Chương 15: **KHẢ NĂNG LIÊN TÁC** **MÃ LỆNH**
KHÔNG-ĐƯỢC-QUẢN-LÝ

Chương 16: **CÁC GIAO DIỆN VÀ MẪU THÔNG DỤNG**

Chương 17: **SỰ HÒA HỢP VỚI MÔI TRƯỜNG WINDOWS**



Quyển sách này sử dụng các quy ước như sau:

Về font chữ

- **Chữ in nghiêng**—Dùng cho tên riêng, tên file và thư mục, và đôi khi để nhấn mạnh.
- **Chữ với bề rộng cố định** (font *Courie New*)—Dùng cho các đoạn chương trình, và cho các phần tử mã lệnh như câu lệnh, tùy chọn, biến, đặc tính, khóa, hàm, kiểu, lớp, không gian tên, phương thức, module, thuộc tính, thông số, giá trị, đối tượng, sự kiện, phương thức thụ lý sự kiện, thẻ *XML*, thẻ *HTML*, nội dung file, và kết xuất từ các câu lệnh.
- **Chữ in đậm với bề rộng cố định**—Dùng trong các đoạn chương trình để nêu bật một phần quan trọng của mã lệnh hoặc dùng cho các dòng lệnh, câu lệnh *SQL*.

Về ký hiệu



Vấn đề



Thủ thuật



Giải pháp



Ghi chú



Để chạy được những ví dụ mẫu đi kèm quyển sách này, bạn sẽ cần những phần mềm sau đây:

- **Microsoft .NET Framework SDK version 1.1**
- **Microsoft Visual Studio .NET 2003**
- **Microsoft Windows 2000, Windows XP,**
hoặc **Microsoft Windows Server 2003**
- **Microsoft SQL Server 2000** hoặc **MSDE**
đối với các mục trong chương 10
- **Microsoft Internet Information Services (IIS)**
đối với một số mục trong chương 7 và chương 12

Yêu cầu tối thiểu về phần cứng là bộ vi xử lý *Pentium II* 450 MHz, với dung lượng *RAM* tối thiểu là 128 MB nếu bạn đang sử dụng *Microsoft Windows 2000*, và là 256 MB nếu bạn đang sử dụng *Windows XP*, *Windows 2000 Server*, hay *Windows Server 2003*. Bạn cần khoảng 5 GB dung lượng đĩa cứng còn trống để cài đặt *Visual Studio .NET 2003*. Những giá trị này là mức tối thiểu, quá trình phát triển sẽ dễ dàng hơn trên một hệ thống với dung lượng *RAM* lớn và đĩa cứng còn trống nhiều.

Mặc dù bản hiện thực *.NET Framework* cho *Windows* của *Microsoft* là tiêu điểm của quyển sách này, một mục tiêu quan trọng là cấp một

tài nguyên hữu ích cho những lập trình viên *C#* không quan tâm đến nền mà họ đang làm việc hoặc công cụ mà họ truy xuất. Ngoài những chủ đề đặc biệt không được hỗ trợ trên tất cả nền *.NET* (như *Windows Form*, *ADO.NET*, và *ASP.NET*), nhiều ví dụ mẫu trong quyển sách này đều hợp lệ trên tất cả bản hiện thực *.NET*.



Mã lệnh được cấp ở dạng tập các giải pháp và dự án *Visual Studio .NET 2003*, được tổ chức theo chương và số đề mục. Mỗi chương là một giải pháp độc lập, và mỗi đề mục là một dự án độc lập bên trong giải pháp của chương. Một vài đề mục trong chương 11 và chương 12 trình bày về lập trình mạng gồm những dự án độc lập có chứa các phần client và server trong giải pháp của đề mục.

Mặc dù tất cả những ví dụ mẫu được cấp ở dạng dự án *Visual Studio .NET*, nhưng hầu hết đều bao gồm một file nguồn đơn mà bạn có thể biên dịch và chạy độc lập với *Visual Studio .NET*. Nếu không sử dụng *Visual Studio .NET 2003*, bạn có thể định vị mã nguồn cho một đề mục cụ thể bằng cách duyệt cấu trúc thư mục của ví dụ mẫu. Ví dụ, để tìm mã nguồn cho mục 4.3, bạn sẽ tìm nó trong thư mục *Chuong04\04-03*. Nếu sử dụng trình biên dịch dòng lệnh thì phải bảo đảm rằng bạn đã thêm tham chiếu đến tất cả các assembly cần thiết.

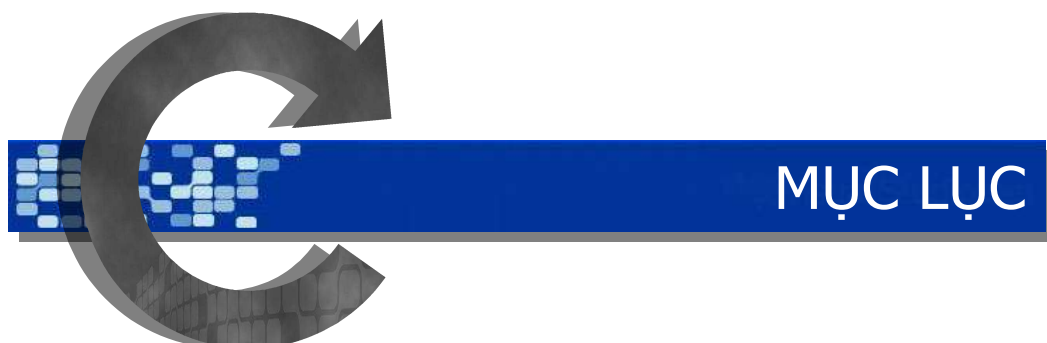
Một số ứng dụng mẫu yêu cầu các đối số dòng lệnh (sẽ được mô tả trong phần văn bản của đề mục). Nếu sử dụng *Visual Studio .NET*, bạn có thể nhập các đối số này trong *Project Properties* (mục *Debugging* của phần *Configuration Properties*). Nhớ rằng, nếu cần nhập tên thư mục hay file có chứa khoảng trắng thì bạn cần đặt tên đầy đủ trong dấu nháy kép.

Tất cả ví dụ truy xuất dữ liệu *ADO.NET* được tạo với *SQL Server 2000*. Chúng cũng có thể được sử dụng với *SQL Server 7* và *MSDE*.

Visual Studio .NET có chứa các kịch bản *SQL* để cài đặt các cơ sở dữ liệu mẫu *Northwind* và *Pubs* nếu chúng chưa hiện diện (các file *instnwnd.sql* và *instpubs.sql* trong thư mục *C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\ v1.1\Samples\Setup*). Bạn có thể chạy các kịch bản này bằng *Query Analyzer* (với *SQL Server*) hay *OSQL.exe* (với *MSDE*).

Để sử dụng các đề mục trong chương 7 và chương 12, bạn cần chép chúng vào thư mục *I:\CSharp* (đường dẫn này là mã cứng trong các file dự án *Visual Studio .NET*). Bạn cũng sẽ cần tạo một thư mục ảo có tên là *CSharp* ánh xạ đến *I:\CSharp*. Bạn có thể cài đặt phép ánh xạ này bằng *IIS Manager*. Thực hiện theo các bước dưới đây:

1. Khởi chạy *IIS Manager* (chọn *Start | Control Panel | Administrative Tools | Internet Information Services*).
2. Khởi chạy *Virtual Directory Wizard* trong *IIS Manager* bằng cách nhấp phải vào *Default Web Site* và chọn *New | Virtual Directory* từ menu ngữ cảnh.
3. Nhấp *Next* để bắt đầu. Mẫu thông tin đầu tiên là bí danh *CSharp*. Nhấp *Next* để tiếp tục.
4. Mẫu thông tin thứ hai là thư mục vật lý *I:\CSharp*. Nhấp *Next* để tiếp tục.
5. Cửa sổ thuật sĩ cuối cùng cho phép bạn điều chỉnh quyền cho thư mục ảo. Bạn nên sử dụng các thiết lập mặc định. Nhấp *Next*.
6. Nhấp *Finish* để kết thúc trình thuật sĩ. Bạn sẽ thấy thư mục ảo này trong phần cây của *IIS Manager*.
7. Khai triển thư mục ảo *CSharp* trong *IIS* thành thư mục nằm trong *CSharp\Chuong07\07-01*.
8. Nhấp phải vào thư mục này, chọn *Properties*, rồi nhấp vào nút *Create* trong thẻ *Directory* để chuyển thư mục này thành thư mục ứng dụng *Web*.
9. Lặp lại bước 8 cho mỗi mục trong chương 7.
10. Theo trình tự đã được trình bày trong các bước 7-9, tạo thư mục ứng dụng *Web* cho các đề mục 12.2, 12.3, 12.4, và 12.6 trong chương 12.



LỜI NÓI ĐẦU.....	7
CẤU TRÚC CỦA SÁCH.....	10
QUY ƯỚC.....	12
YÊU CẦU VỀ HỆ THỐNG.....	15
CÁCH SỬ DỤNG ĐĨA CD.....	18
MỤC LỤC.....	20
Chương 1: PHÁT TRIỂN ỨNG DỤNG.....	29
1. Tạo ứng dụng Console.....	31
2. Tạo ứng dụng dựa-trên-Windows.....	33
3. Tạo và sử dụng module.....	37
4. Tạo và sử dụng thư viện.....	39
5. Truy xuất các đối số dòng lệnh.....	40
6. Chọn biên dịch một khối mã vào file thực thi.....	42
7. Truy xuất một phần tử chương trình có tên trùng với một từ khóa.....	45

8. Tạo và quản lý cặp khóa tên mạnh.....	45
9. Tạo tên mạnh cho assembly.....	47
10. Xác minh một assembly tên mạnh không bị sửa đổi.....	49
11. Hoàn việc ký assembly.....	50
12. Ký assembly với chữ ký số Authenticode.....	52
13. Tạo và thiết lập tin tưởng một SPC thử nghiệm.....	54
14. Quản lý Global Assembly Cache.....	56
15. Ngăn người khác dịch ngược mã nguồn của bạn.....	56

Chương 2: THAO TÁC DỮ LIỆU

59

1. Thao tác chuỗi một cách hiệu quả.....	61
2. Mã hóa chuỗi bằng các kiểu mã hóa ký tự.....	62
3. Chuyển các kiểu giá trị cơ bản thành mảng kiểu byte.....	65
4. Mã hóa dữ liệu nhị phân thành văn bản.....	67
5. Sử dụng biểu thức chính quy để kiểm tra dữ liệu nhập.....	70
6. Sử dụng biểu thức chính quy đã được biên dịch.....	72
7. Tạo ngày và giờ từ chuỗi.....	75
8. Cộng, trừ, so sánh ngày giờ.....	76
9. Sắp xếp một mảng hoặc một ArrayList.....	78
10. Chép một tập hợp vào một mảng.....	79
11. Tạo một tập hợp kiểu mạnh.....	80
12. Lưu một đối tượng khả-tuần-tự-hóa vào file.....	81

Chương 3: MIỀN ỨNG DỤNG, CƠ CHẾ PHẢN CHIẾU, VÀ SIÊU DỮ LIỆU

86

1. Tạo miền ứng dụng.....	88
2. Chuyển các đối tượng qua lại các miền ứng dụng.....	90
3. Tránh nạp các assembly không cần thiết vào miền ứng dụng.....	91
4. Tạo kiểu không thể vượt qua biên miền ứng dụng.....	92
5. Nạp assembly vào miền ứng dụng hiện hành.....	92
6. Thực thi assembly ở miền ứng dụng khác.....	94
7. Thể hiện hóa một kiểu trong miền ứng dụng khác.....	95
8. Truyền dữ liệu giữa các miền ứng dụng.....	101
9. Giải phóng assembly và miền ứng dụng.....	103
10. Truy xuất thông tin Type.....	104
11. Kiểm tra kiểu của một đối tượng.....	106
12. Tạo một đối tượng bằng cơ chế phản chiếu.....	107
13. Tạo một đặc tính tùy biến.....	110
14. Sử dụng cơ chế phản chiếu để kiểm tra các đặc tính của một phân tử chương trình.....	113

Chương 4: TIỂU TRÌNH, TIẾN TRÌNH, VÀ SỰ ĐỒNG BỘ

115

1. Thực thi phương thức với thread-pool.....	117
2. Thực thi phương thức một cách bất đồng bộ.....	121
3. Thực thi phương thức bằng Timer.....	129
4. Thực thi phương thức bằng cách ra hiệu đối tượng WaitHandle.....	132
5. Thực thi phương thức bằng tiểu trình mới.....	135
6. Điều khiển quá trình thực thi của một tiểu trình.....	137

7. Nhận biết khi nào một tiến trình kết thúc.....	142
8. Đồng bộ hóa quá trình thực thi của nhiều tiến trình.....	143
9. Tạo một đối tượng tập hợp có tính chất an-toàn-về-tiến-trình.....	148
10. Khởi chạy một tiến trình mới.....	149
11. Kết thúc một tiến trình.....	152
12. Bảo đảm chỉ có thể chạy một thể hiện của ứng dụng tại một thời điểm.....	154

Chương 5: XML **157**

1. Hiển thị cấu trúc của một tài liệu XML trong TreeView.....	159
2. Chèn thêm nút vào tài liệu XML.....	164
3. Chèn thêm nút vào tài liệu XML một cách nhanh chóng.....	166
4. Tìm một nút khi biết tên của nó.....	169
5. Thu lấy các nút XML trong một không gian tên XML cụ thể.....	170
6. Tìm các phần tử với biểu thức XPath.....	172
7. Đọc và ghi XML mà không phải nạp toàn bộ tài liệu vào bộ nhớ.....	175
8. Xác nhận tính hợp lệ của một tài liệu XML dựa trên một Schema.....	178
9. Sử dụng XML Serialization với các đối tượng tùy biến.....	184
10. Tạo XML Schema cho một lớp .NET.....	188
11. Tạo lớp từ một XML Schema.....	188
12. Thực hiện phép biến đổi XSL.....	189

Chương 6: WINDOWS FORM **193**

1. Thêm điều kiểm vào form lúc thực thi.....	195
2. Liên kết dữ liệu vào điều kiểm.....	197
3. Xử lý tất cả các điều kiểm trên form.....	199
4. Theo vết các form khả kiến trong một ứng dụng.....	200
5. Tìm tất cả các form trong ứng dụng MDI.....	201
6. Lưu trữ kích thước và vị trí của form.....	203
7. Buộc ListBox cuộn xuống.....	205
8. Chỉ cho phép nhập số vào TextBox.....	206
9. Sử dụng ComboBox có tính năng auto-complete.....	207
10. Sắp xếp ListView theo cột bất kỳ.....	211
11. Liên kết menu ngữ cảnh vào điều kiểm.....	213
12. Sử dụng một phần menu chính cho menu ngữ cảnh.....	214
13. Tạo form đa ngôn ngữ.....	217
14. Tạo form không thể di chuyển được.....	219
15. Làm cho form không đường viền có thể di chuyển được.....	220
16. Tạo một icon động trong khay hệ thống.....	222
17. Xác nhận tính hợp lệ của đầu vào cho một điều kiểm.....	223
18. Thực hiện thao tác kéo-và-thả.....	226
19. Sử dụng trợ giúp cảm-ngữ-cảnh.....	228
20. Áp dụng phong cách Windows XP.....	229
21. Thay đổi độ đục của form.....	231

Chương 7: ASP.NET VÀ WEB FORM **234**

1. Chuyển hướng người dùng sang trang khác.....	236
2. Duy trì trạng thái giữa các yêu cầu của trang.....	237
3. Tạo các biến thành viên có trạng thái cho trang.....	243
4. Đáp ứng các sự kiện phía client với JavaScript.....	244
5. Hiển thị cửa sổ pop-up với JavaScript.....	247
6. Thiết lập focus cho điều khiển.....	249
7. Cho phép người dùng upload file.....	250
8. Sử dụng IIS authentication.....	253
9. Sử dụng Forms authentication.....	257
10. Thực hiện xác nhận tính hợp lệ có-chọn-lựa.....	260
11. Thêm động điều khiển vào Web Form.....	263
12. Trả về động một bức hình.....	266
13. Nạp điều khiển người dùng bằng mã lệnh.....	270
14. Sử dụng page-caching và fragment-caching.....	275
15. Dừng lại dữ liệu với ASP.NET Cache.....	276
16. Kích hoạt việc gỡ rối ứng dụng Web.....	280
17. Thay đổi quyền đã cấp cho mã ASP.NET.....	284

Chương 8: ĐỒ HỌA, ĐA PHƯƠNG TIỆN, VÀ IN ẤN

287

1. Tìm tất cả các font đã được cài đặt.....	289
2. Thực hiện "hit testing" với shape.....	291
3. Tạo form có hình dạng tùy biến.....	295
4. Tạo điều khiển có hình dạng tùy biến.....	297
5. Thêm tính năng cuộn cho một bức hình.....	301
6. Thực hiện chụp màn hình Desktop.....	303
7. Sử dụng "double buffering" để tăng tốc độ vẽ lại.....	305
8. Hiển thị hình ở dạng thumbnail.....	308
9. Phát tiếng "beep" của hệ thống.....	310
10. Chạy file audio.....	311
11. Chạy file video.....	313
12. Lấy thông tin về các máy in đã được cài đặt.....	317
13. In văn bản đơn giản.....	321
14. In văn bản có nhiều trang.....	324
15. In text dạng wrapping.....	328
16. Hiển thị print-preview.....	330
17. Quản lý tác vụ in.....	333
18. Sử dụng Microsoft Agent.....	338

Chương 9: FILE, THƯ MỤC, VÀ I/O

346

1. Truy xuất các thông tin về file hay thư mục.....	348
2. Thiết lập các thuộc tính của file và thư mục.....	353
3. Chép, chuyển, xóa file hay thư mục.....	354
4. Tính kích thước của thư mục.....	357
5. Truy xuất thông tin phiên bản của file.....	359
6. Sử dụng TreeView để hiển thị cây thư mục just-in-time.....	360
7. Đọc và ghi file văn bản.....	363
8. Đọc và ghi file nhị phân.....	365

9. Đọc file một cách bất đồng bộ.....	367
10. Tìm file phù hợp một biểu thức wildcard.....	370
11. Kiểm tra hai file có trùng nhau hay không.....	371
12. Thao tác trên đường dẫn file.....	373
13. Xác định đường dẫn tương ứng với một file hay thư mục.....	374
14. Làm việc với đường dẫn tương đối.....	375
15. Tạo file tạm.....	376
16. Lấy dung lượng đĩa còn trống.....	377
17. Hiện thị các hộp thoại file.....	379
18. Sử dụng không gian lưu trữ riêng.....	382
19. Theo dõi hệ thống file để phát hiện thay đổi.....	384
20. Truy xuất cổng COM.....	386

Chương 10: CƠ SỞ DỮ LIỆU 389

1. Kết nối cơ sở dữ liệu.....	392
2. Sử dụng connection-pooling.....	394
3. Thực thi câu lệnh SQL hoặc thủ tục tồn trữ.....	397
4. Sử dụng thông số trong câu lệnh SQL hoặc thủ tục tồn trữ.....	400
5. Xử lý kết quả của truy vấn SQL bằng data-reader.....	403
6. Thu lấy tài liệu XML từ truy vấn SQL Server.....	407
7. Nhận biết tất cả các thể hiện SQL Server 2000 trên mạng.....	411
8. Đọc file Excel với ADO.NET.....	413
9. Sử dụng Data Form Wizard.....	415
10. Sử dụng Crystal Report Wizard.....	424

Chương 11: LẬP TRÌNH MẠNG 435

1. Download file thông qua HTTP.....	437
2. Download và xử lý file bằng stream.....	438
3. Lấy trang HTML từ một website có yêu cầu xác thực.....	440
4. Hiện thị trang web trong ứng dụng dựa-trên-Windows.....	442
5. Lấy địa chỉ IP của máy tính hiện hành.....	446
6. Phân giải tên miền thành địa chỉ IP.....	447
7. "Ping" một địa chỉ IP.....	448
8. Giao tiếp bằng TCP.....	452
9. Lấy địa chỉ IP của client từ kết nối socket.....	457
10. Thiết lập các tùy chọn socket.....	459
11. Tạo một TCP-server hỗ-trợ đa-tiểu-trình.....	460
12. Sử dụng TCP một cách bất đồng bộ.....	463
13. Giao tiếp bằng UDP.....	467
14. Gửi e-mail thông qua SMTP.....	470
15. Gửi và nhận e-mail với MAPI.....	471

Chương 12: DỊCH VỤ WEB XML VÀ REMOTING 474

1. Tránh viết mã cứng cho địa chỉ URL của dịch vụ Web XML.....	477
2. Sử dụng kỹ thuật response-caching trong dịch vụ Web XML.....	478
3. Sử dụng kỹ thuật data-caching trong dịch vụ Web XML.....	479

4. Tạo phương thức web hỗ trợ giao dịch	482
5. Thiết lập thông tin xác thực cho dịch vụ Web XML.....	485
6. Gọi bất đồng bộ một phương thức web.....	486
7. Tạo lớp khả-truy-xuất-từ-xa.....	488
8. Đăng ký tất cả các lớp khả-truy-xuất-từ-xa trong một assembly.....	494
9. Quản lý các đối tượng ở xa trong IIS.....	496
10. Phát sinh sự kiện trên kênh truy xuất từ xa	497
11. Kiểm soát thời gian sống của một đối tượng ở xa	502
12. Kiểm soát phiên bản của các đối tượng ở xa.....	504
13. Tạo phương thức một chiều với dịch vụ Web XML hay Remoting.....	506

Chương 13: BẢO MẬT **509**

1. Cho phép mã lệnh có-đô-tin-cây-một-phần sử dụng assembly tên mạnh của bạn.....	512
2. Vô hiệu bảo mật truy xuất mã lệnh.....	514
3. Vô hiệu việc kiểm tra quyền thực thi.....	516
4. Bảo đảm bộ thực thi cấp cho assembly một số quyền nào đó.....	517
5. Giới hạn các quyền được cấp cho assembly.....	519
6. Xem các yêu cầu quyền được tạo bởi một assembly.....	520
7. Xác định mã lệnh có quyền nào đó lúc thực thi hay không.....	522
8. Hạn chế ai đó thừa kế các lớp của bạn và chép đề các thành viên lớp.....	523
9. Kiểm tra chứng cứ của một assembly.....	525
10. Xử lý chứng cứ khi nạp một assembly	527
11. Xử lý bảo mật bộ thực thi bằng chứng cứ của miền ứng dụng	529
12. Xử lý bảo mật bộ thực thi bằng chính sách bảo mật của miền ứng dụng.....	531
13. Xác định người dùng hiện hành có là thành viên của một nhóm Windows nào đó hay không.....	535
14. Hạn chế những người dùng nào đó thực thi mã lệnh của bạn.....	538
15. Giả nhận người dùng Windows.....	543

Chương 14: MẬT MÃ **548**

1. Tạo số ngẫu nhiên.....	550
2. Tính mã băm của password.....	552
3. Tính mã băm của file.....	554
4. Kiểm tra mã băm.....	555
5. Bảo đảm tính toàn vẹn dữ liệu bằng mã băm có khóa.....	558
6. Bảo vệ file bằng phép mật hóa đối xứng.....	560
7. Truy lại khóa đối xứng từ password	566
8. Gửi một bí mật bằng phép mật hóa bất đối xứng.....	568
9. Lưu trữ khóa bất đối xứng một cách an toàn	574
10. Trao đổi khóa phiên đối xứng một cách an toàn.....	577

Chương 15: KHẢ NĂNG LIÊN TÁC MÃ LỆNH KHÔNG-ĐƯỢC-QUẢN-LÝ **584**

1. Gọi một hàm trong một DLL không-được-quản-lý.....	586
2. Lấy handle của một điều kiểm, cửa sổ, hoặc file.....	590

3. Gọi một hàm không-được-quản-lý có sử dụng cấu trúc.....	591
4. Gọi một hàm không-được-quản-lý có sử dụng callback.....	594
5. Lấy thông tin lỗi không-được-quản-lý.....	595
6. Sử dụng thành phần COM trong .NET-client.....	597
7. Giải phóng nhanh thành phần COM.....	600
8. Sử dụng thông số tùy chọn.....	600
9. Sử dụng điều khiển ActiveX trong .NET-client.....	602
10. Tạo thành phần .NET dùng cho COM-client.....	603

Chương 16: CÁC GIAO DIỆN VÀ MẪU THÔNG DỤNG 605

1. Hiện thực kiểu khả-tuần-tự-hóa (serializable type).....	607
2. Hiện thực kiểu khả-sao-chép (cloneable type).....	614
3. Hiện thực kiểu khả-so-sánh (comparable type).....	617
4. Hiện thực kiểu khả-liệt-kê (enumerable type).....	622
5. Hiện thực lớp khả-hủy (disposable class).....	629
6. Hiện thực kiểu khả-định-dạng (formattable type).....	633
7. Hiện thực lớp ngoại lệ tùy biến.....	636
8. Hiện thực đối số sự kiện tùy biến.....	640
9. Hiện thực mẫu Singleton.....	642
10. Hiện thực mẫu Observer.....	643

Chương 17: SỰ HÒA HỢP VỚI MÔI TRƯỜNG WINDOWS 651

1. Truy xuất thông tin môi trường.....	653
2. Lấy giá trị của một biến môi trường.....	657
3. Ghi một sự kiện vào nhật ký sự kiện Windows.....	658
4. Truy xuất Windows Registry.....	659
5. Tạo một dịch vụ Windows.....	663
6. Tạo một bộ cài đặt dịch vụ Windows.....	668
7. Tạo shortcut trên Desktop hay trong Start menu.....	671

PHỤ LỤC A: GIỚI THIỆU MỘT SỐ CÔNG CỤ .NET 676

A.1 Biên dịch các đoạn mã ngắn với Snippet Compiler	676
A.2 Xây dựng biểu thức chính quy với Regulator	678
A.3 Sinh mã với CodeSmith.....	679
A.4 Viết kiểm thử đơn vị với NUnit	681
A.5 Kiểm soát mã lệnh với FxCop	683
A.6 Khảo sát assembly với .NET Reflector	684
A.7 Lập tài liệu mã lệnh với NDoc.....	686
A.8 Tạo dựng giải pháp với NAnt.....	689
A.9 Chuyển đổi phiên bản ASP.NET với ASP.NET Version Switcher.....	691
A.10 Chuyển đổi phiên bản dự án với Visual Studio .NET Project Converter.....	692
A.11 Chuyển mã nguồn VB.NET sang C# với VB.NET to C# Converter.....	693
A.12 Chuyển mã nguồn C# sang VB.NET với Convert C# to VB.NET.....	693
A.13 Xây dựng website quản trị cơ sở dữ liệu với ASP.NET Maker 1.1.....	694

PHỤ LỤC B: THUẬT NGỮ ANH - VIỆT..... 697

TÀI LIỆU THAM KHẢO.....	705
-------------------------	-----



1

PHÁT TRIỂN ỨNG DỤNG

Chương này trình bày một số kiến thức nền tảng, cần thiết trong quá trình phát triển một ứng dụng C#. Các mục trong chương sẽ trình bày chi tiết các vấn đề sau đây:

- Xây dựng các ứng dụng *Console* và *Windows Form* (mục 1.1 và 1.2).
- Tạo và sử dụng đơn thể mã lệnh và thư viện mã lệnh (mục 1.3 và 1.4).
- Truy xuất đối số dòng lệnh từ bên trong ứng dụng (mục 1.5).
- Sử dụng các chỉ thị biên dịch để tùy biến việc biên dịch mã nguồn (mục 1.6).
- Truy xuất các phần tử chương trình (được xây dựng trong ngôn ngữ khác) có tên xung đột với các từ khóa C# (mục 1.7).
- Tạo và xác minh tên mạnh cho assembly (mục 1.8, 1.9, 1.10, và 1.11).
- Ký một assembly bằng chữ ký số *Microsoft Authenticode* (mục 1.12 và 1.13).
- Quản lý những assembly chia sẻ được lưu trữ trong *Global Assembly Cache* (mục 1.14).
- Ngăn người dùng dịch ngược assembly của bạn (mục 1.15).

 **Tất cả các công cụ được thảo luận trong chương này đều có trong *Microsoft .NET Framework* hoặc *.NET Framework SDK*.**

Các công cụ thuộc *Framework* nằm trong thư mục chính của phiên bản *Framework* mà bạn đang sử dụng (mặc định là `\WINDOWS\Microsoft.NET\Framework\v1.1.4322` nếu bạn sử dụng *.NET Framework version 1.1*). Quá trình cài đặt *.NET* sẽ tự động thêm thư mục này vào đường dẫn môi trường của hệ thống.

Các công cụ được cung cấp cùng với *SDK* nằm trong thư mục *Bin* của thư mục cài đặt *SDK* (mặc định là `\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`). Thư mục này không được thêm vào đường dẫn một cách tự động, vì vậy bạn phải tự thêm nó vào để dễ dàng truy xuất các công cụ này.

Hầu hết các công cụ trên đều hỗ trợ hai dạng đối số dòng lệnh: ngắn và dài. Chương này luôn trình bày dạng dài vì dễ hiểu hơn (nhưng bù lại bạn phải gõ nhiều hơn). Đối với dạng ngắn, bạn hãy tham khảo tài liệu tương ứng trong *.NET Framework SDK*.

1.

Tạo ứng dụng Console

?

Bạn muốn xây dựng một ứng dụng không cần giao diện người dùng đồ họa (*GUI*), thay vào đó hiển thị kết quả và đọc dữ liệu nhập từ dòng lệnh.

✂

Hiện thực một phương thức tĩnh có tên là `Main` dưới các dạng sau trong ít nhất một file mã nguồn:

- `public static void Main();`
- `public static void Main(string[] args);`
- `public static int Main();`

- `public static int Main(string[] args);`

Sử dụng đối số `/target:exe` khi biên dịch assembly của bạn bằng trình biên dịch C# (`csc.exe`).

Mặc định trình biên dịch C# sẽ xây dựng một ứng dụng *Console* trừ khi bạn chỉ định loại khác. Vì lý do này, không cần chỉ định `/target:exe`, nhưng thêm nó vào sẽ rõ ràng hơn, hữu ích khi tạo các kịch bản biên dịch sẽ được sử dụng bởi các ứng dụng khác hoặc sẽ được sử dụng lặp đi lặp lại trong một thời gian. Ví dụ sau minh họa một lớp có tên là `ConsoleUtils` (được định nghĩa trong file `ConsoleUtils.cs`):

```
using System;

public class ConsoleUtils {

    // Phương thức hiển thị lời nhắc và đọc đáp ứng từ console.
    public static string ReadString(string msg) {

        Console.Write(msg);

        return System.Console.ReadLine();

    }

    // Phương thức hiển thị thông điệp.
    public static void WriteString(string msg) {

        System.Console.WriteLine(msg);

    }

    // Phương thức Main dùng để thử nghiệm lớp ConsoleUtils.
    public static void Main() {

        // Yêu cầu người dùng nhập tên.
        string name = ReadString("Please enter your name : ");

        // Hiển thị thông điệp chào mừng.
        WriteString("Welcome to Microsoft .NET Framework, " + name);

    }

}
```


Để xây dựng lớp `ConsoleUtils` thành một ứng dụng *Console* có tên là *ConsoleUtils.exe*, sử dụng lệnh:

```
csc /target:exe ConsoleUtils.cs
```

Bạn có thể chạy file thực thi trực tiếp từ dòng lệnh. Khi chạy, phương thức `Main` của ứng dụng *ConsoleUtils.exe* yêu cầu bạn nhập tên và sau đó hiển thị thông điệp chào mừng như sau:

```
Please enter your name : Binh Phuong
```

```
Welcome to Microsoft .NET Framework, Binh Phuong
```

Thực tế, ứng dụng hiếm khi chỉ gồm một file mã nguồn. Ví dụ, lớp `HelloWorld` dưới đây sử dụng lớp `ConsoleUtils` để hiển thị thông điệp “*Hello, world*” lên màn hình (`HelloWorld` nằm trong file *HelloWorld.cs*).

```
public class HelloWorld {

    public static void Main() {

        ConsoleUtils.WriteString("Hello, world");

    }

}
```

Để xây dựng một ứng dụng *Console* gồm nhiều file mã nguồn, bạn phải chỉ định tất cả các file mã nguồn này trong đối số dòng lệnh. Ví dụ, lệnh sau đây xây dựng ứng dụng *MyFirstApp.exe* từ các file mã nguồn *HelloWorld.cs* và *ConsoleUtils.cs*:

```
csc /target:exe /main:HelloWorld /out:MyFirstApp.exe
    HelloWorld.cs ConsoleUtils.cs
```

Đối số `/out` chỉ định tên của file thực thi sẽ được tạo ra. Nếu không được chỉ định, tên của file thực thi sẽ là tên của file mã nguồn đầu tiên—trong ví dụ trên là *HelloWorld.cs*. Vì cả hai lớp `HelloWorld` và `ConsoleUtils` đều có phương thức `Main`, trình biên dịch không thể tự động quyết định đâu là điểm nhập cho file thực thi. Bạn phải sử dụng đối số `/main` để chỉ định tên của lớp chứa điểm nhập cho ứng dụng của bạn.

2.

Tạo ứng dụng dựa-trên-Windows



Bạn cần xây dựng một ứng dụng cung cấp giao diện người dùng đồ họa (GUI) dựa-trên-Windows Form.



Hiện thực một phương thức tĩnh `Main` trong ít nhất một file mã nguồn. Trong `Main`, tạo một thể hiện của một lớp thừa kế từ lớp `System.Windows.Forms.Form` (đây là form chính của ứng dụng). Truyền đối tượng này cho phương thức tĩnh `Run` của lớp `System.Windows.Forms.Application`. Sử dụng đối số `/target:winexe` khi biên dịch assembly của bạn bằng trình biên dịch C# (`csc.exe`).

Việc xây dựng một ứng dụng có giao diện người dùng đồ họa *Windows* đơn giản hoàn toàn khác xa việc phát triển một ứng dụng dựa-trên-Windows hoàn chỉnh. Tuy nhiên, bất kể viết

một ứng dụng đơn giản như *Hello World* hay viết phiên bản kế tiếp cho *Microsoft Word*, bạn cũng phải thực hiện những việc sau:

- Tạo một lớp thừa kế từ lớp `System.Windows.Forms.Form` cho mỗi form cần cho ứng dụng.
- Trong mỗi lớp form, khai báo các thành viên mô tả các điều kiểm trên form, ví dụ `Button`, `Label`, `ListBox`, `TextBox`. Các thành viên này nên được khai báo là `private` hoặc ít nhất cũng là `protected` để các phần tử khác của chương trình không truy xuất trực tiếp chúng được. Nếu muốn cho phép truy xuất các điều kiểm này, hiện thực các thành viên cần thiết trong lớp form để cung cấp việc truy xuất gián tiếp (kiểm soát được) đến các điều kiểm nằm trong.
- Trong lớp form, khai báo các phương thức thụ lý các sự kiện do các điều kiểm trên form sinh ra, chẳng hạn việc nhấp vào `Button`, việc nhấn phím khi một `TextBox` đang tích cực. Các phương thức này nên được khai báo là `private` hoặc `protected` và tuân theo mẫu sự kiện *.NET* chuẩn (sẽ được mô tả trong mục 16.10). Trong các phương thức này (hoặc trong các phương thức được gọi bởi các các phương thức này), bạn sẽ định nghĩa các chức năng của ứng dụng.
- Khai báo một phương thức khởi dựng cho lớp form để tạo các điều kiểm trên form và cấu hình trạng thái ban đầu của chúng (kích thước, màu, nội dung...). Phương thức khởi dựng này cũng nên liên kết các phương thức thụ lý sự kiện của lớp với các sự kiện tương ứng của mỗi điều kiểm.
- Khai báo phương thức tĩnh `Main`—thường là một phương thức của lớp tương ứng với form chính của ứng dụng. Phương thức này là điểm bắt đầu của ứng dụng và có các dạng như đã được đề cập ở mục 1.1. Trong phương thức `Main`, tạo một thể hiện của form chính và truyền nó cho phương thức tĩnh `Application.Run`. Phương thức `Run` hiển thị form chính và khởi chạy một vòng lặp thông điệp chuẩn trong tiểu trình hiện hành, chuyển các tác động từ người dùng (nhấn phím, nhấp chuột...) thành các sự kiện gửi đến ứng dụng.

Lớp `WelcomeForm` trong ví dụ dưới đây minh họa các kỹ thuật trên. Khi chạy, nó yêu cầu người dùng nhập vào tên rồi hiển thị một `MessageBox` chào mừng.

```
using System.Windows.Forms;
```

```
public class WelcomeForm : Form {
```

```
    // Các thành viên private giữ tham chiếu đến các điều kiểm.
```

```
    private Label label1;
```

```
    private TextBox textBox1;
```

```
    private Button button1;
```

```
    // Phương thức khởi dựng (tạo một thể hiện form
```

```
// và cấu hình các điều kiểm trên form).
public WelcomeForm() {

    // Tạo các điều kiểm trên form.
    this.labell = new Label();
    this.textBox1 = new TextBox();
    this.button1 = new Button();

    // Tạm hoãn layout logic của form trong khi
    // chúng ta cấu hình và bố trí các điều kiểm.
    this.SuspendLayout();

    // Cấu hình các Label (hiển thị yêu cầu).
    this.labell.Location = new System.Drawing.Point(16, 36);
    this.labell.Name = "labell";
    this.labell.Size = new System.Drawing.Size(128, 16);
    this.labell.TabIndex = 0;
    this.labell.Text = "Please enter your name:";

    // Cấu hình TextBox (nhận thông tin từ người dùng).
    this.textBox1.Location = new System.Drawing.Point(152, 32);
    this.textBox1.Name = "textBox1";
    this.textBox1.TabIndex = 1;
    this.textBox1.Text = "";

    // Cấu hình Buton (người dùng nhấn vào sau khi nhập tên).
    this.button1.Location = new System.Drawing.Point(109, 80);
    this.button1.Name = "button1";
    this.button1.TabIndex = 2;
    this.button1.Text = "Enter";
    this.button1.Click += new System.EventHandler(this.button1_Click);

    // Cấu hình WelcomeForm và thêm các điều kiểm.
    this.ClientSize = new System.Drawing.Size(292, 126);
    this.Controls.Add(this.button1);
    this.Controls.Add(this.textBox1);
    this.Controls.Add(this.labell);
    this.Name = "form1";
```

```
this.Text = "Microsoft .NET Framework";

// Phục hồi layout logic của form ngay khi
// tất cả các điều kiểm đã được cấu hình.
this.ResumeLayout(false);
}

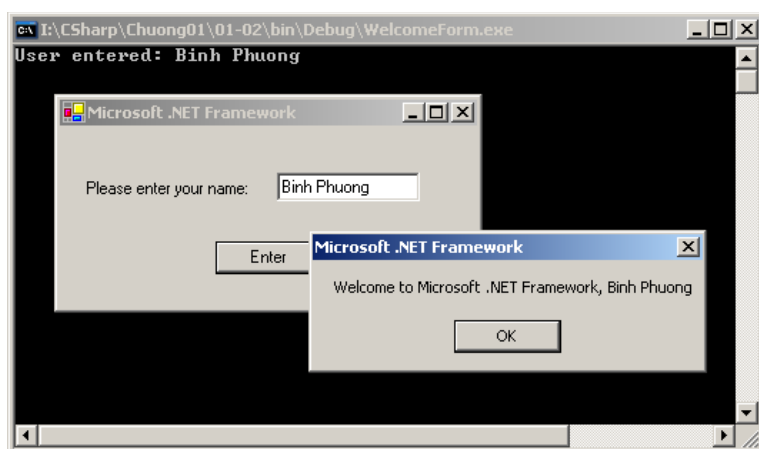
// Điểm nhập của ứng dụng (tạo một thẻ hiện form, chạy vòng lặp
// thông điệp chuẩn trong tiêu trình hiện hành - vòng lặp chuyển
// các tác động từ người dùng thành các sự kiện đến ứng dụng).
public static void Main() {

    Application.Run(new WelcomeForm());
}

// Phương thức thụ lý sự kiện
// (được gọi khi người dùng nhấn vào nút Enter).
private void button1_Click(object sender, System.EventArgs e) {

    // Ghi ra Console.
    System.Console.WriteLine("User entered: " + textBox1.Text);

    // Hiển thị lời chào trong MessageBox.
    MessageBox.Show("Welcome to Microsoft .NET Framework, "
        + textBox1.Text, "Microsoft .NET Framework");
}
}
```



Hình 1.1 Một ứng dụng Windows Form đơn giản

Để xây dựng lớp `WelcomeForm` (trong file `WelcomeForm.cs`) thành một ứng dụng, sử dụng lệnh:

```
csc /target:winexe WelcomeForm.cs
```

Đổi số `/target:winexe` báo cho trình biên dịch biết đây là ứng dụng dựa-trên-*Windows*. Do đó, trình biên dịch sẽ xây dựng file thực thi sao cho không có cửa sổ *Console* nào được tạo ra khi bạn chạy ứng dụng. Nếu bạn sử dụng `/target:exe` khi xây dựng một ứng dụng *Windows Form* thay cho `/target:winexe` thì ứng dụng vẫn làm việc tốt, nhưng sẽ tạo ra một cửa sổ *Console* khi chạy. Mặc dù điều này không được ưa chuộng trong một ứng dụng hoàn chỉnh, cửa sổ *Console* vẫn hữu ích nếu bạn cần ghi ra các thông tin gỡ rối hoặc đăng nhập khi đang phát triển và thử nghiệm một ứng dụng *Windows Form*. Bạn có thể ghi ra *Console* bằng phương thức `Write` và `WriteLine` của lớp `System.Console`.

Ứng dụng `WelcomeForm.exe` trong hình 1.1 hiển thị lời chào người dùng có tên là *Binh Phuong*. Phiên bản này của ứng dụng được xây dựng bằng đổi số `/target:exe`, nên có cửa sổ *Console* để hiển thị kết quả của dòng lệnh `Console.WriteLine` trong phương thức thụ lý sự kiện `button1_Click`.



Việc xây dựng một ứng dụng *GUI* đồ sộ thường tốn nhiều thời gian do phải tạo đối tượng, cấu hình và liên kết nhiều form và điều khiển. Nhưng may mắn là *Microsoft Visual Studio .NET* tự động hóa hầu hết các hoạt động này. Nếu không có công cụ như *Microsoft Visual Studio .NET* thì việc xây dựng một ứng dụng đồ họa đồ sộ sẽ rất lâu, nhàm chán và dễ sinh ra lỗi.

3.

Tạo và sử dụng module



Bạn cần thực hiện các công việc sau:

- Tăng hiệu quả thực thi và sử dụng bộ nhớ của ứng dụng bằng cách bảo đảm rằng bộ thực thi nạp các kiểu ít được sử dụng chỉ khi nào cần thiết.

- Biên dịch các kiểu được viết trong *C#* thành một dạng có thể sử dụng lại được trong các ngôn ngữ *.NET* khác.
- Sử dụng các kiểu được phát triển bằng một ngôn ngữ khác bên trong ứng dụng *C#* của bạn.



Sử dụng đối số `/target:module` (của trình biên dịch *C#*) để xây dựng mã nguồn *C#* của bạn thành một module. Sử dụng đối số `/addmodule` để kết hợp các module hiện có vào assembly của bạn.

Module là các khối cơ bản tạo dựng nên các assembly *.NET*. Module bao gồm một file đơn chứa:

- Mã ngôn ngữ trung gian (*Microsoft Intermediate Language—MSIL*): Được tạo từ mã nguồn *C#* trong quá trình biên dịch.
- Siêu dữ liệu (*metadata*): Mô tả các kiểu nằm trong module.
- Các tài nguyên (*resource*): Chẳng hạn icon và string table, được sử dụng bởi các kiểu trong module.

Assembly gồm một hay nhiều module và một manifest. Khi chỉ có một module, module và manifest thường được xây dựng thành một file cho thuận tiện. Khi có nhiều module, assembly là một nhóm luận lý của nhiều file được triển khai như một thể thống nhất. Trong trường hợp này, manifest có thể nằm trong một file riêng hay chung với một trong các module.

Việc xây dựng một assembly từ nhiều module gây khó khăn cho việc quản lý và triển khai assembly; nhưng trong một số trường hợp, cách này có nhiều lợi ích, bao gồm:

- Bộ thực thi sẽ chỉ nạp một module khi các kiểu định nghĩa trong module này được yêu cầu. Do đó, khi có một tập các kiểu mà ứng dụng ít khi dùng, bạn có thể đặt chúng trong một module riêng mà bộ thực thi chỉ nạp khi cần. Việc này có các lợi ích sau:
 - Tăng hiệu quả thực thi, đặc biệt khi ứng dụng được nạp qua mạng.
 - Giảm thiểu nhu cầu sử dụng bộ nhớ.
- Khả năng sử dụng nhiều ngôn ngữ khác nhau để viết các ứng dụng chạy trên bộ thực thi ngôn ngữ chung (*Common Language Runtime—CLR*) là một thế mạnh của *.NET Framework*. Tuy nhiên, trình biên dịch *C#* không thể biên dịch mã nguồn được viết bằng *Microsoft Visual Basic .NET* hay *COBOL .NET* trong assembly của bạn. Bạn phải sử dụng trình biên dịch của ngôn ngữ đó biên dịch mã nguồn thành *MSIL* theo một cấu trúc mà trình biên dịch *C#* có thể hiểu được—đó là module. Tương tự, nếu muốn lập trình viên của các ngôn ngữ khác sử dụng các kiểu được phát triển bằng *C#*, bạn phải xây dựng chúng thành một module.

Để biên dịch file nguồn *ConsoleUtils.cs* thành một module, sử dụng lệnh:

```
csc /target:module ConsoleUtils.cs
```

Lệnh này sẽ cho kết quả là một file có tên là *ConsoleUtils.netmodule*. Phần mở rộng *netmodule* là phần mở rộng mặc định cho module, và tên file trùng với tên file nguồn *C#*.

Bạn cũng có thể xây dựng một module từ nhiều file nguồn, cho kết quả là một file (module) chứa *MSIL* và siêu dữ liệu cho các kiểu chứa trong tất cả file nguồn. Ví dụ, lệnh:

```
csc /target:module ConsoleUtils.cs WindowsUtils.cs
```

biên dịch hai file nguồn *ConsoleUtils.cs* và *WindowsUtils.cs* thành một module có tên là *ConsoleUtils.netmodule*.

Tên của module được đặt theo tên file nguồn đầu tiên trừ khi bạn chỉ định cụ thể bằng đối số */out*. Ví dụ, lệnh:

```
csc /target:module /out:Utilities.netmodule
    ConsoleUtils.cs WindowsUtils.cs
```

sẽ cho kết quả là file *Utilities.netmodule*.

Để xây dựng một assembly gồm nhiều module, sử dụng đối số */addmodule*. Ví dụ, để xây dựng file thực thi *MyFirstApp.exe* từ hai module: *WindowsUtils.netmodule* và *ConsoleUtils.netmodule* và hai file nguồn: *SourceOne.cs* và *SourceTwo.cs*, sử dụng lệnh:

```
csc /out:MyFirstApp.exe /target:exe
    /addmodule:WindowsUtils.netmodule,ConsoleUtils.netmodule
    SourceOne.cs SourceTwo.cs
```

Lệnh này sẽ cho kết quả là một assembly gồm các file sau:

- *MyFirstApp.exe*: Chứa manifest cũng như *MSIL* cho các kiểu được khai báo trong hai file nguồn *SourceOne.cs* và *SourceTwo.cs*.
- *ConsoleUtils.netmodule* và *WindowsUtils.netmodule*: Giờ đây là một phần của assembly nhưng không thay đổi sau khi biên dịch. (Nếu bạn chạy *MyFirstApp.exe* mà không có các file *netmodule*, ngoại lệ *System.IO.FileNotFoundException* sẽ bị ném).

4.

Tạo và sử dụng thư viện



Bạn cần xây dựng một tập các chức năng thành một thư viện để nó có thể được tham chiếu và tái sử dụng bởi nhiều ứng dụng.



Để tạo thư viện, sử dụng đối số */target:library* khi biên dịch assembly của bạn bằng trình biên dịch C# (*csc.exe*). Để tham chiếu thư viện, sử dụng đối số */reference* và chỉ định tên của thư viện khi biên dịch ứng dụng.

Mục 1.1 minh họa cách xây dựng ứng dụng *MyFirstApp.exe* từ hai file mã nguồn *ConsoleUtils.cs* và *HelloWorld.cs*. File *ConsoleUtils.cs* chứa lớp *ConsoleUtils*, cung cấp các phương thức đơn giản hóa sự tương tác với *Console*. Các chức năng này của lớp *ConsoleUtils* cũng có thể hữu ích cho các ứng dụng khác. Để sử dụng lại lớp này, thay vì gộp cả mã nguồn của nó vào mỗi ứng dụng, bạn có thể xây dựng nó thành một thư viện, khiến các chức năng này có thể truy xuất được bởi nhiều ứng dụng.

Để xây dựng file *ConsoleUtils.cs* thành một thư viện, sử dụng lệnh:

```
csc /target:library ConsoleUtils.cs
```

Lệnh này sinh ra một file thư viện có tên là *ConsoleUtils.dll*.

Để tạo một thư viện từ nhiều file mã nguồn, liệt kê tên các file này ở cuối dòng lệnh. Bạn có thể sử dụng đối số `/out` để chỉ định tên thư viện, nếu không, tên thư viện được đặt theo tên của file mã nguồn đầu tiên. Ví dụ, để tạo thư viện *MyFirstLibrary.dll* từ hai file mã nguồn *ConsoleUtils.cs* và *WindowsUtils.cs*, sử dụng lệnh:

```
csc /out:MyFirstLibrary.dll /target:library
    ConsoleUtils.cs WindowsUtils.cs
```

Trước khi phân phối thư viện cho người khác sử dụng, bạn nên tạo tên mạnh (*strong-name*) để không ai có thể chỉnh sửa assembly của bạn. Việc đặt tên mạnh cho thư viện còn cho phép người khác cài đặt nó vào *Global Assembly Cache*, giúp việc tái sử dụng dễ dàng hơn (xem mục 1.9 về cách đặt tên mạnh cho thư viện của bạn và mục 1.14 về cách cài đặt một thư viện có tên mạnh vào *Global Assembly Cache*). Ngoài ra, bạn có thể đánh dấu thư viện của bạn với chữ ký *Authenticode* để người dùng biết bạn là tác giả của thư viện (xem mục 1.12 về cách đánh dấu thư viện với *Authenticode*).

Để biên dịch một assembly có sử dụng các kiểu được khai báo trong các thư viện khác, bạn phải báo cho trình biên dịch biết cần tham chiếu đến thư viện nào bằng đối số `/reference`. Ví dụ, để biên dịch file *HelloWorld.cs* (trong mục 1.1) trong trường hợp lớp *ConsoleUtils* nằm trong thư viện *ConsoleUtils.dll*, sử dụng lệnh:

```
csc /reference:ConsoleUtils.dll HelloWorld.cs
```

Bạn cần chú ý ba điểm sau:

- Nếu tham chiếu nhiều hơn một thư viện, bạn cần phân cách tên các thư viện bằng dấu phẩy hoặc chấm phẩy, nhưng không sử dụng khoảng trắng. Ví dụ:
`/reference:ConsoleUtils.dll,WindowsUtils.dll`
- Nếu thư viện không nằm cùng thư mục với file mã nguồn, bạn cần sử dụng đối số `/lib` để chỉ định thư mục chứa thư viện. Ví dụ:
`/lib:c:\CommonLibraries,c:\Dev\ThirdPartyLibs`
- Nếu thư viện cần tham chiếu là một assembly gồm nhiều file, bạn cần tham chiếu file có chứa manifest (xem thông tin về assembly gồm nhiều file trong mục 1.3).

5.

Truy xuất các đối số dòng lệnh

?

Bạn cần truy xuất các đối số được chỉ định trên dòng lệnh khi thực thi ứng dụng.

✕

Sử dụng một dạng của phương thức `Main`, trong đó nhận đối số dòng lệnh dưới dạng một mảng chuỗi. Ngoài ra, có thể truy xuất đối số dòng lệnh từ bất cứ đâu trong mã nguồn của bạn bằng các thành viên tĩnh của lớp `System.Environment`.

Khai báo phương thức `Main` thuộc một trong các dạng sau để truy xuất đối số dòng lệnh dưới dạng một mảng chuỗi:

- `public static void Main(string[] args) {}`

- `public static int Main(string[] args) {}`

Khi chạy, đối số `args` sẽ chứa một chuỗi cho mỗi giá trị được nhập trên dòng lệnh và nằm sau tên ứng dụng. Phương thức `Main` trong ví dụ dưới đây sẽ duyệt qua mỗi đối số dòng lệnh được truyền cho nó và hiển thị chúng ra cửa sổ *Console*:

```
public class CmdLineArgExample {

    public static void Main(string[] args) {

        // Duyệt qua các đối số dòng lệnh.
        foreach (string s in args) {
            System.Console.WriteLine(s);
        }
    }
}
```

Khi thực thi *CmdLineArgExample* với lệnh:

```
CmdLineArgExample "one \"two\"    three" four 'five    six'
```

ứng dụng sẽ tạo ra kết xuất như sau:

```
one "two"    three
four
'five
six'
```

Chú ý rằng, khác với *C* và *C++*, tên của ứng dụng không nằm trong mảng chứa các đối số. Tất cả ký tự nằm trong dấu nháy kép (") được xem như một đối số, nhưng dấu nháy đơn (') chỉ được xem như ký tự bình thường. Nếu muốn sử dụng dấu nháy kép trong đối số, đặt ký tự vạch ngược (\) trước nó. Tất cả các khoảng trắng đều bị bỏ qua trừ khi chúng nằm trong dấu nháy kép.

Nếu muốn truy xuất đối số dòng lệnh ở nơi khác (không phải trong phương thức `Main`), bạn cần xử lý các đối số dòng lệnh trong phương thức `Main` và lưu trữ chúng để sử dụng sau này.

Ngoài ra, bạn có thể sử dụng lớp `System.Environment`, lớp này cung cấp hai thành viên tĩnh trả về thông tin dòng lệnh: `CommandLine` và `GetCommandLineArgs`.

- Thuộc tính `CommandLine` trả về một chuỗi chứa toàn bộ dòng lệnh. Tùy thuộc vào hệ điều hành ứng dụng đang chạy mà thông tin đường dẫn có đứng trước tên ứng dụng hay không. Các hệ điều hành *Windows NT 4.0*, *Windows 2000*, và *Windows XP* không chứa thông tin đường dẫn, trong khi *Windows 98* và *Windows ME* thì lại chứa.
- Phương thức `GetCommandLineArgs` trả về một mảng chuỗi chứa các đối số dòng lệnh. Mảng này có thể được xử lý giống như mảng được truyền cho phương thức `Main`, tuy nhiên phần tử đầu tiên của mảng này là tên ứng dụng.

6.

Chọn biên dịch một khối mã vào file thực thi

?

Bạn cần chọn một số phần mã nguồn sẽ được biên dịch trong file thực thi.

✕

Sử dụng các chỉ thị tiền xử lý `#if`, `#elif`, `#else`, và `#endif` để chỉ định khối mã nào sẽ được biên dịch trong file thực thi. Sử dụng đặc tính `System.Diagnostics.ConditionalAttribute` để chỉ định các phương thức mà sẽ chỉ được gọi tùy theo điều kiện. Điều khiển việc chọn các khối mã bằng các chỉ thị `#define` và `#undef` trong mã nguồn, hoặc sử dụng đối số `/define` khi chạy trình biên dịch `C#`.

Nếu muốn ứng dụng của bạn hoạt động khác nhau tùy vào các yếu tố như nền hoặc môi trường mà ứng dụng chạy, bạn có thể kiểm tra điều kiện khi chạy bên trong mã nguồn và kích hoạt các hoạt động cần thiết. Tuy nhiên, cách này làm mã nguồn lớn lên và ảnh hưởng đến hiệu năng. Một cách tiếp cận khác là xây dựng nhiều phiên bản của ứng dụng để hỗ trợ các nền và môi trường khác nhau. Mặc dù cách này khắc phục được các vấn đề về độ lớn của mã nguồn và việc giảm hiệu năng, nhưng nó không phải là giải pháp tốt khi phải giữ mã nguồn khác nhau cho mỗi phiên bản. Vì vậy, `C#` cung cấp các tính năng cho phép bạn xây dựng các phiên bản tùy biến của ứng dụng chỉ từ một mã nguồn.

Các chỉ thị tiền xử lý cho phép bạn chỉ định các khối mã sẽ được biên dịch vào file thực thi chỉ nếu các ký hiệu cụ thể được định nghĩa lúc biên dịch. Các ký hiệu hoạt động như các “công tắc” on/off, chúng không có giá trị mà chỉ là “đã được định nghĩa” hay “chưa được định nghĩa”. Để định nghĩa một ký hiệu, bạn có thể sử dụng chỉ thị `#define` trong mã nguồn hoặc sử dụng đối số trình biên dịch `/define`. Ký hiệu được định nghĩa bằng `#define` có tác dụng đến cuối file định nghĩa nó. Ký hiệu được định nghĩa bằng `/define` có tác dụng trong tất cả các file đang được biên dịch. Để bỏ một ký hiệu đã định nghĩa bằng `/define`, `C#` cung cấp chỉ thị `#undef`, hữu ích khi bạn muốn bảo đảm một ký hiệu không được định nghĩa trong các file nguồn cụ thể. Các chỉ thị `#define` và `#undef` phải nằm ngay đầu file mã nguồn, trên cả các chỉ thị `using`. Các ký hiệu có phân biệt chữ hoa-thường.

Trong ví dụ sau, biến `platformName` được gán giá trị tùy vào các ký hiệu `winXP`, `win2000`, `winNT`, hoặc `win98` có được định nghĩa hay không. Phần đầu của mã nguồn định nghĩa các ký hiệu `win2000` và `released` (không được sử dụng trong ví dụ này), và bỏ ký hiệu `win98` trong trường hợp nó được định nghĩa trên dòng lệnh trình biên dịch.

```
#define win2000
#define release
#undef win98
```

```
using System;
```

```
public class ConditionalExample {
```

```
    public static void Main() {
```

```
// Khai báo chuỗi chứa tên của nền.
string platformName;

#if winXP      // Biên dịch cho Windows XP
    platformName = "Microsoft Windows XP";
#elif win2000 // Biên dịch cho Windows 2000
    platformName = "Microsoft Windows 2000";
#elif winNT    // Biên dịch cho Windows NT
    platformName = "Microsoft Windows NT";
#elif win98    // Biên dịch cho Windows 98
    platformName = "Microsoft Windows 98";
#else         // Nền không được nhận biết
    platformName = "Unknown";
#endif

Console.WriteLine(platformName);
}
}
```

Để xây dựng lớp `ConditionalExample` (chứa trong file *ConditionalExample.cs*) và định nghĩa các ký hiệu `winXP` và `DEBUG` (không được sử dụng trong ví dụ này), sử dụng lệnh:

```
csc /define:winXP;DEBUG ConditionalExample.cs
```

Cấu trúc `#if .. #endif` đánh giá các mệnh đề `#if` và `#elif` chỉ đến khi tìm thấy một mệnh đề đúng, nghĩa là nếu có nhiều ký hiệu được định nghĩa (chẳng hạn, `winXP` và `win2000`), thứ tự các mệnh đề là quan trọng. Trình biên dịch chỉ biên dịch đoạn mã nằm trong mệnh đề đúng. Nếu không có mệnh đề nào đúng, trình biên dịch sẽ biên dịch đoạn mã nằm trong mệnh đề `#else`.

Bạn cũng có thể sử dụng các toán tử luận lý để biên dịch có điều kiện dựa trên nhiều ký hiệu. Bảng 1.1 tóm tắt các toán tử được hỗ trợ.

Bảng 1.1 Các toán tử luận lý được hỗ trợ bởi chỉ thị `#if .. #endif`

Toán tử	Ví dụ	Mô tả
<code>==</code>	<code>#if winXP == true</code>	Bằng. Đúng nếu <code>winXP</code> được định nghĩa. Tương đương với <code>#if winXP</code> .
<code>!=</code>	<code>#if winXP != true</code>	Không bằng. Đúng nếu <code>winXP</code> không được định nghĩa. Tương đương với <code>#if !winXP</code> .
<code>&&</code>	<code>#if winXP && release</code>	Phép <i>AND</i> luận lý. Đúng nếu <code>winXP</code> và <code>release</code> được định nghĩa.

	<code>#if winXP release</code>	Phép <i>OR</i> luận lý. Đúng nếu winXP hoặc release được định nghĩa.
()	<code>#if (winXP win2000) && release</code>	Dấu ngoặc đơn cho phép nhóm các biểu thức. Đúng nếu winXP hoặc win2000 được định nghĩa, đồng thời release cũng được định nghĩa.



Bạn không nên lạm dụng các chỉ thị biên dịch có điều kiện và không nên viết các biểu thức điều kiện quá phức tạp; nếu không, mã nguồn của bạn sẽ trở nên dễ nhầm lẫn và khó quản lý—đặc biệt khi dự án của bạn càng lớn.

Một cách khác không linh hoạt nhưng hay hơn chỉ thị tiền xử lý `#if` là sử dụng đặc tính `System.Diagnostics.ConditionalAttribute`. Nếu bạn áp dụng `ConditionalAttribute` cho một phương thức, trình biên dịch sẽ bỏ qua mọi lời gọi phương thức đó nếu ký hiệu do `ConditionalAttribute` chỉ định không được định nghĩa tại điểm gọi. Trong đoạn mã sau, `ConditionalAttribute` xác định rằng phương thức `DumpState` chỉ được biên dịch vào file thực thi nếu ký hiệu `DEBUG` được định nghĩa khi biên dịch.

```
[System.Diagnostics.Conditional("DEBUG")]
public static void DumpState() {//...}
```

Việc sử dụng `ConditionalAttribute` giúp đặt các điều kiện gọi một phương thức tại nơi khai báo nó mà không cần các chỉ thị `#if`. Tuy nhiên, bởi vì trình biên dịch thật sự bỏ qua các lời gọi phương thức, nên mã của bạn không thể phụ thuộc vào các giá trị trả về từ phương thức. Điều này có nghĩa là bạn có thể áp dụng `ConditionalAttribute` chỉ với các phương thức trả về `void`.

Bạn có thể áp dụng nhiều thể hiện `ConditionalAttribute` cho một phương thức, tương đương với phép *OR* luận lý. Các lời gọi phương thức `DumpState` dưới đây chỉ được biên dịch nếu `DEBUG` hoặc `TEST` được định nghĩa.

```
[System.Diagnostics.Conditional("DEBUG")]
[System.Diagnostics.Conditional("TEST")]
public static void DumpState() {//...}
```

Việc thực hiện phép *AND* luận lý cần sử dụng phương thức điều kiện trung gian, khiến cho mã trở nên quá phức tạp, khó hiểu và khó bảo trì. Ví dụ dưới đây cần phương thức trung gian `DumpState2` để định nghĩa cả hai ký hiệu `DEBUG` và `TEST`.

```
[System.Diagnostics.Conditional("DEBUG")]
public static void DumpState() {
    DumpState2();
}

[System.Diagnostics.Conditional("TEST")]
public static void DumpState2() {//...}
```



Các lớp `Debug` và `Trace` thuộc không gian tên `System.Diagnostics` sử dụng đặc tính `ConditionalAttribute` trong nhiều phương thức của chúng. Các phương thức của lớp `Debug` tùy thuộc vào việc định nghĩa ký hiệu `DEBUG`, còn các phương thức của lớp `Trace` tùy thuộc vào việc định nghĩa ký hiệu `TRACE`.

7.

Truy xuất một phần tử chương trình có tên trùng với một từ khóa



Bạn cần truy xuất một thành viên của một kiểu, nhưng tên kiểu hoặc tên thành viên này trùng với một từ khóa của `C#`.



Đặt ký hiệu `@` vào trước các tên trùng với từ khóa.

`.NET Framework` cho phép bạn sử dụng các thành phần phần mềm (*software component*) được phát triển bằng các ngôn ngữ `.NET` khác bên trong ứng dụng `C#` của bạn. Mỗi ngôn ngữ đều có một tập từ khóa (hoặc từ dành riêng) cho nó và có các hạn chế khác nhau đối với các tên mà lập trình viên có thể gán cho các phần tử chương trình như kiểu, thành viên, và biến. Do đó, có khả năng một thành phần được phát triển trong một ngôn ngữ khác tình cờ sử dụng một từ khóa của `C#` để đặt tên cho một phần tử nào đó. Ký hiệu `@` cho phép bạn sử dụng một từ khóa của `C#` làm định danh và khắc phục việc đụng độ tên. Đoạn mã sau tạo một đối tượng kiểu `operator` và thiết lập thuộc tính `volatile` của nó là `true` (cả `operator` và `volatile` đều là từ khóa của `C#`):

```
// Tạo đối tượng operator.
@operator Operator1 = new @operator();

// Thiết lập thuộc tính volatile của operator.
Operator1.@volatile = true;
```

8.

Tạo và quản lý cặp khóa tên mạnh



Bạn cần tạo một cặp khóa công khai và khóa riêng (*public key* và *private key*) để gán tên mạnh cho assembly.



Sử dụng công cụ *Strong Name* (`sn.exe`) để tạo cặp khóa và lưu trữ chúng trong một file hoặc trong một kho chứa khóa *Cryptographic Service Provider*.



Cryptographic Service Provider (*CSP*) là một phần tử của *Win32 CryptoAPI*, cung cấp các dịch vụ như mật hóa, giải mật hóa và tạo chữ ký số. *CSP* còn cung cấp các tiện ích cho kho chứa khóa (*key container*) như sử dụng giải thuật mật hóa mạnh và các biện pháp bảo mật của hệ điều hành để bảo vệ nội dung của kho chứa khóa. *CSP* và *CryptoAPI* không được đề cập đầy đủ trong quyển sách này, bạn hãy tham khảo thêm trong tài liệu *SDK*.

Để tạo một cặp khóa mới và lưu trữ chúng trong file có tên là `MyKey.snk`, thực thi lệnh `sn -k MyKey.snk` (phần mở rộng `.snk` thường được sử dụng cho các file chứa khóa tên mạnh). File

được tạo ra chứa cả khóa công khai và khóa riêng. Bạn có thể sử dụng lệnh `sn -tp MyKey.snk` để xem khóa công khai, lệnh này cho kết xuất như sau:

```
Microsoft (R) .NET Framework Strong Name Utility Version 1.1.4322.573
```

```
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.
```

```
Public key is
```

```
07020000002400005253413200040000010001008bb302ef9180bf717ace00d570dd649821f24ed578
fdccf1bc4017308659c126570204bc4010fdd1907577df1c2292349d9c2de33e49bd991a0a5bc9b69e
5fd95bafad658a57b8236c5bd9a43be022a20a52c2bd8145448332d5f85e9ca641c26a4036165f2f35
3942b643b10db46c82d6d77bbc210d5a7c5aca84d7acb52cc1654759c62aa34988...
```

```
Public key token is f7241505b81b5ddc
```

Token của khóa công khai là 8 byte cuối của mã băm được tính ra từ khóa công khai. Vì khóa công khai quá dài nên *.NET* sử dụng token cho mục đích hiển thị, và là một cơ chế ngăn gọn cho các assembly khác tham chiếu khóa công khai (chương 14 sẽ thảo luận tổng quát về mã băm).

Như tên gọi của nó, khóa công khai (hoặc token của khóa công khai) không cần được giữ bí mật. Khi bạn tạo tên mạnh cho assembly (được thảo luận trong mục 1.9), trình biên dịch sẽ sử dụng khóa riêng để tạo một chữ ký số (một mã băm đã-được-mật-hóa) của assembly manifest. Trình biên dịch nhúng chữ ký số và khóa công khai vào assembly để người dùng có thể kiểm tra chữ ký số.

Việc giữ bí mật khóa riêng là cần thiết vì người truy xuất vào khóa riêng của bạn có thể thay đổi assembly và tạo một tên mạnh mới—khiến cho khách hàng của bạn không biết mã nguồn đã bị sửa đổi. Không có cơ chế nào để loại bỏ các khóa tên mạnh đã bị tổn hại. Nếu khóa riêng bị tổn hại, bạn phải tạo khóa mới và phân phối phiên bản mới của assembly (được đặt tên mạnh bằng các khóa mới). Bạn cũng cần thông báo cho khách hàng biết là khóa đã bị tổn hại và họ nên sử dụng phiên bản nào—trong trường hợp này, bạn bị mất cả tiền bạc và uy tín. Có nhiều cách để bảo vệ khóa riêng của bạn; sử dụng cách nào là tùy vào các yếu tố như:

- Cấu trúc và tầm cỡ của tổ chức.
- Quá trình phát triển và phân phối ứng dụng.
- Phần mềm và phần cứng hiện có.
- Yêu cầu của khách hàng.



Thông thường, một nhóm nhỏ các cá nhân đáng tin cậy (được gọi là *signing authority*) sẽ có trách nhiệm đảm bảo an toàn cho các khóa tên mạnh của công ty và ký mọi assembly trước khi chúng được phân phối. Khả năng trì hoãn ký assembly (sẽ được thảo luận ở mục 1.11) tạo điều kiện thuận lợi cho việc ứng dụng mô hình này và tránh được việc bạn phải phân phối khóa riêng cho mọi thành viên của nhóm phát triển.

Công cụ *Strong Name* còn cung cấp tính năng sử dụng kho chứa khóa *CSP* để đơn giản hóa việc bảo mật các khóa tên mạnh. Một khi đã tạo một cặp khóa trong một file, bạn có thể cài đặt các khóa này vào kho chứa khóa *CSP* và xóa file đi. Ví dụ, để lưu trữ cặp khóa nằm trong file *MyKey.snk* vào một kho chứa khóa *CSP* có tên là *StrongNameKeys*, sử dụng lệnh `sn -i MyKeys.snk StrongNameKeys` (mục 1.9 sẽ giải thích cách sử dụng các khóa tên mạnh được lưu trữ trong một kho chứa khóa *CSP*).

Một khía cạnh quan trọng của kho chứa khóa *CSP* là có các kho chứa khóa dựa-theo-người-dùng và có các kho chứa khóa dựa-theo-máy. Cơ chế bảo mật của *Windows* bảo đảm người dùng chỉ truy xuất được kho chứa khóa dựa-theo-người-dùng của chính họ. Tuy nhiên, bất kỳ người dùng nào của máy đều có thể truy xuất kho chứa khóa dựa-theo-máy.

Theo mặc định, công cụ *Strong Name* sử dụng kho chứa khóa dựa-theo-máy, nghĩa là mọi người đăng nhập vào máy và biết tên của kho chứa khóa đều có thể ký một assembly bằng các khóa tên mạnh của bạn. Để công cụ *Strong Name* sử dụng kho chứa khóa dựa-theo-người-dùng, sử dụng lệnh `sn -m n`; khi muốn trở lại kho chứa khóa dựa-theo-máy, sử dụng lệnh `sn -m y`. Lệnh `sn -m` sẽ cho biết công cụ *Strong Name* hiện được cấu hình là sử dụng kho chứa khóa dựa-theo-người-dùng hay dựa-theo-máy.

Để xóa các khóa tên mạnh từ kho *StrongNameKeys* (cũng như xóa cả kho này), sử dụng lệnh `sn -d StrongNameKeys`.

9.

Tạo tên mạnh cho assembly



Bạn cần tạo tên mạnh cho một assembly để nó:

- Có một định danh duy nhất, cho phép gán các quyền cụ thể vào assembly khi cấu hình *Code Access Security Policy* (chính sách bảo mật cho việc truy xuất mã lệnh).
- Không thể bị sửa đổi và sau đó mạo nhận là nguyên bản.
- Hỗ trợ việc đánh số phiên bản và các chính sách về phiên bản (*version policy*).
- Có thể được chia sẻ trong nhiều ứng dụng, và được cài đặt trong *Global Assembly Cache (GAC)*.



Sử dụng các đặc tính (*attribute*) mức-assembly để chỉ định nơi chứa cặp khóa tên mạnh, và có thể chỉ định thêm số phiên bản và thông tin bản địa cho assembly. Trình biên dịch sẽ tạo tên mạnh cho assembly trong quá trình xây dựng.

Để tạo tên mạnh cho một assembly bằng trình biên dịch *C#*, bạn cần các yếu tố sau:

- Một cặp khóa tên mạnh nằm trong một file hoặc một kho chứa khóa *CSP* (xem mục 1.8 về cách tạo cặp khóa tên mạnh).
- Sử dụng các đặc tính mức-assembly để chỉ định nơi trình biên dịch có thể tìm thấy cặp khóa tên mạnh đó.
 - Nếu cặp khóa nằm trong một file, áp dụng đặc tính `System.Reflection.AssemblyKeyFileAttribute` cho assembly và chỉ định tên file chứa các khóa.

- Nếu cặp khóa nằm trong một kho chứa khóa *CSP*, áp dụng đặc tính `System.Reflection.AssemblyKeyNameAttribute` cho assembly và chỉ định tên của kho chứa khóa.

Ngoài ra, bạn có thể tùy chọn:

- Áp dụng đặc tính `System.Reflection.AssemblyCultureAttribute` cho assembly để chỉ định thông tin bản địa mà assembly hỗ trợ (Bạn không thể chỉ định bản địa cho các assembly thực thi vì assembly thực thi chỉ hỗ trợ bản địa trung lập).
- Áp dụng đặc tính `System.Reflection.AssemblyVersionAttribute` cho assembly để chỉ định phiên bản của assembly.

Đoạn mã dưới đây (trong file *HelloWorld.cs*) minh họa cách sử dụng các đặc tính (phần in đậm) để chỉ định khóa, bản địa, và phiên bản cho assembly:

```
using System;

using System.Reflection;

[assembly:AssemblyKeyName("MyKeys")]
[assembly:AssemblyCulture("")]
[assembly:AssemblyVersion("1.0.0.0")]

public class HelloWorld {

    public static void Main() {

        Console.WriteLine("Hello, world");

    }

}
```

Để tạo một assembly tên mạnh từ đoạn mã trên, tạo các khóa tên mạnh và lưu trữ chúng trong file *MyKeyFile* bằng lệnh `sn -k MyKeyFile.snk`. Sau đó, sử dụng lệnh `sn -i MyKeyFile.snk MyKeys` để cài đặt các khóa vào một kho chứa khóa *CSP* có tên là *MyKeys*. Cuối cùng, sử dụng lệnh `csc HelloWorld.cs` để biên dịch file *HelloWorld.cs* thành một assembly tên mạnh.



Bạn cũng có thể sử dụng công cụ *Assembly Linker (al.exe)* để tạo assembly tên mạnh, cách này cho phép chỉ định các thông tin tên mạnh trên dòng lệnh thay vì sử dụng các đặc tính trong mã nguồn. Cách này hữu ích khi bạn không muốn nhúng các đặc tính tên mạnh vào file nguồn và khi bạn sử dụng kịch bản để xây dựng những cây mã nguồn đồ sộ. Xem thêm thông tin về *Assembly Linker* trong tài liệu *.NET Framework SDK*.

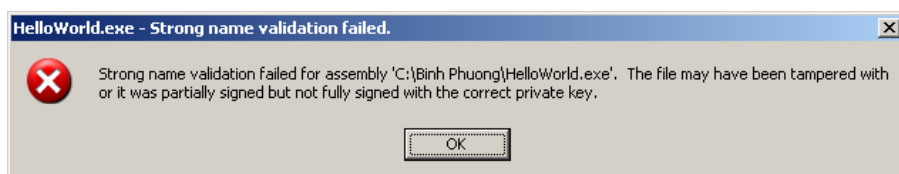
10. Xác minh một assembly tên mạnh không bị sửa đổi

? Bạn cần xác minh rằng một assembly tên mạnh chưa hề bị sửa đổi sau khi nó được biên dịch.

✕ Sử dụng công cụ *Strong Name (sn.exe)* để xác minh tên mạnh của assembly.

Mỗi khi nạp một assembly tên mạnh, bộ thực thi .NET lấy mã băm đã-được-mật-hóa (được nhúng trong assembly) và giải mật hóa với khóa công khai (cũng được nhúng trong assembly). Sau đó, bộ thực thi tính mã băm của assembly manifest và so sánh nó với mã băm vừa-được-giải-mật-hóa. Quá trình xác minh này sẽ nhận biết assembly có bị thay đổi sau khi biên dịch hay không.

Nếu một quá trình xác minh tên mạnh thất bại với một assembly thực thi, bộ thực thi sẽ hiển thị hộp thoại như hình 1.2. Nếu cố nạp một assembly đã thất bại trong quá trình xác minh, bộ thực thi sẽ ném ngoại lệ `System.IO.FileLoadException` với thông điệp “*Strong name validation failed*”.



Hình 1.2 Lỗi khi cố thực thi một assembly tên mạnh đã bị sửa đổi

Ngoài việc tạo và quản lý các khóa tên mạnh (đã được thảo luận trong mục 1.8), công cụ *Strong Name* còn cho phép xác minh các assembly tên mạnh. Để xác minh assembly tên mạnh *HelloWorld.exe* không bị sửa đổi, sử dụng lệnh `sn -vf HelloWorld.exe`. Đối số `-v` yêu cầu công cụ *Strong Name* xác minh tên mạnh của một assembly xác định, đối số `-f` buộc thực hiện việc xác minh tên mạnh ngay cả nó đã bị vô hiệu trước đó cho một assembly nào đó. (Bạn có thể sử dụng đối số `-vr` để vô hiệu việc xác minh tên mạnh đối với một assembly, ví dụ `sn -vr HelloWorld.exe`; mục 1.11 sẽ trình bày lý do tại sao cần vô hiệu việc xác minh tên mạnh).

Nếu assembly này được xác minh là không đổi, bạn sẽ thấy kết xuất như sau:

```
Microsoft (R) .NET Framework Strong Name Utility Version 1.1.4322.573
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.
```

```
Assembly 'HelloWorld.exe' is valid
```

Tuy nhiên, nếu assembly này đã bị sửa đổi, bạn sẽ thấy kết xuất như sau:

```
Microsoft (R) .NET Framework Strong Name Utility Version 1.1.4322.573
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.
```

```
Failed to verify assembly -- Unable to format error message 8013141A
```

11.

Hoãn việc ký assembly

?

Bạn cần tạo một assembly tên mạnh, nhưng không muốn mọi thành viên trong nhóm phát triển truy xuất khóa riêng của cặp khóa tên mạnh.

✕

Trích xuất và phân phối khóa công khai của cặp khóa tên mạnh. Làm theo hướng dẫn trong mục 1.9 để tạo tên mạnh cho assembly. Áp dụng đặc tính `System.Reflection.AssemblyDelaySignAttribute` cho assembly để chỉ định nó là assembly sẽ được ký sau. Sử dụng đối số `-vr` của công cụ *Strong Name (sn.exe)* để vô hiệu việc xác minh tên mạnh cho assembly này.

Các assembly tham chiếu đến assembly tên mạnh sẽ chứa token của assembly được tham chiếu, nghĩa là assembly được tham chiếu phải được tạo tên mạnh trước khi được tham chiếu. Trong một môi trường phát triển mà assembly thường xuyên được xây dựng lại, mỗi người phát triển và kiểm thử đều cần có quyền truy xuất cặp khóa tên mạnh của bạn—đây là một cơ chế bảo mật chủ yếu.

Thay vì phân phối khóa riêng cho mọi thành viên của nhóm phát triển, *.NET Framework* cung cấp cơ chế hoãn việc ký một assembly (được gọi là *delay signing*), theo đó bạn có thể tạo tên mạnh không hoàn chỉnh cho assembly (tạm gọi là tên mạnh bán phần). Tên mạnh bán phần này chỉ chứa khóa công khai và token của khóa công khai (cần thiết để tham chiếu assembly), nhưng chưa chỗ cho chữ ký sẽ được tạo ra từ khóa riêng sau này.

Khi quá trình phát triển hoàn tất, *signing authority* (người chịu trách nhiệm về việc bảo mật và việc sử dụng cặp khóa tên mạnh) sẽ ký lại assembly đã bị hoãn trước đó để hoàn thành tên mạnh cho nó. Chữ ký được tính toán dựa trên khóa riêng và được nhúng vào assembly, và giờ đây bạn đã có thể phân phối assembly.

Khi hoãn việc ký một assembly, bạn chỉ cần truy xuất khóa công khai của cặp khóa tên mạnh. Không có nguy cơ bảo mật nào từ việc phân phối khóa công khai, và *signing authority* phải phân phối khóa công khai đến mọi thành viên của nhóm phát triển. Để trích xuất khóa công khai từ file *MyKeys.snk* và ghi nó vào file *MyPublicKey.snk*, sử dụng lệnh `sn -p MyKeys.snk MyPublicKey.snk`. Nếu bạn lưu trữ cặp khóa tên mạnh trong một kho chứa khóa *CSP* có tên là *MyKeys*, sử dụng lệnh `sn -pc MyKeys MyPublicKey.snk` để trích xuất khóa công khai ra rồi lưu trữ nó vào file *MyPublicKey.snk*.

Ví dụ dưới đây áp dụng các đặc tính đã được thảo luận trong mục 1.9 để khai báo phiên bản, bản địa, và nơi chứa khóa công khai. Đồng thời áp dụng đặc tính `AssemblyDelaySign(true)` cho assembly để báo cho trình biên dịch biết bạn muốn trì hoãn việc ký assembly.

```
using System;

using System.Reflection;

[assembly:AssemblyKeyFile("MyPublicKey.snk")]
[assembly:AssemblyCulture("")]
[assembly:AssemblyVersion("1.0.0.0")]
```

```
[assembly:AssemblyDelaySign(true)]

public class HelloWorld {

    public static void Main() {

        Console.WriteLine("Hello, world");

    }

}
```

Khi cố nạp một assembly bị hoãn ký, bộ thực thi sẽ nhận ra assembly này có tên mạnh và cố xác minh assembly (như được thảo luận trong mục 1.10). Nhưng vì không có chữ ký số nên bạn phải vô hiệu chức năng xác minh này bằng lệnh `sn -Vr HelloWorld.exe`.

Khi quá trình phát triển hoàn tất, bạn cần ký lại assembly để hoàn thành tên mạnh cho assembly. Công cụ *Strong Name* cho phép thực hiện điều này mà không cần thay đổi mã nguồn hoặc biên dịch lại assembly, tuy nhiên, bạn phải có quyền truy xuất khóa riêng của cặp khóa tên mạnh. Để ký lại assembly có tên là *HelloWorld.exe* với cặp khóa nằm trong file *MyKeys.snk*, sử dụng lệnh `sn -R HelloWorld.exe MyKeys.snk`. Nếu cặp khóa được lưu trữ trong một kho chứa khóa *CSP* có tên là *MyKeys*, sử dụng lệnh `sn -Rc HelloWorld.exe MyKeys`.

Sau khi đã ký lại assembly, bạn phải mở chức năng xác minh tên mạnh cho assembly bằng đối số `-Vu` của công cụ *Strong Name*, ví dụ `sn -Vu HelloWorld.exe`. Để kích hoạt lại việc xác minh tên mạnh cho tất cả các assembly đã bị bạn vô hiệu trước đó, sử dụng lệnh `sn -Vx`. Sử dụng lệnh `sn -v1` để xem danh sách các assembly đã bị vô hiệu chức năng này.



Khi sử dụng assembly ký sau, bạn nên so sánh các lần xây dựng khác nhau của assembly để bảo đảm chúng chỉ khác nhau ở chữ ký. Điều này chỉ có thể thực hiện được nếu assembly đã được ký lại bằng đối số `-R` của công cụ *Strong Name*. Sử dụng lệnh `sn -D assembly1 assembly2` để so sánh hai assembly.

```
C:\WINDOWS\System32\cmd.exe
I:\CSharp\Chuong01\01-11>sn -k MyKeys.snk

Microsoft (R) .NET Framework Strong Name Utility Version 1.1.4322.573
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Key pair written to MyKeys.snk

I:\CSharp\Chuong01\01-11>sn -p MyKeys.snk MyPublicKey.snk

Microsoft (R) .NET Framework Strong Name Utility Version 1.1.4322.573
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Public key written to MyPublicKey.snk

I:\CSharp\Chuong01\01-11>cd bin\Debug

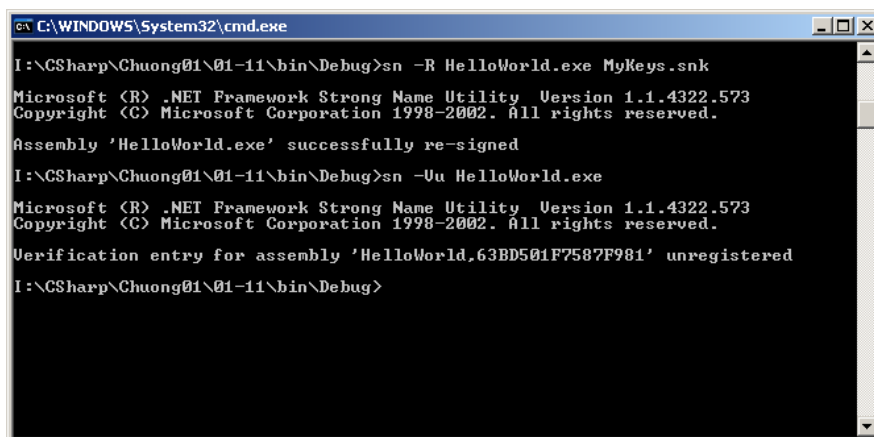
I:\CSharp\Chuong01\01-11\bin\Debug>sn -Ur HelloWorld.exe

Microsoft (R) .NET Framework Strong Name Utility Version 1.1.4322.573
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Verification entry added for assembly 'HelloWorld,63BD501F7587F981'

I:\CSharp\Chuong01\01-11\bin\Debug>
```

Hình 1.3 Tạm hoãn việc ký assembly



```

C:\WINDOWS\System32\cmd.exe

I:\CSharp\Chuong01\01-11\bin\Debug>sn -R HelloWorld.exe MyKeys.snk
Microsoft (R) .NET Framework Strong Name Utility Version 1.1.4322.573
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Assembly 'HelloWorld.exe' successfully re-signed

I:\CSharp\Chuong01\01-11\bin\Debug>sn -Uu HelloWorld.exe
Microsoft (R) .NET Framework Strong Name Utility Version 1.1.4322.573
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Verification entry for assembly 'HelloWorld,63BD501F7587F981' unregistered

I:\CSharp\Chuong01\01-11\bin\Debug>

```

Hình 1.4 Ký lại assembly

12.

Ký assembly với chữ ký số Authenticode

- ? Bạn cần ký một assembly bằng *Authenticode* để người dùng biết bạn chính là người phát hành (*publisher*) và assembly không bị sửa đổi sau khi ký.
- ✕ Sử dụng công cụ *File Signing* (*signcode.exe*) để ký assembly với *Software Publisher Certificate (SPC)* của bạn.

Tên mạnh cung cấp một định danh duy nhất cũng như chứng minh tính toàn vẹn của một assembly, nhưng nó không xác minh ai là người phát hành assembly này. Do đó, *.NET Framework* cung cấp kỹ thuật *Authenticode* để ký assembly. Điều này cho phép người dùng biết bạn là người phát hành và xác nhận tính toàn vẹn của assembly. Chữ ký *Authenticode* còn được sử dụng làm chứng cứ (*evidence*) cho assembly khi cấu hình chính sách bảo mật truy xuất mã lệnh (*Code Access Security Policy*—xem mục 13.9 và 13.10).

Để ký một assembly với chữ ký *Authenticode*, bạn cần một *SPC* do một *Certificate Authority (CA)* cấp. *CA* được trao quyền để cấp *SPC* (cùng với nhiều kiểu chứng chỉ khác) cho các cá nhân hoặc công ty sử dụng. Trước khi cấp một chứng chỉ, *CA* có trách nhiệm xác nhận những người yêu cầu và bảo đảm họ ký kết không sử dụng sai các chứng chỉ do *CA* cấp.

Để có được một *SPC*, bạn nên xem *Microsoft Root Certificate Program Members* tại [<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/rootcertprog.asp>].

Ở đây, bạn có thể tìm thấy danh sách các *CA*, nhiều *CA* trong số đó có thể cấp cho bạn một *SPC*. Với mục đích thử nghiệm, bạn có thể tạo một *SPC* thử nghiệm theo quá trình sẽ được mô tả trong mục 1.13. Tuy nhiên, bạn không thể phân phối phần mềm được ký với chứng chỉ thử nghiệm này. Vì một *SPC* thử nghiệm không do một *CA* đáng tin cậy cấp, nên hầu hết người dùng sẽ không tin tưởng assembly được ký bằng *SPC* thử nghiệm này.

Khi đã có một *SPC*, sử dụng công cụ *File Signing* để ký assembly của bạn. Công cụ *File Signing* sử dụng khóa riêng của *SPC* để tạo một chữ ký số và nhúng chữ ký này cùng phần

công khai của *SPC* vào assembly (bao gồm khóa công khai). Khi xác minh một assembly, người dùng sử dụng khóa công khai để giải mã mã băm đã-được-mật-hóa, tính toán lại mã băm của assembly, và so sánh hai mã băm này để bảo đảm chúng là như nhau. Khi hai mã băm này trùng nhau, người dùng có thể chắc chắn rằng bạn đã ký assembly, và nó không bị thay đổi từ khi bạn ký.

Ví dụ, để ký một assembly có tên là *MyAssembly.exe* với một *SPC* nằm trong file *MyCert.spc* và khóa riêng nằm trong file *MyPrivateKey.pvk*, sử dụng lệnh:

```
signcode -spc MyCert.spc -v MyPrivateKey.pvk MyAssembly.exe
```

Trong ví dụ này, công cụ *File Signing* sẽ hiển thị một hộp thoại như hình 1.5, yêu cầu bạn nhập mật khẩu (được sử dụng để bảo vệ khóa riêng trong file *MyPrivateKey.pvk*).



Hình 1.5 Công cụ File Signing yêu cầu nhập mật khẩu khi truy xuất file chứa khóa riêng

Bạn cũng có thể truy xuất khóa và chứng chỉ trong các kho chứa. Bảng 1.2 liệt kê các đối số thường dùng nhất của công cụ *File Signing*. Bạn hãy tham khảo tài liệu *.NET Framework SDK* để xem tất cả các đối số.

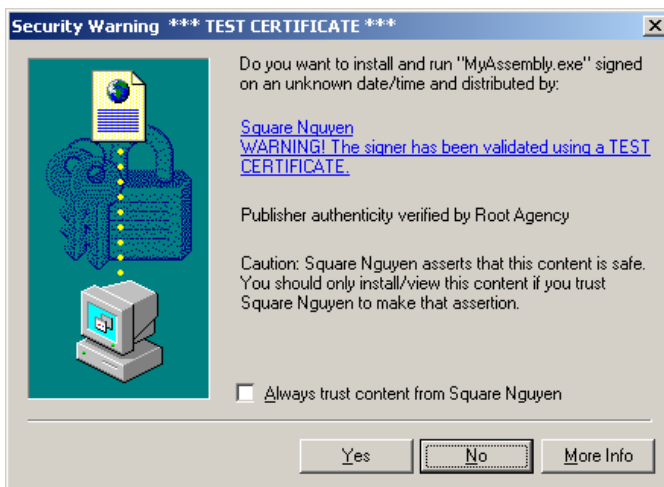
Bảng 1.2 Các đối số thường dùng của công cụ File Signing

Đối số	Mô tả
-k	Chỉ định tên của kho chứa khóa riêng <i>SPC</i>
-s	Chỉ định tên của kho chứa <i>SPC</i>
-spc	Chỉ định tên file chứa <i>SPC</i>
-v	Chỉ định tên file chứa khóa riêng <i>SPC</i>

Để ký một assembly gồm nhiều file, bạn cần chỉ định tên file chứa assembly manifest. Nếu muốn sử dụng cả tên mạnh và *Authenticode* cho assembly, bạn phải tạo tên mạnh cho assembly trước (xem cách tạo tên mạnh cho assembly trong mục 1.9).

Để kiểm tra tính hợp lệ của một file được ký với chữ ký *Authenticode*, sử dụng công cụ *Certificate Verification (chktrust.exe)*. Ví dụ, sử dụng lệnh `chktrust MyAssembly.exe` để kiểm tra file *MyAssembly.exe*. Nếu chưa cấu hình cho hệ thống để nó tin tưởng *SPC* dùng để ký assembly, bạn sẽ thấy hộp thoại tương tự như hình 1.6, hiển thị thông tin về người phát hành và cho bạn chọn là có tin tưởng người phát hành đó hay không (chứng chỉ trong hình 1.6 là một chứng chỉ thử nghiệm được tạo theo quá trình được mô tả trong mục 1.13).

Nếu bạn nhấp *Yes*, hoặc trước đó đã chọn là luôn tin tưởng *SPC*, công cụ *Certificate Verification* xác nhận tính hợp lệ của chữ ký và assembly.



Hình 1.6 Công cụ Certificate Verification

13.

Tạo và thiết lập tin tưởng một SPC thử nghiệm

?

Bạn cần tạo một SPC để thử nghiệm.

✂

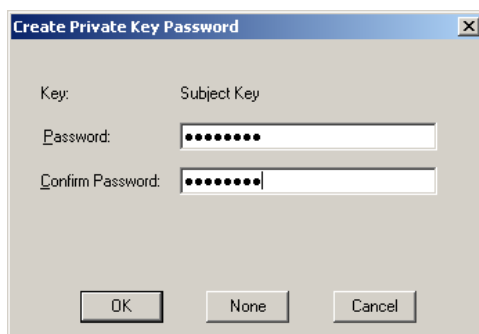
Sử dụng công cụ *Certificate Creation (makecert.exe)* để tạo một chứng chỉ X.509 và sử dụng công cụ *Software Publisher Certificate (cert2spc.exe)* để tạo một SPC từ chứng chỉ X.509 này. Thiết lập tin tưởng chứng chỉ thử nghiệm bằng công cụ *Set Registry (setreg.exe)*.

Để tạo một SPC thử nghiệm cho một nhà phát hành phần mềm có tên là *Square Nguyen*, trước hết sử dụng công cụ *Certificate Creation* để tạo một chứng chỉ X.509. Lệnh:

```
makecert -n "CN=Square Nguyen" -sk MyKeys TestCertificate.cer
```

sẽ tạo một file có tên là *TestCertificate.cer* chứa một chứng chỉ X.509, và lưu trữ khóa riêng tương ứng trong một kho chứa khóa CSP có tên là *MyKeys* (được tạo tự động nếu chưa tồn tại). Bạn cũng có thể ghi khóa riêng vào file bằng cách thay *-sk* bằng *-sv*. Ví dụ, để ghi khóa riêng vào một file có tên là *PrivateKeys.pvk*, sử dụng lệnh:

```
makecert -n "CN=Square Nguyen" -sv PrivateKey.pvk TestCertificate.cer
```



Hình 1.7 Công cụ Certificate Creation nhắc nhập mật khẩu để bảo vệ file chứa khóa riêng

Nếu bạn ghi khóa riêng vào file, công cụ *Certificate Creation* sẽ nhắc bạn nhập mật khẩu để bảo vệ file này (xem hình 1.7).

Công cụ *Certificate Creation* hỗ trợ nhiều đối số, bảng 1.3 liệt kê một vài đối số thường dùng. Xem thêm tài liệu *.NET Framework SDK* về công cụ *Certificate Creation*.

Bảng 1.3 Các đối số thường dùng của công cụ Certificate Creation

Đối số	Mô tả
-e	Chỉ định ngày chứng chỉ không còn hiệu lực.
-m	Chỉ định khoảng thời gian (tính bằng tháng) mà chứng chỉ còn hiệu lực.
-n	Chỉ định một tên <i>X.500</i> tương ứng với chứng chỉ. Đây là tên của người phát hành phần mềm mà người dùng thấy khi họ xem chi tiết của <i>SPC</i> tạo ra.
-sk	Chỉ định tên <i>CSP</i> giữ khóa riêng.
-ss	Chỉ định tên kho chứng chỉ (công cụ <i>Certificate Creation</i> sẽ lưu chứng chỉ <i>X.509</i> trong đó).
-sv	Chỉ định tên file giữ khóa riêng.

Khi đã tạo một chứng chỉ *X.509* bằng công cụ *Certificate Creation*, cần chuyển chứng chỉ này thành một *SPC* bằng công cụ *Software Publisher Certificate Test* (*cert2spc.exe*). Để chuyển *TestCertificate.cer* thành một *SPC*, sử dụng lệnh:

```
cert2spc TestCertificate.cer TestCertificate.spc
```

Công cụ *Software Publisher Certificate Test* không có đối số tùy chọn nào.

Bước cuối cùng để sử dụng *SPC* thử nghiệm là thiết lập tin tưởng *CA* thử nghiệm gốc (*root test CA*); đây là người phát hành mặc định các chứng chỉ thử nghiệm. Bước này chỉ cần lệnh `setreg 1 true` của công cụ *Set Registry* (*setreg.exe*). Khi kết thúc thử nghiệm *SPC*, bỏ thiết lập tin tưởng đối với *CA* thử nghiệm bằng lệnh `setreg 1 false`. Bây giờ, bạn có thể sử dụng *SPC* thử nghiệm để ký assembly với *Authenticode* như quá trình mô tả ở mục 1.12.

14.***Quản lý Global Assembly Cache*****?**

Bạn cần thêm hoặc loại bỏ assembly từ *Global Assembly Cache (GAC)*.

✕

Sử dụng công cụ *Global Assembly Cache (gacutil.exe)* từ dòng lệnh để xem nội dung của *GAC*, cũng như thêm hoặc loại bỏ assembly.

Trước khi được cài đặt vào *GAC*, assembly phải có tên mạnh (xem mục 1.9 về cách tạo tên mạnh cho assembly). Để cài đặt assembly có tên là *SomeAssembly.dll* vào *GAC*, sử dụng lệnh `gacutil /i SomeAssembly.dll`.

Để loại bỏ *SomeAssembly.dll* ra khỏi *GAC*, sử dụng lệnh `gacutil /u SomeAssembly`. Chú ý không sử dụng phần mở rộng *.dll* để nói đến assembly một khi nó đã được cài đặt vào *GAC*.

Để xem các assembly đã được cài đặt vào *GAC*, sử dụng lệnh `gacutil /l`. Lệnh này sẽ liệt kê tất cả các assembly đã được cài đặt trong *GAC*, cũng như danh sách các assembly đã được biên dịch trước sang dạng nhị phân và cài đặt trong *NGEN* cache. Sử dụng lệnh `gacutil /l SomeAssembly` để tránh phải tìm hết danh sách xem một assembly đã được cài đặt chưa.



***.NET Framework* sử dụng *GAC* chỉ khi thực thi, trình biên dịch *C#* sẽ không tìm trong *GAC* bất kỳ tham chiếu ngoại nào mà assembly của bạn tham chiếu đến. Trong quá trình phát triển, trình biên dịch *C#* phải truy xuất được một bản sao cục bộ của bất kỳ assembly chia sẻ nào được tham chiếu đến. Bạn có thể chép assembly chia sẻ vào thư mục mã nguồn của bạn, hoặc sử dụng đối số `/lib` của trình biên dịch *C#* để chỉ định thư mục mà trình biên dịch có thể tìm thấy các assembly cần thiết trong đó.**

15.***Ngăn người khác dịch ngược mã nguồn của bạn*****?**

Bạn muốn bảo đảm assembly *.NET* của bạn không bị dịch ngược.

✕

Xây dựng các giải pháp dựa-trên-server nếu có thể để người dùng không truy xuất assembly được. Nếu bạn phải phân phối assembly thì không có cách nào để ngăn người dùng dịch ngược chúng. Cách tốt nhất có thể làm là sử dụng kỹ thuật *obfuscation* và các thành phần đã được biên dịch thành mã lệnh nguyên sinh (*native code*) để assembly khó bị dịch ngược hơn.

Vì assembly *.NET* bao gồm một tập các mã lệnh và siêu dữ liệu được chuẩn hóa, độc lập nền tảng mô tả các kiểu nằm trong assembly, nên chúng tương đối dễ bị dịch ngược. Điều này cho phép các trình dịch ngược dễ dàng tạo được mã nguồn rất giống với mã gốc, đây sẽ là vấn đề khó giải quyết nếu mã của bạn có chứa các thông tin hoặc thuật toán cần giữ bí mật.

Cách duy nhất để đảm bảo người dùng không thể dịch ngược assembly là không cho họ lấy được assembly. Nếu có thể, hiện thực các giải pháp dựa-trên-server như các ứng dụng *Microsoft ASP.NET* và dịch vụ *Web XML*. Với một chính sách bảo mật tốt ở server, không ai có thể truy xuất assembly, do đó không thể dịch ngược chúng.

Nếu việc xây dựng các giải pháp dựa-trên-server là không phù hợp, bạn có hai tùy chọn sau đây:

- Sử dụng một *obfuscator* để khiến cho assembly của bạn khó bị dịch ngược (*Visual Studio .NET 2003* có chứa phiên bản *Community* của một *obfuscator*, có tên là *Dotfuscator*). *Obfuscator* sử dụng nhiều kỹ thuật khác nhau khiến cho assembly khó bị dịch ngược; nguyên lý của các kỹ thuật này là:
 - Đổi tên các trường và các phương thức *private* nhằm gây khó khăn cho việc đọc và hiểu mục đích của mã lệnh.
 - Chèn các lệnh dòng điều khiển khiến cho người khác khó có thể lần theo logic của ứng dụng.
- Chuyển những phần của ứng dụng mà bạn muốn giữ bí mật thành các đối tượng *COM* hay các *DLL* nguyên sinh, sau đó sử dụng *P/Invoke* hoặc *COM Interop* để gọi chúng từ ứng dụng được-quản-lý của bạn (xem chương 15 về cách gọi mã lệnh không-được-quản-lý).

Không có cách tiếp cận nào ngăn được những người có kỹ năng và quyết tâm dịch ngược mã nguồn của bạn, nhưng chúng sẽ làm cho công việc này trở nên khó khăn đáng kể và ngăn được hầu hết những kẻ tò mò thông thường.

Nguy cơ một ứng dụng bị dịch ngược không chỉ riêng cho *C#* hay *.NET*. Một người quyết tâm có thể dịch ngược bất kỳ phần mềm nào nếu anh ta có kỹ năng và thời gian.

2

THAO TÁC DỮ LIỆU

Hầu hết các ứng dụng đều cần thao tác trên một loại dữ liệu nào đó. *Microsoft .NET Framework* cung cấp nhiều kỹ thuật để đơn giản hóa hay nâng cao hiệu quả các thao tác dữ liệu thông dụng. Chương này sẽ đề cập các kỹ thuật sau:

- Thao tác chuỗi một cách hiệu quả (mục 2.1).
- Mô tả các kiểu dữ liệu cơ sở bằng các kiểu mã hóa khác nhau (mục 2.2, 2.3, và 2.4).
- Sử dụng biểu thức chính quy để xác nhận tính hợp lệ và thao tác chuỗi (mục 2.5 và 2.6).
- Làm việc với ngày và giờ (mục 2.7 và 2.8).
- Làm việc với mảng và tập hợp (mục 2.9, 2.10, và 2.11).
- Tuân tự hóa trạng thái đối tượng và lưu nó vào file (mục 2.12).

1. *Thao tác chuỗi một cách hiệu quả*

? **Bạn cần thao tác trên nội dung của một đối tượng `String` và tránh chi phí của việc tự động tạo các đối tượng `String` mới do tính không đổi của đối tượng `String`.**

✖ **Sử dụng lớp `System.Text.StringBuilder` để thực hiện các thao tác, sau đó chuyển kết quả thành `String` bằng phương thức `StringBuilder.ToString`.**

Các đối tượng `String` trong *.NET* là không đổi, nghĩa là một khi đã được tạo thì chúng không thể bị thay đổi. Ví dụ, nếu bạn tạo một `String` bằng cách nối một số ký tự hoặc chuỗi, thì khi thêm một phần tử mới vào cuối `String` hiện có, bộ thực thi sẽ tạo ra một `String` mới chứa kết quả (chứ không phải `String` cũ bị thay đổi). Do đó sẽ nảy sinh chi phí đáng kể nếu ứng dụng của bạn thường xuyên thao tác trên `String`.

Lớp `StringBuilder` khắc phục vấn đề này bằng cách cung cấp một bộ đệm ký tự, và cho phép thao tác trên nội dung của nó mà bộ thực thi không phải tạo đối tượng mới để chứa kết quả sau mỗi lần thay đổi. Bạn có thể tạo một đối tượng `StringBuilder` rỗng hoặc được khởi tạo là nội dung của một `String` hiện có. Sau đó, thao tác trên nội dung của `StringBuilder` này bằng các phương thức nạp chồng (cho phép bạn chèn, thêm dạng chuỗi của các kiểu dữ liệu khác nhau). Cuối cùng, gọi `StringBuilder.ToString` để chuyển nội dung hiện tại của `StringBuilder` thành một `String`.

Khi bạn thêm dữ liệu mới vào chuỗi, có hai thuộc tính quan trọng ảnh hưởng đến hoạt động của `StringBuilder` là `Capacity` và `Length`. `Capacity` mô tả kích thước của bộ đệm `StringBuilder`, còn `Length` mô tả kích thước của chuỗi ký tự trong bộ đệm. Nếu việc thêm dữ liệu mới vào `StringBuilder` làm kích thước chuỗi (`Length`) vượt quá kích thước bộ đệm (`Capacity`) thì `StringBuilder` sẽ cấp phát bộ đệm mới để chứa chuỗi. Nếu thiếu cẩn thận, việc cấp phát bộ đệm này có thể phủ định lợi ích của việc sử dụng `StringBuilder`. Do đó, nếu biết chính xác kích thước của chuỗi, hoặc biết kích thước tối đa của chuỗi, bạn có thể tránh việc cấp phát bộ đệm quá mức cần thiết bằng cách thiết lập thuộc tính `Capacity` hoặc chỉ định kích thước bộ đệm lúc tạo `StringBuilder`. Khi thiết lập các thuộc tính `Capacity` và `Length`, cần chú ý các điểm sau:

- Nếu bạn thiết lập giá trị `Capacity` nhỏ hơn giá trị `Length`, thuộc tính `Capacity` sẽ ném ngoại lệ `System.ArgumentOutOfRangeException`.
- Nếu bạn thiết lập giá trị `Length` nhỏ hơn kích thước của chuỗi hiện có trong bộ đệm, chuỗi sẽ bị cắt bớt phần lớn hơn.
- Nếu bạn thiết lập giá trị `Length` lớn hơn kích thước của chuỗi, bộ đệm sẽ được "lấp" thêm các khoảng trống cho bằng với `Length`. Việc thiết lập giá trị `Length` lớn hơn giá trị `Capacity` sẽ tự động điều chỉnh `Capacity` cho bằng với `Length`.

Phương thức `ReverseString` dưới đây minh họa cách sử dụng lớp `StringBuilder` để đảo một chuỗi. Nếu không sử dụng lớp `StringBuilder` để thực hiện thao tác này thì sẽ tốn chi phí đáng kể, đặc biệt khi chuỗi nguồn dài. Việc khởi tạo `StringBuilder` với kích thước bằng chuỗi nguồn bảo đảm không cần phải cấp phát lại bộ đệm trong quá trình đảo chuỗi.

```
public static string ReverseString(string str) {

    // Kiểm tra các trường hợp không cần đảo chuỗi.
    if (str == null || str.Length == 1) {
        return str;
    }

    // Tạo một StringBuilder với sức chứa cần thiết.
    System.Text.StringBuilder revStr =
        new System.Text.StringBuilder(str.Length);

    // Duyệt ngược chuỗi nguồn từng ký tự một
    // và thêm từng ký tự đọc được vào StringBuilder.
    for (int count = str.Length-1; count > -1; count--) {
        revStr.Append(str[count]);
    }

    // Trả về chuỗi đã được đảo.
    return revStr.ToString();
}
```

2.

Mã hóa chuỗi bằng các kiểu mã hóa ký tự

?

Bạn cần trao đổi dữ liệu dạng ký tự với các hệ thống sử dụng kiểu mã hóa khác với *UTF-16* (kiểu mã hóa này được sử dụng bởi *C#*).



Sử dụng lớp `System.Text.Encoding` và các lớp con của nó để chuyển đổi ký tự giữa các kiểu mã hóa khác nhau.

Unicode không phải là kiểu mã hóa duy nhất, cũng như *UTF-16* không phải cách duy nhất biểu diễn ký tự *Unicode*. Khi ứng dụng cần trao đổi dữ liệu ký tự với các hệ thống bên ngoài (đặc biệt là các hệ thống cũ), dữ liệu cần phải được chuyển đổi giữa *UTF-16* và kiểu mã hóa mà hệ thống đó hỗ trợ.

Lớp trừu tượng `Encoding`, và các lớp con của nó cung cấp các chức năng để chuyển ký tự qua lại giữa nhiều kiểu mã hóa khác nhau. Mỗi thể hiện của lớp con hỗ trợ việc chuyển đổi giữa *UTF-16* và một kiểu mã hóa khác. Phương thức tĩnh `Encoding.GetEncoding` nhận vào tên hoặc số hiệu trang mã (*code page number*) của một kiểu mã hóa và trả về thể hiện của lớp mã hóa tương ứng.

Bảng 2.1 liệt kê một vài kiểu mã ký tự và số hiệu trang mã mà bạn phải truyền cho phương thức `GetEncoding` để tạo ra thể hiện của lớp mã hóa tương ứng. Bảng này cũng cung cấp các thuộc tính tĩnh của lớp `Encoding` đại diện cho phương thức `GetEncoding` tương ứng.

Bảng 2.1 Các lớp mã hóa ký tự

Kiểu mã hóa	Lớp	Sử dụng
<i>ASCII</i>	<code>ASCIIEncoding</code>	<code>GetEncoding(20127)</code> hay thuộc tính <code>ASCII</code>
Mặc định (<i>kiểu mã hóa hiện hành trên hệ thống</i>)	<code>Encoding</code>	<code>GetEncoding(0)</code> hay thuộc tính <code>Default</code>
<i>UTF-7</i>	<code>UTF7Encoding</code>	<code>GetEncoding(65000)</code> hay thuộc tính <code>UTF7</code>
<i>UTF-8</i>	<code>UTF8Encoding</code>	<code>GetEncoding(65001)</code> hay thuộc tính <code>UTF8</code>
<i>UTF-16 (Big Endian)</i>	<code>UnicodeEncoding</code>	<code>GetEncoding(1201)</code> hay thuộc tính <code>BigEndianUnicode</code>
<i>UTF-16 (Little Endian)</i>	<code>UnicodeEncoding</code>	<code>GetEncoding(1200)</code> hay thuộc tính <code>Unicode</code>
<i>Windows OS</i>	<code>Encoding</code>	<code>GetEncoding(1252)</code>

Sau khi đã lấy được đối tượng lớp `Encoding` hỗ trợ kiểu mã hóa thích hợp, sử dụng phương thức `GetBytes` để chuyển chuỗi nguồn (được mã hóa theo *UTF-16*) thành mảng kiểu byte chứa các ký tự được mã hóa theo kiểu cần chuyển, và sử dụng `GetString` để chuyển mảng byte thành chuỗi đích. Ví dụ dưới đây trình bày cách sử dụng một vài lớp mã hóa:

```
using System;
using System.IO;
using System.Text;

public class CharacterEncodingExample {
```

```
public static void Main() {

    // Tạo file giữ các kết quả.
    using (StreamWriter output = new StreamWriter("output.txt")) {

        // Tạo và ghi ra file một chuỗi chứa ký hiệu của số PI.
        string srcString = "Area = \u03A0r^2";
        output.WriteLine("Source Text : " + srcString);

        // Ghi các byte được mã hóa theo UTF-16
        // của chuỗi nguồn ra file.
        byte[] utf16String = Encoding.Unicode.GetBytes(srcString);
        output.WriteLine("UTF-16 Bytes: {0}",
            BitConverter.ToString(utf16String));

        // Chuyển chuỗi nguồn được mã hóa theo UTF-16
        // thành UTF-8 và ASCII
        byte[] utf8String = Encoding.UTF8.GetBytes(srcString);
        byte[] asciiString = Encoding.ASCII.GetBytes(srcString);

        // Ghi mảng các byte được mã hóa theo UTF-8 và ASCII ra file.
        output.WriteLine("UTF-8 Bytes: {0}",
            BitConverter.ToString(utf8String));
        output.WriteLine("ASCII Bytes: {0}",
            BitConverter.ToString(asciiString));

        // Chuyển các byte được mã hóa theo UTF-8 và ASCII
        // thành chuỗi được mã hóa theo UTF-16 và ghi ra file.
        output.WriteLine("UTF-8 Text : {0}",
            Encoding.UTF8.GetString(utf8String));
        output.WriteLine("ASCII Text : {0}",
            Encoding.ASCII.GetString(asciiString));

        // Ghi dữ liệu xuống file và đóng file.
        output.Flush();
    }
}
```



```

        output.Close();
    }
}

```

Chạy `CharacterEncodingExample` sẽ tạo ra file `output.txt`. Mở file này trong một trình soạn thảo có hỗ trợ *Unicode*, bạn sẽ thấy kết quả như sau:

```

Source Text : Area = Πr^2
UTF-16 Bytes: 41-00-72-00-65-00-61-00-20-00-3D-00-20-00-A0-03-72-00-5E-00-32-00
UTF-8  Bytes: 41-72-65-61-20-3D-20-CE-A0-72-5E-32
ASCII  Bytes: 41-72-65-61-20-3D-20-3F-72-5E-32
UTF-8  Text : Area = Πr^2
ASCII  Text : Area = ?r^2

```

Chú ý rằng, nếu sử dụng *UTF-16* thì mỗi ký tự được mã hóa bởi 2 byte, nhưng vì hầu hết các ký tự đều là ký tự chuẩn nên byte cao là 0 (nếu sử dụng *little-endian* thì byte thấp viết trước). Do đó, hầu hết các ký tự đều được mã hóa bởi những số giống nhau trong ba kiểu mã hóa, ngoại trừ ký hiệu *PI* được mã hóa khác (được in đậm trong kết quả ở trên). Để mã hóa *PI* cần 2 byte, đòi hỏi này được *UTF-8* hỗ trợ nên thể hiện được *Π*, trong khi đó *ASCII* chỉ sử dụng một byte nên thay *PI* bằng mã 3F, đây là mã của dấu hỏi (?).

 **Nếu chuyển các ký tự *Unicode* sang *ASCII* hoặc một kiểu mã hóa khác thì có thể mất dữ liệu. Bất kỳ ký tự *Unicode* nào có mã ký tự không biểu diễn được trong kiểu mã hóa đích sẽ bị bỏ qua khi chuyển đổi.**

Lớp `Encoding` cũng cung cấp phương thức tĩnh `Convert` để đơn giản hóa việc chuyển một mảng byte từ kiểu mã hóa này sang kiểu mã hóa khác không phải qua trung gian *UTF-16*. Ví dụ, dòng mã sau chuyển trực tiếp các byte trong mảng `asciiString` từ *ASCII* sang *UTF-8*:

```

byte[] utf8String = Encoding.Convert(Encoding.ASCII, Encoding.UTF8,
    asciiString);

```

3. Chuyển các kiểu giá trị cơ bản thành mảng kiểu byte

? **Bạn cần chuyển các kiểu giá trị cơ bản thành mảng kiểu byte.**

 **Lớp `System.BitConverter` cung cấp các phương thức tĩnh rất tiện lợi cho việc chuyển đổi qua lại giữa các mảng kiểu byte và hầu hết các kiểu giá trị cơ bản—trừ kiểu `decimal`. Để chuyển một giá trị kiểu `decimal` sang mảng kiểu byte, bạn cần sử dụng đối tượng `System.IO.BinaryWriter` để ghi giá trị đó vào một thể hiện `System.IO.MemoryStream`, sau đó gọi phương thức `MemoryStream.ToArray`. Để có một giá trị `decimal` từ một mảng kiểu byte, bạn cần tạo một đối tượng `MemoryStream` từ mảng kiểu byte, sau đó sử dụng thể hiện `System.IO.BinaryReader` để đọc giá trị này từ `MemoryStream`.**

Phương thức tĩnh `GetBytes` của lớp `BitConverter` cung cấp nhiều phiên bản nạp chồng cho phép chuyển hầu hết các kiểu giá trị cơ bản sang mảng kiểu byte. Các kiểu được hỗ trợ là `bool`, `char`, `double`, `short`, `int`, `long`, `float`, `ushort`, `uint`, và `ulong`. Lớp `BitConverter` cũng cung cấp các phương thức tĩnh cho phép chuyển các mảng kiểu byte thành các kiểu giá trị chuẩn như `ToBoolean`, `ToUInt32`, `toDouble`,... Ví dụ sau minh họa cách chuyển các giá trị `bool` và `int` thành mảng kiểu byte, và ngược lại. Đối số thứ hai trong `ToBoolean` và `ToUInt32` cho biết vị trí (tính từ 0) trong mảng byte mà `BitConverter` sẽ lấy các byte kể từ đó để tạo giá trị dữ liệu.

```
byte[] b = null;

// Chuyển một giá trị bool thành mảng kiểu byte và hiển thị.
b = BitConverter.GetBytes(true);
Console.WriteLine(BitConverter.ToString(b));

// Chuyển một mảng kiểu byte thành giá trị bool và hiển thị.
Console.WriteLine(BitConverter.ToBoolean(b, 0));

// Chuyển một giá trị int thành mảng kiểu byte và hiển thị.
b = BitConverter.GetBytes(3678);
Console.WriteLine(BitConverter.ToString(b));

// Chuyển một mảng kiểu byte thành giá trị int và hiển thị.
Console.WriteLine(BitConverter.ToInt32(b, 0));
```

Đối với kiểu `decimal`, lớp `BitConverter` không hỗ trợ, nên bạn phải sử dụng thêm `MemoryStream` và `BinaryWriter`.

```
// Tạo mảng kiểu byte từ giá trị decimal.
public static byte[] DecimalToByteArray(decimal src) {

    // Tạo một MemoryStream làm bộ đệm chứa dữ liệu nhị phân.
    using (MemoryStream stream = new MemoryStream()) {

        // Tạo một BinaryWriter để ghi dữ liệu nhị phân vào stream.
        using (BinaryWriter writer = new BinaryWriter(stream)) {

            // Ghi giá trị decimal vào BinaryWriter/MemoryStream.
            writer.Write(src);
```

```

        // Trả về mảng kiểu byte.
        return stream.ToArray();
    }
}

```

Để chuyển một mảng kiểu byte thành một giá trị decimal, sử dụng `BinaryReader` để đọc từ `MemoryStream`.

```

// Tạo giá trị decimal từ mảng kiểu byte.
public static decimal ByteArrayToDecimal (byte[] src) {

    // Tạo một MemoryStream chứa mảng.
    using (MemoryStream stream = new MemoryStream(src)) {

        // Tạo một BinaryReader để đọc từ stream.
        using (BinaryReader reader = new BinaryReader(stream)) {

            // Đọc và trả về giá trị decimal từ
            // BinaryReader/MemoryStream.
            return reader.ReadDecimal();
        }
    }
}

```



Lớp `BitConverter` cũng cung cấp phương thức `ToString` để tạo một `String` chứa giá trị mảng. Gọi `ToString` và truyền đối số là một mảng byte sẽ trả về một `String` chứa giá trị thập lục phân của các byte trong mảng, các giá trị này cách nhau bởi dấu gạch nối, ví dụ “34-A7-2C”. Tuy nhiên, không có phương thức nào tạo một mảng kiểu byte từ một chuỗi theo định dạng này.

4.

Mã hóa dữ liệu nhị phân thành văn bản



Bạn cần chuyển dữ liệu nhị phân sang một dạng sao cho có thể được lưu trữ trong một file văn bản *ASCII* (chẳng hạn file *XML*), hoặc được gửi đi trong e-mail.



Sử dụng các phương thức tĩnh `ToBase64String` và `FromBase64String` của lớp `System.Convert` để chuyển đổi qua lại giữa dữ liệu nhị phân và chuỗi được mã hóa theo *Base64*.

Base64 là một kiểu mã hóa cho phép bạn mô tả dữ liệu nhị phân như một dãy các ký tự *ASCII* để nó có thể được chèn vào một file văn bản hoặc một e-mail, mà ở đó dữ liệu nhị phân không được cho phép. *Base64* làm việc trên nguyên tắc sử dụng 4 byte để chứa 3 byte dữ liệu nguồn và đảm bảo mỗi byte chỉ sử dụng 7 bit thấp để chứa dữ liệu. Điều này có nghĩa là mỗi byte dữ liệu được mã hóa theo *Base64* có dạng giống như một ký tự *ASCII*, nên có thể được lưu trữ hoặc truyền đi bất cứ nơi đâu cho phép ký tự *ASCII*.

Lớp `Convert` cung cấp hai phương thức `ToBase64String` và `FromBase64String` để mã hóa và giải mã *Base64*. Tuy nhiên, trước khi mã hóa *Base64*, bạn phải chuyển dữ liệu thành mảng kiểu byte; và sau khi giải mã, bạn phải chuyển mảng kiểu byte trở về kiểu dữ liệu thích hợp (xem lại mục 2.2 và 2.3).

Ví dụ sau minh họa cách sử dụng lớp `Convert` để mã hóa và giải mã *Base64* với chuỗi *Unicode*, giá trị `int`, giá trị `decimal`. Đối với giá trị `decimal`, bạn phải sử dụng lại các phương thức `ByteArrayToDecimal` và `DecimalToByteArray` trong mục 2.3.

```
// Mã hóa Base64 với chuỗi Unicode.
public static string StringToBase64 (string src) {

    // Chuyển chuỗi thành mảng kiểu byte.
    byte[] b = Encoding.Unicode.GetBytes(src);

    // Trả về chuỗi được mã hóa theo Base64.
    return Convert.ToBase64String(b);
}

// Giải mã một chuỗi Unicode được mã hóa theo Base64.
public static string Base64ToString (string src) {

    // Giải mã vào mảng kiểu byte.
    byte[] b = Convert.FromBase64String(src);

    // Trả về chuỗi Unicode.
    return Encoding.Unicode.GetString(b);
}

// Mã hóa Base64 với giá trị decimal.
public static string DecimalToBase64 (decimal src) {

    // Chuyển giá trị decimal thành mảng kiểu byte.
```

```
byte[] b = DecimalToByteArray(src);

// Trả về giá trị decimal được mã hóa theo Base64.
return Convert.ToBase64String(b);
}

// Giải mã một giá trị decimal được mã hóa theo Base64.
public static decimal Base64ToDecimal (string src) {

    // Giải mã vào mảng kiểu byte.
    byte[] b = Convert.FromBase64String(src);

    // Trả về giá trị decimal.
    return ByteArrayToDecimal(b);
}

// Mã hóa Base64 với giá trị int.
public static string IntToBase64 (int src) {

    // Chuyển giá trị int thành mảng kiểu byte.
    byte[] b = BitConverter.GetBytes(src);

    // Trả về giá trị int được mã hóa theo Base64.
    return Convert.ToBase64String(b);
}

// Giải mã một giá trị int được mã hóa theo Base64.
public static int Base64ToInt (string src) {

    // Giải mã vào mảng kiểu byte.
    byte[] b = Convert.FromBase64String(src);

    // Trả về giá trị int.
    return BitConverter.ToInt32(b, 0);
}
```

5. Sử dụng biểu thức chính quy để kiểm tra dữ liệu nhập

? Bạn cần kiểm tra dữ liệu nhập vào có đúng với cấu trúc và nội dung được quy định trước hay không. Ví dụ, bạn muốn bảo đảm người dùng nhập địa chỉ IP, số điện thoại, hay địa chỉ e-mail hợp lệ.

✕ Sử dụng biểu thức chính quy để bảo đảm dữ liệu nhập đúng cấu trúc và chỉ chứa các ký tự được quy định trước đối với từng dạng thông tin.

Khi ứng dụng nhận dữ liệu từ người dùng hoặc đọc dữ liệu từ file, bạn nên giả định dữ liệu này là chưa chính xác và cần được kiểm tra lại. Một nhu cầu kiểm tra khá phổ biến là xác định các số điện thoại, số thẻ tín dụng, địa chỉ e-mail có đúng dạng hay không. Việc kiểm tra cấu trúc và nội dung của dữ liệu không đảm bảo dữ liệu là chính xác nhưng giúp loại bỏ nhiều dữ liệu sai và đơn giản hóa việc kiểm tra sau này. Biểu thức chính quy (*regular expression*) cung cấp một cơ chế rất tốt để kiểm tra một chuỗi có đúng với cấu trúc quy định trước hay không, do đó bạn có thể lợi dụng tính năng này cho mục đích kiểm tra dữ liệu nhập.

Trước tiên, bạn phải xác định cú pháp của biểu thức chính quy cho phù hợp với cấu trúc và nội dung của dữ liệu cần kiểm tra, đây là phần khó nhất khi sử dụng biểu thức chính quy. Biểu thức chính quy được xây dựng trên hai yếu tố: trực kiện (*literal*) và siêu ký tự (*metacharacter*). Trực kiện mô tả các ký tự có thể xuất hiện trong mẫu mà bạn muốn so trùng; siêu ký tự hỗ trợ việc so trùng các ký tự đại diện (wildcard), tầm trị, nhóm, lặp, điều kiện, và các cơ chế điều khiển khác. Ở đây không thảo luận đầy đủ về cú pháp biểu thức chính quy (tham khảo tài liệu *.NET SDK* để hiểu thêm về biểu thức chính quy), nhưng bảng 2.2 sẽ mô tả các siêu ký tự thường dùng.

Bảng 2.2 Các siêu ký tự thường dùng

Siêu ký tự	Mô tả
.	Mọi ký tự trừ ký tự xuống dòng (<code>\n</code>).
<code>\d</code>	Ký tự chữ số thập phân (digit).
<code>\D</code>	Ký tự không phải chữ số (non-digit).
<code>\s</code>	Ký tự whitespace (như khoảng trắng, tab...).
<code>\S</code>	Ký tự non-whitespace.
<code>\w</code>	Ký tự word (gồm mẫu tự, chữ số, và dấu gạch dưới).
<code>\W</code>	Ký tự non-word.
<code>^</code>	Bắt đầu một chuỗi hoặc dòng.
<code>\A</code>	Bắt đầu một chuỗi.
<code>\$</code>	Kết thúc một chuỗi hoặc dòng.
<code>\Z</code>	Kết thúc một chuỗi.

	Ngăn cách các biểu thức có thể so trùng, ví dụ <code>AAA ABA ABB</code> sẽ so trùng với <code>AAA</code> , <code>ABA</code> , hoặc <code>ABB</code> (các biểu thức được so trùng từ trái sang).
<code>[abc]</code>	So trùng với một trong các ký tự trong nhóm, ví dụ <code>[AbC]</code> sẽ so trùng với <code>A</code> , <code>b</code> , hoặc <code>C</code> .
<code>[^abc]</code>	So trùng với bất cứ ký tự nào không thuộc các ký tự trong nhóm, ví dụ <code>[^AbC]</code> sẽ không so trùng với <code>A</code> , <code>b</code> , or <code>C</code> nhưng so trùng với <code>B</code> , <code>F</code> ,...
<code>[a-z]</code>	So trùng với bất kỳ ký tự nào thuộc khoảng này, ví dụ <code>[A-C]</code> sẽ so trùng với <code>A</code> , <code>B</code> , hoặc <code>C</code> .
<code>()</code>	Xác định một biểu thức con sao cho nó được xem như một yếu tố đơn lẻ đối với các yếu tố được trình bày trong bảng này.
<code>?</code>	Xác định có một hoặc không có ký tự hoặc biểu thức con đứng trước nó, ví dụ <code>A?B</code> so trùng với <code>B</code> , <code>AB</code> , nhưng không so trùng với <code>AAB</code> .
<code>*</code>	Xác định không có hoặc có nhiều ký tự hoặc biểu thức con đứng trước nó, ví dụ <code>A*B</code> so trùng với <code>B</code> , <code>AB</code> , <code>AAB</code> , <code>AAAB</code> ,...
<code>+</code>	Xác định có một hoặc có nhiều ký tự hoặc biểu thức con đứng trước nó, ví dụ <code>A+B</code> so trùng với <code>AB</code> , <code>AAB</code> , <code>AAAB</code> ,... nhưng không so trùng với <code>B</code> .
<code>{n}</code>	Xác định có đúng <code>n</code> ký tự hoặc biểu thức con đứng trước nó, ví dụ <code>A{2}</code> chỉ so trùng với <code>AA</code> .
<code>{n, }</code>	Xác định có ít nhất <code>n</code> ký tự hoặc biểu thức con đứng trước nó, ví dụ <code>A{2, }</code> so trùng với <code>AA</code> , <code>AAA</code> , <code>AAAA</code> ,... nhưng không so trùng với <code>A</code> .
<code>{n, m}</code>	Xác định có từ <code>n</code> đến <code>m</code> ký tự đứng trước nó, ví dụ <code>A{2,4}</code> so trùng với <code>AA</code> , <code>AAA</code> , và <code>AAAA</code> nhưng không so trùng với <code>A</code> hoặc <code>AAAAA</code> .

Khi dữ liệu cần kiểm tra càng phức tạp thì cú pháp của biểu thức chính quy cũng càng phức tạp. Ví dụ, dễ dàng kiểm tra dữ liệu nhập chỉ chứa số hay có chiều dài tối thiểu, nhưng kiểm tra một *URL* khá phức tạp. Bảng 2.3 liệt kê một số biểu thức chính quy dùng để kiểm tra các kiểu dữ liệu thông dụng.

Bảng 2.3 Một số biểu thức chính quy thông dụng

Kiểu dữ liệu nhập	Mô tả	Biểu thức chính quy
Số	Chỉ chứa các chữ số thập phân; ví dụ 5, hoặc 5683874674.	<code>^\d+\$</code>
<i>PIN</i>	Chứa 4 chữ số thập phân, ví dụ 1234.	<code>^\d{4}\$</code>
Mật khẩu đơn giản	Chứa từ 6 đến 8 ký tự; ví dụ <code>ghTd6f</code> hoặc <code>b8c7hogh</code> .	<code>^\w{6,8}\$</code>
Số thẻ tín dụng	Chứa dữ liệu phù hợp với cấu trúc của hầu hết các loại số thẻ tín dụng, ví dụ 4921835221552042 hoặc 4921-8352-2155-2042.	<code>^\d{4}-?\d{4}-?\d{4}-?\d{4}\$</code>

Địa chỉ e-mail	<code>[\w-]+</code> nghĩa là chứa một hoặc nhiều ký tự word hoặc dấu gạch ngang, ví dụ <code>some-body@adatum.com</code>	<code>^[\w-]+@([\w-]+\.)+[\w-]+\$</code>
<i>HTTP</i> hoặc <i>HTTPS</i> URL	Dữ liệu là một URL dựa-trên-HTTP hay dựa-trên-HTTPS, ví dụ <code>http://www.microsoft.com</code>	<code>^https?:\/\/([\w-]+\.)+[\w-]+(\/[\w- %=]*)?\$</code>

Một khi đã biết cú pháp của biểu thức chính quy, bạn tạo một đối tượng `System.Text.RegularExpressions.Regex` bằng cách truyền cho phương thức khởi dựng của nó chuỗi chứa biểu thức chính quy. Sau đó, gọi phương thức `IsMatch` của đối tượng `Regex` và truyền chuỗi cần kiểm tra, phương thức này trả về một giá trị luận lý cho biết chuỗi có hợp lệ không. Cú pháp của biểu thức chính quy sẽ chỉ định `Regex` so trùng toàn bộ chuỗi hay chỉ so trùng một phần của chuỗi (xem `^`, `\A`, `$`, và `\z` trong bảng 2.2)

Phương thức `ValidateInput` dưới đây minh họa cách kiểm tra chuỗi nhập bằng biểu thức chính quy:

```
public static bool ValidateInput(string regex, string input) {

    // Tạo đối tượng Regex dựa trên biểu thức chính quy.
    Regex r = new Regex(regex);

    // Kiểm tra dữ liệu nhập có trùng với biểu thức chính quy hay không.
    return r.IsMatch(input);

}
```

Bạn có thể sử dụng đối tượng `Regex` để kiểm tra nhiều chuỗi, nhưng không thể thay đổi biểu thức chính quy được gán cho nó; bạn phải tạo một đối tượng `Regex` mới tương ứng với một cấu trúc mới. Phương thức `ValidateInput` ở trên tạo ra một đối tượng `Regex` mới mỗi lần được gọi, thay vào đó bạn có thể sử dụng phương thức tĩnh nạp chồng `IsMatch`.

```
public static bool ValidateInput(string regex, string input) {

    // Kiểm tra dữ liệu nhập có trùng với biểu thức chính quy hay không.
    return Regex.IsMatch(input, regex);

}
```

6.

Sử dụng biểu thức chính quy đã được biên dịch

?

Bạn cần giảm thiểu các tác động lên hiệu năng của ứng dụng khi các biểu thức chính quy phức tạp được sử dụng thường xuyên.



Khi khởi tạo đối tượng `System.Text.RegularExpressions.Regex`, hãy truyền thêm tùy chọn `Compiled` thuộc kiểu liệt kê `System.Text.RegularExpressions.RegexOptions` để biên dịch biểu thức chính quy thành *Microsoft Intermediate Language (MSIL)*.

Theo mặc định, khi bạn tạo đối tượng `Regex`, mẫu biểu thức chính quy do bạn xác định trong phương thức khởi dựng được biên dịch thành một dạng trung gian (không phải *MSIL*). Mỗi lần bạn sử dụng đối tượng `Regex`, bộ thực thi phiên dịch dạng trung gian này và áp dụng nó để kiểm tra chuỗi. Với các biểu thức chính quy phức tạp được sử dụng thường xuyên, việc phiên dịch lặp đi lại có thể gây tác động xấu lên hiệu năng của ứng dụng.

Khi tùy chọn `RegexOptions.Compiled` được chỉ định, bộ thực thi sẽ biên dịch biểu thức chính quy thành *MSIL*. *MSIL* này được gọi là mã just-in-time (*JIT*), được biên dịch thành mã máy nguyên sinh trong lần thực thi đầu tiên, giống như mã assembly thông thường. Biểu thức chính quy được biên dịch cũng được sử dụng giống như đối tượng `Regex`, việc biên dịch chỉ giúp thực thi nhanh hơn.

Tuy nhiên, việc biên dịch biểu thức chính quy cũng có vài nhược điểm. Trước tiên, trình biên dịch *JIT* phải làm việc nhiều hơn, dẫn đến chậm quá trình biên dịch, đặc biệt khi tạo biểu thức chính quy được biên dịch khi ứng dụng khởi động. Thứ hai, biểu thức chính quy được biên dịch vẫn tồn tại trong bộ nhớ khi không còn được sử dụng nữa, nó không bị bộ thu gom rác (*Garbage Collector*) xóa đi như các biểu thức chính quy thông thường. Vùng nhớ bị chiếm chỉ được giải phóng khi chương trình kết thúc, hoặc khi bạn giải phóng miền ứng dụng.

Dòng mã sau minh họa cách tạo một đối tượng `Regex` được biên dịch thành *MSIL*:

```
Regex reg = new Regex(@"[\w-]+@([\w-]+\.)+[\w-]+",
    RegexOptions.Compiled);
```

Ngoài ra, phương thức tĩnh `Regex.CompileToAssembly` cho phép bạn tạo một biểu thức chính quy được biên dịch và ghi nó vào một assembly khác. Nghĩa là bạn có thể tạo một assembly chứa các biểu thức chính quy để sử dụng cho nhiều ứng dụng sau này. Để biên dịch một biểu thức chính quy và lưu nó vào một assembly, thực hiện các bước sau:

1. Tạo một mảng `System.Text.RegularExpressions.RegexCompilationInfo` đủ lớn để chứa các đối tượng `RegexCompilationInfo`, mỗi đối tượng ứng với một biểu thức chính quy cần được biên dịch.
2. Tạo một đối tượng `RegexCompilationInfo` cho mỗi biểu thức chính quy và truyền đối số cho phương thức khởi dựng để xác định các thuộc tính của biểu thức chính quy này. Các thuộc tính thông dụng là:

- `IsPublic`— giá trị `bool` xác định lớp biểu thức chính quy được tạo ra có tầm vực là công khai hay không.
- `Name`— một `String` xác định tên của lớp.
- `Namespace`— một `String` xác định không gian tên của lớp.
- `Pattern`— một `String` xác định mẫu mà biểu thức chính quy sẽ so trùng (xem chi tiết ở mục 2.5).
- `Options`— một giá trị thuộc kiểu liệt kê `System.Text.RegularExpressions.RegexOptions` xác định các tùy chọn cho biểu thức chính quy.

3. Tạo một đối tượng `System.Reflection.AssemblyName` để xác định tên của assembly mà phương thức `Regex.CompileToAssembly` sẽ tạo ra.
4. Gọi phương thức `Regex.CompileToAssembly`, truyền các đối số là mảng `RegexCompilationInfo` và đối tượng `AssemblyName`.

Quá trình trên tạo ra một assembly chứa các khai báo lớp cho từng biểu thức chính quy được biên dịch, mỗi lớp dẫn xuất từ `Regex`. Để sử dụng một biểu thức chính quy đã được biên dịch trong assembly, bạn cần tạo đối tượng biểu thức chính quy này và gọi các phương thức của nó giống như khi tạo nó với phương thức khởi dựng `Regex` bình thường. Bạn nhớ thêm tham chiếu tới assembly khi sử dụng các lớp biểu thức chính quy nằm trong nó.

Đoạn mã sau minh họa cách tạo một assembly có tên là *MyRegex.dll*, chứa hai biểu thức chính quy có tên là `PinRegex` và `CreditCardRegex`:

```
using System.Text.RegularExpressions;
using System.Reflection;

public class CompiledRegexExample {

    public static void Main() {

        // Tạo mảng chứa các đối tượng RegexCompilationInfo.
        RegexCompilationInfo[] regexInfo = new RegexCompilationInfo[2];

        // Tạo đối tượng RegexCompilationInfo cho PinRegex.
        regexInfo[0] = new RegexCompilationInfo(@"^\d{4}$",
            RegexOptions.Compiled, "PinRegex", "", true);

        // Tạo đối tượng RegexCompilationInfo cho CreditCardRegex.
        regexInfo[1] = new RegexCompilationInfo(
            @"^\d{4}-?\d{4}-?\d{4}-?\d{4}$",
            RegexOptions.Compiled, "CreditCardRegex", "", true);

        // Tạo đối tượng AssemblyName để định nghĩa assembly.
        AssemblyName assembly = new AssemblyName();
        assembly.Name = "MyRegEx";

        // Tạo các biểu thức chính quy được biên dịch.
        Regex.CompileToAssembly(regexInfo, assembly);
    }
}
```

```
}
}
```

7. Tạo ngày và giờ từ chuỗi

? Bạn cần tạo một thể hiện `System.DateTime` mô tả giờ, ngày được chỉ định trong một chuỗi.

✕ Sử dụng phương thức `Parse` hoặc `ParseExact` của lớp `DateTime`.

Có nhiều cách mô tả ngày, giờ; ví dụ *1st June 2004*, *1/6/2004*, *6/1/2004*, *1-Jun-2004* cùng chỉ một ngày; *16:43* và *4:43 PM* cùng chỉ một giờ. Lớp `DateTime` cung cấp phương thức tĩnh `Parse` rất linh hoạt, cho phép tạo thể hiện `DateTime` từ nhiều cách mô tả khác nhau trong chuỗi.

Phương thức `Parse` rất mạnh trong việc tạo đối tượng `DateTime` từ một chuỗi cho trước. Nó có thể xử lý một chuỗi chỉ chứa một phần thông tin hay chứa thông tin sai, và thay thế các giá trị thiếu bằng các giá trị mặc định. Ngày mặc định là ngày hiện tại, giờ mặc định là *12:00:00 AM*. Nếu sau mọi cố gắng, `Parse` không thể tạo đối tượng `DateTime`, nó sẽ ném ngoại lệ `System.FormatException`. Ví dụ sau minh họa tính linh hoạt của `Parse`:

```
// 01/09/2004 12:00:00 AM
DateTime dt1 = DateTime.Parse("Sep 2004");

// 05/09/2004 02:15:33 PM
DateTime dt2 = DateTime.Parse("Sun 5 September 2004 14:15:33");

// 05/09/2004 12:00:00 AM
DateTime dt3 = DateTime.Parse("5,9,04");

// 05/09/2004 02:15:33 PM
DateTime dt4 = DateTime.Parse("5/9/2004 14:15:33");

// 07/10/2004 02:15:00 PM (giả sử ngày hiện tại là 07/10/2004)
DateTime dt5 = DateTime.Parse("2:15 PM");
```

Phương thức `Parse` linh hoạt và có thể tự sửa lỗi. Tuy nhiên, mức độ linh hoạt này không cần thiết trong trường hợp bạn muốn bảo đảm các chuỗi phải theo một định dạng nhất định. Khi đó, sử dụng phương thức `ParseExact` thay cho `Parse`. Dạng đơn giản nhất của `ParseExact` nhận ba đối số: chuỗi chứa ngày giờ, chuỗi định dạng xác định cấu trúc mà chuỗi chứa ngày giờ phải tuân theo, và một tham chiếu `IFormatProvider` cung cấp thông tin đặc thù về bản địa. Nếu `IFormatProvider` là `null`, thông tin về bản địa của tiến trình (*thread*) hiện hành sẽ được sử dụng.

Nếu ngày giờ trong chuỗi đang xét không đúng với định dạng quy định, `ParseExact` sẽ ném ngoại lệ `System.FormatException`. Chuỗi định dạng được sử dụng tương tự như khi bạn chỉ

định chuỗi đại diện cho một đối tượng `DateTime`. Điều này có nghĩa là bạn có thể sử dụng cả định dạng chuẩn lẫn định dạng tùy biến. Tham khảo phần tài liệu cho lớp `System.Globalization.DateTimeFormatInfo` trong tài liệu *.NET Framework SDK* để có thông tin đầy đủ về tất cả các kiểu định dạng.

```
// Chỉ phân tích các chuỗi chứa LongTimePattern.
DateTime dt6 = DateTime.ParseExact("2:13:30 PM",
    "h:mm:ss tt", null);

// Chỉ phân tích các chuỗi chứa RFC1123Pattern.
DateTime dt7 = DateTime.ParseExact(
    "Sun, 05 Sep 2004 14:13:30 GMT",
    "ddd, dd MMM yyyy HH':'mm':'ss 'GMT'", null);

// Chỉ phân tích các chuỗi chứa MonthDayPattern.
DateTime dt8 = DateTime.ParseExact("September 03",
    "MMMM dd", null);
```

8.

Cộng, trừ, so sánh ngày giờ



Bạn cần thực hiện các phép tính số học cơ bản hay phép so sánh trên ngày, giờ.



Sử dụng các cấu trúc `DateTime` và `TimeSpan` (hỗ trợ các toán tử số học và so sánh).

Một đối tượng `DateTime` mô tả một thời điểm xác định (chẳng hạn 4:15 AM, ngày 21 tháng 04 năm 1980), trong khi đối tượng `TimeSpan` mô tả một khoảng thời gian (chẳng hạn 2 giờ, 35 phút). Bạn có thể cộng, trừ, so sánh các đối tượng `TimeSpan` và `DateTime`.

Thực chất, cả `DateTime` và `TimeSpan` đều sử dụng tick để mô tả thời gian—1 tick bằng 100 nano-giây (một nano-giây bằng một phần tỷ (10^{-9}) giây). `TimeSpan` lưu khoảng thời gian của nó là số tick bằng khoảng thời gian đó, `DateTime` lưu số *tick* đã trôi qua kể từ 12:00:00 khuya ngày 1 tháng 1 năm 0001 sau công nguyên. Cách tiếp cận này và việc sử dụng toán tử nạp chồng giúp `DateTime` và `TimeSpan` dễ dàng hỗ trợ các phép tính số học và so sánh. Bảng 2.4 tóm tắt các toán tử mà hai cấu trúc này hỗ trợ.

Bảng 2.4 Các toán tử được cung cấp bởi `DateTime` và `TimeSpan`

Toán tử	<code>TimeSpan</code>	<code>DateTime</code>
Gán (=)	Vì <code>TimeSpan</code> là một cấu trúc nên phép gán trả về một bản sao, không phải một tham chiếu.	Vì <code>DateTime</code> là một cấu trúc nên phép gán trả về một bản sao, không phải một tham chiếu.

Cộng (+)	Cộng hai đối tượng <code>TimeSpan</code> .	Cộng một <code>TimeSpan</code> vào một <code>DateTime</code> .
Trừ (-)	Trừ hai đối tượng <code>TimeSpan</code> .	Trừ một <code>DateTime</code> cho một <code>DateTime</code> hoặc một <code>TimeSpan</code> .
Bằng (==)	So sánh hai đối tượng <code>TimeSpan</code> và trả về <code>true</code> nếu bằng nhau.	So sánh hai đối tượng <code>DateTime</code> và trả về <code>true</code> nếu bằng nhau.
Không bằng (!=)	So sánh hai đối tượng <code>TimeSpan</code> và trả về <code>true</code> nếu không bằng nhau.	So sánh hai đối tượng <code>DateTime</code> và trả về <code>true</code> nếu không bằng nhau.
Lớn hơn (>)	Xác định một đối tượng <code>TimeSpan</code> có lớn hơn một đối tượng <code>TimeSpan</code> khác hay không.	Xác định một đối tượng <code>DateTime</code> có lớn hơn một đối tượng <code>DateTime</code> khác hay không.
Lớn hoặc bằng (>=)	Xác định một đối tượng <code>TimeSpan</code> có lớn hơn hoặc bằng một đối tượng <code>TimeSpan</code> khác hay không.	Xác định một đối tượng <code>DateTime</code> có lớn hơn hoặc bằng một đối tượng <code>DateTime</code> khác hay không.
Nhỏ hơn (<)	Xác định một đối tượng <code>TimeSpan</code> có nhỏ hơn một đối tượng <code>TimeSpan</code> khác hay không.	Xác định một đối tượng <code>DateTime</code> có nhỏ hơn một đối tượng <code>DateTime</code> khác hay không.
Nhỏ hoặc bằng (<=)	Xác định một đối tượng <code>TimeSpan</code> có nhỏ hơn hoặc bằng một đối tượng <code>TimeSpan</code> khác hay không.	Xác định một đối tượng <code>DateTime</code> có nhỏ hơn hoặc bằng một đối tượng <code>DateTime</code> khác hay không.
Âm (-)	Trả về một giá trị đảo dấu của một <code>TimeSpan</code> .	Không hỗ trợ.
Dương (+)	Trả về chính <code>TimeSpan</code> .	Không hỗ trợ.

Cấu trúc `DateTime` cũng hiện thực các phương thức `AddTicks`, `AddMilliseconds`, `AddSeconds`, `AddMinutes`, `AddHours`, `AddDays`, `AddMonths`, và `AddYears`. Mỗi phương thức này cho phép bạn cộng (hoặc trừ bằng các giá trị âm) phần tử thời gian thích hợp với đối tượng `DateTime`. Các phương thức này và các toán tử được liệt kê trong bảng 2.4 không làm thay đổi `DateTime` gốc —thay vào đó chúng sẽ tạo một đối tượng mới với giá trị đã được thay đổi. Đoạn mã dưới đây trình bày cách sử dụng các toán tử để thao tác các cấu trúc `DateTime` và `TimeSpan`:

```
// Tạo một TimeSpan mô tả 2.5 ngày.
TimeSpan timespan1 = new TimeSpan(2,12,0,0);
// Tạo một TimeSpan mô tả 4.5 ngày.
TimeSpan timespan2 = new TimeSpan(4,12,0,0);
// Tạo một TimeSpan mô tả 1 tuần.
TimeSpan oneWeek = timespan1 + timespan2;

// Tạo một DateTime với ngày giờ hiện tại.
DateTime now = DateTime.Now;
```

```
// Tạo một DateTime mô tả 1 tuần trước đây.
DateTime past = now - oneWeek;

// Tạo một DateTime mô tả 1 tuần trong tương lai.
DateTime future = now + oneWeek;
```

9.

Sắp xếp một mảng hoặc một ArrayList



Bạn cần sắp xếp các phần tử trong một mảng hoặc một ArrayList.



Sử dụng phương thức `ArrayList.Sort` để sắp xếp ArrayList và phương thức tĩnh `Array.Sort` để sắp xếp mảng.

Dạng đơn giản nhất của `Sort` là sắp xếp các đối tượng nằm trong một mảng hoặc `ArrayList` khi các đối tượng này có hiện thực giao diện `System.IComparable` và có kiểu giống nhau—tất cả các kiểu dữ liệu cơ bản đều hiện thực `IComparable`. Đoạn mã dưới đây minh họa cách sử dụng phương thức `Sort`:

```
// Tạo một mảng mới và thêm phần tử vào.
int[] array = {4, 2, 9, 3};

// Sắp xếp mảng.
Array.Sort(array);

// Hiển thị nội dung của mảng đã được sắp xếp.
foreach (int i in array) { Console.WriteLine(i);}

// Tạo một ArrayList mới và thêm phần tử vào.
ArrayList list = new ArrayList(4);
list.Add("Phong");
list.Add("Phuong");
list.Add("Khoa");
list.Add("Tam");

// Sắp xếp ArrayList.
list.Sort();

// Hiển thị nội dung của ArrayList đã được sắp xếp.
foreach (string s in list) { Console.WriteLine(s);}
```

Để sắp xếp các đối tượng không hiện thực `Comparable`, bạn cần truyền cho phương thức `Sort` một đối tượng hiện thực giao diện `System.Collections.IComparer`. Hiện thực của `IComparer` phải có khả năng so sánh các đối tượng nằm trong mảng hoặc `ArrayList` (xem mục 16.3 để biết cách hiện thực `Comparable` và `IComparer`).

10.

Chép một tập hợp vào một mảng

?

Bạn cần chép nội dung của một tập hợp vào một mảng.

✖

Sử dụng phương thức `ICollection.CopyTo` (được hiện thực bởi tất cả các lớp tập hợp), hoặc sử dụng phương thức `ToArray` (được hiện thực bởi các tập hợp `ArrayList`, `Stack`, `Queue`).

Các phương thức `ICollection.CopyTo` và `ToArray` có cùng chức năng, chúng chép các phần tử trong một tập hợp vào một mảng. Sự khác biệt nằm ở chỗ `CopyTo` chép vào một mảng đã có, trong khi `ToArray` tạo ra một mảng mới rồi chép vào đó.

`CopyTo` nhận hai đối số: một mảng và một chỉ số. Mảng này là đích của quá trình sao chép và phải có kiểu tương thích với các phần tử của tập hợp. Nếu kiểu không tương thích hay không có sự chuyển đổi ngầm từ kiểu phần tử của tập hợp sang kiểu phần tử của mảng thì ngoại lệ `System.InvalidCastException` sẽ bị ném. Chỉ số là một vị trí trong mảng mà bắt đầu từ đó các phần tử của tập hợp sẽ được chép vào. Nếu chỉ số này lớn hơn hoặc bằng chiều dài của mảng, hoặc số phần tử của tập hợp vượt quá sức chứa của mảng thì ngoại lệ `System.ArgumentException` sẽ bị ném. Đoạn mã sau minh họa cách sử dụng `CopyTo` để chép nội dung của một `ArrayList` vào một mảng:

```
// Tạo một ArrayList mới và thêm phần tử vào.
ArrayList list = new ArrayList(5);
list.Add("Phuong");
list.Add("Phong");
list.Add("Nam");
list.Add("Tam");
list.Add("Nhan");

// Tạo một string[] và sử dụng ICollection.CopyTo
// để chép nội dung của ArrayList.
string[] array1 = new string[5];
list.CopyTo(array1, 0);
```

Các lớp `ArrayList`, `Stack`, và `Queue` cũng hiện thực phương thức `ToArray`, phương thức này tự động tạo một mảng với kích thước đủ để chứa các phần tử của của tập hợp. Nếu bạn không truyền đối số cho `ToArray`, nó sẽ trả về một `object[]` bất chấp kiểu của các đối tượng trong tập hợp. Tuy nhiên, bạn có thể truyền một đối tượng `System.Type` để chỉ định kiểu của mảng (Bạn phải ép mảng kiểu mạnh về đúng kiểu). Ví dụ sau minh họa cách sử dụng `ToArray` cho `ArrayList` ở trên:

```
// Sử dụng ArrayList.ToArray để tạo một object[]
// từ nội dung của tập hợp.
object[] array2 = list.ToArray();

// Sử dụng ArrayList.ToArray để tạo một string[] kiểu mạnh
// từ nội dung của tập hợp.
string[] array3 =
    (string[])list.ToArray(System.Type.GetType("System.String"));
```

11.

Tạo một tập hợp kiểu mạnh



Bạn cần tạo một tập hợp chỉ chứa các phần tử thuộc một kiểu nhất định.



Tạo một lớp dẫn xuất từ lớp `System.Collections.CollectionBase` hay `System.Collections.DictionaryBase`, và hiện thực các phương thức an-toàn-về-kiểu-dữ-liệu (*type-safe*) để thao tác trên tập hợp.

Các lớp `CollectionBase` và `DictionaryBase` có thể đóng vai trò các lớp cơ sở để dẫn xuất ra các lớp tập hợp an-toàn-kiểu mà không phải hiện thực lại các giao diện chuẩn: `IDictionary`, `ICollection`, và `IEnumerable`.

- `CollectionBase`— dùng cho các tập hợp dựa-trên-`ICollection` (như `ArrayList`). Thực chất, `CollectionBase` duy trì tập hợp bằng một đối tượng `ArrayList` chuẩn, có thể được truy xuất thông qua thuộc tính bảo vệ `List`.
- `DictionaryBase`— dùng cho các tập hợp dựa-trên-`IDictionary` (như `Hashtable`). Thực chất, `DictionaryBase` duy trì tập hợp bằng một đối tượng `Hashtable` chuẩn, có thể được truy xuất thông qua thuộc tính bảo vệ `Dictionary`.

Đoạn mã sau hiện thực một tập hợp tên mạnh (dựa trên lớp `CollectionBase`) để thể hiện một danh sách các đối tượng `System.Reflection.AssemblyName`.

```
using System.Reflection;
using System.Collections;

public class AssemblyNameList : CollectionBase {

    public int Add(AssemblyName value) {

        return this.List.Add(value);
    }

    public void Remove(AssemblyName value) {
```



```

        this.List.Remove(value);
    }

    public AssemblyName this[int index] {

        get {
            return (AssemblyName)this.List[index];
        }

        set {
            this.List[index] = value;
        }
    }

    public bool Contains(AssemblyName value) {

        return this.List.Contains(value);
    }

    public void Insert(int index, AssemblyName value) {

        this.List.Insert(index, value);
    }
}

```

Cả hai lớp `CollectionBase` và `DictionaryBase` đều hiện thực một tập các phương thức được-bảo-vệ có tiếp đầu ngữ `On*`. Các phương thức này (chẳng hạn `OnClear`, `OnClearComplete`, `OnGet`, `OnGetComplete`,...) thường được chép đè ở các lớp dẫn xuất nhằm cho phép bạn hiện thực các chức năng tùy biến cần thiết để quản lý tập hợp kiểu mạnh. Các lớp `CollectionBase` và `DictionaryBase` sẽ gọi phương thức phù hợp trước và sau khi việc chỉnh sửa được thực hiện trên tập hợp nằm dưới thông qua thuộc tính `List` hay `Dictionary`.

12.

Lưu một đối tượng khả-tuần-tự-hóa vào file

?

Bạn cần lưu một đối tượng khả-tuần-tự-hóa và các trạng thái của nó vào file, sau đó giải tuần tự hóa khi cần.

✕

Sử dụng một *formatter* để tuần tự hóa đối tượng và ghi nó vào một `System.IO.FileStream`. Khi cần truy xuất đối tượng, sử dụng *formatter* cùng kiểu để đọc dữ liệu được-tuần-tự-hóa từ file và giải tuần tự hóa đối tượng. Thư viện

lớp *.NET Framework* cung cấp các hiện thực *formatter* sau đây để tuần tự hóa đối tượng theo dạng nhị phân hay *SOAP*:

- `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter`
- `System.Runtime.Serialization.Formatters.Soap.SoapFormatter`

Lớp `BinaryFormatter` và `SoapFormatter` có thể được sử dụng để tuần tự hóa một đối tượng của bất kỳ kiểu nào được gắn với đặc tính `System.SerializableAttribute`. `BinaryFormatter` sinh ra một stream dữ liệu nhị phân mô tả đối tượng và trạng thái của nó, trong khi `SoapFormatter` sinh ra một tài liệu *SOAP*.

Cả hai lớp `BinaryFormatter` và `SoapFormatter` đều hiện thực giao diện `System.Runtime.Serialization.IFormatter`, giao diện này định nghĩa hai phương thức: `Serialize` và `Deserialize`.

- `Serialize`— nhận một tham chiếu `System.Object` và một tham chiếu `System.IO.Stream` làm đối số, tuần tự hóa `Object` và ghi nó vào `Stream`.
- `Deserialize`— nhận một tham chiếu `Stream` làm đối số, đọc dữ liệu của đối tượng được-tuần-tự-hóa từ `Stream`, và trả về một tham chiếu `Object` đến đối tượng được-giải-tuần-tự-hóa. Bạn phải ép tham chiếu `Object` này về kiểu thích hợp.

Để gọi các phương thức `Serialize` và `Deserialize` của lớp `BinaryFormatter`, mã lệnh của bạn phải được cấp phần tử `SerializationFormatter` của lớp `System.Security.Permissions.SecurityPermission`.

Để gọi các phương thức `Serialize` và `Deserialize` của lớp `SoapFormatter`, mã lệnh của bạn phải được cấp quyền “tin tưởng tuyệt đối” (*full trust*) vì assembly `System.Runtime.Serialization.Formatters.Soap.dll` (lớp `SoapFormatter` được khai báo bên trong assembly này) không cho phép các mã lệnh chỉ được-tin-cậy-một-phần (*partially trusted caller*) sử dụng nó. Tham khảo mục 13.1 để có thêm thông tin về mã lệnh được-tin-cậy-một-phần.

Lớp `BinarySerializationExample` dưới đây minh họa cách sử dụng lớp `BinaryFormatter` để tuần tự hóa một `System.Collections.ArrayList` chứa danh sách tên người vào một file. Sau đó, `ArrayList` được giải tuần tự hóa từ file và nội dung của nó sẽ được hiển thị trong cửa sổ *Console*.

```
using System.IO;
using System.Collections;
using System.Runtime.Serialization.Formatters.Binary;

public class BinarySerializationExample {

    public static void Main() {

        // Tạo và cấu hình ArrayList để tuần tự hóa.
        ArrayList people = new ArrayList();
        people.Add("Phuong");
```

```

people.Add("Phong");
people.Add("Nam");

// Tuần tự hóa đối tượng ArrayList.
FileStream str = File.Create("people.bin");
BinaryFormatter bf = new BinaryFormatter();
bf.Serialize(str, people);
str.Close();

// Giải tuần tự hóa đối tượng ArrayList.
str = File.OpenRead("people.bin");
bf = new BinaryFormatter();
people = (ArrayList)bf.Deserialize(str);
str.Close();

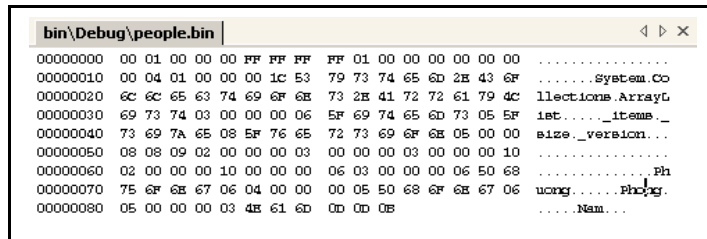
// Hiển thị nội dung của đối tượng ArrayList
// đã-được-giải-tuần-tự-hóa.
foreach (string s in people) {

    System.Console.WriteLine(s);
}
}

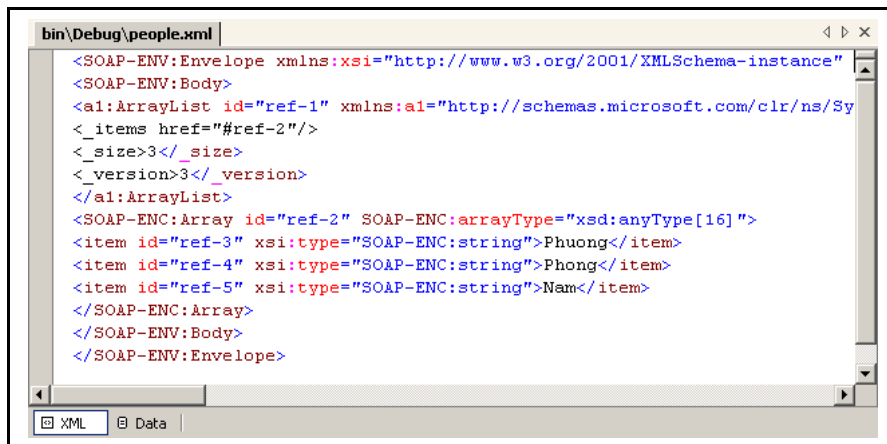
```

Bạn có thể sử dụng lớp `SoapFormatter` theo cách như được trình bày trong lớp `BinarySerializationExample` ở trên, chỉ cần thay mỗi thể hiện của lớp `BinaryFormatter` bằng thể hiện của lớp `SoapFormatter` và thay đổi chỉ thị `using` để nhập không gian tên `System.Runtime.Serialization.Formatters.Soap`. Ngoài ra, bạn cần thêm một tham chiếu đến `System.Runtime.Serialization.Formatters.Soap.dll` khi biên dịch mã. File `SoapSerializationExample.cs` trong đĩa CD đính kèm sẽ trình bày cách sử dụng lớp `SoapFormatter`.

Hình 2.1 và 2.2 dưới đây minh họa hai kết quả khác nhau khi sử dụng lớp `BinaryFormatter` và `SoapFormatter`. Hình 2.1 trình bày nội dung của file `people.bin` được tạo ra khi sử dụng `BinaryFormatter`, hình 2.2 trình bày nội dung của file `people.xml` được tạo ra khi sử dụng `SoapFormatter`.



Hình 2.1 Nội dung file people.bin



Hình 2.2 Nội dung file people.xml

3

MIỀN ỨNG DỤNG,
CƠ CHẾ PHẢN CHIẾU,
VÀ SIÊU DỮ LIỆU

Sức mạnh và tính linh hoạt của *Microsoft .NET Framework* được nâng cao bởi khả năng kiểm tra và thao tác các kiểu và siêu dữ liệu lúc thực thi. Các mục trong chương này sẽ trình bày các khía cạnh thông dụng của miền ứng dụng (*application domain*), cơ chế phản chiếu (*reflection*), và siêu dữ liệu (*metadata*), bao gồm:

- Tạo và hủy các miền ứng dụng (mục 3.1 và 3.9).
- Làm việc với các kiểu và các đối tượng khi sử dụng nhiều miền ứng dụng (mục 3.2, 3.3, 3.4, và 3.8).
- Làm việc với thông tin `Type` (mục 3.10 và 3.11).
- Tạo động các đối tượng và nạp động các assembly lúc thực thi (mục 3.5, 3.6, 3.7, và 3.12).
- Tạo và kiểm tra các đặc tính tùy biến (các mục 3.13 và 3.14).

1.

Tạo miền ứng dụng



Bạn cần tạo một miền ứng dụng mới.



Sử dụng phương thức tĩnh `CreateDomain` của lớp `System.AppDomain`.

Dạng thức đơn giản nhất của phương thức `CreateDomain` nhận một đối số kiểu `string` chỉ định tên thân thiện cho miền ứng dụng mới. Các dạng thức khác cho phép bạn chỉ định chứng cứ (*evidence*) và các thiết lập cấu hình cho miền ứng dụng mới. Chứng cứ được chỉ định bằng đối tượng `System.Security.Policy.Evidence`; mục 13.11 trình bày các tác động của chứng cứ khi bạn tạo một miền ứng dụng. Các thiết lập cấu hình được chỉ định bằng đối tượng `System.AppDomainSetup`.

Lớp `AppDomainSetup` chứa các thông tin cấu hình cho một miền ứng dụng. Bảng 3.1 liệt kê các thuộc tính thường được sử dụng nhất của lớp `AppDomainSetup` khi tạo các miền ứng dụng. Các thuộc tính này có thể được truy xuất sau khi tạo thông qua các thành viên của đối tượng `AppDomain`, và một số có thể thay đổi lúc thực thi; bạn hãy tham khảo tài liệu *.NET Framework SDK* về lớp `AppDomain` để hiểu chi tiết hơn.

Bảng 3.1 Các thuộc tính thông dụng của lớp `AppDomainSetup`

Thuộc tính	Mô tả
<code>ApplicationBase</code>	Thư mục mà <i>CRL</i> sẽ xét trong quá trình dò tìm các assembly riêng. Kỹ thuật dò tìm (<i>probing</i>) sẽ được thảo luận trong mục 3.5. Thực tế, <code>ApplicationBase</code> là thư mục gốc cho ứng dụng đang thực thi. Theo mặc định, đây là thư mục chứa assembly. Có thể đọc được thuộc tính này sau khi tạo miền ứng dụng bằng thuộc tính <code>AppDomain.BaseDirectory</code> .

ConfigurationFile	Tên của file cấu hình, được sử dụng bởi mã đã được nạp vào miền ứng dụng. Có thể đọc được thuộc tính này sau khi tạo miền ứng dụng bằng phương thức <code>AppDomain.GetData</code> với khóa <code>APP_CONFIG_FILE</code> .
DisallowPublisherPolicy	Quy định phần <i>publisher policy</i> của file cấu hình ứng dụng có được xét đến hay không khi xác định phiên bản của một assembly tên mạnh để nối kết. <i>Publisher policy</i> sẽ được thảo luận trong mục 3.5.
PrivateBinPath	Danh sách các thư mục cách nhau bởi dấu chấm phẩy mà bộ thực thi sẽ sử dụng khi dò tìm các assembly riêng. Các thư mục này có vị trí tương đối so với thư mục được chỉ định trong <code>ApplicationBase</code> . Có thể đọc được thuộc tính này sau khi tạo miền ứng dụng bằng thuộc tính <code>AppDomain.RelativeSearchPath</code> . Có thể thay đổi thuộc tính này lúc thực thi bằng phương thức <code>AppendPrivatePath</code> và <code>ClearPrivatePath</code> .

Ví dụ dưới đây trình bày cách tạo và cấu hình một miền ứng dụng:

```
// Khởi tạo một đối tượng của lớp AppDomainSetup.
AppDomainSetup setupInfo = new AppDomainSetup();

// Cấu hình các thông tin cài đặt cho miền ứng dụng.
setupInfo.ApplicationBase = @"C:\MyRootDirectory";
setupInfo.ConfigurationFile = "MyApp.config";
setupInfo.PrivateBinPath = "bin;plugins;external";

// Tạo một miền ứng dụng mới (truyền null làm đối số chúng cú).
// Nhớ lưu một tham chiếu đến AppDomain mới vì nó
// không thể được thu lấy theo bất kỳ cách nào khác.
AppDomain newDomain = AppDomain.CreateDomain(
    "My New AppDomain",
    new System.Security.Policy.Evidence(),
    setupInfo);
```



Bạn phải duy trì một tham chiếu đến đối tượng `AppDomain` vừa tạo bởi vì không có cơ chế nào để liệt kê các miền ứng dụng hiện có từ bên trong mã được-quản-lý.

2. Chuyển các đối tượng qua lại các miền ứng dụng

? Bạn cần chuyển các đối tượng qua lại giữa các miền ứng dụng như các đối số hay các giá trị trả về.

✕ Sử dụng các đối tượng *marshal-by-value* hay *marshal-by-reference*.

Hệ thống *.NET Remoting* (sẽ được thảo luận trong chương 12) giúp việc gọi các đối tượng qua lại các miền ứng dụng trở nên dễ dàng. Tuy nhiên, nếu bạn chưa quen với *.NET Remoting*, kết quả có thể rất khác so với mong đợi. Thực ra, vấn đề gây khó khăn khi dùng nhiều miền ứng dụng là sự tương tác với *.NET Remoting* và cách thức luân chuyển đối tượng qua các miền ứng dụng.

Tất cả các kiểu dữ liệu có thể chia thành ba loại: *nonremotable*, *marshal-by-value (MBV)*, và *marshal-by-reference (MBR)*. Kiểu *nonremotable* không thể vượt qua biên miền ứng dụng và không thể dùng làm các đối số hay các giá trị trả về của các lời gọi trong môi trường liên miền ứng dụng. Kiểu *nonremotable* sẽ được thảo luận trong mục 3.4.

Kiểu *MBV* là kiểu khả-tuần-tự-hóa. Khi một đối tượng kiểu *MBV* được chuyển qua một miền ứng dụng khác như là đối số hay giá trị trả về, hệ thống *.NET Remoting* sẽ tuần tự hóa trạng thái hiện tại của đối tượng, chuyển dữ liệu đó sang miền ứng dụng đích, và tạo một bản sao của đối tượng với cùng trạng thái như đối tượng gốc. Kết quả là tồn tại bản sao của đối tượng ở cả hai miền ứng dụng. Hai đối tượng này ban đầu giống nhau hoàn toàn, nhưng độc lập nhau, nên việc thay đổi đối tượng này không ảnh hưởng đến đối tượng kia. Dưới đây là ví dụ một kiểu khả-tuần-tự-hóa có tên là `Employee`, được chuyển qua một miền ứng dụng khác bằng trị (xem mục 16.1 để biết cách tạo kiểu khả-tuần-tự-hóa).

```
[System.Serializable]
public class Employee {

    // Hiện thực các thành viên ở đây.

    $

}
```

Kiểu *MBR* là lớp dẫn xuất từ lớp `System.MarshalByRefObject`. Khi một đối tượng kiểu *MBR* được chuyển qua một miền ứng dụng khác như đối số hay giá trị trả về, hệ thống *.NET Remoting* sẽ tạo một đối tượng proxy cho đối tượng *MBV* cần chuyển trong miền ứng dụng đích. Đối tượng đại diện thực hiện các hành vi hoàn toàn giống với đối tượng *MBR* mà nó đại diện. Thực ra, khi thực hiện một hành vi trên đối tượng đại diện, hệ thống *.NET Remoting* thực hiện ngầm việc chuyển lời gọi và các đối số cần thiết đến miền ứng dụng nguồn, và tại đó thực hiện lời gọi hàm trên đối tượng *MBR* gốc. Kết quả được trả về thông qua đối tượng đại diện. Dưới đây là một phiên bản khác của lớp `Employee`, được chuyển qua một miền ứng dụng khác bằng tham chiếu thay vì bằng trị (xem mục 12.7 để biết chi tiết về cách tạo kiểu *MBR*).

```
public class Employee : System.MarshalByRefObject {
```

```
// Hiện thực các thành viên ở đây.
$
}
```

3. *Tránh nạp các assembly không cần thiết vào miền ứng dụng*

? Bạn cần chuyển một tham chiếu đối tượng qua lại giữa các miền ứng dụng khác nhau; tuy nhiên, bạn không muốn CLR nạp siêu dữ liệu mô tả kiểu của đối tượng vào các miền ứng dụng trung gian.

X Đóng gói tham chiếu đối tượng trong một `System.Runtime.Remoting.ObjectHandle` và khi cần truy xuất đối tượng thì khôi phục lại.

Khi bạn truyền một đối tượng *marshal-by-value* (MBV) qua các miền ứng dụng, bộ thực thi sẽ tạo một thể hiện mới của đối tượng này trong miền ứng dụng đích. Điều này có nghĩa là bộ thực thi phải nạp assembly chứa siêu dữ liệu mô tả kiểu của đối tượng vào các miền ứng dụng. Do đó, việc truyền các tham chiếu MBV qua các miền ứng dụng trung gian sẽ dẫn đến việc bộ thực thi nạp các assembly không cần thiết vào các miền ứng dụng này. Một khi đã được nạp thì các assembly thừa này sẽ không được giải phóng khỏi miền ứng dụng nếu không giải phóng cả miền ứng dụng chứa chúng (xem mục 3.9).

Lớp `ObjectHandle` cho phép bạn đóng gói tham chiếu đối tượng để truyền qua các miền ứng dụng mà bộ thực thi không phải nạp thêm assembly. Khi đối tượng này đến miền ứng dụng đích, bạn có thể khôi phục tham chiếu đối tượng, bộ thực thi sẽ nạp các assembly cần thiết và cho phép bạn truy xuất đến đối tượng như bình thường. Để đóng gói một đối tượng (ví dụ `System.Data.DataSet`), bạn có thể thực hiện như sau:

```
// Tạo một DataSet mới.
System.Data.DataSet data1 = new System.Data.DataSet();

// Cấu hình/thêm dữ liệu cho DataSet.
$

// Đóng gói DataSet.
System.Runtime.Remoting.ObjectHandle objHandle =
    new System.Runtime.Remoting.ObjectHandle(data1);
```

Để khôi phục một đối tượng, sử dụng phương thức `ObjectHandle.Unwrap` và ép kiểu trả về cho phù hợp, ví dụ:

```
// Khôi phục DataSet từ ObjectHandle.
System.Data.DataSet data2 =
    (System.Data.DataSet)objHandle.Unwrap();
```

4. Tạo kiểu không thể vượt qua biên miền ứng dụng

- ?** Bạn cần tạo một kiểu dữ liệu sao cho các thể hiện của kiểu này không thể được truy xuất từ mã lệnh ở các miền ứng dụng khác.
- ✕** Phải chắc chắn kiểu dữ liệu thuộc dạng *nonremotable*, tức là không thể tuần tự hóa cũng như không dẫn xuất từ lớp `MarshalByRefObject`.

Đôi khi bạn muốn kiểu dữ liệu nào đó chỉ được giới hạn truy xuất trong phạm vi của miền ứng dụng. Để tạo kiểu dữ liệu dạng *nonremotable*, phải chắc rằng kiểu này không phải là khả-tuần-tự-hóa và cũng không dẫn xuất (trực tiếp hay gián tiếp) từ lớp `MarshalByRefObject`. Những điều kiện này sẽ đảm bảo rằng trạng thái của đối tượng không thể được truy xuất từ các miền ứng dụng khác (các đối tượng này không thể được sử dụng làm đối số hay giá trị trả về trong các lời gọi phương thức liên miền ứng dụng).

Điều kiện kiểu dữ liệu không phải là khả-tuần-tự-hóa được thực hiện dễ dàng do một lớp không thừa kế khả năng tuần tự hóa từ lớp cha của nó. Để bảo đảm một kiểu không phải là khả-tuần-tự-hóa, bạn phải chắc chắn rằng đặc tính `System.SerializableAttribute` không được áp dụng khi khai báo kiểu.

