

ĐẶNG VĂN ĐỨC - LÊ QUỐC HƯNG

LẬP TRÌNH WINDOWS BẰNG VISUAL C++

NHÀ XUẤT BẢN GIÁO DỤC



ĐẶNG VĂN ĐỨC (*chủ biên*)
LÊ QUỐC HÙNG

LẬP TRÌNH WINDOWS BẰNG VISUAL C++

(Tái bản lần thứ ba)

NHÀ XUẤT BẢN GIÁO DỤC

Chịu trách nhiệm xuất bản:

Chủ tịch HĐQT kiêm Tổng Giám đốc NGÔ TRẦN ÁI
Phó Tổng Giám đốc kiêm Tổng Biên tập NGUYỄN QUÝ THAO

Biên tập:

NGÔ ÁNH TUYẾT

Trình bày bìa:

TẠ TRỌNG TRÍ

Chế bản:

PHÒNG CHẾ BẢN (NXB GIÁO DỤC)

MỤC LỤC

LỜI NÓI ĐẦU	9
CHƯƠNG 1. TỔNG QUAN VỀ CÁC LỚP CỦA MFC	11
1.1 Cơ sở của lập trình Windows	11
1.2 Giới thiệu MFC	12
1.3 <i>CObject</i> , cha của tất cả các lớp	13
1.4 Các lớp kiến trúc ứng dụng	14
1.5 Các lớp đối tượng trực quan	15
1.6 Các lớp chức năng chung	16
1.7 Các hàm <i>AfxXxx</i>	17
1.8 Thí dụ HELLO1	18
1.9 Đối tượng ứng dụng	20
1.10 Hàm <i>InitInstance</i>	21
1.11 Các thành phần của lớp <i>CWinApp</i>	22
1.12 Nạp chồng hàm <i>CWinApp</i>	24
1.13 MFC sử dụng đối tượng ứng dụng như thế nào?	24
1.14 Đối tượng cửa sổ	25
1.15 Tạo lập cửa sổ	26
1.16 Vẽ trên vùng làm việc của cửa sổ	29
1.17 Ánh xạ thông điệp	30
1.18 Hàm <i>AfxRegisterWndClass</i>	33
1.19 Tóm tắt	37
CHƯƠNG 2. SỬ DỤNG BÚT VẼ VÀ MÀU TÔ	39
2.1 Vẽ bằng GDI	39
2.2 DC trong <i>OnPaint</i>	40
2.3 DC bên ngoài <i>OnPaint</i>	41
2.4 Vẽ bằng DC	42
2.5 Bút vẽ	45
2.6 Chổi tô	47
2.7 Xóa bút và chổi tô	49
2.8. Thí dụ ứng dụng <i>GdiObj1</i>	50
2.8.1 Hàm <i>OnCreate</i>	56
2.8.2 Hàm <i>ShowPenStyles</i>	58

2.8.3 Hàm <i>ShowPenWidths</i>	59
2.8.4 Hàm <i>ShowBrushStyles</i>	59
2.8.5 Xem những gì ta đã vẽ	60
2.9. Thêm thanh trượt vào cửa sổ	60
2.9.1 Đặt dải trượt, vị trí và kích thước trang cho thanh trượt	60
2.9.2 Xử lý các thông điệp của thanh trượt	63
2.9.3 Trượt nội dung cửa sổ	65
2.9.4 Làm thanh trượt một cách dễ dàng	67
2.9.5 Lớp <i>CScrollView</i>	67
2.9.6 Ứng dụng <i>GdiObj2</i>	69
CHƯƠNG 3. XỬ LÝ ĐẦU VÀO TỪ CHUỘT	77
3.1 Nhận thông điệp chuột trong vùng làm việc	77
3.2 Ứng dụng TicTac	81
3.2.1 Hộp thông báo	89
3.2.2 Không có cửa sổ khung?	91
3.3 Các thông điệp chuột ngoài vùng làm việc	94
3.4 Bắt giữ (<i>capturing</i>) chuột	96
3.5 Thực hiện bắt giữ chuột	97
3.6 Con chạy	102
3.7 Tự tạo con chạy	106
3.8 Các vấn đề khác của chuột	107
CHƯƠNG 4. BÀN PHÍM	109
4.1 Các thông điệp gõ phím	109
4.2 Mã phím ảo	112
4.3 Trạng thái phím <i>Shift</i> và lật	112
4.4 Các thông điệp ký tự	115
4.5 Con chạy soạn thảo	117
4.6 Ứng dụng <i>VisualKB</i>	120
4.7 Quản lý con chạy soạn thảo	121
4.8 Nhập và soạn thảo văn bản	134
4.9 Quản lý con chạy chuột	135
4.10 Các điểm cần quan tâm	136
4.11 Các vấn đề khác của bàn phím	137

CHƯƠNG 5. THỰC ĐƠN, THANH CÔNG CỤ VÀ THANH TRẠNG THÁI	139
5.1 Thực đơn và tài nguyên thực đơn	139
5.2 Nạp thực đơn và phím lệnh cấp tốc	142
5.3 Các lệnh và ánh xạ thông điệp	144
5.4 Dải lệnh	145
5.5 Trình điều khiển cập nhật	147
5.6 Thanh công cụ.....	148
5.7 Thanh trạng thái	152
5.8 Thực đơn ngữ cảnh	154
5.9 Gộp tất cả lại	156
CHƯƠNG 6. HỘP THOẠI, TẤM ĐẶC TÍNH VÀ CÁC ĐIỀU KHIỂN	177
6.1 Hộp thoại và lớp <i>CDialog</i>	177
6.2 Các lớp điều khiển MFC	179
6.3 Tập hợp lại	183
6.4 Trao đổi dữ liệu hộp thoại và kiểm tra	186
6.5 Làm phím <i>bitmap</i>	189
6.6 Ứng dụng NAP.....	191
6.7 Hộp thoại <i>Modeless</i>	198
6.8 Tấm đặc tính	199
6.9 Ứng dụng PSD	203
6.10 Các vấn đề khác sẽ xem xét	214
CHƯƠNG 7. KIẾN TRÚC TÀI LIỆU/QUAN SÁT	215
7.1 Các vấn đề cơ bản của tài liệu/quan sát	215
7.2 Đối tượng tài liệu.....	218
7.3 Đối tượng quan sát	221
7.4 Đối tượng cửa sổ khung	223
7.5 Tạo lập động các đối tượng.....	223
7.6 <i>Serialization</i>	224
7.7 Mẫu tài liệu SDI	225
7.8 Đăng ký kiểu tài liệu	227
7.9 Đường dẫn lệnh	227
7.10 Chỉ danh của lệnh và cài đặt lệnh mặc định.....	229
7.11 Thí dụ về một ứng dụng tài liệu/quan sát	231
7.12 Triển vọng của tài liệu/quan sát	252

CHƯƠNG 8. IN VÀ QUAN SÁT TRƯỚC KHI IN.....	253
8.1 Tài liệu và Quan sát.....	253
8.2 Kiến trúc in của Windows	253
8.3 Thủ tục <i>Abort</i> và hộp thoại <i>Abort</i>	256
8.4 Cuộn in.....	258
8.5 Kiến trúc in của MFC	258
8.5.1 Chi tiết hơn về hàm <i>OnPreparePrinting</i>	259
8.5.2 <i>OnBeginPrinting</i> và <i>OnEndPrinting</i>	260
8.5.3 <i>OnPrepareDC</i>	261
8.5.4 <i>OnPrint</i>	262
8.5.5 <i>CView::OnFilePrint</i> và các hàm điều khiển lệnh khác	263
8.6 Quan sát trước khi in	264
8.7 Thí dụ mẫu vẽ in.....	265
8.8 Thí dụ in hoàn hảo hơn.....	270
8.9 Cách tiếp cận duy nhất đến vào/ra (<i>serialization</i>).....	282
8.9.1 Cho phép phím chọn hoạt động.....	283
8.9.2 Trợ giúp đánh số trang mặc định.....	285
8.10 Liệt kê máy in	285
CHƯƠNG 9. LIÊN KẾT CƠ SỞ DỮ LIỆU.....	287
9.1 Giới thiệu SQL	287
9.1.1 Khai thác dữ liệu bằng SQL.....	287
9.1.2 Kết nối các bảng bằng SQL	290
9.1.3 Sắp xếp các bản ghi	292
9.2 Trợ giúp cơ sở dữ liệu của MFC.....	292
9.2.1 DAO và ODBC.....	293
9.2.2 Các lớp trợ giúp cách tiếp cận DAO.....	293
9.2.2 Các lớp trợ giúp cách tiếp cận ODBC.....	294
9.3 Tạo ứng dụng cơ sở dữ liệu	295
9.3.1 Đăng ký cơ sở dữ liệu ODBC.....	295
9.3.2 Sử dụng <i>AppWizard</i> để tạo chương trình ODBC.....	297
9.3.3 Tìm hiểu cấu trúc chương trình.....	300
9.3.4 Sắp xếp dữ liệu trong <i>recordset</i>	307
9.3.4 Sử dụng đối tượng <i>recordset</i> thứ hai.....	308
CHƯƠNG 10. NHÚNG VÀ LIÊN KẾT ĐỐI TƯỢNG.....	319
10.1 Tài liệu đa hợp.....	320

10.2 OLE làm việc như thế nào?	320
10.3 Các lớp MFC trợ giúp OLE.....	321
10.4 Cài đặt trình chứa OLE	323
10.4.1 Khởi động trình chứa OLE.....	323
10.4.2 Lớp <i>CWrxContainerItem</i>	325
10.4.3 Quản lý đa đối tượng nhúng.....	330
10.5 Cài đặt trình dịch vụ OLE.....	336
10.5.1 Tăng cường chức năng vẽ cho chương trình.....	336
10.5.2 Cài đặt các chức năng dịch vụ	344
CHƯƠNG 11. ĐIỀU KHIỂN ActiveX	347
11.1 <i>ActiveX</i> và OLE	347
11.1.1 Điều khiển OLE là gì?	347
11.1.2 Điều khiển <i>ActiveX</i> là gì?	347
11.1.3 Điều khiển OLE làm việc như thế nào?	348
11.2 Cài đặt điều khiển <i>ActiveX</i>	352
11.2.1 Phát sinh điều khiển <i>ActiveX</i> cơ sở	352
11.2.2 Định nghĩa đối tượng tín hiệu giao thông	354
11.2.3 Sử dụng đối tượng <i>CTrafficSignal</i>	359
11.2.4 Sử dụng các đặc tính có sẵn và bao quanh	360
11.2.5 Cài đặt thuộc tính mới cho điều khiển	361
11.2.6 Làm đèn tín hiệu hoạt động	363
11.2.7 Cài đặt sự kiện cho điều khiển <i>ActiveX</i>	366
CHƯƠNG 12. LẬP TRÌNH INTERNET	369
12.1 Mở rộng dịch vụ Web bằng ISAPI	370
12.1.1 Các lớp MFC của ISAPI.....	371
12.1.2 Chương trình mở rộng dịch vụ Internet.....	372
12.1.3 Chương trình lọc sử dụng ISAPI.....	394
12.2 Ứng dụng Internet sử dụng <i>WinInet</i>	405
12.2.1 Xây dựng chương trình Internet bằng MFC <i>WinInet</i>	407
12.2.2 Các bước xây dựng một chương trình Internet	408
12.2.3 Chương trình TEAR	412
12.2.4 Quản lý thông tin trạng thái Internet trong quá trình thực hiện.....	422
CHƯƠNG 13. HƯỚNG DẪN SỬ DỤNG DEVELOPER STUDIO	423
13.1 Các thành phần của hệ thống	423

13.1.1 Chương trình soạn thảo.....	423
13.1.2 Chương trình dịch.....	423
13.1.3 Chương trình liên kết.....	423
13.1.4 Các thư viện của C++	423
13.1.5 Các công cụ khác.....	424
13.2 Sử dụng môi trường phát triển của Visual C++	424
13.3 <i>Project</i> và <i>Project Workspace</i>	425
13.4 Tạo lập <i>project</i>	427
13.5 Xây dựng chương trình	431
13.6 Cài đặt các lựa chọn trong Visual C++	431
13.6.1 Cài đặt các lựa chọn cho <i>Developer Studio</i>	432
13.6.2 Cài đặt các lựa chọn cho <i>project</i>	432
13.7 Tạo chương trình Windows bằng công cụ <i>AppWizard</i>	434
13.7.1 Tạo chương trình thực hiện trong Windows (.EXE).....	434
13.7.2 Tạo chương trình điều khiển <i>ActiveX</i>	436
13.7.3 Tạo chương trình mở rộng dịch vụ Internet sử dụng các lớp <i>ISAPI</i>	438
13.7.4 Tạo chương trình lọc sử dụng các lớp <i>ISAPI</i>	439
13.8 Gỡ rối (<i>Debug</i>) chương trình trong <i>Developer Studio</i>	441
13.8.1 Đặt điểm dừng.....	441
13.8.2 Thực hiện gỡ rối chương trình.....	442
13.8.3 Kết thúc quá trình gỡ rối.....	443
13.8.4 Gỡ rối cho chương trình lọc và trình mở rộng dịch vụ Internet.....	443
MỘT SỐ THUẬT NGỮ.....	445
TÀI LIỆU THAM KHẢO	448

LỜI NÓI ĐẦU

Ngay sau khi xuất hiện ngôn ngữ C thì lập trình hướng thủ tục trở nên rất thông dụng vào những năm 1980. Lập trình hướng thủ tục là công cụ hữu hiệu để viết các chương trình tương đối lớn. Nhưng với các chương trình phức tạp hơn thì quan điểm lập trình hướng thủ tục tỏ ra gặp khó khăn để xây dựng được chương trình chạy ổn định, dễ bảo dưỡng và cho phép sử dụng lại mã trình. Thực tế cho thấy hơn 50% sản phẩm phần mềm không được sử dụng ngay và chúng phải thay đổi theo yêu cầu người dùng vì các lý do khác nhau như phần mềm có lỗi, thay đổi khuôn mẫu dữ liệu, nâng cấp chức năng ... Để giải quyết vấn đề nói trên, các phương pháp lập trình mới đã được nghiên cứu, trong đó có phương pháp lập trình hướng đối tượng. Lập trình hướng đối tượng là cách tổ chức và phát triển chương trình để hạn chế một số nhược điểm của phương pháp lập trình hướng thủ tục bằng cách kết hợp các đặc trưng tốt nhất của lập trình hướng thủ tục với các quan niệm mới, mạnh hơn như lớp, thừa hưởng, đa trị ...

Visual C/C++ của Microsoft trợ giúp hữu hiệu phương pháp lập trình mới nói trên. Thư viện MFC (Microsoft Foundation Class) của Visual C++ tỏ ra khá hiệu quả, chúng làm giảm nhẹ sức lực của người lập trình theo quan điểm hướng đối tượng. Chương trình xây dựng trên cơ sở MFC có tính thích nghi cao, sẵn sàng chạy trên các thế hệ Windows khác nhau. Người mới bắt đầu học lập trình có thể cho rằng Visual C++ là ngôn ngữ khó học. Họ đúng vì bản thân ngôn ngữ C/C++ đã bao gồm các cú pháp khó nhớ như ++, --, [,]... Tuy nhiên, khi đã sử dụng thành thạo ngôn ngữ này thì họ sẽ không muốn rời bỏ nó vì các lợi thế lớn lao so với các ngôn ngữ khác. Trước hết là tính mềm dẻo, lập trình viên có khả năng điều khiển môi trường lập trình để xây dựng được các chương trình đáp ứng mọi yêu cầu thực tế với chất lượng cao. Vì xuất phát từ ngôn ngữ của hệ điều hành, cho nên Visual C++ cho khả năng tận dụng mọi sức mạnh của hệ thống cho chương trình ứng dụng. Tiếp theo, Visual C++ cho lại chương trình ngắn gọn và thực hiện với tốc độ rất cao, đôi khi so sánh được với chương trình viết bằng hợp ngữ. Hơn nữa, những phiên bản cuối cùng của Visual C++ đã có nhiều cải tiến giao diện trực quan khi lập trình. Khi sử dụng chúng, bạn đọc có thể tạo nhanh ra các chương trình đơn giản như sử dụng các ngôn ngữ "phát triển nhanh ứng dụng" (Rapid Application Development - RAD). Đặc biệt, chức năng *wizard* của chúng đã làm giảm thiểu thời gian ban đầu để tạo lập khung chương trình. Đồng thời chúng cho phép dễ dàng phát sinh các ứng dụng liên quan đến Internet, cơ sở dữ liệu (CSDL) ...

Nội dung tài liệu

Không tập trung vào hướng dẫn sử dụng giao diện Visual C++ như nhiều tài liệu khác, trong tài liệu này các bước xây dựng chương trình ứng dụng bằng MFC sẽ được giới thiệu kỹ lưỡng hơn. Đồng thời nguyên lý hoạt động của các thành tố chương trình trong môi trường Windows cũng được xem xét để bạn đọc dễ dàng mở rộng chương trình theo nhu cầu. Tuy nhiên, tài liệu cũng sẽ mô tả cách sử dụng *wizard*, giúp bạn đọc phát sinh nhanh khung chương trình hay các chương trình

đơn giản.

Chương đầu của tài liệu trình bày tổng quan về thư viện lớp và cấu trúc của chương trình MFC. Các chương tiếp theo (chương 2 - chương 8) hướng dẫn sử dụng thư viện MFC vào việc xây dựng các thành phần cơ bản cho ứng dụng Windows như giao diện đồ họa, sử dụng bàn phím, chuột, máy in, hộp thoại, kiến trúc tài liệu/quản sát... Chương 9 trình bày các bước kết nối và tương tác với CSDL. Chương này cho thấy việc xây dựng ghép nối CSDL trên cơ sở hộp thoại dễ dàng như thế nào. Chương 10 và chương 11 giới thiệu một trong các khía cạnh quan trọng nhất của việc phát triển phần mềm hiện nay và trong tương lai là các ứng dụng liên quan đến công nghệ "nhúng và liên kết đối tượng" (OLE, ActiveX). Chương 12 hướng dẫn người đọc xây dựng các ứng dụng trên Internet cả phía máy trạm và phía máy chủ như thăm nhập tài nguyên của Internet, xây dựng mở rộng dịch vụ Internet ... Chương cuối của tài liệu hướng dẫn người đọc sử dụng phương pháp lập trình trực quan của Visual C++ 5.0 để biên soạn, dịch, liên kết, chạy thử và gỡ rối chương trình ứng dụng. Toàn bộ nội dung của tài liệu là kiến thức cơ bản về lập trình Windows bằng MFC.

Để sử dụng tài liệu có hiệu quả

Nếu bạn đọc chưa có cơ hội lập trình bằng ngôn ngữ C/C++ thì nên học nhanh qua chúng để hiểu được các khái niệm cơ bản như lớp, thừa hưởng, hàm ảo, hàm nạp chồng, ... Các kiến thức này có thể tìm thấy trong tài liệu "*Ngôn ngữ lập trình C++*", Ngô Trung Việt, Nhà xuất bản Giao thông vận tải, 1995. Nếu chưa làm quen với phong cách lập trình thông điệp của Windows thì hãy tham khảo nhanh tài liệu "*Lập trình C trên Windows*", Đặng Văn Đức, Nhà xuất bản Khoa học kỹ thuật, 1998. Bạn đọc có thể sử dụng các chương trình dịch Visual C++ của Microsoft, phiên bản 4.2 hay các phiên bản mới hơn trên Windows 32 bit để thực hành các thí dụ trình bày trong tài liệu này.

Tài liệu này dành cho ai

Tài liệu "*Lập trình Windows bằng Visual C++*" được hình thành từ các kinh nghiệm sử dụng Visual C++ 5.0 để xây dựng các chương trình ứng dụng khác nhau cho các nhu cầu trong và ngoài nước, từ các bài giảng cho sinh viên của một số trường đại học. Chúng tôi cho rằng tài liệu này sẽ có ích cho các sinh viên, học viên cao học và cho tất cả những ai muốn xây dựng các ứng dụng tin học có chất lượng cao trong môi trường Windows.

Địa chỉ liên lạc

Trong quá trình biên soạn chúng tôi đã nhận được rất nhiều những góp ý quý giá của các chuyên gia tin học. Tuy nhiên, cũng như nhiều tài liệu xuất bản lần đầu, tài liệu này chắc chắn không tránh khỏi các sai sót. Chúng tôi luôn xin chân thành cảm ơn mọi ý kiến đóng góp của bạn đọc. Địa chỉ liên lạc: dvduc@ioit.ncst.ac.vn

Các tác giả

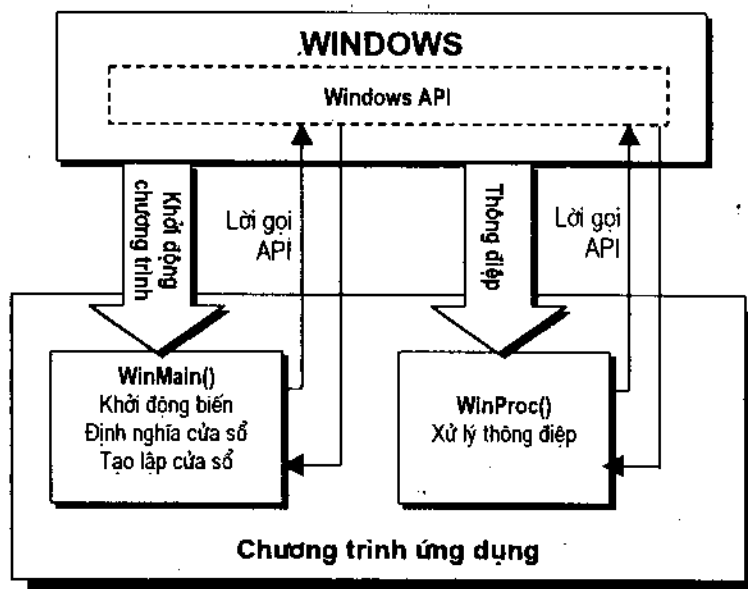
CHƯƠNG 1

TỔNG QUAN VỀ CÁC LỚP CỦA MFC

Trước đây phần lớn các ứng dụng Windows được viết bằng ngôn ngữ C. Ngày nay các môi trường lập trình mới đã cho phép viết các chương trình thương mại có chất lượng cao bằng C, C++, Pascal, BASIC và các ngôn ngữ khác. Trong đó, ngôn ngữ C++ đã được các lập trình viên chuyên nghiệp chọn để sử dụng thay cho C. Kết hợp với thư viện lớp, C++ trừu tượng giao diện lập trình ứng dụng (*Application Programming Interface - API*) của Windows nhờ các lớp cơ sở để biểu diễn các đối tượng chung như cửa sổ, thực đơn... C++ làm giảm các khó khăn cho những người lập trình ít kinh nghiệm. Tài liệu này mô tả cách sử dụng MFC (*Microsoft Foundation Class*) để viết các ứng dụng Windows. Khi mô tả tổng quát về MFC chúng tôi giả sử rằng người đọc đã quen biết mô hình lập trình Windows trên cơ sở thông điệp và các đặc tính cơ bản của C++ như lớp, thừa hưởng... Người đọc có thể tìm hiểu sâu hơn về các vấn đề này trong các tài liệu [1] và [2].

1.1 CƠ SỞ CỦA LẬP TRÌNH WINDOWS

Tài liệu [1] đã mô tả nguyên lý cơ bản về lập trình Windows trên cơ sở thông điệp. Một chương trình Windows có hai thành phần chính: khởi động chương trình và xử lý các thông điệp cửa sổ như mô tả trong hình 1.1.



Hình 1.1 Kiến trúc chương trình Windows

Kiến trúc trong hình 1.1 là lõi của của mọi chương trình Windows. Ta có thể thấy hai thành phần chính của chương trình là hàm **WinMain** và hàm **WinProc**. Windows sẽ gọi hàm **WinMain** khi bắt đầu chạy chương trình ứng dụng. Hàm **WinProc** được Windows gọi để xử lý các thông điệp gửi đến cửa sổ ứng dụng. Hàm **WinMain** thực hiện mọi khởi động và thiết lập cửa sổ để làm giao diện với người sử dụng. Chúng còn chứa vòng lặp để lấy các thông điệp cho ứng dụng từ hàng đợi trong hệ thống. Hàm **WinProc** xử lý toàn bộ các thông điệp đang xếp hàng. Vòng lặp thông điệp của hàm **WinMain** sẽ phân phát các thông điệp trong hàng đợi cho các hàm quản lý cửa sổ xử lý. **WinProc** có trách nhiệm liên lạc với người sử dụng bằng cách xử lý các thông điệp do người sử dụng phát sinh như dịch hay nhấn phím chuột, nhấn bàn phím, Các thông điệp trong hàng đợi là các thông điệp do người sử dụng phát sinh ra. Các thông điệp không xếp hàng nhưng yêu cầu Windows phải gọi trực tiếp **WinProc** là các thông điệp do chương trình phát sinh hay thông điệp liên quan đến quản lý cửa sổ.

Các thí dụ trong tài liệu [1] sử dụng giao diện lập trình ứng dụng Windows (Windows API). *Windows API* là một phần của hệ điều hành Windows. Chúng bao gồm hàng nghìn hàm để liên lạc với Windows và các hàm dịch vụ khác để tạo ra các ứng dụng chạy trong môi trường Windows. Toàn bộ quá trình tương tác giữa chương trình và người sử dụng là do Windows quản lý. Chương trình ứng dụng nhận thông tin từ người dùng thông qua thông điệp Windows. Mọi chương trình viết trên Windows đều sử dụng *Windows API*.

Windows API được Microsoft phát triển trước khi ngôn ngữ C++ được sử dụng rộng rãi. Tuy nhiên, ngày nay chúng đã được Microsoft tổ chức lại để được sử dụng trong các chương trình viết bằng nhiều ngôn ngữ khác nhau, kể cả ngôn ngữ hướng đối tượng.

1.2 GIỚI THIỆU MFC

MFC là tập hợp các lớp định nghĩa sẵn. Chúng biểu diễn cách tiếp cận hướng đối tượng đến lập trình Windows và chúng gói các Windows API. Rất nhiều lập trình viên C++ đã chọn thư viện MFC như là thư viện lớp để phát triển ứng dụng của mình. MFC cho phép lập trình viên dành nhiều thời gian hơn cho phát triển các thành phần cấu trúc chương trình và ít phải lo lắng về giao diện của Windows. Đặc biệt, MFC làm đơn giản tiến trình phát triển mã trình cho các loại máy có hệ điều hành khác nhau: Windows 16 bit, Win32, MacIntosh.

Hiện nay MFC 6.0 có hơn 130 lớp, trong đó một số lớp được trực tiếp sử dụng và một số khác làm các lớp cơ sở cho lớp riêng của lập trình viên. Rất nhiều lớp khá đơn giản, chẳng hạn như lớp **CPoint** (chỉ chứa toạ độ x, y của điểm). Một số lớp khác lại rất phức tạp, thí dụ lớp **CWnd** chứa cả cửa sổ mà ta quan sát thấy trên màn hình.

Trong chương trình MFC, giao diện lập trình rất ít khi được gọi trực tiếp. Thay vì đó là tạo ra

các đối tượng của các lớp MFC và gọi các hàm thành phần của chúng. Rất nhiều hàm thành phần MFC là lớp bao bọc mỏng xung quanh API; và rất nhiều hàm, thậm chí, có cùng tên với các hàm API. Điều này giúp lập trình viên C có thể chuyển nhanh sang MFC. Thí dụ, khi dịch chuyển cửa sổ, trong C ta gọi hàm API với tên `SetWindowsPos`. Khi tìm tên hàm này trong MFC ta thấy MFC cũng trợ giúp chúng. Đó là hàm thành phần của lớp `CWnd` khi cửa sổ được coi là đối tượng còn `SetWindowPos` là một hành động mà ta muốn thực hiện trên đối tượng đó.

MFC còn là khung (*framework*) ứng dụng. MFC giúp định nghĩa cấu trúc của ứng dụng và quản lý rất nhiều các công việc khác nữa. Bắt đầu với `CWinApp`, là lớp biểu diễn chính ứng dụng, MFC bao gồm hầu hết mọi thao tác chương trình. Khung ứng dụng cung cấp hàm `WinMain`, đến lượt `WinMain` gọi các hàm thành phần của đối tượng ứng dụng để làm cho chương trình chạy. Một trong các hàm thành phần của `CWinApp` mà `WinMain` gọi đến là hàm `Run`, chúng chứa vòng lặp thông điệp và lệnh chạy chương trình. MFC còn trợ giúp các tài liệu (*document*) và quan sát (*view*), cho phép tách dữ liệu của chương trình khỏi biểu diễn đồ họa của chúng. MFC cũng đưa ra một số hàm API riêng, độc lập với các lớp và được dùng chung. Chúng ta hãy xem xét sơ bộ một số lớp và hàm API có trong MFC.

1.3 CObject, CHA CỦA TẤT CẢ CÁC LỚP

Khi quan sát biểu đồ phân cấp MFC ta sẽ thấy phần lớn các lớp MFC là lớp suy diễn trực tiếp hoặc gián tiếp từ `CObject`. `CObject` cung cấp ba kiểu cơ bản để xây dựng các lớp thừa hưởng từ chúng:

- Vào/ra dữ liệu (*Serialization*)
- Thông tin về lớp khi chương trình đang chạy (*Run-time*)
- Thông báo kiểm tra lỗi

Serialization (vào/ra) là tiến trình giúp các đối tượng tự truyền đến hoặc truyền đi từ phương tiện lưu trữ như đĩa từ, *clipboard*. `CObject` có hai hàm thành phần làm nhiệm vụ xử lý vào/ra: `IsSerializable` và `Serialize`. Chương trình có thể gọi hàm `IsSerializable` của đối tượng suy diễn từ `CObject` để xác định xem chúng có trợ giúp vào/ra hay không.

Đặc trưng khác mà các lớp suy diễn từ `CObject` thừa hưởng là trợ giúp việc lấy thông tin các đối tượng của chương trình đang chạy. Cho trước con trỏ `CObject` đến lớp suy diễn từ chúng, tiện ích thông tin kiểu run-time cho phép chương trình biết được tên lớp của đối tượng cũng như các thông tin khác. `CObject` cung cấp hai hàm thành phần để làm việc này: `IsKindOf` và `GetRuntimeClass`.

Đặc trưng thứ ba mà `CObject` thêm vào các lớp suy diễn của chúng là trợ giúp kiểm tra lỗi. Hàm

thành phần ảo `AssertValid` của `CObject` yêu cầu đối tượng thực hiện kiểm tra tính hợp lý trên chính nó (trạng thái bên trong của đối tượng) nhờ một dãy toán tử xen, tách (`>>` hay `<<`). Hàm thành phần ảo `Dump` thực hiện hiển thị quá trình kiểm tra các thành phần dữ liệu của đối tượng, có ích khi gỡ rối chương trình và kiểm tra các đối tượng mà nó sử dụng.

`CObject` còn cung cấp các ích lợi khác cho các lớp suy diễn. Thí dụ, nạp chồng (*overload*) toán tử `new`, `delete` để tránh lỗi bộ nhớ. Tầm quan trọng của lớp MFC cơ sở nhất này sẽ trở nên rõ ràng hơn khi lập trình viên quen thuộc hơn với chúng.

1.4 CÁC LỚP KIẾN TRÚC ỨNG DỤNG

Các lớp kiến trúc ứng dụng hình thành khuôn dạng và cấu trúc của các ứng dụng MFC. `CWinApp` biểu diễn chính ứng dụng. Toàn bộ ứng dụng trên cơ sở MFC đều suy diễn lớp từ `CWinApp` và tạo các đối tượng ứng dụng từ các lớp suy diễn. `CWinApp` bao gồm các thành phần dữ liệu chung như `m_lpCmdLine`, chúng lưu con trỏ đến các tham số dòng lệnh (*command-line*) sẽ chuyển đến ứng dụng. Chúng còn chứa các hàm thành phần như `GetProfileString` để đọc tập khởi động của chương trình hay các tệp INI riêng, và hàm `InitInstance`. Ứng dụng sẽ nạp chồng hàm này để thực hiện các công việc ban đầu như lập cửa sổ.

Nhóm lớp này còn bao gồm `CDocument`, lớp cơ sở cho các tài liệu (*document*) trong các ứng dụng sử dụng kiến trúc tài liệu/quan sát (*document/view*) của MFC. Hiểu đơn giản, một “tài liệu” là biểu diễn trừu tượng của các dữ liệu chương trình.

`CDocument` cho phép ứng dụng bảo quản dữ liệu tách biệt khỏi các thành phần khác của chương trình, bao gồm các “quan sát” của dữ liệu được vẽ trên màn hình. Trong chương trình tài liệu/quan sát thì dữ liệu là đối tượng. `CWinApp`, `CDocument`, và rất nhiều các lớp khác được suy diễn từ `CCmdTarget`, nó là lớp cơ sở cho kiến trúc ánh xạ thông điệp (*message-map*) của MFC. Chương trình MFC thường không sử dụng các thủ tục cửa sổ để xử lý các thông điệp. Các hàm thành phần sẽ quản lý các thông điệp; khung ứng dụng kích hoạt hàm thành phần khi các thông điệp tới. Chỉ các lớp suy diễn từ `CCmdTarget` mới có thể nhận và xử lý thông điệp. Chính vì vậy mà toàn bộ các lớp ứng dụng, cửa sổ, tài liệu và quan sát đều được suy diễn trực tiếp hay gián tiếp từ `CCmdTarget`.

Một lớp quan trọng khác là `CWinThread`. Một đối tượng `CWinThread` biểu diễn một luồng thực hiện. Các hàm thành phần của chúng như `CreateThread`, `SetThreadPriority` và `SuspendThread` cung cấp các công cụ mà chương trình MFC sử dụng để tạo và xử lý các luồng thực hiện. Tiến trình (*process*) là chương trình thực hiện được, bao gồm mã trình và dữ liệu, chúng có riêng vùng địa chỉ nhớ và được làm chủ một số tài nguyên của hệ thống. Mỗi tiến trình phải có ít nhất một luồng thực hiện. Một luồng thực hiện là một đơn vị thực hiện, chúng gồm tập trình tự các mã lệnh, giá trị của các thanh ghi CPU và ngăn xếp. Trong Windows 16 bit thì mỗi tiến trình

chỉ có một luồng thực hiện. CWinApp tự nó suy diễn từ CWinThread, bởi vì trong môi trường đa luồng thực hiện như Windows 95/98 thì mỗi ứng dụng bắt đầu được thực hiện như một luồng.

1.5 CÁC LỚP ĐỐI TƯỢNG TRỰC QUAN

Các lớp đối tượng trực quan (*visual object*) bao gồm phần lớn các đối tượng quan sát được trên màn hình khi chương trình Windows chạy, bao gồm cửa sổ, hộp thoại, thực đơn, điều khiển và nhiều đối tượng khác. Chúng còn bao gồm cả ngữ cảnh thiết bị (*Device Context - DC*) của Windows và các đối tượng giao diện thiết bị đồ họa (*Graphics Device Interface - GDI*) như *bitmaps*, tô màu (*brushes*), và bút vẽ (*pens*). Một trong những lớp trừu tượng nhất của loại này là CView, chúng là bản đối trọng của CDocument và là lớp cơ sở cho mọi quan sát dữ liệu của ứng dụng.

Gốc của mọi lớp đối tượng trực quan là CWnd, chúng được sử dụng như lớp cơ sở cho mọi lớp cửa sổ. CWnd định nghĩa các chức năng cơ bản của cửa sổ và chứa hàng chục hàm thành phần để xử lý cửa sổ theo nhiều cách khác nhau. CWnd còn định nghĩa các trả lời mặc định của cửa sổ cho hầu hết các thông điệp. CWnd đôi khi được sử dụng trực tiếp, nhưng thường xuyên hơn là sử dụng các lớp suy diễn từ lớp suy diễn của CWnd hay lập các đối tượng từ các lớp mà MFC suy diễn từ CWnd. Lớp suy diễn từ CWnd bao gồm CFrameWnd để mô hình hoá ứng xử của cửa sổ khung ghép nối đơn tài liệu (*Single-Document Interface - SDI* - cửa sổ mức đỉnh dùng để thiết kế ứng dụng quản lý một tài liệu tại một thời điểm); CControlBar sử dụng như lớp cơ sở cho thanh công cụ (*toolbars*), thanh trạng thái (*status bars*), và các điều khiển (*controls*) của MFC; CDialog gói các chức năng của hộp thoại; CButton, CListBox, và CScrollBar tương ứng với phím, hộp danh sách, thanh trượt. Lớp mở rộng giao diện lập trình Windows CSplitterWnd là lớp phái sinh của CWnd, chúng cho phép thêm một cách dễ dàng chức năng “bẻ - splitter” các cửa sổ như ta thường thấy trong ứng dụng Windows như Visual C++.

Lớp CView cũng suy diễn từ CWnd. CView là lớp cơ sở cho tất cả các lớp quan sát trợ giúp kiến trúc ứng dụng tài liệu/quan sát. Các lớp quan sát khác bao gồm CScrollView để thêm các thanh trượt vào CView; CFormView trợ giúp quan sát theo khuôn mẫu được lập từ trước; CRecordView kết nối quan sát kiểu CFormView với đối tượng CSDL CRecordset; và lớp CEditView, sử dụng hộp soạn thảo của Windows để xem dữ liệu văn bản, đồng thời còn trợ giúp các thao tác in, tìm kiếm, thay thế, cắt dán, sao chép.

Các lớp đối tượng trực quan khác không suy diễn từ CWnd mà đi trực tiếp từ CObject. Lớp CDC chứa các chức năng ngữ cảnh thiết bị của Windows. Các hàm thành phần của CDC như DrawText và LineTo bao phủ hầu hết các hàm GDI để có khả năng vẽ trên cửa sổ. Các hàm suy diễn từ CDC như CPaintDC, CWindowDC, và CClientDC biểu diễn các trường hợp đặc biệt của lớp CDC. Thí dụ, lớp CPaintDC gọi tự động các hàm API BeginPaint và EndPaint từ cấu từ

(*constructor*) và huỷ tử (*destructor*) của chúng, làm đơn giản quá trình xử lý trong thông điệp WM_PAINT. Trong khi đó, lớp **CMetaFileDC** cung cấp khả năng làm việc với các siêu tệp của GDI.

Lớp **CGdiObject** được sử dụng như lớp cơ sở của các đối tượng GDI như bút, chổi, và cơ chữ, cùng với các lớp suy diễn từ **CGdiObject** như **CPen**, **CBrush**, và **CFont** biểu diễn chính các đối tượng. Trong MFC, thiết lập bút GDI có nghĩa là lập đối tượng từ lớp **CPen** và gọi các hàm thành phần để xác định các đặc tính của bút. Một khi đã được chọn cho DC nhờ hàm thành phần **SelectObject** của lớp **CDC**, hay suy diễn từ **CDC**, thì bút có thể được sử dụng để vẽ. Quy trình lập đối tượng, chọn nó cho DC, sử dụng nó để vẽ là mô hình GDI quen thuộc của lập trình viên C. Các lớp khác suy diễn từ **CGdiObject** bao gồm **CBitmap**; **CPalette** biểu diễn bộ mẫu lô gíc; **CRgn** chứa các đối tượng GDI mà ta đã quen biết.

Lớp cuối cùng cũng thuộc họ đối tượng quan sát là **CMenu**, chúng cung cấp ghép nối hướng đối tượng cho thực đơn. **CMenu** thuộc vào các lớp thuận tiện cho người dùng nhất, chúng bao gồm mọi chức năng mà ta muốn thực hiện với thực đơn, từ nạp đến sửa đổi thực đơn để kiểm tra các mục của thực đơn; về cơ bản là hỗn hợp của các hàm thực đơn khác nhau có trong API. Khung ứng dụng còn làm việc ngoài **CMenu** để tăng sức mạnh hệ thống thực đơn. Thí dụ, quản lý các thông điệp thực đơn có thể được đăng ký để được gọi nhờ con trỏ đến đối tượng **CCmdUI** mỗi khi thực đơn hiện ra. Các hàm thành phần của đối tượng **CCmdUI** cho phép quản lý thông điệp làm mới trạng thái mục thực đơn tương ứng - làm hiệu lực hay vô hiệu lực, đánh dấu hay không đánh dấu trước khi thực đơn hiện ra.

1.6 CÁC LỚP CHỨC NĂNG CHUNG

Không phải tất cả các lớp MFC được thiết kế để gói API của Windows. MFC còn có rất nhiều lớp khác thực hiện các công việc như quản lý các biến xâu và vào/ra tệp. Lớp cực kỳ hữu hiệu **CString** định nghĩa dữ liệu xâu có kiểu cơ sở với các toán tử chuẩn như **=**, **+=**, **<** và **>** để thực hiện gán, ghép và so sánh. Lớp **CFile** và các suy diễn từ chúng thực hiện các hàm vào/ra chuẩn như **Read** và **Write** và cài đặt chúng trong ngữ cảnh đối tượng tệp. Đối tượng **CFile** thường được truy nhập gián tiếp qua các đối tượng **CArchive**, chúng biểu diễn dòng dữ liệu nhị phân và sử dụng các toán tử nạp chồng *insertion* và *extraction* (**>>** và **<<**) để vào/ra các kiểu dữ liệu chuẩn cũng như toàn bộ đối tượng với đĩa.

CException là lớp cơ sở trừu tượng để sử dụng cơ chế quản lý lỗi *try/throw/catch* của C++. Một trong các lớp suy diễn từ **CException** là **CFileException**, chúng mô tả các lỗi phát sinh từ các thao tác tệp. Các thành phần dữ liệu chung chứa mã lỗi. Lớp khác hay được sử dụng là **CMemoryException** cho lỗi tràn bộ nhớ. Lớp **CArchiveException** cho lỗi vào/ra, **COleException** cho lỗi OLE, và **CResourceException** cho lỗi khi nạp tài nguyên của Windows.

MFC còn cung cấp các hàm tổng thể như **AfxThrowFileException** và **AfxThrowOleException**. Các lớp tập hợp (*collection*) của MFC cung cấp thư viện lớp trợ giúp các cấu trúc dữ liệu chung như các mảng và danh sách liên kết. Các lớp mảng – **CByteArray**, **CIntArray**, **CStringArray**, ...v.v. – gói các mảng của kiểu dữ liệu cơ sở và các đối tượng có thể thay đổi động độ lớn, không như các mảng tĩnh trong chương trình C. Các lớp mảng là tư tưởng để lưu dữ liệu tài liệu của ứng dụng. Thí dụ, tệp soạn thảo có thể cài đặt như đối tượng **CByteArray**. Trong các lớp danh sách như **CStringList** và **CObList** có cài đặt danh sách liên kết hai chiều, các thành phần của chúng chứa con trỏ đến thành phần trước và sau trong danh sách. Các hàm thành phần như **InsertBefore**, **InsertAfter**, và **RemoveAt** thêm các thao tác xen, bỏ. Các lớp ánh xạ chứa bảng đối tượng và dữ liệu kiểu khóa hay đã được ánh xạ bằng các kiểu dữ liệu khác. Lớp **CMapWordToPtr** tạo khóa từng biến trỏ nơi lưu giá trị từ 16-bit do nơi gọi cung cấp. Cho trước giá trị khóa, con trỏ tương ứng dễ dàng được khai thác nhờ gọi hàm **Lookup** của đối tượng. MFC cũng cung cấp các bản tổng quát của các lớp này dưới hình thức mẫu lớp (*template*) của C++. Cái đáng kể nhất của các lớp này là chúng trợ giúp việc vào/ra, toàn bộ mảng, danh sách, bảng có thể khai thác từ đĩa chỉ bằng một dòng mã lệnh. Các lớp này sẽ được sử dụng nhiều lần khi phát triển ứng dụng.

Các lớp đa năng khác là **CPoint**, **CSize**, **CRect**, **CTime**, và **CTimeSpan**. **CPoint** và **CRect** biểu diễn các điểm và hình chữ nhật, thật hữu hiệu vì chúng thường làm tham số cho các hàm thành phần MFC. Các lớp chẩn đoán **CDumpContext** và **CMemoryState** hiển thị vùng nhớ và phát hiện lỗi bộ nhớ gây ra khi không xóa được các đối tượng lập bằng toán tử *new*.

1.7 CÁC HÀM AfxXxx

Không phải tất cả các hàm mà MFC cung cấp là hàm thành phần lớp. MFC còn cho API dưới dạng các hàm tổng thể với tên bắt đầu là **Afx**. (A và f trong **Afx** là “*application framework*”). Các hàm này được gọi bất kỳ đâu trong ứng dụng MFC, không như hàm thành phần là chúng phải có đối tượng hay con trỏ đối tượng.

Bảng 1.1 liệt kê một vài hàm **AfxXxx**. Hàm **AfxBeginThread** thường được gọi thay **CWinThread::CreateThread** vì chúng tạo và thực hiện luồng thực hiện trong một bước. Hàm **AfxMessageBox** tương đương với hàm **MessageBox** của Windows và không như **CWnd::MessageBox**, chúng có thể được gọi mà không cần con trỏ đến lớp suy diễn từ **CWnd**. **AfxFormatString1** thường được nối với **AfxMessageBox** để thực hiện tạo dạng xâu kiểu **printf**. **AfxGetApp** dùng để lấy con trỏ đến đối tượng ứng dụng khai báo trong tệp khác, và **AfxGetInstanceHandle** lấy chỉ danh làm tham số của hàm API của Windows. Cuối cùng, **AfxRegisterWndClass** được sử dụng khi MFC không thực hiện đăng ký lớp của sổ mặc định.

Bảng 1.1 Các mẫu hàm AfxXxx

Tên hàm	Mô tả
AfxAbort	Kết thúc vô điều kiện một ứng dụng. Thông thường được gọi khi có lỗi vô phương cứu chữa.
AfxBeginThread	Lập luồng thực hiện mới và bắt đầu thực hiện chúng.
AfxEndThread	Kết thúc luồng thực hiện đang thực hiện.
AfxFormatString1	Thay thế một xâu vào vị trí %1 trong xâu tài nguyên và sao chép xâu kết quả vào đối tượng CString.
AfxMessageBox	Hiển thị hộp thông báo của Windows.
AfxGetApp	Cho lại con trỏ đến đối tượng ứng dụng.
AfxGetInstanceHandle	Cho lại handle chỉ ra bản ứng dụng hiện hành.
AfxRegisterWndClass	Đăng ký một lớp cửa sổ riêng cho ứng dụng MFC.

1.8 THÍ DỤ HELLO1

Bây giờ ta đã có thể xây dựng được một ứng dụng MFC đơn giản. Thí dụ **HELLO1** mô tả một vài nguyên lý cơ bản của chương trình MFC nhằm hiển thị xâu “Hello, MFC” trong cửa sổ. Trong đó, các lớp CWinApp và CFrameWnd được sử dụng. Tiếp theo sẽ bàn luận vấn đề ánh xạ thông điệp (*message map*), đó là cơ chế để MFC dẫn các thông điệp mà ứng dụng nhận được vào các hàm thành phần trong MFC và các lớp suy diễn của MFC. Cuối cùng sẽ xem xét lớp CPaintDC, dùng để vẽ đồ thị và văn bản trong cửa sổ khi có thông điệp WM_PAINT, và lớp CRect, một trong các lớp đa năng.

**Hình 1.2** Kết quả của chương trình HELLO1

Chương trình 1.1 bao gồm hai tệp mã nguồn của HELLO1. HELLO1.H chứa các khai báo của hai lớp suy diễn mà HELLO1 sử dụng. HELLO1.CPP chứa mã gốc C++, là cài đặt hai lớp này.

Hình 1.2 là kết quả chạy chương trình 1.1.

Chương trình 1.1 HELLO1

Tệp HELLO1.H

```
class CMyApp : public CWinApp
{
public:
    BOOL InitInstance ();
};
class CMainWindow : public CFrameWnd
{
public:
    CMainWindow ();
protected:
    afx_msg void OnPaint ();
    DECLARE_MESSAGE_MAP ()
};
```

//-----

Tệp HELLO1.CPP

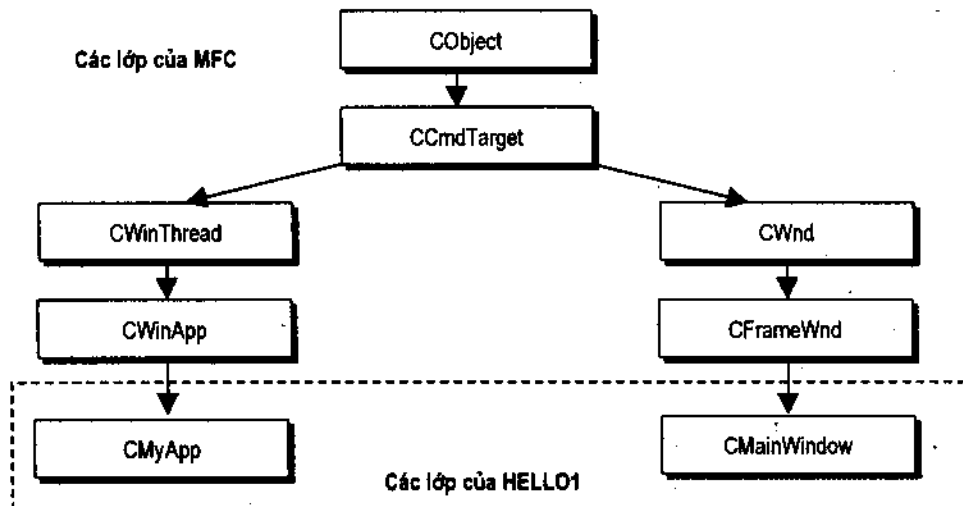
```
#include <afxwin.h>
#include "hello1.h"
CMyApp myApp;
// Các hàm thành phần của lớp CMyApp
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}
// ánh xạ thông điệp và các hàm thành phần của CMainWindow
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_PAINT ()
END_MESSAGE_MAP ()
//-----
CMainWindow::CMainWindow ()
{
    Create (NULL, "Hello1");
```

```

}
//-----
void CMainWindow::OnPaint ()
{
    CRect rect;
    CPaintDC dc (this);
    GetClientRect (rect);
    dc.DrawText ("Hello, MFC", -1, rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
}
//-----

```

Chương trình 1.1 cho thấy MFC cung cấp rất nhiều lớp. Khi gộp chúng lại với nhau ta sẽ có khung làm việc của chương trình ứng dụng. Lập trình viên chỉ việc xen thêm các đoạn mã trình thực hiện các yêu cầu riêng để có được ứng dụng đầy đủ. Hình 1.3 là sơ đồ quan hệ của các lớp cơ bản tạo nên ứng dụng HELLO1.



Hình 1.3 Các lớp của ứng dụng HELLO1

1.9 ĐỐI TƯỢNG ỨNG DỤNG

Lõi của ứng dụng trên cơ sở MFC là đối tượng ứng dụng, lớp của chúng được suy diễn từ lớp CWinApp của MFC. CWinApp cung cấp các hàm thành phần để khởi động chương trình ứng dụng, thực hiện các thao tác chung của ứng dụng như ghi đĩa, đồng thời chứa các hàm nạp chồng như Run. Mỗi ứng dụng MFC có một và chỉ một đối tượng ứng dụng mà thôi, đối tượng này phải được

khai báo là tổng thể (*global*).

Mỗi ứng dụng MFC sẽ suy diễn một lớp từ CWinApp và nạp chồng hàm thành phần InitInstance để tạo lập cửa sổ chính của ứng dụng. Lớp ứng dụng suy diễn của thí dụ này gọi là CMyApp và đối tượng của chúng được khai báo tổng thể. Dòng lệnh tạo đối tượng như sau:

```
CMyApp myApp;
```

đặt trong tệp HELLO1.CPP. Khai báo CMyApp trong HELLO1.H như sau:

```
class CMyApp : public CWinApp
{
public:
    BOOL InitInstance ();
};
```

Hàm InitInstance chỉ được gọi một lần, ngay sau khi ứng dụng bắt đầu chạy và trước khi cửa sổ ứng dụng được hiển thị. Cài đặt mặc định của InitInstance - trong CWinApp - chỉ chứa một lệnh:

```
return TRUE;
```

Vậy mọi chương trình MFC đều phải nạp chồng hàm InitInstance và tạo cửa sổ để giao diện với thế giới bên ngoài. InitInstance là nơi thuận tiện để thực hiện mọi công việc vào thời điểm chương trình khởi động. Nếu khởi động tốt thì InitInstance cho lại TRUE.

1.10 HÀM InitInstance

Hàm InitInstance trong chương trình HELLO1.CPP lập cửa sổ HELLO1 bằng cách khai báo đối tượng của lớp CMainWnd. Dòng lệnh

```
m_pMainWnd = new CMainWnd ();
```

khởi tạo đối tượng cửa sổ và gán chúng vào đối tượng ứng dụng bằng cách lưu địa chỉ vào myApp.m_pMainWnd. Sau khi cửa sổ được lập, InitInstance hiển thị chúng (nhớ rằng cửa sổ chỉ nhìn thấy khi lập chúng với thuộc tính WS_VISIBLE) nhờ gọi các hàm của đối tượng cửa sổ ShowWindow và UpdateWindow, tương tự như các hàm API ShowWindow và UpdateWindow đã được khảo sát trước đây:

```
m_pMainWnd->ShowWindow (m_nCmdShow);
m_pMainWnd->UpdateWindow ();
```

Để gọi hàm API của Windows từ MFC, hãy đặt toán tử :: vào trước tên hàm như sau:

```
::UpdateWindow (hwnd);
```

Từ đây, ta phân biệt API của Windows với các hàm thành phần của MFC nhờ toán tử ::.

ShowWindow chấp nhận một tham số, đó là số nguyên chỉ ra trạng thái hiển thị của sổ. HELLO1 chuyển giá trị lưu trong biến m_nCmdShow của đối tượng ứng dụng cho ShowWindow, nơi lưu tham số nCmdShow mà WinMain đã nhận được từ Windows. m_nCmdShow thường có giá trị SW_SHOWNORMAL, cho biết cửa sổ hiển thị bình thường, không ở trạng thái to nhất/nhỏ nhất. InitInstance luôn sử dụng m_nCmdShow thay vì giá trị cứng nhắc SW_XXX.

UpdateWindow hoàn thiện công việc ShowWindow bằng yêu cầu vẽ lại vùng làm việc (*client area*). Khi hoàn thành công việc, InitInstance cho giá trị TRUE để ứng dụng có thể tiếp tục chạy.

1.11 CÁC THÀNH PHẦN CỦA LỚP CWinApp

Các biến m_pMainWnd và m_nCmdShow là hai thành phần dữ liệu của CWinApp (bảng 1.2).

Bảng 1.2 Các thành phần dữ liệu của CWinApp

Tên biến	Mô tả
m_pszAppName	Con trỏ đến tên ứng dụng. Mặc định là tên tệp .EXE. Có thể lấy được nhờ hàm AfxGetAppName của MFC.
m_hInstance	Chỉ danh bản hiện hành của ứng dụng. Tương đương hInstance chuyển cho WinMain. Có thể lấy được nhờ hàm AfxGetInstanceHandle và AfxGetResourceHandle của MFC.
m_hPrevInstance	Chỉ danh của bản sao trước; luôn có giá trị NULL trong các ứng dụng Win32.
m_lpCmdLine	Con trỏ đến dòng lệnh chương trình. Tương đương lpCmdLine chuyển cho WinMain.
m_nCmdShow	Trạng thái ban đầu mà cửa sổ ứng dụng cần hiển thị. Tương đương với tham số nCmdShow chuyển cho WinMain.
m_bHelpMode	Khác 0 nếu ứng dụng trong chế độ trợ giúp ngữ cảnh, được nhập khi nhấn Shift-F1; trường hợp khác có giá trị 0.
m_pActiveWnd	Nếu khác NULL, chứa con trỏ đến đối tượng của sổ chính của ứng dụng chứa OLE nếu ứng dụng là trình dịch vụ OLE được kích hoạt ngay trong trình chứa.
m_pszExeName	Con trỏ đến tên mô đun chương trình (thí dụ, HELLO1).
m_pszHelpFilePath	Con trỏ đến đường dẫn và tên của tệp trợ giúp (thí dụ, C:\MFCAPPS\PART 1\HELLO1.HLP).
m_pszProfileName	Con trỏ đến tệp INI của chương trình (Thí dụ, HELLO1.INI).
m_pszRegistryKey	Con trỏ đến tên khoá lưu tham số bố trí (setting) ứng dụng trong đăng ký hệ thống. Lập bởi gọi SetRegistryKey.

Nếu ta có con trỏ đến đối tượng ứng dụng (đối tượng của CWinApp) thì có thể thăm nhập các thành phần khác nữa. Nếu chưa có con trỏ này thì có thể có nó nhờ hàm AfxGetApp. Thí dụ, lệnh:

```
AfxMessageBox (AfxGetApp ()->m_pszHelpFilePath);
```

sử dụng AfxGetApp để thăm nhập đường dẫn lưu trong m_pszHelpFilePath và AfxMessageBox để hiển thị chúng trong hộp thông báo. Một vài thành phần dữ liệu của đối tượng ứng dụng có thể thăm nhập qua các hàm Afx. Hàm AfxGetAppName, cho lại con trỏ đến m_pszAppName và

AfxGetInstanceHandle cho lại chỉ danh của bản đang chạy lưu trong m_hInstance. Chỉ danh của bản đang chạy cần thiết khi gọi các API của Windows từ chương trình MFC, nhưng nếu chỉ sử dụng các hàm MFC thì chúng không bao giờ được dùng đến.

Bảng 1.3 Các hàm thành phần của CWinApp

<i>Tên hàm</i>	<i>Mô tả</i>
SetDialogBkColor	Lập màu nền mặc định cho các hộp thoại
LoadCursor	Nạp tài nguyên con chạy từ tệp chương trình EXE.
LoadStandardCursor	Nạp tài nguyên con chạy do Windows cung cấp.
LoadIcon	Nạp tài nguyên biểu tượng từ tệp chương trình EXE.
LoadStandardIcon	Nạp tài nguyên biểu tượng do Windows cung cấp.
GetProfileInt	Đọc giá trị nguyên vào tệp INI hay đăng ký
WriteProfileInt	Viết giá trị nguyên vào tệp INI hay đăng ký
GetProfileString	Đọc giá trị xâu vào tệp INI hay đăng ký
WriteProfileString	Viết giá trị xâu vào tệp INI hay đăng ký
SetRegistryKey	Điều khiển đối tượng ứng dụng để sử dụng đăng ký thay tệp INI để lưu trạng thái chương trình, và chỉ ra tên của khoá đăng ký rằng sẽ nhận ra thông tin được lưu ở đó.

Các hàm GetProfileInt và GetProfileString đọc giá trị số hay xâu từ tệp INI của ứng dụng. Nếu hàm SetRegistryKey đã được gọi thì thông tin được lấy từ phần riêng của ứng dụng trong sổ đăng ký (*registry*) Windows 95/98 (hay Windows NT). Các hàm WriteProfileInt và WriteProfileString là đối xứng với GetProfileInt và GetProfileString để lưu các giá trị. Một trong các ứng dụng của hàm này là lưu lại các thông tin như kích thước và vị trí của cửa sổ ứng dụng chính cho lần khởi động sau. Thí dụ cài đặt InitInstance sau đây sử dụng GetProfileInt để đọc tọa độ từ sổ đăng ký, sau đó sử dụng các số liệu này để nạp các giá trị ban đầu cho cửa sổ chính của ứng dụng:

```
UINT x, y, width, height;
SetRegistryKey("MyCompanyName");
x = GetProfileInt("WndProperties", "XPos", 0);
y = GetProfileInt("WndProperties", "YPos", 0);
width = GetProfileInt("WndProperties", "Width", 320);
height = GetProfileInt("WndProperties", "Height", 240);
m_pMainWnd = new CMainWindow();
m_pMainWnd->SetWindowPos(NULL, x, y, width, height, SWP_NOZORDER);
m_pMainWnd->ShowWindow(m_nCmdShow);
m_pMainWnd->UpdateWindow();
return TRUE;
```

SetWindowPos là hàm thành phần của lớp CWnd để dịch chuyển, thay đổi kích cỡ cửa sổ. Muốn sử dụng đoạn chương trình này, phải nạp chồng hàm ExitInstance của CWinApp, sử dụng WriteProfileInt để lưu các giá trị của cửa sổ vào sổ đăng ký.

Như đã biết, ta cũng có thể viết ứng dụng MFC không sử dụng đối tượng suy diễn từ CWinApp, nhưng đó là trường hợp đặc biệt. Trong trường hợp này, để có thể sử dụng các đối tượng ứng dụng thì lập trình viên phải viết lấy hàm WinMain, bởi vì hàm WinMain của MFC đã gắn chặt với các lời gọi đến các đối tượng ứng dụng.

1.12 NẠP CHỒNG HÀM CWinApp

InitInstance là một hàm thành phần ảo của CWinApp mà ta có thể nạp chồng. Một số hàm khác cũng có khả năng được nạp chồng (xem bảng 1.4). ExitInstance là hàm hay được nạp chồng nhất sau hàm InitInstance. Chúng được gọi ngay trước khi ứng dụng kết thúc, đây là nơi tốt nhất để thực hiện việc dọn dẹp ứng dụng, giải phóng bộ nhớ đã cấp phát ở hàm InitInstance và lưu các thông số chương trình lên đĩa. Cài đặt mặc định hàm ExitInstance của lớp CWinApp có gọi hàm bảo vệ SaveStdProfileSettings để lưu một số tham số chương trình cho khung ứng dụng. Giá trị cho lại của ExitInstance là mã ra của WinMain.

Bảng 1.4 Các hàm nạp chồng của CWinApp

<i>Tên hàm</i>	<i>Mô tả</i>
InitInstance	Được gọi ngay sau khi ứng dụng khởi động. Nạp chồng để tạo cửa sổ chính của ứng dụng và thực hiện vài khởi động khác (hầu như luôn nạp chồng).
ExitInstance	Được gọi ngay trước khi kết thúc ứng dụng. Nạp chồng để phá hủy đối tượng và các tài nguyên cấp phát ở InitInstance hay thực hiện các dọn dẹp khác (thường nạp chồng).
OnIdle	Được gọi trong vòng lặp thông điệp ứng dụng khi không có thông điệp để xử lý, khi ứng dụng rỗi. Nạp chồng để thực hiện xử lý nền (đôi khi nạp chồng).
Run	Được gọi để "chạy" ứng dụng bằng thực hiện vòng lặp thông điệp. Nạp chồng để cài đặt vòng lặp thông điệp riêng (ít khi nạp chồng).
PreTranslateMessage	Được gọi khi ở trong vòng lặp thông điệp để xử lý phím lệnh liên quan đến thực đơn (ít khi nạp chồng).

Các hàm nạp chồng hữu ích khác bao gồm OnIdle, Run, và PreTranslateMessage. OnIdle quản lý việc thực hiện tiến trình nền như chờ in ra máy in, hay dọn dẹp các cấu trúc dữ liệu trong bộ nhớ. Vì OnIdle được gọi khi ứng dụng rỗi rãi, đó là cơ chế tuyệt vời để thực hiện tiến trình khác mà không cần thread thứ hai. Run và PreTranslateMessage không hữu hiệu như OnIdle cho nên ít khi nạp chồng, tuy nhiên chúng được nạp chồng khi thay thế vòng lặp thông điệp của CWinApp bằng vòng lặp riêng để cho phép xử lý các thông điệp đặc biệt trước khi chuyển đến chương trình.

1.13 MFC SỬ DỤNG ĐỐI TƯỢNG ỨNG DỤNG NHU THẾ NÀO?

Nếu muốn xem ứng dụng MFC làm việc ra sao và nhiệm vụ của MFC là gì thì phải xem đến mã nguồn. Khi cài đặt Visual C++, ta đã có ngay mã nguồn cho MFC. Một trong các tệp nguồn có tên là WINMAIN.CPP, chúng chứa hàm WinMain mà MFC cung cấp cho ứng dụng. Hàm này sử dụng đối tượng ứng dụng, vì vậy chúng phải được khai báo là tổng thể. Các biến và đối tượng tổng thể

được lập trước khi WinMain thực hiện.

Ngay sau khi khởi động, WinMain do MFC cung cấp sẽ gọi hàm AfxWinInit để khởi động khung ứng dụng. Nhiệm vụ của AfxWinInit là sao chép các tham số hInstance, hPrevInstance, lpCmdLine, và nCmdShow, đã chuyển cho WinMain, tới thành phần dữ liệu của đối tượng ứng dụng và đăng ký một vài lớp cửa sổ. Sau đó WinMain gọi hàm InitApplication của đối tượng ứng dụng nếu hPrevInstance là NULL, tiếp theo là gọi hàm InitInstance. Vì hPrevInstance luôn có giá trị NULL trong môi trường Win32, một ứng dụng MFC 32-bit có thể dễ dàng sử dụng InitApplication để tự khởi động như InitInstance. Trong Windows 16-bit, InitApplication chỉ được gọi khi không có bản sao của chương trình đang chạy. Nếu bất kỳ hàm nào trong số các hàm AfxWinInit, InitApplication, và InitInstance mà cho lại 0 thì có lỗi, khi ấy WinMain gọi AfxWinTerm để dọn dẹp và kết thúc bằng lệnh return.

Chỉ khi toàn bộ các hàm trên cho lại TRUE, WinMain mới thực hiện bước tiếp theo. Lệnh:

```
pApp->Run();
```

gọi hàm Run của đối tượng ứng dụng, chúng thực hiện vòng lặp thông điệp và gọi OnIdle khi hàng chờ thông điệp rỗng. Vòng lặp thông điệp lặp lại cho đến khi gặp WM_QUIT trong hàng đợi, thời điểm này Run nhảy ra vòng lặp, gọi ExitInstance, và trở về WinMain. WinMain kết thúc bằng gọi AfxWinTerm để dọn dẹp rồi thực hiện lệnh return để kết thúc ứng dụng.

1.14 ĐỐI TƯỢNG CỦA SỔ

Đối tượng quan trọng thứ hai trong ứng dụng HELLO1 là đối tượng cửa sổ. Chương trình C thường lập cửa sổ chính của nó bằng gọi hàm ::RegisterClass để đăng ký lớp cửa sổ và hàm ::CreateWindow để tạo cửa sổ trên cơ sở lớp đó. Chương trình MFC lập cửa sổ bằng khai báo đối tượng cửa sổ C++. Trong HELLO1.CPP việc này được thực hiện như sau:

```
m_pMainWnd = new CMainWindow ();
```

Toán tử new lập đối tượng, và cấu tử (*constructor*) của đối tượng sẽ kiến tạo cửa sổ mà ta thấy trên màn hình. Con trỏ do new cho lại có thể sử dụng để tham nhập các hàm thành phần của đối tượng cửa sổ.

Lớp CMainWindow được khai báo như sau trong tệp HELLO1.H:

```
class CMainWindow : public CFrameWnd
{
public:
    CMainWindow ();
protected:
    afx_msg void OnPaint ();
```

```
DECLARE_MESSAGE_MAP()
};
```

Lớp cơ sở của CMainWindow là CFrameWnd, và nó được suy diễn từ CWnd, để tạo lập cửa sổ chính của ứng dụng SDI. MFC gọi chúng là cửa sổ khung. Trong lớp CMainWindow ta thấy khai báo hàm cấu tử (CMainWindow) và hàm thành phần tên OnPaint. Ngoài ra còn thấy dòng lệnh là DECLARE_MESSAGE_MAP. Hàm thành phần OnPaint xử lý thông điệp WM_PAINT gửi đến cửa sổ. DECLARE_MESSAGE_MAP là macro dùng để khai báo ánh xạ (*map*) thông điệp của lớp.

1.15 TẠO LẬP CỬA SỔ

Hàm cấu tử của CMainWindow chỉ có một dòng lệnh, đó là lời gọi hàm Create của đối tượng cửa sổ:

```
Create(NULL, "Hello1");
```

Hàm này tạo cửa sổ khung. Đây là một trong khoảng 20 hàm thành phần mà CFrameWnd định nghĩa thêm ngoài các hàm mà nó thừa hưởng từ các lớp cơ sở. Mẫu hàm của CFrameWnd::Create như sau:

```
BOOL Create(LPCTSTR lpszClassName,
            LPCTSTR lpszWindowName,
            DWORD dwStyle = WS_OVERLAPPEDWINDOW,
            const RECT& rect = rectDefault,
            CWnd* pParentWnd = NULL,
            LPCTSTR lpszMenuName = NULL,
            DWORD dwExStyle = 0,
            CCreateContext* pContext = NULL);
```

Hàm trên cho thấy sáu trong tám tham số được định nghĩa mặc định cho hàm Create. HELLO1 chỉ xác định giá trị cho hai tham số đầu, các giá trị còn lại chấp nhận mặc định. Tham số đầu của Create, lpszClassName, là tên của lớp mà cửa sổ sẽ lập theo. Nếu tham số này có giá trị NULL thì cửa sổ khung mặc định được lập trên cơ sở lớp đã được đăng ký tự động khi ứng dụng MFC bắt đầu chạy. Nếu muốn, ứng dụng có thể đăng ký thêm các lớp cửa sổ nhờ hàm API ::CreateWindow hay hàm của MFC AfxRegisterWndClass. Tham số lpszWindowName chỉ ra tên sẽ xuất hiện trên đỉnh của cửa sổ. Thông thường đây là tên của ứng dụng, đôi khi chúng còn chứa cả tên của tài liệu. Tham số dwStyle cho kiểu của cửa sổ. Mặc định là WS_OVERLAPPEDWINDOW, chúng lập cửa sổ chuẩn chứa thanh tiêu đề (*title bar*); thực đơn hệ thống (*system menu*); các phím dân to nhất, co nhỏ nhất và đóng (*minimize, maximize, close buttons*); biên co dãn (*resizing border*). Bảng 1.5 liệt kê các giá trị có thể cho tham số dwStyle, chúng có cùng giá trị mà lập trình viên C đã quen biết khi gọi hàm CreateWindow. Thí dụ, ta thêm thanh trượt ngang vào cửa sổ nhờ tổ hợp WS_OVERLAPPEDWINDOW và WS_VSCROLL bằng toán tử OR:

```
Create (NULL, "Hello1", WS_OVERLAPPEDWINDOW | WS_VSCROLL);
```

Nếu muốn lập cửa sổ giống WS_OVERLAPPEDWINDOW, nhưng thiếu phím *minimize*, ta có thể viết:

```
Create (NULL, "Hello1", WS_OVERLAPPED | WS_CAPTION |  
WS_SYSMENU | WS_MAXIMIZEBOX | WS_THICKFRAME);
```

Bảng 1.5. Các kiểu chuẩn của cửa sổ

Kiểu	Mô tả
WS_BORDER	Tạo cửa sổ với khung mỏng
WS_CAPTION	Tạo cửa sổ với thanh tiêu đề
WS_CHILD	Tạo cửa sổ con. Sử dụng để tạo điều khiển như phím và các cửa sổ tài liệu MDI.
WS_CLIPCHILDREN	Chỉ ra rằng vùng cửa sổ con chiếm là vùng clip khi cửa sổ cha vẽ lại.
WS_CLIPSIBLINGS	Sử dụng cho cửa sổ con để chỉ rằng con có thể clipp với con khác khi vẽ.
WS_DISABLED	Cửa sổ sẽ bị vô hiệu lực.
WS_DLGMFRAME	Lập cửa sổ có viền như hộp thoại.
WS_GROUP	Cửa sổ sẽ là thứ nhất trong nhóm cửa sổ mà ta có thể chuyển bằng tab. Sử dụng ban đầu cho cửa sổ con trong hộp thoại.
WS_HSCROLL	Lập cửa sổ có thanh trượt ngang.
WS_MAXIMIZE	Lập cửa sổ maximized.
WS_MAXIMIZEBOX	Lập cửa sổ có phím maximize trên thanh tiêu đề.
WS_MINIMIZE	Lập cửa sổ minimized.
WS_MINIMIZEBOX	Lập cửa sổ có phím minimize trên thanh tiêu đề.
WS_OVERLAPPED	Lập cửa sổ maximize và không có cửa sổ cha.
WS_OVERLAPPEDWINDOW	Lập cửa sổ với các kiểu sau WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, và WS_MAXIMIZEBOX.
WS_POPUP	Lập cửa sổ như WS_OVERLAPPED nhưng không có thanh tiêu đề.
WS_POPUPWINDOW	Lập cửa sổ popup với các kiểu sau WS_POPUP, WS_BORDER, và WS_SYSMENU.
WS_SYSMENU	Lập cửa sổ với hộp thực đơn hệ thống thanh ghi tiêu đề.
WS_TABSTOP	Cửa sổ có thể được chọn bằng phím tab. Sử dụng ban đầu cho cửa sổ con trong hộp thoại.
WS_THICKFRAME	Lập cửa sổ với resizing border.
WS_VISIBLE	Cửa sổ ban đầu nhìn thấy ngay. Nếu không có kiểu này mà muốn thấy cửa sổ thì phải gọi hàm ShowWindow.
WS_VSCROLL	Lập cửa sổ có thanh trượt dọc.

Các kiểu cửa sổ được đặt trong tham số dwExStyle. Các kiểu mở rộng đã thấy trong Windows 3.1 với hàm ::CreateWindowEx. Các tham số của kiểu mở rộng đặt trong đối thứ nhất của hàm. Trong MFC, các kiểu này được đặt trong biến dwExStyle và hàm CFrameWnd::Create sẽ chuyển đến ::CreateWindowEx. Bảng 1.6 liệt kê một vài kiểu mở rộng được sử dụng cho Windows 95/98 và Windows NT.

Bảng 1.6 Các kiểu mở rộng của cửa sổ

<i>Kiểu</i>	<i>Mô tả</i>
WS_EX_ACCEPTFILES	Tạo cửa sổ chấp nhận di-và-thả tệp
WS_EX_DLGMODALFRAME	Tạo cửa sổ với viền dày kiểu đối thoại
WS_EX_NOPARENTNOTIFY	Chỉ ra là cửa sổ con không thông báo cho cha nó khi được lập hay phá hủy
WS_EX_TOPMOST	Tạo cửa sổ luôn ở trên các cửa sổ khác mặc dù nó không tích cực
WS_EX_TRANSPARENT	Tạo cửa sổ luôn ở dưới các cửa sổ khác mặc dù nó tích cực

Giá trị WS_EX_TOPMOST dùng để lập cửa sổ “luôn ở trên đỉnh (stay-on-top)” luôn được nhìn thấy ngay cả khi dấu nó sau các cửa sổ khác trên màn hình. WS_EX_DLGMODALFRAME lập cửa sổ có khung hộp thoại mỏng và không thay đổi kích cỡ. Sau tham số dwStyle là tham số rect (là tham chiếu C++ đến đối tượng CRect, hay địa chỉ của cấu trúc RECT của C) kích cỡ và vị trí ban đầu của cửa sổ được xác định. Mặc định là rectDefault, là biến thành phần tĩnh của lớp CFrameWnd cho Windows biết chọn giá trị mặc định nào cho kích cỡ và vị trí của cửa sổ. Nếu muốn, ta có thể chỉ ra kích cỡ, vị trí cửa sổ ban đầu và nạp vào đối tượng CRect rồi chuyển nó cho hàm Create.

Thí dụ, góc trên trái của cửa sổ rộng 320, cao 240 sẽ đặt vào vị trí 32, 64 của màn hình nhờ các dòng lệnh như sau:

```
CRect rect (32, 64, 352, 304);
Create (NULL, "Hello1", WS_OVERLAPPEDWINDOW, rect);
```

Chú ý, hai tham số sau của hàm rect trên đây cho biết góc dưới, phải chứ không phải độ rộng, cao.

Tham số pParentWnd của hàm Create chỉ ra cửa sổ cha. Windows lưu các cửa sổ do chương trình ứng dụng tạo ra trong một cơ sở dữ liệu phân cấp giữ mối quan hệ của từng cửa sổ với các cửa sổ khác trong hệ thống. Trong quan hệ cha-con, cửa sổ con sẽ gửi cho cửa sổ cha sự kiện có tác động đến cửa sổ cha. Quan hệ hộp thoại với các điều khiển là thí dụ quan hệ cha-con. Khi nhấn phím (*điều khiển*), hộp thoại được thông báo để trả lời phím bị nhấn. Thông báo lại (*notification*) là thông điệp do cửa sổ con gửi đến cửa sổ cha. Khi tạo cửa sổ mức đỉnh như cửa sổ HELLO1, pParentWnd nhận giá trị NULL để chỉ ra rằng chúng không có cửa sổ cha. (Có thể nói mỗi cửa sổ đều có cửa sổ cha, khi pParentWnd=NULL có nghĩa là cửa sổ cha là cửa sổ cả màn hình). Khi cửa sổ con được lập thì biến pParentWnd lưu con trỏ đến đối tượng cửa sổ cha.

Tham số lpszMenuName của hàm Create chỉ ra thực đơn liên quan đến cửa sổ. Giá trị NULL của tham số có nghĩa là cửa sổ không có thực đơn.

Tham số cuối của CFrameWnd::Create là pContext, chứa con trỏ đến cấu trúc CCreateContext mà khung ứng dụng sử dụng khi khởi động ứng dụng kiến trúc tài liệu/quan sát.

1.16 VẼ TRÊN VÙNG LÀM VIỆC CỦA CỬA SỔ

Như mọi ứng dụng Windows khác, HELLO1 không vẽ ngay trên màn hình mà công việc này được thực hiện khi có thông điệp WM_PAINT từ Windows báo cho biết phải làm tươi cửa sổ. Thông điệp WM_PAINT được phát sinh từ nhiều lý do khác nhau. Lớp cửa sổ mặc định mà MFC đăng ký cho các cửa sổ khung chứa thuộc tính CS_HREDRAW và CS_VREDRAW, vậy thông điệp WM_PAINT được tự động phát sinh mỗi khi cửa sổ thay đổi kích cỡ theo chiều ngang hay chiều dọc.

Trong HELLO1, hàm CMainWindow::OnPaint thực hiện toàn bộ việc vẽ, chúng được kích hoạt từ thông điệp WM_PAINT. Khi OnPaint được kích hoạt, vùng làm việc của cửa sổ sẽ được xóa bằng màu nền mà MFC đã đăng ký cho lớp cửa sổ. (Trước mỗi thông điệp WM_PAINT, Windows xử lý thông điệp WM_ERASEBKGD. Trình quản lý WM_ERASEBKGD mặc định lấy màu nền từ cấu trúc WNDCLASS). Như vậy, OnPaint không cần thời gian để xóa toàn bộ nội dung cũ của cửa sổ.

