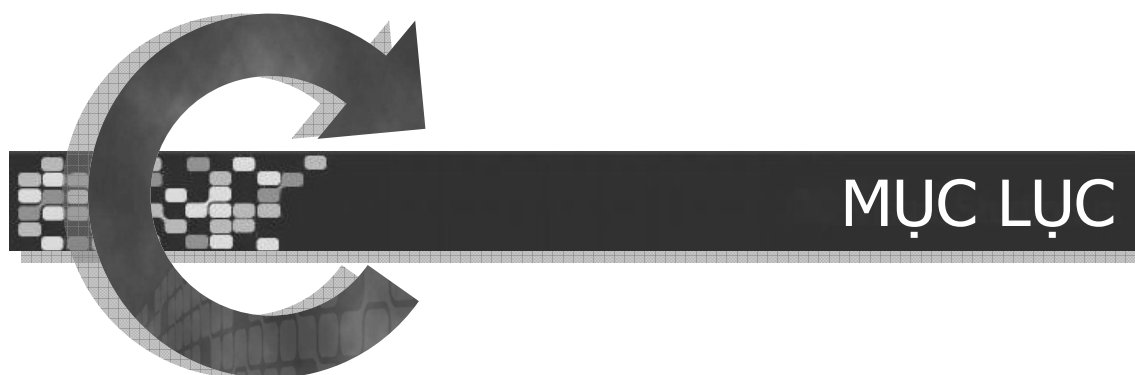




Nguyễn Ngọc Bình Dương - Thái Thanh Phong
tổng hợp & biên dịch

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

NHÀ XUẤT BẢN GIAO THÔNG VẬN TẢI



LỜI NÓI ĐẦU.....	5
CÁU TRÚC CỦA SÁCH.....	7
QUY ƯỚC.....	9
YÊU CẦU VỀ HỆ THỐNG	11
CÁCH SỬ DỤNG ĐĨA CD.....	13
MỤC LỤC	15
Chương 1: PHÁT TRIỂN ỨNG DỤNG	23
1.1 Tạo ứng dụng Console	25
1.2 Tạo ứng dụng dựa-trên-Windows	27
1.3 Tạo và sử dụng module	30
1.4 Tạo và sử dụng thư viện.....	31
1.5 Truy xuất các đối số dòng lệnh	32
1.6 Chọn biên dịch một khối mã vào file thực thi	33
1.7 Truy xuất một phần tử chương trình có tên trùng với một từ khóa.....	36
1.8 Tạo và quản lý cặp khóa tên mạnh	37
1.9 Tạo tên mạnh cho assembly	38
1.10 Xác minh một assembly tên mạnh không bị sửa đổi.....	39
1.11 Hoãn việc ký assembly	40
1.12 Ký assembly với chữ ký số Authenticode	42
1.13 Tạo và thiết lập tin tưởng một SPC thử nghiệm.....	44

1.14 Quản lý Global Assembly Cache	46
1.15 Ngăn người khác dịch ngược mã nguồn của bạn.....	46
Chương 2: THAO TÁC DỮ LIỆU	49
2.1 Thao tác chuỗi một cách hiệu quả	51
2.2 Mã hóa chuỗi bằng các kiểu mã hóa ký tự.....	52
2.3 Chuyển các kiểu giá trị cơ bản thành mảng kiểu byte.....	54
2.4 Mã hóa dữ liệu nhị phân thành văn bản.....	56
2.5 Sử dụng biểu thức chính quy để kiểm tra dữ liệu nhập	57
2.6 Sử dụng biểu thức chính quy đã được biên dịch	60
2.7 Tạo ngày và giờ từ chuỗi.....	62
2.8 Cộng, trừ, so sánh ngày giờ	63
2.9 Sắp xếp một mảng hoặc một ArrayList.....	64
2.10 Chép một tập hợp vào một mảng	65
2.11 Tạo một tập hợp kiểu mạnh.....	66
2.12 Lưu một đối tượng khả-tuần-tự-hóa vào file	67
Chương 3: MIỀN ỨNG DỤNG, CƠ CHẾ PHẢN CHIẾU, VÀ SIÊU DỮ LIỆU	71
3.1 Tạo miền ứng dụng	73
3.2 Chuyển các đối tượng qua lại các miền ứng dụng.....	74
3.3 Tránh nạp các assembly không cần thiết vào miền ứng dụng	75
3.4 Tạo kiểu không thể vượt qua biên miền ứng dụng	76
3.5 Nạp assembly vào miền ứng dụng hiện hành.....	76
3.6 Thực thi assembly ở miền ứng dụng khác.....	78
3.7 Thể hiện hóa một kiểu trong miền ứng dụng khác.....	79
3.8 Truyền dữ liệu giữa các miền ứng dụng	83
3.9 Giải phóng assembly và miền ứng dụng.....	84
3.10 Truy xuất thông tin Type	85
3.11 Kiểm tra kiểu của một đối tượng.....	86
3.12 Tạo một đối tượng bằng cơ chế phản chiếu	87
3.13 Tạo một đặc tính tùy biến	90
3.14 Sử dụng cơ chế phản chiếu để kiểm tra các đặc tính của một phần tử chương trình	91
Chương 4: TIỂU TRÌNH, TIẾN TRÌNH, VÀ SỰ ĐỒNG BỘ	93
4.1 Thực thi phương thức với thread-pool.....	95
4.2 Thực thi phương thức một cách bất đồng bộ.....	98
4.3 Thực thi phương thức bằng Timer.....	104
4.4 Thực thi phương thức bằng cách ra hiệu đối tượng WaitHandle	106
4.5 Thực thi phương thức bằng tiểu trình mới	108
4.6 Điều khiển quá trình thực thi của một tiểu trình	109
4.7 Nhận biết khi nào một tiểu trình kết thúc.....	112
4.8 Đồng bộ hóa quá trình thực thi của nhiều tiểu trình	114
4.9 Tạo một đối tượng tập hợp có tính chất an-toàn-về-tiểu-trình	117
4.10 Khởi chạy một tiến trình mới.....	119
4.11 Kết thúc một tiến trình.....	120
4.12 Bảo đảm chỉ có thể chạy một thể hiện của ứng dụng tại một thời điểm	122

Chương 5: XML	125
5.1 Hiển thị cấu trúc của một tài liệu XML trong TreeView.....	127
5.2 Chèn thêm nút vào tài liệu XML.....	131
5.3 Chèn thêm nút vào tài liệu XML một cách nhanh chóng.....	132
5.4 Tìm một nút khi biết tên của nó.....	134
5.5 Thu lấy các nút XML trong một không gian tên XML cụ thể.....	135
5.6 Tìm các phần tử với biểu thức XPath.....	136
5.7 Đọc và ghi XML mà không phải nạp toàn bộ tài liệu vào bộ nhớ.....	139
5.8 Xác nhận tính hợp lệ của một tài liệu XML dựa trên một Schema.....	141
5.9 Sử dụng XML Serialization với các đối tượng tùy biến.....	145
5.10 Tạo XML Schema cho một lớp .NET.....	148
5.11 Tạo lớp từ một XML Schema.....	149
5.12 Thực hiện phép biến đổi XSL.....	149
Chương 6: WINDOWS FORM	153
6.1 Thêm điều kiểm vào form lúc thực thi.....	155
6.2 Liên kết dữ liệu vào điều kiểm.....	157
6.3 Xử lý tất cả các điều kiểm trên form.....	158
6.4 Theo vết các form khả kiến trong một ứng dụng.....	159
6.5 Tìm tất cả các form trong ứng dụng MDI.....	160
6.6 Lưu trữ kích thước và vị trí của form.....	161
6.7 Buộc ListBox cuộn xuống.....	163
6.8 Chỉ cho phép nhập số vào TextBox.....	164
6.9 Sử dụng ComboBox có tính năng auto-complete.....	165
6.10 Sắp xếp ListView theo cột bất kỳ.....	167
6.11 Liên kết menu ngữ cảnh vào điều kiểm.....	169
6.12 Sử dụng một phần menu chính cho menu ngữ cảnh.....	169
6.13 Tạo form đa ngôn ngữ.....	171
6.14 Tạo form không thể di chuyển được.....	173
6.15 Làm cho form không đường viền có thể di chuyển được.....	174
6.16 Tạo một icon động trong khay hệ thống.....	176
6.17 Xác nhận tính hợp lệ của đầu vào cho một điều kiểm.....	177
6.18 Thực hiện thao tác kéo-và-thả.....	178
6.19 Sử dụng trợ giúp cảm-ngữ-cảnh.....	180
6.20 Áp dụng phong cách Windows XP.....	181
6.21 Thay đổi độ đục của form.....	183
Chương 7: ASP.NET VÀ WEB FORM	185
7.1 Chuyển hướng người dùng sang trang khác.....	187
7.2 Duy trì trạng thái giữa các yêu cầu của trang.....	188
7.3 Tạo các biến thành viên có trạng thái cho trang.....	192
7.4 Đáp ứng các sự kiện phía client với JavaScript.....	193
7.5 Hiển thị cửa sổ pop-up với JavaScript.....	195
7.6 Thiết lập focus cho điều kiểm.....	197
7.7 Cho phép người dùng upload file.....	197
7.8 Sử dụng IIS authentication.....	200