Bạn cần lưu ý khi đảm bảo một lớp không được truyền bằng tham chiếu. Nhiều lớp trong thư viện lớp `.NET` dẫn xuất trực tiếp hay gián tiếp từ `MarshalByRefObject`; bạn phải cẩn thận không dẫn xuất lớp của bạn từ các lớp này. Những lớp cơ sở thông dụng dẫn xuất từ `MarshalByRefObject` bao gồm: `System.ComponentModel.Component`, `System.IO.Stream`, `System.IO.TextReader`, `System.IO.TextWriter`, `System.NET.WebRequest`, và `System.Net.WebResponse` (xem tài liệu *.NET Framework SDK* để có danh sách đầy đủ các lớp dẫn xuất từ `MarshalByRefObject`).

5. Nạp assembly vào miền ứng dụng hiện hành

- ?** Bạn cần nạp một assembly vào miền ứng dụng lúc thực thi.
- ✕** Sử dụng phương thức tĩnh `Load` hay `LoadFrom` của lớp `System.Reflection.Assembly`.

Bộ thực thi tự động nạp các assembly mà assembly của bạn tham chiếu đến lúc biên dịch. Tuy nhiên, bạn cũng có thể chỉ thị cho bộ thực thi nạp assembly. Các phương thức `Load` và `LoadFrom` đều thực hiện một công việc là nạp một assembly vào miền ứng dụng hiện hành, và cả hai đều trả về một đối tượng `Assembly` mô tả assembly vừa được nạp. Sự khác biệt giữa hai phương thức là danh sách các đối số được cung cấp để nhận dạng assembly cần nạp, và cách thức bộ thực thi định vị assembly này.

Phương thức `Load` cung cấp nhiều dạng thức cho phép chỉ định assembly cần nạp, bạn có thể sử dụng một trong những dạng sau:

- Một `string` chứa tên đầy đủ hay tên riêng phần để nhận dạng assembly.

- Một `System.Reflection.AssemblyName` mô tả chi tiết về assembly.
- Một mảng byte chứa dữ liệu cấu thành assembly.

Thông thường, tên của assembly được sử dụng để nạp assembly. Tên đầy đủ của một assembly bao gồm: tên, phiên bản, bản địa, và token khóa công khai, được phân cách bởi dấu phẩy (ví dụ: `System.Data, Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089`). Để chỉ định một assembly không có tên mạnh, sử dụng `PublicKeyToken=null`. Bạn cũng có thể sử dụng tên ngắn để nhận dạng một assembly nhưng ít nhất phải cung cấp tên của assembly (không có phần mở rộng). Đoạn mã dưới đây trình bày các cách sử dụng phương thức `Load`:

```
// Nạp assembly System.Data dùng tên đầy đủ.
string name1 = "System.Data,Version=1.0.5000.0," +
    "Culture=neutral,PublicKeyToken=b77a5c561934e089";
Assembly a1 = Assembly.Load(name1);

// Nạp assembly System.Xml dùng AssemblyName.
AssemblyName name2 = new AssemblyName();
name2.Name = "System.Xml";
name2.Version = new Version(1,0,5000,0);
name2.CultureInfo = new CultureInfo("");
name2.SetPublicKeyToken(
    new byte[] {0xb7,0x7a,0x5c,0x56,0x19,0x34,0xe0,0x89});
Assembly a2 = Assembly.Load(name2);

// Nạp assembly SomeAssembly dùng tên ngắn.
Assembly a3 = Assembly.Load("SomeAssembly");
```

Khi phương thức `Load` được gọi, bộ thực thi thực hiện quá trình định vị và nạp assembly. Dưới đây sẽ tóm tắt quá trình này; bạn tham khảo tài liệu *.NET Framework SDK* để biết thêm chi tiết.

1. Nếu bạn chỉ định assembly tên mạnh, phương thức `Load` sẽ áp dụng *version policy* (chính sách phiên bản) và *publisher policy* (chính sách nhà phát hành) để cho phép khả năng “chuyển tiếp” (redirect) đến một phiên bản assembly khác. *Version policy* được chỉ định trong file cấu hình máy tính hay ứng dụng của bạn bằng phần tử `<bindingRedirect>`. *Publisher policy* được chỉ định trong các assembly đặc biệt được cài đặt bên trong *GAC* (*Global Assembly Cache*).
2. Một khi đã xác định đúng phiên bản của assembly cần sử dụng, bộ thực thi sẽ cố gắng nạp các assembly tên mạnh từ *GAC*.
3. Nếu assembly không có tên mạnh hoặc không được tìm thấy trong *GAC*, bộ thực thi sẽ tìm phần tử `<codeBase>` trong các file cấu hình máy tính và ứng dụng. Phần tử `<codeBase>` ánh xạ tên của assembly thành một file hay một *URL*. Nếu assembly có tên mạnh, `<codeBase>` có thể chỉ đến bất kỳ vị trí nào kể cả các *URL* dựa-trên-Internet; nếu

không, <codeBase> phải chỉ đến một thư mục có vị trí tương đối so với thư mục ứng dụng. Nếu assembly không tồn tại trong vị trí được chỉ định, phương thức `Load` sẽ ném ngoại lệ `System.IO.FileNotFoundException`.

4. Nếu không có phần tử <codeBase> tương ứng với assembly, bộ thực thi sẽ tìm assembly bằng kỹ thuật *probing*. Quá trình *probing* sẽ tìm file đầu tiên có tên của assembly (với phần mở rộng là *.dll* hay *.exe*) trong các vị trí:
 - Thư mục gốc của ứng dụng.
 - Các thư mục con của thư mục gốc phù hợp với tên và bản địa của assembly.
 - Các thư mục con (của thư mục gốc) do người dùng chỉ định.

Phương thức `Load` là cách dễ nhất để tìm và nạp các assembly, nhưng cũng có thể tốn nhiều chi phí cho việc dò trong nhiều thư mục để tìm các assembly có tên yếu. Phương thức `LoadFrom` cho phép bạn nạp assembly từ một vị trí xác định, nếu không tìm thấy nó sẽ ném ngoại lệ `FileNotFoundException`. Bộ thực thi sẽ không cố tìm assembly như phương thức `Load` —phương thức `LoadFrom` không hỗ trợ *GAC*, *policy*, phần tử <codeBase> hay *probing*. Dưới đây là đoạn mã trình bày cách sử dụng `LoadFrom` để nạp *c:\shared\MySharedAssembly.dll*. Lưu ý rằng, khác với `Load`, `LoadFrom` yêu cầu bạn chỉ định phần mở rộng của file assembly.

```
// Nạp assembly có tên là c:\shared\MySharedAssembly.dll
Assembly a4 = Assembly.LoadFrom(@"c:\shared\MySharedAssembly.dll");
```

6.

Thực thi assembly ở miền ứng dụng khác



Bạn cần thực thi một assembly ở một miền ứng dụng khác với miền ứng dụng hiện hành.



Gọi phương thức `ExecuteAssembly` của đối tượng `AppDomain` đại diện cho miền ứng dụng, và chỉ định tên của assembly cần thực thi.

Nếu bạn có một assembly khả-thực-thi và muốn nạp để thực thi nó trong một miền ứng dụng, phương thức `ExecuteAssembly` sẽ giúp bạn. Phương thức `ExecuteAssembly` có bốn dạng thức khác nhau. Dạng thức đơn giản nhất chỉ nhận vào một kiểu `string` chứa tên của assembly cần thực thi; bạn có thể chỉ định một file cục bộ hay một *URL*. Một dạng thức khác cho phép bạn chỉ định chứng cứ (*evidence*) cho assembly (xem mục 13.10) và các đối số để truyền đến điểm nhập của assembly (tương đương với các đối số dòng lệnh).

Phương thức `ExecuteAssembly` nạp assembly được chỉ định và thực thi phương thức được định nghĩa trong siêu dữ liệu là điểm nhập của assembly (thường là phương thức `Main`). Nếu assembly được chỉ định không có khả năng thực thi, `ExecuteAssembly` sẽ ném ngoại lệ `System.Runtime.InteropServices.COMException`. Bộ thực thi không thực thi assembly trong một tiểu trình mới, vì thế quyền kiểm soát sẽ không trả về cho đến khi quá trình thực thi của assembly kết thúc. Do `ExecuteAssembly` nạp một assembly bằng tên riêng phần (chỉ có tên file), *CLR* sẽ không dùng *GAC* hay *probing* để tìm assembly (xem mục 3.5 để biết thêm chi tiết).

Ví dụ dưới đây trình bày cách sử dụng phương thức `ExecuteAssembly` để nạp và thực thi một assembly. Lớp `ExecuteAssemblyExample` tạo một `AppDomain` và thực thi chính nó trong `AppDomain` bằng phương thức `ExecuteAssembly`. Kết quả là có hai bản sao của `ExecuteAssemblyExample` được nạp vào hai miền ứng dụng khác nhau.

```
using System;

public class ExecuteAssemblyExample {

    public static void Main(string[] args) {

        // Nếu assembly đang thực thi trong một AppDomain
        // có tên thân thiện là "NewAppDomain"
        // thì không tạo AppDomain mới. Điều này sẽ
        // tránh một vòng lặp vô tận tạo AppDomain.
        if (AppDomain.CurrentDomain.FriendlyName != "NewAppDomain") {

            // Tạo miền ứng dụng mới có tên là "NewAppDomain".
            AppDomain domain = AppDomain.CreateDomain("NewAppDomain");

            // Thực thi assembly này trong AppDomain mới và
            // truyền mảng các đối số dòng lệnh.
            domain.ExecuteAssembly("ExecuteAssemblyExample.exe",
                                   null, args);
        }

        // Hiển thị các đối số dòng lệnh lên màn hình
        // cùng với tên thân thiện của AppDomain.
        foreach (string s in args) {

            Console.WriteLine(AppDomain.CurrentDomain.FriendlyName +
                               " : " + s);
        }
    }
}
```

7.

Thể hiện hóa một kiểu trong miền ứng dụng khác

?

Bạn cần thể hiện hóa một kiểu trong một miền ứng dụng khác với miền ứng dụng hiện hành.

Gọi phương thức `CreateInstance` hay `CreateInstanceFrom` của đối tượng `AppDomain` đại diện cho miền ứng dụng đích.

Việc sử dụng phương thức `ExecuteAssembly` (đã được thảo luận trong mục 3.6) không mấy khó khăn; nhưng khi phát triển các ứng dụng phức tạp có sử dụng nhiều miền ứng dụng, chắc chắn bạn muốn kiểm soát quá trình nạp các assembly, tạo các kiểu dữ liệu, và triệu gọi các thành viên của đối tượng bên trong miền ứng dụng.

Các phương thức `CreateInstance` và `CreateInstanceFrom` cung cấp nhiều phiên bản nạp chồng giúp bạn kiểm soát quá trình tạo đối tượng. Các phiên bản đơn giản nhất sử dụng phương thức khởi dựng mặc định của kiểu, nhưng cả hai phương thức này đều thiết đặt các phiên bản cho phép bạn cung cấp đối số để sử dụng bất kỳ phương thức khởi dựng nào.

Phương thức `CreateInstance` nạp một assembly có tên xác định vào miền ứng dụng bằng quá trình đã được mô tả cho phương thức `Assembly.Load` trong mục 3.5. Sau đó, `CreateInstance` tạo đối tượng cho kiểu và trả về một tham chiếu đến đối tượng mới được đóng gói trong `ObjectHandle` (được mô tả trong mục 3.3). Tương tự như thế đối với phương thức `CreateInstanceFrom`; tuy nhiên, `CreateInstanceFrom` nạp assembly vào miền ứng dụng bằng quá trình đã được mô tả cho phương thức `Assembly.LoadFrom` trong mục 3.5.

 **`AppDomain` cũng cung cấp hai phương thức rất tiện lợi có tên là `CreateInstanceAndUnwrap` và `CreateInstanceFromAndUnwrap`, chúng sẽ tự động khôi phục tham chiếu đến đối tượng đã được tạo từ đối tượng `ObjectHandle`; bạn phải ép đối tượng trả về cho đúng kiểu trước khi sử dụng.**

Nếu bạn sử dụng `CreateInstance` hay `CreateInstanceFrom` để tạo đối tượng kiểu *MBV* trong một miền ứng dụng khác, đối tượng sẽ được tạo nhưng tham chiếu trả về sẽ không chỉ đến đối tượng đó. Do cách thức đối tượng *MBV* vượt qua biên miền ứng dụng, tham chiếu này sẽ chỉ đến một bản sao của đối tượng được tạo tự động trong miền ứng dụng cục bộ. Chỉ khi bạn tạo một kiểu *MBR* thì tham chiếu trả về mới chỉ đến đối tượng trong miền ứng dụng khác (xem mục 3.2 để biết thêm chi tiết về kiểu *MBV* và *MBR*).

Kỹ thuật chung để đơn giản hóa việc quản lý các miền ứng dụng là sử dụng lớp điều khiển (*controller class*). Một lớp điều khiển là một kiểu *MBR* tùy biến. Bạn hãy tạo một miền ứng dụng rồi tạo đối tượng lớp điều khiển trong miền ứng dụng này bằng phương thức `CreateInstance`. Lớp điều khiển hiện thực các chức năng cần thiết cho ứng dụng để thao tác miền ứng dụng và các nội dung của nó. Các chức năng này có thể bao gồm: nạp assembly, tạo thêm miền ứng dụng, dọn dẹp trước khi xóa miền ứng dụng, hay liệt kê các phần tử chương trình (bạn không thể thực hiện ở bên ngoài miền ứng dụng).

Ví dụ dưới đây trình bày cách sử dụng một lớp điều khiển có tên là `PluginManager`. Khi đã được tạo trong một miền ứng dụng, `PluginManager` cho phép bạn tạo đối tượng của các lớp có hiện thực giao diện `IPlugin`, chạy và dừng các plug-in đó, và trả về danh sách các plug-in hiện được nạp.

```
using System;
using System.Reflection;
```



```
using System.Collections;
using System.Collections.Specialized;

// Giao diện chung cho tất cả các plug-in.
public interface IPlugin {
    void Start();
    void Stop();
}

// Một hiện thực đơn giản cho giao diện Iplugin
// để minh họa lớp điều khiển PluginManager.
public class SimplePlugin : IPlugin {

    public void Start() {
        Console.WriteLine(AppDomain.CurrentDomain.FriendlyName +
            ": SimplePlugin starting...");
    }

    public void Stop() {
```

```

        Console.WriteLine(AppDomain.CurrentDomain.FriendlyName +
            ": SimplePlugin stopping...");
    }
}

// Lớp điều khiển, quản lý việc nạp và thao tác
// các plug-in trong miền ứng dụng của nó.
public class PluginManager : MarshalByRefObject {

    // ListDictionary giữ tham chiếu đến các plug-in.
    private ListDictionary plugins = new ListDictionary();

    // Phương thức khởi dựng mặc định.
    public PluginManager() {}

    // Phương thức khởi dựng nhận danh sách các plug-in.
    public PluginManager(ListDictionary pluginList) {

        // Nạp các plug-in đã được chỉ định.
        foreach (string plugin in pluginList.Keys) {

            this.LoadPlugin((string)pluginList[plugin], plugin);
        }
    }

    // Nạp assembly và tạo plug-in được chỉ định.
    public bool LoadPlugin(string assemblyName, string pluginName) {

        try {

            // Nạp assembly.
            Assembly assembly = Assembly.Load(assemblyName);

            // Tạo plug-in mới.
            IPlugin plugin =

```

```
(IPlugin)assembly.CreateInstance(pluginName, true);

if (plugin != null) {

    // Thêm plug-in mới vào ListDictionary.
    plugins[pluginName] = plugin;

    return true;

} else {
    return false;
}
} catch {
    return false;
}
}

public void StartPlugin(string plugin) {

    // Lấy một plug-in từ ListDictionary và
    // gọi phương thức Start.
    ((IPlugin)plugins[plugin]).Start();
}
```

```
public void StopPlugin(string plugin) {

    // Lấy một plug-in từ ListDictionary và
    // gọi phương thức Stop.
    ((IPlugin)plugins[plugin]).Stop();
}

public ArrayList GetPluginList() {

    // Trả về danh sách các plug-in.
    return new ArrayList(plugins.Keys);
}
}

public class CreateInstanceExample {

    public static void Main() {

        // Tạo một miền ứng dụng mới.
        AppDomain domain1 = AppDomain.CreateDomain("NewAppDomain1");

        // Tạo một PluginManager trong miền ứng dụng mới
        // bằng phương thức khởi dựng mặc định.
        PluginManager manager1 =
            (PluginManager)domain1.CreateInstanceAndUnwrap(
                "CreateInstanceExample", "PluginManager");

        // Nạp một plug-in mới vào NewAppDomain1.
        manager1.LoadPlugin("CreateInstanceExample", "SimplePlugin");

        // Chạy và dừng plug-in trong NewAppDomain1.
        manager1.StartPlugin("SimplePlugin");
        manager1.StopPlugin("SimplePlugin");

        // Tạo một miền ứng dụng mới.
        AppDomain domain2 = AppDomain.CreateDomain("NewAppDomain2");
```

```
// Tạo một ListDictionary chứa các plug-in.
ListDictionary pluginList = new ListDictionary();
pluginList["SimplePlugin"] = "CreateInstanceExample";

// Tạo một PluginManager trong miền ứng dụng mới
// và chỉ định danh sách các plug-in.
PluginManager manager2 =
    (PluginManager)domain1.CreateInstanceAndUnwrap(
        "CreateInstanceExample", "PluginManager", true, 0,
        null, new object[] {pluginList}, null, null, null);

// Hiển thị các plug-in đã được nạp vào NewAppDomain2.
Console.WriteLine("Plugins in NewAppDomain2:");
foreach (string s in manager2.GetPluginList()) {
    Console.WriteLine(" - " + s);
}

// Nhấn Enter để thoát.
Console.ReadLine();
}
}
```

8. *Truyền dữ liệu giữa các miền ứng dụng*

? **Bạn cần một cơ chế đơn giản để truyền dữ liệu trạng thái hay cấu hình giữa các miền ứng dụng.**

✕ **Dùng các phương thức `SetData` và `GetData` của lớp `AppDomain`.**

Dữ liệu có thể được truyền qua các miền ứng dụng như đối số hay trả về khi bạn cho gọi các thành viên của các đối tượng hiện có trong các miền ứng dụng. Việc truyền dữ liệu qua các miền ứng dụng được thực hiện dễ dàng giống như truyền dữ liệu trong cùng một miền ứng dụng.

Mọi miền ứng dụng đều duy trì một vùng đệm dữ liệu (*data cache*) chứa một tập các cặp “tên/giá trị”. Hầu hết nội dung của vùng đệm dữ liệu phản ánh các thiết lập cấu hình của miền ứng dụng, như các giá trị từ đối tượng `AppDomainSetup` được cung cấp trong quá trình tạo miền ứng dụng (xem mục 3.1). Vùng đệm dữ liệu này có thể được sử dụng để trao đổi dữ liệu giữa các miền ứng dụng hay lưu trữ các giá trị tạm thời dùng trong cùng một miền ứng dụng.

Phương thức `SetData` thực hiện việc kết hợp một khóa dạng chuỗi với một đối tượng và lưu trữ nó vào vùng đệm dữ liệu của miền ứng dụng. Phương thức `GetData` thực hiện công việc ngược lại là lấy lại đối tượng từ vùng đệm dữ liệu thông qua khóa. Nếu mã lệnh trong một

miền ứng dụng gọi phương thức `SetData` hay `GetData` để truy xuất vùng đệm dữ liệu của miền ứng dụng khác, thì đối tượng dữ liệu phải hỗ trợ ngữ nghĩa *marshal-by-value* hay *marshal-by-reference*, nếu không thì ngoại lệ `System.Runtime.Serialization.SerializationException` sẽ bị ném (xem mục 3.3 để biết thêm chi tiết về cách truyền đối tượng qua các miền ứng dụng). Đoạn mã sau trình bày cách sử dụng phương thức `SetData` và `GetData` để truyền một `System.Collections.ArrayList` giữa hai miền ứng dụng.

```
using System;
using System.Reflection;
using System.Collections;

public class ListModifier {

    public ListModifier () {

        // Nhận danh sách từ đệm dữ liệu.
        ArrayList list =
            (ArrayList)AppDomain.CurrentDomain.GetData("People");
        // Thay đổi danh sách.
        list.Add("Tam");
    }
}

public class PassDataExample {

    public static void Main() {

        // Tạo một miền ứng dụng mới.
        AppDomain domain = AppDomain.CreateDomain("Test");

        // Tạo một ArrayList và thêm thông tin vào.
        ArrayList list = new ArrayList();
        list.Add("Phuong");
        list.Add("Phong");
        list.Add("Nam");

        // Đặt danh sách vào vùng đệm dữ liệu của miền ứng dụng mới.
        domain.SetData("People", list);
    }
}
```

```
// Tạo một ListModifier trong miền ứng dụng mới
// sẽ thay đổi nội dung của list trong vùng đệm dữ liệu.
domain.CreateInstance("03-08", "ListModifier");

// Nhận lại danh sách và hiển thị nội dung của nó.
foreach (string s in (ArrayList)domain.GetData("People")) {
    Console.WriteLine(s);
}

// Nhấn Enter để thoát.
Console.ReadLine();
}
}
```

9. *Giải phóng assembly và miền ứng dụng*

? Bạn cần giải phóng các assembly hay các miền ứng dụng lúc thực thi.

✖ Không có cách nào để giải phóng các assembly riêng lẻ. Bạn có thể giải phóng toàn bộ một miền ứng dụng bằng phương thức tĩnh `AppDomain.Unload`, đồng thời với việc giải phóng miền ứng dụng là tất cả các assembly đã được nạp vào miền ứng dụng đó cũng được giải phóng.

Cách duy nhất để giải phóng một assembly là giải phóng cả miền ứng dụng mà nó đã được nạp vào. Đáng tiếc, việc giải phóng một miền ứng dụng cũng sẽ giải phóng luôn tất cả các assembly đã được nạp vào đó. Đây là một giới hạn yêu cầu bạn phải tổ chức và quản lý tốt cấu trúc miền ứng dụng và assembly.

Khi giải phóng một miền ứng dụng bằng phương thức tĩnh `AppDomain.Unload`, bạn cần truyền cho nó một tham chiếu `AppDomain` đến miền ứng dụng cần giải phóng. Bạn không thể giải phóng miền ứng dụng mặc định do `CLR` tạo lúc startup. Đoạn mã dưới đây trình bày cách sử dụng phương thức `Unload`.

```
// Tạo một miền ứng dụng mới.
AppDomain newDomain = AppDomain.CreateDomain("New Domain");

// Nạp assembly vào miền ứng dụng này.
$

// Giải phóng miền ứng dụng.
AppDomain.Unload(newDomain);
```

Phương thức `Unload` chặn các tiểu trình mới đi vào miền ứng dụng được chỉ định và gọi phương thức `Thread.Abort` trên tất cả các tiểu trình hiện đang chạy trong miền ứng dụng này. Nếu tiểu trình gọi phương thức `Unload` hiện đang chạy trong miền ứng dụng cần giải phóng thì một tiểu trình khác sẽ được khởi chạy để thực hiện quá trình giải phóng. Nếu có vấn đề trong

việc giải phóng miền ứng dụng, ngoại lệ `System.CannotUnloadAppDomainException` sẽ bị ném bởi tiểu trình thực hiện quá trình giải phóng.

Trong khi miền ứng dụng đang được giải phóng, *CLR* gọi thực thi các phương thức giải phóng của tất cả các đối tượng trong miền ứng dụng. Tùy thuộc vào số lượng đối tượng và bản chất của các phương thức giải phóng mà quá trình này có thể mất một khoảng thời gian nào đó. Phương thức `AppDomain.IsFinalizingForUnload` trả về `true` nếu miền ứng dụng đang được giải phóng và *CLR* đã bắt đầu giải phóng các đối tượng trong đó; ngược lại, trả về `false`.

10.

Truy xuất thông tin Type

?

Bạn muốn thu lấy đối tượng `System.Type` mô tả một kiểu dữ liệu nhất định.

✕

Sử dụng một trong các cách sau:

- Toán tử `typeof`
- Phương thức tĩnh `GetType` của lớp `System.Type`
- Phương thức `GetType` thuộc một thể hiện của kiểu
- Phương thức `GetNestedType` hay `GetNestedTypes` của lớp `Type`
- Phương thức `GetType` hay `GetTypes` của lớp `Assembly`
- Phương thức `GetType`, `GetTypes`, hay `FindTypes` của lớp `System.Reflection.Module`

Đối tượng `Type` cung cấp một điểm khởi đầu để làm việc với các kiểu dữ liệu bằng cơ chế phản chiếu. Một đối tượng `Type` cho phép bạn kiểm tra siêu dữ liệu của kiểu, thu lấy các thành viên của kiểu, và tạo các đối tượng của kiểu. Do tầm quan trọng của nó, *.NET Framework* cung cấp nhiều cơ chế để lấy tham chiếu đến các đối tượng `Type`.

Phương pháp hiệu quả nhất để thu lấy đối tượng `Type` cho một kiểu cụ thể là sử dụng toán tử `typeof`:

```
System.Type t1 = typeof(System.Text.StringBuilder);
```

Tên kiểu không được đặt trong dấu nháy kép và phải khả phân giải đối với trình biên dịch. Vì tham chiếu được phân giải lúc biên dịch nên assembly chứa kiểu này trở thành phần phụ thuộc tĩnh của assembly và sẽ được liệt kê như thế trong assembly manifest của bạn.

Một cách khác là sử dụng phương thức tĩnh `Type.GetType`, nhận vào một chuỗi chứa tên kiểu. Vì sử dụng chuỗi để chỉ định kiểu nên bạn có thể thay đổi nó lúc thực thi, điều này mở ra cánh cửa đến với thế giới lập trình động bằng cơ chế phản chiếu (xem mục 3.12). Nếu bạn chỉ định tên kiểu, bộ thực thi phải tìm kiểu này trong một assembly đã được nạp. Bạn cũng có thể chỉ định một tên kiểu theo tiêu chuẩn assembly (tham khảo tài liệu *.NET Framework SDK* về phương thức `Type.GetType` để biết cách kết cấu tên kiểu theo tiêu chuẩn assembly). Các lệnh sau trình bày cách sử dụng phương thức `GetType`:

```
// Có phân biệt chữ hoa-thường, trả về null nếu không tìm thấy.
Type t2 = Type.GetType("System.String");
```



```
// Có phân biệt chữ hoa-thường,
// ném ngoại lệ TypeLoadException nếu không tìm thấy.
Type t3 = Type.GetType("System.String", true);

// Không phân biệt chữ hoa-thường,
// ném ngoại lệ TypeLoadException nếu không tìm thấy.
Type t4 = Type.GetType("system.string", true, true);

// Tên kiểu theo tiêu chuẩn assembly.
Type t5 = Type.GetType("System.Data.DataSet,System.Data," +
    "Version=1.0.5000.0,Culture=neutral,
    PublicKeyToken=b77a5c561934e089");
```

Để thu lấy đối tượng `Type` mô tả kiểu của một đối tượng hiện có, hãy sử dụng phương thức `GetType`, được hiện thực bởi `Object` và được thừa kế bởi tất cả các kiểu dữ liệu. Dưới đây là một ví dụ:

```
System.Text.StringBuilder sb = new System.Text.StringBuilder();
Type t6 = sb.GetType();
```

Bảng 3.2 tóm tắt các phương thức khác cũng cung cấp khả năng truy xuất đối tượng `Type`.

Bảng 3.2 Các phương thức trả về đối tượng `Type`

Phương thức	Mô tả
<code>Type.GetNestedType</code>	Lấy đối tượng <code>Type</code> mô tả một kiểu lồng bên trong đối tượng <code>Type</code> hiện có
<code>Type.GetNestedTypes</code>	Lấy một mảng các đối tượng <code>Type</code> mô tả các kiểu lồng bên trong đối tượng <code>Type</code> hiện có
<code>Assembly.GetType</code>	Lấy đối tượng <code>Type</code> mô tả một kiểu được khai báo bên trong assembly
<code>Assembly.GetTypes</code>	Lấy một mảng các đối tượng <code>Type</code> mô tả các kiểu được khai báo bên trong assembly
<code>Module.GetType</code>	Lấy đối tượng <code>Type</code> mô tả một kiểu được khai báo bên trong module
<code>Module.GetTypes</code>	Lấy một mảng các đối tượng <code>Type</code> mô tả các kiểu được khai báo bên trong module
<code>Module.FindTypes</code>	Lấy một mảng đã được lọc, chứa các đối tượng <code>Type</code> mô tả các kiểu được khai báo bên trong module—các kiểu này được lọc bằng một <i>delegate</i> (xác định xem mỗi <code>Type</code> có xuất hiện trong mảng đích hay không)

11.

Kiểm tra kiểu của một đối tượng

?

Bạn muốn kiểm tra kiểu của một đối tượng.

✕

Sử dụng phương thức thừa kế `Object.GetType` để thu lấy `Type` cho đối tượng này. Trong vài trường hợp, bạn cũng có thể sử dụng toán tử `is` và `as` để kiểm tra kiểu của một đối tượng.

Tất cả các kiểu dữ liệu đều thừa kế phương thức `GetType` từ lớp cơ sở `Object`. Như đã được thảo luận trong mục 3.10, phương thức này trả về một tham chiếu `Type` mô tả kiểu của đối tượng. Bộ thực thi duy trì một đối tượng `Type` cho mỗi kiểu được nạp và tất cả các tham chiếu cho kiểu này cùng chỉ đến đối tượng này. Điều này nghĩa là bạn có thể so sánh hai tham chiếu kiểu một cách hiệu quả. Ví dụ dưới đây trình bày cách kiểm tra một đối tượng có phải là `System.IO.StringReader` hay không:

```
// Tạo một StringReader để thử nghiệm.
Object someObject =
    new StringReader("This is a StringReader");

// Kiểm tra xem someObject có phải là một StringReader hay không
// bằng cách thu lấy và so sánh tham chiếu Type (sử dụng toán tử typeof).
if (typeof(System.IO.StringReader) == someObject.GetType()) {
    // Làm gì đó.
    $
}
```

C# cung cấp toán tử `is` để thực hiện nhanh việc kiểm tra như trên. Ngoài ra, `is` sẽ trả về `true` nếu đối tượng cần kiểm tra dẫn xuất từ lớp được chỉ định. Đoạn mã dưới đây kiểm tra xem `someObject` là một thể hiện của `System.IO.TextReader`, hay một lớp dẫn xuất từ `TextReader` (như `StringReader`):

```
// Kiểm tra xem someObject là TextReader,
// hay dẫn xuất từ TextReader bằng toán tử is.
if (someObject is System.IO.TextReader) {
    // Làm gì đó.
    $
}
```

Cả hai cách này đều đòi hỏi kiểu dùng với toán tử `typeof` và `is` phải là kiểu đã biết và khả phân giải lúc biên dịch. Một cách khác linh hoạt hơn (nhưng chậm hơn) là sử dụng phương thức `Type.GetType` để trả về một tham chiếu `Type` cho kiểu được chỉ định. Tham chiếu `Type` không được phân giải cho đến khi thực thi, việc này ảnh hưởng đến hiệu năng, nhưng cho phép bạn thay đổi phép so sánh kiểu lúc thực thi dựa trên giá trị của một chuỗi. Phương thức `IsType` dưới đây sẽ trả về `true` nếu đối tượng thuộc kiểu được chỉ định và sử dụng phương

thức `Type.IsSubclassOf` để kiểm tra đối tượng này có phải là một lớp con của kiểu được chỉ định hay không.

```
public static bool IsType(object obj, string type) {

    Type t = Type.GetType(type, true, true);

    return t == obj.GetType() || obj.GetType().IsSubclassOf(t);

}
```

Cuối cùng, bạn có thể sử dụng toán tử `as` để ép bất kỳ đối tượng nào sang kiểu được chỉ định. Nếu đối tượng không thể bị ép sang kiểu được chỉ định, toán tử `as` sẽ trả về `null`. Điều này cho phép bạn thực hiện các phép ép kiểu an toàn (*safe cast*), nhưng kiểu được so sánh phải là khả phân giải lúc thực thi. Dưới đây là một ví dụ:

```
// Sử dụng toán tử as để thực hiện một phép ép kiểu an toàn.
StringReader reader = someObject as System.IO.StringReader;
if (reader != null) {
    // Làm gì đó.
    $
}
```



Phương thức tĩnh `GetUnderlyingType` của lớp `System.Enum` cho phép bạn thu lấy kiểu thực sự của một kiểu liệt kê.

12.

Tạo một đối tượng bằng cơ chế phản chiếu



Bạn cần tạo một đối tượng bằng cơ chế phản chiếu.



Thu lấy đối tượng `Type` mô tả kiểu của đối tượng cần tạo, gọi phương thức `GetConstructor` của nó để có được đối tượng `System.Reflection.ConstructorInfo` mô tả phương thức khởi dựng cần dùng, sau đó thực thi phương thức `ConstructorInfo.Invoke`.

Bước đầu tiên trong việc tạo một đối tượng bằng cơ chế phản chiếu là thu lấy đối tượng `Type` mô tả kiểu của đối tượng cần tạo (xem mục 3.10 để biết thêm chi tiết). Khi có được đối tượng `Type`, hãy gọi phương thức `GetConstructor` để thu lấy đối tượng `ConstructorInfo` mô tả một trong các phương thức khởi dựng của kiểu này. Dạng thức thông dụng nhất của phương thức `GetConstructor` là nhận một mảng `Type` làm đối số, và trả về đối tượng `ConstructorInfo` mô tả phương thức khởi dựng nhận các đối số được chỉ định trong mảng `Type`. Để thu lấy đối tượng `ConstructorInfo` mô tả phương thức khởi dựng mặc định (không có đối số), bạn hãy truyền cho phương thức `GetConstructor` một mảng `Type` rỗng (sử dụng trường tĩnh `Type.EmptyTypes`); đừng sử dụng `null`, nếu không `GetConstructor` sẽ ném ngoại lệ `System.ArgumentNullException`. Nếu `GetConstructor` không tìm thấy phương thức khởi dựng nào có chữ ký phù hợp với các đối số được chỉ định thì nó sẽ trả về `null`.

Một khi đã có đối tượng `ConstructorInfo` như mong muốn, hãy gọi phương thức `Invoke` của nó. Bạn phải cung cấp một mảng chứa các đối số mà bạn muốn truyền cho phương thức khởi dựng. Phương thức `Invoke` sẽ tạo đối tượng mới và trả về một tham chiếu đến đối tượng đó (bạn phải ép về kiểu phù hợp). Dưới đây là đoạn mã trình bày cách tạo một đối tượng `System.Text.StringBuilder`, chỉ định nội dung ban đầu cho `StringBuilder` và sức chứa của nó.

```
// Thu lấy đối tượng Type cho lớp StringBuilder.
Type type = typeof(System.Text.StringBuilder);

// Tạo Type[] chứa các đối tượng Type cho mỗi đối số
// của phương thức khởi dựng (một chuỗi và một số nguyên).
Type[] argTypes = new Type[] {typeof(System.String),
    typeof(System.Int32)};

// Thu lấy đối tượng ConstructorInfo.
ConstructorInfo cInfo = type.GetConstructor(argTypes);

// Tạo object[] chứa các đối số cho phương thức khởi dựng.
object[] argVals = new object[] {"Some string", 30};

// Tạo đối tượng và ép nó về kiểu StringBuilder.
StringBuilder sb = (StringBuilder)cInfo.Invoke(argVals);
```

Chức năng phản chiếu thường được sử dụng để hiện thực các *factory*. Trong đó, bạn sử dụng cơ chế phản chiếu để thể hiện hóa các lớp thừa kế một lớp cơ sở phổ biến hay hiện thực một giao diện phổ biến. Thông thường, cả lớp cơ sở chung và giao diện chung đều được sử dụng. Lớp cơ sở trừu tượng sẽ hiện thực giao diện và bất kỳ chức năng chung nào, sau đó mỗi hiện thực cụ thể sẽ thừa kế lớp cơ sở.

Không có cơ chế nào để khai báo rằng mỗi lớp cụ thể phải hiện thực các phương thức khởi dựng với các chữ ký cụ thể. Nếu muốn người khác (hãng thứ ba) hiện thực các lớp cụ thể thì bạn phải chỉ rõ (trong tài liệu hướng dẫn) chữ ký của phương thức khởi dựng mà *factory* của bạn gọi. Cách thông thường để tránh vấn đề này là sử dụng phương thức khởi dựng mặc định (không có đối số), sau đó cấu hình đối tượng bằng phương thức và thuộc tính. Ví dụ dưới đây sẽ hiện thực một *factory* dùng để tạo các đối tượng có hiện thực giao diện `IPlugin`.

```
using System;
using System.Reflection;

// Giao diện chung mà tất cả các plug-in phải hiện thực.
public interface IPlugin {
    string Description { get; set; }
```

```

    void Start();
    void Stop();
}

// Lớp cơ sở trừu tượng mà tất cả các plug-in phải dẫn xuất từ đây.
public abstract class AbstractPlugin : IPlugin {

    // Chuỗi chứa lời mô tả plug-in.
    private string description = "";

    // Thuộc tính dùng để lấy lời mô tả plug-in.
    public string Description {
        get { return description; }
        set { description = value; }
    }

    // Khai báo các thành viên của giao diện IPlugin.
    public abstract void Start();
    public abstract void Stop();
}

// Một hiện thực đơn giản cho giao diện IPlugin
// để minh họa lớp PluginFactory.
public class SimplePlugin : AbstractPlugin {

    // Hiện thực phương thức Start.
    public override void Start() {
        Console.WriteLine(Description + ": Starting...");
    }

    // Hiện thực phương thức Stop.
    public override void Stop() {
        Console.WriteLine(Description + ": Stopping...");
    }
}

// Factory dùng để tạo các đối tượng của IPlugin.
public sealed class PluginFactory {

    public static IPlugin CreatePlugin(string assembly,

```

```

        string pluginName, string description) {

    // Thu lấy đối tượng Type cho plug-in được chỉ định.
    Type type = Type.GetType(pluginName + ", " + assembly);

    // Thu lấy đối tượng ConstructorInfo.
    ConstructorInfo cInfo = type.GetConstructor(Type.EmptyTypes);

    // Tạo đối tượng và ép nó về kiểu StringBuilder.
    IPlugin plugin = (IPlugin)cInfo.Invoke(null);

    // Cấu hình IPlugin mới.
    plugin.Description = description;

    return plugin;
}
}

```

Câu lệnh sau đây sẽ tạo đối tượng `SimplePlugin` bằng lớp `PluginFactory`:

```

IPlugin plugin = PluginFactory.CreatePlugin(
    "CreateObjectExample", // Tên assembly
    "SimplePlugin",        // Tên lớp plug-in
    "A Simple Plugin"      // Lời mô tả plug-in
);

```



Lớp `System.Activator` cung cấp hai phương thức tĩnh `CreateInstance` và `CreateInstanceFrom` dùng để tạo các đối tượng dựa trên đối tượng `Type` hay chuỗi chứa tên kiểu. Xem tài liệu *.NET Framework SDK* để biết thêm chi tiết.

13.

Tạo một đặc tính tùy biến



Bạn cần tạo ra một đặc tính theo ý bạn.



Tạo một lớp dẫn xuất từ lớp cơ sở trừu tượng `System.Attribute`. Hiện thực các phương thức khởi dựng, các trường, và các thuộc tính để cho phép người dùng cấu hình đặc tính. Sử dụng `System.AttributeUsageAttribute` để định nghĩa:

- Những phần tử chương trình nào là đích của đặc tính
- Bạn có thể áp dụng nhiều thể hiện của đặc tính cho một phần tử chương trình hay không

- **Đặc tính có được thừa kế bởi các kiểu dẫn xuất hay không**

Đặc tính cung cấp một cơ chế tổng quát cho việc kết hợp thông tin khai báo (siêu dữ liệu) với các phần tử chương trình. Siêu dữ liệu này nằm trong assembly đã được biên dịch, cho phép các chương trình thu lấy nó thông qua cơ chế phản chiếu lúc thực thi (xem mục 3.14.) Các chương trình khác, đặc biệt là *CLR*, sử dụng thông tin này để xác định cách thức tương tác và quản lý các phần tử chương trình.

Để tạo một đặc tính tùy biến thì hãy dẫn xuất một lớp từ lớp cơ sở trừu tượng `System.Attribute`. Các lớp đặc tính tùy biến phải là `public` và có tên kết thúc bằng “`Attribute`”. Một đặc tính tùy biến phải có ít nhất một phương thức khởi dựng công khai. Các đối số của phương thức khởi dựng sẽ trở thành các đối số vị trí (*positional parameter*) của đặc tính. Như với bất kỳ lớp nào khác, bạn có thể khai báo nhiều phương thức khởi dựng, cho phép người dùng tùy chọn sử dụng các tập khác nhau của các đối số vị trí khi áp dụng đặc tính. Bất kỳ thuộc tính và trường đọc/ghi công khai nào do đặc tính khai báo đều trở thành đối số được nêu tên (*named parameter*).

Để điều khiển cách thức người dùng áp dụng đặc tính, hãy áp dụng đặc tính `AttributeUsageAttribute` cho đặc tính tùy biến của bạn. Đặc tính `AttributeUsageAttribute` hỗ trợ một đối số vị trí và hai đối số được nêu tên, được mô tả trong bảng 3.3. Các giá trị mặc định chỉ định giá trị sẽ được áp dụng cho đặc tính tùy biến của bạn nếu bạn không áp dụng `AttributeUsageAttribute` hoặc không chỉ định giá trị cho một thông số.

Bảng 3.3 Các thành viên thuộc kiểu liệt kê `AttributeUsage`

Thông số	Kiểu	Mô tả	Mặc định
<code>ValidOn</code>	vị trí	Một thành viên thuộc kiểu liệt kê <code>System.AttributeTargets</code> , chỉ định phần tử chương trình mà đặc tính sẽ có hiệu lực trên đó.	<code>AttributeTargets.All</code>
<code>AllowMultiple</code>	được nêu tên	Đặc tính có thể được chỉ định nhiều lần cho một phần tử hay không.	<code>false</code>
<code>Inherited</code>	được nêu tên	Đặc tính có được thừa kế bởi các lớp dẫn xuất hay các thành viên được chép đè hay không.	<code>true</code>

Ví dụ dưới đây trình bày cách tạo một đặc tính tùy biến có tên là `AuthorAttribute`, cho phép bạn xác định tên và công ty của người tạo ra lớp hay assembly. `AuthorAttribute` khai báo một phương thức khởi dựng công khai, nhận một chuỗi chứa tên tác giả. Điều này yêu cầu người sử dụng `AuthorAttribute` phải luôn cung cấp một đối số vị trí chứa tên tác giả. `Company` là thuộc tính công khai (có thể dùng làm đối số được nêu tên), `Name` là thuộc tính chỉ-đọc (không thể dùng làm đối số được nêu tên).

```
using System;
```

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Assembly,
    AllowMultiple = true, Inherited = false)]
public class AuthorAttribute : System.Attribute {
```

```

private string company; // Công ty của tác giả
private string name;    // Tên tác giả

// Khai báo phương thức khởi dựng công khai.
public AuthorAttribute(string name) {
    this.name = name;
    company = "";
}

// Khai báo thuộc tính Company có quyền set/get.
public string Company {
    get { return company; }
    set { company = value; }
}

// Khai báo thuộc tính Name chỉ-đọc.
public string Name{
    get { return name;}
}
}

```

Dưới đây là cách sử dụng AuthorAttribute:

```

// Khai báo Square Nguyen là tác giả của assembly.
[assembly:Author("Square Nguyen", Company = "Goldsoft Ltd.")]

// Khai báo Square Nguyen là tác giả của lớp.
[Author("Square Nguyen", Company = "Goldsoft Ltd.")]
public class SomeClass {
    $
}

// Khai báo Stephen Chow là tác giả của lớp.
[Author("Stephen Chow")]
public class SomeOtherClass {
    $
}

```


}

14. *Sử dụng cơ chế phản chiếu để kiểm tra các đặc tính của một phần tử chương trình*

? Bạn muốn sử dụng cơ chế phản chiếu để kiểm tra các đặc tính tùy biến đã được áp dụng cho một phần tử chương trình.

✕ Gọi phương thức `GetCustomAttributes` của đối tượng dẫn xuất từ lớp `System.Reflection.MemberInfo` (đối tượng này mô tả phần tử chương trình cần kiểm tra).

Tất cả các lớp mô tả các phần tử chương trình đều dẫn xuất từ lớp `MemberInfo`. Lớp này bao gồm `Type`, `EventInfo`, `FieldInfo`, `PropertyInfo`, và `MethodBase`. `MethodBase` có thêm hai lớp con: `ConstructorInfo` và `MethodInfo`. Bạn có thể gọi phương thức `GetCustomAttributes` nếu có được bất kì đối tượng nào của các lớp này. Phương thức này sẽ trả về mảng chứa các đặc tính tùy biến đã được áp dụng cho phần tử chương trình. Chú ý là mảng này chỉ chứa các đặc tính tùy biến chứ không chứa các đặc tính có sẵn trong thư viện các lớp cơ sở của *.NET Framework*.

Phương thức `GetCustomAttributes` có hai dạng thức. Dạng đầu tiên nhận một đối số `bool` để xác định phương thức này có trả về các đặc tính được thừa kế từ các lớp cha hay không. Dạng thứ hai nhận thêm một đối số `Type` có vai trò như một bộ lọc, kết quả trả về là các đặc tính thuộc kiểu đã được chỉ định bởi `Type`.

Ví dụ sau sử dụng đặc tính tùy biến `AuthorAttribute` đã được khai báo trong mục 3.13 và áp dụng nó vào lớp `GetCustomAttributesExample`. Phương thức `Main` sẽ gọi phương thức `GetCustomAttributes`, lọc các đặc tính để kết quả trả về chỉ có các thể hiện của `AuthorAttribute`. Bạn có thể thực hiện ép kiểu an toàn các đặc tính này về tham chiếu `AuthorAttribute` và truy xuất các thành viên của chúng mà không cần sử dụng cơ chế phản chiếu.

```
using System;
```

```
[Author("Stephen Chau")]
```

```
[Author("Square Nguyen", Company = "Goldsoft Ltd.")]
```

```
public class GetCustomAttributesExample {
```

```
    public static void Main() {
```

```
        // Lấy đối tượng Type cho chính lớp này.
```

```
        Type type = typeof(GetCustomAttributesExample);
```

```
        // Lấy các đặc tính cho kiểu này. Sử dụng bộ lọc để
```

```
        // kết quả trả về chỉ có các thể hiện của AuthorAttribute.
```

```
object[] attrs =
    type.GetCustomAttributes(typeof(AuthorAttribute), true);

// Liệt kê các đặc tính.
foreach (AuthorAttribute a in attrs) {
    Console.WriteLine(a.Name + ", " + a.Company);
}

// Nhấn Enter để kết thúc.
Console.ReadLine();
}
}
```

4

TIỂU TRÌNH, TIẾN TRÌNH, VÀ SỰ ĐỒNG BỘ

Một trong những điểm mạnh của hệ điều hành *Microsoft Windows* là cho phép nhiều chương trình (tiến trình—*process*) chạy đồng thời và cho phép mỗi tiến trình thực hiện nhiều tác vụ đồng thời (bằng nhiều tiểu trình—*thread*). Chương này sẽ trình bày cách kiểm soát các tiến trình và các tiểu trình trong các ứng dụng dựa vào các tính năng do thư viện lớp *.NET Framework* cung cấp. Các mục trong chương này sẽ trình bày cách thực hiện các vấn đề sau:

- Sử dụng các kỹ thuật và các tính năng khác nhau của *.NET Framework* để tạo các tiểu trình mới (mục 4.1 đến 4.5).
- Kiểm soát quá trình thực thi của một tiểu trình để biết được khi nào nó kết thúc (mục 4.6 và 4.7).
- Đồng bộ hóa quá trình thực thi của nhiều tiểu trình (mục 4.8 và 4.9).
- Chạy và dừng các tiến trình mới (mục 4.10 và 4.11).
- Bảo đảm rằng tại một thời điểm chỉ có thể chạy một thể hiện của ứng dụng (mục 4.12).

1. *Thực thi phương thức với thread-pool*

? **Bạn cần thực thi một phương thức bằng một tiểu trình trong thread-pool của bộ thực thi.**

✖ **Khai báo một phương thức chứa mã lệnh cần thực thi; phương thức này phải trả về void và chỉ nhận một đối số. Sau đó, tạo một thể hiện của ủy nhiệm `System.Threading.WaitCallback` tham chiếu đến phương thức này. Tiếp tục, gọi phương thức tĩnh `QueueUserWorkItem` của lớp `System.Threading.ThreadPool`, và truyền thể hiện ủy nhiệm đã tạo làm đối số. Bộ thực thi sẽ xếp thể hiện ủy nhiệm này vào hàng đợi và thực thi nó khi một tiểu trình trong thread-pool sẵn sàng.**

Nếu ứng dụng sử dụng nhiều tiểu trình có thời gian sống ngắn hay duy trì một số lượng lớn các tiểu trình đồng thời thì hiệu năng có thể giảm sút bởi các chi phí cho việc tạo, vận hành và hủy các tiểu trình. Ngoài ra, trong một hệ thống hỗ trợ đa-tiểu-trình, các tiểu trình thường ở trạng thái rỗi suốt một khoảng thời gian dài để chờ điều kiện thực thi phù hợp. Việc sử dụng thread-pool sẽ cung cấp một giải pháp chung nhằm cải thiện tính quy mô và hiệu năng của các hệ thống hỗ trợ đa-tiểu-trình.

.NET Framework cung cấp một hiện thực đơn giản cho thread-pool mà chúng ta có thể truy xuất thông qua các thành viên tĩnh của lớp `ThreadPool`. Phương thức `QueueUserWorkItem` cho phép bạn thực thi một phương thức bằng một tiểu trình trong thread-pool (đặt công việc vào hàng đợi). Mỗi công việc được mô tả bởi một thể hiện của ủy nhiệm `WaitCallback` (tham chiếu đến phương thức cần thực thi). Khi một tiểu trình trong thread-pool sẵn sàng, nó nhận công việc kế tiếp từ hàng đợi và thực thi công việc này. Khi đã hoàn tất công việc, thay vì kết thúc, tiểu trình này quay về thread-pool và nhận công việc kế tiếp từ hàng đợi.

Việc sử dụng thread-pool của bộ thực thi giúp đơn giản hóa việc lập trình hỗ trợ đa-tiểu-trình. Tuy nhiên, cần lưu ý đây là thread-pool được hiện thực đơn giản, chỉ nhằm mục đích sử dụng chung. Trước khi quyết định sử dụng thread-pool này, cần xem xét các điểm sau:

- Bộ thực thi quy định số tiểu trình tối đa được cấp cho thread-pool; bạn không thể thay đổi số tối đa này bằng các tham số cấu hình hay từ bên trong mã được-quản-lý. Giới hạn mặc định là 25 tiểu trình cho mỗi *CPU* trong hệ thống. Số tiểu trình tối đa trong thread-pool không giới hạn số các công việc đang chờ trong hàng đợi.
- Cũng như việc cho phép bạn sử dụng thread-pool để thực thi mã lệnh một cách trực tiếp, bộ thực thi còn sử dụng thread-pool cho nhiều mục đích bên trong, bao gồm việc thực thi phương thức một cách bất đồng bộ (xem mục 4.2) và thực thi các sự kiện định thời (xem mục 4.3). Tất cả các công việc này có thể dẫn đến sự tranh chấp giữa các tiểu trình trong thread-pool; nghĩa là hàng đợi có thể trở nên rất dài. Mặc dù độ dài tối đa của hàng đợi chỉ bị giới hạn bởi số lượng bộ nhớ còn lại cho tiến trình của bộ thực thi, nhưng hàng đợi quá dài sẽ làm kéo dài quá trình thực thi của các công việc trong hàng đợi.
- Bạn không nên sử dụng thread-pool để thực thi các tiến trình chạy trong một thời gian dài. Vì số tiểu trình trong thread-pool là có giới hạn, nên chỉ một số ít tiểu trình thuộc các tiến trình loại này cũng sẽ ảnh hưởng đáng kể đến toàn bộ hiệu năng của thread-pool. Đặc biệt, bạn nên tránh đặt các tiểu trình trong thread-pool vào trạng thái đợi trong một thời gian quá dài.
- Bạn không thể điều khiển lịch trình của các tiểu trình trong thread-pool, cũng như không thể thay đổi độ ưu tiên của các công việc. Thread-pool xử lý các công việc theo thứ tự như khi bạn thêm chúng vào hàng đợi.
- Một khi công việc đã được đặt vào hàng đợi thì bạn không thể hủy hay dừng nó.

Ví dụ dưới đây trình bày cách sử dụng lớp `ThreadPool` để thực thi một phương thức có tên là `DisplayMessage`. Ví dụ này sẽ truyền `DisplayMessage` đến thread-pool hai lần, lần đầu không có đối số, lần sau có đối số là đối tượng `MessageInfo` (cho phép kiểm soát thông tin mà tiểu trình sẽ hiển thị).

```
using System;
using System.Threading;

// Lớp dùng để truyền dữ liệu cho phương thức DisplayMessage
// khi nó được thực thi bằng thread-pool.
public class MessageInfo {

    private int iterations;
    private string message;

    // Phương thức khởi dựng nhận các thiết lập cấu hình cho tiểu trình.
    public MessageInfo(int iterations, string message) {

        this.iterations = iterations;
```

```

        this.message = message;
    }

    // Các thuộc tính dùng để lấy các thiết lập cấu hình.
    public int Iterations { get { return iterations; } }
    public string Message { get { return message; } }
}

public class ThreadPoolExample {

    // Hiển thị thông tin ra cửa sổ Console.
    public static void DisplayMessage(object state) {

        // Ép đổi số state sang MessageInfo.
        MessageInfo config = state as MessageInfo;

        // Nếu đối số config là null, không có đối số nào được
        // truyền cho phương thức ThreadPool.QueueUserWorkItem;
        // sử dụng các giá trị mặc định.
        if (config == null) {

            // Hiển thị một thông báo ra cửa sổ Console ba lần.
            for (int count = 0; count < 3; count++) {

                Console.WriteLine("A thread-pool example.");

                // Vào trạng thái chờ, dùng cho mục đích minh họa.
                // Tránh đưa các tiểu trình của thread-pool
                // vào trạng thái chờ trong các ứng dụng thực tế.
                Thread.Sleep(1000);
            }

        } else {

            // Hiển thị một thông báo được chỉ định trước
            // với số lần cũng được chỉ định trước.
            for (int count = 0; count < config.Iterations; count++) {

```

```
        Console.WriteLine(config.Message);

        // Vào trạng thái chờ, dùng cho mục đích minh họa.
        // Tránh đưa các tiểu trình của thread-pool
        // vào trạng thái chờ trong các ứng dụng thực tế.
        Thread.Sleep(1000);
    }
}

public static void Main() {

    // Tạo một đối tượng ủy nhiệm, cho phép chúng ta
    // truyền phương thức DisplayMessage cho thread-pool.
    WaitCallback workMethod =
        new WaitCallback(ThreadPoolExample.DisplayMessage);

    // Thực thi DisplayMessage bằng thread-pool (không có đối số).
    ThreadPool.QueueUserWorkItem(workMethod);

    // Thực thi DisplayMessage bằng thread-pool (truyền một
    // đối tượng MessageInfo cho phương thức DisplayMessage).
    MessageInfo info =
        new MessageInfo(5, "A thread-pool example with arguments.");

    ThreadPool.QueueUserWorkItem(workMethod, info);

    // Nhân Enter để kết thúc.
    Console.WriteLine("Main method complete. Press Enter.");
    Console.ReadLine();
}
}
```


2.

Thực thi phương thức một cách bất đồng bộ

- ?** Bạn cần thực thi một phương thức và tiếp tục thực hiện các công việc khác trong khi phương thức này vẫn chạy trong một tiểu trình riêng biệt. Sau khi phương thức đã hoàn tất, bạn cần lấy trị trả về của nó.
- X** Khai báo một ủy nhiệm có chữ ký giống như phương thức cần thực thi. Sau đó, tạo một thể hiện của ủy nhiệm tham chiếu đến phương thức này. Tiếp theo, gọi phương thức `BeginInvoke` của thể hiện ủy nhiệm để thực thi phương thức của bạn. Kế đến, sử dụng phương thức `EndInvoke` để kiểm tra trạng thái của phương thức cũng như thu lấy trị trả về của nó nếu đã hoàn tất.

Khi cho gọi một phương thức, chúng ta thường thực hiện một cách đồng bộ; nghĩa là mã lệnh thực hiện lời gọi phải đi vào trạng thái dừng (block) cho đến khi phương thức được thực hiện xong. Đây là cách cần thiết khi mã lệnh yêu cầu quá trình thực thi phương thức phải hoàn tất trước khi nó có thể tiếp tục. Tuy nhiên, trong một số trường hợp, bạn lại cần thực thi phương thức một cách bất đồng bộ; nghĩa là bạn cho thực thi phương thức này trong một tiểu trình riêng trong khi vẫn tiếp tục thực hiện các công việc khác.

.NET Framework hỗ trợ chế độ thực thi bất đồng bộ, cho phép bạn thực thi bất kỳ phương thức nào một cách bất đồng bộ bằng một ủy nhiệm. Khi khai báo và biên dịch một ủy nhiệm, trình biên dịch sẽ tự động sinh ra hai phương thức hỗ trợ chế độ thực thi bất đồng bộ: `BeginInvoke` và `EndInvoke`. Khi bạn gọi phương thức `BeginInvoke` của một thể hiện ủy nhiệm, phương thức được tham chiếu bởi ủy nhiệm này được xếp vào hàng đợi để thực thi bất đồng bộ. Quyền kiểm soát quá trình thực thi được trả về cho mã gọi `BeginInvoke` ngay sau đó, và phương thức được tham chiếu sẽ thực thi trong ngữ cảnh của tiểu trình sẵn sàng trước tiên trong thread-pool.

Các đối số của phương thức `BeginInvoke` gồm các đối số được chỉ định bởi ủy nhiệm, cộng với hai đối số dùng khi phương thức thực thi bất đồng bộ kết thúc. Hai đối số này là:

- Một thể hiện của ủy nhiệm `System.AsyncCallback` tham chiếu đến phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ kết thúc. Phương thức này sẽ được thực thi trong ngữ cảnh của một tiểu trình trong thread-pool. Truyền giá trị `null` cho đối số này nghĩa là không có phương thức nào được gọi và bạn phải sử dụng một cơ chế khác để xác định khi nào phương thức thực thi bất đồng bộ kết thúc (sẽ được thảo luận bên dưới).
- Một tham chiếu đối tượng mà bộ thực thi sẽ liên kết với quá trình thực thi bất đồng bộ. Phương thức thực thi bất đồng bộ không thể sử dụng hay truy xuất đến đối tượng này, nhưng mã lệnh của bạn có thể sử dụng nó khi phương thức này kết thúc, cho phép bạn liên kết thông tin trạng thái với quá trình thực thi bất đồng bộ. Ví dụ, đối tượng này cho phép bạn ánh xạ các kết quả với các thao tác bất đồng bộ đã được khởi tạo trong trường hợp bạn khởi tạo nhiều thao tác bất đồng bộ nhưng sử dụng chung một phương thức callback để xử lý việc kết thúc.

Phương thức `EndInvoke` cho phép bạn lấy trị trả về của phương thức thực thi bất đồng bộ, nhưng trước hết bạn phải xác định khi nào nó kết thúc. Dưới đây là bốn kỹ thuật dùng để xác định một phương thức thực thi bất đồng bộ đã kết thúc hay chưa:

- **Blocking**—dừng quá trình thực thi của tiểu trình hiện hành cho đến khi phương thức thực thi bất đồng bộ kết thúc. Điều này rất giống với sự thực thi đồng bộ. Tuy nhiên, nếu linh hoạt chọn thời điểm chính xác để đưa mã lệnh của bạn vào trạng thái dừng (block) thì bạn vẫn còn cơ hội thực hiện thêm một số việc trước khi mã lệnh đi vào trạng thái này.
- **Polling**—lặp đi lặp lại việc kiểm tra trạng thái của phương thức thực thi bất đồng bộ để xác định nó kết thúc hay chưa. Đây là một kỹ thuật rất đơn giản, nhưng nếu xét về mặt xử lý thì không được hiệu quả. Bạn nên tránh các vòng lặp chặt làm lãng phí thời gian của bộ xử lý; tốt nhất là nên đặt tiểu trình thực hiện polling vào trạng thái nghỉ (sleep) trong một khoảng thời gian bằng cách sử dụng `Thread.Sleep` giữa các lần kiểm tra trạng thái. Bởi kỹ thuật polling đòi hỏi bạn phải duy trì một vòng lặp nên hoạt động của tiểu trình chờ sẽ bị giới hạn, tuy nhiên bạn có thể dễ dàng cập nhật tiến độ công việc.
- **Waiting**—sử dụng một đối tượng dẫn xuất từ lớp `System.Threading.WaitHandle` để báo hiệu khi phương thức thực thi bất đồng bộ kết thúc. Waiting là một cải tiến của kỹ thuật polling, nó cho phép bạn chờ nhiều phương thức thực thi bất đồng bộ kết thúc. Bạn cũng có thể chỉ định các giá trị time-out cho phép tiểu trình thực hiện waiting dừng lại nếu phương thức thực thi bất đồng bộ đã diễn ra quá lâu, hoặc bạn muốn cập nhật định kỳ bộ chỉ trạng thái.
- **Callbacks**—Callback là một phương thức mà bộ thực thi sẽ gọi khi phương thức thực thi bất đồng bộ kết thúc. Mã lệnh thực hiện lời gọi không cần thực hiện bất kỳ thao tác kiểm tra nào, nhưng vẫn có thể tiếp tục thực hiện các công việc khác. Callback rất linh hoạt nhưng cũng rất phức tạp, đặc biệt khi có nhiều phương thức thực thi bất đồng bộ chạy đồng thời nhưng sử dụng cùng một callback. Trong những trường hợp như thế, bạn phải sử dụng các đối tượng trạng thái thích hợp để so trùng các phương thức đã hoàn tất với các phương thức đã khởi tạo.

Lớp `AsyncExecutionExample` trong ví dụ dưới đây mô tả cơ chế thực thi bất đồng bộ. Nó sử dụng một ủy nhiệm có tên là `AsyncExampleDelegate` để thực thi bất đồng bộ một phương thức có tên là `LongRunningMethod`. Phương thức `LongRunningMethod` sử dụng `Thread.Sleep` để mô phỏng một phương thức có thời gian thực thi dài.

```
// Ủy nhiệm cho phép bạn thực hiện việc thực thi bất đồng bộ
// của AsyncExecutionExample.LongRunningMethod.
public delegate DateTime AsyncExampleDelegate(int delay, string name);

// Phương thức có thời gian thực thi dài.
public static DateTime LongRunningMethod(int delay, string name) {

    Console.WriteLine("{0} : {1} example - thread starting.",
        DateTime.Now.ToString("HH:mm:ss.ffff"), name);
```

```
// Mô phỏng việc xử lý tốn nhiều thời gian.
Thread.Sleep(delay);

Console.WriteLine("{0} : {1} example - thread finishing.",
    DateTime.Now.ToString("HH:mm:ss.ffff"), name);

// Trả về thời gian hoàn tất phương thức.
return DateTime.Now;
}
```

`AsyncExecutionExample` chứa năm phương thức diễn đạt các cách tiếp cận khác nhau về việc kết thúc phương thức thực thi bất đồng bộ. Dưới đây sẽ mô tả và cung cấp mã lệnh cho các phương thức đó.

1. Phương thức `BlockingExample`

Phương thức `BlockingExample` thực thi bất đồng bộ phương thức `LongRunningMethod` và tiếp tục thực hiện công việc của nó trong một khoảng thời gian. Khi xử lý xong công việc này, `BlockingExample` chuyển sang trạng thái dừng (block) cho đến khi phương thức `LongRunningMethod` kết thúc. Để vào trạng thái dừng, `BlockingExample` thực thi phương thức `EndInvoke` của thể hiện ủy nhiệm `AsyncExampleDelegate`. Nếu phương thức `LongRunningMethod` kết thúc, `EndInvoke` trả về ngay lập tức, nếu không, `BlockingExample` chuyển sang trạng thái dừng cho đến khi phương thức `LongRunningMethod` kết thúc.

```
public static void BlockingExample() {

    Console.WriteLine(Environment.NewLine +
        "*** Running Blocking Example ***");

    // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null cho
    // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
    AsyncExampleDelegate longRunningMethod =
        new AsyncExampleDelegate(LongRunningMethod);

    IAsyncResult asyncResult = longRunningMethod.BeginInvoke(2000,
        "Blocking", null, null);

    // Thực hiện công việc khác cho đến khi
    // sẵn sàng đi vào trạng thái dừng.
    for (int count = 0; count < 3; count++) {
        Console.WriteLine("{0} : Continue processing until " +
            "ready to block...",
```

```

        DateTime.Now.ToString("HH:mm:ss.ffff"));
        Thread.Sleep(200);
    }

    // Đi vào trạng thái dùng cho đến khi phương thức
    // thực thi bất đồng bộ kết thúc và thu lấy kết quả.
    Console.WriteLine("{0} : Blocking until method is " +
        "complete...", DateTime.Now.ToString("HH:mm:ss.ffff"));
    DateTime completion =
        longRunningMethod.EndInvoke(asyncResult);

    // Hiển thị thông tin kết thúc.
    Console.WriteLine("{0} : Blocking example complete.",
        completion.ToString("HH:mm:ss.ffff"));
}

```

2. Phương thức `PollingExample`

Phương thức `PollingExample` thực thi bất đồng bộ phương thức `LongRunningMethod` và sau đó thực hiện vòng lặp *polling* cho đến khi `LongRunningMethod` kết thúc. `PollingExample` kiểm tra thuộc tính `IsComplete` của thể hiện `IAsyncResult` (được trả về bởi `BeginInvoke`) để xác định phương thức `LongRunningMethod` đã kết thúc hay chưa, nếu chưa, `PollingExample` sẽ gọi `Thread.Sleep`.

```

public static void PollingExample() {

    Console.WriteLine(Environment.NewLine +
        "*** Running Polling Example ***");

    // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null cho
    // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
    AsyncExampleDelegate longRunningMethod =
        new AsyncExampleDelegate(LongRunningMethod);

    IAsyncResult asyncResult = longRunningMethod.BeginInvoke(2000,
        "Polling", null, null);

    // Thực hiện polling để kiểm tra phương thức thực thi
    // bất đồng bộ kết thúc hay chưa. Nếu chưa kết thúc

```

```
// thì đi vào trạng thái chờ trong 300 mini-giây
// trước khi thực hiện polling lần nữa.
Console.WriteLine("{0} : Poll repeatedly until method is " +
    "complete...", DateTime.Now.ToString("HH:mm:ss.ffff"));
while(!asyncResult.IsCompleted) {
    Console.WriteLine("{0} : Polling...",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
    Thread.Sleep(300);
}

// Thu lấy kết quả của phương thức thực thi bất đồng bộ.
DateTime completion =
    longRunningMethod.EndInvoke(asyncResult);

// Hiển thị thông tin kết thúc.
Console.WriteLine("{0} : Polling example complete.",
    completion.ToString("HH:mm:ss.ffff"));
}
```

3. Phương thức `WaitingExample`

Phương thức `WaitingExample` thực thi bất đồng bộ phương thức `LongRunningExample` và sau đó chờ cho đến khi `LongRunningMethod` kết thúc. `WaitingExample` sử dụng thuộc tính `AsyncWaitHandle` của thể hiện `IAsyncResult` (được trả về bởi `BeginInvoke`) để có được một `WaitHandle` và sau đó gọi phương thức `WaitOne` của `WaitHandle`. Việc sử dụng giá trị time-out cho phép `WaitingExample` dừng quá trình đợi để thực hiện công việc khác hoặc dừng hoàn toàn nếu phương thức thực thi bất đồng bộ diễn ra quá lâu.

```
public static void WaitingExample() {

    Console.WriteLine(Environment.NewLine +
        "*** Running Waiting Example ***");

    // Gọi LongRunningMethod một cách bất đồng bộ. Truyền null cho
    // cả ủy nhiệm callback và đối tượng trạng thái bất đồng bộ.
    AsyncExampleDelegate longRunningMethod =
        new AsyncExampleDelegate(LongRunningMethod);

    IAsyncResult asyncResult = longRunningMethod.BeginInvoke(2000,
        "Waiting", null, null);
```

```

// Đợi phương thức thực thi bất đồng bộ kết thúc.
// Time-out sau 300 mili-giây và hiển thị trạng thái ra
// của sổ Console trước khi tiếp tục đợi.
Console.WriteLine("{0} : Waiting until method is complete...",
    DateTime.Now.ToString("HH:mm:ss.ffff"));
while(!asyncResult.AsyncWaitHandle.WaitOne(300, false)) {
    Console.WriteLine("{0} : Wait timeout...",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
}

// Thu lấy kết quả của phương thức thực thi bất đồng bộ.
DateTime completion =
    longRunningMethod.EndInvoke(asyncResult);

// Hiển thị thông tin kết thúc.
Console.WriteLine("{0} : Waiting example complete.",
    completion.ToString("HH:mm:ss.ffff"));
}

```

4. Phương thức `WaitAllExample`

Phương thức `WaitAllExample` thực thi bất đồng bộ phương thức `LongRunningMethod` nhiều lần và sau đó sử dụng mảng các đối tượng `WaitHandle` để đợi cho đến khi tất cả các phương thức kết thúc.

```

public static void WaitAllExample() {

    Console.WriteLine(Environment.NewLine +
        "*** Running WaitAll Example ***");

    // Một ArrayList chứa các thể hiện IAsyncResult
    // cho các phương thức thực thi bất đồng bộ.
    ArrayList asyncResults = new ArrayList(3);

    // Gọi ba lần LongRunningMethod một cách bất đồng bộ.
    // Truyền null cho cả ủy nhiệm callback và đối tượng
    // trạng thái bất đồng bộ. Thêm thể hiện IAsyncResult
    // cho mỗi phương thức vào ArrayList.
    AsyncExampleDelegate longRunningMethod =

```

```

        new AsyncExampleDelegate(LongRunningMethod);

asyncResults.Add(longRunningMethod.BeginInvoke(3000,
    "WaitAll 1", null, null));

asyncResults.Add(longRunningMethod.BeginInvoke(2500,
    "WaitAll 2", null, null));

asyncResults.Add(longRunningMethod.BeginInvoke(1500,
    "WaitAll 3", null, null));

// Tạo một mảng các đối tượng WaitHandle,
// sẽ được sử dụng để đợi tất cả các phương thức
// thực thi bất đồng bộ kết thúc.
WaitHandle[] waitHandles = new WaitHandle[3];

for (int count = 0; count < 3; count++) {

    waitHandles[count] =
        ((IAsyncResult)asyncResults[count]).AsyncWaitHandle;
}

// Đợi cả ba phương thức thực thi bất đồng bộ kết thúc.
// Time-out sau 300 mili-giây và hiển thị trạng thái ra
// của sổ Console trước khi tiếp tục đợi.
Console.WriteLine("{0} : Waiting until all 3 methods are " +
    "complete...", DateTime.Now.ToString("HH:mm:ss.ffff"));
while(!WaitHandle.WaitAll(waitHandles, 300, false)) {
    Console.WriteLine("{0} : WaitAll timeout...",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
}

// Kiểm tra kết quả của mỗi phương thức và xác định
// thời gian phương thức cuối cùng kết thúc.
DateTime completion = DateTime.MinValue;

foreach (IAsyncResult result in asyncResults) {

```

```

        DateTime time = longRunningMethod.EndInvoke(result);
        if (time > completion) completion = time;
    }

    // Hiển thị thông tin kết thúc.
    Console.WriteLine("{0} : WaitAll example complete.",
        completion.ToString("HH:mm:ss.ffff"));
}

```

5. Phương thức `CallbackExample`

Phương thức `CallbackExample` thực thi bất đồng bộ phương thức `LongRunningMethod` và truyền một thẻ hiện ủy nhiệm `AsyncCallback` (tham chiếu đến phương thức `CallbackHandler`) cho phương thức `BeginInvoke`. Phương thức `CallbackHandler` sẽ được gọi một cách tự động khi phương thức `LongRunningMethod` kết thúc, kết quả là phương thức `CallbackExample` vẫn tiếp tục thực hiện công việc.

```

public static void CallbackExample() {

    Console.WriteLine(Environment.NewLine +
        "*** Running Callback Example ***");

    // Gọi LongRunningMethod một cách bất đồng bộ. Truyền một
    // thẻ hiện ủy nhiệm AsyncCallback tham chiếu đến
    // phương thức CallbackHandler. CallbackHandler sẽ
    // tự động được gọi khi phương thức thực thi bất đồng bộ
    // kết thúc. Truyền một tham chiếu đến thẻ hiện ủy nhiệm
    // AsyncExampleDelegate như một trạng thái bất đồng bộ;
    // nếu không, phương thức callback không thể truy xuất
    // thẻ hiện ủy nhiệm để gọi EndInvoke.
    AsyncExampleDelegate longRunningMethod =
        new AsyncExampleDelegate(LongRunningMethod);

    IAsyncResult asyncResult = longRunningMethod.BeginInvoke(2000,
        "Callback", new AsyncCallback(CallbackHandler),
        longRunningMethod);

    // Tiếp tục với công việc khác.
    for (int count = 0; count < 15; count++) {

```



```

        Console.WriteLine("{0} : Continue processing...",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
        Thread.Sleep(200);
    }
}

// Phương thức xử lý việc kết thúc bất đồng bộ bằng callbacks.
public static void CallbackHandler(IAsyncResult result) {

    // Trích tham chiếu đến thể hiện AsyncExampleDelegate
    // từ thể hiện IAsyncResult.
    AsyncExampleDelegate longRunningMethod =
        (AsyncExampleDelegate) result.AsyncState;

    // Thu lấy kết quả của phương thức thực thi bất đồng bộ.
    DateTime completion = longRunningMethod.EndInvoke(result);

    // Hiển thị thông tin kết thúc.
    Console.WriteLine("{0} : Callback example complete.",
        completion.ToString("HH:mm:ss.ffff"));
}

```

3.

Thực thi phương thức bằng Timer

?

Bạn cần thực thi một phương thức trong một tiểu trình riêng theo chu kỳ hay ở một thời điểm xác định.

✕

Khai báo một phương thức trả về void và chỉ nhận một đối tượng làm đối số. Sau đó, tạo một thể hiện ủy nhiệm `System.Threading.TimerCallback` tham chiếu đến phương thức này. Tiếp theo, tạo một đối tượng `System.Threading.Timer` và truyền nó cho thể hiện ủy nhiệm `TimerCallback` cùng với một đối tượng trạng thái mà `Timer` sẽ truyền cho phương thức của bạn khi `Timer` hết hiệu lực. Bộ thực thi sẽ chờ cho đến khi `Timer` hết hiệu lực và sau đó gọi phương thức của bạn bằng một tiểu trình trong thread-pool.

Thông thường, rất hữu ích khi thực thi một phương thức ở một thời điểm xác định hay ở những thời khoảng xác định. Ví dụ, bạn cần sao lưu dữ liệu lúc 1:00 AM mỗi ngày hay xóa vùng đệm dữ liệu mỗi 20 phút. Lớp `Timer` giúp việc định thời thực thi một phương thức trở nên dễ dàng, cho phép bạn thực thi một phương thức được tham chiếu bởi ủy nhiệm `TimerCallback` ở những thời khoảng nhất định. Phương thức được tham chiếu sẽ thực thi trong ngữ cảnh của một tiểu trình trong thread-pool.

Khi tạo một đối tượng `Timer`, bạn cần chỉ định hai thời khoảng (thời khoảng có thể được chỉ định là các giá trị kiểu `int`, `long`, `uint`, hay `System.TimeSpan`):

- Giá trị đầu tiên là thời gian trễ (tính bằng mili-giây) để phương thức của bạn được thực thi lần đầu tiên. Chỉ định giá trị 0 để thực thi phương thức ngay lập tức, và chỉ định `System.Threading.Timeout.Infinite` để tạo `Timer` ở trạng thái chưa bắt đầu (*unstarted*).
- Giá trị thứ hai là khoảng thời gian mà `Timer` sẽ lặp lại việc gọi phương thức của bạn sau lần thực thi đầu tiên. Nếu bạn chỉ định giá trị 0 hay `Timeout.Infinite` thì `Timer` chỉ thực thi phương thức một lần duy nhất (với điều kiện thời gian trễ ban đầu không phải là `Timeout.Infinite`). Đối số thứ hai có thể cung cấp bằng các trị kiểu `int`, `long`, `uint`, hay `System.TimeSpan`.

Sau khi tạo đối tượng `Timer`, bạn cũng có thể thay đổi các thời khoảng được sử dụng bởi `Timer` bằng phương thức `Change`, nhưng bạn không thể thay đổi phương thức sẽ được gọi. Khi đã dùng xong `Timer`, bạn nên gọi phương thức `Timer.Dispose` để giải phóng tài nguyên hệ thống bị chiếm giữ bởi `Timer`. Việc hủy `Timer` cũng hủy luôn phương thức đã được định thời thực thi.

Lớp `TimerExample` dưới đây trình bày cách sử dụng `Timer` để gọi một phương thức có tên là `TimerHandler`. Ban đầu, `Timer` được cấu hình để gọi `TimerHandler` sau hai giây và lặp lại sau một giây. Ví dụ này cũng trình bày cách sử dụng phương thức `Timer.Change` để thay đổi các thời khoảng.

```
using System;
using System.Threading;

public class TimerExample {

    // Phương thức sẽ được thực thi khi Timer hết hiệu lực.
    // Hiển thị một thông báo ra cửa sổ Console.
    private static void TimerHandler(object state) {

        Console.WriteLine("{0} : {1}",
            DateTime.Now.ToString("HH:mm:ss.ffff"), state);
    }

    public static void Main() {

        // Tạo một thẻ hiện ủy nhiệm TimerCallback mới
        // tham chiếu đến phương thức tĩnh TimerHandler.
        // TimerHandler sẽ được gọi khi Timer hết hiệu lực.
```

```

TimerCallback handler = new TimerCallback(TimerHandler);

// Tạo một đối tượng trạng thái, đối tượng này sẽ được
// truyền cho phương thức TimerHandler.
// Trong trường hợp này, một thông báo sẽ được hiển thị.
string state = "Timer expired.";

Console.WriteLine("{0} : Creating Timer.",
    DateTime.Now.ToString("HH:mm:ss.ffff"));

// Tạo một Timer, phát sinh lần đầu tiên sau hai giây
// và sau đó là mỗi giây.
using (Timer timer = new Timer(handler, state, 2000, 1000)) {

    int period;

    // Đọc thời khoảng mới từ Console cho đến khi
    // người dùng nhập 0. Các giá trị không hợp lệ
    // sẽ sử dụng giá trị mặc định là 0 (dùng ví dụ).
    do {

        try {
            period = Int32.Parse(Console.ReadLine());
        } catch {
            period = 0;
        }

        // Thay đổi Timer với thời khoảng mới.
        if (period > 0) timer.Change(0, period);

    } while (period > 0);
}

// Nhấn Enter để kết thúc.
Console.WriteLine("Main method complete. Press Enter.");
Console.ReadLine();
}
}

```

Mặc dù `Timer` thường được sử dụng để gọi thực thi các phương thức ở những thời khoảng, nhưng nó cũng cung cấp cách thức để thực thi một phương thức ở một thời điểm xác định. Bạn cần phải tính toán khoảng thời gian từ thời điểm hiện tại đến thời điểm cần thực thi. Ví dụ dưới đây sẽ thực hiện điều này:

```
public static void RunAt(DateTime execTime) {

    // Tính khoảng thời gian từ thời điểm hiện tại
    // đến thời điểm cần thực thi.
    TimeSpan waitTime = execTime - DateTime.Now;

    if (waitTime < new TimeSpan(0)) waitTime = new TimeSpan(0);

    // Tạo một thể hiện ủy nhiệm TimerCallback mới
    // tham chiếu đến phương thức tĩnh TimerHandler.
    // TimerHandler sẽ được gọi khi Timer hết hiệu lực.
    TimerCallback handler = new TimerCallback(TimerHandler);

    // Tạo một Timer chỉ phát sinh một lần tại thời điểm
    // được chỉ định. Chỉ định thời khoảng thứ hai là -1
    // để ngăn Timer thực thi lặp lại phương thức.
    new Timer(handler, null, waitTime, new TimeSpan(-1));
}
```

4. Thực thi phương thức bằng cách ra hiệu đối tượng *WaitHandle*

?

Bạn muốn thực thi một hay nhiều phương thức một cách tự động khi một đối tượng dẫn xuất từ lớp `System.Threading.WaitHandle` đi vào trạng thái *signaled*.

✕

Tạo một thể hiện ủy nhiệm `System.Threading.WaitOrTimerCallback` tham chiếu đến phương thức cần thực thi. Sau đó, đăng ký thể hiện ủy nhiệm và đối tượng `WaitHandle` với thread-pool bằng phương thức tĩnh `ThreadPool.RegisterWaitForSingleObject`.

Bạn có thể sử dụng các lớp dẫn xuất từ `WaitHandle` (đã được thảo luận trong mục 4.2) để gọi thực thi một phương thức. Bằng phương thức `RegisterWaitForSingleObject` của lớp `ThreadPool`, bạn có thể đăng ký thể hiện ủy nhiệm `WaitOrTimerCallback` với thread-pool khi một đối tượng dẫn xuất từ `WaitHandle` đi vào trạng thái *signaled*. Bạn có thể cấu hình thread-pool để thực thi phương thức chỉ một lần hay tự động đăng ký lại phương thức mỗi khi `WaitHandle` đi vào trạng thái *signaled*. Nếu `WaitHandle` đã ở trạng thái *signaled* khi bạn gọi `RegisterWaitForSingleObject`, phương thức sẽ thực thi ngay lập tức. Phương thức

Unregister của đối tượng `System.Threading.RegisteredWaitHandle` (được trả về bởi phương thức `RegisterWaitForSingleObject`) được sử dụng để hủy bỏ việc đăng ký.

Lớp thường được dùng làm bộ kích hoạt là `AutoResetEvent`, nó sẽ tự động chuyển sang trạng thái *unsignaled* sau khi ở trạng thái *signaled*. Tuy nhiên, bạn cũng có thể thay đổi trạng thái *signaled* theo ý muốn bằng lớp `ManualResetEvent` hay `Mutex`. Ví dụ dưới đây trình bày cách sử dụng một `AutoResetEvent` để kích hoạt thực thi một phương thức có tên là `EventHandler`.

```
using System;
using System.Threading;

public class EventExecutionExample {

    // Phương thức sẽ được thực thi khi AutoResetEvent đi vào trạng
    // thái signaled hoặc quá trình đợi hết thời gian (time-out).
    private static void EventHandler(object state, bool timedout) {

        // Hiển thị thông báo thích hợp ra cửa sổ Console
        // tùy vào quá trình đợi đã hết thời gian hay
        // AutoResetEvent đã ở trạng thái signaled.
        if (timedout) {

            Console.WriteLine("{0} : Wait timed out.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
        } else {

            Console.WriteLine("{0} : {1}",
                DateTime.Now.ToString("HH:mm:ss.ffff"), state);
        }
    }

    public static void Main() {

        // Tạo một AutoResetEvent ở trạng thái unsignaled.
        AutoResetEvent autoEvent = new AutoResetEvent(false);

        // Tạo một thẻ hiện ủy nhiệm WaitOrTimerCallback
        // tham chiếu đến phương thức tĩnh EventHandler.
        // EventHandler sẽ được gọi khi AutoResetEvent đi vào
        // trạng thái signaled hay quá trình đợi hết thời gian.
```

```

WaitOrTimerCallback handler =
    new WaitOrTimerCallback(EventHandler);

// Tạo đối tượng trạng thái (được truyền cho phương thức
// thụ lý sự kiện khi nó được kích hoạt). Trong trường hợp
// này, một thông báo sẽ được hiển thị.
string state = "AutoResetEvent signaled.";

// Đăng ký thẻ hiện ủy nhiệm để đợi AutoResetEvent đi vào
// trạng thái signaled. Thiết lập giá trị time-out là 3 giây.
RegisteredWaitHandle handle =
    ThreadPool.RegisterWaitForSingleObject(autoEvent, handler,
        state, 3000, false);

Console.WriteLine("Press ENTER to signal the AutoResetEvent" +
    " or enter \"Cancel\" to unregister the wait operation.");

while (Console.ReadLine().ToUpper() != "CANCEL") {

    // Nếu "Cancel" không được nhập vào Console,
    // AutoResetEvent sẽ đi vào trạng thái signal,
    // và phương thức EventHandler được thực thi.
    // AutoResetEvent sẽ tự động trở về trạng thái unsignaled.
    autoEvent.Set();
}

// Hủy bỏ việc đăng ký quá trình đợi.
Console.WriteLine("Unregistering wait operation.");
handle.Unregister(null);

// Nhấn Enter để kết thúc.
Console.WriteLine("Main method complete. Press Enter.");
Console.ReadLine();
}
}

```

5. *Thực thi phương thức bằng tiểu trình mới*

- ?** Bạn muốn thực thi mã lệnh trong một tiểu trình riêng, và muốn kiểm soát hoàn toàn quá trình thực thi và trạng thái của tiểu trình đó.
- X** Khai báo một phương thức trả về `void` và không có đối số. Sau đó, tạo một thể hiện ủy nhiệm `System.Threading.ThreadStart` tham chiếu đến phương thức này. Tiếp theo, tạo một đối tượng `System.Threading.Thread` mới, và truyền thể hiện ủy nhiệm cho phương thức khởi dựng của nó. Kế đến, gọi phương thức `Thread.Start` để bắt đầu thực thi phương thức của bạn.

Để tăng độ linh hoạt và mức độ kiểm soát khi hiện thực các ứng dụng hỗ trợ đa-tiểu-trình, bạn phải trực tiếp tạo và quản lý các tiểu trình. Đây là cách tiếp cận phức tạp nhất trong việc lập trình hỗ trợ đa-tiểu-trình, nhưng đó cũng là cách duy nhất vượt qua những hạn chế cố hữu trong các cách tiếp cận sử dụng các tiểu trình trong thread-pool, như đã được thảo luận trong bốn mục trước. Lớp `Thread` cung cấp một cơ chế mà qua đó bạn có thể tạo và kiểm soát các tiểu trình. Để tạo và chạy một tiểu trình mới, bạn hãy tiến hành theo các bước sau:

1. Tạo một đối tượng ủy nhiệm `ThreadStart` tham chiếu đến phương thức chứa mã lệnh mà bạn muốn dùng một tiểu trình mới để chạy nó. Giống như các ủy nhiệm khác, `ThreadStart` có thể tham chiếu đến một phương thức tĩnh hay phương thức của một đối tượng. Phương thức được tham chiếu phải trả về `void` và không có đối số.
2. Tạo một đối tượng `Thread`, và truyền thể hiện ủy nhiệm `ThreadStart` cho phương thức khởi dựng của nó. Tiểu trình mới có trạng thái ban đầu là `Unstarted` (một thành viên thuộc kiểu liệt kê `System.Threading.ThreadState`).
3. Gọi thực thi phương thức `Start` của đối tượng `Thread` để chuyển trạng thái của nó sang `ThreadState.Running` và bắt đầu thực thi phương thức được tham chiếu bởi thể hiện ủy nhiệm `ThreadStart` (nếu bạn gọi phương thức `Start` quá một lần, nó sẽ ném ngoại lệ `System.Threading.ThreadStateException`).

Vì ủy nhiệm `ThreadStart` khai báo không có đối số, bạn không thể truyền dữ liệu trực tiếp cho phương thức được tham chiếu. Để truyền dữ liệu cho tiểu trình mới, bạn phải cấu hình dữ liệu là khả truy xuất đối với mã lệnh đang chạy trong tiểu trình mới. Cách tiếp cận thông thường là tạo một lớp đóng gói cả dữ liệu cần cho tiểu trình và phương thức được thực thi bởi tiểu trình. Khi muốn chạy một tiểu trình mới, bạn hãy tạo một đối tượng của lớp này, cấu hình trạng thái cho nó, và rồi chạy tiểu trình. Dưới đây là một ví dụ:

```
using System;
using System.Threading;

public class ThreadExample {

    // Các biến giữ thông tin trạng thái.
    private int iterations;
    private string message;
    private int delay;
```

```
public ThreadExample(int iterations, string message, int delay) {

    this.iterations = iterations;
    this.message = message;
    this.delay = delay;
}

public void Start() {

    // Tạo một thẻ hiện ủy nhiệm ThreadStart
    // tham chiếu đến DisplayMessage.
    ThreadStart method = new ThreadStart(this.DisplayMessage);

    // Tạo một đối tượng Thread và truyền thẻ hiện ủy nhiệm
    // ThreadStart cho phương thức khởi dụng của nó.
    Thread thread = new Thread(method);

    Console.WriteLine("{0} : Starting new thread.",
        DateTime.Now.ToString("HH:mm:ss.ffff"));

    // Khởi chạy tiểu trình mới.
    thread.Start();
}

private void DisplayMessage() {

    // Hiển thị thông báo ra của sổ Console với số lần
    // được chỉ định (iterations), nghỉ giữa mỗi thông báo
    // một khoảng thời gian được chỉ định (delay).
    for (int count = 0; count < iterations; count++) {

        Console.WriteLine("{0} : {1}",
            DateTime.Now.ToString("HH:mm:ss.ffff"), message);

        Thread.Sleep(delay);
    }
}
```



```

public static void Main() {

    // Tạo một đối tượng ThreadExample.
    ThreadExample example =
        new ThreadExample(5, "A thread example.", 500);

    // Khởi chạy đối tượng ThreadExample.
    example.Start();

    // Tiếp tục thực hiện công việc khác.
    for (int count = 0; count < 13; count++) {
        Console.WriteLine("{0} : Continue processing...",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
        Thread.Sleep(200);
    }

    // Nhấn Enter để kết thúc.
    Console.WriteLine("Main method complete. Press Enter.");
    Console.ReadLine();
}
}

```

6. *Điều khiển quá trình thực thi của một tiểu trình*

? **Bạn cần nắm quyền điều khiển khi một tiểu trình chạy và dừng, và có thể tạm dừng quá trình thực thi của một tiểu trình.**

✕ **Sử dụng các phương thức `Abort`, `Interrupt`, `Resume`, `Start`, và `Suspend` của `Thread` mà bạn cần điều khiển.**

Các phương thức của lớp `Thread` được tóm tắt trong bảng 4.1 cung cấp một cơ chế điều khiển mức cao lên quá trình thực thi của một tiểu trình. Mỗi phương thức này trở về tiểu trình đang gọi ngay lập tức. Tuy nhiên, trạng thái của tiểu trình hiện hành đóng vai trò quan trọng trong kết quả của lời gọi phương thức, và trạng thái của một tiểu trình có thể thay đổi nhanh chóng. Kết quả là, bạn phải viết mã để bắt và thụ lý các ngoại lệ có thể bị ném khi bạn cố điều khiển quá trình thực thi của một `Thread`.

Lớp `ThreadControlExample` dưới đây trình bày cách sử dụng các phương thức được liệt kê trong bảng 4.1. Ví dụ này khởi chạy một tiểu trình thứ hai, hiển thị định kỳ một thông báo ra cửa sổ *Console* và rồi đi vào trạng thái nghỉ (*sleep*). Bằng cách nhập các lệnh tại đầu nhắc lệnh, bạn có thể gián đoạn, tạm hoãn, phục hồi, và hủy bỏ tiểu trình thứ hai.

Bảng 4.1 Điều khiển quá trình thực thi của một tiểu trình

Phương thức	Mô tả
Abort	Kết thúc một tiểu trình bằng cách ném ngoại lệ <code>System.Threading.ThreadAbortException</code> trong mã lệnh đang được chạy. Mã lệnh của tiểu trình bị hủy có thể bắt ngoại lệ <code>ThreadAbortException</code> để thực hiện việc dọn dẹp, nhưng bộ thực thi sẽ tự động ném ngoại lệ này lần nữa để bảo đảm tiểu trình kết thúc, trừ khi <code>ResetAbort</code> được gọi. Abort trở về ngay lập tức, nhưng bộ thực thi xác định chính xác khi nào ngoại lệ bị ném, do đó bạn không thể cho rằng tiểu trình đã kết thúc bởi Abort đã trở về. Bạn nên sử dụng các kỹ thuật được mô tả trong mục 4.7 nếu cần xác định khi nào tiểu trình này thật sự kết thúc. Một khi đã hủy một tiểu trình, bạn không thể khởi chạy lại nó.
Interrupt	Ném ngoại lệ <code>System.Threading.ThreadInterruptedException</code> (trong mã lệnh đang được chạy) lúc tiểu trình đang ở trạng thái <code>WaitSleepJoin</code> . Điều này nghĩa là tiểu trình này đã gọi <code>Sleep</code> , <code>Join</code> (mục 4.7); hoặc đang đợi <code>WaitHandle</code> ra hiệu (để đi vào trạng thái <i>signaled</i>) hay đang đợi một đối tượng dùng cho sự đồng bộ tiểu trình (mục 4.8). Nếu tiểu trình này không ở trạng thái <code>WaitSleepJoin</code> , <code>ThreadInterruptedException</code> sẽ bị ném sau khi tiểu trình đi vào trạng thái <code>WaitSleepJoin</code> .
Resume	Phục hồi quá trình thực thi của một tiểu trình đã bị tạm hoãn (xem phương thức <code>Suspend</code>). Việc gọi <code>Resume</code> trên một tiểu trình chưa bị tạm hoãn sẽ sinh ra ngoại lệ <code>System.Threading.ThreadStateException</code> trong tiểu trình đang gọi.
Start	Khởi chạy tiểu trình mới; xem mục 4.5 để biết cách sử dụng phương thức <code>Start</code> .
Suspend	Tạm hoãn quá trình thực thi của một tiểu trình cho đến khi phương thức <code>Resume</code> được gọi. Việc tạm hoãn một tiểu trình đã bị tạm hoãn sẽ không có hiệu lực, nhưng việc gọi <code>Suspend</code> trên một tiểu trình chưa khởi chạy hoặc đã kết thúc sẽ sinh ra ngoại lệ <code>ThreadStateException</code> trong tiểu trình đang gọi.

```
using System;
using System.Threading;

public class ThreadControlExample {

    private static void DisplayMessage() {

        // Lặp đi lặp lại việc hiển thị một thông báo ra cửa sổ Console.
```

```

while (true) {

    try {

        Console.WriteLine("{0} : Second thread running. Enter"
            + " (S)uspend, (R)esume, (I)nterrupt, or (E)xit.",
            DateTime.Now.ToString("HH:mm:ss.ffff"));

        // Nghỉ 2 giây.
        Thread.Sleep(2000);

    } catch (ThreadInterruptedException) {

        // Tiểu trình đã bị gián đoạn. Việc bắt ngoại lệ
        // ThreadInterruptedException cho phép ví dụ này
        // thực hiện hành động phù hợp và tiếp tục thực thi.
        Console.WriteLine("{0} : Second thread interrupted.",
            DateTime.Now.ToString("HH:mm:ss.ffff"));

    } catch (ThreadAbortException abortEx) {

        // Đối tượng trong thuộc tính
        // ThreadAbortException.ExceptionState được cung cấp
        // bởi tiểu trình đã gọi Thread.Abort.
        // Trong trường hợp này, nó chứa một chuỗi
        // mô tả lý do của việc hủy bỏ.
        Console.WriteLine("{0} : Second thread aborted ({1})",
            DateTime.Now.ToString("HH:mm:ss.ffff"),
            abortEx.ExceptionState);

        // Mặc dù ThreadAbortException đã được thụ lý,
        // bộ thực thi sẽ ném nó lần nữa để bảo đảm
        // tiểu trình kết thúc.
    }

}

}

public static void Main() {

```

```

// Tạo một đối tượng Thread và truyền cho nó một thẻ hiện
// ủy nhiệm ThreadStart tham chiếu đến DisplayMessage.
Thread thread = new Thread(new ThreadStart(DisplayMessage));

Console.WriteLine("{0} : Starting second thread.",
    DateTime.Now.ToString("HH:mm:ss.ffff"));

// Khởi chạy tiểu trình thứ hai.
thread.Start();

// Lặp và xử lý lệnh do người dùng nhập.
char command = ' ';

do {

    string input = Console.ReadLine();
    if (input.Length > 0) command = input.ToUpper()[0];
    else command = ' ';

    switch (command) {

        case 'S':
            // Tạm hoãn tiểu trình thứ hai.
            Console.WriteLine("{0} : Suspending second thread.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));
            thread.Suspend();
            break;

        case 'R':
            // Phục hồi tiểu trình thứ hai.
            try {
                Console.WriteLine("{0} : Resuming second " +
                    "thread.",
                    DateTime.Now.ToString("HH:mm:ss.ffff"));
                thread.Resume();
            }

```

```

    } catch (ThreadStateException) {
        Console.WriteLine("{0} : Thread wasn't " +
            "suspended.",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
    }
    break;

case 'I':
    // Gián đoạn tiểu trình thứ hai.
    Console.WriteLine("{0} : Interrupting second " +
        "thread.",
        DateTime.Now.ToString("HH:mm:ss.ffff"));
    thread.Interrupt();
    break;

case 'E':
    // Hủy bỏ tiểu trình thứ hai và truyền một đối tượng
    // trạng thái cho tiểu trình đang bị hủy,
    // trong trường hợp này là một thông báo.
    Console.WriteLine("{0} : Aborting second thread.",
        DateTime.Now.ToString("HH:mm:ss.ffff"));

    thread.Abort("Terminating example.");

    // Đợi tiểu trình thứ hai kết thúc.
    thread.Join();
    break;
}
} while (command != 'E');

// Nhấn Enter để kết thúc.
Console.WriteLine("Main method complete. Press Enter.");
Console.ReadLine();
}
}

```

7.

Nhận biết khi nào một tiểu trình kết thúc**Bạn muốn biết khi nào một tiểu trình đã kết thúc.****Sử dụng thuộc tính `IsAlive` hay phương thức `Join` của lớp `Thread`.**

Cách dễ nhất để kiểm tra một tiểu trình đã kết thúc hay chưa là kiểm tra thuộc tính `Thread.IsAlive`. Thuộc tính này trả về `true` nếu tiểu trình đã được khởi chạy nhưng chưa kết thúc hay bị hủy.

Thông thường, bạn sẽ cần một tiểu trình để đợi một tiểu trình khác hoàn tất việc xử lý của nó. Thay vì kiểm tra thuộc tính `IsAlive` trong một vòng lặp, bạn có thể sử dụng phương thức `Thread.Join`. Phương thức này khiến tiểu trình đang gọi dừng lại (block) cho đến khi tiểu trình được tham chiếu kết thúc. Bạn có thể tùy chọn chỉ định một khoảng thời gian (giá trị `int` hay `TimeSpan`) mà sau khoảng thời gian này, `Join` sẽ hết hiệu lực và quá trình thực thi của tiểu trình đang gọi sẽ phục hồi lại. Nếu bạn chỉ định một giá trị time-out, `Join` trả về `true` nếu tiểu trình đã kết thúc, và `false` nếu `Join` đã hết hiệu lực.

Ví dụ dưới đây thực thi một tiểu trình thứ hai và rồi gọi `Join` để đợi tiểu trình thứ hai kết thúc. Vì tiểu trình thứ hai mất 5 giây để thực thi, nhưng phương thức `Join` chỉ định giá trị time-out là 3 giây, nên `Join` sẽ luôn hết hiệu lực và ví dụ này sẽ hiển thị một thông báo ra cửa sổ *Console*.

```
using System;

using System.Threading;

public class ThreadFinishExample {

    private static void DisplayMessage() {

        // Hiển thị một thông báo ra cửa sổ Console 5 lần.
        for (int count = 0; count < 5; count++) {

            Console.WriteLine("{0} : Second thread",
                               DateTime.Now.ToString("HH:mm:ss.ffff"));

            // Nghỉ 1 giây.
            Thread.Sleep(1000);
        }
    }
}
```

```

public static void Main() {

    // Tạo một thẻ hiện ủy nhiệm ThreadStart
    // tham chiếu đến DisplayMessage.
    ThreadStart method = new ThreadStart(DisplayMessage);

    // Tạo một đối tượng Thread và truyền thẻ hiện ủy nhiệm
    // ThreadStart cho phương thức khởi dụng của nó.
    Thread thread = new Thread(method);

    Console.WriteLine("{0} : Starting second thread.",
        DateTime.Now.ToString("HH:mm:ss.ffff"));

    // Khởi chạy tiểu trình thứ hai.
    thread.Start();

    // Dừng cho đến khi tiểu trình thứ hai kết thúc,
    // hoặc Join hết hiệu lực sau 3 giây.
    if (!thread.Join(3000)) {

        Console.WriteLine("{0} : Join timed out !!",
            DateTime.Now.ToString("HH:mm:ss.ffff"));
    }

    // Nhấn Enter để kết thúc.
    Console.WriteLine("Main method complete. Press Enter.");
    Console.ReadLine();
}
}

```

8. Đồng bộ hóa quá trình thực thi của nhiều tiểu trình

- ? **Bạn cần phối hợp các hoạt động của nhiều tiểu trình để bảo đảm sử dụng hiệu quả các tài nguyên dùng chung, và bạn không làm sai lệch dữ liệu dùng chung khi một phép chuyển ngữ cảnh tiểu trình (*thread context switch*) xảy ra trong quá trình thay đổi dữ liệu.**
- ✂ **Sử dụng các lớp `Monitor`, `AutoResetEvent`, `ManualResetEvent`, và `Mutex` (thuộc không gian tên `System.Threading`).**

Thách thức lớn nhất trong việc viết một ứng dụng hỗ trợ đa-tiểu-trình là bảo đảm các tiểu trình làm việc trong sự hòa hợp. Việc này thường được gọi là “đồng bộ hóa tiểu trình” và bao gồm:

- Bảo đảm các tiểu trình truy xuất các đối tượng và dữ liệu dùng chung một cách phù hợp để không gây ra sai lạc.
- Bảo đảm các tiểu trình chỉ thực thi khi thật sự cần thiết và phải đảm bảo rằng chúng chỉ được thực thi với chi phí tối thiểu khi chúng rồi.

Cơ chế đồng bộ hóa thông dụng nhất là lớp `Monitor`. Lớp này cho phép một tiểu trình đơn thu lấy chốt (*lock*) trên một đối tượng bằng cách gọi phương thức tĩnh `Monitor.Enter`. Bằng cách thu lấy chốt trước khi truy xuất một tài nguyên hay dữ liệu dùng chung, ta chắc chắn rằng chỉ có một tiểu trình có thể truy xuất tài nguyên đó cùng lúc. Một khi đã hoàn tất với tài nguyên, tiểu trình này sẽ giải phóng chốt để tiểu trình khác có thể truy xuất nó. Khối mã thực hiện công việc này thường được gọi là vùng hành căng (*critical section*).

Bạn có thể sử dụng bất kỳ đối tượng nào đóng vai trò làm chốt, và sử dụng từ khóa `this` để thu lấy chốt trên đối tượng hiện tại. Điểm chính là tất cả các tiểu trình khi truy xuất một tài nguyên dùng chung phải thu lấy cùng một chốt. Các tiểu trình khác khi thu lấy chốt trên cùng một đối tượng sẽ block (đi vào trạng thái `WaitSleepJoin`) và được thêm vào hàng sẵn sàng (*ready queue*) của chốt này cho đến khi tiểu trình chủ giải phóng nó bằng phương thức tĩnh `Monitor.Exit`. Khi tiểu trình chủ gọi `Exit`, một trong các tiểu trình từ hàng sẵn sàng sẽ thu lấy chốt. Nếu tiểu trình chủ không giải phóng chốt bằng `Exit`, tất cả các tiểu trình khác sẽ block vô hạn định. Vì vậy, cần đặt lời gọi `Exit` bên trong khối `finally` để bảo đảm nó được gọi cả khi ngoại lệ xảy ra.

Vì `Monitor` thường xuyên được sử dụng trong các ứng dụng hỗ trợ đa-tiểu-trình nên `C#` cung cấp hỗ trợ mức-ngôn-ngữ thông qua lệnh `lock`. Khối mã được gói trong lệnh `lock` tương đương với gọi `Monitor.Enter` khi đi vào khối mã này, và gọi `Monitor.Exit` khi đi ra khỏi mã này. Ngoài ra, trình biên dịch tự động đặt lời gọi `Monitor.Exit` trong khối `finally` để bảo đảm chốt được giải phóng khi một ngoại lệ bị ném.

Tiểu trình chủ (sở hữu chốt) có thể gọi `Monitor.Wait` để giải phóng chốt và đặt tiểu trình này vào hàng chờ (*wait queue*). Các tiểu trình trong hàng chờ cũng có trạng thái là `WaitSleepJoin` và sẽ tiếp tục block cho đến khi tiểu trình chủ gọi phương thức `Pulse` hay `PulseAll` của lớp `Monitor`. Phương thức `Pulse` di chuyển một trong các tiểu trình từ hàng chờ vào hàng sẵn sàng, còn phương thức `PulseAll` thì di chuyển tất cả các tiểu trình. Khi một tiểu trình đã được di chuyển từ hàng chờ vào hàng sẵn sàng, nó có thể thu lấy chốt trong lần giải phóng kế tiếp. Cần hiểu rằng các tiểu trình thuộc hàng chờ sẽ không thu được chốt, chúng sẽ đợi vô hạn định cho đến khi bạn gọi `Pulse` hay `PulseAll` để di chuyển chúng vào hàng sẵn sàng. Sử dụng `Wait` và `Pulse` là cách phổ biến khi thread-pool được sử dụng để xử lý các item từ một hàng đợi dùng chung.

Lớp `ThreadSyncExample` dưới đây trình bày cách sử dụng lớp `Monitor` và lệnh `lock`. Ví dụ này khởi chạy ba tiểu trình, mỗi tiểu trình (lần lượt) thu lấy chốt của một đối tượng có tên là `consoleGate`. Kế đó, mỗi tiểu trình gọi phương thức `Monitor.Wait`. Khi người dùng nhấn `Enter` lần đầu tiên, `Monitor.Pulse` sẽ được gọi để giải phóng một tiểu trình đang chờ. Lần thứ

hai người dùng nhấn *Enter*, `Monitor.PulseAll` sẽ được gọi để giải phóng tất cả các tiểu trình đang chờ còn lại.

```
using System;
using System.Threading;

public class ThreadSyncExample {

    private static object consoleGate = new Object();

    private static void DisplayMessage() {

        Console.WriteLine("{0} : Thread started, acquiring lock...",
            DateTime.Now.ToString("HH:mm:ss.ffff"));

        // Thu lấy chốt trên đối tượng consoleGate.
        try {

            Monitor.Enter(consoleGate);

            Console.WriteLine("{0} : {1}",
                DateTime.Now.ToString("HH:mm:ss.ffff"),
                "Acquired consoleGate lock, waiting...");

            // Đợi cho đến khi Pulse được gọi trên đối tượng consoleGate.
            Monitor.Wait(consoleGate);

            Console.WriteLine("{0} : Thread pulsed, terminating.",
                DateTime.Now.ToString("HH:mm:ss.ffff"));

        } finally {

            Monitor.Exit(consoleGate);
        }
    }

    public static void Main() {
```

```

// Thu lấy chốt trên đối tượng consoleGate.
lock (consoleGate) {

    // Tạo và khởi chạy ba tiểu trình mới
    // (chạy phương thức DisplayMesssage).
    for (int count = 0; count < 3; count++) {

        (new Thread(new ThreadStart(DisplayMessage))).Start();

    }

}

Thread.Sleep(1000);

// Đánh thức một tiểu trình đang chờ.
Console.WriteLine("{0} : {1}",
    DateTime.Now.ToString("HH:mm:ss.ffff"),
    "Press Enter to pulse one waiting thread.");

Console.ReadLine();

// Thu lấy chốt trên đối tượng consoleGate.
lock (consoleGate) {

    // Pulse một tiểu trình đang chờ.
    Monitor.Pulse(consoleGate);

}

// Đánh thức tất cả các tiểu trình đang chờ.
Console.WriteLine("{0} : {1}",
    DateTime.Now.ToString("HH:mm:ss.ffff"),
    "Press Enter to pulse all waiting threads.");

Console.ReadLine();

// Thu lấy chốt trên đối tượng consoleGate.

```

```

lock (consoleGate) {

    // Pulse tất cả các tiểu trình đang chờ.
    Monitor.PulseAll(consoleGate);

}

// Nhấn Enter để kết thúc.
Console.WriteLine("Main method complete. Press Enter.");
Console.ReadLine();

}
}

```

Các lớp thông dụng khác dùng để đồng bộ hóa tiểu trình là các lớp con của lớp `System.Threading.WaitHandle`, bao gồm `AutoResetEvent`, `ManualResetEvent`, và `Mutex`. Thể hiện của các lớp này có thể ở trạng thái *signaled* hay *unsignaled*. Các tiểu trình có thể sử dụng các phương thức của các lớp được liệt kê trong bảng 4.2 (được thừa kế từ lớp `WaitHandle`) để đi vào trạng thái `WaitSleepJoin` và đợi trạng thái của một hay nhiều đối tượng dẫn xuất từ `WaitHandle` biến thành *signaled*.

Bảng 4.2 Các phương thức của `WaitHandle` dùng để đồng bộ hóa quá trình thực thi của các tiểu trình

Phương thức	Mô tả
<code>WaitAny</code>	Tiểu trình gọi phương thức tĩnh này sẽ đi vào trạng thái <code>WaitSleepJoin</code> và đợi bất kỳ một trong các đối tượng <code>WaitHandle</code> thuộc một mảng <code>WaitHandle</code> biến thành <i>signaled</i> . Bạn cũng có thể chỉ định giá trị <code>time-out</code> .
<code>WaitAll</code>	Tiểu trình gọi phương thức tĩnh này sẽ đi vào trạng thái <code>WaitSleepJoin</code> và đợi tất cả các đối tượng <code>WaitHandle</code> trong một mảng <code>WaitHandle</code> biến thành <i>signaled</i> . Bạn cũng có thể chỉ định giá trị <code>time-out</code> . Phương thức <code>WaitAllExample</code> trong mục 4.2 đã trình bày cách sử dụng phương thức <code>WaitAll</code> .
<code>WaitOne</code>	Tiểu trình gọi phương thức này sẽ đi vào trạng thái <code>WaitSleepJoin</code> và đợi một đối tượng <code>WaitHandle</code> cụ thể biến thành <i>signaled</i> . Phương thức <code>WaitingExample</code> trong mục 4.2 đã trình bày cách sử dụng phương thức <code>WaitOne</code> .

Điểm khác biệt chính giữa các lớp `AutoResetEvent`, `ManualResetEvent`, và `Mutex` là cách thức chúng chuyển trạng thái từ *signaled* thành *unsignaled*, và tính khả kiến (*visibility*) của chúng. Lớp `AutoResetEvent` và `ManualResetEvent` là cục bộ đối với một tiến trình. Để ra hiệu một `AutoResetEvent`, bạn hãy gọi phương thức `Set` của nó, phương thức này chỉ giải phóng một tiểu trình đang đợi sự kiện. `AutoResetEvent` sẽ tự động trở về trạng thái *unsignaled*. Ví dụ trong mục 4.4 đã trình bày cách sử dụng lớp `AutoResetEvent`.

Lớp `ManualResetEvent` phải được chuyển đổi qua lại giữa *signaled* và *unsignaled* bằng phương thức `Set` và `Reset` của nó. Gọi `Set` trên một `ManualResetEvent` sẽ đặt trạng thái của nó là *signaled*, giải phóng tất cả các tiểu trình đang đợi sự kiện. Chỉ khi gọi `Reset` mới làm cho `ManualResetEvent` trở thành *unsignaled*.

Một `Mutex` là *signaled* khi nó không thuộc sở hữu của bất kỳ tiểu trình nào. Một tiểu trình giành quyền sở hữu `Mutex` lúc khởi động hoặc sử dụng một trong các phương thức được liệt kê trong bảng 4.2. Quyền sở hữu `Mutex` được giải phóng bằng cách gọi phương thức `Mutex.ReleaseMutex` (ra hiệu `Mutex` và cho phép một tiểu trình khác thu lấy quyền sở hữu này). Thuận lợi chính của `Mutex` là bạn có thể sử dụng chúng để đồng bộ hóa các tiểu trình qua các biên tiến trình. Mục 4.12 đã trình bày cách sử dụng `Mutex`.

Ngoài các chức năng vừa được mô tả, điểm khác biệt chính giữa các lớp `WaitHandle` và lớp `Monitor` là lớp `Monitor` được hiện thực hoàn toàn bằng mã lệnh được-quản-lý, trong khi các lớp `WaitHandle` cung cấp vỏ bọc cho các chức năng bên dưới của của hệ điều hành. Điều này dẫn đến hệ quả là:

- Sử dụng lớp `Monitor` đồng nghĩa với việc mã lệnh của bạn sẽ khả chuyển hơn vì không bị lệ thuộc vào khả năng của hệ điều hành bên dưới.
- Bạn có thể sử dụng các lớp dẫn xuất từ `WaitHandle` để đồng bộ hóa việc thực thi của các tiểu trình được-quản-lý và không-được-quản-lý, trong khi lớp `Monitor` chỉ có thể đồng bộ hóa các tiểu trình được-quản-lý.

9. Tạo một đối tượng tập hợp có tính chất an-toàn-về-tiểu-trình

?

Bạn muốn nhiều tiểu trình có thể đồng thời truy xuất nội dung của một tập hợp một cách an toàn.

✕

Sử dụng lệnh `lock` để đồng bộ hóa các tiểu trình truy xuất đến tập hợp, hoặc truy xuất tập hợp thông qua một vỏ bọc có tính chất an-toàn-về-tiểu-trình (*thread-safe*).

Theo mặc định, các lớp tập hợp chuẩn thuộc không gian tên `System.Collections` và `System.Collections.Specialized` sẽ hỗ trợ việc nhiều tiểu trình đồng thời đọc nội dung của tập hợp. Tuy nhiên, nếu một hay nhiều tiểu trình này sửa đổi tập hợp, nhất định bạn sẽ gặp rắc rối. Đó là vì hệ điều hành có thể làm đứt quãng các hành động của tiểu trình trong khi tập hợp chỉ mới được sửa đổi một phần. Điều này sẽ đưa tập hợp vào một trạng thái vô định, chắc chắn khiến cho một tiểu trình khác truy xuất tập hợp thất bại, trả về dữ liệu sai, hoặc làm hỏng tập hợp.

✍

Sử dụng “đồng bộ hóa tiểu trình” sẽ sinh ra một chi phí hiệu năng. Cứ để tập hợp là không-an-toàn-về-tiểu-trình (*non-thread-safe*) như mặc định sẽ cho hiệu năng tốt hơn đối với các trường hợp có nhiều tiểu trình không được dùng đến.

Tất cả các tập hợp thông dụng nhất đều hiện thực một phương thức tĩnh có tên là `Synchronized`; bao gồm các lớp: `ArrayList`, `Hashtable`, `Queue`, `SortedList`, và `Stack` (thuộc không gian tên `System.Collections`). Phương thức `Synchronized` nhận một đối tượng tập hợp

(với kiểu phù hợp) làm đối số và trả về một đối tượng cung cấp một vỏ bọc được-đồng-bộ-hóa (*synchronized wrapper*) bao lấy đối tượng tập hợp đã được chỉ định. Đối tượng vỏ bọc này có cùng kiểu với tập hợp gốc, nhưng tất cả các phương thức và thuộc tính dùng để đọc và ghi tập hợp bảo đảm rằng chỉ một tiểu trình có khả năng truy xuất nội dung của tập hợp cùng lúc. Đoạn mã dưới đây trình bày cách tạo một `Hashtable` có tính chất an-toàn-về-tiểu-trình (bạn có thể kiểm tra một tập hợp có phải là an-toàn-về-tiểu-trình hay không bằng thuộc tính `IsSynchronized`).

```
// Tạo một Hashtable chuẩn.
Hashtable hUnsync = new Hashtable();

// Tạo một vỏ bọc được-đồng-bộ-hóa.
Hashtable hSync = Hashtable.Synchronized(hUnsync);
```

Các lớp tập hợp như `HybridDictionary`, `ListDictionary`, và `StringCollection` (thuộc không gian tên `System.Collections.Specialized`) không hiện thực phương thức `Synchronized`. Để cung cấp khả năng truy xuất an-toàn-về-tiểu-trình đến thể hiện của các lớp này, bạn phải hiện thực quá trình đồng bộ hóa (sử dụng đối tượng được trả về từ thuộc tính `SyncRoot`) như được trình bày trong đoạn mã dưới đây:

```
// Tạo một NameValueCollection.
NameValueCollection nvCollection = new NameValueCollection();

// Thu lấy chốt trên NameValueCollection trước khi thực hiện sửa đổi.
lock (((ICollection)nvCollection).SyncRoot) {
    // Sửa đổi NameValueCollection...
}
```

Chú ý rằng lớp `NameValueCollection` dẫn xuất từ lớp `NameObjectCollectionBase`, lớp cơ sở này sử dụng cơ chế hiện thực giao diện tường minh để hiện thực thuộc tính `ICollection.SyncRoot`. Như đã được trình bày, bạn phải ép `NameValueCollection` về `ICollection` trước khi truy xuất thuộc tính `SyncRoot`. Việc ép kiểu là không cần thiết đối với các lớp tập hợp chuyên biệt như `HybridDictionary`, `ListDictionary`, và `StringCollection` (các lớp này không sử dụng cơ chế hiện thực giao diện tường minh để hiện thực `SyncRoot`).

Nếu cần sử dụng rộng khắp lớp tập hợp đã được đồng bộ hóa, bạn có thể đơn giản hóa mã lệnh bằng cách tạo một lớp mới dẫn xuất từ lớp tập hợp cần sử dụng. Kế tiếp, chép đè các thành viên của lớp cơ sở cung cấp khả năng truy xuất nội dung của tập hợp và thực hiện đồng bộ hóa trước khi gọi thành viên lớp cơ sở tương đương. Bạn có thể sử dụng lệnh `lock` một cách bình thường để đồng bộ hóa đối tượng được trả về bởi thuộc tính `SyncRoot` của lớp cơ sở như đã được thảo luận ở trên. Tuy nhiên, bằng cách tạo lớp dẫn xuất, bạn có thể hiện thực các kỹ thuật đồng bộ hóa cao cấp hơn, chẳng hạn sử dụng `System.Threading.ReaderWriterLock` để cho phép nhiều tiểu trình đọc nhưng chỉ một tiểu trình ghi.

10.

Khởi chạy một tiến trình mới

?

Bạn cần thực thi một ứng dụng trong một tiến trình mới.



Sử dụng đối tượng `System.Diagnostics.ProcessStartInfo` để chỉ định các chi tiết cho ứng dụng cần chạy. Sau đó, tạo đối tượng `System.Diagnostics.Process` để mô tả tiến trình mới, gán đối tượng `ProcessStartInfo` cho thuộc tính `StartInfo` của đối tượng `Process`, và rồi khởi chạy ứng dụng bằng cách gọi `Process.Start`.

Lớp `Process` cung cấp một dạng biểu diễn được-quản-lý cho một tiến trình của hệ điều hành và cung cấp một cơ chế đơn giản mà thông qua đó, bạn có thể thực thi cả ứng dụng được-quản-lý lẫn không-được-quản-lý. Lớp `Process` hiện thực bốn phiên bản nạp chồng cho phương thức `Start` (bạn có thể sử dụng phương thức này để khởi chạy một tiến trình mới). Hai trong số này là các phương thức tĩnh, cho phép bạn chỉ định tên và các đối số cho tiến trình mới. Ví dụ, hai lệnh dưới đây đều thực thi *Notepad* trong một tiến trình mới:

```
// Thực thi notepad.exe, không có đối số.
Process.Start("notepad.exe");

// Thực thi notepad.exe, tên file cần mở là đối số.
Process.Start("notepad.exe", "SomeFile.txt");
```

Hai dạng khác của phương thức `Start` yêu cầu bạn tạo đối tượng `ProcessStartInfo` được cấu hình với các chi tiết của tiến trình cần chạy; việc sử dụng đối tượng `ProcessStartInfo` cung cấp một cơ chế điều khiển tốt hơn trên các hành vi và cấu hình của tiến trình mới. Bảng 4.3 tóm tắt một vài thuộc tính thông dụng của lớp `ProcessStartInfo`.

Bảng 4.3 Các thuộc tính của lớp `ProcessStartInfo`

Thuộc tính	Mô tả
<code>Arguments</code>	Các đối số dùng để truyền cho tiến trình mới.
<code>ErrorDialog</code>	Nếu <code>Process.Start</code> không thể khởi chạy tiến trình đã được chỉ định, nó sẽ ném ngoại lệ <code>System.ComponentModel.Win32Exception</code> . Nếu <code>ErrorDialog</code> là <code>true</code> , <code>Start</code> sẽ hiển thị một thông báo lỗi trước khi ném ngoại lệ.
<code>FileName</code>	Tên của ứng dụng. Bạn cũng có thể chỉ định bất kỳ kiểu file nào mà bạn đã cấu hình ứng dụng kết giao với nó. Ví dụ, nếu bạn chỉ định một file với phần mở rộng là <code>.doc</code> hay <code>.xls</code> , <i>Microsoft Word</i> hay <i>Microsoft Excel</i> sẽ chạy.
<code>WindowStyle</code>	Một thành viên thuộc kiểu liệt kê <code>System.Diagnostics.ProcessWindowStyle</code> , điều khiển cách thức hiển thị của cửa sổ. Các giá trị hợp lệ bao gồm: <code>Hidden</code> , <code>Maximized</code> , <code>Minimized</code> , và <code>Normal</code> .
<code>WorkingDirectory</code>	Tên đầy đủ của thư mục làm việc.

Khi đã hoàn tất với một đối tượng `Process`, bạn nên hủy nó để giải phóng các tài nguyên hệ thống—gọi `Close`, `Dispose`, hoặc tạo đối tượng `Process` bên trong tầm vực của lệnh `using`. Việc hủy một đối tượng `Process` không ảnh hưởng lên tiến trình hệ thống nằm dưới, tiến trình này vẫn sẽ tiếp tục chạy.

Ví dụ dưới đây sử dụng `Process` để thực thi *Notepad* trong một cửa sổ ở trạng thái phóng to và mở một file có tên là `C:\Temp\file.txt`. Sau khi tạo, ví dụ này sẽ gọi phương thức `Process.WaitForExit` để dừng tiểu trình đang chạy cho đến khi tiến trình kết thúc hoặc giá trị time-out (được chỉ định trong phương thức này) hết hiệu lực.

```
using System;
using System.Diagnostics;

public class StartProcessExample {

    public static void Main () {

        // Tạo một đối tượng ProcessStartInfo và cấu hình cho nó
        // với các thông tin cần thiết để chạy tiến trình mới.
        ProcessStartInfo startInfo = new ProcessStartInfo();

        startInfo.FileName = "notepad.exe";
        startInfo.Arguments = "file.txt";
        startInfo.WorkingDirectory = @"C:\Temp";
        startInfo.WindowStyle = ProcessWindowStyle.Maximized;
        startInfo.ErrorDialog = true;

        // Tạo một đối tượng Process mới.
        using (Process process = new Process()) {

            // Gán ProcessStartInfo vào Process.
            process.StartInfo = startInfo;

            try {

                // Khởi chạy tiến trình mới.
                process.Start();

                // Đợi tiến trình mới kết thúc trước khi thoát.
                Console.WriteLine("Waiting 30 seconds for process to" +
                    " finish.");
                process.WaitForExit(30000);

            } catch (Exception ex) {
```

```

        Console.WriteLine("Could not start process.");
        Console.WriteLine(ex);
    }

}

// Nhấn Enter để kết thúc.
Console.WriteLine("Main method complete. Press Enter.");
Console.ReadLine();
}
}

```

11.

Kết thúc một tiến trình

?

Bạn muốn kết thúc một tiến trình (một ứng dụng hay một dịch vụ).

✕

Thu lấy đối tượng `Process` mô tả tiến trình hệ điều hành cần kết thúc. Đối với các ứng dụng dựa-trên-*Windows*, hãy gọi phương thức `Process.CloseMainWindow` để gửi một thông điệp đến cửa sổ chính của ứng dụng. Đối với các ứng dụng dựa-trên-*Windows* bỏ qua `CloseMainWindow`, hay đối với các ứng dụng không-dựa-trên-*Windows*, gọi phương thức `Process.Kill`.

Nếu khởi chạy một tiến trình mới từ mã lệnh được-quản-lý bằng lớp `Process` (đã được thảo luận trong mục 4.10), bạn có thể kết thúc tiến trình mới bằng đối tượng `Process` mô tả tiến trình này. Bạn cũng có thể thu lấy các đối tượng `Process` chỉ đến các tiến trình khác hiện đang chạy bằng các phương thức tĩnh của lớp `Process` (được tóm tắt trong bảng 4.4).

Bảng 4.4 Các phương thức dùng để thu lấy các tham chiếu `Process`

Phương thức	Mô tả
<code>GetCurrentProcess</code>	Trả về đối tượng <code>Process</code> mô tả tiến trình hiện đang tích cực.
<code>GetProcessById</code>	Trả về đối tượng <code>Process</code> mô tả tiến trình với <i>ID</i> được chỉ định.
<code>GetProcesses</code>	Trả về mảng các đối tượng <code>Process</code> mô tả tất cả các tiến trình hiện đang tích cực.
<code>GetProcessesByName</code>	Trả về mảng các đối tượng <code>Process</code> mô tả tất cả các tiến trình hiện đang tích cực với tên thân thiện được chỉ định. Tên thân thiện là tên của file thực thi không tính phần mở rộng và đường dẫn; ví dụ, <i>notepad</i> hay <i>calc</i> .

Một khi đã có đối tượng `Process` mô tả tiến trình cần kết thúc, bạn cần gọi phương thức `CloseMainWindow` hay phương thức `Kill`. Phương thức `CloseMainWindow` gửi một thông điệp đến cửa sổ chính của ứng dụng dựa-trên-*Windows*. Phương thức này có cùng tác dụng như thể

người dùng đóng cửa sổ chính bằng trình đơn hệ thống, và nó cho cơ hội ứng dụng thực hiện việc tắt một cách bình thường. `CloseMainWindow` sẽ không kết thúc các ứng dụng không có cửa sổ chính hoặc các ứng dụng có cửa sổ chính bị vô hiệu (có thể vì một hộp thoại hiện đang được mở). Với những tình huống như thế, `CloseMainWindow` sẽ trả về `false`.

`CloseMainWindow` trả về `true` nếu thông điệp được gửi thành công, nhưng không bảo đảm tiến trình thật sự kết thúc. Ví dụ, các ứng dụng dùng để soạn thảo dữ liệu thường sẽ cho cơ hội người dùng lưu lại các dữ liệu chưa được lưu nếu nhận được thông điệp này. Người dùng thường có cơ hội hủy bỏ việc đóng cửa sổ với những tình huống như thế. Điều này nghĩa là `CloseMainWindow` sẽ trả về `true`, nhưng ứng dụng vẫn cứ chạy khi người dùng hủy bỏ. Bạn có thể sử dụng phương thức `Process.WaitForExit` để báo hiệu việc kết thúc tiến trình và thuộc tính `Process.HasExited` để kiểm tra tiến trình đã kết thúc hay chưa. Và bạn cũng có thể sử dụng phương thức `Kill`.

Phương thức `Kill` kết thúc một tiến trình ngay lập tức; người dùng không có cơ hội dừng việc kết thúc, và tất cả các dữ liệu chưa được lưu sẽ bị mất. `Kill` là tùy chọn duy nhất để kết thúc các ứng dụng dựa-trên-*Windows* không đáp lại `CloseMainWindow` và để kết thúc các ứng dụng không-dựa-trên-*Windows*.

Ví dụ dưới đây khởi chạy một thể hiện mới của *Notepad*, đợi 5 giây, sau đó kết thúc tiến trình *Notepad*. Trước tiên, ví dụ này kết thúc tiến trình bằng `CloseMainWindow`. Nếu `CloseMainWindow` trả về `false`, hoặc tiến trình *Notepad* vẫn cứ chạy sau khi `CloseMainWindow` được gọi, ví dụ này sẽ gọi `Kill` và buộc tiến trình *Notepad* kết thúc; bạn có thể buộc `CloseMainWindow` trả về `false` bằng cách bỏ mặc hộp thoại *File Open* mở.

```
using System;
using System.Threading;
using System.Diagnostics;

public class TerminateProcessExample {
    public static void Main () {
        // Tạo một Process mới và chạy notepad.exe.
        using (Process process = Process.Start("notepad.exe")) {

            // Đợi 5 giây và kết thúc tiến trình Notepad.
            Console.WriteLine("Waiting 5 seconds before terminating" +
                " notepad.exe.");
            Thread.Sleep(5000);

            // Kết thúc tiến trình Notepad.
            Console.WriteLine("Terminating Notepad with " +
                "CloseMainWindow.");

            // Gửi một thông điệp đến cửa sổ chính.
```

```

        if (!process.CloseMainWindow()) {

            // Không gửi được thông điệp. Kết thúc Notepad bằng Kill.
            Console.WriteLine("CloseMainWindow returned false - " +
                " terminating Notepad with Kill.");
            process.Kill();

        } else {

            // Thông điệp được gửi thành công; đợi 2 giây
            // để chúng thực việc kết thúc trước khi viện đến Kill.
            if (!process.WaitForExit(2000)) {

                Console.WriteLine("CloseMainWindow failed to" +
                    " terminate - terminating Notepad with Kill.");
                process.Kill();
            }
        }
    }

    // Nhân Enter để kết thúc.
    Console.WriteLine("Main method complete. Press Enter.");
    Console.ReadLine();
}
}

```

12.

Bảo đảm chỉ có thể chạy một thể hiện của ứng dụng tại một thời điểm

- ?** Bạn cần bảo đảm rằng, tại một thời điểm chỉ có thể chạy một thể hiện của ứng dụng.
- X** Tạo một đối tượng `System.Threading.Mutex` và bảo ứng dụng thu lấy quyền sở hữu đối tượng này lúc khởi động.

`Mutex` cung cấp một cơ chế để đồng bộ hóa quá trình thực thi của các tiểu trình vượt qua biên tiến trình và còn cung cấp một cơ chế tiện lợi để bảo rằng chỉ một thể hiện của ứng dụng đang chạy. Bằng cách cố thu lấy quyền sở hữu một đối tượng `Mutex` lúc khởi động và thoát nếu

không thể thu được `Mutex`, bạn có thể bảo đảm rằng chỉ một thể hiện của ứng dụng đang chạy. Ví dụ dưới đây sử dụng một `Mutex` có tên là `MutexExample` để bảo đảm chỉ một thể hiện của ví dụ có thể thực thi.

```
using System;
using System.Threading;

public class MutexExample {

    public static void Main() {

        // Giá trị luận lý cho biết ứng dụng này
        // có quyền sở hữu Mutex hay không.
        bool ownsMutex;

        // Tạo và lấy quyền sở hữu một Mutex có tên là MutexExample.
        using (Mutex mutex =
            new Mutex(true, "MutexExample", out ownsMutex)) {

            // Nếu ứng dụng sở hữu Mutex, nó có thể tiếp tục thực thi;
            // nếu không, ứng dụng sẽ thoát.
            if (ownsMutex) {

                Console.WriteLine("This application currently owns the" +
                    " mutex named MutexExample. Additional instances" +
                    " of this application will not run until you" +
                    " release the mutex by pressing Enter.");

                Console.ReadLine();

                // Giải phóng Mutex.
                mutex.ReleaseMutex();

            } else {

                Console.WriteLine("Another instance of this" +
                    " application already owns the mutex named" +
                    " MutexExample. This instance of the" +
                    " application will terminate.");
```

```
        }  
    }  
  
    // Nhấn Enter để kết thúc.  
    Console.WriteLine("Main method complete. Press Enter.");  
    Console.ReadLine();  
}  
}
```

5

XML

Một trong những khía cạnh đáng chú ý nhất của *Microsoft .NET Framework* là sự tích hợp sâu sắc với *XML*. Trong nhiều ứng dụng *.NET*, bạn sẽ không nhận thấy rằng mình đang sử dụng các kỹ thuật *XML*—chúng sẽ được sử dụng phía hậu trường khi bạn tuần tự hóa một *Microsoft ADO.NET DataSet*, gọi một dịch vụ *Web XML*, hoặc đọc các thiết lập ứng dụng trong một file cấu hình *Web.config*. Trong các trường hợp khác, bạn sẽ muốn làm việc trực tiếp với không gian tên *System.Xml* để thao tác dữ liệu *XML*. Các tác vụ *XML* thông thường không chỉ phân tích một file *XML* mà còn xác nhận tính hợp lệ của nó dựa trên một *XML Schema*, áp dụng phép biến đổi *XSL* để tạo một tài liệu hay trang *HTML* mới, và tìm kiếm một cách thông minh với *XPath*. Các mục trong chương này trình bày các vấn đề sau:

- Các kỹ thuật dùng để đọc, phân tích, và thao tác dữ liệu *XML* (mục 5.1, 5.2, 5.3, và 5.7).
- Duyệt một tài liệu *XML* để tìm các nút cụ thể theo tên (mục 5.4), theo không gian tên (mục 5.5), hay theo biểu thức *XPath* (mục 5.6).
- Xác nhận tính hợp lệ của một tài liệu *XML* dựa trên một *XML Schema* (mục 5.8).
- Tuần tự hóa một đối tượng thành *XML* (mục 5.9), tạo *XML Schema* cho một lớp (mục 5.10), và tạo mã nguồn cho lớp dựa trên một *XML Schema* (mục 5.11).
- Biến đổi một tài liệu *XML* thành một tài liệu khác bằng *XSLT stylesheet* (mục 5.12).

1. *Hiện thị cấu trúc của một tài liệu XML trong TreeView*

? Bạn cần hiện thị cấu trúc và nội dung của một tài liệu *XML* trong một ứng dụng dựa-trên-*Windows*.

X Nạp tài liệu *XML* bằng lớp *System.Xml.XmlDocument*. Sau đó, viết một phương thức để chuyển một *XmlNode* thành một *System.Windows.Forms.TreeNode*, rồi gọi nó một cách đệ quy để duyệt qua toàn bộ tài liệu.

.NET Framework cung cấp nhiều cách khác nhau để xử lý các tài liệu *XML*. Cách mà bạn sử dụng tùy thuộc vào tác vụ cần lập trình. Một trong số đó là lớp *XmlDocument*. Lớp này cung cấp một dạng biểu diễn trong-bộ-nhớ cho một tài liệu *XML*, tuân theo *W3C Document Object Model (DOM)*; cho phép bạn duyệt qua các nút theo bất kỳ hướng nào, chèn và loại bỏ nút, và thay đổi động cấu trúc lúc chạy. Bạn hãy vào [<http://www.w3c.org>] để biết thêm chi tiết về *DOM*.

Để sử dụng lớp *XmlDocument*, bạn chỉ việc tạo một thể hiện của lớp này rồi gọi phương thức *Load* cùng với một tên file, một *Stream*, một *TextReader*, hay một *XmlReader* (bạn cũng có thể cung cấp một *URL* chỉ đến một tài liệu *XML*). Thể hiện *XmlDocument* sẽ chứa tất cả các nút (dạng cây) có trong tài liệu nguồn. Điểm nhập (*entry point*) dùng để truy xuất các nút này là phần tử gốc, được cấp thông qua thuộc tính *XmlDocument.DocumentElement*. Đây là một đối tượng *XmlElement*, có thể chứa nhiều đối tượng *XmlNode* lồng bên trong, các đối tượng này có thể chứa nhiều đối tượng *XmlNode* nữa, và cứ tiếp tục như thế. Một *XmlNode* là phần cấu thành cơ bản của một file *XML*. Một nút *XML* có thể là một phần tử (*element*), một đặc tính (*attribute*), lời chú thích, hay text.

Khi làm việc với `XmlNode` hay một lớp dẫn xuất từ đó (như `XmlElement` hay `XmlAttribute`), bạn có thể sử dụng các thuộc tính cơ bản sau đây:

- `ChildNodes` là tập hợp các nút lồng bên trong ở mức đầu tiên.
- `Name` là tên của nút.
- `NodeType` là một thành viên thuộc kiểu liệt kê `System.Xml.XmlNodeType`, cho biết kiểu của nút (phần tử, đặc tính, text...).
- `Value` là nội dung của nút, nếu đó là nút text hay nút *CDATA*.
- `Attributes` là tập hợp các nút mô tả các đặc tính được áp dụng cho phần tử.
- `InnerText` là chuỗi chứa giá trị (text) của nút hiện hành và tất cả các nút lồng bên trong.
- `InnerXml` là chuỗi chứa thẻ đánh dấu XML cho tất cả các nút lồng bên trong.
- `OuterXml` là chuỗi chứa thẻ đánh dấu XML cho nút hiện hành và tất cả các nút lồng bên trong.

Ví dụ dưới đây duyệt qua tất cả các nút của một `XmlDocument` (bằng thuộc tính `ChildNodes` và một phương thức đệ quy) rồi hiển thị chúng trong một `TreeView`.

```
using System;
using System.Windows.Forms;
using System.Xml;

public class XmlTreeDisplay : System.Windows.Forms.Form{
    private System.Windows.Forms.Button cmdLoad;
    private System.Windows.Forms.Label lblFile;
    private System.Windows.Forms.TextBox txtXmlFile;
    private System.Windows.Forms.TreeView treeXml;

    // (Bỏ qua phần mã designer.)

    private void cmdLoad_Click(object sender, System.EventArgs e) {

        // Xóa cây.
        treeXml.Nodes.Clear();

        // Nạp tài liệu XML.
        XmlDocument doc = new XmlDocument();
        try {
            doc.Load(txtXmlFile.Text);
        }catch (Exception err) {
```



```

        MessageBox.Show(err.Message);
        return;
    }

    // Đổ dữ liệu vào TreeView.
    ConvertXmlNodeToTreeNode(doc, treeXml.Nodes);

    // Mở rộng tất cả các nút.
    treeXml.Nodes[0].ExpandAll();
}

private void ConvertXmlNodeToTreeNode(XmlNode xmlNode,
    TreeNodeCollection treeNodes) {

    // Thêm một TreeNode mô tả XmlNode này.
    TreeNode newTreeNode = treeNodes.Add(xmlNode.Name);

    // Tùy biến phần text cho TreeNode dựa vào
    // kiểu và nội dung của XmlNode.
    switch (xmlNode.NodeType) {

        case XmlNodeType.ProcessingInstruction:
        case XmlNodeType.XmlDeclaration:
            newTreeNode.Text = "<?" + xmlNode.Name + " " +
                xmlNode.Value + "?>";
            break;

        case XmlNodeType.Element:
            newTreeNode.Text = "<" + xmlNode.Name + ">";
            break;

        case XmlNodeType.Attribute:
            newTreeNode.Text = "ATTRIBUTE: " + xmlNode.Name;
            break;

        case XmlNodeType.Text:
        case XmlNodeType.CDATA:
            newTreeNode.Text = xmlNode.Value;
            break;

        case XmlNodeType.Comment:
            newTreeNode.Text = "<!--" + xmlNode.Value + "-->";
    }
}

```

```

        break;
    }

    // Gọi phương thức này một cách đệ quy cho mỗi đặc tính
    // (XmlAttribute là một lớp con của XmlNode).
    if (xmlNode.Attributes != null) {

        foreach (XmlAttribute attribute in xmlNode.Attributes) {
            ConvertXmlNodeToTreeNode(attribute, newTreeNode.Nodes);
        }
    }

    // Gọi phương thức này một cách đệ quy cho mỗi nút con.
    foreach (XmlNode childNode in xmlNode.ChildNodes) {
        ConvertXmlNodeToTreeNode(childNode, newTreeNode.Nodes);
    }
}
}

```

Xét file XML dưới đây (*ProductCatalog.xml*):

```

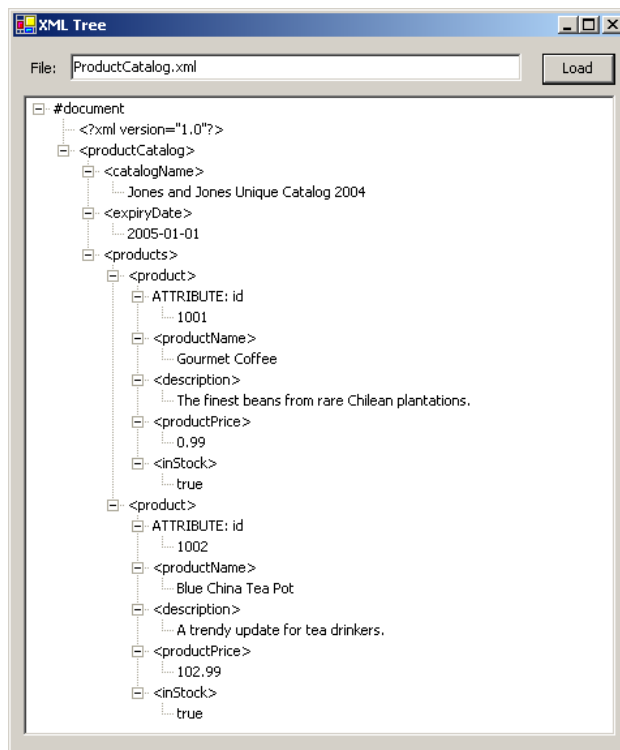
<?xml version="1.0" ?>
<productCatalog>
    <catalogName>Jones and Jones Unique Catalog 2004</catalogName>
    <expiryDate>2005-01-01</expiryDate>
    <products>
        <product id="1001">
            <productName>Gourmet Coffee</productName>
            <description>The finest beans from rare Chilean
            plantations.</description>
            <productPrice>0.99</productPrice>
            <inStock>true</inStock>
        </product>
        <product id="1002">
            <productName>Blue China Tea Pot</productName>
            <description>A trendy update for tea drinkers.</description>
            <productPrice>102.99</productPrice>
            <inStock>true</inStock>
        </product>
    </products>
</productCatalog>

```

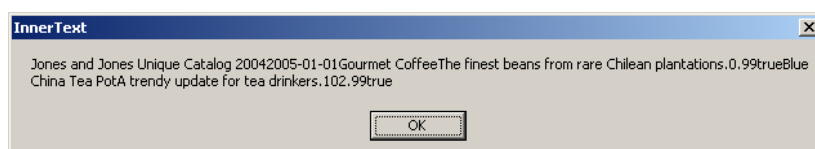
```

</product>
</products>
</productCatalog>

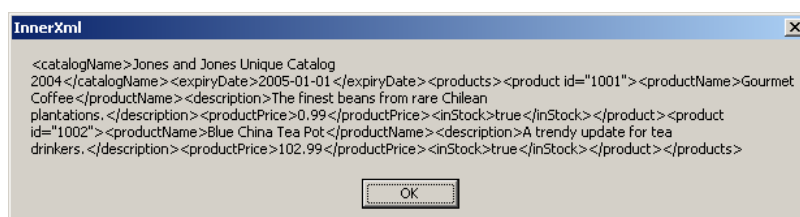
```



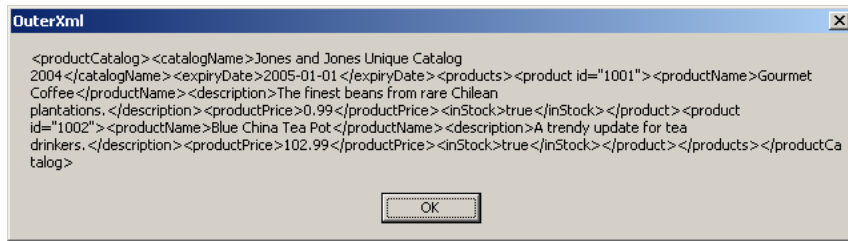
Hình 5.1 Cấu trúc của một tài liệu XML



Hình 5.2 InnerText của phần tử gốc



Hình 5.3 InnerXml của phần tử gốc



Hình 5.4 OuterXml của phần tử gốc

2.

Chèn thêm nút vào tài liệu XML



Bạn cần điều chỉnh một tài liệu XML bằng cách chèn vào dữ liệu mới, hoặc bạn muốn tạo một tài liệu hoàn toàn mới trong bộ nhớ.



Tạo nút bằng một phương thức của XmlDocument (như CreateElement, CreateAttribute, CreateNode...). Kế tiếp, chèn nó vào bằng một phương thức của XmlNode (như InsertAfter, InsertBefore, hay AppendChild).

Chèn một nút vào XmlDocument bao gồm hai bước: tạo nút rồi chèn nó vào vị trí thích hợp. Sau đó, bạn có thể gọi XmlDocument.Save để lưu lại những thay đổi.

Để tạo một nút, bạn sử dụng một trong các phương thức của XmlDocument bắt đầu bằng từ Create, tùy thuộc vào kiểu của nút. Việc này bảo đảm nút sẽ có cùng không gian tên như phần còn lại của tài liệu (bạn cũng có thể cung cấp một không gian tên làm đối số). Kế tiếp, bạn phải tìm một nút phù hợp và sử dụng một trong các phương thức chèn của nó để thêm nút mới vào.

Ví dụ dưới đây trình bày kỹ thuật này bằng cách tạo một tài liệu XML mới:

```
using System;
using System.Xml;
```

```
public class GenerateXml {
```

```
    private static void Main() {
```

```
        // Tạo một tài liệu mới rỗng.
```

```
        XmlDocument doc = new XmlDocument();
```

```
        XmlNode docNode = doc.CreateXmlDeclaration("1.0", "UTF-8", null);
```

```
        doc.AppendChild(docNode);
```

```
        // Tạo và chèn một phần tử mới.
```

```
XmlNode productsNode = doc.CreateElement("products");
doc.AppendChild(productsNode);

// Tạo một phần tử lồng bên trong (cùng với một đặc tính).
XmlNode productNode = doc.CreateElement("product");
XmlAttribute productAttribute = doc.CreateAttribute("id");
productAttribute.Value = "1001";
productNode.Attributes.Append(productAttribute);
productsNode.AppendChild(productNode);

// Tạo và thêm các phần tử con cho nút product này
// (cùng với dữ liệu text).
XmlNode nameNode = doc.CreateElement("productName");
nameNode.AppendChild(doc.CreateTextNode("Gourmet Coffee"));
productNode.AppendChild(nameNode);
XmlNode priceNode = doc.CreateElement("productPrice");
priceNode.AppendChild(doc.CreateTextNode("0.99"));
productNode.AppendChild(priceNode);

// Tạo và thêm một nút product khác.
productNode = doc.CreateElement("product");
productAttribute = doc.CreateAttribute("id");
productAttribute.Value = "1002";
productNode.Attributes.Append(productAttribute);
productsNode.AppendChild(productNode);
nameNode = doc.CreateElement("productName");
nameNode.AppendChild(doc.CreateTextNode("Blue China Tea Pot"));
productNode.AppendChild(nameNode);
priceNode = doc.CreateElement("productPrice");
priceNode.AppendChild(doc.CreateTextNode("102.99"));
productNode.AppendChild(priceNode);

// Lưu tài liệu.
doc.Save(Console.Out);
Console.ReadLine();
}
}
```

Tài liệu được tạo ra trông giống như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
  <product id="1001">
    <productName>Gourmet Coffee</productName>
    <productPrice>0.99</productPrice>
  </product>
  <product id="1002">
    <productName>Blue China Tea Pot</productName>
    <productPrice>102.99</productPrice>
  </product>
</products>
```

3. *Chèn thêm nút vào tài liệu XML một cách nhanh chóng*

- ? **Bạn cần chèn thêm nút vào một tài liệu XML mà không phải dùng đến mã lệnh dài dòng.**
- ✂ **Viết các phương thức trợ giúp (nhận vào tên thẻ và nội dung của nút) để chèn nút vào tài liệu XML. Cách khác, sử dụng phương thức `XmlDocument.CloneNode` để sao lại các nhánh của một `XmlDocument`.**

Chèn một nút vào `XmlDocument` cần nhiều mã lệnh. Có nhiều cách thu ngắn mã lệnh này. Một cách là tạo một lớp trợ giúp (*helper*) gồm các phương thức mức-cao để chèn nút vào tài liệu. Ví dụ, bạn có thể viết phương thức `AddElement` để tạo một phần tử mới, chèn nó vào, và thêm text (đây là ba thao tác cần thiết khi chèn phần tử).

Ví dụ dưới đây là một lớp trợ giúp như thế:

```
using System;
using System.Xml;

public class XmlHelper {

    public static XmlNode AddElement(string tagName,
        string textContent, XmlNode parent) {

        XmlNode node = parent.OwnerDocument.CreateElement(tagName);
        parent.AppendChild(node);
```

```

        if (textContent != null) {

            XmlNode content;
            content = parent.OwnerDocument.CreateTextNode(textContent);
            node.AppendChild(content);
        }
        return node;
    }

    public static XmlNode AddAttribute(string attributeName,
        string textContent, XmlNode parent) {

        XmlAttribute attribute;
        attribute = parent.OwnerDocument.CreateAttribute(attributeName);
        attribute.Value = textContent;
        parent.Attributes.Append(attribute);

        return attribute;
    }
}

```

Bây giờ bạn có thể viết mã lệnh để tạo một tài liệu *XML* (giống mục 5.2) với cú pháp đơn giản hơn như sau:

```

public class GenerateXml {

    private static void Main() {

        // Tạo tài liệu.
        XmlDocument doc = new XmlDocument();
        XmlNode docNode = doc.CreateXmlDeclaration("1.0", "UTF-8", null);
        doc.AppendChild(docNode);
        XmlNode products = doc.CreateElement("products");
        doc.AppendChild(products);

        // Thêm hai product.
        XmlNode product = XmlHelper.AddElement("product", null,
            products);
        XmlHelper.AddAttribute("id", "1001", product);
        XmlHelper.AddElement("productName", "Gourmet Coffee", product);
    }
}

```

```

        XmlHelper.AddElement("productPrice", "0.99", product);

        product = XmlHelper.AddElement("product", null, products);
        XmlHelper.AddAttribute("id", "1002", product);
        XmlHelper.AddElement("productName", "Blue China Tea Pot",
            product);
        XmlHelper.AddElement("productPrice", "102.99", product);

        // Lưu tài liệu.
        doc.Save(Console.Out);

        Console.ReadLine();
    }
}

```

Bạn cũng có thể lấy các phương thức trợ giúp (như `AddAttribute` và `AddElement`) làm các phương thức thể hiện trong một lớp tùy biến dẫn xuất từ `XmlDocument`.

Một cách khác để đơn giản hóa việc viết *XML* là sao lại các nút bằng phương thức `XmlNode.CloneNode`. Phương thức này nhận một đối số luận lý. Nếu giá trị này là `true`, `CloneNode` sẽ sao lại toàn bộ nhánh, với tất cả các nút lồng bên trong.

Ví dụ dưới đây tạo một nút `product` mới bằng cách sao lại nút đầu tiên:

```

// (Thêm nút product đầu tiên.)

// Tạo một product mới dựa vào product hiện có.
product = product.CloneNode(true);

// Điều chỉnh dữ liệu.
product.Attributes[0].Value = "1002";
product.ChildNodes[0].ChildNodes[0].Value = "Blue China Tea Pot";
product.ChildNodes[1].ChildNodes[0].Value = "102.99";

// Thêm phần tử mới.
products.AppendChild(product);

```

Chú ý trong trường hợp này, có một số giả định được áp đặt lên các nút hiện có (ví dụ, giả định con đầu tiên của nút luôn là `productName`, và con thứ hai luôn là `productPrice`). Nếu giả định này không bảo đảm đúng, bạn cần phải xét tên của nút.

4.

Tìm một nút khi biết tên của nó

- ?** Bạn cần thu lấy một nút cụ thể trong một `XmlDocument`, và bạn biết tên của nó nhưng không biết vị trí của nó.
- X** Sử dụng phương thức `XmlDocument.GetElementsByTagName`, phương thức này sẽ dò tìm toàn bộ tài liệu và trả về tập hợp `System.Xml.XmlNodeList` chứa các nút được so trùng.

Lớp `XmlDocument` cung cấp phương thức `GetElementsByTagName` dùng để tìm ra các nút có tên cho trước. Nó trả về kết quả là một tập hợp các đối tượng `XmlNode`.

Đoạn mã dưới đây trình bày cách sử dụng `GetElementsByTagName` để tính tổng giá các item trong một danh mục bằng cách thu lấy tất cả các phần tử có tên là "productPrice":

```
using System;
using System.Xml;

public class FindNodesByName {

    private static void Main() {

        // Nạp tài liệu.
        XmlDocument doc = new XmlDocument();
        doc.Load("ProductCatalog.xml");

        // Thu lấy tất cả price.
        XmlNodeList prices = doc.GetElementsByTagName("productPrice");

        decimal totalPrice = 0;
        foreach (XmlNode price in prices) {

            // Lấy phần text bên trong của mỗi phần tử được so trùng.
            totalPrice += Decimal.Parse(price.ChildNodes[0].Value);
        }

        Console.WriteLine("Total catalog value: " +
            totalPrice.ToString());
        Console.ReadLine();
    }
}
```

Bạn cũng có thể dò tìm một phần tài liệu *XML* bằng phương thức `XmlElement.GetElementsByTagName` (phương thức này sẽ dò tất cả các nút con để tìm ra nút

trùng khớp). Để sử dụng phương thức này, trước hết lấy một `XmlNode` tương ứng với một phần tử, kế đó ép đối tượng này thành một `XmlElement`. Ví dụ dưới đây trình bày cách tìm nút `price` bên dưới phần tử `product` đầu tiên:

```
// Thu lấy tham chiếu đến product đầu tiên.
XmlNode product = doc.GetElementsByTagName("products")[0];

// Tìm nút price bên dưới product này.
XmlNode price =
    ((XmlElement)product).GetElementsByTagName("productPrice")[0];
Console.WriteLine("Price is " + price.InnerText);
```

Nếu các phần tử của bạn có chứa đặc tính `ID`, bạn cũng có thể sử dụng một phương thức có tên là `GetElementById` để thu lấy phần tử có giá trị `ID` trùng khớp.

5. *Thu lấy các nút XML trong một không gian tên XML cụ thể*

?

Bạn cần thu lấy các nút trong một không gian tên cụ thể bằng một `XmlDocument`.

X

Sử dụng phiên bản nạp chồng của phương thức `XmlDocument.GetElementsByTagName` (yêu cầu một tên không gian tên làm đối số). Ngoài ra, áp dụng dấu hoa thị (*) vào đối số tên thẻ nếu bạn muốn so trùng tất cả các thẻ.

Nhiều tài liệu *XML* chứa các nút thuộc nhiều không gian tên khác nhau. Ví dụ, tài liệu *XML* mô tả một bài báo khoa học có thể sử dụng một kiểu đánh dấu riêng để biểu thị các phương trình toán học và các biểu đồ vector. Hoặc một tài liệu *XML* với các thông tin về đặt hàng có thể kết hợp các thông tin về khách hàng và đơn đặt hàng cùng với một hồ sơ vận chuyển. Tương tự, một tài liệu *XML* mô tả một giao dịch thương mại có thể bao gồm những phần thuộc cả hai công ty, và những phần này được viết theo ngôn ngữ đánh dấu riêng.

Một tác vụ thông thường trong lập trình *XML* là thu lấy các phần tử thuộc một không gian tên cụ thể. Bạn có thể thực hiện tác vụ này với phiên bản nạp chồng của phương thức `XmlDocument.GetElementsByTagName` (yêu cầu một tên không gian tên làm đối số). Bạn có thể sử dụng phương thức này để tìm các thẻ theo tên, hoặc tìm tất cả các thẻ trong không gian tên đã được chỉ định nếu bạn áp dụng dấu hoa thị vào đối số tên thẻ.

Ví dụ, tài liệu *XML* phức hợp dưới đây bao gồm các thông tin về đơn đặt hàng và khách hàng trong hai không gian tên khác nhau là `http://mycompany/OrderML` và `http://mycompany/ClientML`.

```
<?xml version="1.0" ?>
<ord:order xmlns:ord="http://mycompany/OrderML"
  xmlns:cli="http://mycompany/ClientML">

  <cli:client>
```

```

    <cli:firstName>Sally</cli:firstName>
    <cli:lastName>Sergeyeva</cli:lastName>
</cli:client>

```

```

<ord:orderItem itemNumber="3211"/>
<ord:orderItem itemNumber="1155"/>

```

```

</ord:order>

```

Và chương trình dưới đây sẽ chọn tất cả các thẻ trong không gian tên `http://mycompany/OrderML`:

```

using System;
using System.Xml;

public class SelectNodesByNamespace {

    private static void Main() {

        // Nạp tài liệu.
        XmlDocument doc = new XmlDocument();
        doc.Load("Order.xml");

        // Thu lấy tất cả các thẻ đặt hàng.
        XmlNodeList matches = doc.GetElementsByTagName("?",
            "http://mycompany/OrderML");

        // Hiển thị thông tin.
        Console.WriteLine("Element \tAttributes");
        Console.WriteLine("***** \t*****");

        foreach (XmlNode node in matches) {

            Console.Write(node.Name + "\t");
            foreach (XmlAttribute attribute in node.Attributes) {
                Console.Write(attribute.Value + " ");
            }
            Console.WriteLine();
        }
    }
}

```

```

        Console.ReadLine();
    }
}

```

Kết xuất của chương trình này như sau:

```

Element          Attributes
*****          *****

ord:order         http://mycompany/OrderML  http://mycompany/ClientML
ord:orderItem     3211
ord:orderItem     1155

```

6.

Tìm các phần tử với biểu thức XPath



Bạn cần duyệt một tài liệu XML để tìm các nút theo một tiêu chuẩn tìm kiếm cấp cao. Ví dụ, bạn có thể muốn duyệt một nhánh cụ thể của một tài liệu XML để tìm các nút có các đặc tính nào đó hoặc chứa một số lượng nút con lồng bên trong.



Thực thi một biểu thức XPath bằng phương thức `SelectNodes` hay `SelectSingleNode` của lớp `XmlDocument`.

Lớp `XmlNode` định nghĩa hai phương thức dùng để tìm kiếm dựa vào biểu thức *Xpath* là `SelectNodes` và `SelectSingleNode`. Hai phương thức này thao tác trên tất cả các nút con. Vì `XmlDocument` thừa kế từ `XmlNode` nên bạn có thể gọi `XmlDocument.SelectNodes` để dò tìm toàn bộ một tài liệu.

Xét tài liệu XML mô tả một đơn đặt hàng gồm hai item:

```

<?xml version="1.0"?>
<Order id="2004-01-30.195496">
  <Client id="ROS-930252034">
    <Name>Remarkable Office Supplies</Name>
  </Client>

  <Items>
    <Item id="1001">
      <Name>Electronic Protractor</Name>
      <Price>42.99</Price>
    </Item>
    <Item id="1002">
      <Name>Invisible Ink</Name>

```

```

        <Price>200.25</Price>
    </Item>
</Items>
</Order>

```

Cú pháp của *XPath* sử dụng ký hiệu giống như đường dẫn. Ví dụ, đường dẫn `/Order/Items/Item` cho biết phần tử `<Item>` lồng bên trong phần tử `<Items>`, và phần tử `<Items>` lồng bên trong phần tử gốc `<Order>`. Ví dụ dưới đây sử dụng một đường dẫn tuyệt đối để tìm tên của tất cả các item trong một đơn đặt hàng:

```

using System;
using System.Xml;

public class XPathSelectNodes {

    private static void Main() {

        // Nạp tài liệu.
        XmlDocument doc = new XmlDocument();
        doc.Load("orders.xml");

        // Thu lấy tên của tất cả các item.
        // Việc này không thể hoàn tất dễ dàng với phương thức
        // GetElementsByTagName(), vì các phần tử Name được sử dụng
        // bên trong các phần tử Item và các phần tử Client, và do đó
        // cả hai kiểu này đều sẽ được trả về.
        XmlNodeList nodes = doc.SelectNodes("/Order/Items/Item/Name");

        foreach (XmlNode node in nodes) {
            Console.WriteLine(node.InnerText);
        }

        Console.ReadLine();
    }
}

```

Kết xuất của chương trình này như sau:

```

Electronic Protractor
Invisible Ink

```

XPath cung cấp một cú pháp tìm kiếm mạnh. Do không thể giải thích tất cả các biến thể của nó chỉ trong một mục ngắn như thế này, nên bảng 5.1 chỉ trình bày các phần chính trong một

biểu thức *XPath* và các ví dụ mô tả cách làm việc của chúng với tài liệu *XML* ở trên. Để hiểu chi tiết hơn, bạn hãy tham khảo tài liệu *W3C XPath* tại [<http://www.w3.org/TR/xpath>].

Bảng 5.1 Cú pháp của biểu thức *XPath*

Biểu thức	Mô tả
/	Bắt đầu một đường dẫn tuyệt đối (chọn từ nút gốc). /Order/Items/Item chọn tất cả các phần tử <i>Item</i> là con của một phần tử <i>Items</i> , mà bản thân <i>Items</i> là con của phần tử gốc <i>Order</i> .
//	Bắt đầu một đường dẫn tương đối (chọn nút bất cứ đâu). //Item/Name chọn tất cả các phần tử <i>Name</i> là con của một phần tử <i>Item</i> , bất chấp chúng xuất hiện ở đâu trong tài liệu.
@	Chọn một đặc tính của một nút. /Order/@id chọn đặc tính có tên là <i>id</i> từ phần tử gốc <i>Order</i> .
*	Chọn bất cứ phần tử nào trong đường dẫn. /Order/* chọn nút <i>Items</i> và <i>Client</i> vì cả hai đều nằm trong phần tử gốc <i>Order</i> .
	Kết hợp nhiều đường dẫn. /Order/Items/Item/Name Order/Client/Name chọn các nút <i>Name</i> dùng để mô tả một <i>Client</i> và các nút <i>Name</i> dùng để mô tả một <i>Item</i> .
.	Cho biết nút (mặc định) hiện hành. Nếu nút hiện hành là một <i>Order</i> , biểu thức ./Items chỉ đến các item liên quan với đơn đặt hàng đó.
..	Cho biết nút cha. //Name/.. chọn phần tử là cha của một <i>Name</i> , gồm các phần tử <i>Client</i> và <i>Item</i> .
[]	Định nghĩa tiêu chuẩn chọn lựa (<i>selection criteria</i>), có thể kiểm tra giá trị của một nút bên trong hay của một đặc tính. /Order[@id="2004-01-30.195496"] chọn các phần tử <i>Order</i> với giá trị đặc tính cho trước. /Order/Items/Item[Price > 50] chọn các sản phẩm có giá trên \$50. /Order/Items/Item[Price > 50 and Name="Laser Printer"] chọn các sản phẩm trùng khớp với cả hai tiêu chuẩn.
starts-with	Hàm này thu lấy các phần tử dựa vào phần text khởi đầu của phần tử nằm bên trong. /Order/Items/Item[starts-with(Name,"C")] tìm tất cả các phần tử <i>Item</i> có phần tử <i>Name</i> bắt đầu bằng mẫu tự <i>C</i> .

position	Hàm này thu lấy các phần tử dựa vào vị trí. /Order/Items/Item[position()=2] chọn phần tử Item thứ hai.
count	Hàm này đếm số phần tử. Bạn cần chỉ định tên của phần tử con cần đếm hoặc dấu hoa thị (*) cho tất cả các phần tử con. /Order/Items/Item[count(Price)=1] thu lấy các phần tử Item có đúng một phần tử Price lồng bên trong.

 **Biểu thức XPath và tất cả tên phần tử và đặc tính mà bạn sử dụng trong đó luôn có phân biệt chữ hoa-thường, vì bản thân XML có phân biệt chữ hoa-thường.**

7. Đọc và ghi XML mà không phải nạp toàn bộ tài liệu vào bộ nhớ

? Bạn cần đọc XML từ một stream, hoặc ghi nó ra một stream. Tuy nhiên, bạn muốn xử lý từng nút một, không phải nạp toàn bộ vào bộ nhớ với một XmlDocument.

✕ Để ghi XML, hãy tạo một XmlTextWriter bọc lấy một stream và sử dụng các phương thức Write (như WriteStartElement và WriteEndElement). Để đọc XML, hãy tạo một XmlTextReader bọc lấy một stream và gọi phương thức Read để dịch chuyển từ nút này sang nút khác.

Lớp XmlTextWriter và XmlTextReader đọc/ghi XML trực tiếp từ stream từng nút một. Các lớp này không cung cấp các tính năng dùng để duyệt và thao tác tài liệu XML như XmlDocument, nhưng hiệu năng cao hơn và vết bộ nhớ nhỏ hơn, đặc biệt khi bạn làm việc với các tài liệu XML cực kỳ lớn.

Để ghi XML ra bất kỳ stream nào, bạn có thể sử dụng XmlTextWriter. Lớp này cung cấp các phương thức Write dùng để ghi từng nút một, bao gồm:

- WriteStartDocument—ghi phần khởi đầu của tài liệu; và WriteEndDocument, đóng bất kỳ phần tử nào đang mở ở cuối tài liệu.
- WriteStartElement—ghi một thẻ mở (*opening tag*) cho phần tử bạn chỉ định. Kế đó, bạn có thể thêm nhiều phần tử lồng bên trong phần tử này, hoặc bạn có thể gọi WriteEndElement để ghi thẻ đóng (*closing tag*).
- WriteElementString—ghi một phần tử, cùng với một thẻ mở, một thẻ đóng, và nội dung text.
- WriteAttributeString—ghi một đặc tính cho phần tử đang mở gần nhất, cùng với tên và giá trị.

Sử dụng các phương thức này thường cần ít mã lệnh hơn là tạo một XmlDocument bằng tay, như được trình bày trong mục 5.2 và 5.3.

Để đọc XML, bạn sử dụng phương thức Read của XmlTextReader. Phương thức này tiến reader đến nút kế tiếp, và trả về true. Nếu không còn nút nào nữa, nó sẽ trả về false. Bạn có thể thu lấy thông tin về nút hiện tại thông qua các thuộc tính của XmlTextReader (bao gồm Name, Value, và NodeType).

Để nhận biết một phần tử có các đặc tính hay không, bạn phải kiểm tra thuộc tính `HasAttributes` và rồi sử dụng phương thức `GetAttribute` để thu lấy các đặc tính theo tên hay theo chỉ số. Lớp `XmlTextReader` chỉ có thể truy xuất một nút tại một thời điểm, và nó không thể dịch chuyển ngược hay nhảy sang một nút bất kỳ. Do đó, tính linh hoạt của nó kém hơn lớp `XmlDocument`.

Ứng dụng dưới đây ghi và đọc một tài liệu *XML* bằng lớp `XmlTextWriter` và `XmlTextReader`. Tài liệu này giống với tài liệu đã được tạo trong mục 5.2 và 5.3 bằng lớp `XmlDocument`.

```
using System;
using System.Xml;
using System.IO;
using System.Text;

public class ReadWriteXml {

    private static void Main() {

        // Tạo file và writer.
        FileStream fs = new FileStream("products.xml", FileMode.Create);
        XmlTextWriter w = new XmlTextWriter(fs, Encoding.UTF8);

        // Khởi động tài liệu.
        w.WriteStartDocument();
        w.WriteStartElement("products");

        // Ghi một product.
        w.WriteStartElement("product");
        w.WriteAttributeString("id", "1001");
        w.WriteElementString("productName", "Gourmet Coffee");
        w.WriteElementString("productPrice", "0.99");
        w.WriteEndElement();

        // Ghi một product khác.
        w.WriteStartElement("product");
        w.WriteAttributeString("id", "1002");
        w.WriteElementString("productName", "Blue China Tea Pot");
        w.WriteElementString("productPrice", "102.99");
```



```
w.WriteEndElement();

// Kết thúc tài liệu.
w.WriteEndElement();
w.WriteEndDocument();
w.Flush();
fs.Close();

Console.WriteLine("Document created. " +
    "Press Enter to read the document.");
Console.ReadLine();

fs = new FileStream("products.xml", FileMode.Open);
XmlTextReader r = new XmlTextReader(fs);

// Đọc tất cả các nút.
while (r.Read()) {

    if (r.NodeType == XmlNodeType.Element) {

        Console.WriteLine();
        Console.WriteLine("<" + r.Name + ">");

        if (r.HasAttributes) {

            for (int i = 0; i < r.AttributeCount; i++) {
                Console.WriteLine("\tATTRIBUTE: " +
                    r.GetAttribute(i));
            }
        }

        else if (r.NodeType == XmlNodeType.Text) {
            Console.WriteLine("\tVALUE: " + r.Value);
        }

    }

    Console.ReadLine();
}
```

}

8. *Xác nhận tính hợp lệ của một tài liệu XML dựa trên một Schema*

?

Bạn cần xác nhận tính hợp lệ của một tài liệu XML bằng cách bảo đảm nó tuân theo một XML Schema.

✕

Sử dụng lớp `System.Xml.XmlValidatingReader`. Tạo một thể hiện của lớp này, nạp Schema vào tập hợp `XmlValidatingReader.Schemas`, dịch chuyển qua từng nút một bằng cách gọi `XmlValidatingReader.Read`, và bắt bất cứ ngoại lệ nào. Để tìm tất cả các lỗi trong một tài liệu mà không phải bắt ngoại lệ, hãy thụ lý sự kiện `ValidationEventHandler`.

Một XML Schema (giản đồ XML) định nghĩa các quy tắc mà một kiểu tài liệu XML cho trước phải tuân theo. Các quy tắc này định nghĩa:

- Các phần tử và đặc tính có thể xuất hiện trong tài liệu.
- Các kiểu dữ liệu cho phần tử và đặc tính.
- Cấu trúc của tài liệu, bao gồm các phần tử nào là con của các phần tử khác.
- Thứ tự và số lượng các phần tử con xuất hiện trong tài liệu.
- Các phần tử nào là rỗng, có thể chứa text, hay đòi hỏi các giá trị cố định.

Bàn sâu về các tài liệu XML Schema vượt quá phạm vi của chương này, nhưng bạn có thể tìm hiểu nó thông qua một ví dụ đơn giản. Mục này sẽ sử dụng tài liệu XML mô tả danh mục sản phẩm đã được trình bày trong mục 5.1.

Ở mức cơ bản nhất, XML Schema Definition (XSD) được sử dụng để định nghĩa các phần tử có thể xuất hiện trong tài liệu XML. Bản thân tài liệu XSD được viết theo dạng XML, và bạn sử dụng một phần tử đã được định nghĩa trước (có tên là `<element>`) để chỉ định các phần tử sẽ cần thiết trong tài liệu đích. Đặc tính `type` cho biết kiểu dữ liệu. Ví dụ dưới đây là tên sản phẩm:

```
<xsd:element name="productName" type="xsd:string" />
```

Và ví dụ dưới đây là giá sản phẩm:

```
<xsd:element name="productPrice" type="xsd:decimal" />
```

Bạn có thể tìm hiểu các kiểu dữ liệu Schema tại [<http://www.w3.org/TR/xmlschema-2>]. Chúng ảnh xạ đến các kiểu dữ liệu .NET và bao gồm `string`, `int`, `long`, `decimal`, `float`, `dateTime`, `boolean`, `base64Binary`...

Cả `productName` và `productPrice` đều là các kiểu đơn giản vì chúng chỉ chứa dữ liệu dạng ký tự. Các phần tử có chứa các phần tử lồng bên trong được gọi là các kiểu phức tạp. Bạn có thể lồng chúng vào nhau bằng thẻ `<sequence>` (nếu thứ tự là quan trọng) hay thẻ `<all>` (nếu thứ tự là không quan trọng). Dưới đây là cách lắp phần tử `<product>` vào danh mục sản phẩm. Chú ý

rằng, các đặc tính luôn được khai báo sau các phần tử, và chúng không được nhóm với thẻ `<sequence>` hay `<all>` vì thứ tự không quan trọng.

```
<xsd:complexType name="product">
  <xsd:sequence>
    <xsd:element name="productName" type="xsd:string"/>
    <xsd:element name="productPrice" type="xsd:decimal"/>
    <xsd:element name="inStock" type="xsd:boolean"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:integer"/>
</xsd:complexType>
```

Theo mặc định, một phần tử có thể xuất hiện đúng một lần trong một tài liệu. Nhưng bạn có thể cấu hình điều này bằng cách chỉ định các đặc tính `maxOccurs` và `minOccurs`. Ví dụ dưới đây không giới hạn số lượng sản phẩm trong danh mục:

```
<xsd:element name="product" type="product" maxOccurs="unbounded" />
```

Dưới đây là *Schema* cho danh mục sản phẩm:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Định nghĩa product (kiểu phức). -->
  <xsd:complexType name="product">
    <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="productPrice" type="xsd:decimal"/>
      <xsd:element name="inStock" type="xsd:boolean"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:integer"/>
  </xsd:complexType>

  <!-- Đây là cấu trúc mà tài liệu phải tuân theo.
  Bắt đầu với phần tử productCatalog. -->
  <xsd:element name="productCatalog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="catalogName" type="xsd:string"/>
        <xsd:element name="expiryDate" type="xsd:date"/>

        <xsd:element name="products">
```

```

        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="product" type="product"
                    maxOccurs="unbounded" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

</xsd:schema>

```

Lớp `XmlValidatingReader` thực thi tất cả các quy tắc *Schema* này—bảo đảm tài liệu là hợp lệ—và nó cũng kiểm tra tài liệu *XML* đã được chỉnh dạng hay chưa (nghĩa là không có các ký tự bất hợp lệ, tất cả các thẻ mở đều có một thẻ đóng tương ứng, v.v...). Để kiểm tra một tài liệu, hãy dùng phương thức `XmlValidatingReader.Read` để duyệt qua từng nút một. Nếu tìm thấy lỗi, `XmlValidatingReader` dựng lên sự kiện `ValidationEventHandler` với các thông tin về lỗi. Nếu muốn, bạn có thể thụ lý sự kiện này và tiếp tục kiểm tra tài liệu để tìm thêm lỗi. Nếu bạn không thụ lý sự kiện này, ngoại lệ `XmlException` sẽ được dựng lên khi bắt gặp lỗi đầu tiên và quá trình kiểm tra sẽ bị bỏ dở. Để kiểm tra một tài liệu đã được chỉnh dạng hay chưa, bạn có thể sử dụng `XmlValidatingReader` mà không cần đến *Schema*.

Ví dụ kế tiếp trình bày một lớp tiện ích dùng để hiển thị tất cả các lỗi trong một tài liệu *XML* khi phương thức `ValidateXml` được gọi. Các lỗi sẽ được hiển thị trong một cửa sổ *Console*, và một biến luận lý được trả về để cho biết quá trình kiểm tra thành công hay thất bại.