Hàm OnPaint bắt đầu bằng tạo lập đối tượng kiểu CRect và lập đối tượng của CPaintDC với tên dc:

```
CRect rect;
CPaintDC dc (this);
```

Lớp CPaintDC của MFC gói ngữ cảnh thiết bị của Windows, chúng được suy diễn từ lớp CDC. Chúng có thêm một khả năng quan trọng là tự gọi hàm BeginPaint và EndPaint để ứng dụng không phải làm việc này nữa. BeginPaint được cấu tử của lớp gọi, còn EndPaint được huỷ từ gọi. Trong MFC, việc vẽ trên màn hình luôn phải thông qua một đối tượng CDC nào đó, nhưng CPaintDC chỉ được sử dụng bên trong OnPaint. Hơn thế nữa, việc tạo các đối tượng CPaintDC trong ngăn xếp là rất quan trọng, vì huỷ từ của nó sẽ được tự động gọi khi OnPaint kết thúc. Có thể tạo đối tượng CPaintDC bằng toán tử new nếu muốn, nhưng nhớ rằng phải xóa đối tượng đó trước khi OnPaint kết thúc. Nếu không thì chương trình sẽ bị treo vì vòng lặp vô hạn gây ra do thông điệp WM_PAINT được phát sinh mãi mãi trong hàng đợi. Tiếp theo, OnPaint gọi hàm thành phần GetClientRect, (lớp CMainWindow của chúng thừa hưởng trực tiếp từ CWnd), để sao chép tọa độ vùng làm việc của cửa sổ hiện hành vào rect:

```
GetClientRect (rect);
```

Sau đó chúng gọi hàm DrawText của đối tượng dc để hiển thị xâu ký tự "Hello, MFC" trong cửa sổ:

```
dc.DrawText ("Hello, MFC", -1, rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
```

Hàm DrawText là hàm đa năng để hiển thị xâu ký tự trong cửa sổ. Chúng có bốn tham số: con trỏ đến xâu sẽ hiển thị, tổng số ký tự trong xâu (hay -1 nếu xâu có kết thúc là 0x00), địa chỉ của

cấu trúc RECT hay đối tượng CRect, các cờ cho khuôn dạng văn bản sẽ hiển thị. Bảng 1.7 liệt kê các giá trị khuôn dạng mà DrawText chấp nhận. Các giá trị sử dụng trong HELLO1 cho DrawText biết phải hiển thị một dòng, chỉnh giữa cả hai chiều. Vì đối tượng rect lưu giá trị của cả vùng làm việc trên cửa sổ nên xâu "Hello, MFC" sẽ được hiển thị vào giữa cửa sổ.

Bảng 1.7 Các cờ khuôn dạng của DrawText

<i>Cờ</i>	<i>Mô tả</i>
DT_BOTTOM	Chỉnh xâu ở đáy khuôn.
DT_CALCRECT	Tính kích thước khuôn cần cho hiển thị xâu mà không cần vẽ xâu ngay. Toa độ giới hạn được lưu vào cấu trúc RECT hay đối tượng CRect.
DT_CENTER	Chỉnh giữa xâu trong khung.
DT_EXPANDTABS	Tăng thêm khoảng cách cho ký tự tab. Mặc định có giá trị 8, chúng có thể thay đổi nhờ chỉ ra thuộc tính DT_TABSTOP và gộp tab stop setting trong tham số chứa các cờ này (xem dưới đây).
DT_EXTERNALLEADING	Gộp external leading của co chữ khi tính khoảng cách dòng.
DT_LEFT	Chỉnh xâu thẳng lề trái của khung.
DT_NOCLIP	Xâu ký tự không bị clip khung (xâu có thể hiển thị ra ngoài khung).
DT_NOPREFIX	Ký tự & (ampersand) được coi là thông thường. Mặc định & báo hiệu gạch chân của ký tự sau.
DT_RIGHT	Chỉnh xâu ký tự thẳng lề phải của khung.
DT_SINGLELINE	Bỏ qua các ký tự CR, LF.
DT_TABSTOP	Byte cao của tham số lưu tổng số space giữa tab stops.
DT_TOP	Chỉnh xâu ký tự từ đỉnh của khung.
DT_VCENTER	Chỉnh giữa xâu theo chiều ngang của khung.
DT_WORDBREAK	Tự động bẻ các từ nếu xâu ký tự vượt ra ngoài khung.

DrawText sử dụng co chữ (*font*) đã gán cho dc. Vì HELLO1 không gán co chữ cho dc nên co chữ mặc định được sử dụng. Giả sử muốn hiển thị co chữ arial, 72-điểm thì trước hết phải tạo co chữ và gán nó cho dc nhờ hàm SelectObject của lớp CPaintDC.

1.17 ẢNH XẠ THÔNGIỆP

Khi Windows gửi thông điệp WM_PAINT cho HELLO1, thì bằng cách nào Windows có thể gọi CMainWindow::OnPaint? Câu trả lời là nhờ *ảnh xạ thông điệp*. Ảnh xạ thông điệp là bảng tương quan giữa thông điệp với các hàm thành phần quản lý các thông điệp đó. Bất kỳ lớp nào suy diễn từ CCmdTarget đều có thể chứa ảnh xạ thông điệp. Với mỗi thông điệp mà ta muốn xử lý, hãy viết hàm thành phần của lớp và thêm đầu vào tương ứng cho mỗi ảnh xạ thông điệp lớp. MFC sẽ làm các việc còn lại, chúng đảm bảo chắc chắn rằng hàm thành phần tương ứng được gọi mỗi khi có thông điệp tới.

Bước thứ nhất để thiết lập chương trình trả lời thông điệp là thêm các hàm thành phần quản lý chúng. HELLO1 chỉ quản lý một kiểu thông điệp - WM_PAINT - và trách nhiệm xử lý các thông điệp WM_PAINT là hàm OnPaint của lớp CMainWindow. OnPaint không cho lại giá trị và không nhận đối số, đó là một trong các tên hàm mà MFC nhận ra là chúng thuộc trình quản lý thông điệp. Cho trước tên thông điệp với tiền tố WM_XXX, thí dụ WM_PAINT, có thể dễ dàng suy ra hệ thống tên của MFC dùng cho hàm quản lý thông điệp: thay thế WM_XXX bằng On và chuyển toàn bộ phần còn lại sang chữ nhỏ trừ ký tự đầu. WM_PAINT trở thành OnPaint, và WM_LBUTTONDOWN trở thành OnLButtonDown. Sau đó tra tài liệu MFC để có mẫu hàm chính xác. (Tiến trình này được đơn giản hoá khi sử dụng công cụ như Visual C++ *ClassWizard*, vì chúng tự sinh mẫu hàm cho ta). OnPaint không nhận đối nào, nhưng hàm khác, như OnLButtonDown, lại có. Các đối được chuyển cho hàm quản lý thông điệp sử dụng cho cùng mục đích như wParam và lParam trong C. Bảng 1.8 là danh sách các thông điệp WM_XXX và mẫu hàm MFC tương ứng. Tiền tố `afx_msg` ở đầu mẫu hàm cho biết hàm là trình quản lý thông điệp.

Bảng 1.8 Các thông điệp WM_XXX và mẫu hàm MFC tương ứng

Thông điệp	Nguyên hàm
WM_CHAR	<code>afx_msg void OnChar (UINT, UINT, UINT)</code>
WM_CREATE	<code>afx_msg int OnCreate (LPCREATESTRUCT)</code>
WM_DESTROY	<code>afx_msg void OnDestroy ()</code>
WM_LBUTTONDOWN	<code>afx_msg void OnLButtonDown (UINT, CPoint)</code>
WM_LBUTTONUP	<code>afx_msg void OnLButtonUp (UINT, CPoint)</code>
WM_MOUSEMOVE	<code>afx_msg void OnMouseMove (UINT, Cpoint)</code>
WM_PAINT	<code>afx_msg void OnPaint ()</code>
WM_SIZE	<code>afx_msg void OnSize (UINT, int, int)</code>

Bước thứ hai chuẩn bị cho ứng dụng nhận và xử lý các thông điệp là liên kết các thông điệp và các trình quản lý của chúng bằng ánh xạ thông điệp. Mỗi lớp có chứa trình quản lý thông điệp thì phải có ánh xạ thông điệp. Ánh xạ thông điệp được khai báo như các thành phần lớp khác trong khai báo lớp, nhưng đầu vào của chúng được định nghĩa ở ngoài khai báo lớp, thông thường các tệp CPP chứa mã của các hàm thành phần lớp.

Lệnh sau đây trong HELLO1.H khai báo ánh xạ thông điệp của CMainWindow:

```
DECLARE_MESSAGE_MAP()
```

DECLARE_MESSAGE_MAP là macro định nghĩa trong AFXWIN.H. Macro này sẽ thêm 3 thành phần sau vào lớp:

- Mảng riêng (*private array*) của cấu trúc kiểu AFX_MSGMAP_ENTRY với tên `_MessageEntries`, làm nhiệm vụ dẫn thông điệp vào trình quản lý của chúng.

- Cấu trúc bảo vệ (*protected*) kiểu `AFX_MSGMAP` với tên `messageMap` để trỏ đến gốc của mảng `_MessageEntries` và cho biết tổng số đầu vào nó có.
- Hàm bảo vệ ảo (*protected virtual*) với tên `GetMessageMap` cho lại địa chỉ của mảng `messageMap`.

Khi chương trình chạy, các thành phần trên cho phép MFC xác định xem lớp cho trước có thể quản lý kiểu thông điệp đó hay không, nếu có, thì địa chỉ của trình quản lý thông điệp là gì. Nếu thông điệp đến và lớp thiếu đầu vào ánh xạ thông điệp cho chúng, MFC kiểm soát ánh xạ thông điệp của lớp cơ sở, với cách làm như vậy cho đến khi thấy được lớp có hàm thành phần tương ứng hay khẳng định rằng thông điệp đó phải được chuyển đến hàm `DefWindowProc`. Thông thường lệnh `DECLARE_MESSAGE_MAP` là lệnh cuối trong khai báo lớp, nhưng không bắt buộc như vậy. Cũng có thể khai báo các thành phần sau `DECLARE_MESSAGE_MAP` nhưng phải có từ khóa tương ứng để đảm bảo tính đúng đắn của miền tác động: *public*, *protected*, hay *private*.

Các ánh xạ phải khai báo giữa `BEGIN_MESSAGE_MAP` và `END_MESSAGE_MAP`, thí dụ:

```
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_PAINT ()
END_MESSAGE_MAP ()
```

Tham số của `BEGIN_MESSAGE_MAP` cho biết lớp, nơi mà ánh xạ thông điệp khai báo và lớp cơ sở của chúng. MFC cần biết lớp cơ sở để có thể chuyển các thông điệp không quản lý được lên trên theo thang. Các macro đầu vào thông điệp sẽ xây dựng ánh xạ thông điệp, mỗi *macro* thêm một đầu vào trong mảng khai báo trong `DECLARE_MESSAGE_MAP`. Macro `END_MESSAGE_MAP` đánh dấu kết thúc mảng.

Ánh xạ thông điệp của `HELLO1` chỉ chứa một đầu vào. `ON_WM_PAINT` là macro đầu vào thông điệp tương ứng với thông điệp `WM_PAINT`. Các macro `ON_WM_XXX` không bao giờ có đối và luôn nhận ra các thông điệp có cùng tên `WM_XXX`. Tuy nhiên, không phải tất cả các macro đầu vào đều bắt đầu bằng `ON_WM_`. Đầu vào của thông điệp `WM_COMMAND` sinh ra khi ta nhấn vào mục của thực đơn được chỉ ra như sau:

```
ON_COMMAND (IDM_EXIT, OnExit)
```

Macro `ON_COMMAND` giúp MFC chia nhỏ thông điệp `WM_COMMAND`, nó được thực đơn và các cửa sổ điều khiển sử dụng để gửi thông báo ngược đến cha nó và chuyển dẫn đến chương trình quản lý chúng. Macro `ON_COMMAND` trên đây nối hàm `OnExit` với các thông điệp `WM_COMMAND` được phát sinh khi mục thực đơn có ID là `IDM_EXIT` được chọn. Trong chương trình C thường sử dụng kiến trúc switch-case bên trong trình quản lý `WM_COMMAND`.

1.18 HÀM `AfxRegisterWndClass`

Khi ứng dụng MFC bắt đầu chạy, MFC đăng ký các lớp cửa sổ mặc định với các kiểu khác nhau. Cấu trúc `WNDCLASS` được đăng ký cho cửa sổ khung SDI để xác định các tính chất có thể thay đổi, bao gồm:

- Kiểu lớp (*class style*)
- Con chạy của lớp (*class cursor*)
- Tô màu nền của lớp (*class background brush*)
- Biểu tượng của lớp (*class icon*)

Kiểu của lớp xác định các đặc trưng ứng xử của cửa sổ như có nhận thông điệp thông báo lại khi chuột được nhấn đúp trên khung hay không. Con chạy của lớp là con trỏ hiển thị khi con trỏ chuột đặt trên cửa sổ, chổi tô nền của lớp định nghĩa màu nền mặc định của cửa sổ. (Đặc biệt, nó định nghĩa màu được sử dụng để xóa bên trong cửa sổ khi có thông điệp `WM_ERASEBKGD`). Cuối cùng, biểu tượng của lớp xác định biểu tượng ứng dụng. Lớp mặc định của MFC sử dụng kiểu `CS_DBLCLKS | CS_HREDRAW | CS_VREDRAW`, có nghĩa cửa sổ nhận thông báo và sẽ vẽ lại khi thay đổi kích cỡ. Con chạy của lớp là mũi tên, màu nền của lớp là màu hệ thống `COLOR_WINDOW`, còn biểu tượng của lớp là biểu tượng mặc định của Windows hay chúng được nhận ra bởi ID `AFX_IDI_STD_ICON`. Một cách cho phép thay đổi các thuộc tính này là hàm `AfxRegisterWndClass`, chúng sẽ đăng ký lớp riêng và truyền tên lớp cho hàm `Create`. Cửa sổ mà ta lập sẽ sử dụng các đặc tính này.

`AfxRegisterWndClass` chấp nhận bốn tham số theo trình tự sau: kiểu lớp (xem bảng 1.9), chỉ danh con chạy của lớp, chỉ danh màu nền của lớp, chỉ danh biểu tượng của lớp. Nó cho lại tên lớp để truyền cho `Create`. Các thông tin khác có thể tìm xem trong tài liệu về công cụ phát triển chương trình Windows (*Windows Software Development Kit - SDK*). Chỉ danh của con chạy và biểu tượng có thể lấy từ hàm `API LoadStandardCursor` và `LoadIcon` của `CWinApp`. Còn chỉ danh chổi có thể lấy từ nhiều nguồn, kể cả hàm `GetStockObject` của Windows, và chúng cho lại chỉ danh của chổi có sẵn ("*stock*" brushes) do Windows định nghĩa. Ta phải chỉ ra kiểu lớp, còn các thông số khác là tùy chọn. Khi cho bất kỳ tham số nào một giá trị 0 thì có nghĩa là yêu cầu MFC sử dụng giá trị mặc định. Các lệnh:

```
Cstring className = AfxRegisterWndClass (CS_HREDRAW | CS_VREDRAW,
    myApp.LoadStandardCursor (IDC_CROSS),
    (HBRUSH) ::GetStockObject (LTGRAY_BRUSH),
    myApp.LoadIcon ("MyIcon"));
Create (className, "Hello1");
```

trước hết đăng ký lớp cửa sổ có kiểu `CS_HREDRAW` và `CS_VREDRAW`, con chạy hình chữ thập, màu nền, ghi sáng, và biểu tượng có tên *MyIcon* mà nó được nạp từ tệp ứng dụng (.EXE); sau đó

chúng tạo cửa sổ trên cơ sở lớp này.

Bảng 1.9 Các kiểu lớp cửa sổ

<i>Kiểu lớp</i>	<i>Mô tả</i>
CS_BYTEALIGNCLIENT	Đảm bảo rằng vùng làm việc của cửa sổ luôn gán bắt đầu từ byte (byte boundary) trong đệm video để tăng tốc độ vẽ.
CS_BYTEALIGNWINDOW	Đảm bảo rằng chính cửa sổ luôn gán bắt đầu từ byte (byte boundary) trong đệm video để tăng tốc chuyển và co dãn.
CS_CLASSDC	Chỉ rằng toàn bộ cửa sổ của lớp này có thể chia sẻ DC.
CS_DBLCLKS	Chỉ rằng cửa sổ được báo lại bằng thông điệp WM_LBUTTONDOWNCLK, WM_MBUTTONDOWNCLK, và WM_RBUTTONDOWNCLK khi nhấn đúp trên khung.
CS_GLOBALCLASS	Chỉ rằng lớp có thể được đăng ký tổng thể để toàn bộ ứng dụng sử dụng. Thường làm các điều khiển như phím và hộp danh sách cho toàn bộ hệ thống.
CS_HREDRAW	Cửa sổ có thể nhận thông điệp WM_PAINT khi thay đổi kích thước ngang.
CS_NOCLOSE	Vô hiệu lực lệnh Close trên thực đơn hệ thống và phím close trên thanh tiêu đề.
CS_OWNDC	Mỗi cửa sổ được lập từ lớp này có thể có DC riêng. Hiệu quả khi tối ưu việc vẽ lại vì ứng dụng không cần xác định lại DC riêng mỗi khi cần đến DC.
CS_PARENTDC	Cửa sổ con được lập từ lớp này thừa hưởng DC của cha nó.
CS_SAVEBITS	Vùng trên màn hình bị chiếm bởi cửa sổ được lập từ lớp này có thể được lưu trong dạng bitmap để vẽ lại cho nhanh. Thường được sử dụng cho thực đơn và các cửa sổ có thời gian sử dụng ngắn.
CS_VREDRAW	Cửa sổ có thể nhận thông điệp WM_PAINT khi thay đổi kích thước dọc.

Chương trình 1.2 là bản nâng cấp của HELLO1 với tên HELLO2, chúng sử dụng các lệnh mô tả trên để tạo lớp cửa sổ. Hình 1.4 cho cửa sổ kết quả sau khi dịch, liên kết và chạy chương trình.

Chương trình 1.2 HELLO2

Tệp HELLO2.H

```
class CMyApp : public CWinApp
{
public:
    BOOL InitInstance ();
};
//-----
class CMainWindow : public CFrameWnd
{
public:
    CMainWindow ();
protected:
    afx_msg void OnPaint ();
```

```

DECLARE_MESSAGE_MAP ()
{
//-----
Tệp HELLO2.RC

Mylcon ICON hello2.ico
//-----

Tệp HELLO2.CPP

#include <afxwin.h>
#include "hello2.h"
CMyApp myApp;
// CMyApp member functions
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}

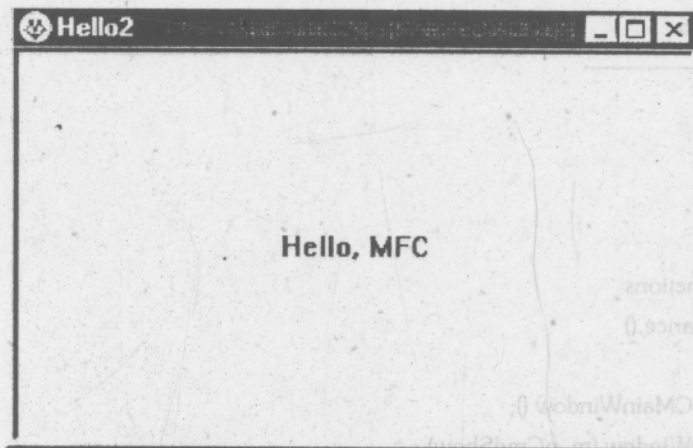
// Các hàm thành phần và ánh xạ thông điệp của CMainWindow
BEGIN_MESSAGE_MAP (CMainWindow, CWnd)
    ON_WM_PAINT ()
END_MESSAGE_MAP ()
//-----
CMainWindow::CMainWindow ()
{
    CString wndClass = AfxRegisterWndClass (CS_HREDRAW | CS_VREDRAW,
        myApp.LoadCursor (IDC_CROSS),
        (HBRUSH)::GetStockObject (LTGRAY_BRUSH),
        myApp.LoadIcon ("Mylcon"));
    Create (wndClass, "Hello2");
}
//-----
void CMainWindow::OnPaint ()
{
    CRect rect;
    CPaintDC dc (this);
    GetClientRect (rect);
    dc.SetBkMode (TRANSPARENT);

```

```
dc.DrawText ("Hello, MFC", -1, rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
```

```
}
```

```
//-----
```



Hình 1.4 Kết quả chương trình 1.2

Hãy chú ý đến mẫu nền của cửa sổ và lệnh:

```
dc.SetBkMode (TRANSPARENT);
```

trong `OnPaint`. Hàm `SetBkMode` lập chế độ màu nền của xâu ký tự thành trong suốt để hàm `DrawText` không đặt nền trắng sau dòng chữ "Hello, MFC". Chế độ mặc định của chữ là `OPAQUE`, chúng làm `DrawText` đặt nền sau ký tự bằng màu nền của ký tự đã chọn cho DC. Ngoài ra `HELLO2` còn có thêm tệp mã nguồn `HELLO2.RC` với một dòng lệnh:

```
Mylcon ICON hello2.ico
```

Biểu tượng được tạo ra nhờ "Visual C++ resource editor", và cũng có thể bằng phần mềm khác. Trong tệp `RC`, lệnh `ICON` cho biết `HELLO2.ICO` là tệp biểu tượng, và `Mylcon` là tên tài nguyên gán cho biểu tượng. Nếu sử dụng môi trường lập trình Visual C++, thì chỉ đơn giản thêm tệp `RC` vào tệp project, chương trình dịch sẽ tự động dịch cho ta.

Nếu muốn tự tạo lập biểu tượng cho ứng dụng thì không cần vất vả với việc gọi `AfxRegisterWndClass`. Với lớp cửa sổ suy diễn từ `CFrameWnd`, đơn giản thay ID của biểu tượng trong tệp `RC` thành `AFX_IDI_STD_FRAME`.

```
AFX_IDI_STD_FRAME ICON hello2.ico
```

MFC nhận ID đặc biệt `AFX_IDI_STD_FRAME` như ID của biểu tượng chuẩn cho cửa sổ SDI.

1.19 TÓM TẮT

Bây giờ nhắc lại vài tư tưởng chính đã trình bày. Đầu tiên, trong chương trình MFC có khai báo tổng thể một đối tượng ứng dụng của lớp suy diễn từ `CWinApp`. Hàm ẩn `WinMain` (do MFC cung cấp) sẽ khởi động chương trình. `WinMain` gọi hàm `InitInstance` của đối tượng ứng dụng, và sau đó gọi hàm `Run` của đối tượng đó để khởi động vòng lặp thông điệp. Đối tượng ứng dụng nạp chồng `InitInstance` và sử dụng chúng để tạo cửa sổ ứng dụng. Quá trình tạo cửa sổ là tiến trình hai bước. Thứ nhất, đối tượng cửa sổ được khai báo từ lớp suy diễn `CFrameWnd`. Tiếp theo, cấu tử của đối tượng sẽ gọi hàm `Create` mà nó thừa hưởng từ `CFrameWnd` để tạo cửa sổ thực. Một khi cửa sổ được lập, `InitInstance` gọi hàm `ShowWindow` và `UpdateWindow` của đối tượng cửa sổ để có thể nhìn thấy chúng trên màn hình và gửi cho chúng thông điệp `WM_PAINT` đầu tiên. Các thông điệp `WM_PAINT` được cơ chế ánh xạ thông điệp của MFC chuyển đổi thành lời gọi đến `CMainWnd::OnPaint`, và `OnPaint` hiển thị xâu ký tự "Hello, MFC" trên vùng làm việc của ứng dụng nhờ lập đối tượng `CPaintDC` và gọi `CPaintDC::DrawText`.

Điều trên đây là hoàn toàn khác với cách mà Windows đã từng được sử dụng để lập trình. Hai bước xây dựng cửa sổ? Các đối tượng ứng dụng? Không còn `WinMain`? MFC đã làm đơn giản vấn đề. Tuy MFC không làm dễ hiểu mã nguồn, nhưng sử dụng thư viện lớp sẽ làm giảm đáng kể mã trình phải viết. Có thể sửa đổi cách ứng xử của bất kỳ lớp MFC nào bằng cách suy diễn chúng (hay lớp cơ sở của chúng) thành lớp của lập trình viên, làm MFC thành công cụ rất hữu hiệu. Lợi thế gia tăng khi sử dụng các đặc tính rất mạnh và phức tạp của Windows như OLE. Với MFC có thể viết ứng dụng OLE và chạy nó không mấy khó khăn.

`HELLO1` và `HELLO2` còn thiếu nhiều thành phần để hình thành chương trình Windows đầy đủ, nhưng nó là bước đầu tốt đẹp trên đường đi. Các chương sau sẽ giới thiệu thêm các thành phần giao diện như thực đơn, hộp thoại, điều khiển, trình diễn cách nhận đầu vào từ chuột, bàn phím, và khai thác ghép nối của MFC với GDI của Windows.

CHƯƠNG 2

SỬ DỤNG BÚT VẼ VÀ MÀU TÔ

Chương này giới thiệu một vài lớp của MFC và các thành phần của chúng giúp lập trình viên vẽ được trong vùng làm việc của cửa sổ.

Phần của Windows thực hiện vẽ đồ họa là modul giao diện thiết bị đồ họa (*Graphics Device Interface - GDI*). Các tệp GDI.EXE và GDI32.DLL của Windows cung cấp các dịch vụ GDI, chúng lưu giữ tương ứng phần 16 hay 32 bit của GDI. Tương tự như các modul DLL khác của Windows, GDI có hai nửa 16 và 32 bit để trợ giúp ứng dụng 16 và cả 32 bit để tương thích với các bản cũ của Windows. Nhưng trong tài liệu này, không phải quan tâm đến việc rẽ đôi đó. Ta chỉ đơn giản gọi dịch vụ GDI hay MFC tương ứng, Windows sẽ hướng lời gọi của lập trình viên vào vị trí thích hợp.

Chương trình mẫu trong chương 1 sử dụng hàm `CDC::DrawText` để đưa xâu ký tự ra cửa sổ. `DrawText` là một trong các hàm thành phần của lớp `CDC` thực hiện đưa ra xâu ký tự và đồ họa. Trước hết chúng ta xem xét chi tiết hơn lớp `CDC` và các lớp suy diễn của chúng, cách sử dụng bút và các chổi tô. Sau đó ta xây dựng chương trình mẫu chỉ ra cách làm việc của các đối tượng này rồi thêm thanh trượt và logic trượt vào chương trình.

2.1 VẼ BẰNG GDI

Khi chương trình Windows vẽ trên màn hình, máy in hay các thiết bị ra khác, chúng không liên lạc trực tiếp với thiết bị; thay vì đó chúng sẽ vẽ trên ngữ cảnh thiết bị `DC` (*Device Context*). Nhưng trước hết phải có chỉ danh của `DC` để Windows sử dụng chúng làm tham số của các hàm GDI.

Khi lập trình Windows bằng MFC, `DC` còn có ý nghĩa lớn hơn nhiều. Đối tượng `DC` gói các hàm GDI mà chương trình gọi để phát sinh dữ liệu cho thiết bị ra. Trong MFC, ta không phải lấy chỉ danh của `DC` và không gọi các hàm ra GDI, hay ít ra là không trực tiếp; thay vì đó ta lấy con trỏ đến đối tượng `DC` (hay lập đối tượng `DC`) và sử dụng con trỏ này để gọi các hàm thành phần của đối tượng. Trình MFC có thể vẽ đường thẳng từ điểm (0,0) đến (100,100) như sau:

```
CDC* pDC = GetDC ();  
pDC->MoveTo (0, 0);  
pDC->LineTo (100, 100);
```

`CWnd::GetDC` cho lại con trỏ đến đối tượng của lớp `CDC`, và các hàm thành phần của đối tượng sẽ được gọi để vẽ. `CDC` chứa chỉ danh của `DC` và các hàm thành phần của chúng gọi qua các hàm GDI trong API của Windows. Trong trường hợp này, `CDC::MoveTo` và `CDC::LineTo` ánh xạ

trực tiếp đến các hàm của API bao gồm `::MoveToEx` (hàm của bản Win32), hàm `::MoveTo` (của bản Win 3.1) và hàm `::LineTo`.

Bước thứ nhất để thực hiện vẽ đồ họa từ chương trình Windows là lấy quyền tham nhập DC. Trong MFC, có nhiều cách để làm việc này; `GetDC` là một trong đó. Khi xong công việc với DC, chúng phải được giải phóng vì Windows chỉ có sẵn năm DC cho toàn bộ hệ thống, và cả năm phải được chia sẻ cho tất cả các trình đang chạy. (Có khả năng trình yêu cầu DC riêng, nhưng đó là trường hợp đặc biệt, chưa bàn đến ở đây). Nếu lấy con trỏ DC bằng hàm `GetDC`, thì phải giải phóng chúng bằng `CWnd::ReleaseDC` như sau:

```
CDC* pDC = GetDC ();
// Các hàm thực hiện vẽ
ReleaseDC (pDC);
```

DC phải được giải phóng càng sớm càng tốt, và `ReleaseDC` phải được gọi tại vị trí cùng thông điệp nơi đã gọi `GetDC`. Nếu lấy DC bằng cách khác - thí dụ, khai báo đối tượng của lớp `CPaintDC` hay `CClientDC` - MFC sẽ tự giải phóng DC cho ta khi đối tượng bị phá hủy. Cách lấy và giải phóng DC phụ thuộc vào hoàn cảnh khi gọi.

2.2 DC TRONG OnPaint

Như đã trình bày trong chương trước, hàm `OnPaint` trong trình MFC sẽ xử lý các thông điệp `WM_PAINT`, hàm này được nối tới hệ thống thông điệp qua ánh xạ thông điệp. `OnPaint` thường là hàm thành phần của lớp cửa sổ khung, mặc dù về kỹ thuật thì nó có thể thuộc bất kỳ lớp nào được suy diễn từ `CCmdTarget`. Thông điệp `WM_PAINT` khác với các thông điệp khác ở một khía cạnh rất quan trọng: nếu hàm xử lý bị sai khi gọi API của Windows với tên là `::BeginPaint` và `::EndPaint`, thì thông điệp `WM_PAINT` sẽ không bị mất đi khỏi hàng đợi thông điệp, chương trình sẽ vẽ lại liên tục trên vùng làm việc. Kết quả là trình sẽ bị tắc trong vòng xử lý thông điệp `WM_PAINT`.

Để lấy DC trong hàm `OnPaint`, hãy sử dụng lớp `CPaintDC` của MFC. Khi đối tượng của lớp `CPaintDC` được lập, chúng tự động gọi `::BeginPaint` để bắt đầu tiến trình vẽ. Khi bị phá hủy, `::EndPaint` được gọi đồng thời. Lệnh:

```
CPaintDC dc (this);
```

lập đối tượng DC với tên `dc` trong ngăn xếp. DC được lập trên cơ sở lớp `CPaintDC` được gọi là DC trong thông điệp vẽ. Hàm `OnPaint` sử dụng DC trong thông điệp vẽ để vẽ đường từ (0,0) đến (100,100) có thể như sau:

```
void CMainWindow::OnPaint ()
{
    CPaintDC dc (this);
```

```
dc.MoveTo (0, 0);
dc.LineTo (100, 100);
}
```

Chú ý rằng không cần gọi `ReleaseDC` ở đây. Khi hàm `OnPaint` kết thúc và đi ra khỏi miền tác động, hủy tử của `CPaintDC` tự gọi `::EndPaint` để bãi bỏ DC. Nếu xây dựng DC trong thông điệp vẽ bằng toán tử `new` như sau:

```
CPaintDC* pDC = new CPaintDC (this);
```

thì phải sử dụng hàm `delete` `pDC` trước khi kết thúc hàm `OnPaint`, nếu không hàm `::EndPaint` sẽ không được gọi.

Khi nhận thông điệp `WM_PAINT` không có nghĩa là phải vẽ lại toàn bộ cửa sổ. Nếu thông điệp `WM_PAINT` được sinh ra vì thay đổi tọa độ góc của cửa sổ, thì chỉ góc ấy cần cập nhật. Thông điệp `WM_PAINT` có kèm theo thông tin mô tả hình chữ nhật cần phải vẽ lại. Nếu muốn, chương trình có thể tối ưu tiến trình vẽ lại nhờ lấy vùng làm việc không còn hợp lệ từ DC và chỉ cập nhật vùng đó mà thôi. Sau khi đối tượng `CPaintDC` được lập, tọa độ của vùng hợp lệ có thể lấy từ trường `rcPaint` của cấu trúc `PAINTSTRUCT` với tên là `m_ps`, nó là thành phần của lớp `CPaintDC`. Trường `rcPaint` có cấu trúc kiểu `RECT`, và được định nghĩa như sau:

```
struct tagRECT {
    int left;
    int top;
    int right;
    int bottom;
} RECT;
```

`rcPaint.left` và `rcPaint.right` lưu tọa độ x của biên trái và biên phải, `rcPaint.top` và `rcPaint.bottom` lưu tọa độ y của biên trên và biên dưới. Mặc định thì các vị trí của cửa sổ được đo bằng *pixel*. Góc trên, trái của vùng làm việc có giá trị (0,0); giá trị của x, y tăng theo chiều phải và chiều xuống dưới. Nếu vùng làm việc của cửa sổ là 200x200 *pixels* và vùng bất hợp lệ là 1/4 phía dưới, trái của cửa sổ thì `rcPaint` sẽ mô tả vùng mà góc trên, trái là (100,100) và góc dưới, phải là (199,199). Không lo ngại việc vẽ lại trên vùng lớn hơn vì GDI tự động làm *clipping* đầu ra, tuy nhiên tốc độ vẽ sẽ giảm.

2.3 DC BÊN NGOÀI OnPaint

Không phải toàn bộ trình đều thực hiện vẽ từ trong `OnPaint`. Giả sử ta viết chương trình `Paintbrush` để vẽ các đối tượng trên màn hình khi người sử dụng nhấn phím chuột. Một cách làm tươi màn hình khi trả lời phím chuột là gọi hàm `CWnd::InvalidateRect` để làm bất hợp lệ toàn bộ hay một phần vùng làm việc của cửa sổ và dẫn tới `OnPaint` thực hiện vẽ. Để làm tăng tốc độ, ta có thể làm tươi toàn bộ màn hình tức thì bằng cách lấy DC ngoài thông điệp vẽ để vẽ.

Nếu đã có con trỏ đến đối tượng cửa sổ, ta có thể lấy DC bằng hàm `GetDC`. Nhưng MFC cung cấp lớp suy diễn từ CDC với tên `CClientDC` để dễ dàng làm việc này hơn. Ta có thể thay:

```
CDC* pDC = GetDC ();
// Các hàm thực hiện vẽ
ReleaseDC (pDC);
```

bằng cách sau:

```
CClientDC dc (this);
// Các hàm thực hiện vẽ
```

Bây giờ dc biểu diễn DC và không cần gọi hàm `ReleaseDC`. Tại sao? Đó là vì cấu trúc của lớp `CClientDC` gọi `::GetDC` và hủy từ của nó gọi `::ReleaseDC`. Vì với DC trong thông điệp vẽ, ta chỉ phải quan tâm đến hủy bỏ DC nếu vì lý do nào đó bắt ta quyết định lập đối tượng của `CClientDC` bằng hàm `new`. Trong trường hợp đó, ta phải gọi `delete` trước khi hàm điều khiển thông điệp kết thúc để đảm bảo chắc chắn là hủy từ được gọi.

Trong vài trường hợp, khi ta muốn vẽ ngoài vùng làm việc của cửa sổ (*nonclient area*), MFC có cung cấp lớp `CWindowDC`. `CWindowDC` làm việc như `CClientDC`, nhưng DC mà nó đại diện lại bao gồm mọi thứ bên trong các biên của cửa sổ. Đôi khi lập trình viên sử dụng `CWindowDC` cho các công việc không bình thường như làm lấy thanh tiêu đề hay làm cửa sổ với góc tròn. `CWindowDC` không phải là cái mà ta cần thường xuyên. Nếu ta muốn vẽ ngoài vùng làm việc của cửa sổ thì hãy chặn lấy thông điệp `WM_NCPAINT` cùng hàm `OnNcPaint` để xác định khi nào ngoài vùng làm việc cần phải vẽ lại. Không như `OnPaint`, hàm `OnNcPaint` không cần gọi `::BeginPaint` và `::EndPaint`.

Với trường hợp là trình cần thâm nhập toàn bộ màn hình, ta phải tạo đối tượng DC của lớp `CClientDC` hoặc `CWindowDC` và truyền con trỏ `NULL` cho cấu trúc thay vì con trỏ đến đối tượng cửa sổ như sau:

```
CClientDC dc(NULL);
```

Các lệnh sau đây vẽ đường thẳng từ góc trên, trái của màn hình đến điểm (100,100).

```
dc.MoveTo (0, 0);
dc.LineTo (100, 100);
```

Các trình chụp ảnh (*capture*) thường sử dụng kỹ thuật này để thâm nhập toàn bộ màn hình.

2.4 VẼ BẰNG DC

Một khi đã có DC để làm việc, ta có thể vẽ. Hai trong nhiều hàm của lớp CDC mà ứng dụng MFC dùng để vẽ màn hình là `MoveTo` và `LineTo`. `MoveTo` lập vị trí hiện hành, đó là con trỏ chỉ ra

một điểm trên màn hình. LineTo vẽ đường thẳng từ điểm hiện hành đến điểm chỉ ra trong hàm, rồi tự động cập nhật vị trí hiện hành. Các lệnh

```
dc.MoveTo (100, 100);
dc.LineTo (200, 100);
dc.LineTo (200, 200);
dc.LineTo (100, 200);
dc.LineTo (100, 100);
```

vẽ hình vuông có mỗi chiều 100 đơn vị. Thí dụ cũng cho thấy không cần gọi MoveTo trước mỗi lần gọi LineTo.

MoveTo và LineTo chỉ là hai trong số rất nhiều hàm của lớp CDC để vẽ trên màn hình. Bảng 2.1 liệt kê một số hàm vẽ thường dùng của lớp CDC. Toàn bộ, CDC gồm hơn 150 hàm thành phần vẽ, lấy thông tin GDI,

Bảng 2.1. Một số hàm vẽ của CDC

Hàm	Mô tả
ArcTo	Vẽ cung elip với bút hiện hành, làm mới vị trí hiện hành đến cuối cung.
Ellipse	Vẽ elip trong hình chữ nhật. Đường biên được vẽ bằng bút hiện hành và tô bằng chổi hiện hành.
FillRect	Tô hình chữ nhật bằng chổi hiện hành.
FloodFill	Tô vùng có hình bất kỳ, có bất kỳ bằng màu hiện hành.
LineTo	Vẽ đường với bút hiện hành và làm tươi vị trí hiện hành tới cuối đường.
MoveTo	Lập vị trí hiện hành để chuẩn bị vẽ.
Pie	Vẽ Pie trong hình chữ nhật. Đường biên được vẽ bằng bút hiện hành và tô bằng chổi hiện hành.
PolyBezier	Vẽ một hay nhiều Bezier splines bằng bút hiện hành.
Polygon	Vẽ đa giác bằng cách nối các trường điểm. Đường biên được vẽ bằng bút hiện hành và tô bằng chổi hiện hành.
Rectangle	Vẽ chữ nhật trong hình chữ nhật. Đường biên được vẽ bằng bút hiện hành và tô bằng chổi hiện hành.
RoundRect	Vẽ Pie trong hình chữ nhật có góc tròn. Đường biên được vẽ bằng bút hiện hành và tô bằng chổi hiện hành.
SetTextColor	Lập màu văn bản.
TextOut	Ra văn bản với màu hiện hành.

Bảng 2.1 giới thiệu các hàm sử dụng bút và màu tô hiện hành. Các bút và màu được biểu diễn trong các lớp **CPen** và **CBrush** của MFC. Hàm **CDC::SelectObject** gán bút và màu cho DC liên quan. Nếu ta không gọi hàm **SelectObject** để thay đổi bút và màu tô hiện hành thì GDI sẽ sử dụng bút và màu tô mặc định của DC. Bút mặc định vẽ đường màu đen, liên tục rộng một *pixel*. Màu tô mặc định là màu trắng và đồng nhất (*solid white*).

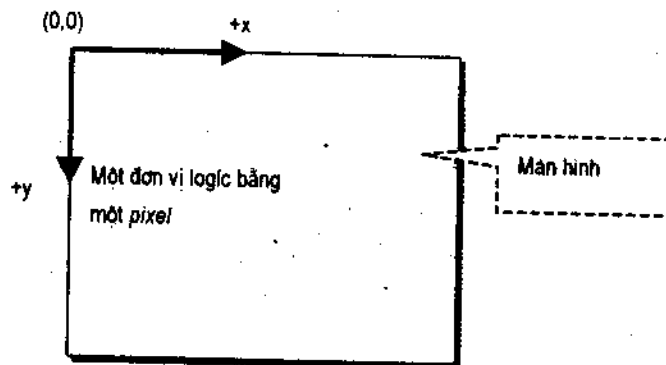
Cũng có thể tạo bút và màu riêng của ta và chọn nó cho DC để thay đổi các thuộc tính của đầu ra. Nếu muốn vẽ vòng tròn đồ đồng nhất với viền đen rộng mười *pixel*, ta phải tạo bút đen đồng

nhất mười *pixel* và chổi màu đỏ đồng nhất rồi gán chúng cho DC nhờ `SelectObject` trước khi gọi hàm `Ellipse`. Giả sử `pPen` là con trỏ đến đối tượng bút, `pBrush` là con trỏ đến đối tượng chổi, và đối tượng dc biểu diễn ngữ cảnh thiết bị DC thì đoạn mã trình có thể như sau:

```
dc.SelectObject (pPen);
dc.SelectObject (pBrush);
Ellipse (0, 0, 100, 100); //tham số của hàm là tọa độ chữ nhật bao elip
```

`SelectObject` được nạp chồng để chấp nhận con trỏ đến các đối tượng của các kiểu khác nhau. Giá trị cho lại của nó là con trỏ đến đối tượng của loại đã được chọn trước cho DC.

Khi sử dụng DC để vẽ, toàn bộ tọa độ là tọa độ logic. Thuộc tính khác của DC là chế độ ánh xạ, chúng xác định cách chuyển đổi tọa độ logic sang tọa độ thiết bị (*pixel trên màn hình*). GDI trợ giúp tám chế độ ánh xạ, chúng được chọn nhờ hàm `CDC::SetMapMode` và hỏi chế độ hiện hành nhờ hàm `CDC::GetMapMode`. Chế độ ánh xạ mặc định là `MM_TEXT`, chúng có đơn vị logic là *pixel* (một đơn vị logic theo trục x, y là một *pixel*) và tọa độ gốc là $(0,0)$ ở góc trên, trái của màn hình. Với DC của vùng làm việc, thì *pixel* góc trên trái là $(0,0)$. Giá trị x, y tăng khi chuyển sang phải và xuống dưới như hình 2.1.



Hình 2.1 Chế độ ánh xạ mặc định

Các chế độ ánh xạ khác sử dụng các phương pháp khác để chuyển đổi các tọa độ logic thành tọa độ thiết bị, hướng của hệ trục tọa độ là khác nhau. Thí dụ, trong chế độ ánh xạ `MM_LOMETRIC`, một đơn vị logic tương đương với 0.1 mm trên màn hình, gốc tọa độ mặc định là góc dưới-trái, giá trị x, y tăng theo chiều phải, lên.

Vì DC là tài nguyên chia sẻ, nó liên tục được sử dụng và giải phóng, không thể chọn bút và chổi cho DC rồi hy vọng nó luôn được giữ lại mỗi khi ta cần đến DC sau đó trong chương trình. Một khi rời bỏ DC, các thuộc tính mặc định của nó được khôi phục. Vì vậy, ta phải khởi động lại DC mỗi khi tạo đối tượng DC. Có nhiều cách giữ các thuộc tính của DC để chúng không cần phải nạp lại nhiều lần, vấn đề này sẽ được xem xét chi tiết sau. Một cách làm có thể tránh việc nạp đi nạp lại

DC là lấy DC trong hàm OnCreate và không rời bỏ chúng cho đến khi thông điệp WM_DESTROY đến, nhưng giải pháp này không bình thường vì số lượng DC là giới hạn.

2.5 BÚT VẼ

GDI cung cấp nhiều hàm API để lập bút và chổi tô cho DC, bao gồm `::CreatePen`, `::CreateSolidBrush`, `::CreateHatchBrush`, và `::CreatePatternBrush`. MFC gói các hàm này trong các lớp `CPen` và `CBrush` cho đơn giản, sao cho bút và chổi tô được đề cập đến như là các đối tượng thay vì các chỉ danh do GDI cung cấp.

MFC biểu diễn bút bằng đối tượng của lớp `CPen`. Cách đơn giản nhất để lập bút GDI là xây dựng các đối tượng `CPen` và chuyển cho chúng các tham số định nghĩa bút:

```
CPen pen (PS_SOLID, 1, RGB (255, 0, 0));
```

Phương pháp thứ hai là xây dựng đối tượng bút chưa khởi động rồi gọi hàm thành phần `CreatePen` của chúng:

```
CPen pen ();  
pen.CreatePen (PS_SOLID, 1, RGB (255, 0, 0));
```

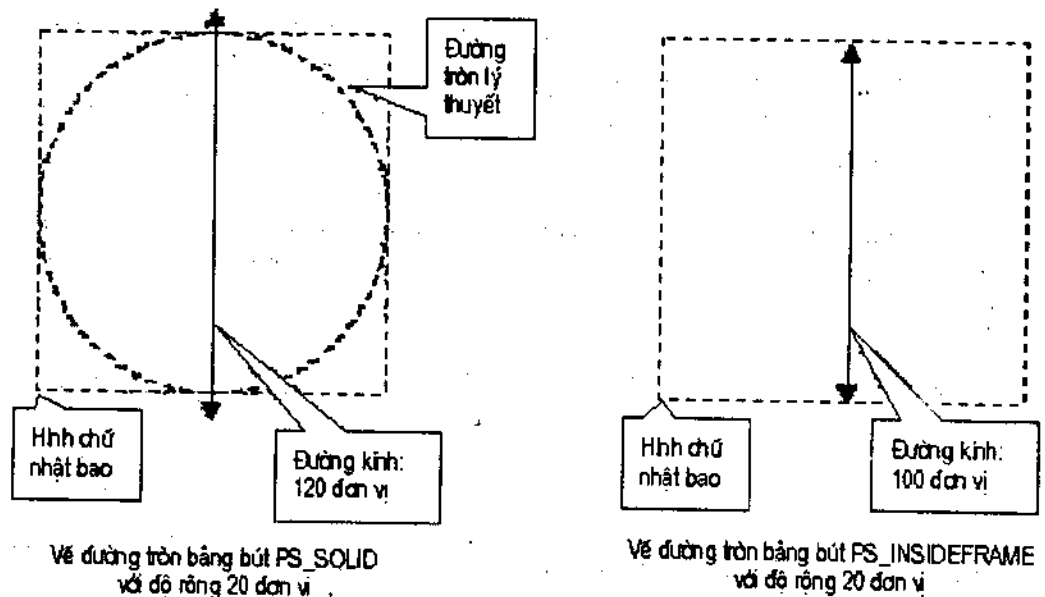
Phương pháp thứ ba là xây dựng đối tượng `CPen` chưa khởi động, làm đầy cấu trúc `LOGPEN` để mô tả bút, sau đó gọi hàm thành phần `CreatePenIndirect` để tạo bút:

```
CPen pen ();  
LOGPEN lp;  
lp.lopnStyle = PS_SOLID;  
lp.lopnWidth = 1;  
lp.lopnColor = RGB (255, 0, 0);  
pen.CreatePenIndirect (&lp);
```

`CreatePen` và `CreatePenIndirect` cho lại `TRUE` nếu lập được bút; ngược lại cho giá trị `FALSE`. Nếu cho phép cấu tử của đối tượng tạo lập bút, thì một ngoại lệ (*exception*) của lớp `CResourceException` được cho lại khi bút không lập được.

Một bút có ba đặc tính: kiểu, độ rộng và màu. Cả ba thí dụ trên đây đều lập bút với kiểu `PS_SOLID`, chúng có độ rộng là một đơn vị logic, màu là đỏ tươi. Tham số đầu tiên chuyển cho cấu tử của lớp `CreatePen`, `CreatePenIndirect`, hay `CPen` chỉ ra kiểu bút. Kiểu bút mô tả loại đường mà bút sẽ vẽ - liên tục, điểm, gạch nối v.v. `PS_SOLID` lập bút liên tục. Kiểu bút đặc biệt `PS_INSIDEFRAME` vẽ đường liên tục hay cung ở bên trong hộp bao - “trong khung” - của một hình đang vẽ như trên hình 2.2. Kiểu bút khác là `PS_NULL`, chúng chẳng vẽ gì cả. Tại sao ta phải lập bút `NULL`? Giả sử rằng ta muốn vẽ hình tròn màu đỏ không có viền. Nếu ta vẽ hình tròn bằng hàm `CDC::Ellipse` của MFC, Windows sẽ tự động viền hình tròn với bút hiện hành đã chọn cho

DC. Không thể bảo cho hàm `Ellipse` rằng ta không muốn viền, nhưng khi chọn bút `NULL` cho DC thì sẽ không nhìn thấy viền của hình tròn. Cách dùng chổi tô `NULL` cũng tương tự. Nếu ta muốn hình tròn có đường viền nhưng không tô bên trong, thì có thể chọn chổi `NULL` cho DC trước khi vẽ.



Hình 2.2 Sử dụng kiểu bút `PS_SOLID` và `PS_INSIDEFRAME`

Tham số thứ hai của hàm tạo bút của đối tượng `CPen` chỉ ra độ rộng bút, tính bằng đơn vị logic. Số đo vật lý phụ thuộc vào chế độ ánh xạ của DC. Trong chế độ mặc định `MM_TEXT`, một đơn vị logic bằng một *pixel*. Khi lập bút, độ rộng mặc định của nó tính bằng *pixel*. Ta có thể tạo các bút `PS_SOLID`, `PS_NULL`, và `PS_INSIDEFRAME` với độ rộng bất kỳ, nhưng các bút `PS_DASH`, `PS_DOT`, `PS_DASHDOT`, và `PS_DASHDOTDOT` chỉ có độ rộng một đơn vị logic. Nếu cho độ rộng của bút là 0 thì bút có độ rộng một *pixel*, không phụ thuộc vào chế độ ánh xạ.

Tham số cuối cùng là màu bút. Windows sử dụng mô hình màu 24 bit RGB trong đó mỗi màu có thể được tạo ra từ các màu đỏ, xanh lá cây, xanh lam (*red, green, blue*) từ 0 đến 255. Thông thường sử dụng *macro* `RGB` để tổ hợp ba giá trị màu đọc lập thành một giá trị `COLORREF` để Windows hiểu được. Vây các lệnh sau:

```
CPen pen (PS_SOLID, 1, RGB (255, 0, 0));
```

tạo bút đỏ sáng với (R=255, G=0, và B=0), còn lệnh:

```
CPen pen (PS_SOLID, 1, RGB (255, 255, 0));
```

tạo bút vàng sáng nhờ tổ hợp đỏ và xanh lá cây. Nếu màn hình không trợ giúp 24 bit màu, Windows sẽ thực hiện trộn màu (*dithering*) để có thể hiển thị màu giống nhất.

Chỉ bút `PS_INSIDEFRAME` có độ rộng hơn một đơn vị logic thì mới có khả năng sử dụng màu trộn. Khi cần, Windows ánh xạ màu của các bút khác thành màu thực gần nhất mà nó có thể hiển thị. Ta có thể chắc chắn việc nhận được màu chính xác mong muốn bằng cách gán các tổ hợp màu cơ bản - tổ hợp *red*, *green* và *blue* các giá trị 192 hay 255. Các màu này là một phần của bộ màu cơ bản mà Windows lập trình cho các thanh ghi màu của mỗi ghép nối màn hình.

Windows định nghĩa trước ba bút cho ta để có thể sử dụng ngay. Các đối tượng có sẵn của GDI được tạo nhờ hàm `CreateStockObject` mà đối tượng bút thừa hưởng từ `CGdiObject`. Các bút có sẵn được nhận ra nhờ các hằng `WHITE_PEN`, `BLACK_PEN`, và `NULL_PEN`. Tất cả đều có độ rộng một *pixel*. Các lệnh:

```
CPen pen;  
pen.CreateStockObject(WHITE_PEN);
```

tạo đối tượng `CPen` và gán bút `WHITE_PEN` có sẵn trong kho cho nó. Các đường vẽ bằng bút này sẽ tương đương khi vẽ bằng bút sau đây:

```
CPen pen;  
pen.CreatePen(PS_SOLID, 0, RGB(255, 255, 255));
```

`BLACK_PEN` là bút mặc định được chọn cho mỗi DC. Chính vì vậy mà các hàm như `CDC::LineTo` vẽ các đường đen độ rộng một *pixel* nếu không chọn bút trước. Các bút có sẵn có thể được chọn cho DC nhờ gọi `CDC::SelectStockObject` mà không cần tạo ra nó. Lệnh sau đây chọn trực tiếp bút `WHITE_PEN` cho DC:

```
dc.SelectStockObject(WHITE_PEN);
```

Lớp `CPen` còn có cấu tử để trợ giúp các kiểu bút như `PS_COSMETIC` và các bút có thể cung cấp khả năng điều khiển các thuộc tính khác của bút như hình dạng của đỉnh cuối. Các kiểu bút này còn mới đối với Windows, lần đầu tiên được đưa vào là cho Windows NT như một phần của giao diện Win32 và nay đã thích ứng cho Windows 95/98. Trong tài liệu này ta chỉ xem xét các kiểu bút cơ bản.

2.6 CHỖI TÔ

Lớp `CBrush` gói các chổi tô của GDI. Chổi tô có ba loại cơ bản: đồng nhất, sọc và mẫu (*solid*, *hatch*, và *pattern*). Chổi đồng nhất tô màu đồng nhất. Màu đồng nhất phải là màu thực, hay nếu phần cứng không cho phép màu tô hiển thị trực tiếp, thì chúng được mô phỏng bởi màu trộn. Chổi sọc sẽ tô một trong sáu kiểu sọc do MFC định nghĩa trước; phỏng theo các mẫu được sử dụng khi vẽ kỹ thuật hay kiến trúc. Chổi mẫu tô bằng bitmap. Lớp `CBrush` cung cấp cấu tử cho mỗi kiểu

chối khác nhau.

Chối đồng màu được lập chỉ bằng một bước, bằng cách truyền giá trị COLORREF cho cấu tử CBrush:

```
CBrush brush (RGB (255, 0, 0));
```

Chúng có thể được tạo theo hai bước bằng cách tạo đối tượng CBrush chưa khởi động và gọi CBrush::CreateSolidBrush:

```
CBrush brush ();  
brush.CreateSolidBrush (RGB (255, 0, 0));
```

Cả hai thí dụ trên tạo chối đồng nhất màu đỏ. Nếu muốn, chối được tạo gián tiếp nhờ CBrush::CreateBrushIndirect. Như CPen, các cấu tử của CBrush thông báo lỗi khi không tạo được chối.

Các chối sọc (*hatch*) được lập nhờ cấu tử chấp nhận cả hai chỉ số của sọc và giá trị COLORREF hoặc bằng CBrush::CreateHatchBrush. Lệnh:

```
CBrush brush (HS_DIAGCROSS, RGB (255, 0, 0));
```

tạo chối sọc, vẽ sọc nghiêng 45° so với đường nằm ngang, tương tự như các lệnh sau:

```
CBrush brush ();  
brush.CreateHatchBrush (HS_DIAGCROSS, RGB (255, 0, 0));
```

HS_DIAGCROSS là một trong sáu kiểu sọc mà ta có thể chọn. Khi vẽ bằng chối sọc, màu nền giữa các kẻ sọc là màu trắng đồng nhất trừ khi ta chỉ định thay đổi màu nền bằng hàm CDC::SetBkColor hay thay đổi chế độ nền từ OPAQUE thành TRANSPARENT nhờ hàm CDC::SetBkMode. Các lệnh sau đây vẽ hình vuông mỗi cạnh 100 đơn vị và tô màu sọc cắt chéo đen và màu nền xám nhạt:

```
CBrush brush (HS_DIAGCROSS, RGB (0, 0, 0));  
dc.SelectObject (&brush);  
dc.SetBkColor (RGB (192, 192, 192));  
Rectangle (0, 0, 100, 100);
```

Ngược lại, các lệnh sau vẽ chữ nhật sọc cắt chéo trên màu nền hiện hành:

```
CBrush brush (HS_DIAGCROSS, RGB (0, 0, 0));  
dc.SelectObject (&brush);  
dc.SetBkMode (TRANSPARENT);  
Rectangle (0, 0, 100, 100);
```

Windows cung cấp sẵn bảy chối: BLACK_BRUSH, DKGRAY_BRUSH, GRAY_BRUSH, LTGRAY_BRUSH, HOLLOW_BRUSH, NULL_BRUSH, và WHITE_BRUSH. Tất cả đều là đồng

nhất. `HOLLOW_BRUSH` và `NULL_BRUSH` là hai cách qui chiếu khác nhau đến cùng một thứ: chổi không vẽ gì. `WHITE_BRUSH` là chổi mặc định của DC. Cũng như bút trong kho, chổi trong kho được tạo nhờ `CGdiObject::CreateStockObject` hay được chọn trực tiếp cho DC nhờ `CDC::SelectStockObject`. Một lợi thế để sử dụng ba chổi kho màu xám là các chổi này luôn được sử dụng màu đồng nhất. Tạo chổi xám cho ứng dụng sẽ liên quan đến việc trộn màu chổi nếu phần cứng màn hình không đủ khả năng hiển thị trực tiếp bóng của màu.

2.7 XOÁ BÚT VÀ CHỐI TÔ

Bút và chổi tô chiếm vùng nhớ. Vì vậy, quan trọng là phải xoá chúng khi ta làm xong công việc. Trường hợp ngoại lệ là khi sử dụng bút và chổi có sẵn thì không cần xoá chúng. Khi ta lập bút hay chổi tô trong ngăn xếp bằng cách tạo đối tượng trực tiếp từ `CPen` và `CBrush`, như sau:

```
CPen pen (PS_SOLID, 1, RGB (255, 0, 0));
CBrush brush (RGB (255, 0, 0));
```

thì các đối tượng GDI liên quan được tự động xoá bằng huỷ từ của các đối tượng khi bút và chổi tô ra ngoài vùng tác động. Nếu tạo đối tượng bút và chổi bằng toán tử `new` của C++, phải chắc chắn là đã bãi bỏ các đối tượng ta lập để huỷ từ của chúng được gọi. Bút và chổi tô GDI liên quan với các đối tượng `CPen` và `CBrush` có thể xoá bằng hàm `CGdiObject::DeleteObject`.

Trong các bản trước của Windows, bút và chổi tô thường là nguyên nhân của sự “rò rỉ” tài nguyên. Do chương trình không xoá các đối tượng do nó lập ra, số lượng tài nguyên hệ thống chưa sử dụng do *Program Manager* thông báo dần dần giảm đi, vì vậy ứng dụng sẽ dừng. Trong Win 95/98 không có vấn đề này vì hệ thống theo dõi các tài nguyên đã cấp phát và tự động xoá chúng khi chương trình kết thúc. Ta vẫn còn phải xoá các đối tượng GDI khi không cần nó nữa sao cho GDI không làm đầy vùng nhớ tài nguyên trước khi kết thúc chương trình. Tưởng tượng xem hàm `OnPaint` lập bốn bút và chổi tô khi nó được gọi, nhưng lại không xoá chúng. Mỗi khi chương trình vẽ lại cửa sổ của chúng hàng trăm lần, `OnPaint` sẽ tạo hàng trăm đối tượng GDI và chiếm vùng nhớ hệ thống dành cho GDI. Ngay tức khắc trình sẽ dừng trả lời các thông điệp `WM_PAINT` vì GDI đã làm tràn bộ nhớ, và việc gọi lập bút và chổi tô sẽ gây ra lỗi. Phải giải thoát bút và chổi tô sớm nhất có thể để tránh vấn đề này. Nếu trình của ta sử dụng cùng bút và chổi tô mãi mãi, thì có thể tiết kiệm thời gian bằng cách lập các bút và chổi này sớm hơn, thí dụ trong hàm `OnCreate`, và xoá chúng ngay trước khi kết thúc chương trình, trong hàm `OnDestroy`.

Khi viết mã trình để vẽ bằng bút và chổi tô GDI, hãy luôn nhớ qui luật của lập trình GDI là: *không bao giờ xoá đối tượng GDI đang được chọn cho DC*. Một hình thức đúng đắn là lưu lại con trỏ đến đối tượng bút và chổi tô mặc định của DC mà hàm `SelectObject` cho lại, và sử dụng con trỏ này để chọn lại đối tượng cũ trước khi xoá đối tượng của ta, hay chọn bút và chổi tô có sẵn cho DC trước khi xoá bút và chổi tô do ta tạo ra. Phương pháp thứ nhất được mô tả bằng các mã trình sau

đây:

```
CPen* pMyPen = new CPen (PS_SOLID, 1, RGB (255, 0, 0));
CPen* pOldPen = dc.SelectObject (pMyPen);
// vẽ bằng bút
dc.SelectObject (pOldPen);
delete pMyPen;
```

Chú ý rằng vì `SelectObject` luôn cho lại con trỏ đến đối tượng mà nó được chọn ra, hai lệnh cuối có thể tổ hợp thành:

```
delete dc.SelectObject (pOldPen);
```

Phương pháp thứ hai làm việc như sau:

```
CPen* pMyPen = new CPen (PS_SOLID, 1, RGB (255, 0, 0));
dc.SelectObject (pMyPen);
// vẽ bằng bút
delete dc.SelectStockObject (BLACK_PEN);
```

Phương pháp nào phù hợp cho ta? Phương pháp thứ hai phù hợp hơn nếu ta chọn vài bút và chổi tô cho DC và cấu trúc của mã lệnh làm khó xác định là cái nào sẽ được chọn trước đó. Hãy chọn đối tượng có sẵn cho DC và chẳng còn phải quan tâm cái nào đã được chọn cho DC trước đó.

2.8. THÍ DỤ ỨNG DỤNG GdiObj1

Cách tốt nhất để học GDI và các lớp MFC liên quan-hay bất kỳ hệ thống đồ họa nào là viết mã trình. Hãy khởi đầu bằng ứng dụng vô cùng đơn giản. Chương trình 2.1 chứa mã nguồn của `GdiObj1`, chúng vẽ đường và hình chữ nhật với bút và chổi của các kiểu khác nhau. Đầu ra được tổ chức thành ba nhóm: Pen Styles, Pen Widths, và Brush Styles như trên hình 2.3. Dưới “Pen Styles”, ta thấy các đường vẽ với bút độ rộng một *pixel* của kiểu tương ứng. Phần “Pen Widths” cho thấy độ rộng bút khác nhau, từ hai đến 24 *pixel*. Cuối cùng, phần “Brush Styles” chứa các hình chữ nhật tô bằng sáu mẫu sọc mà `CBrush` trợ giúp và các chổi tô màu đồng nhất. Mặc dù `GdiObj1` chỉ là ứng dụng đơn giản của các bút và chổi tô, nhưng nó trình bày rất rõ ràng các bước phải trải qua để tạo ra đối tượng GDI, chọn chúng cho DC, dùng nó để vẽ. Chúng cũng vẽ nhiều ảnh với thuộc tính khác nhau như `PS_DASHDOTDOT` và `HS_BDIAGONAL`.

Chương trình 2.1 GDI OBJ1

Tệp `GDI OBJ1.H`

```
class CMyApp : public CWinApp
{
public:
```

```

    BOOL InitInstance ();
};
//-----
class CMainWindow : public CFrameWnd
{
private:
    int m_cxChar;
    int m_cyChar;
    void ShowPenStyles (CDC*, int, int);
    void ShowPenWidths (CDC*, int, int);
    void ShowBrushStyles (CDC*, int, int);
public:
    CMainWindow ();
protected:
    afx_msg int OnCreate (LPCREATESTRUCT);
    afx_msg void OnPaint ();
    DECLARE_MESSAGE_MAP ()
};
//-----
struct STYLES {
    int nStyle;
    char szStyleName[16];
};
//-----

```

Tệp GDIOBJ1.CPP

```

// GdiObj1 mô tả các kiểu bút và chổi tô cơ bản.
#include <afxwin.h>
#include "gdiobj1.h"
CMyApp myApp;
// Các hàm thành phần của CMyApp
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}
// Ánh xạ thông điệp và các hàm thành phần của CMainWindow

```

```

BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_CREATE ()
    ON_WM_PAINT ()
END_MESSAGE_MAP ()
//-----
CMainWindow::CMainWindow ()
{
    Create (NULL, "GdiObj1");
}
//-----
int CMainWindow::OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    TEXTMETRIC tm;
    CDC* pDC = GetDC ();
    pDC->GetTextMetrics (&tm);
    m_cxChar = tm.tmAveCharWidth;
    m_cyChar = tm.tmHeight;
    ReleaseDC (pDC);
    return 0;
}
//-----
void CMainWindow::OnPaint ()
{
    CPaintDC dc (this);
    CFont* pFont = new CFont ();
    pFont->CreateFont (16, 0, 0, 0, FW_NORMAL, FALSE, 0, 0, DEFAULT_CHARSET,
        OUT_CHARACTER_PRECIS, CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH | FF_DONTCARE, "VnTime");
    CFont* pOldFont = dc.SelectObject (pFont);
    ShowPenStyles (&dc, m_cxChar, m_cyChar);
    ShowPenWidths (&dc, m_cxChar*44, m_cyChar);
    ShowBrushStyles (&dc, m_cxChar, m_cyChar * 14);
    dc.SelectObject (pOldFont);
    delete pFont;
}
//-----
void CMainWindow::ShowPenStyles (CDC* pDC, int x, int y)
{
    CPen* pPen;

```

```

CPen* pOldPen;
struct STYLES styles[] = {
    PS_SOLID,    "PS_SOLID",
    PS_DASH,     "PS_DASH",
    PS_DOT,      "PS_DOT",
    PS_DASHDOT,  "PS_DASHDOT",
    PS_DASHDOTDOT, "PS_DASHDOTDOT",
    PS_NULL,     "PS_NULL",
    PS_INSIDEFRAME, "PS_INSIDEFRAME"
};

CString string = "Kiểu bút vẽ (Pen Styles)";
pDC->TextOut (x, y, string);
int dy = (m_cyChar * 3) >> 1;
pDC->SetTextColor (RGB (255, 0, 0));
for (int i=0; i<7; i++) {
    y += dy;
    string = styles[i].szStyleName;
    pDC->TextOut (x + (m_cxChar << 1), y, string);
    pPen = new CPen (styles[i].nStyle, 1, RGB (255, 0, 0));
    pOldPen = pDC->SelectObject (pPen);
    pDC->MoveTo (x + (m_cxChar * 22), y + (m_cyChar >> 1));
    pDC->LineTo (x + (m_cxChar * 38), y + (m_cyChar >> 1));
    delete pDC->SelectObject (pOldPen);
}
}
//-----
void CMainWindow::ShowPenWidths (CDC* pDC, int x, int y)
{
    CPen* pPen;
    CPen* pOldPen;
    int nPenWidths[] = { 2, 8, 16, 24 };
    CString string = "Độ rộng bút vẽ (Pen Widths)";
    pDC->SetTextColor (RGB (0, 0, 0));
    pDC->TextOut (x, y, string);
    int dy = m_cyChar << 1;
    pDC->SetTextColor (RGB (0, 0, 255));
    for (int i=0; i<4; i++) {
        y += dy;
        string.Format ("%d điểm (Pixels)", nPenWidths[i]);
    }
}

```

```

        pDC->TextOut (x + (m_cxChar << 1), y, string);
        pPen = new CPen (PS_SOLID, nPenWidths[i], RGB (0, 0, 255));
        pOldPen = pDC->SelectObject (pPen);
        pDC->MoveTo (x + (m_cxChar * 22), y + (m_cyChar >> 1));
        pDC->LineTo (x + (m_cxChar * 38), y + (m_cyChar >> 1));
        delete pDC->SelectObject (pOldPen);
    }
}

//-----
void CMainWindow::ShowBrushStyles (CDC* pDC, int x, int y)
{
    CBrush* pBrush;
    CBrush* pOldBrush;
    struct STYLES styles[] = {
        HS_BDIAGONAL, "HS_BDIAGONAL",
        HS_CROSS, "HS_CROSS",
        HS_DIAGCROSS, "HS_DIAGCROSS",
        HS_FDIAGONAL, "HS_FDIAGONAL",
        HS_HORIZONTAL, "HS_HORIZONTAL",
        HS_VERTICAL, "HS_VERTICAL"
    };
    CString string = "Kiểu chổi tô (Brush Styles)";
    pDC->SetTextColor (RGB (0, 0, 0));
    pDC->TextOut (x, y, string);
    int dy = m_cyChar * 2;
    for (int i=0; i<6; i++)
    {
        y += dy;
        string = styles[i].szStyleName;
        pDC->TextOut (x + (m_cxChar << 1), y, string);
        CRect rect (x + (m_cxChar * 22), y - m_cyChar+3, x + (m_cxChar * 46), y + m_cyChar-3);
        pBrush = new CBrush (styles[i].nStyle, RGB (0, 255, 0));
        CPoint point (rect.left, rect.top);
        pDC->LPtoDP (&point);
        point.x &= 0x07;
        point.y &= 0x07;
        pDC->SetBrushOrg (point);
        pOldBrush = pDC->SelectObject (pBrush);
        pDC->Rectangle (rect);
        delete pDC->SelectObject (pOldBrush);
    }
}

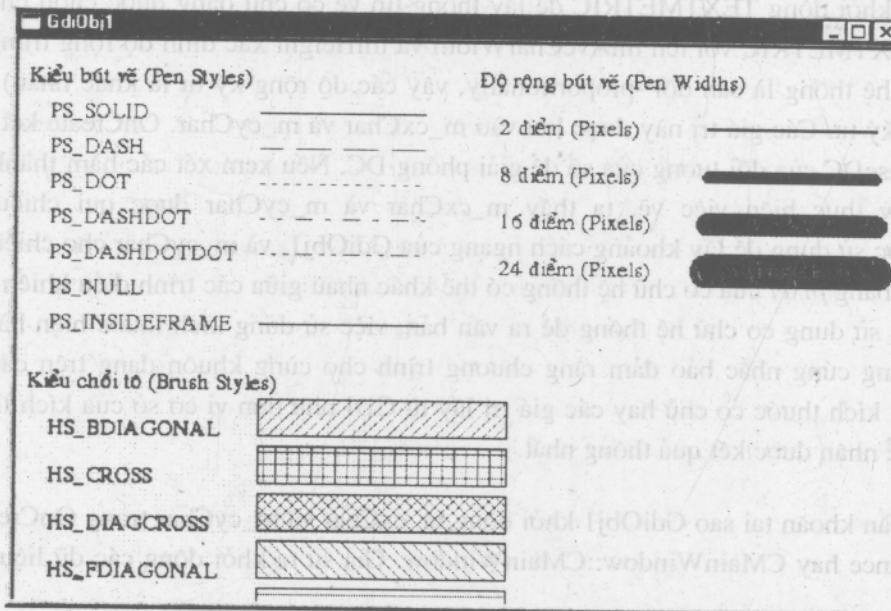
```

```

}
y += dy;
string = "Solid";
pDC->TextOut (x + (m_cxChar << 1), y, string);
pBrush = new CBrush (RGB (0, 255, 0));
pOldBrush = pDC->SelectObject (pBrush);
pDC->Rectangle (x + (m_cxChar * 22), y - m_cyChar+3, x + (m_cxChar * 46), y + m_cyChar-3);
delete pDC->SelectObject (pOldBrush);
}
//-----

```

Quan sát cấu trúc của GdiObj1, đối tượng ứng dụng được tạo từ lớp CMyApp suy diễn từ CWinApp là chính ứng dụng. CMyApp::InitInstance lập cửa sổ khung nhờ khai báo đối tượng từ lớp CMainWindow. CMainWindow::OnPaint quản lý toàn bộ công việc vẽ nhờ gọi các hàm thành phần riêng ShowPenStyles, ShowPenWidths, và ShowBrushStyles. Mỗi hàm này nhận con trỏ đến đối tượng DC và các tọa độ x, y khi bắt đầu vẽ.



Hình 2.3 Kết quả thực hiện chương trình GdiObj1

OnPaint lập đối tượng DC trong thông điệp vẽ với tên dc bằng lệnh sau đây:

```
CPaintDC dc (this);
```

Sau đó chúng chuyển địa chỉ đối tượng cho ShowPenStyles, ShowPenWidths, và

ShowBrushStyles, cùng với các tọa độ x, y chỉ ra vị trí *pixel* trong vùng làm việc của cửa sổ.

2.8.1 Hàm OnCreate

Ngay sau khi được lập, cửa sổ khung GdiObj1 nhận thông điệp WM_CREATE cho biết rằng nó đã được tạo, cho một cơ hội tốt để ứng dụng thực hiện các công việc khởi động phụ thuộc vào sự tồn tại của cửa sổ - cụ thể, việc khởi động thực sự nằm trong các hàm thành phần của lớp CWnd hay suy diễn từ CWnd và chính vì vậy nó đòi hỏi con trả hợp lệ để qui chiếu đối tượng của sổ. Hàm OnCreate của GdiObj1 nạp giá trị ban đầu cho cặp thành phần dữ liệu với tên m_cxChar và m_cyChar bằng độ rộng và cao của ký tự theo cơ chữ hệ thống mặc định.

```
CDC* pDC = GetDC ();
pDC->GetTextMetrics (&tm);
m_cxChar = tm.tmAveCharWidth;
m_cyChar = tm.tmHeight;
ReleaseDC (pDC);
```

Con trỏ DC do GetDC cho lại được lưu vào pDC, và GetTextMetrics được gọi với con trỏ đến cấu trúc chưa khởi động TEXTMETRIC để lấy thông tin về cơ chữ đang được chọn cho DC. Hai trường của TEXTMETRIC với tên tmAveCharWidth và tmHeight xác định độ rộng trung bình của ký tự (cơ chữ hệ thống là cân đối -proportionally, vậy các độ rộng ký tự là khác nhau) và độ cao của khung vẽ ký tự. Các giá trị này được lưu vào m_cxChar và m_cyChar. OnCreate kết thúc bằng gọi hàm ReleaseDC của đối tượng cửa sổ để giải phóng DC. Nếu xem xét các hàm thành phần của CMainWindow thực hiện việc vẽ, ta thấy m_cxChar và m_cyChar được qui chiếu liên tục. m_cxChar được sử dụng để lấy khoảng cách ngang của GdiObj1, và m_cyChar cho chiều đứng. Vì kích thước đo bằng *pixel* của cơ chữ hệ thống có thể khác nhau giữa các trình điều khiển màn hình, và vì GdiObj1 sử dụng cơ chữ hệ thống để ra văn bản, việc sử dụng kích thước hiện hành thay vì một giá trị hằng cứng nhắc bảo đảm rằng chương trình cho cùng khuôn dạng trên các PC khác nhau. Sử dụng kích thước cơ chữ hay các giá trị lấy từ GDI như đơn vị cơ sở của kích thước là kỹ thuật chung để nhận được kết quả thống nhất.

Ta có thể bàn khoản tại sao GdiObj1 khởi động m_cxChar và m_cyChar trong OnCreate thay vì trong InitInstance hay CMainWindow::CMainWindow. Giả sử ta khởi động các dữ liệu này trong InitInstance:

```
m_pMainWnd = new CMainWindow ();
m_pMainWnd->ShowWindow (m_nCmdShow);
m_pMainWnd->UpdateWindow ();
CDC* pDC = m_pMainWnd->GetDC ();
pDC->GetTextMetrics (&tm);
m_pMainWnd->m_cxChar = tm.tmAveCharWidth;
```

```

m_pMainWnd->m_cyChar = tm.tmHeight;
m_pMainWnd->ReleaseDC (pDC);
return TRUE;

```

Sự thật đây là cách làm tồi khi một đối tượng khởi động các dữ liệu thành phần của đối tượng khác, càng sâu hơn càng khó hiểu hơn trong mã trình. Hàm OnPaint của chương trình giả sử m_cxChar và m_cyChar được khởi động trước khi thông điệp WM_PAINT đầu tiên đến. Nhưng UpdateWindow tạo ra thông điệp WM_PAINT đầu tiên, như thí dụ này, khi m_cxChar và m_cyChar vẫn chưa được khởi động lần đầu trước khi tác động đến hàm OnPaint. Ta có thể hiệu chỉnh bằng cách gọi UpdateWindow sau khi khởi động m_cxChar và m_cyChar. Phương án này có thể hay hơn trước, nhưng vẫn chưa ổn khi trình chứa hàm xử lý thông điệp khác liên quan đến thành phần dữ liệu này được khởi động sớm hơn trong trình. Lệnh này xem ra đơn giản:

```
m_pMainWnd = new CMainWindow ();
```

Nhưng vào thời điểm khi lệnh này trở lại, cửa sổ đã nhận nhiều thông điệp, bao gồm WM_CREATE, WM_SIZE, và những thông điệp khác. Vậy, OnCreate là nơi an toàn nhất để khởi động bất kỳ thành phần dữ liệu nào. Điều này quan trọng khi ta thêm hàm OnSize vào trình GdiObj1 có liên quan đến m_cyChar.

Khởi động m_cxChar và m_cyChar trong cấu tử của lớp CMainWindow có được không? Không may, ở đây có vấn đề thời gian nếu các hàm xử lý các thông điệp khác tin tưởng vào giá trị được lưu trong m_cxChar và m_cyChar. Nếu ta viết lại cấu tử như sau:

```

CDC* pDC = GetDC ();
pDC->GetTextMetrics (&tm);
m_cxChar = tm.tmAveCharWidth;
m_cyChar = tm.tmHeight;
ReleaseDC (pDC);
Create (NULL, "GdiObj1");

```

thì những lời gọi GetDC, GetTextMetrics, và ReleaseDC có thể thất bại vì cửa sổ chưa tồn tại trước khi nó được lập nhờ Create. Việc đảo lại thứ tự để Create được gọi trước cũng chẳng giúp ích gì vì cửa sổ sẽ nhận một loạt thông điệp trước khi Create trở lại.

Trên cơ sở này, dưới đây là một quy luật đơn giản phải tuân thủ khi quyết định việc khởi động các thành phần dữ liệu ở trong cấu tử hay là trong hàm OnCreate. Nếu thành phần dữ liệu được hàm xử lý thông điệp khác sử dụng, đặc biệt các thông điệp được tự động gửi đến khi lập cửa sổ, thì khởi động chúng trong OnCreate. Vì WM_CREATE là thông điệp đến sớm nhất mà cửa sổ nhận được, nó đảm bảo rằng biến được khởi động trước mọi thông điệp khác đến. Nếu không, hãy để cấu tử thực hiện khởi động. Tuy nhiên, hãy luôn nhớ rằng các hàm cửa sổ như GetDC và ReleaseDC sẽ không làm việc cho đến khi cửa sổ được tạo ra.