7.9 Sử dụng Forms authentication.....	203
7.10 Thực hiện xác nhận tính hợp lệ có-chọn-lựa	206
7.11 Thêm động điều khiển vào Web Form	207
7.12 Trả về động một bức hình.....	210
7.13 Nạp điều khiển người dùng bằng mã lệnh	213
7.14 Sử dụng page-caching và fragment-caching	216
7.15 Dùng lại dữ liệu với ASP.NET Cache	217
7.16 Kích hoạt việc gỡ rối ứng dụng Web	219
7.17 Thay đổi quyền đã cấp cho mã ASP.NET	222
Chương 8: ĐỒ HỌA, ĐA PHƯƠNG TIỆN, VÀ IN ÁN	225
8.1 Tìm tất cả các font đã được cài đặt	227
8.2 Thực hiện “hit testing” với shape	228
8.3 Tạo form có hình dạng tùy biến	231
8.4 Tạo điều khiển có hình dạng tùy biến	233
8.5 Thêm tính năng cuộn cho một bức hình	236
8.6 Thực hiện chụp màn hình Desktop	237
8.7 Sử dụng “double buffering” để tăng tốc độ vẽ lại	238
8.8 Hiển thị hình ở dạng thumbnail.....	241
8.9 Phát tiếng “beep” của hệ thống.....	242
8.10 Chơi file audio.....	243
8.11 Chơi file video.....	245
8.12 Lấy thông tin về các máy in đã được cài đặt.....	247
8.13 In văn bản đơn giản.....	249
8.14 In văn bản có nhiều trang	252
8.15 In text dạng wrapping	254
8.16 Hiển thị print-preview	256
8.17 Quản lý tác vụ in	258
8.18 Sử dụng Microsoft Agent	262
Chương 9: FILE, THƯ MỤC, VÀ I/O	267
9.1 Truy xuất các thông tin về file hay thư mục.....	269
9.2 Thiết lập các thuộc tính của file và thư mục.....	272
9.3 Chép, chuyển, xóa file hay thư mục.....	273
9.4 Tính kích thước của thư mục.....	275
9.5 Truy xuất thông tin phiên bản của file	276
9.6 Sử dụng TreeView để hiển thị cây thư mục just-in-time.....	277
9.7 Đọc và ghi file văn bản	279
9.8 Đọc và ghi file nhị phân	281
9.9 Đọc file một cách bất đồng bộ	282
9.10 Tìm file phù hợp một biểu thức wildcard.....	284
9.11 Kiểm tra hai file có trùng nhau hay không	285
9.12 Thao tác trên đường dẫn file	286
9.13 Xác định đường dẫn tương ứng với một file hay thư mục	287
9.14 Làm việc với đường dẫn tương đối	288
9.15 Tạo file tạm.....	289
9.16 Lấy dung lượng đĩa còn trống.....	290

9.17	Hiện thị các hộp thoại file.....	291
9.18	Sử dụng không gian lưu trữ riêng.....	293
9.19	Theo dõi hệ thống file để phát hiện thay đổi.....	295
9.20	Truy xuất cổng COM.....	296
Chương 10: CƠ SỞ DỮ LIỆU		299
10.1	Kết nối cơ sở dữ liệu.....	302
10.2	Sử dụng connection-pooling.....	304
10.3	Thực thi câu lệnh SQL hoặc thủ tục tồn trữ.....	306
10.4	Sử dụng thông số trong câu lệnh SQL hoặc thủ tục tồn trữ.....	309
10.5	Xử lý kết quả của truy vấn SQL bằng data-reader.....	311
10.6	Thu lấy tài liệu XML từ truy vấn SQL Server.....	314
10.7	Nhận biết tất cả các thể hiện SQL Server 2000 trên mạng.....	316
10.8	Đọc file Excel với ADO.NET.....	318
10.9	Sử dụng Data Form Wizard.....	319
10.10	Sử dụng Crystal Report Wizard.....	326
Chương 11: LẬP TRÌNH MẠNG		333
11.1	Download file thông qua HTTP.....	335
11.2	Download và xử lý file bằng stream.....	336
11.3	Lấy trang HTML từ một website có yêu cầu xác thực.....	337
11.4	Hiện thị trang web trong ứng dụng dựa-trên-Windows.....	339
11.5	Lấy địa chỉ IP của máy tính hiện hành.....	341
11.6	Phân giải tên miền thành địa chỉ IP.....	342
11.7	"Ping" một địa chỉ IP.....	343
11.8	Giao tiếp bằng TCP.....	345
11.9	Lấy địa chỉ IP của client từ kết nối socket.....	349
11.10	Thiết lập các tùy chọn socket.....	350
11.11	Tạo một TCP-server hỗ trợ đa-tiểu-trình.....	351
11.12	Sử dụng TCP một cách bất đồng bộ.....	353
11.13	Giao tiếp bằng UDP.....	356
11.14	Gửi e-mail thông qua SMTP.....	358
11.15	Gửi và nhận e-mail với MAPI.....	359
Chương 12: DỊCH VỤ WEB XML VÀ REMOTING		361
12.1	Tránh viết mã cứng cho địa chỉ URL của dịch vụ Web XML.....	364
12.2	Sử dụng kỹ thuật response-caching trong dịch vụ Web XML.....	365
12.3	Sử dụng kỹ thuật data-caching trong dịch vụ Web XML.....	366
12.4	Tạo phương thức web hỗ trợ giao dịch.....	367
12.5	Thiết lập thông tin xác thực cho dịch vụ Web XML.....	369
12.6	Gọi bất đồng bộ một phương thức web.....	370
12.7	Tạo lớp khả-truy-xuất-từ-xa.....	372
12.8	Đăng ký tất cả các lớp khả-truy-xuất-từ-xa trong một assembly.....	376
12.9	Quản lý các đối tượng ở xa trong IIS.....	378
12.10	Phát sinh sự kiện trên kênh truy xuất từ xa.....	379
12.11	Kiểm soát thời gian sống của một đối tượng ở xa.....	382
12.12	Kiểm soát phiên bản của các đối tượng ở xa.....	384