```

using System;

using System.Xml;

using System.Xml.Schema;

public class ConsoleValidator {

    // Thiết lập thành true nếu tồn tại ít nhất một lỗi.
    private bool failed;

    public bool Failed {
        get {return failed;}
    }
}

```

```
public bool ValidateXml(string xmlFilename, string schemaFilename) {

    // Tạo validator.
    XmlTextReader r = new XmlTextReader(xmlFilename);
    XmlValidatingReader validator = new XmlValidatingReader(r);
    validator.ValidationType = ValidationType.Schema;

    // Nạp Schema vào validator.
    XmlSchemaCollection schemas = new XmlSchemaCollection();
    schemas.Add(null, schemaFilename);
    validator.Schemas.Add(schemas);

    // Thiết lập phương thức thụ lý sự kiện validation.
    validator.ValidationEventHandler +=
        new ValidationEventHandler(ValidationEventHandler);

    failed = false;
    try {
        // Đọc tất cả dữ liệu XML.
        while (validator.Read())
        {}
    }catch (XmlException err) {
        // Điều này xảy ra khi tài liệu XML có chứa ký tự bất
        // hợp lệ hoặc các thẻ lồng nhau hay đóng không đúng.
        Console.WriteLine("A critical XML error has occurred.");
        Console.WriteLine(err.Message);
        failed = true;
    }finally {
        validator.Close();
    }

    return !failed;
}

private void ValidationEventHandler(object sender,
    ValidationEventArgs args) {
```

```

        failed = true;

        // Hiển thị lỗi validation.
        Console.WriteLine("Validation error: " + args.Message);
        Console.WriteLine();
    }
}

```

Dưới đây là cách sử dụng lớp này để xác nhận tính hợp lệ của danh mục sản phẩm:

```

using System;

public class ValidateXml {

    private static void Main() {

        ConsoleValidator consoleValidator = new ConsoleValidator();
        Console.WriteLine("Validating ProductCatalog.xml.");

        bool success = consoleValidator.ValidateXml("ProductCatalog.xml",
            "ProductCatalog.xsd");
        if (!success) {
            Console.WriteLine("Validation failed.");
        } else {
            Console.WriteLine("Validation succeeded.");
        }

        Console.ReadLine();
    }
}

```

Nếu tài liệu hợp lệ thì sẽ không có thông báo nào xuất hiện, và biến `success` sẽ được thiết lập thành `true`. Nhưng xét xem điều gì sẽ xảy ra nếu bạn sử dụng một tài liệu phá vỡ các quy tắc *Schema*, chẳng hạn file *ProductCatalog_Invalid.xml* như sau:

```

<?xml version="1.0" ?>
<productCatalog>
    <catalogName>Acme Fall 2003 Catalog</catalogName>
    <expiryDate>Jan 1, 2004</expiryDate>

```

```

<products>
  <product id="1001">
    <productName>Magic Ring</productName>
    <productPrice>$342.10</productPrice>
    <inStock>true</inStock>
  </product>
  <product id="1002">
    <productName>Flying Carpet</productName>
    <productPrice>982.99</productPrice>
    <inStock>Yes</inStock>
  </product>
</products>
</productCatalog>

```

Nếu bạn kiểm tra tài liệu này, biến `success` sẽ được thiết lập thành `false` và kết xuất sẽ cho biết các lỗi:

```
Validating ProductCatalog_Invalid.xml.
```

```
Validation error: The 'expiryDate' element has an invalid value according to its
data type. An error occurred at file:///I:/CSharp/Chuong05/05-08/
bin/Debug/ProductCatalog_Invalid.xml, (4, 30).
```

```
Validation error: The 'productPrice' element has an invalid value according to its
data type. An error occurred at file:///I:/CSharp/Chuong05/05-08/
bin/Debug/ProductCatalog_Invalid.xml, (9, 36).
```

```
Validation error: The 'inStock' element has an invalid value according to its data
type. An error occurred at file:///I:/CSharp/Chuong05/05-08/
bin/Debug/ProductCatalog_Invalid.xml, (15, 27).
```

```
Validation failed.
```

Cuối cùng, nếu muốn xác nhận tính hợp lệ của một tài liệu *XML* và rồi xử lý nó, bạn có thể sử dụng `XmlValidatingReader` để quét tài liệu khi nó được đọc vào một `XmlDocument` trong-bộ-nhớ:

```

XmlDocument doc = new XmlDocument();
XmlTextReader r = new XmlTextReader("ProductCatalog.xml");
XmlValidatingReader validator = new XmlValidatingReader(r);

// Nạp Schema vào validator.
validator.ValidationType = ValidationType.Schema;

```

```

XmlSchemaCollection schemas = new XmlSchemaCollection();
schemas.Add(null, "ProductCatalog.xsd");
validator.Schemas.Add(schemas);

// Nạp và kiểm tra tài liệu cùng một lúc.
try {
    doc.Load(validator);
    // (Validation thành công.)
} catch (XmlSchemaException err) {
    // (Validation thất bại.)
}

```

9. Sử dụng XML Serialization với các đối tượng tùy biến

?

Bạn cần sử dụng XML như một định dạng tuần tự hóa (*serialization format*). Tuy nhiên, bạn không muốn xử lý XML trực tiếp trong mã lệnh, mà muốn tương tác với dữ liệu bằng các đối tượng tùy biến.

✕

Sử dụng lớp `System.Xml.Serialization.XmlSerializer` để chuyển dữ liệu từ đối tượng của bạn sang XML, và ngược lại. Bạn cũng có thể đánh dấu mã lệnh của lớp bằng các đặc tính để tùy biến biểu diễn XML của nó.

Lớp `XmlSerializer` cho phép chuyển các đối tượng thành dữ liệu XML, và ngược lại. Lớp này đủ thông minh để tạo đúng các mảng khi nó tìm thấy các phần tử lồng bên trong.

Các yêu cầu khi sử dụng `XmlSerializer`:

- `XmlSerializer` chỉ tuần tự hóa các thuộc tính và các biến công khai.
- Các lớp cần tuần tự hóa phải chứa một phương thức khởi dựng mặc định không có đối số. `XmlSerializer` sẽ sử dụng phương thức khởi dựng này khi tạo đối tượng mới trong quá trình giải tuần tự hóa.
- Các thuộc tính của lớp phải là khả-đọc (*readable*) và khả-ghi (*writable*). Đó là vì `XmlSerializer` sử dụng hàm truy xuất thuộc tính `get` để lấy thông tin và hàm truy xuất thuộc tính `set` để phục hồi dữ liệu sau khi giải tuần tự hóa.

✍

Bạn cũng có thể lưu trữ các đối tượng theo định dạng dựa-trên-XML bằng cách sử dụng *.NET Serialization* và `System.Runtime.Serialization.Formatters.Soap.SoapFormatter`. Trong trường hợp này, bạn chỉ cần làm cho lớp của bạn trở thành khả-tuần-tự-hóa, không cần cung cấp phương thức khởi dựng mặc định hay bảo đảm tất cả các thuộc tính là khả ghi. Tuy nhiên, cách này không cho bạn quyền kiểm soát trên định dạng XML đã-được-tuần-tự-hóa.

Để sử dụng *XML serialization*, trước hết bạn phải đánh dấu các đối tượng dữ liệu với các đặc tính cho biết phép ánh xạ sang *XML*. Các đặc tính này thuộc không gian tên `System.Xml.Serialization` và bao gồm:

- `XmlRoot`—cho biết tên phần tử gốc của file *XML*. Theo mặc định, `XmlSerializer` sẽ sử dụng tên của lớp. Đặc tính này có thể được áp dụng khi khai báo lớp.
- `XmlElement`—cho biết tên phần tử dùng cho một thuộc tính hay biến công khai. Theo mặc định, `XmlSerializer` sẽ sử dụng tên của thuộc tính hay biến công khai.
- `XmlAttribute`—cho biết một thuộc tính hay biến công khai sẽ được tuần tự hóa thành một đặc tính (không phải phần tử), và chỉ định tên đặc tính.
- `XmlEnum`—cấu hình phần text sẽ được sử dụng khi tuần tự hóa các giá trị liệt kê. Nếu bạn không sử dụng `XmlEnum`, tên của hằng liệt kê sẽ được sử dụng.
- `XmlIgnore`—cho biết một thuộc tính hay biến công khai sẽ không được tuần tự hóa.

Ví dụ, xét danh mục sản phẩm đã được trình bày trong mục 5.1. Bạn có thể mô tả tài liệu *XML* này bằng các đối tượng `ProductCatalog` và `Product` như sau:

```
using System;
using System.Xml.Serialization;

[XmlRoot("productCatalog")]
public class ProductCatalog {

    [XmlElement("catalogName")]
    public string CatalogName;

    // Sử dụng kiểu dữ liệu ngày (bỏ qua phần giờ).
    [XmlElement(ElementName="expiryDate", DataType="date")]
    public DateTime ExpiryDate;

    // Cấu hình tên thẻ.
    [XmlArray("products")]
    [XmlArrayItem("product")]
    public Product[] Products;

    public ProductCatalog() {
        // Phương thức khởi dựng mặc định (dùng khi giải tuần tự hóa).
    }

    public ProductCatalog(string catalogName, DateTime expiryDate) {
        this.CatalogName = catalogName;
```

```

        this.ExpiryDate = expiryDate;
    }
}

public class Product {

    [XmlElement("productName")]
    public string ProductName;

    [XmlElement("productPrice")]
    public decimal ProductPrice;

    [XmlElement("inStock")]
    public bool InStock;

    [XmlAttribute(AttributeName="id", DataType="integer")]
    public string Id;

    public Product() {
        // Phương thức khởi dựng mặc định (dùng khi giải tuần tự hóa).
    }

    public Product(string productName, decimal productPrice) {
        this.ProductName = productName;
        this.ProductPrice = productPrice;
    }
}

```

Chú ý rằng, các lớp này sử dụng các đặc tính *XML Serialization* để đổi tên phần tử (sử dụng kiểu ký hiệu *Pascal*¹ trong tên thành viên lớp, và kiểu ký hiệu lửng lạc đà² trong tên thẻ *XML*), cho biết các kiểu dữ liệu không rõ ràng, và chỉ định các phần tử <product> sẽ được lồng bên trong <productCatalog> như thế nào.

Bằng cách sử dụng các lớp tùy biến này và đối tượng `XmlSerializer`, bạn có thể chuyển *XML* thành các đối tượng và ngược lại. Đoạn mã dưới đây tạo một đối tượng `ProductCatalog` mới,

¹ *Pascal casing*: Mẫu tự đầu tiên của các chữ đều viết hoa, ví dụ `SomeOtherName`

² *Camel casing*: Mẫu tự đầu tiên của chữ đầu viết thường, mẫu tự đầu tiên của các chữ đi sau viết hoa, ví dụ `someOtherName`

tuần tự hóa đối tượng thành tài liệu *XML*, giải tuần tự hóa tài liệu thành đối tượng, và rồi hiển thị tài liệu này:

```
using System;
using System.Xml;
using System.Xml.Serialization;
using System.IO;

public class SerializeXml {

    private static void Main() {

        // Tạo danh mục sản phẩm.
        ProductCatalog catalog = new ProductCatalog("New Catalog",
            DateTime.Now.AddYears(1));
        Product[] products = new Product[2];
        products[0] = new Product("Product 1", 42.99m);
        products[1] = new Product("Product 2", 202.99m);
        catalog.Products = products;

        // Tuần tự hóa danh mục ra file.
        XmlSerializer serializer =
            new XmlSerializer(typeof(ProductCatalog));
        FileStream fs =
            new FileStream("ProductCatalog.xml", FileMode.Create);
        serializer.Serialize(fs, catalog);
        fs.Close();

        catalog = null;

        // Giải tuần tự hóa danh mục từ file.
        fs = new FileStream("ProductCatalog.xml", FileMode.Open);
        catalog = (ProductCatalog)serializer.Deserialize(fs);

        // Tuần tự hóa danh mục ra cửa sổ Console.
        serializer.Serialize(Console.Out, catalog);
        Console.ReadLine();
    }
}
```

}

10.**Tạo XML Schema cho một lớp .NET****?**

Bạn cần tạo một XML Schema dựa trên một hay nhiều lớp C#. Điều này cho phép bạn kiểm tra tính hợp lệ của các tài liệu XML trước khi giải tuần tự hóa chúng với `XmlSerializer`.

X

Sử dụng tiện ích dòng lệnh XML Schema Definition Tool (`xsd.exe`—đi kèm với .NET Framework). Chỉ định tên của assembly làm đối số dòng lệnh, và thêm đối số `/t: [TypeName]` để cho biết kiểu cần chuyển đổi.

Mục 5.9 đã trình bày cách sử dụng `XmlSerializer` để tuần tự hóa đối tượng .NET thành XML, và giải tuần tự hóa XML thành đối tượng .NET. Nhưng nếu muốn sử dụng XML như một phương cách để tương tác với các ứng dụng khác, quy trình nghiệp vụ, hay các ứng dụng phi-Framework, bạn sẽ cần xác nhận tính hợp lệ của XML trước khi giải tuần tự hóa nó. Bạn cũng sẽ cần tạo một tài liệu XML Schema định nghĩa cấu trúc và các kiểu dữ liệu được sử dụng trong định dạng XML của bạn, để các ứng dụng khác có thể làm việc với nó. Một giải pháp là sử dụng tiện ích dòng lệnh `xsd.exe`.

Tiện ích `xsd.exe` đi kèm với .NET Framework. Nếu đã cài đặt Microsoft Visual Studio .NET, bạn sẽ tìm thấy nó trong thư mục `C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Bin`. Tiện ích `xsd.exe` có thể tạo ra XML Schema từ một assembly đã được biên dịch. Bạn chỉ cần cung cấp tên file và cho biết lớp mô tả tài liệu XML với đối số `/t: [TypeName]`.

Ví dụ, xét các lớp `ProductCatalog` và `Product` đã được trình bày trong mục 5.9. Bạn có thể tạo XML Schema cho một danh mục sản phẩm với dòng lệnh sau:

```
xsd 05-09.exe /t:ProductCatalog
```

Bạn chỉ cần chỉ định lớp `ProductCatalog` trên dòng lệnh, vì lớp này mô tả tài liệu XML. XML Schema được tạo ra trong ví dụ này (có tên mặc định là `schema0.xsd`) sẽ mô tả đầy đủ một danh mục sản phẩm, với các item sản phẩm lồng bên trong. Bây giờ, bạn có thể sử dụng `XmlValidatingReader` (đã được trình bày trong mục 5.8) để kiểm tra tính hợp lệ của tài liệu XML dựa vào XML Schema này.

11.**Tạo lớp từ một XML Schema****?**

Bạn cần tạo một hay nhiều lớp C# dựa trên một XML Schema; để sau đó, bạn có thể tạo một tài liệu XML theo định dạng phù hợp bằng các đối tượng này và `XmlSerializer`.

X

Sử dụng tiện ích dòng lệnh `xsd.exe` (đi kèm với .NET Framework). Chỉ định tên file Schema làm đối số dòng lệnh, và thêm đối số `/c` để cho biết bạn muốn tạo mã lệnh cho lớp.

Mục 5.10 đã giới thiệu tiện ích dòng lệnh *xsd.exe*, tiện ích này có thể được sử dụng để tạo *XML Schema* dựa trên định nghĩa lớp. Quá trình ngược lại (tạo mã lệnh C# dựa trên một tài liệu *XML Schema*) cũng có thể xảy ra. Việc này hữu ích khi bạn muốn ghi một định dạng *XML* nào đó, nhưng lại không muốn tạo tài liệu này bằng cách ghi từng nút một với lớp *XmlDocument* hay *XmlTextWriter*. Thay vào đó, bằng cách sử dụng *xsd.exe*, bạn có thể tạo ra một tập đầy đủ các đối tượng .NET. Kế đó, bạn có thể tuần tự hóa các đối tượng này thành biểu diễn *XML* bằng *XmlSerializer*, như được mô tả trong mục 5.9.

Để tạo mã lệnh từ một *XML Schema*, bạn chỉ cần cung cấp tên file *Schema* và thêm đối số */c* để cho biết bạn muốn tạo ra lớp. Ví dụ, xét *XML Schema* đã được trình bày trong mục 5.8. Bạn có thể tạo mã lệnh C# từ *Schema* này với dòng lệnh sau:

```
xsd ProductCatalog.xsd /c
```

Lệnh này sẽ tạo ra một file (*ProductCatalog.cs*) gồm hai lớp: *Product* và *productCalalog*. Hai lớp này tương tự với hai lớp đã được tạo trong mục 5.9.

12.

Thực hiện phép biến đổi XSL



Bạn cần biến đổi một tài liệu XML thành một tài liệu khác bằng XSLT stylesheet.



Sử dụng lớp `System.Xml.Xsl.XslTransform`. Nạp XSLT stylesheet bằng phương thức `XslTransform.Load`, và tạo tài liệu kết xuất bằng phương thức `Transform` (cần cung cấp tài liệu nguồn).

XSLT (hay *XSL Transforms*) là một ngôn ngữ dựa-trên-XML, được thiết kế để biến đổi một tài liệu XML thành một tài liệu khác. *XSLT* có thể được sử dụng để tạo một tài liệu XML mới với cùng dữ liệu nhưng được sắp xếp theo một cấu trúc khác hoặc để chọn một tập con dữ liệu trong một tài liệu. Nó cũng có thể được sử dụng để tạo một kiểu tài liệu có cấu trúc khác. *XSLT* thường được sử dụng theo cách này để định dạng một tài liệu XML thành một trang HTML.

XSLT là một ngôn ngữ đa năng, và việc tạo *XSL Transforms* vượt quá phạm vi quyền sách này. Tuy nhiên, bạn có thể học cách tạo các tài liệu *XSLT* đơn giản bằng cách xem một ví dụ cơ bản. Mục này sẽ biến đổi tài liệu *orders.xml* (đã được trình bày trong mục 5.6) thành một tài liệu HTML và rồi hiển thị kết quả. Để thực hiện phép biến đổi này, bạn sẽ cần *XSLT stylesheet* như sau:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0" >

    <xsl:template match="Order">
        <html><body><p>
            Order <b><xsl:value-of select="Client/@id"/></b>
            for <xsl:value-of select="Client/Name"/></p>
        <table border="1">
```

```

        <td>ID</td><td>Name</td><td>Price</td>
        <xsl:apply-templates select="Items/Item"/>
    </table></body></html>
</xsl:template>

<xsl:template match="Items/Item">
    <tr>
        <td><xsl:value-of select="@id"/></td>
        <td><xsl:value-of select="Name"/></td>
        <td><xsl:value-of select="Price"/></td>
    </tr>
</xsl:template>

</xsl:stylesheet>

```

Về cơ bản, mọi *XSL stylesheet* gồm một tập các template. Mỗi template so trùng với các phần tử trong tài liệu nguồn và rồi mô tả các phần tử được so trùng để tạo nên tài liệu kết quả. Để so trùng template, tài liệu *XSLT* sử dụng biểu thức *XPath*, như được mô tả trong mục 5.6.

Stylesheet vừa trình bày ở trên (*orders.xslt*) gồm hai template (là các con của phần tử *stylesheet* gốc). Template đầu tiên trùng khớp với phần tử *Order* gốc. Khi bộ xử lý *XSLT* tìm thấy một phần tử *Order*, nó sẽ ghi ra các thẻ cần thiết để bắt đầu một bảng *HTML* với các tiêu đề cột thích hợp và chèn dữ liệu về khách hàng bằng lệnh **value-of** (ghi ra kết quả dạng text của một biểu thức *XPath*). Trong trường hợp này, các biểu thức *XPath* (*Client/@id* và *Client/Name*) trùng với đặc tính *id* và phần tử *Name*.

Kế tiếp, lệnh **apply-templates** được sử dụng để phân nhánh và xử lý các phần tử *Item* nằm trong. Điều này là cần thiết vì có thể có nhiều phần tử *Item*. Mỗi phần tử *Item* được so trùng bằng biểu thức *Items/Item* (nút gốc *Order* không được chỉ định vì *Order* chính là nút hiện tại). Cuối cùng, các thẻ cần thiết sẽ được ghi ra để kết thúc tài liệu *HTML*.

Nếu thực thi phép biến đổi này trên file *orders.xml* (đã trình bày trong mục 5.6), bạn sẽ nhận được kết quả (tài liệu *HTML*) như sau:

```

<html>
  <body>
    <p>
      Order <b>ROS-930252034</b>
      for Remarkable Office Supplies</p>
    <table border="1">
      <td>ID</td>
      <td>Name</td>

```

```
<td>Price</td>
<tr>
  <td>1001</td>
  <td>Electronic Protractor</td>
  <td>42.99</td>
</tr>
<tr>
  <td>1002</td>
  <td>Invisible Ink</td>
  <td>200.25</td>
</tr>
</table>
</body>
</html>
```

Để áp dụng một *XSLT stylesheet* trong *.NET*, bạn cần sử dụng lớp `XslTransform`. Ứng dụng dưới đây áp dụng phép biến đổi và rồi hiển thị file đã được biến đổi trong cửa sổ trình duyệt web. Trong ví dụ này, mã lệnh đã sử dụng phiên bản nạp chồng của phương thức `Transform` để lưu trực tiếp tài liệu kết quả ra đĩa, mặc dù bạn có thể thu lấy và xử lý nó như một stream bên trong ứng dụng của bạn.

```
using System;
using System.Windows.Forms;
using System.Xml.Xsl;

public class TransformXml : System.Windows.Forms.Form {

    private AxSHDocVw.AxWebBrowser webBrowser;

    // (Bỏ qua phần mã designer.)

    private void TransformXml_Load(object sender, System.EventArgs e) {

        XslTransform transform = new XslTransform();

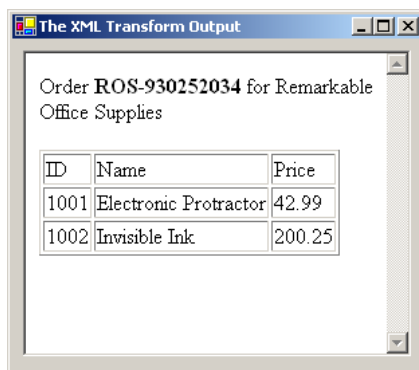
        // Nạp XSL stylesheet.
        transform.Load("orders.xslt");

        // Biến đổi orders.xml thành orders.html.
        transform.Transform("orders.xml", "orders.html", null);
    }
}
```

```

object var = null;
webBrowser.Navigate(
    "file:///\" + Application.StartupPath + @\"orders.html\",
    ref var, ref var, ref var, ref var);
}
}

```



Hình 5.5 Kết xuất stylesheet cho orders.xml



.NET Framework không có điều kiểm nào dùng để thể hiện nội dung *HTML*. Tuy nhiên, chức năng này có thể có được thông qua khả năng liên tác *COM* nếu bạn sử dụng điều kiểm *ActiveX Web Browser* (đi cùng *Microsoft Internet Explorer* và hệ điều hành *Microsoft Windows*). Cửa sổ này có thể hiển thị các file *HTML* cục bộ hay ở xa, và hỗ trợ *JavaScript*, *VBScript*, và tất cả các plug-in cho *Internet Explorer* (xem mục 11.4 để biết cách thêm điều kiểm *Web Browser* vào dự án).

6

WINDOWS FORM

Microsoft .NET Framework chứa một tập phong phú các lớp dùng để tạo các ứng dụng dựa-trên-Windows truyền thống trong không gian tên `System.Windows.Forms`. Các lớp này có phạm vi từ các phần cơ bản như các lớp `TextBox`, `Button`, và `MainMenu` đến các điều khiển chuyên biệt như `TreeView`, `LinkLabel`, và `NotifyIcon`. Ngoài ra, bạn sẽ tìm thấy tất cả các công cụ cần thiết để quản lý các ứng dụng giao diện đa tài liệu (*Multiple Document Interface—MDI*), tích hợp việc trợ giúp cảm-ngữ-cảnh, và ngay cả tạo các giao diện người dùng đa ngôn ngữ—tất cả đều không cần viện đến sự phức tạp của *Win32 API*.

Hầu hết các nhà phát triển C# có thể tự nắm bắt nhanh chóng mô hình lập trình *Windows Form*. Tuy nhiên, có một số thủ thuật và kỹ thuật không tốn nhiều thời gian có thể làm cho việc lập trình *Windows* hiệu quả hơn. Chương này sẽ trình bày các vấn đề sau đây:

- Cách khai thác triệt để các điều khiển, bao gồm thêm chúng vào form lúc thực thi (mục 6.1), liên kết chúng với dữ liệu nào đó (mục 6.2), và xử lý chúng một cách tổng quát (mục 6.3).
- Cách làm việc với form, bao gồm theo vết chúng trong một ứng dụng (mục 6.4), sử dụng *MDI* (mục 6.5), và lưu trữ thông tin về kích thước và vị trí (mục 6.6). Bạn cũng sẽ biết cách tạo form đa ngôn ngữ (mục 6.13) và form không đường viền (mục 6.14 và 6.15).
- Một số thủ thuật khi làm việc với các điều khiển thông dụng như `ListBox` (mục 6.7), `TextBox` (mục 6.8), `ComboBox` (mục 6.9), `ListView` (mục 6.10), và `Menu` (mục 6.11 và 6.12).
- Cách tạo một icon động trong khay hệ thống (mục 6.16).
- Các khái niệm mà bạn có thể áp dụng cho nhiều kiểu điều khiển, bao gồm xác nhận tính hợp lệ (mục 6.17), kéo-và-thả (mục 6.18), trợ giúp cảm-ngữ-cảnh (mục 6.19), phong cách *Windows XP* (mục 6.20), và độ đục của form (mục 6.21).



Hầu hết các mục trong chương này sử dụng các lớp điều khiển, luôn được định nghĩa trong không gian tên `System.Windows.Forms`. Khi đưa vào các lớp này, tên không gian tên đầy đủ không được chỉ định, và `System.Windows.Forms` được thừa nhận.

1.

Thêm điều khiển vào form lúc thực thi



Bạn cần thêm một điều khiển vào form lúc thực thi, không phải lúc thiết kế.



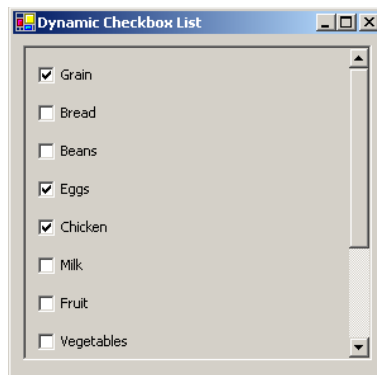
Tạo một đối tượng của lớp điều khiển thích hợp. Kế đó, thêm đối tượng này vào một form hoặc một điều khiển container bằng phương thức `Add` của `ControlCollection`.

Trong một ứng dụng dựa-trên-Windows .NET, không có sự khác biệt nào giữa việc tạo điều khiển lúc thiết kế và việc tạo điều khiển lúc thực thi. Khi bạn tạo một điều khiển lúc thiết kế (sử dụng công cụ *Microsoft Visual Studio .NET*), đoạn mã cần thiết sẽ được thêm vào lớp form, cụ thể là trong một phương thức đặc biệt có tên là `InitializeComponent`. Bạn có thể sử dụng

đoạn mã giống như vậy trong ứng dụng của bạn để tạo điều kiểm. Bạn cần thực hiện các bước sau:

1. Tạo một đối tượng của lớp điều kiểm thích hợp.
2. Cấu hình các thuộc tính của điều kiểm (đặc biệt là kích thước và tọa độ vị trí).
3. Thêm điều kiểm này vào form hoặc điều kiểm container.
4. Ngoài ra, nếu cần thụ lý các sự kiện cho điều kiểm mới, bạn có thể gắn chúng vào các phương thức hiện có.

Mỗi điều kiểm đều cung cấp thuộc tính `Controls` để tham chiếu đến `ControlCollection` chứa tất cả các điều kiểm con của nó. Để thêm một điều kiểm con, bạn cần gọi phương thức `ControlCollection.Add`. Ví dụ sau đây sẽ làm rõ điều này bằng cách tạo động một danh sách các `CheckBox`. Một `CheckBox` được thêm vào cho mỗi item trong một mảng. Tất cả các `CheckBox` được thêm vào một `Panel` (`Panel` có thuộc tính `AutoScroll` là `true` để có thể cuộn qua danh sách các `CheckBox`).



Hình 6.1 Danh sách các `CheckBox` được-tạo-động

```
using System;
using System.Windows.Forms;

public class DynamicCheckBox : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    private void DynamicCheckBox_Load(object sender,
        System.EventArgs e) {

        // Tạo mảng.
        string[] foods = {"Grain", "Bread", "Beans", "Eggs",
            "Chicken", "Milk", "Fruit", "Vegetables",
```

```

        "Pasta", "Rice", "Fish", "Beef");

    int topPosition = 10;
    foreach (string food in foods)
    {
        // Tạo một CheckBox mới.
        CheckBox checkBox = new CheckBox();
        checkBox.Left = 10;
        checkBox.Top = topPosition;
        topPosition += 30;
        checkBox.Text = food;

        // Thêm CheckBox vào form.
        panel.Controls.Add(checkBox);
    }
}

```

2.

Liên kết dữ liệu vào điều kiểm

? Bạn cần liên kết một đối tượng vào một điều kiểm cụ thể (có thể là để lưu trữ vài thông tin nào đó liên quan đến một item cho trước).

✕ Lưu trữ một tham chiếu đến đối tượng trong thuộc tính `Tag` của điều kiểm.

Mọi lớp dẫn xuất từ `System.Windows.Forms.Control` đều cung cấp thuộc tính `Tag` và bạn có thể sử dụng nó để lưu trữ một tham chiếu đến bất kỳ kiểu đối tượng nào. Thuộc tính `Tag` không được điều kiểm hay *Microsoft .NET Framework* sử dụng mà nó được để dành làm nơi lưu trữ các thông tin đặc thù của ứng dụng. Ngoài ra, một vài lớp khác không dẫn xuất từ `Control` cũng cung cấp thuộc tính `Tag`, chẳng hạn các lớp `ListViewItem` và `TreeNode` (trình bày các item trong một `ListView` hoặc `TreeView`). Một lớp không cung cấp thuộc tính `Tag` là `MenuItem`.

Thuộc tính `Tag` được định nghĩa là một kiểu `Object`, nghĩa là bạn có thể sử dụng nó để lưu trữ bất kỳ kiểu giá trị hoặc kiểu tham chiếu nào, từ một số hoặc chuỗi đơn giản cho đến một đối tượng tùy biến do bạn định nghĩa. Khi lấy dữ liệu từ thuộc tính `Tag`, bạn sẽ cần ép (kiểu) đối tượng thành kiểu gốc của nó.

Ví dụ sau đây thêm danh sách các file vào một `ListView`. Đối tượng `FileInfo` tương ứng với mỗi file được lưu trữ trong thuộc tính `Tag`. Khi người dùng nhấp đúp vào một trong các item, ứng dụng sẽ lấy đối tượng `FileInfo` từ thuộc tính `Tag` và hiển thị kích thước file trong một `MessageBox` (xem hình 6.2).

```

using System;
using System.Windows.Forms;

```

```
using System.IO;

public class TagPropertyExample : System.Windows.Forms.Form (

    // (Bỏ qua phần mã designer.)

    private void TagPropertyExample_Load(object sender,
        System.EventArgs e) {

        // Lấy tất cả các file trong thư mục gốc ổ đĩa C.
        DirectoryInfo directory = new DirectoryInfo("C:\\");
        FileInfo[] files = directory.GetFiles();

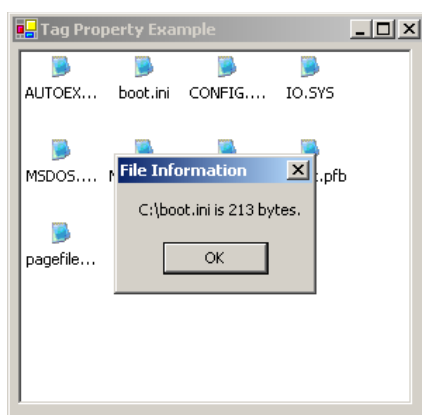
        // Hiển thị tất cả các file trong ListView.
        foreach (FileInfo file in files) {

            ListViewItem item = listView.Items.Add(file.Name);
            item.ImageIndex = 0;
            item.Tag = file;
        }
    }

    private void listView_ItemActivate(object sender,
        System.EventArgs e) {

        // Lấy kích thước file.
        ListViewItem item = ((ListView)sender).SelectedItem[0];
        FileInfo file = (FileInfo)item.Tag;
        string info = file.FullName + " is " + file.Length + " bytes.";

        // Hiển thị kích thước file.
        MessageBox.Show(info, "File Information");
    }
}
```



Hình 6.2 Lưu trữ dữ liệu trong thuộc tính Tag

3. *Xử lý tất cả các điều kiểm trên form*

? Bạn cần thực hiện một tác vụ chung cho tất cả các điều kiểm trên form (ví dụ, lấy hay xóa thuộc tính `Text` của chúng, thay đổi màu hay thay đổi kích thước của chúng).

X Duyệt (đệ quy) qua tập hợp các điều kiểm. Tương tác với mỗi điều kiểm bằng các thuộc tính và phương thức của lớp `Control` cơ sở.

Bạn có thể duyệt qua các điều kiểm trên form bằng tập hợp `Form.Controls`, tập này chứa tất cả các điều kiểm nằm trực tiếp trên bề mặt form. Tuy nhiên, nếu vài điều kiểm trong số đó là điều kiểm container (như `GroupBox`, `Panel`, hoặc `TabPage`), chúng có thể chứa nhiều điều kiểm nữa. Do đó, cần sử dụng kỹ thuật đệ quy để kiểm tra tập hợp `Controls`.

Ví dụ sau đây trình bày một form thực hiện kỹ thuật đệ quy để tìm mọi `TextBox` có trên form và xóa đi toàn bộ text trong đó. Form sẽ kiểm tra mỗi điều kiểm để xác định xem nó có phải là `TextBox` hay không bằng toán tử `typeof`.

```
using System;

using System.Windows.Forms;

public class ProcessAllControls : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    private void cmdProcessAll_Click(object sender, System.EventArgs e) {
        ProcessControls(this);
    }

    private void ProcessControls(Control ctrl) {
```

```

// Bỏ qua điều kiểm tra khi nó là TextBox.
if (ctrl.GetType() == typeof(TextBox)) {
    ctrl.Text = "";
}

// Xử lý các điều kiểm một cách đệ quy.
// Điều này cần thiết khi có một điều kiểm chứa nhiều
// điều kiểm khác (ví dụ, khi bạn sử dụng Panel,
// GroupBox, hoặc điều kiểm container nào khác).
foreach (Control ctrlChild in ctrl.Controls) {
    ProcessControls(ctrlChild);
}
}
}

```

4. *Theo vết các form khả kiến trong một Ứng dụng*

?

Bạn muốn giữ lại vết của tất cả form hiện đang được hiển thị. Đây là trường hợp thường gặp khi bạn muốn một form có thể tương tác với một form khác.

✕

Tạo một lớp giữ các tham chiếu đến các đối tượng Form. Lưu trữ các tham chiếu này bằng biến tĩnh.

.NET không cung cấp cách xác định form nào đang được hiển thị trong một ứng dụng (ngoại trừ ứng dụng *MDI*, sẽ được mô tả trong mục 6.5). Nếu muốn xác định form nào đang tồn tại, form nào đang được hiển thị, hoặc bạn muốn một form có thể gọi các phương thức và thiết lập các thuộc tính của một form khác thì bạn cần phải giữ lại vết của các đối tượng form.

Để thực hiện yêu cầu trên, hãy tạo một lớp gồm các thành viên tĩnh; lớp này có thể theo vết các form đang mở bằng một tập hợp, hay các thuộc tính chuyên biệt. Ví dụ, lớp dưới đây có thể theo vết hai form:

```

public class OpenForms {

    public static Form MainForm;
    public static Form SecondaryForm;
}

```

Khi form chính hoặc form phụ được hiển thị, chúng sẽ tự đăng ký với lớp `OpenForms`. Nơi hợp lý để đặt đoạn mã này là trong phương thức thụ lý sự kiện `Form.Load`.

```

private void MainForm_Load(object sender, EventArgs e) {

```



```
// Đăng ký đối tượng form vừa được tạo.
OpenForms.MainForm = this;
}
```

Bạn có thể sử dụng đoạn mã tương tự để gỡ bỏ tham chiếu khi form bị đóng.

```
private void MainForm_Unload(object sender, EventArgs e) {

    // Gỡ bỏ đối tượng form.
    OpenForms.MainForm = null;
}
```

Bây giờ, một form khác có thể tương tác với form này thông qua lớp `OpenForms`. Ví dụ, dưới đây là cách form chính làm ẩn form phụ:

```
if (OpenForms.SecondaryForm != null) {
    OpenForms.SecondaryForm.Hide();
}
```

Trong cách tiếp cận này, chúng ta giả sử mọi form được tạo chỉ một lần. Nếu bạn có một ứng dụng dựa-trên-tài-liệu (*document-based application*), trong đó, người dùng có thể tạo nhiều đối tượng của cùng một form, bạn cần theo vết các form này bằng một tập hợp. Tập hợp `ArrayList` dưới đây là một ví dụ:

```
public class OpenForms {

    public static Form MainForm;
    public static ArrayList DocForms = new ArrayList();
}
```

Theo đó, form có thể tự thêm vào tập hợp khi cần, như được trình bày trong đoạn mã sau đây:

```
private void DocForm_Load(object sender, EventArgs e) {

    // Đăng ký đối tượng form vừa được tạo.
    OpenForms.DocForms.Add(this);
}
```

5. Tìm tất cả các form trong Ứng dụng MDI



Bạn cần tìm tất cả các form hiện đang được hiển thị trong một ứng dụng giao diện đa tài liệu (*Multiple Document Interface*).



Duyệt qua các form trong tập hợp `MdiChildren` của form *MDI* cha.

.NET Framework có hai “lối tắt” thuận lợi cho việc quản lý các ứng dụng *MDI*: thuộc tính `MdiChildren` và `MdiParent` của lớp `Form`. Bạn có thể xét thuộc tính `MdiParent` của bất kỳ form

MDI con nào để tìm form cha. Bạn có thể sử dụng tập hợp `MdiChildren` của form *MDI* cha để tìm tất cả các form con.

Ví dụ sau đây (xem hình 6.3) sẽ hiển thị tất cả các form con. Mỗi form con gồm một `Label` (chứa thông tin về ngày giờ), và một `Button`. Khi người dùng nhấp vào `Button`, phương thức xử lý sự kiện sẽ duyệt qua tất cả các form con và hiển thị dòng chữ trong `Label` (với thuộc tính chỉ-đọc).

Dưới đây là phần mã cho form con:

```
public class MDIChild : System.Windows.Forms.Form {

    private System.Windows.Forms.Button cmdShowAllWindows;
    private System.Windows.Forms.Label label;

    // (Bỏ qua phần mã designer.)

    public string LabelText {

        get {
            return label.Text;
        }
    }

    private void cmdShowAllWindows_Click(object sender,
        System.EventArgs e) {

        // Duyệt qua tập hợp các form con.
        foreach (Form frm in this.MdiParent.MdiChildren) {

            // Ép kiểu tham chiếu Form thành MDIChild.
            MDIChild child = (MDIChild)frm;
            MessageBox.Show(child.LabelText, frm.Text);
        }
    }

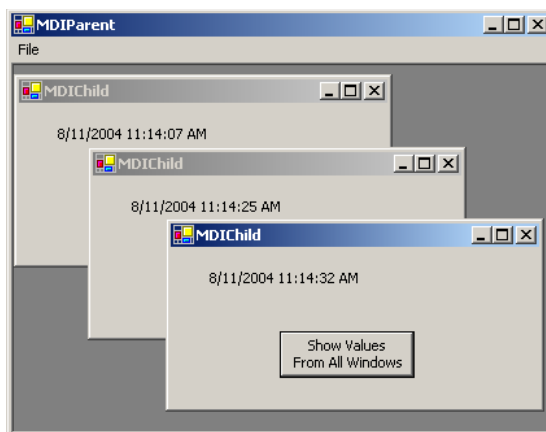
    private void MDIChild_Load(object sender, System.EventArgs e){

        label.Text = DateTime.Now.ToString();
    }
}
```

```
}

```

Chú ý rằng, khi đoạn mã duyệt qua tập hợp các form con, nó phải chuyển (ép kiểu) tham chiếu `Form` thành `MDIChild` để có thể sử dụng thuộc tính `LabelText`.



Hình 6.3 Lấy thông tin từ các form MDI con

6.

Lưu trữ kích thước và vị trí của form

- ?** Bạn cần lưu trữ kích thước và vị trí của một form (có thể thay đổi kích thước được) và phục hồi nó lại trong lần hiển thị form kế tiếp.
- ✕** Lưu trữ các thuộc tính `Left`, `Top`, `Width`, và `Height` của form trong *Windows Registry*.

Windows Registry là nơi lý tưởng để lưu trữ thông tin về vị trí và kích thước cho form. Cụ thể, bạn sẽ lưu trữ thông tin về mỗi form trong một khóa độc lập (có thể sử dụng tên của form làm khóa). Các khóa này sẽ được lưu trữ ngay dưới khóa ứng dụng.

Bạn cần tạo một lớp chuyên biệt để lưu và lấy các thiết lập cho form. Lớp `FormSettingStore` được trình bày dưới đây cung cấp hai phương thức: `SaveSettings`—nhận vào một form và ghi thông tin về kích thước và vị trí của nó vào *Registry*; và `ApplySettings`—nhận vào một form và áp dụng các thiết lập từ *Registry*. Đường dẫn của khóa và tên của khóa con được lưu trữ thành các biến thành viên lớp.

```
using System;
using System.Windows.Forms;
using Microsoft.Win32;

public class FormSettingStore {

    private string regPath;
    private string formName;
    private RegistryKey key;
```

```
public string RegistryPath {
    get {return regPath;}
}

public string FormName {
    get {return formName;}
}

public FormSettingStore(string registryPath, string formName) {

    this.regPath = registryPath;
    this.formName = formName;

    // Tạo khóa nếu nó chưa tồn tại.
    key = Registry.LocalMachine.CreateSubKey(
        registryPath + formName);
}

public void SaveSettings(System.Windows.Forms.Form form) {

    key.SetValue("Height", form.Height);
    key.SetValue("Width", form.Width);
    key.SetValue("Left", form.Left);
    key.SetValue("Top", form.Top);
}

public void ApplySettings(System.Windows.Forms.Form form) {

    form.Height = (int)key.GetValue("Height", form.Height);
    form.Width = (int)key.GetValue("Width", form.Width);
    form.Left = (int)key.GetValue("Left", form.Left);
    form.Top = (int)key.GetValue("Top", form.Top);
}
}
```

Để sử dụng lớp `FormSettingStore`, bạn chỉ cần thêm đoạn mã thụ lý sự kiện dưới đây vào bất kỳ form nào. Đoạn mã này sẽ lưu các thuộc tính của form khi form đóng và phục hồi chúng khi form được nạp.

```
private FormSettingStore formSettings;

private void Form1_Load(object sender, System.EventArgs e) {

    formSettings = new FormSettingStore(@"Software\MyApp\", this.Name);
    formSettings.ApplySettings(this);
}

private void Form1_Closed(object sender, System.EventArgs e) {

    formSettings.SaveSettings(this);
}
```



Nhớ rằng, việc truy xuất *Registry* có thể bị giới hạn căn cứ vào tài khoản người dùng hiện hành và chính sách bảo mật truy xuất mã lệnh (*Code Access Security Policy*). Khi bạn tạo một ứng dụng yêu cầu truy xuất *Registry*, assembly sẽ yêu cầu truy xuất *Registry* bằng yêu cầu quyền tối thiểu (*minimum permission request*—sẽ được mô tả trong mục 13.7).

7.

Buộc ListBox cuộn xuống



Bạn cần cuộn một `ListBox` (bằng mã lệnh) để những item nào đó trong danh sách có thể được nhìn thấy.



Thiết lập thuộc tính `ListBox.TopIndex` (thiết lập item được nhìn thấy đầu tiên).

Trong vài trường hợp, bạn có một `ListBox` lưu trữ một lượng thông tin đáng kể hoặc một `ListBox` mà bạn phải thêm thông tin vào một cách định kỳ. Thường thì thông tin mới nhất (được thêm vào cuối danh sách) lại là thông tin quan trọng hơn thông tin ở đầu danh sách. Một giải pháp là cuộn `ListBox` để có thể nhìn thấy các item vừa mới thêm vào.

Form dưới đây (gồm một `ListBox` và một `Button`) sẽ thêm 20 item vào danh sách rồi cuộn đến trang cuối cùng bằng thuộc tính `TopIndex` (xem hình 6.4):

```
using System;

using System.Windows.Forms;

public class ListBoxScrollTest : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)
```

```

int counter = 0;

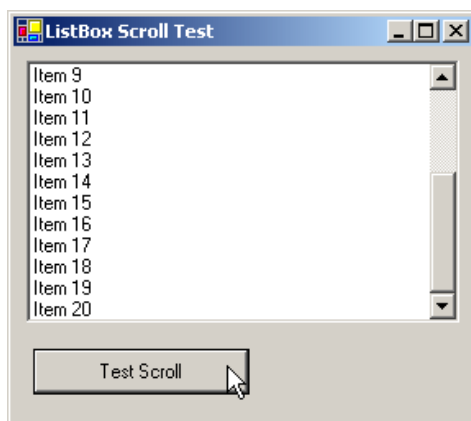
private void cmdTest_Click(object sender, System.EventArgs e) {

    for (int i = 0; i < 20; i++) {

        counter++;
        listBox1.Items.Add("Item " + counter.ToString());
    }

    listBox1.TopIndex = listBox1.Items.Count - 1;
}
}

```



Hình 6.4 Cuộn ListBox đến trang cuối cùng

8.

Chỉ cho phép nhập số vào TextBox

?

Bạn cần tạo một TextBox sao cho TextBox này bỏ qua tất cả các cú nhấn phím không phải số.

X

Thêm phương thức thụ lý sự kiện `TextBox.KeyPress`. Trong phương thức này, thiết lập thuộc tính `KeyPressEventArgs.Handled` là `true` để bỏ qua cú nhấn phím không hợp lệ.

Cách tốt nhất để hiệu chỉnh đầu vào bất hợp lệ là không cho nó được nhập ngay từ đầu. Điều này dễ dàng hiện thực với TextBox vì nó cung cấp sự kiện `KeyPress`, sự kiện này xảy ra sau khi cú nhấn phím được tiếp nhận nhưng trước khi nó được hiển thị. Bạn có thể sử dụng thông

số sự kiện `KeyPressEventArgs` để hủy bỏ cú nhấn phím không hợp lệ bằng cách đặt thuộc tính `Handled` là `true`.

Để đầu vào chỉ là số, bạn cần cho phép một cú nhấn phím chỉ khi nó tương ứng với một số (0 đến 9) hoặc một phím điều khiển đặc biệt (như phím delete hoặc mũi tên). Ký tự vừa nhấn được cấp cho sự kiện `KeyPress` thông qua thuộc tính `KeyPressEventArgs.KeyChar`. Bạn có thể sử dụng hai phương thức tĩnh của lớp `System.Char` là `IsDigit` và `IsControl` để kiểm tra nhanh ký tự.

Dưới đây là phương thức thụ lý sự kiện mà bạn sẽ sử dụng để ngăn đầu vào không phải số:

```
private void textBox1_KeyPress(object sender,
    System.Windows.Forms.KeyPressEventArgs e) {

    if (!Char.IsDigit(e.KeyChar) && !Char.IsControl(e.KeyChar)) {
        e.Handled = true;
    }
}
```

Chú ý rằng đoạn mã này bỏ qua dấu phân cách thập phân. Để cho phép ký tự này, bạn cần sửa lại đoạn mã như sau:

```
// Lấy ký tự phân cách thập phân trên nền này
// ("," đối với US-English).
string decimalString =
    Thread.CurrentThread.CurrentCulture.NumberFormat.CurrencyDecimalSeparator;
char decimalChar = Convert.ToChar(decimalString);

if (Char.IsDigit(e.KeyChar) || Char.IsControl(e.KeyChar)) {}
else if (e.KeyChar == decimalString &&
    textBox1.Text.IndexOf(decimalString) == -1) {}
else {
    e.Handled = true;
}
```

Đoạn mã này chỉ cho phép một dấu phân cách thập phân, nhưng nó không giới hạn số chữ số có thể được dùng. Nó cũng không cho phép nhập số âm (bạn có thể thay đổi điều này bằng cách cho phép dấu trừ “-” là ký tự đầu tiên). Nhớ rằng, đoạn mã này cũng giả định bạn đã nhập không gian tên `System.Threading`.

9. *Sử dụng ComboBox có tính năng auto-complete*

?

Bạn cần tạo một `ComboBox` tự động hoàn tất những gì người dùng gõ vào dựa trên danh sách các item của nó.



Bạn có thể hiện thực một ComboBox có tính năng auto-complete bằng cách tạo một điều kiện tùy biến chép đè phương thức OnKeyPress và OnTextChanged.

Có nhiều biến thể khác nhau đối với điều kiện có tính năng auto-complete. Đôi lúc, điều kiện lấp đầy các giá trị dựa trên danh sách các phần vừa chọn (như *Microsoft Excel* thường làm khi bạn nhập giá trị cho cell) hoặc xổ xuống một danh sách các giá trị gần giống (như *Microsoft Internet Explorer* thường làm khi bạn gõ URL). Bạn có thể tạo một ComboBox có tính năng auto-complete bằng cách thụ lý sự kiện KeyPress và TextChanged, hoặc bằng cách tạo một lớp tùy biến dẫn xuất từ ComboBox và chép đè phương thức OnKeyPress và OnTextChanged.

Trong phương thức OnKeyPress, ComboBox xác định có thực hiện một thay thế auto-complete hay không. Nếu người dùng nhấn một phím ký tự (một mẫu tự chẳng hạn) thì việc thay thế có thể được thực hiện, nhưng nếu người dùng nhấn một phím điều khiển (phím backspace hoặc phím mũi tên chẳng hạn) thì không thực hiện gì cả. Phương thức OnTextChanged thực hiện việc thay thế sau khi việc xử lý phím hoàn tất. Phương thức này tìm item trùng khớp đầu tiên đối với phần text hiện thời, rồi thêm vào phần còn lại của text trùng khớp. Sau khi text được thêm vào, ComboBox sẽ chọn (bôi đen) các ký tự giữa điểm chèn hiện tại và điểm cuối của text. Việc này cho phép người dùng tiếp tục gõ và thay thế auto-complete nếu nó không phải là những gì người dùng muốn.

Dưới đây là phần mã cho lớp AutoCompleteComboBox:

```
using System;
using System.Windows.Forms;

public class AutoCompleteComboBox : ComboBox {

    // Biến cờ dùng khi một phím đặc biệt được nhấn
    // (trong trường hợp này, thao tác thay thế text sẽ bị bỏ qua).
    private bool controlKey = false;

    // Xác định xem phím đặc biệt có được nhấn hay không.
    protected override void OnKeyPress(
        System.Windows.Forms.KeyPressEventArgs e) {

        base.OnKeyPress(e);

        if (e.KeyChar == (int)Keys.Escape) {

            // Xóa text.
            this.SelectedIndex = -1;
            this.Text = "";
            controlKey = true;
        }
    }
}
```



```

    }
    else if (Char.IsControl(e.KeyChar)) {

        controlKey = true;
    }
    else {

        controlKey = false;
    }
}

// Thực hiện thay thế text.
protected override void OnTextChanged(System.EventArgs e) {

    base.OnTextChanged(e);

    if (this.Text != "" && !controlKey) {

        // Tìm kiếm item trùng khớp.
        string matchText = this.Text;
        int match = this.FindString(matchText);

        // Nếu tìm thấy thì chèn nó vào.
        if (match != -1) {

            this.SelectedIndex = match;

            // Chọn (bôi đen) phần text vừa thêm vào để
            // nó có thể được thay thế nếu người dùng tiếp tục gõ.
            this.SelectionStart = matchText.Length;
            this.SelectionLength =
                this.Text.Length - this.SelectionStart;
        }
    }
}
}

```

Để thử nghiệm `AutoCompleteComboBox`, bạn có thể tạo một client đơn giản: thêm `ComboBox` vào form và thêm một số từ (word) vào `ComboBox`. Trong ví dụ này, các từ được lấy từ một file text và `ComboBox` được thêm vào form bằng mã lệnh. Bạn cũng có thể biên dịch lớp

`AutoCompleteComboBox` thành một *Class Library Assembly* độc lập rồi thêm nó vào hộp công cụ, thế là bạn có thể thêm nó vào form lúc thiết kế.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.IO;

public class AutoCompleteComboBoxTest : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

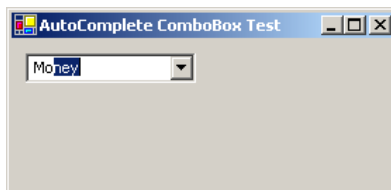
    private void AutoCompleteComboBox_Load(object sender,
        System.EventArgs e) {

        // Thêm ComboBox vào form.
        AutoCompleteComboBox combo = new AutoCompleteComboBox();
        combo.Location = new Point(10,10);
        this.Controls.Add(combo);

        // Thêm một số từ (từ một file text) vào ComboBox.
        FileStream fs = new FileStream("words.txt", FileMode.Open);
        using (StreamReader r = new StreamReader(fs)) {

            while (r.Peek() > -1) {

                string word = r.ReadLine();
                combo.Items.Add(word);
            }
        }
    }
}
```



Hình 6.5 ComboBox có tính năng auto-complete

10.

Sắp xếp ListView theo cột bất kỳ

? Bạn cần sắp xếp một `ListView`, nhưng phương thức nội tại `ListView.Sort` chỉ sắp xếp căn cứ trên cột đầu tiên.

✂ Tạo một hiện thực cho giao diện `System.Collections.IComparer` để có thể sắp xếp các đối tượng `ListViewItem` (kiểu `IComparer` có thể sắp xếp dựa trên bất kỳ tiêu chuẩn nào bạn muốn). Thiết lập thuộc tính `ListView.ListViewItemSorter` với một đối tượng của kiểu `IComparer` trước khi gọi phương thức `ListView.Sort`.

`ListView` cung cấp phương thức `Sort` để sắp các item theo thứ tự alphabet dựa trên phần text trong cột đầu tiên. Nếu muốn sắp xếp dựa trên các giá trị cột khác hoặc sắp thứ tự các item theo bất kỳ cách nào khác, bạn cần tạo một hiện thực tùy biến của giao diện `IComparer`.

Giao diện `IComparer` định nghĩa một phương thức có tên là `Compare`, phương thức này nhận vào hai đối tượng và xác định đối tượng nào sẽ được sắp trước. Lớp tùy biến `ListViewItemComparer` dưới đây hiện thực giao diện `IComparer` và cấp thêm hai thuộc tính: `Column` và `Numeric`. Trong đó, `Column` cho biết cột nào sẽ được sử dụng để sắp xếp; và `Numeric` là một cờ Boolean, được thiết lập là `true` nếu muốn thực hiện việc so sánh theo thứ tự số thay vì so sánh theo thứ tự alphabet.

```
using System;

using System.Collections;

using System.Windows.Forms;

public class ListViewItemComparer : IComparer {

    private int column;
    private bool numeric = false;

    public int Column {

        get {return column;}
        set {column = value;}
    }

    public bool Numeric {

        get {return numeric;}
        set {numeric = value;}
    }
}
```

```
public ListViewItemComparer(int columnIndex) {

    Column = columnIndex;
}

public int Compare(object x, object y) {

    ListViewItem listX = (ListViewItem)x;
    ListViewItem listY = (ListViewItem)y;

    if (Numeric) {

        // Chuyển text thành số trước khi so sánh.
        // Nếu chuyển đổi thất bại, sử dụng giá trị 0.
        decimal listXVal, listYVal;
        try {
            listXVal = Decimal.Parse(listX.SubItems[Column].Text);
        }
        catch {
            listXVal = 0;
        }

        try {
            listYVal = Decimal.Parse(listY.SubItems[Column].Text);
        }
        catch {
            listYVal = 0;
        }

        return Decimal.Compare(listXVal, listYVal);
    }
    else {

        // Giữ nguyên text ở định dạng chuỗi
        // và thực hiện so sánh theo thứ tự alphabetic.
    }
}
```

```

        string listXText = listX.SubItems[Column].Text;
        string listYText = listY.SubItems[Column].Text;

        return String.Compare(listXText, listYText);
    }
}

```

Bây giờ, để sắp xếp `ListView`, bạn cần tạo một đối tượng `ListViewItemComparer`, cấu hình cho nó một cách hợp lý, và rồi thiết lập nó vào thuộc tính `ListView.ListViewItemSorter` trước khi gọi phương thức `ListView.Sort`.

Form dưới đây trình bày một thử nghiệm đơn giản cho `ListViewItemComparer`. Mỗi khi người dùng nhấp vào header của một cột trong `ListView` thì `ListViewItemComparer` sẽ được tạo ra và được sử dụng để sắp xếp danh sách dựa trên cột đó.

```

using System;
using System.Windows.Forms;

public class ListViewItemSort : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    private void ListView1_ColumnClick(object sender,
        System.Windows.Forms.ColumnClickEventArgs e) {

        ListViewItemComparer sorter = new ListViewItemComparer(e.Column);
        ListView1.ListViewItemSorter = sorter;
        ListView1.Sort();
    }
}

```

11.

Liên kết menu ngữ cảnh vào điều kiểm



Bạn cần liên kết một menu ngữ cảnh vào mỗi điều kiểm trên form (các menu này khác nhau). Tuy nhiên, bạn không muốn viết nhiều phương thức thụ lý sự kiện riêng rẽ để hiển thị menu ngữ cảnh cho mỗi điều kiểm.



Viết một phương thức thụ lý sự kiện chung để thu lấy đối tượng `ContextMenu` được kết hợp với điều kiểm, và rồi hiển thị menu này trên điều kiểm.

Bạn có thể liên kết một điều kiểm với một menu ngữ cảnh bằng cách thiết lập thuộc tính `ContextMenu` của điều kiểm. Tuy nhiên, đây chỉ là một thuận lợi—để hiển thị menu ngữ cảnh, bạn phải thu lấy menu và gọi phương thức `Show` của nó. Thông thường, bạn hiện thực logic này trong phương thức thụ lý sự kiện `MouseDown`.

Thực ra, logic dùng để hiển thị menu ngữ cảnh hoàn toàn giống nhau, không quan tâm đến điều kiểm gì. Mọi điều kiểm đều hỗ trợ thuộc tính `ContextMenu` (được thừa kế từ lớp cơ sở `Control`), nghĩa là bạn có thể dễ dàng viết được một phương thức thụ lý sự kiện chung để hiển thị các menu ngữ cảnh cho tất cả các điều kiểm.

Ví dụ, xét một form gồm một `Label`, một `PictureBox`, và một `TextBox`. Bạn có thể viết một phương thức thụ lý sự kiện `MouseDown` cho tất cả các đối tượng này. Đoạn mã dưới đây kết nối tất cả các sự kiện này vào một phương thức thụ lý sự kiện tên là `Control_MouseDown`:

```
this.label1.MouseDown += new MouseEventHandler(this.Control_MouseDown);
this.pictureBox1.MouseDown += new
    MouseEventHandler(this.Control_MouseDown);
this.textBox1.MouseDown += new MouseEventHandler(this.Control_MouseDown);
```

Phần mã thụ lý sự kiện hoàn toàn được dùng chung. Nó chỉ ép kiểu `sender` thành `Control`, kiểm tra menu ngữ cảnh, và hiển thị nó.

```
private void Control_MouseDown(object sender,
    System.Windows.Forms.MouseEventArgs e) {

    if (e.Button == MouseButtons.Right) {

        // Lấy điều kiểm nguồn.
        Control ctrl = (Control)sender;

        if (ctrl.ContextMenu != null) {

            // Hiển thị menu ngữ cảnh.
            ctrl.ContextMenu.Show(ctrl, new Point(e.X, e.Y));

        }

    }
}
```

12.

SỬ DỤNG MỘT PHẦN MENU CHÍNH CHO MENU NGỮ CẢNH

?

Bạn cần tạo một menu ngữ cảnh hiển thị các item giống với một số item trong menu chính của ứng dụng.

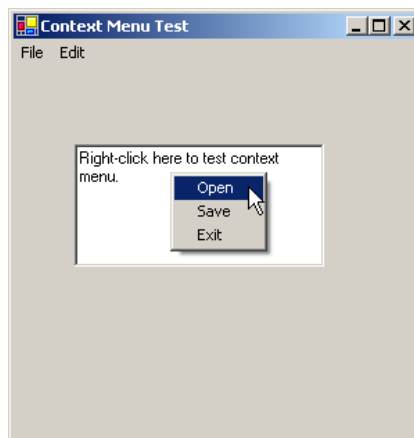
X

Sử dụng phương thức `CloneMenu` của lớp `MenuItem` để sao lại một phần của menu chính.

Trong nhiều ứng dụng, menu ngữ cảnh của một điều kiểm sao lại một phần của menu chính. Tuy nhiên, .NET không cho phép bạn tạo một đối tượng `MenuItem` cùng lúc nằm trong nhiều menu.

Giải pháp là tạo bản sao của một phần menu chính bằng phương thức `CloneMenu`. Phương thức này không chỉ chép các item `MenuItem` (và các submenu), mà còn đăng ký mỗi đối tượng `MenuItem` với cùng phương thức thụ lý sự kiện. Do đó, khi người dùng nhấp vào một item trong menu ngữ cảnh (bản sao), phương thức thụ lý sự kiện tương ứng sẽ được thực thi như thể người dùng nhấp vào item đó trong menu chính.

Ví dụ, xét ứng dụng thử nghiệm trong hình 6.6. Trong ví dụ này, menu ngữ cảnh cho `TextBox` hiển thị các item giống như trong menu *File*. Đây chính là bản sao của các đối tượng `MenuItem`, nhưng khi người dùng nhấp vào một item, phương thức thụ lý sự kiện tương ứng sẽ được thực thi.



Hình 6.6 Chép một phần menu chính vào menu ngữ cảnh

Dưới đây là phần mã cho form để tạo ví dụ này. Nó sẽ sao lại các item trong menu chính khi form được nạp (đáng tiếc là không thể thao tác với các item bản sao lúc thiết kế).

```
using System;
using System.Windows.Forms;
using System.Drawing;

public class ContextMenuCopy : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    private void ContextMenuCopy_Load(object sender,
        System.EventArgs e) {

        ContextMenu mnuContext = new ContextMenu();
```

```
// Chép các item từ menu File vào menu ngữ cảnh.
foreach (MenuItem mnuItem in mnuFile.MenuItems) {

    mnuContext.MenuItems.Add(mnuItem.CloneMenu());
}

// Gắn menu ngữ cảnh vào TextBox.
TextBox1.ContextMenu = mnuContext;
}

private void TextBox1_MouseDown(object sender,
    System.Windows.Forms.MouseEventArgs e) {

    if (e.Button == MouseButtons.Right){

        TextBox1.ContextMenu.Show(TextBox1, new Point(e.X, e.Y));
    }
}

private void mnuOpen_Click(object sender, System.EventArgs e) {

    MessageBox.Show("This is the event handler for Open.");
}

private void mnuSave_Click(object sender, System.EventArgs e) {

    MessageBox.Show("This is the event handler for Save.");
}

private void mnuClick_Click(object sender, System.EventArgs e) {

    MessageBox.Show("This is the event handler for Exit.");
}
}
```


13.

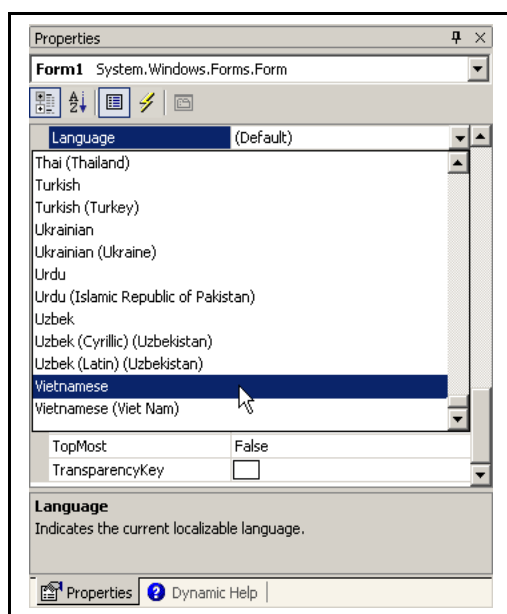
Tạo form đa ngôn ngữ

- ?** Bạn cần tạo một form có thể bản địa hóa (*localizable form*); nghĩa là form này có thể được triển khai ở nhiều ngôn ngữ khác nhau.
- X** Lưu trữ tất cả các thông tin bản địa đặc thù trong các file resource (các file này sẽ được biên dịch thành *Satellite Assembly*).

.NET Framework hỗ trợ sự bản địa hóa (*localization*) thông qua việc sử dụng file resource. Ý tưởng cơ bản là lưu trữ các thông tin bản địa đặc thù (chẳng hạn, phần text của một *Button*) trong một file resource. Sau đó, bạn có thể tạo nhiều file resource cho nhiều bản địa khác nhau rồi biên dịch chúng thành *Satellite Assembly*. Khi chạy ứng dụng, *.NET* sẽ tự động sử dụng đúng *Satellite Assembly* dựa trên các thiết lập bản địa (*locale setting*) của máy tính hiện hành.

Bạn có thể đọc và ghi các file resource bằng mã lệnh. Tuy nhiên, *Visual Studio .NET* cũng hỗ trợ việc thiết kế các form được bản địa hóa:

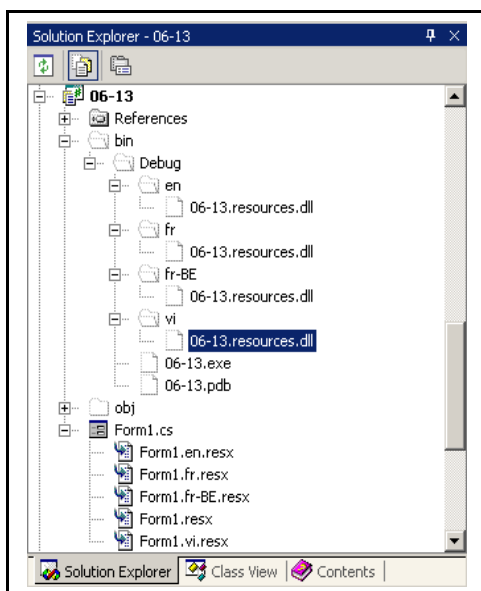
1. Trước tiên, thiết lập thuộc tính *Localizable* của form là *true* trong cửa sổ *Properties*.
2. Thiết lập thuộc tính *Language* của form là bản địa bạn muốn nhập thông tin cho nó (xem hình 6.7). Kế đó, cấu hình các thuộc tính có thể bản địa hóa của tất cả các điều khiển trên form. Thay vì lưu trữ những thay đổi này trong phần mã thiết kế form, *Visual Studio .NET* tạo một file resource mới để lưu trữ dữ liệu của bạn.
3. Lặp lại bước 2 cho mỗi ngôn ngữ bạn muốn hỗ trợ. Mỗi lần như thế, một file resource mới sẽ được tạo ra. Nếu bạn thay đổi thuộc tính *Language* thành bản địa mà bạn đã cấu hình thì các thiết lập trước đó sẽ xuất hiện trở lại, và bạn có thể chỉnh sửa chúng.



Hình 6.7 Chọn một ngôn ngữ để bản địa hóa form

Bây giờ, bạn có thể biên dịch và thử nghiệm ứng dụng trên các hệ thống bản địa khác nhau. *Visual Studio .NET* sẽ tạo một thư mục và một *Satellite Assembly* riêng biệt đối với mỗi file resource trong dự án. Bạn có thể chọn *Project | Show All Files* từ thanh trình đơn của *Visual Studio .NET* để xem các file này được bố trí như thế nào (xem hình 6.8).

Bạn cũng có thể buộc ứng dụng chấp nhận một bản địa cụ thể bằng cách thay đổi thuộc tính `Thread.CurrentUICulture`. Tuy nhiên, bạn phải thay đổi thuộc tính này trước khi form được nạp.



Hình 6.8 Satellite assembly cho bản địa Vietnamese

```
using System;
using System.Windows.Forms;
using System.Threading;
using System.Globalization;

public class MultiLingualForm : System.Windows.Forms.Form {

    private System.Windows.Forms.Label label1;

    // (Bỏ qua phần mã designer.)

    static void Main() {
```

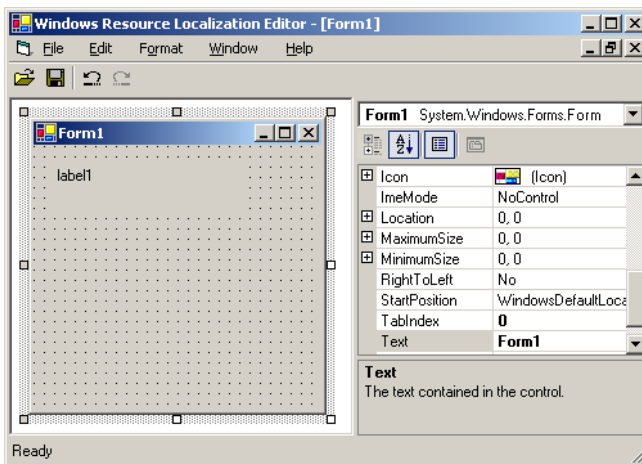
```

Thread.CurrentThread.CurrentUICulture = new CultureInfo("vi");
Application.Run(new MultiLingualForm());
}
}

```



Bạn cũng có thể sử dụng tiện ích *WinRes.exe* (nằm trong thư mục *\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin*) để soạn thảo thông tin resource. Nó cung cấp trình soạn thảo form thu nhỏ nhưng không có chức năng chỉnh sửa mã nguồn, rất hữu dụng cho các nhà phiên dịch và các chuyên gia phi lập trình cần nhập các thông tin bản địa đặc thù.



Hình 6.9 Tiện ích Windows Resource Localization Editor

Ngoài tiện ích trên, bạn cũng có thể sử dụng các chương trình chuyên dùng bản địa hóa các ứng dụng, chẳng hạn *RC-WinTrans* (bạn có thể tải bản dùng thử tại [<http://www.schaudin.com>]). Chương trình này cho phép bạn phát triển các dự án phần mềm đa ngôn ngữ hay bản địa hóa các ứng dụng có sẵn trên nền *Win32*, *.NET*, và *Java*.

14.

Tạo form không thể di chuyển được



Bạn muốn tạo một form chiếm giữ một vị trí cố định trên màn hình và không thể di chuyển được.



Tạo một form không đường viền bằng cách thiết lập thuộc tính `FormBorderStyle` của form là `None`.

Bạn có thể tạo một form không đường viền bằng cách thiết lập thuộc tính `FormBorderStyle` là `None`. Các form này không thể di chuyển được. Và chúng cũng không có kiểu đường viền—nếu muốn có đường viền xanh, bạn phải tự thêm vào bằng cách viết mã hoặc sử dụng hình nền.

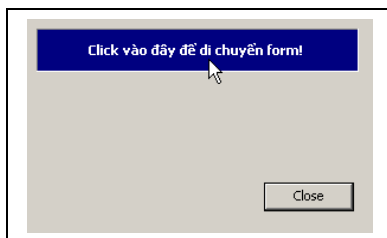
Còn một cách khác để tạo form không thể di chuyển được và có kiểu đường viền giống điều khiển. Trước tiên, thiết lập các thuộc tính `ControlBox`, `MinimizeBox`, và `MaximizeBox` của form là `false`. Kế tiếp, thiết lập thuộc tính `Text` là chuỗi rỗng. Khi đó, form sẽ có đường viền nổi màu xám hoặc đường kẻ màu đen (tùy thuộc vào tùy chọn `FormBorderStyle` mà bạn sử dụng), tương tự như `Button`.

15. *Làm cho form không đường viền có thể di chuyển được*

? Bạn muốn tạo một form không có đường viền nhưng vẫn có thể di chuyển được. Điều này có thể gặp trong trường hợp bạn cần tạo một cửa sổ tùy biến có hình dáng “độc nhất vô nhị” (ví dụ, các ứng dụng game hoặc media player).

✕ Tạo một điều kiện đáp ứng cho các sự kiện `MouseDown`, `MouseUp`, và `MouseMove`; và viết mã để di chuyển form.

Người dùng thường sử dụng thanh tiêu đề để di chuyển form. Tuy nhiên, form không có đường viền cũng không có thanh tiêu đề. Bạn có thể bù vào thiếu hụt này bằng cách thêm một điều khiển vào form để phục vụ cùng mục đích. Ví dụ, form trong hình 6.10 chứa một `Label` hỗ trợ việc kéo rê. Người dùng có thể nhấp vào `Label` này, và rồi kéo rê form đến một vị trí khác trên màn hình trong lúc giữ chuột. Khi người dùng di chuyển chuột, form tự động được di chuyển tương ứng (form được “gắn” với con trỏ chuột).



Hình 6.10 Form không có đường viền nhưng vẫn có thể di chuyển được

Để hiện thực giải pháp này, bạn cần thực hiện các bước sau:

1. Tạo một biến cờ mức-form dùng để theo vết form (form hiện có được kéo rê hay không).
2. Khi người dùng nhấp vào `Label`, cờ sẽ được thiết lập để cho biết form đang ở chế độ kéo rê. Cùng lúc này, vị trí hiện thời của chuột được ghi lại. Bạn cần thêm logic này vào phương thức thụ lý sự kiện `Label.MouseDown`.
3. Khi người dùng di chuyển chuột trên `Label`, form được di chuyển tương ứng để vị trí của chuột trên `Label` vẫn không thay đổi. Bạn cần thêm logic này vào phương thức thụ lý sự kiện `Label.MouseMove`.
4. Khi người dùng thả chuột, chế độ kéo rê được chuyển thành off. Bạn cần thêm logic này vào phương thức thụ lý sự kiện `Label.MouseUp`.

Dưới đây là phần mã hoàn chỉnh cho form:

```
using System;
using System.Windows.Forms;
using System.Drawing;
public class DragForm : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    // Biến cờ dùng để theo vết form.
    // Nếu đang ở chế độ kéo rê, việc di chuyển chuột
    // trên Label sẽ được chuyển thành việc di chuyển form.
    private bool dragging;

    // Lưu trữ offset (vị trí được nhấp vào trên Label).
    private Point pointClicked;

    private void lblDrag_MouseDown(object sender,
        System.Windows.Forms MouseEventArgs e) {

        if (e.Button == MouseButtons.Left) {

            dragging = true;
            pointClicked = new Point(e.X, e.Y);
        }
        else {

            dragging = false;
        }
    }

    private void lblDrag_MouseMove(object sender,
        System.Windows.Forms MouseEventArgs e) {

        if (dragging) {

            Point pointMoveTo;

            // Tìm vị trí hiện tại của chuột trong tọa độ màn hình.
            pointMoveTo = this.PointToScreen(new Point(e.X, e.Y));
```

```

        pointMoveTo.Offset(-pointClicked.X, -pointClicked.Y);

        // Di chuyển form.
        this.Location = pointMoveTo;
    }
}

private void lblDrag_MouseUp(object sender,
    System.Windows.Forms.MouseEventArgs e) {

    dragging = false;
}

private void cmdClose_Click(object sender, System.EventArgs e) {

    this.Close();
}
}

```

16.

Tạo một icon động trong khay hệ thống

- ?** Bạn cần tạo một icon động trong khay hệ thống (chẳng hạn, cho biết tình trạng của một tác vụ đang chạy).
- X** Tạo và hiển thị `NotifyIcon`. Sử dụng một `Timer`, `Timer` này sẽ phát sinh một cách định kỳ (mỗi giây chẳng hạn) và cập nhật thuộc tính `NotifyIcon.Icon`.

Với *.NET Framework* thì rất dễ dàng để hiển thị một icon trong khay hệ thống bằng `NotifyIcon`. Bạn chỉ cần thêm điều kiểm này vào form, cung cấp hình icon bằng thuộc tính `Icon`. Bạn cũng có thể thêm một menu ngữ cảnh vào điều kiểm này bằng thuộc tính `ContextMenu` (tùy chọn). Không giống với các điều kiểm khác, `NotifyIcon` sẽ tự động hiển thị menu ngữ cảnh khi nó được nhấp phải.

Bạn có thể làm động icon trong khay hệ thống bằng cách thay đổi icon định kỳ. Ví dụ, chương trình sau sử dụng tám icon, thể hiện hình mặt trăng từ khuyết đến đầy. Bằng cách dịch chuyển từ hình này sang hình khác, ảo giác về hình động sẽ được tạo ra.

```

using System;
using System.Windows.Forms;
using System.Drawing;

```

```

public class AnimatedSystemTrayIcon : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)
    Icon[] images;
    int offset = 0;

    private void Form1_Load(object sender, System.EventArgs e) {

        // Nạp vào tám icon.
        images = new Icon[8];
        images[0] = new Icon("moon01.ico");
        images[1] = new Icon("moon02.ico");
        images[2] = new Icon("moon03.ico");
        images[3] = new Icon("moon04.ico");
        images[4] = new Icon("moon05.ico");
        images[5] = new Icon("moon06.ico");
        images[6] = new Icon("moon07.ico");
        images[7] = new Icon("moon08.ico");
    }

    private void timer_Elapsed(object sender,
        System.Timers.ElapsedEventArgs e) {

        // Thay đổi icon.
        // Phương thức thụ lý sự kiện này phát sinh mỗi giây một lần.
        notifyIcon.Icon = images[offset];
        offset++;
        if (offset > 7) offset = 0;
    }
}

```

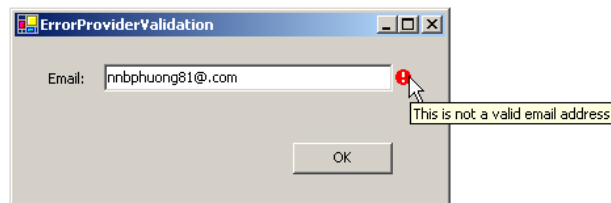
17. *Xác nhận tính hợp lệ của đầu vào cho một điều kiểm*

- ? **Bạn cần cảnh báo cho người dùng khi có dữ liệu không hợp lệ được nhập vào một điều kiểm (như `TextBox`).**
- ✕ **Sử dụng `ErrorProvider` để hiển thị icon lỗi kế bên điều kiểm có lỗi. Kiểm tra lỗi trước khi cho phép người dùng tiếp tục.**

Có một số cách để bạn có thể thực hiện việc xác nhận tính hợp lệ trong một ứng dụng dựa trên *Windows*. Một cách tiếp cận là đáp ứng các sự kiện điều khiển việc xác nhận tính hợp lệ và không cho người dùng thay đổi focus từ điều kiểm này sang điều kiểm khác nếu lỗi xảy ra. Một cách tiếp cận khác là dựng cờ cho điều kiểm có lỗi theo một cách nào đó để người dùng có thể nhìn thấy tất cả lỗi một lượt. Bạn có thể sử dụng cách tiếp cận này trong *.NET* với điều kiểm *ErrorProvider*.

ErrorProvider là một điều kiểm provider đặc biệt, được sử dụng để hiển thị icon lỗi kế bên điều kiểm có lỗi. Bạn có thể hiển thị icon lỗi kế bên một điều kiểm bằng cách sử dụng phương thức *ErrorProvider.SetError*, và chỉ định điều kiểm thích hợp và một chuỗi thông báo lỗi. *ErrorProvider* sẽ hiển thị icon lỗi một cách tự động ở bên phải điều kiểm. Khi người dùng đưa chuột lên icon lỗi, sẽ xuất hiện thông báo chi tiết (xem hình 6.11).

Chỉ cần thêm *ErrorProvider* vào form, bạn có thể sử dụng nó để hiển thị icon lỗi kế bên một điều kiểm bất kỳ. Để thêm *ErrorProvider*, bạn có thể kéo nó vào khay thành phần (*component tray*) hoặc tạo nó bằng mã. Đoạn mã dưới đây kiểm tra nội dung của *TextBox* mỗi khi một phím được nhấn, xác nhận tính hợp lệ của *TextBox* này bằng một biểu thức chính quy (kiểm tra nội dung trong *TextBox* có tương ứng với một địa chỉ e-mail hợp lệ hay không). Nếu nội dung này không hợp lệ, *ErrorProvider* được sử dụng để hiển thị thông báo lỗi. Nếu nội dung này hợp lệ, thông báo lỗi hiện có trong *ErrorProvider* sẽ bị xóa. Cuối cùng, phương thức thụ lý sự kiện *Click* cho nút *OK* sẽ duyệt qua tất cả các điều kiểm trên form và xác nhận rằng không điều kiểm nào có lỗi trước khi cho phép ứng dụng tiếp tục.



Hình 6.11 Form được xác nhận tính hợp lệ với *ErrorProvider*

```
using System;
using System.Windows.Forms;
using System.Text.RegularExpressions;

public class ErrorProviderValidation : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    private void txtEmail_TextChanged(object sender,
        System.EventArgs e) {
```



```

Regex regex;
regex = new Regex(@"\S+@\S+\.\S+");

Control ctrl = (Control)sender;
if (regex.IsMatch(ctrl.Text)) {
    errProvider.SetError(ctrl, "");
}
else {
    errProvider.SetError(ctrl,
        "This is not a valid e-mail address.");
}
}

private void cmdOK_Click(object sender, System.EventArgs e) {

    string errorText = "";
    bool invalidInput = false;

    foreach (Control ctrl in this.Controls) {

        if (errProvider.GetError(ctrl) != "")
        {
            errorText += "    * " + errProvider.GetError(ctrl) + "\n";
            invalidInput = true;
        }
    }

    if (invalidInput) {

        MessageBox.Show(
            "The form contains the following unresolved errors:\n\n" +
            errorText, "Invalid Input", MessageBoxButtons.OK,
            MessageBoxIcon.Warning);
    }
    else {
        this.Close();
    }
}

```

```

    }
}

```

18.

Thực hiện thao tác kéo-và-thả

- ?** Bạn cần sử dụng tính năng kéo-và-thả để trao đổi thông tin giữa hai điều khiển (cũng có thể trong các cửa sổ hoặc các ứng dụng khác nhau)
- ✖** Khởi động thao tác kéo-và-thả bằng phương thức `DoDragDrop` của lớp `Control`, và đáp ứng cho sự kiện `DragEnter` và `DragDrop`.

Thao tác kéo-và-thả cho phép người dùng chuyển thông tin từ nơi này đến nơi khác bằng cách nhấp vào một item và rê nó đến một vị trí khác. Thao tác kéo-và-thả gồm ba bước cơ bản sau đây:

1. Người dùng nhấp vào điều khiển, giữ chuột, và bắt đầu rê. Nếu điều khiển hỗ trợ tính năng kéo-và-thả, nó sẽ thiết lập riêng một vài thông tin.
2. Người dùng rê chuột lên một điều khiển khác. Nếu điều khiển này chấp nhận kiểu nội dung được rê đến, con trỏ chuột sẽ đổi thành hình mũi tên với trang giấy . Nếu không, con trỏ chuột sẽ đổi thành hình tròn với một vạch thẳng bên trong .
3. Khi người dùng thả chuột, dữ liệu được gửi đến điều khiển, và điều khiển này có thể xử lý nó một cách thích hợp.

Để hỗ trợ tính năng kéo-và-thả, bạn phải thụ lý các sự kiện `DragEnter`, `DragDrop`, và `MouseDown`. Ví dụ này sử dụng hai `TextBox`, đây là đoạn mã gắn các phương thức thụ lý sự kiện mà chúng ta sẽ sử dụng:

```

this.TextBox2.MouseDown += new MouseEventHandler(this.TextBox_MouseDown);
this.TextBox2.DragDrop += new DragEventHandler(this.TextBox_DragDrop);
this.TextBox2.DragEnter += new DragEventHandler(this.TextBox_DragEnter);

this.TextBox1.MouseDown += new MouseEventHandler(this.TextBox_MouseDown);
this.TextBox1.DragDrop += new DragEventHandler(this.TextBox_DragDrop);
this.TextBox1.DragEnter += new DragEventHandler(this.TextBox_DragEnter);

```

Để bắt đầu một thao tác kéo-và-thả, bạn hãy gọi phương thức `DoDragDrop` của điều khiển nguồn. Lúc này, bạn cần cung cấp dữ liệu và chỉ định kiểu hoạt động sẽ được hỗ trợ (chép, di chuyển...). Ví dụ dưới đây sẽ khởi tạo một thao tác kéo-và-thả khi người dùng nhấp vào một `TextBox`:

```

private void TextBox_MouseDown(object sender,
    System.Windows.Forms.MouseEventArgs e) {

    TextBox txt = (TextBox)sender;

```

```
txt.SelectAll();
txt.DoDragDrop(txt.Text, DragDropEffects.Copy);
}
```

Để có thể nhận dữ liệu được rê đến, điều kiện phải có thuộc tính `AllowDrop` là `true`. Điều kiện này sẽ nhận sự kiện `DragEnter` khi chuột rê dữ liệu lên nó. Lúc này, bạn có thể kiểm tra dữ liệu đang được rê đến, quyết định xem điều kiện có thể chấp nhận việc thả hay không, và thiết lập thuộc tính `DragEventArgs.Effect` tương ứng, như được trình bày trong đoạn mã dưới đây:

```
private void TextBox_DragEnter(object sender,
    System.Windows.Forms.DragEventArgs e) {

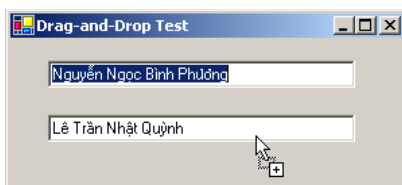
    if (e.Data.GetDataPresent(DataFormats.Text)) {
        e.Effect = DragDropEffects.Copy;
    }
    else {
        e.Effect = DragDropEffects.None;
    }
}
```

Bước cuối cùng là đáp ứng cho sự kiện `DragDrop`, sự kiện này xảy ra khi người dùng thả chuột:

```
private void TextBox_DragDrop(object sender,
    System.Windows.Forms.DragEventArgs e) {

    TextBox txt = (TextBox)sender;
    txt.Text = (string)e.Data.GetData(DataFormats.Text);
}
```

Sử dụng các đoạn mã trên, bạn có thể tạo một ứng dụng thử nghiệm tính năng kéo-và-thả đơn giản (xem hình 6.12), cho phép text được rê từ `TextBox` này đến `TextBox` khác. Bạn cũng có thể rê text từ một ứng dụng khác và thả nó vào một trong hai `TextBox` này.



Hình 6.12 Một ứng dụng thử nghiệm tính năng kéo-và-thả

19.

Sử dụng trợ giúp cảm-ngữ-cảnh

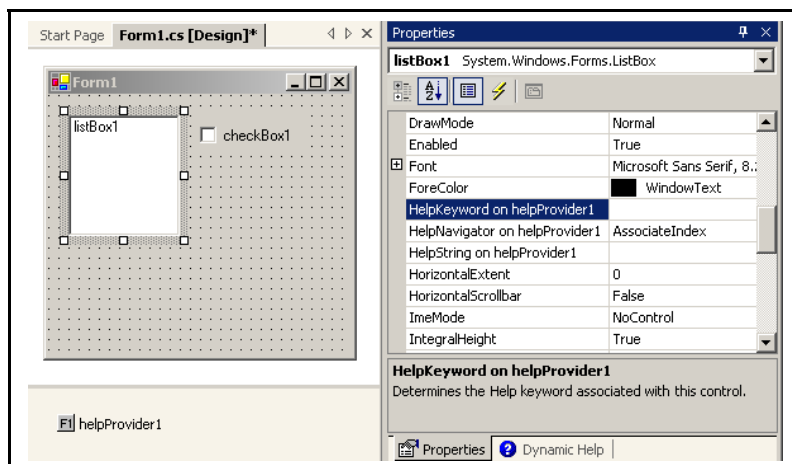
? Bạn muốn hiển thị một chủ đề cụ thể trong file trợ giúp dựa trên điều kiểm hiện đang được chọn.

✂ Sử dụng thành phần `System.Windows.Forms.HelpProvider`, và thiết lập các thuộc tính mở rộng (*extender property*) `HelpKeyword` và `HelpNavigator` cho mỗi điều kiểm.

.NET hỗ trợ tính năng trợ giúp cảm-ngữ-cảnh (*context-sensitive help*) thông qua lớp `HelpProvider`. Lớp này là một điều kiểm mở rộng đặc biệt. Khi bạn thêm nó vào khay thành phần (*component tray*), nó sẽ thêm một số thuộc tính vào tất cả các điều kiểm trên form. Ví dụ, hình 6.13 trình bày một form gồm hai điều kiểm và một `HelpProvider`. `ListBox` (hiện đang được chọn) có thêm các thuộc tính `HelpKeyword`, `HelpNavigator`, và `HelpString` (do `HelpProvider` cấp).

Để sử dụng trợ giúp cảm-ngữ-cảnh với `HelpProvider`, bạn cần thực hiện ba bước sau đây:

1. Thiết lập thuộc tính `HelpProvider.HelpNamespace` là tên của file trợ giúp (chẳng hạn, `myhelp.chm`).
2. Đối với mỗi điều kiểm yêu cầu trợ giúp cảm-ngữ-cảnh, hãy thiết lập thuộc tính mở rộng `HelpNavigator` là `HelpNavigator.Topic`.
3. Đối với mỗi điều kiểm yêu cầu trợ giúp cảm-ngữ-cảnh, hãy thiết lập thuộc tính mở rộng `HelpKeyword` là tên của chủ đề liên kết với điều kiểm này (tên chủ đề phải có trong file trợ giúp và có thể được cấu hình trong các công cụ tạo file trợ giúp).



Hình 6.13 Các thuộc tính mở rộng do `HelpProvider` cấp cho `ListBox`

Nếu người dùng nhấn phím `F1` trong khi một điều kiểm nào đó đang nhận focus, file trợ giúp sẽ được mở một cách tự động và chủ đề liên kết với điều kiểm này sẽ được hiển thị trong cửa

sở trợ giúp. Nếu người dùng nhấn phím *F1* trong khi đang ở trên một điều khiển không có chủ đề trợ giúp (ví dụ, `GroupBox` hoặc `Panel`), các thiết lập trợ giúp cho điều khiển nằm bên trong sẽ được sử dụng. Nếu không có điều khiển nào nằm bên trong hoặc điều khiển nằm bên trong không có thiết lập trợ giúp nào, các thiết lập trợ giúp cho form sẽ được sử dụng. Nếu các thiết lập trợ giúp cho form cũng không có, `HelpProvider` sẽ mở bất kỳ file trợ giúp nào được định nghĩa ở mức dự án. Bạn cũng có thể sử dụng các phương thức của `HelpProvider` để thiết lập hoặc sửa đổi ánh xạ trợ giúp cảm-ngữ-cảnh lúc thực thi.

20.

Áp dụng phong cách Windows XP

? Bạn muốn các điều khiển mang dáng dấp hiện đại của *Windows XP* trên hệ thống *Windows XP*.

X Thiết lập thuộc tính `FlatStyle` là `FlatStyle.System` cho tất cả các điều khiển có hỗ trợ thuộc tính này. Trong *.NET Framework* phiên bản *1.0*, bạn phải tạo một file manifest. Còn trong *.NET Framework* phiên bản *1.1*, bạn chỉ cần gọi phương thức `Application.EnableVisualStyles`.

Phong cách *Windows XP* tự động được áp dụng cho vùng non-client của form (như đường viền, các nút minimize và maximize...). Tuy nhiên, chúng sẽ không được áp dụng cho các điều khiển như `Button` và `GroupBox` trừ khi bạn thực hiện thêm một vài bước nữa.

Trước hết, bạn phải cấu hình tất cả các điều khiển dạng nút trên form (như `Button`, `CheckBox`, và `RadioButton`). Các điều khiển này cung cấp thuộc tính `FlatStyle`, mà thuộc tính này phải được thiết lập là `System`.

Bước kế tiếp tùy thuộc vào phiên bản *.NET* bạn đang sử dụng. Nếu đang sử dụng *.NET Framework* phiên bản *1.1*, bạn chỉ cần gọi phương thức `Application.EnableVisualStyles` trước khi hiển thị form. Ví dụ, bạn có thể khởi tạo ứng dụng với phương thức `Main` như sau:

```
public static void Main() {

    // Kích hoạt visual styles.
    Application.EnableVisualStyles();

    // Hiển thị main form.
    Application.Run(new StartForm)

}
```

Nếu đang sử dụng *.NET Framework* phiên bản *1.0*, bạn không có sự trợ giúp của phương thức `Application.EnableVisualStyles`. Tuy nhiên, bạn vẫn có thể sử dụng phong cách này bằng cách tạo một file manifest cho ứng dụng của bạn. File manifest này (chỉ là một file văn bản thông thường với nội dung *XML*) sẽ báo với *Windows XP* rằng ứng dụng của bạn yêu cầu phiên bản mới của file *comctl32.dll* (file này có trên tất cả các máy tính *Windows XP*). *Windows XP* sẽ đọc và áp dụng các thiết lập từ file manifest một cách tự động, nếu file manifest được đặt trong thư mục ứng dụng và có tên trùng với tên file thực thi ứng dụng cùng

phần mở rộng là *.manifest* (ví dụ, *TheApp.exe* sẽ có file manifest là *TheApp.exe.manifest*—mặc dù nó trông giống có hai phần mở rộng).

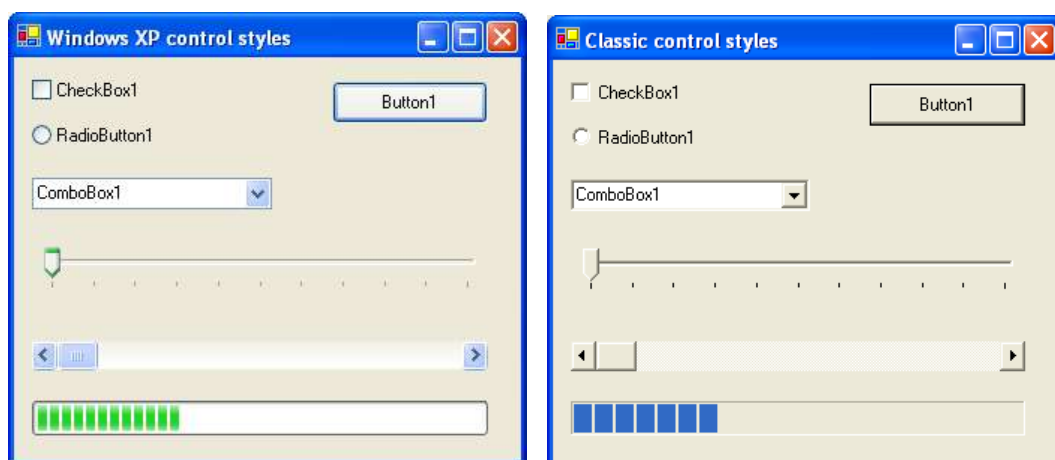
Dưới đây là một file manifest. Bạn có thể chép file này cho các ứng dụng của bạn—chỉ cần đổi tên nó cho phù hợp. Bạn cũng cần đổi giá trị *name* (in đậm) thành tên ứng dụng, mặc dù điều này không thật sự cần thiết.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
    version="1.0.0.0"
    processorArchitecture="X86"
    name="TheApp"
    type="win32" />

<dependency>
<dependentAssembly>
<assemblyIdentity
    type="win32"
    name="Microsoft.Windows.Common-Controls"
    version="6.0.0.0"
    processorArchitecture="X86"
    publicKeyToken="6595b64144ccf1df"
    language="*" />

</dependentAssembly>
</dependency>
</assembly>
```

Phong cách *Windows XP* sẽ không xuất hiện trong môi trường thiết kế của *Visual Studio .NET*. Do đó, để thử nghiệm kỹ thuật này, bạn cần phải chạy ứng dụng. Tuy nhiên, bạn vẫn có thể làm cho môi trường thiết kế của *Visual Studio .NET* hiển thị theo phong cách *Windows XP* bằng cách thêm file *devenv.exe.manifest* vào thư mục *\Program Files\Microsoft Visual Studio .NET 2003\Common7\IDE*.



Hình 6.14 Phong cách Windows XP và phong cách kinh điển

 Nếu bạn áp dụng file manifest cho một ứng dụng đang chạy trên phiên bản *Windows* trước *Windows XP*, nó sẽ bị bỏ qua, và phong cách kinh điển sẽ được sử dụng. Vì lý do này, bạn nên thử nghiệm ứng dụng của bạn cả khi có và không có file manifest.

21.

Thay đổi độ đục của form



Bạn muốn thay đổi độ đục của form để nó trong suốt hơn khi xuất hiện



Thiết lập thuộc tính `Opacity` của form với một giá trị nằm giữa 0% và 100%.

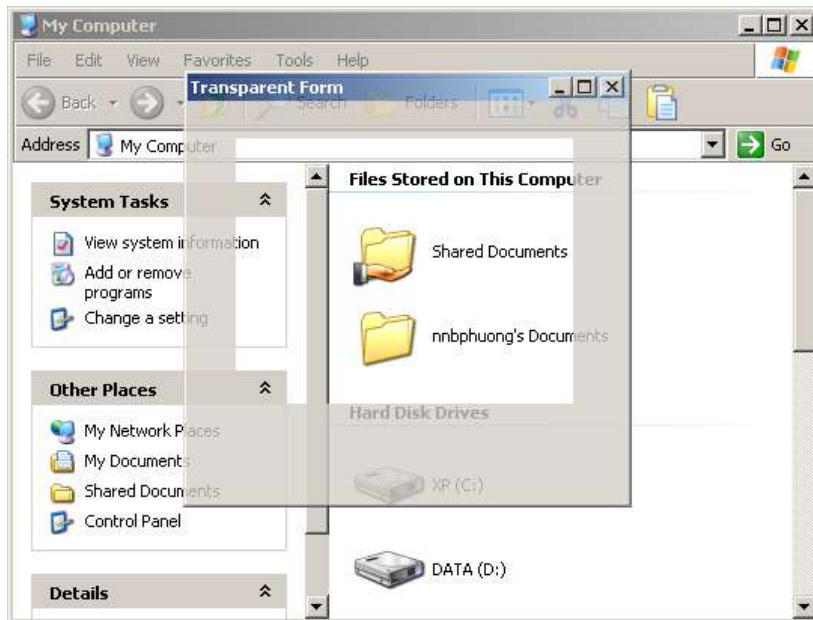
Thuộc tính `Opacity` của một form kiểm soát mức độ đục hay trong của một cửa sổ. Ở mức 100%, form xuất hiện với trạng thái mặc định, nghĩa là không có các vùng trong suốt trên form. Ở mức 0%, form hoàn toàn trong suốt, cũng có nghĩa bạn không thể tương tác với form được nữa.

Khi thay đổi độ đục của một form bằng mã lệnh, bạn phải sử dụng một số thực nằm giữa 0.0 và 1.0:

```
private void Form1_MouseEnter(object sender, System.EventArgs e)
{
    this.Opacity = 1.0;
}

private void Form1_MouseLeave(object sender, System.EventArgs e)
{
    this.Opacity = 0.8;
}
```

Nếu chỉ muốn trong suốt những vùng nào đó trên form, bạn hãy sử dụng thuộc tính `TransparencyKey`. Bạn định nghĩa thuộc tính này là một màu nào đó. Nếu bất kỳ phần nào của form trùng với màu đó, nó sẽ trở nên trong suốt. Hình 6.15 trình bày một form với độ đục 80%. Chúng ta đặt một điều khiển `Panel` lên form và thiết lập màu nền của `Panel` là màu mà ta đã định nghĩa trong thuộc tính `TransparencyKey` của form. Như thế, form sẽ trong suốt trên vùng thuộc `Panel`.



Hình 6.15 Một form với độ đục 80%
và một `Panel` có màu nền giống với thuộc tính `TransparencyKey` của form

Bạn có thể bắt gặp một số ứng dụng dùng hình bitmap làm giao diện người dùng, nhất là các kiểu media player. Bạn có thể tạo kiểu giao diện thế này bằng cách tạo một hình bitmap với những vùng nào đó có màu là màu mà bạn muốn trong suốt. Kế tiếp, thiết lập thuộc tính `BackgroundImage` của form là file bitmap mà bạn đã tạo. Cuối cùng, thiết lập thuộc tính `TransparencyKey` của form là màu mà bạn muốn trong suốt trong hình bitmap.

```
Bitmap Img = (Bitmap) (Bitmap.FromFile("C:\\Example.bmp"));
// Màu tại Pixel(10,10) được sử dụng làm màu trong suốt.
Img.MakeTransparent(Img.GetPixel(10, 10));
this.BackgroundImage = Img;
this.TransparencyKey = Img.GetPixel(10, 10);
```

Bạn cũng có thể gỡ bỏ thanh tiêu đề của form bằng cách thiết lập `FormBorderStyle` là `None` (xem mục 6.14). Để form có thể di chuyển được trong trường hợp này, bạn hãy áp dụng mục

6.15. Trên đây là một cách để tạo form có hình dáng bất thường, một cách khác sẽ được trình bày trong mục 8.3.

7

ASP.NET VÀ WEB FORM

Microsoft ASP.NET là một nền dùng để phát triển các ứng dụng Web, và nó là một phần của Microsoft .NET Framework. ASP.NET cho phép bạn viết dịch vụ Web XML (sẽ được thảo luận trong chương 12) và phát triển website (được thảo luận trong chương này). Các trang ASP.NET sử dụng mô hình điều khiển dựa-trên-sự-kiện, khiến cho việc viết mã cho chúng cũng tương tự như viết mã cho các ứng dụng dựa-trên-Windows thông thường. Tuy nhiên, sự tương tự này có thể là giả tạo. Như hầu hết các nhà phát triển ASP.NET chứng thực, các ứng dụng Web có cách diễn đạt riêng của chúng. Ví dụ, bạn sẽ cần thực hiện thêm các bước để duy trì trạng thái, chuyển thông tin giữa các trang, thụ lý những sự kiện phía client, thực hiện xác thực, và bảo đảm hiệu năng tối ưu khi sử dụng cơ sở dữ liệu. Chương này sẽ xem xét tất cả các vấn đề này.

 **Chương này sẽ không giới thiệu về ASP.NET. Thay vào đó, chương này sẽ giúp những nhà phát triển ASP.NET trung cấp giải quyết những vấn đề thường gặp. Để tìm hiểu căn bản về ASP.NET, hãy vào trang [<http://www.asp.net>] hoặc tham khảo ở các tài liệu khác chuyên về ASP.NET.**

Các mục trong chương này trình bày các vấn đề sau đây:

- Chuyển hướng các yêu cầu của người dùng (mục 7.1).
- Duy trì trạng thái giữa các yêu cầu trang (mục 7.2 và 7.3).
- Sử dụng JavaScript để cải tiến giao diện với các tính năng phía client (mục 7.4, 7.5, và 7.6).
- Cho phép người dùng upload file (mục 7.7).
- Xác thực client theo hai cách: Xác thực tích hợp với Windows (mục 7.8) và xác thực dựa-trên-form (mục 7.9).
- Xác nhận tính hợp lệ của đầu vào mà không sử dụng điều khiển validator của ASP.NET (mục 7.10).
- Tạo động điều khiển web (mục 7.11), hình ảnh (mục 7.12), và điều khiển người dùng (mục 7.13).
- Cải thiện hiệu năng với output-caching (mục 7.14) và data-caching (mục 7.15).
- Giải quyết thông báo lỗi “Unable to start debugging on the Web server” (mục 7.16).
- Thay đổi ngữ cảnh tài khoản Windows mà một ứng dụng ASP.NET chạy trong đó (mục 7.17).

Chương này sử dụng các lớp web cơ bản thuộc không gian tên `System.Web` và các lớp điều khiển web thuộc không gian tên `System.Web.UI.WebControls`. Khi sử dụng các lớp trong các không gian tên này, tên lớp đầy đủ sẽ không được chỉ định.

1.

Chuyển hướng người dùng sang trang khác



Bạn cần chuyển sự thực thi từ một trang ASP.NET sang một trang khác, hoặc bạn muốn chuyển người dùng đến một site hoàn toàn khác.

X Sử dụng phương thức `HttpResponse.Redirect` để chuyển người dùng đến một URL mới, hoặc sử dụng phương thức `HttpServerUtility.Transfer` (nhanh hơn) để chuyển người dùng đến một Web Form ASP.NET khác trên cùng server.

Cách dễ nhất để chuyển người dùng từ một trang này đến một trang khác là sử dụng phương thức `HttpResponse.Redirect` và cấp một URL mới. Bạn có thể truy xuất đối tượng `HttpResponse` hiện tại thông qua đối tượng `HttpContext` hoặc sử dụng thuộc tính `Reponse` của đối tượng `Page` hoặc `Control`. Phương thức thụ lý sự kiện dưới đây (đáp ứng cho một cú nhấp chuột vào `Button`) sẽ chuyển người dùng đến một trang ASP.NET mới:

```
private void cmdRedirect_Click(object sender, System.EventArgs e) {

    Response.Redirect("newpage.aspx");

}
```

Phương thức `Redirect` có thể làm việc với URL tương đối (chỉ đến những tài nguyên trong cùng thư mục ảo), và với URL đầy đủ. URL có thể chỉ đến trang ASP.NET khác, kiểu tài liệu khác (như trang HTML hoặc hình ảnh), và web-server khác.

Phương thức `Redirect` gửi chỉ thị chuyển hướng đến trình duyệt. Kế đó, trình duyệt sẽ yêu cầu trang mới. Kết quả là trình duyệt phải thực hiện hai chuyển đến web-server, và web-server phải xử lý thêm một yêu cầu nữa. Một tùy chọn hiệu quả hơn là sử dụng phương thức `HttpServerUtility.Transfer`, phương thức này sẽ chuyển sự thực thi đến một trang ASP.NET khác trên cùng web-server. Ví dụ:

```
private void cmdRedirect_Click(object sender, System.EventArgs e) {

    Server.Transfer("newpage.aspx");

}
```

Phương thức `Transfer` không cần thêm một chuyển đến client, nhưng nó sẽ không làm việc nếu bạn cần chuyển sự thực thi đến một server khác hoặc một kiểu tài nguyên khác với Web Form (bao gồm trang ASP cổ điển).

2.

Duy trì trạng thái giữa các yêu cầu của trang

?

Bạn cần lưu trữ vài thông tin đặc thù của người dùng giữa các lần postback của trang.

X

Sử dụng *view state* (trạng thái nhìn), *query string argument* (đối số chuỗi truy vấn), *session state* (trạng thái phiên làm việc), hoặc *cookie*, tùy thuộc vào nhu cầu của bạn.

ASP.NET là một mô hình lập trình phi trạng thái (*stateless programming model*). Mỗi khi một postback được phát sinh, mã sẽ nạp vào bộ nhớ, thực thi, và được giải phóng khỏi bộ nhớ. Nếu muốn giữ lại vết của thông tin sau khi mã đã hoàn tất việc xử lý, bạn phải sử dụng các kiểu quản lý trạng thái (*state management*).

ASP.NET cung cấp nhiều cách để lưu trữ thông tin, hay trạng thái, giữa các yêu cầu (*request*). Kiểu trạng thái mà bạn sử dụng cho biết: thông tin sẽ sống bao lâu, sẽ được lưu trữ ở đâu, và sẽ được bảo mật như thế nào. Bảng 7.1 liệt kê những tùy chọn trạng thái khác nhau được cấp bởi ASP.NET. Bảng này không chứa đối tượng *Cache*, đối tượng này cung cấp kho lưu trữ tạm thời và sẽ được mô tả trong mục 7.5.

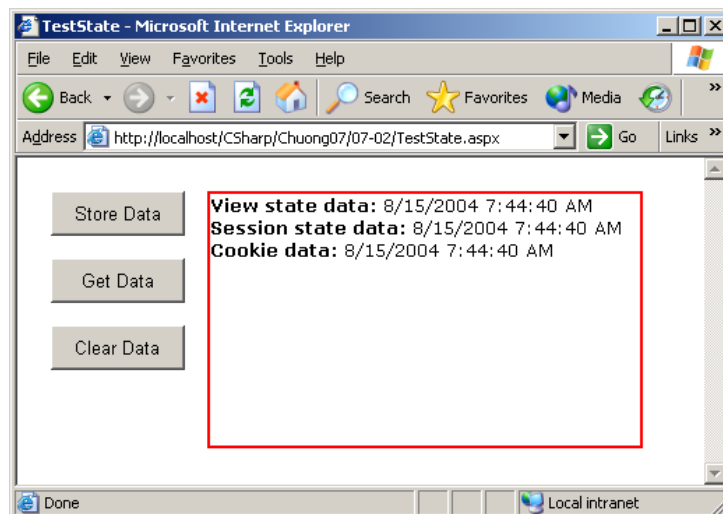
Cú pháp cho các phương pháp lưu trữ dữ liệu là như nhau. Dữ liệu được lưu trữ trong một đối tượng tập hợp và được đánh chỉ mục bằng một tên chuỗi.

Bảng 7.1 Các kiểu quản lý trạng thái

Kiểu trạng thái	Dữ liệu được phép	Vị trí lưu trữ	Thời gian sống	Bảo mật
<i>View state</i>	Tất cả các kiểu dữ liệu khả-tuần-tự-hóa .NET.	Một trường ẩn trong trang web hiện hành.	Bị mất khi người dùng chuyển sang một trang khác.	Mặc định là không an toàn. Tuy nhiên, bạn có thể sử dụng các chỉ thị trang để thực hiện mật hóa và băm để ngăn dữ liệu bị phá.
<i>Query string</i>	Dữ liệu chuỗi.	Chuỗi URL của trình duyệt.	Bị mất khi người dùng nhập một URL mới hoặc đóng trình duyệt. Tuy nhiên, nó có thể được lưu trữ trong một bookmark.	Người dùng có thể nhìn thấy được và chỉnh sửa dễ dàng.
<i>Session state</i>	Tất cả các kiểu dữ liệu khả-tuần-tự-hóa .NET.	Bộ nhớ server (có thể được cấu hình cho một tiến trình hoặc cơ sở dữ liệu bên ngoài).	Hết hiệu lực sau một khoảng thời gian được định nghĩa trước (thường là 20 phút, nhưng khoảng thời gian này có thể thay đổi được).	An toàn vì dữ liệu không bao giờ được chuyển cho client.
<i>Cookie</i>	Dữ liệu chuỗi.	Máy tính client (trong bộ nhớ hoặc một file text, tùy thuộc vào các thiết lập cho thời gian sống của nó).	Được thiết lập bởi lập trình viên. Có thể được sử dụng trong nhiều trang và có thể vẫn còn giữa các lần viếng thăm.	Không an toàn, và có thể bị người dùng chỉnh sửa.

<i>Application state</i>	Tất cả các kiểu dữ liệu khả-tuần-tự-hóa .NET.	Bộ nhớ server.	Thời gian sống của ứng dụng (cho đến khi server được khởi động lại). Không giống với các phương pháp khác, dữ liệu ứng dụng là toàn cục đối với tất cả các người dùng.	An toàn vì dữ liệu không bao giờ được chuyển cho client.
--------------------------	---	----------------	--	--

Hình 7.1 trình bày một trang web thử nghiệm các kiểu quản lý trạng thái khác nhau. Khi người dùng nhấp vào nút *Store Data*, một đối tượng `System.DateTime` sẽ được tạo ra và được lưu trữ trong *view state*, *session state*, và một *cookie* tùy biến. Khi người dùng nhấp vào nút *Get Data*, thông tin này sẽ được lấy ra và hiển thị. Cuối cùng, nút *Clear Data* sẽ xóa thông tin này trong tất cả các trạng thái.



Hình 7.1 Thử nghiệm các kiểu quản lý trạng thái

Dưới đây là phần mã cho trang:

```
using System;
using System.Web;
using System.Web.UI.WebControls;
using System.Web.SessionState;

public class TestState : System.Web.UI.Page {

    protected System.Web.UI.WebControls.Button cmdClear;
    protected System.Web.UI.WebControls.Button cmdStore;
```

```
protected System.Web.UI.WebControls.Button cmdGetData;
protected System.Web.UI.WebControls.Label lblData;

// (Bỏ qua phần mã designer.)

private void cmdStore_Click(object sender, System.EventArgs e) {

    // Tạo đối tượng thử nghiệm.
    DateTime now = DateTime.Now;

    // Lưu trữ đối tượng trong view state.
    ViewState["TestData"] = now;

    // Lưu trữ đối tượng trong session state.
    Session["TestData"] = now;

    // Lưu trữ đối tượng trong một cookie tùy biến.
    // Kiểm tra xem cookie đã tồn tại hay chưa (có tên là 07-02).
    if (Request.Cookies["07-02"] == null) {

        // Tạo cookie.
        HttpCookie cookie = new HttpCookie("07-02");

        // Cookie chỉ có thể lưu trữ dữ liệu chuỗi.
        // Nó có thể lưu trữ nhiều giá trị,
        // mỗi giá trị ứng với một khóa khác nhau.
        cookie["TestData"] = now.ToString();

        // (Bạn có thể chỉnh sửa các thuộc tính
        // của cookie để thay đổi ngày hết hiệu lực.)

        // Gắn cookie vào đáp ứng.
        // Nó sẽ được cung cấp với tất cả các yêu cầu đến
        // site này cho đến khi hết hiệu lực.
        Response.Cookies.Add(cookie);
    }
}
```



```
    }  
}  
  
private void cmdGetData_Click(object sender, System.EventArgs e) {  
  
    lblData.Text = "";  
  
    // Kiểm tra thông tin trong view state.  
    if (ViewState["TestData"] != null) {  
  
        DateTime data = (DateTime)ViewState["TestData"];  
        lblData.Text += "<b>View state data:</b> " +  
            data.ToString() + "<br>";  
  
    }else {  
  
        lblData.Text += "No view state data found.<br>";  
    }  
  
    // Kiểm tra thông tin trong session state.  
    if (Session["TestData"] != null) {  
  
        DateTime data = (DateTime)Session["TestData"];  
        lblData.Text += "<b>Session state data:</b> " +  
            data.ToString() + "<br>";  
  
    }else {  
  
        lblData.Text += "No session data found.<br>";  
    }  
  
    // Kiểm tra thông tin trong cookie tùy biến.  
    HttpCookie cookie = Request.Cookies["07-02"];  
    if (cookie != null) {  
  
        string cookieData = (string)cookie["TestData"];  
        lblData.Text += "<b>Cookie data:</b> " +
```

```

        cookieData + "<br>";

    }else {

        lblData.Text += "No cookie data found.<br>";
    }
}

private void cmdClear_Click(object sender, System.EventArgs e) {

    ViewState["TestData"] = null;
    Session["TestData"] = null;
    // (Bạn cũng có thể sử dụng Session.Abandon để xóa tất cả
    // thông tin trong session state.)

    // Để xóa cookie, bạn phải thay nó thành
    // một cookie đã vượt quá ngày hết hiệu lực.
    HttpCookie cookie = new HttpCookie("07-02");
    cookie.Expires = DateTime.Now.AddDays(-1);
    Response.Cookies.Add(cookie);
}
}

```

Một kiểu trạng thái mà trang này không thể hiện là *query string* (chuỗi truy vấn). *Query string* đòi hỏi một chuyển hướng trang, lý tưởng cho việc chuyển thông tin từ trang này đến trang khác. Để thiết lập thông tin, bạn phải chuyển hướng người dùng đến một trang mới và thêm các đối số *query string* vào cuối *URL*. Bạn có thể sử dụng phương thức `HttpServerUtility.UrlEncode` và `UrlDecode` để bảo đảm dữ liệu chuỗi là *URL* hợp lệ.

```

DateTime now = DateTime.Now;
string data = Server.UrlEncode(now.ToString());
Response.Redirect("newPage.aspx?TestData=" + data);

```

Để lấy thông tin này, bạn có thể sử dụng tập hợp `HttpResponse.QueryString`:

```

// Kiểm tra thông tin trong query string.
if (Request.QueryString["TestData"] != null) {

    string data = Request.QueryString["TestData"];
    data = Server.UrlDecode(data);
}

```

```
lblData.Text += "<b>Found query string data:</b> " + data + "<br>";
}
```

3. *Tạo các biến thành viên có trạng thái cho trang*

? Bạn cần tạo các biến thành viên trong lớp trang và bảo đảm các giá trị của chúng được giữ lại khi trang được post-back.

X Phản ứng với sự kiện `Page.PreRender`, và ghi tất cả các biến thành viên vào *view state*. Phản ứng với sự kiện `Page.Load`, và lấy tất cả các giá trị của các biến thành viên từ *view state*. Phần mã còn lại của bạn giờ đây có thể tương tác với các biến này mà không phải lo lắng các vấn đề về trạng thái.

ASP.NET cung cấp nhiều cơ chế trạng thái, như đã được mô tả trong mục 7.2. Tuy nhiên, bạn không thể sử dụng chúng một cách tự động—tất cả đều đòi hỏi một đoạn mã để đặt thông tin vào và lấy thông tin ra. Bạn có thể thực hiện các công việc này một lần. Khi đó, phần mã còn lại của bạn có thể tương tác trực tiếp với biến thành viên.

Để cách tiếp cận này có thể làm việc được, bạn cần đọc các giá trị của biến vào đầu mỗi postback. Sự kiện `Page.Load` là một chọn lựa lý tưởng cho đoạn mã này vì nó luôn phát sinh trước bất cứ sự kiện điều khiển nào khác. Bạn có thể sử dụng phương thức thụ lý sự kiện `Page.Load` để khởi động tất cả các biến. Ngoài ra, bạn cần lưu trữ tất cả các biến trước khi trang được gửi cho người dùng, sau khi tất cả việc xử lý đã hoàn tất. Trong trường hợp này, bạn có thể đáp ứng cho sự kiện `Page.PreRender`, vì sự kiện này phát sinh sau tất cả các phương thức thụ lý sự kiện khác, chỉ trước khi trang được chuyển thành *HTML* và gửi cho client.

Trang ví dụ sau đây trình bày cách duy trì một biến thành viên của trang có tên là `memberValue`:

```
using System;
using System.Web;
using System.Web.UI.WebControls;

public class StatefulMembers : System.Web.UI.Page {

    // (Bỏ qua phần mã designer.)

    private int memberValue = 0;

    private void Page_Load(object sender, System.EventArgs e) {

        // Nạp lại tất cả các biến thành viên.
        if (ViewState["memberValue"] != null) {
            memberValue = (int)ViewState["memberValue"];
        }
    }
}
```

```

    }
}

private void StatefulMembers_PreRender(object sender,
    System.EventArgs e) {

    // Lưu tất cả các biến thành viên.
    ViewState["memberValue"] = memberValue;

    // Hiển thị giá trị.
    lblCurrent.Text = memberValue.ToString();
}

// (Các phương thức thụ lý sự kiện khác giờ đây
// có thể làm việc trực tiếp với memberValue.)
}

```

4. *Đáp ứng các sự kiện phía client với JavaScript*



Bạn cần thêm mã JavaScript vào một Web Form.



Định nghĩa hàm JavaScript trong một chuỗi, và sử dụng phương thức `Page.RegisterClientScriptBlock` để chèn hàm JavaScript vào trang được trả về. Khi đó, bạn có thể thêm các đặc tính điều khiển để gọi các hàm này.

ASP.NET là một mô hình lập trình đa năng. Đáng tiếc, một khi trang đã được trả về dạng HTML, bạn không thể thực thi bất kỳ mã .NET nào nữa mà không phải phát sinh postback đến server. Hạn chế này làm giảm tính hiệu quả của các trang web có tính tương tác (chẳng hạn, xác nhận tính hợp lệ của đầu vào).

Dĩ nhiên, không có lý do gì khiến bạn không thể trộn chức năng JavaScript phía client vào mã .NET. Mặc dù .NET không chứa bất kỳ giao diện đối tượng nào để tạo JavaScript, nhưng bạn có thể chèn nó vào trang bằng tay. Có một cách để thực hiện việc này là thiết lập đặc tính điều khiển. Ví dụ, TextBox dưới đây sẽ hiển thị một MessageBox khi nó mất focus:

```

TextBox1.Attributes.Add("onBlur",
    "alert('The TextBox has lost focus!');");

```

Thẻ TextBox sẽ được trả về dạng HTML như sau:

```



```

Trong trường hợp này, bạn sử dụng hàm *JavaScript* `alert` (nội tại) và sự kiện *JavaScript* `onBlur` (phát sinh khi một điều kiểm mất focus). Hầu hết các phần tử *HTML* đều hỗ trợ các sự kiện sau đây:

- `onFocus`— xảy ra khi một điều kiểm nhận focus.
- `onBlur`— xảy ra khi một điều kiểm mất focus.
- `onClick`— xảy ra khi người dùng nhấp vào một điều kiểm.
- `onChange`— xảy ra khi người dùng thay đổi giá trị của điều kiểm nào đó.
- `onMouseOver`— xảy ra khi người dùng di chuyển con trỏ chuột trên một điều kiểm.

Một cách khác để chèn mã *JavaScript* là định nghĩa một hàm *JavaScript* trong một biến chuỗi rồi lệnh cho *ASP.NET* chèn nó vào trang web được trả về. Nếu làm theo cách này, bất kỳ điều kiểm nào cũng có thể gọi hàm để đáp ứng cho một sự kiện *JavaScript*.

Ví dụ dưới đây sẽ làm rõ kỹ thuật này bằng một trang web gồm một bảng và một bức hình (xem hình 7.2). Khi người dùng di chuyển chuột lên các ô trong bảng thì hai hàm *JavaScript* tùy biến sẽ được sử dụng: một tạo highlight cho ô hiện tại và một gỡ bỏ highlight khỏi ô trước đó. Ngoài ra, hàm tạo highlight còn thay đổi *URL* của bức hình tùy thuộc vào cột nào đang được chọn. Nếu người dùng đưa chuột lên cột thứ nhất, hình mặt cười sẽ được hiển thị. Nếu người dùng đưa chuột lên cột thứ hai hoặc thứ ba, hình quyển sách với dấu chấm hỏi nhấp nháy sẽ được hiển thị.

```
using System;
using System.Web;
using System.Web.UI.WebControls;

public class JavaScriptTest : System.Web.UI.Page {

    protected System.Web.UI.WebControls.Table Table1;
    protected System.Web.UI.WebControls.Image Image1;

    // (Bỏ qua phần mã designer.)

    private void Page_Load(object sender, System.EventArgs e) {

        // Định nghĩa các hàm JavaScript.
        string highlightScript =
            "<script language=JavaScript> " +
            "function HighlightCell(cell) {" +
            "cell.bgColor = '#C0C0C0';" +
            "if (cell.cellIndex == 0) {" +
            "document.Form1.Image1.src='happy_animation.gif';}" +
            "else {" +
            "document.Form1.Image1.src='question_animation.gif';}" +
```

```

        "};" +
        "</script>";

string unhighlightScript =
    "<script language=JavaScript> " +
    "function UnHighlightCell(cell) {" +
    "cell.bgColor = '#FFFFFF';" +
    "}" +
    "</script>";

// Chèn hàm JavaScript vào trang (nó sẽ xuất hiện
// ngay sau thẻ <form runat=server>).
// Chú ý rằng mỗi khối kịch bản được kết hợp với một tên chuỗi.
// Điều này cho phép nhiều điều kiểm đăng ký cùng khối kịch bản,
// trong khi vẫn bảo đảm nó sẽ được trả về chỉ một lần
// trong trang cuối cùng.
if (!this.IsClientScriptBlockRegistered("Highlight")) {
    Page.RegisterClientScriptBlock("Highlight", highlightScript);
}

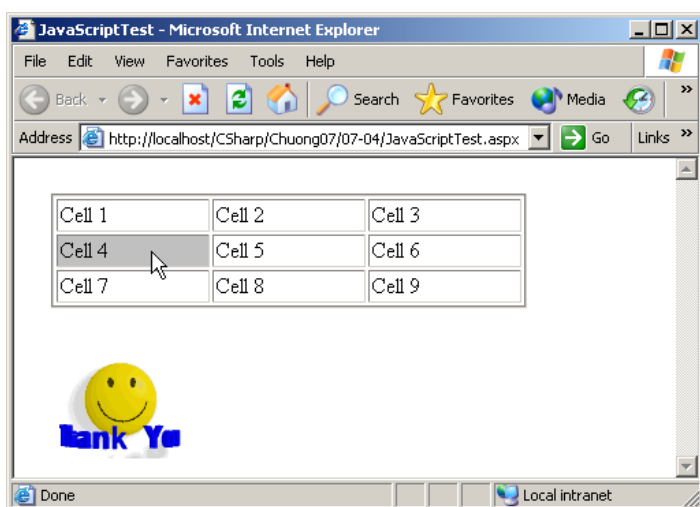
if (!this.IsClientScriptBlockRegistered("UnHighlight")) {
    Page.RegisterClientScriptBlock("UnHighlight",
        unhighlightScript);
}

// Thiết lập đặc tính cho mỗi ô trong bảng.
foreach (TableRow row in Table1.Rows) {

    foreach (TableCell cell in row.Cells) {

        cell.Attributes.Add("onMouseOver", "HighlightCell(this);");
        cell.Attributes.Add("onMouseOut", "UnHighlightCell(this);");
    }
}
}

```



Hình 7.2 Hàm JavaScript tạo highlight cho ô hiện tại



Bạn cần hiểu rõ tính bảo mật của *JavaScript*. Tất cả mã *JavaScript* được trả về trực tiếp trong trang *HTML*, và người dùng có thể xem xét nó. Do đó, bạn đừng bao giờ đặt bất kỳ giải thuật hoặc thông tin bí mật nào vào mã *JavaScript*. Ngoài ra, bạn sử dụng *JavaScript* để kiểm tra tính hợp lệ chỉ là một tiểu xảo, không phải là một phương cách để ngăn các hành động không hợp lệ, bởi người dùng có thể vô hiệu hoặc phá hỏng *JavaScript* trong trình duyệt của họ.

5.

Hiện thị cửa sổ pop-up với JavaScript



Bạn cần hiển thị một cửa sổ pop-up để đáp ứng cho một hành động của người dùng.



Đăng ký một hàm *JavaScript* (hoặc thêm một đặc tính điều khiển sự kiện), và sử dụng hàm *JavaScript* `window.open`.

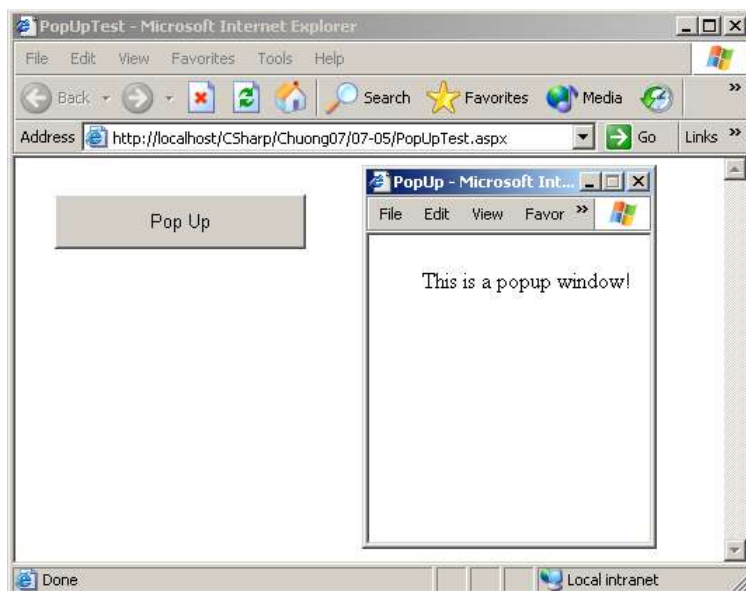
Vì tất cả mã *ASP.NET* thực thi trên server nên không có cách nào để hiển thị một cửa sổ mới từ mã *C#*. Bạn có thể thêm siêu liên kết vào một trang sao cho nó tự động mở trong một cửa sổ mới bằng cách thiết lập đặc tính `target` của thẻ `<a>` là `_blank`, nhưng cách này vẫn không cung cấp khả năng điều khiển kích thước hoặc kiểu của cửa sổ.

Giải pháp để mở cửa sổ pop-up là sử dụng mã *JavaScript*. Cửa sổ pop-up có thể là một trang *HTML*, một trang *ASP.NET*, một file hình, hoặc bất kỳ kiểu tài nguyên nào có thể được mở trong trình duyệt của client. Để mở một cửa sổ pop-up, bạn cần sử dụng hàm `window.open` và chỉ định ba thông số. Thông số thứ nhất là liên kết đến trang mới, thông số thứ hai là tên frame của cửa sổ, và thông số thứ ba là một chuỗi các đặc tính (phân cách bằng dấu phẩy) cấu hình kiểu và kích thước của cửa sổ pop-up. Các đặc tính này bao gồm:

- Đặc tính `height` và `width`—được thiết lập là các giá trị tính theo pixel.

- Đặc tính `toolbar`, `menubar`, và `scrollbars`—có thể được thiết lập là `yes` hay `no`, tùy thuộc vào bạn muốn hiển thị các phần tử này hay không.
- Đặc tính `resizable`—có thể được thiết lập là `yes` hay `no`, tùy thuộc vào bạn muốn đường viền cửa sổ là cố định hay có thể thay đổi kích thước được.

Ví dụ dưới đây trình bày cách mở một trang *ASP.NET* thứ hai trong một cửa sổ pop-up khi người dùng nhấp vào một `Button`.



Hình 7.3 Hiển thị cửa sổ pop-up

Bạn cần thêm đoạn mã này vào phương thức xử lý sự kiện `Page.Load`.

```
string popupScript = "window.open('PopUp.aspx', " +
    "'CustomPopUp', " +
    "'width=200, height=200, menubar=yes, resizable=no')";
```

```
cmdPopUp.Attributes.Add("onClick", popupScript);
```

Và đây là đoạn mã (trong phương thức xử lý sự kiện `Page.Load`) dùng để hiển thị cửa sổ pop-up một cách tự động khi trang được hiển thị:

```
string popupScript = "<script language='javascript'>" +
    "window.open('PopUp.aspx', " +
    "'CustomPopUp', " +
    "'width=200, height=200, menubar=yes, resizable=no') " +
    "</script>";
```



```
Page.RegisterStartupScript("PopupScript", popupScript);
```

6.

Thiết lập focus cho điều kiểm



Bạn cần chỉ định điều kiểm nào sẽ nhận focus khi trang được trả về và gửi cho người dùng.



Tạo một lệnh *JavaScript* để thiết lập focus, và thêm nó vào trang bằng phương thức `Page.RegisterStartupScript`.

Điều kiểm web *ASP.NET* không cung cấp cách để thiết lập focus cho điều kiểm, mà chỉ cung cấp thuộc tính `TabIndex` để thiết lập thứ tự tab. Nhưng thuộc tính này chỉ áp dụng cho *Microsoft Internet Explorer* và không thể được sử dụng để thiết lập focus cho điều kiểm do bạn chọn. Để vượt qua hạn chế này, bạn cần sử dụng đến mã *JavaScript*.

Phương thức dưới đây sẽ thực hiện công việc này. Nó nhận vào một tham chiếu đến bất kỳ đối tượng điều kiểm nào, thu lấy `ClientID` kết giao (mã *JavaScript* phải sử dụng *ID* này để tìm đến điều kiểm), rồi tạo dựng và đăng ký kịch bản để thiết lập focus.

```
private void SetFocus(Control ctrl) {

    // Định nghĩa lệnh JavaScript.
    // Di chuyển focus đến điều kiểm bạn muốn.
    string setFocus = "<script language='javascript'>" +
        "document.getElementById('" + ctrl.ClientID +
        "').focus();</script>";

    // Thêm mã JavaScript vào trang.
    this.RegisterStartupScript("SetFocus", setFocus);
}
```

Nếu thêm phương thức này vào một *Web Form*, bạn có thể gọi `SetFocus` khi cần. Ví dụ dưới đây sẽ thiết lập focus khi trang nạp lần đầu tiên:

```
private void Page_Load(object sender,
    System.EventArgs e) {

    if (!this.IsPostBack) {
        // Chuyển focus đến một TextBox cụ thể.
        SetFocus(TextBox1);
    }
}
```

7.

Cho phép người dùng upload file

?

Bạn cần tạo một trang cho phép người dùng upload một file.

✕

Sử dụng điều khiển `HtmlInputFile`, thiết lập kiểu mã hóa của form là `multipart/form-data`, và lưu file bằng phương thức `HtmlInputFile.PostedFile.SaveAs`.

Vì *ASP.NET* thực thi trên server nên không có cách nào để truy xuất các tài nguyên trên máy client, bao gồm file. Tuy nhiên, bạn có thể sử dụng điều khiển `System.Web.UI.HtmlControls.HtmlInputFile` để cho phép người dùng upload một file. Điều khiển này trả về phần tử *HTML* `<input type="file">`, hiển thị một nút *Browse* và một hộp chứa tên file. Người dùng nhấp vào nút *Browse* và chọn một file. Bước này diễn ra một cách tự động và không đòi hỏi bất kỳ đoạn mã tùy biến nào. Sau đó, người dùng phải nhấp vào một nút khác (bạn phải tạo nút này) để bắt đầu quá trình upload.

Để tạo một trang cho phép upload file, bạn cần thực hiện các bước sau đây:

- Thiết lập kiểu mã hóa của form là `multipart/form-data`. Để thực hiện thay đổi này, tìm thẻ `<form>` trong file `.aspx` và điều chỉnh như sau:

```
<form id="Form1" enctype="multipart/form-data" runat="server">
```

- Thêm điều khiển `HtmlInputFile` vào trang. Trong *Microsoft Visual Studio .NET*, bạn sẽ tìm thấy điều khiển này dưới thẻ *HTML* của hộp công cụ, với tên là *File Field*. Một khi đã thêm điều khiển này, bạn nhấp phải vào nó và chọn *Run As Server Control* để tạo thẻ `<input type="file" runat="server">`.
- Thêm một `Button` khác để bắt đầu quá trình chuyển giao file đã được chỉ định (bằng phương thức `HtmlInputFile.PostedFile.SaveAs`).

Dưới đây là phần mã cho trang upload file:

```
using System;
using System.Web;
using System.Web.UI.WebControls;
using System.IO;

public class UploadPage : System.Web.UI.Page {

    protected System.Web.UI.WebControls.Label lblInfo;
    protected System.Web.UI.WebControls.Button cmdUpload;
    protected System.Web.UI.HtmlControls.HtmlInputFile FileInput;

    // (Bỏ qua phần mã designer.)
```

```
private void cmdUpload_Click(object sender, System.EventArgs e) {

    if (FileInput.PostedFile.FileName == "") {

        // Không file nào được chỉ định.
        lblInfo.Text = "No file specified.";
    } else {

        try {

            if (FileInput.PostedFile.ContentLength > 1048576) {

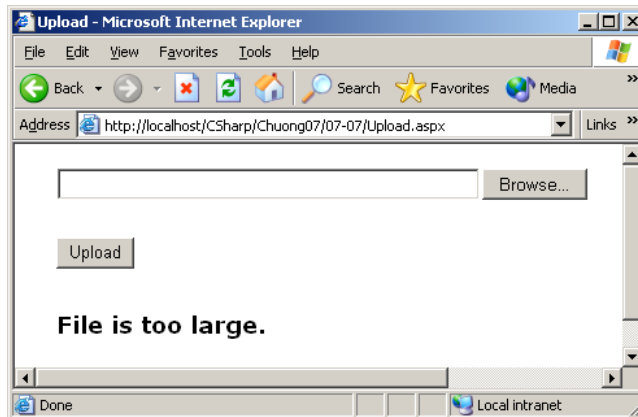
                // Cấm các file lớn hơn 1 megabyte.
                lblInfo.Text = "File is too large.";

            } else {

                // File được lưu vẫn giữ lại tên file gốc của nó.
                string fileName =
                    Path.GetFileName(FileInput.PostedFile.FileName);

                // Tiến trình ASP.NET phải có quyền đối với
                // vị trí nó thực hiện lưu file, nếu không
                // ngoại lệ "access denied" sẽ xảy ra.
                FileInput.PostedFile.SaveAs(fileName);
                lblInfo.Text = "File " + fileName + " uploaded.";
            }
        } catch (Exception err) {

            lblInfo.Text = err.Message;
        }
    }
}
```



Hình 7.4 Trang thử nghiệm upload file

Bạn có thể kiểm tra các thuộc tính của file được chỉ định (bao gồm kích thước của nó) trước khi lưu để ngăn kiểu tấn công từ chối dịch vụ (lừa ứng dụng *ASP.NET* để upload các file lớn làm đầy đĩa cứng). Tuy nhiên, đoạn mã này không ngăn người dùng submit file ngay từ đầu, điều này vẫn có thể làm chậm server và được sử dụng để mở một kiểu tấn công từ chối dịch vụ khác—nắm lấy tất cả các tiêu trình thợ *ASP.NET* đang rảnh. Để ngăn chặn kiểu tấn công này, bạn hãy sử dụng thẻ `<httpRuntime>` trong file *Web.config* để chỉ định kích thước tối đa của file (tính theo kilobyte):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>

    <httpRuntime maxRequestLength="4096" />

    <!-- Bỏ qua các thiết lập khác. -->

  </system.web>
</configuration>
```

Nếu bạn không chỉ định kích thước tối đa, giá trị mặc định 4096 (4 megabyte) sẽ áp dụng. Nếu người dùng cung cấp một file quá lớn, một ngoại lệ sẽ phát sinh ngay khi trang được post-back.



Có một cách khác để gửi file từ client lên web-server là sử dụng dịch vụ *Web XML ASP.NET*. Bạn cần phát triển một ứng dụng dựa-trên-Windows cho phép người dùng chọn một file và rồi giao tiếp với một dịch vụ *Web XML* để chuyển giao thông tin.

8.

Sử dụng IIS authentication



Bạn cần ngăn người dùng truy xuất các trang nào đó trừ khi họ đã được xác thực dựa vào *Windows user account* (tài khoản người dùng *Windows*) trên server.

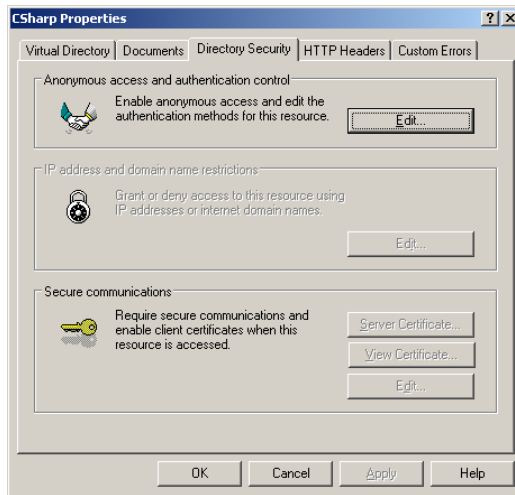


Sử dụng *IIS Windows authentication*. Chọn phương pháp xác thực mà bạn thích, và từ chối truy xuất nặc danh đến thư mục ảo bằng *IIS Manager*. Bạn có thể lấy thông tin về người dùng đã được xác thực từ thuộc tính `Page.User` hoặc lớp `HttpContext`.

IIS và *ASP.NET* sử dụng mô hình bảo mật được sắp lớp (*layered security model*). Khi người dùng yêu cầu một trang *ASP.NET* trên *HTTP*, các bước dưới đây sẽ diễn ra:

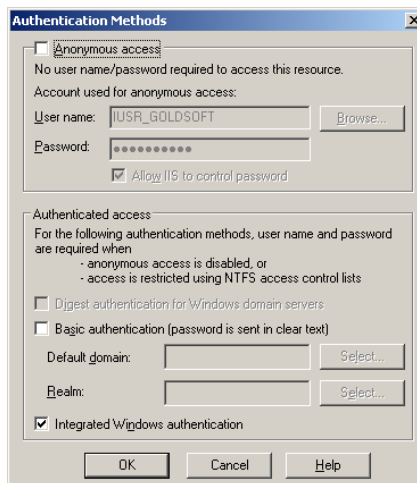
1. *IIS* thực hiện việc xác thực người dùng. Nếu *Anonymous access* được kích hoạt, *IIS* sẽ tự động đăng nhập người dùng với tài khoản nặc danh (*IUSR_[ServerName]*). Nếu không, nó sẽ yêu cầu thông tin xác thực để đăng nhập người dùng với một tài khoản *Windows* khác.
2. Nếu *IIS* xác thực người dùng thành công, nó sẽ chuyển yêu cầu đến *ASP.NET* cùng với thông tin về người dùng đã được xác thực. Theo đó, *ASP.NET* có thể sử dụng các dịch vụ bảo mật của nó dựa vào các thiết lập trong file *Web.config* (ví dụ, không cho người dùng hoặc nhóm cụ thể truy xuất đến các trang hoặc thư mục nào đó). Ngoài ra, mã của bạn có thể hạn chế các hành động bằng cách kiểm tra thông tin về người dùng.
3. Nếu mã *ASP.NET* truy xuất bất kỳ tài nguyên hệ thống nào (ví dụ, mở một file hoặc kết nối đến một cơ sở dữ liệu), hệ điều hành *Windows* sẽ thực hiện những kiểm tra bảo mật của nó. Thông thường, mã ứng dụng *ASP.NET* không thực sự chạy dưới tài khoản của người dùng đã được xác thực. Như thế, những kiểm tra bảo mật này được thực hiện dựa trên tài khoản tiến trình *ASPNET* (được cấu hình bằng file *machine.config*).

Để sử dụng *IIS authentication*, bước đầu tiên là vô hiệu *Anonymous access* cho thư mục ảo. Bạn cần khởi động *IIS Manager* (vào *Start* | *Control Panel* | *Administrative Tools* | *Internet Information Services*). Kế tiếp, nhấp phải vào một thư mục ảo hoặc một thư mục con bên trong thư mục ảo, và chọn *Properties*. Chọn thẻ *Directory Security* (xem hình 7.5).



Hình 7.5 Directory Security

Kế tiếp, nhấp nút *Edit*. Cửa sổ như hình 7.6 sẽ xuất hiện. Trong nửa dưới của cửa sổ, bạn có thể kích hoạt một trong các phương pháp xác thực. Tuy nhiên, không phương pháp nào được sử dụng trừ khi bạn xóa dấu chọn *Anonymous access*.



Hình 7.6 Directory authentication

Bạn có thể kích hoạt nhiều phương pháp xác thực, trong trường hợp này client sẽ sử dụng phương pháp được hỗ trợ mạnh nhất. Nếu *Anonymous access* được kích hoạt thì nó luôn được sử dụng. Các phương pháp xác thực khác nhau được mô tả trong bảng 7.2.

Bảng 7.2 Các kiểu xác thực

Chế độ	Mô tả
<i>Anonymous</i>	Client không cần cung cấp bất kỳ thông tin nào. Người dùng được đăng nhập bằng tài khoản mặc định có sẵn (<i>IUSR_[ServerName]</i>).
<i>Basic</i>	<i>Basic authentication</i> là một phần của chuẩn <i>HTTP 1.0</i> , và nó được hầu hết các trình duyệt và web-server hỗ trợ. Khi sử dụng <i>Basic authentication</i> , trình duyệt yêu cầu người dùng nhập username và password. Thông tin này được chuyển cho <i>IIS</i> , và nó được so trùng với <i>Windows user account</i> cục bộ. <i>Basic authentication</i> nên luôn được phối hợp với <i>SSL</i> vì nó không bảo mật thông tin đăng nhập trước khi chuyển giao.
<i>Digest</i>	<i>Digest authentication</i> gửi một <i>digest</i> (tức mã băm mật mã) trên mạng. Do đó, nó an toàn hơn <i>Basic authentication</i> vì thông tin đăng nhập nếu bị chặn cũng không thể dùng lại được. Bất lợi chủ yếu là <i>Digest authentication</i> chỉ được hỗ trợ trên <i>Internet Explorer 5.0</i> trở lên. Cũng vậy, web-server của bạn cần sử dụng <i>Active Directory</i> hoặc có thể truy xuất đến một <i>Active Directory server</i> .
<i>Integrated</i>	Khi sử dụng <i>Integrated authentication</i> thì <i>Internet Explorer</i> gửi <i>login token</i> cho người dùng hiện hành một cách tự động, miễn là nó ở trên một miền đáng tin cậy. <i>Integrated authentication</i> chỉ được hỗ trợ trên <i>Internet Explorer 2.0</i> trở lên và không thể làm việc trên các proxy-server.

Một khi đã kích hoạt các thiết lập bảo mật cho thư mục ảo thích hợp, bạn cũng nên bảo đảm file *Web.config* được thiết lập là sử dụng *Windows authentication*. Trong một dự án *Visual Studio .NET*, đây là thiết lập mặc định.

```
<configuration>
  <system.web>
    <!-- Bỏ qua các thiết lập khác. -->
    <authentication mode="Windows" />
  </system.web>
</configuration>
```

Vào lúc này, thư mục ảo của bạn sẽ yêu cầu xác thực người dùng và ứng dụng *Web* sẽ có thể lấy thông tin về người dùng. Ngoài ra, bạn có thể thêm các quy tắc phân quyền (*authorization rule*) để ngăn người dùng hoặc nhóm nào đó truy xuất các trang web hoặc thư mục con. Bạn thực hiện điều này bằng cách thêm thẻ `<allow>` và `<deny>` vào phần `<authorization>` của file *Web.config*. Ví dụ, bạn có thể tạo một thư mục con với nội dung file *Web.config* như sau:

```
<configuration>
  <system.web>

    <authorization>
      <deny roles="Guest,Associate" />
      <allow users="nnbphuong81" />
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

```

    </authorization>
</system.web>

```

```
</configuration>
```

ASP.NET xét qua các quy tắc phân quyền theo thứ tự chúng xuất hiện và dừng khi tìm thấy một trùng khớp. Trong ví dụ này, người dùng trong các nhóm *Guest* hoặc *Associate* sẽ tự động bị từ chối. Người dùng *nnpbphuong81* sẽ được phép (trừ khi anh ta là thành viên của một trong hai nhóm bị cấm ở trên). Tất cả các người dùng khác sẽ bị từ chối. Trong trường hợp này, đây là các nhóm và tài khoản người dùng cục bộ. Nếu muốn nói đến một tài khoản miền, bạn hãy sử dụng cú pháp `[DomainName]\[UserName]`.

Đề ý trong ví dụ này, file *Web.config* không chứa phần `<authentication>`. Đó là vì phần này đã được cấu hình trong file *Web.config* thuộc thư mục ứng dụng *Web*. Các thư mục con có thể thiết lập các quy tắc phân quyền của chúng, nhưng chúng không thể thay đổi chế độ xác thực.

Một tùy chọn khác không cho truy xuất đến các trang cụ thể là sử dụng đặc tính `<location>`:

```

<configuration>
  <system.web>
    <!-- Bỏ qua các thiết lập khác. -->
  </system.web>

  <location path="SecurePage.aspx">
    <system.web>
      <authorization>
        <deny roles="Guest" />
      </authorization>
    </system.web>
  </location>

```

```
</configuration>
```

Cuối cùng, bạn có thể viết logic xác thực bằng cách kiểm tra identity của người dùng trong phần mã trang web (sử dụng thuộc tính `Page.User`, thuộc tính này cung cấp đối tượng `WindowsPrincipal`). Bạn có thể lấy tên người dùng từ thuộc tính `WindowsPrincipal.Identity.Name`, và bạn có thể kiểm tra vai trò thành viên trong nhóm bằng phương thức `WindowsPrincipal.IsInRole`. Phần mã cho trang web dưới đây sẽ trình bày các kỹ thuật này:

```

using System;
using System.Web;
using System.Web.UI.WebControls;

```



```

using System.Web.Security.Principal;

public class WindowsSecurityTest : System.Web.UI.Page {

    protected System.Web.UI.WebControls.Label lblMessage;

    // (Bỏ qua phần mã designer.)

    private void Page_Load(object sender, System.EventArgs e) {

        // Thu lấy identity đã được IIS xác thực.
        WindowsIdentity identity = (WindowsIdentity)User.Identity;

        // Kiểm tra xem nó có phải là một Administrator hay không.
        bool isAdmin = User.IsInRole(@"BUILTIN\Administrators");

        // Hiển thị một vài thông tin về identity.
        lblMessage.Text = "You have reached the secured page, " +
            User.Identity.Name + "." +
            "<br><br>Authentication Type: " +
            identity.AuthenticationType.ToString() +
            "<br>Anonymous: " + identity.IsAnonymous.ToString() +
            "<br>Authenticated: " + identity.IsAuthenticated.ToString() +
            "<br>Guest: " + identity.IsGuest.ToString() +
            "<br>System: " + identity.IsSystem.ToString() +
            "<br>Administrator: " + isAdmin.ToString();
    }
}

```

9.

Sử dụng Forms authentication



Bạn cần ngăn người dùng truy xuất các trang nào đó trừ khi họ tự xác thực trước với một trang đăng nhập tùy biến.



Hiện thực *Forms authentication* bằng cách cấu hình thẻ <authentication> trong file *Web.config* của ứng dụng. Bạn phải tạo trang đăng nhập, nhưng *ASP.NET* sẽ giữ lại trạng thái xác thực của người dùng.

Forms authentication là một mô hình bảo mật linh hoạt, cho phép ngăn người dùng chưa được xác thực truy xuất vào trang nào đó. Bạn cần viết mã để thực hiện xác thực, và *ASP.NET* cấp

một *cookie* cho người dùng đã được xác thực. Người dùng không có *cookie* sẽ bị chuyển hướng sang trang đăng nhập khi truy xuất một trang được bảo vệ.

Để hiện thực *Forms authentication*, bạn phải thực hiện các bước sau đây:

- Cấu hình *Forms authentication* bằng thẻ `<authentication>` trong file *Web.config* của ứng dụng.
- Sử dụng các thiết lập trong file *Web.config* để hạn chế người dùng nặc danh truy xuất vào một trang hoặc thư mục cụ thể.
- Tạo trang đăng nhập, và thêm logic xác thực của bạn vào (sử dụng lớp *FormsAuthentication* thuộc không gian tên *System.Web.Security*).

Bước đầu tiên là cấu hình file *Web.config* trong thư mục gốc của ứng dụng để kích hoạt *Forms authentication*, như được trình bày trong đoạn mã dưới đây. Bạn cũng cần chỉ định trang đăng nhập tùy biến (người dùng chưa được xác thực sẽ bị chuyển hướng sang trang này) và thời gian trễ (sau thời gian này, *cookie* sẽ bị loại bỏ). *Authentication cookie* tự động được làm mới với mỗi yêu cầu web.

```
<configuration>
  <system.web>

    <!-- Bỏ qua các thiết lập khác. -->

    <authentication mode="Forms">
      <forms loginUrl="login.aspx" timeout="30" />
    </authentication>

  </system.web>
</configuration>
```

Kế đến, bạn cần thêm quy tắc phân quyền để từ chối người dùng nặc danh. Cách dễ nhất để bảo toàn các trang là tạo một thư mục con cùng với file *Web.config* của nó. File *Web.config* này sẽ từ chối việc truy xuất của các người dùng nặc danh, như được trình bày dưới đây:

```
<configuration>
  <system.web>

    <authorization>
      <deny users="?" />
    </authorization>

    <!-- Bỏ qua các thiết lập khác. -->

  </system.web>
</configuration>
```

Bây giờ, *ASP.NET* sẽ tự động gửi các yêu cầu chưa được xác thực (đối với các trang trong thư mục con) đến trang đăng nhập tùy biến.

Một tùy chọn khác không cho truy xuất đến các trang cụ thể trong thư mục hiện thời là sử dụng thẻ `<location>`:

```
<configuration>
  <system.web>
    <!-- Bỏ qua các thiết lập khác. -->
  </system.web>

  <location path="SecurePage.aspx">
    <system.web>
      <authorization>
        <deny users="?" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

Bạn cũng có thể từ chối các người dùng cụ thể bằng cách nhập danh sách tên người dùng (phân cách bằng dấu phẩy) thay vào ký tự “?” (“?” nghĩa là “tất cả các người dùng nặc danh”). Kế tiếp, bạn cần tạo trang đăng nhập. Trang đăng nhập có thể xác thực người dùng bằng password được viết cứng (phù hợp cho các thử nghiệm đơn giản), cơ sở dữ liệu phía server, hoặc bất kỳ kiểu logic xác thực tùy biến nào. Một khi người dùng đã được xác thực thành công, bạn cần gọi phương thức tĩnh `FormsAuthentication.RedirectFromLoginPage`. Phương thức này sẽ thiết lập *authentication cookie* và chuyển hướng người dùng đến trang được yêu cầu lúc đầu.

Dưới đây là trang đăng nhập sơ bộ, chỉ kiểm tra một password cụ thể khi người dùng nhấp vào nút *Login*:

```
using System;
using System.Web;
using System.Web.UI.WebControls;
using System.Web.Security;

public class LoginPage : System.Web.UI.Page {

    protected System.Web.UI.WebControls.Label lblStatus;
    protected System.Web.UI.WebControls.Button cmdLogin;
    protected System.Web.UI.WebControls.TextBox txtPassword;
    protected System.Web.UI.WebControls.TextBox txtName;
```

```
// (Bỏ qua phần mã designer.)

private void cmdLogin_Click(object sender, System.EventArgs e) {

    if (txtPassword.Text.ToLower() == "secret") {
        FormsAuthentication.RedirectFromLoginPage(
            txtName.Text, false);
    }
    else {
        lblStatus.Text = "Try again.";
    }
}
}
```

Để thử nghiệm trang đăng nhập trên, bạn hãy yêu cầu trang *SecurePage.aspx* (nằm trong thư mục *Secured*). Bạn sẽ bị chuyển hướng sang trang *login.aspx*, và nếu nhập đúng password, bạn sẽ được trả về trang *SecurePage.aspx*.



Hình 7.7 Trang đăng nhập tùy biến

10.

Thực hiện xác nhận tính hợp lệ có-chọn-lựa

?

Bạn cần sử dụng các điều kiểm validator của *ASP.NET*. Tuy nhiên, bạn muốn kiểm tra bằng mã lệnh để có thể xác nhận tính hợp lệ chỉ các điều kiểm hay tập các điều kiểm nào đó, hoặc có thể tùy biến các thông báo lỗi dựa trên đầu vào không hợp lệ.

X

Vô hiệu thuộc tính `EnableClientScript` của mọi điều kiểm validator để trang có thể được post-back. Kế đó, sử dụng phương thức `Page.Validate` để xác nhận tính

hợp lệ của trang hoặc phương thức `BaseValidator.Validate` để xác nhận tính hợp lệ của từng điều kiểm riêng rẽ.

Điều kiểm validator của *ASP.NET* là giải pháp lý tưởng để xác nhận tính hợp lệ của form một cách nhanh chóng. Với điều kiểm validator, bạn có thể xác nhận tính hợp lệ của toàn bộ trang cùng một lúc. Nếu muốn xác nhận tính hợp lệ chỉ một phần form, hoặc muốn quyết định xem có cần xác định tính hợp lệ một điều kiểm nào đó hay không (dựa trên giá trị của một điều kiểm khác chẳng hạn), bạn sẽ cần thực hiện thao tác xác nhận tính hợp lệ có-chọn-lựa.

Bước đầu tiên trong thao tác này là vô hiệu thuộc tính `EnableClientScript` của mọi điều kiểm validator trên trang. Nếu không, việc kiểm tra sẽ được thực hiện tại client thông qua *JavaScript*, trang sẽ không được post-back nếu nó chứa các giá trị không hợp lệ, và phần mã thụ lý sự kiện sẽ không được thực thi. Một khi đã thực hiện thay đổi này, bạn có thể xác nhận tính hợp lệ từng điều kiểm một bằng phương thức `BaseValidator.Validate`, hoặc xác nhận tính hợp lệ toàn bộ trang bằng phương thức `Page.Validate`.

Ví dụ dưới đây thực hiện kiểm tra phía server với hai validator: `RangeValidator` và `RegularExpressionValidator` (xác nhận tính hợp lệ một địa chỉ e-mail). Nếu kiểm tra thất bại, đoạn mã này sẽ duyệt qua tập hợp các validator trên form bằng thuộc tính `Page.Validators`. Mỗi khi tìm thấy một validator có lỗi, nó sẽ tìm điều kiểm tương ứng bằng phương thức `Page.FindControl` rồi hiển thị giá trị lỗi.

```
using System;
using System.Web;
using System.Web.UI.WebControls;

public class SelectiveValidation : System.Web.UI.Page {

    protected System.Web.UI.WebControls.TextBox txtNumber;
    protected System.Web.UI.WebControls.TextBox txtEmail;
    protected System.Web.UI.WebControls.Label lblCustomSummary;
    protected System.Web.UI.WebControls.RegularExpressionValidator
        validatorEmail;
    protected System.Web.UI.WebControls.RangeValidator validatorNumber;
    protected System.Web.UI.WebControls.Button cmdValidate;

    // (Bỏ qua phần mã designer.)

    private void cmdValidate_Click(object sender, System.EventArgs e) {

        // Xác nhận tính hợp lệ của trang.
        this.Validate();
    }
}
```

```
if (!Page.IsValid) {

    lblCustomSummary.Text = "";
    foreach (BaseValidator validator in this.Validators) {

        if (!validator.IsValid) {

            TextBox invalidControl = (TextBox)
                this.FindControl(validator.ControlToValidate);

            lblCustomSummary.Text +=
                "The page contains the following error: <b>" +
                validator.ErrorMessage + "</b>.<br>" +
                "The invalid input is: <b>" +
                invalidControl.Text + "</b>." + "<br><br>";

        }

    }

}

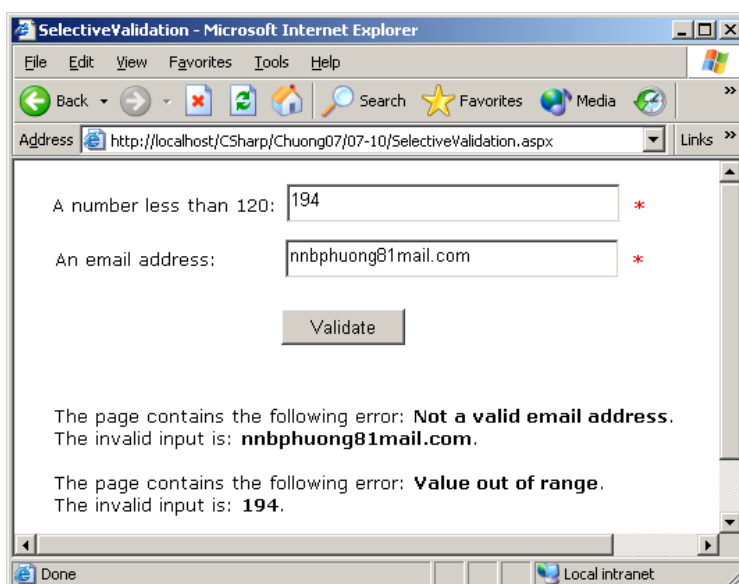
}else {

    lblCustomSummary.Text = "Validation succeeded.";

}

}

}
```



Hình 7.8 Thực hiện thao tác xác nhận tính hợp lệ tùy biến

11.

Thêm động điều kiểm vào Web Form

- ?** Bạn cần thêm một điều kiểm web vào một trang web lúc thực thi và thụ lý các sự kiện của nó.
- X** Tạo một đối tượng điều kiểm, thêm nó vào tập hợp Controls của một điều kiểm container, và sử dụng lệnh `AddHandler` để kết nối bất kỳ phương thức thụ lý sự kiện nào. Bạn phải tạo điều kiểm sau mỗi lần postback.

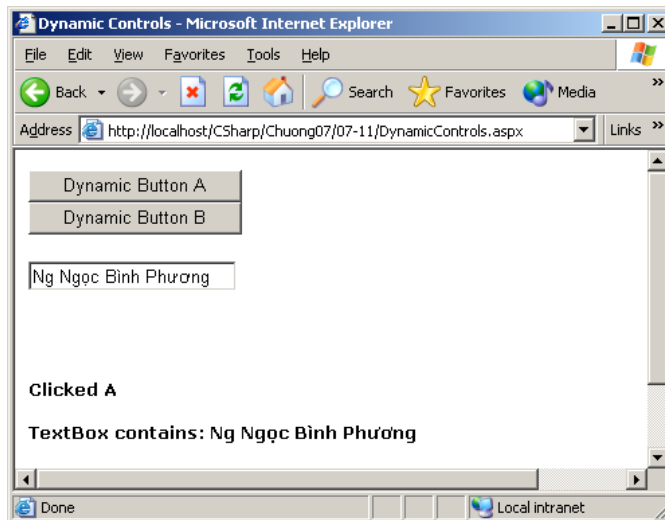
Kỹ thuật thêm điều kiểm web vào trang web tương tự như kỹ thuật thêm điều kiểm *Windows* vào form, nhưng có một vài điểm khác, bao gồm:

- Các điều kiểm được-tạo-động chỉ tồn tại đến lần postback kế tiếp. Nếu cần chúng, bạn phải tái tạo lại khi trang được trả về. Tuy nhiên, đòi hỏi này không ngăn bạn viết mã để thụ lý các sự kiện của chúng.
- Việc định vị các điều kiểm được-tạo-động không mấy dễ dàng. Bạn nên sử dụng điều kiểm trực kiện (*literal control*) chứa mã *HTML* (như `<br>`) để phân cách các điều kiểm được-tạo-động.
- Các điều kiểm được-tạo-động nên được đặt trong một điều kiểm container (như `Panel`) hơn là đặt trực tiếp lên trang. Điều này khiến cho việc định vị chúng dễ dàng hơn.
- Nếu muốn tương tác với điều kiểm sau này, bạn nên cho nó một định danh (*ID*) duy nhất. Bạn có thể sử dụng *ID* này để thu lấy nó từ tập hợp Controls của điều kiểm container.

Nơi tốt nhất để tạo các điều kiểm mới là trong phương thức thụ lý sự kiện `Page.Load` (bảo đảm điều kiểm sẽ được tạo mỗi khi trang được đáp ứng). Ngoài ra, nếu bạn thêm một điều kiểm

nhập (input) sử dụng *view state*, thông tin *view state* sẽ được trả lại cho điều kiểm sau khi sự kiện `Page.Load` phát sinh. Tương tự, vì sự kiện `Page.Load` luôn phát sinh trước khi sự kiện nào khác diễn ra, bạn có thể tái tạo một điều kiểm dựng nên các sự kiện phía server, và phần mã thụ lý sự kiện của nó sẽ diễn ra ngay sau sự kiện `Page.Load`.

Ví dụ dưới đây (xem hình 7.9) sẽ tạo động ba điều kiểm (hai `Button` và một `TextBox`) và định vị chúng bằng điều kiểm trực kiện (đóng vai trò là dấu phân cách). Hai `Button` được kết nối với các phương thức thụ lý sự kiện riêng biệt. `TextBox` được cấp một định danh duy nhất để phần text của nó có thể được thu lấy sau này (trong đáp ứng cho cú nhấp chuột vào `Button`).



Hình 7.9 Các điều kiểm được-tạo-động

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.Security;

public class DynamicControls : System.Web.UI.Page {

    protected System.Web.UI.WebControls.Label lblMessage;
    protected System.Web.UI.WebControls.Panel pnl;

    // (Bỏ qua phần mã designer.)
```



```
private void Page_Load(object sender, System.EventArgs e) {

    // Tạo Button.
    Button dynamicButton = new Button();
    dynamicButton.Text = "Dynamic Button A";

    // Kết nối phương thức thụ lý sự kiện.
    dynamicButton.Click += new EventHandler(cmdDynamicA_Click);

    // Thêm Button vào Panel.
    pnl.Controls.Add(dynamicButton);

    // Thêm dấu ngắt dòng.
    pnl.Controls.Add(new LiteralControl("<br>"));

    // Tạo Button thứ hai.
    dynamicButton = new Button();
    dynamicButton.Text = "Dynamic Button B";
    dynamicButton.Click += new EventHandler(cmdDynamicB_Click);
    pnl.Controls.Add(dynamicButton);

    // Thêm dấu ngắt dòng.
    pnl.Controls.Add(new LiteralControl("<br><br>"));

    // Tạo TextBox.
    TextBox dynamicText = new TextBox();
    pnl.Controls.Add(dynamicText);

    // Gán ID cho TextBox.
    dynamicText.ID = "DynamicText";
}

private void cmdDynamicA_Click(object sender, System.EventArgs e) {

    lblMessage.Text = "Clicked A";
    GetText();
}
```

```

private void cmdDynamicB_Click(object sender, System.EventArgs e) {

    lblMessage.Text = "Clicked B";
    GetText();
}

private void GetText() {
    lblMessage.Text += "<br><br>";

    foreach (Control ctrl in pnl.Controls) {
        if (ctrl.ID == "DynamicText") {
            lblMessage.Text += "TextBox contains: " +
                ((TextBox)ctrl).Text;
        }
    }
}
}

```

Nếu cần tạo động các layout phức tạp (gồm các nhóm điều khiển được tạo dựng trước), bạn có thể chuyển sang sử dụng điều khiển người dùng và nạp động chúng vào trang. Kỹ thuật này sẽ được trình bày trong mục 7.13.

12.

Trả về động một bức hình



Bạn cần trả về động một bức hình (chẳng hạn, để tạo dựng kết xuất dạng biểu đồ hoặc đồ thị).



Tạo dựng bức hình bằng *GDI+* và một đối tượng *System.Drawing.Bitmap* trong bộ-nhớ. Kế đó, bạn có thể ghi nó ra dòng kết xuất (*output stream*), hoặc lưu nó vào ổ đĩa cứng của server và hiển thị nó với điều khiển web *Image*.

Bạn có thể vẽ động các bức hình trong một ứng dụng *Web* bằng cách sử dụng cùng đoạn mã *GDI+* đã sử dụng trong một ứng dụng dựa-trên-*Windows*. Chỉ có điểm khác là cách thức bạn trả về bức hình cuối cùng như thế nào. Về cơ bản, có hai cách tiếp cận mà bạn có thể sử dụng:

- Bạn có thể đưa trực tiếp nội dung nhị phân của bức hình vào thuộc tính *OutputStream* của đối tượng *HttpResponse*. Đây là một cách tiếp cận hay nếu bạn không muốn làm đầy ổ đĩa cứng của server với các file hình không hề được sử dụng lại. Đây cũng là sự lựa chọn tốt nhất nếu bạn cần tạo động các bức hình được thiết kế để phù hợp với đầu vào của người dùng.

- Bạn có thể lưu bức hình vào hệ thống file của web-server và sử dụng thẻ *HTML* `<img>` để hiển thị nó. Đây là sự lựa chọn tốt nếu bạn cần tạo một bức hình sẽ được sử dụng lại, vì tránh được chi phí của việc tái tạo hình liên tục.

Mục này khảo sát cả hai cách tiếp cận trên. Trước tiên, hãy xét cách tạo động một hình mà không lưu nó ra file. Trong ví dụ này, mục đích là tạo một banner đơn giản (xem hình 7.10).



Hình 7.10 Một banner được-tạo-động

Đề ý rằng chỉ có phần text của banner là do người dùng cung cấp (thông qua chuỗi truy vấn). Font, màu, và kích thước được viết mã cứng (mặc dù chúng có thể được thiết lập dễ dàng dựa vào các đối số chuỗi truy vấn khác hoặc dựa vào file *Web.config*).

Đoạn mã dưới đây sẽ trình bày cách tiếp cận này:

```
using System;
using System.Web;
using System.Web.UI.WebControls;
using System.Drawing;
using System.Drawing.Drawing2D;

public class DynamicGraphic : System.Web.UI.Page {

    // (Bỏ qua phần mã designer.)

    private void Page_Load(object sender, System.EventArgs e) {

        // Lấy text từ chuỗi truy vấn.
        // Nếu không có text, chọn mặc định.
        string text = "";
        if (Request.QueryString["image"] == null) {
```

```
        Response.Redirect(Request.Url + "?image=" +
            Server.UrlEncode("This is a test image"));
    }
    else {
        text = Server.UrlDecode(Request.QueryString["image"]);
    }

    // Tạo một hình bitmap trong-bộ-nhớ
    // (rộng 300 pixel và cao 200 pixel).
    int width = 300, height = 200;
    Bitmap bitmap = new Bitmap(width, height);

    // Lấy graphics context của hình bitmap.
    Graphics graphics = Graphics.FromImage(bitmap);

    // Thiết lập màu nền và chất lượng hình.
    // Màu này sẽ trở thành đường viền.
    graphics.Clear(Color.OrangeRed);
    graphics.SmoothingMode = SmoothingMode.AntiAlias;

    // Vẽ một hình chữ nhật.
    graphics.FillRectangle(new SolidBrush(Color.Olive), 5, 5,
        width - 10, height - 10);

    // Chọn font và alignment cho text.
    Font fontBanner = new Font("Verdana", 24, FontStyle.Bold);
    StringFormat stringFormat = new StringFormat();
    stringFormat.Alignment = StringAlignment.Center;
    stringFormat.LineAlignment = StringAlignment.Center;

    // Vẽ text.
    graphics.DrawString(text, fontBanner,
        new SolidBrush(Color.LightYellow),
        new Rectangle(0, 0, width, height), stringFormat);
```

```
// Lưu bức hình vào dòng kết xuất.
bitmap.Save(Response.OutputStream,
    System.Drawing.Imaging.ImageFormat.Gif);

graphics.Dispose();
bitmap.Dispose();
}
}
```

Khi lưu bức hình vào dòng kết xuất, bạn sẽ thế chỗ bất kỳ kết xuất nào khác. Vì thế, bạn không thể sử dụng kỹ thuật này với một trang cũng có chứa *Web controls* hoặc nội dung *HTML* tĩnh. Theo đó, nếu muốn sử dụng một trang phối hợp các bức hình được-tạo-động và các điều kiểm web, bạn cần “bọc” bức hình được-tạo-động trong một điều kiểm hoặc ghi bức hình ra ổ đĩa cứng trước khi hiển thị nó.

Nếu muốn lưu file vào ổ đĩa cứng, bạn cần chuyển phần mã tạo dựng bức hình thành một phương thức độc lập, mà chúng ta sẽ đặt tên là `GenerateBanner`. Kế đó, trong phương thức xử lý sự kiện `Page.Load`, bạn kiểm tra xem file đã tồn tại chưa (sử dụng phương thức tĩnh `File.Exists`). Nếu file chưa tồn tại, bạn tạo nó trong bộ nhớ bằng phương thức `GenerateBanner` và lưu nó bằng phương thức `Bitmap.Save`. Nếu file đã tồn tại, bạn chỉ cần nạp thẳng bức hình này.

Đoạn mã dưới đây sẽ trình bày cách tiếp cận này:

```
using System;
using System.IO;
using System.Web;
using System.Web.UI.WebControls;
using System.Drawing;
using System.Drawing.Drawing2D;

public class DynamicGraphic : System.Web.UI.Page {

    protected System.Web.UI.WebControls.Image imageControl;

    // (Bỏ qua phần mã designer.)

    private Bitmap GenerateBanner() {

        // Tạo dựng bức hình, sử dụng phần mã trong ví dụ ở trên.
    }

    private void Page_Load(object sender, System.EventArgs e) {
```

```

// Thiết lập tên file.
// Giả sử chuỗi truy vấn chứa các ký tự hợp lệ cho tên file.
string fileName = Request.QueryString["image"] + ".gif";

Bitmap bitmap = null;

// Kiểm tra bức hình với phần text này đã tồn tại hay chưa.
if (File.Exists(fileName)) {

    // Nạp bức hình hiện có.
    try {
        bitmap = new Bitmap(fileName);
    } catch {
        bitmap = GenerateBanner();
    }
}
else {
    bitmap = GenerateBanner();

    // Lưu bức hình.
    bitmap.Save(fileName,
        System.Drawing.Imaging.ImageFormat.Gif);
}

// Hiển thị bức hình.
imageControl.ImageUrl = fileName;
}
}

```

13. *Nạp điều khiển người dùng bằng mã lệnh*

- ?** Bạn cần tạo dựng động giao diện người dùng (*user interface*) cho một trang từ một hoặc nhiều điều khiển người dùng (*user control*).
- ✕** Sử dụng phương thức `Page.LoadControl` để tạo đối tượng điều khiển từ file `.ascx`, và rồi thêm nó vào tập hợp `Controls` của một điều khiển container.

Điều khiển người dùng là các nhóm điều khiển độc lập. Như *Web Form*, điều khiển người dùng bao gồm phần layout định nghĩa các điều khiển bên trong (file *.ascx*) và phần code-behind cùng với logic thụ lý sự kiện (file *.cs*). Điều khiển người dùng cho phép bạn sử dụng lại các phần tử giao diện thông thường trên nhiều trang và tạo dựng các giao diện phức tạp từ các khối nhỏ hơn. Một đặc điểm hữu ích của điều khiển người dùng là chúng có thể được nạp bằng mã lệnh, điều này cho phép bạn tạo một giao diện cấu hình cao do bạn thiết kế động y theo người dùng. Bạn chỉ cần nạp điều khiển, cấu hình các thuộc tính của nó, và rồi thêm nó vào một điều khiển container.