2.8.2 Hàm ShowPenStyles

GdiObj1 thực hiện toàn bộ vẽ ra màn hình để trả lời các thông điệp WM_PAINT. Hàm thứ nhất trong ba hàm thành phần riêng mà OnPaint gọi để vẽ là ShowPenStyles, chúng tạo bảy bút với các kiểu khác nhau và sử dụng chúng để vẽ đường thẳng ngang có độ rộng một *pixel*. Vấn đề đầu tiên mà ShowPenStyles làm là hiển thị xâu “Kiểu bút vẽ (*Pen Styles*)” bằng các lệnh sau đây:

```
CString string = "Kiểu bút vẽ (Pen Styles)";
pDC->TextOut(x, y, string);
```

pDC là con trỏ đến đối tượng DC do hàm OnPaint cung cấp, x và y là vị trí khởi đầu của hàm ra. Đối tượng string được lập từ lớp CString của MFC (*chú ý rằng toán tử = của CString được sử dụng để khởi động xâu string*), và TextOut là hàm thành phần của lớp CDC để thực hiện hiển thị văn bản. Nó tương tự như CDC::DrawText trong một nghĩa nào đó, nhưng DrawText thường dùng để tạo khuôn và ra xâu ký tự trong hình chữ nhật, TextOut là hàm tổng quát hơn như hàm puts của C. Mặc định, tọa độ được truyền cho TextOut, chỉ ra góc trên-trái của ký tự trái nhất của xâu string. Đặc tính này, được gọi là *sắp thẳng hàng xâu ký tự*, có thể thay đổi khi gọi CDC::SetTextAlign. Nếu muốn, SetTextAlign được sử dụng để ra lệnh cho TextOut bỏ qua các tham số x, y truyền cho nó và sử dụng vị trí hiện hành được xác định bằng CDC::MoveTo. Lợi thế của phương pháp này là mỗi lần gọi TextOut sẽ tự động cập nhật giá trị x của vị trí hiện hành để cung cấp khoảng cách đúng giữa hai hay nhiều xâu ký tự có độ rộng khác nhau trên cùng một dòng.

Tiếp theo, ShowPenStyles khởi động biến có tên dy bằng giá trị 3/2 của m_cyChar, và gọi hàm SetTextColor của đối tượng DC để thay đổi màu xâu ký tự thành đỏ tươi.

```
int dy = (m_cyChar * 3) >> 1;
pDC->SetTextColor( RGB(255, 0, 0));
```

dy là giá trị nguyên mà ShowPenStyles dùng để đo khoảng cách giữa các đường liên tiếp nhau. SetTextColor chấp nhận giá trị COLORREF xác định màu của xâu được vẽ. SetTextColor xác định màu *dither*, Windows tự động thay thế màu đồng nhất gần nhất. Vì vậy, không phải băn khoăn rằng màu dither làm mờ xâu ký tự của ta.

Bây giờ xem xét lõi của chương trình vẽ. ShowPenStyles thực hiện vòng for để lấy kiểu bút từ trường kiểu, dùng TextOut hiển thị tên kiểu, lập bút, chọn bút cho DC, và vẽ các đường nằm ngang rộng một *pixel* ở phía phải tên kiểu bút. Sau khi vẽ đường, bút gốc (*cái bị loại ra DC khi chọn bút mới*) được chọn lại cho DC và bút mới bị loại. Toàn bộ điều khiển bút được thực hiện bằng năm dòng lệnh sau:

```
pPen = new CPen(styles[i].nStyle, 1, RGB(255, 0, 0));
pOldPen = pDC->SelectObject(pPen);
pDC->MoveTo(x + (m_cxChar * 22), y + (m_cyChar >> 1));
pDC->LineTo(x + (m_cxChar * 46), y + (m_cyChar >> 1));
```

```
delete pDC->SelectObject (pOldPen);
```

Như đã giới thiệu trên đây, bút gốc được chọn lại cho DC và bút mới bị loại trong cùng một bước nhờ giá trị cho lại của SelectObject.

2.8.3 Hàm ShowPenWidths

ShowPenWidths tương tự như ShowPenStyles, chúng vẽ bằng bút có độ rộng khác nhau thay vì kiểu khác nhau. Trường nPenWidths chứa độ rộng của bút và các lệnh sau đây vẽ các đường:

```
pPen = new CPen (PS_SOLID, nPenWidths, RGB (0,0,255));
pOldPen = pDC->SelectObject (pPen);
pDC->MoveTo (x + (m_cxChar * 22), y + (m_cyChar >> 1));
pDC->LineTo (x + (m_cxChar * 46), y + (m_cyChar >> 1));
delete pDC->SelectObject (pOldPen);
```

Cũng như trong ShowPenStyles, mỗi vòng lặp sẽ tạo bút mới, chọn nó cho DC, vẽ một đường bằng hàm MoveTo và LineTo, và xoá bút sau khi đã chọn lại bút gốc cho DC.

Một khía cạnh thú vị của hàm này là cách thức tạo khuôn xâu văn bản để hiển thị ở đầu của mỗi dòng ra. Thay vì các xâu cứng nhắc trong trường như ShowPenStyles làm, ShowPenWidths xây dựng các xâu khi chương trình chạy bằng hàm Format của đối tượng CString:

```
string.Format ("%d điểm (Pixels)", nPenWidths[i]);
```

Format làm việc như hàm printf() của C, chuyển đổi các biến kiểu số thành dạng văn bản và thay thế chúng vào vị trí tương ứng trong xâu khuôn dạng. Lập trình viên quen với C thường sử dụng hàm API của Windows tên là ::wsprintf để tạo dạng văn bản. Hàm Format làm cùng loại việc này cho đối tượng CString mà không phải gọi hàm ngoài.

2.8.4 Hàm ShowBrushStyles

Hàm thứ ba trong ba hàm mà hàm OnPaint của GdiObj1 gọi là để vẽ dãy các hình chữ nhật bao quanh bằng đường đen rộng một *pixel* với các kiểu chổi tô khác nhau. Lần nữa, việc vẽ được thực hiện trong vòng for lấy các kiểu sọc từ một trường và tạo động các chổi tô khác nhau. Sau khi mỗi chổi tô được lập, nó được chọn cho DC và chữ nhật được vẽ bởi các lệnh sau:

```
pOldBrush = pDC->SelectObject (pBrush);
pDC->Rectangle (rect);
```

Vì hàm CDC:Rectangle vẽ chữ nhật có viền bút hiện hành và tô bằng chổi hiện hành, các lệnh đơn giản này vẽ các kiểu chổi HS_XXX. Sau khi mỗi chữ nhật được vẽ, chổi tô cũ được chọn lại cho DC và chổi mới bị loại bằng lệnh:

```
delete pDC->SelectObject (pOldBrush);
```

Khi vòng *for* kết thúc, `ShowBrushStyles` vẽ thêm chữ nhật với chổi tô màu xanh đồng nhất. Trước khi vẽ chữ nhật với chổi sọc, `ShowBrushStyles` đặt gốc chổi tô đến góc trên-trái của chữ nhật để sắp thẳng hàng chổi trên cửa sổ. Đặt lại gốc chổi tô không bắt buộc cần thiết ở đây, vì chữ nhật luôn được vẽ ở cùng vị trí tương đối với góc trên-trái của vùng làm việc. Nhưng khi `GdiObj1` có thêm khả năng trượt, việc đặt chổi tô cho đúng trở nên đặc biệt quan trọng để trình có được đầu ra có chất lượng.

2.8.5 Xem những gì ta đã vẽ

Không may là có một vấn đề nhỏ với đầu ra của `GdiObj1` là trừ khi đang chạy chương trình trên vi màn hình phân giải cao, ta không thể thấy toàn bộ những gì đã vẽ. Thậm chí trên màn hình 1024x768, cửa sổ không thể nhồi nhét tất cả những gì đã vẽ vào màn hình để ta có thể xem được. Và những gì không cho được vào cửa sổ thì chúng bị cắt (*clipp*) bởi GDI. Ta có thể trở lại thí dụ trên và sửa đổi chúng để cho mạnh hơn, quan sát được đầy đủ đầu ra. Vậy, hãy thêm thanh trượt vào cửa sổ.

2.9. THÊM THANH TRƯỢT VÀO CỬA SỔ

Đây là một trong các vấn đề dễ nhất trong lập trình Windows. Đơn giản là lập cửa sổ với thuộc tính kiểu `WS_VSCROLL` hoặc `WS_HSCROLL` (V là dọc - *vertical*, H là ngang - *horizontal*). Nhớ lại rằng tham số thứ ba được chuyển cho `CFrameWnd::Create` chỉ ra kiểu của cửa sổ. Vì vậy, khi ứng dụng lập cửa sổ khung chuẩn bằng lệnh:

```
Create (NULL, "MyApp");
```

thì có thể lập cửa sổ khung chứa thanh trượt đứng bằng lệnh:

```
Create (NULL, "MyApp", WS_OVERLAPPEDWINDOW | WS_VSCROLL);
```

Windows sẽ thêm thanh trượt và sẽ mở rộng cửa sổ sang phải.

Cũng có cách làm phức tạp hơn. Ta tạo riêng cửa sổ thanh trượt như cửa sổ con của cửa sổ khung và đặt nó vào trong viền phải của cha nó. Đôi khi phải làm cách này để tạo thanh trượt nhỏ hơn kích thước cửa sổ hay đặt vào vị trí cố định. Nhưng trong thí dụ `GdiObj1` chỉ cần sử dụng `WS_VSCROLL`.

2.9.1 Đặt dải trượt, vị trí và kích thước trang cho thanh trượt

Trước khi sử dụng thanh trượt, phải nạp trước dải, vị trí và kích thước trang cho chúng. Dải trượt là cặp số nguyên xác định giới hạn thấp nhất, cao nhất của thanh trượt. Vị trí là một giá trị nguyên

chỉ ra vị trí hiện hành trong dải và nó phản ánh vị trí của con chạy của thanh trượt. Kích thước trang là đặc trưng mới có trong Win95/98, xác định vị trí tương đối của cửa sổ với toàn bộ tài liệu. Ta không cần đặt kích thước trang (*Windows sẽ tự lấy kích thước mặc định cho ta*).

Một cách đặt dải và vị trí của thanh trượt là sử dụng hàm `CWnd::SetScrollRange` và `CWnd::SetScrollPos`. Lệnh:

```
SetScrollRange(SB_VERT, 0, 99, TRUE);
```

đặt dải trượt đúng từ 0 đến 99, còn lệnh:

```
SetScrollPos(SB_VERT, 50, TRUE);
```

đặt vị trí hiện hành là 50 và làm cho con chạy chuyển đến giữa thanh trượt. (Với thanh trượt ngang, sử dụng `SB_HORZ` thay cho `SB_VERT`. Windows yêu cầu chỉ báo này để hàm gọi thực hiện cho thanh trượt nào khi cả hai thanh trượt được sử dụng). Thanh trượt quản lý vị trí hiện hành và dải trượt bên trong Windows, ta có thể lấy được giá trị này vào bất kỳ thời điểm nào bằng hàm `GetScrollRange` và `GetScrollPos` của lớp `CWnd`.

Tham số `TRUE` được truyền cho các hàm `SetScrollRange` và `SetScrollPos` chỉ ra rằng thanh trượt cần được hiển thị lại để phản ánh sự thay đổi của nó. Có thể tránh vẽ lại khi cho tham số giá trị `FALSE`. Nếu không chỉ ra giá trị nào thì mặc định `SetScrollRange` và `SetScrollPos` sẽ có giá trị `TRUE`. Thông thường ta muốn thanh trượt vẽ lại mỗi khi sử dụng một trong các hàm này, nhưng không phải khi cả hai được gọi liên tục, vì thanh trượt vẽ lại hai lần trong khoảng thời gian ngắn sẽ gây ra hiệu ứng không mong muốn. Nếu ta lập cả dải và vị trí trong cùng bước bước thì hãy làm như sau:

```
SetScrollRange(SB_VERT, 0, 99, FALSE);
```

```
SetScrollPos(SB_VERT, 50, TRUE);
```

Gọi hàm nào trước cũng được, hàm sau sẽ làm nhiệm vụ vẽ lại. Nhưng nhớ rằng nếu gọi `SetScrollPos` trước, thì hàm gọi này sẽ gây lỗi nếu vị trí mới của con chạy thanh trượt nằm ngoài dải hiện hành.

Trong Windows 95/98, cách hay hơn để lập dải và vị trí thanh trượt là sử dụng hàm `CWnd::SetScrollInfo`. Hơn nữa để có thể lập dải trượt và vị trí trong cùng lời gọi hàm, `SetScrollInfo` còn cung cấp phương tiện để lập kích cỡ trang. `SetScrollInfo` chấp nhận các tham số: `SB_VERT` hay `SB_HORZ`, con trỏ đến cấu trúc `SCROLLINFO` và giá trị `Bool` cho biết có cần vẽ lại hay không. `SCROLLINFO` được khai báo như sau:

```
typedef struct tagSCROLLINFO {
    WORD fMask;
    int nMin;
    int nMax;
```

```
int nPage;
int nPos;
} SCROLLINFO;
```

nMin và nMax xác định dải thanh trượt, nPage xác định kích thước trang, và nPos xác định vị trí. Trường fMask lưu tổ hợp của một hay nhiều cờ bit sau: SIF_DISABLENOSCROLL làm mất tác dụng thanh trượt; SIF_PAGE, chỉ ra rằng nPage lưu cỡ trang; SIF_POS, cho biết nPos lưu vị trí thanh trượt; và SIF_RANGE cho biết nMin và nMax lưu dải thanh trượt. SetScrollInfo bỏ qua các trường mà cờ bit không xác định. Các lệnh sau đây lập vị trí và bỏ qua dải và kích thước trang:

```
SCROLLINFO si;
si.fMask = SIF_POS;
si.nPos = 50;
SetScrollInfo(SB_VERT, &si, TRUE);
```

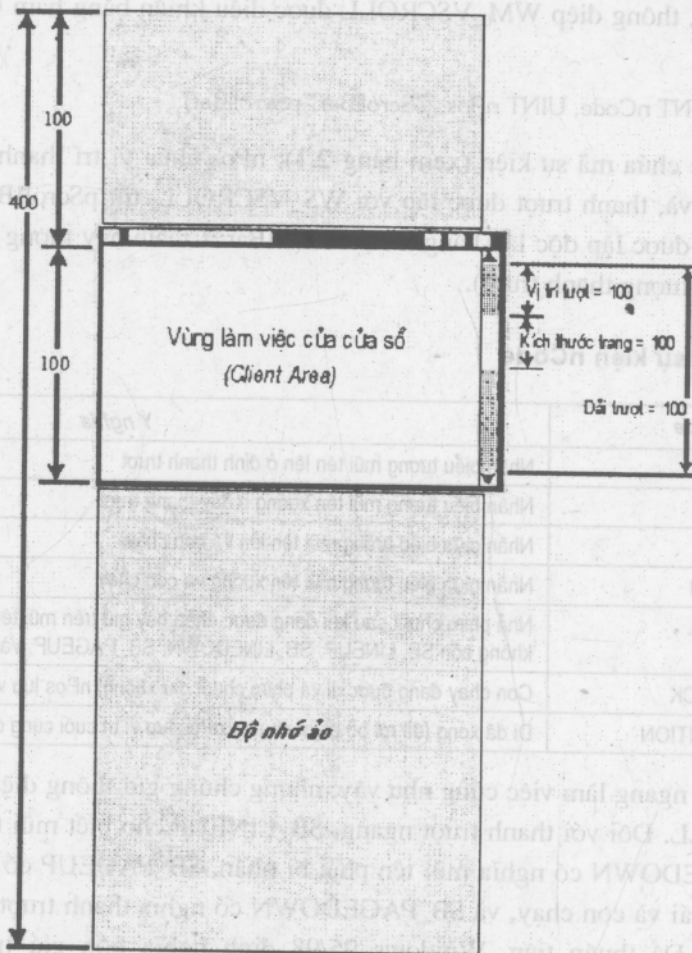
còn các lệnh sau lập dải trượt, vị trí, kích thước trang:

```
SCROLLINFO si;
si.fMask = SIF_RANGE | SIF_POS | SIF_PAGE;
si.nMin = 0;
si.nMax = 99;
si.nPos = 50;
si.nPage = 25;
SetScrollInfo(SB_VERT, &si, TRUE);
```

Hàm bù GetScrollInfo() lấy thông tin do SetScrollInfo lập.

Ta có thể làm thanh trượt biến đi bằng cách lập giới hạn trên của dải bằng giới hạn dưới. Thanh trượt không hoàn toàn mất đi; nó vẫn còn ở đấy, mặc dù ta không nhìn thấy chúng, và - quan trọng hơn - có thể lấy lại nó bằng cách lập lại dải khác không. Đây là một thủ thuật có lợi nếu như ta muốn dấu thanh trượt bởi vì cửa sổ được rộng ra tới điểm mà thanh trượt không cần nữa. Cờ SIF_DISABLENOSCROLL của SetScrollInfo làm thanh trượt không còn nhận được đầu vào, nhưng nó không làm thanh trượt biến đi. Thanh trượt bị mất tác dụng vẫn nhìn thấy trong cửa sổ làm người sử dụng lúng túng, họ băn khoăn tại sao nó vẫn còn đấy mà lại không sử dụng được. Luôn luôn có trường hợp ngoại lệ, nhưng tổng quát thì nên làm mất thanh trượt không cần thiết đi thay vì chỉ làm nó vô hiệu lực. Các thanh trượt cũng có thể dấu đi và hiển thị nhờ hàm CWnd::ShowScrollBar.

Khi lập dải trượt, cỡ trang và vị trí của thanh trượt, quan trọng là tuân thủ mô hình mà Windows 95/98 sử dụng cho thanh trượt, dải biểu diễn kích cỡ của vùng làm việc (scrollable view - workspace), và vị trí hiện hành của cửa sổ tương quan với vùng làm việc. Giả sử cửa sổ cao 100 đơn vị và vùng làm việc mà ta muốn phủ bằng thanh trượt đứng cao 400 đơn vị, như trên hình 2.4.



Hình 2.4 Mô hình thanh trượt của Windows 95/98

Cách dễ nhất để đặt cấu hình thanh trượt là lập dải 0 đến 399 và cỡ trang là 100. Windows sẽ vẽ con chạy thanh trượt bằng 1/4 độ cao của thanh trượt. Khi vị trí thanh trượt là 0, con chạy sẽ ở đỉnh của thanh trượt. Khi con chạy trượt xuống, ta trượt nội dung của cửa sổ lên một khoảng tương ứng với con chạy đã trượt. Và nếu ta giới hạn vị trí cực đại của thanh trượt là 300 (hiệu giữa lượng dải và cỡ trang), đáy của con chạy gặp đáy của thanh trượt cùng thời điểm mà đáy của vùng làm việc trượt vào tầm nhìn trên đáy của cửa sổ. Còn có nhiều cách làm khác, dĩ nhiên, nhưng mô hình đơn giản này là dễ hiểu và dễ nhìn.

2.9.2 Xử lý các thông điệp của thanh trượt

Thanh trượt gửi các thông điệp WM_VSCROLL để thông báo cho cửa sổ là có sự kiện xảy

ra. Trong MFC, thông điệp WM_VSCROLL được điều khiển bằng hàm OnVScroll thông qua ánh xạ thông điệp:

OnVScroll (UINT nCode, UINT nPos, CScrollBar* pScrollBar)

trong đó nCode chứa mã sự kiện (xem bảng 2.1); nPos chứa vị trí thanh trượt nếu con chạy trượt đến vị trí mới; và, thanh trượt được lập với WS_VSCROLL, thì pScrollBar có giá trị NULL. (Với các thanh trượt được lập độc lập bằng hàm CScrollBar::Create hay tương đương, pScrollBar lưu trữ con trỏ đến đối tượng thanh trượt).

Bảng 2.1 Các sự kiện nCode

<i>nCode</i>	<i>Ý nghĩa</i>
SB_LINEUP	Nhấn biểu tượng mũi tên lên ở đỉnh thanh trượt
SB_LINEDOWN	Nhấn biểu tượng mũi tên xuống ở đáy thanh trượt
SB_PAGEUP	Nhấn giữa biểu tượng mũi tên lên và con chạy
SB_PAGEDOWN	Nhấn giữa biểu tượng mũi tên xuống và con chạy
SB_ENDSCROLL	Nhả phím chuột sau khi đang được nhấn hay giữ trên mũi tên hay thanh trượt, kết quả là không còn SB_LINEUP, SB_LINEDOWN, SB_PAGEUP, và SB_PAGEDOWN đi đến nữa.
SB_THUMBTRACK	Con chạy đang được di và phím chuột giữ xuống; nPos lưu vị trí cuối cùng của con chạy
SB_THUMBPOSITION	Di đã xong (đã rời bỏ phím chuột); nPos lưu vị trí cuối cùng của con chạy

Thanh trượt ngang làm việc cũng như vậy, nhưng chúng gửi thông điệp WM_HSCROLL thay vì WM_VSCROLL. Đối với thanh trượt ngang, SB_LINEUP cho biết mũi tên trái trên thanh trượt bị nhấn, SB_LINEDOWN có nghĩa mũi tên phải bị nhấn, SB_PAGEUP có nghĩa thanh trượt bị nhấn giữa mũi tên trái và con chạy, và SB_PAGEDOWN có nghĩa thanh trượt bị nhấn giữa con chạy và mũi tên phải. Để thuận tiện, Windows 95/98 định nghĩa bốn giá trị mới – SB_LINELEFT, SB_LINERIGHT, SB_PAGELEFT, và SB_PAGERIGHT – chúng được sử dụng thay cho UP/DOWN. Nhớ rằng, tài liệu này mô tả thanh trượt đứng, nhưng mọi thứ đều có thể áp dụng cho thanh trượt ngang.

Nếu ta nhấn thanh trượt hoặc mũi tên của thanh trượt và bỏ phím chuột, thì trình sẽ nhận dãy thông điệp SB_LINEUP, SB_LINEDOWN, SB_PAGEUP, hoặc SB_PAGEDOWN liên tiếp, tương tự như mã phím của bàn phím khi giữ phím. SB_ENDSCROLL kết thúc dãy thông điệp UP/DOWN và thông báo rằng đã bỏ phím chuột. Thậm chí chỉ với một lần nhấn thanh trượt hay mũi tên cũng tạo ra thông điệp UP/DOWN tiếp sau SB_ENDSCROLL. Tương tự, khi nhấn và di con chạy thanh trượt sẽ có các thông điệp SB_THUMBTRACK cho biết vị trí mới của con chạy, và nó nhận một thông điệp SB_THUMBPOSITION khi bỏ phím chuột cho biết việc di đã kết thúc.

Trình ứng dụng trả lời các thông điệp này như thế nào? Phần lớn các trình không quan tâm đến thông điệp SB_ENDSCROLL và thay vì trả lời thông điệp SB_LINEUP, SB_LINEDOWN, SB_PAGEUP, và SB_PAGEDOWN. Thông thường trả lời cho thông điệp SB_LINEUP và

SB_LINEDOWN là trượt nội dung của cửa sổ xuống hay lên một dòng và gọi hàm SetScrollPos hoặc SetScrollInfo để lập vị trí thanh trượt mới và làm mới vị trí con chạy. “Dòng” có thể là một pixel, hay độ cao của một dòng văn bản; hoàn toàn phụ thuộc vào ta. Tương tự, trả lời cho các thông điệp SB_PAGEUP và SB_PAGEDOWN là trượt nội dung của cửa sổ xuống hay lên một “trang” trọn vẹn, nó thường được định nghĩa là độ cao của vùng làm việc của cửa sổ, tiếp theo là lời gọi để lập vị trí trượt mới. Khi có bất kỳ sự kiện nào, nhiệm vụ của lập trình viên là cập nhật vị trí thanh trượt khi ta trượt thông tin trong cửa sổ; thanh trượt không tự làm điều đó. Quan điểm ít tổng quát hơn là xử lý thông điệp UP/DOWN là chuyển liên tục con chạy thanh trượt nhờ gọi hàm SetScrollPos hay SetScrollInfo, nhưng trượt cửa sổ chỉ khi có thông điệp SB_ENDSCROLL. Một khi kỹ thuật này được sử dụng thì tránh được sự chuyển động không đều của thanh trượt trong ứng dụng đa phương tiện.

Các thông điệp SB_THUMBTRACK và SB_THUMBPOSITION được xử lý khác nhau. Vì thông điệp SB_THUMBTRACK đi đến khá nhanh khi con chạy bị di, nhiều ứng dụng bỏ qua thông điệp này, chỉ xử lý thông điệp SB_THUMBPOSITION. Vì vậy, cửa sổ không vị trượt cho đến khi rời bỏ con chạy.

Nếu có thể trượt nội dung của cửa sổ nhanh đủ theo kịp các thông điệp SB_THUMBPOSITION, thì ta có thể thêm chức năng khá đẹp vào trình bày trượt tương tác như di con chạy. Khi xử lý các thông điệp UP/DOWN từ điều khiển thanh trượt, vẫn còn phụ thuộc vào lập trình viên việc cập nhật vị trí thanh trượt mỗi khi ta trượt thông tin trong cửa sổ. Windows trang trí chuyển động của con chạy như là chúng được di lên hay xuống, nhưng nếu bị lỗi khi gọi hàm SetScrollPos hoặc SetScrollInfo để trả lời thông điệp SB_THUMBPOSITION hoặc SB_THUMBTRACK, thì con chạy sẽ trượt trở lại vị trí gốc của nó ngay khi rời bỏ con chạy. Khi thông điệp SB_THUMBPOSITION hay SB_THUMBTRACK kích hoạt hàm OnVScroll, tham số hàm nPos lưu vị trí trượt tương ứng với vị trí con chạy hiện hành. Giá trị của nPos không được định nghĩa cho các thông điệp UP/DOWN.

2.9.3 Trượt nội dung cửa sổ

Sau khi đã biết thanh trượt làm việc như thế nào, bây giờ là lúc nghĩ đến việc trượt nội dung của cửa sổ để đáp lại các thông điệp của thanh trượt. Sau đây là một cách tiếp cận.

Trước hết, ta viết hàm OnPaint sao cho chúng sử dụng vị trí trượt để xác định cái gì sẽ xuất hiện trong dòng đầu tiên của cửa sổ. Thí dụ, nếu cửa sổ hiển thị 400 dòng văn bản, và vị trí trượt hiện hành là 100, thì OnPaint bỏ qua 100 dòng và bắt đầu vẽ lại từ dòng 101. Vị trí trượt có thể nhận được từ thanh trượt hay gọi hàm GetScrollPos, hay quản lý chúng bằng thành phần dữ liệu riêng.

Sau đó, khi thông điệp thanh trượt đến, ta phải cập nhật vị trí thanh trượt và gọi hàm

InvalidateRect mà cửa sổ thừa hưởng từ CWnd để yêu cầu vẽ lại. Điều này sẽ kích hoạt OnPaint và vẽ lại cửa sổ sử dụng vị trí trượt mới.

Vấn đề tồn tại của kỹ thuật này là tốc độ chậm. Nếu người sử dụng trượt một dòng một lần, chỉ có một dòng mới sẽ hiển thị trên màn hình; toàn bộ các dòng khác đã ở đấy, nhưng sai vị trí. Chính vì vậy, thật lãng phí khi vẽ lại toàn màn hình. Một quan điểm hiệu quả hơn khi xử lý chẳng hạn các thông điệp SB_LINEUP là sao chép tất cả những gì hiển thị trong cửa sổ xuống một dòng bằng cách dịch chuyển mảng *pixel*, và chỉ vẽ lại đúng hàng trên cùng. Đó là cách làm của hàm ScrollWindow đã được thiết kế cho CWnd.

ScrollWindow trượt vùng làm việc của cửa sổ - một phần hay toàn bộ - lên hay xuống, sang trái hay phải với số lượng một hay nhiều *pixel* nhờ chuyển nhanh khối. Hơn nữa, chúng chỉ làm mất hiệu lực phần cửa sổ không “*phủ kín*” vì trượt, vậy thông điệp WM_PAINT tiếp theo sẽ không phải vẽ lại toàn bộ vùng làm việc. Để trượt nội dung của cửa sổ xuống 10 *pixel*, hàm ScrollWindow thực hiện sao chép khối. Sau đó chúng làm mất hiệu lực hình chữ nhật lưu 10 dòng trên cùng của cửa sổ. Điều này kích hoạt OnPaint để vẽ lại 10 dòng trên. Thậm chí khi OnPaint thử vẽ lại toàn bộ vùng làm việc, hiệu quả vẫn được tăng lên vì phần lớn đầu ra sẽ bị cắt. Hàm OnPaint thực sự thông minh khi xác định được chữ nhật mất hợp lệ và chỉ gọi các hàm thực hiện vẽ lại các *pixel* trong chữ nhật mất hợp lệ đó mà thôi. Do điều này làm phức tạp cho logic lập trình, chúng chỉ nên được thực hiện khi không thoả mãn với phương pháp trên.

Thường là cửa sổ được vẽ lại khi nhận thông điệp WM_PAINT, nhưng thông điệp này chỉ được khai thác sau các thông điệp khác, khi thông điệp WM_PAINT đi đến mà còn thông điệp WM_PAINT trước chưa được xử lý thì các chữ nhật bất hợp lệ được tổ hợp lại và các thông điệp được gộp lại thành một. Kết quả là, vùng không hợp lệ sinh ra khi gọi ScrollWindow không được vẽ lại ngay tức thì. Để trượt trọn chu từng dòng bit, ta có thể gọi UpdateWindow sau ScrollWindow để cập nhật cửa sổ ngay. Lập trình viên có thể băn khoăn về sự khác nhau giữa các hàm CWnd::InvalidateRect và CWnd::UpdateWindow (hay giữa các hàm API của Windows có cùng tên).

ScrollWindow chấp nhận bốn tham số trong đó có hai bắt buộc và hai tùy ý:

```
CWnd::ScrollWindow (int xAmount, int yAmount,  
                    LPCRECT lpRect = NULL, LPCRECT lpClipRect = NULL);
```

trong đó xAmount và yAmount được gán số nguyên chỉ ra tổng số *pixel* sẽ trượt ngang và đứng. Giá trị âm cho trượt trái và lên, và ngược lại, giá trị dương cho trượt phải và xuống. lpRect trỏ đến đối tượng CRect hay cấu trúc RECT cho biết chữ nhật cắt. Nếu lpRect và lpClipRect có giá trị NULL thì trượt toàn bộ vùng làm việc. Lệnh:

```
ScrollWindow (0, 10);
```

trượt mọi thứ trong vùng làm việc của cửa sổ xuống 10 *pixel* và nhắc nhở phải vẽ lại 10 dòng trên cùng. Các lệnh:

```
ScrollWindow (0, 10);
UpdateWindow ();
```

làm cùng công việc, nhưng yêu cầu phải vẽ lại ngay 10 dòng trên cùng.

Ta có thể sử dụng `ScrollWindow` khi ứng dụng hiển thị văn bản, đồ họa hay cả hai. Nhớ rằng trong Windows, mọi thứ đều được hiển thị như ảnh đồ họa, bao gồm các dòng ký tự. Vì vậy, `ScrollWindow` cũng rất hiệu quả khi sử dụng cho trượt bảng xâu ký tự như trượt mảng bitmap lớn.

2.9.4 Làm thanh trượt một cách dễ dàng

Tạo thanh trượt có thể là dễ, nhưng để chúng làm việc được lại yêu cầu phải có nhiều công sức. Windows giúp thực hiện một số công việc quản lý thanh trượt, nhưng chính lập trình viên phải cung cấp logic trượt, phải đảm bảo rằng con chạy có kích cỡ đúng so với khoảng nhìn, làm thanh trượt rời đi nếu cửa sổ tăng đủ lớn, v.v. Có thể thấy trợ giúp thanh trượt phần lớn là công việc bằng tay - hy vọng thư viện lớp có thể giúp ta nhiều trong công việc này.

Hiện tại, cửa sổ trượt là thí dụ hoàn hảo để thấy được MFC làm nhẹ việc lập trình như thế nào. MFC cung cấp lớp đặc biệt gọi là `CScrollView` gói các chức năng của cửa sổ với thanh trượt. Một ứng dụng sử dụng `CScrollView` không cần hàm `OnVScroll` bởi vì thư viện lớp cung cấp hàm quản lý và toàn bộ logic kèm theo. Điều cần phải hiểu rõ là `CScrollView` là lớp quan sát, được thiết kế để sử dụng liên kết với các đối tượng khác của ứng dụng tài liệu/quan sát. Để sử dụng cho ứng dụng loại khác (không phải tài liệu/quan sát) thì việc liên kết thanh trượt với dữ liệu chương trình có phức tạp hơn chút ít.

2.9.5 Lớp `CScrollView`

`CScrollView` là lớp quan sát (*view*) suy diễn từ `CView`. Trong MFC, quan sát là kiểu đặc biệt của cửa sổ thiết kế để hiển thị phần dữ liệu lưu trong tài liệu. Tài liệu cho trước có thể có rất nhiều quan sát liên quan với nó, mỗi quan sát cung cấp một biểu diễn khác nhau của dữ liệu tài liệu. Mặc dù bản chất thực của quan sát thường bị khung làm việc che khuất, nhưng thực tế quan sát cũng chỉ là cửa sổ không khác gì với các cửa sổ khác nhìn thấy trên màn hình. Khi *AppWizard* lập ứng dụng có giao diện đơn cửa sổ tài liệu (*Single Document Interface - SDI*), chúng bao gồm mã trình để khai báo đối tượng của lớp quan sát. Hàm `Create` thừa hưởng từ lớp `CWnd` được gọi để tạo cửa sổ cho quan sát, và cửa sổ sẽ gắn vào cửa sổ khung và kích thước của chúng vừa bằng vùng làm việc của cửa sổ khung. Ta có thể không nhận ra chúng, nhưng khi quan sát bên trong cửa sổ khung SDI do *AppWizard* lập, cái ta nhìn thấy không phải vùng làm việc của cửa sổ khung, mà là cửa sổ quan

sát nằm trên chúng. Điều này cho thấy ta có thể tạo lập quan sát và gắn nó vào cửa sổ khung mà không cần *AppWizard*.

Một khi được lập, đối tượng quan sát kiểu lớp *CScrollView* tự động xử lý thông điệp *WM_VSCROLL* (và nếu muốn thì cả *WM_HSCROLL*) thay cho ứng dụng. Nó còn cung cấp các hàm thành phần hữu hiệu khác mà ứng dụng có thể gọi để khởi động hoặc xử lý quan sát. Điểm hình là hàm *CScrollView::SetScrollSizes*, có cái gì đó gần như *CWnd::SetScrollInfo*.

```
CScrollView::SetScrollSizes (int nMapMode, SIZE sizeTotal,
                             const SIZE& sizePage = sizeDefault, const SIZE& sizeLine = sizeDefault);
```

trong đó, *nMapMode* là chế độ GDI mà cửa sổ quan sát sẽ sử dụng. *SizeTotal* là cấu trúc *SIZE* hay đối tượng *CSize* cho biết kích thước của cửa sổ quan sát trượt được (kích thước của miền làm việc ảo) cho cả hai hướng x, y. *sizePage* xác định kích thước trang, hay giá trị mà cửa sổ quan sát sẽ trượt khi có các thông điệp *SB_PAGEUP*, *SB_PAGEDOWN*, *SB_PAGELEFT*, và *SB_PAGERIGHT*. *sizeLine* là số lượng mà vùng quan sát sẽ trượt khi có các thông điệp *SB_LINEUP*, *SB_LINEDOWN*, *SB_LINELEFT*, và *SB_LINERIGHT*. Giá trị mặc định của *sizePage* và *sizeLine* là 1/10 của *sizeTotal* và 1/10 của *sizePage*. Các lệnh:

```
CSize sizeTotal (400, 400);
CSize sizePage (100, 100);
CSize sizeLine (20, 5);
SetScrollSizes (MM_TEXT, sizeTotal, sizePage, sizeLine);
```

hoặc:

```
SetScrollSizes (MM_TEXT, CSize (400, 400), CSize (100, 100), CSize (20, 5));
```

khởi động miền quan sát trượt chứa khoảng làm việc là *400x400 pixel*, khi nhấn chuột trên thanh trượt thì chúng trượt *100 pixel* lên, xuống, trái hay phải, và trượt *20 pixel* ngang và năm *pixel* dọc khi ấn trên mũi tên thanh trượt. Thông thường, ta sẽ thấy tiện lợi khi gán *sizePage* giá trị độ cao, rộng của cửa sổ để gán kích thước của cửa sổ vào kích thước của con chạy thanh trượt. Nếu ta muốn miền quan sát chỉ có khả năng trượt đứng, ta sẽ nạp 0 vào thành phần x của đối tượng *CSize* và thanh trượt ngang sẽ biến khỏi cửa sổ. Tương tự, cho 0 vào thành phần y để vô hiệu hoá trượt đứng.

Tạo lập “quan sát trượt” và gắn nó cho cửa sổ khung như thế nào? Giả sử ta đang tự viết trình mà không sử dụng xương sống do *AppWizard* tạo ra, suy diễn lớp view từ *CScrollView* và khai báo đối tượng trong hàm *OnCreate* của cửa sổ khung. Sau đó gọi hàm *Create* của đối tượng view như sau:

```
pMyView->Create (NULL, NULL, WS_CHILD | WS_VISIBLE,
                 CRect (0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, NULL);
```

Tổ hợp của thuộc tính kiểu *WS_CHILD* và con trỏ *this* chỉ ra cửa sổ khung như cha của cửa sổ quan sát, view là con của cửa sổ khung.

AFX_IDW_PANE_FIRST gắn quan hệ bằng cách gắn cho cửa sổ quan sát một giá trị ID đặc biệt mà MFC dành cho việc gắn quan sát vào các cửa sổ khung SDI. Và thuộc tính WS_VISIBLE đảm bảo rằng cửa sổ quan sát sẽ thấy được mà không cần gọi ShowWindow như làm cho cửa sổ khung. Kích thước cửa sổ được truyền cho đối tượng CRect là không thích hợp, bởi vì cửa sổ quan sát tự động thay kích cỡ để cho vừa vùng làm việc của cha nó. Một khi cửa sổ quan sát được lập theo cách này, chúng trở thành vùng làm việc của cửa sổ ứng dụng. Nếu muốn, ta có thể chuyển các biến thành phần đã là một phần của lớp cửa sổ chính sang lớp quan sát và cũng hướng các thông điệp đến lớp quan sát.

Bây giờ ta sẽ nâng cấp GdiObj1 để tận dụng ưu việt của logic trượt xây dựng trong MFC.

2.9.6 Ứng dụng GdiObj2

GdiObj2 được mô tả trong chương trình 2.2 và hình 2.5. Điểm khác nhau lớn nhất và đầu tiên giữa GdiObj2 và GdiObj1 là tệp *header* chứa suy diễn cho lớp quan sát với tên CMyView. CMyView suy diễn trực tiếp từ CScrollView và bao gồm các biến m_cxChar và m_cyChar cũng như các hàm thành phần riêng ShowPenStyles, ShowPenWidths, và ShowBrushStyles. CMyView cũng chứa ánh xạ thông điệp để chuyển các thông điệp WM_CREATE và WM_SIZE đến các hàm thành phần OnCreate và OnSize. Điều ngạc nhiên là cả CMyView và CMainWindow không chứa đầu vào ánh xạ thông điệp cho thông điệp WM_PAINT, mặc dù CMyView có chứa hàm OnDraw nhận con trỏ của đối tượng DC làm tham số. Khi quan sát cài đặt của CMyView::OnDraw trong GDI OBJ2.CPP, ta sẽ thấy là nó giống hệt như hàm OnPaint của các chương trình khác.

Chương trình 2.2 GDI OBJ2

Tệp GDI OBJ2.H

```
class CMyApp : public CWinApp
{
public:
    BOOL InitInstance ();
};
//-----
class CMyView : public CScrollView
{
private:
    int m_cxChar;
    int m_cyChar;
    void ShowPenStyles (CDC*, int, int);
    void ShowPenWidths (CDC*, int, int);
```

```

void ShowBrushStyles (CDC* pDC, int, int);
public:
    void OnDraw (CDC* pDC);
protected:
    afx_msg int OnCreate (LPCREATESTRUCT);
    afx_msg void OnSize (UINT, int, int);
    DECLARE_MESSAGE_MAP ()
};
//-----
class CMainWindow : public CFrameWnd
{
public:
    CMainWindow ();
protected:
    afx_msg int OnCreate (LPCREATESTRUCT);
    DECLARE_MESSAGE_MAP ()
};
//-----
struct STYLES {
    int nStyle;
    char szStyleName[16];
};
//-----
#define VHEIGHT (m_cyChar * 50)
//-----

```

Tệp GDIOBJ2.CPP

```

// GdiObj2 thêm logic trượt vào GdiObj1 bằng cách thêm lớp View
// suy diễn từ lớp CScrollView của MFC.
#include <afxwin.h>
#include "GdiObj2.h"
CMyApp myApp;
// Các hàm thành phần của CMyApp
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}

```

```

}
// ánh xạ thông điệp và hàm thành phần của CMainWindow
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_CREATE ()
END_MESSAGE_MAP ()
//-----
CMainWindow::CMainWindow ()
{ Create (NULL, "GdiObj2"); }
//-----
int CMainWindow::OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    CMyView* pMyView = new CMyView ();
    pMyView->Create (NULL, NULL, WS_CHILD | WS_VISIBLE,
        CRect (0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, NULL);
    return 0;
}
//-----
// ánh xạ thông điệp và hàm thành phần của CMyView
BEGIN_MESSAGE_MAP (CMyView, CScrollView)
    ON_WM_CREATE ()
    ON_WM_SIZE ()
END_MESSAGE_MAP ()
//-----
int CMyView::OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    TEXTMETRIC tm;
    CDC* pDC = GetDC ();
    pDC->GetTextMetrics (&tm);
    m_cxChar = tm.tmAveCharWidth;
    m_cyChar = tm.tmHeight;
    ReleaseDC (pDC);
    return 0;
}
//-----
void CMyView::OnSize (UINT nType, int cx, int cy)
{
    SetScrollSizes (MM_TEXT, CSize (0, VHEIGHT), CSize (0, cy), CSize (0, m_cyChar));
}
//-----

```



```

void CMyView::OnDraw (CDC* pDC)
{
    ShowPenStyles (pDC, m_cxChar << 1, m_cyChar);
    ShowPenWidths (pDC, m_cxChar << 1, m_cyChar * 15);
    ShowBrushStyles (pDC, m_cxChar << 1, m_cyChar * 27);
}

//-----
void CMyView::ShowPenStyles (CDC* pDC, int x, int y)
{
    CPen* pPen;
    CPen* pOldPen;
    struct STYLES styles[] = {
        PS_SOLID,    "PS_SOLID",
        PS_DASH,     "PS_DASH",
        PS_DOT,      "PS_DOT",
        PS_DASHDOT,  "PS_DASHDOT",
        PS_DASHDOTDOT, "PS_DASHDOTDOT",
        PS_NULL,     "PS_NULL",
        PS_INSIDEFRAME, "PS_INSIDEFRAME"
    };
    CString string = "Kiểu bút vẽ (Pen Styles)";
    pDC->TextOut (x, y, string);
    int dy = (m_cyChar * 3) >> 1;
    pDC->SetTextColor (RGB (255, 0, 0));
    for (int i=0; i<7; i++) {
        y += dy;
        string = styles[i].szStyleName;
        pDC->TextOut (x + (m_cxChar << 1), y, string);
        pPen = new CPen (styles[i].nStyle, 1, RGB (255, 0, 0));
        pOldPen = pDC->SelectObject (pPen);
        pDC->MoveTo (x + (m_cxChar * 22), y + (m_cyChar >> 1));
        pDC->LineTo (x + (m_cxChar * 46), y + (m_cyChar >> 1));
        delete pDC->SelectObject (pOldPen);
    }
}

//-----
void CMyView::ShowPenWidths (CDC* pDC, int x, int y)
{
    CPen* pPen;
    CPen* pOldPen;

```

```

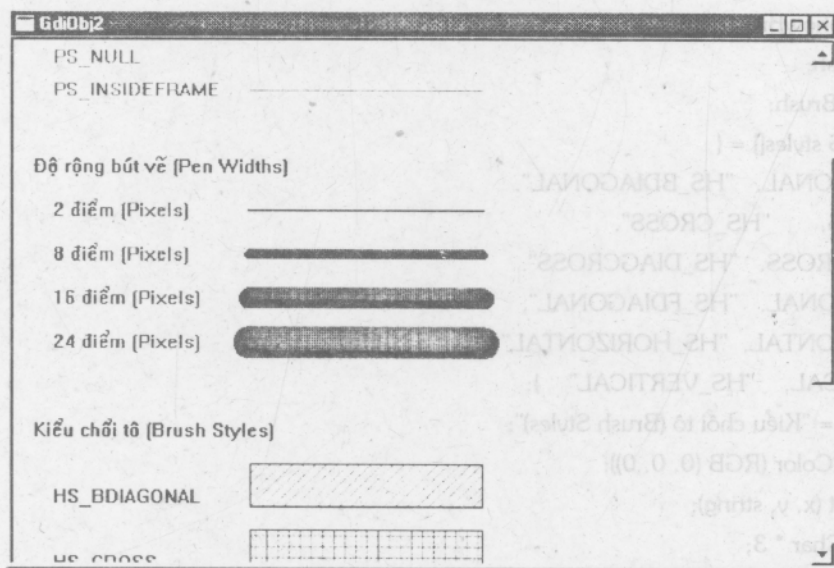
int nPenWidths[] = { 2, 8, 16, 24 };
CString string = "Độ rộng bút vẽ (Pen Widths)";
pDC->SetTextColor (RGB (0, 0, 0));
pDC->TextOut (x, y, string);
int dy = m_cyChar << 1;
pDC->SetTextColor (RGB (0, 0, 255));
for (int i=0; i<4; i++) {
    y += dy;
    string.Format ("%d điểm (Pixels)", nPenWidths[i]);
    pDC->TextOut (x + (m_cxChar << 1), y, string);
    pPen = new CPen (PS_SOLID, nPenWidths[i], RGB (0, 0, 255));
    pOldPen = pDC->SelectObject (pPen);
    pDC->MoveTo (x + (m_cxChar * 22), y + (m_cyChar >> 1));
    pDC->LineTo (x + (m_cxChar * 46), y + (m_cyChar >> 1));
    delete pDC->SelectObject (pOldPen);
}
}
//-----
void CMyView::ShowBrushStyles (CDC* pDC, int x, int y)
{
    CBrush* pBrush;
    CBrush* pOldBrush;
    struct STYLES styles[] = {
        HS_BDIAGONAL, "HS_BDIAGONAL",
        HS_CROSS, "HS_CROSS",
        HS_DIAGCROSS, "HS_DIAGCROSS",
        HS_FDIAGONAL, "HS_FDIAGONAL",
        HS_HORIZONTAL, "HS_HORIZONTAL",
        HS_VERTICAL, "HS_VERTICAL" };
    CString string = "Kiểu chổi tô (Brush Styles)";
    pDC->SetTextColor (RGB (0, 0, 0));
    pDC->TextOut (x, y, string);
    int dy = m_cyChar * 3;
    for (int i=0; i<6; i++) {
        y += dy;
        string = styles[i].szStyleName;
        pDC->TextOut (x + (m_cxChar << 1), y, string);
        CRect rect (x + (m_cxChar * 22), y - m_cyChar, x + (m_cxChar * 46), y + m_cyChar);
        pBrush = new CBrush (styles[i].nStyle, RGB (0, 255, 0));
        CPoint point (rect.left, rect.top);

```

```

pDC->LPtoDP (&point);
point.x &= 0x07;
point.y &= 0x07;
pDC->SetBrushOrg (point);
pOldBrush = pDC->SelectObject (pBrush);
pDC->Rectangle (rect);
delete pDC->SelectObject (pOldBrush);
}
y += dy;
string = "Solid";
pDC->TextOut (x + (m_cxChar << 1), y, string);
pBrush = new CBrush (RGB (0, 255, 0));
pOldBrush = pDC->SelectObject (pBrush);
pDC->Rectangle (x + (m_cxChar * 22), y - m_cyChar, x + (m_cxChar * 46), y + m_cyChar);
delete pDC->SelectObject (pOldBrush);
}
//-----

```



Hình 2.5 Kết quả của chương trình GDI OBJ2

Một trong các lợi thế mà các lớp quan sát của MFC cung cấp là trình tự các thông điệp WM_PAINT. Hàm OnPaint mặc định của lớp quan sát lập đối tượng DC của kiểu CPaintDC và sau đó gọi hàm ảo OnDraw với tham số là con trỏ đến đối tượng đó. Kết quả là toàn bộ công việc mà ta

phải làm để tự vẽ cửa sổ quan sát chỉ là nạp chồng OnDraw và cung cấp logic vẽ cần thiết. Điều này có thể có ít ý nghĩa ở đây. Nhưng trong ngữ cảnh của kiến trúc tài liệu/quan sát thì có ý nghĩa rất lớn vì nó cho phép khung ứng dụng gọi cùng mã lệnh để vẽ, in, và quan sát trước khi in.

Đối tượng quan sát được lập và cửa sổ quan sát được tạo cùng thời điểm của cửa sổ khung trong hàm CMainWindow::OnCreate:

```
CMyView* pMyView = new CMyView ();
pMyView->Create (NULL, NULL, WS_CHILD | WS_VISIBLE,
                CRect (0, 0, 0, 0), this, AFX_IDW_PANE_FIRST, NULL);
```

trong đó, pMyView lưu con trỏ đến đối tượng quan sát.

Việc gọi hàm Create được thực hiện tương tự như đã trình bày trong mục trước, giá trị AFX_IDW_PANE_FIRST của ID cửa sổ con trong hàm xác định duy nhất quan sát. Vì quan sát tự nó là cửa sổ, chúng cũng nhận thông điệp WM_CREATE, và mã khởi tạo m_cxChar và m_cyChar – mã trình là một phần của CMainWindow::OnCreate trong GdiObj1 – bây giờ được đặt trong CMyView::OnCreate.

Hàm OnSize của GdiObj2 gọi hàm SetScrollSizes của quan sát trượt để trả lời thông điệp WM_SIZE, cho biết cửa sổ khung sẽ thay đổi kích thước. CMyView::OnSize đơn giản hơn, chỉ chứa một dòng mã lệnh:

```
SetScrollSizes (MM_TEXT, CSize (0, VHEIGHT), CSize (0, cy), CSize (0, m_cyChar));
```

Không cần thiết nạp 0 cho dài trượt nếu cửa sổ đủ lớn để hiển thị toàn bộ vùng làm việc bởi vì đối tượng quan sát thực hiện chúng một cách tự động. Trách nhiệm của lập trình viên là quản lý vị trí đầu và cuối của thanh trượt bằng lệnh đơn giản trong hàm OnSize, và thậm chí việc đó cũng không cần thiết khi ta bằng lòng để kích thước trang cố định thay cho việc chốt chặt nó vào kích cỡ của cửa sổ. Ta vẫn phải gọi SetScrollSizes một lần, dĩ nhiên, nhưng có thể làm điều này từ CMyView::OnCreate hay cấu tử của CMyView. May thay, việc cung cấp hàm OnSize và thêm một đầu vào ánh xạ thông điệp chỉ là giá thấp phải trả để có được thanh trượt tự động thích nghi với sự thay đổi kích cỡ cửa sổ.

E rằng ta sẽ nhầm lẫn khi nghĩ rằng CScrollView là hoàn hảo và cho rằng ta sẽ không bao giờ phải viết mã trình cho thanh trượt nữa, ở đây có hai vấn đề phải tính tới. Thứ nhất, CScrollView là quan trọng cho ứng dụng đặc biệt này - cửa sổ khung được sử dụng để cung cấp quan sát vùng làm việc ảo - nhưng có cách khác sử dụng cho thanh trượt mà không ánh xạ dễ như những khả năng do CScrollView cung cấp. Hãy nhớ lại ứng dụng đa phương tiện đã đề cập trên đây. Ta cài đặt thanh trượt để cho phép người sử dụng thay đổi dễ dàng các rãnh trên đĩa CD trong ổ CD-ROM. Đối tượng CScrollView chẳng giúp được mấy, vì ở đây không có cái gì để mà nhìn. Vấn đề thứ hai là tốc độ. CScrollView làm việc rất tốt cho GdiObj2 vì vùng làm việc ảo không quá lớn so với cửa sổ.

Nhưng điều gì xảy ra nếu vùng làm việc lớn 30.000 *pixel* và cửa sổ chỉ 100 *pixel*? `CScrollView` vẫn còn làm việc được, nhưng chậm bởi vì hàm `OnDraw` phải cố gắng vẽ cả 30.000 dòng đầu mỗi khi sự kiện trượt xảy ra. Để giải quyết vấn đề này ta cần sửa đổi mã của `OnDraw` để lấy được vị trí trượt nhờ gọi hàm `CScrollView::GetScrollPosition` và giới hạn việc vẽ chỉ trong vùng ta nhìn thấy. Dĩ nhiên phải viết thêm mã trình để làm hàm `OnDraw` trở nên thông minh hơn.

Có một vấn đề còn lỏng lẻo cần phải đề cập đến trong cài đặt này. Ngay sau khi `GdiObj2` khởi động, `CMainWindow::OnCreate` tạo đối tượng quan sát với dòng lệnh sau:

```
CMyView* pMyView = new CMyView ();
```

Vì đối tượng được lập bằng hàm `new` cho nên chúng vẫn còn ở lại trong bộ nhớ sau khi `OnCreate` kết thúc và sự thật chúng không mất đi cho đến khi bị xoá bằng lệnh `delete`. Nhưng ta không tìm thấy lệnh này trong `GdiObj2`. Đối tượng của cửa sổ khung cũng không thấy bị xoá ở đâu; không có lệnh:

```
delete m_pMainWnd;
```

đối xứng với lệnh:

```
m_pMainWnd = new CMainWindow ();
```

trong `CMyApp::InitInstance`. Nhìn từ bên ngoài, hình như là có vấn đề. Trước hết, lập trình viên C++ biết rằng mỗi lệnh `new` phải có lệnh tương ứng `delete` nếu không thì đối tượng sẽ ở lại trong bộ nhớ. Thế thì cái gì xảy ra ở trong thí dụ này?

Đúng như mong đợi thư viện lớp sẽ thực hiện xoá các đối tượng này giúp ta. Chính xác hơn, các đối tượng tự xoá. Chìa khoá của vấn đề này là thông điệp cuối cùng mà cửa sổ nhận được trước khi bị bãi bỏ là `WM_NCDESTROY`. Nếu quan sát trong mã nguồn của `CWnd::OnNcDestroy`, ta thấy rằng chúng gọi hàm ảo với tên `PostNcDestroy`. Cả hai lớp `CFrameWnd` và `CView` nạp chồng hàm `PostNcDestroy` và có thực hiện lệnh:

```
delete this;
```

để tự xoá. Vì vậy, khi cửa sổ khung hay cửa sổ quan sát bị xoá, đối tượng liên quan với cửa sổ cũng sẽ bị xoá. Cửa sổ khung của `GdiObj2` được xoá khi người sử dụng nhấn vào phím đóng trên thanh tiêu đề hay chọn `Close` từ thực đơn hệ thống. Và khi bất kỳ cửa sổ nào bị xoá, cửa sổ con của nó cũng tự động bị xoá - Windows thực hiện việc này. Vậy, việc đóng lại ứng dụng `GdiObj2` sẽ khởi đầu một dãy sự kiện để huỷ bỏ cửa sổ khung, cửa sổ quan sát, và các đối tượng C++ liên quan đến chúng. Mặt khác, không có ý nghĩa gì khi cài đặt lấy hàm `PostNcDestroy` của `CWnd` mà lại không thực hiện việc xoá đối tượng cửa sổ liên quan. Vì vậy, nếu suy diễn lớp cửa sổ riêng trực tiếp từ `CWnd`, ta sẽ phải nạp chồng `PostNcDestroy` trong lớp suy diễn và thực hiện lệnh `delete this`. Nếu không, đối tượng `CWnd` sẽ không được xoá sạch. Ta sẽ xem xét một thí dụ lớp cửa sổ suy diễn từ `CWnd` trong chương sau.

CHƯƠNG 3

XỬ LÝ ĐẦU VÀO TỪ CHUỘT

Phần rất lớn các ứng dụng tin học trên thị trường hiện nay chấp nhận đầu vào từ chuột và bàn phím. Windows 95/98 điều khiển phần lớn loại đầu vào này một cách tự động, thực thi các công việc như chuyển đổi nhấn chuột để kích hoạt các thực đơn cho các thông điệp lệnh để cho ứng dụng biết là thực đơn nào đã được chọn. Như mọi đầu vào ảo của Win95/98, đầu vào chuột và bàn phím đi đến ứng dụng dưới dạng thông điệp. Khi phím chuột nhấn hay nhả, khi dịch chuyển chuột, hay khi phím đang bị nhấn hay nhả, thì chương trình điều khiển chuột hay phím sẽ đặt sự kiện vào hàng đợi hệ thống đặc biệt và được gọi là hàng đợi vào thô (*raw input queue*). Khi chương trình thực hiện, chúng tách thông báo từ hàng đợi thô, tạo thành thông điệp chuột hay phím, truyền thông điệp này cho hàng đợi thông điệp ứng dụng tương ứng. Sau đó thông điệp chuột được xử lý tương tự như mọi thông điệp khác.

Mô hình vào vừa mô tả trên đây khác với mô hình vào của Windows 3.1, trong đó các thông điệp chuột, phím được lưu trong cùng hàng đợi, dùng chung cho tất cả các ứng dụng đang chạy. Các ứng dụng 16 bit vẫn chia sẻ một hàng đợi trong Windows 95/98, nhưng ứng dụng 32 bit có các hàng đợi vào riêng rẽ để tránh ứng dụng nào đó tác động ngược lại ứng dụng khác khi trả lời sai thông điệp chuột và bàn phím. Mỗi ứng dụng không bị giới hạn bởi một hàng đợi thông điệp; trong ứng dụng đa luồng (*multithread*) Win32, mỗi luồng chạy chương trình có thể lấy và chia thông điệp thì đều có một hàng đợi vào riêng. Lợi ích này khó đánh giá hết được trừ khi ta đã thấy ứng dụng 16 bit xử lý thông điệp chậm như thế nào.

May thay, cách hệ điều hành điều khiển các thông điệp chuột và phím là trong suốt đối với ứng dụng. Khi `::SendMessage` được gọi từ vòng lặp thông điệp và có thông điệp chuột, bàn phím đang chờ thì thông điệp được lấy từ hàng đợi và cấp phát cho ứng dụng, không để ý đến nó được lưu ở đâu và như thế nào.

Học cách nhận thông tin từ đầu vào chuột và bàn phím trong Windows chính là học các thông điệp sự kiện của chuột và bàn phím. Trong chương này ta sẽ xem xét các thông điệp chuột; chương sau sẽ xem xét xử lý đầu vào bàn phím. Ta sẽ khảo sát hai chương trình: trò chơi cờ ca-rô biểu diễn cách phản ứng khi nhấn phím chuột và chương trình vẽ, biểu diễn việc chặn đầu vào chuột và tìm hiểu xem các thông điệp của chuột ngoài vùng làm việc được xử lý như thế nào.

3.1 NHẬN THÔNG ĐIỆP CHUỘT TRONG VÙNG LÀM VIỆC

Windows 95/98 sử dụng 20 kiểu thông điệp để thông báo sự kiện chuột. Các thông điệp này rơi

vào hai lĩnh vực: thông điệp chuột trên vùng làm việc, chúng cung cấp sự kiện xảy ra trong vùng làm việc của cửa sổ, và thông điệp chuột ngoài vùng làm việc cho sự kiện ngoài vùng làm việc của cửa sổ, hoặc chuyển dịch của chuột. Hầu như trong chương trình ta bỏ qua các sự kiện ngoài vùng làm việc và để Windows quản lý chúng.

Windows thông báo sự kiện chuột trong vùng làm việc của cửa sổ bằng các thông điệp như trong bảng 3.1. Dĩ nhiên, các thông điệp mà ứng dụng nhận được phụ thuộc vào loại chuột mà ta cài. Ứng dụng sẽ không nhận thông điệp WM_MBUTTON nếu chuột chỉ có hai phím. Và nó không nhận thông điệp WM_RBUTTON nếu chuột chỉ có một phím. Phần lớn chuột trong Windows chỉ có hai phím, cho nên chắc chắn rằng phím phải chuột là có thực. Tuy nhiên, để cho chắc chắn (hay muốn xem chuột có phím thứ ba không), ta có thể gọi hàm `::GetSystemMetrics` như sau:

```
::GetSystemMetrics(SM_CMOUSEBUTTONS);
```

Giá trị cho lại là tổng số phím của chuột, hay cho lại 0 khi chưa cài chuột.

Bảng 3.1 Các thông điệp chuột trong vùng làm việc

<i>Thông điệp</i>	<i>Hàm điều khiển</i>	<i>Gửi khi</i>
WM_LBUTTONDOWN	OnLButtonDown	Phím trái chuột bị nhấn
WM_LBUTTONUP	OnLButtonUp	Phím trái chuột được nhả
WM_LBUTTONDOWNBLCK	OnLButtonDownBlk	Phím trái chuột bị nhấn đúp
WM_MBUTTONDOWN	OnMButtonDown	Phím giữa chuột bị nhấn
WM_MBUTTONUP	OnMButtonUp	Phím giữa chuột được nhả
WM_MBUTTONDBLCLK	OnMButtonDownBlk	Phím giữa chuột bị nhấn đúp
WM_RBUTTONDOWN	OnRButtonDown	Phím phải chuột bị nhấn
WM_RBUTTONUP	OnRButtonUp	Phím phải chuột được nhả
WM_RBUTTONDOWNBLCK	OnRButtonDownBlk	Phím phải chuột bị nhấn đúp
WM_MOUSEMOVE	OnMouseMove	Chuột trượt trên vùng làm việc của cửa sổ

Các thông điệp WM_xBUTTONDOWN và WM_xBUTTONUP thông báo phím nhấn hay nhả. Thông điệp tiếp sau WM_LBUTTONDOWN thường là WM_LBUTTONUP, nhưng không thể đếm được chúng. Nếu ta nhấn phím trái chuột tại vùng làm việc của cửa sổ rồi di con chạy ra ngoài cửa sổ trước khi nhả phím, cửa sổ sẽ nhận thông điệp WM_LBUTTONDOWN nhưng không nhận được thông điệp WM_LBUTTONUP. Rất nhiều chương trình chỉ phản ứng với thông điệp nhấn phím và bỏ qua thông điệp nhả phím trong trường hợp không sử dụng cập thông điệp. Trong trường hợp cần cả hai, chương trình có thể “bắt” chuột khi nhận thông điệp phím nhấn và nhả khi thông điệp nhả phím tới. Ở khoảng giữa, toàn bộ thông điệp thậm chí kể cả các thông điệp bên ngoài cửa sổ đều được hướng tới cửa sổ đang thực hiện “bắt” chuột. Điều này đảm bảo rằng thông điệp nhả phím sẽ được nhận mà không quan tâm đến con chạy ở đâu và khi nào nhả. Khi nhấn đúp vào phím

(khoảng thời gian của nhấn đúp được xác định bởi *Control Panel* của Windows hay hàm `::SetDoubleClickTime`), thông điệp phím nhấn thứ hai được thay thế bởi thông điệp `WM_XBUTTONDOWNBLCLK`. Nhưng điều này chỉ xảy ra khi cửa sổ được lập từ lớp cửa sổ có định nghĩa kiểu `CS_DBLCLKS`. Lớp cửa sổ mặc định mà MFC đăng ký cho cửa sổ khung đã có kiểu `CS_DBLCLKS`, vậy ta chỉ cần quan tâm đến kiểu cửa sổ khi ta đăng ký lớp riêng bằng hàm `AfxRegisterWndClass`. Với cửa sổ `CS_DBLCLKS`, thì hai nhấn nhanh trên phím trái chuột trên vùng làm việc sẽ phát sinh trình tự các thông điệp sau đây:

```
WM_LBUTTONDOWN
WM_LBUTTONUP
WM_LBUTTONDOWNBLCLK
WM_LBUTTONUP
```

Cũng tác động như vậy, nhưng nếu cửa sổ không đăng ký để trả lại thông điệp nhấn đúp thì có các thông điệp:

```
WM_LBUTTONDOWN
WM_LBUTTONUP
WM_LBUTTONDOWN
WM_LBUTTONUP
```

Việc xử lý các thông điệp này phụ thuộc vào lập trình viên nếu họ thấy cần. Việc nhấn hay nhấn đúp cùng phím chuột trên cùng đối tượng thường hướng tới các công việc khác nhau. Thông điệp nhấn đúp luôn đi sau thông điệp nhấn đơn, sao cho các tác động của chúng không dễ tách rời. Giải pháp thông thường là cái nhấn thứ nhất là chọn đối tượng, và cái nhấn thứ hai thực hiện tác động trên chúng. Thí dụ, khi ta nhấn đúp phím phải trên *Folder* của Windows 95/98 *Explorer*, nhấn thứ nhất chọn *Folder*, và nhấn thứ hai mở *Folder* và hiển thị nội dung của chúng.

Các thông điệp `WM_MOUSEMOVE` cho biết con chạy dịch chuyển trong vùng làm việc của cửa sổ. Vì chuột chuyển động, cửa sổ dưới con chạy nhận thông điệp và cho biết vị trí của chuột. Có nhiều chương trình bỏ qua các thông điệp này, tuy nhiên các chương trình vẽ phải sử dụng chúng để tiếp tục cập nhật các vị trí con chạy trên màn hình như là chuột đang chạy. Windows 95/98 sử dụng các thông điệp `WM_MOUSEMOVE` để di chuyển chọn nhóm khi chuột chuyển động trong khi giữ phím trái. Các thông điệp này được sử dụng, nhưng quan trọng là phải xử lý chúng nhanh nhất có thể để tránh chuột chuyển động không trơn tru. Để cho chương trình ứng dụng không bị các thông điệp `WM_MOUSEMOVE` lấn át, Windows không gửi thông điệp cho mỗi *pixel* mà con chạy đi theo trừ khi phản ứng có thể đáp ứng yêu cầu. Khi chuột chạy nhanh thì chương trình hầu như chỉ nhận vài thông điệp cho biết các điểm trên đường chuột đi qua. Chỉ khi chuột đi rất chậm thì vị trí của mọi điểm mới được thông báo lại.

Trong chương trình MFC, các thông điệp `WM_LBUTTONDOWN` được hàm `OnLButtonDown` xử lý, còn các thông điệp `WM_LBUTTONUP` do hàm `OnLButtonUp` xử lý, v.v... Tất cả các hàm

xử lý thông điệp chuột trong vùng làm việc đều có mẫu hàm như sau:

```
void OnMsgName (UINT nFlags, CPoint point);
```

trong đó, *nFlags* chứa dãy các cờ bit cho biết trạng thái của các phím chuột và các phím bàn phím *Shift* và *Ctrl* vào thời điểm sự kiện xảy ra, và vị trí của con chuột. Trong mọi trường hợp, các vị trí của chuột là tọa độ tương đối so với góc trên, trái của vùng làm việc. Đơn vị đo luôn là *pixel*, vậy thông điệp *WM_LBUTTONDOWN* cùng với *point.x = 32* và *point.y = 64* có nghĩa là phím trái chuột được nhấn tại điểm 32x64 kể từ góc trên, trái của vùng làm việc.

Ta có thể xem tham số *nFlags* để biết có phím đặc biệt hay phím bàn phím đang bị nhấn nhờ mặt nạ bit trong bảng 3.2. Lệnh:

```
nFlags & MK_LBUTTON
```

là *TRUE* khi một và chỉ một phím trái chuột bị nhấn, trong khi:

```
nFlags & MK_CONTROL
```

là *TRUE* nếu phím *Ctrl* bị giữ xuống khi có sự kiện xảy ra. Một vài chương trình trả lời khác nhau cho các sự kiện chuột nếu các phím *Shift* hay *Ctrl* được nhấn xuống. Thí dụ, bằng việc kiểm tra bit *MK_CONTROL* trong cờ *nFlags* đi cùng thông điệp *WM_MOUSEMOVE*, chương trình vẽ yêu cầu người sử dụng vẽ đường thẳng nếu phím *Ctrl* bị nhấn như là chuột chuyển động.

Bảng 3.2 Các cờ bit để kiểm tra tham số *nFlags*

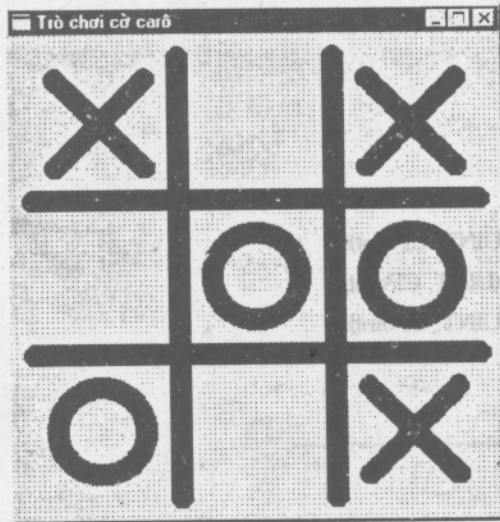
Mặt nạ	Ý nghĩa
<i>MK_LBUTTON</i>	Phím trái chuột bị nhấn
<i>MK_MBUTTON</i>	Phím giữa chuột bị nhấn
<i>MK_RBUTTON</i>	Phím phải chuột bị nhấn
<i>MK_CONTROL</i>	Phím <i>Ctrl</i> bị nhấn
<i>MK_SHIFT</i>	Phím <i>Shift</i> bị nhấn

Nếu cửa sổ chứa cửa sổ con như phím bấm hay thanh trượt, chúng sẽ không nhận thông điệp chuột khi nhấn chuột (hay con chuột chuyển dịch) trên cửa sổ con. Tuân thủ qui luật rằng các thông điệp chuột đến các cửa sổ ở dưới con chuột, các thông điệp sẽ hướng về cửa sổ con. Chính vì vậy, ứng dụng mẫu *GdiObj2* có thể trả lời chuột nhấn trên thanh trượt mà không cần xử lý bất kỳ thông điệp nào. Việc nhấn thanh trượt đứng của cửa sổ sẽ gửi thông điệp chuột đến thủ tục cửa sổ của thanh trượt, và thanh trượt đến lượt mình sẽ gửi thông điệp *WM_VSCROLL* đến cửa sổ quản lý nó. Đó là thủ tục cửa sổ của thanh trượt, được dấu kín sâu trong tệp *USER.EXE* của Windows, xử lý các thông điệp chuột. Thông điệp chuột vùng làm việc có nghĩa là sự kiện chuột xảy ra trong vùng làm việc của cửa sổ của ta, chứ không phải ở nơi đâu khác.

3.2 ỨNG DỤNG TicTac

Để thấy được việc xử lý các thông điệp chuột để như thế nào, hãy làm ứng dụng mẫu lấy đầu vào từ chuột (xem chương trình 3.1 và hình 3.1).

TicTac là chương trình mô phỏng trò chơi cờ ca-rô, phản ứng với ba loại sự kiện chuột trên vùng làm việc: nhấn phím trái, phím phải và nhấn đúp phím trái. Nhấn phím trái trên ô vuông rỗng sẽ đặt dấu X vào đó. Nhấn phím phải sẽ đặt O vào ô rỗng. Nhấn đúp phím trái trên kẻ đen mỏng sẽ xóa lưới chơi và khởi động lại trò chơi. Sau mỗi lần đặt X hay O, chương trình kiểm tra xem có ai thắng không hay là hoà. Hoà sẽ được tuyên bố khi tất cả chín ô vuông đã được làm đầy và không ai điền được ba ô theo đường ngang, dọc hoặc chéo. Ngoài ra chương trình này cũng mô tả xử lý thông điệp chuột, giới thiệu cách sử dụng các hàm MFC như `CWnd::MessageBox`, để hiển thị cửa sổ hộp thông báo, và hàm `CRect::PtInRect` cho biết nhanh chóng là điểm có nằm trong chữ nhật `CRect` hay không.



Hình 3.1 Kết quả thực hiện chương trình TicTac

Chương trình 3.1 TICTAC

Tập TICTAC.H

```
class CMyApp : public CWinApp
{
public:
    BOOL InitInstance ();
};
```

```
//-----
class CMainWindow : public CWnd
{
private:
    CRect m_rect[9];    // Các chữ nhật biểu diễn lưới hình vuông
    int m_nNextChar;    // Ký tự tiếp theo (EX cho X hay OH cho O)
    int m_nGameGrid[9]; // Lưới bàn cờ (0 = ô rỗng)
    int GetRectID (CPoint);
    void DrawBoard (CDC*);
    void DrawX (CDC*, int);
    void DrawO (CDC*, int);
    void ResetGame ();
    void CheckForGameOver ();
    int IsWinner ();
    BOOL IsDraw ();
public:
    CMainWindow ();
protected:
    void PostNcDestroy ();
    afx_msg void OnPaint ();
    afx_msg void OnLButtonDown (UINT, CPoint);
    afx_msg void OnLButtonDbcClick (UINT, CPoint);
    afx_msg void OnRButtonDown (UINT, CPoint);
    DECLARE_MESSAGE_MAP ()
};
//-----
#define EX 1
#define OH 2
//-----
```

Tệp TICTAC.CPP

```
// TicTac biểu diễn quản lý chuột trong chương trình MFC
#include <afxwin.h>
#include "tictac.h"
CMyApp myApp;
// Hàm thành phần của lớp CMyApp
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
```

```

m_pMainWnd->ShowWindow (m_nCmdShow);
m_pMainWnd->UpdateWindow ();
return TRUE;
}

//ánh xạ thông điệp và Hàm thành phần của lớp CMainWindow
BEGIN_MESSAGE_MAP (CMainWindow, CWnd)
ON_WM_PAINT ()
ON_WM_LBUTTONDOWN ()
ON_WM_LBUTTONDBLCLK ()
ON_WM_RBUTTONDOWN ()
END_MESSAGE_MAP ()

//-----
CMainWindow::CMainWindow ()
{
    int nCoords[9][4] = {
        16, 16, 112, 112,
        128, 16, 224, 112,
        240, 16, 336, 112,
        16, 128, 112, 224,
        128, 128, 224, 224,
        240, 128, 336, 224,
        16, 240, 112, 336,
        128, 240, 224, 336,
        240, 240, 336, 336
    };
    m_nNextChar = EX;
    for (int i=0; i<9; i++) {
        m_nGameGrid[i] = 0;
        m_rect[i].SetRect (nCoords[i][0], nCoords[i][1], nCoords[i][2], nCoords[i][3]);
    }
    CString myClass = AfxRegisterWndClass (CS_DBLCLKS, myApp.LoadStandardCursor (IDC_ARROW),
        (HBRUSH) ::GetStockObject (LTGRAY_BRUSH), myApp.LoadStandardIcon (IDI_APPLICATION));
    CreateEx (WS_EX_DLGMODALFRAME, myClass, "Trò chơi cờ caro", WS_OVERLAPPED |
        WS_SYSMENU | WS_CAPTION | WS_MINIMIZEBOX, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, NULL);
    CRect rect (0, 0, 352, 352);
    CalcWindowRect (&rect);
    SetWindowPos (NULL, 0, 0, rect.Width (), rect.Height (),
        SWP_NOZORDER | SWP_NOMOVE | SWP_NOREDRAW);
}

```

```

}
//-----
void CMainWindow::PostNcDestroy ()
{
    delete this;
}
//-----
void CMainWindow::OnPaint ()
{
    CPaintDC dc (this);
    DrawBoard (&dc);
}
//-----
void CMainWindow::OnLButtonDown (UINT nFlags, CPoint point)
{
    if (m_nNextChar != EX) return;
    int nPos = GetRectID (point);
    if (nPos == -1) return;
    if (m_nGameGrid[nPos] != 0) return;
    m_nNextChar = OH;
    m_nGameGrid[nPos] = EX;
    CClientDC dc (this);
    DrawX (&dc, nPos);
    CheckForGameOver ();
}
//-----
void CMainWindow::OnRButtonDown (UINT nFlags, CPoint point)
{
    if (m_nNextChar != OH) return;
    int nPos = GetRectID (point);
    if (nPos == -1) return;
    if (m_nGameGrid[nPos] != 0) return;
    m_nNextChar = EX;
    m_nGameGrid[nPos] = OH;
    CClientDC dc (this);
    DrawO (&dc, nPos);
    CheckForGameOver ();
}
//-----

```

```

void CMainWindow::OnLButtonDbcClk (UINT nFlags, CPoint point)
{
    CClientDC dc (this);
    if (dc.GetPixel (point) == RGB (0, 0, 0))    ResetGame ();
}
//-----
int CMainWindow::GetRectID (CPoint point)
{
    for (int i=0; i<9; i++)
        if (m_rect[i].PtInRect (point))    return i;
    return -1;
}
//-----
void CMainWindow::DrawBoard (CDC* pDC)
{
    CPen pen (PS_SOLID, 16, RGB (0, 0, 0));
    CPen* pOldPen = pDC->SelectObject (&pen);
    pDC->MoveTo (120, 16);
    pDC->LineTo (120, 336);
    pDC->MoveTo (232, 16);
    pDC->LineTo (232, 336);
    pDC->MoveTo (16, 120);
    pDC->LineTo (336, 120);
    pDC->MoveTo (16, 232);
    pDC->LineTo (336, 232);
    for (int i=0; i<9; i++) {
        if (m_nGameGrid[i] == EX)    DrawX (pDC, i);
        else if (m_nGameGrid[i] == OH)    DrawO (pDC, i);
    }
    pDC->SelectObject (pOldPen);
}
//-----
void CMainWindow::DrawX (CDC* pDC, int nPos)
{
    CPen pen (PS_SOLID, 16, RGB (255, 0, 0));
    CPen* pOldPen = pDC->SelectObject (&pen);
    pDC->MoveTo (m_rect[nPos].left + 16, m_rect[nPos].top + 16);
    pDC->LineTo (m_rect[nPos].right - 16, m_rect[nPos].bottom - 16);
    pDC->MoveTo (m_rect[nPos].left + 16, m_rect[nPos].bottom - 16);

```

```

    pDC->LineTo (m_rect[nPos].right - 16, m_rect[nPos].top + 16);
    pDC->SelectObject (pOldPen);
}
//-----
void CMainWindow::DrawO (CDC* pDC, int nPos)
{
    CPen pen (PS_SOLID, 16, RGB (0, 0, 255));
    CPen* pOldPen = pDC->SelectObject (&pen);
    pDC->SelectStockObject (NULL_BRUSH);
    pDC->Ellipse (m_rect[nPos].left + 16, m_rect[nPos].top + 16,
                  m_rect[nPos].right - 16, m_rect[nPos].bottom - 16);
    pDC->SelectObject (pOldPen);
}
//-----
void CMainWindow::CheckForGameOver ()
{
    int nWinner;
    if (nWinner = IsWinner ()) {
        CString string = (nWinner == EX) ? "X thắng!" : "O thắng!";
        MessageBox(string, "Kết thúc", MB_ICONEXCLAMATION | MB_OK);
        ResetGame ();
    }
    else if (IsDraw ()) {
        MessageBox ("Đây là một bản vẽ!", "Kết thúc", MB_ICONEXCLAMATION | MB_OK);
        ResetGame ();
    }
}
//-----
int CMainWindow::IsWinner ()
{
    static int nPattern[8][3] = {
        0, 1, 2,
        3, 4, 5,
        6, 7, 8,
        0, 3, 6,
        1, 4, 7,
        2, 5, 8,
        0, 4, 8,
        2, 4, 6 };

```

```

for (int i=0; i<8; i++) {
    if ((m_nGameGrid[nPattern[i][0]] == EX) &&
        (m_nGameGrid[nPattern[i][1]] == EX) && (m_nGameGrid[nPattern[i][2]] == EX))
        return EX;
    if ((m_nGameGrid[nPattern[i][0]] == OH) &&
        (m_nGameGrid[nPattern[i][1]] == OH) && (m_nGameGrid[nPattern[i][2]] == OH))
        return OH;
}
return 0;
}
//-----
BOOL CMainWindow::IsDraw ()
{
    for (int i=0; i<9; i++)
        if (m_nGameGrid[i] == 0) return FALSE;
    return TRUE;
}
//-----
void CMainWindow::ResetGame ()
{
    m_nNextChar = EX;
    for (int i=0; i<9; i++)
        m_nGameGrid[i] = 0;
    InvalidateRect (NULL);
}
//-----

```

Bước thứ nhất khi xử lý thông điệp chuột, là thêm các thông điệp mà ta quan tâm vào ánh xạ thông điệp. Lớp CMainWindow trong tệp TICTAC.H chứa các khai báo hàm sau đây:

```

afx_msg void OnLButtonDown (UINT, CPoint);
afx_msg void OnLButtonDblClk (UINT, CPoint);
afx_msg void OnRButtonDown (UINT, CPoint);

```

Tương ứng với ánh xạ thông điệp CMainWindow trong TICTAC.CPP gồm các đầu vào:

```

ON_WM_LBUTTONDOWN ()
ON_WM_LBUTTONDBLCLK ()
ON_WM_RBUTTONDOWN ()

```

Các đầu vào ánh xạ thông điệp này dẫn các thông điệp chuột gửi từ cửa sổ ứng dụng đến hàm quản lý thông điệp tương ứng của CMainWindow. Và khi các thông điệp đến thì trò chơi bắt đầu.