12.13 Tạo phương thức một chiều với dịch vụ Web XML hay Remoting	385
Chương 13: BẢO MẬT	387
13.1 Cho phép mã lệnh có-độ-tin-cậy-một-phần sử dụng assembly tên mạnh của bạn..	390
13.2 Vô hiệu bảo mật truy xuất mã lệnh	392
13.3 Vô hiệu việc kiểm tra quyền thực thi	393
13.4 Bảo đảm bộ thực thi cấp cho assembly một số quyền nào đó	394
13.5 Giới hạn các quyền được cấp cho assembly	396
13.6 Xem các yêu cầu quyền được tạo bởi một assembly	397
13.7 Xác định mã lệnh có quyền nào đó lúc thực thi hay không	399
13.8 Hạn chế ai đó thừa kế các lớp của bạn và chép đè các thành viên lớp	400
13.9 Kiểm tra chứng cứ của một assembly	401
13.10 Xử lý chứng cứ khi nạp một assembly.....	402
13.11 Xử lý bảo mật bộ thực thi bằng chứng cứ của miền ứng dụng	404
13.12 Xử lý bảo mật bộ thực thi bằng chính sách bảo mật của miền ứng dụng	406
13.13 Xác định người dùng hiện hành có là thành viên của một nhóm Windows nào đó hay không	409
13.14 Hạn chế những người dùng nào đó thực thi mã lệnh của bạn.....	412
13.15 Giả nhận người dùng Windows	415
Chương 14: MẬT MÃ	419
14.1 Tạo số ngẫu nhiên	421
14.2 Tính mã băm của password.....	422
14.3 Tính mã băm của file	424
14.4 Kiểm tra mã băm	425
14.5 Bảo đảm tính toàn vẹn dữ liệu bằng mã băm có khóa.....	427
14.6 Bảo vệ file bằng phép mật hóa đối xứng	429
14.7 Truy lại khóa đối xứng từ password.....	434
14.8 Gửi một bí mật bằng phép mật hóa bất đối xứng	435
14.9 Lưu trữ khóa bất đối xứng một cách an toàn.....	440
14.10 Trao đổi khóa phiên đối xứng một cách an toàn	442
Chương 15: KHẢ NĂNG LIÊN TÁC MÃ LỆNH KHÔNG-ĐƯỢC-QUẢN-LÝ	447
15.1 Gọi một hàm trong một DLL không-được-quản-lý.....	449
15.2 Lấy handle của một điều kiểm, cửa sổ, hoặc file	452
15.3 Gọi một hàm không-được-quản-lý có sử dụng cấu trúc.....	453
15.4 Gọi một hàm không-được-quản-lý có sử dụng callback	455
15.5 Lấy thông tin lỗi không-được-quản-lý	456
15.6 Sử dụng thành phần COM trong .NET-client	457
15.7 Giải phóng nhanh thành phần COM	459
15.8 Sử dụng thông số tùy chọn.....	460
15.9 Sử dụng điều kiểm ActiveX trong .NET-client	461
15.10 Tạo thành phần .NET dùng cho COM-client	462
Chương 16: CÁC GIAO DIỆN VÀ MẪU THÔNG DỤNG	463
16.1 Hiện thực kiểu khả-tuần-tự-hóa (serializable type)	465
16.2 Hiện thực kiểu khả-sao-chép (cloneable type).....	470
16.3 Hiện thực kiểu khả-so-sánh (comparable type)	473


16.4 Hiện thực kiểu khả-liệt-kê (enumerable type)	476
16.5 Hiện thực lớp khả-hủy (disposable class)	481
16.6 Hiện thực kiểu khả-định-dạng (formattable type)	484
16.7 Hiện thực lớp ngoại lệ tùy biến	486
16.8 Hiện thực đối số sự kiện tùy biến.....	489
16.9 Hiện thực mẫu Singleton	490
16.10 Hiện thực mẫu Observer.....	491
Chương 17: SỰ HÒA HỢP VỚI MÔI TRƯỜNG WINDOWS	497
17.1 Truy xuất thông tin môi trường.....	499
17.2 Lấy giá trị của một biến môi trường	502
17.3 Ghi một sự kiện vào nhật ký sự kiện Windows	503
17.4 Truy xuất Windows Registry	504
17.5 Tạo một dịch vụ Windows.....	507
17.6 Tạo một bộ cài đặt dịch vụ Windows	511
17.7 Tạo shortcut trên Desktop hay trong Start menu.....	513
PHỤ LỤC A: GIỚI THIỆU MỘT SỐ CÔNG CỤ .NET	517
A.1 Biên dịch các đoạn mã ngắn với Snippet Compiler	517
A.2 Xây dựng biểu thức chính quy với Regulator	518
A.3 Sinh mã với CodeSmith.....	519
A.4 Viết kiểm thử đơn vị với NUnit.....	522
A.5 Kiểm soát mã lệnh với FxCop	523
A.6 Khảo sát assembly với .NET Reflector	524
A.7 Lập tài liệu mã lệnh với NDoc	526
A.8 Tạo dựng giải pháp với NAnt	528
A.9 Chuyển đổi phiên bản ASP.NET với ASP.NET Version Switcher.....	530
A.10 Chuyển đổi phiên bản dự án với Visual Studio .NET Project Converter	530
A.11 Chuyển mã nguồn VB.NET sang C# với VB.NET to C# Converter	531
A.12 Chuyển mã nguồn C# sang VB.NET với Convert C# to VB.NET.....	532
A.13 Xây dựng website quản trị cơ sở dữ liệu với ASP.NET Maker 1.1	532
PHỤ LỤC B: THUẬT NGỮ ANH - VIỆT	535
TÀI LIỆU THAM KHẢO	543

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*.

1.1

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.

1.2

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.label1 = 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.label1.Location = new System.Drawing.Point(16, 36);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(128, 16);
        this.label1.TabIndex = 0;
        this.label1.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.label1);
        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 thể 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());

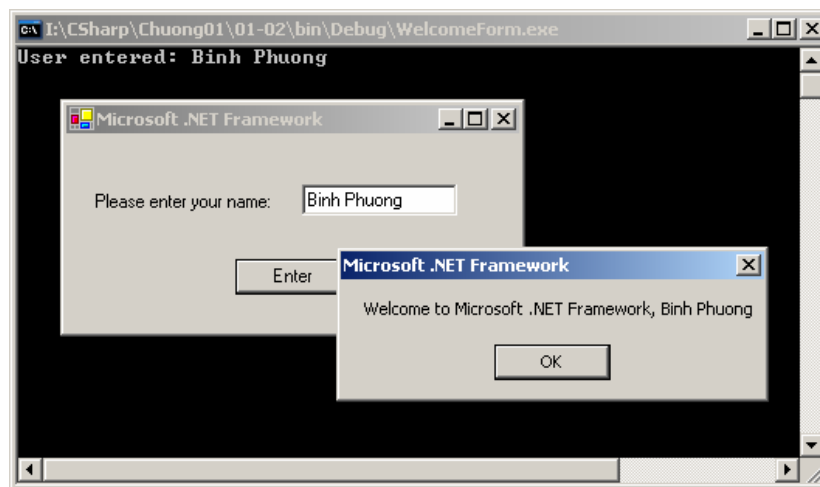
}

// 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
```

Đối 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 đối 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.

1.3

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).

1.4

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).

1.5

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.

1.6

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

1.7

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;
```

1.8

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
07020000002400005253413200040000010001008bb302ef9180bf717ace00d570dd649821f24ed57
8fdccf1bc4017308659c126570204bc4010fdd1907577df1c2292349d9c2de33e49bd991a0a5bc9b6
9e5fd95bafad658a57b8236c5bd9a43be022a20a52c2bd8145448332d5f85e9ca641c26a4036165f2
f353942b643b10db46c82d6d77bbc210d5a7c5aca84d7acb52cc1654759c62aa34988...
```

```
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 ngắn 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`.