Ví dụ, xét trang web đã tạo động các bức hình trong mục 7.12. Một giải pháp theo cách hướng đối tượng hơn có thể hiện thực được là tạo một điều khiển người dùng tùy biến đóng gói bức hình được-tạo-động. Điều khiển người dùng này cho phép trang thiết lập text, font, màu... thông qua các thuộc tính khác nhau.

```
using System;
using System.Web;
using System.Web.UI.WebControls;
using System.Drawing;
using System.Drawing.Drawing2D;

public class DynamicGraphicControl : System.Web.UI.UserControl {

    // (Bỏ qua phần mã designer.)

    private string imageText = "";
    public string ImageText {
        get {
            return imageText;
        }
        set {
            imageText = value;
        }
    }

    private Font textFont;
    public Font TextFont {
        get {
            return textFont;
        }
        set {
            textFont = value;
        }
    }
}
```

```
}

private Size imageSize;
public Size ImageSize {
    get {
        return imageSize;
    }
    set {
        imageSize = value;
    }
}

private Color foreColor;
public Color ForeColor {
    get {
        return foreColor;
    }
    set {
        foreColor = value;
    }
}

private Color backColor;
public Color BackColor {
    get {
        return backColor;
    }
    set {
        backColor = value;
    }
}

private Color borderColor;
public Color BorderColor {
    get {
```



```

        return borderColor;
    }
    set {
        borderColor = value;
    }
}

private void Page_Load(object sender, System.EventArgs e) {

    if (ImageText == "")
        return;

    // Tạo một hình bitmap trong-bộ-nhớ.
    Bitmap bitmap = new Bitmap(ImageSize.Width, ImageSize.Height);

    // lấy graphics context của hình bitmap.
    Graphics graphics = Graphics.FromImage(bitmap);

    // Thiết lập màu nền và chất lượng hình.
    // Màu này sẽ trở thành đường viền.
    graphics.Clear(BorderColor);
    graphics.SmoothingMode = SmoothingMode.AntiAlias;

    // Vẽ một hình chữ nhật.
    graphics.FillRectangle(new SolidBrush(BackColor), 5, 5,
        ImageSize.Width - 10, ImageSize.Height - 10);

    // Thiết lập alignment cho text.
    StringFormat stringFormat = new StringFormat();
    stringFormat.Alignment = StringAlignment.Center;
    stringFormat.LineAlignment = StringAlignment.Center;

    // Vẽ text.
    graphics.DrawString(ImageText, TextFont,
        new SolidBrush(ForeColor),
        new Rectangle(0, 0, ImageSize.Width, ImageSize.Height),
        stringFormat);
}

```

```

        // Lưu bức hình vào dòng kết xuất.
        bitmap.Save(Response.OutputStream,
            System.Drawing.Imaging.ImageFormat.Gif);

        graphics.Dispose();
        bitmap.Dispose();
    }
}

```

Web Form nạp điều khiển người dùng này trong phương thức thụ lý sự kiện `Page.Load`. Điều khiển người dùng được đặt trong một `Panel`. Phương thức `LoadControl` trả về một đối tượng `Control`, và nó được ép kiểu thành lớp điều khiển người dùng thích hợp.

```

using System;
using System.Web;
using System.Web.UI.WebControls;
using System.Drawing;

public class DynamicControlTest : System.Web.UI.Page {

    protected System.Web.UI.WebControls.Panel pnl;

    // (Bỏ qua phần mã designer.)

    private void Page_Load(object sender, System.EventArgs e) {

        // Nạp điều khiển.
        DynamicGraphicControl ctrl;
        ctrl = (DynamicGraphicControl)
            Page.LoadControl("DynamicGraphicControl.ascx");

        // Cấu hình các thuộc tính của điều khiển.
        ctrl.ImageText = "This is a new banner test";
        ctrl.ImageSize = new Size(300, 200);
        ctrl.TextFont = new Font("Verdana", 24, FontStyle.Bold);
        ctrl.BackColor = Color.Olive;
        ctrl.ForeColor = Color.LightYellow;
        ctrl.BorderColor = Color.OrangeRed;
    }
}

```

```
// Thêm điều kiểm vào Panel.
pnl.Controls.Add(ctrl);
}
}
```

Trong *Visual Studio .NET*, lớp điều kiểm người dùng luôn có hiệu lực vì các lớp đã được biên dịch thành *.dll*. Tuy nhiên, nếu điều kiểm người dùng không phải một bộ phận của dự án, bạn sẽ không có lớp điều kiểm người dùng và bạn sẽ không thể truy xuất bất kỳ thuộc tính hay phương thức nào của điều kiểm người dùng. Để khắc phục vấn đề này, bạn có thể tạo một lớp cơ sở hoặc một giao diện định nghĩa các chức năng cơ bản để có thể truy xuất vào bất kỳ điều kiểm người dùng tùy biến nào.



Để tìm hiểu kỹ hơn về kỹ thuật này, bạn hãy download *IBuySpy portal case study* tại [http://www.asp.net/IBS_Portal]. Nó trình bày một layout khả-tùy-biến-cao được tạo dựng hoàn toàn từ các điều kiểm người dùng được-nạp-động.

14.

Sử dụng page-caching và fragment-caching



Bạn cần tăng hiệu năng bằng cách lưu giữ các trang được trả về.



Thêm chỉ thị `OutputCache` vào trang hoặc điều kiểm người dùng, và chỉ định trang sẽ được giữ trong cache bao lâu (tính theo giây).

Việc sử dụng caching vừa phải có thể giảm bớt hiệu ứng thất cổ chai (chẳng hạn, truy xuất cơ sở dữ liệu) và tăng toàn bộ hiệu năng của một website. Caching có hiệu quả lớn trong một site có lưu lượng cao. Ví dụ, xét xem điều gì sẽ xảy ra khi bạn lưu giữ một trang hiển thị kết quả của một truy vấn cơ sở dữ liệu. Nếu bạn lưu giữ trang này trong 1 phút, và trang này nhận được 10 yêu cầu trong khoảng thời gian đó, bạn sẽ giảm được 10 lần chi phí truy xuất cơ sở dữ liệu.

Bạn có thể hiện thực caching một cách dễ dàng—chỉ cần thêm chỉ thị `OutputCache` vào trang web. Chỉ thị này phải được thêm vào file *.aspx*, chứ không phải file *.cs*. Ví dụ dưới đây lưu giữ một trang trong 20 giây:

```
<%@ OutputCache Duration="20" VaryByParam="None" %>
```

Và ví dụ dưới đây lưu giữ một trang trong 20 giây nhưng vẫn duy trì các bản sao tùy vào giá trị của các đối số chuỗi truy vấn:

```
<%@ OutputCache Duration="20" VaryByParam="*" %>
```

Bạn có thể thử nghiệm caching bằng một trang hiển thị ngày và giờ trên server. Bạn sẽ nhận thấy rằng các yêu cầu đến sau (đối với trang này) không khiến cho thời gian được tạo mới. Theo đó, thời gian cũ sẽ được hiển thị cho đến khi trang hết hiệu lực.

Output-caching không hiệu quả trong các trường hợp sau đây:

- Trang của bạn cần tự tùy biến y theo các thiết lập đặc thù của người dùng như thông tin xác thực (đối tượng `User`) hoặc trạng thái (đối tượng `Session`). Trong trường hợp này, nó không tạo cảm giác sử dụng lại cùng một trang cho tất cả các người dùng.

- Trang của bạn chứa các điều kiểm post-back và dựng nên các sự kiện phía server.
- Trang của bạn cần thực hiện một hành động khác (như ghi ra file nhật ký, nhập thông tin vào cơ sở dữ liệu, hoặc thay đổi một biến ứng dụng). Một trang được lưu giữ sẽ sử dụng lại toàn bộ *HTML* đã được trả về; phần mã cho trang bị bỏ qua.
- Trang của bạn có chứa các dữ liệu cần phải được tạo cùng với các dữ liệu hiện hành. Đây là trường hợp đối với tìm kiếm sản phẩm, nhưng không phải là trường hợp đối với danh mục sản phẩm.

Trong các trường hợp này, bạn có thể sử dụng một dạng caching linh hoạt hơn. Bạn có thể sử dụng data-caching (sẽ được mô tả trong mục 7.15) để lưu giữ một đối tượng cụ thể. Hoặc bạn có thể sử dụng fragment-caching để lưu giữ một phần của trang. Để sử dụng fragment-caching, bạn cần tạo một điều kiểm người dùng chứa tất cả nội dung có thể được lưu giữ và thêm chỉ thị `OutputCache` vào điều kiểm người dùng. Khi đó, bạn có thể sử dụng điều kiểm người dùng này trong một trang web. Phần mã cho trang web vẫn sẽ chạy, nhưng phần điều kiểm người dùng có thể được lưu giữ.

15.

Dùng lại dữ liệu với ASP.NET Cache



Bạn cần sử dụng caching, nhưng bạn không thể lưu giữ toàn bộ một trang vì nó chứa một số mã cần phải chạy hoặc một số nội dung cần phải được tạo động.



Sử dụng phương thức `Cache.Insert` để lưu giữ bất kỳ đối tượng nào với chính sách hết hiệu lực trượt (*sliding expiration*) hoặc hết hiệu lực tuyệt đối (*absolute expiration*).

Đối tượng `Cache` cho phép bạn lưu giữ hầu như bất kỳ đối tượng *.NET* nào bằng một khóa chuỗi cùng với chính sách hết hiệu lực do bạn định nghĩa. *ASP.NET* duy trì cache một cách tự động, gỡ bỏ các đối tượng khi chúng hết hiệu lực hoặc khi cạn bộ nhớ.

Có hai kiểu chính sách hết hiệu lực bạn có thể sử dụng khi lưu giữ dữ liệu trong cache. Hết hiệu lực tuyệt đối (*absolute expiration*) làm mất hiệu lực các item đã được lưu giữ sau một khoảng thời gian cố định, gần giống với output-caching. Hết hiệu lực tuyệt đối là cách tiếp cận tốt nhất nếu bạn muốn lưu giữ các thông tin cần được làm tươi định kỳ (như danh mục sản phẩm).

```
// Lưu giữ ObjectToCache trong 10 phút (với khóa là "Catalog").
// TimeSpan.Zero cho biết "không sử dụng sliding expiration".
Cache.Insert("Catalog", ObjectToCache, null,
    DateTime.Now.AddMinutes(10), TimeSpan.Zero);
```

Hết hiệu lực trượt (*sliding expiration*) gỡ bỏ các đối tượng sau một khoảng thời gian không dùng đến. Trong trường hợp này, mỗi khi đối tượng được truy xuất, thời gian sống của nó sẽ được reset. Hết hiệu lực trượt làm việc tốt khi bạn có các thông tin luôn có hiệu lực nhưng luôn không được sử dụng (như dữ liệu thuộc về quá khứ). Thông tin này không cần được làm tươi, nhưng không nên giữ nó trong cache nếu nó không được sử dụng.

```
// Lưu giữ ObjectToCache nếu nó được sử dụng ít nhất
// một lần mỗi 10 phút (với khóa là "Catalog").
// DateTime.MaxValue cho biết "không sử dụng absolute expiration".
Cache.Insert("Catalog", ObjectToCache, null,
    DateTime.MaxValue, TimeSpan.FromMinutes(10));
```

Bạn có thể lấy các item từ cache bằng tên khóa. Tuy nhiên, bạn phải luôn kiểm tra trước xem item có tồn tại hay không và rồi ép nó thành kiểu như mong muốn.

Khi thêm các đối tượng vào cache, cách tốt nhất là tạo một hàm độc lập có thể tái tạo đối tượng khi cần. Ví dụ, nếu đang lưu giữ một `DataSet`, bạn cần tạo một hàm kiểm tra cache và chỉ truy vấn lại cơ sở dữ liệu khi không tìm thấy `DataSet`. Điều này cho phép bạn tránh đi phần xử lý tốn nhiều thời gian nhất—truy vấn cơ sở dữ liệu—trong khi vẫn cho phép mã lệnh của bạn thay đổi hiển thị (chẳng hạn, người dùng yêu cầu sắp thứ tự) hoặc thực hiện các hành động khác.

Ví dụ dưới đây sẽ hiển thị một bảng chứa thông tin về khách hàng được lấy từ một `DataSet`. Phần then chốt là lớp `CustomerDatabase`, đóng gói các chức năng cần thiết để đổ dữ liệu vào `DataSet` và quản lý cache. Vì lớp này không thừa kế từ `Page` nên nó cần sử dụng thuộc tính tĩnh `HttpContext.Current` để lấy tham chiếu đến đối tượng `Cache`.

```
using System;
using System.Data;
using System.Web;
using System.Configuration;
using System.Diagnostics;
using System.Web.Caching;
using System.Data.SqlClient;

public class CustomerDatabase {

    private string connectionString;

    // Lấy tham chiếu đến đối tượng Cache.
    private Cache cache = HttpContext.Current.Cache;

    public CustomerDatabase() {

        // Lấy chuỗi kết nối từ file Web.config.
        connectionString =
            ConfigurationSettings.AppSettings["NorthwindCon"];
    }
}
```

```
public DataSet GetCustomers() {

    DataSet customersDS;

    // Kiểm tra item có nằm trong cache hay không.
    if (cache["Customers"] == null) {

        // Lấy DataSet từ cơ sở dữ liệu.
        customersDS = GetCustomersFromDatabase();

        // Lưu giữ item trong cache
        // cùng với sliding expiration là 60 giây.
        cache.Insert("Customers", customersDS, null,
            DateTime.MaxValue, TimeSpan.FromSeconds(60));

        // Hiển thị thông điệp trong cửa sổ Debug.
        Debug.WriteLine("DataSet created from data source.");

    }else {

        // Hiển thị thông điệp trong cửa sổ Debug.
        Debug.WriteLine("DataSet retrieved from cache.");

        // Lấy item.
        customersDS = (DataSet)cache["Customers"];
    }

    // Trả về DataSet.
    return customersDS;
}

private DataSet GetCustomersFromDatabase() {

    // Tạo DataSet.
    DataSet customersDS = new DataSet();
```

```
// Đổ dữ liệu vào DataSet.
SqlConnection con = new SqlConnection(connectionString);
SqlCommand cmd = new SqlCommand("SELECT * FROM Customers", con);
SqlDataAdapter adapter = new SqlDataAdapter(cmd);

try {

    con.Open();
    adapter.Fill(customersDS, "Customers");
} catch {

    customersDS = null;
}
finally {
    con.Close();
}
```

Bước kế tiếp là tạo một trang web sử dụng lớp `CustomerDatabase`. Trang web dưới đây gồm một `DataGrid` và một `Button`. Mỗi khi người dùng nhấp vào `Button`, trang sẽ gọi phương thức `CustomerDatabase.GetCustomers`. Thông tin được lấy từ cache nếu vẫn còn hiệu lực hoặc được truy vấn lại nếu 60 giây đã trôi qua. Bạn có thể nhận biết `DataSet` có được lấy từ cache hay không bằng cách nhìn vào kết xuất trong cửa sổ *Debug*.

```
using System;
using System.Web;
using System.Web.UI.WebControls;

public class LoginPage : System.Web.UI.Page {

    protected System.Web.UI.WebControls.DataGrid DataGrid1;
    protected System.Web.UI.WebControls.Button cmdGetData;

    // (Bỏ qua phần mã designer.)

    private void cmdGetData_Click(object sender, System.EventArgs e){
        CustomerDatabase custDB = new CustomerDatabase();
        DataGrid1.DataSource = custDB.GetCustomers();
        DataGrid1.DataBind();
    }
}
```

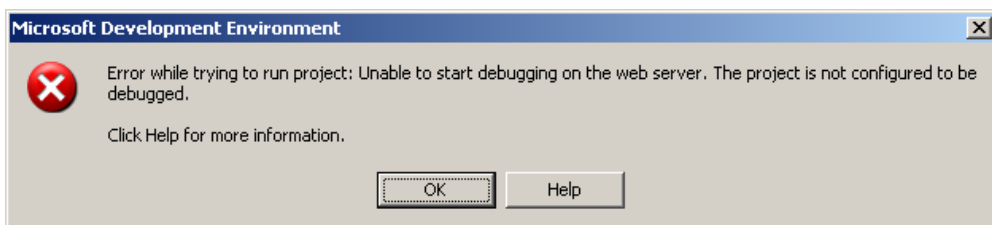
}

16.**Kích hoạt việc gỡ rối ứng dụng Web****?**

Khi thực hiện gỡ rối một ứng dụng *Web* với *Visual Studio .NET*, bạn nhận được lỗi “*Unable to start debugging on the Web server*”.

X

Bảo đảm *Internet Information Services (IIS)* được cài đặt đúng, *IIS* được cài đặt trước *Microsoft .NET Framework*, và *Integrated Windows authentication* được kích hoạt cho thư mục chứa ứng dụng *Web*.



Hình 7.11 Lỗi “Unable to start debugging on the Web server”

Lỗi “*Unable to start debugging on the Web server*” cho biết rằng *Visual Studio .NET* có thể biên dịch ứng dụng *Web* nhưng không thể thực thi nó ở chế độ gỡ rối (*debug mode*). Vấn đề này phát sinh vì nhiều lý do:

- *IIS* chưa được cài đặt hoặc cài đặt không đúng.
- Người dùng đang chạy *Visual Studio .NET* không phải là thành viên của nhóm *Debugger Users* trên web-server.
- Người dùng đang chạy *Visual Studio .NET* không có quyền (*permission*) gỡ rối tiến trình *ASP.NET*. Ví dụ, nếu tiến trình *ASP.NET* đang chạy dưới tài khoản hệ thống cục bộ, người dùng phải có đặc quyền *Administrator* thì mới có thể gỡ rối nó.
- Web-server đang chạy trên phiên bản *Windows* không hỗ trợ gỡ rối, như *Microsoft Windows NT* và *Windows XP Home Edition* (*Windows 2000*, *Windows XP Professional*, *Windows XP Server*, và *Windows Server 2003* đều hỗ trợ gỡ rối).
- Ứng dụng *Web* không có file *Web.config*, hoặc file *Web.config* không kích hoạt gỡ rối.
- Bạn đang chạy *Visual Studio .NET*, và bạn không kích hoạt *Integrated Windows authentication* cho thư mục ảo.

Bước đầu tiên bạn cần thực hiện khi chẩn đoán lý do không thể gỡ rối một ứng dụng *Web* là kiểm tra việc cài đặt *IIS* trên web-server. Để thực hiện điều này, bạn hãy mở <http://localhost/localstart.asp> trong trình duyệt (*localhost* là một bí danh cho máy tính hiện hành). Nếu trang thử nghiệm này không xuất hiện thì có nghĩa là *IIS* không được cài đặt hoặc không được kích hoạt. Bạn cũng có thể bắt đầu ứng dụng *Web* mà không thực hiện gỡ rối bằng cách chọn *Debug | Start Without Debugging* từ thanh trình đơn của *Visual Studio .NET*. Nếu thử nghiệm này thành công thì có nghĩa là *IIS* đã được cài đặt đúng.

Nếu cài đặt *IIS* sau khi cài đặt *Visual Studio .NET* hoặc *.NET Framework*, bạn cần phải "sửa" *.NET Framework* bằng đĩa cài đặt gốc. Để bắt đầu quá trình này, bạn hãy gõ lệnh sau đây vào cửa sổ dòng lệnh (hoặc cửa sổ *Run*), nếu đang dùng phiên bản DVD của *Visual Studio .NET*:

```
<DVD Drive>:\wcu\dotNetFramework\dotnetfx.exe /t:c:\temp
/c:"msiexec.exe /fvecms c:\temp\netfx.msi"
```

Nếu đang dùng phiên bản CD của *Visual Studio .NET*, bạn hãy sử dụng dòng lệnh sau đây cùng với đĩa *Windows Component Update*:

```
<CD Drive>:\dotNetFramework\dotnetfx.exe /t:c:\temp
/c:"msiexec.exe /fvecms c:\temp\netfx.msi"
```

Nếu *IIS* được cài đặt đúng, bước kế tiếp là xác nhận tính hợp lệ của file *Web.config*. File *Web.config* phải có cấu trúc như sau:

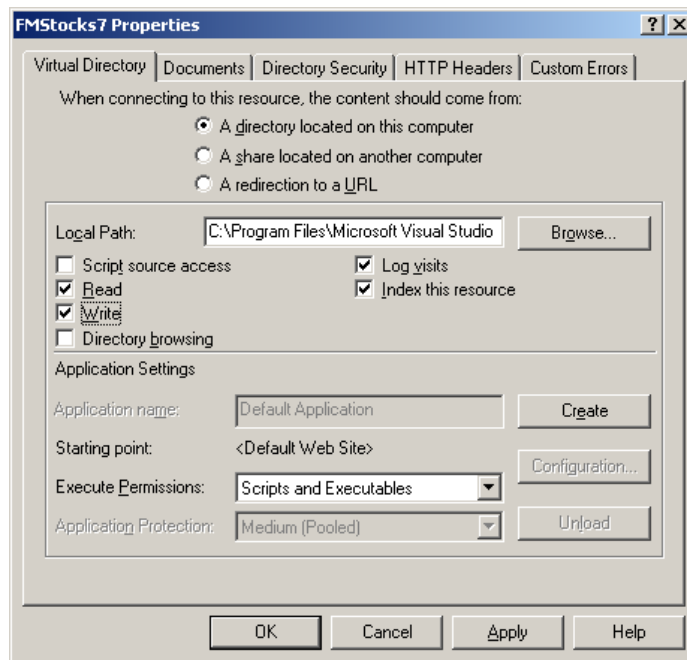
```
<configuration>
  <system.web>
    <compilation defaultLanguage="c#"
      debug="true" >

    <!-- Bỏ qua các thiết lập khác. -->

  </system.web>
</configuration>
```

Theo mặc định, *Visual Studio .NET* thêm thẻ `<compilation>` vào file *Web.config* (được tạo tự động) với đặc tính `debug` được thiết lập là `true`.

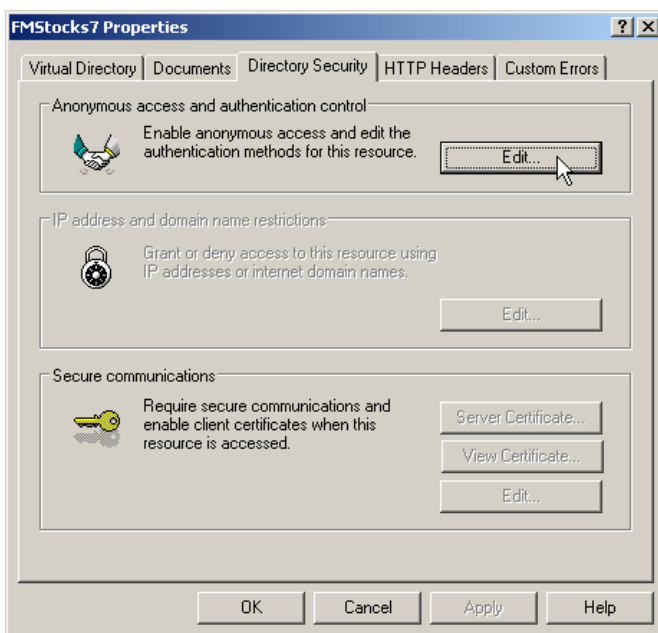
Bước kế tiếp là kiểm tra cấu hình *IIS*. Vấn đề sẽ xảy ra nếu bạn quên tạo thư mục ảo cho ứng dụng hoặc nếu bạn chạy một ứng dụng sau khi đã gỡ bỏ hoặc chỉnh sửa thư mục ảo của nó. Để khắc phục các vấn đề này, bạn hãy hiệu chỉnh các thiết lập cho thư mục ảo trong *IIS Manager* bằng cách chọn *Control Panel | Administrative Tools | Internet Information Services* từ *Start Menu*. Bạn cần kiểm tra xem thư mục ảo đã tồn tại và được cấu hình là một ứng dụng *Web* hay chưa (bạn có thể xem các thiết lập cho thư mục ảo bằng cách nhấp phải vào nó và chọn *Properties*). Ví dụ, thư mục ảo *FMStocks7* (xem hình 7.12) đã tồn tại nhưng chưa được cấu hình là một ứng dụng *Web*. Để giải quyết vấn đề này, bạn chỉ cần nhấp vào nút *Create* trong phần *Application Settings*.



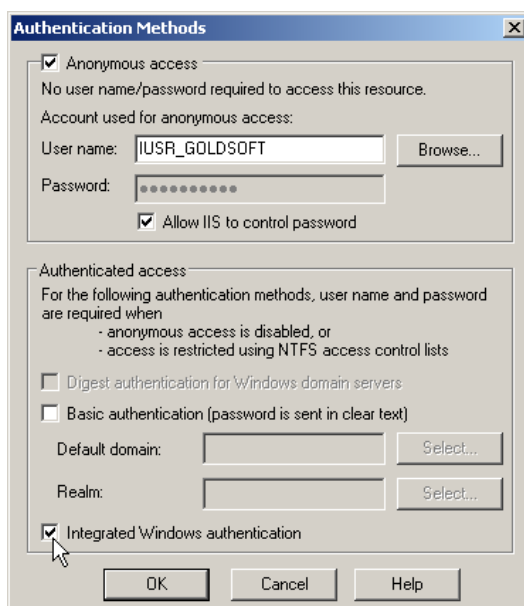
Hình 7.12 Một thư mục ảo không phải là một ứng dụng Web

Một vấn đề khác về cấu hình IIS có thể xảy ra trong *Visual Studio .NET* là không thực hiện xác thực. *Visual Studio .NET* truy xuất web-server cục bộ bằng *Integrated Windows authentication*, ngay cả khi bạn đã kích hoạt *Anonymous authentication* cho thư mục ảo. Điều này nghĩa là thư mục ảo của bạn phải cho phép cả *Anonymous authentication* và *Integrated Windows authentication*. Để cho phép cả hai phương pháp này, bạn cần thực hiện các bước sau đây:

1. Trong *IIS Manager*, nhấp phải vào thư mục ảo cho ứng dụng của bạn và chọn *Properties* (bạn có thể cấu hình kiểu xác thực cho tất cả các thư mục nếu nhấp phải vào thư mục *Web Sites* và chọn *Properties*).
2. Chọn thẻ *Directory Security* (xem hình 7.13).
3. Trong phần *Anonymous access and authentication control*, nhấp nút *Edit*.
4. Trong hộp thoại *Authentication Methods*, bên dưới *Authenticated access*, chọn *Integrated Windows authentication* (xem hình 7.14).
5. Nhấp *OK*.



Hình 7.13 Directory Security



Hình 7.14 Kích hoạt Integrated Windows authentication

17.

Thay đổi quyền đã cấp cho mã ASP.NET

?

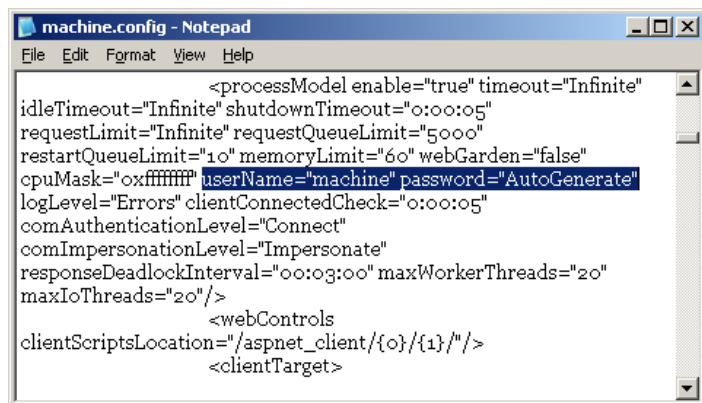
Bạn cần cấp cho phần mã của trang web hoặc dịch vụ *Web* của bạn nhiều đặc quyền *Windows* hơn (như các quyền cho file, thiết lập *Registry*, cơ sở dữ liệu, và các tài nguyên khác), hoặc bạn cần hạn chế các đặc quyền hiện có.

✕

Bạn có thể gán trực tiếp các đặc quyền cho tài khoản *ASPNET* cục bộ, hoặc có thể cấu hình mã *ASP.NET* của bạn để sử dụng một tài khoản hoàn toàn khác bằng cách chỉnh sửa thẻ `<processModel>` trong file *machine.config*. Bạn cũng có thể sử dụng kỹ thuật giả nhận (*impersonation*) để một đoạn mã thực thi với các quyền của người dùng đã được *IIS* xác thực.

Mã *ASP.NET* không chạy dưới tài khoản người dùng đã được *IIS* xác thực hoặc tài khoản mặc định *IUSR_[ServerName]*. Một phần lý do là tài khoản này thường không có đủ đặc quyền đối với mã *ASP.NET* (có thể tạo và xóa các file tạm để quản lý tiến trình biên dịch trang web).

Theo mặc định, các trang *ASP.NET* chạy bằng tài khoản *ASPNET* cục bộ, tài khoản này có một tập các đặc quyền bị giới hạn. Nếu cần mã *ASP.NET* của bạn thực hiện điều gì đó không được phép theo mặc định đối với tài khoản cục bộ (chẳng hạn, ghi ra ổ đĩa cứng trên server), bạn có thể cấp các quyền này cho tiến trình *ASPNET*. Bạn cũng có thể thay đổi thiết lập này bằng cách mở file *machine.config* (nằm trong thư mục `<Windows directory>\Microsoft.NET\Framework\<version>\CONFIG\`) và chỉnh sửa thẻ `<processModel>`.



Hình 7.15 File machine.config

Bạn có thể thiết lập các đặc tính `userName` và `password` cho một người dùng bất kỳ, hoặc bạn có thể đưa vào sử dụng tiến trình *ASPNET* cục bộ (thiết lập `userName` là `machine` và `password` là `AutoGenerate`) hoặc tài khoản hệ thống cục bộ (thiết lập `userName` là `SYSTEM` và `password` là `AutoGenerate`). Vì tài khoản hệ thống cục bộ có đầy đủ quyền trên máy tính nên đừng bao giờ sử dụng tài khoản này ngoại trừ mục đích thử nghiệm. Các thiết lập về tài khoản *ASP.NET* là toàn cục và tất cả các ứng dụng *Web* sẽ chia sẻ tài khoản mà bạn chỉ định.

Bạn cũng có thể thay đổi tài khoản dùng để thực thi các ứng dụng hoặc đoạn mã nào đó bằng kỹ thuật giả nhận. Ví dụ, để cấu hình một ứng dụng *Web* chạy dưới một tài khoản người dùng khác, bạn cần thêm thẻ `<identity>` vào file *Web.config* như sau:

```
<configuration>
  <system.web>
    <!-- Bỏ qua các thiết lập khác. -->

    <identity impersonate="true" name="domain\user" password="pwd"/>

  </system.web>
</configuration>
```

Bạn cũng có thể chỉ thị ứng dụng *Web* sử dụng identity đã được *IIS* xác thực, đó sẽ là tài khoản nặc danh *IUSR_[ServerName]* nếu bạn không sử dụng *Windows authentication* (đã được thảo luận trong mục 7.8). Bạn chỉ cần thêm thẻ `<identity>` mà không phải thêm bất cứ thông tin xác thực nào:

```
<identity impersonate="true"/>
```

Nhớ rằng, đối với kiểu giả nhận này, tài khoản người dùng sẽ yêu cầu truy xuất đọc/ghi đến thư mục *Temporary ASP.NET Files* (nơi lưu trữ các file *ASP.NET* đã được biên dịch—được định vị tại `\[WindowsDirectory]\Microsoft.NET\Framework\[version]\Temporary ASP.NET Files`).

Cuối cùng, bạn cũng có thể sử dụng kỹ thuật giả nhận bằng mã lệnh (vấn đề này sẽ được mô tả chi tiết trong mục 13.15). Tuy vậy, ví dụ dưới đây sẽ làm việc cùng với *Windows authentication*. Nếu *IIS* đã xác thực người dùng (được mô tả trong mục 7.8), identity đó sẽ được thừa nhận khi sử dụng phương thức `WindowsIdentity.Impersonate`. Để sử dụng đoạn mã này, bạn phải nhập không gian tên `System.Security.Principal`.

```
if (User.GetType() == typeof(WindowsPrincipal)) {

    WindowsIdentity id = (WindowsIdentity)User.Identity;
    WindowsImpersonationContext impersonate = id.Impersonate();

    // (Thực hiện các tác vụ với ID này.)

    // Trả lại ID.
    impersonate.Undo();
} else {

    // Người dùng không được xác thực.
    // Ném lỗi hoặc thực hiện các bước khác.

}
```


8

ĐỒ HỌA, ĐA PHƯƠNG TIỆN, VÀ IN ẮN

Đồ họa, video, audio, và in ấn là những dấu hiệu tiêu chuẩn của một client đa năng truyền thống trên hệ điều hành *Microsoft Windows*. Khi tiến đến đa phương tiện, *Microsoft .NET Framework* hỗ trợ cho vài đặc tính này, trong khi bỏ qua các đặc tính khác. Ví dụ, bạn sẽ tìm thấy một tập phức tạp các công cụ dùng để thực hiện việc vẽ trong không gian hai chiều và việc in dựa-trên-sự-khả với *GDI+* và các kiểu thuộc không gian tên *System.Drawing*. Các lớp này hỗ trợ các hàm *Graphics Device Interface (GDI)* nguyên sinh trong *Windows API*; khiến cho việc vẽ các hình dạng phức tạp, làm việc với tọa độ và phép biến hình, và xử lý ảnh dễ dàng hơn. Mặt khác, nếu bạn muốn chơi một file audio, hiển thị một file video, hoặc lấy thông tin về các tác vụ in hiện thời, bạn sẽ cần phải vượt ra ngoài *.NET Framework*.

Các đề mục trong chương này trình bày cách sử dụng các đặc tính nội tại *.NET* và các thư viện *Win32* nguyên sinh thông qua *P/Invoke* hoặc *COM Interop*. Một vài kỹ thuật sẽ được đề cập:

- Tìm và sử dụng các font đã được cài đặt (mục 8.1), vẽ hình cuộn được (mục 8.5) và thumbnail (mục 8.8), cũng như thực hiện chụp màn hình bằng *Win32 API* (mục 8.6).
- Làm việc với các điều khiển tùy biến owner-drawn (mục 8.3 và 8.4) và xử lý các đối tượng đồ họa trên màn hình (mục 8.2 và 8.7).
- Chơi các file audio và video (bao gồm *WAV*, *MP3*, và *MPEG*) bằng thư viện *Quartz* có trong *Windows Media Player* (mục 8.9, 8.10, và 8.11).
- In các văn bản đơn giản và phức tạp (mục 8.13 và 8.14), in text với wrapping (mục 8.15), tạo print preview (mục 8.16), và lấy thông tin về máy in (mục 8.12) và hàng đợi in bằng *WMI* (mục 8.17).

1. *Tìm tất cả các font đã được cài đặt*

? Bạn cần lấy danh sách tất cả các font đã được cài đặt trên máy tính hiện hành.

✕ Tạo đối tượng *System.Drawing.Text.InstalledFontCollection*, tập hợp này chứa các đối tượng *FontFamily* mô tả tất cả các font đã được cài đặt.

Lớp *InstalledFontCollection* cho phép bạn lấy thông tin về các font đã được cài đặt. Đoạn mã dưới đây duyệt qua tập hợp font vừa được tạo; mỗi khi tìm thấy một font, nó sẽ tạo một *Label* mới để hiển thị tên font với diện mạo cho trước (kích thước 14 point). *Label* được thêm vào một *Panel* cuộn được, cho phép người dùng cuộn qua danh sách các font hiện có.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Text;

public class ListFonts : System.Windows.Forms.Form {

    private System.Windows.Forms.Panel pnlFonts;
```

```
// (Bỏ qua phần mã designer.)

private void ListFonts_Load(object sender, System.EventArgs e) {

    // Tạo tập hợp font.
    InstalledFontCollection fontFamilies =
        new InstalledFontCollection();

    // Duyệt qua tất cả các font.
    int offset = 10;
    foreach (FontFamily family in fontFamilies.Families) {

        try {

            // Tạo một Label để hiển thị text (viết ở font này).
            Label fontLabel = new Label();
            fontLabel.Text = family.Name;
            fontLabel.Font = new Font(family, 14);
            fontLabel.Left = 10;
            fontLabel.Width = pnlFonts.Width;
            fontLabel.Top = offset;

            // Thêm Label vào Panel cuộn được.
            pnlFonts.Controls.Add(fontLabel);
            offset += 30;

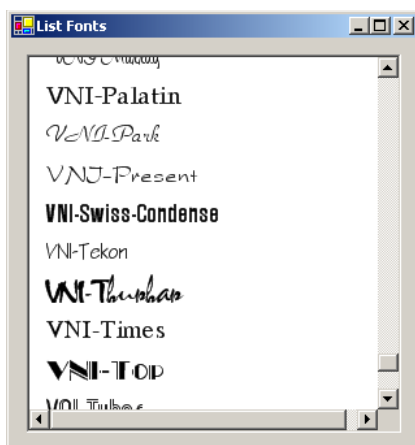
        } catch {

            // Lỗi sẽ xảy ra nếu font được chọn không
            // hỗ trợ normal style (mặc định được sử dụng khi
            // tạo đối tượng Font). Vấn đề này có thể
            // được bỏ qua mà không sao cả.

        }

    }

}
```



Hình 8.1 Danh sách các font đã được cài đặt

2.

Thực hiện “hit testing” với shape

?

Bạn cần nhận biết người dùng có nhấp vào trong một shape hay không.

✕

Kiểm tra điểm mà người dùng đã nhấp vào bằng các phương thức như `Rectangle.Contains` và `Region.IsVisible` (thuộc không gian tên `System.Drawing`), hoặc `GraphicsPath.IsVisible` (thuộc không gian tên `System.Drawing.Drawing2D`), tùy vào kiểu của shape.

Thông thường, nếu sử dụng *GDI+* để vẽ shape trên form, có thể bạn sẽ cần xác định xem khi nào người dùng nhấp vào trong một shape cho trước. *.NET Framework* cung cấp ba phương thức có thể thực hiện công việc này:

- Phương thức `Rectangle.Contains`—nhận vào một điểm và trả về `true` nếu điểm này nằm bên trong hình chữ nhật cho trước. Trong nhiều trường hợp, bạn có thể lấy được hình chữ nhật đối với một kiểu shape khác. Ví dụ, bạn có thể sử dụng `Image.GetBounds` để lấy hình chữ nhật mô tả đường biên của shape. Cấu trúc `Rectangle` là thành viên của không gian tên `System.Drawing`.
- Phương thức `GraphicsPath.IsVisible`—nhận vào một điểm và trả về `true` nếu điểm này nằm bên trong một vùng được định nghĩa bởi `GraphicsPath` khép kín. Vì một `GraphicsPath` có thể chứa nhiều line, shape, và figure nên cách này rất hữu ích nếu bạn muốn kiểm tra một điểm có nằm bên trong một vùng không phải hình chữ nhật hay không. Lớp `GraphicsPath` là một thành viên của không gian tên `System.Drawing.Drawing2D`.
- Phương thức `Region.IsVisible`—nhận vào một điểm và trả về `true` nếu điểm này nằm bên trong một vùng được định nghĩa bởi `Region`. Cũng giống như `GraphicsPath`, `Region` có thể mô tả một shape không phải hình chữ nhật. `Region` là một thành viên của không gian tên `System.Drawing`.

Ví dụ sau đây sẽ tạo một `Rectangle` và một `GraphicsPath`. Theo mặc định, hai shape này có nền màu xanh nhạt. Tuy nhiên, phương thức thụ lý sự kiện `Form.MouseMove` sẽ kiểm tra xem con trỏ chuột có nằm trong một trong hai shape này hay không, và cập nhật màu nền thành hồng tươi nếu con trỏ ở đó.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;

public class HitTesting : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    // Định nghĩa các shape sẽ được sử dụng.
    private GraphicsPath path;
    private Rectangle rectangle;

    // Định nghĩa các cờ để theo vết con trỏ chuột.
    private bool inPath = false;
    private bool inRectangle = false;

    // Định nghĩa các bút vẽ.
    Brush highlightBrush = Brushes.HotPink;
    Brush defaultBrush = Brushes.LightBlue;

    private void HitTesting_Load(object sender, System.EventArgs e) {

        // Tạo các shape.
        path = new GraphicsPath();
        path.AddEllipse(10, 10, 100, 60);
        path.AddCurve(new Point[] {new Point(50, 50),
                                     new Point(10,33), new Point(80,43)});
        path.AddLine(50, 120, 250, 80);
        path.AddLine(120, 40, 110, 50);
        path.CloseFigure();
    }
}
```

```
        rectangle = new Rectangle(100, 170, 220, 120);
    }

    private void HitTesting_Paint(object sender,
        System.Windows.Forms.PaintEventArgs e) {

        Graphics g = e.Graphics;

        // Vẽ shape dựa trên phần chọn hiện tại.
        if (inPath) {

            g.FillPath(highlightBrush, path);
            g.FillRectangle(defaultBrush, rectangle);

        }else if (inRectangle) {

            g.FillRectangle(highlightBrush, rectangle);
            g.FillPath(defaultBrush, path);

        }else {

            g.FillPath(defaultBrush, path);
            g.FillRectangle(defaultBrush, rectangle);
        }
        g.DrawPath(Pens.Black, path);
        g.DrawRectangle(Pens.Black, rectangle);
    }

    private void HitTesting_MouseMove(object sender,
        System.Windows.Forms.MouseEventArgs e) {

        Graphics g = this.CreateGraphics();

        // Thực hiện "hit testing" với hình chữ nhật.
        if (rectangle.Contains(e.X, e.Y)) {

            if (!inRectangle) {
```

```
        inRectangle = true;

        // Đổi màu nền hình chữ nhật.
        g.FillRectangle(highlightBrush, rectangle);
        g.DrawRectangle(Pens.Black, rectangle);
    }

    }else if (inRectangle) {

        inRectangle = false;

        // Phục hồi hình chữ nhật.
        g.FillRectangle(defaultBrush, rectangle);
        g.DrawRectangle(Pens.Black, rectangle);
    }

    // Thực hiện "hit testing" với path.
    if (path.IsVisible(e.X, e.Y)) {

        if (!inPath) {

            inPath = true;

            // Đổi màu nền path.
            g.FillPath(highlightBrush, path);
            g.DrawPath(Pens.Black, path);
        }

    }else if (inPath) {

        inPath = false;

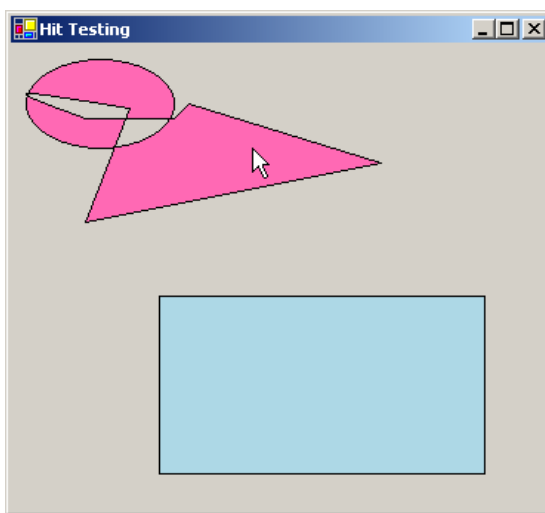
        // Phục hồi path.
        g.FillPath(defaultBrush, path);
```

```

        g.DrawPath(Pens.Black, path);
    }

    g.Dispose();
}
}

```



Hình 8.2 Thực hiện “hit testing” với đối tượng Rectangle và GraphicsPath

Chú ý rằng hoạt động này diễn ra trực tiếp bên trong phương thức thụ lý sự kiện `MouseMove`. Việc vẽ chỉ được thực hiện nếu phần chọn hiện tại thay đổi. Đối với một đoạn mã đơn giản, bạn có thể làm mất hiệu lực toàn bộ form mỗi khi con trỏ chuột di chuyển vào trong hoặc ra khỏi một vùng và thụ lý tất cả việc vẽ trong phương thức thụ lý sự kiện `Form.Paint`, nhưng điều này dẫn đến việc phải vẽ nhiều hơn và tạo nên hiện tượng rung hình (*flicker*) khi toàn bộ form được vẽ lại.

3.

Tạo form có hình dạng tùy biến



Bạn cần tạo một form hoặc điều khiển không phải hình chữ nhật.

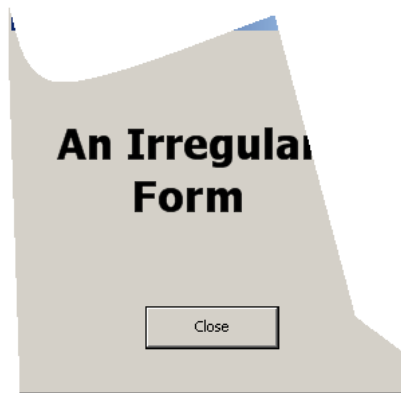


Tạo một đối tượng `System.Drawing.Region` có hình dạng như bạn muốn, và gán nó vào thuộc tính `Form.Region` hoặc `Control.Region`.

Để tạo một form hoặc điều khiển không phải hình chữ nhật, trước hết bạn cần định nghĩa hình dạng mình muốn. Cách tiếp cận dễ nhất là sử dụng đối tượng `System.Drawing.Drawing2D.GraphicsPath`, nó có thể điều tiết bất kỳ sự kết hợp nào của các hình ellipse, chữ nhật, và cung khép kín. Bạn có thể thêm các shape vào một đối tượng `GraphicsPath` bằng các phương thức như `AddEllipse`, `AddRectangle`, và `AddClosedCurve`. Một khi đã hoàn tất việc định nghĩa hình dạng như mong muốn, bạn có thể tạo một đối tượng `Region` từ `GraphicsPath` này—chỉ cần trình ra `GraphicsPath` trong phương thức khởi dựng của

lớp `Region`. Cuối cùng, bạn có thể gán `Region` vào thuộc tính `Form.Region` hoặc `Control.Region`.

Ví dụ dưới đây trình bày cách tạo một form có hình dáng bất thường (xem hình 8.3) bằng hai cung tròn (hai cung này được chuyển thành một figure khép kín bằng phương thức `GraphicsPath.CloseAllFigures`).



Hình 8.3 Form không phải hình chữ nhật

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;

public class IrregularForm : System.Windows.Forms.Form {

    private System.Windows.Forms.Button cmdClose;
    private System.Windows.Forms.Label label1;

    // (Bỏ qua phần mã designer.)

    private void IrregularForm_Load(object sender, System.EventArgs e) {

        GraphicsPath path = new GraphicsPath();
        Point[] pointsA = new Point[] {new Point(0, 0),
            new Point(40, 60), new Point(this.Width - 100, 10)};
        path.AddCurve(pointsA);
```



```

Point[] pointsB = new Point[]
{
    new Point(this.Width - 40, this.Height - 60),
    new Point(this.Width, this.Height),
    new Point(10, this.Height)};
path.AddCurve(pointsB);

path.CloseAllFigures();
this.Region = new Region(path);
}

private void cmdClose_Click(object sender, System.EventArgs e) {

    this.Close();
}
}

```

Đối với ví dụ tạo điều kiện không phải hình chữ nhật, bạn hãy tham khảo mục 8.4.

4. *Tạo điều kiện có hình dạng tùy biến*

- ? **Bạn cần tạo một shape mà người dùng có thể thao tác với nó trên form như kéo rê, thay đổi kích thước....**
- ✕ **Tạo một điều kiện tùy biến, và chép đè painting logic để vẽ shape. Gán shape của bạn vào thuộc tính `Control.Region`. Kế đó, bạn có thể sử dụng `Region` này để thực hiện “hit testing”.**

Nếu muốn tạo một giao diện người dùng phức tạp kết hợp nhiều phần tử được vẽ tùy biến, bạn cần có phương cách để theo vết các phần tử này và cho phép người dùng tương tác với chúng. Cách tiếp cận dễ nhất trong *.NET* là tạo một điều kiện chuyên biệt bằng cách dẫn xuất một lớp từ `System.Windows.Forms.Control`. Kế đó, bạn có thể tùy biến phương cách mà điều kiện này được vẽ dựa theo tập các sự kiện cơ bản của nó.

Điều kiện được trình bày dưới đây mô tả một hình ellipse đơn giản trên form. Tất cả các điều kiện đều được liên hợp với một vùng chữ nhật trên form, do đó điều kiện `EllipseShape` sẽ tạo một ellipse lấp đầy các đường biên này (được cấp thông qua thuộc tính `Control.ClientRectangle`). Một khi shape đã được tạo, thuộc tính `Control.Region` được thiết lập dựa theo biên trên ellipse. Điều này bảo đảm các sự kiện như `MouseMove`, `MouseDown`, `Click...` sẽ xảy ra chỉ khi chuột ở trên ellipse, chứ không phải toàn bộ hình chữ nhật.

Dưới đây là phần mã đầy đủ của lớp `EllipseShape`:

```

using System;

using System.Windows.Forms;

using System.Drawing;

```

```

using System.Drawing.Drawing2D;

public class EllipseShape : System.Windows.Forms.Control {

    private GraphicsPath path = null;

    private void RefreshPath() {

        // Tạo GraphicsPath cho shape và áp dụng nó vào
        // điều khiển bằng cách thiết lập thuộc tính Region.
        path = new GraphicsPath();
        path.AddEllipse(this.ClientRectangle);
        this.Region = new Region(path);
    }

    protected override void OnResize(System.EventArgs e) {

        base.OnResize(e);
        RefreshPath();
        this.Invalidate();
    }

    protected override void OnPaint
        (System.Windows.Forms.PaintEventArgs e) {

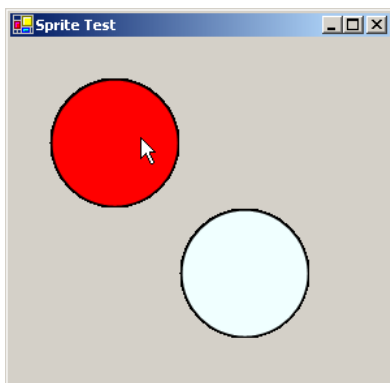
        base.OnPaint(e);
        if (path != null) {

            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
            e.Graphics.FillPath(new SolidBrush(this.BackColor), path);
            e.Graphics.DrawPath(new Pen(this.ForeColor, 4), path);
        }
    }
}

```

Bạn có thể định nghĩa điều khiển `EllipseShape` trong một *Class Library Assembly* độc lập để nó có thể được thêm vào hộp công cụ của *Microsoft Visual Studio .NET* và được sử dụng lúc

thiết kế. Tuy nhiên, ngay cả không thực hiện bước này, cũng dễ dàng tạo được một ứng dụng thử nghiệm đơn giản. Ví dụ dưới đây tạo hai ellipse và cho phép người dùng kéo kéo cả hai vòng quanh form bằng cách giữ chuột xuống và di chuyển con trỏ.



Hình 8.4 Kéo rê các điều khiển có hình dạng tùy biến trên form

```
public class SpriteTest : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    // Các cờ dùng để theo vết chuột khi chế độ kéo rê được kích hoạt.
    private bool isDraggingA = false;
    private bool isDraggingB = false;

    // Các điều khiển có hình dạng ellipse.
    private EllipseShape ellipseA, ellipseB;

    private void SpriteTest_Load(object sender, System.EventArgs e) {

        // Tạo và cấu hình cả hai ellipse.
        ellipseA = new EllipseShape();
        ellipseA.Width = ellipseA.Height = 100;
        ellipseA.Top = ellipseA.Left = 30;
        ellipseA.BackColor = Color.Red;
        this.Controls.Add(ellipseA);

        ellipseB = new EllipseShape();
        ellipseB.Width = ellipseB.Height = 100;
        ellipseB.Top = ellipseB.Left = 130;
```

```

        ellipseB.BackColor = Color.Azure;
        this.Controls.Add(ellipseB);

        // Gắn cả hai ellipse vào cùng tập các phương thức
        // thụ lý sự kiện.
        ellipseA.MouseDown += new MouseEventHandler(Ellipse_MouseDown);
        ellipseA.MouseUp += new MouseEventHandler(Ellipse_MouseUp);
        ellipseA.MouseMove += new MouseEventHandler(Ellipse_MouseMove);

        ellipseB.MouseDown += new MouseEventHandler(Ellipse_MouseDown);
        ellipseB.MouseUp += new MouseEventHandler(Ellipse_MouseUp);
        ellipseB.MouseMove += new MouseEventHandler(Ellipse_MouseMove);
    }

    private void Ellipse_MouseDown(object sender, MouseEventArgs e) {

        // Thu lấy ellipse gây ra sự kiện này.
        Control control = (Control)sender;

        if (e.Button == MouseButtons.Left) {

            control.Tag = new Point(e.X, e.Y);
            if (control == ellipseA) {
                isDraggingA = true;

            }else {
                isDraggingB = true;
            }
        }
    }

    private void Ellipse_MouseUp(object sender, MouseEventArgs e) {

        isDraggingA = false;
        isDraggingB = false;
    }

```

```

    }

    private void Ellipse_MouseMove(object sender, MouseEventArgs e) {

        // Thu lấy ellipse gây ra sự kiện này.
        Control control = (Control)sender;

        if ((isDraggingA && control == ellipseA) ||
            (isDraggingB && control == ellipseB)) {

            // Lấy offset.
            Point point = (Point)control.Tag;

            // Di chuyển điều khiển.
            control.Left = e.X + control.Left - point.X;
            control.Top = e.Y + control.Top - point.Y;
        }
    }
}

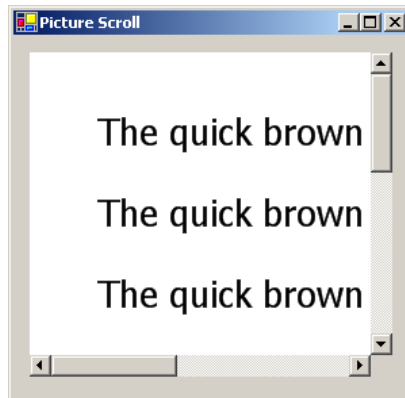
```

5. *Thêm tính năng cuộn cho một bức hình*

- ?** **Bạn cần tạo một bức hình có thể cuộn được (bức hình có nội dung động).**
- X** **Tạo khả năng cuộn tự động cho `System.Windows.Forms.Panel` bằng cách thiết lập `Panel.AutoScroll` là `true` và đặt một `System.Windows.Forms.PictureBox` chứa nội dung bức hình vào trong `Panel`.**

Khi bạn thiết lập `Panel.AutoScroll` là `true`, nếu điều khiển nào đó trong `Panel` vượt quá đường biên của nó, `Panel` sẽ hiển thị thanh cuộn cho phép người dùng chuyển tiếp nội dung. Cách này đặc biệt tốt đối với các bức hình lớn. Bạn có thể nạp hoặc tạo bức hình trong bộ nhớ, gán nó vào một `PictureBox` (không có sự hỗ trợ nội tại nào cho việc cuộn `PictureBox`), và rồi hiển thị `PictureBox` bên trong `Panel`. Chỉ có một vấn đề mà bạn cần nhớ là phải thiết lập kích thước của `PictureBox` bằng với kích thước thật của bức hình bạn muốn hiển thị.

Ví dụ sau đây tạo một bức hình mô tả một văn bản. Bức hình được tạo từ một hình bitmap trong-bộ-nhớ, và nhiều dòng text được thêm vào bằng phương thức `Graphics.DrawString`. Kết đó, bức hình được kết với `PictureBox` (`PictureBox` này được hiển thị trong một `Panel` cuộn được—xem hình 8.5).



Hình 8.5 Thêm tính năng cuộn cho bức hình với nội dung tùy biến

```
using System;
using System.Windows.Forms;
using System.Drawing;

public class PictureScroll : System.Windows.Forms.Form {

    private System.Windows.Forms.PictureBox pictureBox1;
    private System.Windows.Forms.Panel panel1;

    // (Bỏ qua phần mã designer.)

    private void PictureScroll_Load(object sender, System.EventArgs e) {

        string text = "The quick brown fox jumps over the lazy dog.";
        Font font = new Font("Tahoma", 20);

        // Tạo một hình bitmap trong-bộ-nhớ.
        Bitmap b = new Bitmap(600, 600);
        Graphics g = Graphics.FromImage(b);
        g.FillRectangle(Brushes.White, new Rectangle(0, 0, b.Width,
            b.Height));

        // Vẽ nhiều dòng text lên hình bitmap.
        for (int i=0; i < 10; i++) {
```

```

        g.DrawString(text, font, Brushes.Black, 50, 50 + i*60);
    }

    // Hiển thị hình bitmap trong PictureBox.
    pictureBox1.BackgroundImage = b;
    pictureBox1.Size = b.Size;
}
}

```

6.

Thực hiện chụp màn hình Desktop



Bạn cần lấy ảnh chụp của màn hình Desktop hiện thời.



Sử dụng các lời gọi Win32 API GetDesktopWindow, GetDC, và ReleaseDC trong thư viện user32.dll. Ngoài ra, sử dụng GetCurrentObject trong thư viện gdi32.dll.

.NET Framework không cung cấp lớp nào thực hiện việc chụp toàn bộ màn hình (thường được đề cập là cửa sổ Desktop). Tuy nhiên, bạn có thể truy xuất các đặc tính này bằng cách sử dụng P/Invoke với Win32 API.

Bước đầu tiên là tạo một lớp đóng gói các hàm Win32 API bạn cần sử dụng. Lớp dưới đây sẽ khai báo các hàm này và sử dụng chúng trong phương thức công khai Capture để trả về một đối tượng .NET Image chứa cửa sổ Desktop:

```

using System;
using System.Drawing;
using System.Runtime.InteropServices;
using System.Windows.Forms;

public class DesktopCapture {

    [DllImport("user32.dll")]
    private extern static IntPtr GetDesktopWindow();

    [DllImport("user32.dll")]
    private extern static IntPtr GetDC(IntPtr windowHandle);

    [DllImport("gdi32.dll")]
    private extern static IntPtr GetCurrentObject(IntPtr hdc,
        ushort objectType);

    [DllImport("user32.dll")]
    private extern static void ReleaseDC(IntPtr hdc);
}

```

```

const int OBJ_BITMAP = 7;

public static Bitmap Capture() {

    // Lấy Device Context của cửa sổ Desktop.
    IntPtr desktopWindow = GetDesktopWindow();
    IntPtr desktopDC = GetDC( desktopWindow );
    // Lấy GDI handle của bức hình.
    IntPtr desktopBitmap = GetCurrentObject(desktopDC, OBJ_BITMAP);

    // Sử dụng handle để tạo đối tượng .NET Image.
    Bitmap desktopImage = Image.FromHbitmap( desktopBitmap );

    // Giải phóng Device Context và trả về bức hình.
    ReleaseDC(desktopDC);
    return desktopImage;
}
}

```



Hình 8.6 Chụp màn hình Desktop

Bước kế tiếp là tạo một client có thể sử dụng chức năng này. Form dưới đây (xem hình 8.6) sẽ hiển thị bức hình chụp được trong một `PictureBox` (nằm trong một `Panel` cuộn được, như đã được mô tả trong mục 8.5).

```
public class ScreenCapture : System.Windows.Forms.Form {

    private System.Windows.Forms.PictureBox pictureBox1;
    private System.Windows.Forms.Panel panell1;

    // (Bỏ qua phần mã designer.)

    private void cmdCapture_Click(object sender, System.EventArgs e) {

        pictureBox1.Image = DesktopCapture.Capture();
        pictureBox1.Size = pictureBox1.Image.Size;
    }
}
```

7. *Sử dụng “double buffering” để tăng tốc độ vẽ lại*

- ? **Bạn cần tối ưu thao tác vẽ đối với một form thường xuyên được làm tươi, và bạn muốn giảm hiện tượng rung hình (*flicker*).**
- ✕ **Biểu diễn hình ảnh ở dạng hình bitmap trong-bộ-nhớ, rồi chép hình bitmap đã hoàn chỉnh vào form.**

Trong một vài ứng dụng, bạn thường xuyên phải vẽ lại form hoặc điều khiển. Điều này thường gặp khi thể hiện animation (hình động). Ví dụ, bạn có thể sử dụng `Timer` để làm mất hiệu lực form mỗi giây. Khi đó, đoạn mã thực hiện thao tác vẽ có thể vẽ lại một bức hình tại một vị trí mới, tạo cảm giác động. Cách tiếp cận này có một vấn đề: mỗi lần bạn làm mất hiệu lực form, *Windows* sẽ vẽ lại nền cửa sổ (xóa form), và rồi chạy đoạn mã thực hiện thao tác vẽ. Điều này có thể gây ra rung hình đáng kể.

“Double buffering” là một kỹ thuật bạn có thể hiện thực để giảm hiện tượng rung hình. Với “double buffering”, logic vẽ sẽ ghi một hình bitmap trong-bộ-nhớ, và hình này được chép lên form vào cuối quá trình vẽ bằng một thao tác vẽ lại đơn lẻ trong suốt, nhờ đó mà hiện tượng rung hình giảm một cách đáng kể.

Bước đầu tiên khi hiện thực “double buffering” là phải bảo đảm nền của form không tự động được vẽ lại khi form bị mất hiệu lực. Đây là nguyên nhân lớn nhất gây ra rung hình vì nó thay thế bức hình của bạn bằng một frame trống (giả dụ chỉ là một phần nhỏ của một giây). Để ngăn việc vẽ nền, bạn cần chép đè phương thức `OnPaintBackground` của form để nó không nhận hành động nào. Bước thứ hai là sửa đổi đoạn mã thực hiện thao tác vẽ để nó vẽ bức hình thành một hình bitmap trong-bộ-nhớ. Khi hoàn tất, hình bitmap được chép vào form. Cách tiếp cận này bảo đảm làm tươi là một thao tác vẽ lại đơn lẻ, và drawing logic tốn nhiều thời gian đó sẽ không gây ra hiện tượng rung hình.

Ví dụ sau đây trình bày một bức hình (ở đây là logo của *Windows XP*) luân phiên lớn lên và nhỏ lại trên form. Drawing logic được thực hiện trong phương thức thụ lý sự kiện `Form.Paint`, và một `Timer` được sử dụng để làm mất hiệu lực form mỗi 10 mili-giây để bức hình có thể được vẽ lại. Người dùng có thể kích hoạt “double buffering” thông qua một `CheckBox` trên form. Nếu không có “double buffering”, form bị rung đáng kể. Tuy nhiên, khi “double buffering” được kích hoạt, bức hình lớn lên và nhỏ lại một cách mượt mà, không còn độ rung nữa.

```
using System;
using System.Drawing;
using System.Drawing2D;
using System.Windows.Forms;

public class DoubleBuffering : System.Windows.Forms.Form {

    private System.Windows.Forms.CheckBox chkUseDoubleBuffering;
    private System.Windows.Forms.Timer tmrRefresh;

    // (Bỏ qua phần mã designer.)

    // Theo dõi kích thước bức hình,
    // và kiểu animation (lớn lên hoặc nhỏ lại).
    private bool isShrinking = false;
    private int imageSize = 0;

    // Lưu trữ logo sẽ được vẽ lên form.
    private Image image;

    private void DoubleBuffering_Load(object sender,
        System.EventArgs e) {

        // Nạp bức hình logo từ file.
        image = Image.FromFile("image.bmp");

        // Khởi động Timer để làm mất hiệu lực form.
        tmrRefresh.Start();
    }

    private void tmrRefresh_Tick(object sender, System.EventArgs e) {
```

```

// Thay đổi kích thước bức hình dựa vào kiểu animation.
if (isShrinking) {
    imageSize--;

} else {
    imageSize++;
}

// Đổi hướng thay đổi kích thước nếu đến gần biên của form.
if (imageSize > (this.Width - 150)) {
    isShrinking = true;

} else if (imageSize < 1) {
    isShrinking = false;
}

// Vẽ lại form.
this.Invalidate();
}

private void DoubleBuffering_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e) {

    Graphics g;
    Bitmap drawing = null;

    if (chkUseDoubleBuffering.Checked) {

        // "Double buffering" đang được sử dụng.
        // Tạo một bitmap trong-bộ-nhớ mô tả bề mặt của form.
        drawing = new Bitmap(this.Width, this.Height, e.Graphics);
        g = Graphics.FromImage(drawing);

    } else {

        // "Double buffering" không được sử dụng.

```

```

        // Vẽ trực tiếp lên form.
        g = e.Graphics;
    }

    g.SmoothingMode = SmoothingMode.HighQuality;

    // Vẽ nền.
    g.FillRectangle(Brushes.Yellow,
        new Rectangle(new Point(0, 0), this.ClientSize));

    // Vẽ bức hình logo.
    g.DrawImage(image, 50, 50, 50 + imageSize, 50 + imageSize);

    // Nếu sử dụng "double buffering", chép hình bitmap
    // đã hoàn tất trong bộ nhớ vào form.
    if (chkUseDoubleBuffering.Checked) {

        e.Graphics.DrawImageUnscaled(drawing, 0, 0);
        g.Dispose();
    }
}

protected override void OnPaintBackground(
    System.Windows.Forms.PaintEventArgs pevent) {

    // Không làm gì cả.
}
}

```

8.

Hiện thị hình ở dạng thumbnail

?

Bạn cần hiển thị các bức hình trong một thư mục ở dạng thumbnail.

✂

Đọc bức hình từ file bằng phương thức tĩnh `FromFile` của lớp `System.Drawing.Image`. Kế đó, bạn có thể thu lấy hình thumbnail bằng phương thức `Image.GetThumbnailImage`.

Lớp `Image` cung cấp chức năng tạo thumbnail thông qua phương thức `GetThumbnailImage`. Bạn chỉ cần truyền chiều rộng và chiều cao của hình thumbnail bạn muốn (tính bằng pixel), và lớp `Image` sẽ tạo một đối tượng `Image` mới phù hợp với tiêu chuẩn này. Việc khử méo dạng răng cưa (*antialiasing*) được sử dụng khi thu nhỏ bức hình để bảo đảm chất lượng bức hình tốt nhất có thể được, mặc dù một số vết mờ và thiếu hụt một vài tiểu tiết là không thể tránh khỏi (khử méo dạng răng cưa là quá trình loại bỏ các rìa lởm chởm, thường có trong các bức hình đã được thay đổi kích thước, bằng cách tô bóng với một màu trung gian). Ngoài ra, bạn có thể cung cấp một callback để tạo thumbnail một cách bất đồng bộ.

Khi tạo một thumbnail, việc quan trọng là phải bảo đảm tỉ số giữa hai chiều vẫn như cũ (hằng số). Ví dụ, nếu bạn thu nhỏ một bức hình kích thước 200x100 thành một thumbnail kích thước 50x50, chiều rộng sẽ bị nén còn một phần tư và chiều cao bị nén còn một nửa, điều này làm méo bức hình. Để bảo đảm tỉ lệ này vẫn như cũ, bạn có thể thay đổi chiều rộng hoặc chiều cao thành một kích thước cố định rồi điều chỉnh chiều còn lại cho cân xứng.

Ví dụ dưới đây đọc một file bitmap và tạo một thumbnail kích thước không lớn hơn 50x50 (vẫn bảo toàn tỉ lệ gốc):

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class Thumbnails : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    Image thumbnail;

    private void Thumbnails_Load(object sender, System.EventArgs e) {

        Image img = Image.FromFile("test.jpg");
        int thumbnailWidth = 0, thumbnailHeight = 0;

        // Điều chỉnh chiều lớn hơn là 50 pixel.
        // Việc này bảo đảm thumbnail sẽ không lớn hơn 50x50 pixel.
        // Nếu muốn hiển thị nhiều thumbnail, bạn sẽ phải dùng
        // một hình vuông 50x50 pixel cho mỗi thumbnail.
        if (img.Width > img.Height) {

            thumbnailWidth = 50;
            thumbnailHeight = Convert.ToInt32(((50F / img.Width) *
                img.Height));
```

```

    }else {

        thumbnailHeight = 50;
        thumbnailWidth = Convert.ToInt32(((50F / img.Height) *
            img.Width));
    }

    thumbnail = img.GetThumbnailImage(thumbnailWidth,
        thumbnailHeight, null, IntPtr.Zero);
}

private void Thumbnails_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e) {

    e.Graphics.DrawImage(thumbnail, 10, 10);
}
}

```

9.

Phát tiếng “beep” của hệ thống



Bạn cần phát một âm thanh đơn giản, chẳng hạn tiếng “beep” của hệ thống.



Sử dụng một hàm không-được-quản-lý Win32 API như Beep hay sndPlaySound, hoặc gọi hàm Beep của Microsoft Visual Basic .NET.

.NET Framework không chứa bất kỳ lớp được-quản-lý nào thực hiện việc chơi các file âm thanh, ngay cả tiếng “beep” của hệ thống cũng không. Tuy nhiên, bạn có thể dễ dàng vượt qua trở ngại này bằng Win32 API hoặc Visual Basic .NET (cấp hàm Beep thông qua lớp Microsoft.VisualBasic.Interaction). Trong trường hợp thứ hai, bạn phải thêm một tham chiếu đến *Microsoft.VisualBasic.dll* (có trong tất cả các phiên bản của .NET Framework).

Ví dụ sau đây sử dụng cả hàm API Beep và hàm Visual Basic Beep. Chú ý là hàm API sử dụng loa gắn trong của máy tính và phát âm thanh với tần số (tính bằng Hertz, nằm trong khoảng từ 37 đến 32,767) và thời gian (tính bằng mili-giây) cho trước. Cách này sẽ không phát bất kỳ âm thanh nào nếu máy tính không có loa gắn trong. Mặt khác, hàm Visual Basic Beep phát tiếng “beep” chuẩn của hệ thống (là một file WAV). Cách này sẽ không phát bất kỳ âm thanh nào nếu máy tính không có card âm thanh, nếu card âm thanh không được kết nối với loa gắn ngoài, hoặc nếu Windows được cấu hình là không phát âm thanh (thông qua phần *Sounds and Audio Devices* trong *Control Panel*).

```

using System;
using System.Runtime.InteropServices;
using Microsoft.VisualBasic;

public class BeepTest {

    [DllImport("kernel32.dll")]
    private static extern bool Beep(int freq, int dur);

    [STAThread]
    private static void Main(string[] args) {

        // Phát tiếng "beep" tần số 440 Hz trong 100 mili-giây
        // trên internal speaker.
        Console.WriteLine("Win32 API beep test.");
        Beep(440, 100);
        Console.ReadLine();

        // Phát tiếng "beep" mặc định của hệ thống (file WAV).
        Console.WriteLine("VB beep test.");
        Interaction.Beep();
        Console.ReadLine();
    }
}

```

Bạn cũng có thể sử dụng các hàm *Win32 API* để chơi một file âm thanh do bạn chọn. Kỹ thuật này được mô tả trong mục 8.10.

10.

Chơi file audio



Bạn cần chơi một file *WAV* hoặc *MP3*.



Sử dụng hàm *API* `sndPlaySound` (hỗ trợ file *WAV*), hoặc sử dụng thành phần *ActiveMovie* có trong *Windows Media Player* (hỗ trợ file *WAV* và *MP3*)

Để chơi bất kỳ âm thanh nào trong một ứng dụng *.NET*, bạn cần nhận sự giúp đỡ của một thư viện bên ngoài hoặc lời gọi hệ thống. May mắn thay, cả hai tùy chọn này đều dễ dàng thực hiện được.

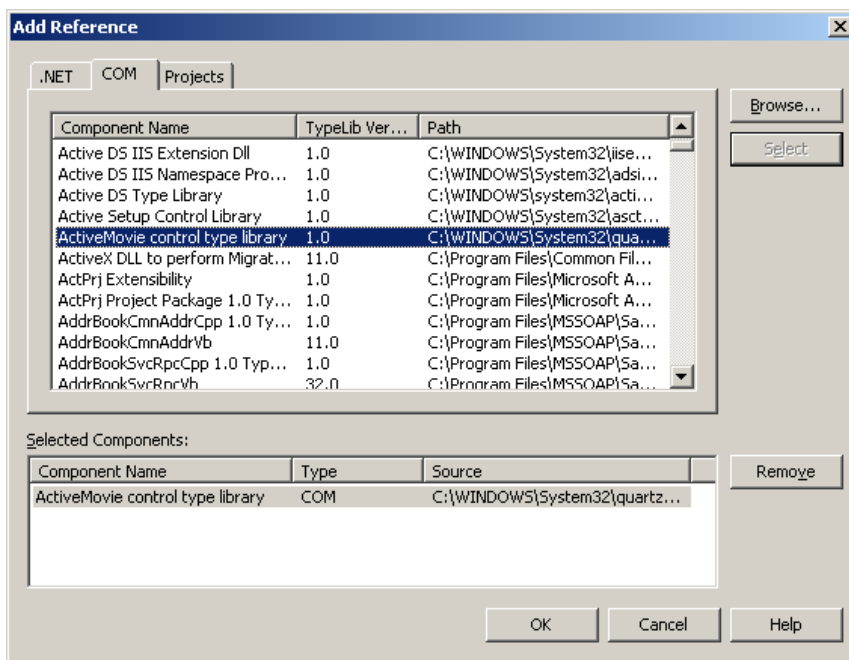
- Thư viện *winmm.dll* (có trong *Windows*) chứa hàm `sndPlaySound` nhận vào tên của một file *WAV* và một thông số chỉ định cách chơi. Bạn có thể chọn chơi âm thanh một cách đồng bộ (gián đoạn việc thực thi của chương trình cho đến khi âm thanh đã hoàn tất), bất đồng bộ, hoặc trong một vòng lặp chạy phía nền.
- Thư viện *Quartz* cung cấp một thành phần *COM* có thể chơi nhiều kiểu file audio, gồm các định dạng *WAV* và *MP3*. Thư viện *Quartz* được cấp thông qua *quartz.dll* và nó là một phần của *Microsoft DirectX* cho *Windows Media Player* và hệ điều hành *Windows*.

Trong ví dụ này, chúng ta sẽ sử dụng cách tiếp cận thứ hai. Bước đầu tiên là tạo một lớp *Interop* có thể quản lý sự tương tác giữa ứng dụng *.NET* và thư viện *Quartz*. Bạn có thể tạo một lớp *C#* cùng với đoạn mã *Interop* này bằng tiện ích *Type Library Importer (tlbimp.exe)* và dòng lệnh sau đây (`[WindowsDir]` là đường dẫn của thư mục cài đặt *Windows*):

```
tlbimp [WindowsDir]\system32\quartz.dll /out:QuartzTypeLib.dll
```

Bạn có thể sử dụng *Visual Studio .NET* để tạo lớp *Interop* bằng cách thêm vào một tham chiếu. Chỉ cần nhấp phải vào dự án của bạn trong *Solution Explorer*, và chọn *Add Reference* từ menu ngữ cảnh. Kế tiếp, chọn thẻ *COM*, và cuộn xuống để chọn *ActiveMovie control type library* (xem hình 8.7).

Một khi lớp *Interop* đã được tạo, bạn có thể làm việc với giao diện `IMediaControl`. Bạn có thể chỉ định file mà bạn muốn chơi bằng `RenderFile`, và có thể điều khiển playback bằng các phương thức như `Run`, `Stop`, và `Pause`. Playback diễn ra trên một tiểu trình độc lập, như thế nó sẽ không block đoạn mã của bạn.



Hình 8.7 Chọn ActiveMovie control type library trong hộp thoại Add Reference

Ví dụ, tiện ích *Console* dưới đây sẽ chơi file audio được chỉ định trong đối số dòng lệnh đầu tiên:

```
using System;

class PlayAudio {

    public static void Main(string[] args) {

        // Lấy tên file được chỉ định trong đối số đầu tiên.
        string filename = args[0];

        // Truy xuất giao diện IMediaControl.
        QuartzTypeLib.FilgraphManager graphManager =
            new QuartzTypeLib.FilgraphManager();
        QuartzTypeLib.IMediaControl mc =
            (QuartzTypeLib.IMediaControl)graphManager;

        // Chỉ định tên file.
        mc.RenderFile(filename);

        // Bắt đầu chơi file audio bất đồng bộ.
        mc.Run();

        Console.WriteLine("Press Enter to continue.");
        Console.ReadLine();
        mc.Stop();
    }
}
```

Bạn cũng có thể sử dụng thư viện *Quartz* để hiển thị file video (sẽ được trình bày trong mục 8.11).

11.

Chơi file video



Bạn cần chơi một file video (như *MPEG*, *AVI*, hoặc *WMV*) ngay trên form.

Sử dụng thành phần *ActiveMovie* có trong *Media Player*. Gắn kết xuất video vào một *PictureBox* trên form bằng cách thiết lập thuộc tính *IVideoWindow.Owner* là thuộc tính *PictureBox.Handle*.

.NET Framework không chứa bất kỳ lớp được-quản-lý nào để tương tác với các file video, nhưng bạn có thể sử dụng chức năng *DirectShow* của thư viện *Quartz* dựa-trên-COM (có trong *Windows Media Player* và hệ điều hành *Windows*). Để biết cách tạo một *Interop Assembly* cho thư viện *Quartz*, bạn hãy tham khảo mục 8.10.

Một khi đã tạo *Interop Assembly*, bạn có thể sử dụng giao diện *IMediaControl* để nạp và chơi một file video. Về cơ bản, kỹ thuật này giống như kỹ thuật đã trình bày trong mục 8.10 với file audio. Tuy nhiên, nếu muốn hiển thị cửa sổ video ngay bên trong giao diện ứng dụng của bạn (hơn là trong một cửa sổ độc lập), bạn phải sử dụng giao diện *IVideoWindow*. Đối tượng *FilgraphManager* có thể được ép kiểu thành giao diện *IMediaControl* và *IVideoWindow*—và nhiều giao diện khác cũng được hỗ trợ như *IBasicAudio* (cho phép bạn cấu hình các thiết lập balance và volume). Với giao diện *IVideoWindow*, bạn có thể gắn kết xuất video vào một đối tượng trên form như *Panel* hoặc *PictureBox*. Để làm được như vậy, bạn cần thiết lập thuộc tính *IVideoWindow.Owner* là handle của điều khiển đó (bạn có thể lấy được handle này bằng thuộc tính *Control.Handle*). Kế tiếp, gọi *IVideoWindow.SetWindowPosition* để thiết lập kích thước và vị trí của cửa sổ. Phương thức này cũng có thể được gọi để thay đổi kích thước video trong quá trình playback (chẳng hạn, khi form bị thay đổi kích thước).

Ví dụ dưới đây cho phép người dùng mở bất kỳ file video nào và chơi nó trong một *PictureBox*. *PictureBox* bị neo đến tất cả các cạnh của form, như thế nó cũng thay đổi kích thước khi form bị thay đổi kích thước. Đoạn mã đáp ứng cho sự kiện *PictureBox.SizeChanged* sẽ thay đổi kích thước của cửa sổ video tương ứng.

```
using System;
using QuartzTypeLib;
using System.Windows.Forms;

public class ShowMovie : System.Windows.Forms.Form {

    private System.Windows.Forms.PictureBox pictureBox1;
    private System.Windows.Forms.Button cmdOpen;

    // (Bỏ qua phần mã designer.)

    // Định nghĩa các hằng dùng để chỉ định window style.
    private const int WM_APP = 0x8000;
    private const int WM_GRAPHNOTIFY = WM_APP + 1;
    private const int EC_COMPLETE = 0x01;
    private const int WS_CHILD = 0x40000000;
    private const int WS_CLIPCHILDREN = 0x20000000;
```

```

// Giữ tham chiếu mức-form đến giao diện Media Control,
// để đoạn mã có thể điều khiển playback cho
// movie được nạp hiện tại.
private IMediaControl mc = null;

// Giữ tham chiếu mức-form đến cửa sổ video trong
// trường hợp nó cần được thay đổi kích thước.
private IVideoWindow videoWindow = null;

private void cmdOpen_Click(object sender, System.EventArgs e) {

    // Cho phép người dùng chọn file.
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter =
        "Media Files|*.mpg;*.avi;*.wma;*.mov;" +
        "*.wav;*.mp2;*.mp3|All Files|*.*";

    if (DialogResult.OK == openFileDialog.ShowDialog()) {

        // Dừng playback đối với movie hiện tại, nếu nó tồn tại.
        if (mc != null) mc.Stop();

        // Nạp file movie.
        FilgraphManager graphManager = new FilgraphManager();
        graphManager.RenderFile(openFileDialog.FileName);

        // Gắn cửa sổ video vào PictureBox trên form.
        try {

            videoWindow = (IVideoWindow)graphManager;
            videoWindow.Owner = (int) pictureBox1.Handle;
            videoWindow.WindowStyle = WS_CHILD | WS_CLIPCHILDREN;
            videoWindow.SetWindowPosition(
                pictureBox1.ClientRectangle.Left,
                pictureBox1.ClientRectangle.Top,
                pictureBox1.ClientRectangle.Width,
                pictureBox1.ClientRectangle.Height);
        }
    }
}

```

```

    } catch {

        // Lỗi có thể xảy ra nếu file không có
        // video source (chẳng hạn, file MP3).
        // Bạn có thể bỏ qua lỗi này và vẫn cho phép
        // playback tiếp tục (không có hình).

    }

    // Bắt đầu playback (bắt đồng bộ).
    mc = (IMediaControl)graphManager;
    mc.Run();
}

private void pictureBox1_SizeChanged(object sender,
    System.EventArgs e) {

    if (videoWindow != null) {

        try {

            videoWindow.SetWindowPosition(
                pictureBox1.ClientRectangle.Left,
                pictureBox1.ClientRectangle.Top,
                pictureBox1.ClientRectangle.Width,
                pictureBox1.ClientRectangle.Height);

        } catch {

            // Bỏ qua ngoại lệ (bị ném khi thay đổi kích thước form)
            // khi file không có video source.

        }

    }

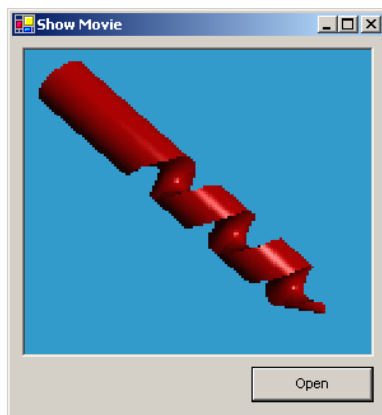
}

```

```
private void ShowMovie_Closing(object sender,
    System.ComponentModel.CancelEventArgs e) {

    if (mc != null) mc.Stop();

}
}
```



Hình 8.8 Chạy file video trong PictureBox trên form

12. *Lấy thông tin về các máy in đã được cài đặt*

? Bạn cần lấy danh sách các máy in đang có hiệu lực trên máy tính.

✂ Đọc tên các máy in đã được cài đặt trong tập hợp `InstalledPrinters` của lớp `System.Drawing.Printing.PrinterSettings`.

Lớp `PrinterSettings` mô tả các thiết lập cho một máy in và thông tin về máy in đó. Ví dụ, bạn có thể sử dụng lớp `PrinterSettings` để xác định các khổ giấy (*paper size*), các nguồn giấy (*paper source*), và các độ phân giải (*resolution*) được hỗ trợ và kiểm tra khả năng in màu hoặc in hai mặt. Ngoài ra, bạn có thể lấy các thiết lập trang mặc định cho lề (*margin*), hướng trang (*orientation*)...

Lớp `PrinterSettings` cung cấp tập hợp tĩnh `InstalledPrinters`, tập hợp này chứa tên của tất cả các máy in đã được cài đặt trên máy tính. Nếu muốn tìm thêm thông tin về các thiết lập cho một máy in cụ thể, bạn cần tạo một đối tượng `PrinterSettings` và thiết lập thuộc tính `PrinterName` cho phù hợp.

Ứng dụng *Console* dưới đây sẽ tìm tất cả các máy in đã được cài đặt trên máy tính và hiển thị thông tin về khổ giấy và độ phân giải được hỗ trợ bởi mỗi máy in.

```
using System;
using System.Drawing.Printing;
```

```
public class ListPrinters {

    private static void Main(string[] args) {

        foreach (string printerName in
            PrinterSettings.InstalledPrinters) {

            // Hiển thị tên máy in.
            Console.WriteLine("Printer: {0}", printerName);

            // Lấy các thiết lập máy in.
            PrinterSettings printer = new PrinterSettings();
            printer.PrinterName = printerName;

            // Kiểm tra tính hợp lệ của máy in.
            // (Bước này cần thiết trong trường hợp bạn đọc tên máy in
            // từ một giá trị do người dùng cấp hoặc một thiết lập trong
            // file registry hay configuration.)
            if (printer.IsValid) {

                // Hiển thị danh sách các độ phân giải hợp lệ.
                Console.WriteLine("Supported Resolutions:");

                foreach (PrinterResolution resolution in
                    printer.PrinterResolutions) {
                    Console.WriteLine("    {0}", resolution);
                }
                Console.WriteLine();

                // Hiển thị danh sách các khổ giấy hợp lệ.
                Console.WriteLine("Supported Paper Sizes:");

                foreach (PaperSize size in printer.PaperSizes) {

                    if (Enum.IsDefined(size.Kind.GetType(), size.Kind)) {
```

```

        Console.WriteLine(" {0}", size);
    }
}
Console.WriteLine();
}
}
Console.ReadLine();
}
}

```

Dưới đây là kết xuất mà ứng dụng này có thể hiển thị:

Printer: SnagIt 7

Supported Resolutions:

```

[PrinterResolution High]
[PrinterResolution Medium]
[PrinterResolution Low]
[PrinterResolution Draft]
[PrinterResolution X=600 Y=600]
[PrinterResolution X=300 Y=300]
[PrinterResolution X=200 Y=200]
[PrinterResolution X=100 Y=100]

```

Supported Paper Sizes:

```

[PaperSize Letter Kind=Letter Height=1100 Width=850]
[PaperSize Legal Kind=Legal Height=1400 Width=850]
[PaperSize Executive Kind=Executive Height=1050 Width=725]
[PaperSize A4 Kind=A4 Height=1169 Width=827]
[PaperSize Envelope #10 Kind=Number10Envelope Height=950 Width=412]
[PaperSize Envelope DL Kind=DLEnvelope Height=866 Width=433]
[PaperSize Envelope C5 Kind=C5Envelope Height=902 Width=638]
[PaperSize Envelope B5 Kind=B5Envelope Height=984 Width=693]
[PaperSize Envelope Monarch Kind=MonarchEnvelope Height=750 Width=387]

```

Printer: Adobe PDF

Supported Resolutions:

```

[PrinterResolution High]
[PrinterResolution Medium]
[PrinterResolution Low]

```

```
[PrinterResolution Draft]
[PrinterResolution X=72 Y=72]
[PrinterResolution X=144 Y=144]
[PrinterResolution X=300 Y=300]
[PrinterResolution X=600 Y=600]
[PrinterResolution X=1200 Y=1200]
[PrinterResolution X=2400 Y=2400]
[PrinterResolution X=3600 Y=3600]
[PrinterResolution X=4000 Y=4000]
```

Supported Paper Sizes:

```
[PaperSize Letter Kind=Letter Height=1100 Width=850]
[PaperSize Tabloid Kind=Tabloid Height=1700 Width=1100]
[PaperSize Ledger Kind=Ledger Height=1100 Width=1700]
[PaperSize Legal Kind=Legal Height=1400 Width=850]
[PaperSize Executive Kind=Executive Height=1050 Width=725]
[PaperSize A3 Kind=A3 Height=1654 Width=1169]
[PaperSize A4 Kind=A4 Height=1169 Width=827]
[PaperSize A2 Kind=A2 Height=2339 Width=1654]
[PaperSize 11 x 17 Kind=Custom Height=1700 Width=1100]
[PaperSize Screen Kind=Custom Height=518 Width=650]
[PaperSize ANSI C Kind=Custom Height=2200 Width=1700]
[PaperSize ANSI D Kind=Custom Height=3400 Width=2200]
[PaperSize ANSI E Kind=Custom Height=4400 Width=3400]
[PaperSize ANSI F Kind=Custom Height=4000 Width=2800]
...
```

Bạn không cần thực hiện cách làm này khi tạo một ứng dụng cung cấp chức năng in. Theo mục 8.13, bạn có thể sử dụng `PrintDialog` để nhắc người dùng chọn một máy in và các thiết lập cho nó. Lớp `PrintDialog` có thể tự động áp dụng các thiết lập của nó cho `PrintDocument` mà không cần viết thêm mã lệnh.



Bạn có thể in một văn bản trong bất kỳ kiểu ứng dụng nào. Tuy nhiên, ứng dụng của bạn phải chứa một tham chiếu đến *System.Drawing.dll*. Nếu đang sử dụng một kiểu dự án trong *Visual Studio .NET* không có tham chiếu này (như ứng dụng *Console*), bạn cần phải thêm nó vào.

13.

In văn bản đơn giản

?

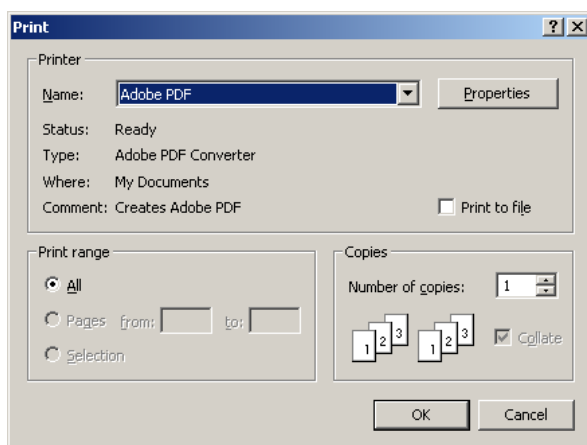
Bạn cần in text hoặc hình.

✕

Thụ lý sự kiện `PrintDocument.PrintPage`, và sử dụng các phương thức `DrawString` và `DrawImage` của lớp `Graphics` để in dữ liệu ra trang.

.NET sử dụng mô hình in dựa-trên-sự-khiến bất đồng bộ (*asynchronous event-based printing model*). Để in một văn bản, bạn cần tạo một đối tượng `System.Drawing.Printing.PrintDocument`, cấu hình các thuộc tính của nó, và rồi gọi phương thức `Print` để thực hiện tác vụ in. Bộ thực thi sẽ phát sinh các sự kiện `BeginPrint`, `PrintPage`, và `EndPrint` của lớp `PrintDocument` trên một tiểu trình mới. Bạn thụ lý các sự kiện này và sử dụng đối tượng `System.Drawing.Graphics` để xuất dữ liệu ra trang. Hình và text được ghi ra trang theo cùng cách như bạn vẽ lên cửa sổ bằng *GDI+*. Tuy nhiên, bạn có thể cần phải theo vết vị trí của bạn trên trang, vì mọi phương thức của lớp `Graphics` đều yêu cầu tọa độ chỉ định nơi cần vẽ.

Các thiết lập cho máy in được cấu hình thông qua các thuộc tính `PrintDocument.PrinterSettings` và `PrintDocument.DefaultPageSettings`. Thuộc tính `PrinterSettings` trả về một đối tượng `PrinterSettings` (đã được mô tả trong mục 8.12) cho biết máy in sẽ được sử dụng. Thuộc tính `DefaultPageSettings` cung cấp đối tượng `PageSettings` cho biết độ phân giải (*resolution*), lề (*margin*), hướng trang (*orientation*)... Bạn có thể cấu hình các thuộc tính này trong mã lệnh, hoặc bạn có thể sử dụng lớp `System.Windows.Forms.PrintDialog` để người dùng thực hiện các thay đổi thông qua hộp thoại in chuẩn của *Windows* (xem hình 8.9). Trong hộp thoại in, người dùng có thể chọn một máy in và chọn số lượng bản in (*number of copies*). Người dùng cũng có thể nhấp vào nút *Properties* để cấu hình các thiết lập nâng cao như cách bố trí trang (*layout*) và độ phân giải máy in (*resolution*). Cuối cùng, người dùng có thể chấp thuận hoặc hủy bỏ thao tác in bằng cách nhấp *OK* hoặc *Cancel*.



Hình 8.9 Hộp thoại Print

Trước khi sử dụng lớp `PrintDialog`, bạn phải gắn nó vào đối tượng `PrintDocument` bằng cách thiết lập thuộc tính `PrintDialog.Document`. Theo đó, bất kỳ sự thay đổi nào do người dùng thực hiện trong hộp thoại in sẽ tự động được áp dụng vào đối tượng `PrintDocument`.

Ví dụ dưới đây là một form chỉ chứa một nút lệnh. Khi người dùng nhấp vào nút này, ứng dụng sẽ tạo một đối tượng `PrintDocument` mới, cho phép người dùng cấu hình các thiết lập in, và rồi bắt đầu thao tác in bất đồng bộ. Phương thức thụ lý sự kiện đáp ứng cho sự kiện `PrintPage` sẽ ghi ra nhiều dòng text và một bức hình.

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Printing;

public class SimplePrint : System.Windows.Forms.Form {

    private System.Windows.Forms.Button cmdPrint;

    // (Bỏ qua phần mã designer.)

    private void cmdPrint_Click(object sender, System.EventArgs e) {

        // Tạo một văn bản và gắn vào phương thức thụ lý sự kiện.
        PrintDocument doc = new PrintDocument();
        doc.PrintPage += new PrintPageEventHandler(this.Doc_PrintPage);

        // Cho phép người dùng chọn một máy in
        // và chỉ định các thiết lập khác.
        PrintDialog dlgSettings = new PrintDialog();
        dlgSettings.Document = doc;

        // Nếu người dùng nhấp OK thì in văn bản.
        if (dlgSettings.ShowDialog() == DialogResult.OK) {

            // Phương thức này trả về tức thì, trước khi tác vụ in
            // bắt đầu. Sự kiện PrintPage sẽ phát sinh bất đồng bộ.
            doc.Print();

        }
    }
}
```

```

    }

    private void Doc_PrintPage(object sender, PrintPageEventArgs e) {

        // Định nghĩa font.
        Font font = new Font("Tahoma", 30);

        // Xác định vị trí trên trang.
        // Trong trường hợp này, chúng ta đọc các thiết lập lề
        // (mặc dù không có gì ngăn chúng ta vượt qua biên lề).
        float x = e.MarginBounds.Left;
        float y = e.MarginBounds.Top;

        // Xác định chiều cao của một dòng (dựa trên font được sử dụng).
        float lineHeight = font.GetHeight(e.Graphics);

        // In năm dòng text.
        for (int i=0; i < 5; i++) {

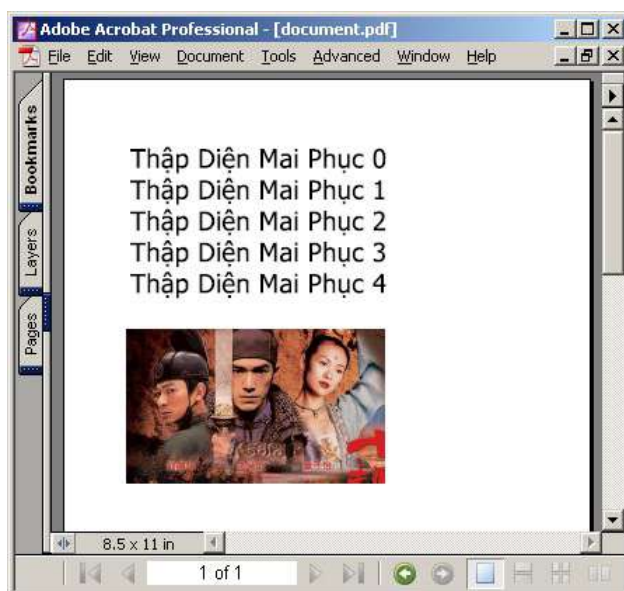
            // Vẽ text với bút vẽ đen, sử dụng font và
            // tọa độ mà chúng ta đã xác định.
            e.Graphics.DrawString("Thập Diện Mai Phục " + i.ToString(),
                font, Brushes.Black, x, y);

            // Dịch xuống một dòng.
            y += lineHeight;
        }
        y += lineHeight;

        // Vẽ hình.
        e.Graphics.DrawImage(Image.FromFile(Application.StartupPath +
            "\\test.bmp"), x, y);
    }
}

```

Ví dụ này có một hạn chế là nó chỉ in một trang đơn. Để in các văn bản phức tạp và nhiều trang hơn, bạn cần phải tạo một lớp chuyên biệt để đóng gói các thông tin về văn bản, trang hiện hành... Kỹ thuật này sẽ được trình bày trong mục 8.14.



Hình 8-10 Văn bản đã được in (sử dụng máy in Adobe PDF)

14.

In văn bản có nhiều trang

- ? **Bạn cần in các văn bản phức tạp gồm nhiều trang và in nhiều văn bản khác nhau cùng một lúc.**
- ✂ **Đặt thông tin muốn in vào một lớp tùy biến dẫn xuất từ `PrintDocument`, và thiết lập thuộc tính `PrintPageEventArgs.HasMorePages` là `true` trong khi vẫn còn trang để in.**

Sự kiện `PrintDocument.PrintPage` cho phép bạn chỉ in một trang đơn. Nếu muốn in nhiều trang hơn, bạn cần thiết lập thuộc tính `PrintPageEventArgs.HasMorePages` là `true` trong phương thức thụ lý sự kiện `PrintPage`. Trong khi `HasMorePages` là `true`, lớp `PrintDocument` vẫn tiếp tục phát sinh các sự kiện `PrintPage` (một sự kiện cho một trang). Tuy nhiên, bạn cần biết là đang in đến trang thứ mấy, dữ liệu gì sẽ được in trên mỗi trang... Để thực hiện điều này, cách hay nhất là tạo một lớp tùy biến.

Lớp `TextDocument` dưới đây thừa kế từ `PrintDocument` và thêm ba thuộc tính: `Text` lưu trữ một mảng các dòng text, `PageNumber` cho biết trang vừa được in, và `Offset` cho biết dòng vừa được in (trong mảng `Text`).

```
public class TextDocument : PrintDocument {

    private string[] text;
```

```

private int pageNumber;
private int offset;

public string[] Text {
    get {return text;}
    set {text = value;}
}

public int PageNumber {
    get {return pageNumber;}
    set {pageNumber = value;}
}

public int Offset {
    get {return offset;}
    set {offset = value;}
}

public TextDocument(string[] text) {
    this.Text = text;
}
}

```

Tùy thuộc vào kiểu tài liệu muốn in, bạn có thể chỉnh sửa lớp này. Ví dụ, bạn có thể lưu trữ một mảng gồm các dữ liệu hình, một vài nội dung sẽ được sử dụng làm header hoặc footer trên mỗi trang, thông tin về font, hoặc tên của file mà bạn muốn đọc thông tin từ nó. Gói các thông tin này vào một lớp đơn sẽ khiến cho việc in nhiều văn bản cùng một lúc dễ dàng hơn.

Phần mã khởi đầu cũng giống như mục 8.13, chỉ khác là bây giờ tạo đối tượng `TextDocument` (thay vì tạo đối tượng `PrintDocument`). Phương thức thụ lý sự kiện `PrintPage` giữ vết của dòng hiện hành và kiểm tra có còn chỗ trống trên trang hay không trước khi thực hiện in dòng kế tiếp. Nếu cần trang mới, thuộc tính `HasMorePages` được thiết lập là `true` và sự kiện `PrintPage` phát sinh lần nữa cho trang kế tiếp. Nếu không, thao tác in xem như hoàn tất.

Phần mã đầy đủ cho form được trình bày dưới đây:

```

using System;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Printing;

public class MultiPagePrint : System.Windows.Forms.Form {

```

```

private System.Windows.Forms.Button cmdPrint;

// (Bỏ qua phần mã designer.)

private void cmdPrint_Click(object sender, System.EventArgs e) {

    // Tạo văn bản gồm 100 dòng.
    string[] printText = new string[100];
    for (int i=0; i < 100; i++) {

        printText[i] = i.ToString();
        printText[i] +=
            ": Thập Diện Mai Phục (House of Flying Daggers)";
    }

    PrintDocument doc = new TextDocument(printText);
    doc.PrintPage += new PrintPageEventHandler(this.Doc_PrintPage);

    PrintDialog dlgSettings = new PrintDialog();
    dlgSettings.Document = doc;

    // Nếu người dùng nhấp OK thì in văn bản.
    if (dlgSettings.ShowDialog() == DialogResult.OK) {
        doc.Print();
    }
}

private void Doc_PrintPage(object sender, PrintPageEventArgs e) {

    // Thu lấy văn bản đã gửi sự kiện này.
    TextDocument doc = (TextDocument)sender;

    // Định nghĩa font và xác định chiều cao.
    Font font = new Font("Tahoma", 10);
    float lineHeight = font.GetHeight(e.Graphics);

```

```

// Tạo các biến lưu giữ vị trí trên trang.
float x = e.MarginBounds.Left;
float y = e.MarginBounds.Top;

// Tăng biến đếm cho trang (cho biết số trang đã được in).
doc.PageNumber += 1;

// In tất cả thông tin vừa khít trên trang.
// Vòng lặp này kết thúc khi dòng kế tiếp vượt quá biên lề,
// hoặc không còn dòng nào để in.

while ((y + lineHeight) < e.MarginBounds.Bottom &&
       doc.Offset <= doc.Text.GetUpperBound(0)) {

    e.Graphics.DrawString(doc.Text[doc.Offset], font,
                           Brushes.Black, x, y);

    // Dịch đến dòng dữ liệu kế tiếp.
    doc.Offset += 1;

    // Dịch xuống một dòng trên trang.
    y += lineHeight;
}

if (doc.Offset < doc.Text.GetUpperBound(0)) {

    // Vẫn còn ít nhất một trang nữa.
    // Cho biết sự kiện này sẽ phát sinh lần nữa.
    e.HasMorePages = true;

} else {

    // Thao tác in đã hoàn tất.
    doc.Offset = 0;
}
}
}

```

15.

In text dạng wrapping

- ?** Bạn cần phân tích một khối text lớn thành các dòng riêng biệt sao cho vừa khít trên một trang.
- X** Sử dụng phương thức nạp chồng `Graphics.DrawString` (phương thức này nhận vào một hình chữ nhật biên).

Thông thường, bạn sẽ cần phá một khối text lớn thành các dòng riêng biệt để có thể in lên trang từng dòng một. *.NET Framework* có thể thực hiện công việc này một cách tự động bằng một phiên bản của phương thức `Graphics.DrawString` (nhận vào một hình chữ nhật biên). Bạn cần chỉ định hình chữ nhật mô tả nơi bạn muốn text sẽ hiển thị. Theo đó, text sẽ được wrap một cách tự động cho vừa khít bên trong đường biên này.

Đoạn mã dưới đây thực hiện cách tiếp cận này:

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Printing;

public class WrappedPrint : System.Windows.Forms.Form {

    private System.Windows.Forms.Button cmdPrint;

    // (Bỏ qua phần mã designer.)

    private void cmdPrint_Click(object sender, System.EventArgs e) {

        // Tạo một văn bản và gắn nó vào phương thức thụ lý sự kiện.
        string text =
            "Windows Server 2003 builds on the core strengths " +
            "of the Windows family of operating systems--security, " +
            "manageability, reliability, availability, and scalability. " +
            "Windows Server 2003 provides an application environment to " +
            "build, deploy, manage, and run XML Web services. " +
            "Additionally, advances in Windows Server 2003 provide many " +
            "benefits for developing applications.";

        PrintDocument doc = new ParagraphDocument(text);
        doc.PrintPage += new PrintPageEventHandler(this.Doc_PrintPage);
```



```

        // Cho phép người dùng chọn một máy in
        // và chỉ định các thiết lập khác.
        PrintDialog dlgSettings = new PrintDialog();
        dlgSettings.Document = doc;

        // Nếu người dùng nhấp OK thì in văn bản.
        if (dlgSettings.ShowDialog() == DialogResult.OK) {
            doc.Print();
        }
    }

    private void Doc_PrintPage(object sender, PrintPageEventArgs e) {

        // Thu lấy văn bản đã gửi sự kiện này.
        ParagraphDocument doc = (ParagraphDocument)sender;

        // Định nghĩa font và text.
        Font font = new Font("Arial", 15);

        e.Graphics.DrawString(doc.Text, font, Brushes.Black,
            e.MarginBounds, StringFormat.GenericDefault);
    }
}

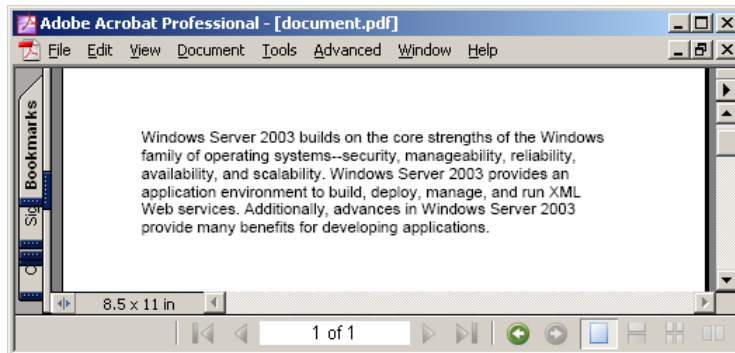
public class ParagraphDocument : PrintDocument {

    private string text;

    public string Text {
        get {return text;}
        set {text = value;}
    }

    public ParagraphDocument(string text) {
        this.Text = text;
    }
}

```



Hình 8.11 Văn bản đã được in (sử dụng máy in Adobe PDF)

16.

Hiển thị *print-preview*

? Bạn cần sử dụng *print-preview* để biết được văn bản khi được in ra sẽ trông như thế nào.

✕ Sử dụng `PrintPreviewDialog` hoặc `PrintPreviewControl` (cả hai đều thuộc không gian tên `System.Windows.Forms`).

.NET cung cấp hai điều khiển có thể nhận vào một đối tượng `PrintDocument`, chạy đoạn mã thực hiện thao tác in, và sử dụng nó để tạo *print-preview* trên màn hình. Hai điều khiển này là:

- `PrintPreviewDialog`—hiển thị *print-preview* trong một cửa sổ độc lập.
- `PrintPreviewControl`—hiển thị *print-preview* trong một form tùy biến.

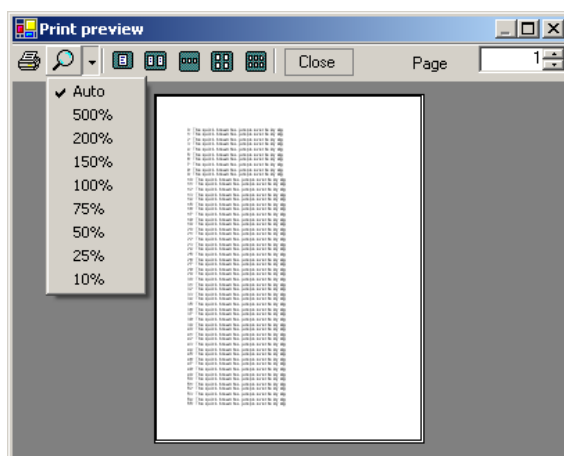
Để sử dụng cửa sổ *print-preview* độc lập, bạn cần tạo đối tượng `PrintPreviewDialog`, ấn định văn bản, và gọi phương thức `PrintPreviewDialog.Show`.

```
PrintPreviewDialog dlgPreview = new PrintPreviewDialog();
dlgPreview.Document = doc;
dlgPreview.Show();
```

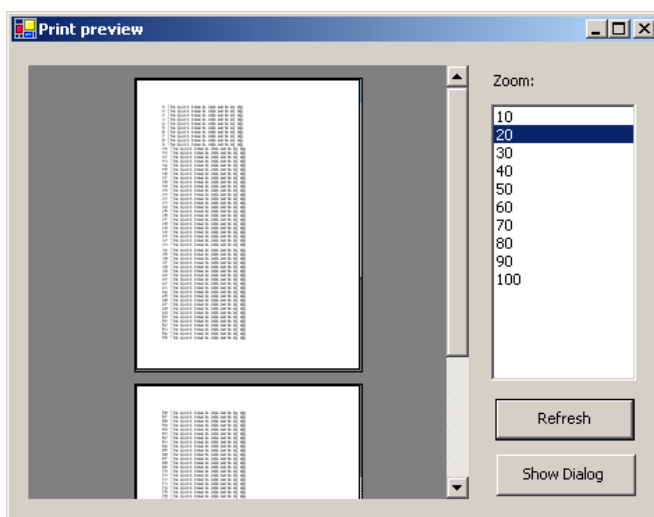
Cửa sổ *print-preview* (xem hình 8.12) cung cấp tất cả các điều khiển cần thiết để di chuyển từ trang này sang trang khác, thu phóng trang... Cửa sổ này cũng cung cấp nút *Print* cho phép người dùng gửi trực tiếp văn bản đến máy in. Bạn có thể biến đổi cửa sổ này bằng cách chỉnh sửa các thuộc tính của `PrintPreviewDialog`.

Bạn cũng có thể thêm `PrintPreviewControl` vào bất kỳ form nào để hiển thị *print-preview* kế bên các thông tin khác. Trong trường hợp này, bạn không cần gọi phương thức `Show`. Ngay khi bạn thiết lập thuộc tính `PrintPreviewControl.Document` thì preview được tạo ra. Để xóa preview, cần thiết lập thuộc tính `Document` là `null`, và để làm tươi preview, cần gán lại thuộc tính `Document`. `PrintPreviewControl` chỉ hiển thị các trang preview, không có thêm điều khiển nào khác. Tuy nhiên, bạn có thể thêm các điều khiển để thực hiện thu phóng trang, lật nhiều trang... Bạn chỉ cần điều chỉnh các thuộc tính của `PrintPreviewControl` cho phù hợp.

Ví dụ, xét form được trình bày trong hình 8.13. Nó sáp nhập một `PrintPreviewControl` và cho phép người dùng chọn thiết lập cho zoom (thu phóng trang).



Hình 8.12 Sử dụng `PrintPreviewDialog` để hiển thị print-preview trong một cửa sổ độc lập



Hình 8.13 Sử dụng `PrintPreviewControl` để hiển thị print-preview trong một cửa sổ tùy biến

Dưới đây là phần mã cho form:

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Printing;

public class PrintPreview : System.Windows.Forms.Form {
```

```

private System.Windows.Forms.PrintPreviewControl printPreviewControl;
private System.Windows.Forms.Button cmdPreview;
private System.Windows.Forms.ListBox lstZoom;
private System.Windows.Forms.Label label1;

// (Bỏ qua phần mã designer.)

private PrintDocument doc;

// (Bỏ qua phần mã cho phương thức thụ lý sự kiện
// PrintDocument.PrintPage - Xem mục 8.14)

private void PrintPreview_Load(object sender, System.EventArgs e) {

    // Thiết lập các zoom được phép.
    for (int i=1; i <= 10; i++) {
        lstZoom.Items.Add((i * 10).ToString());
    }

    // Tạo văn bản gồm 100 dòng.
    string[] printText = new string[100];
    for (int i=0; i < 100; i++) {
        printText[i] = i.ToString();
        printText[i] +=
            ": The quick brown fox jumps over the lazy dog.";
    }

    doc = new TextDocument(printText);
    doc.PrintPage += new PrintPageEventHandler(this.Doc_PrintPage);

    lstZoom.Text = "100";
    printPreviewControl.Zoom = 1;
    printPreviewControl.Document = doc;
    printPreviewControl.Rows = 2;

```

```

    }

    private void cmdPreview_Click(object sender, System.EventArgs e) {

        // Thiết lập zoom.
        printPreviewControl.Zoom = Single.Parse(lstZoom.Text) / 100;

        // Hiển thị cả hai trang, trang này ở trên trang kia.
        printPreviewControl.Rows = 2;

        // Gắn lại PrintDocument để làm tươi preview.
        printPreviewControl.Document = doc;
    }
}

// (Bỏ qua phần mã cho lớp TextDocument - Xem mục 8.14)

```

17.

Quản lý tác vụ in



Bạn cần tạm dừng hoặc phục hồi một tác vụ in hoặc một hàng đợi in.



Sử dụng *Windows Management Instrumentation*. Bạn có thể lấy thông tin từ hàng đợi in bằng một truy vấn với lớp `Win32_PrintJob`, và bạn có thể sử dụng các phương thức `Pause` và `Resume` của các lớp `Win32_PrintJob` và `Win32_Printer` để quản lý hàng đợi.

Windows Management Instrumentation (WMI) cho phép bạn lấy một lượng lớn các thông tin hệ thống bằng một cú pháp giống truy vấn. Một trong các công việc mà bạn có thể thực hiện với *WMI* là lấy danh sách các tác vụ in đang chờ, cùng với thông tin về mỗi tác vụ. Bạn cũng có thể thực hiện các thao tác như in và phục hồi một tác vụ hoặc tất cả các tác vụ cho một máy in. Để sử dụng *WMI*, bạn cần thêm một tham chiếu đến *System.Management.dll*.

Ứng dụng dưới đây thực hiện một truy vấn *WMI* để lấy danh sách tất cả các tác vụ in đang chờ trên máy tính và hiển thị *ID* cho mỗi tác vụ trong một `ListBox`. Khi người dùng chọn một item, một truy vấn đầy đủ hơn sẽ được thực hiện, và các chi tiết về tác vụ in này được hiển thị trong một `TextBox`. Cuối cùng, người dùng có thể nhấp nút *Pause* và *Resume* sau khi chọn một tác vụ để thay đổi trạng thái của nó.

```

using System;
using System.Windows.Forms;
using System.Management;
using System.Collections;

```

```

public class PrintQueueTest : System.Windows.Forms.Form {

    private System.Windows.Forms.ListBox lstJobs;
    private System.Windows.Forms.Button cmdRefresh;
    private System.Windows.Forms.TextBox txtJobInfo;
    private System.Windows.Forms.Button cmdPause;
    private System.Windows.Forms.Button cmdResume;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label2;
    // (Bỏ qua phần mã designer.)

    private void PrintQueueTest_Load(object sender, System.EventArgs e) {

        cmdRefresh_Click(null, null);
    }

    private void cmdRefresh_Click(object sender, System.EventArgs e) {

        // Chọn tất cả các tác vụ in đang chờ.
        string query = "SELECT * FROM Win32_PrintJob";
        ManagementObjectSearcher jobQuery =
            new ManagementObjectSearcher(query);
        ManagementObjectCollection jobs = jobQuery.Get();

        // Thêm các tác vụ trong hàng đợi vào ListBox.
        lstJobs.Items.Clear();
        txtJobInfo.Text = "";
        foreach (ManagementObject job in jobs) {
            lstJobs.Items.Add(job["JobID"]);
        }
    }

    // Phương thức này thực hiện truy vấn WMI và trả về
    // tác vụ WMI cho item hiện đang được chọn trong ListBox.
    private ManagementObject GetSelectedJob() {

```

```

try {

    // Chọn tác vụ in phù hợp.
    string query = "SELECT * FROM Win32_PrintJob " +
        "WHERE JobID='" + lstJobs.Text + "'";
    ManagementObjectSearcher jobQuery =
        new ManagementObjectSearcher(query);
    ManagementObjectCollection jobs = jobQuery.Get();
    IEnumerator enumerator = jobs.GetEnumerator();
    enumerator.MoveNext();
    return (ManagementObject)enumerator.Current;

} catch (InvalidOperationException){

    // Thuộc tính Current của enumerator không hợp lệ
    return null;

}

}

private void lstJobs_SelectedIndexChanged(object sender,
    System.EventArgs e) {

    ManagementObject job = GetSelectedJob();
    if (job == null) {
        txtJobInfo.Text = "";
        return;
    }

    // Hiển thị thông tin về tác vụ.
    string jobInfo = "Document: " + job["Document"].ToString();
    jobInfo += Environment.NewLine;
    jobInfo += "DriverName: " + job["DriverName"].ToString();
    jobInfo += Environment.NewLine;
    jobInfo += "Status: " + job["Status"].ToString();
    jobInfo += Environment.NewLine;
    jobInfo += "Owner: " + job["Owner"].ToString();

```

```

        jobInfo += Environment.NewLine;
        jobInfo += "PagesPrinted: " +
        job["PagesPrinted"].ToString();
        jobInfo += Environment.NewLine;
        jobInfo += "TotalPages: " + job["TotalPages"].ToString();

        if (job["JobStatus"] != null) {
            txtJobInfo.Text += Environment.NewLine;
            txtJobInfo.Text += "JobStatus: " +
                job["JobStatus"].ToString();
        }
        if (job["StartTime"] != null) {
            jobInfo += Environment.NewLine;
            jobInfo += "StartTime: " + job["StartTime"].ToString();
        }

        txtJobInfo.Text = jobInfo;
    }

    private void cmdPause_Click(object sender, System.EventArgs e) {

        if (lstJobs.SelectedIndex == -1) return;
        ManagementObject job = GetSelectedJob();
        if (job == null) return;

        // Tạm dừng tác vụ.
        int returnValue = Int32.Parse(
            job.InvokeMethod("Pause", null).ToString());

        // Hiển thị thông tin về giá trị trả về.
        if (returnValue == 0) {
            MessageBox.Show("Successfully paused job.");
        }
        else {
            MessageBox.Show(
                "Unrecognized return value when pausing job.");
        }
    }

```



```

    }
}

private void cmdResume_Click(object sender, System.EventArgs e) {

    if (lstJobs.SelectedIndex == -1) return;
    ManagementObject job = GetSelectedJob();
    if (job == null) return;

    if ((Int32.Parse(job["StatusMask"].ToString()) & 1) == 1) {

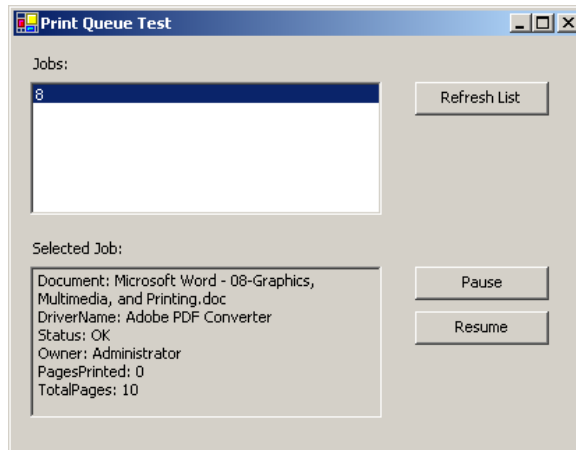
        // Phục hồi tác vụ.
        int returnValue = Int32.Parse(
            job.InvokeMethod("Resume", null).ToString());

        // Hiển thị thông tin về giá trị trả về.
        if (returnValue == 0) {
            MessageBox.Show("Successfully resumed job.");

        }else if (returnValue == 5) {
            MessageBox.Show("Access denied.");

        }else {
            MessageBox.Show(
                "Unrecognized return value when resuming job.");
        }
    }
}
}

```



Hình 8.14 Lấy thông tin từ hàng đợi in

Nhớ rằng các quyền *Windows* có thể ngăn bạn tạm dừng hoặc gỡ bỏ một tác vụ in do một người dùng khác tạo ra. Thực ra, các quyền này còn có thể ngăn bạn lấy thông tin trạng thái và có thể gây ra ngoại lệ.



Các phương thức *WMI* khác mà bạn có thể sử dụng trong một kịch bản in bao gồm *AddPrinterConnection*, *SetDefaultPrinter*, *CancelAllJobs*, và *PrintTestPage*, tất cả đều làm việc với lớp *Win32_Printer*. Để có thêm thông tin về *WMI*, bạn hãy tham khảo tài liệu *MSDN* tại địa chỉ [http://msdn.microsoft.com/library/en-us/wmisdk/wmi/computer_system_hardware_classes.asp].

18.

Sử dụng *Microsoft Agent*



Bạn muốn ứng dụng của mình có sự trợ giúp của những nhân vật hoạt hình như trong *Microsoft Word*, *Microsoft Excel*...



Sử dụng điều khiển *Microsoft Agent*.

Microsoft Agent là một kỹ thuật dùng để thêm các nhân vật hoạt hình có tính tương tác vào ứng dụng hay trang web. Tính tương tác là chức năng chính yếu của kỹ thuật *Microsoft Agent*: các nhân vật *Microsoft Agent* có thể nói và đáp lại đầu vào của người dùng thông qua việc nhận dạng và tổng hợp giọng nói. *Microsoft* sử dụng kỹ thuật này trong các ứng dụng như *Word*, *Excel*, và *PowerPoint*; giúp người dùng tìm câu trả lời cho những câu hỏi và hiểu cách thức hoạt động của ứng dụng. *Microsoft* cung cấp thông tin về kỹ thuật *Microsoft Agent* tại [www.microsoft.com/msagent].

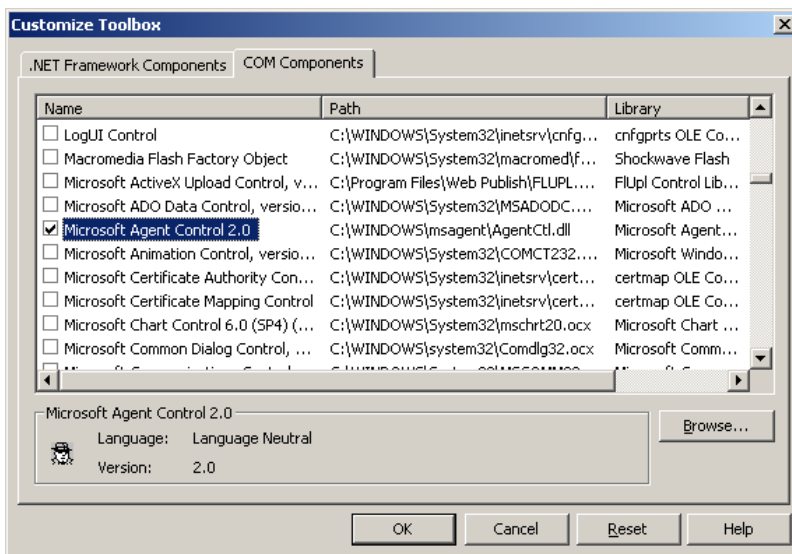
Điều khiển *Microsoft Agent* cho phép người dùng tương tác với các ứng dụng và trang web thông qua giọng nói—dạng giao tiếp tự nhiên nhất của con người. Khi người dùng nói vào micro, điều khiển này sử dụng một bộ máy nhận dạng giọng nói—đây là ứng dụng dịch âm thanh đầu vào từ micro thành ngôn ngữ mà máy tính hiểu được. Điều khiển *Microsoft Agent*

cũng sử dụng một bộ máy text-to-speech—đây là ứng dụng dịch các từ do người dùng nhập vào thành âm thanh có thể nghe được qua loa.

Điều khiển *Microsoft Agent* cho phép bạn truy xuất bốn nhân vật đã được định nghĩa sẵn—*Genie* (vị thần), *Merlin* (thuật sĩ), *Peedy* (con vẹt) và *Robby* (người máy). Mỗi nhân vật có một tập các hành động mà bạn có thể sử dụng trong ứng dụng nhằm minh họa các quan điểm hay chức năng khác nhau. Chẳng hạn, tập các hành động của *Peedy* gồm các dạng bay khác nhau mà bạn có thể sử dụng để dịch chuyển *Peedy* trên màn hình. Bạn cũng có thể tự tạo cho mình các nhân vật hoạt hình với sự trợ giúp của *Microsoft Agent Character Editor* và *Microsoft Linguistic Sound Editing Tool* (có trong đĩa CD đính kèm).

Ví dụ dưới đây minh họa cách xây dựng một ứng dụng đơn giản với điều khiển *Microsoft Agent*. Ứng dụng này gồm hai *ComboBox* dùng để chọn một nhân vật và một hành động. Khi người dùng chọn các *ComboBox* này, nhân vật được chọn sẽ xuất hiện và thực hiện hành động được chọn. Ứng dụng này sử dụng việc nhận dạng và tổng hợp giọng nói để điều khiển các hành động của nhân vật: người dùng có thể bảo nhân vật thực hiện hành động bằng cách nhấn phím [*Scroll Lock*] và rồi đọc tên hành động vào micro. Ví dụ này cũng cho phép người dùng chuyển sang một nhân vật mới bằng cách gọi tên nhân vật, và còn tạo thêm một lệnh tùy biến là *MoveToMouse*. Ngoài ra, nhân vật cũng sẽ đọc bất cứ text nào mà người dùng nhập vào *TextBox*. Trước khi chạy ví dụ này, bạn phải cài đặt điều khiển *Microsoft Agent*, bộ máy nhận dạng giọng nói, bộ máy text-to-speech, và các định nghĩa nhân vật (có trong đĩa CD đính kèm).

Để có thể thêm điều khiển *Microsoft Agent* vào dự án, bạn hãy nhấp phải vào hộp công cụ và chọn *Add/Remove Items*. Kế đó, vào thẻ *COM Components*, và chọn *Microsoft Agent 2.0*. Như thế, *Microsoft Agent* sẽ được thêm vào vào hộp công cụ. Khi bạn thả điều khiển này lên form, các *Interop Assembly* cần thiết sẽ được sinh ra và được thêm vào dự án.



Hình 8.15 Chọn Microsoft Agent Control 2.0 trong cửa sổ Customize Toolbox

```
public class FrmAgent : System.Windows.Forms.Form
```

```

{
    // (Bỏ qua phần mã designer.)

    // Đối tượng agent hiện tại.
    private AgentObjects.IAgentCtlCharacter mSpeaker;

    // Phương thức thụ lý sự kiện KeyDown của txtLocation.
    private void txtLocation_KeyDown(object sender,
        System.Windows.Forms.KeyEventArgs e)
    {
        if (e.KeyCode == Keys.Enter)
        {
            // Thiết lập nơi lưu trữ nhân vật vào txtLocation.
            string location = txtLocation.Text;

            // Khởi tạo các nhân vật.
            try
            {
                // Nạp các nhân vật vào đối tượng agent.
                mainAgent.Characters.Load("Genie", location & "Genie.acs")
                mainAgent.Characters.Load("Merlin", location + "Merlin.acs");
                mainAgent.Characters.Load("Peedy", location & "Peedy.acs")
                mainAgent.Characters.Load("Robby", location & "Robby.acs")

                // Vô hiệu txtLocation và kích hoạt các điều khiển khác.
                txtLocation.Enabled = false;
                txtSpeech.Enabled = true;
                cmdSpeak.Enabled = true;
                characterCombo.Enabled = true;
                actionsCombo.Enabled = true;

                // Thiết lập nhân vật hiện tại là Merlin và hiện nhân vật này.
                mSpeaker = mainAgent.Characters["Merlin"];
                GetAnimationNames(); // Thu lấy danh sách hành động.
            }
            catch { }
        }
    }
}

```

```

        mSpeaker.Show(0);
    }
    catch (FileNotFoundException)
    {
        MessageBox.Show("Invalid character location", "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

// Phương thức thụ lý sự kiện Click của nút Speak.
private void cmdSpeak_Click(System.Object sender, System.EventArgs e)
{
    // Nếu txtSpeech rỗng, nhân vật nhắc người dùng nhập text vào txtSpeech.
    if (txtSpeech.Text == "")
    {
        mSpeaker.Speak("Please type the words you want me to speak", "");
    }
    else
    {
        mSpeaker.Speak(txtSpeech.Text, "");
    }
}

// Phương thức thụ lý sự kiện Click của agent.
private void mainAgent_ClickEvent(object sender,
    AxAgentObjects._AgentEvents_ClickEvent e)
{
    mSpeaker.Play("Confused");
    mSpeaker.Speak("Why are you poking me?", "");
    mSpeaker.Play("RestPose");
}

// Phương thức thụ lý sự kiện SelectedIndexChanged của characterCombo
// (người dùng chọn nhân vật mới).
private void characterCombo_SelectedIndexChanged(System.Object sender,
    System.EventArgs e)

```

```

{
    ChangeCharacter(characterCombo.Text);
}

// Ẩn nhân vật hiện tại và hiện nhân vật mới.
private void ChangeCharacter(string name)
{
    mSpeaker.Hide(0);
    mSpeaker = mainAgent.Characters[name];
    GetAnimationNames(); // Sinh lại danh sách hành động.
    mSpeaker.Show(0);
}

// Thu lấy tên các hành động và đưa vào actionsCombo.
private void GetAnimationNames()
{
    // Bảo đảm tính an toàn về tiến trình.
    object synclockObject = (this);
    Monitor.Enter(synclockObject);
    try
    {
        // Lấy tên các hành động.
        IEnumerator enumerator =
            mainAgent.Characters.Character(mSpeaker.Name).
            AnimationNames.GetEnumerator();
        string voiceString;

        // Xóa actionsCombo.
        actionsCombo.Items.Clear();
        mSpeaker.Commands.RemoveAll();

        // Thêm tất cả các hành động (cho phép ra lệnh bằng giọng nói).
        while (enumerator.MoveNext())
        {
            voiceString = Convert.ToString(enumerator.Current);

```

```

        voiceString = voiceString.Replace("_", "underscore");
        actionsCombo.Items.Add(enumerator.Current);

        mSpeaker.Commands.Add(Convert.ToString(enumerator.Current), ""
            voiceString, true, false);
    }

    // Thêm lệnh tùy biến.
    mSpeaker.Commands.Add("MoveToMouse", "MoveToMouse",
        "Move To Mouse", true, true);
}
finally
{
    Monitor.Exit(synclockObject);
}
}

// Phương thức thụ lý sự kiện SelectedIndexChanged của actionsCombo
// (người dùng chọn hành động mới).
private void actionsCombo_SelectedIndexChanged(System.Object sender,
    System.EventArgs e)
{
    mSpeaker.Stop(null);
    mSpeaker.Play(actionsCombo.Text);
    mSpeaker.Play("RestPose");
}

// Phương thức thụ lý sự kiện Command của agent.
private void mainAgent_Command(System.Object sender,
    AxAgentObjects._AgentEvents_CommandEvent e)
{
    AgentObjects.IAgentCtlUserInput command =
        ((AgentObjects.IAgentCtlUserInput) (e.userInput));

    // Đổi nhân vật nếu người dùng đọc tên nhân vật.
    if (command.Voice == "Peedy" || command.Voice == "Robby" ||
        command.Voice == "Merlin" || command.Voice == "Genie")

```

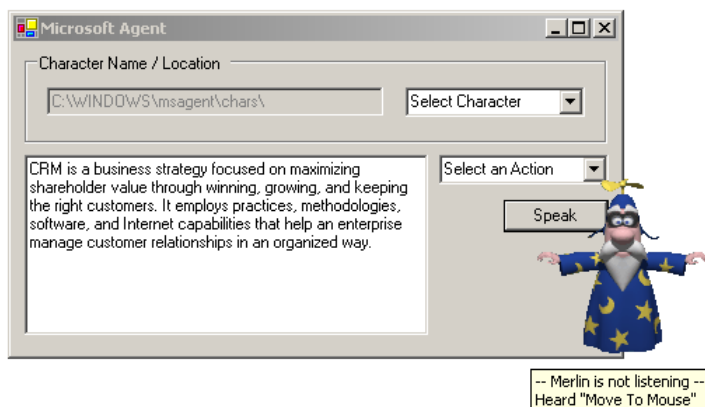
```

{
    ChangeCharacter(command.Voice);
    return;
}

// Gửi agent đến chuột.
if (command.Name == "MoveToMouse")
{
    mSpeaker.MoveTo(Convert.ToInt16(Cursor.Position.X - 60),
        Convert.ToInt16(Cursor.Position.Y - 60), null);
    return;
}

// Thực hiện hành động mới.
mSpeaker.Stop(null);
mSpeaker.Play(command.Name);
}
}

```



Hình 8.16 Ứng dụng thử nghiệm Microsoft Agent

9

FILE, THƯ MỤC, VÀ I/O

Các lớp I/O của *Microsoft .NET* gồm hai loại chính. Loại thứ nhất truy xuất thông tin từ hệ thống file và cho phép thực hiện các thao tác trên hệ thống file (như chép file, chuyển thư mục). Hai lớp tiêu biểu là `FileInfo` và `DirectoryInfo`. Loại thứ hai quan trọng hơn, gồm rất nhiều lớp cho phép đọc và ghi dữ liệu từ mọi kiểu stream. Stream có thể tương ứng với một file nhị phân hay văn bản, một file trong không gian lưu trữ riêng, một kết nối mạng, hoặc một vùng đệm bộ nhớ. Trong mọi trường hợp, cách thức tương tác với stream đều như nhau. Trong chương này, chúng ta sẽ xem xét các lớp hệ thống file và các lớp dựa-trên-stream.

Các mục trong chương này trình bày các vấn đề sau:

- Truy xuất và sửa đổi các thông tin của một file hay một thư mục (các mục 9.1, 9.2, 9.4, 9.5, và 9.16).
- Chép, di chuyển, xóa file hay thư mục (mục 9.3).
- Hiển thị động một cây thư mục trong một ứng dụng dựa-trên-*Windows* (mục 9.6) và sử dụng các hộp thoại file (mục 9.17).
- Đọc và ghi file văn bản (mục 9.7) và file nhị phân (mục 9.8), cũng như tạo file tạm (mục 9.15) và file trong một không gian lưu trữ riêng của người dùng (mục 9.18).
- Đọc file một cách bất đồng bộ (mục 9.9).
- Tìm file (mục 9.10) và kiểm tra hai file có trùng nhau hay không (mục 9.11).
- Làm việc với các chuỗi có chứa thông tin đường dẫn (mục 9.12 đến 9.14).
- Theo dõi sự thay đổi của hệ thống file (mục 9.19).
- Ghi ra cổng *COM* (mục 9.20).

1. Truy xuất các thông tin về file hay thư mục

? Bạn cần truy xuất các thông tin về một file hay một thư mục, chẳng hạn ngày tạo hay các thuộc tính của chúng.

✕ Tạo đối tượng `System.IO.FileInfo` cho file hay đối tượng `System.IO.DirectoryInfo` cho thư mục. Sau đó, truyền đường dẫn tới file hay thư mục đó trong phương thức khởi dựng. Các thông tin cần thiết sẽ được truy xuất thông qua các thuộc tính của đối tượng.

Để tạo một đối tượng `FileInfo` hay `DirectoryInfo`, bạn cần truyền đường dẫn tương đối hay đầy đủ trong phương thức khởi dựng của nó. Bạn có thể lấy các thông tin về file hay thư mục thông qua các thuộc tính của đối tượng tương ứng. Bảng 9.1 liệt kê các thành viên của lớp `FileInfo` và `DirectoryInfo`:

Bảng 9.1 Các thành viên của `FileInfo` và `DirectoryInfo`

Thành viên	Thuộc lớp	Mô tả
------------	-----------	-------

Exists	FileInfo và DirectoryInfo	Trả về true hay false, tùy thuộc vào file hay thư mục có tồn tại ở đường dẫn được chỉ định hay không.
Attributes	FileInfo và DirectoryInfo	Trả về một hoặc nhiều giá trị thuộc kiểu liệt kê System.IO.FileAttributes, cho biết các thuộc tính của file hay thư mục.
CreationTime, LastAccessTime, và LastWriteTime	FileInfo và DirectoryInfo	Trả về các thể hiện System.DateTime cho biết khi nào một file hay thư mục được tạo ra, truy xuất lần cuối, và cập nhật lần cuối.
FullName, Name, và Extension	FileInfo và DirectoryInfo	Trả về một chuỗi chứa tên đầy đủ, tên thư mục hay tên file (cùng phần mở rộng), và phần mở rộng.
Length	FileInfo	Trả về kích thước file (tính theo byte).
DirectoryName và Directory	FileInfo	DirectoryName trả về chuỗi chứa tên của thư mục cha; Directory trả về đối tượng DirectoryInfo mô tả thư mục cha, cho phép bạn truy xuất thêm thông tin về nó.
Parent và Root	DirectoryInfo	Trả về đối tượng DirectoryInfo mô tả thư mục cha hay thư mục gốc.
CreateSubdirectory	DirectoryInfo	Tạo một thư mục bên trong thư mục được mô tả bởi đối tượng DirectoryInfo, tên thư mục cần tạo được truyền cho phương thức. Trả về đối tượng DirectoryInfo mô tả thư mục con vừa tạo.
GetDirectories	DirectoryInfo	Trả về mảng các đối tượng DirectoryInfo, mỗi đối tượng mô tả một thư mục con trong thư mục này.
GetFiles	DirectoryInfo	Trả về mảng các đối tượng FileInfo, mỗi đối tượng mô tả một file trong thư mục này.

Chú ý hai điểm quan trọng khi sử dụng các đối tượng FileInfo và DirectoryInfo:

- Tất cả các thuộc tính của đối tượng FileInfo và DirectoryInfo được đọc ngay lần đầu tiên bạn truy xuất một thuộc tính nào đó. Nếu file hay thư mục thay đổi sau thời điểm này, bạn phải gọi phương thức Refresh để cập nhật các thuộc tính.
- Sẽ không có lỗi nếu bạn chỉ định một đường dẫn không tương ứng với một file hay thư mục đang tồn tại khi tạo một đối tượng FileInfo hay DirectoryInfo. Thay vào đó, bạn sẽ nhận được một đối tượng mô tả file hay thư mục không tồn tại—thuộc tính Exists của nó có giá trị false. Bạn có thể sử dụng đối tượng này để tạo file hay thư mục bằng cách gọi phương thức Create của nó. Nếu bạn truy xuất thuộc tính khác, ngoại lệ FileNotFoundException hay DirectoryNotFoundException sẽ bị ném.

Ứng dụng *Console* dưới đây nhận một đường dẫn file từ dòng lệnh, và rồi hiển thị thông tin về file và thư mục chứa file.

```
using System;
using System.IO;

public class FileInformation {

    private static void Main(string[] args) {

        if (args.Length == 0) {

            Console.WriteLine("Please supply a file name:");
            return;
        }

        FileInfo file = new FileInfo(args[0]);

        // Hiển thị thông tin file.
        Console.WriteLine("Checking file: " + file.Name);
        Console.WriteLine("File exists: " + file.Exists.ToString());

        if (file.Exists) {

            Console.Write("File created: ");
            Console.WriteLine(file.CreationTime.ToString());
            Console.Write("File last updated: ");
            Console.WriteLine(file.LastWriteTime.ToString());
            Console.Write("File last accessed: ");
            Console.WriteLine(file.LastAccessTime.ToString());
            Console.Write("File size (bytes): ");
            Console.WriteLine(file.Length.ToString());
            Console.Write("File attribute list: ");
            Console.WriteLine(file.Attributes.ToString());

        }

        Console.WriteLine();

        // Hiển thị thông tin thư mục.
        DirectoryInfo dir = file.Directory;
```

```

        Console.WriteLine("Checking directory: " + dir.Name);
        Console.WriteLine("In directory: " + dir.Parent.Name);
        Console.Write("Directory exists: ");
        Console.WriteLine(dir.Exists.ToString());

        if (dir.Exists) {
            Console.Write("Directory created: ");
            Console.WriteLine(dir.CreationTime.ToString());
            Console.Write("Directory last updated: ");
            Console.WriteLine(dir.LastWriteTime.ToString());
            Console.Write("Directory last accessed: ");
            Console.WriteLine(dir.LastAccessTime.ToString());
            Console.Write("Directory attribute list: ");
            Console.WriteLine(dir.Attributes.ToString());
            Console.WriteLine("Directory contains: " +
                dir.GetFiles().Length.ToString() + " files");
        }

        Console.ReadLine();
    }
}

```

Nếu bạn thực thi lệnh **FileInformation c:\windows\win.ini**, kết xuất có thể như sau:

```

Checking file: win.ini
File exists: True
File created: 2001-08-23 8:00:00 AM
File last updated: 2003-03-22 9:55:16 AM
File last accessed: 2003-05-26 2:21:53 PM
File size (bytes): 2128
File attribute list: Archive

```

```

Checking directory: windows
In directory: c:\
Directory exists: True
Directory created: 2000-01-01 8:03:33 AM

```



```
Directory last updated: 2003-05-26 2:25:25 PM
Directory last accessed: 2003-05-26 2:25:25 PM
Directory attribute list: Directory
Directory contains: 147 files
```



Bạn có thể sử dụng các phương thức tính của lớp File và Directory thay cho các phương thức của lớp FileInfo và DirectoryInfo, nhưng bạn phải truyền tên file hay đường dẫn mỗi lần gọi. Trong trường hợp thực hiện nhiều thao tác với cùng một file hay thư mục thì sử dụng các lớp FileInfo và DirectoryInfo nhanh hơn, vì chúng thực hiện kiểm tra bảo mật chỉ một lần.

2.

Thiết lập các thuộc tính của file và thư mục



Bạn cần kiểm tra hay thay đổi các thuộc tính của file hay thư mục.



Tạo đối tượng System.IO.FileInfo cho file hay tạo đối tượng System.IO.DirectoryInfo cho thư mục. Sau đó, sử dụng các toán tử AND (&) và OR (|) để thay đổi giá trị của thuộc tính Attributes.

Các thuộc tính FileInfo.Attributes và DirectoryInfo.Attributes mô tả các thuộc tính của file như *archive*, *system*, *hidden*, *read-only*, *compressed*, và *encrypted* (tham khảo thêm trong tài liệu MSDN). Vì một file có thể có nhiều thuộc tính nên Attributes là một tập các giá trị kiểu liệt kê. Để kiểm tra hay thay đổi một thuộc tính đơn lẻ, bạn cần sử dụng các phép toán trên bit.

Ví dụ sau nhận vào một file và kiểm tra thuộc tính *read-only*:

```
using System;

using System.IO;

public class Attributes {

    private static void Main() {

        // Giả sử file này có thuộc tính archive và read-only.
        FileInfo file = new FileInfo("data.txt");

        // Lệnh này sẽ hiển thị chuỗi "ReadOnly, Archive".
        Console.WriteLine(file.Attributes.ToString());

        // Điều kiện dưới đây sai, vì còn có thuộc tính khác
        // đã được thiết lập.
        if (file.Attributes == FileAttributes.ReadOnly) {
            Console.WriteLine("File is read-only (faulty test).");
        }
    }
}
```

```

    }

    // Điều kiện dưới đây đúng, vì nó chỉ lọc ra
    // thuộc tính read-only.
    if ((file.Attributes & FileAttributes.ReadOnly) ==
        FileAttributes.ReadOnly) {
        Console.WriteLine("File is read-only (correct test).");
    }

    Console.ReadLine();
}
}

```

 Để hiểu được ví dụ trên, bạn cần biết rằng `Attributes` được tạo thành (ở dạng nhị phân) bởi một dãy các chữ số 0 và 1, chẳng hạn *00010011*. Mỗi chữ số 1 cho biết một thuộc tính được thiết lập, mỗi chữ số 0 cho biết một thuộc tính không được thiết lập. Khi bạn sử dụng phép *AND*, nó sẽ so sánh mỗi chữ số này với mỗi chữ số tương ứng trong giá trị liệt kê. Ví dụ, nếu bạn *AND* giá trị *00100001* (mô tả thuộc tính *archive* và *read-only*) với giá trị liệt kê *00000001* (mô tả thuộc tính *read-only*), kết quả sẽ là *00000001* (chỉ có được chữ số 1 khi ở cả hai vị trí tương ứng đều là 1).

Khi thiết lập một thuộc tính, bạn cũng phải sử dụng phép toán trên bit. Trong trường hợp này, bạn cần cẩn thận để không vô ý xóa các thuộc tính khác.

```

// Chỉ thêm thuộc tính read-only.
file.Attributes = file.Attributes | FileAttributes.ReadOnly;

// Chỉ xóa thuộc tính read-only.
file.Attributes = file.Attributes & ~FileAttributes.ReadOnly;

```

3.

Chép, chuyển, xóa file hay thư mục

?

Bạn cần chép, chuyển, xóa một file hay thư mục

X

Tạo đối tượng `System.IO.FileInfo` cho file hay đối tượng `System.IO.DirectoryInfo` cho thư mục, truyền đường dẫn cho phương thức khởi dựng. Sử dụng các phương thức của đối tượng để chép, chuyển, xóa.

Các lớp `FileInfo` và `DirectoryInfo` cung cấp nhiều phương thức dùng để thao tác trên file và thư mục. Bảng 9.2 và 9.3 trình bày các phương thức của lớp `FileInfo` và `DirectoryInfo`.

Bảng 9.2 Các phương thức dùng để thao tác đối tượng `FileInfo`

Phương thức	Mô tả
-------------	-------

CopyTo	Chép một file sang đường dẫn mới, tên file được chỉ định trong đối số. Nó cũng trả về một đối tượng <code>FileInfo</code> mô tả file mới được chép. Bạn có thể truyền thêm một thông số tùy chọn có giá trị <code>true</code> để cho phép chép đè.
Create và CreateText	Create tạo file được chỉ định và trả về một đối tượng <code>FileStream</code> dùng để ghi ra file. CreateText cũng thực hiện như thế, nhưng trả về đối tượng <code>StreamWriter</code> gói lấy stream. Xem mục 9.7 và 9.8 để có thêm thông tin về việc ghi file.
Open, OpenRead, OpenText, và OpenWrite	Mở một file (nếu nó tồn tại). OpenRead và OpenText mở file trong chế độ chỉ-đọc, trả về một đối tượng <code>FileStream</code> hay <code>StreamReader</code> . OpenWrite mở file trong chế độ chỉ-ghi, trả về một đối tượng <code>FileStream</code> . Xem thêm mục 9.7 và 9.8 để có thêm thông tin về việc đọc file.
Delete	Xóa file (nếu nó tồn tại).
MoveTo	Chuyển một file đến đường dẫn mới, tên file được chỉ định trong đối số. MoveTo cũng được sử dụng để đổi tên một file mà không chuyển chỗ.

Bảng 9.3 Các phương thức dùng để thao tác đối tượng `DirectoryInfo`

Phương thức	Mô tả
Create	Tạo thư mục được chỉ định. Nếu đường dẫn chỉ định nhiều thư mục chưa tồn tại, tất cả sẽ được tạo một lượt.
CreateSubdirectory	Tạo một thư mục với tên cụ thể bên trong thư mục được mô tả bởi đối tượng <code>DirectoryInfo</code> . Nó cũng trả về một đối tượng <code>DirectoryInfo</code> mô tả thư mục con.
Delete	Xóa một thư mục (nếu nó tồn tại). Nếu muốn xóa một thư mục có chứa các thư mục khác, bạn phải sử dụng phương thức nạp chồng <code>Delete</code> chấp nhận một thông số có tên là <code>recursive</code> và thiết lập nó là <code>true</code> .
MoveTo	Chuyển một thư mục đến đường dẫn mới. MoveTo có thể được sử dụng để đổi tên một thư mục mà không chuyển chỗ.

Lớp `DirectoryInfo` không có phương thức nào dùng để sao chép thư mục. Tuy nhiên, bạn có thể dễ dàng viết được một phương thức như thế dựa trên kỹ thuật đệ quy và phương thức `CopyTo` của đối tượng `FileInfo`:

```
using System;
using System.IO;
```

```
public class FileSystemUtil {
```

```

public static void CopyDirectory(DirectoryInfo source,
    DirectoryInfo destination) {

    if (!destination.Exists) {
        destination.Create();
    }

    // Chép tất cả file.
    FileInfo[] files = source.GetFiles();
    foreach (FileInfo file in files) {

        file.CopyTo(Path.Combine(destination.FullName, file.Name));
    }

    // Xử lý các thư mục con.
    DirectoryInfo[] dirs = sourceDir.GetDirectories();
    foreach (DirectoryInfo dir in dirs) {

        // Lấy thư mục đích.
        string destinationDir = Path.Combine(destination.FullName,
            dir.Name);

        // Gọi đệ quy CopyDirectory().
        CopyDirectory(dir, new DirectoryInfo(destinationDir));
    }
}

```

Ví dụ sau sử dụng phương thức vừa viết ở trên để chép thư mục, đường dẫn các thư mục được truyền qua dòng lệnh:

```

public class CopyDir {

    private static void Main(string[] args) {

        if (args.Length != 2) {

```

```

        Console.WriteLine("usage:  " +
            "CopyDir [sourcePath] [destinationPath]");
        return;
    }

    DirectoryInfo sourceDir = new DirectoryInfo(args[0]);
    DirectoryInfo destinationDir = new DirectoryInfo(args[1]);

    FileSystemUtil.CopyDirectory(new DirectoryInfo(sourceDir),
        new DirectoryInfo(destinationDir));

    Console.WriteLine("Copy complete.");
    Console.ReadLine();
}
}

```

4.

Tính kích thước của thư mục



Bạn cần tính kích thước của tất cả file nằm trong một thư mục (hoặc cả trong các thư mục con của nó).



Duyệt qua tất cả file trong thư mục, tính tổng các thuộc tính `FileInfo.Length` của chúng. Sử dụng kỹ thuật đệ quy để tính cho cả các file nằm trong các thư mục con.

Lớp `DirectoryInfo` không có thuộc tính nào trả về thông tin kích thước. Tuy nhiên, bạn có thể dễ dàng tính được kích thước của tất cả các file nằm trong một thư mục bằng thuộc tính `FileInfo.Length`.

Phương thức dưới đây sử dụng kỹ thuật trên và có thể tùy chọn duyệt đệ quy qua các thư mục con:

```

using System;

using System.IO;

public class FileSystemUtil {

    public static long CalculateDirectorySize(DirectoryInfo directory,
        bool includeSubdirectories) {

        long totalSize = 0;

        // Duyệt tất cả các file trong thư mục.

```

```
FileInfo[] files = directory.GetFiles();
foreach (FileInfo file in files) {
    totalSize += file.Length;
}

// Duyệt tất cả các thư mục con.
if (includeSubdirectories) {

    DirectoryInfo[] dirs = directory.GetDirectories();
    foreach (DirectoryInfo dir in dirs) {
        totalSize += CalculateDirectorySize(dir, true);
    }
}

return totalSize;
}
}
```

Và dưới đây là ứng dụng thử nghiệm phương thức trên:

```
using System;
using System.IO;

public class CalculateDirSize {

    private static void Main(string[] args) {

        if (args.Length == 0) {

            Console.WriteLine("Please supply a directory path.");
            return;
        }

        DirectoryInfo dir = new DirectoryInfo(args[0]);
        Console.WriteLine("Total size: " +
            FileSystemUtil.CalculateDirectorySize(dir, true).ToString() +
            " bytes.");
        Console.ReadLine();
    }
}
```

```
    }
}
```

5. Truy xuất thông tin phiên bản của file

? Bạn cần truy xuất các thông tin về phiên bản của file như *publisher, revision number, comment...*

✂ Sử dụng phương thức tĩnh `GetVersionInfo` của lớp `System.Diagnostics.FileVersionInfo`.

.NET Framework cho phép bạn truy xuất các thông tin về file mà không cần dựa vào *Windows API*. Bạn chỉ cần sử dụng lớp `FileVersionInfo` và gọi phương thức `GetVersionInfo` với đối số là tên file. Kể đó, bạn có thể truy xuất thông tin thông qua các thuộc tính của `FileVersionInfo`.

```
using System;
using System.Diagnostics;

public class VersionInfo {

    private static void Main(string[] args) {

        if (args.Length == 0) {

            Console.WriteLine("Please supply a file name.");
            return;
        }

        FileVersionInfo info = FileVersionInfo.GetVersionInfo(args[0]);

        // Hiển thị các thông tin về phiên bản.
        Console.WriteLine("Checking File: " + info.FileName);
        Console.WriteLine("Product Name: " + info.ProductName);
        Console.WriteLine("Product Version: " + info.ProductVersion);
        Console.WriteLine("Company Name: " + info.CompanyName);
        Console.WriteLine("File Version: " + info.FileVersion);
        Console.WriteLine("File Description: " + info.FileDescription);
        Console.WriteLine("Original Filename: " + info.OriginalFilename);
        Console.WriteLine("Legal Copyright: " + info.LegalCopyright);
        Console.WriteLine("InternalName: " + info.InternalName);
        Console.WriteLine("IsDebug: " + info.IsDebug);
```

```

        Console.WriteLine("IsPatched: " + info.IsPatched);
        Console.WriteLine("IsPreRelease: " + info.IsPreRelease);
        Console.WriteLine("IsPrivateBuild: " + info.IsPrivateBuild);
        Console.WriteLine("IsSpecialBuild: " + info.IsSpecialBuild);

        Console.ReadLine();
    }
}

```

Dưới đây là kết xuất khi bạn chạy lệnh **VersionInfo c:\windows\explorer.exe**:

```

Checking File: c:\windows\explorer.exe

Product Name: Microsoft® Windows® Operating System
Product Version: 6.00.2600.0000
Company Name: Microsoft Corporation
File Version: 6.00.2600.0000 (xpclient.010817-1148)
File Description: Windows Explorer
Original Filename: EXPLORER.EXE
Legal Copyright: © Microsoft Corporation. All rights reserved.
InternalName: explorer
IsDebug: False
IsPatched: False
IsPreRelease: False
IsPrivateBuild: False
IsSpecialBuild: False®

```

6. *Sử dụng TreeView để hiển thị cây thư mục just-in-time*

- ?** Bạn cần hiển thị một cây thư mục trong **TreeView**. Tuy nhiên, việc lấp đầy cấu trúc cây thư mục khi khởi động tốn quá nhiều thời gian.
- X** Thêm cấp thư mục đầu tiên vào **TreeView**, và thêm một nút giả (ẩn) vào mỗi nhánh. Phản ứng lại sự kiện **TreeView.BeforeExpand** để thêm các thư mục con vào một nhánh trước khi nó được hiển thị.

Bạn có thể sử dụng kỹ thuật đệ quy để xây dựng toàn bộ cây thư mục. Tuy nhiên, việc quét hệ thống file theo cách này có thể chậm, đặc biệt đối với các ổ đĩa lớn. Vì lý do này, các phần mềm quản lý file chuyên nghiệp (bao gồm *Windows Explorer*) sử dụng một kỹ thuật khác: chỉ hiển thị những thông tin nào người dùng cần đến.

TreeView rất thích hợp với cách tiếp cận này vì nó cung cấp sự kiện `BeforeExpand` (sự kiện này phát sinh trước khi một cấp mới được hiển thị). Bạn có thể sử dụng một *placeholder* (như dấu hoa thị hay nút rỗng) trong tất cả các nhánh chưa được thêm vào. Điều này cho phép bạn thêm vào các phần của cây thư mục khi chúng được hiển thị.

Để sử dụng kiểu giải pháp này, bạn cần ba yếu tố sau:

- Phương thức `Fill`—thêm một cấp mới vào một thư mục. Bạn sẽ sử dụng phương thức này để thêm vào các cấp khi chúng được mở rộng.
- Phương thức thụ lý sự kiện `Form.Load`—sử dụng `Fill` để tạo cây với cấp đầu tiên.
- Phương thức thụ lý sự kiện `TreeView.BeforeExpand`—phản ứng khi người dùng mở rộng một nút và gọi `Fill` nếu thông tin của thư mục này chưa được thêm.

Dưới đây là phần mã cho form:

```
using System;
using System.IO;
using System.Drawing;
using System.Windows.Forms;

public class DirectoryTree : System.Windows.Forms.Form {

    private System.Windows.Forms.TreeView treeDirectory;

    // (Bỏ qua phần mã designer.)

    private void Fill(TreeNode dirNode) {

        DirectoryInfo dir = new DirectoryInfo(dirNode.FullPath);

        // Một ngoại lệ có thể bị ném nếu bạn không có
        // đủ quyền thao tác trên file hay thư mục.
        // Bạn có thể bắt và bỏ qua ngoại lệ này.
        foreach (DirectoryInfo dirItem in dir.GetDirectories()) {

            // Thêm nút giả cho thư mục.
            TreeNode newNode = new TreeNode(dirItem.Name);
            dirNode.Nodes.Add(newNode);
            newNode.Nodes.Add("");

        }
    }

    private void DirectoryTree_Load(object sender, System.EventArgs e) {
```

```

// Thiết lập nút đầu tiên.
TreeNode rootNode = new TreeNode("C:\\");
treeDirectory.Nodes.Add(rootNode);

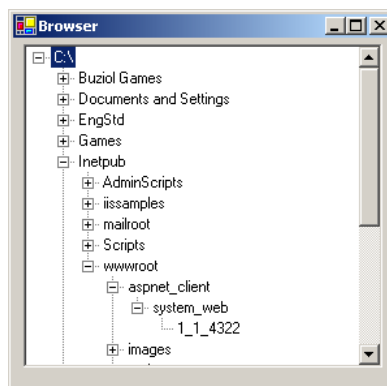
// Thêm cấp thứ nhất và mở rộng nó.
Fill(rootNode);
treeDirectory.Nodes[0].Expand();
}

private void treeDirectory_BeforeExpand(object sender,
    System.Windows.Forms.TreeViewCancelEventArgs e) {

    // Nếu tìm thấy một nút giả, xóa nó và đọc các thư mục thật.
    if (e.Node.Nodes[0].Text == "**") {

        e.Node.Nodes.Clear();
        Fill(e.Node);
    }
}
}

```



Hình 9.1 Một cây thư mục với TreeView

7.

Đọc và ghi file văn bản

- ?** Bạn cần ghi dữ liệu vào một file văn bản theo kiểu mã hóa *ASCII*, *Unicode*, hay *UTF-8*.
- X** Tạo một đối tượng `System.IO.FileStream` tham chiếu đến file. Để ghi file, hãy gọi `FileStream` trong một `System.IO.StreamWriter` và sử dụng phương thức nạp chồng `Write`. Để đọc file, hãy gọi `FileStream` trong một `System.IO.StreamReader` và sử dụng phương thức `Read` hay `ReadLine`.

.NET cho phép bạn ghi hay đọc văn bản bằng lớp `StreamWriter` và `StreamReader`. Khi ghi dữ liệu với `StreamWriter`, hãy sử dụng phương thức `StreamWriter.Write`. Phương thức này được nạp chồng để hỗ trợ tất cả các kiểu dữ liệu thông thường trong *C# .NET*, bao gồm chuỗi, ký tự, số nguyên, số thực dấu chấm động, số thập phân,... Tuy nhiên, phương thức `Write` luôn chuyển dữ liệu thành văn bản. Nếu muốn chuyển văn bản này trở về kiểu ban đầu thì bạn nên sử dụng `WriteLine` để bảo đảm mỗi giá trị được đặt trên một dòng riêng.

Có nhiều cách mô tả một chuỗi dưới dạng nhị phân, tùy thuộc vào cách mã hóa. Các kiểu mã hóa thông thường là:

- *ASCII*—sử dụng 7 bit để mã hóa mỗi ký tự trong chuỗi. Dữ liệu được mã hóa theo *ASCII* không thể chứa các ký tự *Unicode* mở rộng. Khi sử dụng kiểu mã hóa *ASCII* trong *.NET*, các bit được đệm thêm để mảng byte kết quả sẽ có 1 byte cho mỗi ký tự.
- *Full Unicode*, hay *UTF-16*—sử dụng 16 bit để mã hóa mỗi ký tự trong chuỗi, nên mảng byte kết quả sẽ có 2 byte cho mỗi ký tự.
- *UTF-7 Unicode*—sử dụng 7 bit cho các ký tự *ASCII* bình thường và nhiều cặp 7 bit cho các ký tự mở rộng. Kiểu mã hóa này chủ yếu dùng cho các giao thức 7 bit, chẳng hạn *mail*.
- *UTF-8 Unicode*—sử dụng 8 bit cho các ký tự *ASCII* bình thường và nhiều cặp 8 bit cho các ký tự mở rộng. Mảng byte kết quả sẽ có 1 byte cho mỗi ký tự (giả sử không có ký tự mở rộng).

.NET cung cấp một lớp cho mỗi kiểu mã hóa trong không gian tên `System.Text`. Khi sử dụng `StreamReader` và `StreamWriter`, bạn có thể chỉ định kiểu mã hóa hoặc sử dụng kiểu mặc định là *UTF-8*.

Khi đọc thông tin, sử dụng phương thức `Read` hay `ReadLine` của lớp `StreamReader`. Phương thức `Read` đọc một ký tự, hay số ký tự do bạn chỉ định, và trả về một ký tự hay mảng ký tự. Phương thức `ReadLine` trả về một chuỗi chứa toàn bộ nội dung một hàng.

Ứng dụng *Console* dưới đây minh họa việc ghi và đọc một file văn bản:

```
using System;
using System.IO;
using System.Text;

public class TextFileTest {
```

```
private static void Main() {

    // Tạo file mới.
    FileStream fs = new FileStream("test.txt", FileMode.Create);

    // Tạo một writer và chỉ định kiểu mã hóa.
    // Kiểu mặc định (UTF-8) hỗ trợ ký tự Unicode,
    // nhưng mã hóa các ký tự chuẩn giống như ASCII
    StreamWriter w = new StreamWriter(fs, Encoding.UTF8);

    // Ghi một số thập phân, một chuỗi, và một ký tự.
    w.WriteLine(124.23M);
    w.WriteLine("Test string");
    w.WriteLine('!');

    // Bảo đảm tất cả dữ liệu được ghi từ buffer.
    w.Flush();

    // Đóng file.
    w.Close();
    fs.Close();

    Console.WriteLine("Press Enter to read the information.");
    Console.ReadLine();

    // Mở file trong chế độ chỉ-đọc.
    fs = new FileStream("test.txt", FileMode.Open);
    StreamReader r = new StreamReader(fs, Encoding.UTF8);

    // Đọc dữ liệu và chuyển nó về kiểu thích hợp.
    Console.WriteLine(Decimal.Parse(r.ReadLine()));
    Console.WriteLine(r.ReadLine());
    Console.WriteLine(Char.Parse(r.ReadLine()));

    r.Close();
    fs.Close();
}
```

```

        Console.ReadLine();
    }
}

```

8.

Đọc và ghi file nhị phân



Bạn cần ghi dữ liệu vào file nhị phân với kiểu dữ liệu mạnh.



Tạo một đối tượng `System.IO.FileStream` tham chiếu đến file. Để ghi file, hãy gói `FileStream` trong một `System.IO.BinaryWriter` và sử dụng phương thức nạp chồng `Write`. Để đọc file, hãy gói `FileStream` trong một `System.IO.BinaryReader` và sử dụng phương thức `Read` phù hợp với kiểu dữ liệu.

.NET cho phép bạn ghi hay đọc dữ liệu nhị phân bằng lớp `BinaryWriter` và `BinaryReader`. Khi ghi dữ liệu với `BinaryWriter`, hãy sử dụng phương thức `BinaryWriter.Write`. Phương thức này được nạp chồng để hỗ trợ tất cả kiểu dữ liệu thông thường trong *C#.NET*, bao gồm chuỗi, ký tự, số nguyên, số thực dấu chấm động, số thập phân,... Thông tin sau đó được mã hóa thành một dãy các byte và ghi vào file. Bạn có thể chỉ định kiểu mã hóa cho chuỗi bằng một phương thức khởi dựng nạp chồng nhận một đối tượng `System.Text.Encoding` làm đối số (đã được mô tả trong mục 9.7).

Sử dụng file nhị phân để thao tác với các kiểu dữ liệu thì khá phức tạp, vì khi truy xuất thông tin, bạn phải sử dụng một trong những phương thức `Read` kiểu mạnh của `BinaryReader`. Ví dụ: muốn truy xuất dữ liệu dạng thập phân thì phải sử dụng `ReadDecimal`; còn muốn đọc một chuỗi thì phải sử dụng `ReadString` (`BinaryWriter` luôn ghi lại chiều dài của chuỗi khi ghi chuỗi vào file để tránh lỗi).

Ứng dụng *Console* dưới đây minh họa việc ghi và đọc một file nhị phân:

```

using System;

using System.IO;

public class BinaryFileTest {

    private static void Main() {

        // Tạo file và tạo writer.
        FileStream fs = new FileStream("test.txt", FileMode.Create);
        BinaryWriter w = new BinaryWriter(fs);

        // Ghi một số thập phân, hai chuỗi, và một ký tự.
        w.Write(124.23M);
        w.Write("Test string");
    }
}

```

```
w.Write("Test string 2");
w.Write('!');

// Bảo đảm tất cả dữ liệu được ghi từ buffer.
w.Flush();

// Đóng file.
w.Close();
fs.Close();

Console.WriteLine("Press Enter to read the information.");
Console.ReadLine();

// Mở file trong chế độ chỉ-đọc.
fs = new FileStream("test.txt", FileMode.Open);

// Hiển thị dữ liệu thô trong file.
StreamReader sr = new StreamReader(fs);
Console.WriteLine(sr.ReadToEnd());
Console.WriteLine();

// Đọc dữ liệu và chuyển nó về kiểu thích hợp.
fs.Position = 0;
BinaryReader br = new BinaryReader(fs);
Console.WriteLine(br.ReadDecimal());
Console.WriteLine(br.ReadString());
Console.WriteLine(br.ReadString());
Console.WriteLine(br.ReadChar());

fs.Close();

Console.ReadLine();
}
}
```

9.

Đọc file một cách bất đồng bộ

- ?** Bạn cần đọc dữ liệu từ một file mà không phải dừng quá trình thực thi mã lệnh của bạn. Kỹ thuật này thường được sử dụng khi file được lưu trữ trong một nơi có tốc độ truy xuất chậm (chẳng hạn một đĩa mạng).
- X** Tạo một lớp để đọc file một cách bất đồng bộ. Bắt đầu đọc một khối dữ liệu bằng phương thức `FileStream.BeginRead`, và truyền phương thức callback. Khi callback được kích hoạt, gọi `FileStream.EndRead` để truy xuất dữ liệu, xử lý nó, và đọc khối dữ liệu kế tiếp với `BeginRead`.

`FileStream` hỗ trợ hoạt động bất đồng bộ thông qua phương thức `BeginRead` và `EndRead`. Sử dụng các phương thức này, bạn có thể đọc một khối dữ liệu trên một trong các tiểu trình do thread-pool cung cấp mà không cần sử dụng trực tiếp các lớp tiểu trình trong không gian tên `System.Threading`.

Khi đọc file một cách bất đồng bộ, bạn cần xác định kích thước khối dữ liệu trong một lần đọc. Tùy trường hợp, bạn có thể muốn đọc một khối dữ liệu nhỏ (ví dụ, chép từng khối một sang file khác) hoặc khối dữ liệu tương đối lớn (ví dụ, bạn cần một lượng thông tin nhất định trước khi xử lý việc gì đó). Bạn chỉ định kích thước khối khi gọi `BeginRead`, và truyền một bộ đệm để chứa dữ liệu. Vì `BeginRead` và `EndRead` cần truy xuất nhiều mẫu thông tin giống nhau (chẳng hạn `FileStream`, bộ đệm, kích thước khối,...), bạn nên đóng gói mã lệnh đọc file bất đồng bộ trong một lớp.

Với lớp `AsyncProcessor` trong ví dụ dưới đây, phương thức công khai `StartProcess` bắt đầu quá trình đọc bất đồng bộ. Mỗi khi quá trình đọc hoàn tất, `OnCompletedRead` được kích hoạt và khối dữ liệu được xử lý. Nếu còn dữ liệu trong file, một quá trình đọc bất đồng bộ mới sẽ được khởi chạy. Kích thước khối bộ nhớ là 2 KB (2048 byte).

```
using System;
using System.IO;
using System.Threading;

public class AsyncProcessor {

    private Stream inputStream;

    // Kích thước mỗi khối dữ liệu là 2 KB.
    private int bufferSize = 2048;

    public int BufferSize {
        get {return bufferSize;}
        set {bufferSize = value;}
    }
}
```

```
// Bộ đệm chứa dữ liệu.
private byte[] buffer;

public AsyncProcessor(string fileName) {

    buffer = new byte[bufferSize];

    // Mở file, truyền giá trị true để hỗ trợ truy xuất bất đồng bộ.
    inputStream = new FileStream(fileName, FileMode.Open,
        FileAccess.Read, FileShare.Read, bufferSize, true);
}

public void StartProcess() {

    // Bắt đầu quá trình đọc bất đồng bộ.
    inputStream.BeginRead(buffer, 0, buffer.Length,
        new AsyncCallback(OnCompletedRead), null);
}

private void OnCompletedRead(IAsyncResult asyncResult) {

    // Một khối đã được đọc. Truy xuất dữ liệu.
    int bytesRead = inputStream.EndRead(asyncResult);

    // Nếu không đọc được byte nào, stream đang ở cuối file.
    if (bytesRead > 0) {

        // Tạm dừng để giả lập việc xử lý dữ liệu.
        Console.WriteLine("\t[ASYNC READER]: Read one block.");
        Thread.Sleep(TimeSpan.FromMilliseconds(20));

        // Bắt đầu đọc khối kế tiếp.
        inputStream.BeginRead(buffer, 0, buffer.Length,
            new AsyncCallback(OnCompletedRead), null);
    }
    else {
```



```

        // Kết thúc.
        Console.WriteLine("\t[ASYNC READER]: Complete.");
        inputStream.Close();
    }
}

```

Ví dụ dưới sử dụng `AsyncProcessor` để đọc một file có kích thước 1 MB.

```

public class AsynchronousIO {

    public static void Main() {

        // Tạo file thử nghiệm có kích thước 1MB.
        FileStream fs = new FileStream("test.txt", FileMode.Create);
        fs.SetLength(1000000);
        fs.Close();

        // Bắt đầu xử lý bất đồng bộ trên một tiến trình khác.
        AsyncProcessor asyncIO = new AsyncProcessor("test.txt");
        asyncIO.StartProcess();

        // Cùng thời điểm này, thực hiện công việc khác.
        // Ở đây chúng ta sẽ lặp trong 10 giây.
        DateTime startTime = DateTime.Now;
        while (DateTime.Now.Subtract(startTime).TotalSeconds < 10) {

            Console.WriteLine("[MAIN THREAD]: Doing some work.");

            // Tạm dừng để giả lập một thao tác tốn nhiều thời gian.
            Thread.Sleep(TimeSpan.FromMilliseconds(100));
        }

        Console.WriteLine("[MAIN THREAD]: Complete.");
        Console.ReadLine();

        // Xóa file thử.
        File.Delete("test.txt");
    }
}

```

```
}
```

Và đây là kết xuất khi chạy ứng dụng thử nghiệm:

```
[MAIN THREAD]: Doing some work.
    [ASYNC READER]: Read one block.
    [ASYNC READER]: Read one block.
[MAIN THREAD]: Doing some work.
    [ASYNC READER]: Read one block.
    [ASYNC READER]: Read one block.
    [ASYNC READER]: Read one block.
    [ASYNC READER]: Read one block.
[MAIN THREAD]: Doing some work.
    [ASYNC READER]: Read one block.
    [ASYNC READER]: Read one block.
    [ASYNC READER]: Read one block.
. . .
```

10.

Tìm file phù hợp một biểu thức wildcard

?

Bạn cần xử lý nhiều file có điểm chung, dựa vào biểu thức lọc như **.dll* hay *mysheet20???.xls*.

✕

Sử dụng phiên bản nạp chồng của phương thức `System.IO.DirectoryInfo.GetFiles` nhận một biểu thức lọc và trả về một mảng các đối tượng `FileInfo`.

Các đối tượng `DirectoryInfo` và `Directory` đều cho phép dò trong thư mục hiện hành để tìm các file phù hợp với một biểu thức lọc. Các biểu thức này thường sử dụng các ký tự wildcard như `?` và `*`. Bạn cũng có thể sử dụng kỹ thuật tương tự để lấy các thư mục phù hợp với một mẫu nhất định bằng phương thức nạp chồng `DirectoryInfo.GetDirectories`.

Ví dụ dưới đây sẽ lấy tên của tất cả các file trong một thư mục phù hợp với một biểu thức lọc. Thư mục và biểu thức lọc được truyền qua dòng lệnh.