Hàm `OnLButtonDown` của `TicTac` xử lý các nhấn trên phím trái chuột trong vùng làm việc của cửa sổ. Giả sử sau khi kiểm tra biến thành phần `m_nNextChar` của lớp `CMainWindow` để xác định đến lượt X chứ không phải O. `OnLButtonDown` gọi hàm thành phần riêng `GetRectID` để xác định xem có nhấn chuột vào một trong chín ô vuông của lưới trò chơi hay không. Cấu tử `CMainWindow` khởi động mảng các đối tượng `CRect` có tên `m_rect` với tọa độ vuông. `GetRectID` sử dụng vòng `for` để xác định xem vị trí của con chuột có nằm trong hình vuông nào đó hay không.

```
for (int i=0; i<9; i++)
{
    if (m_rect[i].PtInRect (point)) return i;
}
return -1;
```

`CRect::PtInRect` cho lại `TRUE` nếu điểm xác định bởi đối tượng `CPoint` nằm trong chữ nhật biểu diễn bằng đối tượng `CRect`, hay `FALSE` khi không phải như vậy. Nếu `PtInRect` cho lại `TRUE` với bất kỳ chữ nhật nào trong mảng `m_rect` thì `GetRectID` cho lại ID của chữ nhật. ID là số nguyên từ 0 đến 8, trong đó 0 biểu diễn hình vuông tại góc trên trái của lưới, 1 là hình vuông phía phải của nó, 2 là hình vuông góc trên phải, 3 là hình vuông trái nhất của hàng thứ hai, v.v.. Mỗi hình vuông có một thành phần tương ứng trong mảng `m_nGameGrid`, chúng lưu giữ giá trị khởi đầu là 0, có nghĩa là toàn bộ hình vuông rỗng. Nếu không có lời gọi hàm `PtInRect` nào cho lại giá trị `TRUE` thì hàm `GetRectID` cho lại giá trị -1 để chỉ ra rằng chuột được nhấn bên ngoài các hình vuông, là nguyên nhân làm hàm `OnLButtonDown` bỏ qua nhấn chuột. Tuy nhiên, nếu `GetRectID` cho lại ID hợp lệ và hình vuông tương ứng là rỗng (có giá trị 0) thì hàm `OnLButtonDown` ghi lại X vào mảng `m_nGameGrid` rồi gọi `CMainWindow::DrawX` để vẽ X trong hình vuông. `DrawX` tạo bút đỏ 16 pixel và vẽ hai đường chéo nhau nhờ các hàm `MoveTo` và `LineTo`.

`OnRButtonDown` làm việc rất giống `OnLButtonDown`, chỉ khác là chúng vẽ O trong hình vuông rỗng thay cho vẽ X. Chương trình thực hiện việc vẽ là `CMainWindow::DrawO`. Nên vẽ O bằng hàm `CDC::Ellipse`, `DrawO` chọn chổi tô `NULL` cho ngữ cảnh thiết bị:

```
pDC->SelectStockObject (NULL_BRUSH);
```

Làm như vậy tránh được việc tô màu bên trong O. (Cách khác, ta có thể chọn đối tượng có sẵn `LTGRAY_BRUSH` cho ngữ cảnh thiết bị để bên trong O có cùng màu nền của cửa sổ, nhưng sử dụng chổi tô `NULL` sẽ nhanh hơn). Sau đó O sẽ được vẽ chỉ bằng một lệnh sau:

```
pDC->Ellipse (m_rect[nPos].left + 16,
              m_rect[nPos].top + 16,
              m_rect[nPos].right - 16,
              m_rect[nPos].bottom - 16);
```

Tọa độ chữ nhật lưu trong thành phần tương ứng của mảng `m_rect` được có lại 16 pixel cho cả bốn hướng để có hộp bao tròn góc. Kết quả là O nằm giữa hình vuông nơi nó được vẽ. Nhấn đúp

lên lưới ngăn cách các hình vuông sẽ xoá X và O và bắt đầu lần chơi mới. Vì ta vẫn chưa khảo sát thực đơn hay phím điều khiển, cho nên giao diện sử dụng này được dùng mặc dù nó là rất tồi. Nhấn đúp phím trái chuột được xử lý nhờ hàm `CMainWindow::OnLButtonDbClick`, chúng gồm các lệnh đơn giản sau:

```
CClientDC dc (this);
if (dc.GetPixel (point) == RGB (0, 0, 0))
    ResetGame ();
```

Để xác định xem nhấn đúp có xảy ra trên đường đen dày của lưới hay không, `OnLButtonDbClick` gọi `CDC::GetPixel` để lấy màu của *pixel* dưới con chạy và so sánh với màu đen (`RGB (0, 0, 0)`). Nếu đúng thì gọi hàm `ResetGame`. Nếu không, lời gọi `OnLButtonDbClick` trở về, nhấn đúp được bỏ qua. Sử dụng màu của *pixel* dưới con chạy là kỹ thuật hiệu quả để xác định nhấn phím trên những vùng có hình không đều, nhưng chú ý rằng không được sử dụng các màu thứ cấp mà một số màn hình phải thực hiện dither. Các màu thuần túy đen (`RGB (0, 0, 0)`) và trắng (`RGB (255, 255, 255)`) có trên mọi PC mà Windows chạy được.

Để phù hợp với các hướng dẫn giao diện người dùng của Windows 95/98, các ứng dụng không nên sử dụng phím phải chuột để thực hiện ứng dụng kiểu TicTac. Thay vào đó, ứng dụng sẽ đáp lại nhấn đúp phím phải chuột bằng kích hoạt thực đơn ngữ cảnh nào đó như trong Shell của Windows 95/98. Khi thông điệp `WM_RBUTTONDOWN` được chuyển cho hệ thống để xử lý mặc định, Windows 95/98 đặt thông điệp `WM_CONTEXTMENU` vào hàng đợi thông điệp ứng dụng. Đặc trưng mới này của Windows 95/98 sẽ được xem xét trong chương sau.

3.2.1 Hộp thông báo

Trước khi ra khỏi ứng dụng, cả `OnLButtonDown` và `OnRButtonDown` gọi `CheckForGameOver` để biết trò chơi đã kết thúc chưa, nếu cần thì sẽ xoá lưới chơi. Nếu người chơi đã có được hàng ba ký hiệu X hoặc O, hay không còn hình vuông rỗng nào nữa, `CheckForGameOver` gọi hàm `MessageBox` của lớp cửa sổ để kích hoạt hộp thông báo để công bố thắng hay hoà (xem hình 3.2).



Hình 3.2 Hộp thông báo

MessageBox là công cụ vô cùng hiệu quả vì chúng cung cấp phương tiện để liên lạc với người sử dụng và có thể lấy giá trị trả lời. Nguyên hàm của chúng như sau:

```
CWnd::int MessageBox(LPCTSTR lpszText,
LPCTSTR lpszCaption = NULL,
UINT nType = MB_OK);
```

trong đó lpszText chỉ ra văn bản sẽ xuất hiện trong thân hộp thông báo, lpszCaption chỉ ra tiêu đề của hộp thông báo, và nType chứa một hay nhiều cờ bit xác định các đặc tính của hộp thông báo. Giá trị cho lại cho biết phím nào bị nhấn để bãi bỏ hộp thông báo. Các tiêu đề của lpszText và lpszCaption có thể là các đối tượng của CString hay con trỏ đến tiêu đề ký tự. (Bởi vì lớp CString nạp chồng toán tử LPCTSTR, ta luôn có thể truyền đối tượng CString đến hàm chấp nhận kiểu dữ liệu LPCTSTR). Giá trị NULL của lpszCaption sẽ hiển thị tiêu đề "Error" trên thanh tiêu đề.

Bảng 3.3 Các kiểu hộp thông báo và giá trị cho lại

Kiểu phím	Giá trị cho lại có thể
MB_ABORTRETRYIGNORE	Abort, Retry, ignore
	IDABORT, IDRETRY, IDIGNORE
MB_OK	OK
	IDOK
MB_OKCANCEL	OK, Cancel
	IDOK, IDCANCEL
MB_RETRYCANCEL	Retry, Cancel
	IDRETRY, IDCANCEL
MB_YESNO	Yes, No
	IDYES, IDNO
MB_YESNOCANCEL	Yes, No, Cancel
	IDYES, IDNO, IDCANCEL

Bảng 3.3 liệt kê sáu kiểu hộp thông báo và các giá trị tương ứng cho lại, với phím nhấn mặc định kiểu đậm (**boldface**). Trong các hộp thông báo đa phím, phím thứ nhất (phía trái nhất) thông thường là phím nhấn mặc định. Ta có thể làm cho phím hai hay ba là mặc định nhờ phép OR của MB_DEFBUTTON2 và MB_DEFBUTTON3 với giá trị kiểu của hộp thông báo. Lệnh:

```
MessageBox("Your document contains unsaved data. Save it?",
"My Application", MB_YESNOCANCEL | MB_DEFBUTTON3);
```

hiển thị hộp thông báo với ba phím - Yes, No, và Cancel - và giá trị cho lại từ hàm MessageBox sẽ là IDYES, IDNO, hoặc IDCANCEL, nhưng phím Cancel sẽ là phím mặc định.

Mặc định, các hộp thông báo có kiểu *modal*. Nếu muốn, ta có thể thêm MB_SYSTEMMODAL vào tham số nType để có hộp thông báo *modal* của Windows. Hộp thông báo *modal* của Windows

chỉ nên sử dụng khi có lỗi lớn, yêu cầu phải chú ý ngay chẳng hạn như hỏng dữ liệu đĩa từ.

Ta có thể thêm ảnh vào hộp thông báo nhờ MB_ICON. MB_ICONINFORMATION hiển thị ký tự "i" (có nghĩa "information") trong góc trên trái của hộp thông báo. Thông thường chúng được sử dụng khi cung cấp thông tin cho người dùng, không kèm theo câu hỏi:

```
MessageBox("No errors found. Click OK to continue",  
    "My Application", MB_ICONINFORMATION | MB_OK);
```

MB_ICONQUESTION sử dụng biểu tượng dấu hỏi và được dùng cho câu hỏi kiểu "Save before closing?". MB_ICONSTOP hiển thị vòng tròn đỏ với ký tự X và thông thường dùng để thông báo xảy ra lỗi không thể khắc phục được - thí dụ, lỗi tràn bộ nhớ yêu cầu phải dừng chương trình. Cuối cùng, MB_ICONEXCLAMATION hiển thị tam giác màu vàng và dấu chấm than. Chương trình 3.1 chỉ ra thí dụ hộp thông báo sử dụng kiểu MB_ICONEXCLAMATION. MFC cung cấp chức năng tương tự hàm thành phần CWnd::MessageBox dưới hình thức hàm tổng thể AfxMessageBox. Cả hai tương tự nhau, nhưng AfxMessageBox thêm khả năng liên kết hộp thông báo với tiện ích trợ giúp trực tuyến của chương trình. AfxMessageBox là không thể thay thế được khi lỗi nghiêm trọng xảy ra trong hàm InitInstance của đối tượng ứng dụng trước khi tạo cửa sổ, và ta muốn thông báo lỗi cho người sử dụng. MessageBox yêu cầu con trỏ CWnd còn hợp lệ và vì vậy không thể gọi chúng khi cửa sổ chính chưa được lập hay sau khi nó bị phá hủy. AfxMessageBox không yêu cầu con trỏ kiểu như vậy cho nên nó có thể được gọi bất kỳ khi nào.

3.2.2 Không có cửa sổ khung?

Thí dụ TicTac khác với thí dụ mô tả trong chương trước ở một khía cạnh quan trọng. Thay vì sử dụng cửa sổ khung suy diễn từ CFrameWnd, chúng suy diễn lớp cửa sổ trực tiếp từ CWnd. Cửa sổ "Trò chơi cờ caro" không thể thay đổi kích thước vì kích thước của lưới chơi là cố định. Và vì lý do thẩm mỹ, chúng có khung WS_EX_DLGMODALFRAME mỏng như ta nhìn thấy trên nhiều hộp thoại khác và nhiều cửa sổ có kích thước cố định khác. (Tổ hợp với chổi nền xám sáng, chúng sẽ cho cảm giác 3D trong Windows 95/98). Các thuộc tính này không phù hợp lắm với định nghĩa của MFC cho cửa sổ khung, vậy giải pháp tự nhiên là xây dựng lớp cửa sổ phái sinh từ CWnd. Phụ thuộc vào loại ứng dụng được viết mà ta nên làm điều này hay không.

Lập lớp cửa sổ phái sinh từ CWnd khá đơn giản. Khởi đầu, xây dựng lớp cửa sổ phái sinh từ CWnd thay cho CFrameWnd, như trong tệp TICTAC.H. Tương ứng, trong macro BEGIN_MESSAGE_MAP, phải chắc chắn là chỉ ra CWnd, chứ không phải CFrameWnd, là lớp cơ sở. Sau đó, trong cấu tử của cửa sổ, sử dụng hàm AfxRegisterWndClass để đăng ký lớp cửa sổ kiểu WNDCLASS với các thuộc tính lớp mong muốn, và gọi CWnd::Create hay CWnd::CreateEx để tạo cửa sổ. Việc đăng ký lớp cửa sổ riêng là không bắt buộc vì ta có thể sử dụng lớp mà MFC đã đăng ký, nhưng rất thường xuyên ta thấy rằng khi suy diễn lớp cửa sổ C++ mới từ CWnd, ta cũng sẽ

muốn chỉ ra luôn các thuộc tính của lớp WNDCLASS. Sau đây là mã trình tạo cửa sổ được liệt kê trong tệp TICTAC.CPP:

```
CString myClass = AfxRegisterWndClass (CS_DBLCLKS, myApp.LoadStandardCursor(IDC_ARROW),
    (HBRUSH) ::GetStockObject (LTGRAY_BRUSH),
    myApp.LoadStandardIcon (IDI_APPLICATION));
CreateEx (WS_EX_DLGMODALFRAME, myClass, "Trò chơi cờ caro", WS_OVERLAPPED |
    WS_SYSMENU | WS_CAPTION | WS_MINIMIZEBOX, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, NULL);
```

Lớp cửa sổ đã được đăng ký nhờ hàm AfxRegisterWndClass và được gán kiểu lớp CS_DBLCLKS, vậy cửa sổ sẽ nhận các thông điệp nhấn kép. CS_HREDRAW và CS_VREDRAW không được sử dụng ở đây bởi vì cửa sổ có kích thước không thay đổi. Con chạy mũi tên chuẩn và biểu tượng ứng dụng Windows được đăng ký như con chạy và biểu tượng của lớp, màu nền cửa sổ mặc định là màu xám sáng nhờ gọi hàm ::GetStockObject để lấy chỉ số điều khiển của đối tượng có sẵn LTGRAY_BRUSH. Một khi lớp được đăng ký, cửa sổ sẽ được tạo bởi CWnd::CreateEx. Thông thường ta gọi hàm CWnd::Create, nhưng vì WS_EX_DLGMODALFRAME là kiểu mở rộng của cửa sổ, cho nên phải gọi hàm CreateEx. Quan sát thấy kiểu cửa sổ bao gồm WS_OVERLAPPED, WS_SYSMENU, WS_CAPTION và WS_MINIMIZEBOX, nhưng không có WS_MAXIMIZEBOX. Chúng sẽ tạo cửa sổ với thanh tiêu đề, thực đơn hệ thống, mũi tên *minimize*, không có mũi tên *maximize*.

Kiểu WS_THICKFRAME sử dụng cho cửa sổ có khả năng thay đổi kích thước biên, các cạnh và các góc của chúng có thể được kéo bằng chuột. Kiểu cửa sổ WS_EX_DLGMODALFRAME tạo biên dày và không phản ứng với chuột. Nếu thay kiểu cửa sổ WS_EX_DLGMODALFRAME trong lời gọi của hàm CreateEx bằng kiểu WS_OVERLAPPEDWINDOW (hay đơn giản là thêm WS_THICKFRAME vào các kiểu đã có sẵn) thì ta có cửa sổ thay đổi được kích thước như cửa sổ khung.

Trình TicTac gán giá trị CW_USEDEFAULT cho tham số kích thước và vị trí của cửa sổ trong lời gọi hàm CreateEx. Kích thước của cửa sổ TicTac có kích cỡ phù hợp với lưới chơi. Xem các lệnh sau:

```
CRect rect (0, 0, 352, 352);
CalcWindowRect (&rect);
SetWindowPos (NULL, 0, 0, rect.Width (), rect.Height (),
    SWP_NOZORDER | SWP_NOMOVE | SWP_NOREDRAW);
```

Lệnh đầu tiên tạo đối tượng CRect để lưu kích thước mong muốn (thí dụ 352x352 pixel). Các giá trị không thể truyền trực tiếp đến hàm CreateEx, bởi vì các tham số kích thước của hàm CreateEx cho biết kích thước của chữ nhật của sổ chứ không phải kích thước của vùng làm việc của cửa sổ. Chữ nhật cửa sổ bao gồm cả vùng không phải của người sử dụng, thí dụ thanh tiêu đề, sẽ

khác nhau giữa các chương trình điều khiển màn hình và độ phân giải của chúng, kích thước của chữ nhật cửa sổ phải được tính từ chữ nhật làm việc và cửa sổ có kích thước vừa khít. Hàm `CWnd::CalcWindowRect` của MFC là công cụ tuyệt vời cho công việc này. Cho trước con trỏ đến đối tượng `CRect` chứa kích thước mong muốn của vùng làm việc cửa sổ, hàm `CalcWindowRect` tính chữ nhật cửa sổ tương ứng và lưu tọa độ vào đối tượng `CRect`. Sau đó kích thước chữ nhật đã hiệu chỉnh được sử dụng cho lời gọi `CWnd::SetWindowPos` để làm lại kích thước cửa sổ. Mấu chốt là hàm `CalcWindowRect` phải được gọi sau khi cửa sổ được lập, nếu không thì hàm `CalcWindowRect` không có cách gì để biết được các đặc tính của ngoài vùng làm việc. (Sự thật, việc gọi hàm `CalcWindowRect` trước hàm `Create` hay `CreateEx` trong gờ rời của chương trình sẽ dẫn đến lỗi trong *assertion* của MFC). Chính vì vậy hàm `CalcWindowRect` phải được gọi sau `CreateEx`.

`SetWindowPos` là hàm đa năng cho phép kích cỡ, định vị, vị trí theo trật tự z (trật tự trước-sau xác định cửa sổ nào nằm trên đỉnh của cái khác) sẽ được thay đổi. Chúng cũng được sử dụng thay cho `CWnd::ShowWindow` để dấu hay hiển thị cửa sổ.

```
BOOL SetWindowPos (const CWnd* pWndInsertAfter,
                  int x, int y, int cx, int cy, UINT nFlags)
```

trong đó `pWndInsertAfter` cho cửa sổ biết là chúng sẽ ở sau cửa sổ khác theo trật tự z. Các giá trị đặc biệt `wndTop` và `wndBottom` là đặt cửa sổ trên hay dưới đáy theo trật tự z, còn `wndTopMost` làm cho cửa sổ là ở trên cùng - là cửa sổ luôn ở trên các cửa sổ khác, thậm chí cả khi chúng không ở đỉnh của trật tự z - và `wndNoTopMost` là không phải cửa sổ trên đỉnh nhất. Một cách khác để làm cửa sổ là cửa sổ ở vị trí trên cùng nhờ hàm `SetWindowPos` là chỉ ra kiểu mở rộng `WS_EX_TOPMOST` khi cửa sổ được tạo bằng `CreateEx`. (Kiểu bù `WS_EX_TRANSPARENT` có tác động ngược lại, làm cho cửa sổ xuất hiện sau các cửa sổ khác, không phụ thuộc vào vị trí của chúng trong trật tự z). Các tham số x và y được truyền đến `SetWindowPos` là vị trí mới của cửa sổ trong tọa độ màn hình. Tham số cx và cy cho biết kích thước của cửa sổ tính bằng *pixel*. Cuối cùng, tham số `nFlags` chứa cờ tùy chọn. Ba cờ hay dùng nhất là `SWP_NOMOVE`, cho hàm `SetWindowPos` biết là phải bỏ qua các đầu vào x và y và để lại cửa sổ ở vị trí cũ của nó; `SWP_NOSIZE`, làm kích cỡ cửa sổ không thay đổi; và `SWP_NOZORDER`, ra lệnh cho `SetWindowPos` bỏ qua `pWndInsertAfter` và không làm thay đổi trật tự z của cửa sổ. Còn `SWP_NOSIZE | SWP_NOZORDER` trong cờ tùy chọn, làm dịch chuyển cửa sổ nhưng không thay đổi kích cỡ và vị trí trong trật tự z. TicTac sử dụng `SWP_NOMOVE`, `SWP_NOZORDER` và đặt cờ `SWP_NOREDRAW` làm cửa sổ không bị vẽ lại. Nhiều khi không cần vẽ lại cửa sổ.

Vấn đề cuối cùng được xem xét ở đây là sử dụng lớp cửa sổ phái sinh trực tiếp từ `CWnd` khi chúng được tạo lập, đối tượng cửa sổ phải được xóa. Các đối tượng `CFrameWnd` tự xóa khi các cửa sổ gắn vào nó bị xóa nhờ hàm nạp chồng `PostNcDestroy` để thực hiện thao tác:

```
delete this;
```

Bởi vì `CWnd::PostNcDestroy` không xóa đối tượng cửa sổ cho nên lớp cửa sổ phái sinh phải tự xóa đối tượng. Vì lý do này, TicTac bao gồm cả hàm `PostNcDestroy` để xóa đối tượng cửa sổ của lớp `CMainWindow` ngay trước khi chương trình kết thúc:

```
void CMainWindow::PostNcDestroy ()
{
    delete this;
}
```

Việc cài đặt `PostNcDestroy` được xem là chặt chẽ khi ta xây dựng lớp cửa sổ phái sinh từ `CWnd`.

3.3 CÁC THÔNG điệp CHUỘT NGOÀI VÙNG LÀM VIỆC

Khi phím chuột được nhấn hay chuyển động ngoài vùng làm việc của cửa sổ, Windows gửi thông điệp ngoài vùng làm việc để cho biết là có sự kiện. Bảng 3.4 là danh sách các thông điệp loại này mà ứng dụng có thể nhận được, nó tương đương với loại thông điệp chuột trong vùng người sử dụng. Tuy nhiên, nguyên hàm của hàm điều khiển thông điệp có khác.

```
void OnMsgName (UINT nHitTest, CPoint point);
```

Bảng 3.4 Các thông điệp chuột trong vùng không phải của người sử dụng

Thông điệp	Hàm điều khiển	Gửi khi
WM_NCLBUTTONDOWN	OnNcLButtonDown	Phím trái chuột bị nhấn
WM_NCLBUTTONUP	OnNcLButtonUp	Phím trái chuột được nhả
WM_NCLBUTTONDOWNBLCLK	OnNcLButtonDownBlCk	Nhấn đúp phím trái chuột
WM_NCMBUTTONDOWN	OnNcMButtonDown	Nhấn phím giữa chuột
WM_NCMBUTTONUP	OnNcMButtonUp	Nhả phím giữa
WM_NCMBUTTONDOWNBLCLK	OnNcMButtonDownBlCk	Nhấn đúp phím giữa
WM_NCRBUTTONDOWN	OnNcRButtonDown	Nhấn phím phải
WM_NCRBUTTONUP	OnNcRButtonUp	Nhả phím phải
WM_NCRBUTTONDOWNBLCLK	OnNcRButtonDownBlCk	Nhấn đúp phím phải
WM_NCMOUSEMOVE	OnNcMouseMove	Chuột chuyển dịch trên vùng không phải của người sử dụng

Cũng như các thông điệp chuột trong vùng làm việc, tham số *point* của hàm trên đây sẽ chỉ ra vị trí trong cửa sổ khi xảy ra sự kiện. Nhưng đối với thông điệp chuột ngoài vùng làm việc, *point.x* và *point.y* chứa toạ độ màn hình thay vì toạ độ vùng làm việc. Nếu muốn, toạ độ màn hình có thể chuyển đổi sang toạ độ vùng làm việc nhờ hàm `ScreenToClient` mà cửa sổ thừa hưởng từ `CWnd`. Tham số *nHitTest* thay tham số *nFlags* chứa mã *hit-test* cho biết phần ngoài vùng làm việc có xảy ra sự kiện.

Bảng 3.5 Các giá trị mẫu Hit-test ngoài vùng làm việc

Giá trị	Vị trí tương ứng
HTCAPTION	Thanh tiêu đề
HTGROWBOX	Phím restore-size (như HTSIZE)
HTHSCROLL	Thanh trượt ngang của cửa sổ
HTMENU	Thanh thực đơn
HTREDUCE	Phím minimize
HTSIZE	Phím restore-size
HTSYSMENU	Hộp thực đơn hệ thống
HTVSCROLL	Thanh trượt đứng
HTZOOM	Phím maximize

Bảng 3.5 là danh sách một vài giá trị *hit-test* có thể lập trình viên quan tâm; danh sách đầy đủ được tìm thấy trong tài liệu của thông điệp WM_NCHITTEST hay hàm CWnd::OnNcHitTest của MFC. Không như nhấn đúp trong vùng làm việc của cửa sổ, thông điệp nhấn đúp ở ngoài vùng làm việc được chuyển hay không phụ thuộc vào việc cửa sổ đã được đăng ký với kiểu CS_DBLCLKS hay không.

Thông thường trình không xử lý thông điệp chuột ngoài vùng làm việc nhưng cho phép Windows xử lý chúng. Windows cung cấp trả lời mặc định tương ứng và thông thường nó phát sinh ra các thông điệp khác cho cửa sổ. Thí dụ, khi Windows xử lý thông điệp WM_NCLBUTTONDOWNBLCLK với giá trị *hit-test* bằng HTSYSMENU, chúng sẽ phát sinh thông điệp WM_SYSCOMMAND với wParam bằng SC_CLOSE, nó khởi đầu một loạt hành động để đóng cửa sổ. Nếu muốn làm mất tác dụng đặc trưng này trong chương trình, hãy thực hiện như hàm điều khiển thông điệp sau:

```
void CMainWindow::OnNcLButtonDownBlk (UINT nHitTest, CPoint point)
{
    if (nHitTest != HTSYSMENU)
        CWnd::OnNcLButtonDownBlk (nHitTest, point);
}
```

Việc gọi hàm OnNcLButtonDown của lớp cơ sở sẽ chuyển thông điệp đến Windows và cho phép xử lý mặc định. Việc trở lại mà không gọi lớp cơ sở tránh cho Windows thấy được phím bị nhấn. Nếu làm như vậy thì nhấn đúp trong hộp thực đơn hệ thống sẽ không đóng cửa sổ nữa. (Dĩ nhiên, người dùng vẫn còn có thể đóng cửa sổ bằng nhấn hộp thực đơn hệ thống và chọn Close. Nếu muốn, ta cũng có thể làm vô hiệu đặc trưng này bằng chặn lại các thông điệp WM_SYSCOMMAND và lọc ra lệnh SC_CLOSE). Các giá trị *hit-test* khác có thể được sử dụng để tự đặt trả lời của cửa sổ cho các sự kiện khác từ ngoài vùng làm việc.

Trước khi cửa sổ nhận bất kỳ thông điệp nào trong hay ngoài vùng làm việc, nó nhận thông điệp WM_NCHITTEST với tham số CPoint chứa toạ độ màn hình của điểm nơi xảy ra sự kiện. Windows khảo sát các giá trị x và y trong đối tượng CPoint và sử dụng thông tin này để phát sinh các thông điệp trong hay ngoài vùng làm việc. Tuy có thể tự xử lý các thông điệp WM_NCHITTEST bằng cách thêm hàm OnNcHitTest và đầu vào ánh xạ thông điệp tương ứng trong ứng dụng, rất ít khi ta phải làm như vậy. Một khả năng sử dụng hàm OnNcHitTest là làm vô hiệu tạm thời toàn bộ đầu vào chuột như sau:

```
UINT CMainWindow::OnNcHitTest (CPoint point)
{
    return m_bIgnoreInput ? HTNOWHERE : CWnd::OnNcHitTest (point);
}
```

Nếu m_bIgnoreInput là TRUE, OnNcHitTest cho lại HTNOWHERE sao cho không có thông điệp trong hay ngoài vùng làm việc theo sau; nếu m_bIgnoreInput là FALSE, thì hàm OnNcHitTest của lớp cơ sở được gọi để làm dễ dàng các thao tác thông thường với cửa sổ. Cách tốt nhất để vô hiệu đầu vào là gọi hàm CWnd::EnableWindow với tham số FALSE, như sau:

```
EnableWindow (FALSE);
```

Sau đó, lệnh:

```
EnableWindow (TRUE);
```

làm cửa sổ có hiệu lực trở lại và khôi phục lại khả năng nhận đầu vào của chúng.

3.4 BẮT GIỮ (Capturing) CHUỘT

Một vấn đề thường nảy sinh trong chương trình là khi xử lý các thông điệp chuột thì việc nhận thông điệp nhấn phím không có nghĩa là ứng dụng sẽ nhận thông điệp nhả phím. Giả sử ta viết chương trình vẽ có khả năng lưu tham số *point* chuyển cho hàm OnLButtonDown và sử dụng chúng như điểm chốt để vẽ đường thẳng mà điểm kết thúc của chúng theo vị trí con chạy khi chuyển dịch chuột, hành động này được gọi là “căng dải cao su (*rubber-banding*)” theo đường thẳng. Khi thông điệp WM_LBUTTONUP đến, ta sẽ xoá đường dải cao su và thay vào đó là vẽ đường thực. Nhưng cái gì sẽ xảy ra nếu người sử dụng chuyển chuột ra ngoài vùng làm việc của cửa sổ trước khi nhả phím? Ta sẽ không bao giờ nhận được thông điệp WM_LBUTTONUP. Hậu quả là, đường thẳng dải cao su sẽ bị treo lại và đường thẳng thực sẽ không được vẽ.

Windows cung cấp một giải pháp cho vấn đề này bằng cách cho phép ứng dụng “bắt giữ” chuột và tiếp tục nhận các thông điệp chuột không phụ thuộc vào việc con chạy đang ở đâu trên màn hình. Chuột được bắt giữ nhờ hàm SetCapture của CWnd và loại bỏ nhờ hàm CWnd::ReleaseCapture.

Giữa hai lời gọi hàm này, tất cả các thông điệp chuột phát sinh bất kỳ đâu trên màn hình đều được hướng đến cửa sổ nơi thực hiện bắt giữ. Các thông điệp trong vùng làm việc tiếp tục thông báo các tọa độ của vị trí con chạy, các tọa độ này có thể có giá trị âm khi con chạy được đặt vào phía trái hay phía trên của góc trên-trái của vùng làm việc, và có giá trị dương khi vượt quá kích thước của vùng làm việc cửa sổ. Chỉ tồn tại một chương trình có thể bắt giữ chuột vào một thời điểm. Nếu muốn, ta có thể xác định nơi mà chuột bị bắt giữ (và nếu có thì ai thực hiện bắt giữ) nhờ gọi hàm `CWnd::GetCapture`. Hàm `GetCapture` cho lại con trỏ `CWnd` chỉ ra cửa sổ đang làm chủ chuột hay `NULL` nếu chuột không bị bắt giữ. Ta có thể kiểm tra để xem chương trình của ta đã bắt giữ chuột hay không bằng lệnh như sau:

```
if (GetCapture () == this)
```

Vậy, việc bắt giữ chuột giải quyết vấn đề đường thẳng dải cao su như thế nào? Bằng bắt giữ chuột khi trả lời thông điệp `WM_LBUTTONDOWN` và huỷ bỏ nó khi thông điệp `WM_LBUTTONUP` đến, ta đã được đảm bảo việc nhận thông điệp `WM_LBUTTONUP` khi nhả phím chuột. Vì vậy, chắc chắn xoá được đường thẳng dải cao su và vẽ đường thẳng thực thay chúng.

3.5 THỰC HIỆN BẮT GIỮ CHUỘT

Ứng dụng `MouseCap` (xem chương trình 3.2 và hình 3.3) là chương trình vẽ đơn giản, cho phép người sử dụng vẽ đường thẳng bằng chuột. Để vẽ đường thẳng, nhấn và giữ phím trái chuột bất kỳ đâu trong vùng làm việc của cửa sổ và di con chạy trong khi phím chuột đang bị nhấn. Vì chuột chuyển dịch, một đường thẳng mảnh (*dải cao su*) được vẽ giữa điểm khởi đầu (nơi nhấn chuột) và con chạy. Khi không còn nhấn phím chuột, đường thẳng mảnh bị xoá và đường thẳng màu đỏ rộng 16 *pixel* được vẽ thay thế chúng. Logic bắt giữ chuột đặc biệt có ý nghĩa khi chương trình tiếp tục trả lời các sự kiện chuột nếu chuột chuyển dịch ra ngoài vùng làm việc trong khi di chuột. Như vậy, đường thẳng tiếp tục được vẽ như chuột được dịch chuyển mà không cần để ý tới vị trí rồi bỏ phím chuột, đường thẳng màu đỏ được vẽ giữa hai điểm. Tất nhiên đường thẳng không bao giờ được kéo dài ra ngoài vùng làm việc của cửa sổ. Khi rời bỏ phím chuột bên ngoài vùng làm việc, đường thẳng được vẽ cho đến biên của vùng làm việc theo hướng của điểm nơi rời bỏ phím chuột.

Chương trình 3.2 MOUSECAP

Tệp `MOUSECAP.H`

```
class CMyApp : public CWinApp
{
public:
    BOOL InitInstance ();
};
```

```

class CMainWindow : public CFrameWnd
{
private:
    BOOL m_bTracking;
    BOOL m_bCaptureEnabled;
    CPoint m_pointFrom;
    CPoint m_pointTo;
public:
    CMainWindow ();
protected:
    afx_msg void OnLButtonDown (UINT, CPoint);
    afx_msg void OnLButtonUp (UINT, CPoint);
    afx_msg void OnMouseMove (UINT, CPoint);
    afx_msg void OnNclButtonDown (UINT, CPoint);
    DECLARE_MESSAGE_MAP ()
};

```

Tệp MOUSECAP.CPP

```

// MouseCap biểu diễn bắt giữ chuột
#include <afxwin.h>
#include "mousecap.h"
CMyApp myApp;
// Hàm thành phần của lớp CMyApp
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}
// Ánh xạ thông điệp và các hàm thành phần của CMainWindow
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_LBUTTONDOWN ()
    ON_WM_LBUTTONUP ()
    ON_WM_MOUSEMOVE ()
    ON_WM_NCLBUTTONDOWN ()
END_MESSAGE_MAP ()
//-----

```

```

CMainWindow::CMainWindow ()
{
    m_bTracking = FALSE;
    m_bCaptureEnabled = TRUE;
    Create (NULL, "Bắt giữ (capture) chuột");
    ::SetClassLong (m_hWnd, GCL_HCURSOR, (LONG) myApp.LoadStandardCursor (IDC_CROSS));
}

//-----
void CMainWindow::OnLButtonDown (UINT nFlags, CPoint point)
{
    m_pointFrom = point;
    m_pointTo = point;
    m_bTracking = TRUE;
    if (m_bCaptureEnabled) SetCapture ();
}

//-----
void CMainWindow::OnLButtonUp (UINT nFlags, CPoint point)
{
    if (m_bTracking) {
        m_bTracking = FALSE;
        if (GetCapture () == this) ReleaseCapture ();
        CClientDC dc (this);
        dc.SetROP2 (R2_XORPEN);
        dc.SelectStockObject (WHITE_PEN);
        dc.MoveTo (m_pointFrom);
        dc.LineTo (m_pointTo);
        dc.SetROP2 (R2_COPYPEN);
        CPen pen (PS_SOLID, 16, RGB (255, 0, 0));
        dc.SelectObject (&pen);
        dc.MoveTo (m_pointFrom);
        dc.LineTo (point);
    }
}

//-----
void CMainWindow::OnMouseMove (UINT nFlags, CPoint point)
{
    if (m_bTracking) {
        CClientDC dc (this);
        dc.SetROP2 (R2_XORPEN);
    }
}

```

```

        dc.SelectStockObject (WHITE_PEN);
        dc.MoveTo (m_pointFrom);
        dc.LineTo (m_pointTo);
        dc.MoveTo (m_pointFrom);
        dc.LineTo (point);
        m_pointTo = point;
    }
}
//-----
void CMainWindow::OnNcLButtonDown (UINT nHitTest, CPoint point)
{
    if (nHitTest == HTCAPTION) {
        m_bCaptureEnabled = m_bCaptureEnabled ? FALSE : TRUE;
        SetWindowText (m_bCaptureEnabled ?
            "Có khả năng bắt giữ (capture) chuột" :
            "Không có khả năng bắt giữ (capture) chuột");
    }
    CFrameWnd::OnNcLButtonDown (nHitTest, point);
}
//-----

```

Phần lớn các động tác được thực hiện ở trong các trình điều khiển OnLButtonDown, OnMouseMove, và OnLButtonUp của MouseCap. OnLButtonDown bắt đầu tiến trình vẽ bằng khởi tạo các biến là dữ liệu thành phần của lớp CMainWindow:

```

    m_pointFrom = point;
    m_pointTo = point;
    m_bTracking = TRUE;

```

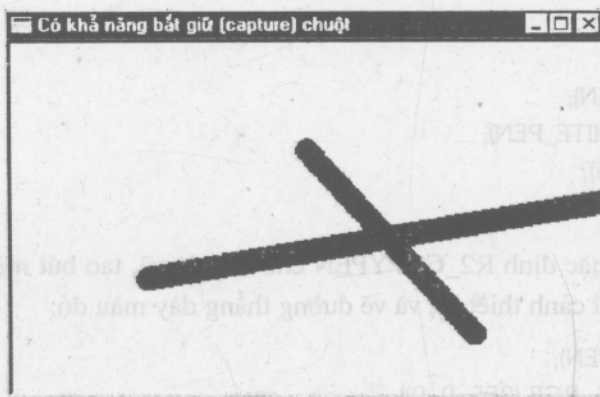
trong đó, m_pointFrom và m_pointTo là điểm đầu và điểm cuối của dải cao su. m_pointTo được hàm OnMouseMove cập nhật liên tục khi di chuột. m_bTracking được nạp giá trị TRUE khi phím trái bị nhấn và giá trị FALSE khi phím được thả, chúng được sử dụng như cờ cho hàm OnMouseMove và hàm OnLButtonUp để biết được dải cao su đã được vẽ chưa. Chỉ khi m_bCaptureEnabled có giá trị TRUE thì hàm OnLButtonDown mới bắt giữ chuột:

```

    if (m_bCaptureEnabled)
        SetCapture ();

```

trong đó, m_bCaptureEnabled được nhận giá trị TRUE nhờ cấu tử CMainWindow. Nó được lật giữa TRUE và FALSE nhờ hàm OnNcLButtonDown của chương trình để có thể bật hay tắt khả năng bắt giữ chuột, cho ta tự nhận biết được tác động như thế nào của việc bắt giữ chuột.



Hình 3.3 Kết quả thực hiện chương trình MouseCap

Công việc của `OnMouseMove` là xóa đường thẳng dài cao su cũ và vẽ đường mới mỗi khi dịch chuyển chuột. Điều này được thực hiện trong các dòng lệnh sau:

```
dc.MoveTo (m_pointFrom);
dc.LineTo (m_pointTo);
dc.MoveTo (m_pointFrom);
dc.LineTo (point);
```

Trước hết, đặt `m_pointTo` vào điểm nơi lần gọi tiếp của `OnMouseMove` sẽ xóa đường vừa vẽ. Sau khi khai báo đối tượng của lớp `CClientDC` để có ngữ cảnh thiết bị nhưng trước khi vẽ, `OnMouseMove` thực hiện dòng lệnh sau:

```
dc.SetROP2 (R2_XORPEN);
dc.SelectStockObject (WHITE_PEN);
```

Trong chương trước ta đã bàn luận về một vài thuộc tính của ngữ cảnh thiết bị như bút, màu tô hiện hành và chế độ ánh xạ. Một thuộc tính chưa được đề cập đến là chế độ vẽ, chúng xác định cách vẽ các *pixel* khi tổ hợp bút, chổi tô với các *pixel* trên màn hình. Chế độ vẽ mặc định là `R2_COPYPEN`, chúng sao chép các *pixel* lên bề mặt màn hình. Việc sử dụng hàm `SetROP2` (“*Set Raster Operation To*”) của lớp `CDC` để lập chế độ vẽ `R2_XORPEN` đặt cấu hình cho ngữ cảnh thiết bị sao cho các bit màu được vẽ bằng các *pixel* bút thực hiện XOR với các bit màu trong các *pixel* màn hình. Vẽ bằng chế độ `R2_XORPEN` và `WHITE_PEN` (tất cả các bit màu của chúng có giá trị 1) sẽ làm đảo các *pixel* màn hình bằng cách đổi 1 thành 0 và 0 thành 1, đảm bảo rằng đường thẳng dài cao su có thể nhìn thấy không phụ thuộc vào màu của nền khi vẽ lại. Quan trọng hơn, việc vẽ cùng đường thẳng trên cùng vị trí sẽ xóa đường thẳng nhờ khôi phục lại màu *pixel* gốc.

Khi phím trái được nhấn, hàm `OnLButtonUp` sẽ kết thúc. Sau khi gán giá trị `FALSE` cho

`m_bTracking` và rời phím chuột, nó xoá đường thẳng dài cao su được vẽ ở lần gọi `OnMouseMove` cuối cùng:

```
CClientDC dc (this);
dc.SetROP2 (R2_XORPEN);
dc.SelectStockObject (WHITE_PEN);
dc.MoveTo (m_pointFrom);
dc.LineTo (m_pointTo);
```

Sau đó lập lại chế độ mặc định `R2_COPYPEN` cho chế độ vẽ, tạo bút màu đỏ đồng nhất với độ rộng *16 pixel*, gán cho ngử cảnh thiết bị, và vẽ đường thẳng dày màu đỏ:

```
dc.SetROP2 (R2_COPYPEN);
CPen pen (PS_SOLID, 16, RGB (255, 0, 0));
dc.SelectObject (&pen);
dc.MoveTo (m_pointFrom);
dc.LineTo (point);
```

Công việc đã xong, `OnLButtonUp` trở về và việc vẽ đường thẳng đã kết thúc. `MouseCap` sẽ trở lại vòng lặp thông điệp để chờ nhấn phím trái chuột tiếp theo. Hình 3.6 là cửa sổ `MouseCap` sau khi vẽ một vài đường thẳng.

Sau khi đã vẽ vài đường, hãy nhấn thanh tiêu đề của cửa sổ để tác động hàm `OnNcLButtonDown` và lật cờ `m_bCaptureEnabled` từ `TRUE` sang `FALSE`. Tiêu đề của cửa sổ được thay đổi từ "Có khả năng bắt giữ (capture) chuột" thành "Không có khả năng bắt giữ (capture) chuột". `OnNcLButtonDown` bẫy phím trái của chuột ở ngoài vùng làm việc của cửa sổ. Hàm điều khiển sử dụng hàm `SetWindowText` mà `CMainWindow` thừa hưởng từ `CWnd` để thay đổi tiêu đề cửa sổ nếu mã *hit-test* trong `nHitTest` có giá trị `HTCAPTION`, chúng chỉ ra rằng phím chuột được nhấn trên thanh tiêu đề. Sau đó thông điệp được truyền đến hàm `OnNcLButtonDown` của lớp cơ sở để chuột không dừng hoạt động ở ngoài vùng làm việc của cửa sổ. Bây giờ vẽ một vài đường thẳng khi khả năng bắt giữ chuột vô hiệu. Đặc biệt, chú ý rằng nếu dịch chuột ra ngoài cửa sổ trong khi đang cao su theo đường thẳng, thì đường thẳng bị tê liệt cho đến khi chuột trở lại vùng làm việc của cửa sổ. Nếu ta bỏ phím chuột khi con chạy ở ngoài cửa sổ, chương trình mất đồng bộ. Đường thẳng dài cao su theo chuột khi dịch chuyển nó trở lại bên trong cửa sổ (ngay cả khi không nhấn phím chuột nữa), và trở thành cố định trong cửa sổ khi vẽ đường thẳng khác. Nhấn thanh tiêu đề lần nữa để khởi động lại khả năng bắt giữ chuột và chương trình sẽ tự trở lại trạng thái bình thường.

3.6 CON CHẠY

Một khía cạnh của quản lý chuột chưa được đề cập đến là cách thức thay đổi hình dạng con chạy. Thay vì con chạy hình mũi tên thông thường, `MouseCap` sử dụng con chạy gạch chéo thông dụng cho các chương trình vẽ và tô. Đó là hai trong 13 con chạy được định nghĩa trước trong

Windows cho lập trình viên, nếu không con chạy nào trong số con chạy định nghĩa trước thích hợp thì có thể tạo con chạy riêng.

Thứ nhất, mỗi khi chuột dịch chuyển, Windows xoá con chạy từ vị trí cũ của chúng bằng cách vẽ lại nền phía sau chúng. Sau đó, trước khi vẽ con chạy trên vị trí mới, Windows dùng để xác định cửa sổ nào chứa con chạy và khảo sát cấu trúc WNDCLASS liên quan để có được chỉ số điều khiển của con chạy. Sau đó nó vẽ con chạy trên màn hình sử dụng chỉ số điều khiển này. Kết quả, con chạy tự động được làm mới khi chúng chuyển dịch trên màn hình để phản ánh lớp con chạy của cửa sổ dưới nó. Thí dụ khi dịch con chạy trong điều khiển soạn thảo (*edit control*) chúng thay đổi từ mũi tên thành thanh đứng, hay *I-beam*. Điều này xảy ra được vì Windows đã đăng ký lớp tổng thể với tên EDIT cho các điều khiển soạn thảo và chỉ ra *I-beam* là lớp con chạy của chúng.

Cách thức để thay đổi hình dạng con chạy là đăng ký lớp cửa sổ riêng và chỉ ra kiểu con chạy mong muốn như là con chạy của lớp. AfxRegisterWndClass đăng ký lớp cửa sổ, và một trong các tham số mà nó tiếp nhận là chỉ số điều khiển của con chạy của lớp. Hàm CWinApp::LoadStandardCursor của MFC khai thác chỉ số điều khiển của con chạy để có một trong các kiểu con chạy định nghĩa trước. Ứng dụng TicTac đã sử dụng lớp cửa sổ với IDC_ARROW như con chạy của lớp và sau đó tạo lập cửa sổ từ lớp này bằng các lệnh sau:

```
CString myClass = AfxRegisterWndClass (CS_DBLCLKS,
    myApp.LoadStandardCursor (IDC_ARROW),
    (HBRUSH) ::GetStockObject (LTGRAY_BRUSH),
    myApp.LoadStandardIcon (IDI_APPLICATION));
CreateEx (WS_EX_DLGMODALFRAME, myClass, "Trò chơi cờ caro",
    WS_OVERLAPPED | WS_SYSMENU | WS_CAPTION | WS_MINIMIZEBOX,
    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, NULL);
```

Nếu ta muốn TicTac sử dụng con chạy gạch chéo thì phải chỉ ra giá trị IDC_CROSS trong lời gọi LoadStandardCursor:

```
CString myClass = AfxRegisterWndClass (CS_DBLCLKS,
    myApp.LoadStandardCursor (IDC_CROSS),
    (HBRUSH) ::GetStockObject (LTGRAY_BRUSH),
    myApp.LoadStandardIcon (IDI_APPLICATION));
```

Sau đó, mặc định, Windows sẽ hiển thị con chạy gạch chéo mỗi khi con trỏ chuột đặt trên cửa sổ TicTac. TicTac quy chiếu trực tiếp đối tượng ứng dụng (myApp) bởi vì myApp được khai báo trong cùng tệp mã nguồn như mã đăng ký lớp cửa sổ. Nếu myApp đặt trong tệp riêng, đối tượng ứng dụng sẽ được tham nhập nhờ hàm AfxGetApp. Mã đăng ký lớp sẽ như sau đây:

```
CString myClass = AfxRegisterWndClass (CS_DBLCLKS,
    AfxGetApp ()->LoadStandardCursor (IDC_CROSS),
```



```
(HBRUSH) ::GetStockObject (LTGRAY_BRUSH).
AfxGetApp ()->LoadStandardIcon (IDI_APPLICATION));
```

Hàm `AfxGetApp` thuận tiện khi ta viết chương trình MFC lớn với các lớp được cài đặt trong các tệp CPP riêng biệt.

`MouseCap` có quan điểm hơi khác khi tạo con chạy của lớp. Thay cho đăng ký lớp của sổ riêng, chúng sử dụng lớp cửa sổ mặc định do MFC cung cấp cho các cửa sổ khung và lập con chạy của lớp thành `IDC_CROSS` sau khi sử dụng hàm `API ::SetClassLong`.

```
::SetClassLong (m_hWnd, GCL_HCURSOR, (LONG) myApp.LoadStandardCursor (IDC_CROSS));
```

Hàm `::SetClassLong` thay đổi thuộc tính của lớp cửa sổ bằng cách nạp chồng giá trị tương ứng trong cấu trúc `WNDCLASS`. Tham số thứ nhất là chỉ danh cửa sổ mà lớp của chúng sẽ được biến đổi. Với đối tượng `CWnd`, `m_hWnd` lưu chỉ danh cửa sổ tương ứng. Điều quan trọng là gọi hàm `::SetClassLong` sau khi cửa sổ đã được lập, không được gọi trước, nếu không thì `m_hWnd` sẽ không giữ chỉ danh cửa sổ hợp lệ của cửa sổ và lời gọi sẽ thất bại. Nhận dạng `GCL_HCURSOR` sẽ cho hàm `::SetClassLong` biết thuộc tính nào của lớp sẽ thay đổi - trong trường hợp này là con chạy chuột của lớp. Và, dĩ nhiên, lời gọi `LoadStandardCursor` sẽ lấy ra chỉ danh con chạy. Kiểu của chỉ danh được ép thành `LONG` để tránh lỗi dịch. (`LONG` được sử dụng như kiểu dữ liệu chung bởi vì `::SetClassLong` chấp nhận nhiều kiểu dữ liệu khác nhau như: chỉ danh con chạy, chỉ danh biểu tượng, ...). Windows cũng cung cấp hàm `API ::GetClassLong` để lấy ra thuộc tính lớp. Như vậy, ta có thể lấy được chỉ danh con chạy lớp của cửa sổ bằng lệnh sau:

```
HCURSOR hCursor = ::GetClassLong (m_hWnd, GCL_CURSOR);
```

Khi ứng dụng đáp lời thông điệp bằng cách thực hiện nhiệm vụ xử lý trong thời gian dài, trong khi đó không có thông điệp nào sẽ được xử lý, thì nên chuyển con chạy thành đồng hồ cát (*hourglass*) để gợi nhớ rằng ứng dụng đang “bận”. Windows cung cấp hình dạng con chạy cho ta; nhận dạng của nó là `IDC_WAIT`. Các hàm `BeginWaitCursor` và `EndWaitCursor` của lớp `CCmdTarget` của MFC thay đổi con chạy thành đồng hồ cát và ngược trở lại. Mỗi lần gọi đến `BeginWaitCursor` làm tăng bộ đếm qui chiếu bên trong, và con chạy đồng hồ cát sẽ được hiển thị mãi cho đến khi số đếm lùi về 0. Nếu gọi hai lần hàm `BeginWaitCursor` thì cũng phải gọi hai lần `EndWaitCursor` để khôi phục lại con chạy thông thường. Các hàm này luôn được gọi bên trong miền tác động của cùng hàm xử lý thông điệp để tránh việc Windows vẽ lại con chạy lớp khi nhận được thông điệp `WM_MOUSEMOVE`. Nếu làm như sau chương trình sẽ không làm việc:

```
void CMainWindow::OnLButtonDown (UINT nFlags, CPoint point)
{
    BeginWaitCursor ();
}
void CMainWindow::OnLButtonUp (UINT nFlags, CPoint point)
{

```

```
EndWaitCursor ();
}
```

Ngay khi chuột được dịch chuyển sau khi phím trái chuột bị nhấn, đồng hồ cát sẽ biến mất. Nhưng nếu ta làm như sau:

```
void CMainWindow::OnLButtonDown (UINT nFlags, CPoint point)
{
    BeginWaitCursor ();
    .....
    EndWaitCursor ();
}
```

nó sẽ làm việc tốt vì không có thông điệp WM_MOUSEMOVE nào được xử lý giữa các lời gọi BeginWaitCursor và EndWaitCursor.

BeginWaitCursor, EndWaitCursor, và RestoreWaitCursor thực hiện công việc của chúng nhờ gọi hàm DoWaitCursor của CWinApp với các tham số vào là 1, -1, và 0. Nếu muốn, ta có thể thay thế con chạy “chờ” thành cái gì đó không phải là đồng hồ cát bằng cách nạp chồng DoWaitCursor và cung cấp cài đặt riêng của nó trong lớp phái sinh CWinApp. Ta có thể sử dụng cài đặt mặc định của CWinApp::DoWaitCursor trong tệp mã nguồn APPUI.CPP của MFC như là mô hình.

Cách khác để tạo lấy con chạy của chương trình ứng dụng là đăng ký lớp cửa sổ với con chạy NULL cho lớp (hay lập giá trị NULL cho con chạy của lớp nhờ hàm ::SetClassLong), và sau đó gọi ::SetCursor mỗi khi nhận được thông điệp WM_MOUSEMOVE.

```
CMainWindow::CMainWindow ()
{
    Create (NULL, "My Application");
    ::SetClassLong (m_hWnd, GCL_HCURSOR, NULL);
}

void CMainWindow::OnMouseMove (UINT nFlags, CPoint point)
{
    ::SetCursor (m_hCursor);
}
```

trong đó, m_hCursor lưu chỉ số điều khiển của con chạy. Không bắt buộc nạp NULL cho con chạy của lớp, nhưng nếu không làm được điều đó thì sẽ gây ra hiệu ứng nháy màn hình, khó chịu khi chuột di động bởi vì Windows vẽ lại nhanh con chạy của lớp trước mỗi lần gọi ::SetCursor. Con chạy NULL của lớp tránh cho Windows làm việc đó và cho phép ::SetCursor thực hiện dễ dàng công việc của chúng.

Tại sao ta đã phải sử dụng kỹ thuật này thay cho đăng ký m_hCursor như con chạy của lớp và để cho Windows thực hiện công việc còn lại? Giả sử ta viết ứng dụng hiển thị con chạy mũi tên

khi con chạy ở nửa trên của cửa sổ và con chạy *I-beam* khi con chạy ở nửa dưới cửa sổ. Con chạy của lớp không đủ trong trường hợp này. Đăng ký con chạy trong lớp là NULL như nói trên đây, khởi động cập biến thành phần `m_hCursorArrow` và `m_hCursorIBeam` bằng các chỉ số điều khiển của con chạy, và sau đó sử dụng hàm `OnMouseMove` để lập hình dạng con chạy khi cần thiết:

```
void CMainWindow::OnMouseMove (UINT nFlags, CPoint point)
{
    CRect rect;
    GetClientRect (rect);
    ::SetCursor ((point.y < (rect.Height () / 2)) ? ::SetCursor (m_hCursorArrow) :
        ::SetCursor (m_hCursorIBeam);
}
```

Bây giờ con chạy sẽ thay đổi từ mũi tên sang *I-beam* hay ngược lại khi nó vượt qua đường thẳng tưởng tượng chia màn hình thành hai nửa. Và bởi vì lớp con chạy là NULL, màn hình sẽ không nhấp nháy.

Đôi khi phải làm cho con chạy biến mất, hãy thực hiện như sau:

```
::ShowCursor (FALSE);
```

và khôi phục lại nó nhờ hàm:

```
::ShowCursor (TRUE);
```

Bên trong, Windows sẽ quản lý số đếm hiển thị, chúng được tăng khi gọi `::ShowCursor (TRUE)` và giảm khi gọi `::ShowCursor (FALSE)`. Số đếm được nạp giá trị ban đầu là 0 nếu chuột được cài và -1 nếu không có chuột, và con chạy được hiển thị mỗi khi số đếm lớn hơn hay bằng 0. Như vậy, nếu gọi `::ShowCursor (FALSE)` hai lần để dấu con chạy thì phải gọi hai lần `::ShowCursor (TRUE)` để hiển thị lại.

3.7 TỰ TẠO CON CHẠY

Windows không hạn chế ta sử dụng các kiểu con chạy định nghĩa trước nhờ hàm `LoadStandardCursor`. Nếu không có con chạy nào phù hợp, ta có thể tạo con chạy nhờ *Windows SDK Image Editor*, đó là *editor* tài nguyên cài đặt trong Visual C++. Con chạy chỉ là *bitmap* (hiện tại là hai *bitmap*), và tài nguyên *bitmap* được lập dễ dàng và cài vào ứng dụng Windows.

Khởi đầu bằng vẽ ảnh con chạy trong lưới *32x32 pixel* bằng *editor* tài nguyên và lưu ảnh vào tệp, giả sử với tên `MYCURSOR.CUR`. Tiếp theo, lập tệp mô tả tài nguyên cho ứng dụng và gộp dòng lệnh sau đây vào:

```
MyCursor CURSOR mycursor.cur
```

Việc này tạo tài nguyên con chạy mà ta có thể sử dụng nhờ gọi hàm `CWinApp::LoadCursor`. Bây giờ, thay việc gọi `LoadStandardCursor` để nạp 1 trong các con chạy xác định trước, ta gọi hàm `LoadCursor` để nạp tài nguyên con chạy với tên *MyCursor*:

```
m_hMyCursor = AfxGetApp()->LoadCursor ("MyCursor");
```

Chỉ danh của con chạy đã được lưu vào `m_hMyCursor` sau đó có thể được chuyển cho bất kỳ hàm nào có yêu cầu lấy chỉ danh con chạy làm đối số, bao gồm các hàm `AfxRegisterWndClass`, `::SetClassLong`, và `::SetCursor`.

3.8 CÁC VẤN ĐỀ KHÁC CỦA CHUỘT

Như đã nói trên đây, việc gọi hàm `::GetSystemMetrics` với đối `SM_CMOUSEBUTTONS` sẽ truy nhập hệ thống để lấy tổng số phím của chuột. Không có hàm tương đương trong MFC với `::GetSystemMetrics`, vậy ta phải gọi trực tiếp chúng. Các tham số khác liên quan đến chuột của hàm `::GetSystemMetrics` bao gồm `SM_CXDOUBLECLK` và `SM_CYDOUBLECLK`, chúng chỉ ra khoảng cách cực đại tính bằng *pixel* theo hai hướng ngang và dọc có thể tách hai nửa của nhấn đúp, và `SM_SWAPBUTTON`, cho lại giá trị khác 0 nếu người sử dụng đổi chỗ các phím trái và phải của chuột nhờ Control Panel. Khi các phím chuột được đổi chỗ, phím trái phát sinh các thông điệp `WM_LBUTTONDOWN` và phím phải phát sinh thông điệp `WM_RBUTTONDOWN`. Thông thường, ta không cần quan tâm đến vấn đề này, nhưng nếu vì lý do nào đó, ứng dụng muốn đảm bảo chắc chắn rằng phím trái thực sự ở phía trái của chuột, thì có thể sử dụng hàm `::GetSystemMetrics` để xác định xem các phím có bị đổi chỗ không.

Các hàm API `::SetDoubleClickTime` và `::GetDoubleClickTime` cho phép chương trình ứng dụng lập và truy nhập thời gian nhấn đúp của chuột - thời gian cực đại được phép giữa các nhấn khi phím chuột được nhấn đúp. Lệnh:

```
::GetDoubleClickTime ();
```

cho lại thời gian nhấn đúp tính bằng mili giây (ms), trong khi:

```
::SetDoubleClickTime (250);
```

lập 250 ms vào thời gian nhấn đúp. Khi nhấn liên tục cùng phím, Windows sử dụng thời gian nhấn đúp và các giá trị `SM_CXDOUBLECLK`, `SM_CYDOUBLECLK` cho lại bởi `::GetSystemMetrics` để xác định xem nhấn thứ hai có phải là nhấn đúp không.

Như trình bày trên đây, hàm xử lý các thông điệp chuột có thể xác định phím chuột nào được nhấn bằng cách kiểm tra tham số `nFlags` chuyển cho hàm xử lý thông điệp bởi MFC. Cũng có khả năng xác định xem phím chuột có bị nhấn bên ngoài hàm xử lý thông điệp chuột nhờ gọi `::GetKeyState` hay `::GetAsyncKeyState` với tham số `VK_LBUTTONDOWN`, `VK_MBUTTONDOWN`, hoặc `VK_RBUTTONDOWN` hay không. `::GetKeyState` chỉ được gọi trong hàm xử lý thông điệp bàn phím để

cho lại trạng thái của phím chuột vào thời điểm thông điệp bàn phím được phát sinh. `::GetAsyncKeyState` có thể được gọi từ bất kỳ hàm xử lý thông điệp nào. Chúng làm việc thời gian thực, cho lại trạng thái của phím chuột vào thời điểm hàm được gọi. Giá trị cho lại từ hàm:

```
::GetKeyState (VK_LBUTTON);
```

hay từ hàm:

```
::GetAsyncKeyState (VK_LBUTTON);
```

chỉ ra rằng phím chuột phải bị nhấn. Đổi chỗ các phím chuột không làm ảnh hưởng `::GetAsyncKeyState`, vậy nếu ta sử dụng hàm này thì phải sử dụng hàm `::GetSystemMetrics` để xác định xem phím có bị đổi cho nhau không. Lệnh:

```
::GetAsyncKeyState (::GetSystemMetrics (SM_SWAPBUTTON) ? VK_RBUTTON : VK_LBUTTON);
```

kiểm tra trạng thái của phím trái chuột đồng bộ, và tự động truy nhập phím chuột phải nếu các phím đổi chỗ.

Windows cung cấp hai hàm API với tên `::GetCursorPos` và `SetCursorPos` để khởi tạo và nhận vị trí con chạy bằng tay. Hàm có tên `::ClipCursor` cấm con chạy vào vùng đặc biệt của màn hình. `::ClipCursor` nhận con trỏ đến cấu trúc `RECT` mô tả, trong tọa độ màn hình, chữ nhật cắt. Vì con chạy là tài nguyên chung chia sẻ cho mọi ứng dụng, ứng dụng nào sử dụng `::ClipCursor` phải giải phóng con chạy nhờ gọi hàm:

```
::ClipCursor (NULL);
```

trước khi kết thúc, nếu không thì con chạy sẽ luôn bị khoá vào vùng *clip* cho đến khi khởi động lại hệ thống.

CHƯƠNG 4

BÀN PHÍM

Dù là sự kiện của chuột hay của bàn phím, ứng dụng sẽ được thông báo là có sự kiện đầu vào thông qua thông điệp. Chương trước chúng ta đã xem xét đầu vào chuột trong ngữ cảnh và khung ứng dụng MFC. Chương này sẽ tập trung vào bàn phím. Tương tự như chuột, bàn phím là tài nguyên phần cứng dùng chung và có thể chia sẻ cho nhiều chương trình. Windows® 95/98 quyết định sẽ gửi thông điệp chuột đến cửa sổ nào nhờ kết quả khảo sát cửa sổ dưới con chạy. Còn các thông điệp bàn phím được hướng vào cửa sổ có tiêu điểm vào (*input focus*). Thông thường cửa sổ có tiêu điểm vào là cửa sổ chính của ứng dụng đang tích cực, nhưng nó cũng có thể là cửa sổ con của cửa sổ chính, hay thậm chí là điều khiển trong hộp thoại. Nếu cửa sổ ứng dụng không có cửa sổ con, thì việc xử lý bàn phím sẽ tương đối dễ dàng. Nếu tiêu điểm chuyển đến điều khiển con, các thông điệp bàn phím đi đến cửa sổ chính sẽ dừng. Lập trình viên thường phải vượt qua vấn đề này bằng cách đảm bảo rằng tiêu điểm vào luôn ở trong cửa sổ chính. (Đôi khi cần đến phương sách sử dụng phương tiện hữu hiệu hơn, như phân lớp (*subclassing*) cửa sổ con hay tạo móc (*hook*) thông điệp để điều khiển dòng thông điệp. Cả hai kỹ thuật này sẽ không được bàn ở đây).

Windows thông báo cho cửa sổ là chúng chuẩn bị nhận hay mất tiêu điểm vào bằng các thông điệp WM_SETFOCUS và WM_KILLFOCUS, được các chương trình MFC xử lý trong các hàm OnSetFocus và OnKillFocus. Một ứng dụng có thể chuyển tiêu điểm vào đến cửa sổ khác nhờ hàm ::SetFocus của API và lấy chỉ danh cửa sổ sẽ nhận tiêu điểm vào để làm tham số:

```
::SetFocus (hwnd);
```

Hàm ::GetFocus của API và hàm tương đương trong MFC, CWnd::GetFocus, nhận ra cửa sổ có tiêu điểm vào. Trong Win32, cả hai hàm cho lại NULL nếu cửa sổ đang chiếm tiêu điểm vào không liên quan đến *thread* hiện hành.

4.1 CÁC THÔNG ĐIỆP GỠ PHÍM

Windows thông báo sự kiện phím nhấn và phím nhả cho ứng dụng bằng cách gửi thông điệp gõ phím WM_KEYDOWN và WM_KEYUP đến cửa sổ có tiêu điểm vào. Nếu các phím khác được gõ trong khi đang giữ xuống phím khác nữa thì các thông điệp kết quả WM_KEYDOWN và WM_KEYUP sẽ tách các thông điệp WM_KEYDOWN và WM_KEYUP để báo hiệu phím đang bị giữ xuống là nhấn hay nhả. Windows thông báo các sự kiện bàn phím theo thứ tự xảy ra, do vậy khảo sát dòng thông điệp gõ phím đi đến chương trình ứng dụng, ta có thể nói chính xác người sử dụng nhấn phím gì và vào thời điểm nào.

Nếu muốn biết khi nào các phím *Page Up* hay *Page Down* được nhấn để chương trình có phản ứng phù hợp, hãy xử lý các thông điệp `WM_KEYDOWN` và kiểm tra mã phím để nhận ra phím *Page Up* hoặc *Page Down*. Nếu muốn biết cả thời điểm nhả phím thì phải xử lý cả thông điệp `WM_KEYUP`. Với phần lớn các phím, ta có thể bỏ qua các thông điệp nhấn và nhả phím và yêu cầu Windows gửi thông điệp ký tự chỉ ra cái mà người sử dụng gõ. (Điều này sẽ trình bày sau đây, chúng xảy ra tự động khi ta sử dụng đối tượng ứng dụng suy diễn từ `CWinApp`). Sử dụng các ký tự thay cho các gõ phím thô làm đơn giản việc xử lý, trình ứng dụng bỏ qua trạng thái phím như phím *Shift* bị nhấn, phím *Caps Lock* tắt hay mở, ...

Cứ mỗi phím sẽ phát sinh ra các thông điệp `WM_KEYDOWN` và `WM_KEYUP`. Có hai phím đặc biệt là *Alt* và *F10*, chúng là phím hệ thống. Mỗi khi một trong các phím này bị nhấn hay nhả, cửa sổ nhận các thông điệp `WM_SYSKEYDOWN` và `WM_SYSKEYUP` thay cho các thông điệp gõ phím không phải của hệ thống. Nếu các phím khác bị nhấn trong khi giữ phím *Alt*, chúng cũng phát sinh các thông điệp phím hệ thống.

Các ứng dụng sẽ xử lý các thông điệp gõ phím bằng cách cung cấp hàm quản lý thông điệp cho các thông điệp mà ta quan tâm và đưa chúng vào vòng xử lý thông điệp thông qua ánh xạ thông điệp. Các thông điệp `WM_KEYDOWN`, `WM_KEYUP`, `WM_SYSKEYDOWN`, và `WM_SYSKEYUP` được xử lý bằng các hàm thành phần lớp `OnKeyDown`, `OnKeyUp`, `OnSysKeyDown`, và `OnSysKeyUp`. Khi được kích hoạt, các hàm điều khiển này nhận thông tin về gõ phím, bao gồm cả mã để nhận ra phím đã bị nhấn hay nhả. Các hàm quản lý thông điệp có nguyên hàm như sau:

```
void OnMsgName (UINT nChar, UINT nRepCnt, UINT nFlags);
```

Trong đó, `nChar` chỉ ra phím bị nhấn hay nhả. `nRepCnt` là số đếm nhắc lại: tổng số gõ phím được lưu trong thông điệp. Thông thường `nRepCnt` có giá trị 1 cho thông điệp `WM_KEYDOWN` hay `WM_SYSKEYDOWN` và luôn có giá trị 1 cho các thông điệp `WM_KEYUP` hay `WM_SYSKEYUP`. Nếu các thông điệp nhấn phím đi đến quá nhanh đến nỗi ứng dụng không giữ được chúng thì Windows sẽ tổ hợp hai hay nhiều thông điệp `WM_KEYDOWN` hoặc `WM_SYSKEYDOWN` thành một, đồng thời làm tăng số đếm tương ứng. Phần lớn các chương trình bỏ qua số đếm nhắc lại và xử lý các thông điệp tổ hợp phím nhấn (các thông điệp có `nRepCnt` lớn hơn 1) như gõ phím đơn để tránh tình trạng tràn mà chương trình vẫn tiếp tục trượt để trả lời thông điệp ngay cả khi đã bỏ ngón tay ra khỏi phím. Không như BIOS bàn phím của PC, lưu các gõ phím và thông báo từng cái riêng rẽ, logic quản lý bàn phím của Windows cung cấp hàng rào chống lại tràn bàn phím.

Tham số `nFlags` truyền đến hàm quản lý thông điệp gõ phím chứa mã quét OEM (*Original Equipment Manufacturers*) 8 bit của phím và các cờ bit chỉ ra trạng thái chuyển tiếp, trạng thái phím trước đó, mã ngữ cảnh, và phím có phải là phím mở rộng hay không (xem bảng 4.1).

Bảng 4.1 Tham số nFlags chuyển cho hàm quản lý thông điệp gõ phím

<i>Bit(s)</i>	<i>Ý nghĩa</i>	<i>Mô tả</i>
0-7	OEM scan code	8-bit OEM scan code do phần cứng bàn phím tạo ra
8	Cờ phím mở rộng	1 nếu là phím mở rộng, 0 nếu không
9-12	Dự trữ	N/A
13	Mã ngữ cảnh	1 nếu giữ xuống phím Alt, 0 ngược lại
14	Trạng thái phím trước	1 nếu phím trước đó bị nhấn, 0 nếu nhả
15	Trạng thái quá độ	0 nếu phím bị nhấn, 1 nếu bỏ phím

Cờ của các phím mở rộng cho phép ứng dụng phân biệt giữa các phím đúp thường thấy trong nhiều bàn phím. Các bàn phím 101- và 102-key được sử dụng cho phần lớn các PC, cờ này được lập cho các phím *Ctrl* và *Alt* trên phía phải của bàn phím, các phím *Home*, *End*, *Insert*, *Delete*, *Page Up*, *Page Down*, và các phím trượt ở giữa phần chính của bàn phím và phần chứa phím số, phím *Enter* và *slash (/)*. Với các phím khác, cờ phím mở rộng là 0. Mã quét OEM có giá trị 8 bit chỉ ra phím trong BIOS. Phần lớn các ứng dụng bỏ qua trường này, bởi vì các phím thường được nhận ra nhờ các mã phím ảo. (Nếu muốn, các mã quét được chuyển đổi sang mã phím ảo nhờ hàm `::MapVirtualKey`). Trạng thái chuyển đổi, trạng thái phím trước đó, mã ngữ cảnh thường không được quan tâm, nhưng chúng rất hữu hiệu trong một số trường hợp.

Bảng 4.2 Các thông điệp typematic

<i>Thông điệp</i>	<i>Mã phím ảo</i>	<i>Trạng thái phím trước</i>
WM_KEYDOWN	VK_SHIFT	0
WM_KEYDOWN	VK_SHIFT	1
WM_KEYDOWN	VK_SHIFT	1
WM_KEYDOWN	VK_SHIFT	1
WM_KEYDOWN	VK_SHIFT	1
WM_KEYDOWN	VK_SHIFT	1
WM_KEYDOWN	VK_SHIFT	1
WM_KEYDOWN	VK_SHIFT	1
WM_KEYDOWN	VK_SHIFT	1
WM_KEYDOWN	VK_SHIFT	1
WM_KEYUP	VK_SHIFT	1

Giá trị 1 của trạng thái phím trước cho biết có gõ phím hay không – gõ phím được sinh ra khi phím được nhấn và giữ xuống trong một khoảng thời gian. Thí dụ, giữ phím Shift trong một giây, chúng sẽ tạo ra một xâu liên tiếp thông điệp, các mã phím ảo, trạng thái phím trước đó được chỉ ra trong bảng 4.2.

Để không phải quan tâm tới các thông điệp tạo ra bởi tác động *typematic*, ta bỏ qua các thông điệp nhấn phím xuống với trạng thái phím trước đó có giá trị 1. Giá trị trạng thái quá độ là thông

tin thừa đối với nhiều chương trình, bởi vì chúng luôn có giá trị 0 cho các thông điệp WM_KEYDOWN và WM_SYSKEYDOWN và giá trị 1 cho các thông điệp WM_KEYUP và WM_SYSKEYUP. Cuối cùng, mã ngữ cảnh chỉ ra phím *Alt* có được nhấn hay không khi thông điệp được phát sinh. Với một số ngoại lệ (thông thường là không quan trọng), chúng có giá trị 1 cho các thông điệp WM_SYSKEYDOWN và WM_SYSKEYUP và giá trị 0 cho thông điệp WM_KEYDOWN và WM_KEYUP.

Tổng quát, ứng dụng không cần xử lý các thông điệp WM_SYSKEYDOWN và WM_SYSKEYUP, nhưng phải cho phép Windows xử lý thay. Nếu các thông điệp này không tìm thấy đường đi đến hàm DefWindowProc, tổ hợp phím hệ thống như *Alt-Tab* và *Alt-Esc* sẽ không làm việc nữa. Windows rất mềm dẻo, chuyển tất cả các thông điệp chuột và bàn phím qua ứng dụng của lập trình viên trước, thậm chí ngay cả các thông điệp chỉ có ý nghĩa với hệ điều hành. Quản lý không phù hợp các thông điệp gõ phím hệ thống và các thông điệp ngoài vùng làm việc của chuột, đặc biệt, thất bại trong việc chuyển các thông điệp đến hệ điều hành thì có thể gây ra các phản ứng không đợi trước của hệ thống.

4.2 MÃ PHÍM ẢO

Giá trị quan trọng nhất chuyển cho hàm quản lý các thông điệp gõ phím là giá trị *nChar* cho biết phím bị nhấn hay đã được nhả. Để cho ứng dụng không phải dựa vào các giá trị cố định hay mã quét OEM khác nhau từ bàn phím này đến bàn phím khác, Windows cung cấp các mã phím ảo như trong bảng 4.3. Nếu xem kỹ bên trong tệp WINUSER.H, nơi định nghĩa các mã phím ảo, ta sẽ không thấy một số mã phím, bao gồm VK_SELECT, VK_PRINT, VK_EXECUTE, và VK_F13 đến VK_F24. Các mã này được sử dụng trong các loại máy tính khác và không thể được sinh trên các bàn phím theo tiêu chuẩn IBM. Các phím mà Windows không cung cấp các mã phím ảo - các phím chữ và số, các phím biểu tượng như ; và] - phải được tránh xa khi xử lý các thông điệp-phím nhấn, phím nhả bởi vì ID của phím có thể khác nhau trên các bàn phím quốc tế. Điều này không có nghĩa là chương trình của ta không thể nhập các ký tự phân cách và các ký tự mà không tồn tại các ID kiểu VK_XXX; chỉ đơn thuần là có cách tốt hơn để làm việc này hơn là phụ thuộc vào các thông điệp nhấn phím, nhả phím. Cách tốt hơn này là các thông điệp ký tự và chúng sẽ được đề cập đến ở dưới đây.

4.3 TRẠNG THÁI PHÍM SHIFT VÀ LẬT

Khi viết hàm xử lý thông điệp cho các thông điệp WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN, và WM_SYSKEYUP, ta cần biết rằng các phím *Shift*, *Ctrl*, hoặc *Alt* có bị nhấn hay không trước khi quyết định trả lời như thế nào. Thông tin trạng thái *Shift* liên quan đến các phím *Shift* và *Ctrl* sẽ không được mã hoá trong các thông điệp bàn phím như thông điệp chuột, vậy Windows cung cấp hàm ::GetKeyState. Cho trước mã phím ảo, ::GetKeyState cho biết phím

tương ứng đang bị nhấn và, trong trường hợp của phím lật (*toggle*) như *Num Lock*, trạng thái tương ứng đang là bật hay tắt.