1.9

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*.**

1.10

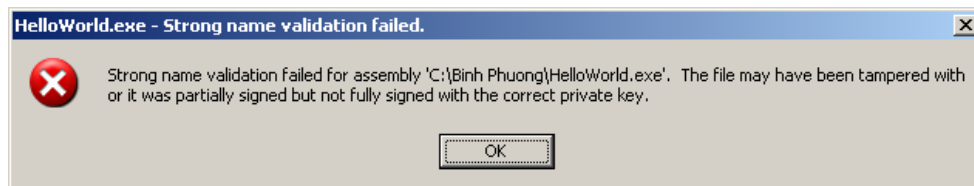
Xác minh một assembly tên mạnh không bị sửa đổi

- ?** 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
```

1.11

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 nguy cơ 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");


    }

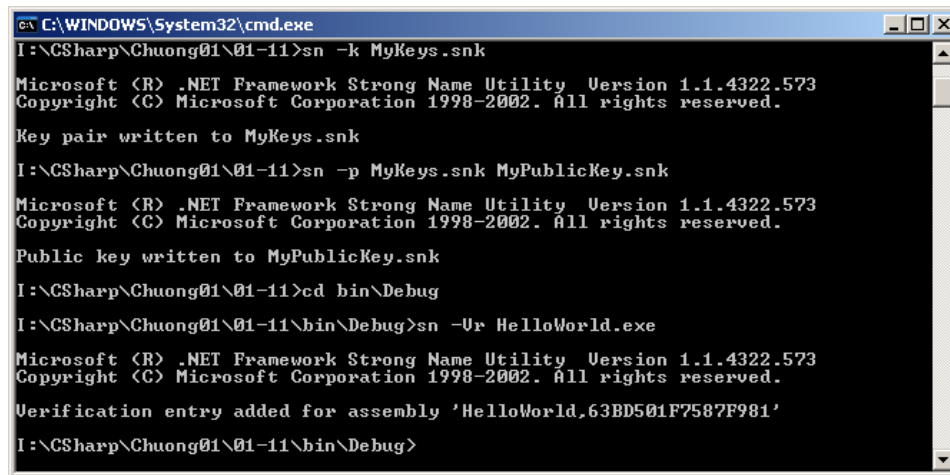
}
```

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

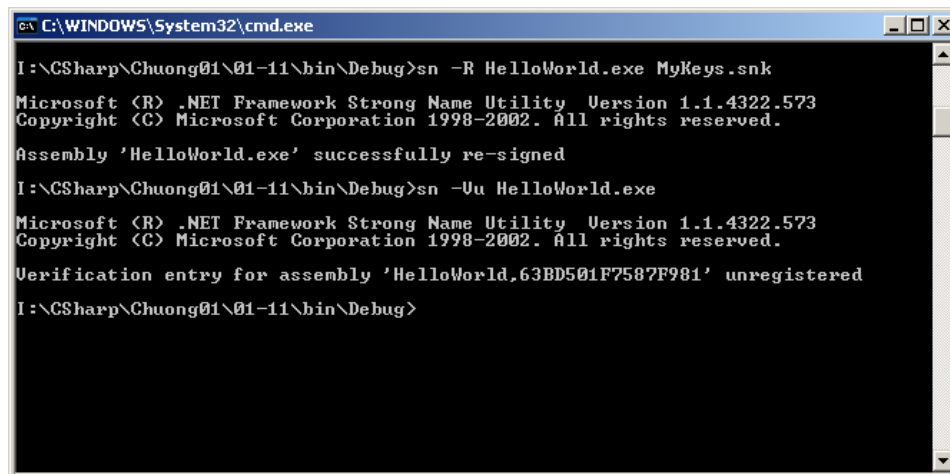
I:\CSharp\Chuong01\01-11>cd bin\Debug
I:\CSharp\Chuong01\01-11\bin\Debug>sn -Ur HelloWorld.exe

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



```
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

1.12

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.

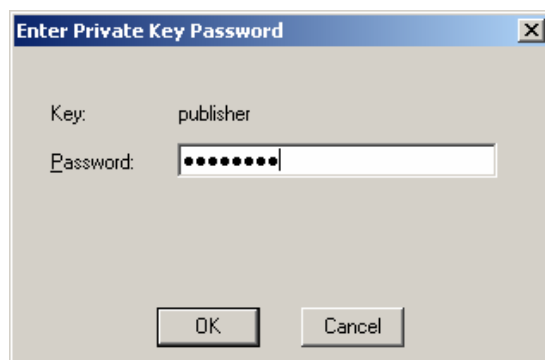
Để có được một *SPC*, bạn nên xem *Microsoft Root Certificate Program Members* tại [<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/rootcertprog.asp>]. Ở đây, bạn có thể tìm thấy danh sách các *CA*, nhiều *CA* trong số đó có thể cấp cho bạn một *SPC*. Với mục đích thử 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 hóa mã băm đã-được-mã-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.

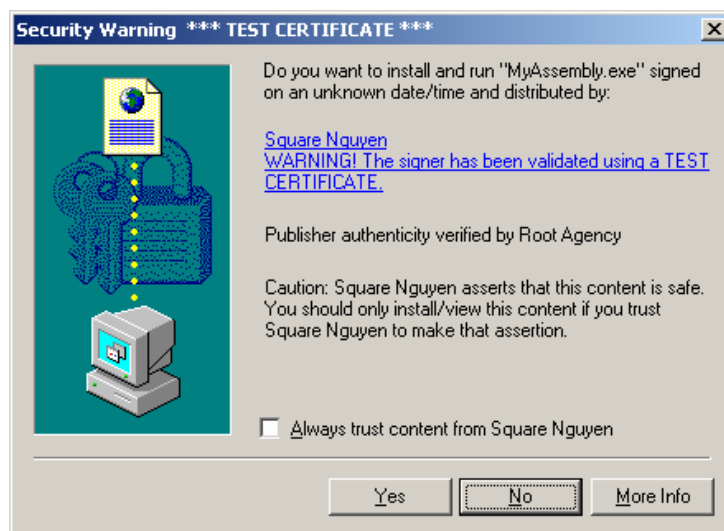
Bảng 1.2 Các đối số thường dùng của công cụ File Signing

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

1.13

Tạo và thiết lập tin tưởng một *SPC* thử nghiệm



Bạn cần tạo một *SPC* để thử nghiệm.



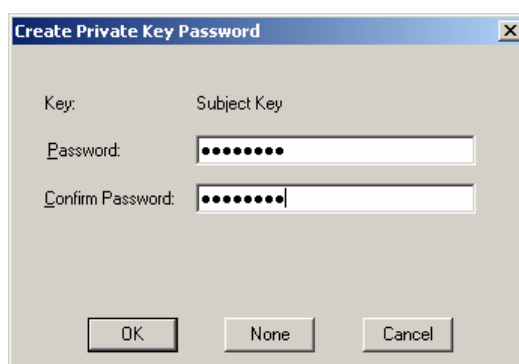
Sử dụng công cụ *Certificate Creation (makecert.exe)* để tạo một chứng chỉ *X.509* và sử dụng công cụ *Software Publisher Certificate (cert2spc.exe)* để tạo một *SPC* từ chứng chỉ *X.509* này. Thiết lập tin tưởng chứng chỉ thử nghiệm bằng công cụ *Set Registry (setreg.exe)*.

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.

1.14

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.**

1.15

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 nhưng 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.

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 mục 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 `Systems.Windows.Forms` được thừa nhận.**

6.1

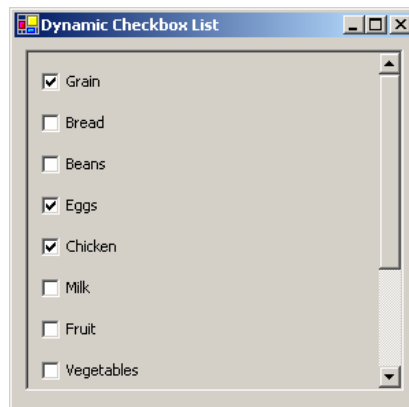
Thêm điều khiển vào form lúc thực thi

- ? 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 khiển. Bạn cần thực hiện các bước sau:

1. Tạo một đối tượng của lớp điều khiển thích hợp.
2. Cấu hình các thuộc tính của điều khiển (đặc biệt là kích thước và tọa độ vị trí).
3. Thêm điều khiển này vào form hoặc điều khiển container.
4. Ngoài ra, nếu cần thụ lý các sự kiện cho điều khiển mới, bạn có thể gắn chúng vào các phương thức hiện có.

Mỗi điều khiển đều cung cấp thuộc tính `Controls` để tham chiếu đến `ControlCollection` chứa tất cả các điều khiển con của nó. Để thêm một điều khiển 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);
        }
    }
}
```

6.2

Liên kết dữ liệu vào điều khiển

? Bạn cần liên kết một đối tượng vào một điều khiển 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 khiển.