```
using System;

using System.IO;

public class WildcardTest {

    private static void Main(string[] args) {

        if (args.Length != 2) {

            Console.WriteLine(
```

```

        "USAGE: WildcardTest [directory] [filterExpression]");
    return;
}

DirectoryInfo dir = new DirectoryInfo(args[0]);
FileInfo[] files = dir.GetFiles(args[1]);

// Hiển thị tên và kích thước file.
foreach (FileInfo file in files) {
    Console.WriteLine("Name: " + file.Name + " ");
    Console.WriteLine("Size: " + file.Length.ToString());
}

Console.ReadLine();
}
}

```

Nếu muốn tìm trong thư mục con, bạn cần sử dụng đệ quy. Nhiều mục trong chương sử dụng kỹ thuật đệ quy để xử lý file, chẳng hạn 9.3 và 9.4.

11. *Kiểm tra hai file có trùng nhau hay không*

? Bạn cần so sánh nội dung của hai file và xác định chúng có trùng nhau hay không.

✕ Tính mã băm của mỗi file bằng lớp `System.Security.Cryptography.HashAlgorithm` rồi so sánh các mã băm.

Có nhiều cách để so sánh nhiều file. Ví dụ, có thể xét một phần của file xem có giống nhau, hoặc đọc cả file so sánh từng byte. Cả hai cách trên đều đúng, nhưng trong một số trường hợp, sử dụng mã băm thuận tiện hơn.

Một giải thuật băm sinh ra một dạng nhị phân đặc trưng (với kích thước nhỏ, thường khoảng 20 byte) cho file. Có khả năng hai file khác nhau có cùng mã băm, nhưng khả năng này hầu như không xảy ra. Thực tế, cả những thay đổi nhỏ nhất (chẳng hạn, chỉ thay đổi một bit của file nguồn) cũng có 50% khả năng thay đổi các bit của mã băm. Do đó, mã băm thường được sử dụng để phát hiện dữ liệu bị sửa đổi (mã băm sẽ được đề cập chi tiết hơn trong chương 14).

Để tạo một mã băm, trước hết bạn phải tạo một đối tượng `HashAlgorithm` bằng phương thức tĩnh `HashAlgorithm.Create`. Sau đó gọi `HashAlgorithm.ComputeHash` để nhận một mảng byte chứa mã băm.

Ví dụ dưới đây đọc hai tên file từ đối số dòng lệnh và kiểm tra hai file này có trùng nhau hay không:

```

using System;
using System.IO;

```

```
using System.Security.Cryptography;

public class CompareFiles {

    private static void Main(string[] args) {

        if (args.Length != 2) {
            Console.WriteLine("USAGE: CompareFiles [fileName] " +
                               [fileName]);
            return;
        }

        Console.WriteLine("Comparing " + args[0] + " and " + args[1]);

        // Tạo đối tượng băm.
        HashAlgorithm hashAlg = HashAlgorithm.Create();

        // Tính mã băm cho file thứ nhất.
        FileStream fsA = new FileStream(args[0], FileMode.Open);
        byte[] hashBytesA = hashAlg.ComputeHash(fsA);
        fsA.Close();

        // Tính mã băm cho file thứ hai.
        FileStream fsB = new FileStream(args[1], FileMode.Open);
        byte[] hashBytesB = hashAlg.ComputeHash(fsB);
        fsB.Close();

        // So sánh mã băm.
        if (BitConverter.ToString(hashBytesA) ==
            BitConverter.ToString(hashBytesB)) {

            Console.WriteLine("Files match.");

        }else {

            Console.WriteLine("No match.");
        }
    }
}
```

```

        Console.ReadLine();
    }
}

```

Các mã băm được so sánh bằng cách chuyển chúng thành chuỗi. Bạn cũng có thể duyệt qua mảng và so sánh từng byte. Cách này nhanh hơn một ít, nhưng việc chuyển 20 byte thành chuỗi không tốn nhiều chi phí nên không cần thiết.

12.

Thao tác trên đường dẫn file

? Bạn cần lấy một phần đường dẫn file hoặc kiểm tra một đường dẫn file có ở dạng chuẩn hay không.

X Xử lý đường dẫn bằng lớp `System.IO.Path`. Bạn có thể sử dụng `Path.GetFileName` để lấy tên file từ đường dẫn, `Path.ChangeExtension` để thay đổi phần mở rộng của đường dẫn, và `Path.Combine` để tạo đường dẫn đầy đủ mà không cần quan tâm thư mục của bạn đã có ký tự phân cách thư mục (`\`) hay chưa.

Thường khó thao tác với các đường dẫn file vì có vô số cách để mô tả một thư mục. Ví dụ, bạn có thể sử dụng đường dẫn tuyệt đối (`C:\Temp`), đường dẫn UNC (`\\MyServer\MyShare\Temp`), hoặc một trong các đường dẫn tương đối (`C:\Temp\MyFiles\..\` hay `C:\Temp\MyFiles\..\Temp`).

Cách dễ nhất để xử lý các đường dẫn file là sử dụng các phương thức tĩnh của lớp `Path` để bảo đảm có thông tin đúng. Ví dụ, đoạn mã sau trích tên file từ một đường dẫn file:

```

string filename = @"..\System\MyFile.txt";
filename = Path.GetFileName(filename);

```

// filename bây giờ là "MyFile.txt".

Và đoạn mã sau sử dụng `Path.Combine` để thêm tên file vào đường dẫn thư mục:

```

string filename = @"..\..\myfile.txt";
string fullPath = @"c:\Temp";

```

```

filename = Path.GetFileName(filename);
fullPath = Path.Combine(fullPath, filename);

```

// fullPath bây giờ là "c:\Temp\myfile.txt".

Cách này có ưu điểm là ký tự phân cách thư mục (`\`) sẽ tự động được thêm vào đường dẫn nếu cần thiết.

Lớp `Path` cũng cung cấp các phương thức hữu ích sau đây để thao tác trên thông tin đường dẫn:

- `ChangeExtension`—thay đổi phần mở rộng của file. Nếu phần mở rộng mới không được chỉ định, phần mở rộng hiện tại sẽ bị xóa.

- `GetDirectoryName`—trả về thông tin của các thư mục nằm giữa ký tự phân cách thư mục (`\`) đầu và cuối.
- `GetFileNameWithoutExtension`—tương tự như `GetFileName`, nhưng bỏ phần mở rộng.
- `GetFullPath`—không có tác dụng đối với đường dẫn tuyệt đối, và nó sử dụng thư mục hiện hành để đổi một đường dẫn tương đối thành đường dẫn tuyệt đối. Ví dụ, nếu `C:\Temp\` là thư mục hiện hành, gọi `GetFullPath` cho file `test.txt` sẽ trả về `C:\Temp\test.txt`.
- `GetPathRoot`—trả về chuỗi chứa thư mục gốc (ví dụ, `"C:\"`). Đối với đường dẫn tương đối, nó trả về tham chiếu rỗng.
- `HasExtension`—trả về `true` nếu đường dẫn kết thúc với phần mở rộng.
- `IsPathRooted`—trả về `true` nếu đường dẫn là tuyệt đối, `false` nếu đường dẫn là tương đối.



Trong hầu hết trường hợp, một ngoại lệ sẽ bị ném nếu bạn truyền đường dẫn không hợp lệ cho một trong các phương thức này (chẳng hạn, đường dẫn có chứa các ký tự không hợp lệ). Tuy nhiên, những đường dẫn không hợp lệ do chứa các ký tự wildcard sẽ không làm sinh ra ngoại lệ.

13. Xác định đường dẫn tương ứng với một file hay thư mục



Bạn có một đường dẫn (ở dạng chuỗi), và cần xác định nó tương ứng với một thư mục hay một file.



Kiểm tra đường dẫn với phương thức `Directory.Exists` và `File.Exists`.

Cả hai lớp `System.IO.Directory` và `System.IO.File` đều có phương thức `Exists`.

- `Directory.Exists`—trả về `true` nếu đường dẫn (tương đối hoặc tuyệt đối) tương ứng với một thư mục đang tồn tại.
- `File.Exists`—trả về `true` nếu đường dẫn tương ứng với một file đang tồn tại.

Sử dụng hai phương thức này, bạn có thể nhanh chóng xác định một đường dẫn có tương ứng với một file hay thư mục hay không, như ví dụ sau:

```
using System;

using System.IO;

public class FileOrPath {

    private static void Main(string[] args) {

        foreach (string arg in args) {
```

```

        Console.Write(arg);

        if (Directory.Exists(arg)) {
            Console.WriteLine(" is a directory");
        }
        else if (File.Exists(arg)) {
            Console.WriteLine(" is a file");
        }
        else {
            Console.WriteLine(" does not exist");
        }
    }
    Console.ReadLine();
}
}

```

14.

Làm việc với đường dẫn tương đối



Bạn cần thiết lập thư mục làm việc hiện hành để có thể sử dụng đường dẫn tương đối trong mã lệnh của bạn.



Sử dụng phương thức tĩnh `GetCurrentDirectory` và `SetCurrentDirectory` của lớp `System.IO.Directory`.

Đường dẫn tương đối tự động được diễn dịch dựa trên quan hệ với thư mục hiện hành. Bạn có thể lấy thư mục hiện hành bằng phương thức `Directory.GetCurrentDirectory` hoặc thay đổi nó bằng phương thức `Directory.SetCurrentDirectory`. Ngoài ra, bạn có thể sử dụng phương thức tĩnh `GetFullPath` của lớp `System.IO.Path` để chuyển đường dẫn tương đối thành đường dẫn tuyệt đối.

Dưới đây là một ví dụ minh họa:

```

using System;

using System.IO;

public class RelativeDirTest {

    private static void Main() {

        Console.WriteLine("Using: " + Directory.GetCurrentDirectory());
        Console.WriteLine("The relative path 'file.txt' " +
            "will automatically become: " +

```

```

        Path.GetFullPath("file.txt") + "");

    Console.WriteLine();

    Console.WriteLine("Changing current directory to C:\\");
    Directory.SetCurrentDirectory("C:\\");

    Console.WriteLine("Now the relative path 'file.txt' " +
        "will automatically become '" +
        Path.GetFullPath("file.txt") + "'");

    Console.ReadLine();
}
}

```

Kết xuất của ứng dụng này có thể như sau (nếu bạn chạy ứng dụng trong thư mục *C:\Temp*):

```

Using: C:\Temp
The relative path 'file.txt' will automatically become 'C:\Temp\file.txt'
Changing current directory to C:\
The relative path 'file.txt' will automatically become 'C:\file.txt'

```



Nếu sử dụng đường dẫn tương đối, bạn nên thiết lập thư mục làm việc khi bắt đầu tương tác với file. Nếu không, có thể ảnh hưởng đến sự an toàn của hệ thống nếu người dùng cố ý thay đổi thư mục làm việc để ứng dụng của bạn truy xuất hoặc ghi đè file hệ thống.

15.

Tạo file tạm



Bạn cần tạo một file sẽ được đặt trong thư mục tạm của người dùng cụ thể và sẽ có tên duy nhất, để nó không đụng độ với các file tạm được sinh ra bởi các chương trình khác.



Sử dụng phương thức tĩnh `GetTempFileName` của lớp `System.IO.Path`, phương thức này trả về một đường dẫn kết hợp đường dẫn đến thư mục tạm với một tên file được tạo ngẫu nhiên.

Có nhiều cách để tạo file tạm. Trong các trường hợp đơn giản, bạn chỉ cần tạo một file trong thư mục ứng dụng, có thể sử dụng một *GUID* hoặc một tem thời gian kết hợp với một giá trị ngẫu nhiên làm tên file. Tuy nhiên, lớp `Path` hỗ trợ một phương thức giúp bạn đỡ tốn công hơn. Nó sẽ tạo ra một file với tên duy nhất trong thư mục tạm của người dùng hiện hành, chẳng hạn *C:\Documents and Settings\username\Local Settings\Temp\tmpac9.tmp*.


```

using System;
using System.IO;

public class TemporaryFile {

    private static void Main() {

        string tempFile = Path.GetTempFileName();

        Console.WriteLine("Using " + tempFile);
        FileStream fs = new FileStream(tempFile, FileMode.Open);

        // (Ghi dữ liệu.)

        fs.Close();

        // Xóa file.
        File.Delete(tempFile);

        Console.ReadLine();
    }
}

```

16.

Lấy dung lượng đĩa còn trống

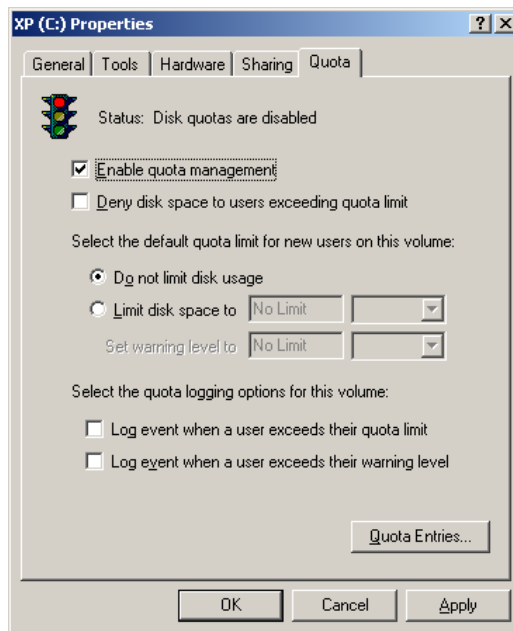
?

Bạn cần xét một ổ đĩa và xác định còn bao nhiêu byte trống.

✕

Sử dụng hàm không-được-quản-lý *Win32 API* `GetDiskFreeSpaceEx`, (hàm này được khai báo trong *kernel32.dll*).

Không lớp nào trong các lớp về hệ thống file của *.NET* cho phép xác định dung lượng đĩa còn trống. Tuy nhiên, bạn có thể dễ dàng lấy được thông tin này bằng hàm *Win32 API* `GetDiskFreeSpaceEx`. Hàm này sẽ trả về dung lượng tổng cộng, dung lượng còn trống, và dung lượng còn trống có thể sử dụng được (nhà quản trị có thể sử dụng *Disk Quota Management* để hạn chế dung lượng mà người dùng có thể sử dụng được).



Hình 9.2 Disk Quota Management

Ứng dụng *Console* dưới đây minh họa kỹ thuật này:

```
using System;
using System.Runtime.InteropServices;

public class GetFreeSpace {

    [DllImport("kernel32.dll", EntryPoint="GetDiskFreeSpaceExA" )]
    private static extern long GetDiskFreeSpaceEx(
        string lpDirectoryName, out long lpFreeBytesAvailableToCaller,
        out long lpTotalNumberOfBytes, out long lpTotalNumberOfFreeBytes);

    private static void Main() {

        long result, total, free, available;
        result = GetDiskFreeSpaceEx("c:", out available, out total,
            out free);
    }
}
```

```

if (result != 0) {

    Console.WriteLine("Total Bytes: {0:N}", total);
    Console.WriteLine("Free Bytes: {0:N}", free);
    Console.WriteLine("Available Bytes: {0:N}", available);

}

Console.ReadLine();

}

}

```

17.

Hiển thị các hộp thoại file

?

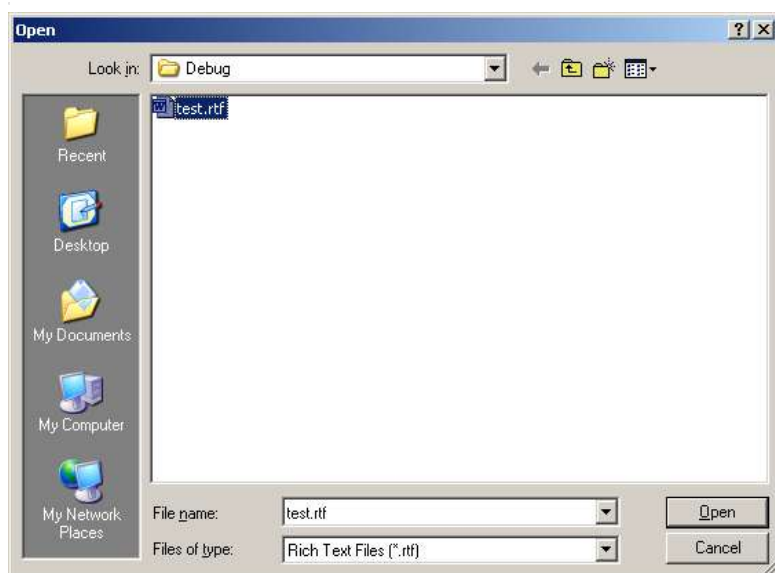
Bạn cần hiện các hộp thoại *Windows* chuẩn để mở, lưu file, và để chọn thư mục.

✕

Sử dụng các lớp `OpenFileDialog`, `SaveFileDialog`, và `FolderBrowserDialog` thuộc không gian tên `System.Windows.Forms`. Gọi phương thức `ShowDialog` để hiển thị hộp thoại, xét giá trị trả về để xác định người dùng đã nhấn *OK* hay *Cancel*, và lấy thông tin từ thuộc tính `FileName` hay `SelectedPath`.

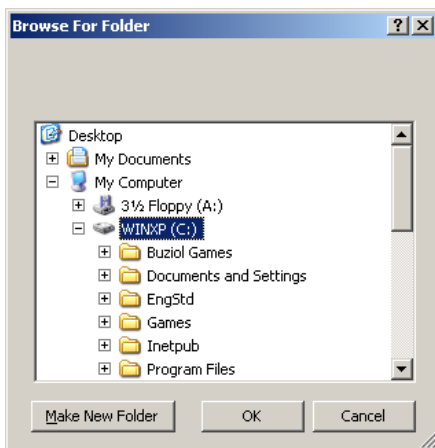
.NET cung cấp các đối tượng bọc lấy nhiều hộp thoại *Windows* chuẩn, bao gồm các hộp thoại dùng để mở và lưu file, và để chọn thư mục. Tất cả các lớp này đều thừa kế từ `System.Windows.Forms.CommonDialog`, bao gồm:

- `OpenFileDialog`—cho phép người dùng chọn một file. Tên file và đường dẫn có thể được lấy từ thuộc tính `FileName` (hay tập hợp `FileNames`, nếu bạn cho phép chọn nhiều file bằng cách thiết lập `Multiselect` là `true`). Ngoài ra, bạn có thể sử dụng thuộc tính `Filter` để chọn định dạng file và thiết lập thuộc tính `CheckFileExists` để kiểm tra tính hợp lệ (xem hình 9.3).



Hình 9.3 OpenFileDialog

- `SaveFileDialog`—cho phép người dùng chỉ định một file mới. Tên file và đường dẫn có thể được lấy từ thuộc tính `FileName`. Bạn có thể sử dụng thuộc tính `Filter` để chọn định dạng file và thiết lập các thuộc tính `CreatePrompt` và `OverwritePrompt` để hiển thị thông báo xác nhận khi người dùng chọn một file mới hay file đã tồn tại.
- `FolderBrowserDialog`—cho phép người dùng chọn (và tạo) một thư mục. Đường dẫn đã chọn có thể được lấy từ thuộc tính `SelectedPath`. Ngoài ra, bạn có thể thiết lập thuộc tính `ShowNewFolderButton` để hiển thị nút *Make New Folder* (xem hình 9.4).

**Hình 9.4** FolderBrowserDialog

Khi sử dụng `OpenFileDialog` hay `SaveFileDialog`, bạn cần thiết lập chuỗi lọc (chỉ định các phần mở rộng được phép). Chuỗi lọc được phân cách bởi ký tự "|" theo định dạng: "[*Nhãn*] | [*Danh sách các phần mở rộng được phân cách bởi dấu chấm phẩy*] | [*Nhãn*] | [*Danh sách các phần mở rộng được phân cách bởi dấu chấm phẩy*] | ...". Bạn cũng có thể thiết lập thuộc tính `Title` (tiêu đề) và `InitialDirectory` (thư mục ban đầu).

Ứng dụng dưới đây cho phép người dùng nạp tài liệu vào một `RichTextBox`, sửa nội dung, và lưu tài liệu đã được sửa (sử dụng lớp `OpenFileDialog` và `SaveFileDialog` để mở và lưu tài liệu).

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class SimpleEditForm : System.Windows.Forms.Form {

    private System.Windows.Forms.MenuItem mnuFile;
    private System.Windows.Forms.MenuItem mnuOpen;
```

```

private System.Windows.Forms.MenuItem mnuSave;
private System.Windows.Forms.MenuItem mnuExit;
private System.Windows.Forms.RichTextBox rtDoc;

// (Bỏ qua phần mã designer.)

private void mnuOpen_Click(object sender, System.EventArgs e) {

    OpenFileDialog dlg = new OpenFileDialog();
    dlg.Filter = "Rich Text Files (*.rtf)|*.RTF|" +
        "All files (*.*)|*.*";
    dlg.CheckFileExists = true;
    dlg.InitialDirectory = Application.StartupPath;

    if (dlg.ShowDialog() == DialogResult.OK) {
        rtDoc.LoadFile(dlg.FileName);
        rtDoc.Enabled = true;
    }
}

private void mnuSave_Click(object sender, System.EventArgs e) {

    SaveFileDialog dlg = new SaveFileDialog();
    dlg.Filter = "RichText Files (*.rtf)|*.RTF|Text Files (*.txt)|
        *.TXT" +
        "|All files (*.*)|*.*";
    dlg.CheckFileExists = true;
    dlg.InitialDirectory = Application.StartupPath;

    if (dlg.ShowDialog() == DialogResult.OK) {
        rtDoc.SaveFile(dlg.FileName);
    }
}

private void mnuExit_Click(object sender, System.EventArgs e) {

    this.Close();
}

```

}

18.**Sử dụng không gian lưu trữ riêng****?**

Bạn cần lưu dữ liệu vào file, nhưng ứng dụng của bạn không được cấp `FileIOPermission` để sử dụng ổ đĩa cứng.

X

Sử dụng các lớp `IsolatedStorageFile` và `IsolatedStorageFileStream` thuộc không gian tên `System.IO.IsolatedStorage`. Các lớp này cho phép ứng dụng ghi dữ liệu vào một file trong thư mục của một người dùng cụ thể mà không cần được cấp phép truy xuất trực tiếp ổ đĩa cứng cục bộ.

.NET Framework hỗ trợ không gian lưu trữ riêng, tức là cho phép bạn đọc và ghi vào hệ thống file ảo của người dùng cụ thể mà *CLR* quản lý. Khi bạn tạo các file lưu trữ riêng, dữ liệu tự động được lưu vào một nơi duy nhất trong đường dẫn profile của người dùng (thông thường đường dẫn này có dạng `C:\Documents and Settings\[username]\Local Settings\Application Data\IsolatedStorage\[guid_identifier]`).

Một lý do để sử dụng không gian lưu trữ riêng là trao cho một ứng dụng có-độ-tin-cây-một-phần có khả năng hạn chế khi lưu trữ dữ liệu (xem mục 13.1 để có thêm thông tin về mã lệnh có-độ-tin-cây-một-phần). Ví dụ, chính sách bảo mật *CLR* mặc định cấp cho mã lệnh cục bộ có `FileIOPermission` không hạn chế, tức là có quyền đọc và ghi bất kỳ file nào. Mã lệnh thực thi từ một máy chủ ở xa trên mạng Intranet cục bộ tự động được cấp ít quyền hơn—thiếu mất `FileIOPermission`, nhưng có `IsolatedStoragePermission`, tức là có khả năng sử dụng không gian lưu trữ riêng (chính sách bảo mật cũng hạn chế dung lượng tối đa có thể được sử dụng trong không gian lưu trữ riêng). Một lý do khác để sử dụng không gian lưu trữ riêng là bảo vệ dữ liệu tốt hơn. Ví dụ, đối với dữ liệu trong không gian lưu trữ riêng của một người dùng, những người dùng khác không phải là nhà quản trị sẽ không được quyền truy xuất.

Đoạn mã dưới đây minh họa cách truy xuất không gian lưu trữ riêng:

```
using System;
using System.IO;
using System.IO.IsolatedStorage;

public class IsolatedStoreTest {

    private static void Main() {

        // Tạo không gian lưu trữ riêng cho người dùng hiện hành.
        IsolatedStorageFile store =
            IsolatedStorageFile.GetUserStoreForAssembly();
```

```

// Tạo một thư mục tại gốc của không gian lưu trữ riêng.
store.CreateDirectory("MyFolder");

// Tạo một file trong không gian lưu trữ riêng.
Stream fs = new IsolatedStorageFileStream(
    "MyFile.txt", FileMode.Create, store);

StreamWriter w = new StreamWriter(fs);

// Ghi file như bình thường.
w.WriteLine("Test");
w.Flush();
fs.Close();

Console.WriteLine("Current size: " + store.CurrentSize.ToString());
Console.WriteLine("Scope: " + store.Scope.ToString());

Console.WriteLine("Contained files include:");
string [] files = store.GetFileNames("*.");
foreach (string file in files) {
    Console.WriteLine(file);
}

Console.ReadLine();
}
}

```

Theo mặc định, mỗi không gian lưu trữ riêng được tách biệt bởi người dùng và assembly. Điều này có nghĩa là khi cùng một người chạy cùng một ứng dụng, ứng dụng này sẽ truy xuất dữ liệu trong cùng không gian lưu trữ riêng. Tuy nhiên, bạn có thể tách biệt thêm bởi miền ứng dụng để nhiều thể hiện của cùng một ứng dụng nhận các không gian lưu trữ riêng khác nhau, như ví dụ sau:

```

// Truy xuất không gian lưu trữ riêng của người dùng
// và assembly hiện tại (tương tự ví dụ trên).
store = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
    IsolatedStorageScope.Assembly, null, null);

// Truy xuất không gian lưu trữ riêng của người dùng, assembly,
// và miền ứng dụng hiện hành. Nói cách khác, dữ liệu này chỉ được

```

```
// truy xuất bởi thể hiện hiện tại của ứng dụng.
store = IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
    IsolatedStorageScope.Assembly | IsolatedStorageScope.Domain,
    null, null);
```

File được lưu trữ như một phần profile của người dùng, vì vậy người dùng có thể truy xuất các file này từ bất kỳ máy nào trong mạng LAN nếu roaming profile³ đã được cấu hình (trong trường hợp này, phải thiết lập cờ `IsolatedStorageFile.Roaming` khi tạo không gian lưu trữ). Bằng cách để .NET Framework và CLR cung cấp các mức cách ly, bạn không phải duy trì sự tách biệt giữa các file, và không phải lo việc không hiểu rõ cơ chế làm việc sẽ gây mất dữ liệu quan trọng.

19. Theo dõi hệ thống file để phát hiện thay đổi

- ?** Bạn cần phản ứng khi hệ thống file thay đổi tại một đường dẫn cụ thể (chẳng hạn sửa hay tạo file).
- ✕** Sử dụng thành phần `System.IO.FileSystemWatcher`, chỉ định file hoặc đường dẫn cần theo dõi, và thụ lý các sự kiện `Created`, `Deleted`, `Renamed`, và `Changed`.

Khi liên kết nhiều ứng dụng và các quá trình nghiệp vụ, thường cần tạo một chương trình thụ động và chỉ trở nên tích cực khi một file được tạo ra hoặc bị thay đổi. Bạn có thể tạo kiểu chương trình thế này bằng cách quét định kỳ qua một thư mục, nhưng lại gặp phải vấn đề cân bằng. Quét càng thường xuyên, càng tốn tài nguyên hệ thống. Quét càng ít, càng lâu phát hiện được thay đổi. Cách tiện nhất là sử dụng lớp `FileSystemWatcher` để phản ứng trực tiếp các sự kiện file của Windows.

Để sử dụng `FileSystemWatcher`, bạn phải tạo một thể hiện và thiết lập các thuộc tính sau:

- `Path`—chỉ định đường dẫn cần theo dõi.
- `Filter`—chỉ định kiểu file cần theo dõi.
- `NotifyFilter`—chỉ định kiểu thay đổi cần theo dõi.

`FileSystemWatcher` sinh ra bốn sự kiện chính: `Created`, `Deleted`, `Renamed`, và `Changed`. Các sự kiện này cung cấp thông tin qua đối số `FileSystemEventArgs`, bao gồm tên file (`Name`), đường dẫn đầy đủ (`FullPath`), kiểu thay đổi (`ChangeType`). Sự kiện `Renamed` cung cấp thể hiện `RenamedEventArgs` dẫn xuất từ `FileSystemEventArgs`, và thêm thông tin về tên file ban đầu (`OldName` và `OldFullPath`). Nếu cần, bạn có thể vô hiệu các sự kiện này bằng cách thiết lập thuộc tính `FileSystemWatcher.EnableRaisingEvents` là `false`. Dễ dàng thụ lý các sự kiện `Created`, `Deleted`, và `Renamed`. Nhưng nếu muốn sử dụng sự kiện `Changed`, bạn cần sử dụng thuộc tính `NotifyFilter` để chỉ định kiểu thay đổi cần theo dõi. Bằng không, chương trình của bạn có thể bị sa lầy bởi một loạt các sự kiện khi file bị thay đổi.

³ Roaming profile được tạo bởi người quản trị hệ thống và được lưu trữ trên một server. Profile này có sẵn mỗi khi bạn đăng nhập vào bất kỳ máy tính nào trên mạng. Bất cứ thay đổi nào cũng khiến roaming profile được cập nhật lại trên server.

Thuộc tính `NotifyFilter` có thể được thiết lập bằng cách kết hợp các giá trị thuộc kiểu liệt kê `System.IO.NotifyFilters: Attributes, CreationTime, DirectoryName, FileName, LastWrite, LastAccess, Security, và Size`.

Ví dụ dưới đây thụ lý sự kiện `Created` và `Deleted`. Và thử nghiệm các sự kiện này bằng cách tạo ra một file thử nghiệm.

```
using System;
using System.IO;
using System.Windows.Forms;

public class FileWatcherTest {

    private static void Main() {

        // Cấu hình FileSystemWatcher.
        FileSystemWatcher watch = new FileSystemWatcher();
        watch.Path = Application.StartupPath;
        watch.Filter = "*.txt";
        watch.IncludeSubdirectories = true;

        // Đăng ký các phương thức thụ lý sự kiện.
        watch.Created += new FileSystemEventHandler(OnCreatedOrDeleted);
        watch.Deleted += new FileSystemEventHandler(OnCreatedOrDeleted);
        watch.EnableRaisingEvents = true;

        Console.WriteLine("Press Enter to create a file.");
        Console.ReadLine();

        if (File.Exists("test.bin")) {
            File.Delete("test.bin");
        }

        FileStream fs = new FileStream("test.bin", FileMode.Create);
        fs.Close();

        Console.WriteLine("Press Enter to terminate the application.");
        Console.WriteLine();
        Console.ReadLine();
    }
}
```

```
// Phát sinh khi một file mới được tạo ra
// trong thư mục cần theo dõi.
private static void OnCreatedOrDeleted(object sender,
    FileSystemEventArgs e) {

    // Hiển thị thông báo.
    Console.WriteLine("\tNOTIFICATION: " + e.FullPath +
        "' was " + e.ChangeType.ToString());
    Console.WriteLine();
}
}
```

20.

Truy xuất cổng COM

?

Bạn cần gửi dữ liệu trực tiếp đến một cổng tuần tự (*serial port*).

✕

Win32 API cung cấp các hàm không-được-quản-lý trong thư viện *kernel32.dll* để trực tiếp đọc và ghi các byte đến cổng tuần tự. Bạn có thể nhập các hàm này vào ứng dụng hoặc sử dụng *Microsoft Communications ActiveX control (MSComm.ocx—có trong Microsoft Visual Studio 6)*.

.NET không cung cấp bất kỳ giao diện được-quản-lý nào để thao tác với các cổng tuần tự. Do đó, những ai cần chức năng này sẽ phải làm việc với các cơ chế tương đối phức tạp.

Một hướng giải quyết là tạo một vỏ bọc .NET cho *Microsoft Communications Control (MSComm.ocx)*. Điều kiểm này cung cấp một mô hình đối tượng mức-cao để làm việc với cổng tuần tự. Tuy nhiên, bạn phải thu lấy điều kiểm này thông qua *Visual Studio 6* (bạn có thể chỉ chọn cài đặt các thành phần *ActiveX* khi cài đặt *Visual Studio 6*, tốn khoảng 5MB). Để có thêm thông tin, bạn hãy tham khảo tài liệu *Visual Studio 6*.

Một hướng giải quyết khác là nhập các hàm API từ thư viện *kernel32.dll*. Cần cẩn thận khi sử dụng phương pháp này vì bạn phải sử dụng đúng kiểu dữ liệu C# và duy trì layout của các cấu trúc bộ nhớ. May mắn là, vấn đề này đã được *Justin Harrell (jharrell@aciss.com)* giải quyết với một lớp C# tùy biến có tên là *ComPort*. Mã lệnh của lớp này khá dài, bạn hãy xem trong đĩa CD đính kèm.

Bạn có thể thêm lớp *ComPort* vào ứng dụng của bạn và sử dụng đoạn mã sau để tương tác với một cổng COM.

```
ComPort port = new ComPort();
```

```
try {
```

```
    // Cấu hình cho cổng.
```

```
port.BaudRate = 9600;
port.Parity = 0;
port.PortNum = 1;
port.ReadTimeout = 10;
port.StopBits = 1;
port.ByteSize = 1;

// Mở cổng.
port.Open();

// Ghi dữ liệu.
port.Write(new byte[1]);

// Đóng cổng.
port.Close();

}catch (ApplicationException err) {

    Console.WriteLine(err.Message);
}
```


10

CƠ SỞ DỮ LIỆU

Trong *Microsoft .NET Framework*, việc truy xuất đến các loại data-source (nguồn dữ liệu) được cho phép thông qua một nhóm các lớp có tên là *Microsoft ADO.NET*. Mỗi loại data-source được hỗ trợ thông qua một data-provider (bộ cung cấp dữ liệu). Mỗi data-provider gồm tập các lớp không chỉ hiện thực tập giao diện chuẩn, mà còn cung cấp chức năng duy nhất của data-source mà nó hỗ trợ. Các lớp này mô tả về kết nối, câu lệnh, thông số, data-adapter (bộ điều hợp dữ liệu), và data-reader (bộ đọc dữ liệu) mà qua đó, bạn có thể tương tác với một loại data-source. Bảng 10.1 liệt kê các hiện thực data-provider trong *.NET*.

Bảng 10.1 Các hiện thực data-provider trong *.NET Framework*

Data-provider	Mô tả
<i>.NET Framework Data Provider for ODBC</i>	Cung cấp kết nối đến mọi data-source có hiện thực giao diện <i>ODBC</i> ; bao gồm <i>Microsoft SQL Server</i> , <i>Oracle</i> , và <i>Microsoft Access</i> . Các lớp data-provider nằm trong không gian tên <code>System.Data.Odbc</code> và có tiền tố <code>Odbc</code> .
<i>.NET Framework Data Provider for OLE DB</i>	Cung cấp kết nối đến mọi data-source có hiện thực giao diện <i>OLE DB</i> ; bao gồm <i>Microsoft SQL Server</i> , <i>MSDE</i> , <i>Oracle</i> , và <i>Jet</i> . Các lớp data-provider nằm trong không gian tên <code>System.Data.OleDb</code> và có tiền tố <code>OleDb</code> .
<i>.NET Framework Data Provider for Oracle</i>	Cung cấp kết nối đến <i>Oracle</i> . Các lớp data-provider nằm trong không gian tên <code>System.Data.OracleClient</code> và có tiền tố <code>Oracle</code> .
<i>.NET Framework Data Provider for SQL Server</i>	Cung cấp kết nối đến <i>Microsoft SQL Server</i> phiên bản 7 và mới hơn (gồm cả <i>MSDE</i>) bằng cách liên lạc trực tiếp với <i>SQL Server</i> mà không cần sử dụng <i>ODBC</i> hay <i>OLE DB</i> . Các lớp data-provider nằm trong không gian tên <code>System.Data.SqlClient</code> và có tiền tố <code>Sql</code> .
<i>.NET Compact Framework Data Provider for SQL Server CE</i>	Cung cấp kết nối đến <i>Microsoft SQL Server CE</i> . Các lớp data-provider nằm trong không gian tên <code>System.Data.SqlServerCe</code> và có tiền tố <code>SqlCe</code> .

Chương này mô tả một vài khía cạnh thường được sử dụng nhất của *ADO.NET*. Tuy nhiên, *ADO.NET* là một phần con mở rộng của thư viện lớp *.NET Framework* và chứa một lượng lớn các chức năng cao cấp. Do đó, để có thể hiểu rõ hơn về *ADO.NET*, bạn nên tìm đọc một quyển sách khác chuyên về *ADO.NET*. Những đề mục trong chương này trình bày các vấn đề sau:

- Cách tạo, cấu hình, mở, và đóng kết nối cơ sở dữ liệu (mục 10.1).
- Cách sử dụng connection-pooling để cải thiện hiệu năng và tính quy mô của các ứng dụng có sử dụng kết nối cơ sở dữ liệu (mục 10.2).
- Cách thực thi các câu lệnh *SQL* và các thủ tục tồn trữ (*Stored Procedure*), và cách sử dụng các thông số để cải thiện tính linh hoạt của chúng (mục 10.3 và 10.4).
- Cách xử lý kết quả được trả về từ truy vấn cơ sở dữ liệu (mục 10.5 và 10.6).
- Cách nhận biết tất cả các đối tượng *SQL Server* đang có hiệu lực trên mạng (mục 10.7).

- Đọc file *Excel* với *ADO.NET* (mục 10.8).
- Cách sử dụng *Data Form Wizard* (mục 10.9) và *Crystal Report Wizard* (mục 10.10).

 Những đề mục trong chương này sử dụng cơ sở dữ liệu mẫu *Northwind* (do *Microsoft* cấp) để làm rõ những kỹ thuật được thảo luận.

1.

Kết nối cơ sở dữ liệu



Bạn cần mở một kết nối đến một cơ sở dữ liệu.



Tạo một đối tượng kết nối phù hợp với kiểu cơ sở dữ liệu mà bạn cần kết nối; tất cả các đối tượng kết nối đều hiện thực giao diện `System.Data.IDbConnection`. Cấu hình đối tượng kết nối bằng cách thiết lập thuộc tính `ConnectionString` của nó. Mở kết nối bằng cách gọi phương thức `Open` của đối tượng kết nối.

Bước đầu tiên trong việc truy xuất cơ sở dữ liệu là mở một kết nối đến cơ sở dữ liệu. Giao diện `IDbConnection` mô tả một kết nối cơ sở dữ liệu, và mỗi data-provider chứa một hiện thực duy nhất. Dưới đây là danh sách các hiện thực `IDbConnection` cho năm data-provider chuẩn:

- `System.Data.Odbc.OdbcConnection`
- `System.Data.OleDb.OleDbConnection`
- `System.Data.OracleClient.OracleConnection`
- `System.Data.SqlServerCe.SqlCeConnection`
- `System.Data.SqlClient.SqlConnection`

Bạn cấu hình một đối tượng kết nối bằng một chuỗi kết nối. Chuỗi kết nối là một tập các cặp giá trị tên được phân cách bằng dấu chấm phẩy. Bạn có thể cung cấp một chuỗi kết nối làm đối số trong phương thức khởi dựng hoặc bằng cách thiết lập thuộc tính `ConnectionString` của đối tượng kết nối trước khi mở kết nối. Mỗi hiện thực lớp kết nối yêu cầu bạn cung cấp những thông tin khác nhau trong chuỗi kết nối. Bạn hãy tham khảo tài liệu về thuộc tính `ConnectionString` đối với mỗi hiện thực để biết được những giá trị mà bạn có thể chỉ định. Dưới đây là một số thiết lập:

- Tên server cơ sở dữ liệu đích
- Tên cơ sở dữ liệu cần mở vào lúc đầu
- Giá trị timeout của kết nối
- Cơ chế connection-pooling (xem mục 10.2)
- Cơ chế xác thực dùng khi kết nối đến các cơ sở dữ liệu được bảo mật, bao gồm việc cung cấp username và password

Một khi đã được cấu hình, gọi phương thức `Open` của đối tượng kết nối để mở kết nối đến cơ sở dữ liệu. Kế đó, bạn có thể sử dụng đối tượng kết nối để thực thi những câu lệnh dựa vào data-source (sẽ được thảo luận trong mục 10.3). Các thuộc tính của đối tượng kết nối cũng

cho phép bạn lấy thông tin về trạng thái của một kết nối và những thiết lập được sử dụng để mở kết nối. Khi đã hoàn tất một kết nối, bạn nên gọi phương thức `Close` để giải phóng các tài nguyên hệ thống và kết nối cơ sở dữ liệu nằm dưới. `IDbConnection` được thừa kế từ `System.IDisposable`, nghĩa là mỗi lớp kết nối sẽ hiện thực phương thức `Dispose`. Phương thức này sẽ tự động gọi `Close`, cho nên lệnh `using` là một cách rất rõ ràng và hiệu quả khi sử dụng đối tượng kết nối trong mã lệnh.

Để đạt được hiệu năng tối ưu trong việc truy cập dữ liệu thì phải mở kết nối cơ sở dữ liệu càng chậm càng tốt, và khi đã hoàn tất thì ngắt kết nối càng sớm càng tốt. Việc này bảo đảm rằng, bạn không truy xuất tới kết nối cơ sở dữ liệu trong một thời gian dài và mã lệnh có cơ hội cao nhất để giữ lấy kết nối. Điều này đặc biệt quan trọng nếu bạn đang sử dụng `connection-pooling`.

Đoạn mã dưới đây trình bày cách sử dụng lớp `SqlConnection` để mở một kết nối đến *SQL Server* đang chạy trên máy cục bộ có sử dụng *Integrated Windows Security* (bảo mật tích hợp với *Windows*). Để truy xuất đến một máy từ xa, chỉ cần thay đổi `data-source` từ `localhost` thành tên của đối tượng cơ sở dữ liệu.

```
// Tạo đối tượng SqlConnection rỗng.
using (SqlConnection con = new SqlConnection()) {

    // Cấu hình chuỗi kết nối của đối tượng SqlConnection.
    con.ConnectionString =
        "Data Source = localhost;" + // Đối tượng SQL Server cục bộ
        "Database = Northwind;" +   // Cơ sở dữ liệu mẫu Northwind
        "Integrated Security=SSPI"; // Integrated Windows Security

    // Mở kết nối cơ sở dữ liệu.
    con.Open();

    // Hiển thị thông tin về kết nối.
    if (con.State == ConnectionState.Open) {
        Console.WriteLine("SqlConnection Information:");
        Console.WriteLine("  Connection State = " + con.State);
        Console.WriteLine("  Connection String = " + con.ConnectionString);
        Console.WriteLine("  Database Source = " + con.DataSource);
        Console.WriteLine("  Database = " + con.Database);
        Console.WriteLine("  Server Version = " + con.ServerVersion);
        Console.WriteLine("  Workstation Id = " + con.WorkstationId);
        Console.WriteLine("  Timeout = " + con.ConnectionTimeout);
        Console.WriteLine("  Packet Size = " + con.PacketSize);
    } else {
        Console.WriteLine("SqlConnection failed to open.");
    }
}
```

```

        Console.WriteLine(" Connection State = " + con.State);
    }
    // Cuối khối using, Dispose sẽ gọi Close.
}

```

Đoạn mã dưới đây trình bày một chuỗi kết nối dùng để mở một kết nối đến cơ sở dữ liệu ở trên (nếu bạn đang sử dụng *OLE DB Data Provider* để thực hiện kết nối):

```

// Tạo một đối tượng OleDbConnection rỗng.
using (OleDbConnection con = new OleDbConnection()) {

    // Cấu hình chuỗi kết nối của đối tượng OleDbConnection.
    con.ConnectionString =
        "Provider = SQLOLEDB;" +           // OLE DB Provider for SQL Server
        "Data Source = localhost;" +       // Đối tượng SQL Server cục bộ
        "Initial Catalog = Northwind;" +   // Cơ sở dữ liệu mẫu Northwind
        "Integrated Security=SSPI";        // Integrated Windows Security

    // Mở kết nối cơ sở dữ liệu.
    con.Open();

    S
}

```

2.

Sử dụng connection-pooling



Bạn muốn duy trì một pool chứa các kết nối đang mở để cải thiện hiệu năng và tính quy mô cho một hệ thống lớn.



Cấu hình pool bằng cách sử dụng các thiết lập trong chuỗi kết nối của đối tượng kết nối.

Connection-pooling làm giảm đáng kể tổng phí liên hợp với việc tạo và hủy kết nối cơ sở dữ liệu. Connection-pooling cũng cải thiện tính quy mô của các giải pháp bằng cách giảm số lượng kết nối đồng thời mà một cơ sở dữ liệu phải duy trì—đa số thường “ngồi không” suốt một phần đáng kể thuộc thời gian sống của chúng. Với connection-pooling, thay vì tạo và mở một đối tượng kết nối mới mỗi khi cần, bạn có thể lấy kết nối đã mở từ pool. Khi bạn đã hoàn tất việc sử dụng kết nối, thay vì đóng nó, bạn trả nó về cho pool và cho phép đoạn mã khác sử dụng nó.

Theo mặc định, *SQL Server* và *Oracle Data Provider* cung cấp chức năng connection-pooling. Một pool sẽ hiện diện đối với mỗi chuỗi kết nối do bạn chỉ định khi mở một kết nối mới. Mỗi khi bạn mở một kết nối mới với chuỗi kết nối đã được sử dụng qua, nó sẽ được lấy từ pool

hiện có. Chỉ khi bạn chỉ định một chuỗi kết nối khác thì data-provider mới tạo một pool mới. Bạn có thể điều khiển các đặc tính của pool bằng cách sử dụng các thiết lập trong chuỗi kết nối được mô tả trong bảng 10.2.

 **Một khi đã được tạo, pool sẽ tồn tại cho đến khi tiến trình kết thúc.**

Bảng 10.2 Các thiết lập trong chuỗi kết nối dùng để điều khiển Connection Pooling

Thiết lập	Mô tả
Connection Lifetime	Chỉ định thời gian tối đa (tính bằng giây) mà một kết nối được phép sống trong pool trước khi nó bị đóng. “Tuổi” của một kết nối được kiểm tra chỉ khi kết nối được trả về cho pool. Thiết lập này cần thiết trong việc thu nhỏ kích thước pool nếu pool không được sử dụng nhiều và cũng bảo đảm tính cân bằng tải được thực hiện tối ưu trong môi trường cơ sở dữ liệu gom tụ. Giá trị mặc định là 0, có nghĩa là kết nối tồn tại trong thời gian sống của tiến trình hiện thời.
Connection Reset	Chỉ được <i>SQL Server Data Provider</i> hỗ trợ. Chỉ định kết nối có được reset hay không khi chúng được lấy từ pool. Giá trị <code>True</code> bảo đảm trạng thái của kết nối được reset nhưng cần phải thông báo cho cơ sở dữ liệu. Giá trị mặc định là <code>True</code> .
Pooling	Thiết lập là <code>False</code> để có được kết nối không lấy từ pool. Giá trị mặc định là <code>True</code> .
Max Pool Size	Chỉ định số lượng kết nối tối đa cần có trong pool. Các kết nối được tạo và thêm vào pool khi được yêu cầu cho đến khi đạt đến con số này. Nếu một yêu cầu kết nối được thực hiện nhưng không còn kết nối trống thì lời gọi sẽ block cho đến khi có một kết nối có hiệu lực. Giá trị mặc định là 100.
Min Pool Size	Chỉ định số lượng kết nối tối thiểu cần có trong pool. Lúc tạo pool thì số kết nối này được tạo và thêm vào pool. Trong quá trình duy trì định kỳ hoặc khi một kết nối được yêu cầu, các kết nối sẽ được thêm vào pool để bảo đảm số lượng kết nối tối thiểu có hiệu lực. Giá trị mặc định là 0.

Đoạn mã dưới đây mô tả cấu hình của một pool: chứa tối thiểu 5 kết nối và tối đa 15 kết nối, kết nối sẽ hết hiệu lực sau 10 phút (600 giây) và được reset mỗi khi một kết nối được lấy từ pool.

```
// Thu lấy pooled connection.
using (SqlConnection con = new SqlConnection()) {

    // Cấu hình chuỗi kết nối của đối tượng SqlConnection.
    con.ConnectionString =
        "Data Source = localhost;" +      // Đối tượng SQL Server cục bộ
        "Database = Northwind;" +        // Cơ sở dữ liệu mẫu Northwind
        "Integrated Security = SSPI;" +   // Integrated Windows Security
```

```

        "Min Pool Size = 5;" +           // Kích thước tối thiểu của pool
        "Max Pool Size = 15;" +         // Kích thước tối đa của pool
        "Connection Reset = True;" +    // Reset kết nối mỗi khi sử dụng
        "Connection Lifetime = 600";    // Thời gian sống tối đa

// Mở kết nối cơ sở dữ liệu.
con.Open();

// Truy xuất cơ sở dữ liệu...
$

// Cuối khối using, Dispose sẽ gọi Close,
// trả kết nối về cho pool để tái sử dụng.
}

```

Đoạn mã dưới đây mô tả cách sử dụng thiết lập `Pooling` để có được một đối tượng kết nối không phải lấy từ pool. Điều này cần thiết khi ứng dụng của bạn sử dụng một kết nối đơn "sống lâu".

```

// Thu lấy non-pooled connection.
using (SqlConnection con = new SqlConnection()) {

    // Cấu hình chuỗi kết nối của đối tượng SqlConnection.
    con.ConnectionString =
        "Data Source = localhost;" +    // Đối tượng SQL Server cục bộ
        "Database = Northwind;" +      // Cơ sở dữ liệu mẫu Northwind
        "Integrated Security = SSPI;" + // Integrated Windows Security
        "Pooling = False";              // Chỉ định non-pooled connection

// Mở kết nối cơ sở dữ liệu.
con.Open();

// Truy xuất cơ sở dữ liệu...
$

// Cuối khối using, Dispose sẽ gọi Close,
// đóng non-pooled connection.
}

```

ODBC và *OLE DB Data Provider* cũng hỗ trợ connection-pooling, nhưng chúng không hiện thực connection-pooling bên trong các lớp *.NET*, nên bạn không thể cấu hình pool theo cách như *SQL Server* hay *Oracle Data Provider*. Connection-pooling trong *ODBC* được quản lý bởi *ODBC Driver Manager* và được cấu hình bằng công cụ *ODBC Data Source Administrator* trong *Control Panel*. Connection-pooling trong *OLE DB* được quản lý bởi hiện thực *OLE DB* nguyên sinh; bạn có thể làm mất hiệu lực pooling bằng cách thêm thiết lập “*OLE DB Services=-4;*” vào chuỗi kết nối. *SQL Server CE Data Provider* không hỗ trợ connection-pooling, vì tại một thời điểm *SQL Server CE* chỉ hỗ trợ một kết nối.

3. Thực thi câu lệnh SQL hoặc thủ tục tồn trữ

? Bạn cần thực thi một câu lệnh *SQL* hoặc một thủ tục tồn trữ trên một cơ sở dữ liệu.

X Tạo một đối tượng câu lệnh phù hợp với kiểu cơ sở dữ liệu mà bạn định sử dụng; tất cả các đối tượng câu lệnh đều hiện thực giao diện `System.Data.IDbCommand`. Cấu hình đối tượng câu lệnh bằng cách thiết lập các thuộc tính `CommandType` và `CommandText` của nó. Thực thi câu lệnh bằng một trong các phương thức `ExecuteNonQuery`, `ExecuteReader`, hay `ExecuteScalar` tùy thuộc vào kiểu câu lệnh và kết quả của nó.

Giao diện `IDbCommand` mô tả một câu lệnh cơ sở dữ liệu, và mỗi data-provider chứa một hiện thực duy nhất. Dưới đây là danh sách các hiện thực `IDbCommand` cho năm data-provider chuẩn:

- `System.Data.Odbc.OdbcCommand`
- `System.Data.OleDb.OleDbCommand`
- `System.Data.OracleClient.OracleCommand`
- `System.Data.SqlClient.SqlCeCommand`
- `System.Data.SqlClient.SqlCommand`

Để thực thi một câu lệnh dựa trên một cơ sở dữ liệu, bạn phải có một kết nối đang mở (đã được thảo luận trong mục 10.1) và một đối tượng câu lệnh đã được cấu hình phù hợp với kiểu cơ sở dữ liệu đang truy xuất. Bạn có thể tạo đối tượng câu lệnh một cách trực tiếp bằng phương thức khởi dựng, nhưng cách đơn giản hơn là sử dụng phương thức `CreateCommand` của đối tượng kết nối. Phương thức `CreateCommand` trả về một đối tượng câu lệnh (đúng kiểu data-provider) và cấu hình nó với các thông tin cơ sở được lấy từ kết nối mà bạn đã sử dụng để tạo câu lệnh. Trước khi thực thi câu lệnh, bạn phải cấu hình các thuộc tính được mô tả trong bảng 10.3

Bảng 10.3 Các thuộc tính thông dụng của đối tượng câu lệnh

Thuộc tính	Mô tả
<code>CommandText</code>	Chuỗi chứa câu lệnh <i>SQL</i> hoặc tên của thủ tục tồn trữ. Nội dung của thuộc tính <code>CommandText</code> phải tương thích với giá trị bạn chỉ định trong thuộc tính <code>CommandType</code> .

CommandTimeout	Số nguyên (int) chỉ định số giây đợi câu lệnh trả về trước khi hết thời gian và ngoại lệ xảy ra. Mặc định là 30 giây.
CommandType	Một giá trị thuộc kiểu liệt kê <code>System.Data.CommandType</code> , chỉ định kiểu câu lệnh được mô tả bởi đối tượng câu lệnh. Đối với hầu hết các data-provider, giá trị hợp lệ là <code>StoredProcedure</code> (khi bạn muốn thực thi một thủ tục tồn trữ), và <code>Text</code> (khi bạn muốn thực thi một câu lệnh <i>SQL</i> dạng text). Nếu đang sử dụng <i>OLE DB Data Provider</i> , bạn có thể chỉ định <code>TableDirect</code> khi muốn trả về toàn bộ nội dung của một hoặc nhiều bảng; hãy tham khảo tài liệu <i>.NET Framework SDK</i> để biết thêm chi tiết. Mặc định là <code>Text</code> .
Connection	Đối tượng <code>IDbConnection</code> , cung cấp kết nối đến cơ sở dữ liệu mà bạn sẽ thực thi câu lệnh trên đó. Nếu bạn tạo câu lệnh bằng phương thức <code>IDbConnection.CreateCommand</code> , thuộc tính này sẽ tự động được thiết lập thành đối tượng <code>IDbConnection</code> mà bạn đã tạo câu lệnh từ nó.
Parameters	Đối tượng <code>System.Data.IDataParameterCollection</code> , chứa tập các thông số để thay thế vào câu lệnh (xem mục 10.4 để biết cách sử dụng thông số).
Transaction	Đối tượng <code>System.Data.IDbTransaction</code> , mô tả phiên giao dịch mà câu lệnh được đưa vào đó (xem tài liệu <i>.NET Framework SDK</i> để biết thêm chi tiết về phiên giao dịch).

Một khi bạn đã cấu hình đối tượng câu lệnh thì có nhiều cách để thực thi nó, tùy thuộc vào bản chất của câu lệnh, kiểu dữ liệu do câu lệnh trả về, và bạn muốn xử lý dữ liệu theo định dạng nào.

Để thực thi một câu lệnh như **INSERT**, **DELETE**, hoặc **CREATE TABLE** (không trả về dữ liệu trong cơ sở dữ liệu), bạn hãy gọi `ExecuteNonQuery`. Đối với các câu lệnh **UPDATE**, **INSERT**, và **DELETE**, phương thức `ExecuteNonQuery` trả về một số nguyên cho biết số hàng bị tác động bởi câu lệnh. Đối với các câu lệnh khác như **CREATE TABLE**, `ExecuteNonQuery` trả về -1. Ví dụ dưới đây sử dụng **UPDATE** để chỉnh sửa một bản ghi.

```
public static void ExecuteNonQueryExample(IDbConnection con) {

    // Tạo và cấu hình câu lệnh mới.
    IDbCommand com = con.CreateCommand();
    com.CommandType = CommandType.Text;
    com.CommandText = "UPDATE Employees SET Title = 'Sales Director' +
        " WHERE EmployeeId = '5'";

    // Thực thi câu lệnh và xử lý kết quả.
    int result = com.ExecuteNonQuery();
}
```

```

if (result == 1) {
    Console.WriteLine("Employee title updated.");
} else {
    Console.WriteLine("Employee title not updated.");
}
}

```

Để thực thi một câu lệnh trả về một tập kết quả như lệnh **SELECT** hoặc thủ tục tồn trữ, bạn hãy sử dụng phương thức `ExecuteReader`. Phương thức này trả về một đối tượng `IDataReader` (sẽ được thảo luận trong mục 10.5) mà qua nó bạn có thể truy xuất đến dữ liệu kết quả. Hầu hết các data-provider cũng cho phép bạn thực thi nhiều câu lệnh *SQL* trong một lời gọi phương thức `ExecuteReader`; ví dụ trong mục 10.5 sẽ giải thích điều này và trình bày cách truy xuất mỗi tập kết quả.

Đoạn mã dưới đây sử dụng phương thức `ExecuteReader` để thực thi thủ tục tồn trữ “*Ten Most Expensive Products*” (mười sản phẩm đắt nhất) từ cơ sở dữ liệu *Northwind* và hiển thị kết quả trong cửa sổ *Console*:

```

public static void ExecuteReaderExample(IDbConnection con) {

    // Tạo và cấu hình câu lệnh mới.
    IDbCommand com = con.CreateCommand();
    com.CommandType = CommandType.StoredProcedure;
    com.CommandText = "Ten Most Expensive Products";

    // Thực thi câu lệnh và xử lý kết quả.
    using (IDataReader reader = com.ExecuteReader()) {

        Console.WriteLine("Price of the Ten Most Expensive Products.");

        while (reader.Read()) {

            // Hiển thị chi tiết về sản phẩm.
            Console.WriteLine("  {0} = {1}",
                reader["TenMostExpensiveProducts"],
                reader["UnitPrice"]);
        }
    }
}

```

Nếu muốn thực thi một truy vấn, nhưng chỉ cần giá trị thuộc cột đầu tiên của hàng đầu tiên trong dữ liệu kết quả, bạn hãy sử dụng phương thức `ExecuteScalar`. Giá trị trả về là một tham chiếu đối tượng và bạn cần ép nó về đúng kiểu. Dưới đây là ví dụ:

```
public static void ExecuteScalarExample(IDbConnection con) {

    // Tạo và cấu hình câu lệnh mới.
    IDbCommand com = con.CreateCommand();
    com.CommandType = CommandType.Text;
    com.CommandText = "SELECT COUNT(*) FROM Employees";

    // Thực thi câu lệnh và ép kiểu kết quả.
    int result = (int)com.ExecuteScalar();

    Console.WriteLine("Employee count = " + result);
}
```



Các hiện thực `IDbCommand` trong *Oracle* và *SQL Data Provider* có hiện thực các phương thức thực thi câu lệnh bổ sung. Mục 10.6 sẽ mô tả cách sử dụng phương thức `ExecuteXmlReader` do lớp `SqlCommand` cung cấp. Bạn hãy tham khảo tài liệu *.NET Frameworks SDK* để biết thêm chi tiết về các phương thức bổ sung `ExecuteOracleNonQuery` và `ExecuteOracleScalar` do lớp `OracleCommand` cung cấp.

4. Sử dụng thông số trong câu lệnh SQL hoặc thủ tục tồn trữ



Bạn cần thiết lập các đối số của một thủ tục tồn trữ hoặc sử dụng các thông số trong một câu lệnh *SQL* để cải thiện tính linh hoạt.



Tạo đối tượng thông số phù hợp với kiểu đối tượng câu lệnh mà bạn dự định thực thi; tất cả các đối tượng thông số đều hiện thực giao diện `System.Data.IDataParameter`. Cấu hình kiểu dữ liệu, giá trị, và hướng của đối tượng thông số và thêm chúng vào tập hợp thông số của đối tượng câu lệnh bằng phương thức `IDbCommand.Parameters.Add`.

Tất cả các đối tượng câu lệnh đều hỗ trợ việc sử dụng thông số, do đó bạn có thể thực hiện các công việc sau:

- Thiết lập các đối số của thủ tục tồn trữ
- Lấy các giá trị trả về từ thủ tục tồn trữ
- Đổi các giá trị thành các câu lệnh text lúc thực thi

Giao diện `IDataParameter` mô tả một thông số và mỗi data-provider chứa một hiện thực duy nhất. Dưới đây là danh sách các hiện thực `IDataParameter` cho năm data-provider chuẩn:

- `System.Data.Odbc.OdbcParameter`
- `System.Data.OleDb.OleDbParameter`
- `System.Data.OracleClient.OracleParameter`
- `System.Data.SqlServerCe.SqlCeParameter`
- `System.Data.SqlClient.SqlParameter`

Các thuộc tính của đối tượng thông số mô tả những thứ mà một đối tượng câu lệnh cần khi thực thi một câu lệnh dựa trên data-source. Bảng 10.4 mô tả các thuộc tính thường được sử dụng khi cấu hình cho thông số.

Bảng 10.4 Các thuộc tính của đối tượng thông số

Thuộc tính	Mô tả
<code>DbType</code>	Giá trị thuộc kiểu liệt kê <code>System.Data.DbType</code> , chỉ định kiểu dữ liệu chứa trong thông số. Các giá trị thường được sử dụng là <code>String</code> , <code>Int32</code> , <code>DateTime</code> , và <code>Currency</code> .
<code>Direction</code>	Giá trị thuộc kiểu liệt kê <code>System.Data.ParameterDirection</code> , cho biết hướng truyền dữ liệu cho thông số. Các giá trị hợp lệ là <code>Input</code> , <code>InputOutput</code> , <code>Output</code> , và <code>ReturnValue</code> .
<code>IsNullable</code>	Giá trị <code>bool</code> , cho biết thông số có chấp nhận giá trị <code>null</code> hay không.
<code>ParameterName</code>	Chuỗi chứa tên thông số.
<code>Value</code>	Đối tượng chứa giá trị của thông số.

Để sử dụng thông số với một câu lệnh text, bạn phải cho biết vị trí cần thay thế giá trị thông số bên trong câu lệnh. *ODBC*, *OLE DB*, và *SQL Server CE Data Provider* hỗ trợ các thông số vị trí (*positional parameter*); vị trí của mỗi thông số được nhận biết bằng dấu chấm hỏi (?). Ví dụ, câu lệnh dưới đây cho biết có hai vị trí cần được thay thế bằng các giá trị thông số:

```
UPDATE Employees SET Title = ? WHERE EmployeeId = ?
```

SQL Server và *Oracle Data Provider* hỗ trợ các thông số được đặt tên (*named parameter*), cho phép bạn chỉ định mỗi vị trí thông số bằng một tên với biểu tượng @ đứng trước. Dưới đây là câu lệnh tương đương như trên nhưng sử dụng thông số được đặt tên:

```
UPDATE Employees SET Title = @title WHERE EmployeeId = @id
```

Để chỉ định các giá trị thông số để thay thế vào một câu lệnh, bạn phải tạo các đối tượng thông số đúng kiểu và thêm chúng vào tập hợp thông số của đối tượng lệnh (có thể được truy xuất thông qua thuộc tính `Parameters`). Bạn có thể thêm các thông số được đặt tên theo thứ tự bất kỳ, nhưng bạn phải thêm các thông số vị trí theo đúng thứ tự mà chúng xuất hiện trong câu lệnh text. Khi bạn thực thi câu lệnh, giá trị của mỗi thông số sẽ được thay thế vào trong chuỗi lệnh trước khi câu lệnh được thực thi dựa trên data-source.

Phương thức `ParameterizedCommandExample` được trình bày ở đây mô tả cách sử dụng thông số trong lệnh **UPDATE** của *SQL Server*. Các đối số của phương thức `ParameterizedCommandExample` gồm một `SqlConnection` đang mở và hai chuỗi. Giá trị của hai chuỗi này được thay thế vào trong lệnh **UPDATE** bằng thông số. Ví dụ sau đây trình bày hai

cách để tạo đối tượng thông số: phương thức `IDbCommand.CreateParameter`, và phương thức `IDbCommand.Parameters`. Bạn cũng có thể tạo các đối tượng thông số bằng phương thức khởi dựng và cấu hình chúng bằng các đối số của phương thức khởi dựng hoặc thông qua việc thiết lập các thuộc tính của chúng.

```
public static void ParameterizedCommandExample(SqlConnection con,
    string employeeID, string title) {

    // Tạo và cấu hình một câu lệnh mới chứa 2 thông số được đặt tên.
    SqlCommand com = con.CreateCommand();
    com.CommandType = CommandType.Text;
    com.CommandText = "UPDATE Employees SET Title = @title" +
        " WHERE EmployeeId = @id";

    // Tạo đối tượng SqlParameter cho thông số title.
    SqlParameter p1 = com.CreateParameter();
    p1.ParameterName = "@title";
    p1.SqlDbType = SqlDbType.VarChar;
    p1.Value = title;
    com.Parameters.Add(p1);

    // Sử dụng cú pháp tốc ký để thêm thông số id.
    com.Parameters.Add("@id", SqlDbType.Int).Value = employeeID;

    // Thực thi câu lệnh và xử lý kết quả.
    int result = com.ExecuteNonQuery();

}
```

Khi sử dụng thông số để thực thi thủ tục tồn trữ, bạn phải cung cấp các đối tượng thông số đáp ứng cho mỗi đối số do thủ tục tồn trữ yêu cầu—gồm cả đối số input và output. Bạn phải thiết lập thuộc tính `Direction` của mỗi thông số như được mô tả trong bảng 10.4 (mặc định là `Input`). Nếu thủ tục tồn trữ có giá trị trả về, thông số giữ giá trị trả về (với thuộc tính `Direction` là `ReturnValue`) phải là thông số đầu tiên được thêm vào tập hợp thông số. Ví dụ dưới đây sử dụng thông số để thực thi thủ tục tồn trữ:

```
public static void StoredProcedureExample(SqlConnection con,
    string category, string year) {

    // Tạo và cấu hình một câu lệnh mới.
```

```

SqlCommand com = con.CreateCommand();
com.CommandType = CommandType.StoredProcedure;
com.CommandText = "SalesByCategory";

// Tạo đối tượng SqlParameter cho thông số category.
com.Parameters.Add("@CategoryName", SqlDbType.NVarChar).Value
    = category;

// Tạo đối tượng SqlParameter cho thông số year.
com.Parameters.Add("@OrdYear", SqlDbType.NVarChar).Value = year;

// Thực thi câu lệnh và xử lý kết quả.
using (IDataReader reader = com.ExecuteReader()) {
    $
}
}

```

5. *Xử lý kết quả của truy vấn SQL bằng data-reader*

? Bạn cần xử lý dữ liệu chứa trong đối tượng `System.Data.IDataReader` (đối tượng này được trả về khi bạn thực thi phương thức `IDbCommand.ExecuteReader`—đã được thảo luận trong mục 10.3).

✕ Sử dụng các thành viên của đối tượng `IDataReader` để duyệt tuần tự các hàng trong tập kết quả và truy xuất các item dữ liệu chứa trong mỗi hàng.

Giao diện `IDataReader` mô tả một data-reader, đây là một cơ chế chỉ-tiến, chỉ-đọc (*forward-only, read-only*) để truy xuất kết quả của truy vấn *SQL*. Mỗi data-provider chứa một hiện thực `IDataReader` duy nhất. Dưới đây là danh sách các hiện thực `IDataReader` cho năm data-provider chuẩn:

- `System.Data.Odbc.OdbcDataReader`
- `System.Data.OleDb.OleDbDataReader`
- `System.Data.OracleClient.OracleDataReader`
- `System.Data.SqlClient.SqlCeDataReader`
- `System.Data.SqlClient.SqlDataReader`

Giao diện `IDataReader` thừa kế giao diện `System.Data.IDataRecord`. Các giao diện này khai báo các chức năng truy xuất dữ liệu và cấu trúc của dữ liệu có trong tập kết quả. Bảng 10.5 mô tả vài thành viên thông dụng của giao diện `IDataReader` và `IDataRecord`.

Ngoài các thành viên được liệt kê trong bảng 10.5, data-reader còn cung cấp một tập các phương thức thực hiện việc lấy dữ liệu đã được định kiểu từ hàng hiện tại. Mỗi phương thức sau đây nhận vào một đối số nguyên cho biết chỉ số (đánh từ 0) của cột mà dữ liệu sẽ được trả

về từ cột này: `GetBoolean`, `GetByte`, `GetBytes`, `GetChar`, `GetChars`, `GetDateTime`, `GetDecimal`, `GetDouble`, `GetFloat`, `GetGuid`, `GetInt16`, `GetInt32`, `GetInt64`, `GetString`, `GetValue`, và `GetValues`.

Bảng 10.5 Các thành viên thông dụng của các lớp data-reader

Thành viên	Mô tả
Thuộc tính	
FieldCount	Lấy số cột trong hàng hiện tại.
IsClosed	Trả về <code>true</code> nếu <code>IDataReader</code> bị đóng; <code>false</code> nếu nó hiện đang mở.
Item	Trả về một đối tượng mô tả giá trị của cột cụ thể trong hàng hiện tại. Cột có thể được chỉ định bằng một chỉ số nguyên (đánh số từ 0) hoặc một chuỗi chứa tên cột. Bạn phải ép giá trị trả về thành kiểu phù hợp. Đây là bộ chỉ mục (<i>indexer</i>) cho các lớp data-reader.
Phương thức	
GetDataTypeName	Lấy tên của kiểu dữ liệu đối với một cột cụ thể.
GetFieldType	Lấy đối tượng <code>System.Type</code> mô tả kiểu dữ liệu của giá trị chứa trong cột cụ thể (cột này được chỉ định bằng một chỉ số nguyên—đánh số từ 0).
GetName	Lấy tên của cột cụ thể (cột này được chỉ định bằng một chỉ số nguyên—đánh số từ 0).
GetOrdinal	Lấy số thứ tự cột (đánh số từ 0) ứng với một tên cột cụ thể.
GetSchemaTable	Trả về đối tượng <code>System.Data.DataTable</code> chứa siêu dữ liệu mô tả các cột có trong <code>IDataReader</code> .
IsDBNull	Trả về <code>true</code> nếu giá trị trong cột cụ thể chứa giá trị <code>null</code> ; nếu không thì trả về <code>false</code> .
NextResult	Nếu <code>IDataReader</code> chứa nhiều tập kết quả vì có nhiều lệnh được thực thi, <code>NextResult</code> sẽ di chuyển đến các tập kết quả kế tiếp. Theo mặc định, <code>IDataReader</code> được bố trí tại tập kết quả đầu tiên.
Read	Dịch reader đến bản ghi kế tiếp. Reader luôn bắt đầu tại bản ghi đầu tiên.

`Data-reader Server` và `Oracle` cũng chứa các phương thức thực hiện việc lấy dữ liệu thuộc các kiểu dữ liệu đặc trưng của data-source. Ví dụ, `SqlDataReader` chứa các phương thức như `GetSqlByte`, `GetSqlDecimal`, và `GetSqlMoney`, và `OracleDataReader` chứa các phương thức như `GetOracleLob`, `GetOracleNumber`, và `GetOracleMonthSpan`. Bạn hãy tham khảo tài liệu *.NET Framework SDK* để biết thêm chi tiết.

Khi đã hoàn tất với data-reader, bạn nên gọi phương thức `Close` để có thể sử dụng lại kết nối cơ sở dữ liệu. `IDataReader` thừa kế `System.IDisposable`; nghĩa là mỗi lớp data-reader đều hiện thực phương thức `Dispose`. Phương thức này sẽ tự động gọi `Close`, cho nên lệnh `using` là một cách rất rõ ràng và hiệu quả khi sử dụng data-reader.

Ví dụ dưới đây mô tả cách sử dụng data-reader để xử lý nội dung của hai tập kết quả được trả về bởi việc thực thi một truy vấn gồm hai lệnh `SELECT`. Tập kết quả đầu tiên được liệt kê và hiển thị trong cửa sổ *Console*. Tập kết quả thứ hai được duyệt qua để tìm và hiển thị thông tin siêu dữ liệu.

```
using System;
using System.Data;
using System.Data.SqlClient;

public class DataReaderExample {

    public static void Main() {

        // Tạo đối tượng SqlConnection mới.
        using (SqlConnection con = new SqlConnection()) {

            // Cấu hình chuỗi kết nối của đối tượng SqlConnection.
            con.ConnectionString = "Data Source = localhost;" +
                "Database = Northwind; Integrated Security=SSPI";

            // Tạo và cấu hình câu lệnh mới.
            SqlCommand com = con.CreateCommand();
            com.CommandType = CommandType.Text;
            com.CommandText = "SELECT BirthDate, FirstName, LastName " +
                "FROM Employees ORDER BY BirthDate; " +
                "SELECT * FROM Employees";

            // Mở kết nối cơ sở dữ liệu và thực thi câu lệnh.
            con.Open();

            // Thực thi câu lệnh và thu lấy SqlDataReader.
            using (SqlDataReader reader = com.ExecuteReader()) {

                // Xử lý tập kết quả đầu tiên
                // và hiển thị nội dung của tập kết quả.
                Console.WriteLine("Employee Birthdays (By Age).");
```

```

while (reader.Read()) {

    Console.WriteLine("{0,18:D} - {1} {2}",
        reader.GetDateTime(0), // Lấy dữ liệu định kiểu
        reader["FirstName"],   // Sử dụng chỉ số chuỗi
        reader[2]);            // Sử dụng chỉ số thứ tự
}

// Xử lý tập kết quả thứ hai và hiển thị chi tiết
// về các cột và các kiểu dữ liệu trong tập kết quả.
reader.NextResult();
Console.WriteLine("Employee Table Metadata.");
for (int field = 0; field < reader.FieldCount; field++) {

    Console.WriteLine("  Column Name:{0}  Type:{1}",
        reader.GetName(field),
        reader.GetDataTypeName(field));
}

}

}

Console.ReadLine();
}
}

```

6. Thu lấy tài liệu XML từ truy vấn SQL Server

- ? **Bạn cần thực thi một truy vấn dựa trên *SQL Server 2000* hoặc *MSDE* và lấy các kết quả dạng XML.**
- ✕ **Sử dụng mệnh đề `FOR XML` trong truy vấn *SQL* để trả về kết quả dạng XML. Thực thi câu lệnh bằng phương thức `SqlCommand.ExecuteReader`, kết quả trả về là một đối tượng `System.Xml.XmlReader` mà thông qua nó bạn có thể truy xuất dữ liệu XML.**

SQL Server 2000 và *MSDE* trực tiếp hỗ trợ XML. Bạn chỉ cần thêm mệnh đề `FOR XML AUTO` vào cuối truy vấn *SQL* để cho biết rằng các kết quả sẽ được trả về ở dạng XML. Theo mặc định, dạng XML này không phải là một tài liệu XML đầy đủ. Thay vào đó, nó trả về kết quả của mỗi bản ghi theo từng phần tử (*element*) riêng rẽ, với tất cả các trường (*field*) đều là đặc tính (*attribute*). Ví dụ, truy vấn sau đây:

```
SELECT CustomerID, CompanyName FROM Customers FOR XML AUTO
```

sẽ trả về *XML* với cấu trúc như sau:

```
<Customers CustomerID="ALFKI" CompanyName="Alfreds Futterkiste"/>
<Customers CustomerID="ANTON" CompanyName="Antonio Moreno Taquería"/>
<Customers CustomerID="GOURL" CompanyName="Gourmet Lanchonetes"/>
$
```

Bạn có thể thêm từ khóa `ELEMENTS` vào cuối truy vấn để định dạng kết quả theo các phần tử lồng nhau. Ví dụ, truy vấn sau đây:

```
SELECT CustomerID, CompanyName FROM Customers FOR XML AUTO, ELEMENTS
```

sẽ trả về *XML* với cấu trúc như sau:

```
<Customers>
  <CustomerID>ALFKI</CustomerID>
  <CompanyName>Alfreds Futterkiste</CompanyName>
</Customers>
<Customers>
  <CustomerID>ANTON</CustomerID>
  <CompanyName>Antonio Moreno Taquería</CompanyName>
</Customers>
<Customers>
  <CustomerID>GOURL</CustomerID>
  <CompanyName>Gourmet Lanchonetes</CompanyName>
</Customers>
$
```



Bạn cũng có thể định dạng kết quả một cách chi tiết hơn bằng cú pháp `FOR XML EXPLICIT`. Ví dụ, cú pháp này cho phép bạn chuyển một vài trường (*field*) thành đặc tính (*attribute*) và các trường khác thành phần tử (*element*). Bạn hãy tham khảo một quyển sách chuyên về *SQL Server* để biết thêm chi tiết.

Ví dụ dưới đây trình bày cách lấy kết quả dạng *XML* bằng mệnh đề `FOR XML` và phương thức `ExecuteXmlReader`. Chú ý rằng, kết nối không thể được sử dụng cho bất kỳ câu lệnh nào khác trong khi `XmlReader` đang mở. Bạn nên xử lý kết quả càng nhanh càng tốt và phải luôn đóng `XmlReader` lại (Chương 5 có rất nhiều ví dụ trình bày cách sử dụng lớp `XmlReader`).

```
using System;
using System.Xml;
using System.Data;
using System.Data.SqlClient;

public class XmlQueryExample {
```



```
public static void Main() {

    // Tạo đối tượng SqlConnection mới.
    using (SqlConnection con = new SqlConnection()) {

        // Cấu hình chuỗi kết nối của đối tượng SqlConnection.
        con.ConnectionString = "Data Source = localhost;" +
            "Database = Northwind; Integrated Security=SSPI";

        // Tạo và cấu hình câu lệnh mới có chứa FOR XML AUTO.
        SqlCommand com = con.CreateCommand();
        com.CommandType = CommandType.Text;
        com.CommandText = "SELECT CustomerID, CompanyName" +
            " FROM Customers FOR XML AUTO";

        // Khai báo XmlReader để nó có thể được tham chiếu trong
        // khối finally (bảo đảm đóng nó lại sau khi sử dụng).
        XmlReader reader = null;

        try {
            // Mở kết nối cơ sở dữ liệu.
            con.Open();

            // Thực thi câu lệnh và lấy XmlReader
            // để truy xuất các kết quả.
            reader = com.ExecuteXmlReader();

            while (reader.Read()) {

                Console.WriteLine("Element: " + reader.Name);
                if (reader.HasAttributes) {
                    for (int i = 0; i < reader.AttributeCount; i++) {

                        reader.MoveToAttribute(i);
                        Console.WriteLine("  {0}: {1}",
                            reader.Name, reader.Value);
                    }
                }
            }
        }
    }
}
```

```

        reader.MoveToElement();
        Console.WriteLine();
    }
}
} catch (Exception ex) {

    Console.WriteLine(ex.ToString());
} finally {

    // Bảo đảm reader đã đóng.
    if (reader != null) reader.Close();
}
}

Console.ReadLine();
}
}

```

Dưới đây là một vài kết xuất từ ứng dụng thử nghiệm này:

```

Element: Customers  CustomerID: ALFKI  CompanyName: Alfreds Futterkiste
Element: Customers  CustomerID: ANTON  CompanyName: Antonio Moreno Taquería
Element: Customers  CustomerID: GOURL  CompanyName: Gourmet Lanchonetes
...

```

Thay vì làm việc với `XmlReader` và truy xuất dữ liệu một cách tuần tự, bạn có thể đọc dữ liệu *XML* vào `System.Xml.XmlDocument`. Theo cách này, tất cả dữ liệu được lấy vào bộ nhớ, và kết nối cơ sở dữ liệu có thể đóng lại. Kế đó, bạn có thể tiếp tục tương tác với tài liệu *XML* (Chương 5 có rất nhiều ví dụ trình bày cách sử dụng lớp `XmlDocument`). Dưới đây là đoạn mã mà bạn sẽ cần:

```

XmlDocument doc = new XmlDocument();

// Tạo đối tượng SqlConnection mới.
using (SqlConnection con = new SqlConnection()) {

    // Cấu hình chuỗi kết nối của đối tượng SqlConnection.
    con.ConnectionString = "Data Source = localhost;" +
        "Database = Northwind; Integrated Security=SSPI";
}
}

```

```
// Tạo và cấu hình câu lệnh mới có chứa FOR XML AUTO.
SqlCommand com = con.CreateCommand();
com.CommandType = CommandType.Text;
com.CommandText =
    "SELECT CustomerID, CompanyName FROM Customers FOR XML AUTO";

// Mở kết nối cơ sở dữ liệu.
con.Open();

// Load dữ liệu XML vào XmlDocument. Cần phải tạo trước một
// phần tử gốc để có thể đặt mỗi phần tử hàng kết quả vào đó.
XmlReader reader = com.ExecuteXmlReader();
doc.LoadXml("<results></results>");

// Tạo XmlNode từ phần tử XML kế tiếp (được đọc từ reader).
XmlNode newNode = doc.ReadNode(reader);

while (newNode != null) {

    doc.DocumentElement.AppendChild(newNode);
    newNode = doc.ReadNode(reader);
}
}

// Xử lý XmlDocument đã ngắt kết nối.
Console.WriteLine(doc.OuterXml);
§
```

7. Nhận biết tất cả các thể hiện SQL Server 2000 trên mạng

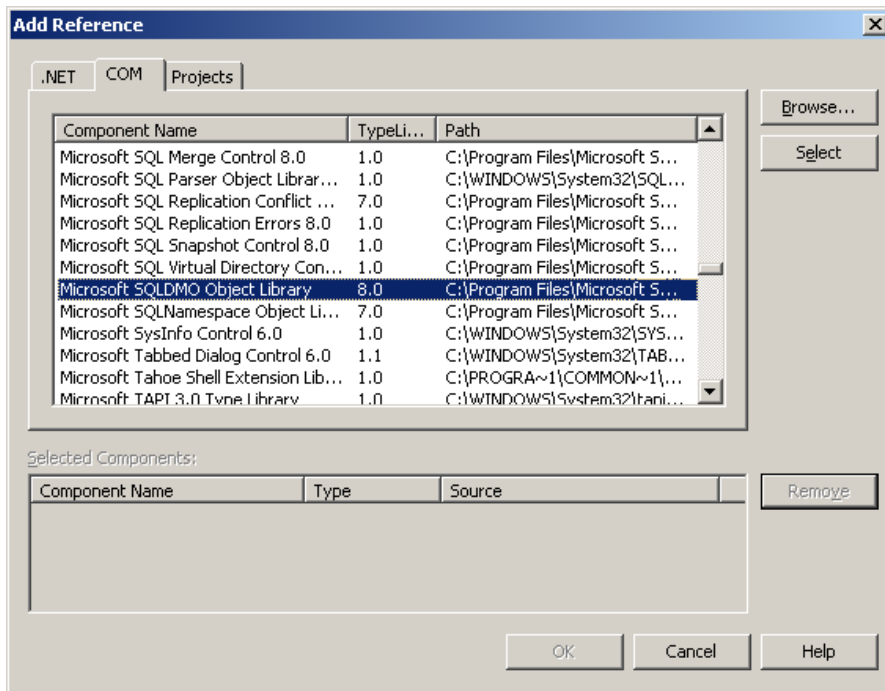
?

Bạn cần lấy danh sách tất cả các thể hiện của *SQL Server 2000* có thể truy xuất được trên mạng.

✕

Sử dụng *COM Interop* để truy xuất chức năng của *Microsoft SQLDMO Object Library*. Tạo một đối tượng *Application* rồi gọi phương thức *ListAvailableSQLServers* của nó. *ListAvailableSQLServers* sẽ trả về đối tượng *NameList*, là một tập hợp chứa tên của mỗi đối tượng *SQL Server 2000* được tìm thấy trên mạng.

Thư viện lớp *.NET Framework* không có chức năng tìm các *SQL Server* chưa biết; tuy nhiên, công việc này không mấy khó khăn với *Microsoft SQLDMO Object Library* (được truy xuất qua *COM Interop*). Mục 15.6 sẽ trình bày chi tiết cách tạo một *Interop Assembly* thực hiện việc truy xuất đến một thành phần *COM*. Nếu đang sử dụng *Microsoft Visual Studio .NET*, bạn hãy thêm một tham chiếu đến *Microsoft SQLDMO Object Library* được liệt kê trong thẻ *COM* của hộp thoại *Add Reference* (xem hình 10.1).



Hình 10.1 Chọn Microsoft SQLDMO Object Library trong hộp thoại Add Reference

Nếu không có *Visual Studio .NET*, bạn hãy sử dụng *Type Library Importer (tlbimp.exe)* để tạo một *Interop Assembly* cho file *sqldma.dll* (thường nằm trong thư mục *\Program Files\Microsoft SQL Server\80\Tools\Binn*).



Có một vấn đề đã được tìm thấy trong bản gốc *SQLDMO Object Library*. Để có thể chạy được dự án này, bạn cần phải cài đặt *SQL Server Service Pack 2* hoặc mới hơn.

Giả sử bạn sử dụng các thiết lập mặc định khi tạo *Interop Assembly* cho mình, trước hết bạn cần nhập không gian tên *SQLDMO*. Để lấy được danh sách các *SQL Server* đang có hiệu lực, bạn hãy tạo một đối tượng *SQLDMO.Application* và gọi phương thức *ListAvailableSQLServers* của nó. Mỗi chuỗi trong đối tượng trả về *SQLDMO.NameList* là tên của một *SQL Server* đang có hiệu lực. Bạn có thể sử dụng các tên này trong chuỗi kết nối hoặc hiển thị chúng trong một danh

sách cho người dùng chọn. Ví dụ dưới đây sẽ hiển thị tên của tất cả các *SQL Server* có thể truy xuất được trong cửa sổ *Console*:

```
using System;
using SQLDMO;

public class SQLDMOExample {

    public static void Main() {

        // Thu lấy danh sách tất cả các SQL Server có hiệu lực.
        SQLDMO.Application app = new SQLDMO.Application();
        SQLDMO.NameList names = app.ListAvailableSQLServers();

        // Xử lý tập hợp NameList.
        if (names.Count == 0) {

            Console.WriteLine("No SQL Servers visible on the network.");

        } else {

            // Hiển thị danh sách các SQL Server có hiệu lực.
            Console.WriteLine("SQL Servers visible : " + names.Count);

            foreach (string name in names) {

                Console.WriteLine("  Name : " + name);

            }

        }

        Console.ReadLine();

    }

}
```

8.

Đọc file Excel với ADO.NET



Bạn muốn thu lấy hay chèn dữ liệu vào một tài liệu *Microsoft Excel* bằng *ADO.NET*.



Sử dụng *ODBC.NET provider* kết hợp với *Microsoft Excel ODBC Driver*.

Không có *OLE DB provider* hay provider được-quản-lý nào cho *Excel*. Tuy nhiên, bạn có thể sử dụng *Microsoft Excel ODBC Driver* (được cài đặt mặc định cùng với *Excel*) kết hợp với *ODBC .NET provider* (đi kèm với *.NET Framework 1.1* và *Visual Studio .NET 2003*).

Trong chuỗi kết nối, bạn cần chỉ định driver mà bạn đang sử dụng và tên file *Excel*. Ví dụ dưới đây chỉ đến file *test.xls* trong thư mục startup của ứng dụng:

```
private string ConnectionString = "Driver={Microsoft Excel Driver
(*.xls)};DriverId=790;Dbq=" + Application.StartupPath + "\\test.xls;";
```

Sau khi kết nối, bạn có thể thực hiện hai kiểu thao tác: **SELECT** hay **INSERT**. Thay vì sử dụng bảng, bạn chọn hay chèn dựa vào tên sheet. Tên sheet phải kết thúc bằng dấu đô la (\$) và được đặt trong dấu ngoặc vuông (nếu không, sẽ sinh ra lỗi cú pháp). Định dạng bị bỏ qua, và hàng đầu tiên tự động được sử dụng làm các tên cột.

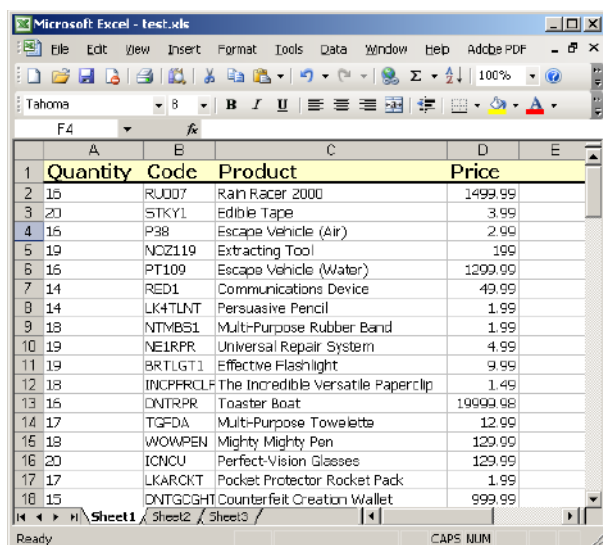
Ví dụ dưới đây trích và hiển thị tất cả các hàng trong *Sheet1*. Hình 10.2 là file *Excel* gốc. Hình 10.3 là dữ liệu được trình bày trên form.

```
private void ExcelView_Load (System.Object sender, System.EventArgs e)
{

    OdbcConnection Con = new OdbcConnection(ConnectionString);
    OdbcDataAdapter Adapter = new OdbcDataAdapter("SELECT * FROM [Sheet1$]",
        Con);
    DataSet Ds = new DataSet();

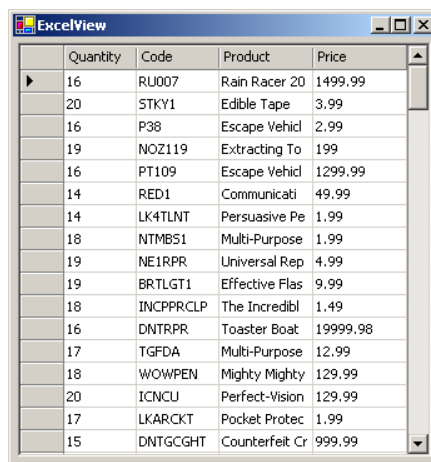
    try
    {
        Con.Open();
        Adapter.Fill(Ds, "Sheet1");
    }
    catch (Exception Err)
    {
        MessageBox.Show(Err.ToString());
    }
    finally
    {
        Con.Close();
    }

    grid.DataSource = Ds.Tables["Sheet1"];
}
```



	A	B	C	D	E
1	Quantity	Code	Product	Price	
2	16	RU007	Rain Racer 2000	1499.99	
3	20	STKY1	Edible Tape	3.99	
4	16	P38	Escape Vehicle (Air)	2.99	
5	19	NOZ119	Extracting Tool	199	
6	16	PT109	Escape Vehicle (Water)	1299.99	
7	14	RED1	Communications Device	49.99	
8	14	LK4TLNT	Persuasive Pencil	1.99	
9	18	NTMB51	Multi-Purpose Rubber Band	1.99	
10	19	NE1RPR	Universal Repair System	4.99	
11	19	BRTLGT1	Effective Flashlight	9.99	
12	18	INCPRLCLP	The Incredible Versatile Paperclip	1.49	
13	16	DNTRPR	Toaster Boat	19999.98	
14	17	TGFDA	Multi-Purpose Towalette	12.99	
15	18	WOWPEN	Mighty Mighty Pen	129.99	
16	20	ICNCU	Perfect-Vision Glasses	129.99	
17	17	LKARCKT	Pocket Protector Rocket Pack	1.99	
18	15	DNTGCGHT	Counterfeit Creation Wallet	999.99	

Hình 10.2 File Excel gốc



	Quantity	Code	Product	Price
16	RU007	Rain Racer 20	1499.99	
20	STKY1	Edible Tape	3.99	
16	P38	Escape Vehicle	2.99	
19	NOZ119	Extracting To	199	
16	PT109	Escape Vehicl	1299.99	
14	RED1	Communicati	49.99	
14	LK4TLNT	Persuasive Pe	1.99	
18	NTMB51	Multi-Purpose	1.99	
19	NE1RPR	Universal Rep	4.99	
19	BRTLGT1	Effective Flas	9.99	
18	INCPRLCLP	The Incredibl	1.49	
16	DNTRPR	Toaster Boat	19999.98	
17	TGFDA	Multi-Purpose	12.99	
18	WOWPEN	Mighty Mighty	129.99	
20	ICNCU	Perfect-Vision	129.99	
17	LKARCKT	Pocket Protec	1.99	
15	DNTGCGHT	Counterfeit Cr	999.99	

Hình 10.3 Dữ liệu Excel trong ứng dụng .NET

Một cách tiếp cận khác là sử dụng *Automation* để vận hành *Excel* thông qua các giao diện *COM* do nó cung cấp. Cách này đòi hỏi bạn sử dụng *COM Interop* và các đối tượng của *Excel*, và chỉ làm việc khi *Excel* đã được cài đặt trên máy tính. Tuy vậy, nó cung cấp rất nhiều chức năng cho việc tương tác với dữ liệu bảng tính.

9.

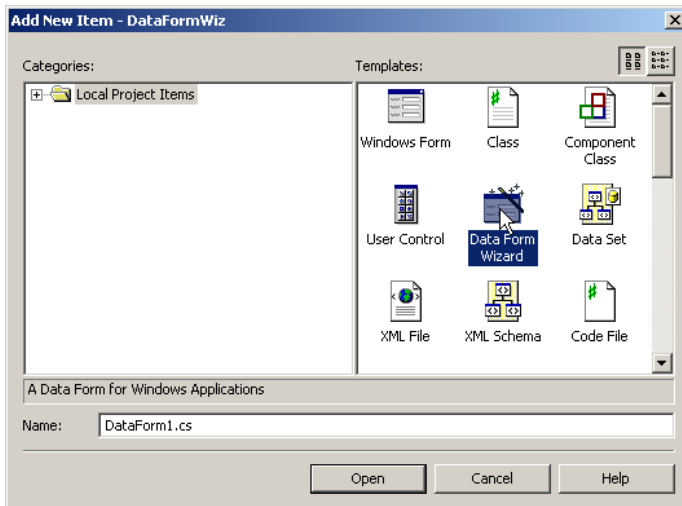
Sử dụng Data Form Wizard

? Bạn muốn xây dựng một ứng dụng cơ sở dữ liệu với đầy đủ chức năng nhưng không phải viết bất cứ dòng mã nào.

✂ Sử dụng *Data Form Wizard*.

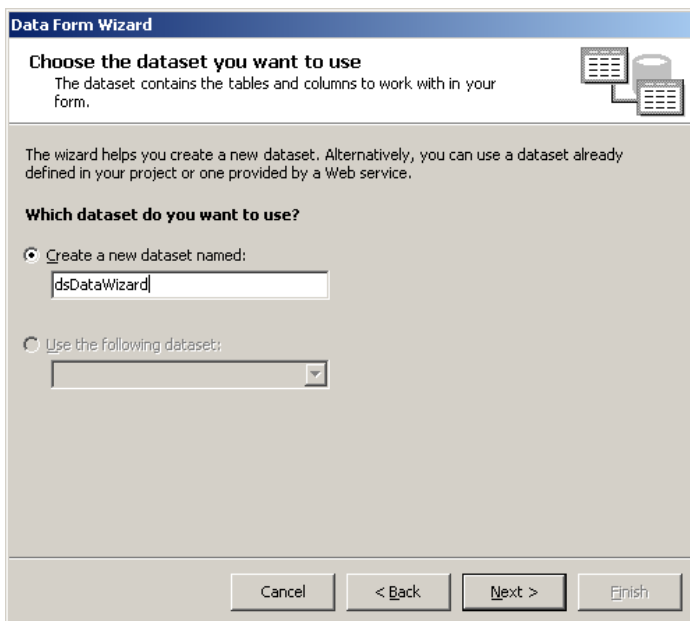
Để sử dụng *Data Form Wizard*, bạn hãy thực hiện các bước dưới đây:

1. Tạo một dự án mới, chọn mẫu *Empty Project*. Đặt tên dự án là *DataFormWiz*. Nhấp *OK*.
2. Từ thanh trình đơn chính của *IDE*, chọn *Project | Add New Item* để hiển thị hộp thoại *Add New Item* (xem hình 10.4).



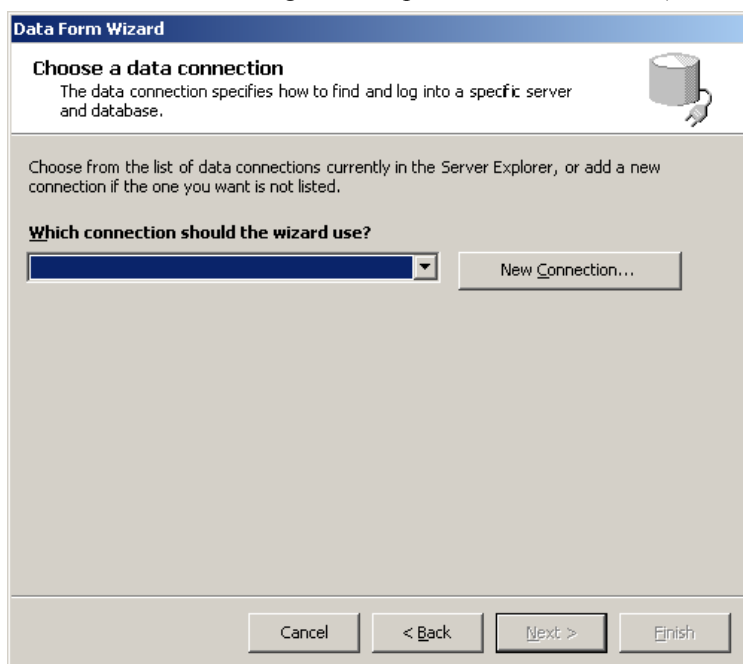
Hình 10.4 Hộp thoại Add New Item

3. Chọn *Data Form Wizard*, và giữ nguyên tên mặc định *DataForm1.cs*. Nhấp *Open* để thêm *Data Form Wizard* vào dự án. Ngay khi bạn nhấp *Open*, *Data Form Wizard* sẽ khởi chạy. Nhấp *Next*.
4. Đặt tên cho tập dữ liệu mới là *dsDataWizard* (xem hình 10.5). Nhấp *Next*.



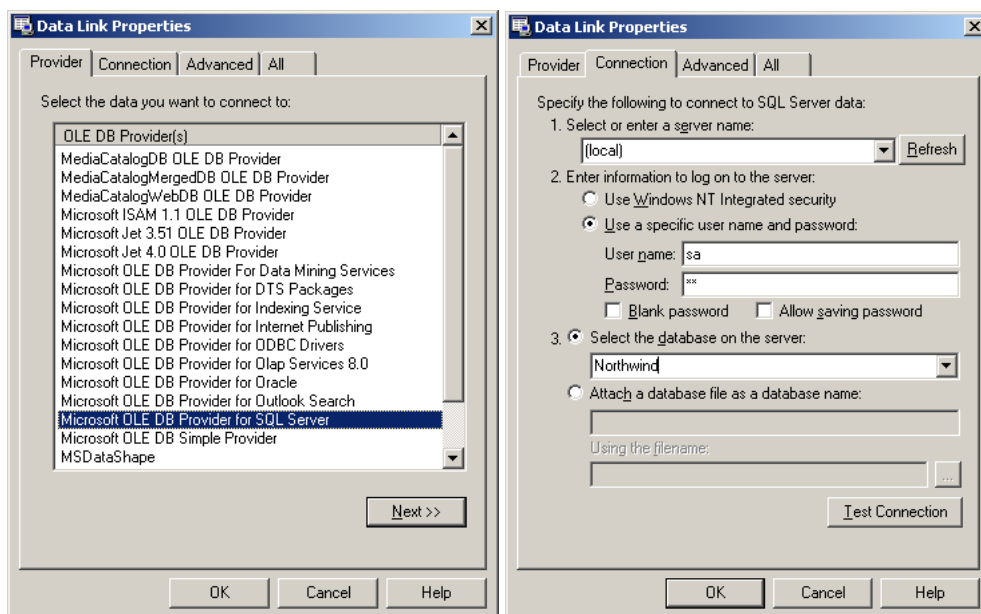
Hình 10.5 Tạo một tập dữ liệu mới với tên là dsDataWizard

5. Tạo một kết nối mới bằng cách nhấp nút *New Connection* (xem hình 10.6).



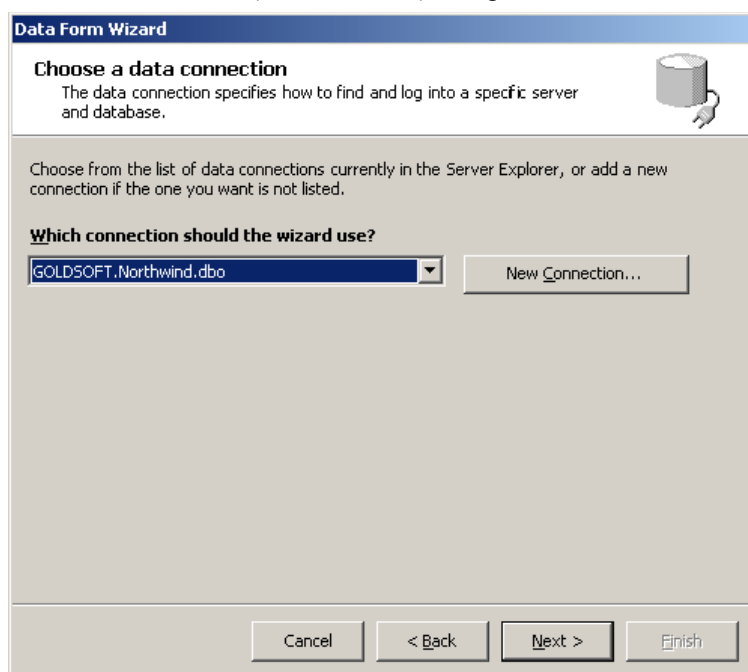
Hình 10.6 Nhấp nút New Connection để tạo kết nối mới

6. Kết nối đến cơ sở dữ liệu *Northwind* của *SQL Server* trong hộp thoại *Data Link Properties* (xem hình 10.7). Nhấp *OK*.



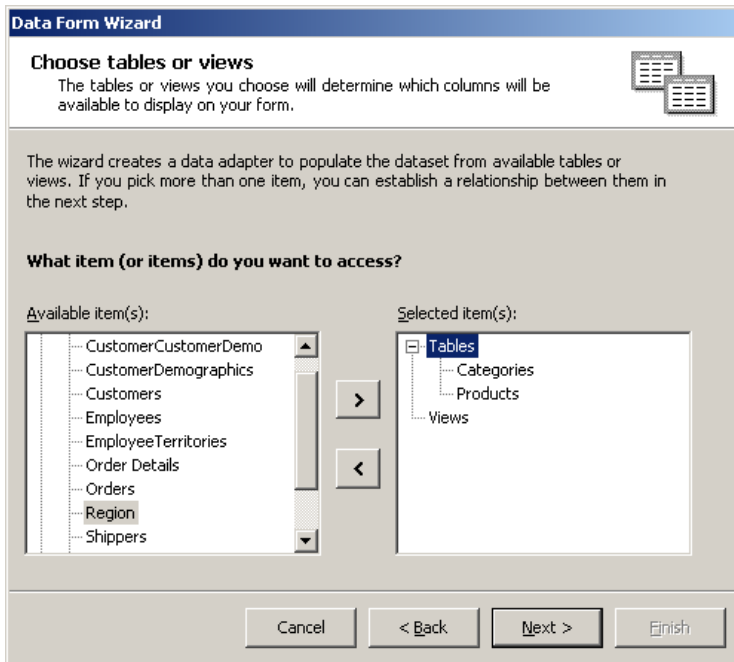
Hình 10.7 Hộp thoại Data Link Properties

7. Chọn kết nối vừa tạo (xem hình 10.8). Nhấp *Next*.



Hình 10.8 Chọn kết nối Northwind vừa mới tạo

8. Thêm bảng *Categories* và *Products* vào danh sách *Selected Item(s)* (xem hình 10.9). Nhấp *Next*.



Hình 10.9 Thêm bảng Categories và Products

9. Chúng ta cần đặt tên cho quan hệ giữa các bảng. Gõ *CategoriesProducts* vào hộp *Name*. *Categories* là bảng cha với khóa chính là *CategoryID*. *Products* là bảng con với khóa ngoại là *CategoryID* (xem hình 10.10). Nhấp nút > để thêm quan hệ này vào hộp *Relations* bên phải. Nhấp *Next*.

Data Form Wizard

Create a relationship between tables
The wizard will use the relationships to generate code that keeps the tables synchronized as you work with them.

Relationships are based on common keys between tables. Name your new relation, choose the parent and child tables and key fields, and then add it to the relations list using the arrow button.

Name: CategoriesProducts

Parent table: Categories

Child table: Products

Keys: CategoryID

Relations:

Cancel < Back Next > Finish

Hình 10.10 Tạo quan hệ

10. Trong cửa sổ kế, chúng ta chọn các bảng và các trường dữ liệu cần hiển thị. Giữ nguyên tất cả các trường dữ liệu trừ trường *Picture* thuộc bảng *Categories* (xem hình 10.11). Nhấp *Next*.

Data Form Wizard

Choose tables and columns to display on the form
Your form can display any of the tables and columns available in the dataset.

If you display more than one table on the form, the tables have a master-detail relationship

What tables and columns do you want to display on the form?

Master or single table: Categories

Detail table: Products

Columns:

- ☒ CategoryID
- ☒ CategoryName
- ☒ Description
- ☐ Picture

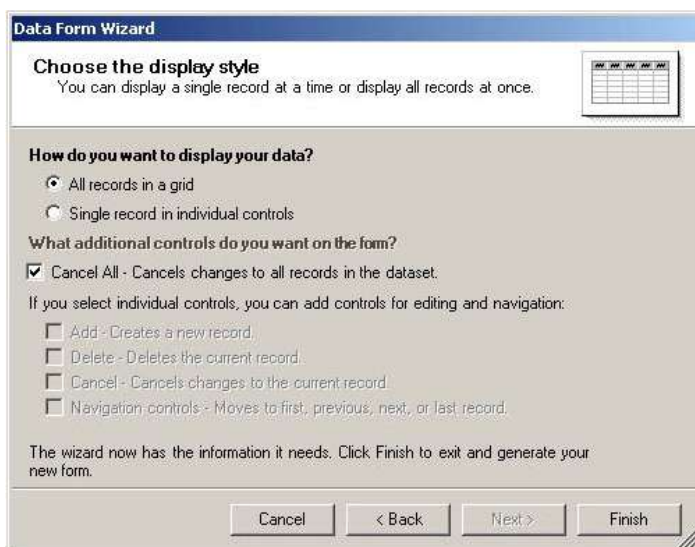
Columns:

- ☒ ProductID
- ☒ ProductName
- ☒ SupplierID
- ☒ CategoryID
- ☒ QuantityPerUnit
- ☒ UnitPrice

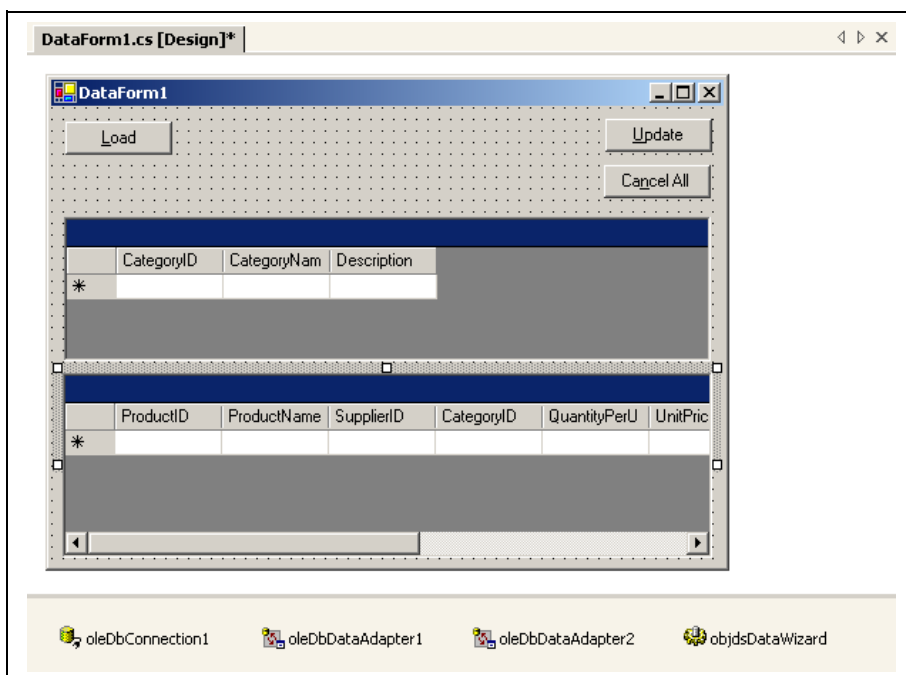
Cancel < Back Next > Finish

Hình 10.11 Chọn các bảng và các trường dữ liệu cần hiển thị

11. Chúng ta muốn hiển thị tất cả các bản ghi trong một khung lưới, cho nên giữ nguyên các thiết lập mặc định (xem hình 10.12). Nhấp *Finish* để kết thúc trình thuật sĩ. Sau vài giây, *DataForm* mới sẽ được kết sinh và hiển thị (xem hình 10.13).

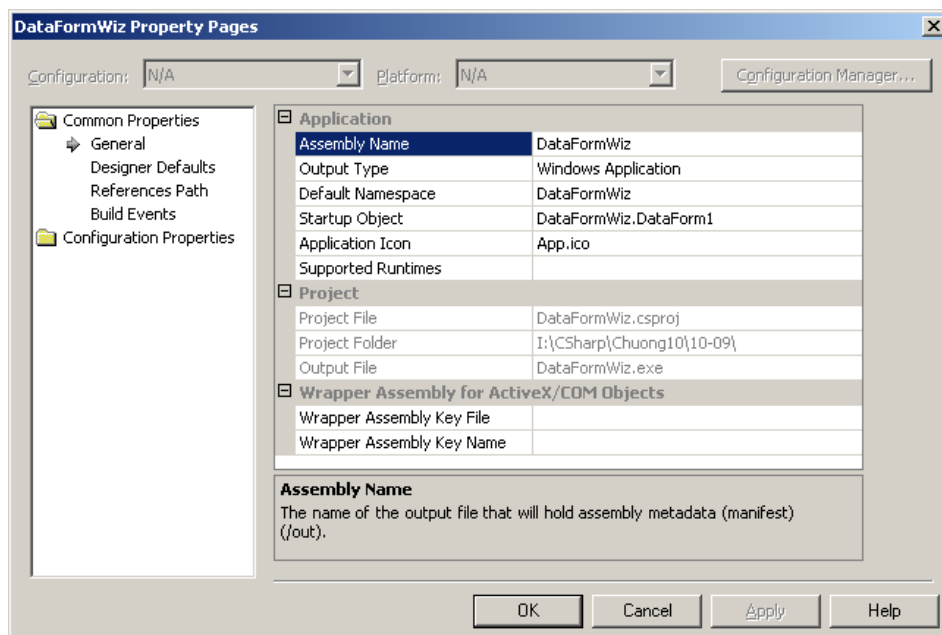


Hình 10.12 Giữ nguyên các thiết lập hiển thị mặc định



Hình 10.13 DataForm1.vb

12. Cuối cùng, bạn phải báo cho ứng dụng biết form *Data Form Wizard* sẽ là đối tượng startup. Nhấp phải vào dự án trong *Solution Explorer*, và chọn *Properties*. Chọn *DataFormWiz.DataForm1* từ danh sách *Startup object*. Nhấp *OK*.



Hình 10.14 Chọn DataFormWiz.DataForm1 từ danh sách Startup object

Nhấn *F5* để chạy chương trình. Nhấp nút *Load* để thiết lập kết nối cơ sở dữ liệu và thu lấy dữ liệu (xem hình 10.15). Cuộn bảng *Categories*, và chọn một một bản ghi. Để ý rằng bảng *Products* bên dưới thay đổi theo quan hệ mà chúng ta đã xây dựng. Bạn có thể thay đổi bất cứ trường dữ liệu nào trong bất kỳ bản ghi nào. Nếu nút *Update* được nhấp vào, tất cả những thay đổi được thực hiện trong các khung lưới dữ liệu sẽ được truyền về nguồn dữ liệu *Northwind*. Và bạn không phải viết bất kỳ dòng mã nào!

CategoryID	CategoryName	Description
4	Dairy Product	Cheeses
5	Grains/Cereal	Breads, crack
6	Meat/Poultry	Prepared me
7	Produce	Dried fruit an

ProductID	ProductName	SupplierID	CategoryID	QuantityPerU	UnitPrice
9	Mishi Kobe Ni	4	6	18 - 500 g pk	97
17	Alice Mutton	7	6	20 - 1 kg tins	39
29	Thüringer Ro	12	6	50 bags x 30	123.79
53	Perth Pasties	24	6	48 pieces	32.8
54	Tourtière	25	6	16 pies	7.45
55	Bâté chinois	26	6	24 boxes x 2	24

Hình 10.15 Dự án Data Form Wizard

10.

Sử dụng Crystal Report Wizard



Bạn muốn tạo *Crystal Report* từ một nguồn dữ liệu.



Xây dựng một bản báo cáo bằng *Crystal Report Wizard*. Sau đó, sử dụng điều kiện kiểm tra *CrystalReportViewer* để trình bày bản báo cáo này trên form.

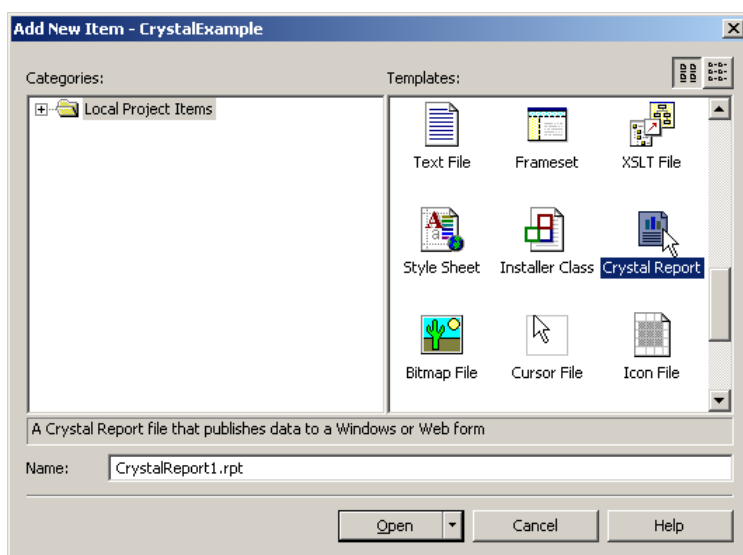
Crystal Reports for Visual Studio .NET là công cụ báo cáo chuẩn dành cho *Visual Studio .NET*. Bạn có thể tiếp quản những báo cáo này trên nền *Web* và *Windows*, và phân bổ chúng ở dạng dịch vụ *Web* trên một server.

Mục này sẽ trình bày từng bước một cách sinh báo cáo từ một nguồn dữ liệu. Trước tiên, bạn hãy tạo một dự án ứng dụng *Windows* mới với tên là *CrystalExample*.

Để có thể hiển thị *Crystal Report*, bạn cần thực hiện hai bước chính: thứ nhất là xây dựng một bản báo cáo, và thứ hai là thêm một điều kiện kiểm tra *CrystalReportViewer* vào form để trình bày báo cáo.

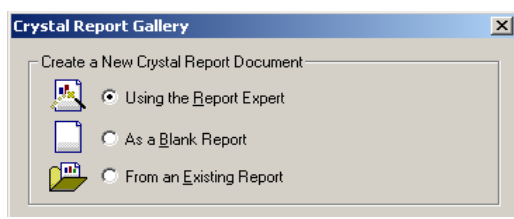
Việc xây dựng một bản báo cáo được mô tả trong 11 bước nhỏ dưới đây:

1. Từ thanh trình đơn chính của *IDE*, chọn *Project | Add New Item* và chọn *Crystal Report* (xem hình 10.16). Nhấp *Open*.



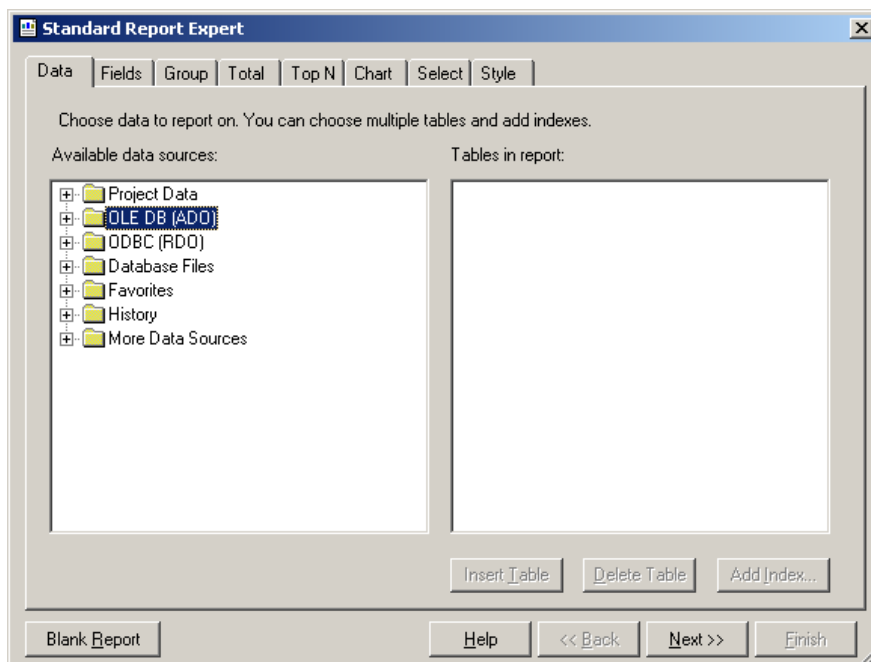
Hình 10.16 Thêm Crystal Report vào dự án

2. *Crystal Report Wizard* sẽ nhắc ta chọn kiểu báo cáo cần xây dựng (xem hình 10.17). Giữ nguyên báo cáo chuẩn mặc định, rồi nhấp *OK*.



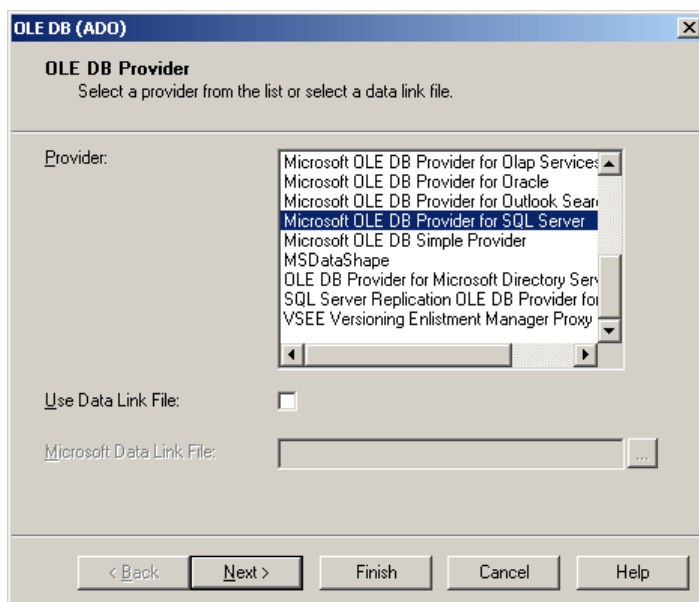
Hình 10.17 Giữ nguyên báo cáo chuẩn mặc định

3. *Crystal Report Wizard* hiển thị hộp thoại *Standard Report Expert* (xem hình 10.18). Nhấp vào dấu cộng kế thư mục *OLE DB (ADO)* trong hộp danh sách *Available Data Sources*.



Hình 10.18 Hộp thoại Standard Report Expert

4. Bạn sẽ thấy hộp thoại *OLE DB (ADO)* (xem hình 10.19). Chọn *Microsoft OLE DB Provider for SQL Server*, rồi nhấp *Next*.



Hình 10.19 Chọn Microsoft OLE DB Provider for SQL Server

5. Kế tiếp, chúng ta cần báo với trình thuật sĩ cơ sở dữ liệu nào sẽ được kết nối. Chọn cơ sở dữ liệu *Northwind* (xem hình 10.20). Nhấp *Next*.

OLE DB (ADD)

Connection Information
Provide necessary information to log on to the chosen data source.

Server: (local) [v]

User ID: sa

Password: [password field]

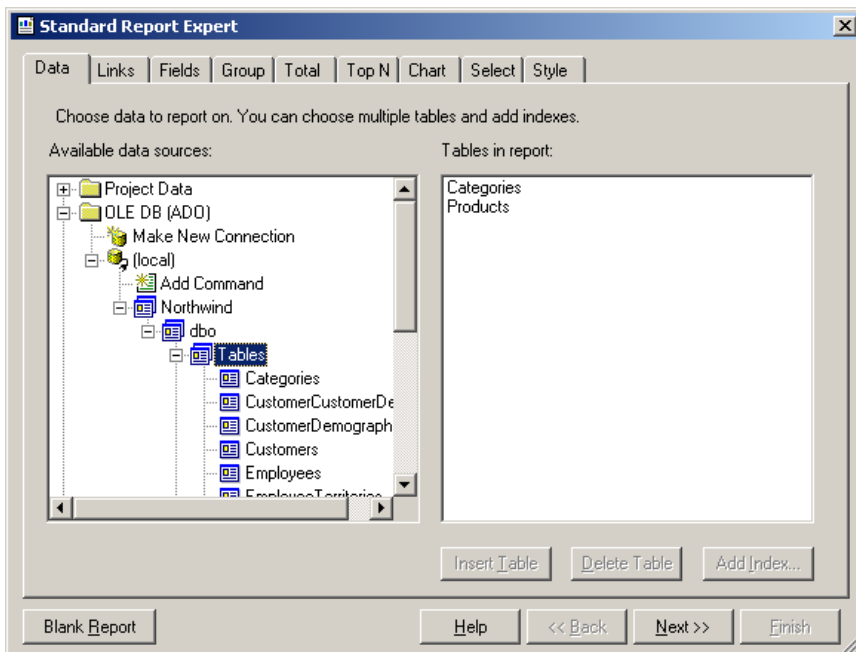
Database: Northwind [v]

Integrated Security: ☐

< Back **Next >** Finish Cancel Help

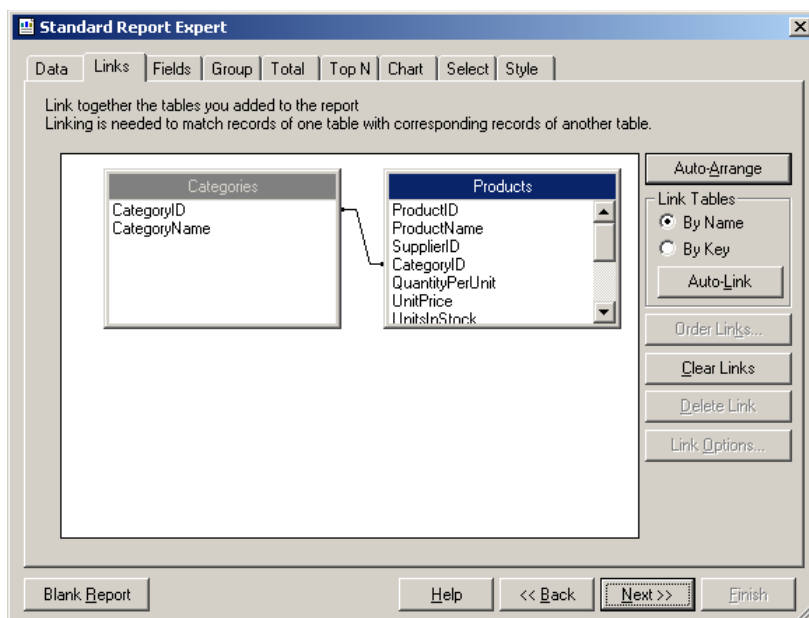
Hình 10.20 Chọn cơ sở dữ liệu Northwind

6. *Crystal Report Wizard* hiển thị hộp thoại *Advanced Information*. Chúng ta không cần thay đổi thông tin nào cho ví dụ này, cho nên nhấp *Finish*. Trong cửa sổ *Standard Report Expert*, chọn bảng *Categories* và *Products* cho bản báo cáo của chúng ta (xem hình 10.21). Nhấp *Next*.



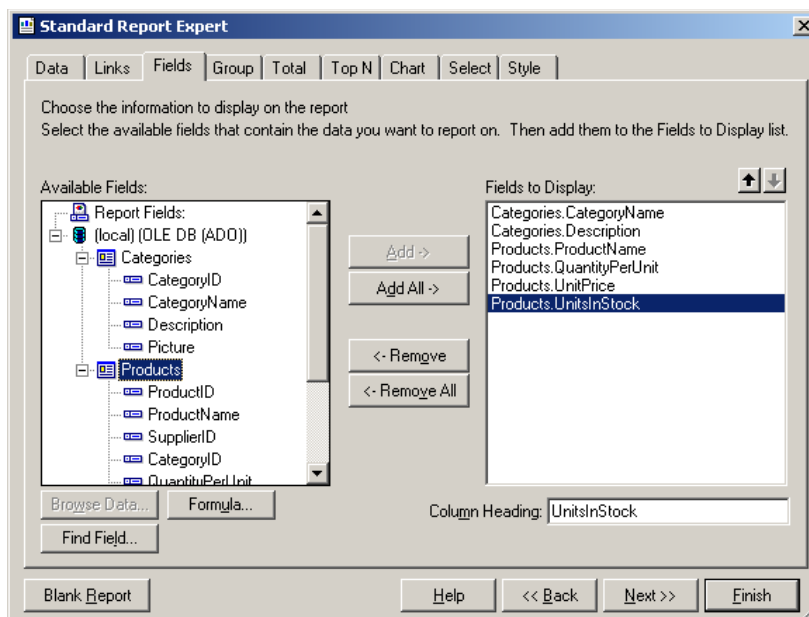
Hình 10.21 Chọn bảng Categories và Products

- Trong thẻ *Links* của hộp thoại *Standard Report Expert*, giữ nguyên các mặc định như hình 10.22. Bạn có thể thấy trình thuật sĩ ánh xạ khóa chính trong *Categories* đến khóa ngoại trong *Products*. Nhấp *Next* để hiển thị thẻ *Fields*.



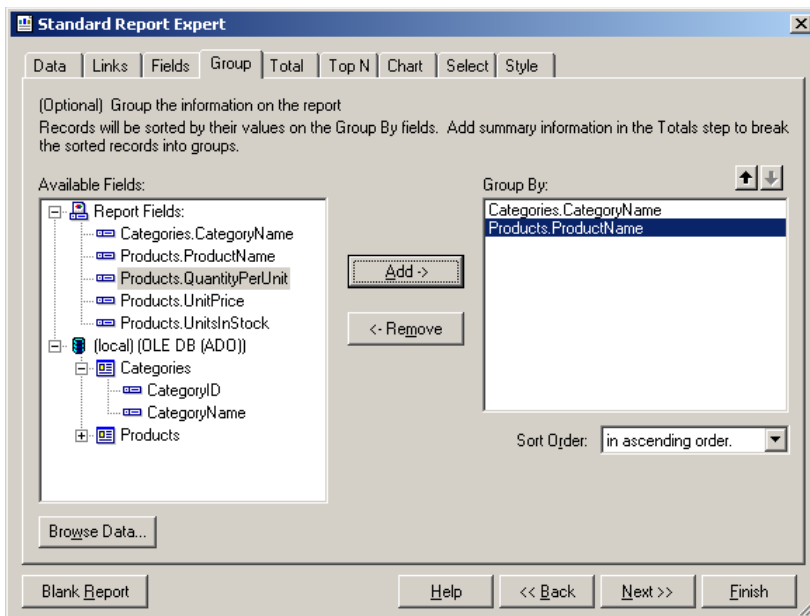
Hình 10.22 Khóa chính trong Categories được ánh xạ đến khóa ngoại trong Products

8. Từ bảng *Categories*, chọn *CategoryName* và *Description*. Từ bảng *Products*, chọn *ProductName*, *QuantityPerUnit*, *UnitPrice*, và *UnitsInStock* (xem hình 10.23). Nhấp *Next* để hiển thị thẻ *Group*.



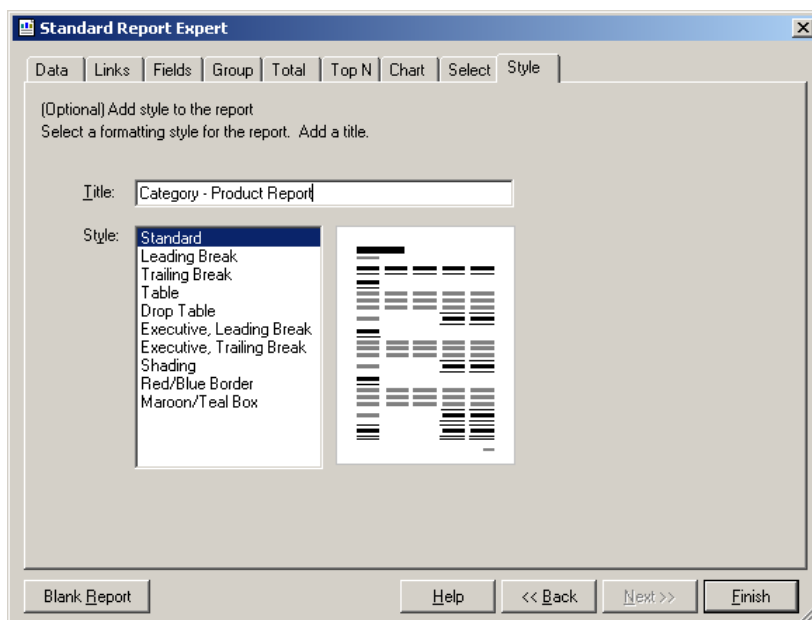
Hình 10.23 Chọn các trường dữ liệu cần hiển thị

9. Chọn *CategoryName* và *ProductName* để phân nhóm (xem hình 10.24).



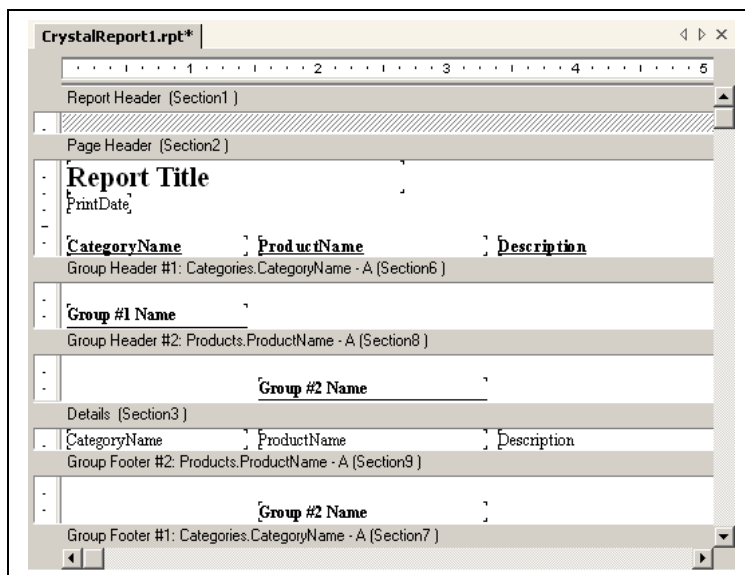
Hình 10.24 Chọn *CategoryName* và *ProductName* để phân nhóm

10. Nhấp *Next* nhiều lần để chấp nhận các thiết lập mặc định cho các thẻ *Total*, *Top N*, *Chart*, và *Select*. Khi đến thẻ cuối cùng (thẻ *Style*), gõ tiêu đề *Category - Product Report* (xem hình 10.25).



Hình 10.25 Gõ tiêu đề Category - Product Report

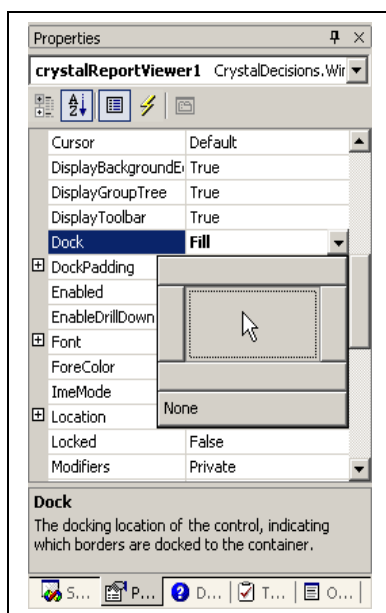
11. Nhấp *Finish*. File *CrystalReport1.rpt* (xem hình 10.26) sẽ được thêm vào dự án.



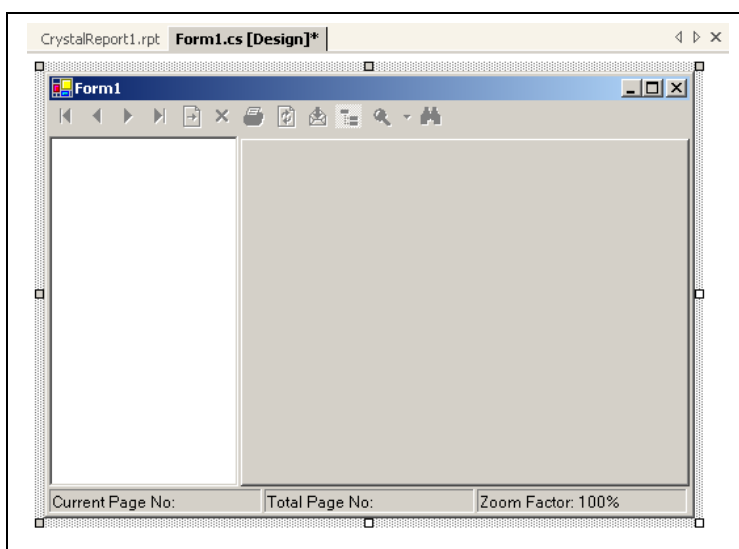
Hình 10.26 Bản báo cáo đã được thiết kế

Chúng ta đã xây dựng xong bản báo cáo. Bước thứ hai là thiết lập một cơ chế để xem bản báo cáo này. Bước này khá dễ dàng. Bạn hãy kéo điều khiển *CrystalReportViewer* từ hộp công cụ vào form mặc định.

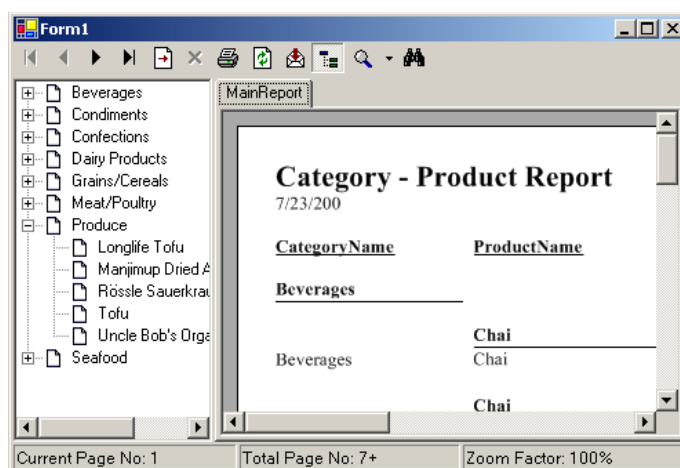
Nhấp phải vào *CrystalReportViewer*, và chọn *Properties*. Tìm thuộc tính *Dock* và chọn *Fill* (xem hình 10.27). Tùy chọn này sẽ khiến *CrystalReportViewer* lấp đầy vùng client của form. Kế tiếp, tìm thuộc tính *ReportSource*, và nhấp *Browse*. Chọn file *CrystalReport1.rpt* mà chúng ta vừa xây dựng. Bây giờ, bạn hãy chạy chương trình để xem kết quả (xem hình 10.29).



Hình 10.27 Tìm thuộc tính *Dock* và chọn *Fill*



Hình 10.28 Form báo cáo hoàn chỉnh



Hình 10.29 Trình bày báo cáo với Crystal Report

Như bạn có thể thấy, điều khiển `CrystalReportViewer` khá tinh vi. Bạn có thể sử dụng các chức năng có sẵn để phân trang và in báo cáo. Bạn có thể nhấn nút *Export Report* (hình phong bì với mũi tên đỏ) để xuất báo cáo thành một file *Adobe Acrobat (.pdf)*, *Microsoft Excel (.xls)*, *Microsoft Word (.doc)*, hay *Rich Text Document (.rtf)*. Bạn cũng có thể phóng to bản báo cáo hoặc tìm kiếm text bên trong bản báo cáo.

Trình thuật sĩ này mạnh đến nỗi bạn không phải viết dòng mã nào cả. Một file mã nguồn báo cáo, chứa một lớp báo cáo cho bản báo cáo này, tự động được sinh ra. Lớp báo cáo này có lớp cơ sở là `ReportClass`.

11

LẬP TRÌNH MẠNG

Microsoft .NET Framework bao gồm một tập các lớp dùng để lập trình mạng thuộc hai không gian tên: *System.Net* và *System.Net.Sockets*. Các lớp này hỗ trợ mọi thứ, từ lập trình dựa-trên-socket với *TCP/IP* cho đến download file và trang *HTML* từ web thông qua *HTTP*. Hai không gian tên này cũng là nền tảng cho hai nền networking cấp cao hơn—*Remoting* và dịch vụ *Web XML*. Hai nền này sẽ được đề cập chi tiết trong chương 12. Chương này sẽ trình bày các vấn đề sau:

- Lấy tài nguyên từ web thông qua *HTTP* (mục 11.1, 11.2, và 11.3).
- Hiển thị một trang web trong một ứng dụng dựa-trên-*Windows* bằng điều khiển *Web Browser* (mục 11.4).
- Lấy địa chỉ *IP* và thông tin *DNS* về máy tính hiện hành và các miền khác trên *World Wide Web* (mục 11.5 và 11.6).
- Gửi thông điệp “ping” (mục 11.7) và giao tiếp bằng giao thức *TCP* và *UDP* (mục 11.8 đến 11.13).
- Gửi và nhận e-mail (mục 11.14 và 11.15).

1.

Download file thông qua HTTP



Bạn cần một cách thật nhanh và đơn giản để download một file từ một website thông qua *HTTP*.



Sử dụng phương thức tĩnh *DownloadFile* của lớp *System.Net.WebClient*.

.NET Framework cung cấp vài cơ chế dùng để gửi dữ liệu thông qua *HTTP*. Một trong những cách dễ nhất là sử dụng lớp *System.Net.WebClient*. Nó cung cấp những phương thức mức-cao như *DownloadFile* và *UploadFile*. Các phương thức này không có sự hỗ trợ nội tại nào cho giao tiếp bất đồng bộ, hay xác thực. Nếu cần các tính năng này, bạn có thể sử dụng các chức năng phức tạp hơn do lớp *WebRequest* và *WebResponse* cung cấp (sẽ được mô tả trong mục 11.2 và 11.3).

Chương trình ví dụ dưới đây sẽ download file *winXP.gif* từ *localhost* và lưu vào đĩa.

```
using System;
using System.Net;
using System.IO;

public class Download {

    private static void Main() {

        string remoteUri =
            "http://localhost/winXP.gif";
        string localFileName = "winXP.gif";

        WebClient client = new WebClient();
```

```

        Console.WriteLine("Downloading file " +
            remoteUri + " to " + Path.GetFullPath(localFileName));

        // Thực hiện download.
        client.DownloadFile(remoteUri, localFileName);
        Console.WriteLine("Download complete.");

        Console.ReadLine();
    }
}

```

2. *Download và xử lý file bằng stream*

- ?** Bạn cần lấy một file từ một website, nhưng không muốn lưu trực tiếp vào đĩa. Thay vào đó, bạn muốn xử lý ngay trong ứng dụng của mình.
- ✖** Sử dụng lớp `WebRequest` để tạo yêu cầu, lớp `WebResponse` để nhận đáp ứng từ web-server, và một số dạng reader (`StreamReader` đối với dữ liệu *HTML* hay text, hoặc `BinaryReader` đối với dữ liệu nhị phân) để phân tích đáp ứng đó.

Download một file cần bốn bước cơ bản sau:

1. Sử dụng phương thức tĩnh `Create` của lớp `System.Net.WebRequest` để chỉ định trang bạn cần. Phương thức này trả về một đối tượng dẫn xuất từ `WebRequest`, phụ thuộc vào kiểu *Uniform Resource Identifier (URI)* bạn sử dụng. Ví dụ, nếu tài nguyên là *HTTP* (với cụm từ *http://*), nó sẽ tạo ra đối tượng `HttpWebRequest`; nếu tài nguyên là file (với cụm từ *file://*), nó sẽ tạo ra đối tượng `FileWebRequest`. Bạn có thể thiết lập thời gian trễ thông qua thuộc tính `WebRequest.Timeout`.
2. Sử dụng phương thức `GetResponse` của đối tượng `WebRequest`, phương thức này trả về một đối tượng `WebResponse` cho trang. Nếu yêu cầu vượt qua thời gian trễ thì ngoại lệ `WebException` sẽ bị ném.
3. Tạo một `StreamReader` hoặc một `BinaryReader` cho `WebResponse`.
4. Công việc cuối cùng là xử lý stream này, chẳng hạn ghi nó ra file rồi hiển thị trong ứng dụng của bạn...

Đoạn mã dưới đây sẽ lấy và hiển thị một file ảnh và nội dung *HTML* của một trang web.

```

using System;
using System.Net;
using System.IO;
using System.Drawing;
using System.Windows.Forms;

```

```
public class DownloadForm : System.Windows.Forms.Form {

    private System.Windows.Forms.PictureBox picBox;
    private System.Windows.Forms.TextBox textBox;

    // (Bỏ qua phần mã designer.)

    private void DownloadForm_Load(object sender, System.EventArgs e) {

        string picUri =
            "http://localhost/winXP.gif";
        string htmlUri =
            "http://localhost/iishelp/iis/misc/default.asp";

        // Tạo yêu cầu.
        WebRequest requestPic = WebRequest.Create(picUri);
        WebRequest requestHtml = WebRequest.Create(htmlUri);

        // Nhận đáp ứng. Công việc này sẽ mất nhiều
        // thời gian, đặc biệt khi file cần lấy quá lớn.
        WebResponse responsePic = requestPic.GetResponse();
        WebResponse responseHtml = requestHtml.GetResponse();

        // Đọc response stream.
        Image downloadedImage =
            Image.FromStream(responsePic.GetResponseStream());
        StreamReader r =
            new StreamReader(responseHtml.GetResponseStream());
        string htmlContent = r.ReadToEnd();
        r.Close();

        // Hiển thị ảnh.
        picBox.Image = downloadedImage;

        // Hiển thị nội dung dạng text của trang HTML.
        textBox.Text = htmlContent;
    }
}
```



Hình 11.1 Download nội dung của một trang web

Để download file lớn một cách hiệu quả, bạn có thể sử dụng kỹ thuật bất đồng bộ đã được mô tả trong chương 4. Bạn cũng có thể sử dụng phương thức `WebRequest.BeginGetResponse`, phương thức này không chặn mã lệnh của bạn và sẽ gọi thủ tục callback khi nhận được đáp ứng.

3. Lấy trang HTML từ một website có yêu cầu xác thực

- ? **Bạn cần thu lấy một file từ một website, nhưng website đó yêu cầu một số thông tin xác thực.**
- ✕ **Sử dụng lớp `WebRequest` và `WebResponse` đã được mô tả trong mục 11.2. Tuy nhiên, trước khi gửi yêu cầu, bạn phải cấu hình thuộc tính `WebRequest.Credentials` với các thông tin xác thực.**

Một số website yêu cầu thông tin xác thực từ người dùng. Khi kết nối thông qua trình duyệt, thông tin này có thể được cung cấp một cách trong suốt (ví dụ, một intranet-site cục bộ có sử dụng *Integrated Windows authentication*), hoặc trình duyệt có thể yêu cầu thông tin này thông qua một hộp thoại đăng nhập. Khi truy xuất một trang web bằng lập trình thì mã lệnh của bạn cần phải cung cấp các thông tin này.

Cách tiếp cận mà bạn sử dụng tùy thuộc vào kiểu xác thực mà website sử dụng.

- Nếu website sử dụng *Basic authentication* hay *Digest authentication*, bạn cần gửi kết hợp username và password, bằng cách tạo một đối tượng `System.Net.NetworkCredential` và gán nó vào thuộc tính `WebRequest.Credentials`.
- Nếu website sử dụng *Integrated Windows authentication*, bạn thực hiện tương tự như trên. Bạn có thể lấy các thông tin đăng nhập của người dùng hiện hành từ đối tượng `System.Net.CredentialCache`.
- Nếu website đòi hỏi chứng chỉ, bạn cần nạp chứng chỉ từ một file bằng lớp `System.Security.Cryptography.X509Certificates.X509Certificate`, và thêm nó vào tập hợp `HttpWebRequest.ClientCertificates`.

Dưới đây là đoạn mã ví dụ cho cả ba cách tiếp cận trên:

```
using System;
using System.Net;
using System.Security.Cryptography.X509Certificates;

public class DownloadWithAuthentication {

    private static void Main() {

        string uriBasic, uriIntegrated, uriCertificate;

        // Xác thực username và password với Basic authentication.
        WebRequest requestA = WebRequest.Create(uriBasic);
        requestA.Credentials =
            new NetworkCredential("userName", "password");
        requestA.PreAuthenticate = true;

        // Đăng nhập người dùng hiện hành với
        // Integrated Windows authentication.
        WebRequest requestB = WebRequest.Create(uriIntegrated);
        requestB.Credentials = CredentialCache.DefaultCredentials;
        requestB.PreAuthenticate = true;

        // Xác thực người dùng bằng chữ ký điện tử.
        HttpWebRequest requestC = (HttpWebRequest)
            WebRequest.Create(uriCertificate);
        X509Certificate cert =
            X509Certificate.CreateFromCertFile(@"c:\user.cer");
```

```
requestC.ClientCertificates.Add(cert);

// (Bây giờ bắt đầu lấy và xử lý đáp ứng.)
}
}
```



Nhớ rằng, nếu muốn sử dụng kiểu xác thực chứng chỉ, bạn phải nạp chứng chỉ từ file. Không có cách nào để tạo đối tượng `X509Certificate` từ một chứng chỉ trong kho chứng chỉ của máy tính.

4. *Hiển thị trang web trong Ứng dụng dựa-trên-Windows*



Bạn cần hiển thị một trang *HTML* (hay bất kỳ kiểu tài liệu nào mà *Microsoft Internet Explorer* hỗ trợ) trong một ứng dụng dựa-trên-*Windows*.

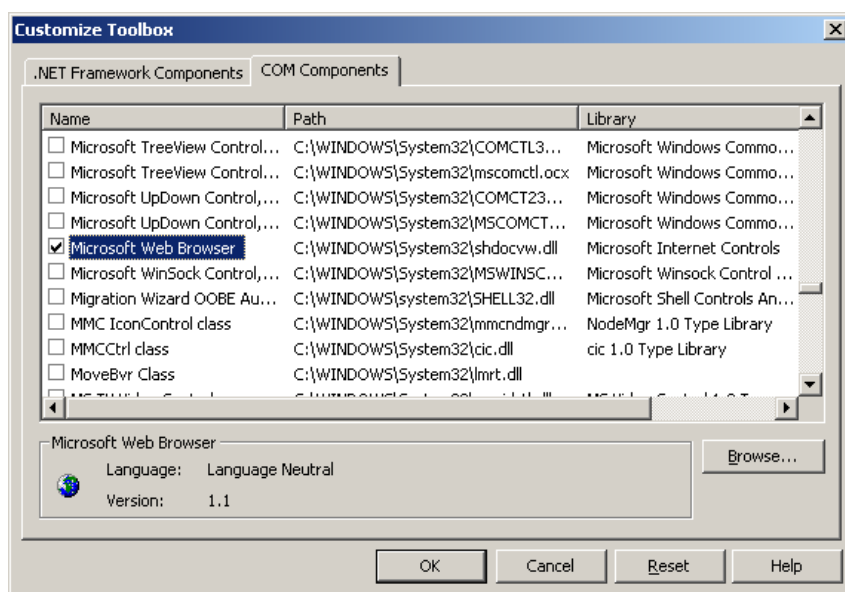


Sử dụng điều khiển *ActiveX Web Browser* (đi cùng với *Internet Explorer*).

.NET Framework không cung cấp điều khiển nào dùng để dịch nội dung *HTML*. Tuy nhiên, chức năng này lại rất cần thiết cho việc hiển thị một nội dung *HTML* cục bộ (kể cả những tài liệu *HTML* đóng gói theo định dạng file trợ giúp của *Windows*) cũng như các thông tin nào đó từ web.

Để hiển thị một trang *HTML*, bạn có thể thêm một cửa sổ *Internet Explorer* vào ứng dụng dựa-trên-*Windows*. Cửa sổ này không chỉ hỗ trợ *HTML*, mà còn hỗ trợ *JavaScript*, *Microsoft Visual Basic Scripting Edition (VBScript)*, *ActiveX control*, và rất nhiều plug-in khác, tùy thuộc vào cấu hình hệ thống của bạn (có thể gồm cả *Microsoft Word*, *Microsoft Excel*, và *Adobe Acrobat Reader*). Bạn còn có thể sử dụng điều khiển *Web Browser* để duyệt một thư mục trong ổ đĩa cục bộ, hay hiển thị các file trên một *FTP (File Transfer Protocol) site*.

Để thêm *Web Browser* vào một dự án trong *Microsoft Visual Studio .NET*, bạn hãy nhấp phải vào hộp công cụ và chọn *Add/Remove Items*. Sau đó, chọn thẻ *COM Components*, và chọn *Microsoft Web Browser* (xem hình 11.2). Theo đó, điều khiển *Microsoft Web Browser* sẽ được thêm vào hộp công cụ. Khi bạn thả điều khiển này vào form, các *Interop Assembly* cần thiết sẽ được tạo ra và thêm vào dự án của bạn.



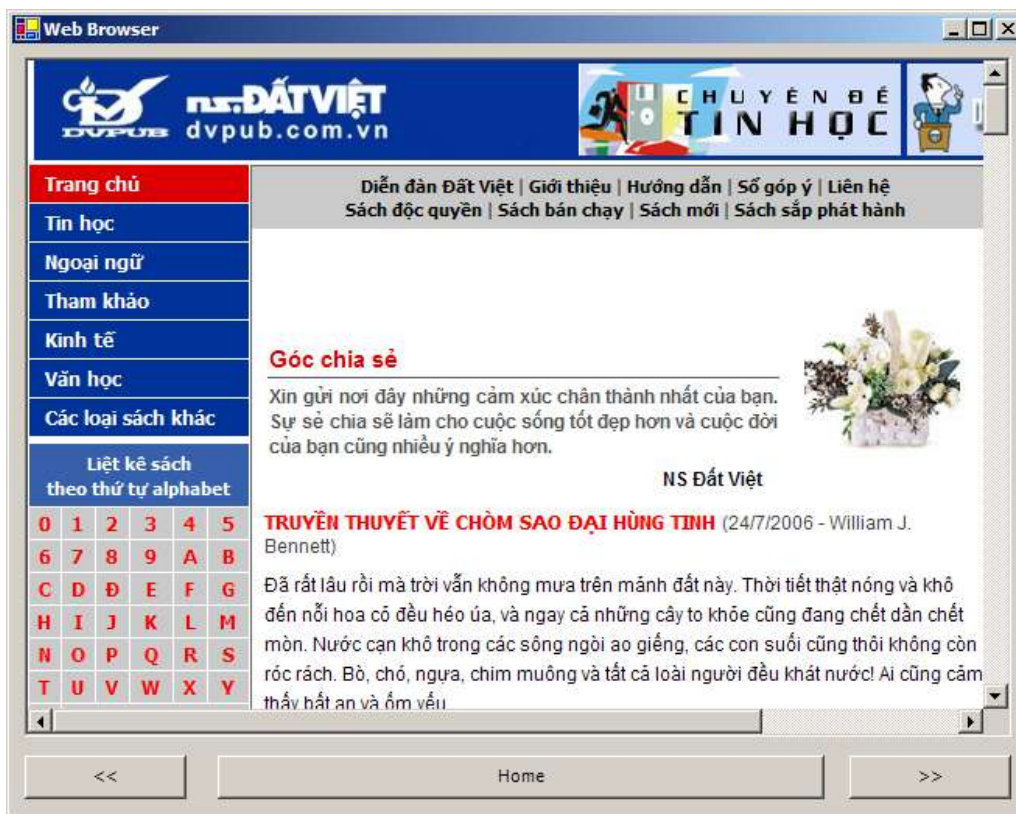
Hình 11.2 Chọn Microsoft Web Browser trong hộp thoại Customize Toolbox

Nếu không sử dụng *Visual Studio .NET*, bạn cần tạo ra một vỏ bọc bằng tiện ích *Type Library Importer (Tlbimp.exe)*, sẽ được giải thích trong mục 15.6.

Khi sử dụng điều khiển *Web Browser*, bạn thường cần các phương thức sau:

- *Navigate* (và *Navigate2*)— Chuyển trang sang *URL* do bạn chỉ định.
- *GoBack* và *GoForward*— Lái sang các trang trong danh sách history.
- *GoHome*— Lái sang trang mặc định (home) được thiết lập trên máy tính hiện hành, và *GoSearch*— Hiển thị trang tìm kiếm.

Người dùng có thể vào một trang mới bằng cách nhấp vào một liên kết nào đó trong trang hiện hành (nếu nó tồn tại). Bạn có thể lấy *URL* hiện hành từ thuộc tính *LocationURL* và xác định điều kiểm vẫn còn xử lý trang đó hay không bằng cách kiểm tra thuộc tính *Busy*. Ngoài ra, bạn có thể phản ứng với nhiều sự kiện khác nhau.



Hình 11.3 Sử dụng điều khiển Web Browser

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class WebBrowser : System.Windows.Forms.Form {

    private AxSHDocVw.AxWebBrowser explorer;
    private System.Windows.Forms.Button cmdBack;
    private System.Windows.Forms.Button cmdHome;
    private System.Windows.Forms.Button cmdForward;

    // (Bỏ qua phần mã designer.)
```

```

private void WebBrowser_Load(object sender, System.EventArgs e) {

    object nullObject = null;
    object uri = "http://www.dvpub.com.vn";
    explorer.Navigate2(ref uri, ref nullObject, ref nullObject,
        ref nullObject, ref nullObject);
}

private void cmdHome_Click(object sender, System.EventArgs e) {

    explorer.GoHome();
}

private void cmdForward_Click(object sender, System.EventArgs e) {

    try {
        explorer.GoForward();
    } catch {
        MessageBox.Show("Already on last page.");
    }
}

private void cmdBack_Click(object sender, System.EventArgs e) {

    try {
        explorer.GoBack();
    } catch {
        MessageBox.Show("Already on first page.");
    }
}
}

```

Hầu hết các phương thức của điều khiển *Web Browser* đều yêu cầu một vài thông số. Vì các phương thức này không được nạp chồng, và vì *C#* không hỗ trợ tham số tùy chọn, nên bạn phải cung cấp giá trị cho mọi tham số. Bạn không thể sử dụng tham chiếu `null` được, vì chúng là các thông số `ref`. Thay vào đó, bạn hãy tạo ra một biến đối tượng chứa tham chiếu `null`, sau đó cung cấp nó cho các thông số mà bạn không cần sử dụng. Kỹ thuật này sẽ được mô tả chi tiết trong mục 15.8.

5.

Lấy địa chỉ IP của máy tính hiện hành

Bạn cần lấy địa chỉ IP của máy tính hiện hành, có thể là để sử dụng sau này trong mã lệnh networking.



Sử dụng phương thức `GetHostName` và `GetHostByName` của lớp `System.Net.Dns`.

Lớp `Dns` cung cấp các dịch vụ phân giải tên miền. Bạn có thể gọi phương thức `GetHostName` để lấy về tên host của máy tính hiện hành. Sau đó, bạn có thể dịch tên này sang địa chỉ IP bằng phương thức `GetHostByName`. Đây là một ví dụ:

```
using System;
using System.Net;

public class GetIPAddress {

    private static void Main() {

        // Lấy tên host của máy tính hiện hành.
        string hostName = Dns.GetHostName();

        // Lấy địa chỉ IP trùng khớp đầu tiên.
        string ipAddress =
            Dns.GetHostByName(hostName).AddressList[0].ToString();

        Console.WriteLine("Host name: " + hostName);
        Console.WriteLine("IP address: " + ipAddress);

        Console.ReadLine();
    }
}
```

Phương thức `GetHostByName` trả về danh sách các địa chỉ IP có hiệu lực. Trong rất nhiều trường hợp, danh sách này chỉ có một phần tử.



Khi sử dụng địa chỉ IP trong giao tiếp mạng, bạn có thể sử dụng địa chỉ **127.0.0.1 để chỉ đến máy tính hiện hành thay cho địa chỉ IP thực tế của nó.**

6.

Phân giải tên miền thành địa chỉ IP

- ?** Bạn muốn xác định địa chỉ *IP* của một máy tính dựa vào tên miền của nó bằng cách thực hiện một truy vấn *Domain Name System (DNS)*.
- X** Sử dụng phương thức `GetHostByName` của lớp `System.Net.Dns` với đối số là tên miền.

Trên web, các địa chỉ *IP* có thể truy xuất công khai thường được ánh xạ đến tên miền để dễ nhớ hơn. Ví dụ, địa chỉ *207.171.185.16* được ánh xạ đến tên miền *www.amazon.com*. Để xác định địa chỉ *IP* khi có tên miền, máy tính cần liên lạc với một *DNS-server*.

Quá trình phân giải tên miền được thực hiện một cách trong suốt khi bạn sử dụng lớp `System.Net.Dns`. Lớp này cho phép lấy địa chỉ *IP* của một tên miền bằng phương thức `GetHostByName`. Dưới đây là đoạn mã trình bày cách lấy danh sách các địa chỉ *IP* được ánh xạ đến tên miền *www.microsoft.com*.

```
using System;
using System.Net;

public class ResolveIP {

    private static void Main() {

        foreach (IPAddress ip in
            Dns.GetHostByName("www.microsoft.com").AddressList) {

            Console.WriteLine(ip.AddressFamily.ToString() + ": ");
            Console.WriteLine(ip.ToString());
        }

        Console.ReadLine();
    }
}
```

Khi chạy đoạn mã trên, bạn sẽ thấy kết xuất như sau:

```
InterNetwork: 207.46.249.222
InterNetwork: 207.46.134.222
InterNetwork: 207.46.249.27
InterNetwork: 207.46.134.155
InterNetwork: 207.46.249.190
```

7.

“Ping” một địa chỉ IP

- ?** Bạn muốn kiểm tra một máy tính có online hay không và đo thời gian đáp ứng (*response time*) của nó.
- X** Gửi một thông điệp “ping”. Thông điệp này được gửi bằng giao thức *Internet Control Message Protocol (ICMP)* với một raw-socket.

Một thông điệp “ping” giao tiếp với một thiết bị tại một địa chỉ *IP* cụ thể, gửi một thông điệp thử nghiệm, và yêu cầu thiết bị này đáp ứng lại. Để đo thời gian kết nối giữa hai máy tính, bạn có thể đo thời gian cho một đáp ứng.

Mặc dù thông điệp “ping” đơn giản hơn các kiểu giao tiếp khác, nhưng hiện thực một tiện ích “ping” trong *.NET* đòi hỏi một lượng lớn mã lệnh networking mức-thấp và phức tạp. Thư viện lớp *.NET* không có sẵn giải pháp nào—thay vào đó, bạn phải sử dụng raw-socket và một số mã lệnh cực kỳ dài.

Tuy nhiên, đã có ít nhất một nhà phát triển giải quyết được vấn đề “ping”. Dưới đây là mã lệnh do *Lance Olson*, một nhà phát triển của *Microsoft*, cung cấp. Mã lệnh này cho phép “ping” một host bằng tên hay địa chỉ *IP* và đo lượng mili-giây cho một đáp ứng.

```
using System;
using System.Net;
using System.Net.Sockets;

public class Pinger {

    public static int GetPingTime(string host) {

        int dwStart = 0, dwStop = 0;

        // Tạo một raw-socket.
        Socket socket = new Socket(AddressFamily.InterNetwork,
            SocketType.Raw, ProtocolType.Icmp);

        // Lấy IPEndPoint của server, và chuyển nó thành EndPoint.
        IPEndPoint serverHE = Dns.GetHostByName(host);
        IPEndPoint ipepServer =
            new IPEndPoint(serverHE.AddressList[0], 0);
        EndPoint epServer = (ipepServer);
```



```
// Thiết lập endpoint cho máy client.
IPHostEntry fromHE = Dns.GetHostByName(Dns.GetHostName());
EndPoint ipEndPointFrom =
    new IPEndPoint(fromHE.AddressList[0], 0);
EndPoint EndPointFrom = (ipEndPointFrom);

// Tạo packet.
int PacketSize = 0;
IcmpPacket packet = new IcmpPacket();
for (int j = 0; j < 1; j++) {
    packet.Type = ICMP_ECHO;
    packet.SubCode = 0;
    packet.CheckSum = UInt16.Parse("0");
    packet.Identifier = UInt16.Parse("45");
    packet.SequenceNumber = UInt16.Parse("0");

    int PingData = 32;
    packet.Data = new Byte[PingData];

    for (int i = 0; i < PingData; i++)
        packet.Data[i] = (byte) '#';

    PacketSize = PingData + 8;

    Byte [] icmp_pkt_buffer = new Byte [PacketSize];
    int index = 0;
    index = Serialize(packet,
        icmp_pkt_buffer, PacketSize, PingData);

    // Tính checksum cho packet.
    double double_length = Convert.ToDouble(index);
    double dtemp = Math.Ceiling(double_length / 2);
    int cksum_buffer_length = Convert.ToInt32(dtemp);
    UInt16[] cksum_buffer = new UInt16[cksum_buffer_length];
    int icmp_header_buffer_index = 0;

    for (int i = 0; i < cksum_buffer_length; i++) {
```

```

        cksum_buffer[i] = BitConverter.ToUInt16(icmp_pkt_buffer,
            icmp_header_buffer_index);
        icmp_header_buffer_index += 2;
    }

    UInt16 u_cksum = checksum(cksum_buffer, cksum_buffer_length);
    packet.CheckSum = u_cksum;

    // Tuần tự hóa packet.
    byte[] sendbuf = new byte[PacketSize];
    index = Serialize(packet, sendbuf, PacketSize, PingData);

    // Bắt đầu tính giờ.
    dwStart = System.Environment.TickCount;
    socket.SendTo(sendbuf, PacketSize, 0, epServer);

    // Nhận đáp ứng, và ngừng tính giờ.
    byte[] ReceiveBuffer = new byte[256];
    socket.ReceiveFrom(ReceiveBuffer, 256, 0, ref EndPointFrom);
    dwStop = System.Environment.TickCount - dwStart;
}

// Dọn dẹp và trả về thời gian "ping" (tính theo giây).
socket.Close();
return (int)dwStop;
}

private static int Serialize(IcmpPacket packet, byte[] buffer,
    int packetSize, int pingData) {

    // (Bỏ qua phương thức private dùng để tuần tự hóa packet.)
}

private static UInt16 checksum(UInt16[] buffer, int size) {

```

```

        // (Bỏ qua phương thức private dùng để tính checksum.)
    }
}

```

```

public class IcmpPacket {
    public byte Type;
    public byte SubCode;
    public UInt16 CheckSum;
    public UInt16 Identifier;
    public UInt16 SequenceNumber;
    public byte[] Data;
}

```

Bạn có thể sử dụng phương thức tĩnh `Pinger.GetPingTime` với một địa chỉ *IP* hay một tên miền. Phương thức `GetPingTime` trả về lượng mili-giây trôi qua trước khi một đáp ứng được tiếp nhận. Dưới đây là đoạn mã thử nghiệm trên ba website:

```

public class PingTest {

    private static void Main() {

        Console.WriteLine("Milliseconds to contact www.yahoo.com:" +
            Pinger.GetPingTime("www.yahoo.com").ToString());
        Console.WriteLine("Milliseconds to contact www.seti.org:" +
            Pinger.GetPingTime("www.seti.org").ToString());
        Console.WriteLine("Milliseconds to contact the local computer:" +
            Pinger.GetPingTime("127.0.0.1").ToString());

        Console.ReadLine();
    }
}

```

Thử nghiệm “ping” cho phép bạn xác minh các máy tính khác có online hay không. Nó cũng có thể hữu ích khi ứng dụng của bạn cần đánh giá những máy tính khác nhau (ở xa) nhưng cho cùng nội dung để xác định máy nào có thời gian giao tiếp mạng thấp nhất.



Một yêu cầu “ping” có thể không thành công nếu bị firewall ngăn lại. Ví dụ, nhiều site bỏ qua yêu cầu “ping” vì sợ bị sa vào một luồng “ping” cùng một lúc sẽ làm cản trở server (thực chất là một tấn công từ chối dịch vụ).

8.

Giao tiếp bằng TCP



Bạn cần gửi dữ liệu giữa hai máy tính trên một network bằng kết nối *TCP/IP*.



Một máy tính (server) phải lắng nghe bằng lớp `System.Net.Sockets.TcpListener`. Mỗi khi một kết nối được thiết lập, cả hai máy tính đều có thể giao tiếp bằng lớp `System.Net.Sockets.TcpListener`.

TCP là một giao thức đáng tin cậy dựa-trên-kết-nối, cho phép hai máy tính giao tiếp thông qua một network. Để tạo một kết nối *TCP*, một máy tính phải đóng vai trò là server và bắt đầu lắng nghe trên một endpoint cụ thể (endpoint được định nghĩa là một địa chỉ *IP*, cho biết máy tính và số port). Một máy tính khác phải đóng vai trò là client và gửi một yêu cầu kết nối đến endpoint mà máy tính thứ nhất đang lắng nghe trên đó. Một khi kết nối được thiết lập, hai máy tính có thể trao đổi các thông điệp với nhau. Cả hai máy tính chỉ đơn giản đọc/ghi từ một `System.Net.Sockets.NetworkStream`.



Mặc dù một kết nối *TCP* luôn cần có một server và một client, nhưng không lý do gì một ứng dụng không thể là cả hai. Ví dụ, trong một ứng dụng *peer-to-peer*, một tiểu trình được sử dụng lắng nghe các yêu cầu đến (đóng vai trò là một server) trong khi một tiểu trình khác được sử dụng để khởi tạo các kết nối đi (đóng vai trò là một client). Trong ví dụ đi kèm mục này, client và server là các ứng dụng riêng rẽ và được đặt trong các thư mục con riêng.

Một khi kết nối *TCP* được thiết lập, hai máy tính có thể gửi bất kỳ kiểu dữ liệu nào bằng cách ghi dữ liệu đó ra `NetworkStream`. Tuy nhiên, ý tưởng hay là bắt đầu thiết kế một ứng dụng mạng bằng cách định nghĩa giao thức mức-ứng-dụng mà client và server sẽ sử dụng để giao tiếp. Giao thức này chứa các hằng mô tả các lệnh được phép, bảo đảm mã lệnh của ứng dụng không chứa các chuỗi giao tiếp được viết cứng.

```
namespace SharedComponent {
```

```
    public class ServerMessages {
        public const string AcknowledgeOK = "OK";
        public const string AcknowledgeCancel = "Cancel";
        public const string Disconnect = "Bye";
    }
```

```
    public class ClientMessages {
        public const string RequestConnect = "Hello";
        public const string Disconnect = "Bye";
    }
```

```
}
```

Trong ví dụ này, bảng từ vựng được định nghĩa sẵn chỉ là cơ bản. Bạn có thể thêm nhiều hằng hơn nữa tùy thuộc vào kiểu ứng dụng. Ví dụ, trong một ứng dụng truyền file, client có thể gửi một thông điệp để yêu cầu một file. Sau đó, server có thể đáp lại bằng một acknowledgment (*ACK*) và trả về các chi tiết của file (kích thước file chẳng hạn). Những hằng này sẽ được biên dịch thành một *Class Library Assembly* riêng, và cả client và server đều phải tham chiếu đến assembly này.

Đoạn mã dưới đây là một khuôn dạng cho một *TCP*-server cơ bản. Nó lắng nghe trên một port cố định, nhận kết nối đến đầu tiên và rồi đợi client yêu cầu ngừng kết nối. Tại thời điểm này, server có thể gọi phương thức `TcpListener.AcceptTcpClient` lần nữa để đợi client kế tiếp. Nhưng thay vào đó, nó sẽ đóng lại.

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
using SharedComponent;

public class TcpServerTest {

    private static void Main() {

        // Tạo listener trên port 8000.
        TcpListener listener =
            new TcpListener(IPAddress.Parse("127.0.0.1"), 8000);

        Console.WriteLine("About to initialize port.");
        listener.Start();
        Console.WriteLine("Listening for a connection...");

        try {

            // Đợi yêu cầu kết nối, và trả về TcpClient.
            TcpClient client = listener.AcceptTcpClient();
            Console.WriteLine("Connection accepted.");

            // Thu lấy network stream.
            NetworkStream stream = client.GetStream();

            // Tạo BinaryWriter để ghi ra stream.
            BinaryWriter w = new BinaryWriter(stream);
```

```

        // Tạo BinaryReader để đọc từ stream.
        BinaryReader r = new BinaryReader(stream);

        if (r.ReadString() == ClientMessages.RequestConnect) {

            w.Write(ServerMessages.AcknowledgeOK);
            Console.WriteLine("Connection completed.");

            while (r.ReadString() != ClientMessages.Disconnect)
            {}

            Console.WriteLine();
            Console.WriteLine("Disconnect request received.");
            w.Write(ServerMessages.Disconnect);
        } else {
            Console.WriteLine("Could not complete connection.");
        }

        // Đóng socket.
        client.Close();
        Console.WriteLine("Connection closed.");

        // Đóng socket nằm dưới (ngừng lắng nghe yêu cầu mới).
        listener.Stop();
        Console.WriteLine("Listener stopped.");
    } catch (Exception err) {
        Console.WriteLine(err.ToString());
    }

    Console.ReadLine();
}
}

```

Đoạn mã dưới đây là một khuôn dạng cho một *TCP*-client cơ bản. Nó tiếp xúc với server tại địa chỉ *IP* và port được chỉ định. Trong ví dụ này, địa chỉ loopback (*127.0.0.1*—chỉ đến máy tính hiện hành) được sử dụng. Nhớ rằng kết nối *TCP* yêu cầu hai port: một tại server và một

tại client. Tuy nhiên, chỉ cần chỉ định port tại server, còn port tại client có thể được chọn động lúc thực thi từ các port có sẵn.

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
using SharedComponent;

public class TcpClientTest {

    private static void Main() {

        TcpClient client = new TcpClient();

        try {

            Console.WriteLine("Attempting to connect to the server " +
                "on port 8000.");
            client.Connect(IPAddress.Parse("127.0.0.1"), 8000);
            Console.WriteLine("Connection established.");

            // Thu lấy network stream.
            NetworkStream stream = client.GetStream();

            // Tạo BinaryWriter để ghi ra stream.
            BinaryWriter w = new BinaryWriter(stream);

            // Tạo BinaryReader để đọc từ stream.
            BinaryReader r = new BinaryReader(stream);

            w.Write(ClientMessages.RequestConnect);

            if (r.ReadString() == ServerMessages.AcknowledgeOK) {

                Console.WriteLine("Connected.");
                Console.WriteLine("Press Enter to disconnect.");
                Console.ReadLine();
                Console.WriteLine("Disconnecting...");
```

```
        w.Write(ClientMessages.Disconnect);
    } else {
        Console.WriteLine("Connection not completed.");
    }

    // Đóng connection socket.
    client.Close();
    Console.WriteLine("Port closed.");
} catch (Exception err) {
    Console.WriteLine(err.ToString());
}

Console.ReadLine();
}
```

Dưới đây là transcript phía server:

```
About to initialize port.
Listening for a connection...
Connection accepted.
Connection completed.

Disconnect request received.
Connection closed.
Listener stopped.
```

Và dưới đây là transcript phía client:

```
Attempting to connect to the server on port 8000.
Connection established.
Connected.
Press Enter to disconnect.

Disconnecting...
Port closed.
```


9. *Lấy địa chỉ IP của client từ kết nối socket*

? Ứng dụng server cần xác định địa chỉ *IP* của client sau khi nó chấp nhận một kết nối.

✕ Sử dụng phương thức `AcceptSocket` của lớp `TcpListener` để lấy lớp mức-thấp là `System.Net.Sockets.Socket` thay vì là `TcpClient`. Sử dụng thuộc tính `Socket.RemoteEndPoint` để lấy địa chỉ *IP* của client.

Lớp `TcpClient` không cho phép bạn thu lấy socket nằm dưới hay bất cứ thông tin nào về port và địa chỉ *IP* của client. Lớp này có cung cấp thuộc tính `Socket`, nhưng thuộc tính này là được-bảo-vệ (protected) và do đó không thể truy xuất được từ các lớp phi dẫn xuất. Để truy xuất socket nằm dưới, bạn có hai tùy chọn:

- Tạo một lớp tùy biến dẫn xuất từ `TcpClient`. Lớp này có thể truy xuất thuộc tính được-bảo-vệ `Socket` và trưng nó ra thông qua một thuộc tính mới. Sau đó, bạn phải sử dụng lớp tùy biến này thay cho `TcpClient`.
- Bỏ qua lớp `TcpClient` bằng cách sử dụng phương thức `TcpListener.AcceptSocket`. Bạn vẫn có thể sử dụng các lớp mức-cao là `BinaryReader` và `BinaryWriter` để đọc/ghi dữ liệu, nhưng bạn cần phải tạo `NetworkStream` trước (sử dụng socket).

Mục này sử dụng cách thứ hai. Dưới đây là phiên bản sửa đổi của server trong mục 11.8:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
using SharedComponent;

public class TcpServerTest {

    private static void Main() {

        // Tạo listener trên port 8000.
        TcpListener listener =
            new TcpListener(IPAddress.Parse("127.0.0.1"), 8000);

        Console.WriteLine("About to initialize port.");
        listener.Start();
        Console.WriteLine("Listening for a connection...");

        try {
```

```

// Đợi yêu cầu kết nối, và trả về một Socket.
Socket socket = listener.AcceptSocket();

Console.WriteLine("Connection accepted.");

// Tạo network stream.
NetworkStream stream = new NetworkStream(socket);

// Tạo BinaryWriter để ghi ra stream.
BinaryWriter w = new BinaryWriter(stream);

// Tạo BinaryReader để đọc từ stream.
BinaryReader r = new BinaryReader(stream);

if (r.ReadString() == ClientMessages.RequestConnect) {

    w.Write(ServerMessages.AcknowledgeOK);
    Console.WriteLine("Connection completed.");

    // Lấy địa chỉ IP của client.
    Console.WriteLine("The client is from IP address: " +
        ((IPEndPoint)socket.RemoteEndPoint).Address.ToString());
    Console.WriteLine("The client uses local port: " +
        ((IPEndPoint)socket.RemoteEndPoint).Port.ToString());

    while (r.ReadString() != ClientMessages.Disconnect)
    {}

    Console.WriteLine();
    Console.WriteLine("Disconnect request received.");
    w.Write(ServerMessages.Disconnect);
} else {
    Console.WriteLine("Could not complete connection.");
}

// Đóng socket.
socket.Close();

```

```

        Console.WriteLine("Connection closed.");

        // Đóng socket nằm dưới (ngừng lắng nghe yêu cầu mới).
        listener.Stop();

        Console.WriteLine("Listener stopped.");
    } catch (Exception err) {
        Console.WriteLine(err.ToString());
    }

    Console.ReadLine();
}
}

```

10.

Thiết lập các tùy chọn socket

? Bạn cần thiết lập các tùy chọn socket mức-thấp, chẳng hạn các tùy chọn cho biết *send timeout* và *receive timeout*.

✕ Sử dụng phương thức `Socket.SetSocketOption`. Bạn có thể thiết lập các thuộc tính của socket được sử dụng để lắng nghe các yêu cầu hoặc các thuộc tính của socket được sử dụng cho một phiên client cụ thể.

Bạn có thể sử dụng phương thức `Socket.SetSocketOption` để thiết lập một số thuộc tính socket mức-thấp. Khi gọi phương thức này, bạn cần cung cấp ba đối số sau đây:

- Một giá trị thuộc kiểu liệt kê `SocketOptionLevel`, cho biết kiểu socket mà thiết lập này sẽ áp dụng cho nó (bao gồm IP, IPv6, Socket, Tcp, Udp).
- Một giá trị thuộc kiểu liệt kê `SocketOptionName`, cho biết thiết lập socket mà bạn muốn thay đổi (xem danh sách các giá trị của `SocketOptionName` trong tài liệu *.NET Framework*).
- Một giá trị mô tả thiết lập mới. Giá trị này thường là một số nguyên, nhưng cũng có thể là một mảng byte hay một kiểu đối tượng.

Ví dụ dưới đây sẽ thiết lập send-timeout của socket:

```

// Thao tác gửi sẽ hết hiệu lực nếu không nhận được
// thông tin xác nhận trong vòng 1000 mili-giây.
socket.SetSocketOption(SocketOptionLevel.Socket,
    SocketOptionName.SendTimeout, 1000);

```

Chú ý rằng, để truy xuất socket mô tả một kết nối client/server, bạn phải sử dụng phương thức `TcpListener.AcceptSocket` thay cho phương thức `TcpListener.AcceptTcpClient` (đã được thảo luận trong mục 11.9).

Bạn cũng có thể thiết lập các tùy chọn cho socket được sử dụng bởi `TcpListener` để theo dõi các yêu cầu kết nối. Tuy nhiên, bạn phải thực hiện thêm một vài bước nữa. Lớp `TcpListener`

cung cấp thuộc tính `Socket`, nhưng khả năng truy xuất của nó là `protected`, nghĩa là bạn không thể truy xuất nó một cách trực tiếp. Thay vào đó, bạn phải dẫn xuất một lớp mới từ `TcpListener`:

```
public class CustomTcpListener : TcpListener {

    public Socket Socket {
        get {return base.Server;}
    }

    public CustomTcpListener(IPAddress ip, int port) : base(ip, port) {}
}
```

Bây giờ, bạn có thể sử dụng lớp này khi tạo một `TcpListener`. Ví dụ dưới đây sử dụng cách tiếp cận này để thiết lập một tùy chọn socket:

```
CustomTcpListener listener =
    new CustomTcpListener(IPAddress.Parse("127.0.0.1"), 8000);

listener.Socket.SetSocketOption(SocketOptionLevel.Socket,
    SocketOptionName.ReceiveTimeout, 1000);

// (Sử dụng CustomTcpListener giống như đã sử dụng TcpListener.)
```

11. Tạo một TCP-server hỗ-trợ-đa-tiểu-trình

- ? **Bạn muốn tạo một TCP-server có thể cùng lúc xử lý nhiều TCP-client.**
- ✖ **Sử dụng phương thức `AcceptTcpClient` của lớp `TcpListener`. Mỗi khi có một client mới kết nối đến, khởi chạy một tiểu trình mới để xử lý yêu cầu và gọi `TcpListener.AcceptTcpClient` lần nữa.**

Một endpoint *TCP* (địa chỉ *IP* và port) có thể phục vụ nhiều kết nối. Thực ra, hệ điều hành đảm đương phần lớn công việc giúp bạn. Những gì bạn cần làm là tạo một đối tượng thợ (*worker object*) trên server để xử lý mỗi kết nối trong một tiểu trình riêng.