Bảng 4.3. Mã phím ảo

<i>Mã phím ảo</i>	<i>Phím tương ứng</i>
VK_F1 đến VK_F12	Các phím chức năng từ F1 đến F12
VK_NUMPAD0 đến VK_NUMPAD9	Các phím số 0 đến 9 khi Num Lock ở trạng thái bật
VK_CANCEL	Ctrl-Break
VK_RETURN	Enter
VK_BACK	Backspace
VK_TAB	Tab
VK_CLEAR	Numeric keypad 5 khi Num Lock tắt
VK_SHIFT	Shift
VK_CONTROL	Ctrl
VK_MENU	Alt
VK_PAUSE	Pause
VK_ESCAPE	Esc
VK_SPACE	Space bar
VK_PRIOR	Page Up và PgUp
VK_NEXT	Page Down và PgDn
VK_END	End
VK_HOME	Home
VK_LEFT	Left arrow
VK_UP	Up arrow
VK_RIGHT	Right arrow
VK_DOWN	Down arrow
VK_SNAPSHOT	Print Screen
VK_INSERT	Insert và Ins
VK_DELETE	Delete và Del
VK_MULTIPLY	Numeric keypad *
VK_ADD	Numeric keypad +
VK_SUBTRACT	Numeric keypad -
VK_DECIMAL	Numeric keypad .
VK_DIVIDE	Numeric keypad /
VK_CAPITAL	Caps Lock
VK_NUMLOCK	Num Lock
VK_SCROLL	Scroll Lock

Lệnh:

```
::GetKeyState (VK_SHIFT)
```

cho lại giá trị âm nếu phím Shift đang bị nhấn, hay giá trị không âm nếu ngược lại. Tương tự, lệnh:

```
::GetKeyState (VK_CONTROL)
```

cho lại giá trị âm nếu phím *Ctrl* đang bị nhấn. Như vậy các lệnh trong ngoặc đơn trong đoạn mã lệnh sau đây trích ra từ hàm *OnKeyDown* sẽ chỉ được thực hiện khi phím *Ctrl-Left* (phím mũi tên trái) bị nhấn:

```
if ((nChar == VK_LEFT) && (::GetKeyState (VK_CONTROL) < 0))
{
//
}
```

Để nhận biết phím *Alt*, đơn giản chỉ cần kiểm tra bit mã ngữ cảnh trong tham số *nFlags* hay gọi *::GetKeyState* với tham số *VK_MENU*. Thông thường, thậm chí ngay cả khi không cần thiết, cửa sổ của ta sẽ nhận thông điệp *WM_SYSKEYDOWN* hoặc *WM_SYSKEYUP* thay vì các thông điệp *WM_KEYDOWN* hoặc *WM_KEYUP*. Nói cách khác, ngữ cảnh thông điệp đã cho ta biết hầu hết điều mà ta cần biết về phím *Alt*. Thêm nữa, ta có thể sử dụng *VK_LBUTTON*, *VK_MBUTTON*, và *VK_RBUTTON* để xác định xem phím trái, giữa, phải của chuột có bị nhấn không. Ứng dụng có thể sử dụng hàm *::GetKeyState* để xác định các phím *Num Lock*, *Caps Lock*, và *Scroll Lock* đang ở trạng thái tắt hay mở. Còn các bit cao của mã cho lại chỉ ra rằng phím đang bị nhấn hay không (phát sinh số âm khi bit cao có giá trị 1), bit thấp, bit thứ 0, chỉ trạng thái của phím lật. Dòng lệnh sau:

```
::GetKeyState (VK_NUMLOCK) & 0x01
```

có giá trị *TRUE* (khác 0) nếu phím *Num Lock* là bật và *FALSE* (*zero*) trong trường hợp ngược lại. Kỹ thuật tương tự áp dụng cho các phím *VK_CAPITAL* (*Caps Lock*) và *VK_SCROLL* (*Scroll Lock*). Quan trọng là lọc bỏ tất cả trừ bit thấp nhất của mã cho lại trước khi kiểm tra, bởi vì, bit cao của chúng còn chỉ ra chính bản thân phím có bị nhấn hay không.

Trong mọi trường hợp, *::GetKeyState* thông báo trạng thái của phím bàn phím hay phím chuột vào thời điểm thông điệp bàn phím được phát sinh, chứ không phải là trạng thái hiện tại của phím vào thời điểm chính xác mà hàm được gọi. Có nghĩa rằng ta không phải bận tâm về phím trượt không bị nhấn trước khi hàm quản lý thông điệp hỏi trạng thái phím. Hàm *::GetKeyState* hầu như không bao giờ được gọi bên ngoài hàm xử lý thông điệp bàn phím bởi vì thông tin mà nó cho lại chỉ hợp lệ sau khi thông điệp bàn phím đã được khai thác từ hàng đợi thông điệp. Nếu ta thực sự cần trạng thái hiện hành của phím chuột hay bàn phím, hay nếu ta muốn kiểm tra phím chuột hay bàn phím bên ngoài hàm xử lý thông điệp bàn phím thì hãy sử dụng hàm *::GetAsyncKeyState*.

4.4 CÁC THÔNG điệp KÝ TỰ

Một trong các vấn đề mà ta sẽ gặp là dựa vào các thông điệp phím lên và phím xuống để nhập dữ liệu từ bàn phím như thí dụ dưới đây. Giả sử ta viết ứng dụng nhập văn bản để chuyển các phím ký tự cố nghĩa thành các ký tự trên màn hình. Khi phím A bị nhấn thì thông điệp WM_KEYDOWN đi đến cùng với mã phím ảo VK_A. Trước khi đẩy ký tự A ra màn hình, ta gọi hàm `::GetKeyState` để xác định xem phím Shift có bị nhấn hay không. Nếu bị nhấn thì sẽ cho ra chữ A; trường hợp ngược lại cho ra chữ a. Cho đến đây mọi việc đều tốt đẹp. Nhưng cái gì sẽ xảy ra khi phím *Caps Lock* cũng tích cực (đang đóng)? *Caps Lock* lật lại hiệu ứng của phím *Shift*, chuyển đổi A thành a và a thành A. Tồn tại 4 hoán vị của ký tự A mà ta phải xem xét (xem bảng 4.4).

Bảng 4.4 Bốn hoán vị của chữ A

Mã phím ảo	VK_SHIFT	CapsLock	Kết quả
0x41	No	mở	a
0x41	Yes	mở	A
0x41	No	đóng	A
0x41	Yes	đóng	a

Trong khi phải vượt qua vấn đề này bằng cách viết mã trình xử lý toàn bộ các trạng thái *shift* và lật có thể xảy ra, nhiệm vụ của ta còn bị phức tạp vì người sử dụng đôi khi còn phải nhấn cả phím *Ctrl* xuống. Tuy nhiên vấn đề chỉ phức tạp xảy ra khi ứng dụng chạy trên các loại bàn phím khác nhau. Người Mỹ nhấn *Shift-0* để nhập ký hiệu đóng ngoặc đơn (")". Nhưng *Shift-0* sẽ phát sinh ký hiệu = trên phần lớn các bàn phím quốc tế và dấu nháy (*apostrophe*) trên bàn phím Đức. Người sử dụng sẽ không đánh giá cao chương trình nếu các ký tự mà chương trình của ta hiện lên lại không hợp với các phím mà ta gõ.

Vì vậy, Windows cung cấp thông điệp WM_CHAR. Để cho dấu vào ký tự được thực hiện độc lập với kiểu bàn phím, hàm `::TranslateMessage` được gọi từ vòng lặp thông điệp trước khi chuyển giao chúng. Các thông điệp WM_KEYDOWN/UP và WM_SYSKEYDOWN/UP được dịch thành thông điệp WM_CHAR và WM_SYSCHAR để mô tả ký tự, có tính đến tất cả các trạng thái shift, lật và sự khác nhau của các loại bàn phím. MFC quan tâm vấn đề này thay ta nhờ gọi hàm `::TranslateMessage` trong vòng lặp thông điệp bên trong CWinApp. Khi ta xử lý các thông điệp ký tự thì không phải bàn khoăn về mã phím ảo và trạng thái *shift* trừ những phím (như các phím chức năng, phím trượt, và các phím *Page Up* và *Page Down*) không tồn tại ký tự tương ứng. Thay vào đó, chúng ta sẽ nhận được mã ANSI (*American National Standards Institute*). Giả sử *Lock* không ở trạng thái đóng thì việc nhấn *Shift-A* sẽ phát sinh trình tự các thông điệp như bảng 4.5.

Bảng 4.5 Nhấn Shift-A

<i>Thông điệp</i>	<i>Mã phím ảo</i>	<i>Mã ANSI</i>
WM_KEYDOWN	VK_SHIFT	
WM_KEYDOWN	0x41	
WM_CHAR		0x41 (A)
WM_KEYUP	0x41	
WM_KEYUP	VK_SHIFT	

Bây giờ có thể an toàn bỏ qua các thông điệp phím lên và phím xuống bởi vì mọi thứ cần biết về gõ phím đã được mã hoá trong thông điệp WM_CHAR. Nếu phím *Alt* bị nhấn, ứng dụng sẽ nhận thông điệp WM_SYSCHAR (xem bảng 4.6).

Bảng 4.6 Nhấn Alt-A

<i>Thông điệp</i>	<i>Mã phím ảo</i>	<i>Mã ANSI</i>
WM_SYSKEYDOWN	VK_SHIFT	
WM_SYSKEYDOWN	0x41	
WM_SYSCHAR		0x41 (A)
WM_SYSKEYUP	0x41	
WM_SYSKEYUP	VK_SHIFT	

Vì các tổ hợp phím với *Alt*, nói chung, thường được sử dụng cho mục đích riêng, phần lớn các ứng dụng bỏ qua các thông điệp WM_SYSCHAR và chỉ xử lý WM_CHAR.

Ứng dụng MFC xử lý các thông điệp WM_CHAR trong hàm thành phần `OnChar` và đầu vào ánh xạ thông điệp liên quan trong lớp cửa sổ và lớp quan sát như sau:

```
void OnChar (UINT nChar, UINT nRepCnt, UINT nFlags);
```

Trong đó, `nRepCnt` và `nFlags` có cùng ý nghĩa như khi xử lý các thông điệp gõ phím. `nChar` lưu mã ký tự ANSI. Một điều phải thận trọng khi hiển thị các ký tự nhận được thông qua các thông điệp WM_CHAR (hoặc WM_SYSCHAR) là co chữ mà ta đang sử dụng là co chữ ANSI chứ không phải là OEM. Các ký tự có mã từ 32 đến 127, các ký tự trong co chữ ANSI và OEM là như nhau. Ngoài dải này, dĩ nhiên, các ký tự sẽ khác nhau. Nếu người dùng bàn phím Đức gõ chữ *O* hoa với biến âm sắc (...), thì ứng dụng sẽ nhận thông điệp WM_CHAR với `nChar` bằng 214. Nếu ta hiển thị ký tự này bằng co chữ ANSI thì chúng sẽ hiển thị đúng. Nếu sử dụng co chữ OEM thì Windows sẽ hiển thị ký tự đồ họa kỳ quặc. May thay, co chữ hệ thống mặc định mà Windows chọn cho ngữ cảnh thiết bị là co chữ ANSI, cũng giống như phần lớn các co chữ Windows khác. Chỉ một ngoại lệ là co chữ OEM_FIXED_FONT là 1 trong các co chữ có sẵn mà ta có thể chọn nhờ hàm `CDC::SelectStockObject`.

Khi cần chuyển đổi xâu mã ký tự từ OEM sang ANSI hay ngược lại, lớp `CString` của MFC cung

cấp hàm với tên `AnsiToOem` và `OemToAnsi`. Một ứng dụng của hàm `OemToAnsi` là chuyển đổi tệp văn bản được lập từ ứng dụng trong môi trường MS-DOS thành chuẩn ANSI để hiển thị trong các ứng dụng của Windows. Luôn nhớ rằng việc chuyển đổi có thể không hoàn hảo. Nếu ký tự trong xâu chuyển đổi không tồn tại trong tập ký tự khác thì Windows sẽ cố hết sức để tìm ra cái có thể chấp nhận được nhờ bảng chuyển đổi do trình điều khiển bàn phím cung cấp. Nhưng kết quả là đầu ra không được như gốc.

4.5 CON CHẠY SOẠN THẢO

Thanh đứng nhỏ mà nhiều ứng dụng dùng để đánh dấu điểm nơi ký tự tiếp theo sẽ xen vào khi nhập văn bản được gọi là con chạy soạn thảo (*caret*).

Bảng 4.7 Các hàm con chạy soạn thảo của CWnd

Hàm	Mô tả
<code>CreateCaret</code>	Lập con chạy soạn thảo từ bitmap
<code>CreateSolidCaret</code>	Lập con chạy soạn thảo trông như bloc đặc hay đường
<code>CreateGrayCaret</code>	Lập con chạy soạn thảo trông như bloc xám hay đường
<code>GetCaretPos</code>	Lấy vị trí con chạy soạn thảo hiện hành
<code>SetCaretPos</code>	Đặt vị trí của con chạy soạn thảo
<code>ShowCaret</code>	Hiển thị con chạy soạn thảo
<code>HideCaret</code>	Đấu con chạy soạn thảo đi

Lớp `CWnd` của MFC cung cấp bảy hàm như trong bảng 4.7 để xử lý con chạy soạn thảo. Một hàm quan trọng thiếu trong bảng, hàm `::DestroyCaret`, phải được gọi trực tiếp từ Windows API bởi vì chúng không có tương đương trong MFC.

Con chạy soạn thảo là tài nguyên chung, như con chạy chuột. Chỉ có một con chạy soạn thảo và chỉ một mà thôi trong chương trình tại một thời điểm. Vì vậy, chương trình không thể sử dụng con chạy soạn thảo theo mọi cách mà ta muốn; phải lưu tâm đến các chương trình khác vì có thể chúng cần nó. Quy luật sử dụng con chạy soạn thảo đơn giản như trình bày dưới đây. Cửa sổ sử dụng con chạy soạn thảo sẽ tạo chúng nhờ hàm `CreateCaret`, `CreateSolidCaret`, hay `CreateGrayCaret` khi chúng nhận tiêu điểm vào, bãi bỏ con chạy soạn thảo nhờ hàm `::DestroyCaret` khi nó mất tiêu điểm vào.

- Một khi con chạy soạn thảo được tạo lập, nó vẫn chưa được hiển thị cho đến khi gọi hàm `ShowCaret`. Nó có thể được dấu đi nhờ gọi `HideCaret`. Nếu gọi lần nữa hàm `HideCaret`, thì số lần gọi `ShowCaret` cũng phải như vậy để nhìn thấy lại con chạy soạn thảo.
- Khi vẽ trong cửa sổ có con chạy soạn thảo từ bên ngoài hàm `OnPaint`, thì con chạy soạn

thảo phải được dấu đi để tránh vẽ chồng. Con chạy soạn thảo có thể được hiển thị lại sau khi vẽ xong.

- Chương trình dịch chuyển con chạy soạn thảo bằng gọi hàm `SetCaretPos`. Windows không tự chuyển con chạy soạn thảo cho ta; trình của ta phải xử lý các thông điệp bàn phím đi đến (cũng có thể là thông điệp của chuột) và xử lý con chạy soạn thảo cho phù hợp. Nếu muốn, vị trí hiện hành của con chạy soạn thảo có thể biết được nhờ hàm `GetCaretPos`.

Hàm quản lý thông điệp sau đây tạo con chạy soạn thảo, đặt vị trí cho chúng, hiển thị chúng khi cửa sổ nhận tiêu điểm vào:

```
void CMainWindow::OnSetFocus (CWnd* pWnd)
{
    CreateSolidCaret (2, m_cyChar);
    SetCaretPos (m_pointCaretPos);
    ShowCaret ();
}
```

Còn hàm quản lý thông điệp sau lưu vị trí con chạy soạn thảo và dấu nó đi, bãi bỏ chúng khi mất tiêu điểm vào:

```
void CMainWindow::OnKillFocus (CWnd* pWnd)
{
    HideCaret ();
    m_pointCaretPos = GetCaretPos ();
    ::DestroyCaret ();
}
```

Trong thí dụ trên, `m_cyChar` lưu độ cao của con chạy soạn thảo, `m_pointCaretPos` lưu vị trí của nó, chúng được lưu lại khi tiêu điểm vào bị mất và được phục hồi khi lấy lại tiêu điểm vào. Vì chỉ có một cửa sổ là có được tiêu điểm vào ở một thời điểm và các thông điệp bàn phím sẽ được hướng tới cửa sổ có tiêu điểm vào, cách tiếp cận đến điều khiển con chạy soạn thảo này đảm bảo rằng cửa sổ nào quản lý bàn phím thì sẽ có con chạy soạn thảo. Nếu tất các chương trình đều ứng xử như vậy thì sẽ không bao giờ có vấn đề là hai chương trình ganh đua để có con chạy soạn thảo vào cùng thời điểm.

Hàm tạo con chạy soạn thảo có hai mục đích: xác định hình dạng con chạy soạn thảo và xác nhận người chủ của nó. Con chạy soạn thảo là *bitmap*, vậy có thể xác định hình dạng bất kỳ bằng cách cung cấp bitmap cho hàm `CWnd::CreateCaret`. Để dễ dàng xác định hình dạng con chạy soạn thảo, hãy để hàm `CreateSolidCaret` làm việc đó cho ta (dễ dàng vì không cần *bitmap* ở đây). `CreateSolidCaret` tạo con chạy soạn thảo khối đen, phụ thuộc vào hình dạng mà ta định nghĩa, có thể là chữ nhật, đường ngang, đứng, hay tương tự. Trong thí dụ `OnSetFocus` trên đây, hàm

```
CreateSolidCaret (2, m_cyChar);
```

tạo con chạy soạn thảo đường đứng rộng hai *pixel* độ cao bằng độ cao của ký tự của co chữ hiện hành (*m_cyChar*). Đây là cách truyền thống để tạo ra con chạy soạn thảo để sử dụng cho co chữ tỷ lệ, trong khi nhiều chương trình lấy độ rộng của con chạy soạn thảo bằng biến hệ thống, thí dụ giá trị *SM_CXBORDER* lấy từ hàm *::GetSystemMetrics*. Với co chữ có độ rộng cố định, nên sử dụng con chạy soạn thảo khối với độ rộng và cao bằng một ký tự, như sau:

```
CreateSolidCaret (m_cxChar, m_cyChar);
```

Con chạy soạn thảo khối không có ý nghĩa với co chữ khoảng cách tỷ lệ bởi vì độ rộng các ký tự khác nhau. Hàm *CreateGrayCaret* của lớp *CWnd* làm việc như hàm *CreateSolidCaret* trừ khi chúng tạo con chạy soạn thảo màu xám thay vì khối đen. Kích thước của con chạy soạn thảo được biểu diễn trong đơn vị logic, vậy nếu ta thay đổi chế độ ánh xạ trước khi gọi hàm tạo con chạy soạn thảo thì kích thước của chúng sẽ biến đổi theo.

Như đề cập trên đây, chính lập trình viên (chứ không phải Windows) dịch chuyển con chạy soạn thảo. Hàm *CWnd::SetCaretPos* đặt vị trí con chạy soạn thảo, chấp nhận đối tượng *CPoint* gói các tọa độ *x, y* của vị trí con chạy mới. Đặt vị trí con chạy soạn thảo trong xâu ký tự tương đối dễ nếu sử dụng co chữ có độ rộng cố định, bởi vì có thể tính giá trị *x* mới trong xâu nhờ nhân vị trí ký tự của con chạy soạn thảo với độ rộng của chúng. Nếu co chữ có khoảng cách tỷ lệ, thì phải làm thêm một số việc. Hàm *CDC::GetTextExtent* và *CDC::GetTabbedTextExtent* của MFC cho phép ứng dụng xác định độ rộng (đơn vị logic) của xâu ký tự đưa ra trong co chữ tỷ lệ. (Hàm *GetTabbedTextExtent* được sử dụng nếu xâu chứa các ký tự *tab*). Cho trước vị trí ký tự, trình có thể xác định vị trí con chạy soạn thảo tương ứng nhờ gọi hàm *GetTextExtent* hoặc *GetTabbedTextExtent* để tính độ rộng của các ký tự cho đến vị trí ký tự. Nếu xâu "Hello, world" được hiển thị vào vị trí xác định bởi biến *point*, các lệnh sau đây đặt con chạy soạn thảo sau ký tự *w* (ký tự thứ tám):

```
CSize size = dc.GetTextExtent ("Hello, w", 8);
SetCaretPos (CPoint (point.x + size.cx, point.y));
```

GetTextExtent cho lại đối tượng *CSize* mà các thành phần *cx, cy* của chúng phản ánh độ rộng, cao của xâu. Đặt vị trí của con chạy soạn thảo vào phía phải của điểm nơi xâu bắt đầu một khoảng *cx* đơn vị thì sau khi gọi hàm này con chạy soạn thảo sẽ ở giữa *w* và *o* trong "world."

Việc đặt con chạy soạn thảo sẽ tương đối khó hơn với co chữ tỷ lệ và không có độ rộng của từng ký tự để làm việc, đây là tình huống gặp phải khi viết hàm *OnLButtonDown* để đặt lại vị trí con chạy soạn thảo khi phím chuột trái bị nhấn. Giả sử ứng dụng quản lý biến *m_nCurrentPos* lưu vị trí hiện hành của ký tự, vị trí trong xâu nơi sẽ xen thêm ký tự. Để dàng tính vị trí con chạy soạn thảo mới khi nhấn phím trượt trái hay phải: làm giảm hay tăng *m_nCurrentPos* và gọi *GetTextExtent* hay *GetTabbedTextExtent* với vị trí mới của ký tự để tính offset mới. Nhưng cái gì xảy ra nếu phím

trái chuột được nhấn vào vị trí tùy ý trong xâu? Chẳng có quan hệ nào giữa điểm nơi nhấn phím chuột và `m_nCurrentPos`, vậy ta phải sử dụng độ chênh lệch giữa vị trí chuột và điểm bắt đầu của xâu để biết vị trí ký tự, sau đó tính vị trí con chạy soạn thảo cuối cùng. Điều này không tránh khỏi việc thực hiện lặp vì cả hàm Windows API cũng như hàm thành phần của lớp MFC đều không chấp nhận xâu, *offset* của *pixel* và cho lại ký tự trên *offset* đó. May thay, thật không quá khó để tự viết hàm này. Ta sẽ xem thực hiện chúng như thế nào.

4.6 ỨNG DỤNG VisualKB

Hãy gộp tất cả những gì đã khảo sát cho đến bây giờ bằng cách phát triển một ứng dụng mẫu, chấp nhận vào văn bản từ bàn phím, hiển thị văn bản trong cửa sổ, và cho phép người sử dụng thực hiện các chức năng soạn thảo văn bản đơn giản bao gồm dịch chuyển con chạy soạn thảo với các phím trượt và chuột. Với mục đích học tập nên hãy thêm khả năng trượt cửa sổ để quan sát các thông điệp bàn phím mà chương trình nhận được và các tham số kèm theo. Chương trình, VisualKB, mô tả một vài kỹ thuật quản lý văn bản có khoảng cách tỷ lệ. Nó cũng là công cụ khảo sát dòng thông điệp đi đến từ bàn phím và xem chi tiết thông điệp nào là do nhấn phím nào hay tổ hợp phím nào phát sinh.

Thông điệp bàn phím							
MFC							
Message	Char	Rep	Scan	Ext	Con	Prv	Tran
WM_KEYDOWN	20	1	58	0	0	0	0
WM_KEYUP	20	1	58	0	0	1	1
WM_KEYDOWN	77	1	50	0	0	0	0
WM_CHAR	77	1	50	0	0	0	0
WM_KEYUP	77	1	50	0	0	1	1
WM_KEYDOWN	70	1	33	0	0	0	0
WM_CHAR	70	1	33	0	0	0	0
WM_KEYUP	70	1	33	0	0	1	1
WM_KEYDOWN	67	1	46	0	0	0	0
WM_CHAR	67	1	46	0	0	0	0
WM_KEYUP	67	1	46	0	0	1	1
WM_SYSKEYDOWN	18	1	56	0	1	0	0

Hình 4.1 Kết quả chạy chương trình VisualKB

Hình 4.1 là cửa sổ VisualKB sau khi khởi động chương trình và nhấn phím của các ký tự M, F, C và phím Alt. Các ký tự nhấn xuất hiện trên chữ nhật ở cửa sổ phía trên, các thông điệp phím tương ứng được hiển thị trong chữ nhật thông điệp phía dưới. Các thông điệp đầu và cuối được phát

sinh khi phím trượt được nhấn và nhả. Ở giữa ta nhìn thấy các thông điệp WM_KEYDOWN, WM_CHAR, và WM_KEYUP phát sinh từ các phím M, F, và C. Chú ý rằng các thông điệp WM_CHAR đến giữa các thông điệp WM_KEYDOWN và WM_KEYUP cho biết phím ký tự được nhấn và nhả. Bên phải mỗi tên thông điệp trong chữ nhật thông điệp, VisualKB hiển thị các tham số thông điệp. Char là mã phím ảo hay mã ký tự chuyển cho hàm xử lý thông điệp trong nChar. Rep là số đếm lần nhắc lại trong nRepCnt. Scan là mã quét OEM lưu trong các bit 0-7 của tham số cờ nFlags, và Ext, Con, Prv, Tran biểu diễn giá trị của cờ phím mở rộng, mã ngữ cảnh, trạng thái phím trước đó, giá trị trạng thái quá độ. VisualKB còn hiển thị các thông điệp WM_SYSKEYDOWN, WM_SYSCHAR, và WM_SYSKEYUP, chúng được kiểm tra bằng nhấn tổ hợp phím với Alt.

Hơn nữa để vào văn bản, ta có thể sử dụng các phím soạn thảo với VisualKB sau đây:

- Các phím trượt trái, phải dịch con chạy soạn thảo đi một ký tự sang trái, phải. *Home* và *End* chuyển con chạy soạn thảo đến đầu hay cuối dòng. Con chạy soạn thảo còn được chuyển đến vị trí mới khi nhấn phím trái chuột.
- Phím *Backspace* xóa ký tự bên trái con chạy soạn thảo và dịch con chạy soạn thảo đi một vị trí sang trái.
- Các phím *Esc* và *Enter* xóa văn bản và khởi động lại vị trí con chạy soạn thảo vào phía trái của dòng.

Các ký tự được nhập vào ở chế độ đè (*overstrike*). Không cần thêm đặc trưng như trượt ngang, chế độ xen ở đây để tránh độ phức tạp không cần thiết của trình. Ngoài ra, thực tế có thể tránh viết lại các lệnh trên nhờ gọi điều khiển soạn thảo (*edit control*) của Windows, chúng cung cấp khả năng tương tự để nhập văn bản bao gồm cả cắt, dán, trượt và nhiều hơn nữa. Trừ khi ta viết riêng ứng dụng xử lý văn bản lớn, còn điều khiển soạn thảo có hầu hết những cái mà ta cần.

4.7 QUẢN LÝ CON CHẠY SOẠN THẢO

Bây giờ hãy khảo sát mã nguồn VisualKB trong chương trình 4.1. VisualKB đăng ký các hàm *OnSetFocus* và *OnKillFocus* trong ánh xạ thông điệp của lớp *CMainWindow* để thông báo lại khi cửa sổ của chúng nhận hay mất tiêu điểm vào. Khi cửa sổ VisualKB nhận tiêu điểm vào, chúng lập con chạy soạn thảo nhờ hàm *CreateSolidCaret*, đặt vị trí con chạy soạn thảo vào điểm lưu trong *m_pointCaretPos* nhờ gọi *SetCaretPos*, và hiển thị con chạy soạn thảo nhờ gọi *ShowCaret*, đúng như mã trình mẫu đã chỉ ra:

```
void CMainWindow::OnSetFocus (CWnd* pWnd)
{
    CreateSolidCaret (2, m_cyChar); // rộng hai đơn vị logic
    SetCaretPos (m_pointCaretPos);
    ShowCaret ();
}
```

Chương trình 4.1 VISUALKB**Tệp VISUALKB.H**

```

#define MAX_STRINGS 12
class CMyApp : public CWinApp
{
public:
    BOOL InitInstance ();
};
//-----
class CMainWindow : public CWnd
{
private:
    int m_cxChar;           // Độ rộng trung bình của ký tự
    int m_cyChar;           // Độ cao của ký tự
    int m_cyLine;           // Độ cao của dòng
    int m_nTextPos;         // Vị trí văn bản hiện hành
    int m_nTabStops[7];     // Tab stops
    int m_nTextLimit;       // Rightmost text extent
    int m_nMsgPos;          // Vị trí trong trường m_messages
    HCURSOR m_hCursorArrow; // Chỉ danh con chuột mũi tên
    HCURSOR m_hCursorIBeam; // Chỉ danh con chuột I-beam
    CPoint m_pointTextOrigin; // Text origin
    CPoint m_pointHeaderOrigin; // Header origin
    CPoint m_pointUpperMsgOrigin; // Origin of top message string
    CPoint m_pointLowerMsgOrigin; // Origin of bottom message string
    CPoint m_pointCaretPos; // Saved caret position
    CRect m_rectTextBox;    // Text-entry rectangle
    CRect m_rectMsgBox;     // Message-string rectangle
    CRect m_rectScroll;     // Scroll rectangle
    CString m_text;         // The input text
    CString m_messages[MAX_STRINGS]; // Message text strings
    int GetNearestPos (CPoint);
    void PositionCaret (CDC* = NULL);
    void DrawInputText (CDC*);
    void Draw3DRectangle (CDC*, CRect&);

```

```

void ShowMessage (CString&, UINT, UINT, UINT);
void DrawMessages (CDC*);
public:
    CMainWindow ();
protected:
    void PostNcDestroy ();
    afx_msg int OnCreate (LPCREATESTRUCT);
    afx_msg void OnPaint ();
    afx_msg void OnSetFocus (CWnd*);
    afx_msg void OnKillFocus (CWnd*);
    afx_msg void OnLButtonDown (UINT, CPoint);
    afx_msg void OnMouseMove (UINT, CPoint);
    afx_msg void OnKeyDown (UINT, UINT, UINT);
    afx_msg void OnKeyUp (UINT, UINT, UINT);
    afx_msg void OnSysKeyDown (UINT, UINT, UINT);
    afx_msg void OnSysKeyUp (UINT, UINT, UINT);
    afx_msg void OnChar (UINT, UINT, UINT);
    afx_msg void OnSysChar (UINT, UINT, UINT);
    DECLARE_MESSAGE_MAP ()
};
//-----

```

Tệp VISUALKB.CPP

```

// VisualKB biểu diễn kỹ thuật quản lý chuột, bàn phím trong ứng dụng nhập
// văn bản và hiển thị các thông điệp phím trên màn hình.
#include <afxwin.h>
#include "visualkb.h"
CMyApp myApp;
// Các hàm thành phần của CMyApp
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}
// Các hàm thành phần và ánh xạ thông điệp của CMainWindow
BEGIN_MESSAGE_MAP (CMainWindow, CWnd)
    ON_WM_CREATE ()

```

```

ON_WM_PAINT ()
ON_WM_SETFOCUS ()
ON_WM_KILLFOCUS ()
ON_WM_LBUTTONDOWN ()
ON_WM_MOUSEMOVE ()
ON_WM_KEYDOWN ()
ON_WM_KEYUP ()
ON_WM_SYSKEYDOWN ()
ON_WM_SYSKEYUP ()
ON_WM_CHAR ()
ON_WM_SYSCHAR ()
END_MESSAGE_MAP ()

//-----
CMainWindow::CMainWindow ()
{
    m_nTextPos = 0;
    m_nMsgPos = 0;
    m_hCursorArrow = myApp.LoadStandardCursor (IDC_ARROW);
    m_hCursorIBeam = myApp.LoadStandardCursor (IDC_IBEAM);
    CString myClass = AfxRegisterWndClass (0, NULL, (HBRUSH) ::GetStockObject (LTGRAY_BRUSH),
        myApp.LoadStandardIcon (IDI_APPLICATION));
    CreateEx (WS_EX_DLGMODALFRAME, myClass, "Thông điệp bàn phím",
        WS_OVERLAPPED | WS_SYSMENU | WS_CAPTION | WS_MINIMIZEBOX, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, NULL);
}

//-----
int CMainWindow::OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    TEXTMETRIC tm;
    CDC* pDC = GetDC ();
    pDC->GetTextMetrics (&tm);
    m_cxChar = tm.tmAveCharWidth;
    m_cyChar = tm.tmHeight;
    m_cyLine = tm.tmHeight + tm.tmExternalLeading;
    ReleaseDC (pDC);
    m_rectTextBox.SetRect (16, 16, (m_cxChar * 64) + 16, (m_cyChar * 2) + 16);
    m_rectMsgBox.SetRect (16, (m_cyChar * 5) + 16, (m_cxChar * 64) + 16,
        (m_cyLine * MAX_STRINGS) + (m_cyChar * 7) + 16);
    m_rectScroll.SetRect (m_cxChar + 16, (m_cyChar * 7) + 16,

```

```

        (m_cxChar * 63) + 16, (m_cyLine * MAX_STRINGS) + (m_cyChar * 6) + 16);
m_pointTextOrigin.x = m_cxChar + 16;
m_pointTextOrigin.y = (m_cyChar / 2) + 16;
m_pointCaretPos = m_pointTextOrigin;
m_nTextLimit = (m_cxChar * 63) + 16;
m_pointHeaderOrigin.x = m_cxChar + 16;
m_pointHeaderOrigin.y = (m_cyChar * 4) + 16;
m_pointUpperMsgOrigin.x = m_cxChar + 16;
m_pointUpperMsgOrigin.y = (m_cyChar * 6) + 16;
m_pointLowerMsgOrigin.x = m_cxChar + 16;
m_pointLowerMsgOrigin.y = (m_cyChar * 6) + (m_cyLine * (MAX_STRINGS - 1)) + 16;
m_nTabStops[0] = (m_cxChar * 24) + 16;
m_nTabStops[1] = (m_cxChar * 30) + 16;
m_nTabStops[2] = (m_cxChar * 36) + 16;
m_nTabStops[3] = (m_cxChar * 42) + 16;
m_nTabStops[4] = (m_cxChar * 46) + 16;
m_nTabStops[5] = (m_cxChar * 50) + 16;
m_nTabStops[6] = (m_cxChar * 54) + 16;
CRect rect (0, 0, m_rectMsgBox.right + 16, m_rectMsgBox.bottom + 16);
CalcWindowRect (&rect);
SetWindowPos (NULL, 0, 0, rect.Width (), rect.Height (),
              SWP_NOZORDER | SWP_NOMOVE | SWP_NOREDRAW);
return 0;
}

//-----
void CMainWindow::PostNcDestroy ()
{
    delete this;
}

//-----
void CMainWindow::OnPaint ()
{
    CPaintDC dc (this);
    static CString header = "Message\\tChar\\tRep\\tScan\\tExt\\tCon\\tPrv\\tTran";
    dc.SetBkMode (TRANSPARENT);
    dc.TabbedTextOut (m_pointHeaderOrigin.x, m_pointHeaderOrigin.y, header, header.GetLength (),
                     sizeof (m_nTabStops), m_nTabStops, m_pointHeaderOrigin.x);
    Draw3DRectangle (&dc, m_rectTextBox);
    Draw3DRectangle (&dc, m_rectMsgBox);
}

```

```

    DrawInputText (&dc);
    DrawMessages (&dc);
}

//-----
void CMainWindow::OnSetFocus (CWnd* pWnd)
{
    CreateSolidCaret (2, m_cyChar);
    SetCaretPos (m_pointCaretPos);
    ShowCaret ();
}

//-----
void CMainWindow::OnKillFocus (CWnd* pWnd)
{
    HideCaret ();
    m_pointCaretPos = GetCaretPos ();
    DestroyCaret ();
}

//-----
void CMainWindow::OnLButtonDown (UINT nFlags, CPoint point)
{
    if (m_rectTextBox.PtInRect (point)) {
        m_nTextPos = GetNearestPos (point);
        PositionCaret ();
    }
}

//-----
void CMainWindow::OnMouseMove (UINT nFlags, CPoint point)
{
    SetCursor (m_rectTextBox.PtInRect (point) ? m_hCursorIBeam : m_hCursorArrow);
}

//-----
void CMainWindow::OnKeyDown (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    CString msg = "WM_KEYDOWN";
    ShowMessage (msg, nChar, nRepCnt, nFlags);
    switch (nChar) {
    case VK_LEFT:
        if (m_nTextPos != 0) {
            m_nTextPos--;

```

```

        PositionCaret ();
    }

    break;

case VK_RIGHT:
    if (m_nTextPos != m_text.GetLength ()) {
        m_nTextPos++;
        PositionCaret ();
    }

    break;

case VK_HOME:
    m_nTextPos = 0;
    PositionCaret ();
    break;

case VK_END:
    m_nTextPos = m_text.GetLength ();
    PositionCaret ();
    break;
}

//-----
void CMainWindow::OnChar (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    CString msg = "WM_CHAR";
    ShowMessage (msg, nChar, nRepCnt, nFlags);
    CClientDC dc (this);
    switch (nChar) {
        case VK_ESCAPE:
        case VK_RETURN:
            m_text.Empty ();
            m_nTextPos = 0;
            break;
        case VK_BACK:
            if (m_nTextPos != 0) {
                m_text = m_text.Left (m_nTextPos - 1) + m_text.Right (m_text.GetLength () - m_nTextPos);
                m_nTextPos--;
            }

            break;
        default:
            if ((nChar >= 0) && (nChar <= 31))

```



```

        return:
    if (m_nTextPos == m_text.GetLength ()) {
        m_text += nChar;
        m_nTextPos++;
    }
    else
        m_text.SetAt (m_nTextPos++, nChar);
    CSize size = dc.GetTextExtent (m_text, m_text.GetLength ());
    if ((m_pointTextOrigin.x + size.cx) > m_nTextLimit) {
        m_text = nChar;
        m_nTextPos = 1;
    }
    break:
}
HideCaret ();
DrawInputText (&dc);
PositionCaret (&dc);
ShowCaret ();
}

//-----
void CMainWindow::OnKeyUp (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    CString msg = "WM_KEYUP";
    ShowMessage (msg, nChar, nRepCnt, nFlags);
    CWnd::OnKeyUp (nChar, nRepCnt, nFlags);
}

//-----
void CMainWindow::OnSysKeyDown (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    CString msg = "WM_SYSKEYDOWN";
    ShowMessage (msg, nChar, nRepCnt, nFlags);
    CWnd::OnSysKeyDown (nChar, nRepCnt, nFlags);
}

//-----
void CMainWindow::OnSysChar (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    CString msg = "WM_SYSCHAR";
    ShowMessage (msg, nChar, nRepCnt, nFlags);
    CWnd::OnSysChar (nChar, nRepCnt, nFlags);
}

```

```

}
//-----
void CMainWindow::OnSysKeyUp (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    CString msg = "WM_SYSKEYUP";
    ShowMessage (msg, nChar, nRepCnt, nFlags);
    CWnd::OnSysKeyUp (nChar, nRepCnt, nFlags);
}
//-----
void CMainWindow::PositionCaret (CDC* pDC)
{
    BOOL bRelease = FALSE;
    if (pDC == NULL) {
        pDC = GetDC ();
        bRelease = TRUE;
    }
    CPoint point = m_pointTextOrigin;
    CString temp = m_text.Left (m_nTextPos);
    point.x += (pDC->GetTextExtent (temp, temp.GetLength ())).cx;
    SetCaretPos (point);
    if (bRelease)
        ReleaseDC (pDC);
}
//-----
int CMainWindow::GetNearestPos (CPoint point)
{
    if (point.x <= m_pointTextOrigin.x) return 0;
    CClientDC dc (this);
    int nLen = m_text.GetLength ();
    if (point.x >= (m_pointTextOrigin.x + (dc.GetTextExtent (m_text, nLen)).cx))
        return nLen;
    int i = 0;
    int nPrevChar = m_pointTextOrigin.x;
    int nNextChar = m_pointTextOrigin.x;
    while (nNextChar < point.x) {
        i++;
        nPrevChar = nNextChar;
        nNextChar = m_pointTextOrigin.x +
            (dc.GetTextExtent (m_text.Left (i), i)).cx;
    }
}

```

```

    }
    return ((point.x - nPrevChar) < (nNextChar - point.x)) ? i - 1 : i;
}

//-----
void CMainWindow::DrawInputText (CDC* pDC)
{
    pDC->SelectStockObject (NULL_PEN);
    pDC->SelectStockObject (LTGRAY_BRUSH);
    pDC->SetBkColor (RGB (192, 192, 192));
    pDC->TextOut (m_pointTextOrigin.x, m_pointTextOrigin.y, m_text);
    pDC->Rectangle (m_pointTextOrigin.x + (pDC->GetTextExtent (m_text, m_text.GetLength ()).cx,
        m_rectTextBox.top + 1, m_rectTextBox.right - 1, m_rectTextBox.bottom - 1);
}

//-----
void CMainWindow::Draw3DRectangle (CDC* pDC, CRect& rect)
{
    CPen* pPen = new CPen (PS_SOLID, 1, RGB (128, 128, 128));
    pDC->SelectObject (pPen);
    pDC->MoveTo (rect.left, rect.bottom - 2);
    pDC->LineTo (rect.left, rect.top);
    pDC->LineTo (rect.right - 1, rect.top);
    delete pDC->SelectStockObject (WHITE_PEN);
    pDC->MoveTo (rect.right - 1, rect.top + 1);
    pDC->LineTo (rect.right - 1, rect.bottom - 1);
    pDC->LineTo (rect.left, rect.bottom - 1);
}

//-----
void CMainWindow::SendMessage (CString& msg, UINT nChar, UINT nRepCnt, UINT nFlags)
{
    CString string;
    string.Format ("%t %u\\t %u\\t %u\\t %u\\t %u\\t %u\\t %u\\t",
        nChar, nRepCnt, nFlags & 0xFF,
        (nFlags >> 8) & 0x01,
        (nFlags >> 13) & 0x01,
        (nFlags >> 14) & 0x01,
        (nFlags >> 15) & 0x01);
    ScrollWindow (0, -m_cyLine, &m_rectScroll);
    ValidateRect (m_rectScroll);
    CClientDC dc (this);
}

```

```

dc.SetBkColor (RGB (192, 192, 192));
m_messages[m_nMsgPos] = msg + string;
dc.TabbedTextOut (m_pointLowerMsgOrigin.x, m_pointLowerMsgOrigin.y,
                  m_messages[m_nMsgPos], m_messages[m_nMsgPos].GetLength (),
                  sizeof (m_nTabStops), m_nTabStops, m_pointLowerMsgOrigin.x);
if (++m_nMsgPos == MAX_STRINGS)    m_nMsgPos = 0;
|
//-----
void CMainWindow::DrawMessages (CDC* pDC)
{
    int pos = m_nMsgPos;
    for (int i=0; i<MAX_STRINGS; i++) {
        pDC->TabbedTextOut (m_pointUpperMsgOrigin.x, m_pointUpperMsgOrigin.y + (m_cyLine * i),
                           m_messages[pos], m_messages[pos].GetLength (), sizeof (m_nTabStops),
                           m_nTabStops, m_pointUpperMsgOrigin.x);
        if (++pos == MAX_STRINGS)    pos = 0;
    }
|
//-----

```

Khi thông điệp WM_KILLFOCUS đến thì hàm OnKillFocus dấu con chạy soạn thảo, lưu vị trí hiện hành sao cho chúng được khôi phục ở lần gọi OnSetFocus sau, cuối cùng là bãi bỏ con chạy soạn thảo.

```

void CMainWindow::OnKillFocus (CWnd* pWnd)
{
    HideCaret ();
    m_pointCaretPos = GetCaretPos ();
    ::DestroyCaret ();
|

```

Hàm OnCreate sẽ nạp tọa độ của ký tự bên trái nhất trong trường nhập văn bản của VisualKB vào m_pointCaretPos. Chúng sẽ được khởi động lại đúng vào vị trí con chạy soạn thảo hiện hành mỗi khi cửa sổ mất tiêu điểm vào. Vì vậy việc gọi hàm SetCaretPos trong OnSetFocus là để đặt con chạy soạn thảo vào đầu trường nhập văn bản khi trình lần đầu tiên kích hoạt, và khôi phục vào vị trí mà nó chiếm trước đó trong các lần kích hoạt tiếp theo.

Hàm OnKeyDown dịch con chạy soạn thảo khi các phím trượt trái, phải, Home, và End bị nhấn. Không phím nào trong đó phát sinh thông điệp WM_CHAR, vậy ta phải xử lý thông điệp WM_KEYDOWN nếu ta biết khi nào nó bị nhấn. Cấu trúc *switch-case* thực hiện trình tự quản lý tương ứng trên cơ sở mã phím ảo chuyển cho nChar. Hàm quản lý phím trượt trái (mã phím ảo của

nó là VK_LEFT) bao gồm các dòng lệnh sau đây:

```
case VK_LEFT:
    if (m_nTextPos != 0) {
        m_nTextPos--;
        PositionCaret ();
    }
    break;
```

Không có gì bí hiểm ở đây. `m_nTextPos` là vị trí nơi ký tự tiếp theo sẽ xen vào chuỗi văn bản. Chuỗi văn bản tự nó được lưu trong `m_text` của đối tượng `CString`. Hàm `PositionCaret` là hàm thành phần của `CMainWindow` sử dụng hàm `GetTextExtent` để tìm vị trí *pixel* trong chuỗi văn bản tương ứng với vị trí ký tự lưu trong `m_nTextPos`, sau đó dịch con chạy soạn thảo đến vị trí đó nhờ hàm `SetCaretPos`. Sau khi kiểm tra `m_nTextPos` để đảm bảo nó không chạy quá khi dịch con chạy soạn thảo sang trái, hàm quản lý VK_LEFT giảm `m_nTextPos` và gọi `PositionCaret` để chuyển con chạy soạn thảo. Nếu `m_nTextPos` là 0, cho biết con chạy soạn thảo đã sẵn sàng đặt vào tận cùng bên trái của trường vào, “gõ phím” bị bỏ qua. Các hàm quản lý VK_XXX khác thực hiện tương tự. Thí dụ, hàm quản lý VK_END chuyển con chạy soạn thảo đến cuối chuỗi văn bản nhờ dòng lệnh

```
m_nTextPos = m_text.GetLength ();
PositionCaret ();
```

`GetLength` là hàm thành phần của `CString`, chúng cho lại độ dài của chuỗi. Sử dụng đối tượng `CString` để lưu văn bản vào `VisualKB` làm cho việc quản lý văn bản đơn giản hơn nhiều và thuận tiện hơn so với khi chuỗi chỉ được quản lý như là trường các ký tự. Thí dụ, tất cả những gì mà hàm điều khiển `OnChar` phải làm khi thêm ký tự mới đến cuối chuỗi, sẽ được thực hiện như sau:

```
m_text += nChar;
```

Khi nói đến quản lý chuỗi thì có lẽ không có gì đơn giản hơn. Trong tệp `VISUALKB.CPP` có một số hàm thành phần và toán tử của đối tượng `CString`, bao gồm `Left`, cho lại đối tượng `CString` chứa x ký tự bên trái của chuỗi; `Right`, cho lại y ký tự bên phải nhất; và `Format`, thực hiện tạo dạng chuỗi như `printf`.

Thật đáng tiếc nếu `VisualKB` không làm gì với chuột, vậy ta sẽ thêm hàm `OnLButtonDown` để có thể dịch con chạy soạn thảo nhờ nhấn phím trái chuột trong mảng nhập văn bản. Điều này cho cơ hội khảo sát hàm nhận điểm nơi nhấn phím chuột làm tham số và cho lại vị trí ký tự tương ứng trong chuỗi văn bản.

```
void CMainWindow::OnLButtonDown (UINT nFlags, CPoint point)
{
    if (m_rectTextBox.PtInRect (point)) {
        m_nTextPos = GetNearestPos (point);
        PositionCaret ();
    }
```

Trong đó, `m_rectTextBox` là chữ nhật bao quanh mảng văn bản. Sau khi gọi hàm `CRect::PInRect` để xác định xem việc nhấn phím xảy ra bên trong chữ nhật hay không (và trở lại mà không làm gì nếu không nhấn phím), `OnLButtonDown` tính toán giá trị mới cho `m_nTextPos` bằng `GetNearestPos` và gọi hàm *friend* có tên `PositionCaret` để đặt lại con chạy soạn thảo cho phù hợp. `GetNearestPos`, là thành phần của lớp `CMainWindow`, thực hiện phần lớn các công việc. `GetNearestPos` trước hết kiểm tra xem chuột có bị nhấn vào bên trái của chuỗi ký tự hay không và cho lại vị trí mới của ký tự là 0 nếu có nhấn:

```
if (point.x <= m_pointTextOrigin.x)
    return 0;
```

`m_pointTextOrigin` giữ tọa độ của góc trên trái chuỗi ký tự; đó là các tọa độ chuyển cho hàm `TextOut` khi chuỗi ký tự được vẽ trên màn hình. Tiếp theo, hàm `GetNearestPos` cho lại giá trị nguyên bằng độ dài chuỗi nếu chuột đã bị nhấn bên ngoài phía phải của chuỗi:

```
CClientDC dc (this);
int nLen = m_text.GetLength ();
if (point.x >= (m_pointTextOrigin.x + (dc.GetTextExtent (m_text, nLen)).cx))
    return nLen;
```

Kết quả? Nếu chuột được nhấn trong chữ nhật văn bản nhưng bên phải của ký tự phía phải nhất hiển thị được thì con chạy soạn thảo được dịch đến cuối chuỗi.

Nếu `GetNearestPos` không thực hiện lệnh *return nLen* trên đây, thì con chạy đã được nhấn bên trong mảng nhập văn bản, ở vị trí nào đó trong khoảng giữa đầu trái và phải của chuỗi. Tiếp theo, `GetNearestPos` khởi động 3 biến và thực hiện vòng lặp gọi `GetTextExtent` cho đến khi `nPrevChar` và `nNextChar` lưu các giá trị tọa độ x của điểm nơi nhấn phím:

```
while (nNextChar < point.x) {
    i++;
    nPrevChar = nNextChar;
    nNextChar = m_pointTextOrigin.x +
        (dc.GetTextExtent (m_text.Left (i), i)).cx;
}
```

Khi qua vòng lặp, `i` chứa số vị trí ký tự ở phía phải của nơi nhấn phím, và `i-1` chứa vị trí của ký tự tính từ phía trái. Bây giờ, tìm vị trí ký tự đơn giản là việc tìm ra xem `point.x` gần `nNextChar` hay gần `nPrevChar` hơn và cho lại `i` hay `i-1`.

```
return ((point.x - nPrevChar) < (nNextChar - point.x)) ? i - 1 : i;
```

Cho trước điểm mồi trong vùng làm việc của cửa sổ, hàm `GetNearestPos` cho lại vị trí phù hợp của ký tự trong chuỗi `m_text`. Cài đặt này không có hiệu quả cao vì phải gọi nhiều lần hàm

GetTextExtent. (Vòng lặp khởi đầu với ký tự bên trái nhất trong xâu và dịch phải qua mỗi ký tự một lần cho đến khi tìm thấy ký tự nằm ngay bên phải của điểm nơi nhấn chuột). Cài đặt thực sự thông minh của hàm GetNearestPos có thể thực hiện tốt hơn bằng sử dụng cách tiếp cận nửa nhị phân, bắt đầu từ giữa xâu ký tự và lặp sang trái hay phải với tổng số ký tự bằng nửa vùng mà nó chưa được phủ cho đến ký tự bên trái và phải của điểm nơi nhấn phím. Vị trí ký tự trong xâu dài 128 ký tự có thể được xác định chỉ bằng dưới tám lần gọi GetTextExtent. Còn cách tiếp cận có sử dụng GetNearestPos cần đến 128 lần gọi.

4.8 NHẬP VÀ SOẠN THẢO VĂN BẢN

Toàn bộ mã trình thực hiện nhập, soạn thảo xâu hiển thị trong mảng nhập văn bản (và lưu ở `m_text`) được mô tả trong hàm `CMainWindow::OnChar`. Chiến lược xử lý của `OnChar` được tóm tắt như sau:

- Hiển thị thông điệp trên màn hình
- Sửa đổi xâu văn bản nhờ mã ký tự trong `nChar`
- Vẽ xâu văn bản đã sửa trong màn hình
- Đặt lại con chạy soạn thảo

Bước một bao gồm gọi hàm `CMainWindow::ShowMessage`, chúng ta sẽ xem xét kỹ hơn sau. Xâu văn bản được sửa đổi như thế nào trong bước hai phụ thuộc vào mã ký tự lưu trong `nChar` là gì. Nếu ký tự là `escape` hay cho lại `VK_ESCAPE` hoặc `VK_RETURN`, `m_text` được xoá bằng hàm `Empty` (hàm khác của lớp `CString`) và nạp 0 vào `m_nTextPos`. Nếu ký tự là `Backspace` (`VK_BACK`) và `m_nTextPos` khác 0, thì ký tự tại `m_nTextPos-1` bị xoá và giảm `m_nTextPos`. Nếu ký tự có giá trị trong khoảng 0 đến 31, thì chúng được bỏ qua. Làm việc này để lọc bỏ các ký tự điều khiển để khỏi hiển thị các ký tự vô nghĩa. (Không may là việc này lọc luôn cả ký tự `tab`. Như vậy là không thực tế, nhưng để đơn giản mã trình của `VisualKB`). Nếu `nChar` chứa bất kỳ ký tự nào khác thì nó được thêm vào `m_text` ở vị trí ký tự hiện hành và tăng `m_nTextPos` cho phù hợp.

Bây giờ, `m_text` được sửa đổi bằng ký tự vừa nhập, hàm `OnChar` dấu con chạy soạn thảo và tiếp tục bước ba để hiển thị xâu đã sửa đổi. Chúng bao gồm gọi hàm `CMainWindow::DrawInputText`, sử dụng hàm `TextOut` để ra văn bản. Trước khi gọi `TextOut`, `DrawInputText` lập màu nền của DC thành xám sáng (`RGB(192, 192, 192)`) sao cho các *pixel* sau các ký tự ra bằng `TextOut` sẽ có màu xám như màu nền cửa sổ thay vì màu trắng mặc định. (Lập chế độ nền văn bản thành `TRANSPARENT` nhờ `SetBkMode` sẽ không tốt hơn bởi vì các *pixel* đen từ xâu các ký tự cũ sẽ được hiện ra). Sau khi `TextOut` trở lại, `DrawInputText` thực hiện một vài điều ngạc nhiên: bút `NULL` và chổi xám sáng được chọn cho DC, chúng vẽ chữ nhật có cạnh là từ phía phải nhất của xâu ký tự đến sườn phải của chữ nhật bao quanh mảng vào văn bản. Việc này xoá tất cả các *pixel* đen cho đến phía phải của xâu ký tự, nó trở nên quan trọng nếu `m_text` ngắn hơn trước đó, thí dụ khi ký tự bị

xoá do nhấn *Backspace*. Thật lãng phí khi xoá mọi cái cho đến phía phải của xâu ký tự mỗi khi nó được hiển thị, nhưng tiếp cận này làm đơn giản logic lập trình bằng cách chỉ cần gọi một hàm để vẽ chính xác xâu trên màn hình mà không quan tâm đến cái gì có trước đó. Và xoá mọi thứ ở phía phải của xâu sẽ hạn chế nhấp nháy có thể xảy ra nếu ta thực hiện việc xoá toàn bộ mảng nhập văn bản và vẽ đề lên mỗi khi gọi hàm `DrawInputString`.

`OnChar` kết thúc bằng gọi `PositionCaret` để đặt lại con chạy soạn thảo sử dụng giá trị trong `m_nTextPos` và `ShowCaret` để hiển thị lại con chạy soạn thảo. Để xem cái gì xảy ra nếu ta không dấu con chạy soạn thảo trước khi cập nhật cửa sổ, thử loại các hàm `HideCaret` và `ShowCaret` của `OnChar`, dịch lại chương trình rồi gõ vài ký tự. Khi quan sát kết quả, chắc chắn ta thấy sự cần thiết của việc dấu và hiện con chạy soạn thảo, thậm chí đôi khi thấy con chạy soạn thảo nhấp nháy.

4.9 QUẢN LÝ CON CHẠY CHUỘT

Giả sử ta chuyển dịch con chạy chuột (*cursor*) vòng quanh trong `VisualKB`, chú ý rằng nó thay đổi từ mũi tên khi ở ngoài mảng nhập văn bản thành I-beam khi ở trong chúng. Vấn đề này tương đối dễ thực hiện. Cấu tử của `CMainWindow` sử dụng `CWinApp::LoadStandardCursor` để lấy được một cặp chỉ danh con chạy, nó được lưu vào `m_hCursorArrow` và `m_hCursorIBeam`:

```
m_hCursorArrow = myApp.LoadStandardCursor (IDC_ARROW);
m_hCursorIBeam = myApp.LoadStandardCursor (IDC_IBEAM);
```

Sau đó nó sử dụng `AfxRegisterWndClass` để đăng ký lớp cửa sổ với con chạy `NULL` và tạo cửa sổ `VisualKB` từ lớp:

```
CString myClass = AfxRegisterWndClass (0, NULL, (HBRUSH) ::GetStockObject (LTGRAY_BRUSH),
                                     myApp.LoadStandardIcon (IDI_APPLICATION));
CreateEx (WS_EX_DLGMODALFRAME, myClass, "Visual Keyboard",
          WS_OVERLAPPED | WS_SYSMENU | WS_CAPTION | WS_MINIMIZEBOX,
          CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
          CW_USEDEFAULT, NULL, NULL, NULL);
```

Hàm `OnMouseMove` hoàn thiện công việc bằng lời gọi `::SetCursor` mỗi khi dịch chuột để lập con chạy thành `m_hCursorIBeam` hay `m_hCursorArrow`, phụ thuộc vào vị trí con chạy:

```
void CMainWindow::OnMouseMove (UINT nFlags, CPoint point)
{
    ::SetCursor (m_rectTextBox.PtInRect (point) ? m_hCursorIBeam : m_hCursorArrow);
}
```

Kết quả? Khi con chạy đến gần văn bản nơi ta đang gõ vào ứng dụng, chúng đổi thành I-beam, do vậy có thể định vị chính xác con chạy soạn thảo trước khi di chuyển nó bằng chuột. Bên ngoài chữ nhật văn bản nó lại trở lại hình dạng thông thường - mũi tên. Và bởi vì con chạy trong lớp là

NULL, ta không nhìn thấy nhấp nháy khi chuyển chuột.

Để thấy rõ ý nghĩa của “nháy”, hãy thử thay đổi chỉ số điều khiển con chạy NULL khi gọi `AfxRegisterWndClass` thành `myApp.LoadStandardCursor (IDC_ARROW)` rồi dịch lại chương trình. Khi dịch con chạy qua mảng nhập văn bản, sẽ thấy chúng nhấp nháy qua lại giữa mũi tên và I-beam khi Windows vẽ chúng và hàm `::SetCursor` cũng vẽ lại chúng mỗi khi dịch chuột. Không cần phải nói, người sử dụng sẽ thấy khó chịu. Việc đăng ký con chạy lớp NULL khi ta định thay đổi con chạy trong hàm `OnMouseMove` sẽ tránh được vấn đề này.

4.10 CÁC ĐIỂM CẦN QUAN TÂM

VisualKB gọi hàm `CMainWindow::ShowMessage` để hiển thị lại thông điệp trên cửa sổ màn hình. `ShowMessage` lập khuôn mẫu xâu ra mới nhờ gọi hàm `CString::Format`, sao chép chúng vào nơi ít được sử dụng nhất trong mảng `m_messages`, trượt nội dung hiện hành của cửa sổ lên một dòng, và gọi `CDC::TabbedTextOut` để hiển thị xâu thông điệp mới ở dòng cuối. `TabbedTextOut` được sử dụng thay cho `TextOut` để tạo cột trên đầu ra. (Nếu thiếu các ký tự *tab*, sẽ khó, hầu như không thể thực hiện việc đặt các ký tự của cơ chữ tỷ lệ thành cột). Các thiết đặt *tab stop* được khởi tạo trong hàm `OnCreate` sử dụng các giá trị trên cơ sở độ rộng trung bình của ký tự được lưu trong mảng `m_nTabStops`, địa chỉ của chúng được truyền cho `TabbedTextOut` mỗi khi được gọi. Các xâu thông điệp lưu trong mảng `m_messages` do vậy hàm `OnPaint` có thể vẽ lại các thông điệp trên cửa sổ. Thành phần dữ liệu `m_nMsgPos` của `CMainWindow` đánh dấu vị trí trong mảng nơi sẽ sao chép xâu tiếp theo vào. Chúng được tăng mỗi khi gọi `ShowMessage` (nó lại trở về 0 khi tới giới hạn của mảng) sao cho `m_messages` luôn chứa 12 thông điệp bàn phím cuối cùng được hiển thị ra màn hình.

Các hàm điều khiển thông điệp như `OnSysKeyDown` và `OnSysKeyUp` trong VisualKB chỉ để hiển thị lại các thông điệp bàn phím, phải gọi hàm điều khiển lớp cơ sở tương ứng trước khi trở lại:

```
void CMainWindow::OnSysKeyDown (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    //...
    CWnd::OnSysKeyDown (nChar, nRepCnt, nFlags);
}
```

Nếu không làm như vậy thì các lệnh chuẩn như *Alt-Spacebar* và *Alt-Tab* sẽ không làm việc nữa. Đây là điểm vô cùng quan trọng. Khi ta bắt thông điệp, phải truyền toàn bộ những gì mà ta không xử lý cho lớp cơ sở. Chúng đặc biệt quan trọng đối với các thông điệp phím hệ thống và các thông điệp chuột ngoài vùng làm việc, bởi vì cả hai đóng vai trò quan trọng làm cho cửa sổ ứng dụng có thể ứng xử như mọi cửa sổ khác trong màn hình.

Hàm `CMainWindow::Draw3DRectangle` là điểm quan tâm cuối cùng trong mã nguồn của trình

VisualKB, chúng vẽ chữ nhật bao quanh mảng nhập xâu ký tự và hiển thị thông điệp bàn phím. Đây không phải chữ nhật thông thường; chúng xuất hiện như 3D. Cho trước con trỏ đến ngữ cảnh thiết bị và qui chiếu đến đối tượng CRect mô tả chữ nhật, Draw3DRectangle vẽ chữ nhật 3D bằng đường biên màu xám đen (*RGB (128, 128, 128)*) ở phía trái và trên và đường trắng nhờ bút WHITE_PEN ở phía phải và dưới. Điều này làm “ánh sáng” chiếu trên cửa sổ đi từ phía trên, trái, chúng phù hợp với mô hình ánh sáng của Windows 95/98 dành cho biên cửa sổ.

4.11 CÁC VẤN ĐỀ KHÁC CỦA BÀN PHÍM

Có hai thông điệp bàn phím vẫn chưa được bàn luận tới bởi vì chúng ít được sử dụng. Nhiều bàn phím Windows quốc tế cho phép người sử dụng nhập ký tự có dấu bằng cách gõ phím “chết” biểu diễn dấu sau ký tự của nó. Hàm ::TranslateMessage dịch thông điệp WM_KEYUP tương ứng phím “chết” thành thông điệp WM_DEADCHAR (và thông điệp WM_SYSKEYUP sinh ra bởi phím “chết” thành WM_SYSDEADCHAR) và đặt chúng vào hàng đợi thông điệp của ứng dụng. Windows cung cấp logic tổ hợp các thông điệp này với thông điệp ký tự để sản sinh ký tự có dấu, vậy thông điệp phím “chết” thường được truyền cho xử lý mặc định. Nhưng một số ứng dụng nhân từng thông điệp phím “chết” và hiển thị dấu tương ứng. Sau đó gõ phím tiếp theo sẽ thay thế dấu bằng ký tự có dấu. Điều này làm người sử dụng dễ chịu hơn do tránh việc gõ phím “chết” như là phím “mù”.

Ứng dụng có thể điều khiển tốc độ nhấp nháy của con chạy soạn thảo nhờ hàm của Windows ::SetCaretBlinkTime và có thể lấy giá trị hiện hành nhờ hàm ::GetCaretBlinkTime. Lệnh:

```
::SetCaretBlinkTime (500);
```

nạp khoảng nháy là 500ms. Vì con chạy soạn thảo là tài nguyên chia sẻ, ứng dụng cần lưu thời gian nháy trước khi thay đổi nó và khôi phục giá trị gốc trước khi mất tiêu điểm vào. Tốt nhất là không nên thay đổi chúng và sử dụng giá trị được lập ngoài chương trình, trong *Control Panel*.

Lập trình viên thường sử dụng bộ ký tự ASCII, thông thường sử dụng các hàm C như *toupper* và *tolower* để chuyển đổi từ ký tự thường sang ký tự hoa và ngược lại. Đôi khi sử dụng “mẹo” để chuyển bằng cách xoá các bit 5 và 8 của mã ký tự như sau:

```
ch &= 0xDF; // Chuyển ch sang chữ hoa
ch |= 0x20; // Chuyển ch sang chữ nhỏ
```

Các cách này đều không hay cho chương trình Windows. Chúng có thể làm việc tốt với các ký tự theo qui định, nhưng không làm việc với các ký tự có dấu như ký tự quốc tế. Để thực hiện chuyển đổi này, hãy dùng hàm của Windows::AnsiUpper và ::AnsiLower hay hàm MakeLower và MakeUpper của CString. Điều này tránh được lỗi khi ứng dụng sử dụng tập mã quốc tế.

CHƯƠNG 5

THỰC ĐƠN, THANH CÔNG CỤ VÀ THANH TRẠNG THÁI

Thực đơn đẩy xuống (*pull-down*) có thể là thành phần giao diện được chấp nhận rộng rãi nhất trên thế giới. Hầu như bất kỳ ai ngồi vào trước máy tính đều thấy thực đơn, đồng thời đều biết rằng việc nhấn phím vào mục nào đó trên thanh thực đơn sẽ hiển thị danh sách lệnh.

Trong ứng dụng Windows® 95/98, thực đơn thường được định nghĩa từ bên ngoài chương trình, tại mẫu tài nguyên (*resource template*) và được nạp khi chương trình chạy, tuy nhiên thực đơn cũng có thể được tự động tạo ra khi chương trình chạy. Khi chọn mục của thực đơn, Windows 95/98 gửi cho cửa sổ chủ thực đơn đó thông điệp WM_COMMAND với từ thấp của WPARAM chứa chỉ danh ID (là số nguyên) của mục. Trong ứng dụng MFC, hàm thành phần tương ứng tự động kích hoạt thông qua ánh xạ thông điệp. Khung ứng dụng của MFC làm tăng sức mạnh mẫu thực đơn-thông điệp bằng cách cung cấp cơ chế dễ sử dụng để đồng bộ các mục thực đơn với thông tin trạng thái do ứng dụng quản lý, dẫn đường cho các thông điệp lệnh giữa các đối tượng khác nhau trong ứng dụng tài liệu/quan sát (*document/view*). Hơn nữa các đối tượng xây dựng từ lớp CToolBar và CStatusBar của MFC dễ tích hợp với thực đơn để cung cấp phím nhấn tương đương cho các lệnh chung và văn bản trợ giúp cho các mục thực đơn. Một khi đã làm thực đơn theo MFC, ta sẽ không còn muốn làm theo cách cũ nữa.

Chương này trình bày cho lập trình viên mới kể cả những ai đã có kinh nghiệm một phương pháp quản lý thực đơn trong ngữ cảnh của MFC và khung ứng dụng. Ta sẽ khảo sát việc lập thực đơn và xử lý chọn mục thực đơn, viết trình điều khiển cho mục thực đơn, cung cấp phím lệnh cấp tốc (*keyboard shortcuts - accelerators*) cho mục thực đơn, tăng cường ghép nối thực đơn với thanh công cụ và thanh trạng thái; và tạo thực đơn ngữ cảnh bằng nhấn phím phải. Có nhiều vấn đề phải bàn, ta hãy đi lần lượt. Trước hết, hãy xem xét việc tạo thực đơn cơ bản và cách nối nó vào ứng dụng.

5.1 THỰC ĐƠN VÀ TÀI NGUYÊN THỰC ĐƠN

Cách dễ nhất để định nghĩa thực đơn là thêm mẫu thực đơn vào tệp RC của ứng dụng. Một khi được dịch, mẫu sẽ trở thành tài nguyên nhị phân được liên kết vào tệp EXE và được tham chiếu bằng tên hay chỉ danh ID tài nguyên.

Khi xây dựng lõi ứng dụng bằng Visual C++™, AppWizard sẽ tạo mẫu thực đơn tương tự như dưới đây. Trong đó, IDR_MAINFRAME là ID (số nguyên) của tài nguyên. Ta có thể chỉ ra xâu

hay chỉ danh ID nguyên bất kỳ mà ta muốn. PRELOAD và DISCARDABLE xác định các thuộc tính của thực đơn.

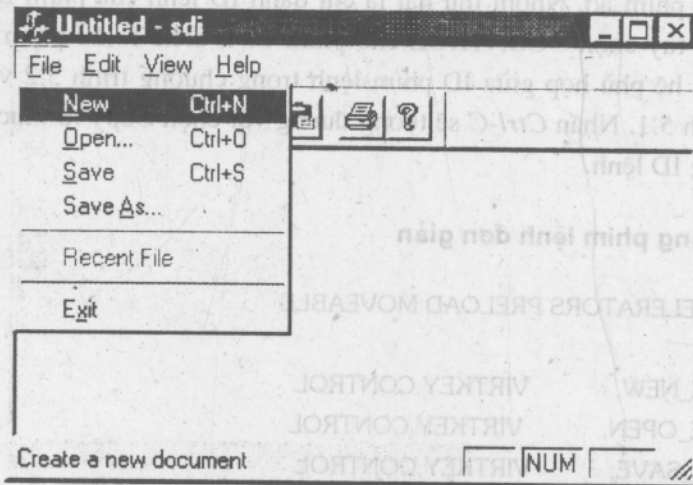
Chương trình 5.1 Mẫu thực đơn đơn giản

```
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&New\tCtrl+N",      ID_FILE_NEW
    MENUITEM "&Open...\tCtrl+O",  ID_FILE_OPEN
    MENUITEM "&Save\tCtrl+S",      ID_FILE_SAVE
    MENUITEM "Save &As...",       ID_FILE_SAVE_AS
    MENUITEM SEPARATOR
    MENUITEM "Recent File",       ID_FILE_MRU_FILE1,GRAYED
    MENUITEM SEPARATOR
    MENUITEM "E&xit",             ID_APP_EXIT
  END
  POPUP "&Edit"
  BEGIN
    MENUITEM "&Undo\tCtrl+Z",      ID_EDIT_UNDO
    MENUITEM SEPARATOR
    MENUITEM "Cu&t\tCtrl+X",       ID_EDIT_CUT
    MENUITEM "&Copy\tCtrl+C",      ID_EDIT_COPY
    MENUITEM "&Paste\tCtrl+V",     ID_EDIT_PASTE
  END
  POPUP "&View"
  BEGIN
    MENUITEM "&Toolbar",          ID_VIEW_TOOLBAR
    MENUITEM "&Status Bar",      ID_VIEW_STATUS_BAR
  END
  POPUP "&Help"
  BEGIN
    MENUITEM "&About MyApp...",   ID_APP_ABOUT
  END
END
```

Các lệnh giữa BEGIN và END định nghĩa nội dung thực đơn. Các lệnh POPUP chỉ ra mục thực đơn mức đỉnh (*top-level*) - là các mục xuất hiện trong thanh thực đơn trên đỉnh cửa sổ. Mỗi mục trong thực đơn mức đỉnh là một thực đơn khác, gọi là thực đơn *pop-up*. Lệnh MENUITEM định

nghĩa các mục thực đơn của thực đơn *pop-up*. Các xâu trong dấu nháy định nghĩa tên của mục thực đơn, ký tự "&" xác định phím lệnh.

Ký tự \t tách xâu ký tự của mục thực đơn ra khỏi tên phím lệnh, chúng sẽ xen ký tự tab sao cho phím lệnh được chỉnh hàng trong thực đơn. Đôi khi lập trình viên sử dụng \a để căn phải xâu ký tự phím lệnh. Hình 5.1 là thực đơn sinh ra từ mẫu trong chương trình 5.1 khi chọn thực đơn File.



Hình 5.1 Thực đơn tệp đơn giản

Các giá trị ID_XXX đi sau xâu mục thực đơn trong lệnh MENUITEM là ID của lệnh. Chúng là số nguyên được tải vào từ thập của thông điệp WM_COMMAND khi người sử dụng chọn mục thực đơn. Mỗi mục thực đơn được gán ID duy nhất có giá trị từ 1, đến 0xDFFF. Lập trình viên MFC thường sử dụng lệnh #define để định nghĩa chỉ danh ID của mục thực đơn trong RESOURCE.H. Các chỉ danh ID trong chương trình 5.1 đã được định nghĩa trong tệp AFXRES.H của MFC, vậy ta không cần phải tự định nghĩa chúng. Sự thật, nếu ta gộp AFXRES.H vào tệp RC và thêm cả các lệnh #define tương ứng với bất kỳ ID nào định nghĩa trước, chương trình ứng dụng sẽ không dịch được. Để tránh đụng độ này, lập trình viên MFC thường phải thêm tiền tố IDM_ vào các mục thực đơn tự định nghĩa. Một khi không muốn sử dụng ID định nghĩa sẵn thì đừng gộp tệp AFXRES.H vào chương trình.

Khi định nghĩa mục thực đơn bằng lệnh MENUITEM, ta có thể chỉ ra trạng thái khởi đầu của mục. Từ khoá GRAYED đi kèm mục thực đơn *File/Recent File* trong hình 5.1 làm "xám" mục thực đơn, làm cho nó không chọn được. Từ khoá hay dùng khác nữa là CHECKED, chúng đặt dấu tích (✓) bên cạnh mục thực đơn. Các ứng dụng Windows viết bằng C với công cụ phát triển phần mềm của Microsoft (*Software Development Kit - SDK*) thường hay sử dụng kỹ thuật này, nhưng

trong ứng dụng MFC lại không vì cơ chế cập nhật giao diện với người dùng (*User Interface - UI*) của khung ứng dụng quản lý tự động trạng thái các mục của thực đơn. Dưới đây ta sẽ xem xét kỹ hơn về cập nhật UI của MFC.

Khi *AppWizard* sinh thực đơn mặc định, chúng cũng tạo ra bảng phím lệnh cấp tốc đi kèm theo. Bảng phím lệnh do *AppWizard* phát sinh cho thực đơn trong chương trình 5.1 được nhắc lại trong chương trình 5.2. Mỗi dòng trong bảng chỉ ra một phím hay tổ hợp phím lệnh. Nhóm thứ nhất trong mỗi dòng là mã phím ảo. Nhóm thứ hai là chỉ danh ID lệnh của phím lệnh cấp tốc. Sau từ khoá *VIRTKEY* là từ tùy chọn - *CONTROL* cho phím *Ctrl*, *SHIFT* cho phím *Shift*, và *ALT* cho phím *Alt*. Chú ý quan hệ phù hợp giữa ID phím lệnh trong chương trình 5.2 và ID của mục thực đơn trong chương trình 5.1. Nhấn *Ctrl-C* sẽ tương đương với chọn *Copy* từ thực đơn *Edit* bởi vì cả hai đều được gán cùng ID lệnh.

Chương trình 5.2 Bảng phím lệnh đơn giản

```
IDR_MAINFRAME ACCELERATORS PRELOAD MOVEABLE
BEGIN
    "N",      ID_FILE_NEW,      VIRTKEY,CONTROL
    "O",      ID_FILE_OPEN,     VIRTKEY,CONTROL
    "S",      ID_FILE_SAVE,     VIRTKEY,CONTROL
    "Z",      ID_EDIT_UNDO,     VIRTKEY,CONTROL
    "X",      ID_EDIT_CUT,      VIRTKEY,CONTROL
    "C",      ID_EDIT_COPY,     VIRTKEY,CONTROL
    "V",      ID_EDIT_PASTE,    VIRTKEY,CONTROL
    VK_BACK,  ID_EDIT_UNDO,     VIRTKEY,ALT
    VK_DELETE, ID_EDIT_CUT,     VIRTKEY,SHIFT
    VK_INSERT, ID_EDIT_COPY,    VIRTKEY,CONTROL
    VK_INSERT, ID_EDIT_PASTE,   VIRTKEY,SHIFT
    VK_F6,    ID_NEXT_PANE,     VIRTKEY
    VK_F6,    ID_PREV_PANE,     VIRTKEY,SHIFT
END
```

5.2 NẠP THỰC ĐƠN VÀ PHÍM LỆNH CẤP TỐC

Trước khi thực đơn xuất hiện trong ứng dụng, chúng phải được nạp và gán vào cửa sổ. Việc này thông thường được thực hiện trong một bước bằng cách chuyển chỉ danh ID của thực đơn cho hàm *Create* mà nó được gọi khi tạo cửa sổ, nhưng nó cũng có thể được thực hiện bằng cách xây dựng đối tượng *CMenu*, nạp thực đơn nhờ *CMenu::LoadMenu*, và gán nó vào cửa sổ nhờ *CWnd::SetMenu*. Lệnh sau đây tạo cửa sổ khung trên cơ sở lớp *CFrameWnd* của MFC và nạp, gán tự động vào thực đơn có chỉ danh ID tài nguyên là *IDR_MAINFRAME*:

```
Create (NULL, "My Application", WS_OVERLAPPEDWINDOW,
    rectDefault, NULL, MAKEINTRESOURCE (IDR_MAINFRAME));
```

Tương tự, các lệnh sau đây nạp thực đơn tách biệt và gắn nó vào cửa sổ khung đang tồn tại:

```
CMenu menu;
menu.LoadMenu (IDR_MAINFRAME);
SetMenu (&menu);
menu.Detach ();
```

Kỹ thuật cuối cùng và có hiệu quả để nạp và gắn thực đơn khi chương trình đang chạy (*on-the-fly*), đó là cái mà lập trình viên muốn làm khi ứng dụng trợ giúp cả thực đơn dài và ngắn. Nếu cửa sổ đang nhìn thấy trên màn hình khi SetMenu được gọi, phải đảm bảo chắc chắn là có gọi CWnd::DrawMenuBar để thực đơn được vẽ lại phản ánh sự thay đổi. Trong thí dụ trên đây, hàm Detach của đối tượng thực đơn được gọi để tách đối tượng thực đơn từ tài nguyên thực đơn để sau này chúng không bị phá hủy khi thực đơn ra khỏi miền tác động. (Nhìn qua bên trong AFXWIN1.INL và ta sẽ thấy hủy tử CMenu có gọi DestroyMenu). Tài nguyên thực đơn được xóa tự động khi cửa sổ khung bị phá hủy. Ta không cần phải quan tâm tài nguyên bị rò rỉ do thực đơn còn ở lại chừng nào thực đơn còn gắn vào cửa sổ.

Phím lệnh cũng vậy, phải được nạp từ trình ứng dụng. Nạp bảng mã lệnh cho cửa sổ khung khá đơn giản: chỉ cần gọi CFrameWnd::LoadAccelTable với ID tài nguyên của bảng phím lệnh. Lệnh:

```
LoadAccelTable (MAKEINTRESOURCE (IDR_MAINFRAME));
```

nạp bảng mã lệnh có ID là IDR_MAINFRAME và liên kết chúng với cửa sổ khung. Trong chương trình C, bảng mã lệnh không tích cực cho đến khi vòng lặp thông điệp được sửa đổi để xử lý các thông điệp nhờ ::TranslateAccelerator trước khi chuyển chúng cho ::TranslateMessage và ::DispatchMessage. Trong MFC, lớp CFrameWnd nạp chồng hàm ảo CWnd::PreTranslateMessage và gọi hàm ::TranslateAccelerator cho ta nếu bảng phím lệnh đã được nạp - có nghĩa là nếu thành phần dữ liệu m_hAccelTable của đối tượng cửa sổ khung chứa chỉ danh bảng phím lệnh khác NULL.