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 khiển 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();

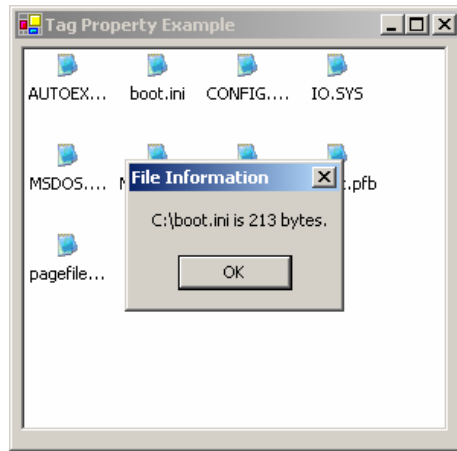
        // Hiển thị tất cả các file trong ListView.
        foreach (FileInfo file in files) {

            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

6.3

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).
- ✂** 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 trừ khi nó là TextBox.
        if (ctrl.GetType() == typeof(TextBox)) {
```

```

        ctrl.Text = "";
    }

    // Xử lý các điều kiện một cách đệ quy.
    // Điều này cần thiết khi có một điều kiện chứa nhiều
    // điều kiện khác (ví dụ, khi bạn sử dụng Panel,
    // GroupBox, hoặc điều kiện container nào khác).
    foreach (Control ctrlChild in ctrl.Controls) {
        ProcessControls(ctrlChild);
    }
}
}

```

6.4 *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);
}
```

6.5

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 thụ 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) {
```

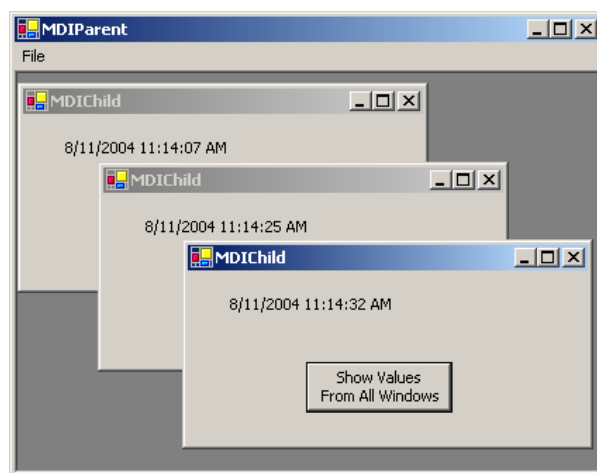
```

        // 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

6.6

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).

6.7

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 {

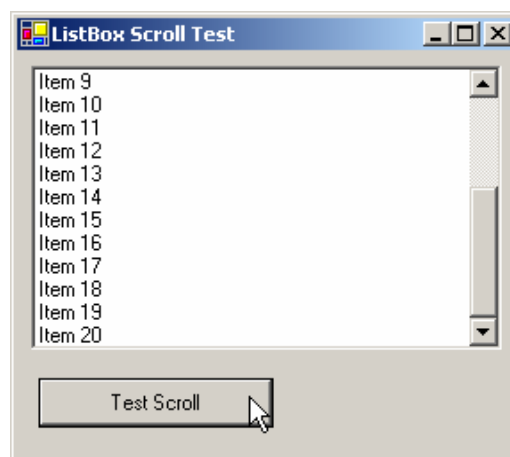
    // (Bỏ qua phần mã designer.)

    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

6.8

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.
- ✕** 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.

6.9

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 đề 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 đề 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) {

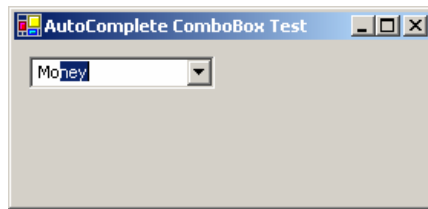
        // Thêm ComboBox vào form.
        AutoCompleteComboBox combo = new AutoCompleteComboBox();
        combo.Location = new Point(10,10);
        this.Controls.Add(combo);

        // 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

6.10

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();
    }
}

```

6.11

Liên kết menu ngữ cảnh vào điều khiển

- ?** Bạn cần liên kết một menu ngữ cảnh vào mỗi điều khiển 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 khiển.
- ✕** Viết một phương thức thụ lý sự kiện chung để thu lấy đối tượng `ContextMenu` được kết hợp với điều khiển, và rồi hiển thị menu này trên điều khiển.

Bạn có thể liên kết một điều khiển với một menu ngữ cảnh bằng cách thiết lập thuộc tính `ContextMenu` của điều khiển. 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 khiển gì. Mọi điều khiển 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 khiển.

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 khiển nguồn.
        Control ctrl = (Control)sender;

        if (ctrl.ContextMenu != null) {

            // Hiển thị menu ngữ cảnh.
            ctrl.ContextMenu.Show(ctrl, new Point(e.X, e.Y));
        }
    }
}
```

6.12

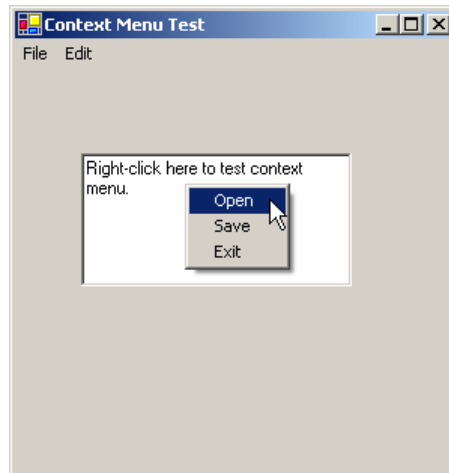
Sử dụng một phần menu chính cho menu ngữ cảnh

- ?** Bạn cần tạo một menu ngữ cảnh hiển thị các item giống với một số item trong menu chính của ứng dụng.
- ✕** 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 khiển 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.");