Xét lớp *TCP-client* và *TCP-server* đã được trình bày trong mục 11.8. Bạn có thể dễ dàng chuyển server này thành một server hỗ-trợ-đa-tiểu-trình để thực hiện nhiều kết nối cùng một lúc. Trước hết, tạo một lớp để tương tác với một client:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
```

```
using System.Threading;
using SharedComponent;

public class ClientHandler {

    private TcpClient client;
    private string ID;

    public ClientHandler(TcpClient client, string ID) {

        this.client = client;
        this.ID = ID;
    }

    public void Start() {

        // Thu lấy network stream.
        NetworkStream stream = client.GetStream();

        // Tạo BinaryWriter để ghi ra stream.
        BinaryWriter w = new BinaryWriter(stream);

        // Tạo BinaryReader để đọc từ stream.
        BinaryReader r = new BinaryReader(stream);

        if (r.ReadString() == ClientMessages.RequestConnect) {

            w.Write(ServerMessages.AcknowledgeOK);
            Console.WriteLine(ID + ": Connection completed.");
            while (r.ReadString() != ClientMessages.Disconnect) {}

            Console.WriteLine(ID + ": Disconnect request received.");
            w.Write(ServerMessages.Disconnect);

        }else {

            Console.WriteLine(ID + ": Could not complete connection.");
        }
    }
}
```

```

        // Đóng socket.
        client.Close();

        Console.WriteLine(ID + ": Client connection closed.");

        Console.ReadLine();
    }
}

```

Kế tiếp, thay đổi mã lệnh của server sao cho nó lặp liên tục, tạo ra các thể hiện `ClientHandler` mới khi cần và chạy chúng trong các tiểu trình mới. Dưới đây là mã lệnh đã được sửa đổi:

```

public class TcpServerTest {

    private static void Main() {

        TcpListener listener =
            new TcpListener(IPAddress.Parse("127.0.0.1"), 8000);
        Console.WriteLine("Server: About to initialize port.");
        listener.Start();
        Console.WriteLine("Server: Listening for a connection...");

        int clientNum = 0;
        while (true) {

            try {

                // Đợi yêu cầu kết nối, và trả về một TcpClient.
                TcpClient client = listener.AcceptTcpClient();
                Console.WriteLine("Server: Connection accepted.");

                // Tạo một đối tượng mới để xử lý kết nối này.
                clientNum++;

                ClientHandler handler =
                    new ClientHandler(client, "Client " +
                        clientNum.ToString());

                // Khởi động đối tượng này làm việc trong

```

```
// một tiêu trình khác.
Thread handlerThread =
    new Thread(new ThreadStart(handler.Start));
handlerThread.IsBackground = true;
handlerThread.Start();

// (Bạn cũng có thể thêm Handler và HandlerThread vào
// một tập hợp để theo dõi các phiên client.)

} catch (Exception err) {
    Console.WriteLine(err.ToString());
}
}
}
```

Dưới đây là transcript phía server của một phiên làm việc với hai client:

```
Server: About to initialize port.
Server: Listening for a connection...
Server: Connection accepted.
Client 1: Connection completed.
Server: Connection accepted.
Client 2: Connection completed.
Client 2: Disconnect request received.
Client 2: Client connection closed.
Client 1: Disconnect request received.
Client 1: Client connection closed.
```

Bạn có thể thêm mã lệnh vào server để nó theo vết các đối tượng thợ hiện hành trong một tập hợp. Làm như thế sẽ cho phép server hủy bỏ các tác vụ này nếu nó cần phải đóng và chỉ được phép một số tối đa client cùng một lúc.

12.

SỬ DỤNG TCP một cách bất đồng bộ

?

Bạn cần ghi dữ liệu ra network-stream từng khối một, mà không phải block phần mã lệnh còn lại. Kỹ thuật này có thể được sử dụng nếu bạn muốn “stream” một file lớn trên mạng.

✕

Tạo một lớp riêng để xử lý kỹ thuật streaming bất đồng bộ. Bạn có thể bắt đầu “stream” một khối dữ liệu bằng phương thức `NetworkStream.BeginWrite` và cung cấp một phương thức callback. Khi callback được kích hoạt thì gửi khối kế tiếp.

Lớp `NetworkStream` hỗ trợ việc sử dụng bất đồng bộ thông qua phương thức `BeginRead` và `BeginWrite`. Sử dụng các phương thức này, bạn có thể gửi hay nhận một khối dữ liệu trên một trong các tiểu trình do thread-pool của bộ thực thi *.NET* cung cấp, mà không block mã lệnh của bạn. Mục này trình bày kỹ thuật ghi bất đồng bộ.

Khi gửi dữ liệu một cách bất đồng bộ, bạn phải gửi dữ liệu nhị phân thô (một mảng byte). Và bạn cần chọn kích thước mỗi lần gửi hay nhận. Ví dụ dưới đây viết lại server từ mục 11.11 sao cho mỗi lớp `ClientHandler` gửi một lượng lớn dữ liệu được đọc từ một file. Dữ liệu này được gửi một cách bất đồng bộ, nghĩa là `ClientHandler` có thể tiếp tục thực hiện các tác vụ khác (trong ví dụ này, nó chỉ việc lấy các thông điệp được gửi từ client).

Một thuận lợi của cách tiếp cận này là toàn bộ nội dung của file chẳng bao giờ nằm trong bộ nhớ một lượt. Thay vào đó, nó được thu lấy ngay trước khi một khối mới được gửi. Một thuận lợi khác nữa là server có thể hủy bỏ thao tác vào bất cứ lúc nào. Ví dụ, nếu client chỉ đọc đến khối dữ liệu thứ ba thì ngắt kết nối, server sẽ thiết lập một biến thành viên luận lý có tên là `fileStop` để báo cho callback không gửi dữ liệu nữa.

Dưới đây là lớp `ClientHandler` đã được sửa đổi (lớp `TcpServerTest` không cần thay đổi gì):

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
using SharedComponent;

public class ClientHandler {

    private TcpClient client;
    private string ID;

    // Kích thước một khối dữ liệu (2 KB).
    private int bufferSize = 2048;
    // Bộ đệm dùng để chứa dữ liệu.
    private byte[] buffer;

    // Dùng để đọc dữ liệu từ một file.
    private FileStream fileStream;
    // Dùng để giao tiếp với client.
    private NetworkStream networkStream;

    // Dấu hiệu ngừng gửi dữ liệu.
    private bool fileStop = false;
```



```
public ClientHandler(TcpClient client, string ID) {

    this.buffer = new byte[bufferSize];
    this.client = client;
    this.ID = ID;
}

public void Start() {

    // Thu lấy network stream.
    networkStream = client.GetStream();

    // Tạo các đối tượng dùng để gửi và nhận text.
    BinaryWriter w = new BinaryWriter(networkStream);
    BinaryReader r = new BinaryReader(networkStream);

    if (r.ReadString() == ClientMessages.RequestConnect) {

        w.Write(ServerMessages.AcknowledgeOK);
        Console.WriteLine(ID + ": Connection completed.");

        string message = "";
        while (message != ClientMessages.Disconnect) {

            message = r.ReadString();
            if (message == ClientMessages.RequestData) {

                // Tên file có thể do client cung cấp, nhưng
                // trong ví dụ này, file thử nghiệm là mã cứng.
                FileStream =
                    new FileStream("test.bin", FileMode.Open);

                // Gửi kích thước file.
                w.Write(fileStream.Length.ToString());

                // Khởi chạy thao tác bất đồng bộ.
                StreamData(null);
            }
        }
    }
}
```

```

    }
    fileStop = true;
    Console.WriteLine(ID + ": Disconnect request received.");
} else {
    Console.WriteLine(ID + ": Could not complete connection.");
}

// Đóng kết nối.
client.Close();
Console.WriteLine(ID + ": Client connection closed.");
Console.ReadLine();
}

private void StreamData(IAsyncResult asyncResult) {

    // Hủy bỏ nếu client ngừng kết nối.
    if (fileStop == true) {
        fileStop = false;
        return;
    }

    if (asyncResult != null) {
        // Một khối đã được ghi một cách bất đồng bộ.
        networkStream.EndWrite(asyncResult);
    }

    // Lấy khối kế tiếp từ file.
    int bytesRead = fileStream.Read(buffer, 0, buffer.Length);

    // Nếu không đọc được byte nào, stream đã đến cuối file.
    if (bytesRead > 0) {

        Console.WriteLine("Streaming new block.");

        // Ghi khối kế tiếp ra network stream.
        networkStream.BeginWrite(buffer, 0, buffer.Length,

```

```

        new AsyncCallback(StreamData), null);
    } else {

        // Kết thúc thao tác.
        Console.WriteLine("File streaming complete.");
        fileStream.Close();
    }
}
}

```

Bạn có thể sử dụng một mẫu tương tự để đọc dữ liệu một cách bất đồng bộ phía client.

13.

Giao tiếp bằng UDP



Bạn cần gửi dữ liệu giữa hai máy tính trên một network bằng *User Datagram Protocol (UDP) stream*.



Sử dụng lớp `System.Net.Sockets.UdpClient`, và sử dụng hai tiểu trình: một để gửi dữ liệu và một để nhận dữ liệu.

UDP là một giao thức phi kết nối, không có bất kỳ điều khiển dòng chảy hay kiểm tra lỗi nào. Khác với *TCP*, *UDP* sẽ không được sử dụng ở những nơi cần đến giao tiếp đáng tin cậy. Tuy nhiên, vì chi phí thấp hơn, *UDP* thường được sử dụng cho các ứng dụng "chatty", tại đó chấp nhận mất một vài thông điệp. Ví dụ, giả sử bạn muốn tạo một network mà trong đó, các client gửi thông tin về nhiệt độ hiện thời tại vị trí của chúng đến một server mỗi vài phút. Bạn có thể sử dụng *UDP* trong trường hợp này vì tần số giao tiếp cao và thiệt hại do mất packet là không đáng kể (vì server có thể tiếp tục sử dụng nhiệt độ nhận được cuối cùng).

Ứng dụng dưới đây sử dụng hai tiểu trình: một để nhận thông điệp và một để gửi thông điệp. Để thử nghiệm ứng dụng này, hãy nạp hai thẻ hiện cùng một lúc. Trên máy tính *A*, cho biết địa chỉ *IP* của máy tính *B*. Trên máy tính *B*, cho biết địa chỉ *IP* của máy tính *A*. Theo đó, bạn có thể gửi qua lại thông điệp dạng text (bạn có thể mô phỏng thử nghiệm này trên một máy đơn bằng cách sử dụng hai port khác nhau và địa chỉ loopback).

```

using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;

public class UdpTest {

    private static int localPort;

    private static void Main() {

```

```
// Định nghĩa endpoint (thông điệp được gửi tại đây).
Console.Write("Connect to IP: ");
string IP = Console.ReadLine();
Console.Write("Connect to port: ");
int port = Int32.Parse(Console.ReadLine());

IPEndPoint remoteEndPoint = new IPEndPoint(IPAddress.Parse(IP),
    port);

// Định nghĩa endpoint cục bộ (thông điệp được nhận tại đây).
Console.Write("Local port for listening: ");
localPort = Int32.Parse(Console.ReadLine());

Console.WriteLine();

// Tạo một tiểu trình mới để nhận thông điệp đến.
Thread receiveThread = new Thread(
    new ThreadStart(ReceiveData));
receiveThread.IsBackground = true;
receiveThread.Start();

UdpClient client = new UdpClient();

try {
    string text;
    do {
        text = Console.ReadLine();

        if (text != "") {

            // Mã hóa dữ liệu thành dạng nhị phân
            // bằng phép mã hóa UTF8.
            byte[] data = Encoding.UTF8.GetBytes(text);

            // Gửi text đến client ở xa.
            client.Send(data, data.Length, remoteEndPoint);
        }
    } while (text != null);
}
```

```

        }
        } while (text != "");
    } catch (Exception err) {
        Console.WriteLine(err.ToString());
    }

    Console.ReadLine();
}

private static void ReceiveData() {

    UdpClient client = new UdpClient(localPort);
    while (true) {

        try {
            // Nhận dữ liệu (byte).
            IPEndPoint anyIP = new IPEndPoint(IPAddress.Any, 0);
            byte[] data = client.Receive(ref anyIP);

            // Chuyển byte thành text bằng phép mã hóa UTF8.
            string text = Encoding.UTF8.GetString(data);

            // Hiển thị text thu được.
            Console.WriteLine(">> " + text);
        } catch (Exception err) {
            Console.WriteLine(err.ToString());
        }
    }
}
}

```

Chú ý rằng, các ứng dụng *UDP* không thể sử dụng *NetworkStream* như các ứng dụng *TCP*. Thay vào đó, chúng phải chuyển tất cả dữ liệu thành một stream bằng một lớp mã hóa, như đã được mô tả trong mục 2.2.

Bạn có thể thử nghiệm ứng dụng này với các client trên máy cục bộ bằng cách sử dụng hai port khác nhau và địa chỉ loopback. Ví dụ, giả sử có hai *UDP*-client: client *A* và client *B*. Dưới đây là transcript đối với client *A*:

```

Connect to IP: 127.0.0.1
Connect to port: 8001
Local port for listening: 8080

```

Hi there!

Và đây là transcript tương ứng đối với client *B* (cùng với thông điệp nhận được):

Connect to IP: 127.0.0.1

Connect to port: 8080

Local port for listening: 8001

>> Hi there!

14.

Gửi e-mail thông qua SMTP



Bạn cần gửi e-mail đến một địa chỉ e-mail bằng một *SMTP-server* (*Simple Mail Transfer Protocol server*).



Sử dụng lớp `SmtpMail` và `MailMessage` thuộc không gian tên `System.Web.Mail`.

Các lớp trong không gian tên `System.Web.Mail` cung cấp một vỏ bọc cho thành phần *Collaboration Data Objects for Windows 2000 (CDO SYS)*. Chúng cho phép bạn soạn và gửi thông điệp e-mail bằng *SMTP*.

Dễ dàng sử dụng các kiểu này. Bạn chỉ cần tạo một đối tượng `MailMessage`, cho biết địa chỉ e-mail của người gửi và người nhận, và đặt nội dung của thông điệp trong thuộc tính `Body`.

```
MailMessage myMessage = new MailMessage();
myMessage.To = "someone@somewhere.com";
myMessage.From = "me@somewhere.com";
myMessage.Subject = "Hello";
myMessage.Priority = MailPriority.High;
myMessage.Body = "This is the message!";
```

Nếu muốn, bạn có thể gửi một thông điệp *HTML* bằng cách thay đổi định dạng của thông điệp và sử dụng các thẻ *HTML*.

```
myMessage.BodyFormat = MailFormat.Html;
myMessage.Body = @"<HTML><HEAD></HEAD>" +
    @"<BODY>This is the message!</BODY></HTML>";
```

Bạn có thể thêm file đính kèm bằng tập hợp `MailMessage.Attachments` và lớp `MailAttachment`.

```
MailAttachment myAttachment = new MailAttachment("c:\\mypic.gif");
myMessage.Attachments.Add(myAttachment);
```

Để gửi thông điệp, bạn chỉ cần cho biết tên của *SMTP-server* và gọi phương thức `SmtpMail.Send`.

```
SmtMail.SmtpServer = "test.mailserver.com";
```

```
SmtMail.Send(myMessage);
```

Tuy nhiên, có một vài vấn đề khi sử dụng lớp `SmtMail` để gửi một thông điệp e-mail. Lớp này cần một *SMTP-server* cục bộ hay một relay-server trên mạng. Ngoài ra, lớp `SmtMail` không hỗ trợ việc xác thực, do đó, nếu *SMTP-server* yêu cầu username và password, bạn sẽ không thể gửi bất kỳ mail nào. Để khắc phục vấn đề này, bạn có thể trực tiếp sử dụng thành phần *CDOYS* thông qua *COM Interop* (giả sử bạn có phiên bản server của *Windows* hay *Microsoft Exchange*).

 **Nhớ rằng, giao thức *SMTP* không được sử dụng để lấy e-mail. Đối với công việc này, bạn cần giao thức *POP3* hay *IMAP*, cả hai giao thức này đều không có trong *.NET Framework*.**

Để có thêm thông tin về cách sử dụng và cấu hình *SMTP-server*, bạn hãy tham khảo các quyển sách chuyên về *IIS*.

15.

Gửi và nhận e-mail với *MAPI*

? **Bạn muốn gửi một thông điệp e-mail, nhưng *SMTP-server* (*Simple Mail Transfer Protocol server*) chưa được cấu hình trên máy tính.**

✕ **Sử dụng *Simple MAPI* (*Messaging Application Programming Interface*) bằng cách nhập hàm cần thiết từ thư viện hệ thống không-được-quản-lý *Mapi32.dll*.**

MAPI là giao diện cho phép bạn tương tác với các tính năng mailing được tích hợp trong hệ điều hành *Windows*. Bạn có thể sử dụng *MAPI* (thông qua các hàm *API* không-được-quản-lý, hoặc thông qua thành phần *MAPI* đi cùng với *Visual Studio 6*) để tương tác với mail-client mặc định (thường là *Microsoft Outlook* hay *Outlook Express*). Các tác vụ bao gồm: lấy thông tin contact từ sổ địa chỉ, lấy thông điệp trong *Inbox*, soạn và gửi thông điệp. Đáng tiếc, không có lớp nào sử dụng *MAPI* trong *.NET Framework*. Tuy nhiên, bạn có thể sử dụng thư viện không-được-quản-lý *Mapi32.dll*.

Thách thức chính khi sử dụng *Simple MAPI* trong *.NET* là marshal các cấu trúc được sử dụng trong *.NET* thành các cấu trúc mà *Simple MAPI* cần, sau đó marshal các cấu trúc do *Simple MAPI* trả về cho ứng dụng *.NET*. Đây không phải là một công việc đơn giản. Tuy nhiên, *Microsoft* cung cấp một giải pháp toàn vẹn trong một thành phần *C#* tổng quát (có thể tải miễn phí). Bạn có thể sử dụng hai dự án dưới đây:

- Một thành phần thư viện lớp (*Class Library Component*) bọc lấy các hàm *Simple MAPI* và làm cho chúng có hiệu lực thông qua các phương thức của lớp.
- Một chương trình (thử nghiệm) sử dụng thành phần này để đăng nhập, đăng xuất, đọc mail, gửi mail...

Mã lệnh của cả hai dự án này không mấy phức tạp, nhưng rất dài nên không trình bày ở đây (bạn hãy xem trong đĩa CD đính kèm).

 **Đối với một ví dụ phức tạp hơn (xây dựng trên thư viện *Simple MAPI* của *Microsoft* để tạo một ứng dụng *Windows Form*), một dự án *C#* mẫu (có thể tải**

miễn phí) được *Thomas Scheidegger* cung cấp tại
[<http://www.codeproject.com/csharp/simplemapidotnet.asp>].

12

DỊCH VỤ WEB XML VÀ REMOTING

Microsoft .NET Framework hỗ trợ hai mô hình lập trình phân tán cấp cao là *Remoting* và dịch vụ *Web XML*. Mặc dù cả hai công nghệ này có nhiều điểm tương đồng (ví dụ, cả hai cùng trừu tượng hóa lời gọi giữa các tiến trình hay giữa các máy tính khác nhau thành lời gọi phương thức của các đối tượng ở xa), nhưng chúng cũng có vài điểm khác nhau cơ bản.

Dịch vụ *Web XML* được xây dựng bằng các chuẩn xuyên-nền và dựa vào khái niệm *XML messaging*. Dịch vụ *Web XML* được thực thi bởi bộ thực thi *ASP.NET*; nghĩa là chúng có được các tính năng của *ASP.NET* như output-caching. Điều này cũng có nghĩa là dịch vụ *Web XML* thuộc dạng phi trạng thái (*stateless*). Nói chung, dịch vụ *Web XML* thích hợp nhất khi bạn cần xuyên biên nền (ví dụ, một *Java-client* gọi một dịch vụ *Web ASP.NET*) hay biên tin cậy (ví dụ, trong các phiên giao dịch thương mại). Trong chương này, chúng ta sẽ bàn một số mục liên quan đến dịch vụ *Web XML* sau:

- Nâng cao tính linh hoạt của các lớp proxy bằng cách không viết mã cứng cho địa chỉ của dịch vụ *Web XML* (mục 12.1).
- Sử dụng kỹ thuật caching để nâng cao hiệu năng và khả năng phục vụ (tính quy mô) của dịch vụ *Web XML* (mục 12.2 và 12.3).
- Tạo phương thức giao dịch cho dịch vụ *Web XML* (mục 12.4).
- Truyền thông tin xác thực cho một dịch vụ *Web XML* bằng proxy (mục 12.5).
- Gọi bất đồng bộ một phương thức của dịch vụ *Web XML* (mục 12.6).

Remoting là một công nghệ đặc trưng của .NET dành cho các đối tượng phân tán và được xem như là một hậu duệ của công nghệ *DCOM*. Công nghệ này lý tưởng cho các hệ thống in-house⁴; trong đó, tất cả các ứng dụng đều được xây dựng trên nền .NET, chẳng hạn backbone của một hệ thống xử lý hóa đơn. *Remoting* cho phép các chuẩn giao tiếp khác nhau, chẳng hạn các thông điệp nhị phân nhỏ gọn và các kết nối *TCP/IP* hiệu quả hơn mà dịch vụ *Web XML* không hỗ trợ. Ngoài ra, *Remoting* còn là công nghệ duy nhất hỗ trợ các đối tượng có trạng thái, và giao tiếp hai chiều thông qua callback. Nó cũng là công nghệ duy nhất cho phép gọi các đối tượng .NET tùy biến qua mạng. Trong chương này, chúng ta sẽ bàn một số mục liên quan đến *Remoting* sau:

- Tạo các đối tượng khả-truy-xuất-từ-xa; đăng ký và quản lý chúng trên *IIS* (mục 12.7, 12.8, và 12.9).
- Phát sinh sự kiện trên các kênh truy xuất từ xa (mục 12.10).
- Kiểm soát thời gian sống và phiên bản của các đối tượng khả-truy-xuất-từ-xa (mục 12.11 và 12.12).
- Hiện thực các phương thức một chiều trong các đối tượng khả-truy-xuất-từ-xa (mục 12.13).

⁴ in-house system: Hệ thống công nghệ thông tin do phòng IT của một công ty xây dựng nên.



Chương này chỉ đưa ra một số kỹ thuật hữu dụng trong việc sử dụng dịch vụ *Web XML* và *Remoting*. Để hiểu sâu hơn, bạn cần tham khảo các sách chuyên về đề tài này.

1. Tránh viết mã cứng cho địa chỉ URL của dịch vụ Web XML



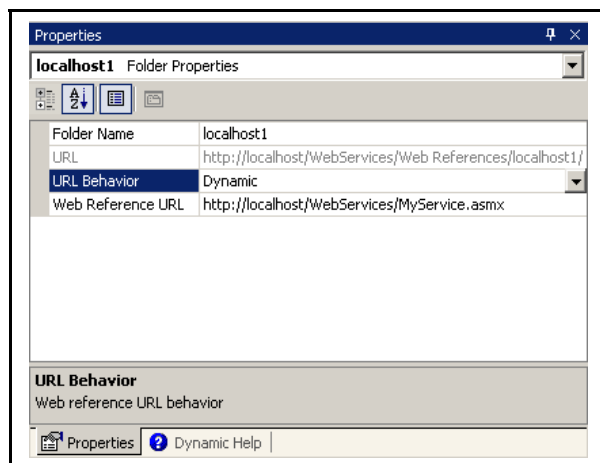
Bạn cần sử dụng một dịch vụ *Web XML* được đặt tại một địa chỉ URL mà địa chỉ này có thể thay đổi sau khi bạn triển khai ứng dụng client.



Sử dụng địa chỉ URL động cho dịch vụ *Web XML*. Khi đó, địa chỉ động này được lấy một cách tự động từ file cấu hình của ứng dụng client. Trong *Microsoft Visual Studio .NET*, bạn có thể cấu hình địa chỉ URL động bằng cách thay đổi tùy chọn *URL Behavior* của *Web Reference*. Bạn cũng có thể sử dụng công cụ *Web Services Description Language (Wsdl.exe)* với đối số `/appsettingurlkey`.

Theo mặc định, khi bạn tạo một lớp proxy thì địa chỉ URL của dịch vụ *Web XML* là mã cứng trong phương thức khởi dựng của lớp proxy này. Bạn có thể chép đề thiết lập này trong mã lệnh bằng cách điều chỉnh thuộc tính `url` của lớp proxy sau khi tạo một thể hiện của nó. Tuy nhiên, có một tùy chọn khác: cấu hình cho lớp proxy sử dụng một địa chỉ URL động.

Trong *Visual Studio .NET*, bạn có thể thực hiện điều này bằng cách chọn *Web Reference* trong cửa sổ *Solution Explorer* và thay đổi tùy chọn *URL Behavior* trong cửa sổ *Properties* (xem hình 12.1).



Hình 12.1 Cấu hình địa chỉ URL cho dịch vụ Web XML

Sau khi bạn thay đổi như trên, địa chỉ URL của dịch vụ *Web XML* sẽ tự động được thêm vào file cấu hình của ứng dụng client. File cấu hình này có tên là *Web.config* đối với các ứng dụng *Web*, và *[AppName].exe.config* đối với các ứng dụng khác (lưu ý, nguồn xuất hiện trong môi trường thiết kế chỉ là *App.config*, nhưng sau đó sẽ được *Visual Studio .NET* đổi tên một cách tự động). Ví dụ dưới đây là thiết lập được sinh tự động trong file cấu hình:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<configuration>

  <appSettings>
    <add key="AppName.ServerName.ServiceName"
      value="http://localhost/Webservices/MyService.asmx"/>
  </appSettings>

</configuration>
```

Bạn cũng có thể sử dụng địa chỉ URL động do công cụ *WSDL.exe* sinh ra. Trong trường hợp này, bạn sử dụng đối số `/appsettingurlkey` để cho biết tên của thiết lập cấu hình mà URL sẽ được lưu trữ ở đó. Bạn phải tạo file cấu hình bằng tay.

```
wsdl /out:Proxy.cs http://localhost/Webservices/MyService.asmx?WSDL
  /appsettingurlkey:MyService
```

Trong cả hai trường hợp, mã lệnh của lớp proxy sẽ được sửa đổi để đọc địa chỉ URL từ file cấu hình. Nếu không tìm thấy giá trị cần thiết, nó sẽ mặc định sử dụng địa chỉ URL được sử dụng trong quá trình phát triển ứng dụng. Cách tiếp cận này cho phép bạn đổi địa chỉ URL của dịch vụ Web XML sau khi đã biên dịch và triển khai ứng dụng (chỉ cần thay đổi file cấu hình).

2. Sử dụng kỹ thuật response-caching trong dịch vụ Web XML



Bạn muốn nâng cao hiệu năng của dịch vụ Web XML bằng cách lưu trữ giá trị trả về của một phương thức web.



Sử dụng response-caching bằng cách thiết lập thuộc tính `CacheDuration` của đặc tính `System.Web.Services.WebMethod`.

Trong *ASP.NET*, dịch vụ Web XML hỗ trợ response-caching giống hệt như các trang web *ASP.NET*. Khi response-caching được kích hoạt, mã lệnh của bạn chỉ thực hiện một lần, và giá trị trả về của phương thức web sẽ được lưu lại và được trả về trong các lần gọi tiếp theo. Đối với *Web Form*, caching được thực hiện cho từng form. Đối với dịch vụ Web XML, caching được kích hoạt và cấu hình riêng cho mỗi phương thức web.

Ví dụ, phương thức web dưới đây trả về giá trị ngày giờ hiện hành trên máy server. Thông tin này được lưu lại trong một phút, nghĩa là các lời yêu cầu tiếp theo trong khoảng thời gian này sẽ nhận lại thông tin đã được lưu trước đó.

```
Using System;
using System.Web.Services;

public class ResponseCaching {

    [WebMethod(CacheDuration=60)]
```

```
public string GetDate() {

    return DateTime.Now.ToString();

}
```

Nếu phương thức web của bạn nhận nhiều đối số, *ASP.NET* sẽ sử dụng lại giá trị đã được lưu chỉ khi các giá trị đối số cũng giống như thế. Nếu bạn có một phương thức nhận tầm giá trị rộng thì caching trở nên không hiệu quả và đôi khi trở nên lãng phí, vì có quá nhiều thông tin cần được lưu nhưng ít khi được sử dụng lại. Ví dụ, sử dụng caching trong một phương thức web thực hiện phép tính dựa trên đầu vào dạng số không phải là một sự lựa chọn tốt. Trái lại, sử dụng caching trong một phương thức web nhận giá trị *ID* của một nhóm nhỏ sản phẩm chắc chắn là một sự lựa chọn tốt. Bao giờ cũng vậy, response-caching bỏ qua mã lệnh của bạn, khiến nó trở nên không đủ tư cách nếu phương thức web của bạn cần thực hiện các hành động khác (chẳng hạn ghi nhật ký), hay phương thức web của bạn phụ thuộc vào một số thông tin khác ngoài các đối số được đưa vào (chẳng hạn thông tin xác thực của người dùng hay dữ liệu trong phiên làm việc).



Một hạn chế của response-caching là nó chỉ cho phép bạn sử dụng lại dữ liệu trong phạm vi của một phương thức web. Nếu bạn muốn sử dụng lại dữ liệu trong nhiều phương thức web khác nhau, kỹ thuật data-caching sẽ hiệu quả hơn.

3. *SỬ DỤNG KỸ THUẬT data-caching trong dịch vụ Web XML*



Bạn cần chạy phương thức web của bạn nhưng vẫn sử dụng một số thông tin đã được lưu. Hoặc, bạn muốn nâng cao hiệu năng của dịch vụ *Web XML* bằng cách lưu trữ một số dữ liệu, và cần sử dụng lại dữ liệu này trong một số phương thức.



Bạn có thể lưu bất cứ đối tượng nào vào cache bằng phương thức `Insert` của đối tượng `System.Web.Caching.Cache`. Bạn có thể truy xuất cache thông qua thuộc tính tĩnh `HttpContext.Current`.

Cách thức làm việc của data-caching với một dịch vụ *Web XML* cũng giống như với một trang web. Bạn có thể lưu trữ dữ liệu vào cache bằng mã trang web và lấy nó về trong một dịch vụ *Web XML*, hay ngược lại. Để có thêm thông tin về data-caching và các kiểu chính sách hết hiệu lực (*expiration policy*) mà nó hỗ trợ, bạn hãy tham khảo mục 7.15.

Điểm khác biệt duy nhất giữa caching trong một dịch vụ *Web XML* và caching trong một trang web là: Trong dịch vụ *Web XML*, bạn không thể thu lấy đối tượng `Cache` như một thuộc tính nội tại; thay vào đó, bạn cần truy xuất cache thông qua thuộc tính tĩnh `HttpContext.Current`.

Ví dụ dưới đây trình bày một dịch vụ *Web XML* với hai phương thức web:

- `GetProductCatalog`: Trả về một `DataSet` với các thông tin về sản phẩm. `DataSet` này có thể được lấy từ cache hoặc được sinh tự động (nếu cần) bằng hàm `GetCustomerDataSet`.
- `GetProductList`: Cũng sử dụng `DataSet` và hàm `GetCustomerDataSet` nhưng chỉ lấy một phần thông tin (tên sản phẩm).

Cả hai phương thức web trên đều có thể sử dụng chung dữ liệu đã được lưu, điều này làm giảm nhẹ gánh nặng đặt lên cơ sở dữ liệu.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.Services;
using System.Web;

public class DataCachingTest {

    private static string connectionString = "Data Source=localhost;" +
        "Initial Catalog=Northwind;user ID=sa";

    [WebMethod()]
    public DataSet GetProductCatalog() {

        // Trả về DataSet (từ cache, nếu có thể).
        return GetCustomerDataSet();
    }

    [WebMethod()]
    public string[] GetProductList() {

        // Truy xuất bảng khách hàng (từ cache, nếu có thể).
        DataTable dt = GetCustomerDataSet().Tables[0];

        // Tạo mảng chứa tên khách hàng
        string[] names = new string[dt.Rows.Count];

        // Đổ dữ liệu vào mảng.
        int i = 0;
        foreach (DataRow row in dt.Rows) {
            names[i] = row["ProductName"].ToString();
            i += 1;
        }
    }
}
```



```
        return names;
    }

    private DataSet GetCustomerDataSet() {

        // Kiểm tra item đã có trong cache chưa.
        DataSet ds = HttpContext.Current.Cache["Products"] as DataSet;

        if (ds == null) {

            string SQL = "SELECT * FROM Products";

            // Tạo các đối tượng ADO.NET.
            SqlConnection con = new SqlConnection(connectionString);
            SqlCommand com = new SqlCommand(SQL, con);
            SqlDataAdapter adapter = new SqlDataAdapter(com);
            ds = new DataSet();

            // Thực thi câu truy vấn.
            try {

                con.Open();
                adapter.Fill(ds, "Products");

                // Lưu item vào cache (trong 60 giây).
                HttpContext.Current.Cache.Insert("Products", ds, null,
                    DateTime.Now.AddSeconds(60), TimeSpan.Zero);
            } catch (Exception err) {
                System.Diagnostics.Debug.WriteLine(err.ToString());
            } finally {
                con.Close();
            }
        }

        return ds;
    }
}
```

4.

Tạo phương thức web hỗ trợ giao dịch

?

Bạn muốn thực thi tất cả các hành động của một phương thức web trong ngữ cảnh của một phiên giao dịch COM+ sao cho chúng chỉ có hai khả năng: hoặc là thành công hoặc là thất bại.

✕

Kích hoạt một phiên giao dịch tự động bằng cách chọn một giá trị thuộc kiểu liệt kê `System.EnterpriseServices.TransactionOption` và áp dụng nó cho thuộc tính `TransactionOption` của đặc tính `WebMethod`.

Trong *ASP.NET*, dịch vụ *Web XML* hỗ trợ các phiên giao dịch tự động (có thể được kích hoạt trên mỗi phương thức). Khi được kích hoạt, bất kỳ nguồn dữ liệu nào có hỗ trợ giao dịch COM+ sẽ tự động được đưa vào phiên giao dịch hiện hành khi nó được sử dụng trong mã lệnh của bạn. Phiên giao dịch sẽ tự động được commit khi phương thức web hoàn tất. Phiên giao dịch này được roll-back khi có bất kỳ ngoại lệ chưa-được-thụ-lý nào xảy ra hoặc bạn gọi phương thức `SetAbort` của lớp `System.EnterpriseServices.ContextUtil`.

Để kích hoạt việc hỗ trợ giao dịch cho một phương thức web, bạn cần thiết lập thuộc tính `TransactionOption` của đặc tính `WebMethod` là `RequiresNew`. Ví dụ, phương thức web (hỗ trợ giao dịch) dưới đây sẽ xóa các bản ghi trong một cơ sở dữ liệu và rồi hủy bỏ phiên giao dịch này. Để sử dụng đoạn mã này, bạn phải thêm một tham chiếu đến *System.EnterpriseServices.dll*.

```
using System;
using System.Data.SqlClient;
using System.Web.Services;
using System.EnterpriseServices;

public class TransactionTest {

    private static string connectionString = "Data Source=localhost;" +
        "Initial Catalog=Northwind;user ID=sa";

    [WebMethod(TransactionOption=TransactionOption.RequiresNew)]
    public void FailedTransaction() {

        // Tạo kết nối.
        SqlConnection con = new SqlConnection(connectionString);

        // Tạo câu truy vấn SQL.
        SqlCommand cmd = new SqlCommand("DELETE * FROM Customers", con);
```

```
// Thực thi câu truy vấn. Tác vụ này sẽ tự động được
// đăng ký như một phần trong phiên giao dịch.
con.Open();
cmd.ExecuteNonQuery();
con.Close();

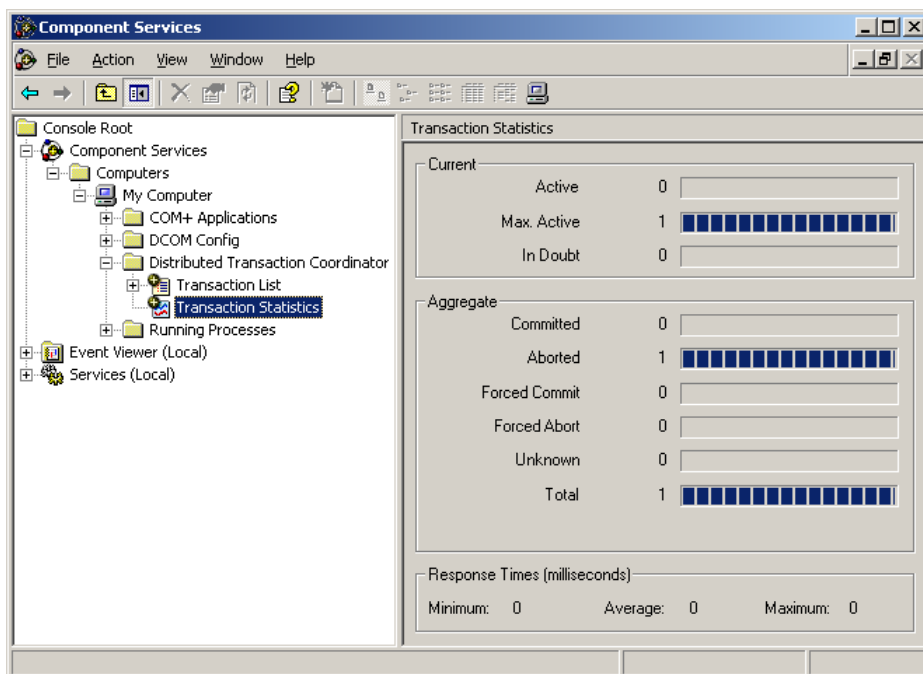
// Gọi một phương thức nào khác.
DoSomething();

// Nếu không có lỗi xảy ra, các thay đổi trong cơ sở dữ liệu
// sẽ được commit ở đây khi phương thức này kết thúc.
}

private void DoSomething() {

    // Hủy bỏ phiên giao dịch.
    ContextUtil.SetAbort();
}
```

Bạn có thể sử dụng tiện ích *Component Service* (trong phần *Administrative Tools* của *Control Panel*) để theo dõi phiên giao dịch trên. Trong tiện ích này, bạn hãy chọn mục *Distributed Transaction Coordinator* của máy tính hiện hành, và xem *Transaction Statistics*. Hình 12.2 cho thấy một phiên giao dịch không thành công khi chạy đoạn mã trên.



Hình 12.2 Theo dõi một phiên giao dịch không thành công

Vì bản chất phi trạng thái (*stateless*) của giao thức *HTTP*, nên một phương thức web chỉ có thể góp phần làm gốc của một phiên giao dịch; nghĩa là bạn không thể đưa nhiều hơn một phương thức web vào cùng một phiên giao dịch. Mặc dù thuộc tính *TransactionOption* chấp nhận tất cả các giá trị *TransactionOption* chuẩn, nhưng các giá trị này không mang ý nghĩa như mong đợi. Ví dụ, *Disabled*, *NotSupported*, và *Supported* đều có cùng tác dụng là vô hiệu việc hỗ trợ giao dịch. Tương tự, cả *Required* và *RequiresNew* đều kích hoạt việc hỗ trợ giao dịch và khởi chạy một phiên giao dịch mới. Bạn nên sử dụng *RequiresNew* trong các phương thức web vì tên của nó tương xứng với hành vi thật sự (!).



Các giao dịch *COM+* làm việc một cách trong suốt với hầu hết các nguồn dữ liệu vì chúng cung cấp các bộ quản lý tài nguyên tương thích. Nhưng luôn nhớ rằng, nếu bạn tương tác với một tài nguyên không hỗ trợ giao dịch, mã lệnh sẽ không được roll-back. Một số hoạt động không phải là giao dịch: ghi file, đặt thông tin vào trạng thái phiên làm việc, và truy xuất một thiết bị phần cứng (như máy in). Mặt khác, các thao tác dữ liệu với hầu hết các hệ cơ sở dữ liệu *Enterprise* (bao gồm *Microsoft SQL Server* và *Oracle*) đều là tương thích *COM+*.

5. Thiết lập thông tin xác thực cho dịch vụ Web XML

? Bạn muốn gửi các thông tin đăng nhập từ client của dịch vụ *Web XML* đến *IIS authentication*.

X Sử dụng thuộc tính `Credentials` của lớp proxy. Bạn có thể tạo một đối tượng `NetworkCredential` mới chứa `username` và `password`; hoặc sử dụng `CredentialCache` để lấy các thông tin xác thực của người dùng hiện hành.

Cũng giống như trang web, dịch vụ *Web XML* có thể được sử dụng cùng với *IIS authentication*. Những gì bạn cần làm là đặt dịch vụ *Web XML* vào một thư mục ảo hạn chế việc truy xuất nặc danh. Tuy nhiên, nếu người dùng có thể cung cấp các thông tin ánh xạ đến một tài khoản người dùng hợp lệ, người dùng này sẽ được xác thực và bạn có thể lấy các thông tin xác thực này thông qua đối tượng `WebService.User`.

Khác với trang web, dịch vụ *Web XML* không có sẵn phương thức nào để thu lấy thông tin xác thực từ client vì dịch vụ *Web XML* được thực thi bởi các ứng dụng khác, chứ không phải bởi người dùng. Do đó, ứng dụng đang tương tác với dịch vụ *Web XML* sẽ chịu trách nhiệm nhập bất kỳ thông tin xác thực cần thiết nào.

Ví dụ sau mô phỏng một dịch vụ *Web XML* có thực hiện xác thực người dùng. `GetIISUser` trả về người dùng đã được *IIS* xác thực. Nếu truy xuất nặc danh được phép thì kết quả sẽ là một chuỗi rỗng. Nếu truy xuất nặc danh bị từ chối thì kết quả sẽ là một chuỗi có dạng `[DomainName]\[UserName]` hay `[ComputerName]\[UserName]`.

```
public class AuthenticationTest : System.Web.Services.WebService {

    // Lấy thông tin về người dùng đã được IIS xác thực.
    [WebMethod()]
    public string GetIISUser() {
        return User.Identity.Name;
    }
}
```

Bước cuối cùng là tạo một client có thể cung cấp các thông tin xác thực. Các thông tin này được nhập thông qua thuộc tính `Credentials` của đối tượng proxy (thiết lập thuộc tính này tương tự như thiết lập thuộc tính `WebRequest.Credentials` khi lấy về một trang web—tham khảo mục 11.3). Đoạn mã dưới đây trình bày cách truy xuất một dịch vụ *Web XML* sử dụng *Basic authentication* (xác thực cơ bản):

```
// Tạo proxy.
localhost.AuthenticationTest proxy = new localhost.AuthenticationTest();

// Tạo thông tin xác thực.
proxy.Credentials = new System.Net.NetworkCredential(
    "myUserName", "myPassword");
```

```
Console.WriteLine(proxy.GetIISUser());
```

Đoạn mã dưới đây trình bày cách truy xuất một dịch vụ *Web XML* sử dụng *Integrated Windows authentication* (xác thực được tích hợp với *Windows*):

```
// Tạo proxy.
```

```
localhost.AuthenticationTest proxy = new localhost.AuthenticationTest();
```

```
// Gán thông tin xác thực của người dùng hiện hành cho lớp proxy.
```

```
proxy.Credentials = System.Net.CredentialCache.DefaultCredentials;
```

```
Console.WriteLine(proxy.GetIISUser());
```

6. Gọi bất đồng bộ một phương thức web

? Bạn cần gọi một phương thức web trong một tiểu trình khác để chương trình của bạn có thể thực hiện các tác vụ khác trong khi chờ đáp ứng.

✕ Sử dụng các phương thức bất đồng bộ có sẵn trong lớp proxy. Các phương thức này có tên là *BeginXXX* và *EndXXX* với *xxx* là tên của phương thức đồng bộ gốc.

Lớp proxy (được tạo tự động) có các tính năng cơ bản mà bạn cần để gọi bất kỳ phương thức web nào một cách bất đồng bộ. Ví dụ, phương thức web dưới đây có chức năng tạm dừng một khoảng thời gian ngẫu nhiên từ 10 đến 19 giây:

```
using System;
```

```
using System.Web.Services;
```

```
public class Wait : System.Web.Services.WebService {
```

```
    [WebMethod]
```

```
    public int Wait() {
```

```
        DateTime start = DateTime.Now;
```

```
        Random rand = new Random();
```

```
        TimeSpan delay = new TimeSpan(0, 0, rand.Next(10, 20));
```

```
        while (DateTime.Now < start.Add(delay)) {}
```

```
        return delay.Seconds;
```

```
    }
```

```
}
```

Với đoạn mã trên, lớp proxy tương ứng sẽ gồm ba phương thức: `Wait`, `BeginWait` và `EndWait`. Phương thức `Wait` gọi phương thức web một cách đồng bộ. Phương thức `BeginWait` khởi chạy phương thức web trong một tiểu trình riêng và trả về ngay lập tức. Phương thức `BeginXXX` luôn nhận nhiều hơn phương thức gốc hai đối số và trả về một đối tượng `IAsyncState`. Hai đối số này được sử dụng để nhập thông tin trạng thái và một callback. Đối tượng `IAsyncState` cho phép bạn xác định khi nào lời gọi kết thúc. Ví dụ, bạn có thể định kỳ kiểm tra thuộc tính `IAsyncState.IsComplete` để xác định lời gọi phương thức đã hoàn tất chưa. Khi nó đã hoàn tất, bạn nhập đối tượng `IAsyncState` vào phương thức `EndWait` để nhận giá trị trả về từ phương thức web. Nếu bạn gọi `EndWait` trước khi phương thức web hoàn tất, mã lệnh của bạn sẽ đợi cho đến khi nó hoàn tất.

Có hai mẫu bất đồng bộ phổ biến dùng cho dịch vụ *Web XML*. Cách thứ nhất là gọi vài phương thức bất đồng bộ một lượt, rồi đợi chúng hoàn tất. Cách này cho phép bạn giảm thời gian đợi tổng cộng, và nó làm việc tốt nhất với đối tượng `System.Threading.WaitHandle`. Ví dụ dưới đây gọi phương thức `Wait` ba lần:

```
using System;
using System.Threading;

public class WaitClient {

    [MTAThread]
    private static void Main() {

        localhost.WaitService proxy = new localhost.WaitService();

        DateTime startDate = DateTime.Now;

        // Gọi ba phương thức một cách bất đồng bộ.
        IAsyncResult handle1 = proxy.BeginWait(null, null);
        IAsyncResult handle2 = proxy.BeginWait(null, null);
        IAsyncResult handle3 = proxy.BeginWait(null, null);

        WaitHandle[] waitHandle = {handle1.AsyncWaitHandle,
                                   handle2.AsyncWaitHandle, handle3.AsyncWaitHandle};

        // Đợi cho cả ba phương thức hoàn tất.
        WaitHandle.WaitAll(waitHandle);

        int totalDelay = proxy.EndWait(handle1) +
                        proxy.EndWait(handle2) + proxy.EndWait(handle3);
```

```

        TimeSpan elapsedTime = DateTime.Now - startDate;

        Console.WriteLine("Completed after " + elapsedTime.ToString());
        Console.WriteLine("Total delay time: " + totalDelay.ToString());
    }
}

```

Trong trường hợp này, thời gian đã trôi qua nhỏ hơn thời gian trì hoãn tổng cộng:

```
Completed after 00:00:20.2591312
```

```
Total delay time: 47
```

Cách thứ hai là sử dụng callback. Bạn cần nhập một ủy nhiệm chỉ định một phương thức cụ thể trong mã lệnh. Khi phương thức web hoàn tất, ủy nhiệm này sẽ được gọi với đối số là một đối tượng `IAsyncResult` thích hợp.

Dưới đây là đoạn mã gọi phương thức `BeginWait` (cùng với một callback):

```
AsyncCallback callback = new AsyncCallback(Callback);
```

```
// Gọi phương thức một cách bất đồng bộ.
```

```
proxy.BeginWait(callback, proxy);
```

Và đây là callback (sẽ được kích hoạt khi thao tác hoàn tất):

```
public static void Callback(IAsyncResult handle) {

    localhost.WaitService proxy =
        (localhost.WaitService)handle.AsyncState;
    int result = proxy.EndWait(handle);
    Console.WriteLine("Waited " + result.ToString());
}

```

7.

Tạo lớp khả-truy-xuất-từ-xa



Bạn muốn tạo một lớp có thể được truy xuất từ một ứng dụng khác hay từ một máy tính khác trên mạng. Tuy vậy, bạn không cần tính tương thích xuyên-nền và bạn muốn có hiệu năng tối ưu.



Làm cho lớp này trở thành khả-truy-xuất-từ-xa (*remotable*) bằng cách dẫn xuất từ lớp `System.MarshalByRefObject`, và tạo một host để đăng ký lớp này với kiến trúc *.NET Remoting*.

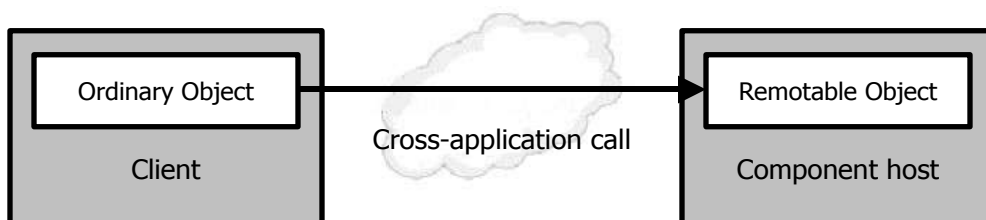
Remoting cho phép bạn làm cho một đối tượng trở nên truy xuất được qua các biên máy và biên tiến trình. Trong khi dịch vụ *Web XML* là giải pháp lý tưởng khi bạn cần chia sẻ các chức

nặng qua các nền hay các biên tin cậy, *Remoting* là lựa chọn tốt nhất cho một hệ thống mà trong đó tất cả các thành phần đều được xây dựng trên nền *.NET* và hệ điều hành *Windows*.

Để sử dụng *.NET Remoting*, bạn cần các phần sau (mỗi phần phải thuộc về một assembly riêng biệt):

- Lớp khả-truy-xuất-từ-xa— Lớp này có thể được truy xuất từ các ứng dụng và các máy tính khác, và phải dẫn xuất từ `System.MarshalByRefObject`.
- Ứng dụng host— Ứng dụng này đăng ký kiểu khả-truy-xuất-từ-xa với kiến trúc *.NET Remoting* bằng lớp `RemotingConfiguration` (thuộc không gian tên `System.Runtime.Remoting`). Miễn là ứng dụng host đang chạy, các client ở xa có thể tạo ra các thể hiện của lớp khả-truy-xuất-từ-xa.
- Ứng dụng client— Ứng dụng này có thể tạo ra các thể hiện của lớp khả-truy-xuất-từ-xa trong tiến trình của host và tương tác với chúng. Client sử dụng lớp `RemotingConfiguration` để đăng ký các kiểu mà nó muốn truy xuất từ xa.

Hình sau mô tả sự tương tác giữa ba phần trên. Trong ví dụ này, chỉ có một client. Tuy nhiên, cũng có thể có nhiều client tạo ra các thể hiện của lớp khả-truy-xuất-từ-xa cùng một lúc. Trong trường hợp này, mỗi client sẽ có một đối tượng khả-truy-xuất-từ-xa của chính nó, và tất cả các đối tượng này sẽ thuộc về miền ứng dụng của host.



Hình 12.3 Sử dụng một lớp khả-truy-xuất-từ-xa

Bước đầu tiên là tạo lớp khả-truy-xuất-từ-xa. Ví dụ, lớp khả-truy-xuất-từ-xa dưới đây trả về một `DataSet`; với cách này, một client có thể truy xuất thông tin từ cơ sở dữ liệu mà không phải trực tiếp kết nối đến cơ sở dữ liệu phía server.

```

using System;
using System.Data;
using System.Data.SqlClient;

public class ProductsDB : MarshalByRefObject {

    private static string connectionString = "Data Source=localhost;" +
        "Initial Catalog=Northwind;Integrated Security=SSPI";

    public DataTable GetProducts() {

        string SQL = "SELECT * FROM Products";
  
```

```

// Tạo các đối tượng ADO.NET.
SqlConnection con = new SqlConnection(connectionString);
SqlCommand com = new SqlCommand(SQL, con);
SqlDataAdapter adapter = new SqlDataAdapter(com);
DataSet ds = new DataSet();

// Thực thi câu truy vấn.
try {
    con.Open();
    adapter.Fill(ds, "Products");
} catch (Exception err) {
    Console.WriteLine(err.ToString());
} finally {
    con.Close();
}

return ds.Tables[0];
}

// Phương thức này kiểm tra Remoting có hoạt động hay không.
public string GetHostLocation() {
    return AppDomain.CurrentDomain.FriendlyName;
}
}

```

Lớp này được định nghĩa trong một *Class Library Assembly* có tên là *RemoteObject.dll*.



Ở mức lý tưởng, đối tượng ở xa sẽ không giữ lại bất kỳ trạng thái nào. Tính chất này cho phép bạn sử dụng chế độ kích hoạt gọi một lần (*single-call activation*), trong đó đối tượng sẽ được tạo ra ngay đầu mỗi lời gọi phương thức và sẽ được giải phóng khi kết thúc (giống như trong dịch vụ *Web XML*). Điều này bảo đảm các đối tượng không chiếm nhiều tài nguyên của server và việc quản lý thời gian sống của chúng trở nên dễ dàng hơn.

Kể đến, bạn phải tạo host—đây là ứng dụng phía server quản lý tất cả các thể hiện của lớp khả-truy-xuất-từ-xa. Bạn có thể sử dụng bất kỳ kiểu ứng dụng *.NET* nào làm host (bao gồm: ứng dụng dựa-trên-*Windows*, dịch vụ *Windows*, và ứng dụng *Console*). Dưới đây là một host đơn giản ở dạng *Console*:

```

using System;
using System.Runtime.Remoting;

public class Server {

    private static void Main() {

        // Đăng ký các lớp khả-truy-xuất-từ-xa với .NET Remoting.
        RemotingConfiguration.Configure("Server.exe.config");

        // Miễn là ứng dụng này đang chạy, các đối tượng ở xa
        // sẽ là khả truy xuất.

        Console.WriteLine("Press a key to shut down the server.");
        Console.ReadLine();

    }
}

```

Chương trình trên sử dụng file cấu hình (*app.config*) để cấu hình các lớp mà nó sẽ hỗ trợ, các cổng mà nó sẽ hỗ trợ cho giao tiếp mạng, và địa chỉ *URI (Uniform Resource Identifier)* mà client sẽ sử dụng để truy xuất đối tượng. Dưới đây là một file cấu hình đơn giản đăng ký lớp *RemoteObjects.RemoteObject* từ *RemoteObject.dll* với địa chỉ cổng là *9080* thông qua giao thức *TCP/IP*. Assembly này phải nằm trong *GAC (Global Assembly Cache)* hoặc trong cùng thư mục với ứng dụng server. File cấu hình cũng cấu hình đối tượng ở xa dùng chế độ kích hoạt gọi-một-lần.

```

<configuration>
  <system.runtime.remoting>
    <application>

      <!-- Định nghĩa đối tượng khả-truy-xuất-từ-xa. -->
      <service>
        <wellknown
          mode = "SingleCall"
          type="RemoteObject.ProductsDB, RemoteObject"
          objectUri="RemoteObject" />
      </service>

      <!-- Định nghĩa giao thức dùng cho truy xuất mạng.
           Bạn có thể sử dụng kênh tcp hay http. -->
      <channels>

```

```

        <channel ref="tcp" port="9080" />
    </channels>

    </application>
</system.runtime.remoting>
</configuration>

```

Host không bao giờ tương tác trực tiếp tới các đối tượng ở xa, những gì nó làm chỉ là đăng ký các kiểu thích hợp với kiến trúc *.NET Remoting*. Sau thời điểm đó, ứng dụng client có thể tạo ra các đối tượng này, và ứng dụng server có thể tiếp tục thực hiện các công việc khác. Tuy nhiên, khi host bị đóng, tất cả các đối tượng sẽ bị hủy, và không thể tạo đối tượng được nữa.

Ứng dụng client sử dụng file cấu hình tương tự như trên để định nghĩa địa chỉ *URL* và kiểu của đối tượng ở xa. Địa chỉ *URL* có định dạng như sau:

```
[Protocol]://[Server]:[PortNumber]/[ObjectURI]
```

Dưới đây là file cấu hình phía client:

```

<configuration>
  <system.runtime.remoting>
    <application>

      <!-- Định nghĩa đối tượng mà ứng dụng này
           muốn truy xuất từ xa. -->

      <client>
        <wellknown type="RemoteObject.ProductsDB, RemoteObject"
          url="tcp://localhost:9080/RemoteObject" />
      </client>

      <!-- Định nghĩa giao thức dùng cho truy xuất mạng.
           Giao thức này phải khớp với giao thức được
           định nghĩa phía server, nhưng địa chỉ cổng có thể
           khác. Địa chỉ cổng 0 nghĩa là "lấy bất kỳ
           một địa chỉ cổng nào còn trống". -->

      <channels>
        <channel ref="tcp" port="0" />
      </channels>

    </application>
  </system.runtime.remoting>

```

```
</configuration>
```

Ứng dụng client sử dụng phương thức `RemotingConfiguration.Configure` để đăng ký các đối tượng mà nó muốn gọi. Sau khi đã đăng ký xong, client có thể tạo đối tượng này giống như tạo đối tượng cục bộ mặc dù nó thật sự nằm trong miền ứng dụng của host. Đoạn mã dưới đây trình bày các bước này:

```
using System;
using System.Runtime.Remoting;
using System.Data;
using RemoteObject;

public class Client {

    private static void Main() {

        // Đăng ký các lớp sẽ được truy xuất từ xa.
        RemotingConfiguration.Configure("Client.exe.config");

        // Tương tác với đối tượng ở xa thông qua proxy.
        ProductsDB proxy = new ProductsDB();

        // Hiển thị tên miền ứng dụng của host.
        Console.WriteLine("Object executing in: " +
            proxy.GetHostLocation());

        // Lấy DataSet và hiển thị nội dung của nó.
        DataTable dt = proxy.GetProducts();

        foreach (DataRow row in dt.Rows) {
            Console.WriteLine(row[1]);
        }

        Console.ReadLine();
    }
}
```

Để tạo một đối tượng ở xa, client cần một tham chiếu đến assembly mà lớp này được định nghĩa trong đó. Điều này cần thêm một bước triển khai nữa, bạn có thể tránh đi bằng cách sử dụng một giao diện có định nghĩa các chức năng được hỗ trợ.



Để chuyển dữ liệu đến đối tượng ở xa (hoặc chuyển dữ liệu từ đối tượng ở xa về), các kiểu dữ liệu dùng cho các tham số và giá trị trả về phải là khả-tuần-tự-hóa (*serializable*—tất cả các kiểu cơ bản đều có tính chất này). Nếu muốn sử dụng các lớp tùy biến để chuyển dữ liệu, bạn phải làm cho các lớp này trở thành khả-tuần-tự-hóa bằng cách áp dụng đặc tính *Serializable* (xem mục 16.1 để biết thêm chi tiết).

8. Đăng ký tất cả các lớp khả-truy-xuất-từ-xa trong một assembly



Bạn muốn đăng ký tất cả các lớp khả-truy-xuất-từ-xa được định nghĩa trong một assembly mà không phải chỉ định chúng trong file cấu hình.



Nạp assembly (có chứa các lớp khả-truy-xuất-từ-xa) bằng cơ chế phản chiếu (*reflection*). Duyệt qua tất cả các kiểu trong đó và sử dụng phương thức `RemotingConfiguration.RegisterWellKnownServiceType` để đăng ký mỗi lớp khả-truy-xuất-từ-xa.

.NET cho phép bạn đăng ký các lớp khả-truy-xuất-từ-xa một cách dễ dàng thông qua việc viết file cấu hình hoặc viết mã chương trình. Xét ví dụ trong mục 12.7. Để đăng ký bằng mã chương trình, trước hết bạn phải gỡ bỏ những khai báo lớp trong file cấu hình trên server như sau:

```
<configuration>
  <system.runtime.remoting>
    <application>

      <channels>
        <channel ref="tcp" port="9080" />
      </channels>

    </application>
  </system.runtime.remoting>
</configuration>
```

Bây giờ, bạn có thể sử dụng kỹ thuật phản chiếu kết hợp với phương thức `RegisterWellKnownServiceType` để đăng ký tất cả các lớp khả-truy-xuất-từ-xa. Tuy nhiên, bạn cần thêm một tham chiếu đến `System.Runtime.Remoting.dll`. Đoạn mã dưới đây tìm các lớp khả-truy-xuất-từ-xa trong `RemoteObject.dll` và đăng ký mỗi lớp:

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
```

```

using System.Runtime.Remoting.Channels.Tcp;
using System.Reflection;

public class Server {

    private static void Main() {

        // Sử dụng file cấu hình để định nghĩa các tùy chọn về mạng.
        RemotingConfiguration.Configure("Server.exe.config");

        // Lấy kênh Remoting đã được đăng ký.
        TcpChannel channel =
            (TcpChannel)ChannelServices.RegisteredChannels[0];

        // Nạp RemoteObject.dll.
        Assembly assembly = Assembly.LoadFrom("RemoteObject.dll");

        // Xử lý tất cả các kiểu trong RemoteObject.dll.
        foreach (Type type in assembly.GetTypes()) {

            // Kiểm tra kiểu có phải là khả-truy-xuất-từ-xa hay không.
            if (type.IsSubclassOf(typeof(MarshalByRefObject))) {

                // Đăng ký kiểu (tên kiểu là địa chỉ URI).
                Console.WriteLine("Registering " + type.Name);
                RemotingConfiguration.RegisterWellKnownServiceType(
                    type, type.Name, WellKnownObjectMode.SingleCall);

                // Xác định địa chỉ URL (kiểu được publish tại đây).
                string[] urls = channel.GetUrlsForUri(type.Name);
                Console.WriteLine(urls[0]);

            }
        }

        Console.WriteLine("Press a key to shut down the server.");
        Console.ReadLine();

    }
}

```

}

9.

Quản lý các đối tượng ở xa trong IIS

?

Bạn muốn tạo một đối tượng khả-truy-xuất-từ-xa trong IIS (để có thể sử dụng SSL hay IIS authentication) thay cho một host chuyên biệt.

✕

Đặt file cấu hình và assembly vào một thư mục ảo, và thay đổi URI sao cho nó kết thúc bằng .rem hay .soap.

Thay vì tạo một host chuyên biệt, bạn có thể quản lý một lớp khả-truy-xuất-từ-xa trong IIS (*Internet Information Services*). Điều này cho phép bạn bảo đảm các lớp khả-truy-xuất-từ-xa sẽ luôn có hiệu lực, và cho phép bạn sử dụng các tính năng của IIS như *SSL Encryption* và *Integrated Windows authentication*.

Để quản lý một lớp khả-truy-xuất-từ-xa trong IIS, trước hết bạn phải tạo một thư mục ảo. Thư mục này chứa hai thứ: file cấu hình dùng để đăng ký các lớp khả-truy-xuất-từ-xa và thư mục *bin* dùng để chứa *Class Library Assembly* tương ứng (hoặc cài đặt assembly vào *GAC*).

File cấu hình này hoàn toàn tương tự với file cấu hình mà bạn sử dụng cho một host tùy biến. Tuy nhiên, bạn phải tuân theo các quy tắc:

- Bạn phải sử dụng kênh *HTTP* (mặc dù có thể sử dụng *Binary formatter* đối với các kích thước thông điệp nhỏ hơn).
- Bạn không thể chỉ cụ thể địa chỉ cổng. IIS lắng nghe tất cả các cổng bạn đã cấu hình trong *IIS Manager* (cổng 80 và 443).
- URI phải kết thúc bằng .rem hay .soap.
- File cấu hình phải có tên là *Web.config*, nếu không nó sẽ bị bỏ qua.

File *Web.config* dưới đây sẽ đăng ký lớp đã được trình bày trong mục 12.7:

```
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="SingleCall"
          type="RemoteObject.ProductsDB, RemoteObject"
          objectUri="RemoteObject.rem" />
      </service>

      <channels>
        <channel ref="http">

          <!-- Gỡ bỏ chú thích dưới đây để sử dụng Binary formatter
```



```

        thay cho SOAP formatter (mặc định). -->
    <!--
        <serverProviders>
            <formatter ref="binary"/>
        </serverProviders>
    -->

    </channel>
</channels>

</application>
</system.runtime.remoting>
</configuration>

```

Client có thể sử dụng đối tượng được quản lý trong *IIS* giống như đối tượng được quản lý trong một host tùy biến. Tuy nhiên, tên thư mục ảo sẽ là một phần của *URI*. Ví dụ, nếu file *Web.config* vừa trình bày ở trên được đặt trong thư mục ảo *http://localhost/RemoteObjects* thì *URL* đầy đủ sẽ là *http://localhost/RemoteObjects/RemoteObject.rem*.

 **Khi quản lý một đối tượng với *IIS*, tài khoản được sử dụng để thực thi đối tượng là tài khoản *ASP.NET* (được định nghĩa trong file *machine.config*). Nếu tài khoản này không có quyền truy xuất cơ sở dữ liệu (là trạng thái mặc định), bạn sẽ gặp lỗi khi chạy ví dụ này. Để giải quyết vấn đề này, bạn hãy xem mục 7.17.**

10. *Phát sinh sự kiện trên kênh truy xuất từ xa*

- ? Bạn cần tạo một client có thể nhận một sự kiện do đối tượng ở xa phát sinh.**
- * Phải chắc rằng bạn đang sử dụng các kênh hai chiều (*bidirectional channel*). Tạo một đối tượng khả-truy-xuất-từ-xa bên client (có thể nhận sự kiện từ server).**

Mặc dù cú pháp thụ lý sự kiện không hề thay đổi khi bạn sử dụng *.NET Remoting*, nhưng bạn cần tạo một client có thể thụ lý sự kiện từ một đối tượng ở xa. Dưới đây là các yêu cầu chính:

- Lớp khả-truy-xuất-từ-xa phải sử dụng chế độ được-client-kích-hoạt (*client-activated*) hay chế độ kích hoạt đơn-nhất (*singleton activation*)— không phải chế độ kích hoạt gọi-một-lần (*single-call activation*). Điều này bảo đảm đối tượng vẫn “còn sống” giữa các lần gọi phương thức, cho phép nó phát sinh sự kiện đến client.
- Client phải sử dụng kênh hai chiều để nó có thể nhận các kết nối do server khởi tạo.
- Đối tượng *EventArgs* phải là khả-tuần-tự-hóa để nó có thể được chuyển qua các biên miền ứng dụng.
- Client phải sử dụng một đối tượng khả-truy-xuất-từ-xa để nhận sự kiện (được gọi là listener). Theo đó, listener sẽ dựng một sự kiện cục bộ mà client có thể xử lý được. Đối

tượng ở xa không thể trực tiếp phát sinh sự kiện đến một lớp bình thường vì lớp bình thường không thể được truy xuất từ các miền ứng dụng khác.

- Bạn phải thay đổi các file cấu hình của client và server để cho phép “full serialization” (điều này không cần thiết với *.NET 1.0*).

Dưới đây là lớp khả-truy-xuất-từ-xa mà bạn có thể sử dụng để phát sinh một sự kiện đến client. Lớp này cung cấp phương thức `StartTask` để khởi chạy một bộ định thời, phát sinh sau một thời gian ngắn (khoảng 10 giây). Khi bộ định thời phát sinh, đối tượng khả-truy-xuất-từ-xa dựng lên sự kiện `TaskComplete`.

```
using System;
using System.Timers;

public delegate void TaskCompleted(object sender,
    TaskCompleteEventArgs e);

public class RemoteObject : MarshalByRefObject {

    public event TaskCompleted TaskComplete;
    private Timer tmr = new Timer();

    public void StartTask() {

        tmr.Interval = 10000;
        tmr.Elapsed += new ElapsedEventHandler(tmrCallback);
        tmr.Start();
    }

    private void tmrCallback(object sender, ElapsedEventArgs e) {

        tmr.Enabled = false;
        if (TaskComplete != null) {
            TaskComplete(this,
                new TaskCompleteEventArgs("Task completed on server"));
        }
    }

    public override object InitializeLifetimeService() {
```

```

        return null;
    }
}

[Serializable()]
public class TaskCompleteEventArgs : EventArgs {

    public string Result;

    public TaskCompleteEventArgs(string result) {
        this.Result = result;
    }
}

```

Bước kế tiếp là định nghĩa một lớp khả-truy-xuất-từ-xa chạy trên client và có thể nhận sự kiện này. Theo đó, lớp này có thể tiếp xúc với client. Lớp `EventListener` dưới đây trình bày một ví dụ như thế—nó chỉ đơn giản dựng lên sự kiện thứ hai, mà client có thể trực tiếp xử lý. Cũng như tất cả các đối tượng khả-truy-xuất-từ-xa, nó sẽ chỉ được truy xuất trong 5 phút, trừ khi bạn thay đổi chính sách “lease” (sẽ được mô tả trong mục 12.11). Có một cách là chép đề phương thức `InitializeLifetimeService` để cho phép đối tượng được sống vĩnh viễn:

```

public class EventListener : MarshalByRefObject {

    public event RemoteObject.TaskCompleted TaskComplete;

    // Thụ lý sự kiện ở xa.
    public void OnTaskComplete(object sender,
        RemoteObject.TaskCompleteEventArgs e) {
        TaskComplete(sender, e);
    }

    public override object InitializeLifetimeService() {
        return null;
    }
}

```

Listener phải được định nghĩa trong một assembly riêng để nó có thể được tham chiếu bởi ứng dụng client và lớp khả-truy-xuất-từ-xa (cả hai đều cần tương tác với nó).

Bây giờ ứng dụng client có thể khởi chạy tác vụ bất đồng bộ thông qua lớp `RemoteObject` và thụ lý sự kiện thông qua lớp `EventListener`. Đoạn mã dưới đây trình bày một client chỉ đơn giản hiển thị thông báo khi nhận được sự kiện:

```
using System;
using System.Windows.Forms;
using System.Runtime.Remoting;

public class ClientForm : System.Windows.Forms.Form {

    private System.Windows.Forms.Button cmdStart;

    // (Bỏ qua phần mã designer.)

    RemoteObject.RemoteObject remoteObj;
    EventListener.EventListener listener;

    private void ClientForm_Load(object sender, System.EventArgs e) {

        RemotingConfiguration.Configure("Client.exe.config");
        remoteObj = new RemoteObject.RemoteObject();
        listener = new EventListener.EventListener();
    }

    private void cmdStart_Click(object sender, System.EventArgs e) {

        // Kết nối phương thức thụ lý sự kiện ở xa.
        remoteObj.TaskComplete += new
            RemoteObject.TaskCompleted(listener.OnTaskComplete);

        // Kết nối phương thức thụ lý sự kiện cục bộ.
        listener.TaskComplete +=
            new RemoteObject.TaskCompleted(TaskComplete);

        remoteObj.StartTask();
        MessageBox.Show("Task has been started.");
    }

    // Định nghĩa phương thức thụ lý sự kiện cục bộ.
```

```
private void TaskComplete(object sender,
    RemoteObject.TaskCompleteEventArgs e) {

    MessageBox.Show("Event received: " + e.Result);

}
}
```

Để có thể làm việc, bạn phải chắc rằng client đang sử dụng các kênh hai chiều. Do đó, thẻ `<channel>` trong file cấu hình phải trông giống như sau:

```
<channel ref="tcp" port="0" />
```

Và không được giống như các ví dụ dưới đây:

```
<channel ref="tcp server" port="0" />
```

```
<channel ref="tcp client" port="0" />
```

Ngoài ra, bạn phải kích hoạt việc hỗ trợ “full serialization”. Nếu không, server sẽ không được phép nhận ủy nhiệm cho phương thức `Listener.TaskCompleted`, và sẽ không thể kết nối đến phương thức thụ lý sự kiện ở xa. Để kích hoạt việc hỗ trợ “full serialization” bên server, bạn cần thay đổi file cấu hình của host như sau:

```
<configuration>
  <system.runtime.remoting>
    <application>

      <client url="tcp://localhost:9080/Server">
        <activated type="RemoteObject.RemoteObject, RemoteObject"/>
      </client>

      <channels>
        <channel ref="tcp" port="0">
          <serverProviders>
            <formatter ref="binary" typeFilterLevel="Full" />
          </serverProviders>
        </channel>
      </channels>

    </application>
  </system.runtime.remoting>
</configuration>
```

Để kích hoạt việc hỗ trợ “full serialization” bên client, bạn cần thay đổi file cấu hình của client như sau:

```
<configuration>
```

```

<system.runtime.remoting>
  <application name="SimpleServer" >

    <service>
      <activated type="RemoteObject.RemoteObject, RemoteObject"/>
    </service>

    <channels>
      <channel ref="tcp" port="9080">
        <serverProviders>
          <formatter ref="binary" typeFilterLevel="Full" />
        </serverProviders>
      </channel>
    </channels>

  </application>
</system.runtime.remoting>
</configuration>

```

11. Kiểm soát thời gian sống của một đối tượng ở xa

- ?** Bạn muốn cấu hình thời gian sống của một đối tượng đơn-nhất hay được-client-kích-hoạt khi nó không còn được sử dụng.
- X** Chỉ định các thiết lập mặc định về thời gian sống trong file cấu hình của host; chép đè phương thức `InitializeLifetimeService` trong lớp ở xa; hoặc hiện thực một cơ chế kiểm soát thời gian sống bên client.

Nếu một đối tượng sử dụng chế độ kích hoạt gọi một lần (*single-call activation*), nó sẽ tự động bị hủy vào cuối mỗi lời gọi phương thức. Điều này khác với các đối tượng được-client-kích-hoạt (*client-activated*) và đơn-nhất (*singleton*), các đối tượng này có thời gian sống lâu hơn vì tuân theo “lifetime lease” (tạm dịch là “hợp đồng cho thuê thời gian sống”). Với các thiết lập mặc định, một đối tượng ở xa sẽ tự động bị hủy nếu nó không hoạt động trong hai phút, miễn là nó đã tồn tại ít nhất năm phút.

Host, đối tượng ở xa, và client đều có thể thay đổi các thiết lập về thời gian sống.

- Host có thể chỉ định các thiết lập mặc định về thời gian sống trong file cấu hình. Các thiết lập này sẽ áp dụng cho tất cả các đối tượng ở xa mà nó quản lý.
- Lớp ở xa có thể chép đè phương thức `GetLifetimeService` để điều chỉnh các thiết lập lease ban đầu bằng đối tượng `ILease`.

- Client có thể gọi phương thức `MarshalByRefObject.GetLifetimeService` với một đối tượng ở xa cụ thể để thu lấy một thể hiện `ILease`. Kế tiếp, client có thể gọi phương thức `ILease.Renew` để chỉ định lượng thời gian tối thiểu mà đối tượng sẽ sống.

Ví dụ dưới đây sử dụng cách tiếp cận thứ nhất (dùng thẻ `<lifetime>` trong file cấu hình của host). Các thiết lập lease áp dụng cho tất cả các đối tượng ở xa do host tạo ra. Sử dụng `m` để chỉ phút hay `s` để chỉ giây. Đối tượng ở xa có thời gian sống ban đầu là 10 phút. Khi client truy xuất đối tượng, thời gian sống của nó tự động được làm mới ít nhất ba phút.

```
<configuration>
  <system.runtime.remoting>
    <application>

      <service>
        <wellknown
          mode = "Singleton"
          type="RemoteObjects.RemoteObject, RemoteObjects"
          objectUri="RemoteObject" />
        </service>

      <channels>
        <channel ref="tcp" port="9080" />
      </channels>

      <lifetime leaseTime = "10M"
        renewOnCallTime = "3M" />

    </application>
  </system.runtime.remoting>
</configuration>
```

Một cách tiếp cận khác là chép đè phương thức `InitializeLifetimeService` để một đối tượng ở xa tự kiểm soát thời gian sống của nó. Bạn có thể thêm đoạn mã dưới đây vào lớp ở xa để nó có thời gian sống mặc định là 10 phút và thời gian làm mới là 5 phút:

```
public override object InitializeLifetimeService() {
    ILease lease = MyBase.InitializeLifetimeService();

    // Lease chỉ có thể được cấu hình nếu nó đang ở trạng thái ban đầu.
    if (lease.CurrentState == LeaseState.Initial) {
        lease.InitialLeaseTime = TimeSpan.FromMinutes(10);
        lease.RenewOnCallTime = TimeSpan.FromMinutes(5);
    }
}
```

```
return lease;
}
```

Nếu muốn đối tượng có thời gian sống vô hạn, bạn chỉ cần trả về một tham chiếu `null` thay vì là đối tượng `ILease`. Trường hợp thông thường nhất là bạn muốn tạo một đối tượng đơn-nhất chạy độc lập (và lâu dài) ngay cả khi client không sử dụng nó.

12. *Kiểm soát phiên bản của các đối tượng ở xa*

- ?** Bạn muốn tạo một host có thể quản lý nhiều phiên bản của một đối tượng.
- X** Cài đặt tất cả các phiên bản của đối tượng vào *GAC*, và đăng ký mỗi phiên bản tại một endpoint *URI* khác biệt.

.NET Remoting không có sự hỗ trợ nội tại nào cho việc đánh phiên bản. Khi một client tạo một đối tượng ở xa, host tự động sử dụng phiên bản trong thư mục cục bộ hoặc, trong trường hợp là một assembly dùng chung, phiên bản mới nhất trong *GAC*. Để hỗ trợ nhiều phiên bản, bạn có ba lựa chọn:

- Tạo các ứng dụng host riêng biệt. Mỗi host sẽ có một phiên bản khác nhau của assembly “đối tượng ở xa” và sẽ đăng ký phiên bản của nó với một *URI* khác nhau. Cách này buộc bạn phải chạy nhiều ứng dụng host cùng một lúc và thiết thực nhất khi bạn đang sử dụng *IIS* (được mô tả trong mục 12.9).
- Tạo một assembly “đối tượng ở xa” hoàn toàn mới (thay vì thay đổi phiên bản). Theo đó, bạn có thể đăng ký các lớp trong hai assembly tại các *URI* khác nhau, sử dụng cùng host.
- Cài đặt tất cả các phiên bản của assembly “đối tượng ở xa” vào *GAC*. Theo đó, bạn có thể tạo một host ánh xạ các *URI* khác nhau đến các phiên bản cụ thể của assembly “đối tượng ở xa”.

Tùy chọn cuối cùng là linh hoạt nhất trong trường hợp bạn cần hỗ trợ nhiều phiên bản. Ví dụ, file cấu hình dưới đây đăng ký hai phiên bản của *RemoteObjects* tại hai endpoint khác nhau. Bạn cần ghi rõ số phiên bản và token khóa công khai khi sử dụng các assembly trong *GAC*. Bạn có thể tìm thấy thông tin này bằng cách xem assembly trong *Windows Explorer GAC plug-in* (vào thư mục *C:\Windows\Assembly*).

```
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <!-- Thông tin type được chia thành ba hàng cho hợp với
              biên trang. Trong file cấu hình, thông tin này phải
              được đặt trên một hàng. -->
```



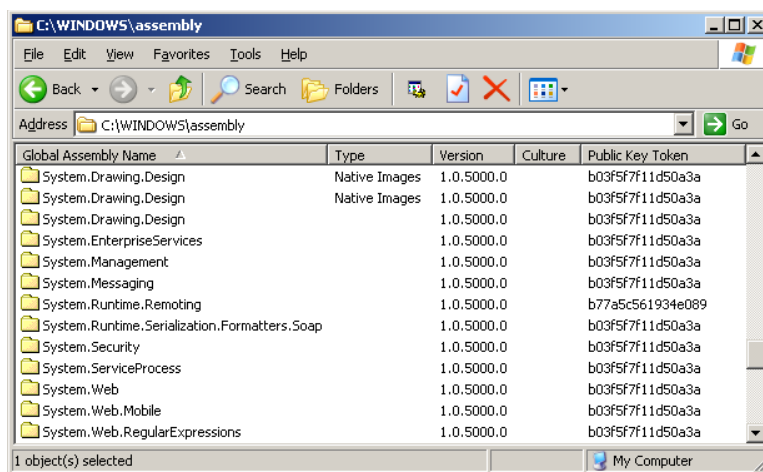
```

<wellknown mode="SingleCall"
  type="RemoteObjects.RemoteObject, RemoteObjects,
    Version 1.0.0.1, Culture=neutral,
    PublicKeyToken=8b5ed84fd25209e1"
  objectUri="RemoteObj" />

<wellknown mode="SingleCall"
  type="RemoteObjects.RemoteObject, RemoteObjects,
    Version 2.0.0.1, Culture=neutral,
    PublicKeyToken=8b5ed84fd25209e1"
  objectUri="RemoteObj_2.0" />
</service>
<channels>
  <channel ref="tcp server" port="9080" />
</channels>
</application>
</system.runtime.remoting>
</configuration>

```

File cấu hình của client không phải thay đổi gì hết (trừ việc cập nhật *URI*, nếu cần). Client “chọn” phiên bản mà nó muốn sử dụng bằng *URI* tương ứng.



Hình 12.4 Thư mục C:\Windows\Assembly

13.

*Tạo phương thức một chiều
với dịch vụ Web XML hay Remoting*

?

Bạn muốn một phương thức web hay một thành phần ở xa thực hiện một tác vụ kéo dài, và bạn không muốn bắt client phải đợi trong lúc mã lệnh đang thực thi.

✕

Tạo một phương thức web một chiều bằng cách áp dụng đặc tính `SoapDocumentMethod` hay `SoapRpcMethod` và thiết lập thuộc tính `OneWay` của đặc tính này là `true`. Tạo một phương thức *Remoting* một chiều bằng cách áp dụng đặc tính `OneWay` thuộc không gian tên `System.Runtime.Remoting.Messaging`.

Với các phương thức một chiều, client gửi một thông điệp yêu cầu, và server đáp ứng tức thì để cho biết rằng phương thức đã bắt đầu quá trình xử lý. Cách làm việc này có các hệ quả sau:

- Client không cần đợi trong lúc mã lệnh đang thực thi.
- Phương thức không thể trả về thông tin nào cho client (thông qua một giá trị trả về hay một thông số `ByRef`).
- Nếu phương thức ném một ngoại lệ chưa-được-thụ-lý, nó sẽ không được truyền về cho client.

Rõ ràng là các phương thức một chiều không phù hợp khi client cần nhận thông tin từ server. Tuy nhiên, chúng lại lý tưởng khi cần khởi chạy một kiểu tác vụ nào đó phía server (chẳng hạn, bắt đầu một công việc xử lý bổ).

Để tạo một phương thức web một chiều, bạn cần áp dụng đặc tính `SoapDocumentMethod` (thuộc không gian tên `System.Web.Services.Protocols`) cho phương thức thích hợp và thiết lập thuộc tính `OneWay` là `true`. Ví dụ dưới đây là một dịch vụ *Web XML* có hai phương thức, mỗi phương thức đều gây trì hoãn 10 giây. Một trong hai phương thức này sử dụng đặc tính `SoapDocumentMethod` (để client không phải đợi 10 giây).

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;

public class OneWayTestWebService {

    [WebMethod()]
    public void DoLongTaskWithWait() {
        // (Bắt đầu một tác vụ kéo dài và khiến client phải đợi.)
        Delay(10);
    }

    [WebMethod, SoapDocumentMethod(OneWay=true)]
    public void DoLongTaskWithoutWait() {
```

```

        // (Bắt đầu một tác vụ kéo dài nhưng không khiến client phải đợi.)
        Delay(10);
    }

    private void Delay(int seconds) {
        DateTime currentTime = DateTime.Now;
        while (DateTime.Now.Subtract(currentTime).TotalSeconds < seconds)
        {}
    }
}

```

Ví dụ trên giả định dịch vụ *Web XML* và client đang sử dụng *SOAP document* (mặc định). Nếu đang sử dụng *Remote Procedure Call (RPC)*, bạn hãy sử dụng đặc tính tương ứng là `SoapRpcMethod` để đánh dấu phương thức một chiều.

Để tạo một phương thức một chiều trong một thành phần trên *Remoting*, bạn cần áp dụng đặc tính `OneWay` (thuộc không gian tên `System.Runtime.Remoting.Messaging`) cho phương thức thích hợp. Đoạn mã dưới đây trình bày một ví dụ giống như trên nhưng với một thành phần ở xa:

```

using System;
using System.Runtime.Remoting.Messaging;

public class OneWayTestRemoting : MarshalByRefObject {

    public void DoLongTaskWithWait() {
        // (Bắt đầu một tác vụ kéo dài và khiến client phải đợi.)
        Delay(10);
    }

    [OneWay()]
    public void DoLongTaskWithoutWait() {
        // (Bắt đầu một tác vụ kéo dài nhưng không khiến client phải đợi.)
        Delay(10);
    }

    private void Delay(int seconds) {
        DateTime currentTime = DateTime.Now;
        while (DateTime.Now.Subtract(currentTime).TotalSeconds < seconds) {}
    }
}

```



Phương thức một chiều không phải là cách duy nhất để loại bỏ việc trì hoãn client. Bạn cũng có thể điều chỉnh client gọi phương thức web một cách bất đồng

bộ. Trong trường hợp này, client sẽ đợi dịch vụ *Web XML* hoàn tất, nhưng nó sẽ đợi trên một tiểu trình khác, và do đó client có thể tiếp tục với các công việc khác. Việc gọi phương thức bất đồng bộ đã được mô tả trong mục 12.6.

13

BẢO MẬT

Mục tiêu chính của *Microsoft .NET Framework* là làm cho việc lập trình trở nên an toàn hơn—đặc biệt lưu tâm đến việc sử dụng mobile code⁵ và các hệ thống phân tán. Hầu hết các hệ điều hành hiện đại (bao gồm *Microsoft Windows*) đều hỗ trợ bảo mật dựa-trên-người-dùng (*User-Based Security*), cho phép bạn kiểm soát các hành động và các tài nguyên mà một người dùng truy xuất đến. Tuy nhiên, do sự phát triển của mạng máy tính, đặc biệt là Internet, sự bảo mật nếu chỉ dựa vào định danh của người dùng trên hệ thống là chưa đủ. Khi quan tâm đến bảo mật, mã lệnh không nên tự động nhận mức tin cậy như mức tin cậy mà bạn đã ấn định cho người đang chạy mã lệnh này.

.NET Framework kết hợp hai mô hình bảo mật bổ sung lẫn nhau (thực hiện nhiều vấn đề liên quan đến bảo mật người dùng và mã lệnh):

- *CAS (Code Access Security)*—Bảo mật truy xuất mã lệnh)
- *RBS (Role-Based Security)*—Bảo mật dựa-trên-vai-trò)

CAS và *RBS* không thay thế hay sao lại các phương tiện bảo mật do hệ điều hành nằm dưới cung cấp. Chúng là các cơ chế độc lập nền, cấp thêm các khả năng bảo mật để nâng cao tính bảo mật tổng thể trong các giải pháp được-quản-lý.

CAS sử dụng các thông tin về nguồn gốc của một assembly đã được thu thập lúc thực thi—đây là chứng cứ (*evidence*)—để xác định xem mã lệnh có thể truy xuất các hành động và tài nguyên nào—đây là quyền (*permission*). Chính sách bảo mật của *.NET Framework*—một tập hợp phân cấp các quy tắc cấu hình—định nghĩa phép ánh xạ giữa chứng cứ và quyền. Thư viện lớp *.NET Framework* sử dụng các yêu cầu quyền (*permission demand* hay *permission request*) để bảo vệ các chức năng quan trọng nhất của nó không bị truy xuất trái phép. Một yêu cầu buộc bộ thực thi bảo đảm rằng: nếu muốn gọi một phương thức được-bảo-vệ thì mã lệnh phải có một quyền cụ thể nào đó. *CAS* bảo đảm rằng: khả năng thực thi của mã lệnh tùy thuộc vào mức độ tin cậy của bạn đối với người tạo ra mã và nguồn gốc của nó, chứ không phải mức độ tin cậy đối với người dùng đang chạy mã. Các mục liên quan đến *CAS* trong chương này thảo luận các vấn đề sau:

- Cho phép mã lệnh có-độ-tin-cậy-một-phần (*partially trusted code*) truy xuất các assembly tên mạnh của bạn (mục 13.1).
- Vô hiệu hoàn toàn *CAS* (mục 13.2) hoặc chỉ vô hiệu việc kiểm tra quyền thực thi (mục 13.3).
- Yêu cầu các quyền truy xuất mã lệnh cụ thể và xác định xem bộ thực thi đã cấp các quyền nào cho mã lệnh của bạn (mục 13.4, 13.5, 13.6, và 13.7).
- Kiểm soát sự thừa kế và chép đề thành viên bằng *CAS* (mục 13.8).
- Xem xét và xử lý chứng cứ của assembly (mục 13.9 và 13.10).
- Xử lý bảo mật bộ thực thi bằng miền ứng dụng (mục 13.11 và 13.12).

RBS cho phép bạn thực hiện các quyết định lúc thực thi (*runtime decision*) dựa trên định danh (*identity*) và các vai trò (*role*) của người dùng mà ứng dụng đang chạy trên danh nghĩa người dùng này. Trên hệ điều hành *Windows*, đây chính là việc thực hiện các quyết định dựa trên tên người dùng *Windows* và các nhóm *Windows* mà người dùng đó thuộc về. Tuy nhiên, *RBS*

⁵ Mobile code là phần mềm được truyền qua một hệ thống mạng và sau đó được thực thi trên hệ thống cục bộ.

cung cấp một cơ chế bảo mật chung không lệ thuộc vào hệ điều hành nằm dưới, cho phép bạn tích hợp vào bất kỳ hệ thống tài khoản người dùng nào. Các mục trong chương này thảo luận các vấn đề sau đây của *.NET RBS*:

- Tích hợp *RBS* với các tài khoản người dùng *Windows* và xác định xem một người dùng có là thành viên của một nhóm *Windows* nào đó hay không (mục 13.13).
- Kiểm soát việc truy xuất đến các chức năng của ứng dụng dựa trên người dùng hiện hành và các vai trò mà người dùng này là một thành viên (mục 13.14).
- Giả nhận một người dùng *Windows* để thực hiện các tác vụ hệ điều hành trên danh nghĩa người dùng đó (mục 13.15).

Các mục liên quan đến *RBS* và *CAS* trong chương này trình bày một số công việc thông thường mà bạn sẽ cần thực hiện trong các ứng dụng, nhưng chúng chỉ mô tả một phần nhỏ trong các khả năng bảo mật của *.NET Framework*. Để hiểu rõ hơn, bạn hãy tham khảo một quyển sách khác chuyên về bảo mật trong *.NET Framework*.

1. Cho phép mã lệnh có-độ-tin-cây-một-phần sử dụng assembly tên mạnh của bạn

? Bạn cần viết một assembly chia sẻ sao cho nó là khả truy xuất đối với mã lệnh có-độ-tin-cây-một-phần (theo mặc định, bộ thực thi không cho phép mã lệnh có-độ-tin-cây-một-phần truy xuất các kiểu và các thành viên nằm trong một assembly tên mạnh).

✕ Áp dụng đặc tính `System.Security.AllowPartiallyTrustedCallersAttribute` cho assembly chia sẻ của bạn.

Để giảm thiểu các nguy cơ bảo mật do mã lệnh nguy hiểm bày ra, bộ thực thi không cho phép các assembly có-độ-tin-cây-một-phần truy xuất đến các assembly tên mạnh. Hạn chế này làm giảm nguy cơ mã lệnh nguy hiểm tấn công vào hệ thống của bạn, nhưng đối với một cách tiếp cận áp chế như thế cần phải có lời giải thích.

Theo quy tắc, các assembly tên mạnh được cài đặt trong *Global Assembly Cache (GAC)* và chứa các chức năng quan trọng được dùng chung giữa nhiều ứng dụng. Điều này hoàn toàn đúng với các assembly cấu thành thư viện lớp *.NET Framework*. Các assembly tên mạnh khác từ các sản phẩm được-phân-bổ-rộng-rãi cũng sẽ nằm trong *GAC* và là khả truy xuất đối với các ứng dụng được-quản-lý. Khả năng hiện diện trong *GAC* cao, tính khả truy xuất dễ dàng, và tầm quan trọng đối với nhiều ứng dụng khác nhau khiến cho các assembly tên mạnh là mục tiêu có khả năng nhất đối với bất cứ hành động phá hoại nào của mã lệnh nguy hiểm được-quản-lý.

Thông thường, mã lệnh có khả năng nguy hiểm là mã được nạp từ các nơi xa—như Internet—ở đó bạn có ít (hay không có) sự kiểm soát nào. Với chính sách bảo mật mặc định, tất cả mã lệnh chạy từ máy cục bộ đều có độ tin cậy toàn phần (*full trust*), trong khi mã lệnh được nạp từ các nơi xa chỉ có độ tin cậy một phần (*partial trust*). Ngăn mã lệnh có-độ-tin-cây-một-phần truy xuất đến các assembly tên mạnh; nghĩa là mã lệnh có-độ-tin-cây-một-phần không có cơ

hội sử dụng các tính năng của assembly cho các mục đích nguy hiểm, và không thể khảo sát assembly để tìm các lỗ hổng có thể khai thác được. Dĩ nhiên, lý thuyết này giả định rằng bạn quản lý chính sách bảo mật một cách phù hợp. Nếu bạn gán tất cả mã lệnh có độ tin cậy toàn phần, không chỉ bất kỳ assembly nào cũng có thể truy xuất assembly tên mạnh của bạn, mà mã lệnh này còn có thể truy xuất tất cả các chức năng của *.NET Framework*. Điều này sẽ là một tai họa bảo mật!



Nếu bạn thiết kế, hiện thực, và thử nghiệm assembly chia sẻ của bạn một cách phù hợp bằng CAS để giới hạn việc truy xuất đến các thành viên quan trọng, bạn không cần áp đặt một hạn chế bao trùm để ngăn mã lệnh có-độ-tin-cậy-một-phần sử dụng assembly của bạn. Tuy nhiên, đối với một assembly bất kỳ, không thể chứng minh rằng không có lỗ hổng bảo mật nào để mã nguy hiểm có thể lợi dụng. Do đó, bạn nên xem xét cẩn thận nhu cầu cho phép mã lệnh có-độ-tin-cậy-một-phần truy xuất assembly tên mạnh trước khi áp dụng đặc tính `AllowPartiallyTrustedCallersAttribute`.

Bộ thực thi ngăn mã lệnh có-độ-tin-cậy-một-phần truy xuất các assembly tên mạnh bằng cách đặt `LinkDemand` vào tập quyền `FullTrust` trên mọi thành viên công-khai (*public*) và được-bảo-vệ (*protected*) của mỗi kiểu khả-truy-xuất-công-khai được định nghĩa trong assembly. Điều này có nghĩa là chỉ những assembly được cấp các quyền tương đương với tập quyền `FullTrust` thì mới có thể truy xuất các kiểu và thành viên trong assembly tên mạnh. Việc áp dụng `AllowPartiallyTrustedCallersAttribute` vào assembly tên mạnh báo với bộ thực thi không buộc `LinkDemand` có hiệu lực trên các thành viên và các kiểu bên trong.



Bộ thực thi chịu trách nhiệm buộc các hoạt động bảo mật ngầm `LinkDemand` có hiệu lực để bảo vệ các assembly tên mạnh; C# assembler không sinh ra các lệnh khai báo `LinkDemand` lúc biên dịch.

Đoạn mã dưới đây trình bày một ứng dụng có sử dụng đặc tính `AllowPartiallyTrustedCallersAttribute`. Chú ý rằng, bạn phải sử dụng tiền tố `assembly:` để báo với trình biên dịch rằng đích của đặc tính này là assembly (còn được gọi là đặc tính toàn cục—*global attribute*). Ngoài ra, không cần thêm phần `Attribute` vào tên của đặc tính—mặc dù bạn có thể thêm vào nếu muốn. Vì bạn nhầm lẫn đến assembly nên đặc tính này phải được đặt sau các lệnh `using` mức trên, nhưng trước các khai báo không gian tên và kiểu.

```
using System.Security;
```

```
[assembly:AllowPartiallyTrustedCallers]
```

```
public class AllowPartiallyTrustedCallersExample {
```

```
    §
```

```
}
```



Thực tế, tất cả các đặc tính toàn cục đều nằm trong một file độc lập với phần mã lệnh còn lại của ứng dụng. Microsoft Visual Studio .NET sử dụng cách tiếp cận này: tạo một file có tên là *AssemblyInfo.cs* để chứa tất cả các đặc tính toàn cục.

Nếu sau khi áp dụng `AllowPartiallyTrustedCallersAttribute` cho assembly, bạn muốn mã lệnh có-độ-tin-cây-một-phần chỉ gọi được một số thành viên nào đó, bạn cần bổ sung `LinkDemand` cho tập quyền `FullTrust` trên các thành viên cần thiết, như được trình bày trong đoạn mã dưới đây:

```
[System.Security.Permissions.PermissionSetAttribute
    (System.Security.Permissions.SecurityAction.LinkDemand,
     Name="FullTrust")]
public void SomeMethod() {
    §
}
```

2.

Vô hiệu bảo mật truy xuất mã lệnh



Bạn cần vô hiệu CAS.



Thiết lập thuộc tính `SecurityEnabled` của lớp `System.Security.SecurityManager` là `false` và lưu lại bằng phương thức `SecurityManager.SavePolicy`. Bạn cũng có thể sử dụng công cụ *Code Access Security Policy* (*Caspol.exe*) và thực thi lệnh `caspol -s off`.

CAS là phần then chốt trong mô hình bảo mật của bộ thực thi .NET, và cũng là phần đặc trưng của nền tảng .NET mà nhiều nền tảng khác không có. Mặc dù CAS được xây dựng với tiêu chí đảm bảo hiệu năng thực thi cao nhất và đã được sử dụng một cách cẩn trọng trong thư viện lớp .NET, nhưng vẫn có một chi phí cho mỗi yêu cầu bảo mật (*security demand*) và stack walk (kết quả) mà bộ thực thi phải thực hiện.

Đôi lúc, bảo mật mức-mã-lệnh có thể không là điều bạn bận tâm, hoặc nhu cầu hiệu năng có thể vượt quá nhu cầu CAS. Trong các trường hợp này, bạn có thể vô hiệu hoàn toàn CAS và loại bỏ chi phí cho việc kiểm tra bảo mật mức-mã-lệnh. Vô hiệu CAS có tác dụng trao cho mã lệnh khả năng thực hiện bất kỳ hành động nào mà .NET Framework hỗ trợ (tương đương với tập quyền `FullTrust`), bao gồm khả năng nạp mã lệnh khác, gọi các thư viện nguyên sinh, và sử dụng con trỏ để trực tiếp truy xuất bộ nhớ.



Bạn chỉ nên vô hiệu CAS vì các lý do hiệu năng sau khi đã tiêu hết tất cả các chừng mực có thể khác để đạt được các đặc điểm hiệu năng mà ứng dụng của bạn đòi hỏi. Việc lập profile cho mã lệnh thường sẽ nhận biết những vùng mà bạn có thể cải thiện đáng kể hiệu năng nhưng không phải vô hiệu CAS. Ngoài ra, bạn cần bảo đảm các tài nguyên hệ thống đã được bảo vệ bằng các cơ chế bảo mật của hệ điều hành (như *Windows ACLs*) trước khi vô hiệu CAS.

Caspol.exe là một tiện ích được cấp cùng với .NET Framework, cho phép bạn cấu hình chính sách bảo mật truy xuất mã lệnh từ dòng lệnh. Khi bạn nhập lệnh `caspol -s off` hoặc `caspol -s on`, tiện ích này sẽ xác lập thuộc tính `SecurityEnabled` của lớp `SecurityManager`. Lớp `SecurityManager` cung cấp tập các phương thức tĩnh để truy xuất dữ liệu và chức năng bảo

mật quan trọng. Đoạn mã dưới đây trình bày cách sử dụng thuộc tính `SecurityEnabled` để vô hiệu và kích hoạt *CAS*:

```
// Vô hiệu CAS.
System.Security.SecurityManager.SecurityEnabled = false;

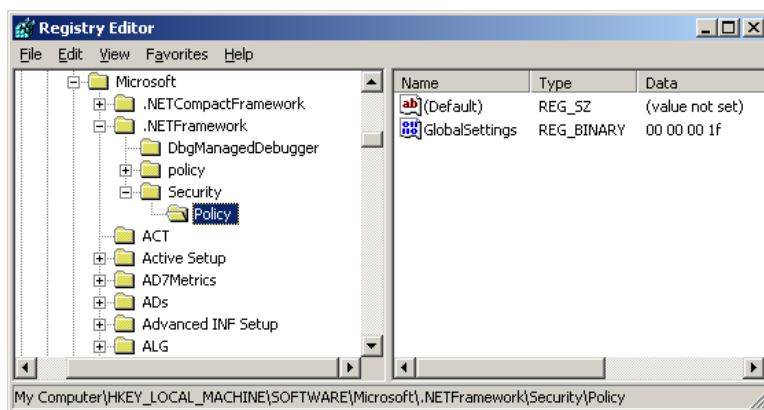
// Lưu cấu hình.
System.Security.SecurityManager.SavePolicy();

// Kích hoạt CAS.
System.Security.SecurityManager.SecurityEnabled = true;

// Lưu cấu hình.
System.Security.SecurityManager.SavePolicy();
```

Để vô hiệu *CAS*, mã lệnh của bạn phải có phần tử `ControlPolicy` của `System.Security.Permissions.SecurityPermission`. Để kích hoạt *CAS*, bạn không cần phải có quyền cụ thể nào.

Thay đổi `SecurityEnabled` sẽ không ảnh hưởng đến hoạt động của *CAS* trong các tiến trình hiện có, và cũng không ảnh hưởng đến các tiến trình mới cho đến khi bạn gọi phương thức `SavePolicy` để lưu trạng thái của `SecurityEnabled` vào *Windows Registry*. Đáng tiếc, *.NET Framework* không bảo đảm những thay đổi của `SecurityEnabled` sẽ tác động đúng đến hoạt động của *CAS* trong tiến trình hiện hành. Do đó, bạn phải thay đổi `SecurityEnabled` rồi mở một tiến trình mới để có được hoạt động đúng như mong muốn.



Hình 13.1 Registry Editor (CAS đã bị vô hiệu)



Trạng thái hiện hành của *CAS* (on/off) được lưu trong khóa `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NETFramework\Security\Policy` của *Windows Registry*. Nếu khóa này không tồn tại, mặc định *CAS* là on.

3.

Vô hiệu việc kiểm tra quyền thực thi

?

Bạn cần ngăn bộ thực thi kiểm tra mỗi assembly được nạp vào có quyền thực thi (*execution permission*) hay không.

✕

Thiết lập thuộc tính `CheckExecutionRights` của lớp `System.Security.SecurityManager` là `false` rồi lưu lại bằng phương thức `SecurityManager.SavePolicy`. Bạn cũng có thể sử dụng công cụ *Code Access Security Policy (Caspol.exe)* và thực thi lệnh `caspol -e off`.

Mỗi khi nạp assembly, bộ thực thi bảo đảm grant-set của assembly này có chứa phần tử `Execution` của `SecurityPermission`. Bộ thực thi hiện thực một quá trình phân giải chính sách “lười” (*lazy policy resolution process*), nghĩa là grant-set của một assembly không được tính cho đến khi có một yêu cầu bảo mật (*security demand*) được thực hiện trên assembly này. Việc kiểm tra quyền thực thi không chỉ buộc bộ thực thi kiểm tra mỗi assembly có quyền thực thi hay không, mà còn gián tiếp tạo ra việc phân giải chính sách đối với mỗi assembly được nạp, phủ nhận lợi ích của việc phân giải chính sách “lười”. Các yếu tố này có thể gây ra sự trì hoãn đáng kể khi các assembly được nạp, đặc biệt khi bộ thực thi nạp nhiều assembly cùng một lúc.

Trong nhiều trường hợp, việc cho phép mã lệnh nạp và chạy không phải là một nguy cơ đáng kể miễn là tất cả các hoạt động và các tài nguyên quan trọng khác đã được bảo vệ bằng *CAS* và bằng cơ chế bảo mật của hệ điều hành. Bộ thực thi *.NET* cho phép bạn vô hiệu việc tự động kiểm tra các quyền thực thi ngay bên trong mã lệnh, hay bằng công cụ *Caspol.exe*.

Khi bạn nhập lệnh `caspol -e off` hoặc `caspol -e on`, *Caspol.exe* sẽ xác lập thuộc tính `CheckExecutionRights` của lớp `SecurityManager`. Bạn có thể thực hiện công việc này bên trong mã lệnh như sau:

```
// Vô hiệu việc kiểm tra quyền thực thi.
System.Security.SecurityManager.CheckExecutionRights = false;

// Lưu cấu hình.
System.Security.SecurityManager.SavePolicy();

// Kích hoạt việc kiểm tra quyền thực thi.
System.Security.SecurityManager.CheckExecutionRights = true;

// Lưu cấu hình.
System.Security.SecurityManager.SavePolicy();
```

Để thay đổi giá trị của `CheckExecutionRights` thì mã lệnh của bạn phải có phần tử `ControlPolicy` của `SecurityPermission`. Thay đổi này sẽ có tác dụng với tiến trình hiện hành ngay tức thì, cho phép bạn nạp các assembly trong lúc thực thi mà bộ thực thi không phải

kiểm tra chúng có quyền thực thi hay không. Tuy nhiên, thay đổi này sẽ không ảnh hưởng đến các tiến trình hiện có. Do vậy, bạn phải lưu thay đổi vào *Windows Registry* (bằng phương thức *SavePolicy*) để nó có tác dụng với các tiến trình mới.

4. Bảo đảm bộ thực thi cấp cho assembly một số quyền nào đó

? Bạn cần bảo đảm bộ thực thi cấp cho assembly của bạn các quyền truy xuất mã lệnh (*code access permission*) mà các quyền này quyết định sự thành công trong hoạt động của ứng dụng.

X Sử dụng các yêu cầu quyền (*permission request*) để chỉ định các quyền truy xuất mã lệnh mà assembly cần phải có. Bạn khai báo các yêu cầu quyền bằng các đặc tính quyền truy xuất mã lệnh ở mức assembly.

Các yêu cầu quyền cho biết các quyền mà mã lệnh phải có thì mới có thể chạy được. Ví dụ, nếu bạn viết một trình xem phim sao cho người dùng có thể sử dụng nó để tải và xem phim từ một server, thật tai hại nếu chính sách bảo mật của người dùng không cho phép chương trình này mở một kết nối mạng đến server. Chương trình của bạn sẽ nạp và chạy, nhưng khi người dùng kết nối đến server để xem phim, chương trình sẽ crash với ngoại lệ *System.Security.SecurityException*. Giải pháp là đưa vào assembly yêu cầu quyền cần thiết để có thể mở một kết nối mạng đến server (*System.Net.WebPermission* hay *System.Net.SocketPermission*, tùy thuộc vào kiểu kết nối bạn cần mở).

Bộ thực thi thực hiện các yêu cầu quyền với nguyên tắc: khoan nạp mã lệnh, điều này tốt hơn là nạp mã lệnh và thất bại sau khi thực hiện một hành động mà nó không có quyền. Vì vậy, nếu sau quá trình phân giải chính sách bảo mật, bộ thực thi xác định được *grant-set* của assembly không thể đáp ứng các yêu cầu quyền của assembly, nó sẽ không nạp assembly và ném ngoại lệ *System.Security.Policy.PolicyException*.

Để khai báo một yêu cầu quyền, bạn phải sử dụng bản sao đặc tính (*attribute counterpart*) của quyền truy xuất mã lệnh bạn cần. Tất cả các quyền truy xuất mã lệnh đều có một bản sao đặc tính mà bạn có thể sử dụng để tạo ra các lệnh bảo mật khai báo (*declarative security statement*)—bao gồm các yêu cầu quyền. Ví dụ, bản sao đặc tính của *SocketPermission* là *SocketPermissionAttribute*, và bản sao đặc tính của *WebPermission* là *WebPermissionAttribute*—Tất cả các quyền và các bản sao đặc tính của chúng cùng theo quy ước đặt tên và là các thành viên của cùng không gian tên.

Ứng dụng *PermissionRequestExample* dưới đây có hai yêu cầu quyền: một cho *SocketPermission* và một cho *SecurityPermission*. Bạn cần nhớ các điều sau:

- Bạn phải khai báo yêu cầu quyền sau bất kỳ lệnh *using* mức trên nào nhưng trước bất kỳ khai báo kiểu hay không gian tên nào.
- Đặc tính phải nhắm đến assembly nên bạn phải thêm tiền tố *assembly:* vào tên đặc tính.
- Không cần thêm phần *Attribute* vào tên đặc tính—mặc dù bạn có thể thêm vào nếu muốn.
- Bạn phải chỉ định *SecurityAction.RequestMinimum* là đối số đầu tiên của đặc tính—giá trị này cho biết đây là một yêu cầu quyền.

- Bạn phải cấu hình đặc tính để mô tả quyền truy xuất mã lệnh mà bạn cần bằng các thuộc tính của đặc tính. Bạn hãy tham khảo tài liệu *.NET Framework SDK* để biết thêm chi tiết về các thuộc tính do mỗi đặc tính bảo mật truy xuất mã lệnh hiện thực.
- Các lệnh yêu cầu quyền không được kết thúc bằng dấu chấm phẩy (;).
- Để tạo nhiều yêu cầu quyền, chỉ cần thêm nhiều lệnh yêu cầu quyền như được trình bày trong ví dụ dưới đây:

```
using System.Net;

using System.Security.Permissions;

// Yêu cầu SocketPermission (cho phép mở một kết nối
// TCP đến host và port được chỉ định).
[assembly:SocketPermission(SecurityAction.RequestMinimum,
    Access = "Connect", Host = "www.fabrikam.com",
    Port = "3538", Transport = "Tcp")]

// Yêu cầu phân tử UnmanagedCode của SecurityPermission,
// (kiểm soát khả năng thực thi mã lệnh không-được-quản-lý).
[assembly:SecurityPermission(SecurityAction.RequestMinimum,
    UnmanagedCode = true)]

public class PermissionRequestExample {
    public static void Main() {
        // Làm gì đó...
    }
}
```

Nếu bạn thực thi ứng dụng `PermissionRequestExample` và chính sách bảo mật không cấp cho assembly này các quyền được yêu cầu, bạn sẽ nhận được ngoại lệ `PolicyException` như dưới đây và ứng dụng sẽ không thực thi. Khi sử dụng chính sách bảo mật mặc định, điều này sẽ xảy ra nếu bạn chạy assembly từ một mạng dùng chung vì các assembly được nạp từ Intranet không được cấp `SocketPermission`.

```
Unhandled Exception: System.Security.Policy.PolicyException:
Required permission cannot be acquired.
```

Khi bạn nạp một assembly từ bên trong mã lệnh (tự động hay bằng tay), và assembly này có chứa các yêu cầu quyền mà chính sách bảo mật không đáp ứng thì phương thức mà bạn sử dụng để nạp assembly sẽ ném ngoại lệ `PolicyException`, do đó bạn phải thụ lý sao cho phù hợp.

5.

Giới hạn các quyền được cấp cho assembly

- ?** Bạn cần hạn chế các quyền truy xuất mã lệnh được cấp cho assembly của bạn, đảm bảo người khác và phần mềm khác không thể biến mã lệnh của bạn thành một cơ chế mà thông qua đó để thực hiện các hành động nguy hiểm hay không mong muốn.
- X** Sử dụng các lệnh bảo mật khai báo (*declarative security statement*) để chỉ định các yêu cầu quyền tùy chọn (*optional permission request*) và các yêu cầu loại trừ quyền (*permission refusal request*) trong assembly của bạn. Các yêu cầu quyền tùy chọn định nghĩa tập tối đa các quyền mà bộ thực thi sẽ cấp cho assembly. Các yêu cầu loại trừ quyền chỉ định các quyền cụ thể mà bộ thực thi sẽ không cấp cho assembly.

Nếu quan tâm đến bảo mật, thật lý tưởng khi mã lệnh của bạn chỉ có các quyền truy xuất mã lệnh cần thiết để thực hiện chức năng của nó. Điều này giảm thiểu nguy cơ người khác và mã lệnh khác sử dụng mã lệnh của bạn để thực hiện các hành động nguy hiểm hay không mong muốn. Vấn đề là, bộ thực thi phân giải các quyền của một assembly bằng chính sách bảo mật (do một người dùng nào đó hay người quản trị cấu hình). Chính sách bảo mật có thể khác nhau tại mỗi nơi mà ứng dụng chạy, và bạn không thể kiểm soát các quyền mà chính sách bảo mật ấn định cho mã lệnh của bạn.

Mặc dù bạn không thể kiểm soát chính sách bảo mật tại tất cả các nơi mà mã lệnh chạy, nhưng .NET Framework cung cấp hai cơ chế mà thông qua đó, bạn có thể loại bỏ các quyền được cấp cho assembly của bạn: yêu cầu loại trừ (*refuse request*) và yêu cầu tùy chọn (*optional request*). Yêu cầu loại trừ cho phép bạn chỉ ra các quyền cụ thể mà bạn không muốn bộ thực thi cấp cho assembly. Sau quá trình phân giải chính sách, nếu grant-set của một assembly chứa bất kỳ quyền nào đã được chỉ định trong một yêu cầu loại trừ, bộ thực thi sẽ loại bỏ quyền đó. Yêu cầu tùy chọn định nghĩa tập tối đa các quyền mà bộ thực thi có thể cấp cho assembly. Nếu grant-set của một assembly chứa bất kỳ quyền nào khác với các quyền đã được chỉ định trong yêu cầu tùy chọn, bộ thực thi sẽ loại bỏ quyền đó. Khác với yêu cầu quyền tối thiểu (đã được thảo luận trong mục 13.4), bộ thực thi sẽ không từ chối nạp assembly của bạn nếu nó không thể cấp tất cả các quyền đã được chỉ định trong yêu cầu tùy chọn.

Bạn có thể coi yêu cầu loại trừ và yêu cầu tùy chọn là các cách chọn lựa để thu được cùng kết quả; cách mà bạn sử dụng tùy thuộc vào số lượng quyền bạn muốn loại bỏ. Nếu muốn loại bỏ một số ít quyền, bạn hãy chọn yêu cầu loại trừ. Tuy nhiên, nếu muốn loại bỏ nhiều quyền, bạn hãy chọn yêu cầu tùy chọn (yêu cầu này gồm một số quyền bạn cần, phần quyền còn lại sẽ tự động bị loại bỏ).

Bạn thêm các yêu cầu tùy chọn và yêu cầu loại trừ vào mã lệnh bằng các lệnh bảo mật khai báo với cú pháp giống như các yêu cầu quyền tối thiểu đã được thảo luận trong mục 13.4. Điểm khác biệt duy nhất là giá trị của `System.Security.Permissions.SecurityAction` mà bạn truyền cho phương thức khởi dựng của đặc tính quyền. Sử dụng `SecurityAction.RequestOptional` để khai báo một yêu cầu tùy chọn và `SecurityAction.RequestRefuse` để khai báo một yêu cầu loại trừ. Cũng giống như các yêu cầu quyền tối thiểu, bạn phải khai báo các yêu cầu tùy chọn và yêu cầu loại trừ là các đặc tính toàn cục bằng cách thêm tiền tố `assembly:` vào tên đặc tính. Ngoài ra, tất cả các yêu cầu phải

xuất hiện sau bất kỳ lệnh `using` mức trên nào nhưng trước bất kỳ khai báo kiểu hay không gian tên nào.

Ví dụ dưới đây minh họa một yêu cầu quyền tùy chọn cho tập quyền `Internet`. Đây là tập quyền được định nghĩa bởi chính sách bảo mật mặc định. Khi bộ thực thi nạp `OptionalRequestExample`, nó sẽ không cấp cho assembly này bất cứ quyền nào không nằm trong tập quyền `Internet` (tham khảo tài liệu *.NET Framework SDK* để biết thêm chi tiết về các quyền trong tập quyền `Internet`).

```
using System.Security.Permissions;

[assembly:PermissionSet(SecurityAction.RequestOptional, Name = "Internet")]

public class OptionalRequestExample {

    public static void Main() {

        // Làm gì đó...

    }

}
```

Trái với `OptionalRequestExample`, ví dụ dưới đây sử dụng một yêu cầu loại trừ để loại bỏ quyền `System.Security.Permissions.FileIOPermission` (mô tả quyền truy xuất ghi đối với ổ đĩa C):

```
using System.Security.Permissions;

[assembly:FileIOPermission(SecurityAction.RequestRefuse, Write = @"C:\")]

public class RefuseRequestExample {

    public static void Main() {

        // Làm gì đó...

    }

}
```

6. *Xem các yêu cầu quyền được tạo bởi một assembly*

?

Bạn cần xem các loại trừ và các yêu cầu quyền khai báo được tạo bên trong một assembly để có thể cấu hình chính sách bảo mật một cách phù hợp, hoặc

nắm được các mặt hạn chế của một thư viện mà bạn muốn gọi từ mã lệnh của bạn.

✕ Sử dụng công cụ *Permissions View* (*Permvview.exe*—được cấp cùng với *.NET Framework SDK*).

Để cấu hình chính sách bảo mật một cách phù hợp, bạn cần biết các nhu cầu quyền truy xuất mã lệnh (*code access permission requirements*) của các assembly mà bạn dự định sẽ chạy. Điều này đúng cho cả các thư viện và các assembly thực thi được truy xuất từ ứng dụng của bạn. Đối với thư viện, bạn cần biết assembly loại bỏ những quyền nào để không sử dụng thư viện này để thực hiện một hành động bị hạn chế (sẽ dẫn đến ngoại lệ `System.Security.SecurityException`).

Công cụ *Permvview.exe* cung cấp một cơ chế đơn giản mà thông qua đó, bạn có thể xem các yêu cầu quyền khai báo (*declarative permission request*) được tạo bên trong một assembly—bao gồm yêu cầu tối thiểu (*minimum request*), yêu cầu tùy chọn (*optional request*), và yêu cầu loại trừ (*refusal request*). Ví dụ, lớp dưới đây khai báo một cầu tối thiểu, một yêu cầu tùy chọn, và một yêu cầu loại trừ:

```
using System.Net;
using System.Security.Permissions;

// Yêu cầu quyền tối thiểu: SocketPermission.
[assembly:SocketPermission(SecurityAction.RequestMinimum,
    Unrestricted = true)]

// Yêu cầu quyền tùy chọn: SecurityPermission.
[assembly:SecurityPermission(SecurityAction.RequestOptional,
    Unrestricted = true)]

// Yêu cầu loại trừ quyền: FileIOPermission.
[assembly:SecurityPermission(SecurityAction.RequestRefuse,
    Unrestricted = true)]

public class PermissionViewExample {
    public static void Main() {
        // Làm gì đó...
    }
}
```

Thực thi lệnh `permview PermissionViewExample.exe` sẽ sinh ra kết xuất như sau. Mặc dù không mấy thân thiện nhưng bạn vẫn có thể đọc kết xuất để xác định các yêu cầu quyền được tạo bởi assembly. Mỗi kiểu trong ba kiểu yêu cầu quyền—tối thiểu, tùy chọn, và loại trừ—được liệt kê dưới một tiêu đề riêng và được kết cấu ở dạng *XML* của một đối tượng `System.Security.PermissionSet`.

Microsoft (R) .NET Framework Permission Request Viewer.
 Version 1.1.4322.510
 Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

minimal permission set:

```
<PermissionSet class="System.Security.PermissionSet" version="1">
  <IPermission class="System.Net.SocketPermission, System,
    Version=1.0.5000.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
    version="1" Unrestricted="true"/>
</PermissionSet>
```

optional permission set:

```
<PermissionSet class="System.Security.PermissionSet" version="1">
  <IPermission class="System.Security.Permissions.SecurityPermission,
    mscorlib, Version=1.0.5000.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" version="1" Unrestricted="true"/>
</PermissionSet>
```

refused permission set:

```
<PermissionSet class="System.Security.PermissionSet" version="1">
  <IPermission class="System.Security.Permissions.SecurityPermission,
    mscorlib, Version=1.0.5000.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" version="1" Unrestricted="true"/>
</PermissionSet>
```



Bằng cách chỉ định đối số `/decl` khi chạy tiện ích *Permview.exe*, bạn có thể xem tất cả các lệnh bảo mật khai báo (*declarative security statement*) nằm trong một assembly. Việc này cho thấy rõ những gì assembly thực hiện bên trong và cho phép bạn cấu hình chính sách bảo mật một cách phù hợp. Tuy nhiên, *Permview.exe* không hiển thị các thao tác bảo mật bắt buộc (*imperative security operation*) nằm trong một assembly. Hiện không có cách nào để trích và tổng kết các thao tác bảo mật bắt buộc được thực thi bên trong một assembly.

7. Xác định mã lệnh có quyền nào đó lúc thực thi hay không



Bạn cần xác định assembly của bạn có một quyền cụ thể nào đó lúc thực thi hay không.



Tạo và cấu hình quyền mà bạn muốn kiểm tra rồi truyền nó cho phương thức tĩnh `IsGranted` của lớp `System.Security.SecurityManager`.

Sử dụng yêu cầu quyền tối thiểu, bạn có thể bảo đảm rằng bộ thực thi cấp cho assembly tập các quyền đã được chỉ định; nếu mã lệnh của bạn đang chạy, bạn có thể chắc rằng nó có các quyền tối thiểu đã được yêu cầu. Tuy nhiên, bạn có thể muốn hiện thực một chức năng có tính “cơ hội” sao cho ứng dụng có thể cung cấp chức năng đó chỉ khi bộ thực thi cấp cho assembly của bạn các quyền phù hợp. Cách tiếp cận này được chính thức hóa một phần bằng các yêu cầu quyền tùy chọn, cho phép bạn định nghĩa một tập các quyền mà mã lệnh của bạn có thể tận dụng nếu chính sách bảo mật cấp chúng, nhưng lại không thiết yếu đối với sự vận hành thành công của mã lệnh (mục 13.5 đã trình bày chi tiết về các yêu cầu quyền tùy chọn).

Vấn đề đối với các yêu cầu quyền tùy chọn là bộ thực thi không có khả năng báo cho assembly rằng nó đã cấp những quyền nào. Bạn có thể sử dụng một thao tác được-bảo-vệ và thất bại một cách êm xuôi khi lời gọi này dẫn đến ngoại lệ `System.Security.SecurityException`. Tuy nhiên, sẽ hiệu quả hơn khi xác định xem bạn có các quyền cần thiết hay không. Sau đó, bạn có thể xây dựng logic vào trong mã lệnh để tránh gọi phải các thành viên được-bảo-vệ mà sẽ gây ra stack walk và dựng nên các ngoại lệ bảo mật. Đoạn mã dưới đây trình bày cách sử dụng phương thức `IsGranted` để xác định xem assembly hiện tại có quyền ghi vào thư mục `C:\Data` hay không. Bạn có thể tạo một lời gọi như thế mỗi khi bạn cần kiểm tra quyền, nhưng sẽ hiệu quả hơn nếu sử dụng một giá trị luận lý để thiết lập một cờ cấu hình cho biết có cho phép người dùng lưu file hay không.

```
// Định nghĩa một biến luận lý cho biết assembly có quyền
// truy xuất ghi đối với thư mục C:\Data hay không.
bool canWrite = false;

// Tạo và cấu hình một đối tượng FileIOPermission mô tả quyền
// truy xuất ghi đối với thư mục C:\Data.
System.Security.Permissions.FileIOPermission fileIOPerm =
    new System.Security.Permissions.FileIOPermission(
        System.Security.Permissions.FileIOPermissionAccess.Write,
        @"C:\Data");

// Kiểm tra assembly hiện tại có quyền đã được chỉ định hay không.
canWrite = System.Security.SecurityManager.IsGranted(fileIOPerm);
```

8.

Hạn chế ai đó thừa kế các lớp của bạn và chép đè các thành viên lớp



Bạn cần kiểm soát những ai có thể thừa kế các lớp của bạn thông qua sự thừa kế (*inheritance*) và một lớp dẫn xuất có thể chép đè những thành viên nào.



Sử dụng các lệnh bảo mật khai báo (*declarative security statement*) để áp dụng thành viên `SecurityAction.InheritanceDemand` vào phân khai báo của các lớp và các thành viên bạn cần bảo vệ.

Các modifier như `sealed`, `public`, `private`, và `virtual` cho phép bạn kiểm soát khả năng của các lớp khác khi các lớp này thừa kế lớp của bạn và chép đề các thành viên của nó. Tuy nhiên, các modifier này không linh hoạt, không có khả năng chọn lọc khi giới hạn những mã lệnh nào có thể thừa kế một lớp hoặc chép đề các thành viên của nó. Ví dụ, bạn muốn chỉ những mã lệnh do công ty hay khoa của bạn viết thì mới có thể thừa kế các lớp nghiệp vụ quan trọng, hoặc chỉ những mã lệnh được nạp từ máy cục bộ thì mới có thể thừa kế các phương thức nào đó. Bằng cách áp dụng `InheritanceDemand` vào khai báo lớp hay thành viên, bạn có thể chỉ định các quyền (lúc thực thi) mà một lớp phải có thì mới có thể thừa kế lớp của bạn hoặc chép đề các thành viên cụ thể nào đó. Nhớ rằng, các quyền của một lớp là các quyền của assembly mà lớp này được khai báo trong đó.

Mặc dù bạn có thể yêu cầu bất kỳ quyền hay tập quyền nào trong `InheritanceDemand`, nhưng thông thường là yêu cầu các quyền định danh (*identity permission*). Quyền định danh mô tả chứng cứ (*evidence*) do một assembly đưa cho bộ thực thi. Nếu một assembly đưa ra các kiểu chứng cứ nào đó lúc nạp, bộ thực thi sẽ tự động gán cho assembly này quyền định danh phù hợp. Quyền định danh cho phép bạn sử dụng các lệnh bảo mật bắt buộc và khai báo (*imperative* và *declarative security statement*) để trực tiếp ra các quyết định bảo mật (*security decision*) căn cứ trên định danh mã (*code identity*) mà không cần trực tiếp đánh giá các đối tượng chứng cứ. Bảng 13-1 liệt kê kiểu quyền định danh được tạo cho mỗi kiểu chứng cứ (kiểu chứng cứ là thành viên của không gian tên `System.Security.Policy`, kiểu quyền định danh là thành viên của không gian tên `System.Security.Permissions`).

Bảng 13.1 Các lớp chứng cứ và các quyền định danh

Lớp chứng cứ	Quyền định danh
<code>ApplicationDirectory</code>	Không
<code>Hash</code>	Không
<code>Publisher</code>	<code>PublisherIdentityPermission</code>
<code>Site</code>	<code>SiteIdentityPermission</code>
<code>StrongName</code>	<code>StrongNameIdentityPermission</code>
<code>Url</code>	<code>UrlIdentityPermission</code>
<code>Zone</code>	<code>ZoneIdentityPermission</code>



Bộ thực thi gán các quyền định danh cho một assembly dựa trên chứng cứ do assembly này đưa ra. Bạn không thể gán thêm quyền định danh cho một assembly thông qua việc cấu hình chính sách bảo mật.

Bạn phải sử dụng cú pháp bảo mật khai báo (*declarative security syntax*) để hiện thực một `InheritanceDemand`, nên bạn phải sử dụng bản sao đặc tính (*attribute counterpart*) của lớp quyền mà bạn muốn yêu cầu. Tất cả các lớp quyền đều có một bản sao đặc tính để tạo các

lệnh bảo mật khai báo—bao gồm `InheritanceDemand`. Ví dụ, bản sao đặc tính của `PublisherIdentityPermission` là `PublisherIdentityPermissionAttribute`, và bản sao đặc tính của `StrongNameIdentityPermission` là `StrongNameIdentityPermissionAttribute`—tất cả các quyền và các bản sao đặc tính của chúng cùng theo quy ước đặt tên và là các thành viên của cùng không gian tên.

Để kiểm soát những mã lệnh nào có thể thừa kế lớp của bạn, hãy áp dụng `InheritanceDemand` khi khai báo lớp. Đoạn mã dưới đây trình bày một lớp được bảo vệ bằng `InheritanceDemand`. Theo đó, chỉ những lớp bên trong các assembly được ký bởi publisher-certificate (nằm trong file `pubcert.cer`) thì mới có thể thừa kế lớp `InheritanceDemandExample`. Nội dung của file `pubcert.cer` được đọc lúc biên dịch, và các thông tin chứng thực cần thiết được gắn vào assembly.

```
[PublisherIdentityPermission(SecurityAction.InheritanceDemand,
    CertFile = @"I:\CSharp\Chuong13\pubcert.cer")]
public class InheritanceDemandExample {
    §
}
```

Để kiểm soát những mã lệnh nào có thể chép đè các thành viên nào đó, bạn hãy áp dụng `InheritanceDemand` khi khai báo thành viên. Xét đoạn mã dưới đây, chỉ những lớp được cấp tập quyền `FullTrust` thì mới có thể chép đè phương thức `SomeProtectedMethod`.

```
[PermissionSet(SecurityAction.InheritanceDemand, Name="FullTrust")]
public void SomeProtectedMethod () {
    §
}
```

9.

Kiểm tra chứng cứ của một assembly



Bạn cần kiểm tra chứng cứ mà bộ thực thi đã gán cho một assembly.



Thu lấy đối tượng `System.Reflection.Assembly` mô tả assembly mà bạn quan tâm. Lấy tập hợp `System.Security.Policy.Evidence` từ thuộc tính `Evidence` của đối tượng `Assembly`, rồi truy xuất các đối tượng chứng cứ bên trong bằng phương thức `GetEnumerator`, `GetEnumerator`, hay `GetAssemblyEnumerator` của lớp `Evidence`.

Lớp `Evidence` mô tả một tập hợp các đối tượng chứng cứ. Thuộc tính chỉ-đọc `Evidence` của lớp `Assembly` trả về một đối tượng tập hợp `Evidence` chứa tất cả các đối tượng chứng cứ mà bộ thực thi đã gán cho assembly khi assembly này được nạp.

Thật ra, lớp `Evidence` chứa hai tập hợp, mô tả hai kiểu chứng cứ khác nhau: chứng cứ host và chứng cứ assembly. Chứng cứ host bao gồm các đối tượng chứng cứ được gán cho assembly bởi bộ thực thi hay mã lệnh đã nạp assembly (mã lệnh này là đáng tin cậy). Chứng cứ assembly mô tả các đối tượng chứng cứ tùy biến được nhúng vào assembly lúc tạo dựng. Lớp `Evidence` hiện thực ba phương thức sau đây để liệt kê các đối tượng chứng cứ bên trong:

- `GetEnumerator`

- GetEnumerator
- GetAssemblyEnumerator

Phương thức GetEnumerator trả về một System.Collections.IEnumerator dùng để liệt kê tất cả các đối tượng chứng cứ bên trong tập hợp Evidence. Phương thức GetEnumerator và GetAssemblyEnumerator trả về một thể hiện IEnumerator liệt kê chỉ các đối tượng chứng cứ từ tập hợp tương ứng.

Ví dụ dưới đây trình bày cách hiển thị chứng cứ host và chứng cứ assembly của một assembly. Chú ý rằng, tất cả các lớp chứng cứ chuẩn đều chép đề phương thức Object.ToString để biểu diễn trạng thái của đối tượng chứng cứ. Mặc dù có liên quan nhưng không phải lúc nào ví dụ này cũng hiển thị chứng cứ mà một assembly sẽ có khi được nạp từ bên trong chương trình của bạn. Runtime host (như *Microsoft ASP.NET* hay *Microsoft Internet Explorer runtime host*) có thể tự ý gán thêm chứng cứ host khi nó nạp một assembly.

```
using System;
using System.Reflection;
using System.Collections;
using System.Security.Policy;

public class ViewEvidenceExample {
    public static void Main(string[] args) {

        // Nạp assembly đã được chỉ định.
        Assembly a = Assembly.LoadFrom(args[0]);

        // Thu lấy tập hợp Evidence từ assembly đã được nạp.
        Evidence e = a.Evidence;

        // Hiển thị chứng cứ host.
        IEnumerator x = e.GetEnumerator();
        Console.WriteLine("HOST EVIDENCE COLLECTION:");
        while(x.MoveNext()) {
            Console.WriteLine(x.Current.ToString());
        }

        // Hiển thị chứng cứ assembly.
        x = e.GetAssemblyEnumerator();
        Console.WriteLine("ASSEMBLY EVIDENCE COLLECTION:");
        while(x.MoveNext()) {
```

```

        Console.WriteLine(x.Current.ToString());
    }
}
}

```

Tất cả các lớp chứng cứ chuẩn do *.NET Framework* cấp đều bất biến, bạn không thể thay đổi các giá trị của chúng sau khi bộ thực thi đã tạo ra rồi gán chúng cho assembly. Ngoài ra, bạn không thể thêm hay loại bỏ các item trong lúc đang liệt kê một tập hợp bằng `IEnumerator`, nếu không, phương thức `MoveNext` sẽ ném ngoại lệ `System.InvalidOperationException`.

10.

Xử lý chứng cứ khi nạp một assembly



Bạn cần xử lý chứng cứ khi nạp một assembly để tác động đến các quyền mà bộ thực thi cấp cho assembly.



Tạo các đối tượng chứng cứ mà bạn muốn gán cho assembly, sau đó thêm chúng vào một thể hiện của lớp `System.Security.Policy.Evidence`, rồi truyền tập hợp `Evidence` cho phương thức dùng để nạp assembly.

Chứng cứ (được chiếm hữu bởi một assembly) định nghĩa định danh (*identity*) của assembly và xác định bộ thực thi cấp các quyền nào cho assembly. Bộ nạp assembly (*Assembly Loader*) chịu trách nhiệm chính trong việc xác định gán chứng cứ gì cho một assembly, nhưng một host đáng tin cậy (như *ASP.NET* hay *Internet Explorer runtime host*) cũng có thể gán chứng cứ cho một assembly. Mã lệnh của bạn có thể gán chứng cứ khi nạp một assembly nếu mã lệnh này có phần tử `ControlEvidence` của `SecurityPermission`.



Nếu bạn nạp một assembly vào một miền ứng dụng hai lần nhưng gán chứng cứ khác vào assembly này mỗi lần như thế, bộ thực thi sẽ ném ngoại lệ `System.IO.FileLoadException`.

Có nhiều phương thức thực hiện việc gán chứng cứ khi nạp một assembly. Một đặc điểm mà tất cả các phương thức này thường có là nhận một tập hợp `Evidence` làm đối số—lớp `Evidence` là một bộ chứa cho các đối tượng chứng cứ. Bạn phải đặt từng đối tượng chứng cứ mà bạn muốn gán cho assembly vào một tập hợp `Evidence` và truyền nó cho phương thức nạp assembly. Nếu bạn gán chứng cứ mới xung đột với chứng cứ được gán bởi bộ nạp assembly, chứng cứ mới sẽ thay thế chứng cứ cũ. Bảng 13-2 liệt kê các lớp và các phương thức của chúng trực tiếp hay gián tiếp nạp một assembly. Mỗi phương thức cung cấp một hay nhiều phiên bản nạp chồng chấp nhận một tập hợp `Evidence`.

Bảng 13.2 Các lớp và các phương thức của chúng cho phép bạn gán chứng cứ cho một assembly

Lớp/Phương thức	Mô tả
Lớp <code>System.Activator</code>	Các phương thức này ảnh hưởng đến miền ứng dụng hiện hành.
<code>CreateInstance</code>	Tạo một kiểu trong miền ứng dụng hiện hành từ assembly được chỉ định.
<code>CreateInstanceFrom</code>	

Lớp <code>System.AppDomain</code>	Các phương thức này ảnh hưởng đến miền ứng dụng được mô tả bởi đối tượng <code>AppDomain</code> (phương thức được gọi trên đó).
<code>CreateInstance</code> <code>CreateInstanceAndUnwrap</code> <code>CreateInstanceFrom</code> <code>CreateInstanceFromAndUnwrap</code>	Tạo một kiểu từ assembly được chỉ định.
<code>DefineDynamicAssembly</code>	Tạo một đối tượng <code>System.Reflection.Emit.AssemblyBuilder</code> , bạn có thể sử dụng nó để tạo động một assembly trong bộ nhớ.
<code>ExecuteAssembly</code>	Nạp và thực thi một assembly có điểm nhập đã được định nghĩa (phương thức <code>Main</code>).
<code>Load</code>	Nạp assembly được chỉ định.
Lớp <code>System.Reflection.Assembly</code>	Các phương thức này ảnh hưởng đến miền ứng dụng hiện hành.
<code>Load</code> <code>LoadFile</code> <code>LoadFrom</code> <code>LoadWithPartialName</code>	Nạp assembly được chỉ định.

Đoạn mã dưới đây trình bày cách sử dụng phương thức `Assembly.Load` để nạp một assembly vào miền ứng dụng hiện hành. Trước khi gọi `Load`, đoạn mã này tạo một tập hợp `Evidence` và sử dụng phương thức `AddHost` của nó để thêm các đối tượng chứng cứ `Site` và `Zone` (các thành viên của không gian tên `System.Security.Policy`).

```
// Tạo các đối tượng chứng cứ Site và Zone mới.
System.Security.Policy.Site siteEvidence = new
    System.Security.Policy.Site("www.microsoft.com");
System.Security.Policy.Zone zoneEvidence = new
    System.Security.Policy.Zone(System.Security.SecurityZone.Trusted);

// Tạo một tập hợp Evidence mới.
System.Security.Policy.Evidence evidence =
    new System.Security.Policy.Evidence();

// Thêm các đối tượng chứng cứ Site và Zone vào tập hợp Evidence
// bằng phương thức AddHost.
```



```
evidence.AddHost(siteEvidence);
evidence.AddHost(zoneEvidence);

// Nạp assembly có tên là "SomeAssembly" và gán các đối tượng Site và
// Zone cho nó. Các đối tượng này sẽ chép đè các đối tượng Site và Zone
// do bộ nạp assembly gán.
System.Reflection.Assembly assembly =
    System.Reflection.Assembly.Load("SomeAssembly", evidence);
```

11. Xử lý bảo mật bộ thực thi bằng chứng cứ của miền ứng dụng

- ?** Bạn cần buộc một giới hạn trên (*upper limit*) lên các quyền đang có hiệu lực với tất cả các assembly được nạp vào một miền ứng dụng cụ thể.
- X** Cấu hình chính sách bảo mật để cấp các quyền phù hợp dựa trên chứng cứ mà bạn dự định gán cho miền ứng dụng. Khi tạo miền ứng dụng bằng phương thức tĩnh `CreateDomain` của lớp `System.AppDomain`, bạn hãy cung cấp một tập hợp `System.Security.Policy.Evidence` chứa các đối tượng chứng cứ của miền ứng dụng. Sau đó, nạp các assembly mà bạn muốn giới hạn các quyền của chúng bên trong miền ứng dụng này.

Đúng như bộ thực thi gán quyền cho assembly dựa trên chứng cứ mà assembly này đưa ra lúc nạp, bộ thực thi cũng gán quyền cho các miền ứng dụng dựa trên chứng cứ của chúng. Bộ thực thi không gán chứng cứ cho các miền ứng dụng theo cách như nó gán cho các assembly vì không có gì để chứng cứ đó tựa vào. Thay vào đó, mã lệnh tạo miền ứng dụng phải gán chứng cứ khi cần.

- ✎** Bộ thực thi chỉ sử dụng các mức chính sách công ty (*enterprise*), máy (*machine*), và người dùng (*user*) để tính các quyền của một miền ứng dụng; các chính sách bảo mật của các miền ứng dụng hiện có không giữ vai trò gì cả. Mục 13.12 sẽ thảo luận chính sách bảo mật miền ứng dụng.

Các miền ứng dụng không có chứng cứ là trong suốt đối với các cơ chế bảo mật truy xuất mã lệnh của bộ thực thi. Các miền ứng dụng được gán chứng cứ có một grant-set dựa trên chính sách bảo mật và đóng một vai trò quan trọng trong việc phân giải các yêu cầu bảo mật CAS. Khi quá trình thực thi ứng dụng xuyên qua một biên miền ứng dụng, bộ thực thi ghi lại việc chuyển tiếp trên call stack. Khi một yêu cầu bảo mật gây nên một stack walk, bản ghi chuyển tiếp miền ứng dụng được xử lý giống như các bản ghi stack khác—bộ thực thi đánh giá grant-set kết giao với bản ghi stack để bảo đảm nó chứa các quyền được yêu cầu. Điều này nghĩa là các quyền của một miền ứng dụng ảnh hưởng đến tất cả mã lệnh được nạp vào miền ứng dụng. Thực tế, miền ứng dụng thiết lập một giới hạn trên lên các khả năng của tất cả mã lệnh được nạp vào nó.

Một ví dụ quan trọng trong sử dụng chứng cứ miền ứng dụng là *Microsoft Internet Explorer*. *Internet Explorer* tạo một miền ứng dụng cho mỗi site mà nó download các điều kiểm được-quản-lý từ đó. Tất cả các điều kiểm được download từ một site cho trước—cũng như các assembly mà chúng nạp—chạy trong cùng miền ứng dụng. Khi *Internet Explorer* tạo miền

ứng dụng cho một site, nó gán chứng cứ `System.Security.Policy.Site` cho miền ứng dụng này. Điều này bảo đảm rằng, nếu các điều kiểm được download nạp một assembly (ngay cả từ đĩa cục bộ), các hành động của assembly này bị ràng buộc bởi các quyền được cấp cho miền ứng dụng dựa trên chứng cứ `Site` và chính sách bảo mật.



Trừ khi bạn gán chứng cứ cho miền ứng dụng một cách tường minh khi tạo nó, miền ứng dụng này không ảnh hưởng gì đến các yêu cầu bảo mật (*security demand*).

Để gán chứng cứ cho một miền ứng dụng, bạn hãy tạo một tập hợp `Evidence` và thêm các đối tượng chứng cứ cần thiết vào đó bằng phương thức `Evidence.AddHost`. Khi tạo miền ứng dụng mới, bạn truyền tập hợp `Evidence` cho một trong các phiên bản nạp chồng của phương thức tĩnh `CreateDomain`. Quá trình phân giải chính sách thường kỳ của bộ thực thi sẽ xác định `grant-set` của miền ứng dụng.

Ứng dụng dưới đây trình bày cách gán chứng cứ cho một miền ứng dụng. Trong đó, ứng dụng nạp mã lệnh từ một publisher cụ thể vào một miền ứng dụng publisher cụ thể. Bằng cách gán cho miền ứng dụng chứng cứ `System.Security.Policy.Publisher` mô tả publisher của phần mềm, ví dụ này hạn chế các khả năng của mã lệnh được nạp vào miền ứng dụng. Sử dụng chính sách bảo mật, bạn có thể gán mã lệnh của publisher một tập quyền cực đại tương xứng với mức tin cậy bạn đặt vào publisher.

```
using System;
using System.Security.Policy;
using System.Security.Cryptography.X509Certificates;

public class AppDomainEvidenceExample {

    public static void Main() {

        // Tạo một miền ứng dụng mới cho mỗi publisher mà ứng dụng này
        // sẽ nạp mã lệnh của nó. Truyền cho phương thức CreateAppDomain
        // tên công ty, và tên của file chứa chứng chỉ X.509v3
        // của công ty này.
        AppDomain appDom1 = CreateAppDomain("Litware", "litware.cer");
        AppDomain appDom2 = CreateAppDomain("Fabrikam", "fabrikam.cer");

        // Nạp mã lệnh từ các publisher vào miền ứng dụng phù hợp
        // để thực thi.
    }
}
```

```
// Phương thức này tạo một miền ứng dụng mới để nạp và chạy mã lệnh
// trong đó từ một publisher cụ thể. Đối số name chỉ định tên của
// miền ứng dụng. Đối số certFile chỉ định tên của file chứa
// một chứng chỉ X.509v3 cho publisher mà mã lệnh của nó sẽ
// được chạy trong miền ứng dụng mới.
private static AppDomain CreateAppDomain(string name,
    string certFile){

    // Tạo một đối tượng X509Certificate mới từ chứng chỉ X.509v3
    // nằm trong file được chỉ định.
    X509Certificate cert =
        X509Certificate.CreateFromCertFile(certFile);

    // Tạo chứng cứ Publisher mới từ đối tượng X509Certificate.
    Publisher publisherEvidence = new Publisher(cert);

    // Tạo một tập hợp Evidence mới.
    Evidence evidence = new Evidence();

    // Thêm chứng cứ Publisher vào tập hợp Evidence.
    evidence.AddHost(publisherEvidence);

    // Tạo một miền ứng dụng mới với tập hợp Evidence
    // chứa chứng cứ Publisher và trả về miền ứng dụng
    // vừa được tạo ra.
    return AppDomain.CreateDomain(name, evidence);
}
}
```

12.

Xử lý bảo mật bộ thực thi bằng chính sách bảo mật của miền ứng dụng

?

Bạn cần kiểm soát (bằng mã lệnh) các quyền được cấp cho các assembly.

✕

Cấu hình (bằng mã lệnh) chính sách bảo mật của miền ứng dụng mà bạn đã nạp các assembly vào đó.

Chính sách bảo mật (*security policy*) bao gồm bốn mức chính sách: công ty (*enterprise*), máy (*machine*), người dùng (*user*), và miền ứng dụng (*application domain*). Bộ thực thi quyết định những quyền nào để cấp cho một assembly bằng cách xác định tập quyền được cấp bởi mỗi mức chính sách rồi tính phép giao (phép *AND* luận lý) của bốn tập quyền. Các quyền nằm trong tập giao là grant-set cuối cùng của assembly.

 **Ngay cả khi các mức chính sách công ty, máy, hay người dùng chỉ định code group `LevelFinal` (chỉ thị bộ thực thi không đánh giá các mức chính sách thấp hơn), bộ thực thi luôn sử dụng mức chính sách của miền ứng dụng để tính grant-set của một assembly.**

Chỉ các mức chính sách công ty, máy, và người dùng là được cấu hình tĩnh bằng *Administrative Tools*. Vì miền ứng dụng không tồn tại bên ngoài ngữ cảnh của bộ thực thi nên không thể cấu hình tĩnh chính sách miền ứng dụng được. Để cấu hình chính sách bảo mật của một miền ứng dụng, bạn phải tạo một mức chính sách (bằng mã lệnh) rồi gán nó cho miền ứng dụng.

Xây dựng một mức chính sách bằng mã lệnh có thể là một công việc lâu dài, tùy thuộc vào độ phức tạp của chính sách bảo mật mà bạn cần diễn tả. Lớp `System.Security.Policy.PolicyLevel` mô tả một mức chính sách bảo mật. Bên trong mỗi đối tượng `PolicyLevel`, bạn phải tạo dựng một cấu trúc phân cấp các code group, định nghĩa các điều kiện thành viên, các tập quyền, và các đặc tính của mỗi code group. Có rất nhiều kiểu được sử dụng để tạo dựng mức chính sách, và thảo luận về các kiểu này là vượt quá phạm vi quyển sách này. Lập trình bảo mật *.NET*, như đã đề cập trước đây, cung cấp thông tin chi tiết về mỗi lớp này và trình bày cách xây dựng một mức chính sách bằng mã lệnh.

 **Thông thường, bạn sẽ phát triển một công cụ trợ giúp việc tạo một mức chính sách và ghi định nghĩa mức chính sách ra file; sau đó, bạn có thể nạp định nghĩa này từ đĩa khi cần. Lớp `PolicyLevel` có hai phương thức nhằm đơn giản hóa quá trình này: `ToXml` đưa một đối tượng `PolicyLevel` về dạng để lưu trữ, và `FromXml` xây dựng lại một đối tượng `PolicyLevel` từ dạng đã được lưu trữ.**

Một khi đã có đối tượng `PolicyLevel` mô tả chính sách bảo mật như mong muốn, bạn có thể gán nó cho một miền ứng dụng. Thu lấy tham chiếu `System.AppDomain` đến miền ứng dụng mà bạn muốn cấu hình, và truyền đối tượng `PolicyLevel` cho phương thức `AppDomain.SetAppDomainPolicy`. Mã lệnh của bạn phải có phần tử `ControlDomainPolicy` của `SecurityPermission` thì mới có thể gọi `SetAppDomainPolicy`. Bạn chỉ có thể gọi `SetAppDomainPolicy` một lần trên mỗi miền ứng dụng; nếu bạn gọi `SetAppDomainPolicy` lần thứ hai, nó sẽ ném ngoại lệ `System.Security.Policy.PolicyException`.

Bạn không phải gán đối tượng `PolicyLevel` cho một miền ứng dụng trước khi nạp các assembly vào miền ứng dụng này. Các assembly được nạp trước khi bạn thiết lập chính sách bảo mật miền ứng dụng có các grant-set chỉ dựa trên các mức chính sách: công ty, máy, và người dùng. Chính sách miền ứng dụng chỉ áp dụng cho các assembly được nạp sau khi nó được cấu hình. Thông thường, bạn sử dụng khả năng này để nạp các assembly chia sẻ đáng-tin-cậy vào miền ứng dụng mà không bị ràng buộc bởi chính sách miền ứng dụng.

Ứng dụng dưới đây trình bày quá trình tạo một mức chính sách và gán nó cho một miền ứng dụng. Mức chính sách này cấp các quyền dựa trên publisher của một assembly—được thể hiện trong các khoản của chứng cứ `System.Security.Policy.Publisher`.

```
using System;
using System.Security;
using System.Security.Policy;
using System.Security.Cryptography.X509Certificates;

public class AppDomainPolicyExample {

    public static void Main() {

        // Tạo một miền ứng dụng mới (để nạp các assembly vào đó).
        AppDomain domain = AppDomain.CreateDomain("modules");

        // Nạp các assembly vào miền ứng dụng mà bạn không muốn
        // bị ràng buộc bởi chính sách bảo mật miền ứng dụng.
        $

        // Cấu hình chính sách bảo mật cho đối tượng AppDomain.
        SetDomainPolicy(domain);

        // Nạp các assembly vào miền ứng dụng được bảo vệ.
        $
    }

    // Phương thức này cấu hình chính sách bảo mật của đối tượng
    // AppDomain được truyền cho nó. Chính sách bảo mật sẽ gán các
    // quyền khác nhau cho mỗi assembly dựa trên publisher của
    // assembly. Những assembly từ các publisher vô danh không được
    // cấp quyền nào cả.
    private static void SetDomainPolicy(AppDomain domain) {

        // Tạo một PolicyLevel mới cho miền ứng dụng.
        PolicyLevel policy = PolicyLevel.CreateAppDomainLevel();

        // Tạo một FirstMatchCodeGroup mới dùng làm nút gốc của hệ
        // thống phân cấp code group. Cấu hình group này sao cho
```

```
// nó trùng khớp với tất cả code. Điều này nghĩa là tất cả
// các assembly đều khởi đầu với một grant-set rỗng cho
// mức chính sách miền ứng dụng.
policy.RootCodeGroup = new FirstMatchCodeGroup(
    new AllMembershipCondition(),
    new PolicyStatement(policy.GetNamedPermissionSet("Nothing"))
);

// Tạo tập các code group để xác định các quyền nào sẽ được cấp
// cho một assembly do một publisher tạo ra. Vì code group gốc
// là một FirstMatchCodeGroup, nên quá trình phân giải chính sách
// chỉ so trùng assembly trên các group con cho đến khi tìm thấy
// trùng khớp đầu tiên. Mỗi code group được tạo với đặc tính
// Exclusive để bảo đảm rằng assembly không lấy thêm quyền
// nào từ các code group khác.

// Tạo code group cấp tập quyền FullTrust cho các
// assembly được phát hành bởi Microsoft.
X509Certificate microsoftCert =
    X509Certificate.CreateFromCertFile("microsoft.cer");

policy.RootCodeGroup.AddChild(new UnionCodeGroup(
    new PublisherMembershipCondition(microsoftCert),
    new PolicyStatement(policy.GetNamedPermissionSet("FullTrust"),
        PolicyStatementAttribute.Exclusive)
));

// Tạo code group cấp tập quyền Internet cho các
// assembly được phát hành bởi Litware, Inc.
X509Certificate litwareCert =
    X509Certificate.CreateFromCertFile("litware.cer");

policy.RootCodeGroup.AddChild(new UnionCodeGroup(
    new PublisherMembershipCondition(litwareCert),
    new PolicyStatement(policy.GetNamedPermissionSet("Internet"),
        PolicyStatementAttribute.Exclusive)
```

```

));

// Tạo code group cấp tập quyền Execution cho các
// assembly được phát hành bởi Fabrikam, Inc.
X509Certificate fabrikamCert =
    X509Certificate.CreateFromCertFile("fabrikam.cer");

policy.RootCodeGroup.AddChild(new UnionCodeGroup(
    new PublisherMembershipCondition(fabrikamCert),
    new PolicyStatement(policy.GetNamedPermissionSet("Execution"),
        PolicyStatementAttribute.Exclusive)
));

// Thêm một code group sau cùng để bắt tất cả các assembly
// không khớp với publisher nào. Gán group này với tập quyền
// Nothing. Vì group này là Exclusive nên assembly sẽ không
// nhận quyền nào từ các group khác, ngay cả từ các
// mức chính sách cao hơn (công ty, máy, và người dùng).
policy.RootCodeGroup.AddChild(new UnionCodeGroup(
    new AllMembershipCondition(),
    new PolicyStatement(policy.GetNamedPermissionSet("Nothing"),
        PolicyStatementAttribute.Exclusive)
));

// Gán chính sách cho miền ứng dụng.
domain.SetAppDomainPolicy(policy);
}
}

```

13. *Xác định người dùng hiện hành có là thành viên của một nhóm Windows nào đó hay không*

? Bạn cần xác định người dùng hiện hành của ứng dụng có phải là thành viên của một nhóm người dùng Windows nào đó hay không.

✕ Thu lấy đối tượng `System.Security.Principal.WindowsIdentity` mô tả người dùng hiện hành bằng phương thức tĩnh `WindowsIdentity.GetCurrent`. Kế tiếp, truyền đối tượng `WindowsIdentity` cho phương thức khởi dựng của lớp `System.Security.Principal.WindowsPrincipal` để thu lấy đối tượng

WindowsPrincipal. Cuối cùng, gọi phương thức `IsInRole` của đối tượng `WindowsPrincipal` để xác định người dùng này có nằm trong một nhóm *Windows* nào đó hay không.

Cơ chế *RBS* của *.NET Framework* trừu tượng hóa các tính năng bảo mật dựa-trên-người-dùng của hệ điều hành nằm dưới thông qua hai giao diện chính sau đây:

- `System.Security.Principal.IIdentity`
- `System.Security.Principal.IPrincipal`

Giao diện `IIdentity` mô tả thực thể mà mã lệnh hiện đang chạy trên danh nghĩa của thực thể này, chẳng hạn một người dùng hoặc tài khoản dịch vụ (*service account*). Giao diện `IPrincipal` mô tả `IIdentity` của thực thể và tập các vai trò mà thực thể này thuộc về. Một vai trò chỉ là một sự phân loại, dùng để nhóm các thực thể với các khả năng bảo mật tương tự nhau, chẳng hạn một nhóm người dùng *Windows*.

Để tích hợp *RBS* với bảo mật người dùng *Windows*, *.NET Framework* cung cấp hai lớp sau đây (hiện thực giao diện `IIdentity` và `IPrincipal`):

- `System.Security.Principal.WindowsIdentity`
- `System.Security.Principal.WindowsPrincipal`

Lớp `WindowsIdentity` hiện thực giao diện `IIdentity` và mô tả một người dùng *Windows*. Lớp `WindowsPrincipal` hiện thực `IPrincipal` và mô tả tập các nhóm *Windows* mà người dùng này thuộc về. Vì *.NET RBS* là một giải pháp chung được thiết kế sao cho độc lập nền, bạn không thể truy xuất các tính năng và các khả năng của tài khoản người dùng *Windows* thông qua giao diện `IIdentity` và `IPrincipal`, bạn phải sử dụng trực tiếp các đối tượng `WindowsIdentity` và `WindowsPrincipal`.

Để xác định người dùng hiện hành có phải là thành viên của một nhóm *Windows* nào đó hay không, trước tiên bạn phải gọi phương thức tĩnh `WindowsIdentity.GetCurrent`. Phương thức này trả về một đối tượng `WindowsIdentity` mô tả người dùng *Windows* mà tiểu trình hiện đang chạy trên danh nghĩa của người dùng này. Kế tiếp, tạo một đối tượng `WindowsPrincipal` mới và truyền đối tượng `WindowsIdentity` cho phương thức khởi dựng. Cuối cùng, gọi phương thức `IsInRole` của đối tượng `WindowsPrincipal` để kiểm tra xem người dùng này có nằm trong một nhóm (vai trò) cụ thể nào đó hay không. `IsInRole` trả về `true` nếu người dùng này là thành viên của nhóm đã được chỉ định, ngược lại trả về `false`.



Bạn có thể thu lấy tham chiếu `IPrincipal` đến một đối tượng `WindowsPrincipal` bằng thuộc tính tĩnh `CurrentPrincipal` của lớp `System.Threading.Thread`. Tuy nhiên, kỹ thuật này tùy thuộc vào cấu hình chính sách *principal* của miền ứng dụng hiện hành; mục 13.14 sẽ thảo luận vấn đề này chi tiết hơn.

Phương thức `IsInRole` có ba phiên bản nạp chồng:

- Phiên bản thứ nhất nhận một chuỗi chứa tên của nhóm cần kiểm tra. Tên nhóm phải có dạng `[DomainName]\[GroupName]` đối với các nhóm dựa-trên-miền và phải có dạng `[MachineName]\[GroupName]` đối với các nhóm được định nghĩa cục bộ. Nếu muốn kiểm

tra tư cách thành viên của một nhóm *Windows* chuẩn, bạn hãy sử dụng dạng `BUILTIN\[GroupName].IsInRole` thực hiện kiểm tra có phân biệt chữ hoa-thường đối với tên nhóm được chỉ định.

- Phiên bản thứ hai nhận một số nguyên (`int`), số này chỉ định một *Windows Role Identifier (RID)*. *RID* cung cấp một cơ chế để nhận biết các nhóm, không phụ thuộc vào ngôn ngữ (*language*) và sự bản địa hóa (*localization*).
- Phiên bản thứ ba nhận một thành viên thuộc kiểu liệt kê `System.Security.Principal.WindowsBuiltInRole`. Kiểu liệt kê này định nghĩa các thành viên mô tả các nhóm *Windows* có sẵn. Bảng 13.3 liệt kê tên, *RID*, và giá trị `WindowsBuiltInRole` cho mỗi nhóm *Windows* chuẩn.

Bảng 13.3 Tên, RID, và giá trị `WindowsBuiltInRole` của các tài khoản có sẵn

Tên tài khoản	RID (Hex)	Giá trị <code>WindowsBuiltInRole</code>
<i>BUILTIN\Account Operators</i>	<i>0x224</i>	<i>AccountOperator</i>
<i>BUILTIN\Administrators</i>	<i>0x220</i>	<i>Administrator</i>
<i>BUILTIN\Backup Operators</i>	<i>0x227</i>	<i>BackupOperator</i>
<i>BUILTIN\Guests</i>	<i>0x222</i>	<i>Guest</i>
<i>BUILTIN\Power Users</i>	<i>0x223</i>	<i>PowerUser</i>
<i>BUILTIN\Print Operators</i>	<i>0x226</i>	<i>PrintOperator</i>
<i>BUILTIN\Replicators</i>	<i>0x228</i>	<i>Replicator</i>
<i>BUILTIN\Server Operators</i>	<i>0x225</i>	<i>SystemOperator</i>
<i>BUILTIN\Users</i>	<i>0x221</i>	<i>User</i>



Lớp `WindowsIdentity` cung cấp các phương thức khởi dựng nạp chồng cho phép bạn thu lấy đối tượng `WindowsIdentity` mô tả một người dùng nào đó (khi chạy trên *Microsoft Windows Server 2003* trở về sau). Bạn có thể sử dụng đối tượng này và phương pháp được mô tả trong mục này để xác định xem người dùng đó có phải là thành viên của một nhóm *Windows* nào đó hay không.

Nếu bạn sử dụng một trong các phương thức khởi dựng này khi chạy trên một phiên bản *Windows* cũ, nó sẽ ném ngoại lệ `System.ArgumentException`. Trên các nền *Windows* trước *Windows Server 2003*, bạn phải sử dụng mã lệnh nguyên sinh (*native code*) để thu lấy *Windows access token* mô tả người dùng cần thiết. Kể đó, bạn có thể sử dụng *access token* này để tạo đối tượng `WindowsIdentity`; mục 13.15 sẽ trình bày cách thu lấy *Windows access token* cho những người dùng cụ thể.

Ứng dụng `WindowsGroupExample` dưới đây trình bày cách kiểm tra người dùng hiện hành có là thành viên của một tập các nhóm *Windows* được nêu tên hay không. Các nhóm này được chỉ định trong các đối số dòng lệnh; bạn nhớ đặt tên máy tính, tên miền, hay `BUILTIN` (đối với các nhóm *Windows* chuẩn) vào trước tên nhóm.

```

using System;
using System.Security.Principal;

public class WindowsGroupExample {

    public static void Main (string[] args) {

        // Thu lấy đối tượng WindowsIdentity
        // mô tả người dùng hiện hành.
        WindowsIdentity identity = WindowsIdentity.GetCurrent();

        // Tạo đối tượng WindowsPrincipal mô tả các khả năng bảo mật
        // của đối tượng WindowsIdentity được chỉ định.
        WindowsPrincipal principal = new WindowsPrincipal(identity);

        // Duyệt qua các đối số dòng lệnh (tên nhóm) và kiểm tra xem
        // người dùng hiện hành có là thành viên của mỗi nhóm hay không.
        foreach (string role in args) {

            Console.WriteLine("Is {0} a member of {1}? = {2}",
                identity.Name, role, principal.IsInRole(role));
        }
    }
}

```

Nếu bạn chạy ví dụ này với tư cách người dùng có tên là `nnpbphuong81` trên một máy tính có tên là `MACHINE` bằng lệnh `WindowsGroupExample BUILTIN\Administrators BUILTIN\Users MACHINE\Accountants`, kết xuất có thể như sau:

```

Is MACHINE\nnpbphuong81 a member of BUILTIN\Administrators? = False
Is MACHINE\nnpbphuong81 a member of BUILTIN\Users? = True
Is MACHINE\nnpbphuong81 a member of MACHINE\Accountants? = True

```

14. Hạn chế những người dùng nào đó thực thi mã lệnh của bạn

?

Bạn cần hạn chế những người dùng nào đó truy xuất các phần tử trong mã lệnh của bạn dựa trên tên người dùng hay các vai trò mà người dùng này là thành viên.



Sử dụng lớp `System.Security.Permissions.PrincipalPermission` và bản sao đặc tính `System.Security.Permissions.PrincipalPermissionAttribute` của lớp này để bảo vệ các phần tử trong chương trình của bạn với các yêu cầu *RBS*.

.NET Framework hỗ trợ cả yêu cầu *RBS* bắt buộc (*imperative RBS demand*) và yêu cầu *RBS* khai báo (*declarative RBS demand*). Lớp `PrincipalPermission` hỗ trợ các lệnh bảo mật bắt buộc, và bản sao đặc tính `PrincipalPermissionAttribute` của lớp này hỗ trợ các lệnh bảo mật khai báo. Các yêu cầu *RBS* sử dụng cú pháp giống như các yêu cầu *CAS*, nhưng các yêu cầu *RBS* chỉ rõ tên mà người dùng hiện hành phải có, hoặc thông thường hơn là các vai trò mà người dùng này là thành viên. Một yêu cầu *RBS* lệnh cho bộ thực thi xét tên và các vai trò của người dùng hiện hành, và nếu chúng không đạt yêu cầu, bộ thực thi sẽ ném ngoại lệ `System.Security.SecurityException`.

Đoạn mã dưới đây trình bày cú pháp của một yêu cầu bảo mật bắt buộc:

```
// Cú pháp của một yêu cầu bảo mật bắt buộc dựa-trên-vai-trò.
public static void SomeMethod() {
    $
    PrincipalPermission perm =
        new PrincipalPermission("UserName", "RoleName");

    perm.Demand();
    $
}
```

Trước tiên, bạn phải tạo một đối tượng `PrincipalPermission` chỉ định tên người dùng và tên vai trò mà bạn yêu cầu, rồi gọi phương thức `Demand` của nó. Bạn chỉ có thể chỉ định một tên người dùng và tên vai trò cho mỗi yêu cầu. Nếu tên người dùng hoặc tên vai trò là `null`, bất kỳ giá trị nào cũng sẽ thỏa mãn yêu cầu. Khác với các quyền truy xuất mã lệnh, một yêu cầu *RBS* không cho kết quả trong một stack walk; bộ thực thi chỉ đánh giá tên người dùng và các vai trò của người dùng hiện hành.

Đoạn mã dưới đây trình bày cú pháp của một yêu cầu bảo mật khai báo:

```
// Cú pháp của một yêu cầu bảo mật khai báo dựa-trên-vai-trò.
[PrincipalPermission(SecurityAction.Demand, Name = "UserName",
    Role = "RoleName")]
public static void SomeMethod() { /*...*/ }
```

Bạn có thể thay các yêu cầu *RBS* khai báo ở mức lớp hay mức thành viên. Các yêu cầu mức-lớp áp dụng cho tất cả các thành viên của lớp trừ khi có một yêu cầu mức-thành-viên chép đè yêu cầu mức-lớp.

Nói chung, bạn có thể tự chọn hiện thực các yêu cầu *RBS* bắt buộc hay khai báo. Tuy nhiên, các yêu cầu bảo mật bắt buộc cho phép bạn tích hợp các yêu cầu *RBS* với logic của mã lệnh để thực hiện những yêu cầu *RBS* phức tạp. Ngoài ra, nếu không biết vai trò hay tên người dùng để yêu cầu lúc biên dịch, bạn phải sử dụng các yêu cầu bắt buộc. Các yêu cầu *RBS* khai báo có thuận lợi là chúng độc lập với logic của mã lệnh và dễ nhận biết hơn. Ngoài ra, bạn có thể

xem các yêu cầu *RBS* khai báo bằng công cụ *Permview.exe* (đã được thảo luận trong mục 13.6). Cho dù hiện thực yêu cầu *RBS* bắt buộc hay khai báo, bạn cũng phải chắc rằng bộ thực thi có thể truy xuất tên và các vai trò của người dùng hiện hành để đánh giá yêu cầu một cách phù hợp.

Lớp `System.Threading.Thread` mô tả một tiểu trình của hệ điều hành (chạy mã lệnh được quản-lý). Thuộc tính tĩnh `CurrentPrincipal` của lớp `Thread` chứa một thể hiện `IPrincipal`—mô tả người dùng mà tiểu trình hiện đang chạy trên danh nghĩa của người này. Ở mức hệ điều hành, mỗi tiểu trình cũng có một *Windows access token* kết giao—mô tả tài khoản *Windows* mà tiểu trình hiện đang chạy trên danh nghĩa của tài khoản này. Bạn cần hiểu rằng thể hiện `IPrincipal` và *Windows access token* là hai thực thể riêng biệt. *Windows* sử dụng *access token* để thực hiện cơ chế bảo mật hệ điều hành, trong khi bộ thực thi *.NET* sử dụng thể hiện `IPrincipal` để đánh giá các yêu cầu *RBS* ở mức ứng dụng. Mặc dù chúng có thể mô tả cùng một người dùng, nhưng điều này không có nghĩa là luôn luôn như vậy.

Theo mặc định, thuộc tính `Thread.CurrentPrincipal` là không xác định. Vì việc thu lấy các thông tin liên quan đến người dùng có thể mất nhiều thời gian, và chỉ một phần nhỏ trong số các ứng dụng sử dụng thông tin này, các nhà thiết kế *.NET* chọn cách khởi dựng "lười" đối với thuộc tính `CurrentPrincipal`. Trước tiên, mã lệnh thu lấy thuộc tính `Thread.CurrentPrincipal`, bộ thực thi gán một thể hiện `IPrincipal` cho thuộc tính này theo logic sau đây:

1. Nếu miền ứng dụng mà tiểu trình hiện hành đang thực thi có một *principal* mặc định, thì bộ thực thi sẽ gán *principal* này cho thuộc tính `Thread.CurrentPrincipal`.

Theo mặc định, miền ứng dụng không có *principal* mặc định. Bạn có thể thiết lập *principal* mặc định của một miền ứng dụng bằng cách gọi phương thức `SetThreadPrincipal` trên một đối tượng `System.AppDomain` mô tả miền ứng dụng bạn muốn cấu hình. Để gọi `SetPrincipalPolicy`, mã lệnh của bạn phải có phần tử `ControlPrincipal` của `SecurityPermission`. Bạn chỉ có thể thiết lập *principal* mặc định một lần cho mỗi miền ứng dụng; lời gọi `SetThreadPrincipal` thứ hai dẫn đến ngoại lệ `System.Security.Policy.PolicyException`.

2. Nếu miền ứng dụng không có *principal* mặc định, chính sách *principal* của miền ứng dụng sẽ xác định hiện thực `IPrincipal` nào sẽ được tạo ra và gán nó cho `Thread.CurrentPrincipal`.

Để cấu hình chính sách *principal* cho một miền ứng dụng, bạn cần thu lấy đối tượng `AppDomain` mô tả miền ứng dụng và gọi phương thức `SetPrincipalPolicy` của đối tượng này. Phương thức `SetPrincipalPolicy` nhận vào một thành viên thuộc kiểu liệt kê `System.Security.Principal.PrincipalPolicy`, giá trị này cho biết kiểu của đối tượng `IPrincipal` sẽ được gán cho `Thread.CurrentPrincipal`. Để gọi `SetPrincipalPolicy`, mã lệnh của bạn phải có phần tử `ControlPrincipal` của `SecurityPermission`. Bảng 13.4 liệt kê các giá trị của `PrincipalPolicy`; giá trị mặc định là `UnauthenticatedPrincipal`.

3. Nếu mã lệnh của bạn có phần tử `ControlPrincipal` của `SecurityPermission`, bạn có thể tự tạo ra đối tượng `IPrincipal` và trực tiếp gán nó cho thuộc tính

`Thread.CurrentPrincipal`. Việc này sẽ ngăn bộ thực thi gán các đối tượng `IPrincipal` mặc định hoặc tạo ra các đối tượng mới dựa trên chính sách *principal*.

Bảng 13.4 Các thành viên thuộc kiểu liệt kê `PrincipalPolicy`

Tên thành viên	Mô tả
<code>NoPrincipal</code>	Không có đối tượng <code>IPrincipal</code> nào được tạo ra, <code>Thread.CurrentPrincipal</code> trả về một tham chiếu <code>null</code> .
<code>UnauthenticatedPrincipal</code>	Một đối tượng <code>System.Security.Principal.GenericPrincipal</code> rỗng được tạo ra và được gán cho <code>Thread.CurrentPrincipal</code> .
<code>WindowsPrincipal</code>	Một đối tượng <code>WindowsPrincipal</code> (mô tả người dùng <i>Windows</i> đã đăng nhập) được tạo ra và được gán cho <code>Thread.CurrentPrincipal</code> .

Bất kể sử dụng phương pháp nào để thiết lập `IPrincipal` cho tiểu trình hiện hành, bạn cũng phải làm như thế trước khi sử dụng các yêu cầu bảo mật *RBS*, nếu không thông tin về người dùng (`IPrincipal`) sẽ không có hiệu lực để bộ thực thi có thể xử lý yêu cầu. Bình thường, khi chạy trên nền *Windows*, bạn thiết lập chính sách *principal* của một miền ứng dụng là `PrincipalPolicy.WindowsPrincipal` để thu lấy thông tin về người dùng *Windows*:

```
// Thu lấy một tham chiếu đến miền ứng dụng hiện hành.
```

```
AppDomain appDomain = System.AppDomain.CurrentDomain;
```

```
// Cấu hình miền ứng dụng hiện hành sao cho sử dụng các
```

```
// principal dựa-trên-Windows.
```

```
appDomain.SetPrincipalPolicy(
```

```
    System.Security.Principal.PrincipalPolicy.WindowsPrincipal);
```

File *RoleBasedSecurityExample.cs* (trong đĩa CD đính kèm) minh họa cách sử dụng các yêu cầu *RBS* bắt buộc và khai báo. Phần thứ nhất trình bày ba phương thức được bảo vệ bằng các yêu cầu *RBS* bắt buộc. Nếu đối tượng `Thread.CurrentPrincipal` không thỏa các đòi hỏi về tên người dùng và tư cách thành viên, yêu cầu sẽ ném ngoại lệ `SecurityException`.

```
public static void ProtectedMethod1() {
```

```
    // Một yêu cầu bảo mật bắt buộc dựa-trên-vai-trò: principal
```

```
    // hiện hành mô tả một định danh với tên là "nnbphuong81",
```

```
    // các vai trò của principal là không quan trọng.
```

```
    System.Security.Permissions.PrincipalPermission perm =
```

```
        new System.Security.Permissions.PrincipalPermission
```

```
            (@"MACHINE\nnbphuong81", null);
```

```
    perm.Demand();
```

```

}

public static void ProtectedMethod2() {

    // Một yêu cầu bảo mật bắt buộc dựa-trên-vai-trò: principal
    // hiện tại là một thành viên của vai trò "Managers" hay
    // "Developers". Khi sử dụng PrincipalPermission, bạn chỉ có thể diễn
    // tả mỗi quan hệ OR. Đó là vì phương thức PrincipalPolicy.Intersect
    // luôn trả về một quyền rỗng trừ khi hai input là như nhau.
    // Tuy nhiên, bạn có thể sử dụng logic của mã lệnh để hiện thực
    // các điều kiện phức tạp hơn. Trong trường hợp này, tên của định
    // danh là không quan trọng.
    System.Security.Permissions.PrincipalPermission perm1 =
        new System.Security.Permissions.PrincipalPermission
            (null, @"MACHINE\Managers");

    System.Security.Permissions.PrincipalPermission perm2 =
        new System.Security.Permissions.PrincipalPermission
            (null, @"MACHINE\Developers");

    perm1.Union(perm2).Demand();
}

public static void ProtectedMethod3() {

    // Một yêu cầu bảo mật bắt buộc dựa-trên-vai-trò: principal
    // hiện tại mô tả một định danh với tên là "nnbphuong81" và
    // là một thành viên của vai trò "Managers".
    System.Security.Permissions.PrincipalPermission perm =
        new System.Security.Permissions.PrincipalPermission
            (@"MACHINE\nnbphuong81", @"MACHINE\Managers");

    perm.Demand();
}

```

Phần thứ hai trình bày ba phương thức được bảo vệ bằng các yêu cầu *RBS* khai báo, tương đương với các yêu cầu *RBS* bắt buộc vừa được trình bày ở trên:

```
// Một yêu cầu bảo mật khai báo dựa-trên-vai-trò: principal hiện tại
// mô tả một định danh với tên là "nnbphuong81", các vai trò của
// principal là không quan trọng.
[PrincipalPermission(SecurityAction.Demand,
    Name = @"MACHINE\nnbphuong81")]
public static void ProtectedMethod1() { /*...*/}

// Một yêu cầu bảo mật khai báo dựa-trên-vai-trò: principal hiện tại
// là một thành viên của vai trò "Managers" hay "Developers". Bạn chỉ
// có thể diễn tả mối quan hệ OR (không thể diễn tả mối quan hệ AND).
// Tên của định danh là không quan trọng.
[PrincipalPermission(SecurityAction.Demand,
    Role = @"MACHINE\Managers")]
[PrincipalPermission(SecurityAction.Demand,
    Role = @"MACHINE\Developers")]
public static void ProtectedMethod2() { /*...*/}

// Một yêu cầu bảo mật khai báo dựa-trên-vai-trò: principal hiện tại
// mô tả một định danh với tên là "nnbphuong81" và là một thành viên
// của vai trò "Managers".
[PrincipalPermission(SecurityAction.Demand,
    Name = @"MACHINE\nnbphuong81",
    Role = @"MACHINE\Managers")]
public static void ProtectedMethod3() { /*...*/}
```

15.