Cách quản lý bảng phím lệnh khi nạp cho cửa sổ không có khung có khác, chúng không trợ giúp bảng phím lệnh như trong CFrameWnd. Giả sử ta xây dựng lớp cửa sổ phái sinh từ CWnd và muốn nạp bảng phím lệnh. Sau đây là cách phải làm. Trước hết, thêm thành phần dữ liệu m_hAccel (kiểu HACCEL) vào lớp cửa sổ. Tiếp theo, ngay vào thời điểm bắt đầu chạy chương trình, có thể trong cấu tử của cửa sổ chính, ngay sau khi gọi Create, hãy sử dụng ::LoadAccelerators để nạp bảng phím lệnh. Chép giá trị cho lại từ ::LoadAccelerators vào m_hAccel. Cuối cùng, trong lớp cửa sổ, nạp chồng PreTranslateMessage và gọi ::TranslateAccelerator với chỉ số lưu trong m_hAccel. Sử dụng giá trị do ::TranslateAccelerator cho lại như giá trị cho lại của PreTranslateMessage để thông điệp sẽ không được dịch và chuyển đi nếu ::TranslateAccelerator đã sử dụng chúng.

Sau đây là đoạn mã trình đồ:

```
// Trong cấu tử của CMyWindow ...
m_hAccel = ::LoadAccelerators(AfxGetInstanceHandle(), MAKEINTRESOURCE (IDR_MAINFRAME));
// Nạp chồng hàm PreTranslateMessage
BOOL CMyWindow::PreTranslateMessage (MSG* pMsg)
{
    return ((hAccel != NULL) && ::TranslateAccelerator (m_hWnd, m_hAccel, pMsg));
}
```

Với khung ứng dụng, cửa sổ kiểu CWnd sẽ sử dụng bảng phím lệnh đúng như trường hợp cửa sổ khung.

Chú ý rằng bảng phím lệnh do ::LoadAccelerators (hay LoadAccelTable) nạp không cần phải xoá trước khi kết thúc, bởi vì Windows tự động xoá chúng.

Nếu quan sát bên trong ứng dụng phát sinh từ *AppWizard*, sẽ không thấy mã trình nạp thực đơn hay bảng mã lệnh. Đó là do *AppWizard* tạo ứng dụng tài liệu/quan sát có sử dụng mẫu tài liệu phái sinh từ lớp CDocTemplate của MFC để xác định quan hệ giữa các cửa sổ khung, tài liệu và quan sát. Trong ứng dụng tài liệu/quan sát, khung ứng dụng nạp thực đơn và bảng phím lệnh. Khi ứng dụng bắt đầu chạy, khung ứng dụng gọi hàm CreateNewFrame của mẫu tài liệu để tạo cửa sổ khung, gọi hàm CreateNewFrame để lập đối tượng cửa sổ khung, sau đó gọi hàm LoadFrame của đối tượng. LoadFrame tạo cửa sổ khung và nạp thực đơn, bảng phím lệnh, biểu tượng, tiêu đề cửa sổ trong cùng một bước. Ta cũng có thể sử dụng LoadFrame trong ứng dụng *non-doc/view*. Giả sử pFrameWnd trở đến cửa sổ khung chưa khởi động giá trị, lệnh

```
pFrameWnd->LoadFrame (IDR_MAINFRAME);
```

tạo cửa sổ khung, gắn nó vào đối tượng cửa sổ khung, và tự động nạp thực đơn, bảng phím lệnh, biểu tượng hay tài nguyên xâu ký tự có chỉ danh ID là IDR_MAINFRAME.

5.3 CÁC LỆNH VÀ ÁNH XẠ THÔNG ĐIỆP

Khi người sử dụng chọn mục trong thực đơn hay nhấn phím lệnh, cửa sổ liên quan sẽ nhận thông điệp WM_COMMAND với từ thấp của wParam lưu chỉ danh ID lệnh và từ cao có giá trị 0 khi chọn thực đơn, hay giá trị 1 khi chọn phím lệnh. Các ứng dụng MFC ánh xạ các sự kiện của cả hai loại này vào hàm thành phần của lớp tương ứng nhờ các đầu vào ánh xạ thông điệp ON_COMMAND. Ánh xạ thông điệp sau của lớp cửa sổ phái sinh từ CFrameWnd với tên CMyFrame liên kết các mục thực đơn *File New* và *File Open*, có chỉ danh tương ứng là ID_FILE_NEW và ID_FILE_OPEN, vào các hàm thành phần với tên OnFileNew và OnFileOpen:

```
BEGIN_MESSAGE_MAP (CMyFrame, CFrameWnd)
    ON_COMMAND (ID_FILE_NEW, OnFileNew)
```



```
ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
END_MESSAGE_MAP
```

Không có gì bí hiểm với các tên hàm; ta có thể sử dụng bất kỳ tên nào mà ta muốn. Như chỉ ra trên đây, `CMyFrame::OnFileNew` và `CMyFrame::OnFileOpen` sẽ được kích hoạt vào thời điểm `New` và `Open` được chọn từ thực đơn `File` hay nhấn phím lệnh tương đương của ứng dụng.

Rất nhiều lập trình viên C quen sử dụng cấu trúc *switch-case* để rẽ nhánh thông điệp `WM_COMMAND` nhờ chỉ danh ID lệnh. Như ta thấy từ thí dụ này, MFC áp đặt mã trình "đẹp hơn" trong cấu trúc thông điệp bằng sử dụng các ánh xạ thông điệp để hướng các thông điệp `WM_COMMAND` tới hàm điều khiển tương ứng trên cơ sở ID lệnh trong `wParam`. Nếu định nghĩa `wParam` có bị thay đổi thay đổi (như đã làm giữa Windows® 3.1 và Windows 95/98), chỉ đơn giản dịch lại thì sẽ cập nhật được chương trình của ta bởi vì macro `ON_COMMAND` sẽ được cập nhật theo.

Trong các ứng dụng không có kiến trúc tài liệu/quan sát của MFC, các thông điệp thực đơn luôn ánh xạ đến các hàm thành phần của cửa sổ của thực đơn. Trong ứng dụng tài liệu/quan sát, các thông điệp thực đơn có thể được sử dụng bất kỳ đâu - trong lớp quan sát, lớp khung, lớp tài liệu, hay thậm chí lớp ứng dụng. Khung ứng dụng thêm các đại lượng khác cho việc xử lý `WM_COMMAND` bằng việc cài đặt hệ thống dẫn lệnh rất mạnh, cho mỗi đối tượng một cơ hội xử lý thông điệp từ thực đơn và điều khiển. Ta sẽ xem xét dẫn lệnh hoạt động qua hai cài đặt trước khi giới thiệu kiến trúc tài liệu/quan sát.

5.4 DẢI LỆNH

Đôi khi hiệu quả hơn nếu xử lý dải các chỉ danh ID của mục thực đơn với một hàm thành phần thay cho cung cấp hàm thành phần riêng rẽ cho chúng. Hãy xem xét ứng dụng vẽ có thực đơn `Color` từ đó người sử dụng có thể chọn các màu đỏ, xanh lá cây, xanh lam (*red*, *green*, hoặc *blue*). Chọn màu từ thực đơn sẽ nạp biến thành phần với tên `m_nCurrentColor` một giá trị ID của mục thực đơn đã được chọn, tiếp theo là thay đổi màu vẽ trên màn hình. Đầu vào ánh xạ thông điệp và các hàm thành phần liên quan cho mỗi mục thực đơn phải được cài đặt như sau:

```
ON_COMMAND(ID_COLOR_RED, OnColorRed)
ON_COMMAND(ID_COLOR_GREEN, OnColorGreen)
ON_COMMAND(ID_COLOR_BLUE, OnColorBlue)
void CMyFrame::OnColorRed ()
{
    m_nCurrentColor = ID_COLOR_RED;
}
void CMyFrame::OnColorGreen ()
{
```

```

    m_nCurrentColor = ID_COLOR_GREEN;
}
void CMyFrame::OnColorBlue ()
{
    m_nCurrentColor = ID_COLOR_BLUE;
}

```

Tuy nhiên, đây không phải cách làm hiệu quả khi xử lý thông điệp từ thực đơn *Color* vì mỗi hàm điều khiển thông điệp thực hiện cùng công việc chính. Và hiệu quả giảm đi nếu có 10 đến 20 màu để chọn thay cho ba màu.

Một cách làm giảm dư thừa trong logic quản lý thực đơn của chương trình là ánh xạ toàn bộ ba mục thực đơn đến cùng hàm thành phần của CMyFrame và khai thác chỉ danh mục thực đơn bằng hàm CWnd::GetCurrentMessage, như sau:

```

ON_COMMAND (ID_COLOR_RED, OnColor)
ON_COMMAND (ID_COLOR_GREEN, OnColor)
ON_COMMAND (ID_COLOR_BLUE, OnColor)
void CMyFrame::OnColor ()
{
    UINT nID = (UINT) LOWORD(GetCurrentMessage ()->wParam);
    m_nCurrentColor = nID;
}

```

Công việc này vẫn còn chưa hoàn hảo vì nó còn phụ thuộc vào giá trị của wParam.

Giải pháp tốt hơn là dùng *macro* ON_COMMAND_RANGE của MFC, chúng ánh xạ dải các ID đến hàm thành phần chung và tải ID của mục đến hàm thành phần như tham số. Bây giờ, giả sử ID_COLOR_RED, ID_COLOR_GREEN, và ID_COLOR_BLUE chứa các giá trị liên tục và ID_COLOR_RED, ID_COLOR_BLUE là giá trị thấp và cao nhất của dải, (thí dụ, ID_COLOR_RED=100, ID_COLOR_GREEN=101, và ID_COLOR_BLUE=102), ta có thể viết lại mã trình cho thực đơn Color như sau:

```

ON_COMMAND_RANGE (ID_COLOR_RED, ID_COLOR_BLUE, OnColor)
void CMyFrame::OnColor (UINT nID)
{
    m_nCurrentColor = nID;
}

```

Khi người sử dụng chọn màu và hàm CMainWindow::OnColor được gọi, nID chứa ID_COLOR_RED, ID_COLOR_GREEN, hoặc ID_COLOR_BLUE để phân biệt việc chọn. Và một lệnh đơn giản đặt giá trị thích hợp cho m_nCurrentColor, không quan tâm đến mục thực đơn nào được chọn.

5.5 TRÌNH ĐIỀU KHIỂN CẬP NHẬT

Trong nhiều ứng dụng, các mục thực đơn thường xuyên được cập nhật để phản ánh trạng thái của ứng dụng. Thí dụ trong thực đơn *Color*, chọn màu có thể đặt dấu chọn (✓) bên cạnh mục thực đơn. Trong thực đơn *Edit*, các mục *Cut*, *Copy* thường bị làm mất hiệu lực khi không có văn bản (hay dữ liệu) được chọn, và mục thực đơn *Paste* chỉ được tích cực khi có cái gì đó trong *Clipboard* để dán.

Thông thường lập trình viên C sử dụng một trong hai cách để đặt trạng thái cho mục thực đơn: cập nhật ngay mục thực đơn khi hành động liên quan xảy ra, hoặc chúng bẫy thông điệp `WM_INITMENUPOPUP` chỉ ra thực đơn *pop-up* sẽ hiển thị và cập nhật toàn bộ các mục trong thực đơn. MFC cung cấp cơ chế đơn giản hơn (và độc lập với loại máy, hệ điều hành) để cập nhật các mục thực đơn. Bằng cách thêm macro `ON_UPDATE_COMMAND_UI` vào ánh xạ thông điệp, ta có thể chỉ rõ các hàm thành phần dùng làm hàm điều khiển cập nhật cho các mục thực đơn. Khi người sử dụng chọn thực đơn, khung ứng dụng bẫy thông điệp `WM_INITMENUPOPUP` và gọi toàn bộ hàm điều khiển cập nhật tương quan với các mục thực đơn. Mỗi hàm điều khiển cập nhật được chuyển cho con trỏ đến đối tượng `CCmdUI` biểu diễn mục phát sinh thông điệp cập nhật, và các hàm thành phần của nó có thể được sử dụng để cập nhật. Hơn nữa, bởi vì `CCmdUI` không là đặc thù cho bất kỳ loại nào của thành phần ghép nối ứng dụng, cùng hàm điều khiển cập nhật có thể sử dụng chung cho cả thực đơn, thanh công cụ, và các đối tượng giao diện với người sử dụng (UI) khác.

Các hàm điều khiển cập nhật được gán đầu vào ánh xạ thông điệp `ON_UPDATE_COMMAND_UI`. Các đầu vào ánh xạ thông điệp, các hàm điều khiển lệnh, và hàm điều khiển cập nhật được chỉ ra trong chương trình 5.3 có thể được sử dụng cho mục thực đơn *Edit Copy* trong ứng dụng cho phép người sử dụng nhập văn bản trong hộp soạn thảo (*edit control*) biểu diễn bằng đối tượng `CEdit` có tên `m_edit`. Mỗi lần thực đơn *Edit* được chọn, `OnUpdateEditCopyUI` bắt đầu hoạt động và gọi `CEdit::GetSel` để xác định xem xâu ký tự có được chọn hay không. Nếu có (nếu `nStart != nEnd`), thì gọi `CCmdUI::Enable` để làm tích cực mục thực đơn *Copy*. Nếu không có xâu ký tự nào được chọn (`nStart = nEnd`), thì mục thực đơn bị làm “mất hiệu lực”.

Chương trình 5.3 Mục thực đơn *Edit Copy*

```
BEGIN_MESSAGE_MAP (CMyFrame, CFrameWnd)
    ON_COMMAND (ID_EDIT_COPY, OnEditCopy)
    ON_UPDATE_COMMAND_UI (ID_EDIT_COPY, OnUpdateEditCopyUI)
    .....
END_MESSAGE_MAP
void CMyFrame::OnEditCopy ()
{
```

```

    m_edit.Copy ();
}
void CMainWindow::OnUpdateEditCopyUI (CCmdUI* pCmdUI)
{
    int nStart, nEnd;
    m_edit.GetSel (nStart, nEnd);
    pCmdUI->Enable(nStart != nEnd);
}

```

Enable là một trong bốn hàm thành phần thuận tiện mà lớp CCmdUI cung cấp để cập nhật các mục thực đơn; những hàm khác được liệt kê trong bảng 5.1. **SetCheck** đánh dấu hay bỏ đánh dấu một mục thực đơn, phụ thuộc vào giá trị khác 0 hay 0 chuyển cho nó (mặc định là khác 0). **SetRadio** “đánh dấu” mục thực đơn bằng dấu chấm nếu không có đối số hay giá trị khác 0, và bỏ dấu chấm nếu 0 được chuyển đến. **SetText** thay tên của mục thực đơn. Chúng nhận con trỏ đến xâu hay đối tượng CString. SetRadio thường được sử dụng để đánh dấu mục được chọn trong tập mục thực đơn loại trừ (tên hàm đề cập đến chức năng tương tự nhóm các phím của *radio*), trong khi SetCheck được sử dụng như chỉ báo bật/tắt.

Bảng 5.1 Các hàm thành phần CCmdUI để cập nhật mục thực đơn

Hàm	Mô tả
Enable	Làm hiệu lực hay làm mất hiệu lực mục thực đơn
SetCheck	Đánh dấu hay bỏ đánh dấu khỏi mục thực đơn bằng dấu chọn
SetRadio	Đánh dấu hay bỏ đánh dấu khỏi mục thực đơn bằng dấu chấm
SetText	Đặt tên cho mục thực đơn

Cũng như ON_COMMAND_RANGE có thể sử dụng cho ánh xạ dài chỉ danh ID lệnh đến một hàm điều khiển, ON_UPDATE_COMMAND_UI_RANGE có thể được sử dụng để ánh xạ dài các ID lệnh đến handler cập nhật chung. Trình quản lý lệnh và cập nhật cho thực đơn *Color* vừa xem xét có thể được viết như sau:

```

ON_COMMAND_RANGE (ID_COLOR_RED, ID_COLOR_BLUE, OnColor)
ON_UPDATE_COMMAND_UI_RANGE (ID_COLOR_RED, ID_COLOR_BLUE, OnUpdateColorUI)
void CMyFrame::OnColor (UINT nID)
{
    m_nCurrentColor = nID;
}
void CMyFrame::OnUpdateColorUI (CCmdUI* pCmdUI)
{
    pCmdUI->SetRadio (pCmdUI->m_nID == m_nCurrentColor);
}

```

Trong đó, m_nID là thành phần dữ liệu chung của CCmdUI để lưu ID của mục thực đơn đã gọi

hàm điều khiển cập nhật cho chúng. So sánh `m_nID` với `m_nCurrentColor` trong lời gọi đến `SetRadio` để đảm bảo rằng chỉ có màu hiện hành được đánh dấu (*bullet*).

5.6 THANH CÔNG CỤ

Ngày nay, càng ngày càng có nhiều ứng dụng sử dụng thanh công cụ. MFC làm thanh công cụ dễ dàng nhờ lớp `CToolBar`, gói cửa sổ thanh công cụ chứa các phím với ảnh kiểu biểu tượng. Mỗi phím được gán một chỉ danh ID lệnh, nếu nhấn phím sẽ phát sinh thông điệp `WM_COMMAND` tương tự chọn mục thực đơn. Các phím thanh công cụ thường được gán cùng ID với các mục thực đơn để cùng một hàm thành phần có thể sử dụng cả hai.

Phím thanh công cụ và mục thực đơn có cùng ID thì cùng chia sẻ *handler* cập nhật. Để cập nhật mục thực đơn, khung ứng dụng gọi hàm điều khiển cập nhật với đối tượng `CCmdUI` tham chiếu phím. Vì vậy, cùng mã trình thực hiện làm có hiệu lực hay vô hiệu lực mục thực đơn *Edit Copy* trên cơ sở trạng thái chọn hiện hành cũng sẽ làm có hiệu lực hay vô hiệu lực phím *Copy* trong thanh công cụ. Sự khác nhau chỉ là mục thực đơn được cập nhật mỗi khi hiển thị thực đơn, trong khi phím thanh công cụ được cập nhật trong chu kỳ rỗi (*idle cycle*) – chu kỳ mà hàng đợi thông điệp rỗng.

Windows 95/98 cung cấp cài đặt thanh công cụ riêng của mình trong thư viện điều khiển. Trong MFC 4.0, khung ứng dụng tự động sử dụng thanh công cụ Windows 95/98 thay cho cài đặt thanh công cụ riêng của đối tượng `CToolBar`. Như vậy, `CToolBar` là cách đơn giản để sử dụng thanh công cụ Windows 95/98 trong ứng dụng của ta mà không cần phải học API và sơ đồ thông điệp mới.

Ưu việt của `CToolBar` là chỉ cần ít công việc phải làm để có thể thêm thanh công cụ vào ứng dụng. Thí dụ, ta cung cấp *bitmap* cho các phím công cụ; MFC phát sinh *bitmap* biểu diễn các phím khi chúng bị nhấn hay vô hiệu lực. Ta cung cấp một mảng chỉ danh ID lệnh, và MFC tạo các phím cho ta. Thêm *ToolTip* – cửa sổ thu nhỏ chứa trợ giúp mà nó xuất hiện khi con chuột lướt qua các phím thanh công cụ khoảng nửa giây hoặc lâu hơn – là chỉ đơn giản thêm bảng xâu ký tự vào tệp RC của ta. Có muốn gán thanh công cụ vào cửa sổ? Không có vấn đề; chỉ gọi hàm `EnableDocking` của thanh công cụ và hàm `EnableDocking` của cửa sổ khung và MFC sẽ làm các công việc còn lại. Với một ít công việc, thanh công cụ thậm chí có thể được sửa đổi để có các điều khiển khác như hộp *combo*.

Toolbars làm việc tốt nhất với cửa sổ khung bởi vì các lớp `CFrameWnd` và `CToolBar` của MFC được thiết kế để làm việc chung. Thanh công cụ thông thường được tạo trong đoạn trình `OnCreate` của cửa sổ khung. Các lệnh sau đây lập thanh công cụ, nạp *bitmap* của các phím và đặt chỉ danh ID phím.

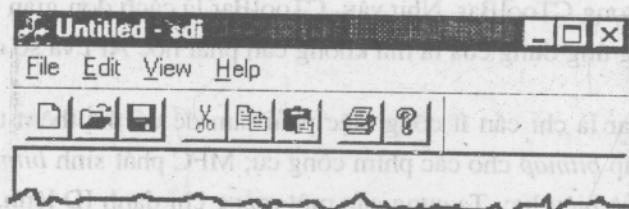
```
static UINT nButtonIDs[] = {
```

```

ID_FILE_NEW,
ID_FILE_OPEN,
ID_FILE_SAVE,
ID_SEPARATOR,
ID_EDIT_CUT,
ID_EDIT_COPY,
ID_EDIT_PASTE,
ID_SEPARATOR,
ID_FILE_PRINT,
ID_APP_ABOUT,
};
m_toolbar.Create(this);
m_toolbar.LoadBitmap(IDR_TOOLBAR);
m_toolbar.SetButtons(nButtonIDs, 10)

```

Trong đó, `m_toolbar` là thành phần của lớp cửa sổ khung, đối tượng *toolbar* được tự động tạo lập khi cửa sổ khung được tạo, và bị phá hủy khi cửa sổ khung bị phá hủy. Thanh công cụ này chứa tám phím và hai đường ngăn cách (*separator*) – là khoảng trống rộng vài *pixel* để tách nhóm phím. Hàm `Create` tạo cửa sổ công cụ; `LoadBitmap` nạp tài nguyên *bitmap* chứa tám ảnh, kích thước mỗi ảnh là 16 *pixel* rộng và 15 *pixel* chiều cao; và `SetButtons` gán ID của phím. Các ID phím được lưu trong mảng `nButtonIDs`, giá trị ID đặc biệt `ID_SEPARATOR` đánh dấu *separators*. Thanh công cụ kết quả được thể hiện trong hình 5.2.



Hình 5.2 Các *bitmap* thanh công cụ

`CToolBar` cung cấp phân loại các hàm thành phần để thay đổi hình dạng và ứng xử của thanh công cụ, truy nhập thông tin. Thí dụ, `CToolBar::SetSizes` được sử dụng để thay đổi kích thước ảnh phím khác với 16x15 *pixel* và kích thước phím tới 24x22 *pixel*. Cho trước chỉ số phím (tính từ 0), `CToolBar::GetItemID` cho lại chỉ danh ID lệnh gán cho phím. Khi quan sát chức năng mà `CToolBar` cung cấp qua các hàm thành phần, đừng quên quan sát cả lớp cơ sở `CControlBar`. Ta sẽ thấy ở đây các hàm thuận tiện như `EnableDocking`, chúng cho biết thanh công cụ có thể gắn ở đâu, và `SetBarStyle`, thay đổi thuộc tính kiểu thanh công cụ. Trong `CControlBar` ta cũng còn thấy biến thành phần chung có tên `m_bAutoDelete`, nếu có giá trị `TRUE`, làm đối tượng `CToolBar` tự động phá hủy khi cửa sổ thanh công cụ bị phá hủy. Bảng 5.2 liệt kê các hàm thanh công cụ do `CToolBar`

và CControlBar cung cấp.

Bảng 5.2 Các hàm thanh công cụ

Hàm	Mô tả
CommandToIndex	Cho trước ID phím, cho lại chỉ số phím tính từ 0
Create	Lập thanh công cụ và gắn nó vào đối tượng CToolBar
EnableDocking	Cho phép gắn thanh công cụ và chỉ ra nơi thanh công cụ có thể được gắn (xem thêm CFrameWnd::EnableDocking)
GetBarStyle	Khai thác kiểu thanh công cụ
GetButtonInfo	Khai thác ID phím, kiểu, và chỉ số ảnh
GetCount	Khai thác tổng số phím và đường tách thực đơn
GetItemID	Cho trước chỉ số, cho lại ID phím
GetItemRect	Cho trước chỉ số, cho lại chữ nhật phím
LoadBitmap	Nạp tải nguyên bitmap chứa ảnh phím
SetBarStyle	Đặt kiểu thanh công cụ
SetButtonInfo	Đặt ID phím, kiểu và chỉ số ảnh
SetButtons	Gán ID lệnh của phím thanh công cụ
SetHeight	Đặt độ cao thanh công cụ
SetSizes	Đặt kích thước ảnh, phím của thanh công cụ

Thêm *ToolTips* vào thanh công cụ khá dễ dàng. Khi tạo thanh công cụ, chỉ cần gộp thêm kiểu CBRS_TOOLTIPS. Hay sử dụng GetBarStyle và SetBarStyle để thêm CBRS_TOOLTIPS sau khi thanh công cụ đã được tạo lập:

```
m_toolbar.SetBarStyle(m_toolbar.GetBarStyle() | CBRS_TOOLTIPS);
```

Bây giờ công việc mà ta phải làm là thêm bảng xâu ký tự vào tệp RC của ứng dụng với xâu ký tự của *ToolTip*. Mỗi tài nguyên xâu ký tự chỉ ra xâu cho *ToolTip* cho mỗi phím và được gán cùng ID như phím. Nếu ta sử dụng ID phím mục thực đơn/thanh công cụ như đã định nghĩa trước, MFC sẽ cung cấp xâu *ToolTip* cho ta khi ta đã gộp tệp AFXRES.H. Xâu *ToolTip* cho những ID phím khác phải được chỉ ra riêng rẽ. Thí dụ, thanh công cụ chứa các phím để chọn *bold*, *italic*, và *underlined text*, và ID phím là IDB_BOLD, IDB_ITALIC, và IDB_UNDERLINE, bảng xâu ký tự có thể như sau:

```
STRINGTABLE
BEGIN
    IDB_BOLD      "\nBold"
    IDB_ITALIC    "\nItalic"
    IDB_UNDELRINE "\nUnderline"
END
```

Các ký tự sau \n là ký tự của *ToolTip* cho phím. Nếu ứng dụng có chứa cả thanh-trạng thái, ta có

thể thêm kiểu **CBRS_FLYBY** vào thanh công cụ và sử dụng cùng bảng xâu ký tự để chỉ ra các ký tự “flyby” – mô tả chi tiết hơn xuất hiện trong thanh trạng thái khi *ToofTip* xuất hiện:

```
STRINGTABLE
BEGIN
    IDB_BOLD      "Select bold text\nBold"
    IDB_ITALIC    "Select italic text\nItalic"
    IDB_UNDELRINE "Select underlined text\nUnderline"
END
```

Nếu thử gắn thanh công cụ và thanh trạng thái của Windows 95/98 mà không tận dụng MFC, ta sẽ thấy rõ tính giản đơn mà khung ứng dụng cho tiến trình.

Sau đây là “mạch nước” liên quan đến sử dụng thanh công cụ. Thanh công cụ đứng vững trong các ứng dụng tài liệu/quan sát và không có kiến trúc tài liệu/quan sát, nhưng nếu ta cho phép thanh công cụ gắn vào ứng dụng và trôi nổi trong ứng dụng không có kiến trúc tài liệu/quan sát, ta sẽ thấy khó theo dõi phần thấy được của vùng làm việc của cửa sổ khung bởi vì vùng làm việc không thay đổi kích thước khi thanh công cụ dịch chuyển. Trong ứng dụng tài liệu/quan sát, vùng làm việc đi theo cửa sổ quan sát chứ không theo cửa sổ khung, và khung ứng dụng tự động thay đổi kích thước cửa sổ quan sát mỗi khi thanh công cụ đặt lại vị trí. Làm thanh công cụ tĩnh trong ứng dụng không có kiến trúc tài liệu/quan sát thì logic chương trình sẽ đơn giản nhiều.

5.7 THANH TRẠNG THÁI

Trong ứng dụng MFC, thanh công cụ, thanh trạng thái và thực đơn luôn đi với nhau bởi vì cả ba cùng làm việc chung. Thanh trạng thái hiển thị trợ giúp cho các mục thực đơn và cho phép thanh công cụ. Trong MFC, thanh trạng thái được biểu diễn bởi đối tượng của lớp `CStatusBar`. Khi ta lập cửa sổ `CStatusBar` trong Windows 95, MFC 4.0 sẽ tự động sử dụng điều khiển thanh trạng thái tìm thấy trong thư viện điều khiển chung. Cũng như thanh công cụ, MFC cung cấp cách tốt nhất để lập trình điều khiển thanh trạng thái Windows 95/98.

Đồ có thanh trạng thái cơ sở hiển thị và chạy - nó chỉ chứa một ô (*pane*) mô tả mục thực đơn hay phím thanh công cụ được chọn - chỉ đòi hỏi có ba dòng mã trình. Thông thường, đối tượng `CStatusBar` được gói trong lớp cửa sổ khung và `CStatusBar::Create` sau đó được gọi từ *handler* `OnCreate` của cửa sổ khung:

```
m_statusbar.Create(this);
UINT nIndicator = ID_SEPARATOR;
m_statusbar.SetIndicators(&nIndicator, 1);
```

Hàm `Create` tạo cửa sổ thanh trạng thái, chỉ ra cửa sổ khung là cha, cho thanh trạng thái kiểu **CBRS_BOTTOM** sao cho nó sẽ được đặt vào đáy của cha nó, và gán chỉ danh ID ảo

AFX_IDW_STATUS_BAR sao cho nó sẽ trở thành một phần của khung ứng dụng. SetIndicators khởi động thanh trạng thái bằng cách cho nó biết ta cần bao nhiêu ô (*pane*) - ở đây là một. ID_SEPARATOR cho khung ứng dụng biết rằng không có xâu ký tự mặc định gắn với ô.

Khi thông điệp WM_MENUSELECT đến, chỉ ra đã chọn thực đơn, cửa sổ khung tự động nạp tài nguyên xâu ký tự tương ứng - xâu có ID cùng với ID của mục thực đơn - và hiển thị xâu ký tự trong ô của thanh trạng thái. Chính vì vậy, hành động đơn giản là thêm tài nguyên xâu ký tự chứa trợ giúp ngữ cảnh cho mỗi mục thực đơn vào tệp tài nguyên đã cho phép khung ứng dụng hiển thị trợ giúp trong thanh trạng thái. Để kết thúc tốt đẹp, ta có thể thêm dòng sau đây vào bảng xâu ký tự:

```
AFX_IDS_IDLEMESSAGE "Ready"
```

Bây giờ "Ready" sẽ được hiển thị trong thanh trạng thái mỗi khi không thực đơn nào được chọn. Thanh trạng thái không bị hạn chế chỉ với một ô. Ta có thể tạo thêm các ô bằng cách chỉ ra ID ô khi gọi SetIndicators. Giả sử ta muốn thêm ô để hiển thị tổng số lần ứng dụng chạy. Nếu ID của ô là ID_INDICATOR_ELAPSED_TIME, lời gọi đến SetIndicators có thể như sau:

```
static UINT nIndicators[] = {
    ID_SEPARATOR,
    ID_INDICATOR_ELAPSED_TIME
};
m_statusbar.SetIndicators (nIndicators, 2);
int nWidth;
UINT nID, nStyle;
m_statusbar.GetPaneInfo (1, nID, nStyle, nWidth);
m_statusbar.SetPaneInfo (1, nID, nStyle, 48);
```

Gọi hàm SetPaneInfo để đặt độ rộng ô 48 *pixel*. Vì SetPaneInfo cũng đặt ID và kiểu ô, phải gọi hàm GetPaneInfo trước để ID và kiểu hiện hành không thay đổi. Không cần thiết phải đặt độ rộng của ô trái nhất vì mặc định nó là ô "đàn hồi" có độ rộng tự động chỉnh để làm đầy dấu cách không sử dụng trong thanh ghi trạng thái.

Ô mới đã được tạo lập, có hai cách cập nhật chúng. Có thể gọi hàm CStatusBar::SetPaneText để thay đổi ký tự hiển thị trong ô mỗi khi thời gian trôi qua thay đổi (có thể trả lời thông điệp đồng hồ), hay cung cấp hàm điều khiển cập nhật cho ID_INDICATOR_ELAPSED_TIME sử dụng CCmdUI::SetText để cập nhật xâu ký tự trong thời gian rồi. Như ta thấy hàm điều khiển cập nhật cũng hoạt động với thanh trạng thái.

Cách khác để cung cấp xâu ký tự cho ô thanh trạng thái là gộp tài nguyên xâu với cùng ID như ô trong tệp tài nguyên của ta. Khung ứng dụng sẽ tự động thay đổi kích thước để cho vừa xâu ký tự, và sẽ hiển thị xâu ký tự khi ô có hiệu lực và dấu nó khi ô mất hiệu lực. Ta làm có hiệu lực và vô

hiệu lực ô như thế nào? Thật đơn giản: cung cấp hàm điều khiển cập nhật và sử dụng hàm `CCmdUI::Enable` để làm có hiệu lực và vô hiệu lực ô tương ứng. Ta cũng có thể sử dụng `CCmdUI::SetText` và `CCmdUI::SetCheck` trong hàm điều khiển cập nhật để sửa đổi ô. `SetText` thay đổi xâu hiển thị trong ô, trong khi `SetCheck` đánh dấu và bỏ đánh dấu ô. Ô đánh dấu được gán kiểu `SBPS_POPOUT` để nó nhô lên từ bề mặt của thanh trạng thái.

Nếu ta muốn gộp các chỉ báo trạng thái cho các phím Caps Lock, Num Lock, và Scroll Lock trong thanh trạng thái, đơn giản là thêm các ô với ID đặc biệt như `ID_INDICATOR_CAPS`, `ID_INDICATOR_NUM`, và `ID_INDICATOR_SCRL`. Chừng nào thanh trạng thái được gán vào cửa sổ khung, khung ứng dụng sẽ tự động hiển thị CAP, NUM, và SCRL tương ứng.

Còn nhiều thứ ta có thể làm với thanh trạng thái; trên đây chỉ là một số điểm nổi bật. Khi làm việc với lớp `CStatusBar` của MFC, ta thấy rằng sẽ có cách để bắt nó làm bất cứ cái gì mà ta muốn. Quan trọng hơn, với khung ứng dụng đằng sau, khả năng của nó sẽ tăng lên rất nhiều.

5.8 THỰC ĐƠN NGŨ CẢNH

Một trong các thành phần giao diện với người dùng (UI) mới trong Windows 95/98 là menu ứng dụng có thể trợ giúp thực đơn ngữ cảnh. Thực đơn ngữ cảnh là thực đơn được hiển thị khi nhấn phím phải trên đối tượng. Tư tưởng chính ở đây là trong môi trường hướng đối tượng, người sử dụng phải có khả năng nghĩ đến mục được vẽ trên màn hình như đối tượng và lấy danh sách các hành động mà nó có thể thực hiện trên bất kỳ đối tượng nào khác thông qua thực đơn ngữ cảnh. Một thí dụ hoàn hảo là *desktop* của Windows 95/98. Bởi vì UI xử lý cửa sổ *desktop* như đối tượng, nhấn phím phải trên nền màn hình sẽ kích hoạt thực đơn ngữ cảnh với các tùy chọn như *Arrange Icons*, *New*, và *Properties*. Qua *Properties*, người ta có thể chọn lại độ phân giải màn hình, *screen saver*, *wallpaper bitmap*, và hơn thế nữa.

Ta có thể tạo thực đơn ngữ cảnh nhờ lớp `CMenu` của MFC và một vài hàm thành phần của chúng. Windows thực hiện chúng dễ dàng nhờ gửi thông điệp `WM_CONTEXTMENU` khi nhấn phím phải chuột bên trong vùng làm việc của cửa sổ. Thực đơn ngữ cảnh chính là thực đơn *pop-up* nó không gắn vào thực đơn *top-level*. Các thực đơn này, thường được gọi là thực đơn *pop-up* trôi nổi, được lập đúng như thực đơn *pop-up* thông thường và hiển thị trên màn hình nhờ hàm `TrackPopupMenu` của `CMenu`. `TrackPopupMenu` có nguyên hàm như sau:

```
BOOL TrackPopupMenu (UINT nFlags, int x, int y, CWnd* pWnd, LPCRECT lpRect = 0)
```

Trong đó, *x* và *y* là vị trí trên màn hình (theo tọa độ màn hình) nơi thực đơn *pop-up* sẽ xuất hiện. *nFlags* chứa các bit cờ chỉ ra hiệu chỉnh ngang thực đơn so với *x* và phím chuột nào có thể được dùng để chọn mục thực đơn. Các cờ gán `TPM_LEFTALIGN`, `TPM_CENTERALIGN`, và `TPM_RIGHTALIGN` cho Windows 95/98 biết rằng *x* xác định vị trí của chỉnh trái, chỉnh giữa,

chính phải của thực đơn, trong khi các cờ `TPM_LEFTBUTTON` và `TPM_RIGHTBUTTON` xác định việc chọn thực đơn bằng phím trái hay phải của chuột. Chỉ một trong các cờ chính hàng được chỉ ra, hoặc cả hai cờ phím được sử dụng. `pWnd` chỉ ra cửa sổ chủ của thực đơn *pop-up* nổi, và `lpRect` trỏ đến đối tượng `CRect` hay cấu trúc `RECT` chứa tọa độ màn hình của chữ nhật nơi người sử dụng nhấn chuột vào để có thực đơn. Nếu `lpRect` là 0, nhấn phím bất kỳ đâu ở bên ngoài thì thực đơn tự động bỏ qua. Giả sử `pMenu` trỏ đến đối tượng thực đơn *pop-up* liên quan đến cửa sổ chính của ứng dụng, lệnh:

```
pMenu->TrackPopupMenu (TPM_LEFTALIGN |
    TPM_LEFTBUTTON | TPM_RIGHTBUTTON, 32, 64, this);
```

hiển thị thực đơn *pop-up* mà góc trên-trái được đặt vào 32 *pixel* phải và 64 *pixel* trái từ góc trên trái của màn hình. (`TrackPopupMenu` luôn sử dụng tọa độ màn hình thay vì tọa độ cửa sổ). Việc chọn có thể được thực hiện từ thực đơn bằng phím trái hay phải của chuột. Trong khi thực đơn được hiển thị, cửa sổ qui chiếu bằng con trỏ này sẽ nhận thông điệp thực đơn thuộc về thực đơn *pop-up* nổi đúng như đối với thực đơn *pop-up* gắn vào thực đơn *top-level*. Một khi thực đơn biến mất, thông điệp sẽ dừng lại cho đến khi thực đơn hiển thị lại.

Thực đơn ngữ cảnh, như thực đơn *top-level*, có thể nạp từ tài nguyên hay định nghĩa bằng chương trình. Mẫu thực đơn cho thực đơn ngữ cảnh chứa ba mục – *Move*, *Copy*, và *Delete* – như sau đây:

```
IDR_CONTEXTMENU MENU
BEGIN
    POPUP ""
    BEGIN
        MENUITEM "&Move",    IDM_MOVE
        MENUITEM "&Copy",    IDM_COPY
        MENUITEM "&Delete",  IDM_DELETE
    END
END
```

Mã trình để hiển thị nó trên màn hình như sau:

```
CMenu menu;
ClientToScreen (&point);
menu.LoadMenu (IDR_CONTEXTMENU);
menu.GetSubMenu(0)->TrackPopupMenu (TPM_LEFTALIGN |
    TPM_LEFTBUTTON | TPM_RIGHTBUTTON, point.x, point.y, this);
```

Trong thí dụ này, `point` là điểm (trong tọa độ cửa sổ) nơi sẽ xuất hiện thực đơn ngữ cảnh. `ClientToScreen` được gọi để chuyển đổi từ tọa độ cửa sổ sang tọa độ màn hình trước khi hiển thị thực đơn. Tài nguyên thực đơn được nạp nhờ `CMenu::LoadMenu`, và con trỏ đến thực đơn *pop-up* không có tên mà thực đơn con của nó có chỉ số 0, sẽ được nhận lại bằng hàm

CMenu::GetSubMenu. Sau đó hàm TrackPopupMenu được gọi để hiển thị thực đơn. Bằng cách ánh xạ IDM_MOVE, IDM_COPY, và IDM_DELETE đến các hàm *handle* thông qua ánh xạ thông điệp, ứng dụng của ta có thể phản ứng lại việc chọn thực đơn ngữ cảnh. Và bởi vì thực đơn được tạo lập trong khung ngăn xếp, tài nguyên thực đơn được nạp nhờ LoadMenu sẽ tự động giải phóng khi thực đơn ra khỏi miền tác động.

Chương trình mẫu Paint hiển thị thực đơn ngữ cảnh khi nhận thông điệp WM_RBUTTONDOWN thay vì thông điệp WM_CONTEXTMENU. Vì trình được viết bằng MFC 3.1, chúng không có macro ON_WM_CONTEXTMENU cho thông điệp WM_CONTEXTMENU. Nhưng trong ứng dụng MFC 4.0, ta có thể sử dụng ON_WM_CONTEXTMENU thay cho ON_WM_RBUTTONDOWN trong ánh xạ thông điệp của ứng dụng.

5.9 GỘP TẤT CẢ LẠI

Paint (xem chương trình 5.4) là chương trình vẽ thô sơ cho phép ta tạo ảnh *bitmap* trong lưới 256 *pixel* rộng và 256 *pixel* cao. Thanh công cụ trên đỉnh của cửa sổ khung cho phép tham nhập các lệnh *New*, *Open*, *Save*, và *Print* chỉ bằng một nhấn phím. Chỉ có lệnh *New* làm việc còn các lệnh khác thông báo là lệnh chưa cài. Thanh trạng thái ở đáy của cửa sổ (xem hình 5.3) hiển thị trợ giúp cho mục thực đơn và các phím thanh công cụ.

Chương trình 5.4 Paint

Tệp PAINT.RC

```
#include "afxres.h"
#include "resource.h"
//-----
IDR_TOOLBAR          BITMAP MOVEABLE PURE "toolbar.bmp"
IDR_TOOLPALETTE      BITMAP MOVEABLE PURE "palette.bmp"

IDR_MAINMENU MENU DISCARDABLE
BEGIN
    POPUP "&Tệp"
    BEGIN
        MENUITEM "&Tệp mới\Ctrl+N".    IDM_FILE_NEW
        MENUITEM "&Mở tệp...\Ctrl+O".  IDM_FILE_OPEN
        MENUITEM SEPARATOR
        MENUITEM "&Ghi tệp\Ctrl+S".    IDM_FILE_SAVE
        MENUITEM "Ghi &tên mới...".    IDM_FILE_SAVEAS
        MENUITEM SEPARATOR
```

```

MENUITEM "&In...\tCtrl+P".      IDM_FILE_PRINT
MENUITEM SEPARATOR
MENUITEM "&Kết thúc".           IDM_FILE_EXIT
END
POPUP "&Công cụ"
BEGIN
    MENUITEM "Đường &cong".      IDM_TOOLS_FREE
    MENUITEM "&Đường thẳng".     IDM_TOOLS_LINE
    MENUITEM "Hình &tròn".       IDM_TOOLS_CIRC
    MENUITEM "Hình chữ &nhật".   IDM_TOOLS_RECT
    MENUITEM "&Tô màu".         IDM_TOOLS_FILL
END
POPUP "&Màu"
BEGIN
    MENUITEM "Đ&en".             IDM_COLOR_BLACK
    MENUITEM "&Xanh lam".        IDM_COLOR_BLUE
    MENUITEM "Xanh &lá cây".     IDM_COLOR_GREEN
    MENUITEM "&Cổ vịt",          IDM_COLOR_CYAN
    MENUITEM "Đỏ".               IDM_COLOR_RED
    MENUITEM "&Cánh xen".        IDM_COLOR_MAGENTA
    MENUITEM "&Vàng".            IDM_COLOR_YELLOW
    MENUITEM "T&rắng".           IDM_COLOR_WHITE
END
POPUP "&Quan sát"
BEGIN
    MENUITEM "Hiển thị thanh &công cụ".  IDM_VIEW_TOOLBAR
    MENUITEM "Hiển thị thanh &trạng thái". IDM_VIEW_STATUSBAR
    MENUITEM "Hiển thị thanh công cụ &vẽ". IDM_VIEW_PALETTE
END
END
//-----
IDR_ACCELERATORS ACCELERATORS MOVEABLE PURE
BEGIN
    "N".      IDM_FILE_NEW,      VIRTKEY, CONTROL
    "O".      IDM_FILE_OPEN,     VIRTKEY, CONTROL
    "S".      IDM_FILE_SAVE,     VIRTKEY, CONTROL
    "P".      IDM_FILE_PRINT,    VIRTKEY, CONTROL
END
//-----

```

STRINGTABLE DISCARDABLE

BEGIN

AFX_IDS_IDLEMESSAGE "Sẵn sàng"

END

STRINGTABLE DISCARDABLE

BEGIN

IDM_FILE_NEW	"Bắt đầu bản vẽ mới\nTệp mới"
IDM_FILE_OPEN	"Mở bản vẽ đã có\nMở tệp"
IDM_FILE_SAVE	"Ghi lại bản vẽ\nGhi tệp"
IDM_FILE_SAVEAS	"Ghi bản vẽ với tên mới"
IDM_FILE_PRINT	"In bản vẽ\nIn"
IDM_FILE_EXIT	"Kết thúc chương trình"
IDM_TOOLS_FREE	"Chọn công cụ vẽ đường cong\nĐường cong"
IDM_TOOLS_LINE	"Chọn công cụ vẽ đường thẳng\nĐường thẳng"
IDM_TOOLS_CIRC	"Chọn công cụ vẽ hình tròn\nHình tròn"
IDM_TOOLS_RECT	"Chọn công cụ vẽ hình chữ nhật\nHình chữ nhật"
IDM_TOOLS_FILL	"Chọn công cụ tô màu\nTô màu"
IDM_COLOR_BLACK	"Chọn màu vẽ đen"
IDM_COLOR_BLUE	"Chọn màu vẽ xanh lam"
IDM_COLOR_GREEN	"Chọn màu vẽ xanh lá cây"
IDM_COLOR_CYAN	"Chọn màu vẽ cổ vịt"
IDM_COLOR_RED	"Chọn màu vẽ đỏ"
IDM_COLOR_MAGENTA	"Chọn màu vẽ cánh xen"
IDM_COLOR_YELLOW	"Chọn màu vẽ vàng"
IDM_COLOR_WHITE	"Chọn màu vẽ trắng"

END

STRINGTABLE DISCARDABLE

BEGIN

IDM_VIEW_TOOLBAR	"Hiển thị/tắt thanh công cụ"
IDM_VIEW_STATUSBAR	"Hiển thị/tắt thanh trạng thái"
IDM_VIEW_PALETTE	"Hiển thị/tắt thanh công cụ vẽ"

END

//-----

Tệp PAINT.H

#define PENWIDTH 12

class CMyApp : public CWinApp

```

{
public:
    BOOL InitInstance ();
};
//-----
class CMainWindow : public CFrameWnd
{
private:
    BOOL      m_bDirty;
    UINT      m_nColor;
    UINT      m_nTool;
    CRect      m_rect;
    CPoint     m_pointFrom;
    CPoint     m_pointTo;
    CBitmap     m_bitmap;
    CDC        m_memDC;
    CToolBar    m_toolBar;
    CToolBar    m_toolPalette;
    CStatusBar  m_statusBar;
    void CreateToolbar ();
    void CreateToolPalette ();
    void CreateStatusBar ();
    void RubberBandLine (CDC*, CPoint);
    void RubberBandCirc (CDC*, CPoint);
    void RubberBandRect (CDC*, CPoint);
    void DrawFree (CDC*, CPoint);
    void DrawLine (CDC*, CPoint);
    void DrawCirc (CDC*, CPoint);
    void DrawRect (CDC*, CPoint);
    void DoFloodFill (CPoint);
    int CreateMemoryDC ();
    void EraseContents ();
    COLORREF GetCurrentColor ();
    CPoint ConvertPoint (CPoint);
    CRect ConvertRect (CRect);
    void ErrorUnimplemented ();
public:
    CMainWindow ();
protected:

```

```

BOOL PreCreateWindow (CREATESTRUCT&);
afx_msg int OnCreate (LPCREATESTRUCT);
afx_msg void OnPaint ();
afx_msg void OnFileNew ();
afx_msg void OnFileOpen ();
afx_msg void OnFileSave ();
afx_msg void OnFileSaveAs ();
afx_msg void OnFilePrint ();
afx_msg void OnFileExit ();
afx_msg void OnSelectTool (UINT);
afx_msg void OnSelectColor (UINT);
afx_msg void OnViewToolBar ();
afx_msg void OnViewStatusBar ();
afx_msg void OnViewToolPalette ();
afx_msg void OnUpdateNewUI (CCmdUI*);
afx_msg void OnUpdateSaveUI (CCmdUI*);
afx_msg void OnUpdateSaveAsUI (CCmdUI*);
afx_msg void OnUpdatePrintUI (CCmdUI*);
afx_msg void OnUpdateToolsUI (CCmdUI*);
afx_msg void OnUpdateColorUI (CCmdUI*);
afx_msg void OnUpdateToolBarUI (CCmdUI*);
afx_msg void OnUpdateStatusBarUI (CCmdUI*);
afx_msg void OnUpdateToolPaletteUI (CCmdUI*);
afx_msg void OnLButtonDown (UINT, CPoint);
afx_msg void OnMouseMove (UINT, CPoint);
afx_msg void OnLButtonUp (UINT, CPoint);
afx_msg void OnRButtonUp (UINT, CPoint);
DECLARE_MESSAGE_MAP ()
};
//-----

```

Tệp RESOURCE.H

```

#define IDM_FILE_NEW      100
#define IDM_FILE_OPEN     101
#define IDM_FILE_SAVE     102
#define IDM_FILE_SAVEAS   103
#define IDM_FILE_PRINT    104
#define IDM_FILE_EXIT     105
#define IDM_TOOLS_FREE    110

```



```

#define IDM_TOOLS_LINE    111
#define IDM_TOOLS_CIRC    112
#define IDM_TOOLS_RECT    113
#define IDM_TOOLS_FILL    114
#define IDM_COLOR_BLACK   120
#define IDM_COLOR_BLUE    121
#define IDM_COLOR_GREEN   122
#define IDM_COLOR_CYAN    123
#define IDM_COLOR_RED     124
#define IDM_COLOR_MAGENTA 125
#define IDM_COLOR_YELLOW  126
#define IDM_COLOR_WHITE   127
#define IDM_VIEW_TOOLBAR  130
#define IDM_VIEW_STATUSBAR 131
#define IDM_VIEW_PALETTE  132
#define IDR_MAINMENU      200
#define IDR_ACCELERATORS  201
#define IDR_TOOLBAR        202
#define IDR_TOOLPALETTE   203
#define IDW_TOOLBAR        300
#define IDW_TOOLPALETTE   301

```

```
//-----
```

Tệp PAINT.CPP

```

#include <afxwin.h>
#include <afxext.h>
#include "resource.h"
#include "paint.h"

CMyApp myApp;
// Hàm thành phần của CMyApp
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}

// ánh xạ thông điệp và hàm thành phần của CMainWindow
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)

```

```

ON_WM_CREATE ()
ON_WM_PAINT ()
ON_COMMAND (IDM_FILE_NEW, OnFileNew)
ON_COMMAND (IDM_FILE_OPEN, OnFileOpen)
ON_COMMAND (IDM_FILE_SAVE, OnFileSave)
ON_COMMAND (IDM_FILE_SAVEAS, OnFileSaveAs)
ON_COMMAND (IDM_FILE_PRINT, OnFilePrint)
ON_COMMAND (IDM_FILE_EXIT, OnFileExit)
ON_COMMAND_RANGE (IDM_TOOLS_FREE, IDM_TOOLS_FILL, OnSelectTool)
ON_COMMAND_RANGE (IDM_COLOR_BLACK, IDM_COLOR_WHITE, OnSelectColor)
ON_COMMAND (IDM_VIEW_TOOLBAR, OnViewToolbar)
ON_COMMAND (IDM_VIEW_STATUSBAR, OnViewStatusBar)
ON_COMMAND (IDM_VIEW_PALETTE, OnViewToolPalette)
ON_UPDATE_COMMAND_UI (IDM_FILE_NEW, OnUpdateNewUI)
ON_UPDATE_COMMAND_UI (IDM_FILE_SAVE, OnUpdateSaveUI)
ON_UPDATE_COMMAND_UI (IDM_FILE_SAVEAS, OnUpdateSaveAsUI)
ON_UPDATE_COMMAND_UI (IDM_FILE_PRINT, OnUpdatePrintUI)
ON_UPDATE_COMMAND_UI_RANGE (IDM_TOOLS_FREE, IDM_TOOLS_FILL, OnUpdateToolsUI)
ON_UPDATE_COMMAND_UI_RANGE (IDM_COLOR_BLACK,
                             IDM_COLOR_WHITE, OnUpdateColorUI)
ON_UPDATE_COMMAND_UI (IDM_VIEW_TOOLBAR, OnUpdateToolbarUI)
ON_UPDATE_COMMAND_UI (IDM_VIEW_STATUSBAR, OnUpdateStatusBarUI)
ON_UPDATE_COMMAND_UI (IDM_VIEW_PALETTE, OnUpdateToolPaletteUI)
ON_WM_LBUTTONDOWN ()
ON_WM_MOUSEMOVE ()
ON_WM_LBUTTONUP ()
ON_WM_RBUTTONUP ()
END_MESSAGE_MAP ()

//-----
CMainWindow::CMainWindow ()
{
    m_bDirty = FALSE;
    m_nColor = IDM_COLOR_RED;
    m_nTool = IDM_TOOLS_FREE;
    Create (NULL, "Paint", WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
            rectDefault, NULL, MAKEINTRESOURCE (IDR_MAINMENU));
    LoadAccelTable (MAKEINTRESOURCE (IDR_ACCELERATORS));
}

//-----

```

```

BOOL CMainWindow::PreCreateWindow (CREATESTRUCT& cs)
{
    if (!CFrameWnd::PreCreateWindow (cs))    return FALSE;
    cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
    return TRUE;
}

//-----
int CMainWindow::OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate (lpCreateStruct) == -1)    return -1;
    if (CreateMemoryDC () == -1)    return -1;
    CreateToolBar ();
    CreateToolPalette ();
    CreateStatusBar ();
    EnableDocking (CBRS_FLOAT_MULTI);
    CPoint point ( 28, 128);
    ClientToScreen (&point);
    FloatControlBar (&m_toolPalette, point);
    CRect rect;
    RecalcLayout ();
    m_toolBar.GetWindowRect (&rect);
    m_rect.SetRect (16, rect.Height () + 16, 272, rect.Height () + 272);
    return 0;
}

//-----
void CMainWindow::OnPaint ()
{
    CPaintDC dc (this);
    dc.SelectStockObject (NULL_BRUSH);
    dc.Rectangle (m_rect.left - 1, m_rect.top - 1, m_rect.right + 1, m_rect.bottom + 1);
    dc.BitBlt (m_rect.left, m_rect.top, 256, 256, &m_memDC, 0, 0, SRCCOPY);
}

//-----
void CMainWindow::OnFileNew ()
{
    EraseContents ();
    InvalidateRect (&m_rect, FALSE);
    m_bDirty = FALSE;
}

```

```

//-----
void CMainWindow::OnFileOpen ()
{
    ErrorUnimplemented ();
}
//-----
void CMainWindow::OnFileSave ()
{
    ErrorUnimplemented ();
}
//-----
void CMainWindow::OnFileSaveAs ()
{
    ErrorUnimplemented ();
}
//-----
void CMainWindow::OnFilePrint ()
{
    ErrorUnimplemented ();
}
//-----
void CMainWindow::OnFileExit ()
{
    SendMessage (WM_CLOSE, 0, 0);
}
//-----
void CMainWindow::OnSelectTool (UINT nID)
{
    m_nTool = nID;
}
//-----
void CMainWindow::OnSelectColor (UINT nID)
{
    m_nColor = nID;
}
//-----
void CMainWindow::OnViewToolbar ()
{
    if (m_toolBar.IsWindowVisible ()) {

```

```

ShowControlBar (&m_toolBar, FALSE, FALSE);
m_rect.top = 16;
m_rect.bottom = 272;
}
else {
    ShowControlBar (&m_toolBar, TRUE, FALSE);
    CRect rect;
    m_toolBar.GetWindowRect (&rect);
    m_rect.OffsetRect (0, rect.Height ());
}
Invalidate ();
}
//-----
void CMainWindow::OnViewStatusBar ()
{
    ShowControlBar (&m_statusBar, m_statusBar.IsWindowVisible () ? FALSE : TRUE, FALSE);
}
//-----
void CMainWindow::OnViewToolPalette ()
{
    ShowControlBar (&m_toolPalette, m_toolPalette.IsWindowVisible () ? FALSE : TRUE, FALSE);
}
//-----
void CMainWindow::OnUpdateNewUI (CCmdUI* pCmdUI)
{
    pCmdUI->Enable (m_bDirty);
}
//-----
void CMainWindow::OnUpdateSaveUI (CCmdUI* pCmdUI)
{
    pCmdUI->Enable (m_bDirty);
}
//-----
void CMainWindow::OnUpdateSaveAsUI (CCmdUI* pCmdUI)
{
    pCmdUI->Enable (m_bDirty);
}
//-----
void CMainWindow::OnUpdatePrintUI (CCmdUI* pCmdUI)

```

```

{
    pCmdUI->Enable (m_bDirty);
}
//-----
void CMainWindow::OnUpdateToolsUI (CCmdUI* pCmdUI)
{
    pCmdUI->SetRadio (pCmdUI->m_nID == m_nTool);
}
//-----
void CMainWindow::OnUpdateColorUI (CCmdUI* pCmdUI)
{
    pCmdUI->SetRadio (pCmdUI->m_nID == m_nColor);
}
//-----
void CMainWindow::OnUpdateToolBarUI (CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck (m_toolBar.IsWindowVisible ());
}
//-----
void CMainWindow::OnUpdateStatusBarUI (CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck (m_statusBar.IsWindowVisible ());
}
//-----
void CMainWindow::OnUpdateToolPaletteUI (CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck (m_toolPalette.IsWindowVisible ());
}
//-----
void CMainWindow::OnLButtonDown (UINT nFlags, CPoint point)
{
    if (m_rect.PtInRect (point)) {
        if (m_nTool == IDM_TOOLS_FILL)
            DoFloodFill (point);
        else {
            m_pointFrom = point;
            m_pointTo = point;
            if (m_nTool == IDM_TOOLS_FREE) {
                CClientDC dc (this);
            }
        }
    }
}

```

```

        CRgn rgn;
        rgn.CreateRectRgn (m_rect.left, m_rect.top, m_rect.right, m_rect.bottom);
        dc.SelectClipRgn (&rgn);
        DrawFree (&dc, point);
    }
    SetCapture ();
}
}
}
//-----
void CMainWindow::OnMouseMove (UINT nFlags, CPoint point)
{
    if (GetCapture () == this) {
        CClientDC dc (this);
        CRgn rgn;
        rgn.CreateRectRgn (m_rect.left, m_rect.top, m_rect.right, m_rect.bottom);
        dc.SelectClipRgn (&rgn);
        switch (m_nTool) {
            case IDM_TOOLS_FREE:
                DrawFree (&dc, point);
                break;
            case IDM_TOOLS_LINE:
                RubberBandLine (&dc, point);
                break;
            case IDM_TOOLS_CIRC:
                RubberBandCirc (&dc, point);
                break;
            case IDM_TOOLS_RECT:
                RubberBandRect (&dc, point);
                break;
        }
    }
}
//-----
void CMainWindow::OnLButtonUp (UINT nFlags, CPoint point)
{
    if (GetCapture () == this) {
        ReleaseCapture ();
        CClientDC dc (this);
    }
}

```

```

CRgn rgn;
rgn.CreateRectRgn (m_rect.left, m_rect.top, m_rect.right, m_rect.bottom);
dc.SelectClipRgn (&rgn);
switch (m_nTool) {
case IDM_TOOLS_LINE:
    DrawLine (&dc, point);
    break;
case IDM_TOOLS_CIRC:
    DrawCirc (&dc, point);
    break;
case IDM_TOOLS_RECT:
    DrawRect (&dc, point);
    break;
}
}

//-----
void CMainWindow::OnRButtonUp (UINT nFlags, CPoint point)
{
    if (m_rect.PtInRect (point))
    {
        ClientToScreen (&point);
        CMenu* pContextMenu = GetMenu ()->GetSubMenu (2);
        pContextMenu->TrackPopupMenu (TPM_LEFTALIGN | TPM_LEFTBUTTON
                                     | TPM_RIGHTBUTTON, point.x, point.y, this);
    }
}

//-----
void CMainWindow::CreateToolBar ()
{
    static UINT nButtonIDs[] = {
        IDM_FILE_NEW,
        IDM_FILE_OPEN,
        ID_SEPARATOR,
        IDM_FILE_SAVE,
        ID_SEPARATOR,
        IDM_FILE_PRINT
    };
    m_toolBar.Create (this, WS_CHILD | WS_VISIBLE |

```



```

        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_TOP, IDW_TOOLBAR);
m_toolBar.LoadBitmap (IDR_TOOLBAR);
m_toolBar.SetButtons (nButtonIDs, 6);
}

//-----
void CMainWindow::CreateToolPalette ()
{
    static UINT nButtonIDs[] = {
        IDM_TOOLS_FREE,
        IDM_TOOLS_LINE,
        IDM_TOOLS_CIRC,
        IDM_TOOLS_RECT,
        IDM_TOOLS_FILL,
    };
    m_toolPalette.Create (this, WS_CHILD | WS_VISIBLE |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_TOP, IDW_TOOLPALETTE);
    m_toolPalette.LoadBitmap (IDR_TOOLPALETTE);
    m_toolPalette.SetButtons (nButtonIDs, 5);
    m_toolPalette.EnableDocking (CBRS_FLOAT_MULTI);
}

//-----
void CMainWindow::CreateStatusBar ()
{
    m_statusBar.Create (this);
    UINT nIndicator = ID_SEPARATOR;
    m_statusBar.SetIndicators (&nIndicator, 1);
}

//-----
void CMainWindow::RubberBandLine (CDC* pDC, CPoint point)
{
    pDC->SetROP2 (R2_XORPEN);
    pDC->SelectStockObject (WHITE_PEN);
    pDC->MoveTo (m_pointFrom);
    pDC->LineTo (m_pointTo);
    m_pointTo = point;
    pDC->MoveTo (m_pointFrom);
    pDC->LineTo (m_pointTo);
}

//-----

```

```

void CMainWindow::RubberBandCirc (CDC* pDC, CPoint point)
{
    pDC->SetROP2 (R2_XORPEN);
    pDC->SelectStockObject (WHITE_PEN);
    pDC->SelectStockObject (NULL_BRUSH);
    CRect rect (m_pointFrom.x, m_pointFrom.y, m_pointTo.x, m_pointTo.y);
    pDC->Ellipse (rect);
    m_pointTo = point;
    rect.SetRect (m_pointFrom.x, m_pointFrom.y, m_pointTo.x, m_pointTo.y);
    pDC->Ellipse (rect);
}

//-----

void CMainWindow::RubberBandRect (CDC* pDC, CPoint point)
{
    pDC->SetROP2 (R2_XORPEN);
    pDC->SelectStockObject (WHITE_PEN);
    pDC->SelectStockObject (NULL_BRUSH);
    CRect rect (m_pointFrom.x, m_pointFrom.y, m_pointTo.x, m_pointTo.y);
    pDC->Rectangle (rect);
    m_pointTo = point;
    rect.SetRect (m_pointFrom.x, m_pointFrom.y, m_pointTo.x, m_pointTo.y);
    pDC->Rectangle (rect);
}

//-----

void CMainWindow::DrawFree (CDC* pDC, CPoint point)
{
    CPen pen (PS_SOLID, PENWIDTH, GetCurrentColor ());
    CPen* pOldPen = pDC->SelectObject (&pen);
    pDC->MoveTo (m_pointFrom);
    pDC->LineTo (point);
    pDC->SelectObject (pOldPen);
    pOldPen = m_memDC.SelectObject (&pen);
    m_memDC.MoveTo (ConvertPoint (m_pointFrom));
    m_memDC.LineTo (ConvertPoint (point));
    m_memDC.SelectObject (pOldPen);
    m_pointFrom = point;
    m_bDirty = TRUE;
}

//-----

```

```

void CMainWindow::DrawLine (CDC* pDC, CPoint point)
{
    CPen pen (PS_SOLID, PENWIDTH, GetCurrentColor ());
    CPen* pOldPen = pDC->SelectObject (&pen);
    pDC->MoveTo (m_pointFrom);
    pDC->LineTo (m_pointTo);
    pDC->SelectObject (pOldPen);
    pOldPen = m_memDC.SelectObject (&pen);
    m_memDC.MoveTo (ConvertPoint (m_pointFrom));
    m_memDC.LineTo (ConvertPoint (m_pointTo));
    m_memDC.SelectObject (pOldPen);
    m_bDirty = TRUE;
}

//-----
void CMainWindow::DrawCirc (CDC* pDC, CPoint point)
{
    CPen pen (PS_SOLID, PENWIDTH, GetCurrentColor ());
    CPen* pOldPen = pDC->SelectObject (&pen);
    pDC->SelectStockObject (NULL_BRUSH);
    CRect rect (m_pointFrom.x, m_pointFrom.y, point.x, point.y);
    pDC->Ellipse (rect);
    pDC->SelectObject (pOldPen);
    pOldPen = m_memDC.SelectObject (&pen);
    CBrush* pOldBrush = (CBrush*) m_memDC.SelectStockObject (NULL_BRUSH);
    m_memDC.Ellipse (ConvertRect (rect));
    m_memDC.SelectObject (pOldBrush);
    m_memDC.SelectObject (pOldPen);
    m_bDirty = TRUE;
}

//-----
void CMainWindow::DrawRect (CDC* pDC, CPoint point)
{
    CPen pen (PS_SOLID, PENWIDTH, GetCurrentColor ());
    CPen* pOldPen = pDC->SelectObject (&pen);
    pDC->SelectStockObject (NULL_BRUSH);
    CRect rect (m_pointFrom.x, m_pointFrom.y, point.x, point.y);
    pDC->Rectangle (rect);
    pDC->SelectObject (pOldPen);
    pOldPen = m_memDC.SelectObject (&pen);

```

```

CBrush* pOldBrush = (CBrush*) m_memDC.SelectStockObject (NULL_BRUSH);
m_memDC.Rectangle (ConvertRect (rect));
m_memDC.SelectObject (pOldBrush);
m_memDC.SelectObject (pOldPen);
m_bDirty = TRUE;
}

//-----
void CMainWindow::DoFloodFill (CPoint point)
{
    CClientDC dc (this);
    CBrush brush (GetCurrentColor ());
    CBrush* pOldBrush = dc.SelectObject (&brush);
    CRgn rgn;
    rgn.CreateRectRgn (m_rect.left, m_rect.top, m_rect.right, m_rect.bottom);
    dc.SelectClipRgn (&rgn);
    dc.ExtFloodFill (point.x, point.y, dc.GetPixel (point), FLOODFILLSURFACE);
    dc.SelectObject (pOldBrush);
    pOldBrush = m_memDC.SelectObject (&brush);
    m_memDC.ExtFloodFill (point.x - m_rect.left, point.y - m_rect.top,
        m_memDC.GetPixel (ConvertPoint (point)), FLOODFILLSURFACE);
    m_memDC.SelectObject (pOldBrush);
    m_bDirty = TRUE;
}

//-----
int CMainWindow::CreateMemoryDC ()
{
    CClientDC dc (this);
    if (!m_memDC.CreateCompatibleDC (&dc)) {
        MessageBox ("Unable to create a memory DC", "Error", MB_ICONSTOP | MB_OK);
        return -1;
    }
    if (!m_bitmap.CreateCompatibleBitmap (&dc, 256, 256)) {
        MessageBox ("Unable to create a bitmap", "Error", MB_ICONSTOP | MB_OK);
        return -1;
    }
    EraseContents ();
    return 0;
}

//-----

```

```

void CMainWindow::EraseContents ()
{
    m_memDC.SelectObject (&m_bitmap);
    CBrush brush (RGB (255, 255, 255));
    m_memDC.FillRect (CRect (0, 0, 256, 256), &brush);
}

//-----
COLORREF CMainWindow::GetCurrentColor ()
{
    static COLORREF crColors[8] = {
        RGB ( 0, 0, 0), // đen
        RGB ( 0, 0, 255), // xanh lam
        RGB ( 0, 255, 0), // xanh lá cây
        RGB ( 0, 255, 255), // cỏ vịt
        RGB (255, 0, 0), // đỏ
        RGB (255, 0, 255), // cánh xen
        RGB (255, 255, 0), // vàng
        RGB (255, 255, 255) // trắng
    };
    return crColors[m_nColor - IDM_COLOR_BLACK];
}

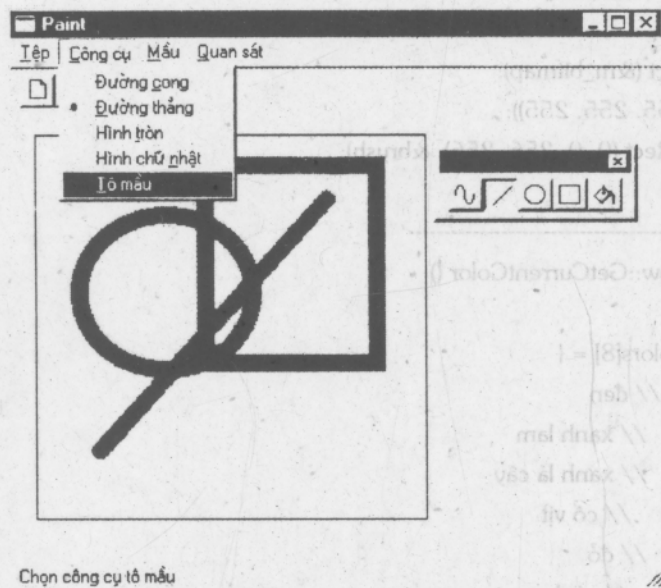
//-----
CPoint CMainWindow::ConvertPoint (CPoint point)
{
    return CPoint (point.x - m_rect.left, point.y - m_rect.top);
}

CRect CMainWindow::ConvertRect (CRect rect)
{
    return CRect (rect.left - m_rect.left, rect.top - m_rect.top,
        rect.right - m_rect.left, rect.bottom - m_rect.top);
}

//-----
void CMainWindow::ErrorUnimplemented ()
{
    MessageBox ("Chức năng chưa cài đặt!", "Error", MB_ICONINFORMATION | MB_OK);
}

//-----

```



Hình 5.3 Kết quả chương trình Paint

Các mục thực đơn còn lại của trình là hoàn toàn hoạt động. Thực đơn *Công cụ* chứa lựa chọn từ năm công cụ vẽ: vẽ tự do, vẽ đường thẳng, vòng tròn, chữ nhật và tô màu. *Công cụ* còn có thể được chọn từ bộ công cụ trôi nổi. Màu vẽ hiện hành được chọn từ thực đơn *Màu* trên đỉnh cửa sổ hay từ thực đơn ngữ cảnh khi nhấn phím phải chuột trên vùng vẽ. Trong thực đơn *Công cụ* và thực đơn màu, công cụ hay màu hiện hành được đánh dấu bằng điểm (bullet). Các mục trong thực đơn *Quan sát* cho phép người sử dụng tắt/mở thanh công cụ, bộ màu công cụ, thanh trạng thái. Thanh công cụ và thanh trạng thái được tắt/mở nhờ hàm `CFrameWnd::ShowControlBar`. Hàm `ShowControlBar` được sử dụng thay cho hàm `ShowWindow` để đảm bảo quản lý chính xác cửa sổ khung nhỏ chứa bộ công cụ trôi nổi.

Khi chạy chương trình Paint, chú ý rằng chọn công cụ từ thực đơn *Công cụ* sẽ tự động đánh dấu phím tương ứng trong bộ công cụ. Hàm quản lý lệnh *Công cụ* (`OnSelectTool`) không làm gì khác ngoài đặt giá trị cho `m_nTool`. Nhưng bộ công cụ được cập nhật ngay tức thì nhờ hàm `OnUpdateToolsUI`. Tương tự, các phím *Tệp mới*, *Ghi tệp* và *In* được đặt có hiệu lực hay không trong mỗi chu kỳ rồi phù hợp với giá trị của `m_bDirty`, chúng được nạp giá trị `FALSE` ban đầu mỗi khi có hành động vẽ. Như vậy, phím sáng ngay khi ta vẽ *pixel* đầu tiên. Paint mô tả một cách tuyệt vời cơ chế UI của MFC tách mã trình làm thay đổi trạng thái ứng dụng từ mã trình cập nhật thành phần UI tương ứng như thế nào.

Ta có thể xem thực đơn, thanh công cụ, thanh trạng thái làm việc cùng nhau như thế nào bằng

cách quan sát thanh trạng thái khi mục thực đơn được chiếu sáng hay con chạy đặt trên phím thanh công cụ hoặc bộ công cụ. Thanh trạng thái hiển thị trợ giúp ngữ cảnh cho các mục thực đơn, và khi *ToolTip* xuất hiện trên phím, xâu ký tự chi tiết hơn sẽ xuất hiện trên thanh trạng thái. Khung ứng dụng MFC thực hiện các công việc này. Ta chỉ việc cung cấp tài nguyên xâu ký tự và khung ứng dụng đảm bảo chúng xuất hiện vào đúng vị trí và đúng thời điểm.

Vì thực đơn màu *pop-up* kích hoạt nhờ phím phải chuột tương đương thực đơn *Pop-up* trong thực đơn chính, hàm quản lý thông điệp `OnRButtonUp` của `Paint` "mượn" thực đơn *Màu* nhờ sử dụng hàm `GetMenu` để lấy con trỏ đến thực đơn chính và `GetSubMenu` để lấy con trỏ đến thực đơn *Màu*. **TrackPopupMenu** sau đó được gọi để hiển thị thực đơn ngữ cảnh:

```
ClientToScreen (&point);
CMenu* pContextMenu = GetMenu ()->GetSubMenu (2);
pContextMenu->TrackPopupMenu (TPM_LEFTALIGN |
    TPM_LEFTBUTTON | TPM_RIGHTBUTTON, point.x, point.y, this);
```

Do thực đơn ngữ cảnh cũng sử dụng chỉ danh ID lệnh như thực đơn *Màu* của thực đơn chính, thanh trạng thái cũng hiển thị trợ giúp cho thực đơn ngữ cảnh. Và hàm `OnUpdateColorUI` đánh dấu màu tương ứng mỗi khi thực đơn ngữ cảnh được hiện ra.

Cửa sổ khung `Paint` được tạo với thuộc tính `WS_CLIPCHILDREN` cho nên các *pixel* vẽ trong vùng làm việc của chúng sẽ không vẽ vào thanh công cụ và thanh trạng thái. Điều này không cần thiết trong ứng dụng tài liệu/quan sát vì việc vẽ bị giới hạn trong cửa sổ quan sát, chúng tự động thay đổi kích thước sao cho chúng chứa phần nhìn thấy của vùng làm việc của cửa sổ khung bên ngoài thanh điều khiển. Trong ứng dụng không có kiến trúc tài liệu/quan sát, phụ thuộc vào lập trình viên để đảm bảo ta sẽ không vẽ vào nơi không được phép.

Chú ý cuối cùng liên quan đến mã trình của thí dụ `Paint` là nạp chồng hàm `CFrameWnd::PreCreateWindow` trong tệp `PAINT.CPP`. Trong MFC 3.1, cài đặt mặc định hàm này có thêm thuộc tính mới `WS_EX_CLIENTEDGE` cho cửa sổ khung khi nó biết là chạy trong Windows 95/98. Điều này làm cho biên cửa sổ được thấy như 3D. Trong ứng dụng tài liệu/quan sát, hàm `CFrameWnd::OnCreateClient` bỏ thuộc tính này đi vì `WS_EX_CLIENTEDGE` được cửa sổ quan sát cung cấp. `Paint` không phải là ứng dụng tài liệu/quan sát, vậy `PreCreateWindow` được nạp chồng để tránh cài đặt mặc định là thêm `WS_EX_CLIENTEDGE`. Nếu không làm điều này, vùng làm việc của cửa sổ khung sẽ có khung là biên trũng xuống và thanh trạng thái và thanh công cụ của `Paint` sẽ không uốn xuống theo biên như trong hình 5.3. Ta phải cung cấp hàm nạp chồng `PreCreateWindow` tương tự như khi ta sử dụng cửa sổ khung trong ứng dụng không có kiến trúc tài liệu/quan sát viết cho Windows 95/98 và ta không muốn khung ứng dụng phải thêm kiểu `WS_EX_CLIENTEDGE`.

HỘP THOẠI, TẤM ĐẶC TÍNH VÀ CÁC ĐIỀU KHIỂN

Một trong nhiều lý do để lập trình viên thích sử dụng MFC là khung ứng dụng hướng đối tượng có khả năng xây dựng các hộp thoại và điều khiển. Nếu ta muốn hiển thị hộp thoại lên màn hình thì hãy quên đi hàm `::DialogBox`; thay vào đó, hãy xây dựng lớp hộp thoại phái sinh từ lớp `CDialog` của MFC, tạo lập đối tượng của lớp và để cho MFC thực hiện các công việc còn lại. Lớp `CDialog` thừa hưởng các chức năng của sổ cơ bản từ `CWnd` và thêm một số “gia vị” của riêng ta để cài đặt hộp thoại. Các điều khiển trong hộp thoại là đối tượng có các hàm thành phần để xử lý chúng. Chúng còn có cơ chế thuận tiện thực hiện trao đổi và “kiểm tra” dữ liệu thoại, chúng thực hiện phần lớn việc nhận vào và gửi ra các thông tin của điều khiển cho ta.

Nếu hộp thoại có quá nhiều các điều khiển, ta có thể dễ dàng chuyển qua sử dụng tấm đặc tính (*property sheet*) với các trang thoại theo bảng. MFC trợ giúp tấm đặc tính bằng các lớp `CPropertySheet` và `CPropertyPage`. Với MFC 4.0, tấm đặc tính MFC sử dụng các cài đặt tấm đặc tính có sẵn trong Windows 95/98 và Windows NT™ 3.51/4.0. Nếu ai đã lập trình tấm đặc tính sử dụng giao diện lập trình ứng dụng 32 bit (Win32® API), thì thấy thật khó khăn để cho chúng làm việc đúng. Với MFC thì việc này thật đơn giản, bởi vì MFC quản lý phần lớn các thông báo tấm đặc tính cho ta. Nếu ai đã sử dụng MFC để làm tấm đặc tính thì không còn muốn làm chúng bằng công cụ phát triển phần mềm (*Software Development Kit - SDK*) như trước đây nữa.

Trong chương này, chúng ta sẽ xem xét các lớp hộp thoại, tấm đặc tính của MFC, cũng như các lớp gói điều khiển cửa sổ con như phím bấm, hộp danh sách. Chúng ta cũng xem xét cách tạo hộp thoại hay tấm đặc tính, ghép nối với điều khiển trong hộp thoại hay tấm đặc tính, xây dựng các lớp điều khiển chuyển mang (*portable*) và khép kín có cài đặt ứng xử riêng, sửa đổi các lớp điều khiển dựng sẵn để thêm các chức năng, tạo các phím tự vẽ (*owner-draw*) mà không khó khăn gì.