}

}

```

6.13

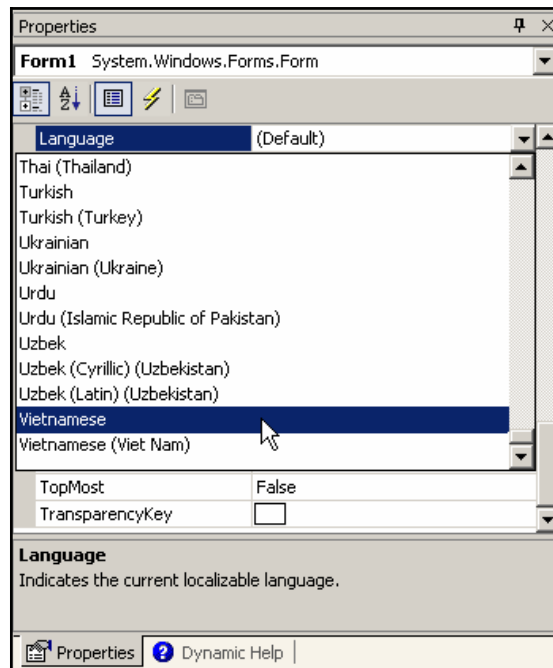
Tạo form đa ngôn ngữ

- ?** 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.
- ✕** 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:

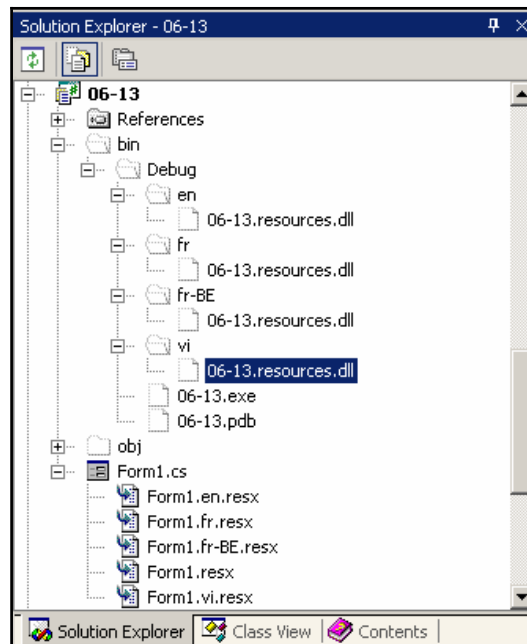
1. Trước tiên, thiết lập thuộc tính `Localizable` của form là `true` trong cửa sổ *Properties*.
2. Thiết lập thuộc tính `Language` của form là bản địa bạn muốn nhập thông tin cho nó (xem hình 6.7). Kế đó, cấu hình các thuộc tính có thể bản địa hóa của tất cả các điều khiển trên form. Thay vì lưu trữ 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;

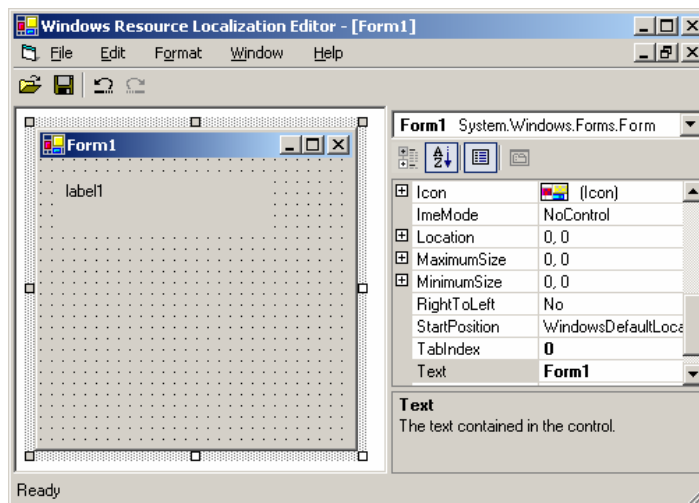
public class MultiLingualForm : System.Windows.Forms.Form {
    private System.Windows.Forms.Label label1;

    // (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*.

6.14

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`.

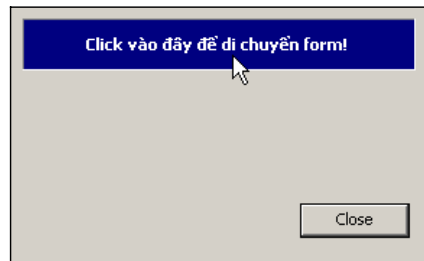
Bạn có thể tạo một form không đường viền bằng cách thiết lập thuộc tính `FormBorderStyle` là `None`. Các form này không thể di chuyển được. Và chúng cũng không có kiểu đường viền—nếu muốn có đường viền xanh, bạn phải tự thêm vào bằng cách viết mã hoặc sử dụng hình nền.

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`.

6.15 *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 khiển 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();
    }
}
```

6.16

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).

✂ 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 khiển này vào form, cung cấp hình icon bằng thuộc tính `Icon`. Bạn cũng có thể thêm một menu ngữ cảnh vào điều khiển này bằng thuộc tính `ContextMenu` (tùy chọn). Không giống với các điều khiển 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;
    }
}
```

6.17

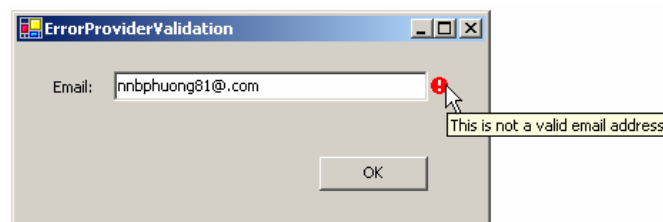
Xác nhận tính hợp lệ của đầu vào cho một điều kiểm

- ?** 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 dừng chờ 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();
        }
    }
}



```

6.18

Thực hiện thao tác kéo-và-thả

- ?** Bạn cần sử dụng tính năng kéo-và-thả để trao đổi thông tin giữa hai điều kiểm (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 kiểm, giữ chuột, và bắt đầu rê. Nếu điều kiểm hỗ trợ tính năng kéo-và-thả, nó sẽ thiết lập riêng một vài thông tin.
2. Người dùng rê chuột lên một điều kiểm khác. Nếu điều kiểm 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 khiển phải có thuộc tính `AllowDrop` là `true`. Điều khiển 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 khiển 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) {

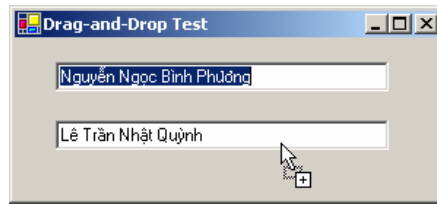
    if (e.Data.GetDataPresent(DataFormats.Text)) {
        e.Effect = DragDropEffects.Copy;
    }
    else {
        e.Effect = DragDropEffects.None;
    }
}
```

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

6.19

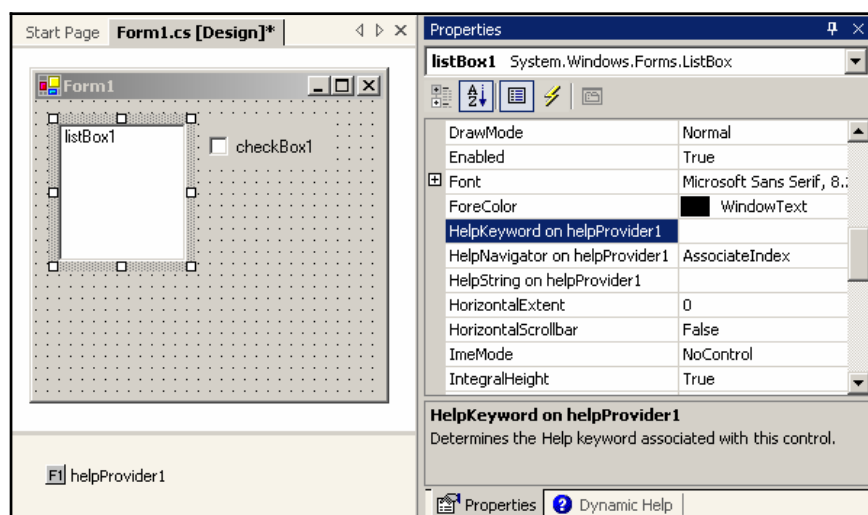
Sử dụng trợ giúp cảm-ngữ-cảnh

- ?** 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:

- Thiết lập thuộc tính `HelpProvider.HelpNamespace` là tên của file trợ giúp (chẳng hạn, `myhelp.chm`).
- Đối với mỗi điều kiểm yêu cầu trợ giúp cảm-ngữ-cảnh, hãy thiết lập thuộc tính mở rộng `HelpNavigator` là `HelpNavigator.Topic`.
- 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 khiển nào đó đang nhận focus, file trợ giúp sẽ được mở một cách tự động và chủ đề liên kết với điều khiển này sẽ được hiển thị trong cửa sổ 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.

6.20

Áp dụng phong cách Windows XP

- ? Bạn muốn các điều khiển mang dáng dấp hiện đại của *Windows XP* trên hệ thống *Windows XP*.**
- ✗ 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 cho 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)
}
```