Giải nhận người dùng Windows

?

Bạn muốn mã lệnh của bạn chạy trong ngữ cảnh của một người dùng Windows nào đó chứ không phải tài khoản người dùng hiện đang tích cực.

✕

Thu lấy đối tượng `System.Security.Principal.WindowsIdentity` mô tả người dùng Windows mà bạn cần giả nhận, rồi gọi phương thức `Impersonate` của đối tượng `WindowsIdentity`.

Mỗi tiểu trình Windows đều có một *access token* kết giao—mô tả tài khoản Windows mà tiểu trình hiện đang chạy trên danh nghĩa của tài khoản này. Hệ điều hành Windows sử dụng *access token* để xác định một tiểu trình có các quyền thích đáng để thực hiện các thao tác được-bảo-vệ trên danh nghĩa của tài khoản này hay không, như đọc/ghi file, khởi động lại hệ thống, và thay đổi thời gian hệ thống.

Theo mặc định, một ứng dụng được-quản-lý chạy trong ngữ cảnh của tài khoản Windows đã thực thi ứng dụng. Điều này là hoàn toàn bình thường, nhưng đôi lúc bạn muốn chạy ứng

dụng trong ngữ cảnh của một tài khoản *Windows* khác. Điều này đúng trong trường hợp các ứng dụng phía server cần xử lý phiên giao dịch dựa trên danh nghĩa của các người dùng kết nối đến server. Thông thường, một ứng dụng server chạy trong ngữ cảnh của tài khoản *Windows* được tạo riêng cho ứng dụng—đây là tài khoản dịch vụ (*service account*). Tài khoản dịch vụ này sẽ có các quyền tối thiểu để truy xuất các tài nguyên hệ thống, làm cho ứng dụng hoạt động như thể đó là người dùng đã kết nối cho phép ứng dụng truy xuất các hoạt động và tài nguyên phù hợp với quyền hạn của người dùng đó. Khi một ứng dụng nắm lấy định danh của một người dùng khác, đây là sự giả nhận (*impersonation*). Nếu được hiện thực đúng, sự giả nhận sẽ đơn giản hóa việc quản trị bảo mật và thiết kế ứng dụng, trong khi vẫn duy trì việc giải trình người dùng.



Như đã thảo luận trong mục 13.14, *Windows access token* và *.NET principal* của một tiểu trình là các thực thể riêng biệt và có thể mô tả những người dùng khác nhau. Kỹ thuật giả nhận được mô tả trong mục này chỉ thay đổi *Windows access token* của tiểu trình hiện hành, chứ không thay đổi *principal* của tiểu trình này. Để thay đổi *principal* của tiểu trình, mã lệnh của bạn phải có phần tử `ControlPrincipal` của `SecurityPermission` và gán một đối tượng `System.Security.Principal.IPrincipal` mới vào thuộc tính `CurrentPrincipal` của `System.Threading.Thread` hiện hành.

Lớp `System.Security.Principal.WindowsIdentity` cung cấp các chức năng mà thông qua đó, bạn có thể thực hiện sự giả nhận. Tuy nhiên, quá trình này tùy thuộc vào ứng dụng của bạn đang chạy trên phiên bản *Windows* nào. Trên *Windows Server 2003* trở về sau, lớp `WindowsIdentity` hỗ trợ các phiên bản nạp của chồng phương thức khởi dựng, cho phép tạo ra các đối tượng `WindowsIdentity` dựa trên tên tài khoản của người dùng cần giả nhận. Trên tất cả các phiên bản *Windows* trước đó, trước hết bạn phải thu lấy `System.IntPtr` chứa tham chiếu đến *Windows access token* mô tả người dùng cần giả nhận. Để thu lấy tham chiếu này, bạn cần sử dụng một phương thức nguyên sinh như `LogonUser` của *Win32 API*.



Vấn đề chủ yếu khi thực hiện sự giả nhận trên *Windows 2000* và *Windows NT* là một tài khoản phải có đặc quyền `SE_TCB_NAME` thì mới có thể thực thi `LogonUser`. Điều này đòi hỏi bạn cấu hình chính sách bảo mật của *Windows* và cấp cho tài khoản quyền “*Act as part of operating system*” (mức tin cậy rất cao). Bạn đừng bao giờ trực tiếp cấp đặc quyền `SE_TCB_NAME` cho các tài khoản người dùng.

Một khi đã có đối tượng `WindowsIdentity` mô tả người dùng cần giả nhận, bạn hãy gọi phương thức `Impersonate` của nó. Từ lúc này, tất cả các hành động mà mã lệnh của bạn thực hiện đều diễn ra trong ngữ cảnh của tài khoản *Windows* đã được giả nhận. Phương thức `Impersonate` trả về đối tượng `System.Security.Principal.WindowsSecurityContext`, đối tượng này mô tả tài khoản tích cực trước khi giả nhận. Để trở về tài khoản cũ, bạn cần gọi phương thức `Undo` của đối tượng `WindowsSecurityContext` này.

Ứng dụng dưới đây trình bày sự giả nhận của một người dùng *Windows*. Ứng dụng này cần hai đối số dòng lệnh: tên tài khoản của người dùng cần giả nhận và password của tài khoản.

Ứng dụng này sử dụng hàm `LogonUser` của *Win32 API* để thu lấy *Windows access token* cho người dùng được chỉ định, giả nhận người dùng này, rồi trở về ngữ cảnh của người dùng cũ. Ví dụ, lệnh **ImpersonationExample nnbphuong81 password** sẽ giả nhận người dùng `nnbphuong81` nếu người dùng đó đã tồn tại trong cơ sở dữ liệu tài khoản cục bộ.

```
using System;
using System.IO;
using System.Security.Principal;
using System.Security.Permissions;
using System.Runtime.InteropServices;

// Bảo đảm assembly có quyền truy xuất mã lệnh không-được-quản-lý
// và có quyền kiểm soát principal của tiểu trình.
[assembly:SecurityPermission(SecurityAction.RequestMinimum,
    UnmanagedCode=true, ControlPrincipal=true)]

public class ImpersonationExample {

    // Định nghĩa các hằng số được sử dụng cùng với hàm LogonUser.
    const int LOGON32_PROVIDER_DEFAULT = 0;
    const int LOGON32_LOGON_INTERACTIVE = 2;

    // Nhập hàm Win32 LogonUser từ advapi32.dll. Chỉ định
    // "SetLastError = true" để có thể truy xuất các mã lỗi của Win32.
    [DllImport("advapi32.dll", SetLastError=true)]
    static extern int LogonUser(string userName, string domain,
        string password, int logonType, int logonProvider,
        ref IntPtr accessToken);

    public static void Main(string[] args) {

        // Tạo một IntPtr mới để giữ lấy access token
        // do hàm LogonUser trả về.
        IntPtr accessToken = IntPtr.Zero;

        // Gọi LogonUser để thu lấy access token cho người dùng
        // được chỉ định. Biến accessToken được truyền cho LogonUser
        // bằng tham chiếu và sẽ chứa tham chiếu đến Windows access token
        // nếu LogonUser thành công.
        int result = LogonUser(
```

```

args[0],                // tên người dùng để đăng nhập
".",                    // sử dụng CSDL tài khoản cục bộ
args[1],                // password của người dùng
LOGON32_LOGON_INTERACTIVE, // tạo một interactive login
LOGON32_PROVIDER_DEFAULT, // sử dụng logon provider mặc định
ref accessToken          // nhận access token handle
);

// Nếu lỗi xảy ra (LogonUser trả về zero), hiển thị lỗi và thoát.
if (result == 0) {

    Console.WriteLine("LogonUser returned error {0}",
        Marshal.GetLastWin32Error());

} else {

    // Tạo một WindowsIdentity mới từ Windows access token.
    WindowsIdentity identity = new WindowsIdentity(accessToken);

    // Hiển thị định danh đang tích cực (trước khi giả nhận).
    Console.WriteLine("Identity before impersonation = {0}",
        WindowsIdentity.GetCurrent().Name);

    // Giả nhận người dùng đã được chỉ định. Đối tượng
    // WindowsImpersonationContext chứa các thông tin
    // cần thiết để trở về ngữ cảnh của người dùng cũ.
    WindowsImpersonationContext impContext =
        identity.Impersonate();

    // Hiển thị định danh đang tích cực (trong lúc giả nhận).
    Console.WriteLine("Identity during impersonation = {0}",
        WindowsIdentity.GetCurrent().Name);

    // *****
    // Thực hiện các hành động với danh nghĩa người dùng được giả nhận
    // *****

```

```
// Trở về người dùng Windows cũ bằng đối tượng
// WindowsImpersonationContext.
impContext.Undo();

// Hiển thị định danh đang tích cực (sau khi giả nhận).
Console.WriteLine("Identity after impersonation = {0}",
    WindowsIdentity.GetCurrent().Name);
}
}
}
```

14

MẬT MÃ

Mật mã (*cryptography*) là một trong những mặt phức tạp nhất của quá trình phát triển phần mềm mà bất kỳ nhà phát triển nào cũng sẽ sử dụng. Lý thuyết kỹ thuật mật mã hiện đại cực kỳ khó hiểu và đòi hỏi một mức kiến thức toán học mà tương đối ít người có được. May mắn là thư viện lớp *.NET Framework* cung cấp các hiện thực để sử dụng cho hầu hết các kỹ thuật mật mã thông dụng và hỗ trợ các giải thuật phổ biến nhất. Chương này sẽ bàn về các vấn đề sau:

- Tạo số ngẫu nhiên (mục 14.1).
- Tạo và xác minh các mã băm mật mã và các mã băm có khóa (mục 14.2, 14.3, 14.4, và 14.5).
- Sử dụng giải thuật đối xứng và không đối xứng để mật hóa và giải mật hóa dữ liệu (mục 14.6 và 14.8).
- Tìm lại, lưu trữ, và chuyển đổi các khóa mật mã (mục 14.7, 14.9, và 14.10).



Khi nghĩ cách áp dụng các kỹ thuật trong chương này vào mã lệnh, bạn nên nhớ rằng mật mã chẳng phải là cái mà bạn hiện thực đơn lẻ. Mật mã không ngang bằng với bảo mật (*security*); sử dụng mật mã chỉ là một phần nhỏ trong việc tạo một giải pháp an toàn.

Đối với những ai chưa quen thuộc với mật mã, dưới đây là định nghĩa của một số từ quan trọng:

- **Encrypt** (động từ, tạm dịch là mật hóa) là mã hóa thông tin theo cách nào đó để mọi người không thể đọc được nó, trừ những ai có khóa.
- **Decrypt** (động từ, tạm dịch là giải mật hóa) là giải mã thông tin đã-được-mật-hóa.
- **Key** là chuỗi các bit dùng để mật hóa và giải mật hóa thông tin.
- **Plaintext** là text chưa-được-mật-hóa hay đã-được-giải-mật-hóa.
- **Ciphertext** là text đã-được-mật-hóa.

1.

Tạo số ngẫu nhiên



Bạn cần tạo một số ngẫu nhiên dùng cho các ứng dụng mật mã và bảo mật.



Sử dụng một bộ tạo số ngẫu nhiên mật mã (*cryptographic random number generator*), chẳng hạn `System.Security.Cryptography.RNGCryptographyServiceProvider`.

Lớp `System.Random` là một bộ tạo số giả ngẫu nhiên, nó sử dụng một giải thuật toán học để mô phỏng việc tạo số ngẫu nhiên. Thực ra, giải thuật này là tất định (*deterministic*), nghĩa là bạn luôn có thể tính được số kế tiếp sẽ là gì dựa trên số đã được tạo trước đó. Điều này nghĩa là các số được tạo bởi lớp `Random` sẽ không phù hợp khi tính bảo mật được ưu tiên, chẳng hạn tạo khóa mật hóa và password.

Khi cần một số ngẫu nhiên không tất định (*nondeterministic*) dùng trong các ứng dụng liên quan đến mật mã hay bảo mật, bạn phải sử dụng bộ tạo số ngẫu nhiên dẫn xuất từ lớp

`System.Security.Cryptography.RandomNumberGenerator`. Đây là một lớp trừu tượng mà tất cả các bộ tạo số ngẫu nhiên cụ thể đều sẽ thừa kế từ nó. Hiện tại, chỉ có một hiện thực là lớp `RNGCryptoServiceProvider`. Lớp này cung cấp một vỏ bọc được-quản-lý cho hàm `CryptGenRandom` của *Win32 CryptoAPI*, và bạn có thể sử dụng để đổ vào một mảng byte các giá trị byte ngẫu nhiên.



Các số do `RNGCryptoServiceProvider` sinh ra không thật sự ngẫu nhiên. Tuy nhiên, chúng “đủ” ngẫu nhiên để đáp ứng yêu cầu cho các ứng dụng mật mã và bảo mật trong hầu hết các môi trường chính phủ và thương mại.

Lớp cơ sở `RandomNumberGenerator` là một *factory* cho các lớp hiện thực dẫn xuất từ đó. Gọi `RandomNumberGenerator.Create("System.Security.Cryptography.RNGCryptoServiceProvider")` sẽ trả về một thể hiện của `RNGCryptoServiceProvider`, và bạn có thể sử dụng nó để tạo số ngẫu nhiên. Ngoài ra, vì `RNGCryptoServiceProvider` là hiện thực duy nhất nên nó sẽ là lớp mặc định được tạo ra khi bạn gọi phương thức `Create` không có đối số: `RandomNumberGenerator.Create()`.

Ví dụ dưới đây tạo một đối tượng `RNGCryptoServiceProvider` và sử dụng nó để tạo các giá trị ngẫu nhiên. Phương thức `GetBytes` đổ vào một mảng byte các giá trị byte ngẫu nhiên. Bạn có thể sử dụng phương thức `GetNonZeroBytes` nếu cần dữ liệu ngẫu nhiên không chứa giá trị zero.

```
using System;
using System.Security.Cryptography;

public class SecureRandomNumberExample {

    public static void Main() {

        // Tạo mảng byte dùng để lưu trữ dữ liệu ngẫu nhiên.
        byte[] number = new byte[32];

        // Tạo bộ tạo số ngẫu nhiên mặc định.
        RandomNumberGenerator rng = RandomNumberGenerator.Create();

        // Tạo dữ liệu ngẫu nhiên.
        rng.GetBytes(number);

        // Hiển thị dữ liệu ngẫu nhiên.
        Console.WriteLine(BitConverter.ToString(number));
    }
}
```



Những nỗ lực tính toán cần thiết để tạo một số ngẫu nhiên với `RNGCryptoServiceProvider` lớn hơn nhiều so với `Random`. Đối với mục đích thường ngày, sử dụng `RNGCryptoServiceProvider` là quá mức cần thiết. Bạn nên xem xét số lượng số ngẫu nhiên cần tạo và mục đích của các số này trước khi quyết định sử dụng `RNGCryptoServiceProvider`. Sử dụng lớp `RNGCryptoServiceProvider` quá mức và không cần thiết có thể ảnh hưởng đáng kể lên hiệu năng của ứng dụng.

2.

Tính mã băm của password



Bạn cần lưu trữ password của người dùng một cách an toàn để bạn có thể sử dụng nó để xác thực người dùng đó trong tương lai.



Đừng lưu trữ password của người dùng ở dạng plaintext vì đây là một nguy cơ bảo mật lớn. Thay vào đó, hãy tạo và lưu trữ một mã băm của password bằng một lớp giải thuật băm dẫn xuất từ lớp `System.Security.Cryptography.HashAlgorithm`. Khi xác thực, tạo mã băm của password và so sánh nó với mã băm đã được lưu trữ.

Các giải thuật băm là các hàm mật mã một chiều, nhận plaintext có chiều dài thay đổi và tạo một giá trị số có kích thước cố định. Chúng là một chiều vì gần như không thể tìm lại plaintext gốc từ mã băm. Các giải thuật băm là tất định (*deterministic*); áp dụng cùng giải thuật băm cho một mẫu plaintext luôn tạo ra cùng mã băm. Điều này khiến mã băm trở nên hữu ích cho việc xác định hai khối plaintext (trong trường hợp này là password) có giống nhau hay không. Mục đích của các giải thuật băm bảo đảm rằng—mặc dù không phải không xảy ra—khả năng hai mẫu plaintext khác nhau tạo ra cùng mã băm là cực kỳ nhỏ. Ngoài ra, không có mối tương quan nào giữa sự giống nhau của hai mẫu plaintext và mã băm của chúng; một khác biệt nhỏ trong plaintext cũng có thể gây ra khác biệt đáng kể trong mã băm.

Khi sử dụng password để xác thực một người dùng, bạn không quan tâm đến nội dung của password do người dùng nhập vào. Bạn chỉ cần biết rằng password được nhập trùng khớp với password mà bạn đã ghi lại cho người dùng đó trong cơ sở dữ liệu tài khoản. Bản chất của các giải thuật băm khiến chúng trở nên lý tưởng trong việc lưu trữ password một cách an toàn. Khi người dùng cung cấp một password mới, bạn phải tạo mã băm của password và lưu trữ mã băm này, rồi loại bỏ password dạng text. Mỗi khi người dùng xác thực với ứng dụng của bạn, tính mã băm của password do người đó cung cấp và so sánh nó với mã băm mà bạn đã lưu trữ.



Người ta thường hỏi cách thu lấy password từ một mã băm. Và câu trả lời là không thể. Mục đích của mã băm là đóng vai trò như một token và bạn có thể tùy ý lưu trữ nó mà không sinh ra lỗ hổng bảo mật nào. Nếu người dùng quên password, bạn không thể tìm lại nó từ mã băm đã được lưu trữ; bạn phải reset tài khoản này thành giá trị mặc định nào đó, hoặc tạo một password mới cho người dùng.

Lớp trừu tượng `HashAlgorithm` cung cấp lớp cơ sở để tất cả các hiện thực giải thuật băm cụ thể dẫn xuất từ đó. Thư viện lớp *.NET Framework* có sáu hiện thực giải thuật băm cụ thể (được liệt kê trong bảng 14.1), mỗi lớp hiện thực là một thành viên của không gian tên `System.Security.Cryptography`. Các lớp với phần đuôi là `CryptoServiceProvider` bọc lấy các chức năng do *Win32 CryptoAPI* cung cấp, trong khi các lớp với phần đuôi là `Managed` được hiện thực hoàn toàn bằng mã lệnh được-quản-lý.

Bảng 14.1 Các hiện thực giải thuật băm

Tên giải thuật	Tên lớp	Kích thước mã băm (bit)
<i>MD5</i>	<code>MD5CryptoServiceProvider</code>	128
<i>SHA</i> hay <i>SHA1</i>	<code>SHA1CryptoServiceProvider</code>	160
<i>SHA1Managed</i>	<code>SHA1Managed</code>	160
<i>SHA256</i> hay <i>SHA-256</i>	<code>SHA256Managed</code>	256
<i>SHA384</i> hay <i>SHA-384</i>	<code>SHA384Managed</code>	384
<i>SHA512</i> hay <i>SHA-512</i>	<code>SHA512Managed</code>	512

Mặc dù bạn có thể trực tiếp tạo ra thể hiện của các lớp giải thuật băm, lớp cơ sở `HashAlgorithm` là một *factory* cho các lớp hiện thực dẫn xuất từ nó. Gọi phương thức tĩnh `HashAlgorithm.Create` với đối số là tên giải thuật sẽ trả về một đối tượng thuộc kiểu đã được chỉ định. Sử dụng *factory* cho phép bạn ghi mã lệnh tổng quát và mã lệnh này có thể làm việc với bất kỳ hiện thực giải thuật băm nào.

Một khi bạn đã có đối tượng `HashAlgorithm`, phương thức `ComputeHash` của nó nhận một mảng byte chứa plaintext và trả về một mảng byte mới chứa mã băm được tạo ra. Bảng 14.1 cho biết kích thước của mã băm (tính bằng bit) được tạo ra bởi mỗi lớp giải thuật băm.

Lớp `HashPasswordExample` dưới đây trình bày cách tạo mã băm từ một chuỗi (password chẳng hạn). Ứng dụng này cần hai đối số dòng lệnh: tên của giải thuật băm cần sử dụng và chuỗi cần tạo mã băm. Vì phương thức `HashAlgorithm.ComputeHash` yêu cầu một mảng byte nên trước hết bạn phải mã hóa chuỗi nhập bằng lớp `System.Text.Encoding` (lớp này cung cấp các cơ chế dùng để chuyển chuỗi thành/từ các định dạng mã hóa ký tự khác nhau).

```
using System;
using System.Text;
using System.Security.Cryptography;

public class HashPasswordExample {

    public static void Main(string[] args) {

        // Tạo HashAlgorithm của kiểu được chỉ định bởi
        // đối số dòng lệnh thứ nhất.
        using (HashAlgorithm hashAlg = HashAlgorithm.Create(args[0])) {
```

```

        // Chuyển chuỗi password (đổi số dòng lệnh thứ hai)
        // thành một mảng byte.
        byte[] pwordData = Encoding.Default.GetBytes(args[1]);

        // Tạo mã băm của password.
        byte[] hash = hashAlg.ComputeHash(pwordData);

        // Hiển thị mã băm của password.
        Console.WriteLine(BitConverter.ToString(hash));
    }
}
}

```

Chạy lệnh `HashPasswordExample SHA1 ThisIsMyPassword` sẽ hiển thị mã băm sau đây:

80-36-31-2F-EA-D9-93-45-79-34-C9-FD-21-EE-8D-05-16-DC-A1-E2

3.

Tính mã băm của file



Bạn cần xác định nội dung của một file có thay đổi theo thời gian hay không.



Tạo mã băm cho nội dung của file bằng phương thức `ComputeHash` của lớp `System.Security.Cryptography.HashAlgorithm`. Lưu trữ mã băm này để sau này so sánh với các mã băm được tạo mới.

Ngoài việc cho phép bạn lưu trữ password một cách an toàn (đã được thảo luận trong mục 14.2), mã băm còn cung cấp một phương cách rất hay để xác định một file có thay đổi hay không. Bằng cách tính toán và lưu trữ mã băm của một file, sau này bạn có thể tính lại mã băm của file này để xác định file có thay đổi trong thời gian chuyển tiếp hay không. Giải thuật băm sẽ sinh ra một mã băm rất khác ngay cả chỉ với một thay đổi rất nhỏ trong file, nên khả năng hai file khác nhau cho ra cùng mã băm là cực kỳ nhỏ.



Các mã băm chuẩn không phù hợp khi gửi cùng với một file để bảo đảm tính toàn vẹn của nội dung file. Nếu ai đó chặn được file trên đường đi, người này có thể dễ dàng thay đổi file và tính lại mã băm. Chúng ta sẽ thảo luận một biến thể của mã băm trong mục 14.5 (mã băm có khóa), mã băm này phù hợp cho việc bảo đảm tính toàn vẹn của file trên đường đi.

Dễ dàng tạo được mã băm của một file với lớp `HashAlgorithm`. Trước hết, thể hiện hóa một trong các hiện thực giải thuật băm dẫn xuất từ lớp `HashAlgorithm` (bạn cần truyền tên giải thuật băm cho phương thức `HashAlgorithm.Create`—xem tên các giải thuật băm hợp lệ trong bảng 14.1). Kế tiếp, thay vì truyền một mảng byte cho phương thức `ComputeHash`, bạn hãy truyền một đối tượng `System.IO.Stream` mô tả file cần được tạo mã băm. Đối tượng

HashAlgorithm xử lý quá trình đọc dữ liệu từ Stream và trả về một mảng byte chứa mã băm cho file.

Lớp HashStreamExample dưới đây trình bày cách tạo mã băm từ một file. Bạn phải chỉ định tên giải thuật băm và tên file làm đối số dòng lệnh, ví dụ `HashStreamExample SHA1 HashStreamExample.cs`.

```
using System;
using System.IO;
using System.Security.Cryptography;

public class HashStreamExample {

    public static void Main(string[] args) {

        // Tạo một HashAlgorithm với kiểu được chỉ định trong
        // đối số dòng lệnh thứ nhất.
        using (HashAlgorithm hashAlg = HashAlgorithm.Create(args[0])) {

            // Mở một FileStream cho file được chỉ định trong
            // đối số dòng lệnh thứ hai.
            using (Stream file = new FileStream(args[1],
                FileMode.Open)) {

                // Tạo mã băm cho nội dung của file.
                byte[] hash = hashAlg.ComputeHash(file);

                // Hiển thị mã băm.
                Console.WriteLine(BitConverter.ToString(hash));
            }
        }
    }
}
```

4.

Kiểm tra mã băm

?

Bạn cần xác minh một password hoặc xác nhận một file vẫn không thay đổi bằng cách so sánh hai mã băm.

✕

Chuyển cả mã băm cũ và mới thành chuỗi thập lục phân, chuỗi *Base64*, hay mảng byte và so sánh chúng.

Bạn có thể sử dụng mã băm để xác định hai mẫu dữ liệu có giống nhau hay không, để không phải lưu trữ hay duy trì việc truy xuất đến dữ liệu gốc. Để xác định dữ liệu có thay đổi theo thời gian hay không, bạn phải tạo và lưu trữ mã băm của dữ liệu gốc. Sau đó, hãy tạo một mã băm khác cho dữ liệu này rồi so sánh mã băm cũ và mới để cho thấy có thay đổi nào xảy ra hay không. Định dạng của mã băm gốc sẽ xác định cách thức phù hợp nhất để kiểm tra mã băm mới được tạo.



Nhiều mục trong chương này sử dụng phương thức `ToString` của lớp `System.BitConverter` để chuyển mảng byte thành giá trị chuỗi thập lục phân khi hiển thị. Mặc dù dễ sử dụng và thích hợp cho mục đích hiển thị, bạn có thể nhận thấy cách này không phù hợp khi lưu trữ mã băm vì nó đặt dấu gạch nối (-) giữa mỗi giá trị byte (ví dụ, 4D-79-3A-C9-...). Ngoài ra, lớp `BitConverter` không cung cấp phương thức nào để phân tích một biểu diễn chuỗi như thế trở về một mảng byte.

Mã băm thường được lưu trữ trong file text ở dạng chuỗi thập lục phân (ví dụ, 89D22213170A9CFF09A392F00E2C6C4EDC1B0EF9) hoặc chuỗi được mã hóa theo *Base64* (ví dụ, idTiExcKnP8Jo5LwDixsTtwbDvk=). Mã băm cũng có thể được lưu trữ trong cơ sở dữ liệu ở dạng giá trị byte thô. Bất kể bạn lưu trữ mã băm theo cách nào, bước đầu tiên khi so sánh mã băm cũ và mới là đưa chúng về một dạng chung.

Đoạn mã dưới đây chuyển mã băm mới (mảng byte) thành chuỗi thập lục phân khi so sánh với mã băm cũ. Ngoài phương thức `BitConverter.ToString` mà chúng ta đã thảo luận ở trên, thư viện lớp *.NET Framework* không cung cấp phương thức nào để chuyển một mảng byte thành chuỗi thập lục phân. Bạn phải viết một vòng lặp đi qua các phần tử của mảng byte, chuyển mỗi byte thành chuỗi, và gắn chuỗi này vào biểu diễn chuỗi thập lục phân của mã băm. Sử dụng `System.Text.StringBuilder` sẽ tránh tạo ra các chuỗi mới không cần thiết mỗi khi vòng lặp gắn giá trị byte kế tiếp vào chuỗi kết quả (xem mục 2.1 để biết thêm chi tiết).

```
// Phương thức dùng để so sánh mã băm mới với
// mã băm có sẵn (được biểu diễn ở dạng chuỗi thập lục phân).
private static bool VerifyHexHash(byte[] hash, string oldHashString) {

    // Tạo biểu diễn chuỗi cho mã băm mới.
    System.Text.StringBuilder newHashString =
        new System.Text.StringBuilder(hash.Length);

    foreach (byte b in hash) {
        newHashString.AppendFormat("{0:X2}", b);
    }

    // So sánh biểu diễn chuỗi của mã băm cũ và mới,
    // và trả về kết quả.
```

```

        return (oldHashString == newHashString.ToString());
    }

```

Trong đoạn mã dưới đây, mã băm mới là một mảng byte và mã băm cũ là một chuỗi được mã hóa theo *Base64*. Đoạn mã này sẽ mã hóa mã băm mới thành chuỗi *Base64* rồi thực hiện phép so sánh chuỗi.

```

// Phương thức dùng để so sánh mã băm mới với
// mã băm có sẵn (được biểu diễn ở dạng chuỗi Base64).
private static bool VerifyB64Hash(byte[] hash, string oldHashString) {

    // Tạo biểu diễn chuỗi Base64 cho mã băm mới.
    string newHashString = System.Convert.ToBase64String(hash);

    // So sánh biểu diễn chuỗi của mã băm cũ và mới,
    // rồi trả về kết quả.
    return (oldHashString == newHashString);
}

```

Cuối cùng, đoạn mã dưới đây so sánh hai mã băm được biểu diễn ở dạng mảng byte. Thư viện lớp *.NET Framework* không có phương thức nào thực hiện kiểu so sánh này, do đó bạn phải viết một vòng lặp để so sánh các phần tử của hai mảng. Đoạn mã này có sử dụng một vài kỹ thuật không tốn nhiều thời gian như: bảo đảm các mảng byte có cùng chiều dài trước khi bắt đầu so sánh chúng, và trả về *false* khi tìm thấy khác biệt đầu tiên.

```

// Phương thức dùng để so sánh mã băm mới với
// mã băm có sẵn (được biểu diễn ở dạng mảng byte).
private static bool VerifyByteHash(byte[] hash, byte[] oldHash) {

    // Nếu một mảng là null, hoặc hai mảng có chiều dài khác nhau
    // thì chúng không bằng nhau.
    if (hash == null || oldHash == null || hash.Length != oldHash.Length)
        return false;

    // Duyệt qua mảng byte và so sánh mỗi giá trị byte.
    for (int count = 0; count < hash.Length; count++) {
        if (hash[count] != oldHash[count]) return false;
    }

    // Hai mã băm bằng nhau.
    return true;
}

```

5. Bảo đảm tính toàn vẹn dữ liệu bằng mã băm có khóa

? Bạn cần chuyển một file cho ai đó và cấp cho người này một phương cách để xác minh tính toàn vẹn của file.

X Cấp cho người nhận một khóa bí mật (*key*). Khóa này có thể là một số được sinh ngẫu nhiên, nhưng nó cũng có thể là một nhóm từ mà bạn và người nhận đã thỏa thuận. Sử dụng khóa cùng với một trong những lớp giải thuật băm có khóa dẫn xuất từ lớp `System.Security.Cryptography.KeyedHashAlgorithm` để tạo mã băm có khóa. Gửi mã băm này cùng với file. Khi nhận được file, người nhận sẽ tạo mã băm có khóa cho file này bằng khóa. Nếu hai mã băm giống nhau, người nhận sẽ biết rằng file này do bạn gửi đến và nó không bị thay đổi trong quá trình chuyển giao.

Mã băm rất hữu ích khi so sánh hai mẫu dữ liệu để xác định chúng có giống nhau hay không (cả khi bạn không thể truy xuất được dữ liệu gốc). Tuy nhiên, bạn không thể sử dụng mã băm để cam đoan với người nhận về tính toàn vẹn của dữ liệu. Nếu có ai đó chặn được dữ liệu, người này có thể thay thế dữ liệu và tạo mã băm mới. Khi người nhận kiểm tra mã băm, nó có vẻ đúng nhưng thực tế dữ liệu không giống với những gì bạn gửi lúc ban đầu.

Một giải pháp đơn giản và hiệu quả cho vấn đề toàn vẹn dữ liệu là mã băm có khóa (*keyed hash code*). Mã băm có khóa cũng tương tự như mã băm bình thường (đã được thảo luận trong mục 14.2 và 14.3); tuy nhiên, mã băm có khóa kết hợp thêm một phần tử dữ liệu bí mật (khóa), phần tử này chỉ có người gửi và người nhận biết. Nếu không có khóa, không ai có thể tạo được mã băm đúng từ tập dữ liệu cho trước.

X Khóa phải được giữ bí mật. Nếu ai đó biết khóa thì có thể tạo ra mã băm có khóa hợp lệ, nghĩa là bạn sẽ không thể xác định họ có thay đổi nội dung của tài liệu hay không. Vì lý do này, bạn không nên chuyển giao hay lưu trữ khóa cùng với tài liệu cần được bảo vệ tính toàn vẹn. Mục 14.10 sẽ cung cấp một cơ chế mà bạn có thể sử dụng để trao đổi khóa một cách an toàn.

Tạo mã băm có khóa cũng tương tự như tạo mã băm bình thường vì lớp trừu tượng `System.Security.Cryptography.KeyedHashAlgorithm` mở rộng lớp `System.Security.Cryptography.HashAlgorithm`. Lớp `KeyedHashAlgorithm` cung cấp một lớp cơ sở để tất cả các giải thuật băm có khóa dẫn xuất từ đó. Thư viện lớp *.NET Framework* có hai hiện thực giải thuật băm có khóa được liệt kê trong bảng 14.2; mỗi hiện thực là một thành viên của không gian tên `System.Security.Cryptography`.

Bảng 14.2 Các hiện thực giải thuật băm có khóa

Giải thuật/Tên lớp	Kích thước khóa (bit)	Kích thước mã băm (bit)
HMACSHA1	bất kỳ	160
MACTripleDES	64, 128, 192	64

Cũng như các giải thuật băm chuẩn, bạn có thể trực tiếp tạo ra các đối tượng giải thuật băm có khóa, hoặc bạn có thể sử dụng phương thức tĩnh `KeyedHashAlgorithm.Create` với đối số là tên giải thuật. Sử dụng *factory* cho phép bạn viết mã lệnh tổng quát và mã lệnh này có thể làm việc với bất kỳ hiện thực giải thuật băm có khóa nào, nhưng theo bảng 14.2, mỗi lớp hỗ trợ các chiều dài khóa khác nhau nên bạn phải cung cấp giá trị này trong mã lệnh tổng quát.

Nếu sử dụng phương thức khởi dựng để tạo đối tượng băm có khóa, bạn có thể truyền khóa cho phương thức này. Khi sử dụng *factory*, bạn phải thiết lập khóa bằng thuộc tính `Key` (được thừa kế từ lớp `KeyedHashAlgorithm`). Một khi đã cấu hình khóa, gọi phương thức `ComputeHash` với đối số là một mảng byte hay một đối tượng `System.IO.Stream`. Giải thuật băm có khóa sẽ xử lý dữ liệu nhập và trả về một mảng byte chứa mã băm có khóa. Bảng 14.2 cho thấy kích thước của mã băm do mỗi giải thuật băm có khóa sinh ra.

Lớp `KeyedHashStreamExample` dưới đây trình bày cách tạo mã băm có khóa từ một file. Bạn phải chỉ định tên file và một khóa làm đối số dòng lệnh. Ứng dụng này sử dụng lớp `HMACSHA1` để tạo mã băm có khóa và rồi hiển thị nó ra cửa sổ *Console*.

```
using System;
using System.IO;
using System.Text;
using System.Security.Cryptography;

public class KeyedHashStreamExample {

    public static void Main(string[] args) {

        // Tạo mảng byte từ chuỗi key (là đối số dòng lệnh thứ hai).
        byte[] key = Encoding.Unicode.GetBytes(args[1]);

        // Tạo một đối tượng HMACSHA1
        // (truyền key cho phương thức khởi dựng).
        using (HMACSHA1 hashAlg = new HMACSHA1(key)) {

            // Mở một FileStream để đọc file (tên file
            // được chỉ định trong đối số dòng lệnh thứ nhất).
            using (Stream file = new FileStream(args[0],
                FileMode.Open)) {

                // Tạo mã băm có khóa cho nội dung file.
                byte[] hash = hashAlg.ComputeHash(file);

                // Hiển thị mã băm có khóa ra cửa sổ Console.
```

```

        Console.WriteLine(BitConverter.ToString(hash));
    }
}
}
}

```

Lệnh **KeyedHashStreamExample** **KeyedHashStreamExample.cs** **secretKey** sẽ sinh ra mã băm như sau:

```
95-95-2A-8E-44-D4-3C-55-6F-DA-06-44-27-79-29-81-15-C7-2A-48
```

Ứng dụng **KeyedHashMessageExample.cs** (có trong đĩa CD đính kèm) trình bày cách tạo một mã băm có khóa từ một chuỗi. Ứng dụng này yêu cầu hai đối số dòng lệnh: một thông điệp và một khóa, và sẽ tạo ra mã băm có khóa cho chuỗi thông điệp bằng khóa này. Ví dụ, lệnh **KeyedHashMessageExample "Two hundred dollars is my final offer" secretKey** sẽ sinh ra mã băm như sau:

```
83-43-0D-9D-07-6F-AA-B7-BC-79-CD-6F-AD-7B-FA-EA-19-D1-24-44
```

6.

Bảo vệ file bằng phép mật hóa đối xứng

?

Bạn cần mật hóa một file bằng giải thuật mật hóa đối xứng (*symmetric encryption*).

✕

Trước hết, bạn phải thể hiện hóa một trong các lớp giải thuật đối xứng cụ thể dẫn xuất từ lớp `System.Security.Cryptography.SymmetricAlgorithm`. Sau đó, gọi phương thức `CreateEncryptor` hay `CreateDecryptor` của đối tượng `SymmetricAlgorithm` để thu lấy một đối tượng có hiện thực giao diện `System.Security.Cryptography.ICryptoTransform`. Sử dụng đối tượng `ICryptoTransform` này kết hợp với một đối tượng `System.Security.Cryptography.CryptoStream` để mật hóa hay giải mật hóa dữ liệu đọc từ một file (được truy xuất bằng một đối tượng `System.IO.FileStream`).

Lớp trừu tượng `SymmetricAlgorithm` cung cấp một lớp cơ sở để tất cả các hiện thực giải thuật đối xứng cụ thể dẫn xuất từ đó. Thư viện lớp *.NET Framework* có bốn hiện thực giải thuật đối xứng cụ thể được liệt kê trong bảng 14.3, mỗi lớp là một thành viên của không gian tên `System.Security.Cryptography`. Các lớp có đuôi là `CryptoServiceProvider` bọc lấy các chức năng do *Win32 CryptoAPI* cung cấp, trong khi các lớp có đuôi là `Managed` (hiện tại chỉ có `RijndaelManaged`) được hiện thực hoàn toàn bằng mã lệnh được-quản-lý. Bảng này cũng cho thấy chiều dài khóa mà mỗi giải thuật hỗ trợ (chiều dài mặc định được in đậm). Nói chung, khóa càng dài, càng khó giải mật hóa ciphertext nếu không có khóa, nhưng cũng có nhiều yếu tố khác cần xem xét.

Bảng 14.3 Các hiện thực giải thuật đối xứng

Tên giải thuật	Tên lớp	Chiều dài khóa (bit)
----------------	---------	----------------------

<i>DES</i>	DESCryptoServiceProvider	64
<i>TripleDES</i> hay <i>3DES</i>	TripleDESCryptoServiceProvider	128, 192
<i>RC2</i>	RC2CryptoServiceProvider	40, 48 56, 64, 72, 80, 88, 96, 104, 112, 120, 128
<i>Rijndael</i>	RijndaelManaged	128, 192, 256

Mặc dù bạn có thể tạo ra các thể hiện của các lớp giải thuật đối xứng một cách trực tiếp, lớp cơ sở `SymmetricAlgorithm` là một *factory* cho các lớp hiện thực cụ thể dẫn xuất từ đó. Gọi phương thức tĩnh `SymmetricAlgorithm.Create` với đối số là tên giải thuật sẽ trả về một đối tượng thuộc kiểu đã được chỉ định. Sử dụng *factory* cho phép bạn viết mã lệnh tổng quát, và mã lệnh này có thể làm việc với bất kỳ hiện thực giải thuật đối xứng nào:

```
string algName = "3DES";
SymmetricAlgorithm alg = SymmetricAlgorithm.Create(algName);
```

 **Nếu bạn gọi `SymmetricAlgorithm.Create` và không chỉ định tên giải thuật, `SymmetricAlgorithm` sẽ trả về một đối tượng `RijndaelManaged`. Nếu bạn chỉ định một giá trị không hợp lệ, `SymmetricAlgorithm` sẽ trả về `null`. Bạn có thể cấu hình các ánh xạ tên/lớp mới bằng file cấu hình (xem tài liệu *.NET Framework SDK* để biết thêm chi tiết).**

Trước khi mật hóa dữ liệu với một trong các lớp giải thuật đối xứng, bạn cần một khóa (*key*) và một vector khởi động (*initialization vector*). Khóa là thông tin bí mật dùng để mật hóa và giải mật hóa dữ liệu. Vector khởi động là dữ liệu ngẫu nhiên được truyền cho giải thuật mật hóa. Bạn phải sử dụng cùng khóa và vector khởi động cho cả mật hóa và giải mật hóa dữ liệu. Tuy nhiên, chỉ có khóa là cần phải được giữ bí mật, bạn có thể lưu trữ hay gửi vector khởi động cùng với dữ liệu đã-được-mật-hóa.

Khóa cho mỗi lớp dẫn xuất từ `SymmetricAlgorithm` có thể được truy xuất thông qua thuộc tính `Key`, và vector khởi động có thể được truy xuất thông qua thuộc tính `IV`. Cách đơn giản nhất và ít lỗi nhất để tạo khóa và vector khởi động mới là để lớp tự tạo chúng giùm bạn. Sau khi đã tạo một đối tượng giải thuật đối xứng, nếu bạn không thiết lập các thuộc tính `Key` và `IV` cho nó, đối tượng này sẽ tự động tạo ra các giá trị mới ngay khi bạn cho gọi một thành viên có sử dụng các giá trị `Key` và `IV`. Một khi đã được thiết lập, đối tượng giải thuật đối xứng sẽ tiếp tục sử dụng các giá trị `Key` và `IV` này. Để thay đổi giá trị của `Key` và `IV`, bạn có thể gán trực tiếp các giá trị mới hoặc gọi phương thức `GenerateKey` và `GenerateIV` (buộc đối tượng giải thuật đối xứng tạo ra các giá trị ngẫu nhiên mới).

Bạn không thể trực tiếp thực hiện mật hóa và giải mật hóa với một đối tượng giải thuật đối xứng. Một khi đã tạo và cấu hình đối tượng giải thuật đối xứng, bạn phải gọi phương thức `CreateEncryptor` hay `CreateDecryptor` của nó để thu lấy một đối tượng có hiện thực giao diện `System.Security.Cryptography.ICryptoTransform`. Kế đó, bạn có thể sử dụng các phương thức của đối tượng `ICryptoTransform` này để mật hóa và giải mật hóa dữ liệu. Tuy nhiên, đối tượng `ICryptoTransform` yêu cầu bạn truyền dữ liệu theo từng khối (có kích thước cố định) và lấp (bằng tay) khối dữ liệu cuối cùng vì khối này ít khi có kích thước đúng.

Giao diện `ICryptoTransform` không quá khó sử dụng, nhưng không mấy thân thiện; do vậy *.NET Framework* kèm thêm lớp `System.Security.Cryptography.CryptoStream`. Đây là lớp dẫn xuất từ `System.IO.Stream`, dùng để đơn giản hóa việc mật hóa và giải mật hóa dữ liệu được đọc từ các đối tượng `Stream` khác. Lớp này cho phép bạn mật hóa và giải mật hóa dữ liệu từ các file và các kết nối mạng một cách dễ dàng bằng một mô hình xử lý quen thuộc, và nó cung cấp cho bạn tất cả các tiện ích quen thuộc khi truy xuất dữ liệu dựa-vào-`Stream`.

Phương thức khởi dựng của `CryptoStream` yêu cầu ba đối số: một `Stream` nằm dưới, một thể hiện của `ICryptoTransform`, và một giá trị thuộc kiểu liệt kê `System.Security.Cryptography.CryptoStreamMode`. Giá trị `CryptoStreamMode` cho biết chế độ của đối tượng `CryptoStream` mới; các giá trị hợp lệ là `Read` và `Write`. Khi bạn gọi phương thức `Read` hay `Write` của `CryptoStream`, `CryptoStream` sẽ sử dụng thể hiện `ICryptoTransform` để mật hóa và giải mật hóa dữ liệu đang truyền qua `CryptoStream`. Đối tượng `CryptoStream` bảo đảm kích thước khối dùng cho thể hiện `ICryptoTransform` luôn đúng.

Cấu hình của một đối tượng `CryptoStream` có tính linh hoạt cao, nhưng có thể hơi khó hiểu. Bảng 14.4 mô tả hoạt động của một đối tượng `CryptoStream` dựa trên chế độ của `CryptoStream` và kiểu thể hiện `ICryptoTransform` được sử dụng trong phương thức khởi dựng của `CryptoStream`.

Bảng 14.4 Hoạt động của đối tượng `CryptoStream`

Chế độ của <code>CryptoStream</code>	Chỉ thị của <code>ICryptoTransform</code>	Mô tả
Read	Mật hóa	<code>Stream</code> nằm dưới chứa plaintext nguồn. <code>CryptoStream.Read</code> ghi ciphertext ra bộ đệm xuất.
Read	Giải mật hóa	<code>Stream</code> nằm dưới chứa ciphertext nguồn. <code>CryptoStream.Read</code> ghi plaintext ra bộ đệm xuất.
Write	Mật hóa	<code>CryptoStream.Write</code> chỉ định plaintext cần mật hóa. <code>Stream</code> nằm dưới nhận ciphertext đã-được-mật-hóa.
Write	Giải mật hóa	<code>CryptoStream.Write</code> chỉ định ciphertext cần giải mật hóa. <code>Stream</code> nằm dưới nhận plaintext đã-được-giải-mật-hóa.

Lớp `SymmetricEncryptionExample` dưới đây trình bày cách sử dụng giải thuật *Triple DES* để mật hóa một file và rồi giải mật hóa file đó. Phương thức `Main` nhận tên của file cần mật hóa làm đối số dòng lệnh. Trước tiên, nó sẽ tạo khóa và vectơ khởi động; sau đó, gọi phương thức `EncryptFile`, kế tiếp là phương thức `DecryptFile`, và sinh ra hai file: file thứ nhất chứa phiên bản đã-được-mật-hóa của file nguồn, file thứ hai chứa phiên bản đã-được-giải-mật-hóa của file đã-được-mật-hóa (giống file nguồn).

```
using System;
using System.IO;
using System.Security.Cryptography;
```

```

public class SymmetricEncryptionExample {

    public static void Main(string[] args) {

        // Tạo một giải thuật Triple DES mới để thu lấy khóa dùng cho
        // ví dụ này. Khóa này sẽ được dùng chung trong các phương thức
        // EncryptFile và DecryptFile. Bình thường, khóa được
        // thỏa thuận giữa người gửi và người nhận, hoặc được gửi
        // (bởi người gửi) cùng với file đã-được-mật-hóa.
        byte[] key;
        byte[] iv;

        using(SymmetricAlgorithm alg =
            SymmetricAlgorithm.Create("3DES")){

            key = alg.Key;
            iv = alg.IV;
        }

        // Mật hóa file. Tiền tố "encrypted" sẽ được thêm vào tên file
        // nguồn và được sử dụng làm tên của file đã-được-mật-hóa.
        EncryptFile(args[0], "encrypted"+args[0], (byte[])key.Clone(),
            (byte[])iv.Clone());

        // Giải mật hóa file đã-được-mật-hóa. Tiền tố "decrypted" sẽ được
        // thêm vào tên file gốc và được sử dụng làm tên của file
        // đã-được-giải-mật-hóa.
        DecryptFile("encrypted"+args[0], "decrypted"+args[0], key, iv);
    }

    // Phương thức dùng để mật hóa một file (bằng giải thuật Triple DES)
    // với key và iv cho trước.
    private static void EncryptFile(string srcFileName,
        string destFileName, byte[] key, byte[] iv) {

        // Tạo các stream để truy xuất file nguồn và file đích.
        Stream srcFile =
            new FileStream(srcFileName, FileMode.Open, FileAccess.Read);
    }

```

```
Stream destFile =
    new FileStream(destFileName, FileMode.Create,
        FileAccess.Write);

// Tạo một giải thuật Triple DES mới để mật hóa file.
using(SymmetricAlgorithm alg =
    SymmetricAlgorithm.Create("3DES")){

    // Cấu hình thuộc tính Key và IV của giải thuật.
    alg.Key = key;
    alg.IV = iv;

    // Tạo một CryptoStream để mật hóa nội dung của
    // Stream nguồn khi nó được đọc. Gọi phương thức
    // CreateEncryptor của SymmetricAlgorithm
    // để nhận thể hiện ICryptoTransform và
    // truyền nó cho CryptoStream.
    CryptoStream cryptoStream = new CryptoStream(srcFile,
                                                alg.CreateEncryptor(),
                                                CryptoStreamMode.Read);

    // Khai báo bộ đệm dùng để đọc dữ liệu từ file nguồn
    // thông qua CryptoStream và ghi nó ra file đích.
    int bufferLength;
    byte[] buffer = new byte[1024];

    // Đọc file nguồn (từng khối 1024 byte) và ghi phiên bản
    // đã-được-mật-hóa ra file đích.
    do {
        bufferLength = cryptoStream.Read(buffer, 0, 1024);
        destFile.Write(buffer, 0, bufferLength);
    } while (bufferLength > 0);

    // Đóng stream và xóa các dữ liệu bí mật.
    destFile.Flush();
    Array.Clear(key, 0, key.Length);
```

```

        Array.Clear(iv, 0, iv.Length);
        cryptoStream.Clear();
        cryptoStream.Close();
        srcFile.Close();
        destFile.Close();
    }
}

// Phương thức dùng để giải mật hóa một file đã-được-mật-hóa bằng
// giải thuật Triple DES với key và iv cho trước.
private static void DecryptFile(string srcFileName,
    string destFileName, byte[] key, byte[] iv) {

    // Tạo các stream để truy xuất file nguồn và file đích.
    Stream srcFile =
        new FileStream(srcFileName, FileMode.Open, FileAccess.Read);
    Stream destFile =
        new FileStream(destFileName, FileMode.Create,
            FileAccess.Write);

    // Tạo một giải thuật Triple DES mới để giải mật hóa file.
    using (SymmetricAlgorithm alg =
        SymmetricAlgorithm.Create("3DES")) {

        // Cấu hình thuộc tính Key và IV của giải thuật.
        alg.Key = key;
        alg.IV = iv;

        // Tạo một CryptoStream để giải mật hóa nội dung của dữ liệu
        // đã-được-mật-hóa khi nó được ghi. Gọi phương thức
        // CreateDecryptor của SymmetricAlgorithm để nhận thể hiện
        // ICryptoTransform và truyền nó cho CryptoStream.
        CryptoStream cryptoStream = new CryptoStream(destFile,
            alg.CreateDecryptor(),
            CryptoStreamMode.Write);

        // Khai báo bộ đệm dùng để đọc dữ liệu từ file đã-được-
        // mật-hóa và ghi ra file đích thông qua CryptoStream.

```

```

int bufferLength;
byte[] buffer = new byte[1024];

// Đọc file đã-được-mật-hóa (từng khối 1024 byte) và ghi
// phiên bản đã-được-giải-mật-hóa ra file đích.
do {
    bufferLength = srcFile.Read(buffer, 0, 1024);
    cryptoStream.Write(buffer, 0, bufferLength);
} while (bufferLength > 0);

// Đóng stream và xóa các dữ liệu bí mật.
cryptoStream.FlushFinalBlock();
Array.Clear(key, 0, key.Length);
Array.Clear(iv, 0, iv.Length);
cryptoStream.Clear();
cryptoStream.Close();
srcFile.Close();
destFile.Close();
}
}
}

```

7.

Truy lại khóa đối xứng từ password



Bạn cần tạo một khóa đối xứng từ một password để người dùng chỉ cần nhớ password và không cần lưu trữ khóa.



Sử dụng lớp `System.Security.Cryptography.PasswordDeriveBytes` để tạo khóa đối xứng từ chuỗi password.

Hiếm có người nào nhớ được giá trị của một khóa đối xứng, và không thực tế khi bắt người dùng nhập các số dài như thế bằng tay. Điều này nghĩa là khóa phải được lưu trữ ở một dạng an toàn sao cho ứng dụng có thể truy xuất được (trong smart card, đĩa mềm, cơ sở dữ liệu hay file). Vấn đề liên quan với việc cấp, phân bổ, truy xuất, và lưu trữ khóa là một trong những mặt khó nhất trong quá trình hiện thực bất kỳ giải pháp mật mã nào (vấn đề này được quy chung về quản lý khóa). Khi cần ghi nhớ một bí mật (khóa), bạn không chỉ lo bảo vệ dữ liệu mà còn phải lo bảo vệ các bí mật dùng để bảo vệ dữ liệu!

Một cách lưu trữ khóa là cấp cho người dùng một password dễ nhớ hơn và sử dụng một giao thức truy lại khóa (*key derivation protocol*) để tạo một khóa đối xứng từ password. Theo đó,

mỗi khi cần mật hóa hay giải mật hóa dữ liệu, người dùng chỉ cần nhập password và máy tính sẽ tạo ra khóa (nếu người dùng nhập cùng password, giao thức này sẽ tạo ra cùng khóa).

 **Truy lại khóa từ những từ hay nhóm từ dễ nhớ làm giảm đáng kể tính ngẫu nhiên của khóa, dẫn đến giảm tính bảo mật được cấp bởi những hàm mật mã có sử dụng khóa đó. Trong trường hợp tồi tệ nhất, hacker có thể đoán được password, và crack dữ liệu của bạn thông qua việc giải mã mật mã (cryptanalysis).**

Thư viện lớp *.NET Framework* có một hiện thực truy lại khóa đối xứng: *PasswordDeriveBytes*. Lớp này sử dụng một giải thuật băm được áp dụng lặp đi lặp lại cho một password để tạo ra một khóa với chiều dài như mong muốn. Khi cấu hình một đối tượng *PasswordDeriveBytes*, bạn có thể chỉ định tên giải thuật băm cũng như số lần lặp (mặc định, *SHA-1* được áp dụng 100 lần). Ngoài ra, bạn cũng cần cung cấp giá trị salt. Salt là dữ liệu ngẫu nhiên mà quá trình truy lại khóa sẽ sử dụng để làm cho khóa tìm được trở nên bền vững hơn đối với các dạng tấn công bằng mật mã. Bạn không cần giữ bí mật cho giá trị salt; bạn phải lưu trữ và sử dụng nó khi truy lại khóa từ password sau này (nếu không có giá trị salt đúng, bạn sẽ không thể truy lại khóa đúng và không thể giải mật hóa được).

 **Bạn không thể tạo khóa bất đối xứng bằng giao thức truy lại khóa. Giải thuật mật hóa bất đối xứng (asymmetric encryption) dựa vào các mối liên hệ toán cụ thể giữa các thành phần khóa công khai (public key) và khóa riêng (private key). Như thế, mỗi giải thuật mật hóa bất đối xứng yêu cầu bạn phải tuân theo một quy trình riêng để có thể tạo ra các khóa mới.**

Ví dụ dưới đây trình bày cách sử dụng lớp *PasswordDeriveBytes* để tạo một khóa đối xứng gồm 64 bit từ một chuỗi password. Ví dụ này yêu cầu hai đối số dòng lệnh: tên giải thuật băm và password (tên của các giải thuật băm đã được liệt kê trong bảng 14.1).

```
using System;

using System;

using System.Security.Cryptography;

public class DerivedKeyExample {

    public static void Main(string[] args) {

        // Sử dụng một bộ tạo số ngẫu nhiên để tạo giá trị salt.
        byte[] salt = new byte[8];

        RandomNumberGenerator.Create().GetBytes(salt);

        // Tạo một đối tượng PasswordDeriveBytes để tạo khóa từ
        // password. Cần cung cấp password nguồn (là đối số dòng
        // lệnh thứ hai) và salt.
        PasswordDeriveBytes pdb =
```

```

        new PasswordDeriveBytes(args[1], salt);

// Thiết lập giải thuật băm dùng để tạo khóa, tên
// giải thuật được chỉ định trong đối số dòng lệnh thứ nhất.
// Giải thuật được sử dụng mặc định là SHA-1.
pdb.HashName = args[0];

// Thiết lập số lần lặp là 200 (đây là số lần giải thuật băm
// được áp dụng cho password để tạo khóa). Mặc định là 100.
pdb.IterationCount = 200;

// Tạo một khóa gồm 8 byte (64 bit) từ password.
// Chiều dài của khóa bị giới hạn bởi chiều dài của
// mã băm - 160 bit đối với SHA-1.
byte[] key = pdb.GetBytes(8);

// Hiển thị khóa và salt.
Console.WriteLine("Key = {0}", BitConverter.ToString(key));
Console.WriteLine("Salt = {0}", BitConverter.ToString(salt));
    }
}

```

Chạy lệnh `DerivedKeyExample SHA1 S0meVereeStr@ngeP@$w0rd` (sử dụng giải thuật băm *SHA-1* để truy lại một khóa gồm 8 byte từ chuỗi "S0meVereeStr@ngeP@\$w0rd") sẽ sinh ra kết xuất tương tự như sau:

```

Key = 53-72-74-5B-A4-88-A4-80
Salt = 70-82-79-F4-3B-F9-DF-D2

```

Chú ý rằng, mỗi khi bạn chạy cùng một lệnh, `DerivedKeyExample` sinh ra khóa khác nhau. Đó là do tác dụng của salt. Nếu bạn bỏ đi dòng lệnh gán một giá trị ngẫu nhiên vào salt (được in đậm trong đoạn mã trên), sau đó biên dịch lại và chạy `DerivedKeyExample`, bạn sẽ nhận thấy ví dụ này luôn tạo ra cùng một khóa với một password cho trước.

8. Gửi một bí mật bằng phép mật hóa bất đối xứng

- ? **Bạn cần sử dụng phép mật hóa bất đối xứng (*asymmetric encryption*) để gửi một bí mật.**
- ✕ **Thể hiện hóa lớp giải thuật bất đối xứng `System.Security.Cryptography.RSACryptoServiceProvider`. Sử dụng phương thức `RSACryptoServiceProvider`.**

Encrypt và khóa công khai (*public key*) của người nhận để mật hóa thông điệp. Sau đó, người nhận sẽ sử dụng phương thức `RSACryptoServiceProvider.Decrypt` và khóa riêng (*private key*) để giải mật hóa bí mật đã-được-mật-hóa.

.NET Framework định nghĩa một hệ thống phân cấp theo lớp cho các giải thuật bất đối xứng tương tự như đã định nghĩa cho các giải thuật đối xứng (đã được thảo luận trong mục 14.6). Tất cả các giải thuật bất đối xứng phải thừa kế một lớp cơ sở trừu tượng chung có tên là `System.Security.Cryptography.AsymmetricAlgorithm`. Có hai hiện thực giải thuật bất đối xứng cụ thể:

- `System.Security.Cryptography.RSACryptoServiceProvider`
- `System.Security.Cryptography.DSACryptoServiceProvider`

Vì có đuôi là `CryptoServiceProvider` nên cả hai lớp này đều bọc lấy các chức năng do *Win32 CryptoAPI* cung cấp. Tuy nhiên, chỉ có lớp `RSACryptoServiceProvider` là hỗ trợ việc mật hóa dữ liệu. Lớp `DSACryptoServiceProvider` hiện thực *Digital Signature Algorithm (DSA)*, bạn có thể sử dụng giải thuật này chỉ để tạo chữ ký số (xem *Federal Information Processing Standard [FIPS] 186-2* tại [<http://www.itl.nist.gov/fipspubs>] để biết thêm chi tiết về *DSA*).

Mặc dù bạn có thể tạo một đối tượng giải thuật bất đối xứng bằng phương thức tĩnh `Create` của lớp cơ sở `AsymmetricAlgorithm`, nhưng bù lại bạn sẽ phải trả giá chút ít cho việc này. Lớp `AsymmetricAlgorithm` không khai báo các phương thức mà `RSACryptoServiceProvider` sử dụng để mật hóa và giải mật hóa dữ liệu. Thay vào đó, bạn phải trực tiếp thể hiện hóa lớp `RSACryptoServiceProvider` bằng một trong các phương thức khởi dựng của nó.

Trước khi mật hóa hay giải mật hóa dữ liệu với đối tượng `RSACryptoServiceProvider`, bạn cần truy xuất các khóa thích hợp. Khóa của giải thuật bất đối xứng khác nhiều so với khóa của giải thuật đối xứng. Thứ nhất, nó có hai thành phần: khóa công khai (*public key*) và khóa riêng (*private key*). Thứ hai, thay vì chỉ là một dãy các byte được sinh ngẫu nhiên, khóa bất đối xứng được tạo theo một cách thức đặc biệt. Có một mối quan hệ toán đặc biệt giữa khóa công khai và khóa riêng; mối quan hệ này cho phép giải thuật bất đối xứng mật hóa dữ liệu bằng một khóa và giải mật hóa dữ liệu bằng một khóa khác. Mỗi giải thuật bất đối xứng sử dụng cách thức tạo khóa của chính nó, và các lớp hiện thực cụ thể đóng gói các chức năng cần thiết để tạo ra các khóa mới.

Khóa công khai không cần được giữ bí mật và chủ sở hữu có thể tùy ý gửi nó cho bạn thông qua e-mail, hoặc post nó lên một website hay một server phân phối khóa để mọi người cùng thấy. Những ai muốn gửi bí mật thì sử dụng khóa công khai để mật hóa bí mật. Sau đó, người nhận sử dụng khóa riêng để giải mật hóa bí mật. Khóa riêng phải được giữ bí mật; những ai sở hữu khóa riêng đều có thể giải mật hóa dữ liệu đã-được-mật-hóa bằng khóa công khai.

Để tạo một bí mật được-mật-hóa-bất-đối-xứng, bạn phải có khóa công khai của người nhận và nạp nó vào một đối tượng `RSACryptoServiceProvider`. Có hai cách nạp khóa công khai:

- Sử dụng phương thức `RSACryptoServiceProvider.ImportParameters` để nhập một cấu trúc `System.Security.Cryptography.RSAParameters`, cấu trúc này chứa thông tin khóa công khai của người nhận. Chủ sở hữu có thể tạo cấu trúc `RSAParameters` bằng phương thức `RSACryptoServiceProvider.ExportParameters` và gửi nó cho bạn. Tuy nhiên, người này có thể gửi cho bạn khóa công khai ở dạng byte, và bạn phải tự nạp giá trị này vào cấu trúc `RSAParameters`.

- Sử dụng phương thức `RSACryptoServiceProvider.FromXmlString` để nạp dữ liệu khóa công khai từ một chuỗi *XML*. Chủ sở hữu có thể tạo dữ liệu *XML* này bằng phương thức `RSACryptoServiceProvider.ToXmlString` và gửi nó cho bạn.



Cả phương thức `ExportParameters` và `ToXmlString` của lớp `RSACryptoServiceProvider` đều nhận một đối số luận lý, nếu là `true`, đối tượng `RSACryptoServiceProvider` sẽ xuất cả khóa công khai và khóa riêng. Bạn chỉ định giá trị này là `false` khi cần xuất khóa cho mục đích phân phối hay lưu trữ.

Một khi đã nạp khóa công khai của người nhận vào đối tượng `RSACryptoServiceProvider`, bạn có thể mật hóa dữ liệu. Giải thuật bất đối xứng chậm hơn giải thuật đối xứng khi mật hóa và giải mật hóa dữ liệu. Vì lý do này, bạn không nên sử dụng giải thuật bất đối xứng để mật hóa lượng dữ liệu lớn. Thông thường, nếu cần mật hóa lượng dữ liệu lớn, bạn nên sử dụng giải thuật đối xứng và rồi mật hóa khóa đối xứng bằng giải thuật bất đối xứng để bạn có thể gửi khóa đối xứng cùng với dữ liệu. Mục 14.10 sẽ thảo luận về vấn đề này. Để bảo đảm tính nhất quán trong việc sử dụng, lớp `RSACryptoServiceProvider` không hỗ trợ mô hình mật hóa và giải mật hóa dựa-trên-`System.IO.Stream` (đã được sử dụng trong mục 14.6).

Để mật hóa dữ liệu với đối tượng `RSACryptoServiceProvider`, bạn hãy gọi phương thức `Encrypt`, truyền cho nó một mảng byte chứa plaintext; `Encrypt` sẽ trả về một mảng byte chứa ciphertext. Phương thức `Encrypt` cũng nhận một đối số luận lý cho biết kiểu *padding* mà đối tượng `RSACryptoServiceProvider` sẽ sử dụng. *Padding* cho biết đối tượng bất đối xứng sẽ xử lý plaintext như thế nào trước khi mật hóa. *Padding* bảo đảm giải thuật bất đối xứng không cần xử lý từng khối dữ liệu, và bảo vệ ciphertext đối với các dạng tấn công bằng mật mã. Diễn giải các dạng *padding* vượt quá phạm vi của quyền sách này. Nói chung, nếu đang sử dụng *Microsoft Windows XP* trở về sau, bạn nên chỉ định đối số *padding* là `true`; nếu không, bạn phải chỉ định đối số *padding* là `false` (nếu không thì `Encrypt` sẽ ném ngoại lệ `System.Security.Cryptography.CryptographicException`).

Giải mật hóa dữ liệu cũng đơn giản như mật hóa dữ liệu. Người nhận cần tạo một đối tượng `RSACryptoServiceProvider` và nạp nó cùng với khóa riêng. Thông thường, khóa này sẽ được lưu trữ trong một kho chứa khóa (*key container*) do *CryptoAPI* quản lý (sẽ được thảo luận kỹ hơn trong mục 14.9). Người nhận gọi `RSACryptoServiceProvider.Decrypt` và truyền cho nó ciphertext mà bạn đã gửi. Người nhận phải chỉ định cơ chế *padding*, và nó cũng phải giống như khi mật hóa dữ liệu. Phương thức `Decrypt` trả về một mảng byte chứa plaintext đã-được-giải-mật-hóa. Nếu plaintext mô tả một chuỗi, người nhận phải chuyển mảng byte thành giá trị chuỗi thích hợp bằng lớp `System.Text.Encoding`.



Lớp `RSACryptoServiceProvider` thừa kế các phương thức có tên là `EncryptValue` và `DecryptValue` từ lớp cha của nó là `System.Security.Cryptography.RSA`. Lớp `RSACryptoServiceProvider` không hiện thực các phương thức này và ném ngoại lệ `System.NotSupportedException` nếu bạn gọi chúng.

Lớp `AsymmetricEncryptionExample` dưới đây trình bày cách sử dụng lớp `RSACryptoServiceProvider` để mật hóa một chuỗi và rồi giải mật hóa:

```
using System;
```

```
using System.Text;
using System.Security.Cryptography;

public class AsymmetricEncryptionExample {

    public static void Main(string[] args) {

        // Khai báo một biến RSAParameters, biến này sẽ chứa
        // thông tin PUBLIC KEY của người nhận.
        RSAParameters recipientsPublicKey;

        // Khai báo một biến CspParameters, biến này sẽ cho biết
        // PRIVATE KEY được lưu trữ trong kho chứa khóa nào.
        // Thông thường, chỉ có người nhận mới có thể truy xuất
        // thông tin này. Với mục đích minh họa, chúng ta sẽ tạo
        // một cặp khóa ngay đầu ví dụ và sử dụng các khóa này
        // cho cả bên gửi và bên nhận.
        CspParameters cspParams = new CspParameters();
        cspParams.KeyContainerName = "MyKeys";

        // Tạo cặp khóa bất đối xứng bằng lớp RSACryptoServiceProvider.
        // Lưu các khóa này vào một kho chứa khóa có tên là "MyKeys"
        // và trích thông tin PUBLIC KEY vào biến recipientsPublicKey.
        using (RSACryptoServiceProvider rsaAlg =
            new RSACryptoServiceProvider(cspParams)) {

            // Cấu hình cho giải thuật lưu khóa vào kho chứa khóa.
            rsaAlg.PersistKeyInCsp = true;

            // Trích PUBLIC KEY.
            recipientsPublicKey = rsaAlg.ExportParameters(false);
        }

        // Hiển thị thông điệp plaintext gốc.
        Console.WriteLine("Original message = {0}", args[0]);

        // Chuyển thông điệp gốc thành mảng byte. Tốt nhất là không
        // truyền các thông tin bí mật ở dạng chuỗi giữa các phương thức.
```

```

byte[] plaintext = Encoding.Unicode.GetBytes(args[0]);

// Mật hóa thông điệp bằng phương thức EncryptMessage.
// Phương thức này cần PUBLIC KEY của người nhận.
byte[] ciphertext = EncryptMessage(plaintext,
    recipientsPublicKey);

// Hiển thị ciphertext do phương thức EncryptMessage trả về.
// Sử dụng phương thức BitConverter.ToString method cho đơn giản
// mặc dù nó chèn dấu gạch nối (-) vào giữa các giá trị byte
// (không đúng với biểu diễn dữ liệu trong bộ nhớ).
Console.WriteLine("Formatted Ciphertext = {0}",
    BitConverter.ToString(ciphertext));

// Giải mật hóa thông điệp (đã-được-mật-hóa) bằng phương thức
// DecryptMessage. Phương thức này cần truy xuất PRIVATE KEY
// của người nhận (chỉ có người nhận mới có thể truy xuất được).
// Chúng ta sẽ truyền cho nó một đối tượng CspParameters
// (cho biết PRIVATE KEY được lưu trữ trong kho chứa khóa nào).
// Giải pháp này an toàn hơn là truyền PRIVATE KEY thô
// giữa các phương thức.
byte[] decData = DecryptMessage(ciphertext, cspParams);

// Chuyển thông điệp đã-được-giải-mật-hóa từ mảng byte
// thành chuỗi và hiển thị nó ra cửa sổ Console.
Console.WriteLine("Decrypted message = {0}",
    Encoding.Unicode.GetString(decData));

// Nhấn Enter để kết thúc.
Console.ReadLine();
}

// Phương thức dùng để mật hóa (theo RSA) một thông điệp bằng
// PUBLIC KEY (nằm trong một cấu trúc RSAParameters).
private static byte[] EncryptMessage(byte[] plaintext,
    RSAParameters rsaParams) {

```

```

// Khai báo mảng byte chứa ciphertext.
byte[] ciphertext = null;

// Tạo một thể hiện của giải thuật RSA.
using (RSACryptoServiceProvider rsaAlg =
    new RSACryptoServiceProvider()) {

    rsaAlg.ImportParameters(rsaParams);

    // Mật hóa plaintext bằng OAEP padding
    // (chỉ được hỗ trợ trên Windows XP trở về sau).
    ciphertext = rsaAlg.Encrypt(plaintext, true);
}

// Xóa các giá trị được giữ trong mảng byte chứa plaintext.
// Điều này bảo đảm dữ liệu bí mật không còn trong bộ nhớ
// sau khi bạn giải phóng tham chiếu đến nó.
Array.Clear(plaintext, 0, plaintext.Length);

return ciphertext;
}

// Phương thức dùng để giải mật hóa một thông điệp (đã-được-mật-hóa-
// theo-RSA) bằng PRIVATE KEY (do đối tượng CspParameters chỉ định).
private static byte[] DecryptMessage(byte[] ciphertext,
    CspParameters cspParams ) {

    // Khai báo mảng byte chứa plaintext (đã-được-giải-mật-hóa).
    byte[] plaintext = null;

    // Tạo một thể hiện của giải thuật RSA.
    using (RSACryptoServiceProvider rsaAlg =
        new RSACryptoServiceProvider(cspParams)) {

        // Giải mật hóa plaintext bằng OAEP padding.
        plaintext = rsaAlg.Decrypt(ciphertext, true);
    }
}

```

```

        return plaintext;
    }
}

```

Lệnh `AsymmetricEncryptionExample` "I love you!" sẽ sinh ra kết xuất tương tự như sau:

Original message = I love you!

Formatted Ciphertext = 1F-53-05-2B-9D-CC-20-6B-5D-D3-D4-0B-C9-5F-CA-FA-C1-61-6C-3B-5B-9E-EA-B9-D0-AF-E5-2B-05-BC-D4-94-DD-71-D6-21-2A-B0-82-6B-16-C0-89-3E-24-B3-B3-A3-15-FE-16-7A-B0-58-14-43-CD-69-1A-FD-08-39-2D-09-A6-41-86-96-78-B4-3D-D6-C7-39-8A-90-84-D6-68-E6-5D-86-32-14-67-51-A7-B7-5A-EF-CF-F4-6D-E4-B0-18-6A-16-2A-AF-54-B7-3C-B8-19-6E-A5-86-BF-3E-B2-6D-17-E3-1D-E8-AD-D1-A8-D9-54-93-8E-F1-E8-5D-AC-4A

Decrypted message = I love you!



Nếu bạn chạy ví dụ này nhiều lần với cùng thông điệp và khóa, ciphertext sẽ khác nhau. Đó là vì cơ chế *padding* sinh ra dữ liệu ngẫu nhiên để tránh các dạng tấn công bằng mật mã. Mặc dù hơi rắc rối nhưng đây chính là cách hành xử mà ta mong đợi.

9.

Lưu trữ khóa bất đối xứng một cách an toàn



Bạn cần lưu trữ cặp khóa bất đối xứng vào một nơi an toàn để ứng dụng của bạn có thể truy xuất được dễ dàng.



Dựa vào chức năng lưu trữ khóa do hai lớp giải thuật bất đối xứng cung cấp (`RSACryptoServiceProvider` và `DSACryptoServiceProvider`—thuộc không gian tên `System.Security.Cryptography`).

Cả hai lớp giải thuật bất đối xứng—`RSACryptoServiceProvider` và `DSACryptoServiceProvider`—đều bọc lấy các chức năng do *CSP* (*Cryptographic Service Provider*—một thành phần của *Win32 CryptoAPI*) hiện thực. Ngoài các dịch vụ như mật hóa, giải mật hóa, và chữ ký số, mỗi *CSP* còn cung cấp một kho chứa khóa (*key container*).

Kho chứa khóa là vùng lưu trữ dành cho các khóa mà *CSP* quản lý; *CSP* sử dụng cơ chế bảo mật của hệ điều hành và phép mật hóa mạnh để bảo vệ nội dung của kho chứa khóa. Kho chứa khóa cho phép ứng dụng dễ dàng truy xuất khóa mà không ảnh hưởng đến tính bảo mật của khóa. Khi gọi các hàm của một *CSP*, ứng dụng cần chỉ định tên của kho chứa khóa và *CSP* sẽ truy xuất các khóa cần thiết. Vì khóa không truyền từ *CSP* đến ứng dụng nên ứng dụng không thể làm hại tính bảo mật của khóa.

Lớp `RSACryptoServiceProvider` và `DSACryptoServiceProvider` cho phép bạn cấu hình hiện thực *CSP* nằm dưới bằng một thể hiện của lớp `System.Security.Cryptography.CspParameters`. Để cấu hình cho một đối tượng `RSACryptoServiceProvider` hay `DSACryptoServiceProvider` sử dụng một kho chứa khóa cụ thể, bạn phải hoàn tất các bước dưới đây:

1. Tạo một đối tượng `CspParameters`.

2. Thiết lập trường `KeyContainerName` của đối tượng `CspParameters` là một giá trị chuỗi mô tả tên của kho chứa khóa cần sử dụng; chuỗi có thể chứa khoảng trắng.
3. Tạo một đối tượng `RSACryptoServiceProvider` hay `DSACryptoServiceProvider`, và truyền đối tượng `CspParameters` làm đối số cho phương thức khởi dựng.

Nếu kho chứa khóa tồn tại bên trong tầm vực của *CSP* và chứa các khóa thích hợp, *CSP* sẽ sử dụng các khóa này khi thực hiện các thao tác mật mã. Nếu kho chứa khóa hay khóa không tồn tại, *CSP* sẽ tự động tạo khóa mới. Để buộc *CSP* lưu trữ các khóa mới được tạo vào kho chứa khóa, bạn phải thiết lập thuộc tính `PersistKeyInCsp` (của đối tượng `RSACryptoServiceProvider` hay `DSACryptoServiceProvider`) là `true`.

Phương thức `LoadKeys` dưới đây là một đoạn trích trong file *StoreAsymmetricKeyExample.cs* (xem đĩa CD đính kèm). `LoadKeys` tạo một đối tượng `RSACryptoServiceProvider` và cấu hình cho nó sử dụng một kho chứa khóa có tên là `MyKeys`. Bằng cách chỉ định `PersistKeyInCsp` là `true`, giải thuật sẽ tự động lưu trữ các khóa mới được tạo vào kho chứa khóa này.

```
// Phương thức này tạo một đối tượng RSACryptoServiceProvider
// và nạp các khóa từ một kho chứa khóa nếu chúng tồn tại; nếu không,
// RSACryptoServiceProvider sẽ tự động tạo các khóa mới và lưu
// chúng vào kho chứa khóa để sử dụng sau này.
public static void LoadKeys(string container) {

    // Tạo một đối tượng CspParameters và thiết lập trường
    // KeyContainerName là tên của kho chứa khóa.
    System.Security.Cryptography.CspParameters cspParams =
        new System.Security.Cryptography.CspParameters();
    cspParams.KeyContainerName = container;

    // Tạo một đối tượng giải thuật RSA và truyền đối tượng
    // CspParameters làm đối số trong phương thức khởi dựng.
    using (System.Security.Cryptography.RSACryptoServiceProvider
        rsaAlg = new
            System.Security.Cryptography.RSACryptoServiceProvider(cspParams)) {

        // Cấu hình cho đối tượng RSACryptoServiceProvider
        // lưu trữ khóa vào kho chứa khóa.
        rsaAlg.PersistKeyInCsp = true;

        // Hiển thị PUBLIC KEY.
        System.Console.WriteLine(rsaAlg.ToXmlString(false));
    }
}
```

Lớp `RSACryptoServiceProvider` và `DSACryptoServiceProvider` không cung cấp phương thức nào trực tiếp gỡ bỏ kho chứa khóa. Để xóa các khóa đã được lưu trữ, bạn hãy thiết lập giá trị của `PersistKeyInCsp` là `false` và gọi phương thức `Clear` hay `Dispose` của đối tượng `RSACryptoServiceProvider` hay `DSACryptoServiceProvider`. Phương thức `DeleteKeys` dưới đây sẽ trình bày kỹ thuật này:

```
// Phương thức này tạo một đối tượng RSACryptoServiceProvider
// và xóa các khóa hiện có khỏi kho chứa khóa.
public static void DeleteKeys(string container) {

    // Tạo một đối tượng CspParameters và thiết lập trường
    // KeyContainerName là tên của kho chứa khóa cần xóa.
    System.Security.Cryptography.CspParameters cspParams =
        new System.Security.Cryptography.CspParameters();
    cspParams.KeyContainerName = container;

    // Tạo một đối tượng giải thuật RSA và truyền đối tượng
    // CspParameters làm đối số trong phương thức khởi dựng.
    using (System.Security.Cryptography.RSACryptoServiceProvider
        rsaAlg = new
            System.Security.Cryptography.RSACryptoServiceProvider(cspParams)) {

        // Cấu hình cho đối tượng RSACryptoServiceProvider
        // không lưu trữ khóa vào kho chứa khóa.
        rsaAlg.PersistKeyInCsp = false;

        // Hiển thị PUBLIC KEY. Vì chúng ta gọi Dispose()
        // sau lời gọi này nên các khóa hiện có sẽ không thay đổi
        // cho đến khi phương thức được gọi lần thứ hai.
        System.Console.WriteLine(rsaAlg.ToXmlString(false));

        // Vì mã lệnh nằm trong khối "using" nên Dispose được gọi
        // trên đối tượng RSACryptoServiceProvider. Vì đối tượng
        // này được cấu hình là không lưu trữ khóa nên kho chứa khóa
        // sẽ bị xóa. Thay vì gọi Dispose(), gọi rsaAlg.Clear()
        // sẽ có cùng tác dụng, vì nó gián tiếp gọi Dispose().
    }
}
```


}

Win32 CryptoAPI hỗ trợ cả *user-key-store* và *machine-key-store*. Hệ điều hành Windows bảo đảm một *user-key-store* chỉ có thể được truy xuất bởi người đã tạo ra nó, nhưng một *machine-key-store* có thể được truy xuất bởi bất kỳ người dùng nào của máy. Theo mặc định, lớp `RSACryptoServiceProvider` và `DSACryptoServiceProvider` sẽ sử dụng *user-key-store*. Bạn có thể chỉ định sử dụng *machine-key-store* bằng cách thiết lập thuộc tính tĩnh `UseMachineKeyStore` của lớp `RSACryptoServiceProvider` hay `DSACryptoServiceProvider` là `true`. Điều này sẽ có tác dụng với tất cả mã lệnh đang chạy trong miền ứng dụng hiện hành. Nếu muốn kiểm soát chặt chẽ hơn, bạn có thể thiết lập thuộc tính `CspParameters.Flags` là giá trị `System.Security.Cryptography.CspProviderFlags.UseMachineKeyStore` trước khi tạo đối tượng mật hóa bất đối xứng.



Bạn nên xét các yêu cầu bảo mật một cách cẩn thận trước khi chọn sử dụng *machine-key-store*. Thực tế là người dùng nào có quyền truy xuất máy đều có thể giành được quyền truy xuất các khóa trong kho lưu trữ, điều này phủ định hầu hết các lợi ích do phép mật hóa bất đối xứng mang lại.

10.

Trao đổi khóa phiên đối xứng một cách an toàn

?

Bạn cần trao đổi dữ liệu đã-được-mật-hóa-đối-xứng với ai đó, và bạn cần một biện pháp an toàn để phân bổ khóa phiên (*session key*) đối xứng cùng với dữ liệu.

✕

Sử dụng cơ chế trao đổi khóa do lớp `System.Security.Cryptography.RSACryptoServiceProvider` hiện thực. Theo cơ chế này, khóa đối xứng sẽ được mật hóa bất đối xứng bằng khóa công khai (*public key*) của người nhận. Theo đó, bạn có thể gửi khóa đối xứng đã-được-mật-hóa cùng với dữ liệu đã-được-mật-hóa. Người nhận phải giải mật hóa khóa đối xứng bằng khóa riêng (*private key*), rồi mới tiến hành giải mật hóa dữ liệu.

Mỗi khi mật hóa dữ liệu (bằng giải thuật đối xứng) để chuyển giao, bạn nên tạo một khóa mới, được gọi là khóa phiên (*session key*). Sử dụng khóa phiên có hai lợi ích chính:

- Nếu ai đó (không được phép) lấy được nhiều khối ciphertext đã được mật hóa bằng cùng một khóa đối xứng, khả năng người đó giải được dữ liệu sẽ tăng cao.
- Nếu ai đó tìm được khóa phiên của bạn, người này chỉ có thể truy xuất được một tập dữ liệu nào đó đã-được-mật-hóa, chứ không phải tất cả các bí mật của bạn ở quá khứ và tương lai.

Vấn đề đối với khóa phiên là phân bổ và bảo mật khóa. Một giải pháp là thỏa thuận một lượng lớn khóa phiên với những người mà bạn cần trao đổi dữ liệu với họ. Thật không may, việc này nhanh chóng trở nên khó quản lý; và thực tế là tất cả các khóa của bạn trong tương lai đều được lưu trữ tại một nơi nào đó, điều này tăng khả năng chúng sẽ bị xâm hại. Cách tốt hơn là gửi khóa phiên theo một dạng được mật hóa mạnh cùng với dữ liệu mà bạn đã mật hóa với khóa đó—quá trình này được gọi là trao đổi khóa (*key exchange*).

Quá trình trao đổi khóa sử dụng phép mật hóa bất đối xứng để mật hóa khóa phiên đối xứng. Nếu muốn gửi dữ liệu cho ai đó, bạn tạo một khóa phiên đối xứng, mật hóa dữ liệu, và rồi mật hóa khóa phiên bằng khóa công khai của người nhận. Khi nhận được dữ liệu, người nhận giải mật hóa khóa phiên bằng khóa riêng của họ, và rồi giải mật hóa dữ liệu. Quan trọng là việc

trao đổi khóa cho phép bạn trao đổi các lượng dữ liệu lớn (đã-được-mật-hóa) với bất cứ ai, thậm chí những người bạn chưa từng tiếp xúc trước đây, miễn là bạn có thể truy xuất khóa công khai đối xứng của họ.

 **Một cách lý tưởng, bạn sử dụng một giải thuật bất đối xứng để mật hóa tất cả dữ liệu, như thế tránh được nhu cầu trao đổi các khóa đối xứng. Tuy nhiên, tốc độ của các giải thuật bất đối xứng khi mật hóa và giải mật hóa dữ liệu khiến chúng không thực tế cho việc sử dụng với các lượng lớn dữ liệu. Sử dụng các giải thuật bất đối xứng để mật hóa các khóa phiên đối xứng là một giải pháp tuy phức tạp hơn, nhưng là tốt nhất ở cả hai mặt: tính linh hoạt và tính nhanh chóng.**

Thư viện lớp *.NET Framework* hỗ trợ việc trao đổi khóa chỉ với giải thuật *RSA*, nhưng bạn phải lựa chọn giữa hai *formatting scheme*: *Optimal Asymmetric Encryption Padding (OAEP)* và *PKCS #1 v 1.5*. Bàn về các *formatting scheme* này vượt quá phạm vi của quyển sách này. Nói chung, bạn nên sử dụng *OAEP formatting* trừ khi bạn có nhu cầu giao tiếp với một hệ thống cũ có sử dụng *PKCS formatting*. Hai lớp dưới đây hiện thực cơ chế trao đổi khóa, mỗi cơ chế ứng với một *formatting scheme*:

- `System.Security.Cryptography.RSAOAEPKeyExchangeFormatter`
- `System.Security.Cryptography.RSAPKCS1KeyExchangeFormatter`

Để chuẩn bị một khóa đối xứng dùng cho trao đổi, bạn phải tạo một đối tượng *formatter* với kiểu như mong muốn và rồi ấn định một đối tượng giải thuật bất đối xứng (`RSACryptoServiceProvider`) cho *formatter* bằng phương thức `SetKey` của *formatter*. Bạn phải cấu hình cho giải thuật bất đối xứng sử dụng khóa công khai của người nhận. Khi đã cấu hình xong, gọi phương thức `CreateKeyExchange` của *formatter* và truyền một mảng byte chứa khóa phiên đối xứng mà bạn cần định dạng. Phương thức `CreateKeyExchange` trả về một mảng byte chứa dữ liệu bạn sẽ gửi đi.

Giải định dạng cho khóa ngược với quá trình định dạng. Có hai lớp *deformatter*, mỗi lớp ứng với một *formatting scheme*.

- `System.Security.Cryptography.RSAOAEPKeyExchangeDeformatter`
- `System.Security.Cryptography.RSAPKCS1KeyExchangeDeformatter`

Để giải định dạng một khóa phiên đã được định dạng, hãy tạo một đối tượng *deformatter* với kiểu phù hợp rồi gọi phương thức `SetKey` của nó để ấn định một đối tượng giải thuật bất đối xứng. Bạn phải nạp khóa riêng của bạn vào giải thuật bất đối xứng. Cuối cùng, gọi phương thức `DecryptKeyExchange` với đối số là dữ liệu trao đổi. Phương thức này trả về một mảng byte chứa khóa phiên đối xứng gốc.

File *KeyExchangeExample.cs* chứa ví dụ minh họa cho việc trao đổi khóa. Phương thức `Main` mô phỏng việc tạo, định dạng, trao đổi, và giải định dạng một khóa phiên đối xứng. Nó sẽ tạo một cặp khóa bất đối xứng để sử dụng cho cả ví dụ này. Thực tế, người gửi (người tạo khóa đối xứng) chỉ có khóa công khai của người nhận; người nhận có khóa riêng (được giữ bí mật).



Cũng như nên sử dụng một khóa đối xứng có chiều dài phù hợp với tính bí mật của dữ liệu đang được bảo vệ, bạn nên mật hóa khóa phiên bằng một giải thuật bất đối xứng và chiều dài khóa ít nhất cũng phải tương đương với khóa đối xứng. Nếu khóa bất đối xứng yếu hơn khóa đối xứng, có khả năng kẻ tấn công sẽ phá vỡ giải thuật bất đối xứng và thu lấy khóa đối xứng thay vì cố giải mật hóa dữ liệu đã-được-mật-hóa-đối-xứng. Xem [<http://www.ietf.org/rfc/rfc3766.txt>] (có trong đĩa CD đính kèm) để biết chi tiết về sự tương đương giữa chiều dài khóa đối xứng và bất đối xứng.

Kế đó, phương thức `Main` gọi phương thức `FormatKeyExchange` với đối số là một mảng byte chứa khóa đối xứng và một đối tượng `RSAParameters` chứa khóa công khai của người nhận. Phương thức `FormatKeyExchange` trả về một mảng byte chứa khóa đối xứng đã-được-mật-hóa và đã-được-định-dạng, chuẩn bị gửi đi. Kế tiếp, phương thức `Main` gọi phương thức `DeformatKeyExchange` với đối số là dữ liệu trao đổi đã được định dạng và đối tượng `CspParameters` chứa một tham chiếu đến kho chứa khóa `MyKeys` (chứa khóa riêng của người nhận). Trong suốt quá trình này, phương thức `Main` sẽ hiển thị khóa phiên gốc, dữ liệu trao đổi đã được định dạng, và cuối cùng là khóa phiên đã được giải định dạng.

```
using System;
using System.Text;
using System.Security.Cryptography;

public class KeyExchangeExample {

    public static void Main() {

        // Khai báo một biến RSAParameters, biến này sẽ
        // chứa thông tin PUBLIC KEY của người nhận.
        RSAParameters recipientsPublicKey;

        // Khai báo một biến CspParameters, biến này sẽ cho biết
        // PRIVATE KEY được lưu trữ trong kho chứa khóa nào.
        // Thông thường, chỉ có người nhận mới có thể truy xuất
        // thông tin này. Với mục đích minh họa, chúng ta sẽ tạo
        // một cặp khóa ngay đầu ví dụ và sử dụng các khóa này
        // cho cả bên gửi và bên nhận.
        CspParameters cspParams = new CspParameters();
        cspParams.KeyContainerName = "MyKeys";

        // Tạo cặp khóa bất đối xứng bằng lớp RSACryptoServiceProvider.
        // Lưu các khóa này vào một kho chứa khóa có tên là "MyKeys"
```

```

// và trích thông tin PUBLIC KEY vào biến recipientsPublicKey.
using (RSACryptoServiceProvider rsaAlg =
    new RSACryptoServiceProvider(cspParams)) {

    // Cấu hình cho giải thuật lưu khóa vào kho chứa khóa.
    rsaAlg.PersistKeyInCsp = true;

    // Trích PUBLIC KEY.
    recipientsPublicKey = rsaAlg.ExportParameters(false);
}

// Tạo giải thuật đối xứng Triple-DES và sử dụng
// khóa được sinh tự động làm khóa phiên.
using (SymmetricAlgorithm symAlg =
    SymmetricAlgorithm.Create("3DES")) {

    // Hiển thị khóa phiên gốc.
    Console.WriteLine("Session Key at Source = {0}\n\r",
        BitConverter.ToString(symAlg.Key));

    // Chuẩn bị khóa phiên đối xứng dùng cho trao đổi
    // (sử dụng phương thức FormatKeyExchange, phương thức
    // này cần khóa dùng để mật hóa và PUBLIC KEY
    // của người nhận).
    byte[] exchangeData =
        FormatKeyExchange(symAlg.Key, recipientsPublicKey);

    // Hiển thị khóa phiên đã-được-mật-hóa (do phương thức
    // FormatKeyExchange trả về).
    Console.WriteLine("Exchange Data = {0}\n\r",
        BitConverter.ToString(exchangeData));

    // ***** GỬI KHÓA *****
    // Bây giờ, khóa phiên có thể được gửi đi bằng các
    // kênh giao tiếp bình thường.

```

```

// Trích khóa phiên từ dữ liệu trao đổi bằng
// phương thức DeformatKeyExchange.
byte[] sessionKey = DeformatKeyExchange(exchangeData,
    cspParams);

// Hiển thị khóa phiên vừa được trích.
Console.WriteLine("Session Key at Destination = {0}\n\r",
    BitConverter.ToString(sessionKey));

// Nhấn Enter để kết thúc.
Console.ReadLine();
}
}

// Phương thức dùng để mật hóa và định dạng khóa phiên đối xứng.
// Để mật hóa khóa phiên, chúng ta cần truy xuất PUBLIC KEY
// của người nhận (trong cấu trúc RSAParameters).
private static byte[] FormatKeyExchange(byte[] sessionKey,
    RSAParameters rsaParams) {

    // Tạo một giải thuật bất đối xứng RSA.
    using (RSACryptoServiceProvider asymAlg =
        new RSACryptoServiceProvider()) {

        // Nạp PUBLIC KEY của người nhận.
        asymAlg.ImportParameters(rsaParams);

        // Tạo một RSA OAEP formatter để định dạng dữ liệu trao đổi.
        RSAOAEPKeyExchangeFormatter formatter
            = new RSAOAEPKeyExchangeFormatter();

        // Chỉ định giải thuật RSA dùng để mật hóa khóa phiên.
        formatter.SetKey(asymAlg);

        // Mật hóa và định dạng khóa phiên rồi trả về kết quả.
        return formatter.CreateKeyExchange(sessionKey);
    }
}

```

```

    }

    // Phương thức dùng để giải mật hóa dữ liệu trao đổi và trích khóa phiên
    // đối xứng. Để giải mật hóa dữ liệu trao đổi, chúng ta cần truy xuất
    // PRIVATE KEY (từ kho chứa khóa do đối số cspParams chỉ định).
    private static byte[] DeformatKeyExchange(byte[] exchangeData,
        CspParameters cspParams) {

        // Tạo một giải thuật bất đối xứng RSA.
        using (RSACryptoServiceProvider asymAlg =
            new RSACryptoServiceProvider(cspParams)) {

            // Tạo một RSA OAEP deformatter để trích khóa phiên
            // từ dữ liệu trao đổi.
            RSAOAEPKeyExchangeDeformatter deformatter
                = new RSAOAEPKeyExchangeDeformatter();

            // Chỉ định giải thuật RSA dùng để giải mật hóa dữ liệu trao đổi.
            deformatter.SetKey(asymAlg);

            // Giải mật hóa dữ liệu trao đổi và trả về khóa phiên.
            return deformatter.DecryptKeyExchange(exchangeData);
        }
    }
}

```

Chạy KeyExchangeExample sẽ sinh ra kết xuất tương tự như sau:

```
Session Key at Source = EE-5B-16-5B-AC-46-3D-72-CC-73-19-D9-0B-8A-19-E2-A6-02-13-
BE-F8-CE-DF-40
```

```
Exchange Data = 60-FA-3B-63-41-25-F1-AD-08-F9-FC-67-CD-C6-FB-3E-0F-C3-62-
C6-3F-5C-C0-7E-D1-60-2D-19-58-07-EE-BB-7C-53-A5-C2-FB-CA-D7-64-FF-BA-33-77-AC-52-
87-5F-75-E7-57-99-01-90-CD-70-36-1E-53-0C-82-C6-CE-B8-BC-8B-C9-39-6F-29-39-5F-6C-
A6-43-E5-B0-A1-42-46-1C-9B-1C-72-EB-5E-67-06-44-C0-CE-AB-70-B8-39-8E-9F-01-E8-49-
51-36-D6-27-09-94-DA-42-CE-79-C2-72-88-4D-CE-63-B4-A0-AC-07-AF-26-A7-76-DE-21-BE-
A5
```

```
Session Key at Destination = EE-5B-16-5B-AC-46-3D-72-CC-73-19-D9-0B-8A-19-E2-A6-
02-13-BE-F8-CE-DF-40
```


15

KHẢ NĂNG LIÊN TÁC
MÃ LỆNH KHÔNG-ĐƯỢC-QUẢN-LÝ

Microsoft .NET Framework là một nền cực kỳ cao vọng, là sự kết hợp của một ngôn ngữ mới (C#), một bộ thực thi được-quản-lý (CLR), một nền cho các ứng dụng Web (Microsoft ASP.NET), và một thư viện lớp rất lớn để xây dựng tất cả các kiểu ứng dụng. Tuy nhiên, .NET Framework không lặp lại các tính năng có trong mã lệnh không-được-quản-lý. Hiện thời, .NET Framework không bao gồm mọi hàm có trong Win32 API, trong khi nhiều doanh nghiệp đang sử dụng các giải pháp phức tạp được xây dựng với các ngôn ngữ dựa-trên-COM như Microsoft Visual Basic 6 và Microsoft Visual C++ 6. May mắn là Microsoft không có ý để những doanh nghiệp đó bỏ đi nền tảng mã lệnh mà họ đã xây dựng khi chuyển sang nền .NET. Thay vào đó, .NET Framework được trang bị với các tính năng interoperability (khả năng liên tác), cho phép bạn sử dụng lại mã lệnh cũ (legacy code) trong các ứng dụng .NET Framework và truy xuất các assembly .NET như thể chúng là các thành phần COM. Chương này sẽ thảo luận các vấn đề sau:

- Cách gọi các hàm thuộc DLL không-được-quản-lý (mục 15.1 đến 15.5).
- Cách sử dụng thành phần COM trong ứng dụng .NET Framework (mục 15.6 đến 15.8).
- Cách sử dụng điều khiển ActiveX trong ứng dụng .NET Framework (mục 15.9).
- Cách tạo một thành phần .NET sao cho một COM-client có thể sử dụng nó (mục 15.10).

1. Gọi một hàm trong một DLL không-được-quản-lý

? **Bạn cần gọi một hàm C trong một DLL. Đây có thể là một hàm của Win32 API hoặc do bạn viết.**

✕ **Khai báo một phương thức trong mã C# mà bạn sẽ sử dụng để truy xuất hàm không-được-quản-lý. Khai báo phương thức này là static và extern, áp dụng đặc tính System.Runtime.InteropServices.DllImportAttribute để chỉ định file DLL và tên của hàm cần dùng.**

Để sử dụng một hàm C từ một thư viện ngoài, bạn chỉ cần khai báo nó một cách thích hợp. CLR sẽ tự động đảm trách phần việc còn lại, bao gồm việc tải DLL vào bộ nhớ khi hàm được gọi và chuyển các thông số từ kiểu dữ liệu .NET thành kiểu dữ liệu C.

Dịch vụ .NET hỗ trợ việc thực thi xuyên-nền này có tên là *PInvoke* (Platform Invoke), và quá trình này thường là trong suốt đối với người sử dụng. Thỉnh thoảng, bạn sẽ cần thực hiện thêm một số việc, chẳng hạn cần hỗ trợ cấu trúc trong-bộ-nhớ (in-memory structure), callback, hay chuỗi có thể thay đổi (mutable string).

PInvoke thường được sử dụng để truy xuất các hàm Win32 API, đặc biệt là các tính năng không có trong các lớp được-quản-lý thuộc .NET Framework. Các ví dụ được trình bày trong chương này sẽ sử dụng *PInvoke* theo cách này. Có ba thư viện chính trong Win32 API:

- *kernel32.dll*—gồm các hàm đặc-trung-hệ-điều-hành như nạp tiến trình, chuyển ngữ cảnh, nhập/xuất file và bộ nhớ.
- *user32.dll*—gồm các hàm dùng để thao tác cửa sổ, trình đơn, hộp thoại, biểu tượng,...

- *gdi32.dll*— gồm các hàm đồ họa dùng để vẽ trực tiếp lên cửa sổ, trình đơn, bề mặt điều khiển, cũng như để in ấn.

Ví dụ, xét các hàm *Win32 API* dùng để đọc và ghi các file *INI*, chẳng hạn *GetPrivateProfileString* và *WritePrivateProfileString* trong *kernel32.dll*. *.NET Framework* không có lớp nào bọc lấy chức năng này. Tuy nhiên, có thể nhập các hàm này bằng đặc tính *DllImportAttribute* như sau:

```
[DllImport("kernel32.DLL", EntryPoint="WritePrivateProfileString")]
private static extern bool WritePrivateProfileString(string lpAppName,
    string lpKeyName, string lpString, string lpFileName);
```

Các đối số trong phương thức *WritePrivateProfileString* phải tương thích với hàm trong *DLL*, nếu không sẽ có lỗi khi gọi nó. Vì phương thức *WritePrivateProfileString* được khai báo để tham chiếu đến một hàm trong *DLL* nên bạn không được viết mã cho nó. Phần *EntryPoint* trong đặc tính *DllImportAttribute* trong ví dụ này là tùy chọn, vì tên phương thức được khai báo đã trùng với tên của hàm trong thư viện ngoài.

Trong ví dụ sau, lớp *IniFileWrapper* khai báo các phương thức riêng tham chiếu tới các hàm *Win32 API*, sau đó gọi chúng từ các phương thức công khai khác dựa trên file được chỉ định:

```
using System;
using System.Text;
using System.Runtime.InteropServices;
using System.Windows.Forms;

public class IniFileWrapper {

    private string filename;

    public string Filename {
        get {return filename;}
    }

    public IniFileWrapper(string filename) {
        this.filename = filename;
    }

    [DllImport("kernel32.dll", EntryPoint="GetPrivateProfileString")]
    private static extern int GetPrivateProfileString(string lpAppName,
        string lpKeyName, string lpDefault, StringBuilder lpReturnedString,
        int nSize, string lpFileName);

    [DllImport("kernel32.dll", EntryPoint="WritePrivateProfileString")]
```

```

private static extern bool WritePrivateProfileString(
    string lpAppName, string lpKeyName,
    string lpString, string lpFileName);

// Bốn hàm sau không được sử dụng trong ví dụ này,
// nhưng được khai báo cho đầy đủ.
[DllImport("kernel32.dll", EntryPoint="WritePrivateProfileInt")]
private static extern int GetPrivateProfileInt(string lpAppName,
    string lpKeyName, int iDefault, string lpFileName) ;

[DllImport("kernel32.dll", EntryPoint="GetPrivateProfileSection")]
    private static extern int GetPrivateProfileSection(
        string lpAppName, byte[] lpReturnedString,
        int nSize, string lpFileName);

[DllImport("kernel32.dll", EntryPoint="WritePrivateProfileSection")]
    private static extern bool WritePrivateProfileSection(
        string lpAppName, byte[] data, string lpFileName);

[DllImport("kernel32.dll",
    EntryPoint="GetPrivateProfileSectionNames")]
private static extern int GetPrivateProfileSectionNames(
    byte[] lpReturnedString, int nSize, string lpFileName);

public string GetIniValue(string section, string key) {

    StringBuilder buffer = new StringBuilder();
    string sDefault = "";
    if (GetPrivateProfileString(section, key, sDefault,
        buffer, buffer.Capacity, filename) != 0) {

        return buffer.ToString();
    } else {
        return null;
    }
}

```

```

public bool WriteIniValue(string section, string key, string value) {

    return WritePrivateProfileString(section, key, value, filename);

}
}

```



Phương thức `GetPrivateProfileString` có một thông số thuộc kiểu `StringBuilder` (`lpReturnedString`). Đó là vì chuỗi này phải là khả đối—khi lời gọi hàm hoàn tất, nó sẽ chứa thông tin của file *INI*. Bất cứ khi nào cần chuỗi khả đối, bạn phải sử dụng `StringBuilder` thay cho `String`. Thông thường, bạn cần tạo `StringBuilder` với một bộ đệm ký tự có kích thước xác định, rồi truyền kích thước này (`nSize`) cho phương thức. Bạn có thể chỉ định số lượng ký tự trong phương thức khởi dựng của `StringBuilder` (xem mục 2.1 để có thêm thông tin về `StringBuilder`).

Để thử nghiệm lớp `IniFileWrapper`, bạn hãy tạo một file *INI* chứa thông tin sau:

```

[SampleSection]
Key1=Value1
Key2=Value2
Key3=Value3

```

Và thực thi đoạn mã sau để đọc và ghi một giá trị trong file *INI*.

```

public class IniTest {

    private static void Main() {

        IniFileWrapper ini = new IniFileWrapper(
            Application.StartupPath + "\\initest.ini");

        string val = ini.GetIniValue("SampleSection", "Key1");
        Console.WriteLine("Value of Key1 in [SampleSection] is: " + val);

        ini.WriteIniValue("SampleSection", "Key1", "New Value");
        val = ini.GetIniValue("SampleSection", "Key1");
        Console.WriteLine("Value of Key1 in [SampleSection] is now: " +
            val);

        ini.WriteIniValue("SampleSection", "Key1", "Value1");
        Console.ReadLine();

    }

}

```

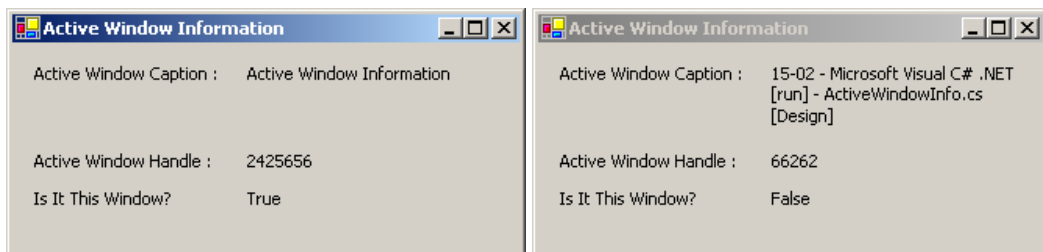
2. *Lấy handle của một điều kiểm, cửa sổ, hoặc file*

? Bạn cần gọi một hàm không-được-quản-lý, và hàm này cần handle của một điều kiểm, cửa sổ, hoặc file.

X Nhiều lớp, bao gồm lớp `FileStream` và tất cả lớp dẫn xuất từ `Control`, trả về handle (thuộc cấu trúc `IntPtr`) thông qua thuộc tính `Handle`. Cũng có lớp trả về thông tin tương tự; ví dụ, lớp `System.Diagnostics.Process` có thêm thuộc tính `Process.MainWindowHandle` ngoài thuộc tính `Handle`.

.NET Framework không che dấu các chi tiết nằm dưới, chẳng hạn handle dùng cho cửa sổ và điều kiểm. Mặc dù không thường sử dụng thông tin này, bạn có thể lấy nó khi cần gọi một hàm không-được-quản-lý và hàm này cần đến nó.

Xét ứng dụng dưới đây, form chính luôn hiển thị trên tất cả các cửa sổ khác bất kể nó có focus hay không (có được chức năng này bằng cách thiết lập thuộc tính `Form.TopMost` là `true`). Form còn có một `Timer` định kỳ gọi các hàm không-được-quản-lý `GetForegroundWindow` và `GetWindowText` để lấy thông tin của cửa sổ hiện đang có focus. Ngoài ra, handle của form chính được lấy thông qua thuộc tính `Form.Handle`, rồi được so sánh với handle của form hiện đang tích cực để kiểm tra form chính đang có focus hay không.



Hình 15.1 Thông tin về cửa sổ đang tích cực

```
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Text;

public class ActiveWindowInfo : System.Windows.Forms.Form {

    // (Bỏ qua phần mã designer.)

    private System.Windows.Forms.Timer tmrRefresh;
    private System.Windows.Forms.Label lblCurrent;
    private System.Windows.Forms.Label lblHandle;
```

```

private System.Windows.Forms.Label lblCaption;

[DllImport("user32.dll")]
private static extern int GetForegroundWindow();

[DllImport("user32.dll")]
private static extern int GetWindowText(int hWnd, StringBuilder text,
    int count);

private void tmrRefresh_Tick(object sender, System.EventArgs e) {

    int chars = 256;
    StringBuilder buff = new StringBuilder(chars);
    int handle = GetForegroundWindow();

    if (GetWindowText(handle, buff, chars) > 0) {

        lblCaption.Text = buff.ToString();
        lblHandle.Text = handle.ToString();
        if (new IntPtr(handle) == this.Handle) {
            lblCurrent.Text = "True";
        } else {
            lblCurrent.Text = "False";
        }
    }
}
}

```



Handle của form được quản lý một cách trong suốt đối với người dùng. Thay đổi thuộc tính nào đó của form có thể khiến cho *C#* tạo một handle mới. Do đó, bạn nên luôn truy xuất handle ngay trước khi sử dụng nó (không nên giữ nó trong một biến để sử dụng trong một thời gian dài).

3. *Gọi một hàm không-được-quản-lý có sử dụng cấu trúc*



Bạn cần gọi một hàm không-được-quản-lý có thông số là một cấu trúc.



Định nghĩa cấu trúc trong mã *C#*. Sử dụng đặc tính `System.Runtime.InteropServices.StructLayoutAttribute` để cấu hình việc cấp bộ nhớ cho cấu trúc. Sử dụng phương thức tĩnh `SizeOf` của lớp

System.Runtime.InteropServices nếu muốn xác định kích thước của cấu trúc theo byte.

Trong mã C# thuần túy, bạn không có khả năng trực tiếp kiểm soát việc cấp bộ nhớ. Thay vào đó, CRL sẽ quyết định khi nào cần đưa dữ liệu vào bộ nhớ để tối ưu hóa hoạt động. Điều này gây rắc rối khi làm việc với các hàm C, vì cấu trúc phải được trữ liên tục trong bộ nhớ. May mắn là .NET đã giải quyết vấn đề này bằng đặc tính `StructLayoutAttribute`, cho phép bạn chỉ định các thành viên của một lớp hay một cấu trúc cho trước sẽ được sắp xếp trong bộ nhớ như thế nào.

Ví dụ, xét hàm `GetVersionEx` trong thư viện `kernel32.dll`. Hàm này nhận một con trỏ chỉ tới cấu trúc `OSVERSIONINFO` và sử dụng nó để trả về thông tin phiên bản của hệ điều hành. Để sử dụng cấu trúc `OSVERSIONINFO` trong mã C#, bạn phải định nghĩa nó với đặc tính `StructLayoutAttribute` như sau:

```
[StructLayout(LayoutKind.Sequential)]
public class OSVersionInfo {

    public int dwOSVersionInfoSize;
    public int dwMajorVersion;
    public int dwMinorVersion;
    public int dwBuildNumber;
    public int dwPlatformId;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst=128)]
    public String szCSDVersion;
}
```

Chú ý rằng, cấu trúc này cũng sử dụng đặc tính `System.Runtime.InteropServices.MarshalAsAttribute` (cần cho các chuỗi có kích thước không đổi). Ở đây, `MarshalAsAttribute` chỉ định chuỗi sẽ được truyền bằng trị và sẽ chứa một bộ đệm gồm 128 ký tự được chỉ định trong cấu trúc `OSVersionInfo`. Trong ví dụ này, `LayoutKind.Sequential` được sử dụng, nghĩa là các kiểu dữ liệu trong cấu trúc được bố trí theo thứ tự mà chúng được liệt kê trong cấu trúc hoặc lớp.

Ngoài `LayoutKind.Sequential`, bạn có thể sử dụng `LayoutKind.Explicit`. Trong trường hợp này, bạn phải sử dụng `FieldOffsetAttribute` để định nghĩa độ dời của các trường. Cách này hữu ích khi bạn muốn lưu trữ các trường một cách linh động hơn, hoặc bạn muốn bỏ qua (không sử dụng) trường nào đó. Ví dụ sau định nghĩa lớp `OSVersionInfo` với `LayoutKind.Explicit`.

```
[StructLayout(LayoutKind.Explicit)]
public class OSVersionInfo {

    [FieldOffset(0)] public int dwOSVersionInfoSize;
```



```

[FieldOffset(4)] public int dwMajorVersion;
[FieldOffset(8)] public int dwMinorVersion;
[FieldOffset(12)] public int dwBuildNumber;
[FieldOffset(16)] public int dwPlatformId;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst=128)]
[FieldOffset(20)] public String szCSDVersion;
}

```

Sau khi đã định nghĩa cấu trúc được sử dụng bởi hàm `GetVersionEx`, bạn có thể khai báo hàm này và sử dụng nó. Ứng dụng dưới đây sẽ trình bày toàn bộ mã lệnh cần thiết. Chú ý, `InAttribute` và `OutAttribute` được áp dụng cho `OSVersionInfo` để biết rằng marshalling sẽ được thực hiện trên cấu trúc này khi nó được truyền cho hàm và khi nó được trả về từ hàm. Ngoài ra, phương thức `Marshal.SizeOf` được sử dụng để tính kích thước của cấu trúc trong bộ nhớ.

```

using System;
using System.Runtime.InteropServices;

public class CallWithStructure {

    // (Bỏ qua lớp OSVersionInfo.)

    [DllImport("kernel32.dll")]
    public static extern bool GetVersionEx([In, Out] OSVersionInfo osv);

    private static void Main() {

        OSVersionInfo osv = new OSVersionInfo();
        osv.dwOSVersionInfoSize = Marshal.SizeOf(osv);

        GetVersionEx(osv);

        Console.WriteLine("Class size: " + osv.dwOSVersionInfoSize);
        Console.WriteLine("Major Version: " + osv.dwMajorVersion);
        Console.WriteLine("Minor Version: " + osv.dwMinorVersion);
        Console.WriteLine("Build Number: " + osv.dwBuildNumber);
        Console.WriteLine("Platform Id: " + osv.dwPlatformId);
        Console.WriteLine("CSD Version: " + osv.szCSDVersion);
        Console.WriteLine("Platform: " + Environment.OSVersion.Platform);
        Console.WriteLine("Version: " + Environment.OSVersion.Version);
        Console.ReadLine();
    }
}

```

```
}
```

Nếu chạy ứng dụng này trên hệ thống *Windows XP*, bạn sẽ thấy thông tin như sau:

```
Class size: 148
Major Version: 5
Minor Version: 1
Build Number: 2600
Platform Id: 2
CSD Version: Service Pack 1
Platform: Win32NT
Version: 5.1.2600.0
```

4. *Gọi một hàm không-được-quản-lý có sử dụng callback*

? Bạn cần gọi một hàm không-được-quản-lý và cho phép nó gọi một hàm khác.

✂ Tạo một ủy nhiệm cho callback. Sử dụng ủy nhiệm này khi định nghĩa và sử dụng hàm không-được-quản-lý.

Nhiều hàm của *Win32 API* sử dụng callback. Ví dụ, nếu muốn lấy tên của tất cả các cửa sổ đang mở, bạn có thể sử dụng hàm `EnumWindows` trong thư viện *user32.dll*. Khi gọi `EnumWindows`, bạn cần truyền cho nó một con trỏ chỉ đến một hàm khác trong mã lệnh của bạn. Hệ điều hành *Windows* sau đó sẽ gọi hàm này mỗi khi tìm thấy một cửa sổ đang mở, và truyền handle của cửa sổ cho nó.

.NET Framework cho phép bạn quản lý việc sử dụng callback mà không cần các con trỏ và các khối mã không an toàn. Thay vào đó, bạn có thể định nghĩa và sử dụng một ủy nhiệm chỉ đến hàm callback. Khi bạn truyền ủy nhiệm cho hàm `EnumWindows`, *CLR* sẽ tự động marshal ủy nhiệm thành con trỏ hàm không-được-quản-lý như mong muốn.

Ví dụ dưới đây sử dụng `EnumWindows` cùng với một callback để hiển thị tên của tất cả các cửa sổ đang mở.

```
using System;
using System.Text;
using System.Runtime.InteropServices;

public class GetWindows {

    // Chữ ký cho hàm callback.
    public delegate bool CallBack(int hwnd, int lParam);

    // Hàm không-được-quản-lý sẽ kích hoạt callback
```

```
// khi duyệt qua các cửa sổ đang mở.
[DllImport("user32.dll")]
public static extern int EnumWindows(CallBack callback, int param);

[DllImport("user32.dll")]
public static extern int GetWindowText(int hWnd,
    StringBuilder lpString, int nMaxCount);

private static void Main() {

    CallBack callBack = new CallBack(DisplayWindowInfo);

    // Yêu cầu hệ điều hành duyệt qua các cửa sổ đang mở,
    // kích hoạt callback với handle của mỗi cửa sổ.
    EnumWindows(callBack, 0);

    Console.ReadLine();
}

// Hàm sẽ nhận callback. Thông số thứ hai
// không được sử dụng nhưng phải được khai báo để
// tương thích với chữ ký của callback.
public static bool DisplayWindowInfo(int hWnd, int lParam) {

    int chars = 100;
    StringBuilder buf = new StringBuilder(chars);
    if (GetWindowText(hWnd, buf, chars) != 0) {
        Console.WriteLine(buf);
    }
    return true;
}
}
```

5.

Lấy thông tin lỗi không-được-quản-lý

?

Bạn cần truy xuất thông tin lỗi (mã lỗi hoặc thông điệp mô tả lỗi) giải thích tại sao một lời gọi *Win32 API* thất bại.

✕

Trong phần khai báo của hàm không-được-quản-lý, thiết lập trường `SetLastError` của đặc tính `DllImportAttribute` là `true`. Nếu có lỗi khi thực thi, gọi

phương thức tĩnh `Marshal.GetLastWin32Error` để truy xuất mã lỗi. Để lấy thông điệp mô tả một mã lỗi cụ thể, sử dụng hàm không-được-quản-lý `FormatMessage`.

Bạn không thể trực tiếp lấy thông tin lỗi bằng hàm không-được-quản-lý `GetLastError`. Vấn đề là, mã lỗi do `GetLastError` trả về có thể không phản ánh lỗi do hàm không-được-quản-lý gây ra. Thay vào đó, nó có thể được thiết lập bởi các lớp *.NET Framework* khác hoặc *CLR*. Bạn có thể lấy thông tin lỗi một cách an toàn bằng phương thức tĩnh `Marshal.GetLastWin32Error`. Phương thức này cần được gọi ngay sau lời gọi hàm không-được-quản-lý, và nó sẽ trả về thông tin lỗi chỉ một lần (các lần gọi `GetLastWin32Error` sau đó sẽ trả về mã lỗi *127*). Ngoài ra, bạn phải thiết lập trường `SetLastError` của đặc tính `DllImportAttribute` là `true`, cho biết những lỗi do hàm này sinh ra sẽ được ghi nhận.

```
[DllImport("user32.dll", SetLastError=true)]
```

Sau đó, bạn có thể sử dụng hàm `FormatMessage` trong thư viện *kernel32.dll* để lấy thông điệp mô tả lỗi từ mã lỗi *Win32*.

Ví dụ, ứng dụng dưới đây muốn hiển thị một `MessageBox`, nhưng lại sử dụng handle không đúng. Mã lỗi được lấy bằng `Marshal.GetLastWin32Error`, và thông điệp mô tả lỗi được lấy bằng `FormatMessage`.

```
using System;

using System.Runtime.InteropServices;

public class TestError {

    [DllImport("kernel32.dll")]
    private unsafe static extern int FormatMessage(int dwFlags,
        int lpSource, int dwMessageId, int dwLanguageId,
        ref String lpBuffer, int nSize, int Arguments);

    [DllImport("user32.dll", SetLastError=true)]
    public static extern int MessageBox(int hWnd, string pText,
        string pCaption, int uType);

    private static void Main() {

        int badWindowHandle = 453;
        MessageBox(badWindowHandle, "Message", "Caption", 0);

        int errorCode = Marshal.GetLastWin32Error();
        Console.WriteLine(errorCode);
        Console.WriteLine(GetErrorMessage(errorCode));
    }
}
```

```

        Console.ReadLine();
    }

    // GetErrorMessage trả về thông điệp lỗi
    // tương ứng với mã lỗi errorCode.
    public static string GetErrorMessage(int errorCode) {

        int FORMAT_MESSAGE_ALLOCATE_BUFFER = 0x00000100;
        int FORMAT_MESSAGE_IGNORE_INSERTS = 0x00000200;
        int FORMAT_MESSAGE_FROM_SYSTEM = 0x00001000;

        int messageSize = 255;
        string lpMsgBuf = "";
        int dwFlags = FORMAT_MESSAGE_ALLOCATE_BUFFER |
            FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS;

        int retVal = FormatMessage(dwFlags, 0, errorCode, 0,
            ref lpMsgBuf, messageSize, 0);

        if (0 == retVal) {
            return null;
        } else {
            return lpMsgBuf;
        }
    }
}

```

Sau đây là kết xuất của ứng dụng:

```

1400
Invalid window handle.

```

6.

Sử dụng thành phần COM trong .NET-client

?

Bạn cần sử dụng thành phần COM trong một .NET-client.

✕

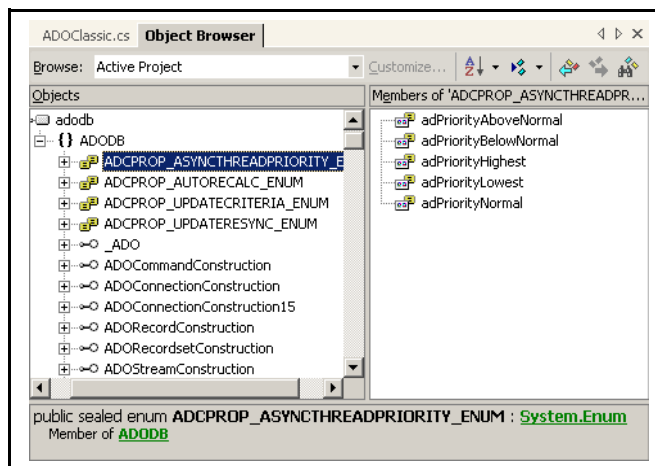
Sử dụng một *Primary Interop Assembly*, nếu có. Nếu không, tạo một *Runtime Callable Wrapper* bằng *Type Library Importer (Tlbimp.exe)*, hoặc tính năng *Add Reference* trong *Visual Studio .NET*.

.NET Framework hỗ trợ khả năng liên tác với COM. Để cho phép .NET-client tương tác với một thành phần COM, .NET sử dụng một *Runtime Callable Wrapper (RCW)*—một lớp proxy đặc biệt đóng vai trò trung gian giữa mã .NET và thành phần COM. RCW sẽ xử lý tất cả chi tiết, bao gồm việc marshal các kiểu dữ liệu, sử dụng các giao diện COM truyền thống, và xử lý các sự kiện COM.

Bạn có ba tùy chọn khi sử dụng một RCW:

- Lấy RCW từ tác giả của thành phần COM. Trong trường hợp này, RCW được gọi là một *Primary Interop Assembly (PIA)*.
- Tạo RCW bằng tiện ích dòng lệnh *Tlbimp.exe* hoặc bằng *Visual Studio .NET*.
- Tự tạo RCW thông qua các kiểu trong không gian tên `System.Runtime.InteropServices` (cách này cực kỳ dài dòng và phức tạp).

Nếu muốn sử dụng *Visual Studio .NET* để tạo RCW, bạn chỉ cần chọn *Project | Add Reference* và sau đó chọn thành phần thích hợp trong thẻ *COM*. *Interop Assembly* sẽ được tạo ra và được thêm vào các tham chiếu của dự án. Sau đó, bạn có thể sử dụng *Object Browser* để khảo sát các không gian tên và các lớp có trong thành phần này.



Hình 15.2 Sử dụng Object Browser để xem các không gian tên và các lớp có trong thành phần COM

Nếu không sử dụng *Visual Studio .NET*, bạn có thể tạo một *Wrapper Assembly* bằng công cụ *Tlbimp.exe*, chỉ cần truyền cho nó tên file chứa thành phần COM. Ví dụ, dòng lệnh sau đây sẽ tạo một RCW với tên file và không gian tên mặc định, giả sử file *MyCOMComponent.dll* đang nằm trong thư mục hiện hành.

```
tlbimp MyCOMComponent.dll
```

Giả sử *MyCOMComponent* chứa một thư viện kiểu có tên là *MyClasses*, file RCW sẽ có tên là *MyClasses.dll* và sẽ trưng các lớp của nó thông qua một không gian tên có tên là *MyClasses*. Bạn có thể cấu hình các tùy chọn này bằng các đối số dòng lệnh (được mô tả chi tiết trong tài

liệu *MSDN*). Ví dụ, bạn có thể sử dụng `/out:[Filename]` để chỉ định tên file assembly, sử dụng `/namespace:[Namespace]` để chỉ định không gian tên. Bạn cũng có thể sử dụng `/keyfile[keyfilename]` để tạo tên mạnh cho assembly, và đặt nó vào *Global Assembly Cache (GAC)*. Sử dụng `/primary` để tạo một *PIA*.

Nếu có thể, bạn nên sử dụng *PIA* thay vì tạo *RCW*. *PIA* hoạt động đáng tin cậy hơn vì được tạo ra bởi nhà phát hành thành phần, và cũng có thể chứa một số tính năng nâng cao. Nếu một *PIA* được đăng ký với hệ thống cho một thành phần *COM* thì *Visual Studio .NET* sẽ tự động sử dụng *PIA* đó khi bạn thêm một tham chiếu đến thành phần *COM*. Ví dụ, *adodb.dll* (có trong *.NET Framework*) cho phép bạn sử dụng các đối tượng *ADO* truyền thống. Nếu bạn thêm một tham chiếu đến thành phần *Microsoft ActiveX Data Objects, Interop Assembly* này sẽ tự động được sử dụng, mà không sinh ra một *RCW* mới. Tương tự, *Microsoft Office XP* cung cấp một *PIA* để nâng cao sự hỗ trợ *.NET* cho *Office Automation* (bạn có thể tải *PIA* này tại địa chỉ [<http://msdn.microsoft.com/downloads/list/office.asp>]).

Ví dụ sau minh họa cách sử dụng *COM Interop* để truy xuất các đối tượng *ADO* truyền thống trong ứng dụng *.NET Framework*.

```
using System;

public class ADOLClassic {

    private static void Main() {

        ADODB.Connection con = new ADODB.Connection();
        string connectionString = "Provider=SQLOLEDB.1;" +
            "Data Source=localhost;" +
            "Initial Catalog=Northwind;Integrated Security=SSPI";
        con.Open(connectionString, null, null, 0);

        object recordsAffected;
        ADODB.Recordset rs = con.Execute("SELECT * From Customers",
            out recordsAffected, 0);

        while (rs.EOF != true) {

            Console.WriteLine(rs.Fields["CustomerID"].Value);
            rs.MoveNext();
        }

        Console.ReadLine();
    }
}
```

7.

Giải phóng nhanh thành phần COM

- ?** Bạn cần bảo đảm một thành phần *COM* được xóa khỏi bộ nhớ ngay tức thì, không phải chờ bộ thu gom rác (*Garbage Collector*) làm việc. Hoặc bạn muốn bảo đảm các đối tượng *COM* được giải phóng theo một thứ tự xác định.
- ✕** Sử dụng phương thức tĩnh `Marshal.ReleaseComObject` và truyền *RCW* thích hợp để giải phóng tham chiếu đến đối tượng *COM* nằm dưới.

COM sẽ đếm các tham chiếu đến đối tượng để xác định khi nào đối tượng sẽ được giải phóng. Khi bạn sử dụng một *RCW*, tham chiếu sẽ được giữ cả khi biến đối tượng vượt khỏi tầm vực. Tham chiếu này chỉ được giải phóng khi bộ thu gom rác giải phóng đối tượng *RCW*. Do đó, bạn không thể kiểm soát việc các đối tượng *COM* sẽ được giải phóng khi nào hoặc theo thứ tự nào.

Để vượt qua hạn chế này, bạn có thể sử dụng phương thức `Marshal.ReleaseComObject`. Trong ví dụ ở mục 15.6, bạn hãy thêm hai dòng sau vào cuối mã lệnh để giải phóng đối tượng `Recordset` và `Connection` nằm dưới.

```
Marshal.ReleaseComObject(rs);
```

```
Marshal.ReleaseComObject(con);
```

- ✎** Về mặt kỹ thuật, phương thức `ReleaseComObject` không thực sự giải phóng đối tượng *COM*, nó chỉ giảm số lượng tham chiếu đến đối tượng. Nếu số tham chiếu là 0, đối tượng *COM* sẽ được giải phóng. Tuy nhiên, nếu cùng thể hiện của một đối tượng *COM* được sử dụng tại nhiều mẫu mã lệnh, nó phải được giải phóng ở các nơi đó trước khi được giải phóng khỏi bộ nhớ.

8.

Sử dụng thông số tùy chọn

- ?** Bạn cần gọi một phương thức trong thành phần *COM* mà không phải truyền tất cả các thông số cần thiết.
- ✕** Sử dụng trường `Type.Missing`.

Hầu hết các phương thức trong *.NET Framework* đều được nạp chồng nhiều lần để bạn có thể gọi phiên bản yêu cầu chỉ những thông số do bạn cung cấp. Mặt khác, *COM* không hỗ trợ việc nạp chồng phương thức. Thay vào đó, các thành phần *COM* thường sử dụng các phương thức với một danh sách dài các thông số tùy chọn. Không may là, *C#* không hỗ trợ thông số tùy chọn, nghĩa là người phát triển phải cung cấp thêm các giá trị không cần thiết khi truy xuất một thành phần *COM*. Và vì các thông số *COM* thường được truyền bằng tham chiếu nên mã lệnh của bạn không thể truyền một tham chiếu `null`, mà phải khai báo một biến đối tượng và rồi truyền biến đó.

Bạn có thể giảm nhẹ vấn đề đến một chừng mực nào đó bằng cách cung cấp trường `Type.Missing` bất cứ khi nào muốn bỏ qua một thông số tùy chọn. Nếu cần truyền một thông

số bằng tham chiếu, bạn chỉ cần khai báo một biến đối tượng, thiết lập nó là `Type.Missing`, và sử dụng nó trong mọi trường hợp:

```
private static object n = Type.Missing;
```

Ví dụ dưới đây sử dụng đối tượng *Word* để tạo và hiển thị một tài liệu. Trong đó, có nhiều phương thức yêu cầu các thông số tùy chọn (được truyền bằng tham chiếu). Việc sử dụng trường `Type.Missing` đơn giản hóa mã lệnh rất nhiều.

```
using System;
```

```
public class OptionalParameters {
```

```
    private static object n = Type.Missing;
```

```
    private static void Main() {
```

```
        // Chạy Word phía nền.
```

```
        Word.ApplicationClass app = new Word.ApplicationClass();
```

```
        app.DisplayAlerts = Word.WdAlertLevel.wdAlertsNone;
```

```
        // Tạo một tài liệu mới (không khả kiến đối với người dùng).
```

```
        Word.Document doc = app.Documents.Add(ref n, ref n, ref n,  
        ref n);
```

```
        Console.WriteLine();
```

```
        Console.WriteLine("Creating new document.");
```

```
        Console.WriteLine();
```

```
        // Thêm một tiêu đề và hai hàng text.
```

```
        Word.Range range = doc.Paragraphs.Add(ref n).Range;
```

```
        range.InsertBefore("Test Document");
```

```
        string style = "Heading 1";
```

```
        object objStyle = style;
```

```
        range.set_Style(ref objStyle);
```

```
        range = doc.Paragraphs.Add(ref n).Range;
```

```
        range.InsertBefore("Line one.\nLine two.");
```

```
        range.Font.Bold = 1;
```

```
        // Hiển thị Print Preview, làm cho Word trở nên khả kiến.
```

```
        doc.PrintPreview();
```

```
        app.Visible = true;
```

```

        Console.ReadLine();
    }
}

```

9. *Sử dụng điều khiển ActiveX trong .NET-client*

- ?** **Bạn cần đặt một điều khiển ActiveX trên một cửa sổ ứng dụng .NET Framework.**
- X** **Sử dụng một RCW (cũng giống như với một thành phần COM bình thường). Để làm việc với điều khiển ActiveX khi thiết kế, thêm nó vào hộp công cụ của Visual Studio .NET.**

.NET Framework hỗ trợ như nhau đối với tất cả các thành phần COM, bao gồm điều khiển ActiveX. Điều khác nhau cơ bản là lớp RCW (cho điều khiển ActiveX) dẫn xuất từ kiểu .NET đặc biệt System.Windows.Forms.AxHost. Về mặt kỹ thuật, bạn thêm AxHost vào form, và nó sẽ giao tiếp với điều khiển ActiveX phía hậu trường. Vì dẫn xuất từ System.Windows.Forms.Control, nên AxHost cũng có các thuộc tính, phương thức, và sự kiện chuẩn như Location, Size, Anchor,.. Nếu RCW được sinh tự động, các lớp AxHost luôn bắt đầu bằng Ax.

Bạn có thể tạo một RCW cho một điều khiển ActiveX cũng giống như cho bất cứ thành phần COM nào khác bằng công cụ Tlbimp.exe hoặc tính năng Add Reference trong Visual Studio .NET, sau đó lập trình để tạo điều khiển. Tuy nhiên, một cách tiếp cận dễ hơn trong Visual Studio .NET là thêm điều khiển ActiveX vào hộp công cụ (xem mục 11.4 để biết thêm chi tiết).

Chẳng có gì xảy ra khi bạn thêm một điều khiển ActiveX vào hộp công cụ. Tuy nhiên, bạn có thể sử dụng biểu tượng trong hộp công cụ để thêm một thể hiện của điều khiển vào form. Lần đầu bạn làm việc này, Visual Studio .NET sẽ tạo một Interop Assembly và thêm nó vào dự án của bạn. Ví dụ, nếu bạn thêm điều khiển Microsoft Masked Edit (không có điều khiển .NET tương đương), Visual Studio .NET sẽ tạo một RCW Assembly có tên là AxInterop.MSMask.dll. Dưới đây là đoạn mã trong vùng designer dùng để tạo một thể hiện của điều khiển này và thêm nó vào form:

```

this.axMaskedTextBox1 = new AxMSMask.AxMaskedTextBox();
((System.ComponentModel.ISupportInitialize)(this.axMaskedTextBox1)).
    BeginInit();

//
// axMaskedTextBox1
//

this.axMaskedTextBox1.Location = new System.Drawing.Point(16, 12);
this.axMaskedTextBox1.Name = "axMaskedTextBox1";
this.axMaskedTextBox1.OcxState = ((System.Windows.Forms.AxHost.State)

```

```
(resources.GetObject("axMaskedTextBox1.OcxState")));

this.axMaskedTextBox1.Size = new System.Drawing.Size(112, 20);
this.axMaskedTextBox1.TabIndex = 0;

this.Controls.Add(this.axMaskedTextBox1);
```

Chú ý rằng, các thuộc tính tùy biến của điều khiển *ActiveX* không được áp dụng trực tiếp thông qua các lệnh thiết lập thuộc tính. Thay vào đó, chúng sẽ được thiết lập theo nhóm khi thuộc tính *OcxState* đã được thiết lập. Tuy nhiên, mã lệnh của bạn có thể sử dụng các thuộc tính này một cách trực tiếp.

10. Tạo thành phần .NET dùng cho COM-client

? Bạn cần tạo một thành phần .NET sao cho một COM-client có thể gọi nó.

✂ Tạo một assembly theo các chỉ dẫn trong mục này. Tạo một thư viện kiểu cho assembly này bằng tiện ích dòng lệnh *Type Library Exporter (Tlbexp.exe)*.

.NET Framework hỗ trợ việc COM-client sử dụng thành phần .NET. Khi COM-client tạo một đối tượng .NET, CLR sẽ tạo một đối tượng được-quản-lý và một *COM Callable Wrapper (CCW)* bọc lấy đối tượng này. COM-client sẽ tương tác với đối tượng thông qua CCW. CLR chỉ tạo một CCW cho một đối tượng được-quản-lý, bất chấp có bao nhiêu COM-client đang sử dụng nó.

Các kiểu cần được truy xuất bởi COM-client phải thỏa mãn các yêu cầu sau:

- Các kiểu được-quản-lý (lớp, giao diện, cấu trúc, hoặc kiểu liệt kê) phải được khai báo là *public*.
- Nếu COM-client cần tạo đối tượng, nó phải có một phương thức khởi dựng mặc định *public*. COM không hỗ trợ các phương thức khởi dựng có chứa thông số.
- Các thành viên của kiểu cần được truy xuất phải là các thành viên *public*. COM-client không truy xuất được các thành viên *private*, *protected*, *internal*, và *static*.

Ngoài ra, bạn nên tuân theo các kinh nghiệm sau:

- Không nên tạo các quan hệ thừa kế giữa các lớp, vì các quan hệ này sẽ không khả kiến đối với COM-client (mặc dù .NET giả lập quan hệ này bằng cách khai báo một giao diện lớp cơ sở dùng chung).
- Các lớp mà bạn trưng ra nên hiện thực một giao diện. Với mục đích kiểm soát phiên bản, bạn có thể sử dụng đặc tính *System.Runtime.InteropServices.GuidAttribute* để chỉ định *GUID* sẽ được gán cho giao diện.
- Nên tạo tên mạnh cho assembly để nó có thể được cài đặt vào *GAC* và được dùng chung cho nhiều client.

Để tạo một đối tượng .NET, COM-client cần một thư viện kiểu (file *.tlb*). File thư viện kiểu có thể được tạo từ một assembly bằng tiện ích dòng lệnh *Tlbexp.exe*. Ví dụ:

```
tlbexp ManagedLibrary.dll
```

Một khi đã tạo ra thư viện kiểu, bạn có thể tham chiếu nó từ một công cụ phát triển không-được-quản-lý. Với *Visual Basic 6*, bạn tham chiếu file *.tlb* từ hộp thoại *Project | Reference*. Trong *Visual C++ 6*, bạn có thể sử dụng lệnh `#import` để nhập các định nghĩa kiểu từ thư viện kiểu.

16

CÁC GIAO DIỆN VÀ MẪU THÔNG DỤNG

Chương này trình bày cách hiện thực các mẫu (*pattern*) sẽ được sử dụng thường xuyên trong quá trình phát triển các ứng dụng *Microsoft .NET Framework*. Một số mẫu được chuẩn hóa bằng các giao diện được định nghĩa trong thư viện lớp *.NET Framework*. Một số khác thì ít cứng nhắc hơn, nhưng vẫn yêu cầu bạn thực hiện các cách tiếp cận cụ thể để thiết kế và hiện thực các kiểu của bạn. Các mục trong chương này mô tả cách:

- Tạo các kiểu khả-tuần-tự-hóa để bạn có thể dễ dàng lưu trữ vào đĩa, gửi qua mạng, hoặc truyền bằng trị qua các biên miền ứng dụng (mục 16.1).
- Cung cấp một cơ chế dùng để tạo bản sao đầy đủ và chính xác của đối tượng (mục 16.2).
- Hiện thực các kiểu sao cho dễ dàng so sánh và sắp xếp (mục 16.3).
- Hỗ trợ việc liệt kê các phần tử trong các tập hợp tùy biến (mục 16.4).
- Bảo đảm rằng một kiểu có sử dụng các tài nguyên không-được-quản-lý sẽ giải phóng các tài nguyên đó khi không còn cần đến chúng nữa (mục 16.5).
- Hiện thị dạng chuỗi của các đối tượng biến đổi dựa trên *format specifier* (mục 16.6).
- Hiện thực các kiểu đối số sự kiện và ngoại lệ tùy biến (bạn sẽ thường xuyên sử dụng chúng trong quá trình phát triển ứng dụng) (mục 16.7 và 16.8).
- Hiện thực các mẫu thiết kế thông dụng *Singleton* và *Observer* bằng các tính năng có sẵn của *C#* và thư viện lớp *.NET Framework* (mục 16.9 và 16.10).

1. *Hiện thực kiểu khả-tuần-tự-hóa (serializable type)*

?

Bạn cần hiện thực một kiểu tùy biến khả-tuần-tự-hóa, cho phép bạn:

- Lưu trữ các thể hiện của kiểu vào kho lưu trữ (file hay cơ sở dữ liệu).
- Chuyển các thể hiện của kiểu qua mạng.
- Truyền các thể hiện của kiểu “bằng trị” qua các biên miền ứng dụng.

✕

Đối với việc tuần tự hóa các kiểu đơn giản, hãy áp dụng đặc tính `System.SerializableAttribute` vào khai báo kiểu. Đối với các kiểu phức tạp hơn, hoặc để kiểm soát nội dung và cấu trúc của dữ liệu được-tuần-tự-hóa, hãy hiện thực giao diện `System.Runtime.Serialization.ISerializable`.

Mục 2.12 đã trình bày cách tuần tự hóa và giải tuần tự hóa một đối tượng bằng các lớp *formatter* (được cấp cùng với thư viện lớp *.NET Framework*). Tuy nhiên, theo mặc định thì các kiểu không là khả-tuần-tự-hóa. Để hiện thực một kiểu tùy biến là khả-tuần-tự-hóa, bạn phải áp dụng đặc tính `SerializableAttribute` vào khai báo kiểu. Khi tất cả các trường dữ liệu trong kiểu đều là khả-tuần-tự-hóa, việc áp dụng `SerializableAttribute` là tất cả những gì cần làm để khiến cho kiểu tùy biến của bạn là khả-tuần-tự-hóa. Nếu bạn hiện thực một lớp tùy biến dẫn xuất từ một lớp cơ sở, lớp cơ sở cũng phải là khả-tuần-tự-hóa.

Mỗi lớp *formatter* chứa logic cần thiết để tuần tự hóa các kiểu được gắn với đặc tính `SerializableAttribute` và sẽ tuần tự hóa tất cả các trường `public`, `protected`, và `private`.

Đoạn mã dưới đây trình bày các khai báo kiểu và khai báo trường của một lớp khả-tuần-tự-hóa có tên là `Employee`.

```
using System;

[Serializable]
public class Employee {

    private string name;
    private int age;
    private string address;
    §
}
```



Các lớp dẫn xuất từ một kiểu khả-tuần-tự-hóa không thừa kế đặc tính `SerializableAttribute`. Để khiến cho các kiểu dẫn xuất là khả-tuần-tự-hóa, bạn phải khai báo chúng là khả-tuần-tự-hóa bằng cách áp dụng đặc tính `SerializableAttribute`.

Bạn có thể ngăn việc tuần tự hóa một trường nào đó bằng cách áp dụng đặc tính `System.NonSerializedAttribute` cho trường này. Bạn nên ngăn việc tuần tự hóa đối với các trường sau:

- Chứa các kiểu dữ liệu không-khả-tuần-tự-hóa.
- Chứa các giá trị có thể không hợp lệ khi đối tượng được giải tuần tự hóa, ví dụ: kết nối cơ sở dữ liệu, địa chỉ bộ nhớ, *ID* của tiểu trình, và handle của tài nguyên không-được-quản-lý.
- Chứa các thông tin nhạy cảm hay riêng tư, ví dụ: mật khẩu, khóa mật hóa, và các chi tiết riêng về người hay tổ chức.
- Chứa các dữ liệu dễ dàng tái tạo hay thu lấy được từ các nguồn khác—đặc biệt khi dữ liệu lớn.

Nếu ngăn việc tuần tự hóa một số trường, bạn phải hiện thực kiểu sao cho bù lại việc những dữ liệu nào đó sẽ không hiện diện khi một đối tượng được giải tuần tự hóa. Đáng tiếc, bạn không thể tạo hay thu lấy các trường dữ liệu bị mất trong một phương thức khởi dựng vì *formatter* không gọi phương thức khởi dựng trong quá trình giải tuần tự hóa đối tượng. Giải pháp thông thường nhất là hiện thực mẫu “*Lazy Initialization*”, trong đó kiểu của bạn sẽ tạo hay thu lấy dữ liệu ngay lần đầu tiên cần đến.

Đoạn mã dưới đây trình bày một phiên bản đã được chỉnh sửa của lớp `Employee` với đặc tính `NonSerializedAttribute` được áp dụng cho trường `address`, nghĩa là *formatter* sẽ không tuần tự hóa giá trị của trường này. Lớp `Employee` hiện thực các thuộc tính công khai dùng để truy xuất các thành viên dữ liệu riêng, là nơi thuận tiện để hiện thực “*Lazy Initialization*” cho trường `address`.


```
using System;

[Serializable]
public class Employee {

    private string name;
    private int age;

    [NonSerialized]
    private string address;

    // Phương thức khởi dựng đơn giản.
    public Employee(string name, int age, string address) {

        this.name = name;
        this.age = age;
        this.address = address;
    }

    // Thuộc tính công khai dùng để truy xuất tên của nhân viên.
    public string Name {
        get { return name; }
        set { name = value; }
    }

    // Thuộc tính công khai dùng để truy xuất tuổi của nhân viên.
    public int Age {
        get { return age; }
        set { age = value; }
    }

    // Thuộc tính công khai dùng để truy xuất địa chỉ của nhân viên.
    // Sử dụng "Lazy Initialization" để thiết lập địa chỉ vì
    // đối tượng được-giải-tuần-tự-hóa sẽ không có giá trị địa chỉ.
    public string Address {
        get {
            if (address == null) {
```

```

        // Nạp địa chỉ từ kho lưu trữ.
    }

    return address;
}

set {
    address = value;
}
}
}

```

Đối với phần lớn các kiểu tùy biến, việc sử dụng đặc tính `SerializableAttribute` và `NonSerializedAttribute` sẽ đáp ứng đủ nhu cầu tuần tự hóa của bạn. Nếu cần kiểm soát quá trình tuần tự hóa, bạn cần hiện thực giao diện `ISerializable`. Các lớp *formatter* sử dụng logic khác nhau khi tuần tự hóa và giải tuần tự hóa thể hiện của các kiểu có hiện thực `ISerializable`. Để hiện thực đúng `ISerializable`, bạn phải:

- Khai báo rằng kiểu của bạn có hiện thực giao diện `ISerializable`.
- Áp dụng đặc tính `SerializableAttribute` vào khai báo kiểu như vừa được mô tả; không sử dụng `NonSerializedAttribute` vì nó sẽ không có tác dụng.
- Hiện thực phương thức `ISerializable.GetObjectData` (được sử dụng trong quá trình tuần tự hóa), phương thức này nhận các kiểu đối số sau:
 - `System.Runtime.Serialization.SerializationInfo`
 - `System.Runtime.Serialization.StreamingContext`
- Hiện thực một phương thức khởi dựng không công khai (được sử dụng trong quá trình giải tuần tự hóa), phương thức này nhận cùng đối số như phương thức `GetObjectData`. Nhớ rằng, nếu bạn có ý định dẫn xuất một số lớp từ lớp khả-tuần-tự-hóa thì phương thức khởi dựng này phải là `protected`.
- Nếu bạn tạo một lớp khả-tuần-tự-hóa từ một lớp cơ sở cũng có hiện thực `ISerializable`, thì phương thức `GetObjectData` và phương thức khởi dựng (dùng để giải tuần tự hóa) của bạn phải gọi các phương thức tương đương trong lớp cha.

Trong quá trình tuần tự hóa, *formatter* sẽ gọi phương thức `GetObjectData` và truyền cho nó các tham chiếu `SerializationInfo` và `StreamingContext`.

- Bạn phải đổ dữ liệu cần tuần tự hóa vào đối tượng `SerializationInfo`. Lớp `SerializationInfo` cung cấp phương thức `AddValue` dùng để thêm dữ liệu. Với mỗi lần gọi `AddValue`, bạn phải chỉ định tên dữ liệu (tên này sẽ được sử dụng trong quá trình giải tuần tự hóa để thu lấy dữ liệu). Phương thức `AddValue` có đến 16 phiên bản nạp chồng, cho phép bạn thêm nhiều kiểu dữ liệu khác nhau vào đối tượng `SerializationInfo`.

- Đối tượng `StreamingContext` cung cấp các thông tin về chủ định và đích của dữ liệu được-tuần-tự-hóa, cho phép bạn chọn tuần tự hóa dữ liệu nào. Ví dụ, bạn có thể cần tuần tự hóa dữ liệu riêng nếu nó được dành cho một miền ứng dụng khác trong cùng tiến trình, nhưng không cần nếu dữ liệu sẽ được ghi ra file.

Trong quá trình giải tuần tự hóa, *formatter* sẽ gọi phương thức khởi dựng việc giải tuần tự hóa, lại truyền cho nó các tham chiếu `SerializationInfo` và `StreamingContext`.

- Kiểu của bạn phải trích dữ liệu đã-được-tuần-tự-hóa từ đối tượng `SerializationInfo` bằng một trong các phương thức `SerializationInfo.Get*`, ví dụ: `GetString`, `GetInt32`, hay `GetBoolean`.
- Đối tượng `StreamingContext` cung cấp các thông tin về nguồn gốc của dữ liệu đã-được-tuần-tự-hóa, phản ánh logic mà bạn đã hiện thực cho việc tuần tự hóa.



Trong quá trình tuần tự hóa chuẩn, *formatter* không sử dụng khả năng của đối tượng `StreamingContext` để biết các chi tiết về nguồn gốc, đích, và chủ định của dữ liệu được-tuần-tự-hóa. Tuy nhiên, nếu muốn thực hiện quá trình tuần tự hóa tùy biến, bạn có thể cấu hình đối tượng `StreamingContext` của *formatter* trước khi bắt đầu quá trình tuần tự hóa và giải tuần tự hóa. Tham khảo tài liệu *.NET Framework SDK* để có thêm thông tin về lớp `StreamingContext`.

Ví dụ dưới đây trình bày phiên bản đã được chỉnh sửa của lớp `Employee`, có hiện thực giao diện `ISerializable`. Trong phiên bản này, lớp `Employee` không tuần tự hóa trường `address` nếu đối tượng `StreamingContext` chỉ định rằng đích của dữ liệu được-tuần-tự-hóa là file. Phương thức `Main` sẽ giải thích việc tuần tự hóa và giải tuần tự hóa của một đối tượng `Employee`.

```
using System;
using System.Runtime.Serialization;

[Serializable]
public class Employee : ISerializable {

    private string name;
    private int age;
    private string address;

    // Phương thức khởi dựng đơn giản.
    public Employee(string name, int age, string address) {

        this.name = name;
        this.age = age;
        this.address = address;
    }
}
```

```

// Phương thức khởi dựng dùng để kích hoạt formatter thực hiện việc
// giải tuần tự hóa một đối tượng Employee. Bạn nên khai báo
// phương thức khởi dựng này là private, hay ít nhất cũng là
// protected để bảo đảm nó không bị gọi quá mức cần thiết.
private Employee(SerializationInfo info, StreamingContext context) {

    // Trích xuất tên và tuổi của Employee (sẽ luôn hiện diện
    // trong dữ liệu đã-được-tuần-tự-hóa bất chấp giá trị
    // của StreamingContext).
    name = info.GetString("Name");
    age = info.GetInt32("Age");

    // Thực hiện trích xuất địa chỉ của Employee
    // (thất bại nếu không có).
    try {
        address = info.GetString("Address");
    } catch (SerializationException) {
        address = null;
    }
}

// Các thuộc tính Name, Age, và Address (đã trình bày ở trên).
$

// Được khai báo bởi giao diện ISerializable, phương thức
// GetObjectData cung cấp cơ chế để formatter thu lấy
// dữ liệu sẽ-được-tuần-tự-hóa.
public void GetObjectData(SerializationInfo inf,
    StreamingContext con){

    // Luôn tuần tự hóa tên và tuổi của Employee.
    inf.AddValue("Name", name);
    inf.AddValue("Age", age);

    // Không tuần tự hóa địa chỉ của Employee nếu StreamingContext
    // cho biết rằng dữ liệu được-tuần-tự-hóa sẽ được ghi ra file.

```

```

        if ((con.State & StreamingContextStates.File) == 0) {

            inf.AddValue("Address", address);
        }
    }

    // Chép đè Object.ToString để trả về chuỗi mô tả Employee.
    public override string ToString() {

        StringBuilder str = new StringBuilder();

        str.AppendFormat("Name: {0}\n\r", Name);
        str.AppendFormat("Age: {0}\n\r", Age);
        str.AppendFormat("Address: {0}\n\r", Address);

        return str.ToString();
    }

    public static void Main(string[] args) {

        // Tạo một đối tượng Employee mô tả Phuong.
        Employee phuong = new Employee("Phuong", 23, "HCM");

        // Hiện thị Phuong.
        Console.WriteLine(phuong);

        // Tuần tự hóa Phuong với đích là một miền ứng dụng khác.
        // Địa chỉ của Phuong sẽ được tuần tự hóa.
        Stream str = File.Create("phuong.bin");
        BinaryFormatter bf = new BinaryFormatter();
        bf.Context =
            new StreamingContext(StreamingContextStates.CrossAppDomain);
        bf.Serialize(str, phuong);
        str.Close();

        // Giải tuần tự hóa và hiện thị Phuong.
        str = File.OpenRead("phuong.bin");
    }

```

```

bf = new BinaryFormatter();
phuong = (Employee)bf.Deserialize(str);
str.Close();
Console.WriteLine(phuong);

// Tuần tự hóa Phuong với đích là file. Trong trường hợp này,
// địa chỉ của Phuong sẽ không được tuần tự hóa.
str = File.Create("phuong.bin");
bf = new BinaryFormatter();
bf.Context = new StreamingContext(StreamingContextStates.File);
bf.Serialize(str, phuong);
str.Close();

// Giải tuần tự hóa và hiển thị Phuong.
str = File.OpenRead("phuong.bin");
bf = new BinaryFormatter();
phuong = (Employee)bf.Deserialize(str);
str.Close();
Console.WriteLine(phuong);

Console.ReadLine();
}

}

```

2. *Hiện thực kiểu khả-sao-chép (cloneable type)*

? Bạn cần tạo một kiểu tùy biến cung cấp một cơ chế đơn giản để lập trình viên tạo bản sao cho các thể hiện của kiểu.

✕ Hiện thực giao diện `System.ICloneable`.

Khi gán một kiểu giá trị sang một kiểu giá trị khác là bạn đã tạo một bản sao của giá trị đó. Không có mối liên hệ nào giữa hai giá trị—một thay đổi trên giá trị này sẽ không ảnh hưởng đến giá trị kia. Tuy nhiên, khi gán một kiểu tham chiếu sang một kiểu tham chiếu khác (ngoại trừ chuỗi—được bộ thực thi xử lý đặc biệt), bạn không tạo một bản sao mới của kiểu tham chiếu. Thay vào đó, cả hai kiểu tham chiếu đều chỉ đến cùng một đối tượng, và những thay đổi trên giá trị của đối tượng đều được phản ánh trong cả hai tham chiếu. Để tạo một bản sao thật của một kiểu tham chiếu, bạn phải “nhái” lại đối tượng mà nó chỉ đến.

Giao diện `ICloneable` nhận dạng một kiểu là khả-sao-chép và khai báo phương thức `Clone` là một cơ chế mà thông qua đó, bạn có thể thu lấy bản sao của một đối tượng. Phương thức `Clone` không nhận đối số nào và trả về một `System.Object`, bất chấp kiểu đang hiện thực là gì. Điều này nghĩa là một khi đã sao một đối tượng, bạn phải ép bản sao về đúng kiểu.

Cách hiện thực phương thức `Clone` cho một kiểu tùy biến tùy thuộc vào các thành viên dữ liệu được khai báo bên trong kiểu. Nếu kiểu tùy biến chỉ chứa các thành viên dữ liệu kiểu giá trị (`int`, `byte`...) và `System.String`, bạn có thể hiện thực phương thức `Clone` bằng cách tạo một đối tượng mới và thiết lập các thành viên dữ liệu của nó có giá trị giống như đối tượng hiện tại. Lớp `Object` (tất cả các kiểu đều dẫn xuất từ đây) chứa phương thức `MemberwiseClone` dùng để tự động hóa quá trình này. Ví dụ dưới đây trình bày một lớp đơn giản có tên là `Employee`, chỉ chứa các thành viên chuỗi. Do đó, phương thức `Clone` dựa vào phương thức thừa kế `MemberwiseClone` để tạo bản sao.

```
using System;

public class Employee : ICloneable {

    public string Name;
    public string Title;

    // Phương thức khởi dựng đơn giản.
    public Employee(string name, string title) {

        Name = name;
        Title = title;
    }

    // Tạo một bản sao bằng phương thức Object.MemberwiseClone
    // vì lớp Employee chỉ chứa các tham chiếu chuỗi.
    public object Clone() {

        return MemberwiseClone();
    }
}
```

Nếu kiểu tùy biến của bạn có chứa các thành viên dữ liệu kiểu tham chiếu, bạn phải quyết định xem phương thức `Clone` của bạn sẽ thực hiện một bản sao cạn (*shallow copy*) hay một bản sao sâu (*deep copy*). Bản sao cạn nghĩa là bất kỳ thành viên dữ liệu kiểu tham chiếu nào trong bản sao đều sẽ chỉ đến đối tượng giống như thành viên dữ liệu kiểu tham chiếu tương ứng trong đối tượng gốc. Bản sao sâu nghĩa là bạn phải sao toàn bộ đồ thị đối tượng (*object graph*) để các thành viên dữ liệu kiểu tham chiếu của bản sao chỉ đến các bản sao (độc lập về mặt vật lý) của các đối tượng được tham chiếu bởi đối tượng gốc.

Dễ dàng hiện thực một bản sao cạn—sử dụng phương thức `MemberwiseClone` vừa được mô tả. Tuy nhiên, một bản sao sâu thường là cái mà lập trình viên mong đợi khi lần đầu tiên sao một đối tượng—nhưng hiếm khi là cái họ lấy. Điều này đặc biệt đúng đối với các lớp tập hợp trong không gian tên `System.Collections`, tất cả đều hiện thực bản sao cạn trong các phương thức `Clone` của chúng. Mặc dù sẽ có ích nếu các tập hợp này hiện thực bản sao sâu, có hai lý do chính để các kiểu (đặc biệt là các lớp tập hợp) không hiện thực bản sao sâu:

- Việc tạo bản sao của một đồ thị đối tượng lớn sẽ tốn nhiều bộ nhớ và thời gian xử lý.
- Các tập hợp thông thường có thể chứa các đồ thị đối tượng sâu và rộng, bao gồm bất kỳ kiểu đối tượng nào. Việc tạo một hiện thực bản sao sâu để phục vụ nhiều thứ như thế là không khả thi vì một số đối tượng trong tập hợp có thể không phải là khả-sao-chép, và một số khác có thể chứa các tham chiếu vòng, khiến quá trình sao chép trở thành một vòng lặp vô tận.

Đối với các tập hợp kiểu mạnh, trong đó bản chất của các phần tử được hiểu và được kiểm soát thì một bản sao sâu có thể là một tính năng rất hữu ích. Ví dụ, `System.Xml.XmlNode` hiện thực một bản sao sâu trong phương thức `Clone`, điều này cho phép bạn tạo đúng bản sao của toàn bộ hệ thống phân cấp đối tượng *XML* chỉ với một lệnh đơn.



Nếu cần sao một đối tượng không hiện thực `ICloneable` nhưng lại là khả-tuần-tự-hóa, bạn có thể tuần tự hóa rồi giải tuần tự hóa đối tượng đó để có được cùng kết quả như khi sao chép. Tuy nhiên, quá trình tuần tự hóa có thể không tuần tự hóa tất cả các thành viên dữ liệu (như đã được thảo luận trong mục 16.1). Cũng vậy, nếu tạo một kiểu khả-tuần-tự-hóa tùy biến, bạn có thể sử dụng quá trình tuần tự hóa vừa được mô tả để thực hiện một bản sao sâu bên trong phương thức `ICloneable.Clone`. Để sao một đối tượng khả-tuần-tự-hóa, bạn hãy sử dụng lớp `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter` để tuần tự hóa đối tượng này thành một đối tượng `System.IO.MemoryStream`, và rồi giải tuần tự hóa đối tượng này từ `System.IO.MemoryStream`.

Lớp `Team` dưới đây hiện thực phương thức `Clone` thực hiện một bản sao sâu. Lớp `Team` chứa một tập hợp các đối tượng `Employee` mô tả một nhóm người. Khi bạn gọi phương thức `Clone` của một đối tượng `Team`, phương thức này sẽ tạo bản sao của mỗi đối tượng `Employee` và thêm nó vào đối tượng `Team` được sao. Lớp `Team` cung cấp một phương thức khởi dựng `private` để đơn giản hóa mã lệnh trong phương thức `Clone` (sử dụng phương thức khởi dựng là cách thông thường để đơn giản hóa quá trình sao chép).

```
using System;

using System.Collections;

public class Team : ICloneable {

    public ArrayList TeamMembers = new ArrayList();

    public Team() {
```



```

    }

    // Phương thức khởi dựng private – được phương thức Clone gọi
    // để tạo một đối tượng Team mới và đổ vào ArrayList của nó
    // bản sao của các đối tượng Employee từ một ArrayList có trước.
    private Team(ArrayList members) {

        foreach (Employee e in members) {

            TeamMembers.Add(e.Clone());

        }

    }

    // Thêm một đối tượng Employee vào Team.
    public void AddMember(Employee member) {

        TeamMembers.Add(member);

    }

    public object Clone() {

        // Tạo một bản sao sâu của Team bằng cách gọi phương thức
        // khởi dựng Team và truyền cho nó ArrayList chứa
        // các thành viên của Team.
        return new Team(this.TeamMembers);

        // Lệnh này sẽ tạo một bản sao cạn của Team:
        // return MemberwiseClone();

    }

}

```

3. *Hiện thực kiểu khả-so-sánh (comparable type)*

- ?** Bạn cần một cơ chế dùng để so sánh các kiểu tùy biến, cho phép bạn dễ dàng sắp xếp tập hợp chứa các thể hiện của kiểu này.
- X** Để cung cấp một cơ chế so sánh chuẩn cho một kiểu, hiện thực giao diện `System.IComparable`. Để hỗ trợ nhiều dạng so sánh, tạo riêng từng kiểu trợ giúp (*helper*) và các kiểu này hiện thực giao diện `System.Collections.IComparer`.

Nếu muốn sắp xếp kiểu của bạn chỉ theo một thứ tự nào đó (như *ID* tăng dần, hay tên theo thứ tự alphabet), bạn nên hiện thực giao diện `IComparable`. Giao diện này định nghĩa phương thức `CompareTo` như sau:

```
int CompareTo(object obj);
```

Đối tượng (`obj`) được truyền cho phương thức phải cùng kiểu với đối tượng đang gọi, nếu không `CompareTo` sẽ ném ngoại lệ `System.ArgumentException`. Giá trị do `CompareTo` trả về được tính như sau:

- Nếu đối tượng hiện tại nhỏ hơn `obj`, trả về một số âm (chẳng hạn, -1).
- Nếu đối tượng hiện tại có cùng giá trị như `obj`, trả về zero.
- Nếu đối tượng hiện tại lớn hơn `obj`, trả về một số dương (chẳng hạn, 1).

Phép so sánh này thực hiện điều gì là tùy thuộc vào kiểu đã hiện thực giao diện `IComparable`. Ví dụ, nếu muốn sắp xếp dựa theo tên, bạn cần thực hiện phép so sánh chuỗi (`String`). Tuy nhiên, nếu muốn sắp xếp dựa theo ngày sinh, bạn cần thực hiện phép so sánh ngày (`System.DateTime`).

Để hỗ trợ nhiều dạng sắp xếp cho một kiểu cụ thể, bạn phải hiện thực riêng rẽ từng kiểu trợ giúp và các kiểu này hiện thực giao diện `IComparer`. Giao diện này định nghĩa phương thức `Compare` như sau:

```
int Compare(object x, object y);
```

Kiểu trợ giúp phải đóng gói logic cần thiết để so sánh hai đối tượng và trả về một giá trị dựa trên logic như sau:

- Nếu `x` nhỏ hơn `y`, trả về một số âm (chẳng hạn, -1).
- Nếu `x` có cùng giá trị như `y`, trả về zero.
- Nếu `x` lớn hơn `y`, trả về một số dương (chẳng hạn, 1).

Lớp `Newspaper` dưới đây hiện thực cả giao diện `IComparable` và `IComparer`. Phương thức `Newspaper.CompareTo` thực hiện phép so sánh không phân biệt chữ hoa-thường hai đối tượng `Newspaper` dựa trên trường `name` của chúng. Một lớp `private` lồng bên trong có tên là `AscendingCirculationComparer` hiện thực `IComparer` và so sánh hai đối tượng `Newspaper` dựa trên trường `circulation` của chúng. Đối tượng `AscendingCirculationComparer` được thu lấy bằng thuộc tính tĩnh `Newspaper.CirculationSorter`.

```
using System;
```

```
using System.Collections;
```

```
public class Newspaper : IComparable {
```

```
    private string name;
```

```
    private int circulation;
```

```

private class AscendingCirculationComparer : IComparer {

    int IComparer.Compare(object x, object y) {

        // Xử lý các tham chiếu null.
        // Null được coi như nhỏ hơn bất cứ giá trị nào khác.
        if (x == null && y == null) return 0;
        else if (x == null) return -1;
        else if (y == null) return 1;

        // Trường hợp x và y tham chiếu đến cùng một đối tượng.
        if (x == y) return 0;

        // Bảo đảm x và y đều là các thể hiện của Newspaper.
        Newspaper newspaperX = x as Newspaper;
        if (newspaperX == null) {

            throw new ArgumentException("Invalid object type", "x");
        }

        Newspaper newspaperY = y as Newspaper;
        if (newspaperY == null) {

            throw new ArgumentException("Invalid object type", "y");
        }

        // So sánh circulation. IComparer quy định rằng:
        //      trả về một số âm nếu x < y
        //      trả về zero nếu x = y
        //      trả về một số dương nếu x > y
        // Dễ dàng hiện thực logic này bằng phép tính số nguyên.
        return newspaperX.circulation - newspaperY.circulation;
    }
}

public Newspaper(string name, int circulation) {

```

```
this.name = name;
this.circulation = circulation;
}

// Khai báo một thuộc tính chỉ-đọc, trả về một thể hiện của
// AscendingCirculationComparer.
public static IComparer CirculationSorter{
    get { return new AscendingCirculationComparer(); }
}

public override string ToString() {

    return string.Format("{0}: Circulation = {1}", name,
        circulation);
}

// Phương thức CompareTo so sánh hai đối tượng Newspaper dựa trên
// phép so sánh trường name (không phân biệt chữ hoa-thường).
public int CompareTo(object obj) {

    // Một đối tượng luôn được coi như lớn hơn null.
    if (obj == null) return 1;

    // Trường hợp đối tượng kia là một tham chiếu đến đối tượng này.
    if (obj == this) return 0;

    // Ép đối tượng kia về Newspaper.
    Newspaper other = obj as Newspaper;

    // Nếu "other" là null, nó không phải là một thể hiện của
    // Newspaper. Trong trường hợp này, CompareTo phải ném
    // ngoại lệ System.ArgumentException.
    if (other == null) {

        throw new ArgumentException("Invalid object type", "obj");

    } else {
```

```

        // Tính giá trị trả về bằng cách thực hiện phép so sánh
        // trường name (không phân biệt chữ hoa-thường).

        // Vì name là chuỗi nên cách dễ nhất là dựa vào khả năng
        // so sánh của lớp String (thực hiện phép so sánh chuỗi
        // có phân biệt bản địa).
        return string.Compare(this.name, other.name, true);
    }
}
}

```

Phương thức `Main` minh họa phép so sánh và khả năng sắp xếp nhờ có hiện thực giao diện `IComparable` và `IComparer`. Phương thức này sẽ tạo một tập hợp `System.Collections.ArrayList` chứa năm đối tượng `Newspaper`, sau đó sắp xếp `ArrayList` hai lần bằng phương thức `ArrayList.Sort`. Lần đầu, thao tác `Sort` sử dụng cơ chế so sánh mặc định của `Newspaper` (thông qua phương thức `IComparable.CompareTo`). Lần sau, thao tác `Sort` sử dụng đối tượng `AscendingCirculationComparer` (thông qua phương thức `IComparer.Compare`).

```

public static void Main() {

    ArrayList newspapers = new ArrayList();

    newspapers.Add(new Newspaper("Tuoi Tre", 125780));
    newspapers.Add(new Newspaper("Echip", 55230));
    newspapers.Add(new Newspaper("Thanh Nien", 235950));
    newspapers.Add(new Newspaper("Phu Nu", 88760));
    newspapers.Add(new Newspaper("Tiep Thi", 5670));

    Console.WriteLine("Unsorted newspaper list:");
    foreach (Newspaper n in newspapers) {
        Console.WriteLine(n);
    }

    Console.WriteLine(Environment.NewLine);
    Console.WriteLine("Newspaper list sorted by name (default order):");
    newspapers.Sort();
    foreach (Newspaper n in newspapers) {
        Console.WriteLine(n);
    }
}

```

```

        Console.WriteLine(Environment.NewLine);
        Console.WriteLine("Newspaper list sorted by circulation:");
        newspapers.Sort(Newspaper.CirculationSorter);
        foreach (Newspaper n in newspapers) {
            Console.WriteLine(n);
        }
    }
}

```

Chạy phương thức `Main` sẽ sinh ra kết quả như sau:

Unsorted newspaper list:

```

Tuoi Tre: Circulation = 125780
Echip: Circulation = 55230
Thanh Nien: Circulation = 235950
Phu Nu: Circulation = 88760
Tiep Thi: Circulation = 5670

```

Newspaper list sorted by name (default order):

```

Echip: Circulation = 55230
Phu Nu: Circulation = 88760
Thanh Nien: Circulation = 235950
Tiep Thi: Circulation = 5670
Tuoi Tre: Circulation = 125780

```

Newspaper list sorted by circulation:

```

Tiep Thi: Circulation = 5670
Echip: Circulation = 55230
Phu Nu: Circulation = 88760
Tuoi Tre: Circulation = 125780
Thanh Nien: Circulation = 235950

```

4.

Hiện thực kiểu khả-liệt-kệ (enumerable type)

?

Bạn cần tạo một kiểu tập hợp sao cho nội dung của nó có thể được liệt kê bằng lệnh `foreach`.

✕

Hiện thực giao diện `System.IEnumerable` trong kiểu tập hợp của bạn. Phương thức `GetEnumerator` của giao diện `IEnumerable` trả về một enumerator—một đối

tượng có hiện thực giao diện `System.IEnumerator`. Giao diện `IEnumerator` định nghĩa các phương thức sẽ được lệnh `foreach` sử dụng để liệt kê tập hợp.

Một bộ chỉ mục bằng số (*numerical indexer*) cho phép bạn duyệt qua các phần tử của một tập hợp bằng vòng lặp `for`. Tuy nhiên, kỹ thuật này không cung cấp mức trừu tượng phù hợp với các cấu trúc dữ liệu phi tuyến, như cây và tập hợp đa chiều. Lệnh `foreach` cung cấp một cơ chế duyệt qua các đối tượng của một tập hợp mà không quan tâm cấu trúc bên trong của chúng là gì.

Để hỗ trợ ngữ nghĩa `foreach`, đối tượng chứa tập hợp phải hiện thực giao diện `System.IEnumerable`. Giao diện này khai báo một phương thức có tên là `GetEnumerator`, phương thức này không nhận đối số và trả về một đối tượng `System.IEnumerator`:

```
IEnumerator GetEnumerator();
```

Đối tượng `IEnumerator` là đối tượng hỗ trợ việc liệt kê các phần tử của tập hợp. Giao diện `IEnumerator` cung cấp một con chạy chỉ-đọc, chỉ-tiến (*read-only, forward-only cursor*) dùng để truy xuất các thành viên của tập hợp nằm dưới. Bảng 16.1 mô tả các thành viên của giao diện `IEnumerator`.

Bảng 16.1 Các thành viên của giao diện `IEnumerator`

Thành viên	Mô tả
Current	Thuộc tính này trả về phần tử dữ liệu hiện tại. Khi enumerator được tạo ra, <code>Current</code> chỉ đến vị trí đứng trước phần tử dữ liệu đầu tiên, nghĩa là bạn phải gọi <code>MoveNext</code> trước khi sử dụng <code>Current</code> . Nếu <code>Current</code> được gọi và enumerator đang đứng trước phần tử đầu tiên hoặc sau phần tử cuối cùng trong tập hợp dữ liệu, <code>Current</code> sẽ ném ngoại lệ <code>System.InvalidOperationException</code> .
MoveNext	Phương thức này dịch chuyển enumerator sang phần tử dữ liệu kế tiếp trong tập hợp; trả về <code>true</code> nếu còn phần tử, trả về <code>false</code> nếu không còn phần tử. Nếu nguồn dữ liệu nằm dưới thay đổi trong thời gian sống của enumerator, <code>MoveNext</code> sẽ ném ngoại lệ <code>InvalidOperationException</code> .
Reset	Phương thức này dịch chuyển enumerator về vị trí đứng trước phần tử đầu tiên trong tập hợp dữ liệu. Nếu nguồn dữ liệu nằm dưới thay đổi trong thời gian sống của enumerator, <code>Reset</code> sẽ ném ngoại lệ <code>InvalidOperationException</code> .

Các lớp `TeamMember`, `Team`, và `TeamMemberEnumerator` minh họa việc hiện thực giao diện `IEnumerable` và `IEnumerator`. Lớp `TeamMember` mô tả một thành viên của một đội:

```
// Lớp TeamMember mô tả một thành viên trong đội.
public class TeamMember {

    public string Name;
    public string Title;

    // Phương thức khởi dựng đơn giản.
```

```

public TeamMember(string name, string title) {

    Name = name;
    Title = title;
}

// Trả về chuỗi mô tả TeamMember.
public override string ToString() {

    return string.Format("{0} ({1})", Name, Title);
}
}

```

Lớp *Team* (mô tả một đội) là một tập hợp các đối tượng *TeamMember*. Lớp này hiện thực giao diện *IEnumerable* và khai báo một lớp có tên là *TeamMemberEnumerator* để cung cấp chức năng liệt kê. Thông thường, các lớp tập hợp sẽ trực tiếp hiện thực cả giao diện *IEnumerable* và *IEnumerator*. Tuy nhiên, sử dụng một lớp enumerator riêng biệt là cách đơn giản nhất để cho phép nhiều enumerator—và nhiều tiểu trình—liệt kê đồng thời các phần tử của *Team*.