6.1 HỘP THOẠI VÀ LỚP `CDialog`

Ta biết rằng hộp thoại chia làm hai loại cơ bản: *modal* và *modeless*. Hộp thoại *modal* hướng mọi dữ liệu vào từ bàn phím hay chuột đến chính nó cho đến khi nó bị mất đi. Hộp thoại *modeless* cho phép dữ liệu vào đến chính nó hay đến các cửa sổ khác trên màn hình. MFC gộp các chức năng của thoại *modal* và *modeless* trong `CDialog`. Cấu tử `CDialog` nhận hai đối số: xâu ký tự hay số nguyên là chỉ danh ID tài nguyên và con trỏ `CWnd` chỉ ra cha của hộp thoại, chúng có thể là `NULL`. Chỉ danh ID tài nguyên xác định mẫu (*template*) của hộp thoại. Cũng như trong ứng dụng SDK, khuôn mẫu của hộp thoại được xác định bởi mẫu gộp trong tệp RC của ứng dụng. Sau khi

đối tượng hộp thoại được lập trong bộ nhớ, ứng dụng sẽ lập hộp thoại trên màn hình nhờ gọi hàm **DoModal** hoặc **Create** của đối tượng. Hàm **DoModal** tạo hộp thoại *modal*, trong khi hàm **Create** tạo hộp thoại *modeless*.

Ta có thể chuyển hộp thoại *modal* ứng dụng thành hộp thoại *modal* hệ thống bằng cách chỉ ra kiểu **DS_SYSMODAL** trong mẫu hộp đối thoại. Trong Windows 16 bit, chúng sẽ tạo hộp thoại *modal* có khả năng hướng toàn bộ dữ liệu vào đến tất cả các cửa sổ khác trong hệ thống. Trong môi trường Win32, hộp thoại *modal* hệ thống không tác động các cửa sổ thuộc các tiến trình khác và khác thoại *modal* chuẩn chút ít trừ khi Windows tự động đặt bit **WS_EX_TOPMOST** của thoại *modal* hệ thống để cho nó hiển thị trên các cửa sổ khác.

Thông thường có thể cài đặt hộp thoại bằng cách lập đối tượng trực tiếp từ lớp **CDialog**. Chương trình 6.1 chỉ ra các lệnh tài nguyên cho hộp thoại đơn giản **About** chứa ba điều khiển: hai điều khiển văn bản tĩnh (*static text*) hiển thị thông tin về chương trình và phím **OK**. Việc hiển thị hộp thoại này chỉ cần hai dòng mã lệnh như sau:

```
CDialog dlg ("About", this);
dlg.DoModal ();
```

Tham số **"About"** là tên của mẫu. Tham số *this* tham chiếu đến cửa sổ của hộp thoại. Nếu tham số này là **NULL** (hay không có, thì **NULL** được sử dụng là mặc định), khung ứng dụng sẽ lập ra cửa sổ mức đỉnh của ứng dụng, là cha của hộp thoại.

Chương trình 6.1 Mẫu cho hộp thoại đơn giản **About**

About DIALOG 32, 32, 186, 98

STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU

CAPTION "About Object Builder"

FONT 8, "MS Sans Serif"

BEGIN

CTEXT "Object Builder Version 1.0", -1, 12, 28, 162, 10

CTEXT "Copyright (c) 1999 by me", -1, 12, 40, 162, 10

DEFPUSHBUTTON "OK", IDOK, 68, 72, 50, 14

END

Bởi vì **DoModal** tạo hộp thoại *modal*, cho nên chúng không trở lại nơi gọi cho đến khi hộp thoại mất đi. Giá trị cho lại là giá trị được truyền cho **EndDialog** để phá hủy hộp thoại. Trừ khi ta xây dựng lớp hộp thoại phái sinh riêng và tự gọi **EndDialog**, thì giá trị sẽ luôn là **IDOK** hay **IDCANCEL**, phụ thuộc vào hộp thoại bị mất do nhấn phím **OK** hay **Cancel**. **DoModal** có thể cho lại **IDCANCEL** thậm chí cả khi không có phím **Cancel**, bởi vì nhấn phím **ESC** tương đương nhấn **Cancel**. Chú ý rằng không cần phải viết thủ tục hộp thoại để xử lý các thông điệp đến chúng. **CDialog** cung cấp thủ tục hộp thoại mặc định cho ta, nó biết làm cái gì khi phím bị nhấn. Nếu ta sử

dụng lợi thế của DDX/DDV (sẽ được mô tả sau), chúng còn biết cả cách khởi động các điều khiển hộp thoại để trả lời lại thông điệp WM_INITDIALOG và lấy lại dữ liệu từ các điều khiển khi nhấn phím OK.

Để thay đổi cách mà hộp thoại đáp lại các phím OK và Cancel hay thêm mã lệnh cho các điều khiển khác, ta phải xây dựng lớp hộp thoại phái sinh riêng. CDialog cung cấp ba hàm có thể nạp chồng để khởi động các điều khiển hộp thoại và trả lời lại các phím OK, Cancel bị nhấn, bao gồm: **OnInitDialog**, **OnOK**, và **OnCancel**. Mặc dù mỗi một trong các hàm này tương ứng với một thông điệp hộp thoại, ta không cần ánh xạ thông điệp để xử lý chúng bởi vì CDialog định nghĩa chúng là hàm ảo. Lát nữa chúng ta sẽ xem thí dụ về cách sử dụng các hàm này. Nhưng trước hết ta hãy xem xét các điều khiển xuất hiện trong hộp thoại và các lớp hộp thoại phái sinh ghép nối với nó như thế nào.

6.2 CÁC LỚP ĐIỀU KHIỂN MFC

Từ bản 1.0, Windows đã cung cấp các lớp cửa sổ cho các đối tượng giao diện với người sử dụng UI như phím nhấn, phím radio, hộp danh sách và thanh trượt. Chúng được biết với cái tên *điều khiển cửa sổ con* bởi vì trong quan hệ cha-con chúng chia sẻ với các cửa sổ khác, các cửa sổ đặc biệt này không phải làm lại mỗi khi cần đến phím và thanh trượt. MFC gói các kiểu điều khiển và đặt chúng vào các lớp của C++ được chỉ ra trong bảng 6.1. Các mục trong cột WNDCLASS là tên lớp cửa sổ định nghĩa trước mà các điều khiển lấy làm cơ sở. MFC 4.0 mở rộng các lớp để gộp cả các điều khiển của Windows 95/98.

Bảng 6.1 Các lớp điều khiển của MFC

Kiểu điều khiển	Lớp MFC	WNDCLASS
Phím nhấn, phím radio, hộp đánh dấu và hộp nhóm	Cbutton	"button"
Điều khiển xâu ký tự tĩnh	Cstatic	"static"
Điều khiển soạn thảo	Cedit	"edit"
Hộp danh sách	CListBox	"listbox"
Hộp danh sách đẩy xuống	CComboBox	"combobox"
Các thanh trượt	CScrollBar	"scrollbar"

Một lợi thế mà ghép nối C++ dành cho các điều khiển là mỗi lớp cung cấp các hàm thành phần để xử lý chúng. Thí dụ, lập trình viên C phải đặt số giới hạn các ký tự được nhập vào điều khiển soạn thảo là 32 bằng cách gửi thông điệp EM_LIMITTEXT cho điều khiển như sau:

```
SendMessage (hwndEditCtrl, EM_LIMITTEXT, 32, 0);
```

Giả sử ctlEdit là đối tượng của kiểu CEdit, lập trình viên MFC thực hiện nó như sau:

```
ctlEdit.LimitText (32);
```

Lợi thế khác của việc sử dụng lớp điều khiển trong MFC là ta có thể sử dụng chúng như lớp cơ sở cho các lớp điều khiển riêng của ta. Nếu chỉ muốn điều khiển soạn thảo chấp nhận số thôi ư? Không có vấn đề gì; chỉ việc xây dựng lớp phái sinh CNumEdit từ CEdit, ánh xạ các thông điệp WM_CHAR vào CNumEdit::OnChar, và viết hàm điều khiển OnChar truyền các phím số đến lớp cơ sở và bỏ qua những cái khác. Rất dễ xây dựng ứng xử của điều khiển theo cách này và cũng dễ tạo ra được các điều khiển riêng. Một thí dụ hoàn hảo của điều khiển riêng là phím tự vẽ (*owner-draw*) có thể tự vẽ thay vì uỷ thác cho cha nó. MFC thậm chí làm cho nó có khả năng trả lời các thông báo riêng của chúng, một đặc trưng quý giá nếu ta sẽ xây dựng các lớp điều khiển có thể sử dụng lại có ứng xử độc lập với cha của chúng.

Có hai phương pháp tạo điều khiển từ lớp điều khiển MFC. Phương pháp thứ nhất thường được sử dụng khi điều khiển được tạo trong cửa sổ mức đỉnh, là xây dựng đối tượng điều khiển và gọi hàm Create hay CreateEx của đối tượng đó. Các lệnh sau đây tạo điều khiển phím bấm có tên Reset từ lớp CButton của MFC:

```
CButton ctlPushButton;
ctlPushButton.Create ("Reset",
    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
    CRect (x, y, cx, cy), this, IDC_RESET);
```

Phương pháp thứ hai được sử dụng trong các hộp thoại. Nếu điều khiển được tạo từ mẫu thoại, chúng không cần tạo lại với Create hoặc CreateEx. Tuy nhiên, nếu điều khiển được thêm nhập từ các hàm thành phần của MFC, thì việc xây dựng đối tượng của lớp điều khiển tương ứng tuỳ thuộc vào lập trình viên. Cách làm dễ nhất là sử dụng hàm CWnd::GetDlgItem, chúng nhận ID điều khiển và cho lại con trỏ CWnd. Ta có thể ép (*cast*) con trỏ CWnd thành con trỏ đến kiểu điều khiển tương ứng để đảm bảo thêm nhập an toàn kiểu. Lệnh sau đây lấy con trỏ CButton cho phím trong hộp thoại:

```
CButton* pPushButton=(CButton*) GetDlgItem (IDC_RESET);
```

Bây giờ pPushButton được sử dụng để xử lý điều khiển bằng các hàm thành phần của CButton. Con trỏ do GetDlgItem cho lại có thể là tạm thời không cần lưu lại để sử dụng sau. Thí dụ, ta không muốn gọi GetDlgItem trong hàm điều khiển OnInitDialog và sử dụng lại con trỏ trong OnOK. Có thể là con trỏ sẽ không còn hợp lệ bởi vì MFC sử dụng thời gian rỗi giữa các thông điệp để xoá các thông điệp tạm thời do GetDlgItem và các hàm khác tạo ra. Một cách tránh hạn chế này là gộp các hàm thành phần riêng cho lại qui chiếu đến từng điều khiển trong lớp hộp thoại của ta. Giả sử hộp thoại chứa hai điều khiển soạn thảo với các ID là IDC_NAME và IDC_PHONE. Nếu lớp hộp thoại chứa các hàm thành phần:

```
CEdit& ctlName () { return *(CEdit*) GetDlgItem (IDC_NAME); }
CEdit& ctlPhone () { return *(CEdit*) GetDlgItem (IDC_PHONE); }
```

thì các điều khiển được thêm nhập như sau:

```
ctlName().LimitText(32);
ctlPhone().LimitText(32);
```

Cách làm khác là gộp hai đối tượng CEdit vào lớp hộp thoại và gắn chúng vào điều khiển soạn thảo bằng hàm **CWnd::Attach** trong OnInitDialog:

```
m_ctlName.Attach(::GetDlgItem(IDC_NAME));
m_ctlPhone.Attach(::GetDlgItem(IDC_PHONE));
```

Bây giờ CEdit thêm nhập một cách đơn giản vào điều khiển soạn thảo:

```
m_ctlName.LimitText(32);
m_ctlPhone.LimitText(32);
```

Bởi vì các đối tượng CEdit là thành phần của lớp hộp thoại, ta không phải xoá chúng khi hộp thoại bị phá hủy.

Nếu ta sử dụng lớp điều khiển phái sinh, nó không đủ để gắn một cách đơn giản điều khiển vào đối tượng của lớp phái sinh. Nếu ta gắn đối tượng CNumEdit vào điều khiển soạn thảo được lập từ mẫu thoại như sau:

```
m_ctlPhone.Attach(::GetDlgItem(IDC_PHONE));
```

thì điều khiển vẫn ứng xử như điều khiển CEdit chuẩn bởi vì hàm Attach không chuyển hướng thông điệp điều khiển qua ánh xạ thông điệp của lớp phái sinh. Giải pháp? Thay vì sử dụng *Attach*, hãy gắn điều khiển vào đối tượng CNumEdit nhờ hàm **SubclassDlgItem** của MFC như sau:

```
m_ctlPhone.SubclassDlgItem(IDC_PHONE, this);
```

Bây giờ điều khiển soạn thảo sẽ ứng xử như CNumEdit.

Điều khiển cửa sổ con liên lạc với cha nó bằng cách gửi thông điệp WM_COMMAND. Trong Win32, từ cao của tham số wParam của thông điệp chứa mã cho lại, từ thấp chứa ID của điều khiển, và lParam chứa chỉ danh của sổ điều khiển. Mã cho lại thông báo cho cha biết rằng sự kiện gây ra bởi nhấn phím hay thay đổi ký tự trong điều khiển soạn thảo. Nhận dạng thông báo phím có tiếp đầu ngữ BN_. Các kiểu thông báo khác bao gồm EN_ cho điều khiển soạn thảo, LBN_ cho hộp danh sách, và CBN_ cho điều khiển combo box. Các điều khiển tĩnh không gửi thông báo, và các điều khiển thanh trượt gửi các thông điệp WM_HSCROLL và WM_VSCROLL thay cho thông điệp WM_COMMAND.

MFC cung cấp các macro ánh xạ thông điệp cho các kiểu thông báo khác nhau để cho ta dễ dán các điều khiển vào các hàm thành phần của lớp hộp thoại. Các dòng mã trình sau sử dụng macro ON_BN_CLICKED để nối các phím mà chỉ danh ID của chúng là IDC_RESET đến hàm thành phần CMyDialog với tên OnReset:

```

BEGIN_MESSAGE_MAP (CMyDialog, CDialog)
    ON_BN_CLICKED (IDC_RESET, OnReset)
END_MESSAGE_MAP ()
CMyDialog::OnReset ()
{
    .....
    // Thực hiện công việc khi phím Reset bị nhấn
    .....
}

```

Tài liệu MFC chứa danh sách đầy đủ các thông báo lại của các điều khiển và *macro* ánh xạ thông điệp tương ứng.

Đôi khi ta muốn điều khiển phải xử lý các thông điệp thông báo (*notification*) lại của chính nó. Giả sử ta viết lớp hộp danh sách sử dụng lại để biểu diễn cấu trúc danh mục đĩa cứng. Trong C, thông báo lại LBN_DBLCLK chỉ ra tên danh mục đã được nhấn, được quản lý bởi cha của hộp danh sách, và cha phải có trách nhiệm thay đổi các danh mục, cập nhật hộp danh sách. Nhưng trong MFC, bởi vì khung ứng dụng “phản ánh” thông báo lại trở về điều khiển nơi gửi chúng nhờ hàm ảo OnChildNotify, hộp danh sách có thể tự quản lý nhấn đúp. Sau đây là mã trình của hộp danh sách quản lý thông báo lại LBN_DBLCLK:

```

BOOL CMyListBox::OnChildNotify (UINT message, WPARAM wParam,
                                LPARAM lParam, LRESULT* pLResult)
{
    if (message == WM_COMMAND)
    {
        if (HIWORD (wParam) == LBN_DBLCLK)
        {
            .....
            // Trả lời nhấn đúp
            .....
            return TRUE;
        }
    }
    return FALSE;
}

```

MFC còn phản ánh lại các thông điệp WM_MEASUREITEM, WM_DRAWITEM, WM_COMPAREITEM, và WM_DELETEITEM đến điều khiển phát sinh ra chúng để các điều khiển tự vẽ có thể tự vẽ lại. Đây là đặc trưng mạnh của khung ứng dụng, bởi vì nó cho phép các lớp điều khiển được viết một cách hoàn toàn khép kín. Và điều khiển khép kín dễ dàng chuyển đổi sang các ứng dụng khác, đây là một trong các lý do để viết mã trình hướng đối tượng.

6.3 TẬP HỢP LẠI

Đến bây giờ ta đã có khái niệm về các điều khiển hộp thoại được quản lý như thế nào, hãy quan sát trên hộp thoại để nhập tên, số điện thoại sử dụng điều khiển soạn thảo trên đây. Giả sử rằng mẫu thoại đã lập sẵn, công việc tiếp theo là xây dựng lớp hộp thoại phái sinh từ CDialog. Đoạn mã trình 6.2 chỉ ra mã trình của lớp hộp thoại phái sinh với tên CMyDialog. Khi lớp hộp thoại được tạo và hàm ảo OnInitDialog được gọi, tên và số điện thoại ban đầu (nếu có) được sao chép từ các biến thành phần m_strName và m_strPhone vào các điều khiển soạn thảo. Khi phím OK được nhấn, trình quản lý OnOK của hộp thoại lấy xâu ký tự từ các điều khiển và sao chép chúng trở lại các biến thành phần. CMyDialog không nạp chồng OnCancel, bởi vì cài đặt mặc định OnCancel của lớp CDialog (thực hiện đơn giản việc bãi bỏ hộp thoại và gửi IDCANCEL đến DoModal) thực hiện công việc khá tốt.

Đoạn mã trình 6.2 Lớp hộp thoại phái sinh đơn giản

```

Class CMyDialog : public CDialog
{
private:
    CEdit& ctlName () { return *(CEdit*) GetDlgItem (IDC_NAME); }
    CEdit& ctlPhone () { return *(CEdit*) GetDlgItem (IDC_PHONE); }
public:
    CString m_strName;
    CString m_strPhone;
    CMyDialog (CWnd* pParentWnd = NULL) :
        CDialog ("NameAndPhoneDialog", pParentWnd) {}
    virtual BOOL OnInitDialog ();
protected:
    virtual void OnOK ();
};
BOOL CMyDialog::OnInitDialog ()
{
    // Trước hết gọi lớp cơ sở
    CDialog::OnInitDialog ();
    // Khởi động điều khiển soạn thảo
    ctlName ().LimitText (32);
    ctlPhone ().LimitText (32);
    ctlName ().SetWindowText (m_strName);
    ctlPhone ().SetWindowText (m_strPhone);
    return TRUE;
}
void CMyDialog::OnOK ()

```

```

{
    // Truy nhập dữ liệu từ điều khiển soạn thảo
    ctlName ().GetWindowText (m_strName);
    ctlPhone ().GetWindowText (m_strPhone);
    // Gọi lớp cơ sở để kết thúc hộp thoại
    CDialog::OnOK ();
}

```

Các biến của CString như m_strName và m_strPhone được khai báo là chung (*public*), không phải là riêng (*private*). Làm như vậy để cửa của hộp thoại có thể nhận được dữ liệu vào và ra hộp thoại. Giả sử lớp cửa sổ chính chứa hai trường CString cùng tên m_strName và m_strPhone chứa tên và số, thành phần dữ liệu m_nCurrentEntry chỉ ra tên và số điện thoại “hiện hành”. Sau đây là mã trình hiển thị hộp thoại:

```

CMyDialog dlg (this);
dlg.m_strName = m_strName[m_nCurrentEntry];
dlg.m_strPhone = m_strPhone[m_nCurrentEntry];
if (dlg.DoModal () == IDOK) {
    m_strName[m_nCurrentEntry] = dlg.m_strName;
    m_strPhone[m_nCurrentEntry] = dlg.m_strPhone;
}

```

Tên và số điện thoại được truyền đến hộp thoại bằng cách sao chép tên và số điện thoại hiện hành vào các thành phần dữ liệu m_strName và m_strPhone của thoại. Ngược lại, tên và số điện thoại được lấy ra từ hộp thoại bằng cách sao chép ngược lại các giá trị của các thành phần này khi hộp thoại bị phá hủy do nhấn phím OK. Kỹ thuật này hạn chế được số biến tổng thể thường được sử dụng để truyền dữ liệu đến và đi từ hộp thoại trong các ứng dụng SDK. Dĩ nhiên, các thành phần dữ liệu m_strName và m_strPhone của lớp hộp thoại phải được khai báo là *public* (hoặc lớp cửa sổ chính được khai báo là *friend* của lớp hộp thoại) cho cách tiếp cận này.

Chú ý rằng mã trình trên đây không chỉ ra tên mẫu trong cấu tử thoại, bởi vì cấu tử CMyDialog được xây dựng trong tên mẫu đã chứa trong khai báo lớp. Đó là kỹ thuật mà lập trình viên MFC thường sử dụng để hình thành sự liên kết giữa lớp hộp thoại và mẫu thoại tương ứng.

Lớp CMyDialog phân biệt ra sao khi điều khiển soạn thảo của số điện thoại là điều khiển CNumEdit thay cho CEdit chuẩn? Đoạn mã trình 6.3 của lớp CMyDialog có sử dụng điều khiển CNumEdit. Chúng cũng chỉ ra mã trình của lớp CNumEdit. CMyDialog bây giờ khai báo thành phần dữ liệu riêng CEdit và CNumEdit ánh xạ đến điều khiển soạn thảo trong OnInitDialog. Điều khiển CEdit được ánh xạ bằng Attach, nhưng điều khiển CNumEdit sử dụng SubclassDlgItem, vậy ánh xạ thông điệp CNumEdit sẽ được sử dụng. Bây giờ, điều khiển soạn thảo *số điện thoại* sẽ chỉ chấp nhận các giá trị số bởi vì các giá trị số (cũng như phím *Backspace*, cũng được thông báo qua OnChar) được truyền đến lớp cơ sở để xử lý.

Đoạn mã trình 6.3 Lớp hộp thoại phát sinh có sử dụng điều khiển CNumEdit

```

class CNumEdit : public CEdit
{
protected:
    afx_msg void OnChar (UINT, UINT, UINT);
DECLARE_MESSAGE_MAP ()
};
BEGIN_MESSAGE_MAP (CNumEdit, CEdit)
    ON_WM_CHAR ()
END_MESSAGE_MAP ()
void CNumEdit::OnChar (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    if (((nChar >= '0') && (nChar <= '9')) || (nChar == VK_BACK))
        CEdit::OnChar (nChar, nRepCnt, nFlags);
}

Class CMyDialog : public CDialog
{
private:
    CEdit m_ctlName;
    CNumEdit m_ctlPhone;
public:
    CString m_strName;
    CString m_strPhone;
    CMyDialog (CWnd* pParentWnd = NULL) :
        CDialog ("NameAndPhoneDialog", pParentWnd) {}
    virtual BOOL OnInitDialog ();
protected:
    virtual void OnOK ();
};

BOOL CMyDialog::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    m_ctlName.Attach (::GetDlgItem (IDC_NAME));
    m_ctlPhone.SubclassDlgItem (IDC_PHONE, this);
    m_ctlName.LimitText (32);
    m_ctlPhone.LimitText (32);
    m_ctlName.SetWindowText (m_strName);
    m_ctlPhone.SetWindowText (m_strPhone);
    return TRUE;
}

void CMyDialog::OnOK ()

```



```

{
    m_ctlName.GetWindowText (m_strName);
    m_ctlPhone.GetWindowText (m_strPhone);
    CDialog::OnOK ();
}

```

6.4 TRAO ĐỔI DỮ LIỆU HỘP THOẠI VÀ KIỂM TRA

Thí dụ này cho thấy lập trình viên MFC quản lý hộp thoại *modal* và các điều khiển như thế nào. Thí dụ còn cho thấy các lớp điều khiển chuẩn như **CEdit** có thể thay đổi được nhờ sử dụng chúng như các lớp cơ sở cho các lớp phái sinh. Nhưng còn một điều quan trọng mà ta vẫn chưa bàn luận đến: trao đổi dữ liệu thoại (*Dialog Data Exchange*) và kiểm tra dữ liệu thoại (*Dialog Data Validation*), quen thuộc hơn với tên DDX và DDV.

Khi sử dụng *ClassWizard* của Visual C++ để tạo nên các lớp hộp thoại, ta có thể sử dụng đặc trưng đặc biệt của *ClassWizard* để tạo các thành phần dữ liệu hộp thoại và liên kết chúng với các điều khiển trong hộp thoại. Thật là dễ: chọn chỉ danh ID của điều khiển từ hộp danh sách (*list box*), nhấn phím *Add Variable*, và nạp tên và đặc tính của biến. Khi hộp thoại được lập, các điều khiển được khởi động để kết hợp với giá trị được gán cho các biến thành phần tương ứng. Khi hộp thoại mất đi vì nhấn phím OK, dữ liệu được nhập vào điều khiển sẽ tìm đường trở lại các biến thành phần. Kiểm tra sơ bộ dữ liệu được thực hiện. Thí dụ nếu giá trị nguyên vào từ điều khiển soạn thảo rơi ra ngoài khoảng mà ta định trước, thì hộp thông báo xuất hiện chỉ ra lỗi cho người sử dụng. Khi hộp thông báo mất đi, hộp thoại vẫn còn trên màn hình và tiêu điểm vào được gán cho điều khiển nào phạm lỗi, cùng với *caret* nhấp nháy báo cho người sử dụng nhập lại dữ liệu hợp lý.

Điều bí mật của khả năng liên kết các điều khiển trong hộp thoại với các thành phần dữ liệu của lớp hộp thoại của *ClassWizard* là cơ chế DDX/DDV của khung ứng dụng. *ClassWizard* thêm hàm thành phần với tên *DoDataExchange* vào lớp hộp thoại của ứng dụng chứa một hay nhiều lời gọi đến các hàm DDX/DDV liên quan đến điều khiển và các thành phần dữ liệu. Với hộp thoại đơn giản chỉ chứa một điểm nối DDX/DDV (chỉ có một điều khiển soạn thảo để nhập giá trị dữ liệu nguyên) *DoDataExchange* được mô tả sau đây:

```

void CMyDialog::DoDataExchange (CDataExchange* pDX)
{
    DDX_Text (pDX, IDC_EDIT, m_nIntVal);
    DDV_MinMaxInt (pDX, m_nIntVal, 0, 100);
}

```

Trong đó, IDC_EDIT là chỉ danh ID của điều khiển và m_nIntVal là biến thành phần liên quan với nó. Tham số pDX truyền cho *DoDataExchange* là con trỏ đến đối tượng **CDataExchange** do khung ứng dụng cung cấp. Đối tượng **CDataExchange** cho các hàm *DDX_Text* và

DDV_MinMaxInt biết cách thông tin chuyển đi - dữ liệu được truyền từ thành phần dữ liệu đến điều khiển hay từ điều khiển đến thành phần dữ liệu. Ta coi DoDataExchange là ánh xạ dữ liệu bởi vì nó gắn các điều khiển hộp thoại vào thành phần dữ liệu hộp thoại đúng như ánh xạ thông điệp gắn các thông điệp vào các hàm thành phần lớp.

ClassWizard lập ánh xạ dữ liệu, nhưng khung ứng dụng làm cho nó hoạt động. Khi hộp thoại được tạo, CDialog::OnInitDialog gọi hàm UpdateData mà đối tượng thoại thừa hưởng từ CWnd và khởi động điều khiển hộp thoại được nạp giá trị FALSE. UpdateData, đến lượt nó, tạo đối tượng CDataExchange và gọi hàm DoDataExchange của hộp thoại, truyền cho nó con trỏ đến đối tượng CDataExchange. Mỗi hàm DDX được DoDataExchange gọi sẽ khởi động điều khiển bằng giá trị của biến thành phần. Trong thí dụ trên, m_nIntVal được chuyển từ số nguyên sang dạng chữ và sao chép chúng vào điều khiển soạn thảo nhờ hàm DDX_Text. Cuối cùng, khi nhấn phím OK, CDialog::OnOK gọi UpdateData với tham số TRUE làm cho DDX_Text thực hiện ngược lại, đúng như chúng đã làm trước đó: đọc xâu ký tự từ điều khiển soạn thảo, chuyển nó sang số nguyên, lưu kết quả trong m_nIntVal. Thêm nữa, hàm DDV_MinMaxInt của MFC kiểm tra xem m_nIntVal có ở trong khoảng 0 đến 100 hay không. Nếu số nguyên ở ngoài, DoDataExchange trở lại và hộp thoại biến mất. Mặt khác, hàm Fail của đối tượng CDataExchange được gọi để hiển thị thông điệp cảnh báo và hộp thoại vẫn còn trên màn hình với tiêu điểm vào trên điều khiển soạn thảo.

DDX_Text và DDV_MinMaxInt là hai trong một số hàm do MFC cung cấp; bảng 6.2 và 6.3 chứa danh sách đầy đủ.

Bảng 6.2 Các hàm DDX

Hàm	Mô tả
DDX_Text	Liên kết biến BYTE, int, UINT, long, DWORD, CString, float, hay double với điều khiển soạn thảo.
DDX_Check	Liên kết biến int với hộp đánh dấu. Các giá trị 0, 1, 2 tương ứng với ba trạng thái có thể của hộp đánh dấu là: không đánh dấu, đánh dấu, và trung gian.
DDX_Radio	Liên kết biến int với nhóm phím radio. Giá trị biến là chỉ số tính từ 0 chỉ ra phím đã đánh dấu.
DDX_LBIndex	Liên kết biến int với hộp danh sách. Giá trị biến là chỉ số của mục đang được chọn.
DDX_LBString	Liên kết biến CString với hộp danh sách và bỏ qua trường hợp khi đặt lựa chọn. Giá trị của biến là xâu ký tự của mục đang được chọn.
DDX_LBStringExact	Liên kết biến CString với hộp danh sách. Giá trị của biến là xâu ký tự của mục đang được chọn.
DDX_CBIndex	Liên kết biến int với hộp combo. Giá trị biến là chỉ số của mục đang được chọn.
DDX_CBString	Liên kết biến CString với hộp combo và bỏ qua trường hợp khi đặt lựa chọn. Giá trị của biến là xâu ký tự của mục đang được chọn.
DDX_CBStringExact	Liên kết biến CString với hộp combo. Giá trị của biến là xâu ký tự của mục đang được chọn.
DDX_Scroll	Liên kết int với thanh trượt. Giá trị của biến tương ứng với vị trí con chạy của thanh trượt.

Bảng 6.3 Các hàm DDV

Hàm	Mô tả
DDV_MinMaxByte	Kiểm tra giá trị biến <i>BYTE</i> có ở trong giá trị giới hạn không.
DDV_MinMaxInt	Kiểm tra giá trị biến <i>int</i> có ở trong giá trị giới hạn không.
DDV_MinMaxLong	Kiểm tra giá trị biến <i>long</i> có ở trong giá trị giới hạn không.
DDV_MinMaxUInt	Kiểm tra giá trị biến <i>UINT</i> có ở trong giá trị giới hạn không.
DDV_MinMaxDWord	Kiểm tra giá trị biến <i>DWORD</i> có ở trong giá trị giới hạn không.
DDV_MinMaxFloat	Kiểm tra giá trị biến <i>float</i> có ở trong giá trị giới hạn không.
DDV_MinMaxDouble	Kiểm tra giá trị biến <i>double</i> có ở trong giá trị giới hạn không.
DDV_MaxChars	Kiểm tra giá trị biến <i>CString</i> không chứa nhiều hơn tổng số ký tự xác định trước hay không.

Cú pháp của các hàm ở trong tệp *header* *AFXDD_.H* của MFC. Hàm *DDX_Text*, chúng truyền ký tự và dữ liệu số đến và đi khỏi điều khiển soạn thảo, chấp nhận nhiều loại dữ liệu khác nhau. Các hàm khác của *DDX* chỉ chấp nhận một loại dữ liệu, thí dụ biến thành phần liên kết với điều khiển hộp đánh dấu (*check box*) phải là *int*. Giá trị nguyên 0, 1 hoặc 2 tương ứng với trạng thái không đánh dấu, đánh dấu hay không xác định của hộp đánh dấu. (Trạng thái không xác định chỉ áp dụng cho hộp đánh dấu ba trạng thái). *DDX_Radio* liên kết nhóm phím *radio* với biến thành phần đặc biệt. Tại đầu vào, biến lưu chỉ số tính từ 0 chỉ ra phím *radio* nào được chọn. Tại đầu ra, chúng lưu chỉ số của phím đang được chọn. Chỉ danh ID của điều khiển được truyền cho *DDX_Radio* chỉ ra phím đầu tiên trong nhóm và MFC đặt cuối nhóm khi chương trình chạy bằng cách quét cho đến khi thấy thuộc tính *WS_GROUP* đánh dấu bắt đầu của nhóm khác. *DDX_LBIndex*, *DDX_LBString*, *DDX_LBStringExact* nhận và đặt lựa chọn trong hộp danh sách nhờ chỉ số nguyên hay giá trị *CString*, và *DDX_CBIndex*, *DDX_CBString*, *DDX_CBStringExact* cũng làm như vậy với hộp *combo*. Cuối cùng, *DDX_Scroll* thực hiện liên kết giữa thanh trượt và thành phần dữ liệu thoại có kiểu *int*. Giá trị của số nguyên kiểu *int* tương ứng với vị trí của con chạy thanh trượt.

Rất cuộc *DDX/DDV* là cách nhẹ nhàng nhất để lấy số liệu vào và ra từ các điều khiển của hộp thoại và thực hiện một số thủ tục đánh giá. Đoạn mã trình 6.4 chỉ ra cách *DDX* và *DDV* đơn giản hoá mã trình cho lớp *CMyDialog* đã được mô tả trên. Khung ứng dụng bây giờ truyền dữ liệu giữa các thành phần *m_strName*, *m_strPhone* và điều khiển soạn thảo, cho nên *OnInitDialog* và *OnOK* không cần nạp chống nữa. Hơn nữa, khung ứng dụng kiểm soát xem xâu nhập vào điều khiển có dài hơn 32 ký tự hay không và hiển thị thông điệp lỗi nếu xâu vượt quá giới hạn. So sánh mã trình 6.4 với mã trình tương đương nhưng không phải là *DDX/DDV* trong mã trình 6.3, ta sẽ thấy chính xác khung ứng dụng làm thay ta những gì. Chú ý rằng lời gọi hàm *DDV* kiểm tra dữ liệu của điều khiển được đặt ngay sau các lời gọi *DDX* cho cùng điều khiển. Điều này là quan trọng bởi vì nó cho phép hàm *DDV* đặt tiêu điểm (*focus*) vào đến điều khiển có lỗi xảy ra.

Đoạn mã trình 6.4 Viết lại lớp CMyDialog để sử dụng DDX và DDV

```

Class CMyDialog : public CDialog
{
public:
    CString m_strName;
    CString m_strPhone;
    CMyDialog (CWnd* pParentWnd = NULL) :
        CDialog ("NameAndPhoneDialog", pParentWnd) {}
protected:
    virtual void DoDataExchange (CDataExchange*);
};

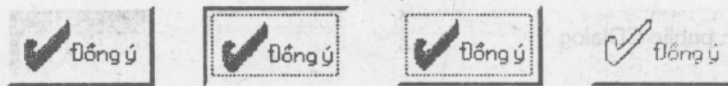
void CMyDialog::DoDataExchange (CDataExchange* pDX)
{
    DDX_Text (pDX, IDC_NAME, m_strName);
    DDV_MaxChars (pDX, m_strName, 32);
    DDX_Text (pDX, IDC_PHONE, m_strPhone);
    DDV_MaxChars (pDX, m_strPhone, 32);
}

```

6.5 LÀM PHÍM BITMAP

Cho đến bây giờ, ý tưởng về việc ứng xử của điều khiển có thể tự thực hiện bằng lớp phái sinh mới từ lớp điều khiển MFC có thể được ghi sâu trong trí não ta. Để làm cho công việc của ta dễ dàng hơn, MFC cung cấp một lớp điều khiển thuận tiện suy diễn từ lớp của chúng. **CBitmapButton** mở rộng phạm vi của lớp **CButton** để hoàn thiện các phím nhấn tự vẽ luôn là những việc lặt vặt khi viết trong C. Với **CBitmapButton** trong tay, các phím nhấn với ảnh trên chúng làm tăng hấp dẫn của ứng dụng và được sử dụng rộng rãi trong các sản phẩm phần mềm thương mại, được dễ dàng lập và quản lý như các phím chỉ có ký tự. Lý do là khung ứng dụng quản lý thông điệp tự vẽ cho ta. Ta chỉ cần cung cấp *bitmap* biểu diễn phím trong các trạng thái khác nhau của chúng.

Tạo phím *bitmap* là đơn giản. Bước thứ nhất là sử dụng chương trình soạn thảo *bitmap* để tạo *bitmap*. **CBitmapButton** chấp nhận cho đến bốn ảnh *bitmap* (xem hình 6.1): thứ nhất miêu tả phím trạng thái bình thường, trạng thái không nhấn; thứ hai mô tả phím bị nhấn; thứ ba mô tả phím không bị nhấn và có tiêu điểm vào; và thứ tư mô tả phím bị vô hiệu (hay “mờ”). Tối thiểu, ta phải cung cấp các *bitmap* “lên” và “xuống” chỉ ra phím ở trạng thái bình thường và nhấn. Khi phím bị nhấn, khung ứng dụng vẽ lại phím *bitmap* xuống làm cho phím như là bị nhấn sâu trong màn hình. Khi bỏ phím ra, nó được vẽ lại *bitmap* lên. Các *bitmap* “liều cự” và “vô hiệu” là tùy ý. Ta có thể cung cấp *bitmap* tiêu cự khi cần. Ta chỉ cần cung cấp *bitmap* vô hiệu lực nếu ta cho phép phím trở nên vô hiệu.



Hình 6.1: Các *Bitmap* tạo phím Đồng ý

Sau khi *bitmap* của phím được tạo và lưu trên tệp như kiểu BMP, bước tiếp theo là chuyển chúng thành tài nguyên *bitmap* bằng cách gộp vào tệp RC của chương trình.

```
CancelU BITMAP cancelu.bmp
CancelD BITMAP canceld.bmp
CancelF BITMAP cancelf.bmp
CancelX BITMAP cancelx.bmp
```

Trong đó, tên tài nguyên: CancelU là *bitmap* lên, CancelD là *bitmap* xuống, CancelF là *bitmap* có tiêu điểm, CancelX là *bitmap* vô hiệu lực. Phần của tên tài nguyên đi trước các chữ này phải trùng với ký tự gán cho phím. Quy định đặt tên này không hoàn toàn cần thiết khi ta tạo phím *bitmap* trong cửa sổ mức đỉnh, nhưng nó rất quan trọng cho các phím *bitmap* của các hộp thoại.

Bước thứ ba và bước cuối cùng là tạo phím và nạp *bitmap* phím. Trong hộp thoại, nó được thực hiện trong một bước nhờ hàm CBitmapButton::AutoLoad, chúng gọi CBitmapButton::LoadBitmaps để nạp *bitmap* bất kể phím nhấn trong các trạng thái khác nhau và SubclassDlgItem để phân lớp điều khiển phím đang tồn tại. *AutoLoad* tạo ra tên tài nguyên cho *bitmap* bằng cách tổ hợp các ký tự của phím với U, D, F, hay X. Kết quả là, *AutoLoad* chuyển phím chuẩn tạo ra từ mẫu thoại thành phím *bitmap* do CBitmapButton quản lý. Nếu mẫu thoại gộp đôi phím nhấn mà nó được định nghĩa như sau:

```
DEFPUSHBUTTON "OK", IDOK, 68, 72, 50, 14, BS_OWNERDRAW
PUSHBUTTON "Cancel", IDCANCEL, 126, 72, 50, 14, BS_OWNERDRAW
```

và nếu lớp hộp thoại gộp các đối tượng CBitmapButton được khai báo như sau:

```
CBitmapButton m_ctlOKButton;
CBitmapButton m_ctlCancelButton;
```

thì các phím có thể được chuyển thành phím *bitmap* trong OnInitDialog bằng các lệnh:

```
m_ctlOKButton.AutoLoad (IDOK, this);
m_ctlCancelButton.AutoLoad (IDCANCEL, this);
```

Các lệnh tài nguyên xác định phím chỉ ra kiểu BS_OWNERDRAW. Nếu ta quên gộp BS_OWNERDRAW, *bitmap* của phím sẽ không hiện ra trên màn hình bởi vì thông điệp không được phát sinh.

6.6 ỨNG DỤNG NAP

Chương trình 6.5 chứa mã nguồn của ứng dụng đơn giản với tên NAP (viết tắt của *Names And Phones*). NAP bao gồm cửa sổ khung mà trong vùng làm việc có hộp danh sách hiển thị tên và số điện thoại (xem hình 6.2). Hộp danh sách là đối tượng của lớp `CNapListBox`, chúng được suy diễn từ `CListBox` (xem NAP.H). `CNapListBox` là hộp danh sách khép kín, nó hiển thị hộp thoại báo cho người sử dụng nhập tên và số điện thoại mới khi hàm `PromptForNewEntry` của nó được gọi. `CNapListBox` có lợi thế của cơ chế hàm ảo `CWnd::OnChildNotify` của MFC để phản ứng khi mục của hộp danh sách được chọn (nhấn đúp). Cha của nó quản lý thông báo lại `LBN_DBLCLK`, còn `CNapListBox` tự biểu diễn tên, số điện thoại hiện đang được chọn trong hộp thoại nơi có thể được sửa đổi. Thậm chí nó còn tự khởi động khi xử lý thông điệp `WM_CREATE` và nạp chồng hàm `PreCreateWindow` sao cho cha của nó chỉ phải chỉ ra kiểu của hộp danh sách là `WS_CHILD` và `WS_VISIBLE`. Nói tóm lại, `CNapListBox` là thí dụ hoàn hảo của lớp điều khiển.

Chương trình 6.5 NAP

Tệp NAP.RC

```
#include "afxres.h"
#include "resource.h"
// ảnh bitmap
OKU          BITMAP MOVEABLE PURE "oku.bmp"
OKD          BITMAP MOVEABLE PURE "okd.bmp"
OKF          BITMAP MOVEABLE PURE "okf.bmp"
OKX          BITMAP MOVEABLE PURE "okx.bmp"
CANCELU     BITMAP MOVEABLE PURE "cancelu.bmp"
CANCELED    BITMAP MOVEABLE PURE "canceled.bmp"
CANCELF     BITMAP MOVEABLE PURE "cancelf.bmp"
//Mẫu thực đơn
MAINMENU MENU DISCARDABLE
BEGIN
    POPUP "&Lựa chọn"
    BEGIN
        MENUITEM "&Tạo mới...",      IDM_NEW
        MENUITEM SEPARATOR
        MENUITEM "&Kết thúc",        IDM_EXIT
    END
END
// Mẫu hộp thoại
EDITENTRY DIALOG DISCARDABLE 0, 0, 184, 87
```

```

STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Thêm/Sửa bản ghi"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT      "&Họ và tên",-1,8,14,35,8
    EDITTEXT   IDC_NAME,49,12,118,12,ES_AUTOHSCROLL
    LTEXT      "Điện &thoại",-1,8,34,34,8
    EDITTEXT   IDC_PHONE,49,32,118,12,ES_AUTOHSCROLL
    CONTROL    "OK",IDOK,"Button",BS_OWNERDRAW | WS_GROUP | WS_TABSTOP, 50,54,40,24
    CONTROL    "Cancel",IDCANCEL,"Button",BS_OWNERDRAW | WS_GROUP |
        WS_TABSTOP,104,54,40,24
END
// Bảng xâu ký tự
STRINGTABLE DISCARDABLE
BEGIN
    IDM_NEW      "Tạo mới"
    IDM_EXIT     "Kết thúc chương trình"
END

```

```
//-----
```

Tập NAP.H

```

class CNumEdit : public CEdit
{
protected:
    afx_msg void OnChar (UINT, UINT, UINT);
    DECLARE_MESSAGE_MAP ()
};
//-----
class CNapListBox : public CListBox
{
private:
    CFont m_font;
public:
    virtual BOOL PreCreateWindow (CREATESTRUCT&);
    void PromptForNewEntry ();
protected:
    virtual BOOL OnChildNotify (UINT, WPARAM, LPARAM, LRESULT*);
    afx_msg int OnCreate (LPCREATESTRUCT);
    DECLARE_MESSAGE_MAP ()
}

```

```
};
//-----
class CMyApp : public CWinApp
{
public:
    BOOL InitInstance ();
};
//-----
class CMainWindow : public CFrameWnd
{
private:
    CListBox m_ctlListBox;
public:
    CMainWindow ();
protected:
    afx_msg int OnCreate (LPCREATESTRUCT);
    afx_msg void OnSize (UINT, int, int);
    afx_msg void OnNew ();
    afx_msg void OnExit ();
    DECLARE_MESSAGE_MAP ()
};
//-----
class CEntryDialog : public CDialog
{
public:
    CString m_strName;
    CString m_strPhone;
    CBitmapButton m_ctlOKButton;
    CBitmapButton m_ctlCancelButton;
    CNumEdit m_ctlPhone;
    CEntryDialog (CWnd* pParentWnd = NULL) :
        CDialog ("EditEntry", pParentWnd) {}
    virtual BOOL OnInitDialog ();
protected:
    virtual void DoDataExchange (CDataExchange*);
    afx_msg void OnUpdateOKButton ();
    DECLARE_MESSAGE_MAP ()
};
//-----
```


Tệp RESOURCE.H

```
#define IDM_NEW      100
#define IDM_EXIT     101
#define IDC_NAME     110
#define IDC_PHONE    111
#define IDC_LISTBOX  200
```

```
//-----
```

Tệp NAP.CPP

```
#include <afxwin.h>
#include <afxext.h>
#include "Resource.h"
#include "Nap.h"
CMyApp myApp;
// Hàm thành phần của lớp CMyApp
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}

// ánh xạ thông điệp và hàm thành phần của CMainWindow
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_CREATE ()
    ON_WM_SIZE ()
    ON_COMMAND (IDM_NEW, OnNew)
    ON_COMMAND (IDM_EXIT, OnExit)
END_MESSAGE_MAP ()

//-----
CMainWindow::CMainWindow ()
{
    Create (NULL, "Danh bạ điện thoại", WS_OVERLAPPEDWINDOW, rectDefault, NULL, "MainMenu");
}

//-----
int CMainWindow::OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate (lpCreateStruct) == -1)    return -1;
    m_ctlListBox.Create (WS_CHILD | WS_VISIBLE, CRect (0, 0, 0, 0), this, IDC_LISTBOX);
```

```

    return 0;
}
//-----
void CMainWindow::OnSize (UINT nType, int cx, int cy)
{
    m_ctlListBox.SetWindowPos (NULL, 0, 0, cx, cy, SWP_NOMOVE | SWP_NOZORDER);
}
//-----
void CMainWindow::OnNew ()
{
    m_ctlListBox.PromptForNewEntry ();
}
//-----
void CMainWindow::OnExit ()
{
    SendMessage (WM_CLOSE, 0, 0);
}
// ánh xạ thông điệp và hàm thành phần của
BEGIN_MESSAGE_MAP (CEntryDialog, CDialog)
    ON_EN_CHANGE (IDC_NAME, OnUpdateOKButton)
END_MESSAGE_MAP ()
//-----
BOOL CEntryDialog::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    m_ctlOKButton.AutoLoad (IDOK, this);
    m_ctlCancelButton.AutoLoad (IDCANCEL, this);
    m_ctlPhone.SubclassDlgItem (IDC_PHONE, this);
    OnUpdateOKButton ();
    CenterWindow ();
    return TRUE;
}
//-----
void CEntryDialog::OnUpdateOKButton ()
{
    CEdit* pCtlName = (CEdit*)GetDlgItem (IDC_NAME);
    m_ctlOKButton.EnableWindow (pCtlName->LineLength ());
}
//-----

```

```

void CEntryDialog::DoDataExchange (CDataExchange* pDX)
{
    CDialog::DoDataExchange (pDX);
    DDX_Text (pDX, IDC_NAME, m_strName);
    DDV_MaxChars (pDX, m_strName, 32);
    DDX_Text (pDX, IDC_PHONE, m_strPhone);
    DDV_MaxChars (pDX, m_strPhone, 32);
}

BEGIN_MESSAGE_MAP (CNapListBox, CListBox)
    ON_WM_CREATE ()
END_MESSAGE_MAP ()

//-----
BOOL CNapListBox::PreCreateWindow (CREATESTRUCT& cs)
{
    if (!CListBox::PreCreateWindow (cs))    return FALSE;
    cs.style |= LBS_SORT | LBS_USETABSTOPS | LBS_NOTIFY | LBS_NOINTEGRALHEIGHT;
    return TRUE;
}

//-----
int CNapListBox::OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    if (CListBox::OnCreate (lpCreateStruct) == -1)    return -1;
    CClientDC dc (this);
    int nHeight = ((dc.GetDeviceCaps (LOGPIXELSY) * 8) / 72);
    m_font.CreateFont (nHeight, 0, 0, 0, FW_NORMAL, 0, 0, 0, DEFAULT_CHARSET,
        OUT_CHARACTER_PRECIS, CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH | FF_DONTCARE, "MS Sans Serif");
    SetFont (&m_font, FALSE);
    SetTabStops (128);
    return 0;
}

//-----
BOOL CNapListBox::OnChildNotify (UINT message, WPARAM wParam, LPARAM lParam, LRESULT*
pLResult)
{
    if (message == WM_COMMAND) {
        UINT nCode = HIWORD (wParam);
        if (nCode == LBN_DBLCLK) {
            CEntryDialog dlg;

```

```

    CString strItem;
    int nIndex = GetCurSel ();
    GetText (nIndex, strItem);
    int nPos = strItem.Find ('\t');
    dlg.m_strName = strItem.Left (nPos);
    dlg.m_strPhone = strItem.Right (strItem.GetLength () - nPos - 1);
    if (dlg.DoModal () == IDOK) {
        strItem = dlg.m_strName + "\t" + dlg.m_strPhone;
        DeleteString (nIndex);
        AddString (strItem);
    }
    SetFocus ();
    return TRUE;
}

return FALSE;
}

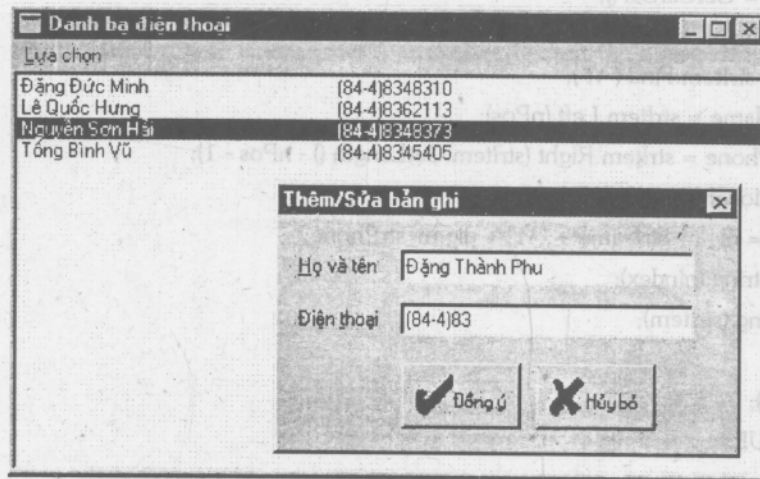
//-----
void CNapListBox::PromptForNewEntry ()
{
    CEntryDialog dlg;
    if (dlg.DoModal () == IDOK) {
        CString strItem = dlg.m_strName + "\t" + dlg.m_strPhone;
        AddString (strItem);
    }
    SetFocus ();
}

// ánh xạ thông điệp và hàm thành phần của CNumEdit
BEGIN_MESSAGE_MAP (CNumEdit, CEdit)
    ON_WM_CHAR ()
END_MESSAGE_MAP ()

//-----
void CNumEdit::OnChar (UINT nChar, UINT nRepCnt, UINT nFlags)
{
    if (((nChar >= '0') && (nChar <= '9')) || (nChar == VK_BACK) ||
        (nChar == '(') || (nChar == ')') || (nChar == '-') || (nChar == ' '))
        CEdit::OnChar (nChar, nRepCnt, nFlags);
}

//-----

```



Hình 6.2 Kết quả thực hiện chương trình NAP

Hộp thoại trong hình 6.2 được lập từ lớp phái sinh `CDialog` có tên `CEntryDialog`. Điều khiển soạn thảo chấp nhận số điện thoại là điều khiển `CNumEdit` tương tự như đã xem xét trên đây.

Ta có thể thấy: mảng *điện thoại* sẽ không chấp nhận bất kỳ ký tự nào ngoài chữ số, ngoặc và gạch ngang. Các phím `OK` và `Cancel` là điều khiển `CBitmapButton` được khởi động trong `OnInitDialog`. Ở đây có thêm thông báo lại `ON_EN_CHANGE` của ánh xạ hộp thoại cho hàm thành phần hàm `OnUpdateOKButton` biết rằng xâu ký tự trong trường *Họ và tên* đã thay đổi. `OnUpdateOKButton` làm hiệu lực hay mất hiệu lực phím `OK` phụ thuộc vào trường *Họ và tên* có chứa ký tự nào hay không. Phím `OK` bị làm mất hiệu lực khi điều khiển soạn thảo là rỗng và làm hiệu lực khi nó chứa ít nhất một ký tự, tránh cho người sử dụng nhập đầu vào rỗng cho hộp danh sách. Cũng chú ý đến việc gọi đến hàm `CenterWindow` trong `OnInitDialog`; chúng chỉnh giữa hộp thoại so với cửa sổ cha nó.

6.7 HỘP THOẠI MODELESS

Một khi đã làm chủ hộp thoại *modal*, ta sẽ thấy hộp thoại *modeless* chỉ là một dạng khác của cái mà ta đã khảo sát. Như hộp thoại *modal*, hộp thoại *modeless* được tạo lập từ mẫu thoại gắn với tệp `EXE` của ta. Các khác nhau cơ bản là:

- Hộp thoại *modal* được hiển thị bằng hàm `CDialog::DoModal`, còn hộp thoại *modeless* hiển thị nhờ `CDialog::Create`. Khác với `DoModal`, hàm `DoModal` không trở lại cho đến khi hộp thoại bị phá hủy còn hàm `Create` trở lại ngay sau khi hộp thoại được lập. Kết quả là hộp thoại vẫn được hiển thị khi đã kết thúc thực hiện hàm `Create`. Nếu muốn,

Create có thể được gọi từ cấu tử của hộp thoại cũng như CFrameWnd::Create thường được gọi từ cấu tử của cửa sổ khung.

- Hộp thoại *modeless* bị mất đi khi gọi hàm DestroyWindow, chứ không phải EndDialog. Ta không được thực hiện hàm điều khiển OnOK hay OnCancel vì cả hai đều gọi hàm EndDialog.
- Vì OnCancel (và OnOK) phải được nạp chồng để tránh việc gọi EndDialog, ta sẽ không bao giờ khai báo đối tượng hộp thoại *modeless* trực tiếp từ lớp CDialog mà phải xây dựng lớp phái sinh riêng. Luôn phải nạp chồng hàm OnCancel trong lớp phái sinh. Cũng phải nạp chồng hàm OnOK nếu hộp thoại có phím OK hay bất kỳ phím nào mà chỉ danh ID của chúng được gán giá trị IDOK.
- Các đối tượng của hộp thoại *modal* thông thường được tạo trong ngăn xếp vậy nó tự động bị phá hủy. Ngược lại, hộp thoại *modeless* thông thường được tạo bằng toán tử *new*, vậy đối tượng của lớp hộp thoại này không tự phá hủy. Để đảm bảo rằng đối tượng hộp thoại được phá hủy khi kết thúc hộp thoại, ta phải nạp chồng hàm ảo hộp thoại PostNcDestroy và thực hiện hàm delete this.

Còn một vài khác nhau giữa hộp thoại *modal* và *modeless* sẽ do MFC quản lý. Thí dụ, các thông điệp đi đến hộp thoại *modeless* được lấy ra và chuyển đến vòng thông điệp chính của ứng dụng, trong khi hộp thoại *modal* sử dụng vòng lặp thông điệp riêng. Ứng dụng viết trong C có sử dụng hộp thoại *modeless* phải lưu chỉ danh cửa sổ hộp thoại và xen lời gọi hàm ::IsDialogMessage. Trong MFC khung ứng dụng làm việc này cho ta.

Một trong các vấn đề sẽ gặp đầu tiên khi ta viết mã trình cho hộp thoại *modeless* là cách thức để hộp thoại liên lạc với cha của nó. Thí dụ hộp thoại *Find* trong các trình soạn thảo văn bản. Một khi người sử dụng nhập xâu ký tự và nhấn phím *Find*, hộp thoại ra lệnh cha nó thực hiện tìm kiếm như thế nào? Hoặc hộp thoại phải gửi cho cha nó thông điệp, hoặc có thể sử dụng hàm AfxGetMainWnd để nhận con trỏ đến cửa sổ chính của ứng dụng và gọi một trong các hàm thành phần của nó. Cách khác là để cho cửa sổ chính xử lý nhấn phím thực hiện bên trong hộp thoại nhờ đặc trưng không công bố của cơ chế dẫn đường-lệnh của MFC. Nếu ta gán chỉ danh ID điều khiển của phím thoại bằng 0x8000 hay lớn hơn, những thông báo lại của phím không được hộp thoại quản lý sẽ được truyền lên cha của chúng. Do đó, cửa sổ chính của ứng dụng có thể cài đặt trong hàm thành phần OnFind và phản ứng trực tiếp khi nhấn phím *Find*, ngay cả khi phím là con của cửa sổ hộp thoại.

6.8 TẦM ĐẶC TÍNH

Một trong các đặc tính hấp dẫn mới của Windows 95/98 là trợ giúp tầm đặc tính (*property sheet*), đó là hộp thoại kiểu bảng chứa các trang điều khiển mà người sử dụng có thể bật qua lại các trang nhờ nhấn chuột. Tầm đặc tính được gộp trong thư viện điều khiển chung của hệ điều hành

Windows 95/98.

Việc xây dựng tấm đặc tính khá khó khăn trong C khi sử dụng SDK và API, nhưng tương đối dễ dàng trong C++ và MFC nhờ khung ứng dụng. Sự thật, việc thêm tấm đặc tính vào ứng dụng MFC không hoàn toàn khác việc thêm hộp thoại. Ứng dụng MFC 6.0 có sử dụng tấm đặc tính và chạy trong Windows 95/98 hay Windows NT 3.51/4.0 sẽ tự động sử dụng chính mã lệnh cài đặt tấm đặc tính của hệ điều hành. Trong các hệ điều hành khác, MFC có cài đặt riêng tấm đặc tính để thay thế.

Các chức năng của tấm đặc tính hầu như được gói trong hai lớp của MFC với tên **CPropertySheet** và **CPropertyPage**, cả hai được định nghĩa trong tệp **AFXDLGS.H**. Lớp **CPropertySheet** suy diễn từ **CWnd** biểu diễn tấm đặc tính. **CPropertyPage** suy diễn từ **CDialog** biểu diễn các trang của tấm đặc tính. Tương tự hộp thoại, tấm đặc tính có thể là *modal* hay *modeless*.

Để tạo tấm đặc tính *modal* ta sử dụng các lớp trên cơ sở **CPropertySheet** và **CPropertyPage**, trước hết thêm mẫu thoại vào tệp RC để định nghĩa các điều khiển và tính chất của từng trang. Trong lệnh **STYLE** của mẫu, kiểu tấm đặc tính **WS_CHILD**, **WS_BORDER**, **WS_CAPTION**, và **WS_DISABLED** được xác định. Sử dụng lệnh **CAPTION** để chỉ ra tiêu đề sẽ xuất hiện trên đỉnh của mỗi trang. Cũng có thể bỏ qua lệnh **CAPTION** và chỉ ra tiêu đề khi xây dựng trang.

Tiếp theo, với mỗi trang trong tấm đặc tính hãy xây dựng lớp phái sinh kiểu hộp thoại từ **CPropertyPage** với thành phần dữ liệu cho các điều khiển (*control*) của trang. Cũng như thực hiện suy diễn lớp hộp thoại thông thường, hãy để dữ liệu thành phần là chung để chúng có thể được tham nhập từ ngoài. Viết ánh xạ dữ liệu (hàm **DoDataExchange**) để truyền dữ liệu giữa các biến thành phần và điều khiển, và kiểm tra dữ liệu nếu muốn.

Sau đó, xây dựng đối tượng tấm đặc tính từ **CPropertySheet** hay lớp phái sinh từ **CPropertySheet** và các đối tượng trang của tấm đặc tính từ lớp **CPropertyPage** mà ta đã suy diễn trước đó. Thêm các trang vào tấm đặc tính nhờ hàm **CPropertySheet::AddPage**. Nếu các trang có kích thước khác nhau (kích thước được chỉ ra trong lệnh **DIALOG** trong mẫu thoại), hãy làm trang đầu là lớn nhất sao cho tấm đặc tính có thể chứa vật lý toàn bộ các trang.

Cuối cùng, khởi động các thành phần dữ liệu trang và gọi hàm **DoModal** của tấm đặc tính để hiển thị trên màn hình. Nếu tấm đặc tính bị phá hủy bằng phím OK, hàm **DoModal** sẽ cho lại **IDOK**. Nếu không thì cho lại **IDCANCEL** - đúng như hộp thoại.

Để đơn giản hoá việc gọi tấm đặc tính, phần lớn lập trình viên MFC xây dựng các lớp tấm đặc tính phái sinh riêng của mình từ **CPropertySheet**, bao gồm các thành phần dữ liệu cho các trang đặc tính sao cho các trang được tự động tạo lập. Tương ứng mỗi lớp trang được để gọn trong cấu tử mặc định gọi cấu tử lớp cơ sở với tên của mẫu hộp thoại liên quan. Cấu tử tấm đặc tính sau đó sẽ gọi **AddPage** để thêm vào từng trang một.

Đoạn mã trình 6.6 chỉ ra việc khai báo lớp tương ứng như thế nào. CFirstPage biểu diễn trang thứ nhất trong tấm đặc tính và CSecondPage biểu diễn trang thứ hai. Tài nguyên hộp thoại liên quan là "FirstPage" và "SecondPage". Với cơ sở hạ tầng này tấm đặc tính *modal* có thể được xây dựng và hiển thị bằng hai lệnh:

```
CMyPropertySheet propsheet (this);
propsheet.DoModal ();
```

Đoạn mã trình 6.6 Tấm đặc tính mẫu và khai báo lớp trang của tấm đặc tính

```
class CFirstPage : public CPropertyPage
{
public:
    // Khai báo các thành phần dữ liệu của CFirstPage ở đây
    CFirstPage () : CPropertyPage ("FirstPage") {}
protected:
    virtual void DoDataExchange (CDataExchange*);
};
//-----
class CSecondPage : public CPropertyPage
{
public:
    // Khai báo các thành phần dữ liệu của CSecondPage ở đây
    CSecondPage () : CPropertyPage ("SecondPage") {}
protected:
    virtual void DoDataExchange (CDataExchange*);
};
//-----
class CMyPropertySheet : public CPropertySheet
{
public:
    CFirstPage m_firstPage;    // Trang thứ nhất
    CSecondPage m_secondPage;  // Trang thứ hai
    // Cấu tử tự động thêm trang
    CMyPropertySheet (CWnd* pParentWnd = NULL) : CPropertySheet ("Properties", pParentWnd)
    {
        AddPage (&m_firstPage);
        AddPage (&m_secondPage);
    }
};
//-----
```

Nếu CFirstPage chứa thành phần dữ liệu CString với tên m_name và CSecondPage chứa biến

UINT với tên `m_age` thì giá trị của thành phần dữ liệu được khởi động và lấy ra trước và sau khi gọi `DoModal` như sau:

```

CMyPropertySheet propsheet (this);
propsheet.m_firstPage.m_name = m_name;
propsheet.m_secondPage.m_age = m_age;
if (propsheet.DoModal () == IDOK) {
    m_name = propsheet.m_firstPage.m_name;
    m_age = propsheet.m_secondPage.m_age;
}

```

Tương tự `CDialog::DoModal`, `CPropertySheet::DoModal` cho lại `IDOK` nếu tấm đặc tính bị phá hủy do nhấn phím *OK* hay *IDCANCEL* trong trường hợp khác.

Mẫu của hộp thoại cho các trang tấm đặc tính không cần phím *OK* và *Cancel* như mẫu hộp thoại qui định bởi vì tấm đặc tính đã cung cấp các phím này rồi. Tấm đặc tính còn gộp phím *Apply* và *Help* mà ta có thể kích hoạt nếu muốn. Phím *Apply* bị vô hiệu khi tấm đặc tính xuất hiện lần đầu và nó có hiệu lực khi trang của tấm đặc tính gọi hàm `SetModified` thừa hưởng từ `CPropertyPage`. Thông thường, các trang của tấm đặc tính đi theo tấm đặc tính có phím *Apply* sẽ sử dụng ánh xạ thông điệp để đón *thông báo* cho biết rằng điều khiển đã bị thay đổi và gọi hàm `SetModified` để trả lời. Hàm điều khiển nhấn phím `ON_BN_CLICKED` trong lớp tấm đặc tính đón nhấn phím *Apply* có chỉ danh ID là `IS_APPLY_NOW`. Hàm *điều khiển nhấn phím* sẽ gọi `UpdateData` với tham số `TRUE` (giả sử sử dụng `DDX/DDV`) để cập nhật các biến thành phần của trang hiện hành, và thông báo lại sự kiện cho cha của tấm đặc tính bằng cách gửi cho chúng thông điệp hay gọi hàm thành phần chung của cha nó. Sau đó nó vô hiệu phím *Apply* bằng cách gọi `SetModified` với tham số `FALSE` cho mỗi trang của tấm đặc tính.

Hàm điều khiển `ON_BN_CLICKED` chỉ gọi `UpdateData` để kích hoạt trang của tấm đặc tính. Điều này là quan trọng bởi vì các trang của tấm đặc tính không được tạo vật lý cho đến khi chúng được kích hoạt bởi người dùng tấm đặc tính. Việc gọi `UpdateData` cho trang của tấm đặc tính mà *tab* của chúng vẫn chưa được nhấn sẽ phát sinh lỗi từ MFC. Khung ứng dụng gọi `UpdateData` cho trang tích cực hiện hành khi *tab* được nhấn để bật sang trang khác, vậy khi phím *Apply* được nhấn, chỉ trang mà các thành phần dữ liệu của nó cần cập nhật là trang tích cực. Ta có thể lấy con trỏ đến trang tích cực hiện hành nhờ hàm `GetActivePage` của lớp `CPropertySheet`. Trình điều khiển `ON_BN_CLICKED` cho phím *Apply* trong tấm đặc tính của đoạn trước có thể viết như sau:

```

// Trong ánh xạ thông điệp
ON_BN_CLICKED (ID_APPLY_NOW, OnApplyNow)
// Trong cài đặt lớp
void CMyPropertySheet::OnApplyNow ()
{
    GetActivePage ()->UpdateData (TRUE);
}

```

```

AfxGetMainWnd()->SendMessage(WM_USER_APPLYNOW, 0, 0);
m_firstPage.SetModified(FALSE);
m_secondPage.SetModified(FALSE);
}

```

Giả sử cửa sổ chính biết cách quản lý thông điệp do người sử dụng định nghĩa WM_USER_APPLYNOW do tấm đặc tính gửi đến, trình điều khiển này sẽ áp dụng toàn bộ các thay đổi vào các trang của tấm đặc tính và vô hiệu phím *Apply* cho đến khi thay đổi khác xảy ra.

Sử dụng DDX và DDV để truyền dữ liệu giữa điều khiển và hàm thành phần trong trang của tấm đặc tính và kiểm tra dữ liệu lấy từ điều khiển là rất thuận tiện; nó cho phép khung ứng dụng xử lý các thông báo lại được truyền đi khi người sử dụng chuyển dịch giữa các trang của tấm đặc tính hay nhấn phím của tấm đặc tính. Thí dụ lần đầu trang của tấm đặc tính được hiển thị, hàm OnInitDialog liên quan được gọi để khởi động trang. Cài đặt mặc định của OnInitDialog gọi UpdateData(FALSE) để khởi động điều khiển các trang. Nếu người sử dụng nhấn *tab* khác để nhảy đến trang khác, hàm ảo OnKillActive của trang hiện hành được gọi và khung ứng dụng gọi UpdateData(TRUE) để lấy ra và kiểm tra dữ liệu của điều khiển. Trong khi đó, trang mới nhận thông báo lại OnSetActive và có thể cả thông báo lại OnInitDialog, nhưng chỉ khi không có hàm nào của DDV của trang trước bị lỗi. Nếu người sử dụng tiếp tục nhấn phím OK của tấm đặc tính, hàm điều khiển OnOK của trang hiện hành được gọi và khung ứng dụng gọi UpdateData(TRUE) để lấy ra và kiểm tra dữ liệu của trang. Để cho DDX/DDV làm việc này, ta không phải nạp chồng bất kỳ hàm nào của chúng. Dĩ nhiên, ta có thể cung cấp cài đặt riêng các hàm OnInitDialog, OnSetActive, OnKillActive, OnOK, và OnCancel của CPropertyPage để thực hiện các xử lý riêng.

Việc xem xét cuối cùng khi lập trình tấm đặc tính là nếu người sử dụng yêu cầu thay đổi mà ta không thể trở lại trang trước được thì gọi CPropertyPage::CancelToClose để thay đổi phím *Cancel* của tấm đặc tính thành phím *Close*. Đây là một phần của nghi thức tấm đặc tính, tránh cho người sử dụng suy nghĩ sai lầm rằng sự thay đổi có thể được bãi bỏ bằng nhấn phím.

6.9 ỨNG DỤNG PSD

Ứng dụng PSD (*Property Sheet Demo*) trong chương trình 6.7 mô tả tấm đặc tính 2-tab. Hình 6.3 chỉ ra cửa sổ PSD và tấm đặc tính của nó. Trang "Nền" điều khiển màu nền và kiểu tô của cửa sổ chính. Trang "Văn bản" điều khiển đặc tính của ký tự sẽ hiển thị. Ta có thể có đầu ra chương trình khác nhau bằng cách thay đổi các đặc tính của tấm và nhấn phím *OK* hay *Apply*. Phím *Apply* để lại tấm đặc tính đang hiển thị, trong khi OK phá hủy tấm đặc tính. Tấm đặc tính là *modal*, vậy cửa sổ chính không thể kích hoạt lại cho đến khi tấm đặc tính vẫn đang hiển thị.

Chương trình 6.7 PSD

Tập PSD.RC

```

#include "afxres.h"
#include "resource.h"
// Mẫu thực đơn
MAINMENU MENU DISCARDABLE
BEGIN
    POPUP "&Lựa chọn"
    BEGIN
        MENUITEM "&Thuộc tính...",          IDM_PROPERTIES
        MENUITEM SEPARATOR
        MENUITEM "Kết thúc",                IDM_EXIT
    END
END

// Mẫu hộp thoại
BACKGROUND DIALOG DISCARDABLE 0, 0, 132, 144
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Nền"
FONT 8, "MS Sans Serif"
BEGIN
    GROUPBOX    "Kiểu tô", -1, 12, 8, 101, 40
    CONTROL     "&Tô kín", IDC_SOLID, "Button", BS_AUTORADIOBUTTON |
        WS_GROUP | WS_TABSTOP, 20, 20, 64, 10
    CONTROL     "Tô theo hàm &Gradient", IDC_GRADIENT, "Button",
        BS_AUTORADIOBUTTON | WS_TABSTOP, 20, 32, 92, 10
    GROUPBOX    "Tô màu", -1, 12, 60, 101, 76, WS_GROUP
    CONTROL     "Đỏ", IDC_RED, "Button", BS_AUTORADIOBUTTON | WS_GROUP |
        WS_TABSTOP, 20, 72, 64, 10
    CONTROL     "Xanh &lá cây", IDC_GREEN, "Button", BS_AUTORADIOBUTTON |
        WS_TABSTOP, 20, 84, 64, 10
    CONTROL     "&Xanh lam", IDC_BLUE, "Button", BS_AUTORADIOBUTTON |
        WS_TABSTOP, 20, 96, 64, 10
    CONTROL     "Cánh &sen", IDC_MAGENTA, "Button", BS_AUTORADIOBUTTON |
        WS_TABSTOP, 20, 108, 64, 10
    CONTROL     "&Cổ vịt", IDC_CYAN, "Button", BS_AUTORADIOBUTTON |
        WS_TABSTOP, 20, 120, 64, 10
END

```

```

TEXT DIALOG DISCARDABLE 0, 0, 131, 143
STYLE WS_CHILD | WS_DISABLED | WS_CAPTION
CAPTION "Văn bản"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT        "&Nội dung",-1,12,16,32,8
    EDITTEXT     IDC_TEXT,47,12,73,12,ES_AUTOHSCROLL | WS_GROUP
    LTEXT        "Độ &cao (điểm)",-1,12,36,76,8
    EDITTEXT     IDC_HEIGHT,92,32,28,12,ES_AUTOHSCROLL | WS_GROUP
    GROUPBOX     "Tác động",-1,12,60,108,52
    CONTROL      "Đạ&m",IDC_BOLD,"Button",BS_AUTOCHECKBOX | WS_GROUP
                | WS_TABSTOP,20,72,80,10
    CONTROL      "N&ghiêng",IDC_ITALIC,"Button",BS_AUTOCHECKBOX |
                WS_GROUP | WS_TABSTOP,20,84,80,10
    CONTROL      "&Bóng",IDC_SHADOW,"Button",BS_AUTOCHECKBOX |
                WS_GROUP | WS_TABSTOP,20,96,80,10

```

```
END
```

```
//-----
```

Tệp PSD.H

```

class CMyApp : public CWinApp
{
public:
    BOOL InitInstance ();
};
//-----
class CBkgndPage : public CPropertyPage
{
public:
    int m_nFillType;
    int m_nFillColor;
    CBkgndPage () : CPropertyPage ("Background") {}
protected:
    virtual void DoDataExchange (CDataExchange*);
    afx_msg void OnChange ();
    DECLARE_MESSAGE_MAP ()
};
//-----

```

```

class CTextPage : public CPropertyPage
{
public:
    CString m_text;
    int m_nHeight;
    BOOL m_bBold;
    BOOL m_bItalic;
    BOOL m_bShadow;
    CTextPage () : CPropertyPage ("Text") {}
protected:
    virtual void DoDataExchange (CDataExchange*);
    afx_msg void OnChange ();
    DECLARE_MESSAGE_MAP ()
};
//-----
class CMyPropertySheet : public CPropertySheet
{
public:
    CBkgndPage m_bkgndPage;
    CTextPage m_textPage;
    CMyPropertySheet (CWnd* = NULL);
protected:
    afx_msg void OnApply ();
    DECLARE_MESSAGE_MAP ()
};
//-----
class CMainWindow : public CFrameWnd
{
private:
    int m_nFillType;
    int m_nFillColor;
    CString m_text;
    int m_nHeight;
    BOOL m_bBold;
    BOOL m_bItalic;
    BOOL m_bShadow;
    void DoSolidFill (CDC*, CRect*);
    void DoGradientFill (CDC*, CRect*);
    void DoDrawText (CDC*, CRect*);

```

```
public:
    CMainWindow ();
    void Apply (CMyPropertySheet*);
protected:
    afx_msg void OnPaint ();
    afx_msg void OnProperties ();
    afx_msg void OnExit ();
    afx_msg void OnAbout ();
    DECLARE_MESSAGE_MAP ()
```

```
};
```

```
//-----
```

Tệp RESOURCE.H

```
#define IDM_PROPERTIES 100
#define IDM_EXIT      101
#define IDC_SOLID     110
#define IDC_GRADIENT  111
#define IDC_RED       112
#define IDC_GREEN     113
#define IDC_BLUE      114
#define IDC_MAGENTA   115
#define IDC_CYAN      116
#define IDC_TEXT      117
#define IDC_HEIGHT    118
#define IDC_BOLD      119
#define IDC_ITALIC    120
#define IDC_SHADOW    121
```

```
//-----
```

Tệp PSD.CPP

```
#include <afxwin.h>
#include <afxdlgs.h>
#include "Resource.h"
#include "Psd.h"
CMyApp myApp;
// Các hàm thành phần CMyApp
BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow ();
    m_pMainWnd->ShowWindow (m_nCmdShow);
```

```

    m_pMainWnd->UpdateWindow ();
    return TRUE;
}

// ánh xạ thông điệp và các hàm thành phần của CMainWindow
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_PAINT ()
    ON_COMMAND (IDM_PROPERTIES, OnProperties)
    ON_COMMAND (IDM_EXIT, OnExit)
END_MESSAGE_MAP ()

//-----
CMainWindow::CMainWindow ()
{
    m_nFillType = 1;
    m_nFillColor = 2;
    m_text = "PopMap";
    m_nHeight = 100;
    m_bBold = TRUE;
    m_bItalic = TRUE;
    m_bShadow = FALSE;
    Create (NULL, "Thí dụ về tầm thuộc tính", WS_OVERLAPPEDWINDOW,
        rectDefault, NULL, "MainMenu");
}

//-----
void CMainWindow::OnPaint ()
{
    CRect rect;
    GetClientRect (&rect);
    CPaintDC dc (this);
    m_nFillType == 1 ? DoGradientFill (&dc, &rect) : DoSolidFill (&dc, &rect);
    DoDrawText (&dc, &rect);
}

void CMainWindow::OnProperties ()
{
    CMyPropertySheet propsheet (this);
    propsheet.m_bkgndPage.m_nFillType = m_nFillType;
    propsheet.m_bkgndPage.m_nFillColor = m_nFillColor;
    propsheet.m_textPage.m_text = m_text;
    propsheet.m_textPage.m_nHeight = m_nHeight;
    propsheet.m_textPage.m_bBold = m_bBold;
}

```

```

propsheet.m_textPage.m_bItalic = m_bItalic;
propsheet.m_textPage.m_bShadow = m_bShadow;
if (propsheet.DoModal() == IDOK) {
    m_nFillType = propsheet.m_bkgndPage.m_nFillType;
    m_nFillColor = propsheet.m_bkgndPage.m_nFillColor;
    m_text = propsheet.m_textPage.m_text;
    m_nHeight = propsheet.m_textPage.m_nHeight;
    m_bBold = propsheet.m_textPage.m_bBold;
    m_bItalic = propsheet.m_textPage.m_bItalic;
    m_bShadow = propsheet.m_textPage.m_bShadow;
    Invalidate (FALSE);
}
}
//-----
void CMainWindow::OnExit ()
{
    SendMessage (WM_CLOSE, 0, 0);
}
//-----
void CMainWindow::Apply (CMyPropertySheet* pMyProp)
{
    m_nFillType = pMyProp->m_bkgndPage.m_nFillType;
    m_nFillColor = pMyProp->m_bkgndPage.m_nFillColor;
    m_text = pMyProp->m_textPage.m_text;
    m_nHeight = pMyProp->m_textPage.m_nHeight;
    m_bBold = pMyProp->m_textPage.m_bBold;
    m_bItalic = pMyProp->m_textPage.m_bItalic;
    m_bShadow = pMyProp->m_textPage.m_bShadow;
    Invalidate (FALSE);
}
//-----
void CMainWindow::DoSolidFill (CDC* pDC, CRect* pRect)
{
    static COLORREF crColor[5] = {
        RGB (255, 0, 0),
        RGB (0, 255, 0),
        RGB (0, 0, 255),
        RGB (255, 0, 255),
        RGB (0, 255, 255) };

```



```

CBrush* pBrush = new CBrush (crColor[m_nFillColor]);
pDC->FillRect (pRect, pBrush);
delete pBrush;
}
//-----
void CMainWindow::DoGradientFill (CDC* pDC, CRect* pRect)
{
    CBrush* pBrush[256];
    for (int i=0; i<256; i++) {
        switch (m_nFillColor) {
            case 0:
                pBrush[i] = new CBrush (RGB (255 - i, 0, 0));
                break;
            case 1:
                pBrush[i] = new CBrush (RGB (0, 255 - i, 0));
                break;
            case 2:
                pBrush[i] = new CBrush (RGB (0, 0, 255 - i));
                break;
            case 3:
                pBrush[i] = new CBrush (RGB (255 - i, 0, 255 - i));
                break;
            case 4:
                pBrush[i] = new CBrush (RGB (0, 255 - i, 255 - i));
                break;
        }
    }
    int nWidth = pRect->Width ();
    int nHeight = pRect->Height ();
    CRect rect;
    for (i=0; i<nHeight; i++) {
        rect.SetRect (0, i, nWidth, i + 1);
        pDC->FillRect (&rect, pBrush[(i * 255) / nHeight]);
    }
    for (i=0; i<256; i++)
        delete pBrush[i];
}
//-----
void CMainWindow::DoDrawText (CDC* pDC, CRect* pRect)

```

```

{
    CFont* pFont = new CFont ();
    int nHeight = ((pDC->GetDeviceCaps (LOGPIXELSY) * m_nHeight) / 72);
    pFont->CreateFont (nHeight, 0, 0, 0, m_bBold ? FW_BOLD : FW_NORMAL, m_bItalic, 0, 0,
        DEFAULT_CHARSET, OUT_CHARACTER_PRECIS, CLIP_CHARACTER_PRECIS,
        DEFAULT_QUALITY, DEFAULT_PITCH | FF_DONTCARE, "Times New Roman");
    pDC->SetBkMode (TRANSPARENT);
    CFont* pOldFont = pDC->SelectObject (pFont);
    if (m_bShadow) {
        CRect rectShadow (pRect);
        rectShadow.OffsetRect (8, 8);
        pDC->SetTextColor (RGB (0, 0, 0));
        pDC->DrawText (m_text, -1, &rectShadow, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
    }
    pDC->SetTextColor (RGB (255, 255, 255));
    pDC->DrawText (m_text, -1, pRect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
    pDC->SelectObject (pOldFont);
    delete pFont;
}
//-----
BEGIN_MESSAGE_MAP (CMyPropertySheet, CPropertySheet)
    ON_BN_CLICKED (ID_APPLY_NOW, OnApply)
END_MESSAGE_MAP ()
//-----
CMyPropertySheet::CMyPropertySheet (CWnd* pParentWnd) : CPropertySheet ("Thuộc tính", pParentWnd)
{
    AddPage (&m_bkgndPage);
    AddPage (&m_textPage);
}
//-----
void CMyPropertySheet::OnApply ()
{
    GetActivePage ()->UpdateData (TRUE);
    ((CMainWindow*) AfxGetMainWnd ())->Apply (this);
    m_bkgndPage.SetModified (FALSE);
    m_textPage.SetModified (FALSE);
}
BEGIN_MESSAGE_MAP (CBkgndPage, CPropertyPage)
    ON_BN_CLICKED (IDC_SOLID, OnChange)

```

```

ON_BN_CLICKED(IDC_GRADIENT, OnChange)
ON_BN_CLICKED(IDC_RED, OnChange)
ON_BN_CLICKED(IDC_GREEN, OnChange)
ON_BN_CLICKED(IDC_BLUE, OnChange)
ON_BN_CLICKED(IDC_MAGENTA, OnChange)
ON_BN_CLICKED(IDC_CYAN, OnChange)
END_MESSAGE_MAP ()

//-----
void CBkgndPage::OnChange ()
{
    SetModified (TRUE);
}

//-----
void CBkgndPage::DoDataExchange (CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange (pDX);
    DDX_Radio (pDX, IDC_SOLID, m_nFillType);
    DDX_Radio (pDX, IDC_RED, m_nFillColor);
}

//-----
BEGIN_MESSAGE_MAP (CTextPage, CPropertyPage)
    ON_EN_CHANGE (IDC_TEXT, OnChange)
    ON_EN_CHANGE (IDC_HEIGHT, OnChange)
    ON_BN_CLICKED (IDC_BOLD, OnChange)
    ON_BN_CLICKED (IDC_ITALIC, OnChange)
    ON_BN_CLICKED (IDC_SHADOW, OnChange)
END_MESSAGE_MAP ()

//-----
void CTextPage::OnChange ()
{
    SetModified (TRUE);
}

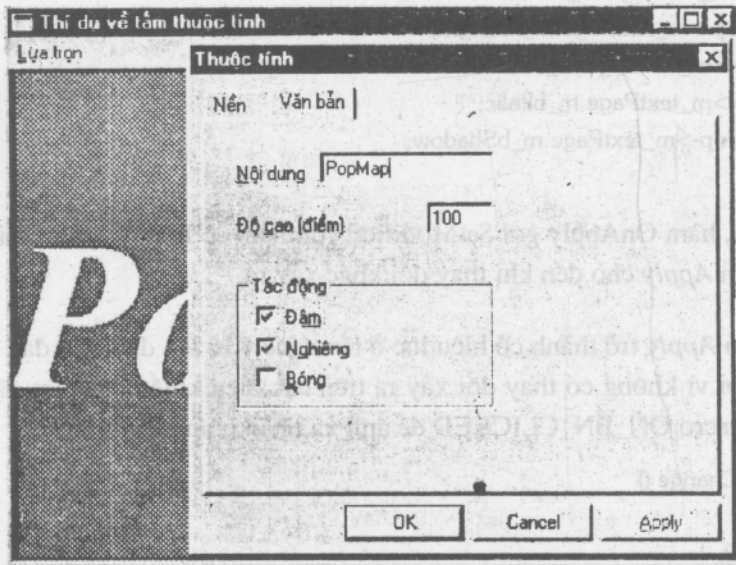
//-----
void CTextPage::DoDataExchange (CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange (pDX);
    DDX_Text (pDX, IDC_TEXT, m_text);
    DDX_Text (pDX, IDC_HEIGHT, m_nHeight);
    DDV_MinMaxInt (pDX, m_nHeight, 8, 192);
}

```

```
DDX_Check(pDX, IDC_BOLD, m_bBold);
DDX_Check(pDX, IDC_ITALIC, m_bItalic);
DDX_Check(pDX, IDC_SHADOW, m_bShadow);
```

```
}
```

```
//-----
```



Hình 6.3 Thí dụ về tấm thuộc tính

Một trong các chi tiết cài đặt mà ta phải quan tâm là các phím *OK* và *Apply* sẽ áp dụng việc thay đổi giá trị của các điều khiển trong tấm đặc tính như thế nào. Phím *OK* được quản lý như phím *OK* trong hộp thoại. Khi *OK* được nhấn, hàm *DoModal* cho lại *IDOK* và nơi gọi (*CMainWindow*) sao chép dữ liệu từ các biến thành phần của các trang của tấm đặc tính vào các biến thành phần riêng của nó. Sau đó chúng gọi *Invalidat*e để cập nhật màn hình. Phím *Apply* được quản lý khác bởi vì nó không làm cho *DoModal* trở lại. Việc nhấn phím *Apply* sẽ được ánh xạ vào *CMyPropertySheet::OnApply*:

```
ON_BN_CLICKED(ID_APPLY_NOW, OnApply)
```

Đến lượt *OnApply* gọi *UpdateData* để “đọc” dữ liệu từ các điều khiển của trang tích cực vào các thành phần dữ liệu liên quan, sử dụng *AfxGetMainWnd* để lấy con trỏ đến cửa sổ chính của ứng dụng, ép (*cast*) con trỏ thành con trỏ *CMainWindow* và gọi hàm thành phần chung *CMainWindow::Apply*:

```
GetActivePage()->UpdateData(TRUE);
((CMainWindow*) AfxGetMainWnd())->Apply(this);
```

CMainWindow::Apply sử dụng con trỏ pMyProp của CMyPropertySheet được truyền vào danh sách tham số để thăm nhập trang của tấm thuộc tính và các thành phần dữ liệu của chúng. Sau đó chúng yêu cầu vẽ lại cửa sổ:

```
m_nFillType = pMyProp->m_bkgndPage.m_nFillType;
m_nFillColor = pMyProp->m_bkgndPage.m_nFillColor;
m_text = pMyProp->m_textPage.m_text;
m_nHeight = pMyProp->m_textPage.m_nHeight;
m_bBold = pMyProp->m_textPage.m_bBold;
m_bItalic = pMyProp->m_textPage.m_bItalic;
m_bShadow = pMyProp->m_textPage.m_bShadow;
Invalidate (FALSE);
```

Sau khi *Apply* trở lại, hàm OnApply gọi SetModified với tham số FALSE cho mỗi trang của tấm đặc tính để vô hiệu phím *Apply* cho đến khi thay đổi khác xảy ra.