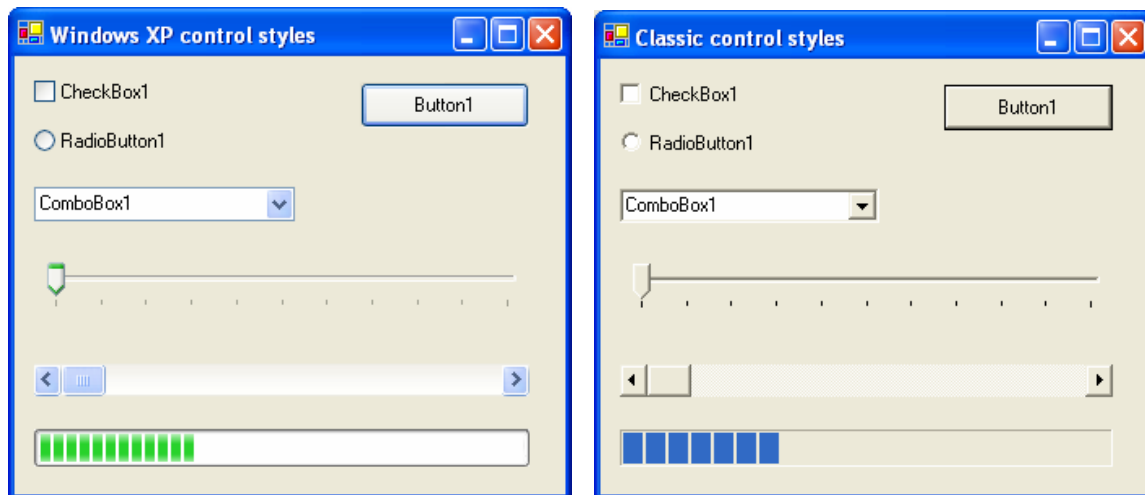
Nếu đang sử dụng *.NET Framework* phiên bản *1.0*, bạn không có sự trợ giúp của phương thức `Application.EnableVisualStyles`. Tuy nhiên, bạn vẫn có thể sử dụng phong cách này bằng cách tạo một file manifest cho ứng dụng của bạn. File manifest này (chỉ là một file văn bản thông thường với nội dung XML) 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.

6.21

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%`.

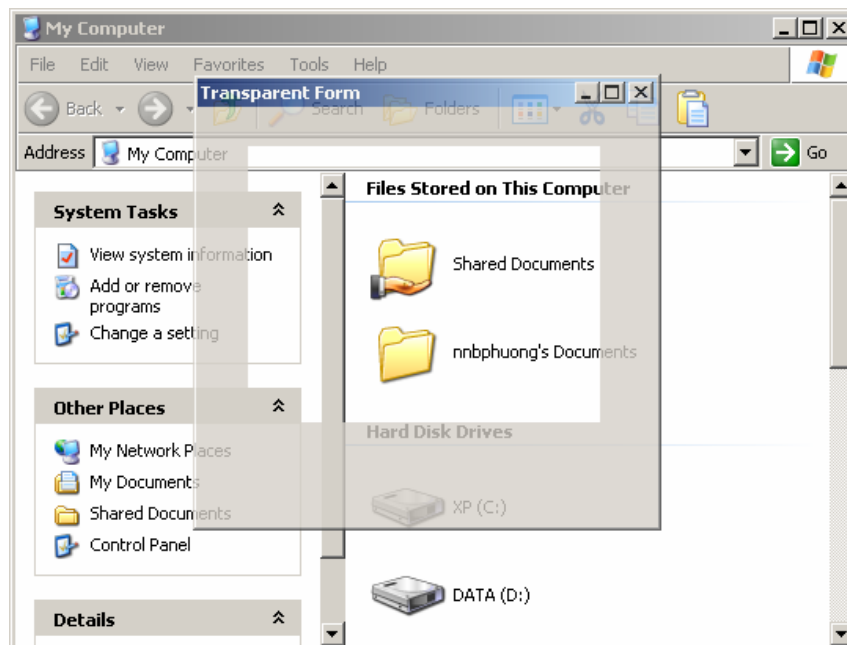
Thuộc tính `Opacity` của một form kiểm soát mức độ đục hay trong của một cửa sổ. Ở mức `100%`, form xuất hiện với trạng thái mặc định, nghĩa là không có các vùng trong suốt trên form. Ở mức `0%`, form hoàn toàn trong suốt, cũng có nghĩa bạn không thể tương tác với form được nữa.

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.

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 *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.

17.1

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();
}
}

```

17.2

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.
- ✕** 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();
}
}

```

17.3

Ghi một sự kiện vào nhật ký sự kiện Windows

- ? Bạn cần ghi một sự kiện vào nhật ký sự kiện *Windows*.**
- ✗ 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();
}
}

```

17.4

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 tính `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.
<code>DeleteValue</code>	Xóa một giá trị với tên được chỉ định khỏi khóa hiện tại.
<code>GetValue</code>	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 <code>GetValue</code> trả về <code>null</code> 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 <code>null</code>) nếu giá trị không tồn tại.
<code>GetValueNames</code>	Trả về mảng kiểu chuỗi chứa tên của tất cả các giá trị trong khóa hiện tại.
<code>SetValue</code>	Tạo (hoặc cập nhật) giá trị với tên được chỉ định. Bạn không thể chỉ định kiểu dữ liệu <i>Registry</i> dùng để lưu trữ dữ liệu; <code>SetValue</code> 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\

```

17.5

Tạo một dịch vụ Windows



Bạn cần tạo một ứng dụng đóng vai trò là một dịch vụ *Windows*.



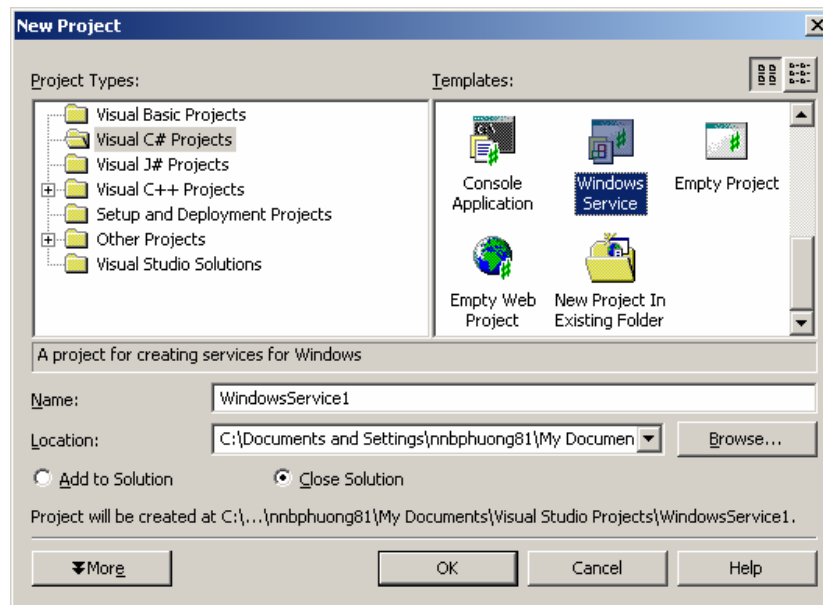
Tạo một lớp thừa kế từ lớp `System.ServiceProcess.ServiceBase`. Sử dụng các thuộc tính thừa kế để điều khiển hành vi của dịch vụ, và chép đề các phương thức thừa kế để hiện thực các chức năng cần thiết. Hiện thực phương thức `Main`, trong đó tạo một thể hiện của lớp dịch vụ và truyền nó cho phương thức 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
<code>OnStart</code>	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.
<code>OnStop</code>	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> .
<code>OnPause</code>	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> .
<code>OnContinue</code>	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> .
<code>OnShutdown</code>	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> .
<code>OnPowerEvent</code>	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> .