Team hiện thực mẫu *Observer* bằng cách sử dụng các thành viên sự kiện và ủy nhiệm để báo cho tất cả các đối tượng *TeamMemberEnumerator* biết *Team* nằm dưới có thay đổi hay không (xem mục 16.10 để có thêm thông tin về mẫu *Observer*). Lớp *TeamMemberEnumerator* là một lớp private lồng bên trong nên bạn không thể tạo các thể hiện của nó, trừ khi thông qua phương thức *Team.GetEnumerator*. Dưới đây là phần mã cho lớp *Team* và *TeamMemberEnumerator*:

```

// Lớp Team mô tả tập hợp các đối tượng TeamMember. Hiện thực giao diện
// IEnumerable để hỗ trợ việc liệt kê các đối tượng TeamMember.
public class Team : IEnumerable {

    // TeamMemberEnumerator là một lớp private lồng bên trong, cung cấp
    // chức năng liệt kê các đối tượng TeamMember trong tập hợp
    // Team. Vì là lớp lồng bên trong nên TeamMemberEnumerator
    // có thể truy xuất các thành viên private của lớp Team.
    private class TeamMemberEnumerator : IEnumerator {

        private Team sourceTeam;

        // Giá trị luận lý cho biết Team nằm dưới có thay đổi hay không.
        private bool teamInvalid = false;
    }
}

```



```

// Giá trị nguyên cho biết TeamMember hiện tại (chỉ số
// trong ArrayList). Giá trị ban đầu là -1.
private int currentMember = -1;

// Phương thức khởi dựng (nhận một tham chiếu đến Team).
internal TeamMemberEnumerator(Team team) {

    this.sourceTeam = team;

    sourceTeam.TeamChange +=
        new TeamChangedEventHandler(this.TeamChange);
}

// Hiện thực thuộc tính IEnumerator.Current.
public object Current {
    get {

        // Nếu TeamMemberEnumerator đứng trước phần tử đầu tiên
        // hoặc sau phần tử cuối cùng thì ném ngoại lệ.
        if (currentMember == -1 ||
            currentMember > (sourceTeam.teamMembers.Count-1)) {

            throw new InvalidOperationException();
        }

        // Nếu không, trả về TeamMember hiện tại.
        return sourceTeam.teamMembers[currentMember];
    }
}

// Hiện thực phương thức IEnumerator.MoveNext.
public bool MoveNext() {

    // Nếu Team nằm dưới bất hợp lệ, ném ngoại lệ.
    if (teamInvalid) {

```

```

        throw new InvalidOperationException("Team modified");
    }

    // Nếu không, tiến đến TeamMember kế tiếp.
    currentMember++;

    // Trả về false nếu ta dịch qua khỏi TeamMember cuối cùng.
    if (currentMember > (sourceTeam.teamMembers.Count-1)) {
        return false;
    } else {
        return true;
    }
}

// Hiện thực phương thức IEnumerator.Reset. Phương thức này
// reset vị trí của TeamMemberEnumerator về đầu tập hợp Team.
public void Reset() {

    // Nếu Team nằm dưới bất hợp lệ, ném ngoại lệ.
    if (teamInvalid) {

        throw new InvalidOperationException("Team modified");
    }

    // Dịch con trỏ currentMember về trước phần tử đầu tiên.
    currentMember = -1;
}

// Phương thức thụ lý sự kiện tập hợp Team nằm dưới thay đổi.
internal void TeamChange(Team t, EventArgs e) {

    // Báo hiệu Team nằm dưới hiện đang bất hợp lệ.
    teamInvalid = true;
}
}

```

```

// Ủy nhiệm dùng để chỉ định chữ ký mà tất cả
// các phương thức thụ lý sự kiện phải hiện thực.
public delegate void TeamChangedEventHandler(Team t, EventArgs e);

// ArrayList dùng để chứa các đối tượng TeamMember.
private ArrayList teamMembers;

// Sự kiện dùng để báo cho TeamMemberEnumerator
// biết Team đã thay đổi.
public event TeamChangedEventHandler TeamChange;

// Phương thức khởi dựng Team.
public Team() {

    teamMembers = new ArrayList();
}

// Hiện thực phương thức IEnumerable.GetEnumerator.
public IEnumerator GetEnumerator() {

    return new TeamMemberEnumerator(this);
}

// Thêm một đối tượng TeamMember vào Team.
public void AddMember(TeamMember member) {

    teamMembers.Add(member);

    if (TeamChange != null) {

        TeamChange(this, null);
    }
}
}

```

Nếu lớp tập hợp của bạn chứa nhiều kiểu dữ liệu khác nhau và bạn muốn liệt kê chúng một cách riêng rẽ, việc hiện thực giao diện `IEnumerable` trên lớp tập hợp này thì vẫn còn thiếu. Trong trường hợp này, bạn cần hiện thực một số thuộc tính trả về các thể hiện khác nhau của

`IEnumerator`. Ví dụ, nếu lớp `Team` mô tả cả các thành viên và các máy tính trong đội, bạn có thể hiện thực các thuộc tính này như sau:

```
// Thuộc tính dùng để liệt kê các thành viên trong đội.
public IEnumerator Members {
    get {
        return new TeamMemberEnumerator(this);
    }
}

// Thuộc tính dùng để liệt kê các computer trong đội.
public IEnumerator Computers {
    get {
        return new TeamComputerEnumerator(this);
    }
}
```

Khi đó, bạn có thể sử dụng các enumerator này như sau:

```
Team team = new Team();

foreach(TeamMember in team.Members) {
    // Làm gì đó...
}

foreach(TeamComputer in team.Computers) {
    // Làm gì đó...
}
```



Lệnh `foreach` cũng hỗ trợ các kiểu có hiện thực một mẫu tương đương với mẫu được định nghĩa bởi giao diện `IEnumerable` và `IEnumerator`, mặc dù kiểu đó không hiện thực các giao diện này. Tuy nhiên, mã lệnh của bạn sẽ rõ ràng hơn và dễ hiểu hơn nếu bạn hiện thực giao diện `IEnumerable`. Bạn hãy xem *C# Language Specification* để biết chi tiết về các yêu cầu của lệnh `foreach` [<http://msdn.microsoft.com/net/ecma>].

5.

Hiện thực lớp khả-hủy (*disposable class*)

Bạn cần tạo một lớp có tham chiếu đến các tài nguyên không-được-quản-lý và cung cấp một cơ chế để người dùng giải phóng các tài nguyên đó một cách tất định.



Hiện thực giao diện `System.IDisposable`, và giải phóng các tài nguyên không-được-quản-lý khi mã client gọi phương thức `IDisposable.Dispose`.

Một đối tượng không được tham chiếu đến vẫn tồn tại trên vùng nhớ động (*heap*) và tiêu thụ các tài nguyên cho đến khi bộ thu gom rác (*Garbage Collector*) giải phóng đối tượng và các tài nguyên. Bộ thu gom rác sẽ tự động giải phóng các tài nguyên được-quản-lý (như bộ nhớ), nhưng nó sẽ không giải phóng các tài nguyên không-được-quản-lý (như file handle và kết nối cơ sở dữ liệu) được tham chiếu bởi các đối tượng được-quản-lý. Nếu một đối tượng chứa các thành viên dữ liệu tham chiếu đến các tài nguyên không-được-quản-lý, đối tượng này phải giải phóng các tài nguyên đó.

Một giải pháp là khai báo một destructor—hay finalizer—cho lớp. Trước khi giải phóng phần bộ nhớ do một thể hiện của lớp sử dụng, bộ thu gom rác sẽ gọi finalizer của đối tượng này. Finalizer có thể thực hiện các bước cần thiết để giải phóng các tài nguyên không-được-quản-lý. Vì bộ thu gom rác chỉ sử dụng một tiểu trình để thực thi tất cả các finalizer, việc sử dụng finalizer có thể bất lợi trong quá trình thu gom rác và ảnh hưởng đến hiệu năng của ứng dụng. Ngoài ra, bạn không thể kiểm soát khi bộ thực thi giải phóng các tài nguyên không-được-quản-lý vì bạn không thể trực tiếp gọi finalizer của một đối tượng, và bạn chỉ có quyền kiểm soát hạn chế trên các hoạt động của bộ thu gom rác bằng lớp `System.GC`.

Bằng cách sử dụng finalizer, *.NET Framework* định nghĩa mẫu *Dispose* như một phương tiện cung cấp quyền kiểm soát khi bộ thực thi giải phóng các tài nguyên không-được-quản-lý. Để hiện thực mẫu *Dispose*, lớp phải hiện thực giao diện `IDisposable`. Giao diện này khai báo một phương thức có tên là `Dispose`; trong đó, bạn phải hiện thực phần mã cần thiết để giải phóng các tài nguyên không-được-quản-lý.

Các thể hiện của các lớp có hiện thực mẫu *Dispose* được gọi là các đối tượng khả-hủy (*disposable object*). Khi mã lệnh đã hoàn tất với một đối tượng khả-hủy, nó sẽ gọi phương thức `Dispose` của đối tượng để giải phóng các tài nguyên không-được-quản-lý, vẫn dựa vào bộ thu gom rác để giải phóng các tài nguyên được-quản-lý của đối tượng. Cần hiểu rằng bộ thực thi không bắt buộc hủy các đối tượng; việc gọi phương thức `Dispose` là nhiệm vụ của client. Tuy nhiên, vì thư viện lớp *.NET Framework* sử dụng mẫu *Dispose* rộng khắp nên *C#* cung cấp lệnh `using` để đơn giản hóa việc sử dụng các đối tượng khả-hủy. Đoạn mã sau trình bày cấu trúc của lệnh `using`:

```
using (FileStream fileStream = new FileStream("SomeFile.txt",
    FileMode.Open)) {
    // Làm gì đó với đối tượng fileStream...
}
```

Dưới đây là một số điểm cần lưu ý khi hiện thực mẫu *Dispose*:

- Mã client nên có khả năng gọi đi gọi lại phương thức `Dispose` mà không gây ra các ảnh hưởng bất lợi.
- Trong các ứng dụng hỗ trợ đa-tiểu-trình, điều quan trọng là chỉ có một tiểu trình thực thi phương thức `Dispose`. Thông thường, bảo đảm sự đồng bộ tiểu trình là nhiệm vụ của mã client, mặc dù bạn có thể hiện thực sự đồng bộ bên trong phương thức `Dispose`.
- Phương thức `Dispose` không nên ném ngoại lệ.
- Vì phương thức `Dispose` dọn dẹp tất cả nên không cần gọi finalizer của đối tượng. Phương thức `Dispose` của bạn nên gọi phương thức `GC.SuppressFinalize` để bảo đảm finalizer không được gọi trong quá trình thu gom rác.
- Hiện thực một finalizer sao cho phương thức `Dispose` sẽ được nó gọi theo một cơ chế an toàn trong trường hợp mã client gọi `Dispose` không đúng. Tuy nhiên, nên tránh tham chiếu đến các đối tượng được-quản-lý trong finalizer vì không rõ trạng thái của đối tượng.
- Nếu một lớp khả-hủy thừa kế một lớp khả-hủy khác, phương thức `Dispose` của lớp con phải gọi phương thức `Dispose` của lớp cha. Gói phần mã của lớp con trong một khối `try` và gọi phương thức `Dispose` của lớp cha trong một mệnh đề `finally` để bảo đảm việc thực thi.
- Các phương thức và thuộc tính khác của lớp nên ném ngoại lệ `System.ObjectDisposedException` nếu mã client thực thi một phương thức trên một đối tượng đã bị hủy.

Lớp `DisposeExample` dưới đây minh họa một hiện thực phổ biến của mẫu *Dispose*:

```
using System;

// Hiện thực giao diện IDisposable.
public class DisposeExample : IDisposable {

    // Phần tử dữ liệu private dùng để báo hiệu
    // đối tượng đã bị hủy hay chưa.
    bool isDisposed = false;

    // Phần tử dữ liệu private dùng để lưu giữ
    // handle của tài nguyên không-được-quản-lý.
    private IntPtr resourceHandle;

    // Phương thức khởi dựng.
    public DisposeExample() {

        // Thu lấy tham chiếu đến tài nguyên không-được-quản-lý.
```

```

        // resourceHandle = ...
    }

    // Destructor/Finalizer.
    ~DisposeExample() {

        // Gọi phiên bản nạp chồng protected của Dispose
        // và truyền giá trị "false" để cho biết rằng
        // Dispose đang được gọi trong quá trình thu gom rác,
        // chú không phải bởi mã consumer.
        Dispose(false);
    }

    // Hiện thực public của phương thức IDisposable.Dispose, được gọi
    // bởi consumer của đối tượng để giải phóng các tài nguyên không-
    // được-quản-lý một cách tất định.
    public void Dispose() {

        // Gọi phiên bản nạp chồng protected của Dispose và truyền
        // giá trị "true" để cho biết rằng Dispose đang được gọi
        // bởi mã consumer, chú không phải bởi bộ thu gom rác.
        Dispose(true);

        // Vì phương thức Dispose thực hiện tất cả việc dọn dẹp cần
        // thiết nên bảo đảm bộ thu gom rác không gọi destructor của lớp.
        GC.SuppressFinalize(this);
    }

    // Phiên bản nạp chồng protected của phương thức Dispose. Đối số
    // disposing cho biết phương thức được gọi bởi mã consumer (true),
    // hay bởi bộ thu gom rác (false).
    protected virtual void Dispose(bool disposing) {

        if (!IsDisposed) {

            if (disposing) {

                // Phương thức này được gọi bởi mã consumer. Gọi

```

```

        // phương thức Dispose của các thành viên dữ liệu
        // được-quản-lý có hiện thực giao diện IDisposable.
        // §
    }

    // Giải phóng tất cả các tài nguyên không-được-quản-lý
    // và thiết lập giá trị của các thành viên dữ liệu
    // được-quản-lý thành null.
    // Close(resourceHandle);
}

// Báo rằng đối tượng này đã bị hủy.
disposed = true;
}

// Trước khi thực thi bất kỳ chức năng nào, bảo đảm rằng
// Dispose chưa được thực thi trên đối tượng.
public void SomeMethod() {

    // Ném một ngoại lệ nếu đối tượng đã bị hủy.
    if (disposed) {

        throw new ObjectDisposedException("DisposeExample");
    }

    // Thực thi chức năng của phương thức...
    // §
}

public static void Main() {

    // Lệnh using bảo đảm phương thức Dispose được gọi
    // cả khi ngoại lệ xảy ra.
    using (DisposeExample d = new DisposeExample()) {

        // Làm gì đó với d...
    }
}

```



```

    }
}
}

```

6. *Hiện thực kiểu khả-định-dạng (formattable type)*

? Bạn cần hiện thực một kiểu có thể được sử dụng theo các *format string*, và có thể tạo ra những biểu diễn chuỗi khác nhau cho nội dung của nó dựa vào *format specifier*.

X Hiện thực giao diện `System.IFormattable`.

Đoạn mã dưới đây minh họa cách sử dụng format specifier (phần in đậm) trong phương thức `WriteLine` của lớp `System.Console`.

```

double a = 345678.5678;
uint b = 12000;
byte c = 254;

Console.WriteLine("a = {0}, b = {1}, and c = {2}", a, b, c);
Console.WriteLine("a = {0:c0}, b = {1:n4}, and c = {2,10:x5}", a, b, c);

```

Khi chạy trên máy với thiết lập bản địa là *English (U.K.)*, đoạn mã này sẽ cho kết xuất như sau (thay đổi nội dung của format specifier sẽ thay đổi định dạng của kết xuất một cách đáng kể mặc dù dữ liệu vẫn không thay đổi):

```

a = 345678.5678, b = 12000, and c = 254
a = £345,679, b = 12,000.0000, and c =          000fe

```

Để kích hoạt việc hỗ trợ format specifier, bạn phải hiện thực giao diện `IFormattable`. Giao diện này khai báo một phương thức có tên là `ToString` với chữ ký như sau:

```
string ToString(string format, IFormatProvider formatProvider);
```

Đối số `format` là một `System.String` chứa format string (chuỗi định dạng). Format string là phần format specifier phía sau dấu hai chấm. Ví dụ, trong format specifier `{2,10:x5}` (ở ví dụ trên), `"x5"` là format string. Format string chứa những chỉ thị mà thể hiện `IFormattable` sẽ sử dụng khi tạo ra dạng chuỗi cho nội dung của nó. Tài liệu *.NET Framework* phát biểu rằng: những kiểu có hiện thực `IFormattable` thì phải hỗ trợ format string "G" (*general*), nhưng những format string được hỗ trợ khác thì phụ thuộc vào hiện thực. Đối số `format` là null nếu format specifier không chứa phần format string, ví dụ `{0}` hay `{1,20}`.

Đối số `formatProvider` là tham chiếu đến một thể hiện `System.IFormatProvider` (dùng để truy xuất các thông tin bản địa—bao gồm các dữ liệu như biểu tượng tiền tệ hay số lượng chữ số thập phân). Theo mặc định, `formatProvider` là null, nghĩa là bạn sẽ sử dụng các thiết lập bản địa của tiểu trình hiện hành (có thể lấy được thông qua phương thức tĩnh `CurrentCulture` của lớp `System.Globalization.CultureInfo`).

.NET Framework chủ yếu sử dụng `IFormattable` để hỗ trợ việc định dạng các kiểu giá trị, nhưng nó có thể được sử dụng cho bất kỳ kiểu nào. Ví dụ, lớp `Person` dưới đây có hiện thực giao diện `IFormattable`. Lớp này chứa danh hiệu và tên của một người, và sẽ trả về tên theo

các định dạng khác nhau tùy vào format string. Lớp `Person` không sử dụng các thiết lập bản địa do đối số `formatProvider` cung cấp.

```
using System;

public class Person : IFormattable {

    // Các thành viên private dùng để lưu trữ danh hiệu
    // và tên của một người.
    private string title;
    private string[] names;

    // Phương thức khởi dựng dùng để thiết lập danh hiệu và tên.
    public Person(string title, params string[] names) {

        this.title = title;
        this.names = names;
    }

    // Chép đè phương thức Object.ToString để trả về
    // tên theo định dạng general.
    public override string ToString() {
        return ToString("G", null);
    }

    // Hiện thực phương thức IFormattable.ToString để trả về
    // tên theo các dạng khác nhau dựa trên format string.
    public string ToString(string format,
        IFormatProvider formatProvider) {

        string result = null;

        // Sử dụng định dạng general nếu format = null.
        if (format == null) format = "G";

        // Nội dung của format string cho biết định dạng của tên.
        switch (format.ToUpper()[0]) {
```

```

        case 'S':
            // Sử dụng dạng short: first-initial và surname.
            result = names[0][0] + ". " + names[names.Length-1];
            break;

        case 'P':
            // Sử dụng dạng polite: title, initials, và surname.
            if (title != null && title.Length != 0) {
                result = title + ". ";
            }
            for (int count = 0; count < names.Length; count++) {
                if (count != (names.Length - 1)) {
                    result += names[count][0] + ". ";
                } else {
                    result += names[count];
                }
            }
            break;

        case 'I':
            // Sử dụng dạng informal: chỉ có first-name.
            result = names[0];
            break;

        case 'G':
        default:
            // Sử dụng dạng mặc định/general: first-name và surname.
            result = names[0] + " " + names[names.Length-1];
            break;
    }
    return result;
}
}

```

Đoạn mã dưới đây trình bày cách sử dụng khả năng định dạng của lớp `Person`:

```

// Tạo một đối tượng Person mô tả một người có tên là
// Mr. Richard Glen David Peters.
Person person =

```

```
new Person("Mr", "Richard", "Glen", "David", "Peters");

// Hiển thị tên bằng nhiều format string khác nhau.
System.Console.WriteLine("Dear {0:G},", person);
System.Console.WriteLine("Dear {0:P},", person);
System.Console.WriteLine("Dear {0:I},", person);
System.Console.WriteLine("Dear {0},", person);
System.Console.WriteLine("Dear {0:S},", person);
```

Khi được thực thi, đoạn mã này sinh ra kết xuất như sau:

```
Dear Richard Peters,
Dear Mr. R. G. D. Peters,
Dear Richard,
Dear Richard Peters,
Dear R. Peters,
```

7.

Hiện thực lớp ngoại lệ tùy biến

?

Bạn cần tạo một lớp ngoại lệ tùy biến sao cho bạn có thể sử dụng cơ chế thụ lý ngoại lệ của bộ thực thi để thụ lý các ngoại lệ đặc-trung-ứng-dụng.

✕

Tạo một lớp khả-tuần-tự-hóa, thừa kế lớp `System.ApplicationException` và hiện thực các phương thức khởi dựng với chữ ký như sau:

```
public CustomException() : base() {}
public CustomException(string message) : base(message) {}
public CustomException(string message, Exception inner)
    : base(message, inner) {}
```

Thêm bất cứ thành viên dữ liệu tùy biến nào mà ngoại lệ cần đến, bao gồm các phương thức khởi dựng và các thuộc tính cần thiết để thao tác các thành viên dữ liệu.

Các lớp ngoại lệ là duy nhất, bạn không được khai báo các lớp mới để hiện thực chức năng mới hay mở rộng. Cơ chế thụ lý ngoại lệ của bộ thực thi (được trưng ra bởi các lệnh: `try`, `catch`, và `finally`) làm việc dựa trên kiểu ngoại lệ bị ném, chứ không phải các thành viên chức năng hay dữ liệu được hiện thực bởi ngoại lệ bị ném.

Nếu cần ném một ngoại lệ, bạn nên sử dụng một lớp ngoại lệ có sẵn trong thư viện lớp *.NET Framework* (nếu tồn tại một lớp phù hợp). Dưới đây là một số ngoại lệ hữu ích:

- `System.ArgumentNullException`—khi mã lệnh truyền một giá trị đối số `null` cho một phương thức không hỗ trợ đối số `null`.
- `System.ArgumentOutOfRangeException`—khi mã lệnh truyền cho phương thức một giá trị đối số không phù hợp (lớn quá hay nhỏ quá).

- `System.FormatException`—khi mã lệnh truyền cho phương thức một đối số `String` chứa dữ liệu không được định dạng đúng.

Nếu không có lớp ngoại lệ nào đáp ứng được nhu cầu của bạn, hoặc bạn cảm thấy ứng dụng của bạn sẽ được lợi từ việc sử dụng các ngoại lệ đặc-trung-ứng-dụng, bạn có thể tạo một lớp ngoại lệ cho mình. Để tích hợp ngoại lệ tùy biến với cơ chế thụ lý ngoại lệ của bộ thực thi và vẫn giữ tính nhất quán với mẫu được hiện thực bởi các lớp ngoại lệ có sẵn, bạn cần:

- Đặt một tên có ý nghĩa cho lớp ngoại lệ tùy biến, kết thúc bằng từ `Exception`, chẳng hạn, `TypeMismatchException` hay `RecordNotFoundException`.
- Thừa kế lớp `ApplicationException`. Về cơ bản, lớp ngoại lệ tùy biến phải thừa kế lớp `System.Exception`, nếu không trình biên dịch sẽ dựng lên lỗi khi bạn ném ngoại lệ. `ApplicationException` thừa kế `Exception` và được đề nghị làm lớp sơ sở cho tất cả các lớp ngoại lệ đặc-trung-ứng-dụng.
- Đánh dấu lớp ngoại lệ tùy biến là `sealed` nếu bạn không muốn các lớp ngoại lệ khác có thể thừa kế nó.
- Hiện thực thêm các thuộc tính và các thành viên dữ liệu để hỗ trợ các thông tin tùy biến mà lớp ngoại lệ này cung cấp.
- Hiện thực ba phương thức khởi dựng `public` với chữ ký như dưới đây và bảo đảm chúng gọi phương thức khởi dựng của lớp cơ sở:

```
public CustomException() : base() {}
public CustomException(string message) : base(message) {}
public CustomException(string message, Exception inner)
    : base(message, inner) {}
```

- Làm cho lớp ngoại lệ tùy biến trở nên khả-tuần-tự-hóa để bộ thực thi có thể marshal các thể hiện của nó qua các biên miền ứng dụng và biên máy. Áp dụng đặc tính `System.SerializableAttribute` thường là đủ cho các lớp ngoại lệ không hiện thực các thành viên dữ liệu tùy biến. Tuy nhiên, vì `Exception` hiện thực giao diện `System.Runtime.Serialization.ISerializable` nên nếu ngoại lệ của bạn có khai báo các thành viên dữ liệu tùy biến, bạn phải chép đè phương thức `ISerializable.GetObjectData` của lớp `Exception` cũng như hiện thực một phương thức khởi dựng giải tuần tự hóa với chữ ký như dưới đây. Nếu lớp ngoại lệ của bạn là `sealed`, đánh dấu phương thức khởi dựng giải tuần tự hóa là `private`; nếu không thì đánh dấu nó là `protected`.

```
private CustomException(SerializationInfo info,
    StreamingContext context) {}
```

Phương thức `GetObjectData` và phương thức khởi dựng giải tuần tự hóa phải gọi phương thức tương đương trong lớp cơ sở để cho phép lớp cơ sở thực hiện tuần tự hóa và giải tuần tự hóa dữ liệu của nó một cách đúng đắn (xem mục 16.1 để biết cách làm cho một lớp trở nên khả-tuần-tự-hóa).

Dưới đây là một lớp ngoại lệ tùy biến có tên là `CustomException` (thừa kế lớp `ApplicationException`). Lớp này khai báo hai thành viên dữ liệu tùy biến: một chuỗi có tên là `stringInfo` và một giá trị luận lý có tên là `booleanInfo`.

```

using System;
using System.Runtime.Serialization;

// Đánh dấu CustomException là Serializable (khả-tuần-tự-hóa).
[Serializable]
public sealed class CustomException : ApplicationException {

    // Các thành viên dữ liệu tùy biến cho CustomException.
    private string stringInfo;
    private bool booleanInfo;

    // Ba phương thức khởi dựng chuẩn; chỉ cần gọi phương thức
    // khởi dựng của lớp cơ sở (System.ApplicationException).
    public CustomException() : base() {}

    public CustomException(string message) : base(message) {}

    public CustomException(string message, Exception inner)
        : base(message, inner) {}

    // Phương thức khởi dựng giải tuần tự hóa (cần cho giao diện
    // ISerialization). Vì CustomException là sealed nên phương thức
    // khởi dựng này là private. Nếu CustomException không phải là
    // sealed thì phương thức khởi dựng này nên được khai báo là
    // protected để các lớp dẫn xuất có thể gọi nó trong quá trình
    // giải tuần tự hóa.
    private CustomException(SerializationInfo info,
        StreamingContext context) : base (info, context) {

        // Giải tuần tự hóa mỗi thành viên dữ liệu tùy biến.
        stringInfo = info.GetString("StringInfo");
        booleanInfo = info.GetBoolean("BooleanInfo");
    }

    // Các phương thức khởi dựng cho phép mã lệnh thiết lập

```

```

// các thành viên dữ liệu tùy biến.
public CustomException(string message, string stringInfo,
    bool booleanInfo): this(message) {
    this.stringInfo = stringInfo;
    this.booleanInfo = booleanInfo;
}

public CustomException(string message, Exception inner,
    string stringInfo, bool booleanInfo) : this(message, inner) {
    this.stringInfo = stringInfo;
    this.booleanInfo = booleanInfo;
}

// Các thuộc tính chỉ-đọc cho phép truy xuất đến các
// thành viên dữ liệu tùy biến.
public string StringInfo {
    get { return stringInfo; }
}

public bool BooleanInfo {
    get { return booleanInfo; }
}

// Phương thức GetObjectData (được khai báo trong giao diện
// ISerializable) được sử dụng trong quá trình tuần tự hóa
// CustomException. Vì CustomException có khai báo các thành
// viên dữ liệu tùy biến nên nó phải chép đè hiện thực
// GetObjectData của lớp cơ sở.
public override void GetObjectData(SerializationInfo info,
    StreamingContext context) {

    // Tuần tự hóa các thành viên dữ liệu tùy biến.
    info.AddValue("StringInfo", stringInfo);
    info.AddValue("BooleanInfo", booleanInfo);

    // Gọi lớp cơ sở để tuần tự hóa các thành viên của nó.
    base.GetObjectData(info, context);
}

```

```
// Chép đè thuộc tính Message của lớp cơ sở (để kèm các
// thành viên dữ liệu tùy biến vào).
public override string Message {
    get {
        string message = base.Message;
        if (stringInfo != null) {
            message += Environment.NewLine +
                stringInfo + " = " + booleanInfo;
        }
        return message;
    }
}
```

Trong các ứng dụng lớn, bạn sẽ thường xuyên hiện thực một vài lớp ngoại lệ tùy biến. Bạn cần lưu tâm đến cách tổ chức các ngoại lệ tùy biến và mã lệnh sẽ sử dụng chúng như thế nào. Nói chung, tránh tạo ra các lớp ngoại lệ mới trừ khi mã lệnh cần nỗ lực bắt ngoại lệ đó; sử dụng các thành viên dữ liệu để thu thông tin, chứ không phải các lớp ngoại lệ. Ngoài ra, tránh phân cấp lớp theo chiều sâu mà nên phân cấp cận, theo chiều rộng.

8.

Hiện thực đối số sự kiện tùy biến

- ? Khi dựng lên một sự kiện, bạn cần truyền một trạng thái đặc-trung-sự-kiện cho các phương thức thụ lý sự kiện.
- ✕ Tạo một lớp đối số sự kiện tùy biến dẫn xuất từ lớp `System.EventArgs`. Khi dựng lên sự kiện, hãy tạo một thể hiện của lớp đối số sự kiện và truyền nó cho các phương thức thụ lý sự kiện.

Khi khai báo các kiểu sự kiện, thông thường bạn sẽ cần truyền trạng thái đặc-trung-sự-kiện cho các phương thức thụ lý sự kiện. Để tạo một lớp đối số sự kiện tùy biến tuân theo mẫu *Event* do *.NET Framework* định nghĩa, bạn cần:

- Dẫn xuất lớp đối số sự kiện tùy biến từ lớp `EventArgs`. Lớp `EventArgs` không chứa dữ liệu và được sử dụng cùng với các sự kiện không cần truyền trạng thái.
- Đặt một tên có ý nghĩa cho lớp đối số sự kiện tùy biến, kết thúc bằng từ `EventArgs`; chẳng hạn, `DiskFullEventArgs` hay `MailReceivedEventArgs`.
- Đánh dấu lớp đối số sự kiện là `sealed` nếu bạn không muốn các lớp đối số sự kiện khác có thể thừa kế nó.

- Hiện thực thêm các thuộc tính và các thành viên dữ liệu để hỗ trợ trạng thái sự kiện mà bạn cần truyền cho các phương thức thụ lý sự kiện. Tốt nhất là làm cho trạng thái sự kiện trở nên bất biến (*immutable*), như vậy bạn nên sử dụng các thành viên dữ liệu `private readonly` và sử dụng các thuộc tính `public` để cho phép truy xuất chỉ-đọc đến các thành viên dữ liệu này.
- Hiện thực một phương thức khởi dựng `public` hỗ trợ cấu hình ban đầu của trạng thái sự kiện.
- Làm cho lớp đối số sự kiện của bạn trở nên khả-tuần-tự-hóa (*serializable*) để bộ thực thi có thể marshal các thể hiện của nó qua các biên miền ứng dụng và biên máy. Áp dụng đặc tính `System.SerializableAttribute` thường là đủ cho các lớp đối số sự kiện. Tuy nhiên, nếu lớp đối số sự kiện có các yêu cầu tuần tự hóa đặc biệt, bạn phải hiện thực giao diện `System.Runtime.Serialization.ISerializable` (xem mục 16.1 để biết cách làm cho một lớp trở nên khả-tuần-tự-hóa).

Đoạn mã dưới đây trình bày một lớp đối số sự kiện tùy biến có tên là `MailReceivedEventArgs`. Giả sử có một mail-server truyền các thể hiện của lớp `MailReceivedEventArgs` cho các phương thức thụ lý sự kiện nhận một thông điệp e-mail. Lớp này chứa các thông tin về người gửi và chủ đề của thông điệp e-mail.

```
using System;
```

```
[Serializable]
```

```
public sealed class MailReceivedEventArgs : EventArgs {
```

```
    // Các thành viên private readonly giữ trạng thái sự kiện
    // (được phân bổ cho tất cả các phương thức thụ lý sự kiện).
    // Lớp MailReceivedEventArgs sẽ cho biết ai đã gửi mail
    // và chủ đề là gì.
    private readonly string from;
    private readonly string subject;
```

```
    // Phương thức khởi dựng (khởi tạo trạng thái sự kiện).
```

```
    public MailReceivedEventArgs(string from, string subject) {
```

```
        this.from = from;
        this.subject = subject;
```

```
    }
```

```
    // Các thuộc tính chỉ-đọc cho phép truy xuất
```

```
    // trạng thái sự kiện.
```

```
    public string From { get { return from; } }
```

```
public string Subject { get { return subject; } }
}
```

9.

Hiện thực mẫu Singleton

?

Bạn cần bảo đảm chỉ có một thể hiện của một kiểu tồn tại ở một thời điểm cho trước và thể hiện đó là khả-truy-xuất đối với tất cả các phần tử của ứng dụng.

✕

Hiện thực kiểu này theo mẫu *Singleton* như sau:

- Hiện thực một thành viên tĩnh `private` để giữ một tham chiếu đến thể hiện của kiểu.
- Hiện thực một thuộc tính tĩnh khả-truy-xuất-công-khai để cho phép truy xuất chỉ-đọc đến thể hiện.
- Hiện thực một phương thức khởi dựng `private` để mã lệnh không thể tạo thêm các thể hiện của kiểu.

Trong tất cả các mẫu được biết đến, có lẽ mẫu *Singleton* được biết đến nhiều nhất và thường được sử dụng nhất. Mục đích của mẫu *Singleton* là bảo đảm chỉ có một thể hiện của một kiểu tồn tại ở một thời điểm cho trước và cho phép truy xuất toàn cục đến các chức năng của thể hiện đó. Đoạn mã dưới đây trình bày một hiện thực của mẫu *Singleton* cho một lớp có tên là `SingletonExample`:

```
public class SingletonExample {

    // Thành viên tĩnh dùng để giữ một tham chiếu đến thể hiện singleton.
    private static SingletonExample instance;

    // Phương thức khởi dựng tĩnh dùng để tạo thể hiện singleton.
    // Một cách khác là sử dụng "Lazy Initialization" trong
    // thuộc tính Instance.
    static SingletonExample () {
        instance = new SingletonExample();
    }

    // Phương thức khởi dựng private dùng để ngăn mã lệnh tạo thêm
    // các thể hiện của kiểu singleton.
    private SingletonExample () {}

    // Thuộc tính public cho phép truy xuất đến thể hiện singleton.
    public static SingletonExample Instance {
```

```

        get { return instance; }
    }

    // Các phương thức public cung cấp chức năng của singleton.
    public void SomeMethod1 () { /*...*/ }
    public void SomeMethod2 () { /*...*/ }
}

```

Để gọi các chức năng của lớp `SingletonExample`, bạn có thể lấy về một tham chiếu đến *singleton* bằng thuộc tính `Instance` và rồi gọi các phương thức của nó. Bạn cũng có thể trực tiếp thực thi các thành viên của *singleton* thông qua thuộc tính `Instance`. Đoạn mã dưới đây trình bày cả hai cách này:

```

// Thu lấy tham chiếu đến singleton và gọi một phương thức của nó.
SingletonExample s = SingletonExample.Instance;
s.SomeMethod1();

// Thực thi chức năng của singleton mà không cần tham chiếu.
SingletonExample.Instance.SomeMethod2();

```

10.

Hiện thực mẫu Observer



Bạn cần hiện thực một cơ chế hiệu quả để một đối tượng (*subject*) báo với các đối tượng khác (*observer*) về những thay đổi trạng thái của nó.



Hiện thực mẫu *Observer* bằng các kiểu ủy nhiệm (đóng vai trò là các con trỏ hàm an-toàn-về-kiểu-dữ-liệu—*type-safe function pointer*) và các kiểu sự kiện.

Cách truyền thống khi hiện thực mẫu *Observer* là hiện thực hai giao diện: một để mô tả *observer* (`IObserver`) và một để mô tả *subject* (`ISubject`). Các đối tượng có hiện thực `IObserver` sẽ đăng ký với *subject*, cho biết chúng muốn được thông báo về các sự kiện quan trọng (như thay đổi trạng thái) tác động đến *subject*. *Subject* chịu trách nhiệm quản lý danh sách các *observer* đã đăng ký và thông báo với chúng khi đáp ứng các sự kiện tác động đến *subject*. *Subject* thường thông báo với *observer* bằng phương thức `Notify` (được khai báo trong giao diện `IObserver`). *Subject* có thể truyền dữ liệu cho *observer* trong phương thức `Notify`, hoặc *observer* có thể cần gọi một phương thức được khai báo trong giao diện `ISubject` để thu lấy thêm các chi tiết về sự kiện.

Mặc dù bạn có thể hiện thực mẫu *Observer* bằng *C#* theo cách vừa được mô tả, nhưng vì mẫu *Observer* quá phổ biến trong các giải pháp phần mềm hiện đại nên *C#* và *.NET Framework* cung cấp các kiểu sự kiện và ủy nhiệm để đơn giản hóa hiện thực của nó. Sử dụng sự kiện và ủy nhiệm nghĩa là bạn không cần khai báo giao diện `IObserver` và `ISubject`. Ngoài ra, bạn không cần hiện thực các logic cần thiết để quản lý và thông báo với các *observer* đã đăng ký—đây chính là phần dễ xảy ra lỗi nhất khi viết mã.

.NET Framework sử dụng một hiện thực cho mẫu *Observer* dựa-trên-sự-kiện và dựa-trên-ủy-nhiệm thường xuyên đến nỗi nó được đặt một cái tên: mẫu *Event*. File *ObserverExample.cs* chứa một hiện thực cho mẫu *Event*. Ví dụ này bao gồm:

- Lớp *Thermostat* (subject)—theo dõi nhiệt độ hiện tại và thông báo với observer khi có sự thay đổi nhiệt độ.
- Lớp *TemperatureChangeEventArgs*—là một hiện thực tùy biến của lớp *System.EventArgs*, được sử dụng để đóng gói dữ liệu cần phân bổ cho observer.
- Ủy nhiệm *TemperatureEventHandler*—định nghĩa chữ ký của phương thức mà tất cả các observer của đối tượng *Thermostat* phải hiện thực, và đối tượng *Thermostat* sẽ gọi phương thức này trong sự kiện thay đổi nhiệt độ.
- Lớp *TemperatureChangeObserver* và *TemperatureAverageObserver*—là các observer của lớp *Thermostat*.

Lớp *TemperatureChangeEventArgs* (được trình bày bên dưới) dẫn xuất từ lớp *System.EventArgs*. Lớp này sẽ chứa tất cả các dữ liệu cần thiết để subject truyền cho các observer khi nó thông báo với chúng về một sự kiện. Nếu không cần truyền dữ liệu, bạn không phải định nghĩa một lớp đối số mới (bạn chỉ cần truyền đối số *null* khi dựng nên sự kiện). Xem mục 16.8 để biết rõ hơn về cách hiện thực lớp đối số sự kiện tùy biến.

```
// Lớp đối số sự kiện chứa thông tin về sự kiện thay đổi nhiệt độ.
```

```
// Một thẻ hiện của lớp này sẽ được truyền cùng với mỗi sự kiện.
```

```
public class TemperatureChangeEventArgs : System.EventArgs {

    // Các thành viên dữ liệu chứa nhiệt độ cũ và mới.
    private readonly int oldTemperature, newTemperature;

    // Phương thức khởi dựng (nhận giá trị nhiệt độ cũ và mới).
    public TemperatureChangeEventArgs(int oldTemp, int newTemp) {

        oldTemperature = oldTemp;
        newTemperature = newTemp;
    }

    // Các thuộc tính dùng để truy xuất các giá trị nhiệt độ.
    public int OldTemperature { get { return oldTemperature; } }
    public int NewTemperature { get { return newTemperature; } }
}
```

Đoạn mã dưới đây trình bày cách khai báo ủy nhiệm *TemperatureEventHandler*. Dựa trên khai báo này, tất cả các observer phải hiện thực một phương thức (tên không quan trọng), trả về

void và nhận hai đối số: một đối tượng `Thermostat` và một đối tượng `TemperatureChangeEventArgs`. Trong quá trình thông báo, đối số `Thermostat` chỉ đến đối tượng `Thermostat` đã dựng nên sự kiện, và đối số `TemperatureChangeEventArgs` chứa các dữ liệu về nhiệt độ cũ và mới.

```
// Ủy nhiệm cho biết chữ ký mà tất cả các phương thức
// thụ lý sự kiện phải hiện thực.
public delegate void TemperatureEventHandler(Thermostat s,
    TemperatureChangeEventArgs e);
```

Lớp `Thermostat` là đối tượng bị giám sát trong mẫu *Observer (Event)* này. Theo lý thuyết, một thiết bị giám sát thiết lập nhiệt độ hiện tại bằng cách gọi thuộc tính `Temperature` trên một đối tượng `Thermostat`. Khi đó, đối tượng `Thermostat` dựng nên sự kiện `TemperatureChange` và gửi một đối tượng `TemperatureChangeEventArgs` đến mỗi observer. Dưới đây là mã lệnh cho lớp `Thermostat`:

```
// Lớp mô tả một bộ ổn nhiệt, là nguồn gốc của các sự kiện thay đổi
// nhiệt độ. Trong mẫu Observer, đối tượng Thermostat là subject.
public class Thermostat {

    // Trường private dùng để giữ nhiệt độ hiện tại.
    private int temperature = 0;

    // Sự kiện dùng để duy trì danh sách các ủy nhiệm observer và
    // dựng nên sự kiện thay đổi nhiệt độ.
    public event TemperatureEventHandler TemperatureChange;

    // Phương thức protected dùng để dựng nên sự kiện TemperatureChange.
    // Vì sự kiện chỉ có thể được phát sinh bên trong kiểu chứa nó nên
    // việc sử dụng phương thức protected cho phép các lớp dẫn xuất
    // có cách hành xử tùy biến và vẫn có thể dựng nên sự kiện
    // của lớp cơ sở.
    virtual protected void RaiseTemperatureEvent
        (TemperatureChangeEventArgs e) {

        if (TemperatureChange != null) {

            TemperatureChange(this, e);
        }
    }

    // Thuộc tính public dùng để lấy và thiết lập nhiệt độ hiện tại.
    // Phía "set" chịu trách nhiệm dựng nên sự kiện thay đổi nhiệt độ
```

```
// để thông báo với tất cả các observer về thay đổi này.
public int Temperature {

    get { return temperature; }

    set {
        // Tạo một đối tượng đối số sự kiện mới để chứa
        // nhiệt độ cũ và mới.
        TemperatureChangeEventArgs e =
            new TemperatureChangeEventArgs(temperature, value);

        // Cập nhật nhiệt độ hiện tại.
        temperature = value;

        // Dựng nên sự kiện thay đổi nhiệt độ.
        RaiseTemperatureEvent(e);
    }
}
}
```

Với mục đích minh họa mẫu *Observer*, ví dụ này có hai kiểu observer: *TemperatureAverageObserver* (hiển thị nhiệt độ trung bình mỗi khi một sự kiện thay đổi nhiệt độ xảy ra) và *TemperatureChangeObserver* (hiển thị thông tin về sự thay đổi mỗi khi một sự kiện thay đổi nhiệt độ xảy ra). Dưới đây là mã lệnh cho hai lớp này:

```
// Observer hiển thị thông tin về sự thay đổi nhiệt độ
// khi có một sự kiện thay đổi nhiệt độ xảy ra.
public class TemperatureChangeObserver {

    // Phương thức khởi dựng (nhận một tham chiếu đến đối tượng
    // Thermostat cần được TemperatureChangeObserver giám sát).
    public TemperatureChangeObserver(Thermostat t) {

        // Tạo một thẻ hiện ủy nhiệm TemperatureEventHandler và
        // đăng ký nó với Thermostat đã được chỉ định.
        t.TemperatureChange +=
            new TemperatureEventHandler(this.TemperatureChange);
    }
}
```

```

// Phương thức thụ lý sự kiện thay đổi nhiệt độ.
public void TemperatureChange(Thermostat sender,
    TemperatureChangeEventArgs temp) {

    System.Console.WriteLine
        ("ChangeObserver: Old={0}, New={1}, Change={2}",
        temp.OldTemperature, temp.NewTemperature,
        temp.NewTemperature - temp.OldTemperature);
}
}

// Observer hiển thị thông tin về nhiệt độ trung bình
// khi có một sự kiện thay đổi nhiệt độ xảy ra.
public class TemperatureAverageObserver {

    // Sum chứa tổng các giá trị nhiệt độ.
    // Count chứa số lần sự kiện thay đổi nhiệt độ xảy ra.
    private int sum = 0, count = 0;

    // Phương thức khởi dụng (nhận một tham chiếu đến đối tượng
    // Thermostat cần được TemperatureAverageObserver giám sát).
    public TemperatureAverageObserver (Thermostat t) {

        // Tạo một thẻ hiện ủy nhiệm TemperatureEventHandler và
        // đăng ký nó với Thermostat đã được chỉ định.
        t.TemperatureChange +=
            new TemperatureEventHandler(this.TemperatureChange);
    }

    // Phương thức thụ lý sự kiện thay đổi nhiệt độ.
    public void TemperatureChange(Thermostat sender,
        TemperatureChangeEventArgs temp) {

        count++;
        sum += temp.NewTemperature;

        System.Console.WriteLine

```

```

        ("AverageObserver: Average={0:F}",
        (double)sum/ (double)count);
    }
}

```

Lớp `Thermostat` định nghĩa một phương thức `Main` (được trình bày bên dưới) để chạy ví dụ này. Sau khi tạo một đối tượng `Thermostat` và hai đối tượng `observer`, phương thức `Main` sẽ nhắc bạn nhập vào một nhiệt độ. Mỗi khi bạn nhập một nhiệt độ mới, đối tượng `Thermostat` sẽ báo cho `observer` hiển thị thông tin ra cửa sổ *Console*.

```

public static void Main() {

    // Tạo một đối tượng Thermostat.
    Thermostat t = new Thermostat();

    // Tạo hai observer.
    new TemperatureChangeObserver(t);
    new TemperatureAverageObserver(t);

    // Lặp để lấy nhiệt độ từ người dùng. Bất cứ giá trị
    // nào không phải số nguyên sẽ khiến vòng lặp kết thúc.
    do {

        System.Console.WriteLine("\n\rEnter current temperature: ");

        try {
            // Chuyển đầu vào của người dùng thành một số
            // nguyên và sử dụng nó để thiết lập nhiệt độ
            // hiện tại của bộ ổn nhiệt.
            t.Temperature =
                System.Int32.Parse(System.Console.ReadLine());

        } catch (System.Exception) {
            // Sử dụng điều kiện ngoại lệ để kết thúc vòng lặp.
            System.Console.WriteLine("Terminating ObserverExample.");
            return;
        }
    } while (true);
}

```



```
}
```

Dưới đây là kết xuất khi chạy *ObserverExample.cs* (các giá trị in đậm là do bạn nhập vào):

```
Enter current temperature: 50
```

```
ChangeObserver: Old=0, New=50, Change=50
```

```
AverageObserver: Average=50.00
```

```
Enter current temperature: 20
```

```
ChangeObserver: Old=50, New=20, Change=-30
```

```
AverageObserver: Average=35.00
```

```
Enter current temperature: 40
```

```
ChangeObserver: Old=20, New=40, Change=20
```

```
AverageObserver: Average=36.67
```


17

SỰ HÒA HỢP VỚI MÔI TRƯỜNG WINDOWS

Microsoft .NET Framework được thiết kế sao cho có thể chạy trên nhiều hệ điều hành khác nhau, nâng cao tính khả chuyển của mã lệnh (*code mobility*) và đơn giản hóa việc tích hợp xuyên-nền (*cross-platform integration*).

Hiện tại, .NET Framework có thể chạy trên các hệ điều hành: *Microsoft Windows*, *FreeBSD*, *Linux*, và *Mac OS X*. Tuy nhiên, nhiều bản hiện thực vẫn chưa hoàn chỉnh hay chưa được chấp nhận rộng rãi. *Microsoft Windows* hiện là hệ điều hành mà .NET Framework được cài đặt nhiều nhất. Do đó, các mục trong chương này tập trung vào các tác vụ đặc trưng cho hệ điều hành *Windows*, bao gồm:

- Lấy các thông tin môi trường *Windows* (mục 17.1 và 17.2).
- Ghi vào nhật ký sự kiện *Windows* (mục 17.3).
- Truy xuất *Windows Registry* (mục 17.4).
- Tạo và cài đặt dịch vụ *Windows* (mục 17.5 và 17.6).
- Tạo shortcut trên *Desktop* hay trong *Start menu* của *Windows* (mục 17.7).

 Phần lớn các chức năng được thảo luận trong chương này được *CLR* bảo vệ bằng các quyền bảo mật truy xuất mã lệnh (*Code Access Security*). Xem chương 13 về bảo mật truy xuất mã lệnh, và xem tài liệu *.NET Framework SDK* về các quyền cần thiết để thực thi từng bộ phận.

1.

Truy xuất thông tin môi trường

? **Bạn cần truy xuất các thông tin về môi trường thực thi mà ứng dụng đang chạy trong đó.**

✂ **Sử dụng các thành viên của lớp `System.Environment`.**

Lớp `Environment` cung cấp một tập các thành viên tĩnh dùng để lấy (và trong một số trường hợp, để sửa đổi) thông tin về môi trường mà một ứng dụng đang chạy trong đó. Bảng 17.1 mô tả các thành viên thường dùng nhất.

Bảng 17.1 Các thành viên thường dùng của lớp `Environment`

Thành viên	Mô tả
Thuộc tính	
<code>CommandLine</code>	Lấy chuỗi chứa dòng lệnh thực thi ứng dụng hiện tại, gồm cả tên ứng dụng; xem chi tiết ở mục 1.5.
<code>CurrentDirectory</code>	Lấy và thiết lập chuỗi chứa thư mục hiện hành của ứng dụng. Ban đầu, thuộc tính này chứa tên của thư mục mà ứng dụng đã chạy trong đó.
<code>HasShutdownStarted</code>	Lấy một giá trị luận lý cho biết <i>CRL</i> đã bắt đầu tắt, hoặc miền ứng dụng đã bắt đầu giải phóng hay chưa.
<code>MachineName</code>	Lấy chuỗi chứa tên máy.

OSVersion	Lấy một đối tượng <code>System.OperatingSystem</code> chứa các thông tin về nền và phiên bản của hệ điều hành nằm dưới. Xem chi tiết bên dưới bảng.
SystemDirectory	Lấy chuỗi chứa đường dẫn đầy đủ của thư mục hệ thống.
TickCount	Lấy một giá trị kiểu <code>int</code> cho biết thời gian (tính bằng mili-giây) từ khi hệ thống khởi động.
UserDomainName	Lấy chuỗi chứa tên miền của người dùng hiện hành. Thuộc tính này sẽ giống với thuộc tính <code>MachineName</code> nếu đây là máy độc lập.
UserInteractive	Lấy một giá trị luận lý cho biết ứng dụng có đang chạy trong chế độ tương tác với người dùng hay không. Trả về <code>false</code> khi ứng dụng là một dịch vụ hoặc ứng dụng <i>Web</i> .
UserName	Lấy chuỗi chứa tên người dùng đã khởi chạy tiểu trình hiện hành.
Version	Lấy một đối tượng <code>System.Version</code> chứa thông tin về phiên bản của <i>CRL</i> .
Phương thức	
ExpandEnvironmentVariables	Thay tên của các biến môi trường trong một chuỗi bằng giá trị của biến; xem chi tiết ở mục 17.2.
GetCommandLineArgs	Trả về một mảng kiểu chuỗi chứa tất cả các phần tử dòng lệnh dùng để thực thi ứng dụng hiện tại, gồm cả tên ứng dụng; xem chi tiết ở mục 1.5.
GetEnvironmentVariable	Trả về chuỗi chứa giá trị của một biến môi trường; xem chi tiết ở mục 17.2.
GetEnvironmentVariables	Trả về một <code>System.Collections.IDictionary</code> chứa tất cả các biến môi trường và các giá trị của chúng; xem chi tiết ở mục 17.2
GetFolderPath	Trả về chuỗi chứa đường dẫn đến một thư mục hệ thống đặc biệt, được xác định bởi kiểu liệt kê <code>System.Environment.SpecialFolder</code> (bao gồm các thư mục cho <i>Internet cache</i> , <i>Cookie</i> , <i>History</i> , <i>Desktop</i> , và <i>Favourite</i> ; xem tài liệu <i>.NET Framework SDK</i> để có danh sách tất cả các giá trị này).

GetLogicalDrives

Trả về mảng kiểu chuỗi chứa tên của tất cả các ổ đĩa luận lý.

Đối tượng `OperatingSystem` (do `OSVersion` trả về) có hai thuộc tính: `Platform` và `Version`. Thuộc tính `Platform` trả về một giá trị thuộc kiểu liệt kê `System.PlatformID`—xác định nền hiện hành; các giá trị hợp lệ là `Win32NT`, `Win32S`, `Win32Windows`, và `WinCE`. Thuộc tính `Version` trả về một đối tượng `System.Version`—xác định phiên bản của hệ điều hành. Để xác định chính xác tên hệ điều hành, bạn phải sử dụng cả thông tin nền và phiên bản, bảng 17.2 dưới đây sẽ liệt kê một số thông tin chi tiết.

Bảng 17.2 Xác định hệ điều hành

PlatformID	Major Version	Minor Version	Hệ điều hành
Win32Windows	4	10	Windows 98
Win32Windows	4	90	Windows Me
Win32NT	4	0	Windows NT 4
Win32NT	5	0	Windows 2000
Win32NT	5	1	Windows XP
Win32NT	5	2	Windows Server 2003

Lớp `AccessEnvironmentExample` trong ví dụ dưới đây sử dụng lớp `Environment` để hiển thị thông tin về môi trường hiện hành.

```
using System;
```

```
public class AccessEnvironmentExample {

    public static void Main() {

        // Dòng lệnh.
        Console.WriteLine("Command line : " + Environment.CommandLine);

        // Thông tin về phiên bản hệ điều hành và môi trường thực thi.
        Console.WriteLine("OS PlatformID : " +
            Environment.OSVersion.Platform);
        Console.WriteLine("OS Major Version : " +
            Environment.OSVersion.Version.Major);
        Console.WriteLine("OS Minor Version : " +
            Environment.OSVersion.Version.Minor);
        Console.WriteLine("CLR Version : " + Environment.Version);

        // Thông tin về người dùng, máy, và miền.
```

```

Console.WriteLine("User Name : " + Environment.UserName);
Console.WriteLine("Domain Name : " + Environment.UserDomainName);
Console.WriteLine("Machine name : " + Environment.MachineName);

// Các thông tin môi trường khác.
Console.WriteLine("Is interactive ? : "
    + Environment.UserInteractive);
Console.WriteLine("Shutting down ? : "
    + Environment.HasShutdownStarted);
Console.WriteLine("Ticks since startup : "
    + Environment.TickCount);

// Hiển thị tên của tất cả các ổ đĩa luận lý.
foreach (string s in Environment.GetLogicalDrives()) {
    Console.WriteLine("Logical drive : " + s);
}

// Thông tin về các thư mục chuẩn.
Console.WriteLine("Current folder : "
    + Environment.CurrentDirectory);
Console.WriteLine("System folder : "
    + Environment.SystemDirectory);

// Liệt kê tất cả các thư mục đặc biệt.
foreach (Environment.SpecialFolder s in
    Enum.GetValues(typeof(Environment.SpecialFolder))) {

    Console.WriteLine("{0} folder : {1}",
        s, Environment.GetFolderPath(s));
}

// Nhấn Enter để kết thúc.
Console.WriteLine("Main method complete. Press Enter.");
Console.ReadLine();
}
}

```


2.

Lấy giá trị của một biến môi trường

?

Bạn cần lấy giá trị của một biến môi trường để sử dụng cho ứng dụng của bạn.

X

Sử dụng các phương thức `GetEnvironmentVariable`, `GetEnvironmentVariables`, và `ExpandEnvironmentVariables` của lớp `Environment`.

Phương thức `GetEnvironmentVariable` trả về chuỗi chứa giá trị của một biến môi trường, còn phương thức `GetEnvironmentVariables` trả về một `IDictionary` chứa tên và giá trị của tất cả các biến môi trường dưới dạng chuỗi. Phương thức `ExpandEnvironmentVariables` cung cấp một cơ chế đơn giản để thay tên biến môi trường bằng giá trị của nó bên trong một chuỗi, bằng cách đặt tên biến môi trường giữa dấu phần trăm (%).

Ví dụ sau minh họa cách sử dụng ba phương thức trên:

```
using System;
using System.Collections;

public class VariableExample {

    public static void Main () {

        // Lấy một biến môi trường thông qua tên.
        Console.WriteLine("Path = " +
            Environment.GetEnvironmentVariable("Path"));
        Console.WriteLine();

        // Thay tên biến môi trường bằng giá trị của nó.
        Console.WriteLine(Environment.ExpandEnvironmentVariables(
            "The Path on %computername% is %Path%"));
        Console.WriteLine();

        // Lấy tất cả các biến môi trường. Hiển thị giá trị
        // của các biến môi trường bắt đầu bằng ký tự 'P'.
        IDictionary vars = Environment.GetEnvironmentVariables();
        foreach (string s in vars.Keys) {
            if (s.ToUpper().StartsWith("P")) {
                Console.WriteLine(s + " = " + vars[s]);
            }
        }
        Console.WriteLine();
    }
}
```

```
// Nhấn Enter để kết thúc.
Console.WriteLine("Main method complete. Press Enter.");
Console.ReadLine();
}
}
```

3. Ghi một sự kiện vào nhật ký sự kiện Windows

? Bạn cần ghi một sự kiện vào nhật ký sự kiện *Windows*.

X Sử dụng các thành viên của lớp `System.Diagnostics.EventLog` để tạo một nhật ký (nếu cần), đăng ký một nguồn sự kiện (*event source*), và ghi sự kiện.

Bạn có thể ghi vào nhật ký sự kiện *Windows* bằng các phương thức tĩnh của lớp `EventLog`, hoặc có thể tạo một đối tượng `EventLog` và sử dụng các thành viên của nó. Dù chọn cách nào, trước khi ghi bạn cần phải quyết định sẽ sử dụng nhật ký nào và đăng ký một nguồn sự kiện cho nhật ký đó. Nguồn sự kiện đơn giản chỉ là một chuỗi (duy nhất) nhận diện ứng dụng của bạn. Một nguồn sự kiện chỉ có thể được đăng ký cho một nhật ký tại một thời điểm.

Theo mặc định, nhật ký sự kiện gồm ba loại: *Application*, *System*, và *Security*. Thông thường, bạn sẽ ghi vào nhật ký *Application*, nhưng cũng có thể ghi vào một nhật ký tùy biến. Bạn không cần phải trực tiếp tạo ra nhật ký tùy biến; khi bạn đăng ký một nguồn sự kiện cho một nhật ký, nếu nhật ký này không tồn tại, nó sẽ được tạo một cách tự động.

Một khi đã chọn nhật ký đích và đã đăng ký nguồn sự kiện tương ứng cho nó, bạn có thể bắt đầu ghi các entry nhật ký bằng phương thức `WriteEntry`. Phương thức này cung cấp các phiên bản nạp chồng cho phép bạn chỉ định một vài hoặc tất cả các giá trị sau:

- Chuỗi chứa nguồn sự kiện cho entry nhật ký (chỉ có ở các phương thức tĩnh).
- Chuỗi chứa thông điệp cho entry nhật ký.
- Giá trị thuộc kiểu liệt kê `System.Diagnostics.EventLogEntryType`, chỉ định kiểu của entry nhật ký. Các giá trị hợp lệ là `Error`, `FailureAlert`, `Information`, `SuccessAudit`, và `Warning`.
- Giá trị kiểu `int` chỉ định *ID* của entry nhật ký.
- Giá trị kiểu `short` chỉ định *category* của entry nhật ký.
- Mảng kiểu `byte` chứa dữ liệu thô tương ứng với entry nhật ký.

Lớp `EventLogExample` trong ví dụ dưới đây trình bày cách sử dụng các phương thức tĩnh của lớp `EventLog` để ghi một entry vào nhật ký sự kiện của máy cục bộ. Lớp `EventLog` cũng cung cấp các phương thức nạp chồng để ghi vào nhật ký sự kiện của các máy ở xa (xem tài liệu *.NET Framework SDK* để biết thêm chi tiết).

```
using System;
using System.Diagnostics;
```

```

public class EventLogExample {

    public static void Main () {

        // Nếu nguồn sự kiện không tồn tại, đăng ký nó với
        // nhật ký Application trên máy cục bộ.
        // Đăng ký một nguồn sự kiện đã tồn tại sẽ
        // sinh ra ngoại lệ System.ArgumentException.
        if (!EventLog.SourceExists("EventLogExample")) {

            EventLog.CreateEventSource("EventLogExample", "Application");
        }

        // Ghi một sự kiện vào nhật ký sự kiện.
        EventLog.WriteEntry(
            "EventLogExample",           // Nguồn sự kiện đã đăng ký
            "A simple test event.",      // Thông điệp cho sự kiện
            EventLogEntryType.Information, // Kiểu sự kiện
            1,                          // ID của sự kiện
            0,                          // Category của sự kiện
            new byte[] {10, 55, 200}    // Dữ liệu của sự kiện
        );

        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }
}

```

4.

Truy xuất Windows Registry



Bạn cần đọc thông tin từ *Registry* hoặc ghi thông tin vào *Registry*.



Sử dụng lớp `Microsoft.Win32.Registry` để lấy về đối tượng `Microsoft.Win32.RegistryKey` mô tả một khóa mức-cơ-sở. Sử dụng các thành viên của đối tượng `RegistryKey` để duyệt cây phân cấp; đọc, sửa, và tạo khóa và giá trị.

Bạn không thể truy xuất trực tiếp các khóa và các giá trị nằm trong *Registry*. Trước hết bạn phải thu lấy đối tượng `RegistryKey` mô tả một khóa mức-cơ-sở, sau đó duyệt qua cây phân cấp của đối tượng này để đến khóa cần tìm. Lớp `Registry` hiện thực bảy trường tĩnh, các

trường này đều trả về đối tượng `RegistryKey` mô tả khóa mức-cơ-sở. Bảng 17.3 trình bày các khóa mức-cơ-sở ứng với các trường này.

Bảng 17.3 Các trường tĩnh của lớp `Registry`

Trường	Ứng với
<code>ClassesRoot</code>	<code>HKEY_CLASSES_ROOT</code>
<code>CurrentConfig</code>	<code>HKEY_CURRENT_CONFIG</code>
<code>CurrentUser</code>	<code>HKEY_CURRENT_USER</code>
<code>DynData</code>	<code>HKEY_DYN_DATA</code>
<code>LocalMachine</code>	<code>HKEY_LOCAL_MACHINE</code>
<code>PerformanceData</code>	<code>HKEY_PERFORMANCE_DATA</code>
<code>Users</code>	<code>HKEY_USERS</code>



Phương thức tĩnh `RegistryKey.OpenRemoteBaseKey` cho phép bạn mở một khóa mức-cơ-sở trên một máy ở xa (xem chi tiết trong tài liệu *.NET Framework SDK*).

Một khi đã có đối tượng `RegistryKey` mô tả khóa mức-cơ-sở, bạn phải duyệt qua cây phân cấp để đến khóa cần tìm. Để hỗ trợ việc duyệt cây, lớp `RegistryKey` cung cấp các thành viên dưới đây:

- Thuộc tính `SubKeyCount`—Trả về số khóa con.
- Phương thức `GetSubKeyNames`—Trả về một mảng kiểu chuỗi chứa tên của tất cả các khóa con.
- Phương thức `OpenSubKey`—Trả về tham chiếu đến một khóa con. Phương thức này có hai phiên bản nạp chồng: phương thức thứ nhất mở khóa trong chế độ chỉ-đọc; phương thức thứ hai nhận một đối số kiểu `bool`, nếu là `true` thì cho phép ghi.

Một khi đã có đối tượng `RegistryKey` mô tả khóa cần tìm, bạn có thể tạo, đọc, cập nhật, hoặc xóa các khóa con và các giá trị bằng các phương thức trong bảng 17.4. Các phương thức sửa đổi nội dung của khóa đòi hỏi bạn phải có đối tượng `RegistryKey` cho phép ghi.

Bảng 17.4 Các phương thức của `RegistryKey` dùng để tạo, đọc, cập nhật, và xóa các khóa và các giá trị Registry

Phương thức	Mô tả
<code>CreateSubKey</code>	Tạo một khóa mới với tên được chỉ định và trả về đối tượng <code>RegistryKey</code> cho phép ghi. Nếu khóa đã tồn tại, phương thức này sẽ trả về một tham chiếu cho phép ghi đến khóa đã tồn tại.
<code>DeleteSubKey</code>	Xóa khóa với tên được chỉ định, khóa này phải không có khóa con (nhưng có thể có giá trị); nếu không, ngoại lệ <code>System.InvalidOperationException</code> sẽ bị ném.
<code>DeleteSubKeyTree</code>	Xóa khóa với tên được chỉ định cùng với tất cả các khóa con của nó.

DeleteValue	Xóa một giá trị với tên được chỉ định khỏi khóa hiện tại.
GetValue	Trả về giá trị với tên được chỉ định từ khóa hiện tại. Giá trị trả về là một đối tượng, bạn phải ép nó về kiểu thích hợp. Dạng đơn giản nhất của GetValue trả về null nếu giá trị không tồn tại. Ngoài ra còn có một phiên bản nạp chồng cho phép chỉ định giá trị trả về mặc định (thay cho null) nếu giá trị không tồn tại.
GetValueNames	Trả về mảng kiểu chuỗi chứa tên của tất cả các giá trị trong khóa hiện tại.
SetValue	Tạo (hoặc cập nhật) giá trị với tên được chỉ định. Bạn không thể chỉ định kiểu dữ liệu Registry dùng để lưu trữ dữ liệu; SetValue sẽ tự động chọn kiểu dựa trên kiểu dữ liệu được lưu trữ.

Lớp RegistryKey có hiện thực giao diện IDisposable; bạn nên gọi phương thức IDisposable.Dispose để giải phóng các tài nguyên của hệ điều hành khi đã hoàn tất với đối tượng RegistryKey.

Lớp RegistryExample trong ví dụ sau nhận một đối số dòng lệnh và duyệt đệ quy cây có gốc là CurrentUser để tìm các khóa có tên trùng với đối số dòng lệnh. Khi tìm được một khóa, RegistryExample sẽ hiển thị tất cả các giá trị kiểu chuỗi nằm trong khóa này. Lớp RegistryExample cũng giữ một biến đếm trong khóa HKEY_CURRENT_USER\RegistryExample.

```
using System;
using Microsoft.Win32;

public class RegistryExample {

    public static void Main(String[] args) {

        if (args.Length > 0) {

            // Mở khóa cơ sở CurrentUser.
            using(RegistryKey root = Registry.CurrentUser) {

                // Cập nhật biến đếm.
                UpdateUsageCounter(root);

                // Duyệt đệ quy để tìm khóa với tên cho trước.
                SearchSubKeys(root, args[0]);

            }

        }

    }
}
```

```

        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }

    public static void UpdateUsageCounter(RegistryKey root) {

        // Tạo một khóa để lưu trữ biến đếm,
        // hoặc lấy tham chiếu đến khóa đã có.
        RegistryKey countKey = root.CreateSubKey("RegistryExample");

        // Đọc giá trị của biến đếm hiện tại, và chỉ định
        // giá trị mặc định là 0. Ép đối tượng về kiểu Int32,
        // và gán vào một giá trị int.
        int count = (Int32)countKey.GetValue("UsageCount", 0);

        // Ghi biến đếm đã được cập nhật trở lại Registry,
        // hoặc tạo một giá trị mới nếu nó chưa tồn tại.
        countKey.SetValue("UsageCount", ++count);
    }

    public static void SearchSubKeys(RegistryKey root,
        String searchKey) {

        // Lặp qua tất cả các khóa con trong khóa hiện tại.
        foreach (string keyname in root.GetSubKeyNames()) {

            try {
                using (RegistryKey key = root.OpenSubKey(keyname)) {
                    if (keyname == searchKey) PrintKeyValues(key);
                    SearchSubKeys(key, searchKey);
                }
            } catch (System.Security.SecurityException) {
                // Bỏ qua SecurityException với chủ định của ví dụ này.
                // Một số khóa con của HKEY_CURRENT_USER được bảo mật
            }
        }
    }
}

```

```

        // và sẽ ném SecurityException khi được mở.
    }

}

}

public static void PrintKeyValues(RegistryKey key) {

    // Hiển thị tên của khóa được tìm thấy,
    // và số lượng giá trị của nó.
    Console.WriteLine("Registry key found : {0} contains {1} values",
        key.Name, key.ValueCount);

    // Hiển thị các giá trị này.
    foreach (string valuenam in key.GetValueNames()) {
        if (key.GetValue(valuenam) is String) {
            Console.WriteLine("  Value : {0} = {1}",
                valuenam, key.GetValue(valuenam));
        }
    }
}
}

```

Khi được thực thi trên máy chạy *Windows XP* với dòng lệnh `RegistryExample Environment`, ví dụ này sẽ cho kết xuất như sau:

```

Registry key found : HKEY_CURRENT_USER\Environment contains 4 values
Value : TEMP =
    C:\Documents and Settings\nnbphuong81\Local Settings\Temp
Value : TMP =
    C:\Documents and Settings\nnbphuong81\Local Settings\Temp
Value : LIB =
    C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Lib\
Value : INCLUDE =
    C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\include\

```

5.

Tạo một dịch vụ Windows

?

Bạn cần tạo một ứng dụng đóng vai trò là một dịch vụ *Windows*.

✕

Tạo một lớp thừa kế từ lớp `System.ServiceProcess.ServiceBase`. Sử dụng các thuộc tính thừa kế để điều khiển hành vi của dịch vụ, và chép đề các phương

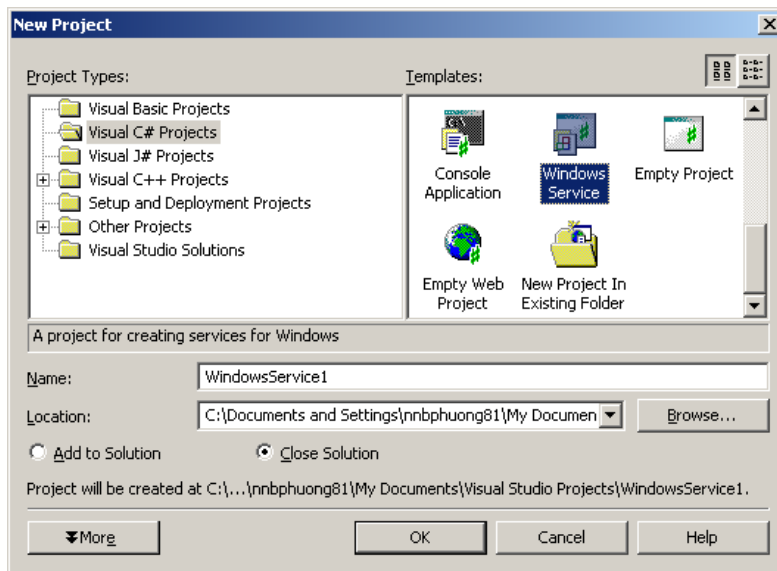
thức thừa kế để hiện thực các chức năng cần thiết. Hiện thực phương thức `Main`, trong đó tạo một thể hiện của lớp dịch vụ và truyền nó cho phương thức tĩnh `ServiceBase.Run`.

Nếu đang sử dụng *Microsoft Visual C#.NET*, bạn có thể dùng mẫu dự án *Windows Service* để tạo một dịch vụ *Windows*. Mẫu này cung cấp sẵn các mã lệnh cơ bản cần cho một lớp dịch vụ, và bạn có thể hiện thực thêm các chức năng tùy biến.

Để tạo một dịch vụ *Windows* bằng tay, bạn phải hiện thực một lớp dẫn xuất từ `ServiceBase`. Lớp `ServiceBase` cung cấp các chức năng cơ bản cho phép *Windows Service Control Manager (SCM)* cấu hình dịch vụ, thi hành dịch vụ dưới nền, và điều khiển thời gian sống của dịch vụ. *SCM* cũng điều khiển việc các ứng dụng khác có thể điều khiển dịch vụ như thế nào.



Lớp `ServiceBase` được định nghĩa trong `System.Serviceprocess`, do đó bạn phải thêm một tham chiếu đến assembly này khi xây dựng lớp dịch vụ.



Hình 17.1 Mẫu dự án Windows Service

Để điều khiển dịch vụ của bạn, *SDM* sử dụng bảy phương thức `protected` thừa kế từ lớp `ServiceBase` (xem bảng 17.5). Bạn cần chép đề các phương thức này để hiện thực các chức năng và cách thức hoạt động của dịch vụ. Không phải tất cả dịch vụ đều hỗ trợ tất cả các thông điệp điều khiển. Các thuộc tính thừa kế từ lớp `ServiceBase` sẽ báo với *SCM* rằng dịch vụ của bạn hỗ trợ các thông điệp điều khiển nào; thuộc tính điều khiển mỗi kiểu thông điệp được ghi rõ trong bảng 17.5.

Bảng 17.5 Các phương thức dùng để điều khiển sự hoạt động của một dịch vụ

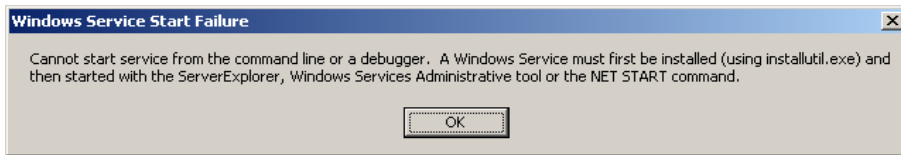
Phương thức	Mô tả
-------------	-------

OnStart	Tất cả các dịch vụ đều phải hỗ trợ phương thức <code>OnStart</code> , <i>SCM</i> gọi phương thức này để khởi động dịch vụ. <i>SCM</i> truyền cho dịch vụ một mảng kiểu chuỗi chứa các đối số cần thiết. Nếu <code>OnStart</code> không trả về trong 30 giây thì <i>SCM</i> sẽ không chạy dịch vụ.
OnStop	Được <i>SCM</i> gọi để dừng một dịch vụ— <i>SCM</i> chỉ gọi <code>OnStop</code> nếu thuộc tính <code>CanStop</code> là <code>true</code> .
OnPause	Được <i>SCM</i> gọi để tạm dừng một dịch vụ— <i>SCM</i> chỉ gọi <code>OnPause</code> nếu thuộc tính <code>CanPauseAndContinue</code> là <code>true</code> .
OnContinue	Được <i>SCM</i> gọi để tiếp tục một dịch vụ bị tạm dừng— <i>SCM</i> chỉ gọi <code>OnContinue</code> nếu thuộc tính <code>CanPauseAndContinue</code> là <code>true</code> .
OnShutdown	Được <i>SCM</i> gọi khi hệ thống đang tắt— <i>SCM</i> chỉ gọi <code>OnShutdown</code> nếu thuộc tính <code>CanShutdown</code> là <code>true</code> .
OnPowerEvent	Được <i>SCM</i> gọi khi trạng thái nguồn mức-hệ-thống thay đổi, chẳng hạn một laptop chuyển sang chế độ suspend. <i>SCM</i> chỉ gọi <code>OnPowerEvent</code> nếu thuộc tính <code>CanHandlePowerEvent</code> là <code>true</code> .
OnCustomCommand	Cho phép mở rộng cơ chế điều khiển dịch vụ với các thông điệp điều khiển tùy biến; xem chi tiết trong tài liệu <i>.NET Framework SDK</i> .

Như được đề cập trong bảng 17.5, phương thức `OnStart` phải trả về trong vòng 30 giây, do đó bạn không nên sử dụng `OnStart` để thực hiện các thao tác khởi động tốn nhiều thời gian. Một lớp dịch vụ nên hiện thực một phương thức khởi động để thực hiện các thao tác khởi động, bao gồm việc cấu hình các thuộc tính thừa kế từ lớp `ServiceBase`. Ngoài các thuộc tính khai báo các thông điệp điều khiển nào được dịch vụ hỗ trợ, lớp `ServiceBase` còn hiện thực ba thuộc tính quan trọng khác:

- `ServiceName`—Là tên được *SCM* sử dụng để nhận dạng dịch vụ, và phải được thiết lập trước khi dịch vụ chạy.
- `AutoLog`—Điều khiển việc dịch vụ có tự động ghi vào nhật ký sự kiện hay không khi nhận thông điệp điều khiển `OnStart`, `OnStop`, `OnPause`, và `OnContinue`.
- `EventLog`—Trả về một đối tượng `EventLog` được cấu hình trước với tên nguồn sự kiện (*event source*) trùng với thuộc tính `ServiceName` được đăng ký với nhật ký *Application* (xem mục 17.3 để có thêm thông tin về lớp `EventLog`).

Bước cuối cùng trong việc tạo một dịch vụ là hiện thực phương thức tĩnh `Main`. Phương thức này phải tạo một thể hiện của lớp dịch vụ và truyền nó cho phương thức tĩnh `ServiceBase.Run`. Nếu muốn chạy nhiều dịch vụ trong một tiến trình, bạn phải tạo một mảng các đối tượng `ServiceBase` và truyền nó cho phương thức `ServiceBase.Run`. Mặc dù các lớp dịch vụ đều có phương thức `Main` nhưng bạn không thể thực thi mã lệnh dịch vụ một cách trực tiếp; bạn sẽ nhận được hộp thông báo như hình 17.2 nếu trực tiếp chạy một lớp dịch vụ. Mục 17.6 sẽ trình bày cách cài đặt dịch vụ trước khi thực thi.



Hình 17.2 Hộp thông báo Windows Service Start Failure

Lớp `ServiceExample` trong ví dụ dưới đây sử dụng một `System.Timers.Timer` để ghi một entry vào nhật ký sự kiện *Windows* theo định kỳ.

```
using System;
using System.Timers;
using System.ServiceProcess;

public class ServiceExample : ServiceBase {

    // Timer điều khiển khi nào ServiceExample ghi vào nhật ký sự kiện.
    private System.Timers.Timer timer;

    public ServiceExample() {

        // Thiết lập thuộc tính ServiceBase.ServiceName.
        ServiceName = "ServiceExample";

        // Cấu hình các thông điệp điều khiển.
        CanStop = true;
        CanPauseAndContinue = true;

        // Cấu hình việc ghi các sự kiện quan trọng vào
        // nhật ký Application.
        AutoLog = true;
    }

    // Phương thức sẽ được thực thi khi Timer hết
    // hiệu lực – ghi một entry vào nhật ký Application.
    private void WriteLogEntry(object sender, ElapsedEventArgs e) {

        // Sử dụng đối tượng EventLog để ghi vào nhật ký sự kiện.
```

```

        EventLog.WriteEntry("ServiceExample active : " + e.SignalTime);
    }

    protected override void OnStart(string[] args) {

        // Lấy chu kỳ ghi sự kiện từ đối số thứ nhất.
        // Mặc định là 5000 mili-giây,
        // và tối thiểu là 1000 mili-giây.
        double interval;

        try {
            interval = System.Double.Parse(args[0]);
            interval = Math.Max(1000, interval);
        } catch {
            interval = 5000;
        }

        EventLog.WriteEntry(String.Format("ServiceExample starting. " +
            "Writing log entries every {0} milliseconds...", interval));

        // Tạo, cấu hình, và khởi động một System.Timers.Timer
        // để gọi phương thức WriteLogEntry theo định kỳ.
        // Các phương thức Start và Stop của lớp System.Timers.Timer
        // giúp thực hiện các chức năng khởi động, tạm dừng, tiếp tục,
        // và dừng dịch vụ.
        timer = new Timer();
        timer.Interval = interval;
        timer.AutoReset = true;
        timer.Elapsed += new ElapsedEventHandler(WriteLogEntry);
        timer.Start();
    }

    protected override void OnStop() {

        EventLog.WriteEntry("ServiceExample stopping...");
        timer.Stop();

        // Giải phóng tài nguyên hệ thống do Timer sử dụng.
    }

```

```

        timer.Dispose();
        timer = null;
    }

    protected override void OnPause() {

        if (timer != null) {
            EventLog.WriteEntry("ServiceExample pausing...");
            timer.Stop();
        }
    }

    protected override void OnContinue() {

        if (timer != null) {
            EventLog.WriteEntry("ServiceExample resuming...");
            timer.Start();
        }
    }

    public static void Main() {

        // Tạo một thẻ hiện của lớp ServiceExample để ghi một
        // entry vào nhật ký Application. Truyền đối tượng này
        // cho phương thức tĩnh ServiceBase.Run.
        ServiceBase.Run(new ServiceExample());
    }
}

```

6.

Tạo một bộ cài đặt dịch vụ Windows

?

Bạn đã tạo một ứng dụng dịch vụ *Windows* và cần cài đặt nó.

✕

Thừa kế lớp `System.Configuration.Install.Installer` để tạo một lớp cài đặt gồm những thông tin cần thiết để cài đặt và cấu hình lớp dịch vụ của bạn. Sử dụng công cụ *Installutil.exe* để thực hiện việc cài đặt.

Như đã đề cập trong mục 17.5, bạn không thể chạy các lớp dịch vụ một cách trực tiếp. Vì dịch vụ được tích hợp mức cao với hệ điều hành *Windows* và thông tin được giữ trong *Registry* nên dịch vụ phải được cài đặt trước khi chạy.

Nếu đang sử dụng *Microsoft Visual Studio .NET*, bạn có thể tạo một bộ cài đặt cho dịch vụ một cách tự động bằng cách nhấp phải vào khung thiết kế của lớp dịch vụ và chọn *Add Installer* từ menu ngữ cảnh. Bộ cài đặt này có thể được gọi bởi các dự án triển khai hoặc công cụ *Installutil.exe* để cài đặt dịch vụ.

Bạn cũng có thể tự tạo một bộ cài đặt cho dịch vụ *Windows* theo các bước sau:

1. Tạo một lớp thừa kế từ lớp *Installer*.
2. Áp dụng đặc tính `System.ComponentModel.RunInstallerAttribute(true)` cho lớp cài đặt.
3. Trong phương thức khởi dựng của lớp cài đặt, tạo một thể hiện của lớp `System.ServiceProcess.ServiceProcessInstaller`. Thiết lập các thuộc tính `Account`, `UserName`, và `Password` của đối tượng `ServiceProcessInstaller` để cấu hình tài khoản mà dịch vụ sẽ chạy.
4. Cũng trong phương thức khởi dựng của lớp cài đặt, tạo một thể hiện của lớp `System.ServiceProcess.ServiceInstaller` cho mỗi dịch vụ cần cài đặt. Sử dụng các thuộc tính của đối tượng `ServiceInstaller` để cấu hình các thông tin về mỗi dịch vụ, bao gồm:
 - **ServiceName**—Chỉ định tên mà *Windows* sử dụng để nhận dạng dịch vụ. Tên này phải trùng với giá trị được gán cho thuộc tính `ServiceBase.ServiceName`.
 - **DisplayName**—Chỉ định tên thân thiện cho dịch vụ.
 - **StartType**—Sử dụng các giá trị thuộc kiểu liệt kê `System.ServiceProcess.ServiceStartMode` để điều khiển việc dịch vụ được khởi động tự động hay bằng tay, hay bị vô hiệu.
 - **ServiceDependsUpon**—lấy một mảng kiểu chuỗi chứa tên các dịch vụ phải được chạy trước khi dịch vụ hiện hành chạy.
5. Sử dụng thuộc tính `Installers` thừa kế từ lớp cơ sở *Installer* để lấy một đối tượng `System.Configuration.Install.InstallerCollection`. Thêm các đối tượng `ServiceProcessInstaller` và tất cả các đối tượng `ServiceInstaller` vào tập hợp này.

Lớp `ServiceInstallerExample` dưới đây là một bộ cài đặt cho lớp `ServiceExample` trong mục 17.5. Dự án mẫu cho mục này chứa cả hai lớp `ServiceExample` và `ServiceInstallerExample`, và tạo ra file thực thi *ServiceInstallerExample.exe*.

```
using System.ServiceProcess;

using System.Configuration.Install;

using System.ComponentModel;

[RunInstaller(true)]

public class ServiceInstallerExample : Installer {
```

```

public ServiceInstallerExample() {

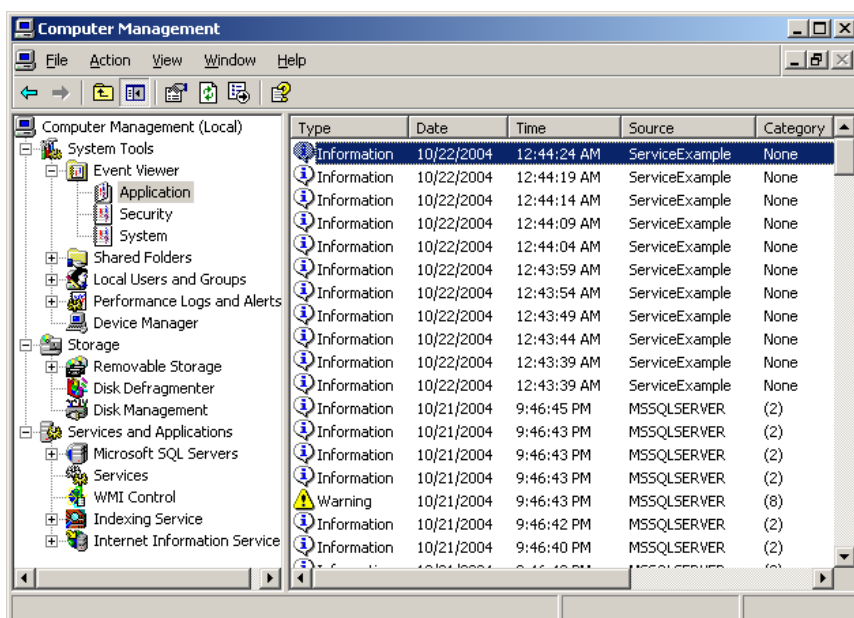
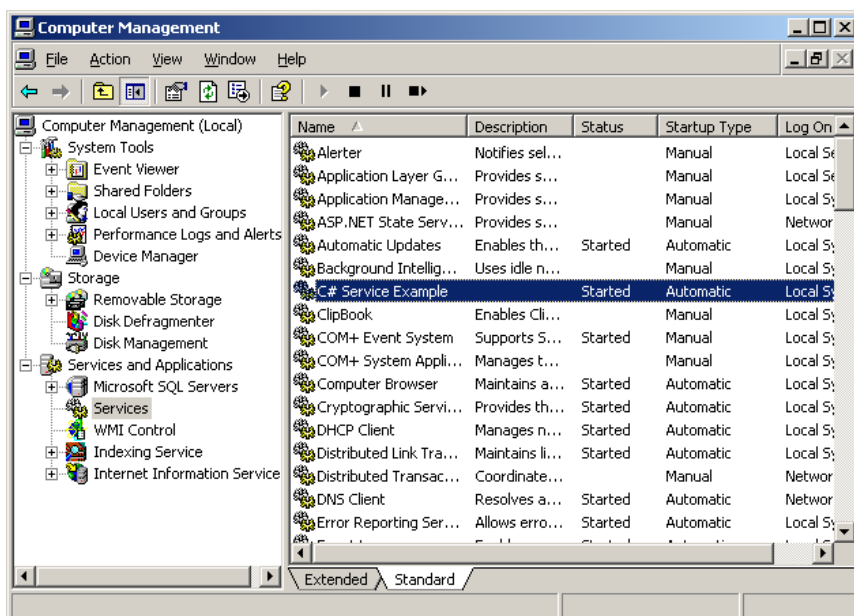
    // Tạo và cấu hình đối tượng ServiceProcessInstaller.
    ServiceProcessInstaller ServiceExampleProcess =
        new ServiceProcessInstaller();
    ServiceExampleProcess.Account = ServiceAccount.LocalSystem;

    // Tạo và cấu hình đối tượng ServiceInstaller.
    ServiceInstaller ServiceExampleInstaller =
        new ServiceInstaller();
    ServiceExampleInstaller.DisplayName =
        "C# Service Example";
    ServiceExampleInstaller.ServiceName = "ServiceExample";
    ServiceExampleInstaller.StartType = ServiceStartModeAutomatic;

    // Thêm đối tượng ServiceProcessInstaller và ServiceInstaller
    // vào tập hợp Installers (thừa kế từ lớp cơ sở Installer).
    Installers.Add(ServiceExampleInstaller);
    Installers.Add(ServiceExampleProcess);
}
}

```

Để cài đặt `ServiceExample`, bạn cần tạo dự án, chuyển đến thư mục chứa file `ServiceInstallerExample.exe` (mặc định là `bin\debug`), rồi thực thi lệnh `Installutil ServiceInstallerExample.exe`. Sau đó, bạn có thể sử dụng *Windows Computer Management* để xem và điều khiển dịch vụ. Mặc dù `StartType` được chỉ định là `Automatic`, dịch vụ này vẫn không được khởi động sau khi cài đặt. Bạn phải khởi động dịch vụ bằng tay (hoặc khởi động lại máy) trước khi dịch vụ ghi các entry vào nhật ký sự kiện. Một khi dịch vụ đã chạy, bạn có thể xem các entry mà nó đã ghi vào nhật ký *Application* bằng *Event Viewer*. Để gỡ bỏ `ServiceExample`, bạn hãy thực thi lệnh `Installutil /u ServiceInstallerExample.exe`.



Hình 17.3 Windows Computer Management

7.

Tạo shortcut trên Desktop hay trong Start menu

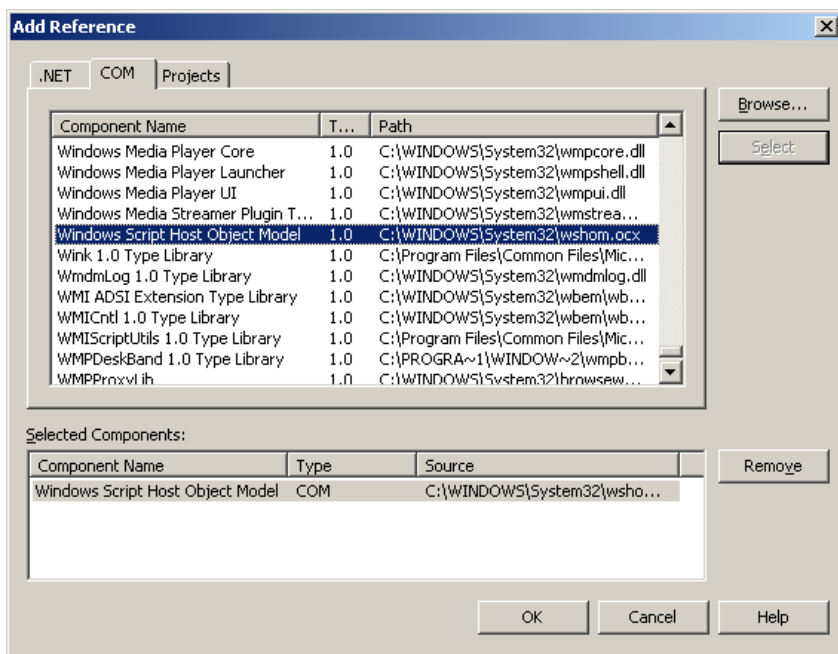
?

Bạn cần tạo một shortcut trên *Desktop* hay trong *Start menu* của người dùng.



Sử dụng *COM Interop* để truy xuất các chức năng của *Windows Script Host*. Tạo và cấu hình một thẻ hiện *IWshShortcut* tương ứng với shortcut. Thư mục chứa shortcut sẽ xác định shortcut xuất hiện trên *Desktop* hay trong *Start menu*.

Thư viện lớp *.NET Framework* không có chức năng tạo shortcut trên *Desktop* hay trong *Start menu*; tuy nhiên, việc này có thể được thực hiện dễ dàng bằng thành phần *Windows Script Host* (được truy xuất thông qua *COM Interop*). Cách tạo *Interop Assembly* để truy xuất một thành phần *COM* đã được trình bày trong mục 15.6. Nếu đang sử dụng *Visual Studio .NET*, bạn hãy thêm một tham chiếu đến *Windows Script Host Object Model* (được liệt kê trong thẻ *COM* của hộp thoại *Add Reference*). Nếu không có *Visual Studio .NET*, bạn hãy sử dụng công cụ *Type Library Importer (Tlbimp.exe)* để tạo một *Interop Assembly* cho file *wshom.ocx* (file này thường nằm trong thư mục *Windows\System32*). Bạn có thể lấy phiên bản mới nhất của *Windows Script Host* tại [<http://msdn.microsoft.com/scripting>].



Hình 17.4 Chọn Windows Script Host Object Model trong hộp thoại Add Reference

Một khi đã tạo và nhập *Interop Assembly* vào dự án, bạn hãy thực hiện các bước sau:

1. Tạo một đối tượng *WshShell* để truy xuất vào *Windows shell*.
2. Sử dụng thuộc tính *SpecialFolders* của đối tượng *WshShell* để xác định đường dẫn đến thư mục sẽ chứa shortcut. Tên của thư mục đóng vai trò là index đối với thuộc tính *SpecialFolders*. Ví dụ, chỉ định giá trị *Desktop* để tạo shortcut trên *Desktop*, và chỉ định giá trị *StartMenu* để tạo shortcut trong *Start menu*. Thuộc tính *SpecialFolders* còn

có thể được sử dụng để lấy đường dẫn đến mọi thư mục đặc biệt của hệ thống; các giá trị thường dùng khác là `AllUsersDesktop` và `AllUsersStartMenu`.

3. Gọi phương thức `CreateShortcut` của đối tượng `WshShell`, và truyền đường dẫn đầy đủ của file shortcut cần tạo (có phần mở rộng là `.lnk`). Phương thức này sẽ trả về một thể hiện `IWshShortcut`.
4. Sử dụng các thuộc tính của thể hiện `IWshShortcut` để cấu hình shortcut. Ví dụ, bạn có thể cấu hình file thực thi được shortcut tham chiếu, biểu tượng dùng cho shortcut, lời mô tả, và phím nóng.
5. Gọi phương thức `Save` của thể hiện `IWshShortcut` để ghi shortcut vào đĩa. Shortcut sẽ nằm trên *Desktop* hay trong *Start menu* (hay một nơi nào khác) tùy vào đường dẫn được chỉ định khi tạo thể hiện `IWshShortcut`.

Lớp `ShortcutExample` trong ví dụ dưới đây tạo shortcut cho *Notepad.exe* trên *Desktop* và trong *Start menu* của người dùng hiện hành. `ShortcutExample` tạo hai shortcut này bằng phương thức `CreateShortcut` và chỉ định hai thư mục khác nhau cho file shortcut. Cách này giúp bạn tạo file shortcut trong bất kỳ thư mục đặc biệt nào được trả về từ thuộc tính `WshShell.SpecialFolders`.

```
using System;
using IWshRuntimeLibrary;

public class ShortcutExample {

    public static void Main() {

        // Tạo shortcut cho Notepad trên Desktop.
        CreateShortcut("Desktop");

        // Tạo shortcut cho Notepad trong Start menu.
        CreateShortcut("StartMenu");

        // Nhấn Enter để kết thúc.
        Console.WriteLine("Main method complete. Press Enter.");
        Console.ReadLine();
    }

    public static void CreateShortcut(string destination) {

        // Tạo một đối tượng WshShell để truy xuất
        // các chức năng của Windows shell.
        WshShell wshShell = new WshShell();
```

```
// Lấy đường dẫn sẽ chứa file Notepad.lnk. Bạn có thể
// sử dụng phương thức System.Environment.GetFolderPath
// để lấy đường dẫn, nhưng sử dụng WshShell.SpecialFolders
// sẽ truy xuất được nhiều thư mục hơn. Bạn cần tạo một
// đối tượng tạm tham chiếu đến chuỗi destination
// để thỏa mãn yêu cầu của phương thức Item.
object destFolder = (object)destination;
string fileName =
    (string)wshShell.SpecialFolders.Item(ref destFolder)
    + @"\"Notepad.lnk";

// Tạo đối tượng shortcut. Tuy nhiên, chẳng có gì được
// tạo ra trong thư mục cho đến khi shortcut được lưu.
IWshShortcut shortcut =
    (IWshShortcut)wshShell.CreateShortcut(fileName);

// Cấu hình đường dẫn file thực thi.
// Sử dụng lớp Environment cho đơn giản.
shortcut.TargetPath =
    Environment.GetFolderPath(Environment.SpecialFolder.System)
    + @"\"notepad.exe";

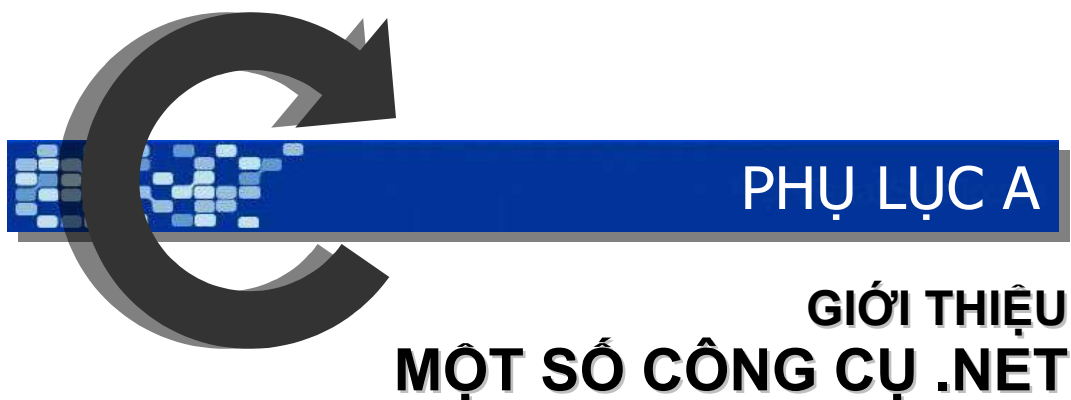
// Thiết lập thư mục làm việc là Personal (My Documents).
shortcut.WorkingDirectory =
    Environment.GetFolderPath(Environment.SpecialFolder.Personal);

// Cung cấp lời mô tả cho shortcut.
shortcut.Description = "Notepad Text Editor";

// Gán phím nóng cho shortcut.
shortcut.Hotkey = "CTRL+ALT+N";

// Cấu hình của sổ Notepad luôn phóng to khi khởi động.
shortcut.WindowStyle = 3;
```

```
// Cấu hình shortcut hiển thị icon đầu tiên trong notepad.exe.  
shortcut.IconLocation = "notepad.exe, 0";  
  
// Lưu file shortcut.  
shortcut.Save();  
}  
}
```



PHỤ LỤC A

GIỚI THIỆU MỘT SỐ CÔNG CỤ .NET

Phần phụ lục này giới thiệu một số công cụ nhỏ (hầu hết là miễn phí) nhưng rất tốt cho các nhà phát triển .NET, trong đó có những công cụ giúp phát triển ứng dụng nhanh hơn và có những công cụ có thể làm thay đổi cách thức viết mã lệnh của bạn.

- Biên dịch các đoạn mã ngắn với *Snippet Compiler*
- Xây dựng biểu thức chính quy với *Regulator*
- Sinh mã với *CodeSmith*
- Viết kiểm thử đơn vị với *NUnit*
- Kiểm soát mã lệnh với *FxCop*
- Khảo sát assembly với *.NET Reflector*
- Lập tài liệu mã lệnh với *NDoc*
- Tạo dựng giải pháp với *NAnt*
- Các công cụ chuyển đổi: *ASP.NET Version Switcher*, *Visual Studio .NET Project Converter*, *VB.NET to C# Converter*, và *Convert C# to VB.NET*
- Xây dựng website quản trị cơ sở dữ liệu với *ASP.NET Maker 1.1*

A.1 Biên dịch các đoạn mã ngắn với *Snippet Compiler*

Snippet Compiler là một ứng dụng nhỏ dùng để viết, biên dịch, và chạy mã lệnh. Công cụ này hữu ích khi bạn có những đoạn mã ngắn và bạn không muốn phải tạo toàn bộ dự án *Visual Studio .NET* (cùng với các file đi kèm) cho chúng.

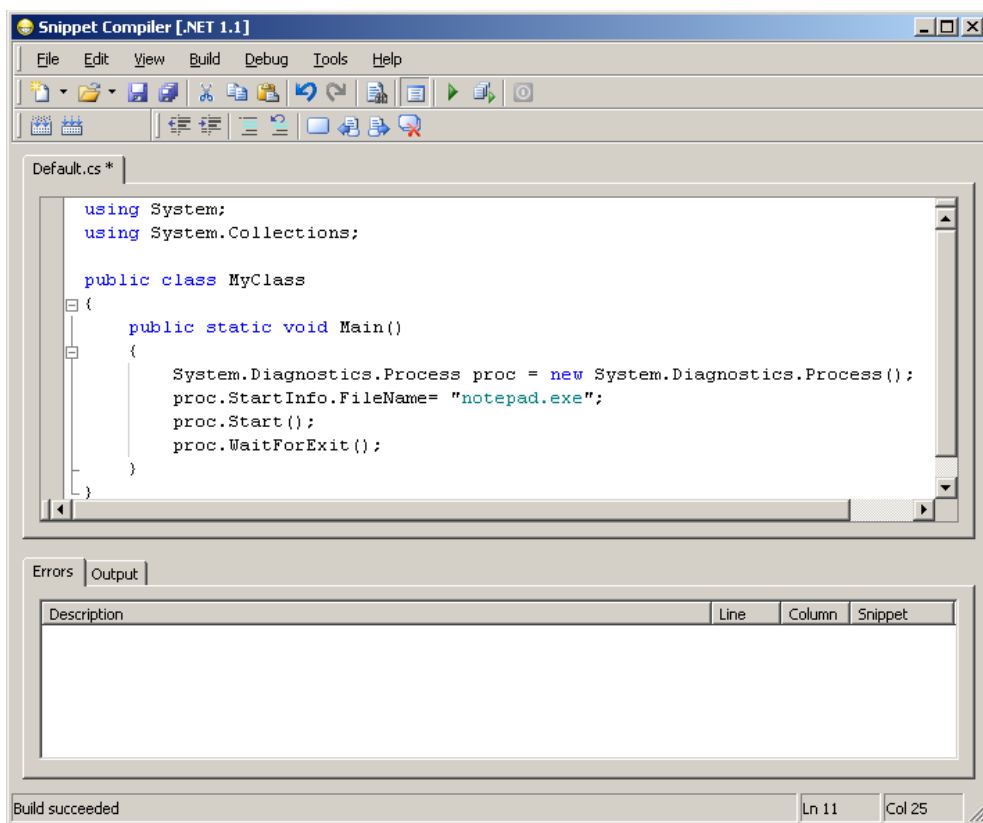
Lấy ví dụ, giả sử bạn muốn chạy một ứng dụng nào đó từ *Microsoft .NET Framework*. Trong *Snippet Compiler*, bạn hãy tạo một ứng dụng *Console* mới. Phần mã lệnh có thể được tạo bên trong phương thức `Main` của ứng dụng này. Đoạn mã dưới đây trình bày cách tạo một thể hiện của *Notepad* từ *.NET Framework*:

```
using System;
using System.Collections;
```

```

public class MyClass
{
    public static void Main()
    {
        System.Diagnostics.Process proc = new System.Diagnostics.Process();
        proc.StartInfo.FileName= "notepad.exe";
        proc.Start();
        proc.WaitForExit();
    }
}

```



Hình A-1 Snippet Compiler

Hình A-1 cho thấy đoạn mã này trong *Snippet Compiler*. Để thử nghiệm đoạn mã này, bạn chỉ việc nhấn nút *Play* (hình tam giác xanh), và nó sẽ chạy ở chế độ gỡ rối. Đoạn mã này sẽ sinh ra một cửa sổ pop-up (ứng dụng *Console*), và *Notepad* sẽ xuất hiện. Khi bạn đóng *Notepad*, ứng dụng *Console* cũng sẽ đóng.

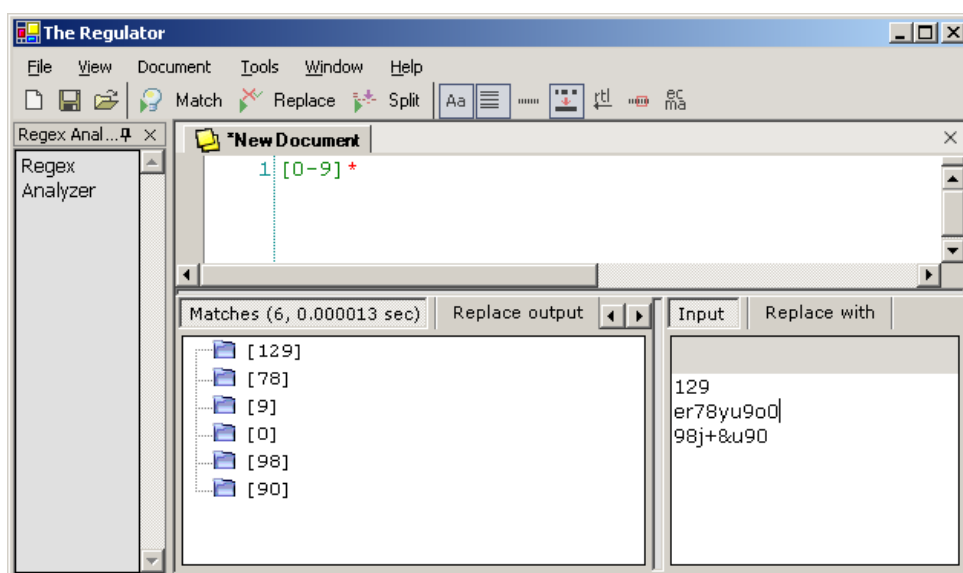


Snippet Compiler được viết bởi *Jeff Key* và có thể được download tại [<http://www.sliver.com/dotnet/SnippetCompiler>].

A.2 Xây dựng biểu thức chính quy với Regulator

Regulator là một công cụ với đầy đủ tính năng dùng để xây dựng và kiểm tra các biểu thức chính quy. Biểu thức chính quy được sử dụng để định nghĩa các mẫu trong những chuỗi dựa trên ký tự, tần số xuất hiện, và thứ tự ký tự. Chúng thường được sử dụng nhiều nhất để làm phương tiện xác nhận tính hợp lệ của đầu vào do người dùng cung cấp hoặc tìm một chuỗi ký tự bên trong một chuỗi lớn hơn—chẳng hạn, khi tìm kiếm một địa chỉ *URL* hay e-mail trên một trang web.

Regulator cho phép bạn nhập một biểu thức chính quy và một đầu vào nào đó (bạn sẽ chạy biểu thức chính quy dựa trên đầu vào này). Bằng cách này, bạn có thể thấy cách thức làm việc của biểu thức chính quy và kết quả trả về trước khi hiện thực nó trong ứng dụng của mình. Hình A-2 cho thấy *Regulator* với một biểu thức chính quy đơn giản.



Hình A-2 Regulator với một biểu thức chính quy đơn giản

Phần document chứa biểu thức chính quy, trong ví dụ này là `[0-9]*`, biểu thức này trùng khớp với bất kỳ dãy chữ số nào. Hộp ở phía dưới phải chứa đầu vào cho biểu thức chính quy, và hộp ở phía dưới trái hiển thị những trùng khớp mà biểu thức chính quy tìm thấy bên trong các đầu vào. Viết và kiểm tra các biểu thức chính quy trong một ứng dụng độc lập như thế này thì dễ hơn nhiều so với thao tác chúng trong ứng dụng của bạn.

Một trong những tính năng hay nhất của *Regulator* là khả năng tìm kiếm thư viện biểu thức chính quy trực tuyến tại [<http://regexlib.com>]. Ví dụ, nếu nhập chuỗi *"phone"* vào hộp tìm

kiểm, bạn sẽ tìm thấy hơn 20 biểu thức chính quy khác nhau trùng khớp với các số điện thoại, bao gồm các biểu thức cho Anh, Úc, và nhiều số điện thoại khác.



Regulator được viết bởi Roy Oshero ve và có thể được download tại [\[http://regex.osherove.com\]](http://regex.osherove.com).

A.3 Sinh mã với CodeSmith

CodeSmith là một công cụ sinh mã dựa-trên-template (khuôn mẫu), sử dụng một cú pháp tương tự như *ASP.NET* để kết sinh bất kỳ kiểu mã hay text nào. Khác với nhiều công cụ sinh mã khác, *CodeSmith* không yêu cầu bạn mô tả một bản thiết kế hay kiến trúc ứng dụng cụ thể. Khi sử dụng *CodeSmith*, bạn có thể kết sinh mọi thứ, từ một tập hợp đơn giản, được-định-kiểu-mạnh đến toàn bộ một ứng dụng.

Khi xây dựng một ứng dụng, bạn thường phải lặp đi lặp lại những tác vụ nào đó, chẳng hạn viết mã truy xuất dữ liệu hay xây dựng các tập hợp tùy biến. *CodeSmith* đặc biệt hữu ích trong những tình huống như vậy, vì bạn có thể viết các template để tự động hóa các tác vụ này, điều này không chỉ cải thiện hiệu năng mà còn giúp bạn bớt nhàm chán.

CodeSmith có sẵn một số template, bao gồm những template cho tất cả các kiểu tập hợp .NET cũng như những template để sinh thủ tục tồn trữ, nhưng sức mạnh thực sự của công cụ này chính là khả năng tạo các template tùy biến.

Template của *CodeSmith* chỉ là file văn bản đơn thuần (có thể được tạo bằng công cụ soạn thảo văn bản bất kỳ), với phần mở rộng là *.cst*.

Template được giới thiệu dưới đây sẽ nhận một chuỗi và rồi xây dựng một lớp dựa vào chuỗi đó. Bước đầu tiên là viết phần header, phần này khai báo ngôn ngữ của template, ngôn ngữ đích, và mô tả vắn tắt về template:

```
<%@ CodeTemplate Language="C#"
    TargetLanguage="C#"
    Description="Car Template" %>
```

Phần kế tiếp của template là các khai báo thuộc tính, các thuộc tính này sẽ được chỉ định mỗi khi template chạy. Ví dụ dưới đây khai báo một thuộc tính chuỗi:

```
<%@ Property Name="ClassName" Type="String" Category="Context"
    Description="Class Name" %>
```

Khai báo này sẽ làm cho thuộc tính *ClassName* xuất hiện trong cửa sổ thuộc tính của *CodeSmith* để nó có thể được chỉ định khi template chạy.

Bước tiếp theo là xây dựng phần thân của template (với mã lệnh tương tự như *ASP.NET*). Như bạn có thể thấy, template này nhận chuỗi nhập và sinh ra một lớp với tên đó. Trong phần thân của template, các thẻ đóng và mở được sử dụng như trong *ASP.NET*. Trong template này, ta chỉ chèn giá trị thuộc tính, nhưng cũng có thể sử dụng bất kỳ kiểu mã .NET nào bên trong các thẻ này.

```
public sealed class <%= ClassName %>
{
    private static volatile <%= ClassName %> _instance;
    private <%= ClassName %>() {}
```

```

private static readonly object _syncRoot = new object();

public static <%= ClassName %> Value
{
    get
    {
        if (_instance == null)
        {
            lock(_syncRoot)
            {
                if (_instance == null)
                {
                    _instance = new <%= ClassName %>();
                }
            }
        }
        return _instance;
    }
}

```

Một khi template đã hoàn tất, bạn hãy nạp nó vào *CodeSmith* (xem hình A-3). Bạn có thể nhận thấy thuộc tính phía bên trái là thuộc tính mà chúng ta đã khai báo trong template.

Nếu bạn nhập “*SingletonClass*” làm tên lớp và nhấp nút *Generate*, *CodeSmith* sẽ sinh ra lớp dưới đây:

```

public sealed class SingletonClass
{
    private static volatile SingletonClass _instance;
    private SingletonClass() {}
    private static readonly object _syncRoot = new object();

    public static SingletonClass Value
    {
        get
        {
            if (_instance == null)
            {

```

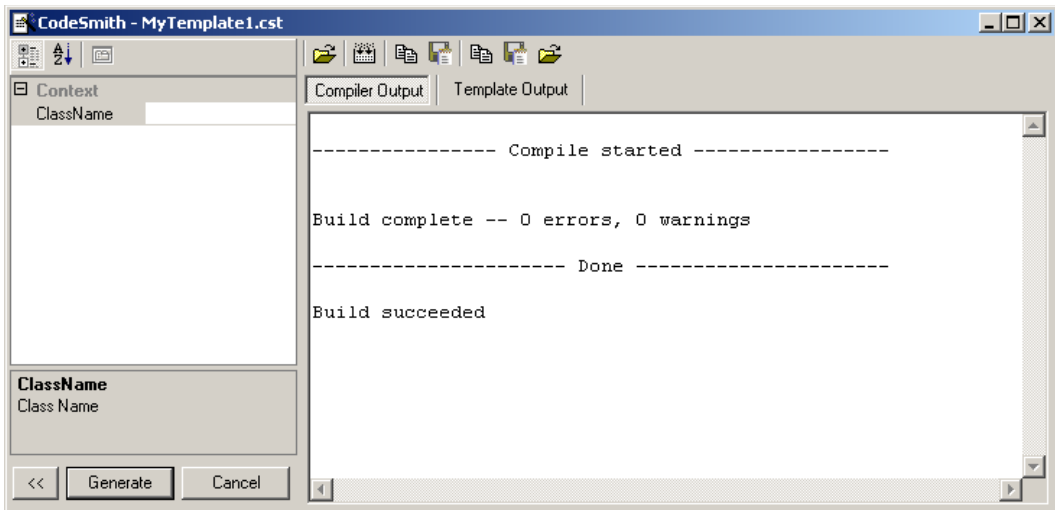


```

        lock(_syncRoot)
        {
            if (_instance == null)
            {
                _instance = new SingletonClass();
            }
        }
    }

    return _instance;
}
}
}

```



Hình A-3 Nạp template vào CodeSmith

CodeSmith tương đối dễ sử dụng và có thể sinh ra những kết quả tuyệt vời nếu được áp dụng một cách đúng đắn. Một trong những phần phổ biến nhất cần được sinh mã là tầng truy xuất dữ liệu. *CodeSmith* có một assembly đặc biệt với tên là *SchemaExplorer*, assembly này có thể được sử dụng để sinh ra các template từ bảng, thủ tục tồn trữ, hay hầu như bất kỳ đối tượng *SQL Server* nào khác.



CodeSmith được viết bởi *Eric J. Smith* và có thể được download (bản dùng thử) tại [<http://www.ericjsmith.net/codesmith/download.aspx>].

A.4 Viết kiểm thử đơn vị với NUnit

NUnit là bộ khung kiểm thử đơn vị được xây dựng cho *.NET Framework*, cho phép bạn viết các phương thức kiểm thử theo ngôn ngữ do bạn chọn để kiểm tra một hàm cụ thể của chương

trình. Ứng dụng *NUnit* cung cấp một bộ khung để viết các kiểm thử đơn vị, cũng như một giao diện đồ họa để chạy các kiểm thử đơn vị và xem kết quả.

Ví dụ, chúng ta cần kiểm tra chức năng của lớp *Hashtable* trong *.NET Framework* để xác định hai đối tượng có thể được thêm vào và lấy ra hay không. Bước đầu tiên là tham chiếu đến assembly *NUnit.Framework* để có thể truy xuất các thuộc tính và phương thức của bộ khung *NUnit*. Bước kế tiếp là tạo một lớp và đánh dấu nó với đặc tính `[TestFixture]` để *NUnit* biết lớp này có chứa phương thức kiểm thử.

```
using System;
using System.Collections;
using NUnit.Framework;

namespace NUnitExample
{
    [TestFixture]
    public class HashtableTest {
        public HashtableTest() {

        }
    }
}
```

Kế tiếp, chúng ta tạo một phương thức và đánh dấu nó với đặc tính `[Test]` để *NUnit* biết đây là phương thức kiểm thử. Trong phương thức này, chúng ta sẽ thiết lập một *Hashtable* và đưa vào đó hai giá trị, sau đó sử dụng phương thức `Assert.AreEqual` để xem chúng ta có thể thu lấy đúng các giá trị mà chúng ta đã đưa vào *Hashtable* hay không:

```
[Test]
public void HashtableAddTest()
{
    Hashtable ht = new Hashtable();

    ht.Add("Key1", "Value1");
    ht.Add("Key2", "Value2");

    Assert.AreEqual("Value1", ht["Key1"], "Wrong object returned!");
    Assert.AreEqual("Value2", ht["Key2"], "Wrong object returned!");
}
```

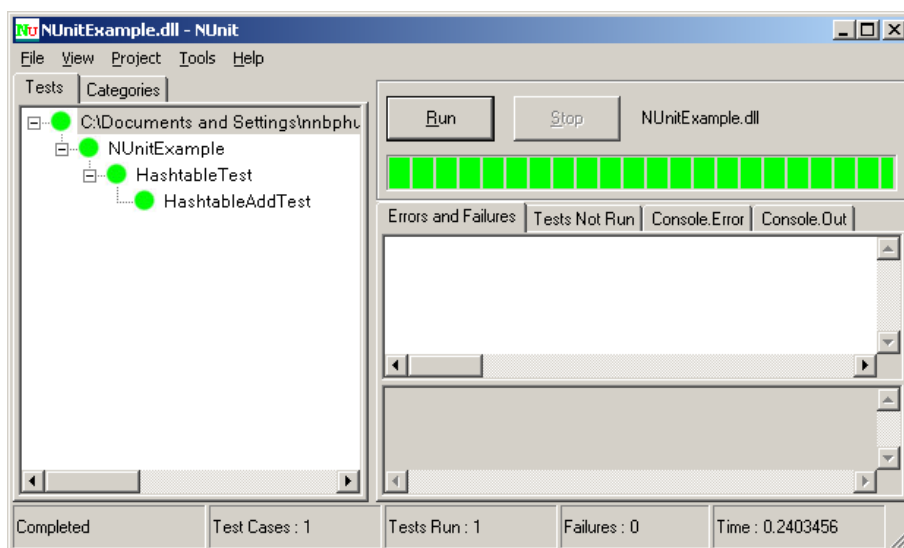
Để chạy phương thức kiểm thử này, bạn cần tạo một dự án *NUnit*, mở assembly đã được sinh ra bên trên và nhấp nút *Run*. Hình A-4 cho thấy kết quả.

Trên đây chỉ là một kiểm thử đơn giản, nhưng cho thấy khả năng của *NUnit*. Có rất nhiều kiểu kiểm thử, cũng như nhiều phương thức *Assert*, có thể được sử dụng để kiểm thử mọi phần trong mã lệnh của bạn.

Một kiểm thử đơn vị có thể lưu và chạy lại mỗi khi bạn sửa đổi mã lệnh, điều này giúp bạn phát hiện lỗi dễ dàng hơn và đảm bảo phát triển ứng dụng tốt hơn.



NUnit là một dự án mã nguồn mở và có thể được download tại [<http://www.nunit.org>].



Hình A-4 NUnit

Cũng có một bản add-in của *NUnit* cho *Visual Studio .NET*, add-in này cho phép bạn trực tiếp chạy các phương thức kiểm thử đơn vị từ *Visual Studio .NET* (có thể được download tại [<http://sourceforge.net/projects/nunitaddin>]).

Để có thêm thông tin về *NUnit* và vị trí của nó trong việc phát triển test-driven (vận hành theo kiểm thử), bạn hãy xem bài viết “*Test-Driven C#: Improve the Design and Flexibility of Your Project with Extreme Programming Techniques*” trong đĩa CD đính kèm.

A.5 Kiểm soát mã lệnh với FxCop

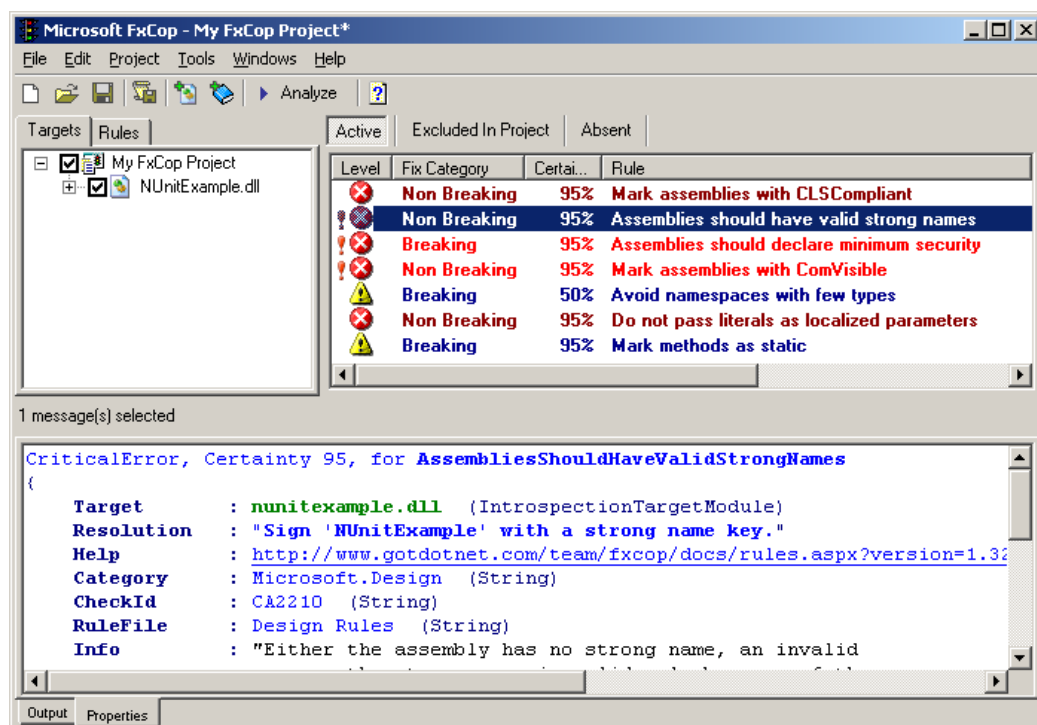
.NET Framework rất mạnh, có nghĩa khả năng tạo ra những ứng dụng tuyệt vời là rất cao, nhưng khả năng tạo ra những chương trình tồi cũng là rất cao. *FxCop* là một trong những công cụ có thể được sử dụng để trợ giúp tạo ra những ứng dụng tốt hơn bằng việc cho phép bạn khảo sát một assembly và kiểm tra tính tương thích của nó với một số quy luật. *FxCop* chứa tập các quy luật do *Microsoft* tạo ra, nhưng bạn cũng có thể tạo ra những quy luật cho mình. Chẳng hạn, nếu muốn tất cả các lớp đều có một phương thức khởi dựng mặc định không đối số, bạn có thể viết một quy luật kiểm tra phương thức khởi dựng trên mỗi lớp của assembly. Để có thêm thông tin về việc tạo những quy luật tùy biến, bạn hãy vào [<http://msdn.microsoft.com/msdnmag/issues/04/06/bugslayer>].

Ví dụ, chúng ta hãy xem *FxCop* phát hiện ra lỗi gì trong assembly *NUnitExample* (đã được trình bày mục A.4). Trước tiên, bạn cần tạo một dự án *FxCop* và đưa assembly này vào. Sau

đó, bạn hãy nhấn *Analyze*, *FxCop* sẽ khảo sát assembly này và đưa ra các thông báo lỗi (xem hình A-5).

FxCop nhận thấy một số vấn đề với assembly này. Bạn có thể nhấp đúp lên một lỗi để xem chi tiết, bao gồm lời mô tả quy luật và nơi mà bạn có thể tìm thấy nhiều thông tin hơn.

FxCop có thể giúp bạn tạo mã lệnh tốt hơn, nhất quán hơn, nhưng nó không thể sửa chữa việc thiết kế tồi hay lập trình kém. *FxCop* cũng không phải là một sự thay thế cho việc kiểm tra mã lệnh (*code review*), nhưng vì nó có thể bắt nhiều lỗi trước khi kiểm tra mã lệnh nên bạn có thể dành nhiều thời gian cho các vấn đề hệ trọng hơn là phải lo lắng về các quy ước đặt tên.



Hình A-5 FxCop đưa ra các thông báo khi khảo sát NUnitExample



FxCop được phát triển bởi *Microsoft* và có thể được download tại [\[http://www.gotdotnet.com/team/fxcop\]](http://www.gotdotnet.com/team/fxcop).

A.6 Khảo sát assembly với .NET Reflector

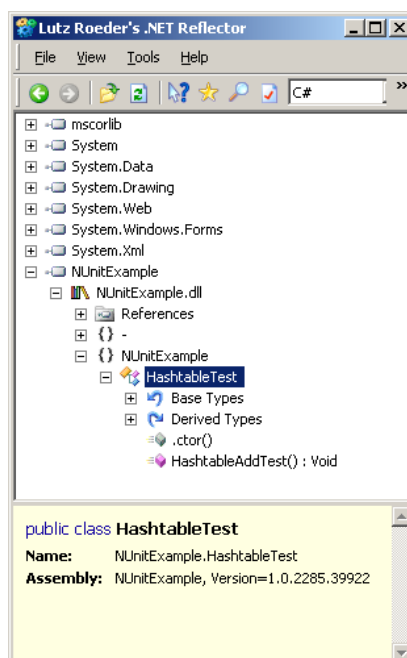
.NET Reflector là một trình duyệt lớp (*class browser*) và trình dịch ngược (*decompiler*), nó có thể khảo sát một assembly và cho bạn thấy tất cả các “bí mật” trong đó. *.NET Framework* đã đưa ra cơ chế phản chiếu (*reflection*) để khảo sát bất kỳ mã lệnh nào dựa-trên-.NET, cho dù nó là một lớp đơn hay toàn bộ một assembly. Cơ chế phản chiếu cũng có thể được sử dụng để thu lấy thông tin về các lớp, phương thức, và thuộc tính khác nhau trong một assembly nào đó. Sử

dụng *.NET Reflector*, bạn có thể duyệt các lớp và các phương thức của một assembly, bạn có thể khảo sát ngôn ngữ trung gian (*Microsoft Intermediate Language—MSIL*) do các lớp và phương thức này sinh ra, và bạn có thể dịch ngược các lớp và phương thức sang *C#* hay *Visual Basic .NET*.

Ví dụ, chúng ta sẽ sử dụng *.NET Reflector* để khảo sát assembly *NUnitExample* (đã được trình bày ở mục A.4). Hình A-6 thể hiện assembly này khi được nạp vào *.NET Reflector*. Bên trong *.NET Reflector* còn có những công cụ mà bạn có thể sử dụng để khảo sát thêm assembly này. Để xem *MSIL* của một phương thức, bạn hãy nhấp vào phương thức này và chọn *Disassembler*.

Ngoài việc xem *MSIL*, bạn cũng có thể xem phương thức ở dạng *C#* bằng cách chọn *Decompiler* từ trình đơn *Tools*. Bạn cũng có thể xem phương thức này được dịch ngược sang *Visual Basic .NET* hay *Delphi* bằng cách thay đổi tùy chọn trong trình đơn *Languages*. Dưới đây là đoạn mã do *.NET Reflector* sinh ra:

```
public void HashtableAddTest()
{
    Hashtable hashtable1 = new Hashtable();
    hashtable1.Add("Key1", "Value1");
    hashtable1.Add("Key2", "Value2");
    Assert.AreEqual("Value1", hashtable1["Key1"],
        "Wrong object returned!");
    Assert.AreEqual("Value2", hashtable1["Key2"],
        "Wrong object returned!");
}
```



Hình A-6 Khảo sát NUnitExample với .NET Reflector

Đoạn mã trên rất giống với đoạn mã mà chúng ta đã viết:

```
public void HashtableAddTest()
{
    Hashtable ht = new Hashtable();

    ht.Add("Key1", "Value1");
    ht.Add("Key2", "Value2");

    Assert.AreEqual("Value1", ht["Key1"], "Wrong object returned!");
    Assert.AreEqual("Value2", ht["Key2"], "Wrong object returned!");
}
```

Mặc dù có một vài khác biệt nhỏ về mã lệnh nhưng chúng giống hệt nhau về chức năng.

Công dụng hay nhất của *.NET Reflector* là khảo sát các assembly và phương thức của *.NET Framework*. *.NET Framework* cung cấp nhiều cách khác nhau để thực hiện các thao tác tương tự nhau. Ví dụ, nếu bạn cần đọc một tập dữ liệu từ XML, có nhiều cách khác nhau để thực hiện điều này: sử dụng *XmlDocument*, *XPathNavigator*, hay *XmlReader*. Bằng cách sử dụng *.NET Reflector*, bạn có thể xem *Microsoft* đã sử dụng gì khi viết phương thức *ReadXml* của *DataSet*, hoặc họ đã làm gì khi đọc dữ liệu từ file cấu hình. *.NET Reflector* cũng rất có ích khi tìm hiểu cách tạo các đối tượng như *HttpHandlers*; và qua đó, bạn biết được cách thức mà nhóm phát triển của *Microsoft* đã xây dựng các đối tượng đó trong *Framework*.

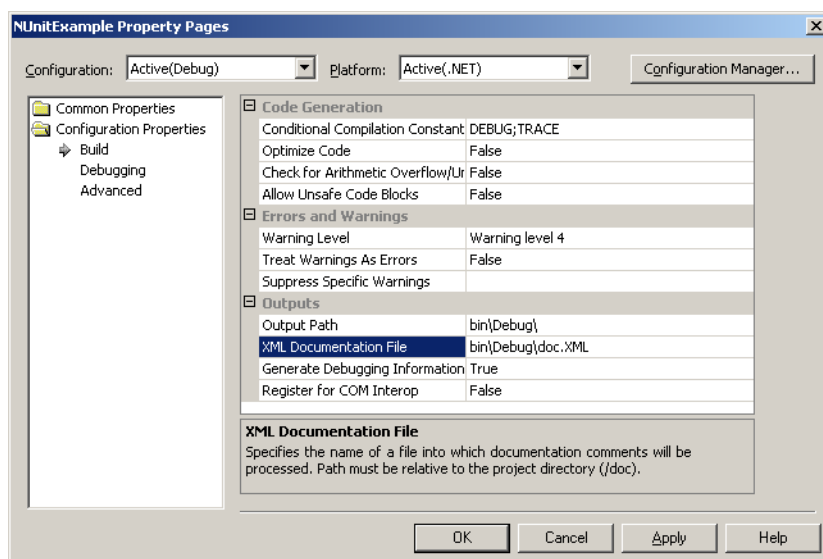


.NET Reflector được viết bởi *Lutz Roeder* và có thể được download tại [<http://www.aisto.com/roeder/dotnet/>].

A.7 Lập tài liệu mã lệnh với NDoc

Việc lập tài liệu mã lệnh gần như là một công việc không mấy hứng thú. Ở đây không nói về tài liệu thiết kế mà là tài liệu cho từng phương thức và thuộc tính của lớp. Công cụ *NDoc* sẽ tự động sinh tài liệu cho mã lệnh của bạn bằng cách sử dụng cơ chế phản chiếu để khảo sát assembly và sử dụng file XML được sinh từ các chú thích XML C# (các chú thích XML chỉ có hiệu lực cho C#, nhưng có một *Visual Studio .NET Power Toy* với tên là *VBCommenter* cũng sẽ thực hiện giống như vậy đối với *Visual Basic .NET*).

Với *NDoc*, bạn vẫn cứ lập tài liệu cho mã lệnh, nhưng lập khi viết mã (trong các chú thích XML). Bước đầu tiên khi sử dụng *NDoc* là mở chức năng sinh chú thích XML đối với assembly của bạn. Nhấp phải vào dự án và chọn *Properties* | *Configuration Properties* | *Build*, rồi nhập một đường dẫn để lưu file XML trong tùy chọn *XML Documentation File* (xem hình A-7). Khi dự án được tạo dựng, một file XML sẽ được sinh ra với tất cả các chú thích XML đi kèm.



Hình A-7 Chọn đường dẫn để lưu tài liệu XML

Dưới đây là phương thức ở mục A.4:

```
/// <summary>
/// This test adds a number of values to the Hashtable collection
/// and then retrieves those values and checks if they match.
/// </summary>

[Test]
public void HashtableAddTest()
{
    // Phần thân phương thức ở đây.
}
```

Phần chú thích *XML* cho phương thức này sẽ được trích xuất và lưu thành file *XML* như sau:

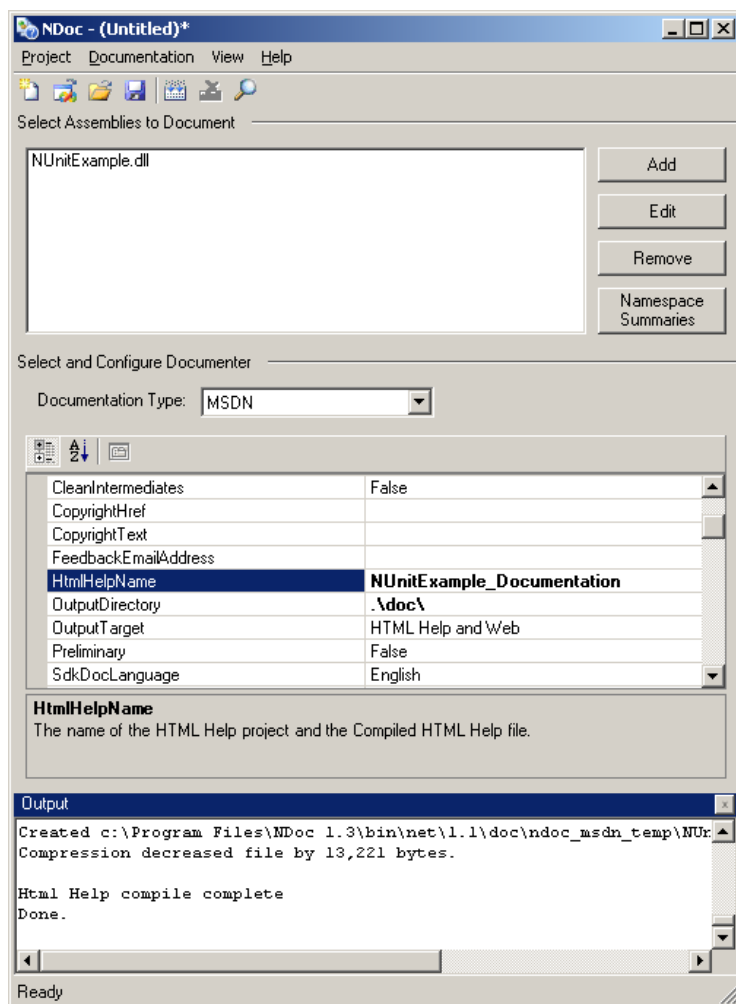
```
<?xml version="1.0" ?>
<doc>
  <assembly>
    <name>NUnitExample</name>
  </assembly>
  <members>
    <member name="M:NUnitExample.HashtableTest.HashtableAddTest">
      <summary>This test adds a number of values to the Hashtable
        collection and then retrieves those values and checks if
        they match.
      </summary>
    </member>
  </members>
</doc>
```

```

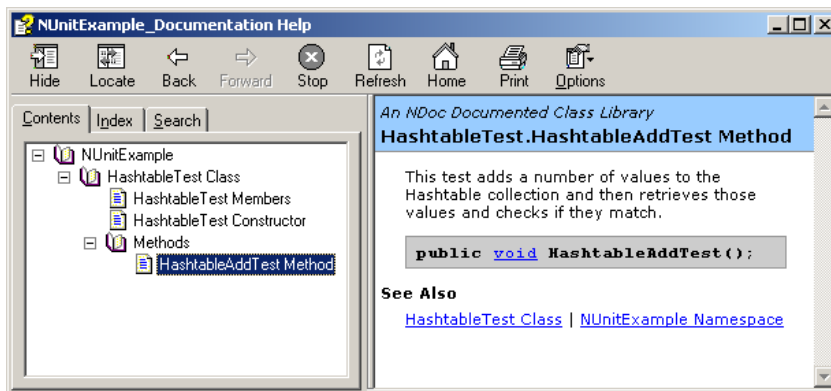
    </summary>
  </member>
</members>
</doc>

```

Bước kế tiếp là nạp assembly và file *XML* vào *NDoc*. Sau đó, nhấp nút *Build Documentation* để chạy quá trình sinh tài liệu (xem hình A-8). Hình A-9 là tài liệu *CHM* do *NDoc* sinh ra.



Hình A-8 NDoc



Hình A-9 Tài liệu CHM do NDoc sinh ra



NDoc là một dự án mã nguồn mở và có thể được download tại [<http://ndoc.sourceforge.net>].

A.8 Tạo dựng giải pháp với NAnt

NAnt là một công cụ tạo dựng dựa-trên-.NET, giúp bạn viết một quy trình tạo dựng dự án cho mình. Khi có nhiều nhà phát triển cùng làm việc trên một dự án, bạn không thể phó thác việc tạo dựng cho từng người. Bạn cũng không muốn phải thường xuyên tạo dựng dự án một cách thủ công. Thay vào đó, bạn viết một quy trình tạo dựng tự động chạy mỗi đêm. *NAnt* cho phép bạn tạo dựng giải pháp, chép file, chạy các kiểm tra *NUnit*, gửi e-mail, và nhiều nữa. Đáng tiếc, *NAnt* thiếu giao diện đồ họa, nhưng nó có một ứng dụng *Console* và các file *XML* chỉ định các tác vụ nào sẽ được hoàn thành trong quá trình tạo dựng. Lưu ý rằng *MSBuild*, một nền tạo dựng mới trong phiên bản *Visual Studio 2005*, cũng có tính năng tương tự như *NAnt*.

Ví dụ, chúng ta cần viết file tạo dựng *NAnt* cho dự án *NUnitExample* ở mục A.4. Trước tiên, bạn hãy tạo một file *XML* với phần mở rộng là *.build*, và đặt nó trong thư mục gốc của dự án:

```
<?xml version="1.0"?>
<project name="NUnit Example" default="build" basedir=".">
  <description>The NUnit Example Project</description>
  <property name="debug" value="true"/>
  <target name="build" description="compiles the source code">
    <csc target="library" output=".\\bin\\debug\\NUnitExample.dll"
      debug="{debug}">
      <references>
        <includes name="C:\\Program Files\\NUnit 2.2\\bin
          \\NUnit.Framework.dll" />
      </references>
    </csc>
  </target>
  <sources>
    <includes name="NUnitExample.cs"/>
  </sources>
</project>
```

```

        </sources>

    </csc>

</target>

</project>

```

Thẻ `project` được sử dụng để đặt tên cho dự án, target mặc định, và thư mục cơ sở. Thẻ này cần có những thẻ con sau:

- Thẻ `description` được sử dụng để đặt một mô tả ngắn gọn về dự án.
- Thẻ `property` được sử dụng để lưu trữ một thiết lập sao cho nó có thể được truy xuất từ bất cứ đâu trong file tạo dựng. Ví dụ này tạo một thuộc tính với tên là `debug`, và thiết lập nó là `true` hay `false` tùy vào bạn có muốn dự án được biên dịch ở cấu hình gỡ rối hay không (thuộc tính này không ảnh hưởng gì đến cách thức tạo dựng dự án; nó chỉ là một biến số mà bạn có thể thiết lập và sẽ được thu về khi bạn thật sự xác định cách thức tạo dựng dự án).
- Kế tiếp là thẻ `target`. Một dự án có thể có nhiều target (có thể được chỉ định khi *NAnt* chạy). Nếu không có target nào được chỉ định, target mặc định sẽ được sử dụng (ta đã thiết lập nó trong thẻ `project`). Trong ví dụ này, target mặc định là `build`. Bên trong thẻ `target`, bạn cần thiết lập tên của target và mô tả những gì mà target này sẽ thực hiện.

Thẻ `csc` được sử dụng để chỉ định những gì sẽ được truyền cho trình biên dịch *C#*. Trước tiên, bạn phải thiết lập target cho thẻ `csc`. Do cần tạo file `.dll` nên ví dụ này thiết lập target là `library`. Kế tiếp, bạn phải thiết lập output cho thẻ `csc`, đây là nơi mà file `.dll` sẽ được tạo. Cuối cùng, bạn cần thiết lập thuộc tính `debug`, cho biết dự án có được biên dịch ở chế độ gỡ rối hay không. Vì đã tạo một thuộc tính trước đó để lưu trữ giá trị này, ta có thể sử dụng chuỗi `$(debug)` để truy xuất giá trị của thuộc tính.

Thẻ `csc` cũng cần có các thẻ con: thẻ `references` cho biết những assembly nào cần được tham chiếu; và thẻ `sources` cho biết những file nào đi kèm. Ví dụ này tham chiếu đến assembly *NUnit.Framework.dll* và chứa file *NUnitExample.cs* .

Để tạo dựng, bạn cần đến thư mục gốc của dự án và thực thi *NAnt.exe* ở đó (xem hình A-10). Nếu tạo dựng thành công, bạn có thể tìm thấy file các `.dll` và `.pdb` trong thư mục *bin* của dự án.

```

C:\WINDOWS\System32\cmd.exe

C:\NUnitExample>NAnt.exe
NAnt 0.85 (Build 0.85.1932.0; rc3; 4/16/2005)
Copyright (C) 2001-2005 Gerry Shaw
http://nant.sourceforge.net

Buildfile: file:///C:/NUnitExample/NUnitExample.build
Target framework: Microsoft .NET Framework 1.1
Target(s) specified: build

build:
C:\NUnitExample\NUnitExample.build(7,14): Element <includes... /> for <assemblyf
ileset... /> is deprecated. Use <include> element instead.
C:\NUnitExample\NUnitExample.build(10,14): Element <includes... /> for <fileset.
.. /> is deprecated. Use <include> element instead.
  [csc] Compiling 1 files to 'C:\NUnitExample\bin\debug\NUnitExample.dll'.

BUILD SUCCEEDED - 0 non-fatal error(s), 2 warning(s)

Total time: 2.1 seconds.

C:\NUnitExample>

```

Hình A-10 Thực thi NAnt.exe tại thư mục gốc của dự án

Tuy không dễ dàng như việc nhấp *Build* trong *Visual Studio*, nhưng *NAnt* là một công cụ rất mạnh khi xây dựng quy trình tạo dựng chạy tự động theo lịch biểu. *NAnt* cũng có các tính năng hữu ích như chạy các kiểm thử đơn vị hay chép các file đi kèm (các tính năng này không được quy trình tạo dựng của *Visual Studio 2003* hỗ trợ).



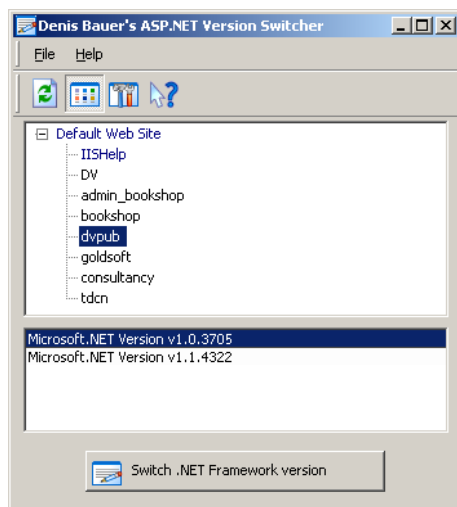
NAnt là một dự án mã nguồn mở và có thể được download tại [<http://sourceforge.net/projects/nant/>].

A.9 Chuyển đổi phiên bản ASP.NET với ASP.NET Version Switcher

Khi thự lý một yêu cầu, *IIS* xem phần mở rộng của file được yêu cầu; và rồi dựa vào các ánh xạ phần mở rộng (*extension mapping*) cho thư mục ảo hay website, nó ủy nhiệm yêu cầu cho một phần mở rộng *ISAPI* hoặc tự thự lý nó. Đây là cách *ASP.NET* làm việc; các ánh xạ phần mở rộng được đăng ký cho tất cả các phần mở rộng *ASP.NET* và hướng chúng đến *aspnet_isapi.dll*.

Khi bạn cài đặt *ASP.NET 1.1*, ánh xạ phần mở rộng được nâng cấp sang phiên bản mới của *aspnet_isapi.dll*. Điều này gây ra lỗi khi một ứng dụng đã được tạo dựng trên *ASP.NET 1.0* lại chạy trên phiên bản *1.1*. Để giải quyết vấn đề này, bạn có thể chuyển tất cả các ánh xạ phần mở rộng trở về phiên bản *1.0* của *aspnet_isapi.dll*, nhưng với 18 ánh xạ phần mở rộng thì quả là vất vả nếu làm thủ công. Đây chính là nơi *ASP.NET Version Switcher* trở nên hữu dụng. Tiện ích nhỏ này có thể được sử dụng để chuyển phiên bản *.NET Framework* của bất kỳ ứng dụng *ASP.NET* nào.

Sử dụng công cụ này rất đơn giản: bạn hãy chọn một ứng dụng và rồi chọn phiên bản *.NET Framework* mà bạn muốn ứng dụng này sử dụng (xem hình A-11). Sau đó, công cụ này sẽ sử dụng tiện ích dòng lệnh *aspnet_regiis.exe* để chuyển ứng dụng về phiên bản *.NET Framework* đã được chọn. Công cụ này càng hữu ích khi có thêm các phiên bản *ASP.NET* và *.NET Framework* mới trong tương lai.

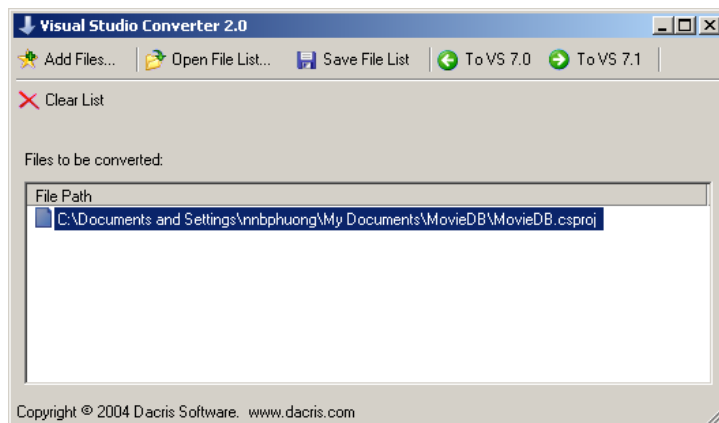


Hình A-11 ASP.NET Version Switcher

ASP.NET Version Switcher được viết bởi Denis Bauer và có thể được download tại [<http://www.denisbauer.com/NETTools/ASPNETVersionSwitcher.aspx>].

A.10 Chuyển đổi phiên bản dự án với Visual Studio .NET Project Converter

Visual Studio .NET Project Converter (xem hình A-12) được sử dụng để chuyển đổi phiên bản của một file dự án *Visual Studio*. Mặc dù chỉ có một ít khác biệt giữa phiên bản 1.0 và 1.1 của *.NET Framework*, nhưng một khi file dự án đã được chuyển từ *Visual Studio .NET 2002* sang *Visual Studio .NET 2003* thì không thể chuyển ngược lại. Đôi khi việc chuyển ngược lại cần thiết. Công cụ này có thể chuyển bất kỳ file giải pháp hay dự án nào từ *Visual Studio 7.1* (*Visual Studio .NET 2003*) về *Visual Studio 7.0* (*Visual Studio .NET 2002*), và ngược lại.

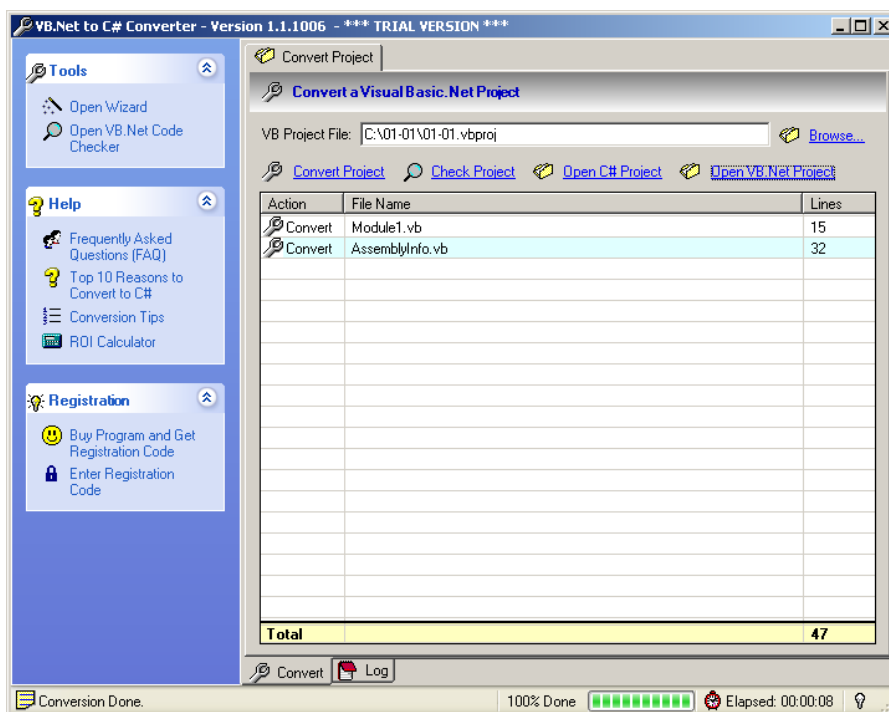


Hình A-12 Visual Studio .NET Project Converter

🔗 *Visual Studio .NET Project Converter* được viết bởi *Dacris Software* và có thể được download tại [<http://www.codeproject.com/macro/vsconvert.asp>].

A.11 Chuyển mã nguồn VB.NET sang C# với VB.NET to C# Converter

VB.NET to C# Converter được sử dụng để chuyển toàn bộ một dự án *VB.NET* sang dự án *C#*. Chương trình này có thể chuyển được một số điểm mà các chương trình khác không thực hiện được; chẳng hạn như các lệnh `ReDim`, các biến dùng chung cục bộ, cơ chế thụ lý sự kiện, các lệnh `Case` phức tạp, các lời gọi *API*...



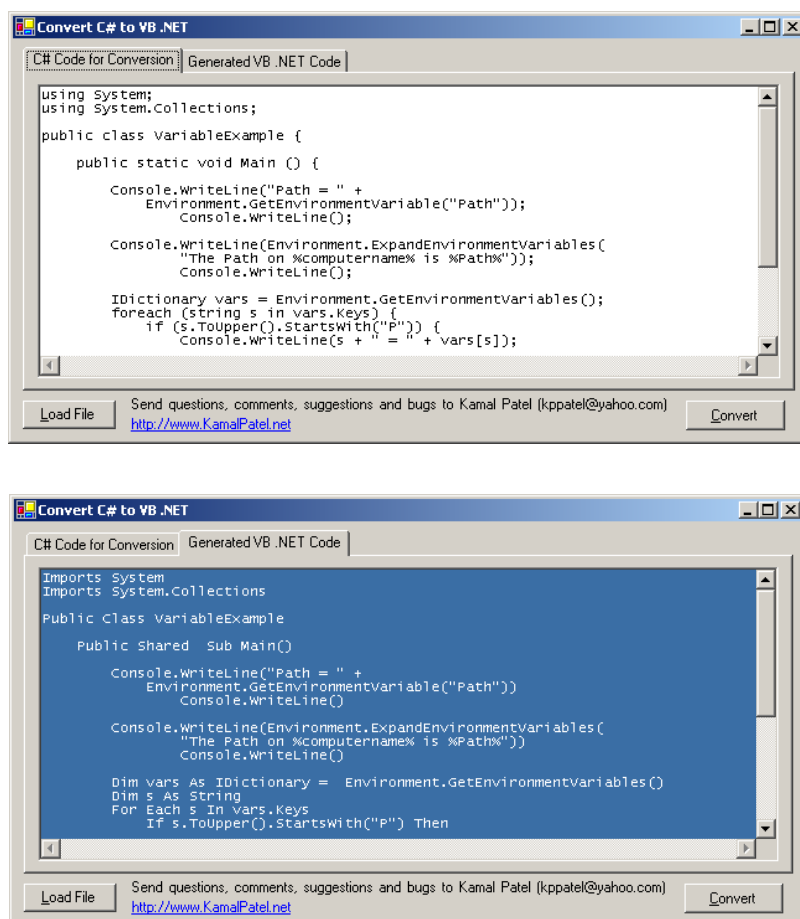
Hình A-13 VB.NET to C# Converter

VB.NET to C# Converter là trình chuyển đổi mã nguồn sang mã nguồn. Sau khi chuyển đổi, mã lệnh của bạn vẫn giữ lại tính dễ đọc vốn có (bao gồm các tên biến và các chú thích). Tính tin cậy của chương trình cũng rất cao, chính xác trên 99% trong hầu hết các lần thử nghiệm.

🔗 *VB.NET to C# Converter* được phát triển bởi *VBConversions* và có thể được download (bản dùng thử) tại [<http://www.vbconversions.com>].

A.12 Chuyển mã nguồn C# sang VB.NET với Convert C# to VB.NET

Ngược với *VB.NET to C# Converter*, *Convert C# to VB.NET* được sử dụng để chuyển mã nguồn *C#* sang *VB.NET*.



Hình A-14 Convert C# to VB.NET



Convert C# to VB.NET được viết bởi *Kamal Patel* và có thể được download tại [<http://www.kamalpatel.net>].

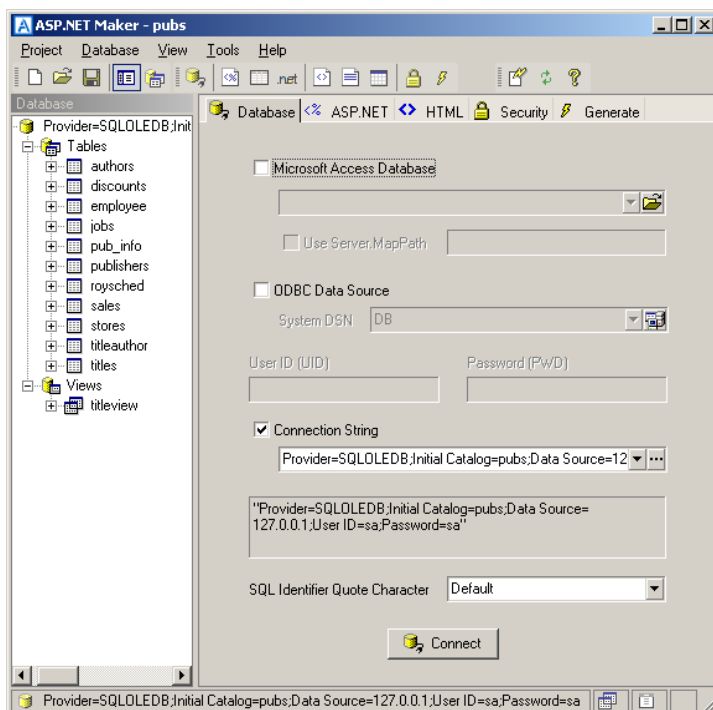
A.13 Xây dựng website quản trị cơ sở dữ liệu với ASP.NET Maker 1.1

ASP.NET Maker là một bộ sinh mã rất mạnh nhưng lại rất dễ sử dụng, giúp bạn nhanh chóng tạo các trang quản trị *ASP.NET* (ngôn ngữ *C#* hay *VB.NET*) từ một nguồn dữ liệu (các cơ sở dữ liệu được hỗ trợ: *Microsoft Access*, *Microsoft SQL Server*, *Oracle*, bất kỳ cơ sở dữ liệu nào với kết nối *ADO* hay *ODBC*). *ASP.NET Maker* được trang bị rất nhiều tính năng hữu ích như khung nhìn drill-down, cơ chế bảo mật cao cấp, tích hợp với *CSS* và *Visual Studio .NET*...

Giả sử bạn cần xây dựng một website dùng cho quản trị cơ sở dữ liệu *pubs* (trong *SQL Server*). Dưới đây là các bước cơ bản:

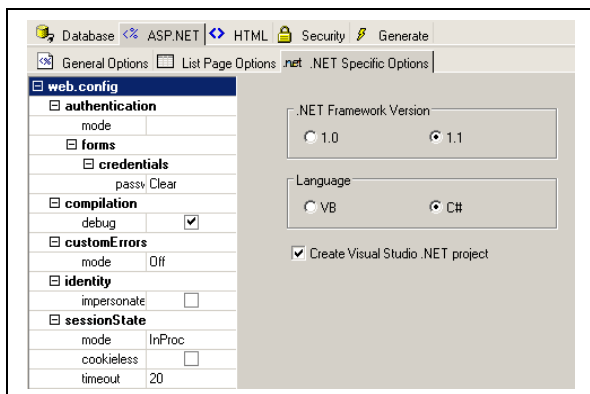
- Trước tiên, vào thẻ *Database*, đánh dấu chọn *Connection String* và nhập chuỗi "Provider=SQLOLEDB;Initial Catalog=pubs;Data Source=127.0.0.1; User

ID=sa;Password=sa” (không có dấu ngoặc kép), rồi nhấp nút *Connect*. Danh sách các bảng và khung nhìn sẽ được nạp và hiển thị phía trái (xem hình A-15). Khi đó, bạn có thể tùy chỉnh mỗi trường trong các bảng và khung nhìn sao cho phù hợp.



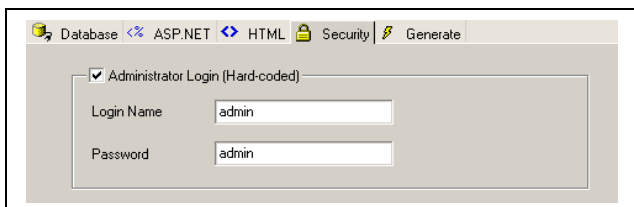
Hình A-15 Nạp cơ sở dữ liệu pubs vào ASP.NET Maker

- Kế tiếp, vào thẻ *ASP.NET*, rồi vào thẻ *.NET Specific Options*, chọn phiên bản *.NET Framework* và ngôn ngữ (*VB.NET* hay *C#*).



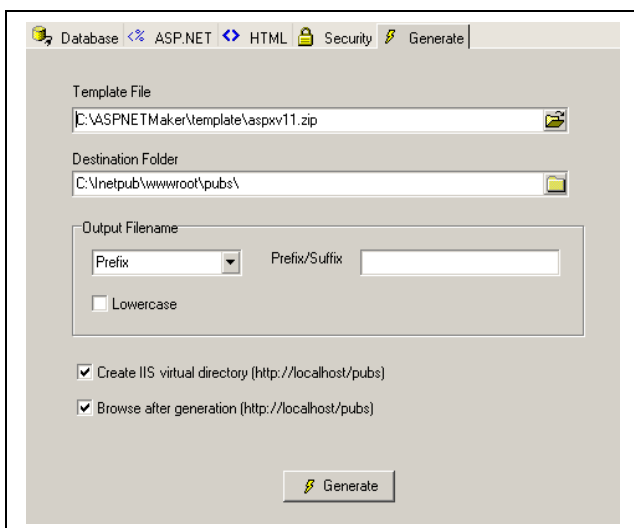
Hình A-16 Thẻ ASP.NET | .NET Specific Options

- Kế tiếp, vào thẻ *Security*, đánh dấu chọn *Administrator Login*, nhập *Login Name* và *Password*. Đây là tài khoản quản trị (sẽ được viết mã cứng).



Hình A-17 Thẻ Security

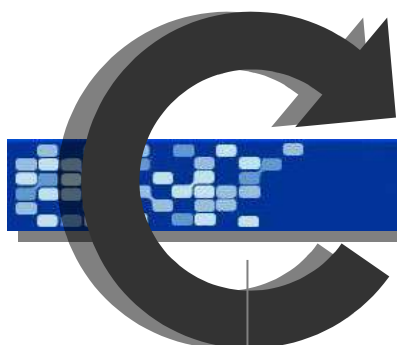
- Cuối cùng, vào thẻ *Generate* và nhấp nút *Generate*. Sau khi xây dựng thành công, bạn hãy vào <http://localhost/pubs> để xem kết quả.



Hình A-18 Thẻ Generate



ASP.NET Maker được phát triển bởi *e.World* và có thể được download (bản dùng thử) tại [<http://www.hkvstore.com>].



PHỤ LỤC B

THUẬT NGỮ ANH – VIỆT

Absolute [<i>adj</i>]	Tuyệt đối
Abstract [<i>adj</i>]	Trừu tượng
Access [<i>v</i>]	Truy xuất
Access modifier	Từ khóa thay đổi tầm truy xuất
Accessibility	Khả năng truy xuất
Account	Tài khoản
ACL [Access Control List]	Danh sách điều khiển truy xuất
Administrator	Người quản trị
Aggregate function	Hàm tập hợp
Algorithm	Giải thuật
API [Application Programming Interface]	Giao diện lập trình ứng dụng
Application	Ứng dụng
Application domain	Miền ứng dụng
Argument	Đối số
Arithmetic	Số học
Array	Mảng
Assembly	Gói kết hợp
Asymmetric [<i>adj</i>]	Bất đối xứng
Asynchronous [<i>adj</i>]	Bất đồng bộ
Attribute	Đặc tính
Authentication	Sự xác thực
Authorization	Sự phân quyền
Availability	Tính khả dụng
Binary	Nhị phân
Block	Khối
Bound	Cận
Boundary	Đường biên / Ranh giới
Breakpoint	Điểm dừng

Browser.....	Trình duyệt
Buffer.....	Bộ đệm
Built-in [<i>adj</i>].....	Nội tại
Cache.....	Kho chứa (truy xuất nhanh)
Caching.....	Cơ chế lưu giữ
CAS [Code Access Security].....	Bảo mật truy xuất mã lệnh
Certificate.....	Chứng chỉ
Channel.....	Kênh
Character.....	Ký tự
Class.....	Lớp
Client.....	Trình khách
Clone [<i>v</i>].....	Sao chép
Cloneable [<i>adj</i>].....	Khả sao chép
CLR [Common Language Runtime].....	Bộ thực thi ngôn ngữ chung
Code.....	Mã lệnh
Collection.....	Tập hợp
Column.....	Cột
Command.....	Lệnh
Communication.....	Sự giao tiếp
Comparable [<i>adj</i>].....	Khả so sánh
Compare [<i>v</i>].....	So sánh
Compatibility.....	Tính tương thích
Compile [<i>v</i>].....	Biên dịch
Compiler.....	Trình biên dịch
Component.....	Thành phần
Component tray.....	Khay thành phần
Configuration.....	Cấu hình
Connection.....	Kết nối
Constant.....	Hằng
Constructor.....	Phương thức khởi dựng
Context.....	Ngữ cảnh
Context-sensitive help.....	Trợ giúp cảm-ngữ-cảnh
Control.....	Điều khiển
Convert [<i>v</i>].....	Chuyển đổi
Convertible [<i>adj</i>].....	Khả chuyển đổi
Cryptography.....	Mật mã
Culture.....	Miền văn hóa
Custom [<i>adj</i>].....	Tùy biến
Data.....	Dữ liệu
Data binding.....	Kỹ thuật kết dữ liệu
Database.....	Cơ sở dữ liệu
De-compile [<i>v</i>].....	Dịch ngược

De-serialize [v].....	Giải tuần tự hóa
Decrypt [v].....	Giải mật hóa
Decryption.....	Sự giải mật hóa
Debug [v].....	Gỡ rối
Debugger.....	Trình gỡ rối
Default.....	Mặc định
Delegate.....	Ủy nhiệm hàm
Deploy [v].....	Triển khai
Destructor.....	Phương thức hủy
Device.....	Thiết bị
Derive [v].....	Dẫn xuất
Dictionary.....	Từ điển
Digital signature.....	Chữ ký số
Directive.....	Chỉ thị
Directory.....	Thư mục
Disposable [adj].....	Khả hủy
Dispose [v].....	Hủy
Distributed [adj].....	Có tính phân tán
Document.....	Tài liệu
Domain.....	Miền
Edit [v].....	Hiệu chỉnh
Editor.....	Trình soạn thảo
Encapsulation.....	Sự đóng gói
Encode [v].....	Mã hóa
Encoding.....	Phép mã hóa
Encrypt [v].....	Mật hóa
Encryption.....	Sự mật hóa
Entry.....	Khoản mục
Enumeration.....	Kiểu liệt kê
Environment.....	Môi trường
Error.....	Lỗi
Event.....	Sự kiện
Event handler.....	Phương thức thụ lý sự kiện
Event log.....	Nhật ký sự kiện
Evidence.....	Chứng cứ
Exception.....	Biệt lệ
Exception handler.....	Phương thức thụ lý biệt lệ
Expiration.....	Sự hết hiệu lực
Export [v].....	Xuất
Expression.....	Biểu thức
Feature.....	Tính năng
Field.....	Trường
File.....	Tập tin
Filter.....	Bộ lọc
Flag.....	Cờ

Flexibility.....	Tính linh hoạt
Form.....	Biểu mẫu
Format.....	Định dạng
FTP [File Transfer Protocol].....	Giao thức truyền file
Function.....	Hàm
Functionality.....	Chức năng
GAC [Global Assembly Cache].....	Kho chứa gói kết hợp toàn cục
GC [Garbage Collector].....	Bộ thu gom rác
Generalization.....	Tính tổng quát hóa
Global [<i>adj</i>].....	Toàn cục
Globalization.....	Sự toàn cầu hóa
Graphics.....	Đồ họa
Group.....	Nhóm
GUI [Graphical User Interface].....	Giao diện người dùng đồ họa
GUID [Globally Unique Identifier].....	Định danh duy nhất toàn cục
Handle [<i>v</i>].....	Thụ lý
Handle.....	Mục quản
Hash [<i>v</i>].....	Băm
Hash code.....	Mã băm
Hashtable.....	Bảng băm
Help.....	Trợ giúp
HTML [HyperText Markup Language].....	Ngôn ngữ đánh dấu siêu văn bản
Hyperlink.....	Siêu liên kết
IDE [Integrated Development Environment].....	Môi trường phát triển tích hợp
Identifier.....	Diện từ
Imperson [<i>v</i>].....	Giả nhận
Impersonation.....	Sự giả nhận
Implement [<i>v</i>].....	Hiện thực
Implementation.....	Bản hiện thực
Import [<i>v</i>].....	Nhập
Index.....	Chỉ mục
Indexer.....	Bộ chỉ mục
Inheritance.....	Sự thừa kế
Initialize [<i>v</i>].....	Khởi tạo
Input.....	Đầu vào / Dữ liệu nhập
Insert [<i>v</i>].....	Chèn
Install [<i>v</i>].....	Cài đặt
Instance.....	Thẻ hiện
Integration.....	Sự tích hợp
Interface.....	Giao diện
Interoperability.....	Khả năng liên tác
IP [Internet Protocol].....	Giao thức Internet

Item.....	Mục chọn
JIT [just-in-time].....	Tức thời / Vừa đúng lúc
Key.....	Khóa
Keyword.....	Từ khóa
Language.....	Ngôn ngữ
Length.....	Chiều dài
Library.....	Thư viện
Lifetime.....	Thời gian sống
Link.....	Liên kết
List.....	Danh sách
Literal.....	Trực kiện
Load [v].....	Nạp
Local [adj].....	Cục bộ
Locale.....	Bản địa
Localization.....	Sự bản địa hóa
Lock.....	Chốt
Logic.....	Mã thi hành chức năng
Loop.....	Vòng lặp
Managed [adj].....	Được quản lý
Management.....	Sự quản lý
Mapping.....	Phép ánh xạ
Member.....	Thành viên
Membership.....	Tư cách thành viên
Memory.....	Bộ nhớ
Menu.....	Trình đơn
Message.....	Thông điệp
Metacharacter.....	Siêu ký tự
Metadata.....	Siêu dữ liệu
Method.....	Phương thức
Model.....	Mô hình
Module.....	Đơn thể
MSIL [Microsoft Intermediate Language].....	Ngôn ngữ trung gian
Multilingual [adj].....	Đa ngôn ngữ
Multithreading.....	Lập trình đa tiểu trình
Native [adj].....	Nguyên sinh
Namespace.....	Không gian tên
Network.....	Mạng
Node.....	Nút
Object.....	Đối tượng
Object-oriented programming.....	Lập trình hướng đối tượng
Operating system.....	Hệ điều hành
Operator.....	Toán tử
Output.....	Đầu ra / Kết xuất
Overload.....	Bản nạp chồng
Override [v].....	Chép đè

Parameter.....	Thông số
Password.....	Mật khẩu
Path.....	Đường dẫn
Pattern.....	Kiểu mẫu
Performance.....	Hiệu năng
Permission.....	Quyền
Pixel.....	Điểm ảnh
Platform.....	Nền
Pointer.....	Con trỏ
Policy.....	Chính sách
Polymorphisme.....	Tính đa hình
Pool.....	Kho dự trữ
Pooling.....	Cơ chế dự trữ
POP3 [Post Office Protocol 3].....	Giao thức nhận mail 3
Port.....	Cổng
Postfix.....	Hậu tố
Prefix.....	Tiền tố
Private [<i>adj</i>].....	Riêng
Privilege.....	Đặc quyền
Procedure.....	Thủ tục
Process.....	Tiến trình
Processor.....	Bộ xử lý
Project.....	Dự án
Property.....	Thuộc tính
Protected [<i>adj</i>].....	Được bảo vệ
Protocol.....	Giao thức
Public [<i>adj</i>].....	Công khai
Query.....	Truy vấn
Queue.....	Hàng đợi
Random.....	Ngẫu nhiên
RBS [Role-Based Security].....	Bảo mật dựa-trên-vai-trò
Record.....	Bản ghi / Mẫu tin
Recursion.....	Sự đệ quy
Reference.....	Tham chiếu
Reflection.....	Cơ chế phản chiếu
Register [<i>v</i>].....	Đăng ký
Regular expression.....	Biểu thức chính quy
Relationship.....	Mối quan hệ
Relative [<i>adj</i>].....	Tương đối
Remotable [<i>adj</i>].....	Khả truy xuất từ xa
Resource.....	Tài nguyên
Reusability.....	Khả năng tái sử dụng

Role.....	Vai trò
Routine.....	Thường trình
Row.....	Hàng / Dòng
Runtime.....	Bộ thực thi
Schema.....	Lược đồ / Khuôn
Script.....	Kịch bản
Security.....	Sự bảo mật
Serialize [v].....	Tuần tự hóa
Serializable [<i>adj</i>].....	Khả tuần tự hóa
Serialization.....	Sự tuần tự hóa
Server.....	Trình chủ
Service.....	Dịch vụ
Session.....	Phiên làm việc
Setting.....	Thiết lập
Shared [<i>adj</i>].....	Được chia sẻ / Dùng chung
Signature.....	Chữ ký
SMTP [Simple Mail Transfer Protocol].....	Giao thức truyền mail đơn giản
SOAP [Simple Object Access Protocol].....	Giao thức truy xuất đối tượng đơn giản
Solution.....	Giải pháp
Specialization.....	Tính chuyên hóa
SQL [Structured Query Language].....	Ngôn ngữ truy vấn có cấu trúc
Stack.....	Ngăn chồng
State.....	Trạng thái
State Stateless [<i>adj</i>].....	Có trạng thái Phi trạng thái
Statement.....	Câu lệnh / Khai báo
Static [<i>adj</i>].....	Tĩnh
Stored procedure.....	Thủ tục tồn trữ
Stream.....	Dòng chảy
String.....	Chuỗi
Strong name.....	Tên mạnh
Strong type.....	Kiểu mạnh
Strongly-named [<i>adj</i>].....	Được định tên mạnh
Strongly-typed [<i>adj</i>].....	Được định kiểu mạnh
Structure.....	Cấu trúc
Symmetric [<i>adj</i>].....	Đối xứng
Synchronization.....	Sự đồng bộ hóa
Synchronous [<i>adj</i>].....	Đồng bộ
System.....	Hệ thống
System tray.....	Khay hệ thống
Table.....	Bảng
Tag.....	Thẻ
Task.....	Tác vụ
Template.....	Khuôn mẫu
Thread.....	Tiểu trình / Mạch trình / Tuyến đoạn
Thread-safe.....	An toàn về tiểu trình

Throw [v].....	Ném
Timestamp.....	Tem thời gian
Tool.....	Công cụ
Toolbox.....	Hộp công cụ
Transaction.....	Phiên giao dịch
Type.....	Kiểu
Type-safe.....	An toàn về kiểu dữ liệu
Unmanaged [<i>adj</i>].....	Không được quản lý
Update.....	Cập nhật
URI [Uniform Resource Identifier].....	Bộ nhận dạng tài nguyên đồng dạng
URL [Uniform Resource Locator].....	Bộ định vị tài nguyên đồng dạng
User.....	Người dùng
Utility.....	Tiện ích
Validation.....	Sự xác nhận tính hợp lệ
Value.....	Giá trị
Variable.....	Biến
Version.....	Phiên bản
Virtual [<i>adj</i>].....	Ảo
Visible [<i>adj</i>].....	Khả kiến
Visual [<i>adj</i>].....	Trực quan
Wildcard.....	Ký tự đại diện
Window.....	Cửa sổ
Worker.....	Thợ
Wrapper.....	Vỏ bọc
WSDL [Web Services Description Language].....	Ngôn ngữ mô tả dịch vụ Web
XML [Extensible Markup Language].....	Ngôn ngữ đánh dấu mở rộng

TÀI LIỆU THAM KHẢO

CÁC GIẢI PHÁP LẬP TRÌNH C#

- [1] Allen Jones. *C# Programmer's Cookbook*. Microsoft Press, 2004.
- [2] John Connell. *Coding Techniques for Microsoft Visual Basic .NET*. Microsoft Press, 2002.
- [3] Harvey M. Deitel, Paul J. Deitel, & Tem R. Nieto. *Visual Basic .NET How to Program, Second Edition*. Prentice Hall, 2002.
- [4] Jose Mojica. *C# and VB .NET Conversion Pocket Reference*. O'Reilly, 2002.
- [5] James Avery. *Ten Must-Have Tools Every Developer Should Download Now*. MSDN Magazine, 2004.
- [6] Karl Moore. *The Ultimate VB.NET and ASP.NET Code Book*. Apress, 2003.
- [7] Matthew MacDonald. *Microsoft Visual Basic .NET Programmer's Cookbook*. Microsoft Press, 2003.
- [8] Mark Schmidt & Simon Robinson. *Microsoft Visual C# .NET 2003 Developer's Cookbook*. Sams Publishing, 2003.
- [9] Dương Quang Thiện. *Lập trình Visual C# thế nào?* Nhà xuất bản Tổng Hợp TP. Hồ Chí Minh, 2005.
- [10] Nguyễn Ngọc Bình Phương & Lê Trần Nhật Quỳnh. *Các giải pháp lập trình Visual Basic .NET*. Nhà xuất bản Giao thông Vận tải, 2006.
- [11] Thái Thanh Phong & Nguyễn Ngọc Bình Phương. *Sổ tay tra cứu VB.NET*. Nhà xuất bản Giao thông Vận tải, 2004.
- [12] Và một số website như www.msdn.microsoft.com, www.codeproject.com, www.msd2d.com, www.developersdex.com, www.windowsforms.net, www.gotdotnet.com, www.codeguru.com, www.developerfusion.com, v.v...

CÁC GIẢI PHÁP LẬP TRÌNH C#

Nguyễn Ngọc Bình Phương – Thái Thanh Phong
tổng hợp & biên dịch

Chịu trách nhiệm xuất bản
TS. Nguyễn Xuân Thủy

Biên tập
Hồ Nguyễn Thị Thanh Thúy

Trình bày bìa
Nguyễn Thị Thanh Thủy

Chế bản & Sửa bản in
Nguyễn Ngọc Bình Phương