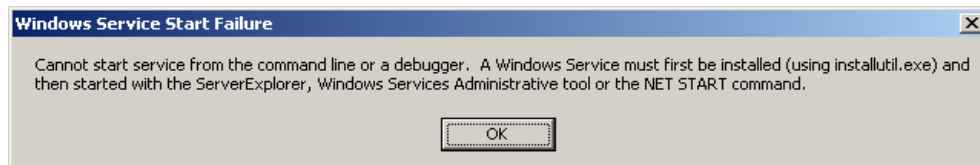
OnCustomCommand

Cho phép mở rộng cơ chế điều khiển dịch vụ với các thông điệp điều khiển tùy biến; xem chi tiết trong tài liệu *.NET Framework SDK*.

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 dựng 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 dùng 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());
}
}
```

17.6

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 thẻ 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 thẻ 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 {

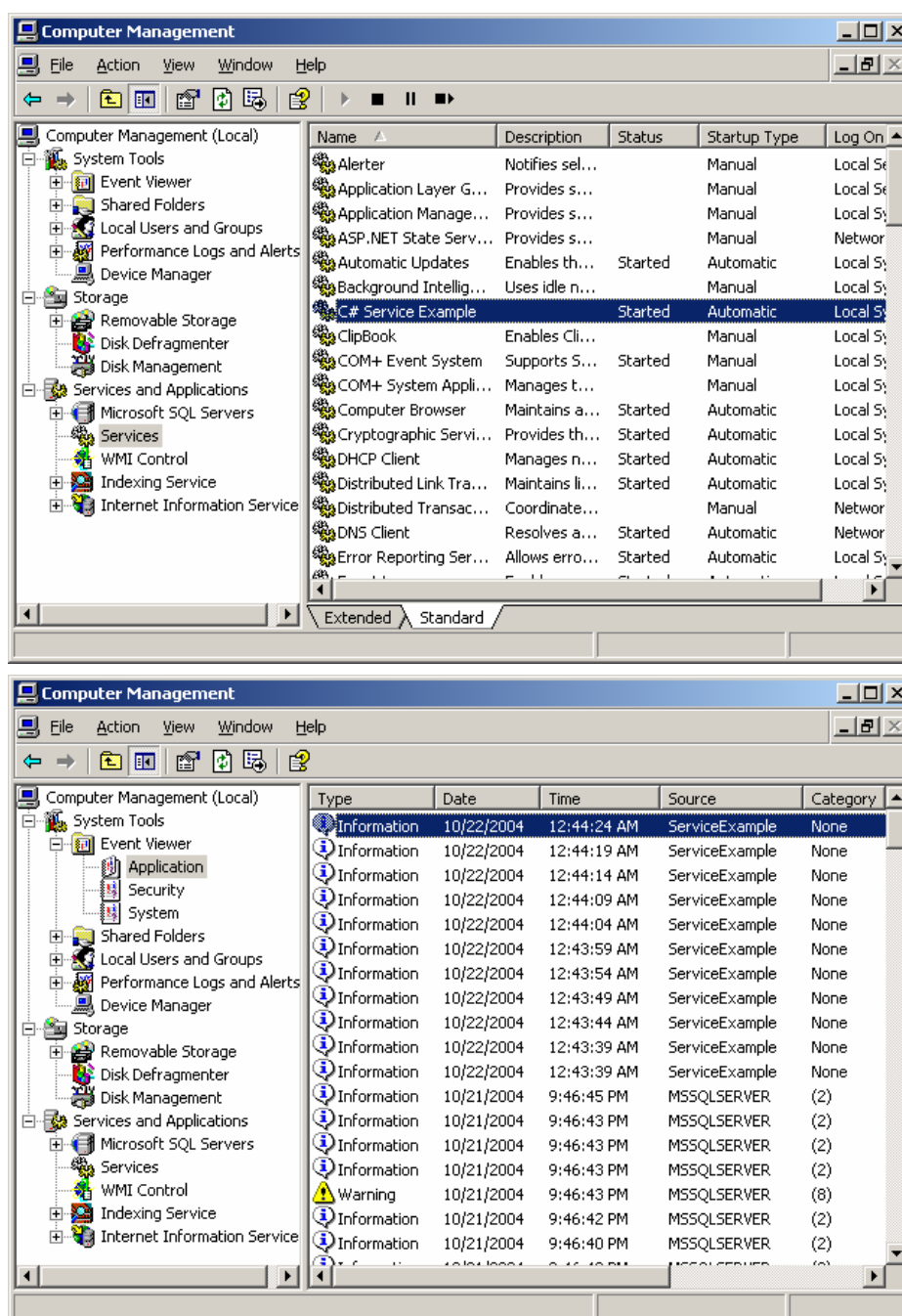
    public ServiceInstallerExample() {

        // 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 = ServiceStartMode.Automatic;

        // 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 dựng 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`.



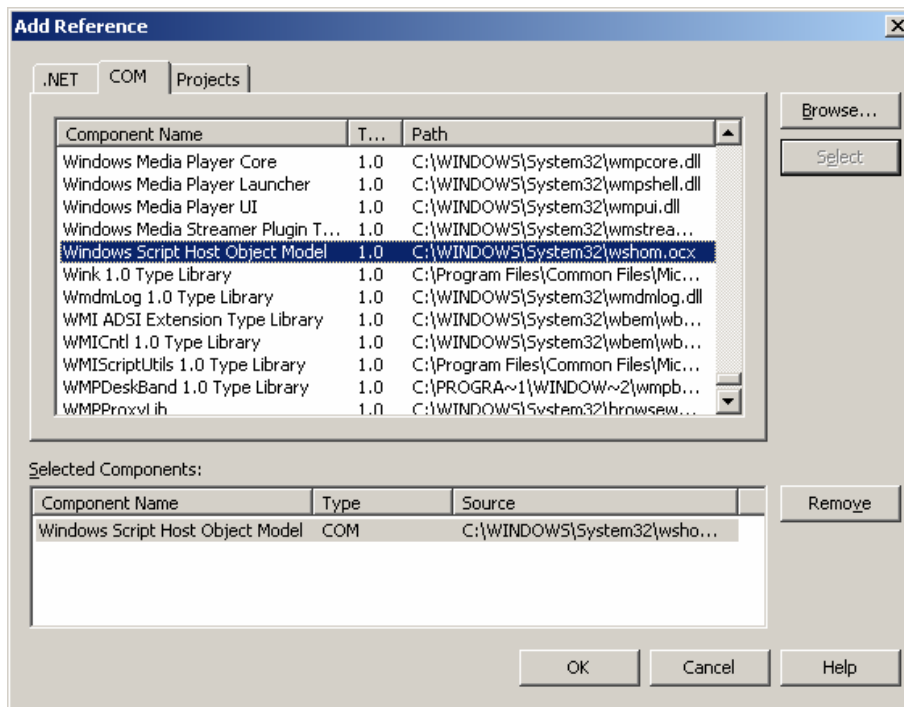
Hình 17.3 Windows Computer Management

17.7

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 thể 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 cửa 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();
}
```