Làm thế nào để phím *Apply* trở thành có hiệu lực ở lần đầu? Khi lần đầu tấm đặc tính xuất hiện, phím bị mất hiệu lực bởi vì không có thay đổi xảy ra trên các điều khiển của tấm đặc tính. Nhưng CBkkgndPage sử dụng macro ON_BN_CLICKED để ánh xạ nhấn phím đến hàm sau:

```
void CBkkgndPage::OnChange ()
{
    SetModified (TRUE);
}
```

Tương tự, CTextPage ánh xạ nhấn phím và sự kiện thay đổi ký tự trong điều khiển soạn thảo đến hàm sau:

```
void CTextPage::OnChange ()
{
    SetModified (TRUE);
}
```

Kết quả là phím *Apply* được tự động có hiệu lực khi thay đổi các điều khiển.

6.10 CÁC VẤN ĐỀ KHÁC SẼ XEM XÉT

Chương sau ta sẽ xem xét một trong các khía cạnh quan trọng nhất của MFC: kiến trúc tài liệu/quan sát (*document/view*). Trong ứng dụng tài liệu/quan sát, dữ liệu được lưu trong đối tượng tài liệu phái sinh từ lớp CDocument của MFC, và nó biểu diễn “quan sát” của dữ liệu trong các cửa sổ được tạo bởi lớp CView của MFC hay lớp phái sinh của chúng. Bạn đã bao giờ quan sát AppWizard tạo ứng dụng SDI hay MDI và thấy tuyệt diệu là chỉ với một ít mã trình có thể tạo được chương trình chức năng? Chương tới ta sẽ tìm ra cái kỳ diệu đó là ở đâu và cho thấy kiến trúc tài liệu/quan sát sẽ cho phép ta bàn đến sức mạnh đầy đủ của MFC và khung ứng dụng của chúng.

CHƯƠNG 7

KIẾN TRÚC TÀI LIỆU/QUAN SÁT

Trong các phiên bản đầu tiên của MFC các ứng dụng được xây dựng như các trình mẫu trong các chương 1 đến chương 6. Một ứng dụng có hai thành phần chính: đối tượng ứng dụng biểu diễn chính ứng dụng và đối tượng cửa sổ biểu diễn cửa sổ ứng dụng. Nhiệm vụ đầu tiên của đối tượng ứng dụng là tạo đối tượng cửa sổ còn nhiệm vụ của đối tượng cửa sổ là xử lý các thông điệp.

MFC 2.0 đã thay đổi phương pháp làm việc sao cho các ứng dụng Windows có thể được viết trên cơ sở kiến trúc tài liệu/quan sát. Kiến trúc này được tiếp tục sử dụng cho đến MFC 6.0. Trong ứng dụng tài liệu/quan sát, đối tượng tài liệu biểu diễn dữ liệu ứng dụng còn một hay nhiều đối tượng quan sát sẽ biểu diễn quan sát của dữ liệu đó. Các đối tượng tài liệu và quan sát cùng xử lý đầu vào của người sử dụng, vẽ ký tự và biểu diễn đồ họa của dữ liệu kết quả. Lớp `CDocument` được sử dụng làm lớp cơ sở cho các đối tượng tài liệu, còn lớp `CView` và các lớp phái sinh của chúng làm lớp cơ sở cho các đối tượng quan sát. Cửa sổ mức trên cùng suy diễn từ `CFrameWnd` hay `CMDIFrameWnd` không còn là tiêu điểm để xử lý các thông điệp, nhưng để làm cái “chứa” quan sát, thanh công cụ, thanh trạng thái, và các đối tượng khác.

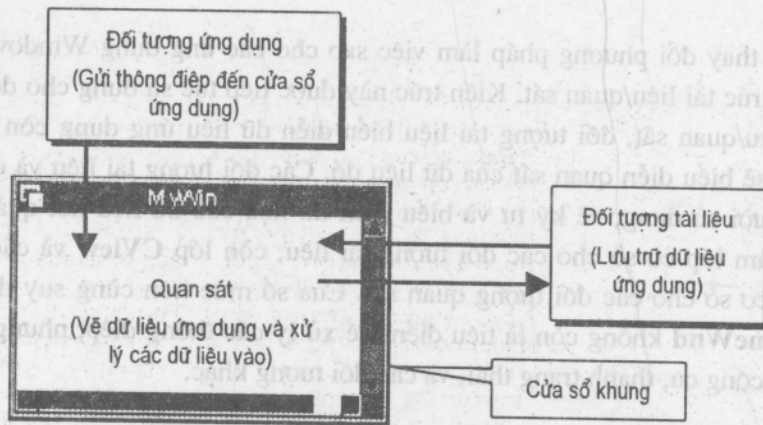
Kiến trúc tài liệu/quan sát đơn giản hoá tiến trình phát triển chương trình. Mã trình để thực hiện các công việc nhỏ khác như báo cho người sử dụng biết là có dữ liệu chưa được cất giữ trước khi ra khỏi soạn thảo tài liệu là do khung ứng dụng cung cấp. Nó cũng cung cấp các đoạn mã trình chuyển tài liệu ứng dụng vào bộ chứa OLE (*container*), đơn giản hoá in ấn, chia sẻ các cửa sổ và các công việc khác.

MFC trợ giúp hai loại ứng dụng tài liệu/quan sát. Loại thứ nhất là ứng dụng ghép nối đơn tài liệu (*Single-Document Interface - SDI*), chúng chỉ cho phép mở một tài liệu một lần. Loại thứ hai là ứng dụng ghép nối đa tài liệu (*Multiple-Document Interface - MDI*), cho phép mở hai hay nhiều tài liệu đồng thời. Ứng dụng MDI trợ giúp nhiều quan sát trên một tài liệu. Nhờ khung ứng dụng mà việc viết ứng dụng đa tài liệu chỉ phức tạp hơn chút ít so với viết ứng dụng đơn tài liệu. Nhưng vì tài liệu “*Windows Interface Guidelines for Software Design (Microsoft Press, 1995)*” và tài liệu trực tuyến của Visual C++ lại gợi ý là nên sử dụng SDI thay cho MDI, nên chương này chỉ giới thiệu tài liệu, quan sát và các khía cạnh khác của kiến trúc tài liệu/quan sát theo hướng SDI. Ta sẽ xem xét chương trình trò chơi *Life* mô tả cách viết ứng dụng tài liệu/quan sát.

7.1 CÁC VẤN ĐỀ CƠ BẢN CỦA TÀI LIỆU/QUAN SÁT

Chúng ta hãy bắt đầu tìm hiểu kiến trúc tài liệu/quan sát thông qua việc xem xét các đối tượng

liên quan khác và quan hệ của chúng. Cửa sổ khung là cửa sổ đỉnh của ứng dụng. Thông thường chúng có kiểu `WS_OVERLAPPEDWINDOW` với khung co giãn, thanh tiêu đề, thực đơn hệ thống, các hộp dẫn, co và đóng cửa sổ. Quan sát là cửa sổ con nằm khít trong cửa sổ khung sao cho chúng trở thành vùng làm việc của cửa sổ cha nó. Khi cửa sổ khung thay đổi kích thước, cửa sổ quan sát tự động thay đổi theo. Như đã đề cập trên, dữ liệu ứng dụng sẽ lưu trong đối tượng tài liệu, phần nhìn thấy của chúng là trên cửa sổ quan sát. Với ứng dụng SDI, cửa sổ khung được lập từ lớp phái sinh của `CFrameWnd`, tài liệu từ lớp phái sinh `CDocument`, và quan sát từ lớp phái sinh của `CView` hay các lớp liên quan như `CScrollView`.



Hình 7.1 Tài liệu/quan sát kiểu SDI

Kiến trúc này có rất nhiều gợi ý cho việc thiết kế và thao tác một chương trình ứng dụng. Trong ứng dụng MFC 1.0, dữ liệu chương trình thường được lưu trong các biến thành phần gán cho lớp cửa sổ khung. “Quan sát” của các dữ liệu này được vẽ nhờ thâm nhập trực tiếp các biến thành phần và sử dụng các hàm GDI gói trong lớp `CDC` để vẽ trên vùng làm việc của cửa sổ khung. Kiến trúc tài liệu/quan sát của MFC 2.x và các phiên bản mới hơn làm tăng thêm khả năng thiết kế chương trình theo mô đun bằng cách gói dữ liệu trong một đối tượng tài liệu độc lập và cung cấp đối tượng quan sát cho đầu ra màn hình của chương trình. Một ứng dụng tài liệu/quan sát không bao giờ chiếm ngữ cảnh thiết bị cho cửa sổ khung để vẽ vào vùng làm việc của cửa sổ này; thay vào đó chúng sử dụng đối tượng ngữ cảnh thiết bị cho cửa sổ quan sát để vẽ vào quan sát. Nó tương tự như vẽ vào cửa sổ khung, nhưng thực tế là toàn bộ sẽ ra cửa sổ quan sát nằm đè trên vùng làm việc của cửa sổ khung. Ta có thể vẽ vào cửa sổ khung nếu ta muốn, nhưng ta sẽ không thấy đầu ra vì vùng làm việc của cửa sổ khung SDI bị quan sát che phủ toàn bộ.

Một trong các khía cạnh thú vị nhất của ứng dụng tài liệu/quan sát SDI là cách thức các đối tượng cửa sổ khung, tài liệu và quan sát được tạo ra. Nếu xem xét bên trong hàm `InitInstance` của lớp ứng dụng do `AppWizard` của Visual C++ 6.0 tạo ra ta có thể thấy như sau:

```

CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate (
    IDR_MAINFRAME,
    RUNTIME_CLASS (CMyDoc),
    RUNTIME_CLASS (CMyFrameWnd),
    RUNTIME_CLASS (CMyView));
AddDocTemplate (pDocTemplate);
CCommandLineInfo cmdInfo;
ParseCommandLine (cmdInfo);
if (!ProcessShellCommand (cmdInfo))
    return FALSE;
return TRUE;

```

Không có một sự tương đồng nào dù là nhỏ nhất so với đoạn mã trình trong hàm `InitInstance` đã mô tả trong các ví dụ trước:

```

m_pMainWnd = new CMainWindow;
m_pMainWnd->ShowWindow (m_nCmdShow);
m_pMainWnd->UpdateWindow ();
return TRUE;

```

Ta hãy quan sát kỹ hơn đoạn mã do *AppWizard* tạo ra để hiểu chính xác chúng làm gì. Hai dòng lệnh đầu tạo đối tượng mẫu tài liệu SDI từ lớp `CSingleDocTemplate` của MFC. Mẫu tài liệu nhận ra các đối tượng của sổ khung, tài liệu, quan sát mà ứng dụng sẽ sử dụng. *Macro* `RUNTIME_CLASS` của MFC cho lại con trỏ đến cấu trúc `CRuntimeClass` của lớp đã chỉ ra, chúng cho phép tạo các đối tượng của lớp này khi chạy. Mẫu tài liệu còn nhận ra chỉ danh của tài nguyên (`IDR_MAINFRAME` là ID được *AppWizard* gán tự động, nhưng ta có thể gán bởi bất kỳ số nguyên khác không nào) của một số tài nguyên có quan hệ gần với cửa sổ khung. Ta sẽ bàn kỹ hơn về ID của tài nguyên trong các trang sau.

Tiếp theo, dòng lệnh:

```
AddDocTemplate (pDocTemplate);
```

thêm mẫu tài liệu vào danh sách các mẫu tài liệu do đối tượng quản lý. Mỗi mẫu tài liệu được đăng ký theo kiểu này sẽ xác định một kiểu tài liệu mà ứng dụng sẽ trợ giúp. Các ứng dụng SDI chỉ đăng ký một loại tài liệu, nhưng các ứng dụng MDI có thể đăng ký nhiều bao nhiêu cũng được.

Các lệnh:

```

CCommandLineInfo cmdInfo;
ParseCommandLine (cmdInfo);

```

sử dụng hàm `CWinApp::ParseCommandLine` của MFC 6.0 để khởi động đối tượng `CCommandLineInfo` bằng các giá trị phản ánh các tham số vào từ dòng lệnh, bao gồm cả tên tệp

tài liệu. Còn các lệnh:

```
if (!ProcessShellCommand (cmdInfo))
    return FALSE;
```

sử dụng tham số dòng lệnh. Ngoài các việc khác, **ProcessShellCommand** gọi **CWinApp::OnFileNew** để khởi động ứng dụng với tài liệu rỗng nếu không có tên tệp trên dòng lệnh, hay gọi **CWinApp::OpenDocumentFile** để khởi động ứng dụng và nạp tài liệu nếu tên tài liệu đã được chỉ ra. Khi chương trình chạy đến giai đoạn này, khung ứng dụng tạo các đối tượng cửa sổ khung, tài liệu và quan sát nhờ thông tin lớp đang chạy chứa trong mẫu tài liệu. **ProcessShellCommand** cho lại **TRUE** nếu khởi động được và **FALSE** nếu không được. Khi giá trị cho lại là **FALSE** thì **InitInstance** cũng cho lại **FALSE** và đóng ứng dụng.

Sau khi ứng dụng được khởi động và các đối tượng cửa sổ khung, tài liệu và quan sát được lập, vòng lặp thông điệp được thực hiện và ứng dụng bắt đầu lấy và xử lý các thông điệp. Không như MFC 1.0, các ứng dụng tài liệu/quan sát chia sẻ các công việc xử lý thông điệp giữa ứng dụng và các đối tượng cửa sổ khung, tài liệu và quan sát. Khung ứng dụng thực hiện rất nhiều công việc ở nền. Trong Windows, chỉ các cửa sổ có thể nhận thông điệp vậy nên khung ứng dụng mới cài đặt cơ chế dẫn lệnh phức tạp dẫn đường cho các thông điệp - đó là khái niệm của MFC cho các thông điệp **WM_COMMAND** phát sinh từ các thực đơn và phím trên thanh công cụ - từ một đối tượng đến đối tượng khác theo trình tự xác định trước, cho đến khi một đối tượng nào đó xử lý thông điệp hay thông điệp chuyển cho **::DefWindowProc**.

7.2 ĐỐI TƯỢNG TÀI LIỆU

Trong ứng dụng tài liệu/quan sát, dữ liệu được lưu trong đối tượng tài liệu của lớp phái sinh từ **CDocument**. Khái niệm “tài liệu” hay gây nhầm lẫn vì nó thường được hiểu dưới cách nhìn của các chương trình xử lý văn bản hay trang tính và các ứng dụng khác làm cho ta suy nghĩ đến tài liệu văn bản truyền thống. Trong thực tế, kiến trúc tài liệu/quan sát còn có ý nghĩa lớn hơn nhiều. Một tài liệu có thể là bất kỳ cái gì. “Tài liệu” trong “tài liệu/quan sát” đề cập đến biểu diễn trừu tượng dữ liệu chương trình, vạch ra đường ngăn cách rõ ràng giữa việc lưu giữ dữ liệu và biểu diễn dữ liệu. Thông thường, đối tượng tài liệu cung cấp các hàm thành phần chung mà các đối tượng khác có thể thâm nhập dữ liệu tài liệu. Mọi quản lý dữ liệu được thực hiện nhờ chính đối tượng tài liệu. Dữ liệu tài liệu thông thường được lưu trong các biến thành phần di theo lớp tài liệu phái sinh. Trong thí dụ **Scribble** của Visual C++, dữ liệu được đưa ra trực tiếp cho các đối tượng khác nhờ khai báo tổng thể, nhưng chúng được gói nghiêm ngặt nhờ thực hiện cục bộ dữ liệu tài liệu và cung cấp các hàm thành phần tổng thể để thâm nhập chúng. Thí dụ, đối tượng tài liệu trong chương trình soạn thảo văn bản có thể lưu trong đối tượng **CByteArray** và cung cấp các hàm **AddChar**, **RemoveChar** sao cho các đối tượng khác (chẳng hạn quan sát) có thể thêm, bớt các ký tự. Các hàm khác như **AddLine** và **DeleteLine** làm phong phú thêm các ghép nối cho đối tượng tài liệu và

các đối tượng tương tác với chúng.

Bảng 7.1 liệt kê một số hàm thành phần quan trọng mà đối tượng tài liệu thừa hưởng từ lớp **CDocument**. Hàm **SetModifiedFlag** được gọi mỗi khi dữ liệu tài liệu thay đổi. Hàm **SetModifiedFlag** sẽ lập cờ bên trong đối tượng tài liệu để cho khung ứng dụng biết rằng tài liệu có dữ liệu chưa được lưu trên đĩa. Ta có thể tự xác định xem tài liệu có chứa dữ liệu chưa lưu lại trên đĩa không nhờ gọi hàm **IsModified** của tài liệu. Hàm **GetTitle** và **GetPathName** lấy tên tệp của tài liệu và đường dẫn đầy đủ của tệp. Cả hai hàm cho lại đối tượng null kiểu **CString** nếu tài liệu không có tiêu đề, như vậy, nếu tài liệu chưa được lưu trên đĩa thì cũng chưa được gán tên tệp. Hàm mới **OnFileSendMail** trong lệnh *Send Mail* của thực đơn *File* của ứng dụng thực hiện vào/ra tài liệu đến/đi từ tệp đệm và gửi tệp như thông điệp thư điện tử. (Một tài liệu có tiêu đề khi chúng được gán tên tệp).

Bảng 7.1 Các thao tác cơ bản của CDocument

<i>Function</i>	<i>Description</i>
GetFirstViewPosition	Cho lại giá trị POSITION để truyền cho GetNextView để đếm các quan sát tài liệu.
GetNextView	Cho lại con trỏ CView trở đến quan sát tiếp theo trong danh sách các quan sát gắn với tài liệu.
GetPathName	Lấy tên tệp và đường dẫn của tài liệu, thí dụ "C:\Documents\Personal\MyFile.Lif." Cho lại xâu rỗng nếu tài liệu chưa được đặt tên.
GetTitle	Lấy tiêu đề tài liệu, thí dụ "MyFile." Cho lại xâu rỗng nếu tài liệu chưa được đặt tên.
IsModified	Cho lại giá trị khác 0 nếu tài liệu chứa dữ liệu chưa lưu lên đĩa và 0 nếu đã lưu.
SetModifiedFlag	Đặt hoặc xóa cờ cho biết tài liệu đã thay đổi.
UpdateAllViews	Làm tươi toàn bộ các quan sát liên quan đến tài liệu bằng cách gọi hàm OnUpdate cho mỗi quan sát.
OnFileSendMail	Cài đặt lệnh <i>Send Mail</i> trong thực đơn <i>File</i> .

Ba trong các hàm của **CDocument** trên bảng 7.1 cho phép đối tượng tài liệu có thể tương tác với quan sát. Hàm **UpdateAllViews** thông báo tới các quan sát liên quan đến tài liệu để tự làm tươi nhờ kích hoạt các hàm ảo **OnUpdate** của quan sát. Trong ứng dụng SDI với chỉ một quan sát, khi có thông tin vào hay trước/sau khi cập nhật dữ liệu tài liệu hàm **UpdateAllViews** thường không được sử dụng để tự làm tươi quan sát. Nhưng trong các ứng dụng MDI, hàm **UpdateAllViews** được gọi khi tài liệu thay đổi để giữ đồng bộ cho tất cả các quan sát. Nếu muốn, đối tượng tài liệu có thể đếm duyệt các quan sát của chúng và liên kết với từng quan sát nhờ các hàm **GetFirstViewPosition** và **GetNextView** để duyệt danh sách quan sát. Đoạn mã lệnh sau mô tả hoạt động của **UpdateAllViews** nhờ gọi riêng rẽ hàm **OnUpdate** của mỗi đối tượng quan sát:

```
POSITION pos = GetFirstViewPosition ();
while (pos != NULL)
    GetNextView (pos)->OnUpdate (NULL, 0, NULL);
```

Đĩ nhiên, chỉ gọi hàm **UpdateAllViews** là để hơn trừ khi phải chuyển danh sách tham số cho các

quan sát khác nhau hay bỏ qua một số quan sát.

CDocument còn cung cấp tập các hàm ảo để nạp chồng. Một số thường xuyên phải nạp chồng trong các lớp tài liệu phái sinh, một số khác rất ít khi nạp chồng. Bốn hàm thường hay được nạp chồng nhất được mô tả trong danh sách trên bảng 7.2.

Bảng 7.2 Các hàm nạp chồng của CDocument

Hàm	Mô tả
OnNewDocument	Khung ứng dụng gọi hàm này khi tài liệu mới được tạo. Nạp chồng để khởi động lại đối tượng tài liệu khi tài liệu mới được tạo.
OnOpenDocument	Khung ứng dụng gọi hàm này khi tài liệu nạp từ đĩa. Nạp chồng để khởi động lại các thành phần dữ liệu không vào/ra của đối tượng tài liệu trước khi tài liệu mới được nạp.
DeleteContents	Khung ứng dụng gọi hàm này để xóa nội dung tài liệu. Nạp chồng để giải phóng vùng nhớ và các tài nguyên khác do tài liệu thuê trước khi bị đóng.
Serialize	Khung ứng dụng gọi hàm này để vào/ra tài liệu từ/đến đĩa. Nạp chồng sẽ cung cấp mã trình vào/ra riêng của tài liệu sao cho tài liệu có thể đọc/ghi.

OnNewDocument khởi động tài liệu mới khi nó được tạo, hàm **OnOpenDocument** khởi động các thành phần dữ liệu không liên quan đến vào/ra của đối tượng tài liệu khi tài liệu mới được nạp từ đĩa từ. Trong ứng dụng SDI, đối tượng tài liệu được tạo ra chỉ một lần và sử dụng lại mỗi khi tài liệu được tạo hay mở. Vì cấu trúc của đối tượng chỉ được thực hiện một lần không phụ thuộc vào việc bao nhiêu tài liệu mở hay đóng, ứng dụng SDI phải thực hiện khởi động một lần trong cấu trúc tài liệu và đặt mã khởi động cho mỗi tài liệu mới trong **OnNewDocument** và **OnOpenDocument**.

Trước khi mở hay tạo tài liệu, khung ứng dụng gọi hàm ảo **DeleteContents** của tài liệu để xóa các dữ liệu có sẵn của tài liệu. Một ứng dụng SDI phải nạp chồng **CDocument::DeleteContents** và giải phóng bất kỳ tài nguyên nào mà tài liệu thuê và thực hiện các chương trình con khác nhằm chuẩn bị cho đối tượng tài liệu được sử dụng lại. Các ứng dụng MDI về tổng thể cũng tuân thủ mô hình này, mặc dù các đối tượng tài liệu MDI khác với các đối tượng tài liệu SDI, được tạo và lập riêng rẽ như là khi người sử dụng tạo, mở, đóng tài liệu.

Khi nạp chồng **OnNewDocument** và **OnOpenDocument**, điều quan trọng là phải gọi các bản lớp cơ sở của chúng. Nếu không, hàm **DeleteContents** sẽ không được gọi khi tài liệu mới được tạo hay nạp và khung ứng dụng không thể khởi động các công việc quan trọng khác. Thí dụ, bản CDocument của **OnOpenDocument** không chỉ gọi **DeleteContents** mà nó còn hiển thị hộp thoại Mở để lấy tên tệp từ người sử dụng và gọi hàm **Serialize** của đối tượng tài liệu để vào/ra dữ liệu tài liệu từ đĩa từ. Hàm nạp chồng trước hết phải gọi cài đặt lớp cơ sở và dừng lại nếu lớp cơ sở cho lại giá trị FALSE, và nó phải cho lại giá trị TRUE để thông báo kết thúc công việc:

```
BOOL CMyDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument()) return FALSE;
```

```

// Khởi động đối tượng tài liệu như là mới...
return TRUE;
}
BOOL CMyDoc::OnOpenDocument (LPCTSTR lpszPathName)
{
    if (!CDocument::OnOpenDocument (lpszPathName))    return FALSE;
    // Khởi động các thành phần của đối tượng tài liệu chưa được khởi
    // động khi tài liệu vào/ra từ đĩa...
    return TRUE;
}

```

Khi mở hay ghi tài liệu, khung ứng dụng gọi hàm `Serialize` của đối tượng tài liệu để đọc hay ghi dữ liệu tài liệu. Vào/ra (*Serialization*) là tiến trình mà đối tượng thực hiện ghi bản ghi vào vật mang thông tin như tệp đĩa, hay lấy ra bằng cách đọc ra. Ta viết hàm `Serialize` để chuyển dữ liệu tài liệu vào và ra; khung ứng dụng làm các công việc còn lại, bao gồm mở tệp để đọc hay ghi và cung cấp đối tượng `CArchive` để tách ta ra khỏi vào/ra đĩa vật lý. Thông thường, toàn bộ tài liệu có thể vào/ra chỉ bằng vài dòng mã lệnh nhờ toán tử xen và tách của `CArchive`. Nếu cần, đối tượng tài liệu có thể thực hiện đọc thô và viết qua đối tượng `CFile` (có thể lấy con trỏ của chúng nhờ hàm `CArchive::GetFile`) hay sử dụng các hàm `Read` và `Write` của đối tượng `CArchive`.

Các hàm nạp chồng khác của `CDocument` ít được sử dụng, nhưng chúng hữu ích trong một số trường hợp, thí dụ khi gọi `OnCloseDocument` để đóng tài liệu; gọi `OnSaveDocument` để lưu tài liệu; `SaveModified` để lưu dữ liệu thay đổi, bao gồm việc hỏi lại người sử dụng; và `ReportSaveLoadException` khi có lỗi vào/ra xảy ra.

7.3 ĐỐI TƯỢNG QUAN SÁT

Trong khi mục tiêu duy nhất của đối tượng tài liệu là lưu dữ liệu ứng dụng, các đối tượng quan sát có hai mục đích: biểu diễn dữ liệu tài liệu trên màn hình và chuyển đổi đầu vào từ người ứng dụng thành các lệnh thao tác tài liệu - trừ các thông điệp chuột và bàn phím, chúng không được chuyển đến đối tượng tài liệu như các thông điệp lệnh. Như vậy tài liệu và quan sát có quan hệ mật thiết và thông tin trao đổi giữa chúng là hai chiều.

Đối tượng tài liệu có thể có số lượng bất kỳ các quan sát liên quan đến nó, nhưng mỗi quan sát chỉ gắn với một đối tượng tài liệu. Khung ứng dụng lưu con trỏ đến đối tượng tương ứng trong thành phần dữ liệu `m_pDocument` của quan sát và con trỏ này được thâm nhập qua hàm thành phần `GetDocument`. Cũng như đối tượng tài liệu có thể nhận ra các quan sát của chúng nhờ các hàm `CDocument::GetFirstViewPosition` và `CDocument::GetNextView`, đối tượng quan sát có thể tìm thấy tài liệu của nó nhờ gọi hàm `GetDocument`. Khi *AppWizard* tạo mã nguồn cho đối tượng quan sát, chúng nạp chồng hàm `GetDocument` của lớp cơ sở để ép (*cast*) `m_pDocument` thành kiểu

tài liệu tương ứng và cho lại kết quả. Điều này cho phép đối tượng tài liệu được thâm nhập một cách an toàn mà không cần ép lại kiểu mỗi khi qui chiếu.

Lớp CView của MFC định nghĩa các tính chất của quan sát, và các lớp quan sát phái sinh chia sẻ chức năng phụ. MFC 6.0 thêm một số lớp quan sát mới vào danh sách quan sát, bao gồm CListView, CRichEditView, CTreeView và CDaoRecordView. Như lớp CDocument, CView và các lớp phái sinh của chúng bao gồm một số hàm thành phần ảo mà ta có thể nạp chồng để thực hiện các thao tác quan sát riêng. Hàm quan trọng nhất là OnDraw, nó được gọi mỗi khi quan sát nhận thông điệp WM_PAINT. Trong ứng dụng không phải cấu trúc tài liệu/quan sát, các thông điệp WM_PAINT được xử lý nhờ hàm OnPaint, hàm này sử dụng đối tượng CPaintDC để vẽ. Trong ứng dụng tài liệu/quan sát, khung ứng dụng nhận thông điệp WM_PAINT và gọi hàm OnDraw của lớp quan sát, chuyển con trỏ CDC cho chúng để vẽ. Không cần ánh xạ thông điệp bởi vì OnDraw là ảo. Hàm OnDraw hiển thị xâu "Hello, MFC" vào giữa màn hình của cửa sổ quan sát như sau:

```
void CMyView::OnDraw (CDC* pDC)
{
    CRect rect;
    GetClientRect (&rect);
    pDC->DrawText ("Hello, MFC", -1, rect, DT_SINGLELINE | DT_CENTER | DT_VCENTER);
}
```

Thật ra, quan sát không tạo đối tượng ngữ cảnh thiết bị riêng cũng có ít nhiều bất tiện. Lý do chính mà khung ứng dụng sử dụng OnDraw là cùng mã trình có thể sử dụng cho cả ra màn hình và máy in. Khi thông điệp WM_PAINT đến, khung ứng dụng chuyển cho đối tượng quan sát một con trỏ đến DC vẽ để cho đầu ra là màn hình. Khi thực hiện in, OnDraw sẽ được chuyển con trỏ đến DC máy in đưa đầu ra đến máy in.

Hai hàm nạp chồng khác của CView cũng có ích khi suy diễn các lớp quan sát riêng là OnInitialUpdate và OnUpdate. Hàm OnInitialUpdate được gọi để khởi động quan sát ngay sau khi nó được lập và, trong ứng dụng SDI, mỗi khi đóng tài liệu, tài liệu mới được mở hay tạo. Cài đặt mặc định gọi OnUpdate, và cài đặt mặc định của OnUpdate sẽ làm mất hiệu lực của vùng làm việc. Sử dụng OnInitialUpdate để khởi động các thành phần dữ liệu của lớp quan sát. Thứ tự, trong lớp phái sinh từ CScrollView, OnInitialUpdate thường gọi hàm CScrollView::SetScrollSizes để khởi động kích thước dòng, trang, tham số trượt của tài liệu. Trong ứng dụng đa quan sát, quan sát có thể xác định khi nào là tích cực và vô hiệu lực nhờ nạp chồng hàm OnActivateView của CView. Tham số thứ nhất chuyển cho OnActivateView có giá trị TRUE nếu quan sát đang tích cực và FALSE nếu không. Tham số thứ hai và ba tương ứng là con trỏ CView chỉ ra các quan sát đang tích cực và không. Nếu các con trỏ giống nhau thì cửa sổ khung của ứng dụng được kích hoạt mà không gây ra thay đổi trong quan sát tích cực. Các đối tượng quan sát đôi khi sử dụng đặc trưng đặc biệt này của hàm OnActivateView để thực hiện bộ màu logic hay các nhiệm vụ khác mà nó chỉ được

thực hiện khi người sử dụng quay lại ứng dụng sau khi làm tích cực ứng dụng khác.

7.4 ĐỐI TƯỢNG CỦA SỔ KHUNG

Trong ứng dụng tài liệu/quan sát, cửa sổ khung xác định vùng làm việc vật lý trên màn hình của ứng dụng và khung quan sát của dữ liệu. Các cửa sổ khung SDI có từ lớp `CFrameWnd`. Các cửa sổ khung MDI mức đỉnh có từ `CMDIFrameWnd`. Lớp `CMDIChildWnd` liên quan xác định ứng xử của các cửa sổ con làm khung cho quan sát MDI và trôi nổi trong khung MDI mức đỉnh.

Các cửa sổ khung là một phần quan trọng của kiến trúc tài liệu/quan sát. Thứ dụ, lớp `CFrameWnd` xây dựng trong các hàm `OnClose` và `OnQueryEndSession` cho người sử dụng cơ hội lưu các thay đổi đến tài liệu nếu đóng cửa sổ ứng dụng hay kết thúc Windows. Trong ứng dụng đa quan sát, cửa sổ khung đổi theo quan sát tích cực và gọi các hàm `OnActivateView` của quan sát khi quan sát tích cực bị thay đổi. Cửa sổ khung cũng quản lý hầu hết các công việc quan trọng như thay đổi kích cỡ của cửa sổ quan sát khi cửa sổ khung bị thay đổi; thanh công cụ xuất hiện hay không; nó bao gồm các hàm thành phần để dấu đi hay hiển thị thanh trạng thái và thanh công cụ.

7.5 TẠO LẬP ĐỘNG CÁC ĐỐI TƯỢNG

Để khung ứng dụng tạo các đối tượng tài liệu, quan sát, và cửa sổ khung trong khi chương trình đang chạy, các lớp từ đó các đối tượng được xây dựng phải trợ giúp một đặc trưng gọi là "tạo động". Cơ chế tạo động đối tượng của MFC là cách cho các ứng dụng đăng ký các lớp theo phương pháp mà khung ứng dụng có thể tạo các đối tượng của các lớp này. MFC cho phép dễ dàng viết các lớp có thể tạo động nhờ các *macro* `DECLARE_DYNCREATE` và `IMPLEMENT_DYNCREATE`. `DECLARE_DYNCREATE` được gọi từ khai báo lớp với tên lớp làm tham số, và `IMPLEMENT_DYNCREATE` được gọi trong tệp cài đặt của lớp với hai tham số: tên lớp và tên của lớp cơ sở. Một đối tượng của lớp sử dụng macro này có thể được tạo lập khi chương trình đang chạy nhờ dòng lệnh như sau:

```
RUNTIME_CLASS(CMyClass)->CreateObject();
```

Về cơ bản không có gì khác với việc dùng toán tử `new` để tạo đối tượng `CMyClass`, nhưng nó vượt qua được nhược điểm của C++ là tránh được các dòng lệnh như sau:

```
CString strClassName = "CMyClass";
CMyClass* ptr = new strClassName;
```

Chương trình dịch sẽ thử xây dựng đối tượng từ lớp với tên "strClassName" bởi vì nó không nhận ra rằng `strClassName` là tên biến chứ không phải tên lớp.

Cái gì xảy ra khi ta viết một lớp có thể tạo động? Thật đơn giản. Macro `DECLARE_DYNCREATE` thêm ba thành phần vào khai báo lớp: thành phần dữ liệu tĩnh có kiểu

CRuntimeClass, hàm ảo có tên GetRuntimeClass, và hàm tĩnh có tên CreateObject. Khi ta viết:

```
DECLARE_DYNAMIC(CMyClass)
```

thì chương trình dịch sẽ hiểu là:

```
public:
    static AFX_DATA CRuntimeClass classCMyClass;
    virtual CRuntimeClass* GetRuntimeClass() const;
    static CObject* PASCAL CreateObject();
```

Macro IMPLEMENT_DYNCREATE khởi động cấu trúc CRuntimeClass với các thông tin như tên lớp, kích thước của các đối tượng được tạo từ lớp, và địa chỉ của lớp cơ sở. Nó cũng cung cấp các mã lệnh cho các hàm GetRuntimeClass và CreateObject. Nếu macro IMPLEMENT_DYNCREATE của MFC được gọi như sau:

```
IMPLEMENT_DYNCREATE(CMyClass, CBaseClass)
```

thì CreateObject được cài đặt như sau:

```
CObject* PASCAL class_name::CreateObject()
{ return new CMyClass; }
```

Một khi lớp có thể tạo động nạp chồng toán tử new thì CreateObject sẽ sử dụng bản nạp chồng.

7.6 SERIALIZATION

Vào/ra (*Serialization*) là tiến trình nhờ nó đối tượng có thể ghi bản ghi vào vật mang thông tin như tệp đĩa hay nạp ra nhờ đọc lại bản ghi. *Serialization* là chủ đề khá rộng nhưng có thể tóm lại như sau: nhờ nạp chồng CDocument::Serialize trong lớp tài liệu và sử dụng đối tượng cho trước CArchive để vào/ra các thành phần dữ liệu của tài liệu, khung ứng dụng sẽ trợ giúp việc cài đặt các lệnh như Mở tệp, Ghi tệp, và Ghi tệp với tên mới trong thực đơn Tệp. Việc đọc ghi tài liệu trở nên vô cùng đơn giản.

Giả sử dữ liệu trong tài liệu bao gồm hai thành phần dữ liệu *int* với tên m_nWidth và m_nHeight. Hàm Serialize của tài liệu có thể như sau:

```
void CMyDoc::Serialize(CArchive& archive)
{
    if (archive.IsStoring())
        archive << m_nWidth << m_nHeight;
    else
        archive >> m_nWidth >> m_nHeight;
}
```

Biến *archive* là đối tượng kiểu CArchive sử dụng như đệm giữa tài liệu và tệp đĩa vật lý biểu

diễn bởi đối tượng `CFile`. Hàm `CArchive::IsStoring` cho lại `TRUE` nếu tài liệu đang được ghi và `FALSE` nếu tài liệu đang được đọc. Lớp `CArchive` nạp/chống các toán tử tách (`<<`) và xen (`>>`) sao cho các kiểu dữ liệu cơ bản như `BYTE`, `WORD`, `DWORD`, `LONG`, `int`, `float`, và `double` có thể vào/ra dễ dàng. Các kiểu dữ liệu của MFC như `CString` và `CRect` cũng có thể được ghi và đọc theo cùng cách đó. MFC 6.0 trợ giúp trực tiếp vào/ra kiểu dữ liệu `int`.

Các lớp có thể vào/ra như các kiểu dữ liệu cơ sở bằng cách suy diễn lớp từ `CObject`, đưa vào một vài *macro*, thêm hàm `Serialize` để vào/ra các thành phần dữ liệu của lớp. MFC xây dựng trợ giúp vào/ra trong rất nhiều lớp của chúng, bao gồm các lớp tập hợp để lưu dữ liệu. Nếu các ký tự và các mã khuôn mẫu ở trong tài liệu xử lý văn bản đã được lưu trong đối tượng `CByteArray` với tên `m_byData`, thì toàn bộ tài liệu - có thể dài 1 byte hay 1 Mbyte - đều có thể vào/ra bằng hàm `Serialize` một dòng lệnh như sau:

```
void CMyDoc::Serialize (CArchive& archive)
{
    m_byData.Serialize (archive);
}
```

7.7 MẪU TÀI LIỆU SDI

Trên đây đã trình bày đối tượng mẫu tài liệu SDI được tạo từ lớp `CSingleDocTemplate`. Cấu tử của đối tượng nhận bốn tham số: số nguyên cho `IDR_MAINFRAME` và ba con trỏ kiểu `RUNTIME_CLASS`. Mục đích của *macro* `RUNTIME_CLASS` đã rõ ràng, ta xem xét kỹ hơn số nguyên của tham số thứ nhất, chúng là ID tài nguyên đa năng xác định các tài nguyên sau đây:

- Biểu tượng ứng dụng
- Thực đơn liên quan đến cửa sổ khung
- Bảng phím lệnh thực đơn theo thực đơn
- Xâu tài liệu

Trong ứng dụng tài liệu/quan sát SDI, khung ứng dụng tạo lập cửa sổ khung ứng dụng nhờ gọi hàm `CFrameWnd::LoadFrame`, nó nhận ID tài nguyên để phân biệt bốn loại tài nguyên vừa nói trên. Như mong đợi, mẫu tài liệu chuyển ID tài nguyên cho `LoadFrame` khi nó tạo lập cửa sổ khung. `LoadFrame` tạo lập cửa sổ khung và nạp thực đơn, bảng phím lệnh, biểu tượng và xâu tài liệu liên quan trong cùng một bước, nhưng để chúng làm việc thì phải gán cùng ID cho tất cả các tài nguyên. Chính vì vậy, tệp `RC` do *AppWizard* sinh ra sử dụng cùng ID (mặc định, `IDR_MAINFRAME`) cho các tài nguyên khác nhau.

Xâu tài liệu là tài nguyên xâu hình thành từ tổ hợp bảy xâu con cách nhau bởi ký tự `\n`, mỗi xâu con mô tả một đặc trưng của cửa sổ khung hay tài liệu liên quan tới mẫu tài liệu. Theo trình tự từ trái sang phải, các xâu con chỉ ra:

- Tên sẽ hiển thị trong thanh tiêu đề của cửa sổ khung (chỉ với các ứng dụng SDI). Thường là tên của ứng dụng, thí dụ "Microsoft Draw".
- Tên mặc định gán cho các tài liệu mới. Nếu không có xâu này thì mặc định là "Untitled".
- Tên mô tả cho kiểu tài liệu đi kèm theo với các kiểu tài liệu khác trong hộp thoại khi người sử dụng chọn New trong thực đơn File, thí dụ "Spreadsheet" hay "Drawing". Đầu vào này cần cho ứng dụng MDI để đăng ký hai hay nhiều kiểu tài liệu.
- Tên mô tả cho các tài liệu kiểu này tổ hợp với phần mở rộng tên tệp mặc định, thí dụ "Draw files (*.DRW)". Xâu này được sử dụng trong mảng kiểu tệp của hộp thoại Open và Save As.
- Phần mở rộng của tên cho các tài liệu của kiểu này, thí dụ DRW.
- Tên không chứa dấu cách chỉ ra kiểu tài liệu trong đăng ký, thí dụ "MicrosoftDraw.Document". Khi ứng dụng gọi CWinApp::RegisterShellFileTypes để đăng ký các kiểu tài liệu của nó, xâu này trở thành giá trị mặc định cho các khóa con HKEY_CLASSES_ROOT được đặt tên theo phần mở rộng tên của tệp tài liệu.
- Tên mô tả cho kiểu tài liệu, thí dụ "Microsoft Draw drawing file", khác với xâu đi trước nó trong xâu tài liệu, xâu con này có thể chứa dấu cách. Nếu ứng dụng gọi CWinApp::RegisterShellFileTypes để đăng ký kiểu tài liệu của nó trong hệ thống, xâu con này là tên kiểu tài liệu có thể đọc được mà Windows 95/98 sẽ hiển thị trong trang đặc tính cho tài liệu.

Có thể để thiếu.xâu con khi tạo tài nguyên xâu tài liệu; từng xâu con có thể thiếu sau dấu \n, nếu giá trị là NULL thì thiếu toàn bộ xâu con. Nếu sử dụng *AppWizard* để xây dựng ứng dụng, xâu tài liệu được tự động tạo ra cho ta. Các lệnh tài nguyên cho một xâu tài liệu SDI điển hình như sau:

```

STRINGTABLE
BEGIN
    IDR_MAINFRAME "Draw\n\nDraw files(*.drw)\n.n.drw\n\nDraw.Document\nDraw Document"
END

```

Khi ứng dụng này bắt đầu chạy với tài liệu trống, cửa sổ khung sẽ có tiêu đề "Untitled - Draw".
Phần mở rộng tên tệp mặc định để lưu tài liệu là DRW, và DRW sẽ là một trong các phần mở rộng
tên tệp sẽ liệt kê trong các hộp thoại *Open* và *Save As*.

Sau khi mẫu tài liệu được tạo, các xâu con đi theo xâu tài liệu có thể lấy ra nhờ hàm `CDocTemplate::GetDocString` của MFC. Thí dụ lệnh sau:

```
CString strDefExt;  
pDocTemplate->GetDocString(strDefExt, CDocTemplate::filterExt);
```

sao chép phần mở rộng tên tệp mặc định vào CString với tên strDefExt.

7.8 ĐĂNG KÝ KIỂU TÀI LIỆU

Ngoài việc tạo ra các tài liệu, quan sát và cửa sổ khung đi kèm theo quan sát, một trong các hàm quan trọng nhất của mẫu tài liệu là đăng ký kiểu tài liệu với hệ điều hành. Trong Windows 95/98, khi nhấn đúp chuột trên biểu tượng tài liệu hay nhấn phím chuột phải rồi chọn thực đơn *Open* từ thực đơn ngữ cảnh, tài liệu và ứng dụng tạo ra nó sẽ được mở. Để làm việc này, kiểu tài liệu phải được đăng ký với Windows, có nghĩa phải ghi các dãy ký tự vào bản đăng ký để chỉ ra phân mở rộng tên tệp tài liệu và lệnh sẽ sử dụng để mở chúng.

Trong ứng dụng Windows, việc đăng ký được thực hiện bằng cách cung cấp tệp REG mà người sử dụng có thể trộn vào bản đăng ký của hệ điều hành hay bằng cách lập trình để viết các đầu vào cần thiết vào bản đăng ký nhờ các hàm API của Win32 như `::RegCreateKey` và `::RegSetValue`. Trong ứng dụng MFC, tiến trình đơn giản hơn nhiều. Gọi hàm `CWinApp::RegisterShellFileTypes` sau khi gọi `AddDocTemplate` trong `InitInstance` sẽ làm thay đổi bản đăng ký để liên kết ứng dụng tài liệu mà nó tạo ra và Windows 95/98.

Hàm `CWinApp` liên quan có tên `EnableShellOpen`, thường được gọi để liên kết với `RegisterShellFileTypes`, sẽ tạo ra liên kết DDE giữa ứng dụng và shell để thêm đặc tính vào các ứng dụng MDI. Nếu ứng dụng MDI chạy và nó đăng ký kiểu tài liệu của chúng với `EnableShellOpen`, và khi người sử dụng mở một trong các biểu tượng tài liệu của ứng dụng qua *shell*, *shell* sẽ không khởi động bản thứ hai của ứng dụng; thay vào đó chúng gửi thông điệp DDE đến bản đang tồn tại để ra lệnh chúng tự mở tài liệu. Như vậy, tài liệu sẽ xuất hiện trong cửa sổ khung mới của khung MDI mức đỉnh, đúng như là nó vừa được mở bằng lệnh *File Open* của ứng dụng.

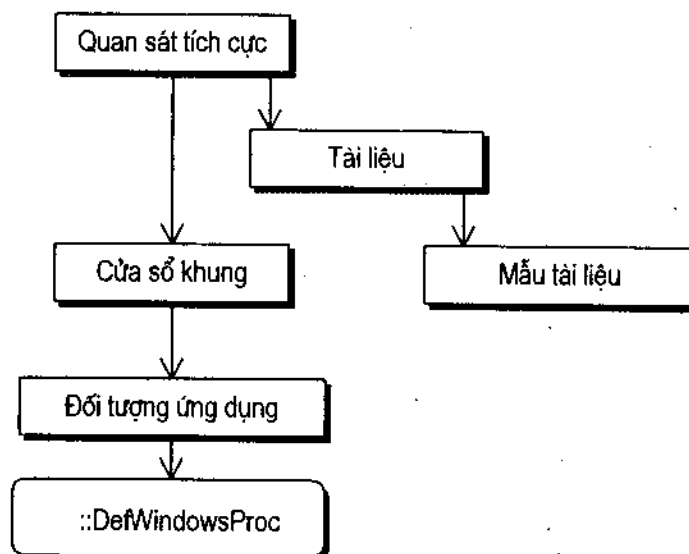
Ta có thể thêm đặc tính di-và-nhả vào ứng dụng tài liệu/quan sát bằng cách gọi hàm `DragAcceptFiles` của cửa sổ khung. Điều này đăng ký cửa sổ để nhận các thông điệp `WM_DROPFILES`. Hàm `OnDropFiles` của `CFrameWnd` trả lời thông báo “nhả” bằng gọi hàm `OnOpenDocument` của đối tượng ứng dụng với tên tệp mà nó vừa “nhả”. Hàm `DragAcceptFiles` thông thường được gọi từ `InitInstance` thông qua con trỏ `m_pMainWnd` lưu trong đối tượng ứng dụng. Trong ứng dụng SDI, `DragAcceptFiles` phải được gọi sau `ProcessShellCommand` bởi vì cửa sổ khung không tồn tại khi chưa gọi `ProcessShellCommand`.

7.9 ĐƯỜNG DẪN LỆNH

Một trong các đặc trưng đáng lưu ý nhất của MFC và kiến trúc tài liệu/quan sát là ứng dụng có thể quản lý các thông điệp lệnh đi tới từ các sự kiện giao diện ứng dụng UI chẳng hạn như chọn thực đơn, hầu như ở bất cứ chỗ nào. Cửa sổ khung là nơi nhận nhiều thông điệp lệnh nhất, nhưng có thể quản lý các thông điệp lệnh trong đối tượng quan sát, đối tượng tài liệu, hay đối tượng ứng

dụng bằng cách đơn giản thêm đầu vào ánh xạ thông điệp trong định nghĩa lớp cho các thông điệp ta muốn quản lý. Điều này có một hiệu quả rất thực tế là ta được tự do quản lý các thông điệp tại bất kỳ lớp nào, không cần quản lý các thông điệp chỉ ở lớp cửa sổ khung.

Trong trình tự chương trình, các thông điệp lệnh đã gửi tới cửa sổ khung SDI theo đường dẫn trên hình 7.2. Quan sát tích cực trước hết nhận thông điệp, tiếp theo chuyển đến đối tượng tài liệu liên quan với quan sát, mẫu tài liệu, cửa sổ khung và cuối cùng là đối tượng ứng dụng. Việc dẫn đường dừng nếu có đối tượng nào đó trên đường xử lý thông điệp, nếu không chúng tiếp tục cho đến `::DefWindowProc`. Đường dẫn tương tự cho các thông điệp lệnh gửi tới các cửa sổ khung MDI.



Hình 7.2 Đường đi của thông điệp lệnh đến các cửa sổ khung SDI

Giá trị của đường dẫn lệnh sẽ trở nên rõ ràng khi ta xem xét ứng dụng tài liệu/quan sát quản lý các lệnh thực đơn như thế nào. Theo qui định, các lệnh *File New*, *File Open*, và *File Exit* được ánh xạ đến đối tượng ứng dụng khi `CWinApp` cung cấp các hàm thành phần `OnFileNew`, `OnFileOpen`, và `OnAppExit` để quản lý chúng. Còn *File Save* và *File Save As*, ngược lại, được ánh xạ đến đối tượng tài liệu, chúng cung cấp các cài đặt mặc định cho cả hai lệnh dưới dạng `CDocument::OnFileSave` và `CDocument::OnFileSaveAs`. Các lệnh để ẩn và hiện thanh công cụ và thanh trạng thái được cửa sổ khung quản lý nhờ các hàm thành phần của `CFrameWnd`, và phần lớn các lệnh khác được lớp quan sát quản lý.

Điểm quan trọng phải ghi nhớ khi xem xét nơi đặt các hàm điều khiển thông điệp, đó là nơi chỉ các thông điệp lệnh là chủ thể dẫn đường. Các thông điệp chuẩn của Windows như `WM_CHAR`, `WM_LBUTTONDOWN`, `WM_CREATE`, và `WM_SIZE` phải được đối tượng mà cửa sổ của chúng

nhận thông điệp quản lý. Các thông điệp chuột, bàn phím thường đi đến quan sát, trong khi phần lớn các thông điệp khác đi đến cửa sổ khung. Các đối tượng tài liệu và ứng dụng không bao giờ nhận thông điệp không phải là lệnh bởi vì chúng không là đối tượng CWnd.

7.10 CHỈ DANH CỦA LỆNH VÀ CÀI ĐẶT LỆNH MẶC ĐỊNH

Khi viết ứng dụng tài liệu/quan sát, thông thường không phải tự viết toàn bộ các điều khiển cho các lệnh thực đơn. CWinApp, CDocument, CFrameWnd, và các lớp khác của MFC cung cấp các cài đặt mặc định cho các lệnh thực đơn dùng chung như *File Open* và *File Save*. Thêm nữa, khung ứng dụng cung cấp một dải các ID của các mục lệnh thực đơn như ID_FILE_OPEN và ID_FILE_SAVE, rất nhiều trong chúng đã gắn trước vào ánh xạ thông điệp của lớp cung cấp các cài đặt lệnh mặc định.

Bảng 7.3 liệt kê nhóm chính của các ID lệnh được định nghĩa trước (xem tệp AFXRES.H của MFC). Cho mỗi ID, bảng cũng cho biết mục thực đơn tương ứng, hàm MFC (nếu có) cung cấp cài đặt mặc định, và nếu có cài đặt mặc định, thì cho biết chương trình quản lý có được gọi tự động hay không (*Prewired=Yes*) hay ID lệnh phải được gắn vào chương trình điều khiển thông qua ánh xạ thông điệp (*Prewired=No*). Thí dụ, các cài đặt mặc định cho các lệnh *File New* và *File Open* do các hàm OnFileNew và OnFileOpen của CWinApp cung cấp, nhưng chúng không nối với ứng dụng trừ khi ta cung cấp các đầu vào ánh xạ thông điệp ON_COMMAND cho chúng. (Nếu ta cho AppWizard sinh xương sống của ứng dụng SDI hay MDI, chúng sẽ viết các đầu vào ON_COMMAND cho OnFileNew, OnFileOpen và các chương trình điều khiển lệnh mặc định cho ta). Mặt khác, hàm CWinApp::OnAppExit, không cần đầu vào ánh xạ thông điệp. Việc ta cần phải làm chỉ là gắn ID ID_APP_EXIT cho lệnh mục thực đơn *File Exit*, và *File Exit* sẽ tự động gọi CWinApp::OnAppExit để kết thúc ứng dụng.

Bảng 7.3 ID lệnh định nghĩa trước và các hàm điều khiển lệnh

ID lệnh	Tên mục	Cài đặt mặc định	Prewired
Thực đơn Tệp (File menu)			
ID_FILE_NEW	Tạo tệp mới (New)	CWinApp::OnFileNew	No
ID_FILE_OPEN	Mở tệp (Open)	CWinApp::OnFileOpen	No
ID_FILE_CLOSE	Đóng tệp (Close)	CDocument::OnFileClose	Yes
ID lệnh	Tên mục	Cài đặt mặc định	Prewired
ID_FILE_SAVE	Ghi tệp (Save)	CDocument::OnFileSave	Yes
ID_FILE_SAVE_AS	Ghi tệp với tên khác (Save As)	CDocument::OnFileSaveAs	Yes
ID_FILE_PAGE_SETUP	Đặt cấu hình trang in (Page Setup)	None	N/A
ID_FILE_PRINT_SETUP	Đặt cấu hình máy in (Print Setup)	CWinApp::OnFilePrintSetup	No
ID_FILE_PRINT	In (Print)	CView::OnFilePrint	No

ID_FILE_PRINT_PREVIEW	Xem trước trang in (Print Preview)	CView::OnFilePrintPreview	No
ID_FILE_SEND_MAIL	Gửi thư (Send Mail)	CDocument::OnFileSendMail	No
ID_FILE_MRU_FILE1-16	N/A	CWinApp::OnOpenRecentFile	Yes
ID_APP_EXIT	Kết thúc (Exit)	CWinApp::OnAppExit	Yes
Thực đơn Soạn thảo (Edit menu)			
ID_EDIT_CLEAR	Xoá (Clear)	None*	N/A
ID_EDIT_CLEAR_ALL	Xoá tất cả (Clear All)	None	N/A
ID_EDIT_COPY	Sao chép (Copy)	None*	N/A
ID_EDIT_CUT	Cắt (Cut)	None*	N/A
ID_EDIT_PASTE	Dán (Paste)	None*	N/A
ID_EDIT_PASTE_LINK	Dán liên kết (Paste Link)	None	N/A
ID_EDIT_PASTE_SPECIAL	Dán đặc biệt (Paste Special)	None	N/A
ID_EDIT_FIND	Tìm kiếm (Find)	None*	N/A
ID_EDIT_REPLACE	Thay thế (Replace)	None*	N/A
ID_EDIT_REPEAT	Lặp lại (Repeat)	None*	N/A
ID_EDIT_SELECT_ALL	Chọn tất (Select All)	None*	N/A
ID_EDIT_UNDO	Hủy hành động (Undo)	None*	N/A
ID_EDIT_REDO	Làm lại (Redo)	None	N/A
Thực đơn Quan sát (View menu)			
ID_VIEW_TOOLBAR	Thanh công cụ (Toolbar)	CFrameWnd::OnBarCheck	Yes
ID_VIEW_STATUS_BAR	Thanh trạng thái (Status Bar)	CFrameWnd::OnBarCheck	Yes
Thực đơn Cửa sổ (Window menu)			
ID_WINDOW_NEW	Mở cửa sổ mới (New Window)	CMDIFrameWnd::OnWindowNew	Yes
ID_WINDOW_ARRANGE	Sắp xếp biểu tượng (Arrange Icons)	CMDIFrameWnd::OnMDIWindowCmd	Yes
ID_WINDOW_CASCADE	Sắp xếp lại các cửa sổ theo hình thác (Cascade)	CMDIFrameWnd::OnMDIWindowCmd	Yes
ID_WINDOW_TILE_HORZ	Sắp xếp lại các cửa sổ theo hàng ngang (Tile Horizontal)	CMDIFrameWnd::OnMDIWindowCmd	Yes
ID_WINDOW_TILE_VERT	Sắp xếp lại các cửa sổ theo hàng dọc (Tile Vertical)	CMDIFrameWnd::OnMDIWindowCmd	Yes
ID_WINDOW_SPLIT	Chia cửa sổ (Split)	CView::OnSplitCmd	Yes
Thực đơn Trợ giúp (Help menu)			
ID_APP_ABOUT	Giới thiệu (About AppName)	None	N/A

* **CEditView** và **CRichEditView** cung cấp cài đặt mặc định và các đầu vào ánh xạ cho các lệnh này.

Một vài lệnh trên bảng 7.3 còn được khung ứng dụng của MFC cung cấp hàm điều khiển cập nhật **ON_UPDATE_COMMAND_UI**. Lần nữa, *AppWizard* giúp viết các đầu vào ánh xạ thông điệp cần thiết cho ta.

Ta không phải sử dụng ID lệnh định nghĩa trong AFXRES.H hay hàm điều khiển lệnh mặc định do khung ứng dụng cung cấp. Ta có thể định nghĩa lấy ID lệnh, cung cấp các đầu vào ánh xạ thông điệp để dán chúng vào hàm điều khiển mặc định, hay thậm chí có thể thay hàm điều khiển lệnh mặc định bằng hàm điều khiển của ta. Nếu ta viết lấy ID lệnh và muốn hiển thị trợ giúp ngữ cảnh trong thanh trạng thái thì phải thêm bảng xâu vào tệp RC của ứng dụng với các xâu trợ giúp cho các ID lệnh mới. Bảng xâu mặc định thêm vào ứng dụng khi gộp AFXRES.H chứa các xâu trợ giúp cho các ID lệnh định nghĩa trước. Tóm lại, ta có thể sử dụng ít hay nhiều trợ giúp của khung ứng dụng tùy ý muốn. Nhưng càng biết nhiều về khung ứng dụng thì càng ít phải viết mã trình.

7.11 THÍ DỤ VỀ MỘT ỨNG DỤNG TÀI LIỆU/QUAN SÁT

Chương trình 7.1 chứa mã nguồn của ứng dụng tài liệu/quan sát SDI với tên *Life*, chúng khác chút ít với các chương trình giới thiệu về tài liệu/quan sát khác. *Life* mô phỏng trò chơi *Life* do tạp chí *Scientific American* đưa ra từ 10/1970. Trò chơi mô phỏng việc sinh ra, cuộc sống, chết và phát sinh lại các tế bào trong vũ trụ bao gồm mảng hai chiều các hình vuông (xem hình 7.3). Ta sẽ vẽ mẫu tế bào trong lưới, sau đó mở rộng các mẫu này bằng cách nhấn liên tục phím F10 hay chọn *Bước đi* từ thực đơn *Lựa chọn*. Mỗi “bước” sẽ mở rộng lưới thêm một thế hệ theo tập các qui luật đơn giản sau:

- Một tế bào sống (màu xanh) với chính xác hai hay ba hàng xóm sống qua được thế hệ tiếp theo.
- Một tế bào rỗng với chính xác hai hàng xóm sẽ đến sống trong thế hệ tiếp theo.
- Tế bào sống với ít hơn hai hay nhiều hơn ba hàng xóm chết trong thế hệ tiếp theo.

Hàng xóm là tế bào bất kỳ nào đó có thể chạm tới tế bào cho trước theo chiều ngang, thẳng đứng hay chéo. Cách thức tính hàng xóm ở cạnh lưới phụ thuộc vào cái gì sẽ được chọn trong thực đơn *Lựa chọn*. *Lựa chọn Đường biên mở* cho phép các tế bào có khoảng trống không giới hạn để phát triển. Trong lưới mở, các tế bào ở cột trái nhất của lưới là hàng xóm của các tế bào cột phải nhất, và các tế bào trên hàng đỉnh là hàng xóm của các tế bào hàng đáy. *Lựa chọn Đường biên đóng* sẽ đóng lưới sao cho tế bào đặt theo cạnh có cực đại năm hàng xóm và tế bào trong góc có cực đại ba.

Chương trình 7.1 Life

Tệp RESOURCE.H

```
#define ID_OPTIONS_STEP          100
#define ID_OPTIONS_OPEN_BOUNDARY 101
#define ID_OPTIONS_CLOSED_BOUNDARY 102
#define IDC_ICONRECT            200
```

```
#define IDC_HORZ          210
#define IDC_VERT          211
#define IDR_MAINFRAME     300
#define CELLWIDTH         16
#define CELLHEIGHT        16
//-----
```

Tệp RESRC1.H

```
#define _APS_NEXT_RESOURCE_VALUE  101
#define _APS_NEXT_COMMAND_VALUE  40001
#define _APS_NEXT_CONTROL_VALUE   1000
#define _APS_NEXT_SYMED_VALUE     101
//-----
```

Tệp LIFE.RC

```
#include "resrc1.h"
#include "afxres.h"
#include "resource.h"
// biểu tượng
IDR_MAINFRAME      ICON    DISCARDABLE    "Life.ico"
// ảnh bitmap
GRAYCELL           BITMAP MOVEABLE PURE   "GrayCell.bmp"
BLUECELL           BITMAP MOVEABLE PURE   "BlueCell.bmp"
// Thực đơn
IDR_MAINFRAME MENU DISCARDABLE
BEGIN
    POPUP "&Tệp"
    BEGIN
        MENUITEM "&Tệp mới\Ctrl+N",      ID_FILE_NEW
        MENUITEM "&Mở tệp...\Ctrl+O",    ID_FILE_OPEN
        MENUITEM SEPARATOR
        MENUITEM "&Ghi tệp\Ctrl+S",      ID_FILE_SAVE
        MENUITEM "Ghi tệp với tên khác...\Ctrl+A", ID_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "Tệp mới nhất", .      ID_FILE_MRU_FILE1
        MENUITEM SEPARATOR
        MENUITEM "&Kết thúc",            ID_APP_EXIT
    END
    POPUP "&Lựa chọn"
    BEGIN
```

```

MENUITEM "Bước đi\F10",          ID_OPTIONS_STEP
MENUITEM SEPARATOR
MENUITEM "&Đường biên mở",      ID_OPTIONS_OPEN_BOUNDARY
MENUITEM "Đường biên đóng",     ID_OPTIONS_CLOSED_BOUNDARY
END
POPUP "&Quan sát"
BEGIN
    MENUITEM "&Thanh trạng thái", ID_VIEW_STATUS_BAR
END
POPUP "Trợ &giúp"
BEGIN
    MENUITEM "&Giới thiệu Life...", ID_APP_ABOUT
END
END
// Tăng tốc
IDR_MAINFRAME ACCELERATORS MOVEABLE PURE
BEGIN
    "N",      ID_FILE_NEW,      VIRTKEY, CONTROL
    "O",      ID_FILE_OPEN,     VIRTKEY, CONTROL
    "S",      ID_FILE_SAVE,     VIRTKEY, CONTROL
    "A",      ID_FILE_SAVE_AS,  VIRTKEY, CONTROL
    VK_F10,   ID_OPTIONS_STEP,  VIRTKEY
END
// Hộp thoại
GRIDSIZE DIALOG DISCARDABLE 0, 0, 148, 60
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Kích thước lưới"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT      "Độ &rộng",-1,12,16,40,8
    LTEXT      "Độ &cao",-1,12,36,40,8
    EDITTEXT   IDC_HORZ,56,12,32,12,ES_AUTOHSCROLL
    EDITTEXT   IDC_VERT,56,32,32,12,ES_AUTOHSCROLL
    DEFPUSHBUTTON "OK",IDOK,100,12,40,12,WS_GROUP
END
ABOUT DIALOG DISCARDABLE 0, 0, 256, 98
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Giới thiệu Life"
FONT 8, "MS Sans Serif"

```



```

BEGIN
    LTEXT      "", IDC_ICONRECT, 14, 12, 80, 74
    LTEXT      "Life version 1.0", -1, 108, 12, 136, 8
    LTEXT      "" "Programming Windows with MFC" "", -1, 108, 36, 136, 8
    DEFPUSHBUTTON "OK", IDOK, 108, 72, 50, 14
    LTEXT      "(GIS Department - VN/IOIT - July 1999)", -1, 108, 56, 136, 8
END
// Bảng xâu ký tự
STRINGTABLE DISCARDABLE
BEGIN
    IDR_MAINFRAME      "Life\n\n\nLife Grids (*.lif)\n.lif\nLife.Grid\nLife Grid"
END
STRINGTABLE DISCARDABLE
BEGIN
    * AFX_IDS_IDLEMESSAGE  "Sẵn sàng"
END
STRINGTABLE DISCARDABLE
BEGIN
    ID_OPTIONS_STEP      "Phát sinh mới theo suy luận"
    ID_OPTIONS_OPEN_BOUNDARY "Cho phép bao bọc xung quanh đường biên"
    ID_OPTIONS_CLOSED_BOUNDARY "Không cho phép bao bọc xung quanh đường biên"
END
STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_NEW          "Tệp mới bản vẽ"
    ID_FILE_OPEN         "Mở tệp bản vẽ"
    ID_FILE_SAVE         "Ghi bản vẽ vào tệp"
    ID_FILE_SAVE_AS      "Ghi bản vẽ với tên khác"
END
STRINGTABLE DISCARDABLE
BEGIN
    ID_FILE_MRU_FILE1    " "
END
STRINGTABLE DISCARDABLE
BEGIN
    ID_APP_ABOUT         "Giới thiệu chương trình"
    ID_APP_EXIT          "Kết thúc chương trình"
END
STRINGTABLE DISCARDABLE

```

```
BEGIN
    ID_VIEW_STATUS_BAR    "Hiển thị/không hiển thị thanh trạng thái"
END
//-----
```

Tệp LIFE.H

```
class CLifeApp : public CWinApp
{
public:
    virtual BOOL InitInstance ();
protected:
    afx_msg void OnAppAbout ();
    DECLARE_MESSAGE_MAP ()
};
//-----
```

Tệp LIFE.CPP

```
#include <afxwin.h>
#include <afxext.h>
#include "Resource.h"
#include "Life.h"
#include "LifeDoc.h"
#include "MainFrame.h"
#include "LifeView.h"
#include "AboutDlg.h"
CLifeApp myApp;
BEGIN_MESSAGE_MAP (CLifeApp, CWinApp)
    ON_COMMAND (ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND (ID_FILE_OPEN, CWinApp::OnFileOpen)
    ON_COMMAND (ID_APP_ABOUT, OnAppAbout)
END_MESSAGE_MAP ()
BOOL CLifeApp::InitInstance ()
{
    SetRegistryKey ("GIS Dept IOIT");
    LoadStdProfileSettings ();
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate (
        IDR_MAINFRAME,          // chỉ danh tài nguyên .
        RUNTIME_CLASS (CLifeDoc), // lớp tài liệu
        RUNTIME_CLASS (CMainFrame), // lớp cửa sổ khung .

```

```

    RUNTIME_CLASS (CLifeView)    // lớp quan sát
};
AddDocTemplate (pDocTemplate);
EnableShellOpen ();
RegisterShellFileTypes ();
CCommandLineInfo cmdInfo;
ParseCommandLine (cmdInfo);
if (!ProcessShellCommand (cmdInfo))    return FALSE;
m_pMainWnd->DragAcceptFiles ();
return TRUE;
}
void CLifeApp::OnAppAbout ()
{
    CAboutDialog dlg;
    dlg.DoModal ();
}
//-----

```

Tệp MAINFRM.H

```

class CMainFrame : public CFrameWnd
{
    DECLARE_DYNCREATE (CMainFrame)
private:
    CStatusBar m_wndStatusBar;
protected:
    afx_msg int OnCreate (LPCREATESTRUCT);
    DECLARE_MESSAGE_MAP ()
};
//-----

```

Tệp MAINFRM.CPP

```

#include <afxwin.h>
#include <afxext.h>
#include "Resource.h"
#include "MainFrame.h"
IMPLEMENT_DYNCREATE (CMainFrame, CFrameWnd)
BEGIN_MESSAGE_MAP (CMainFrame, CFrameWnd)
    ON_WM_CREATE ()
END_MESSAGE_MAP ()
int CMainFrame::OnCreate (LPCREATESTRUCT lpcs)

```

```

{
    if (CFrameWnd::OnCreate (lpcs) == -1)    return -1;
    m_wndStatusBar.Create (this);
    UINT nIndicator = ID_SEPARATOR;
    m_wndStatusBar.SetIndicators (&nIndicator, 1);
    return 0;
}

```

//-----

Tệp LIFEDOC.H

```

class CLifeDoc : public CDocument
{
    DECLARE_DYNCREATE (CLifeDoc)
private:
    int      m_cx;
    int      m_cy;
    BOOL     m_bOpenEdge;
    CByteArray m_byGrid;
    CByteArray m_byBuffer;
    void     SetBackgroundCell (int, int, BOOL);
public:
    CLifeDoc ();
    virtual BOOL OnNewDocument ();
    virtual void DeleteContents ();
    virtual void Serialize (CArchive&);
    CSize     GetGridSize ();
    BOOL     GetCell (int, int);
    BOOL     ToggleCell (int, int);
    int      Evolve ();
protected:
    afx_msg void OnOptionsStep ();
    afx_msg void OnOptionsOpenBoundary ();
    afx_msg void OnOptionsClosedBoundary ();
    afx_msg void OnUpdateOptionsOpenBoundaryUI (CCmdUI*);
    afx_msg void OnUpdateOptionsClosedBoundaryUI (CCmdUI*);
    DECLARE_MESSAGE_MAP ()
};
//-----

```

Tệp LIFEDOC.CPP

```

#include <afxwin.h>
#include "Resource.h"
#include "LifeDoc.h"
#include "SizeDlg.h"
IMPLEMENT_DYNCREATE (CLifeDoc, CDocument)
BEGIN_MESSAGE_MAP (CLifeDoc, CDocument)
    ON_COMMAND (ID_OPTIONS_STEP, OnOptionsStep)
    ON_COMMAND (ID_OPTIONS_OPEN_BOUNDARY, OnOptionsOpenBoundary)
    ON_COMMAND (ID_OPTIONS_CLOSED_BOUNDARY, OnOptionsClosedBoundary)
    ON_UPDATE_COMMAND_UI (ID_OPTIONS_OPEN_BOUNDARY,
                          OnUpdateOptionsOpenBoundaryUI)
    ON_UPDATE_COMMAND_UI (ID_OPTIONS_CLOSED_BOUNDARY,
                          OnUpdateOptionsClosedBoundaryUI)
END_MESSAGE_MAP ()
CLifeDoc::CLifeDoc ()
{
    m_cx = -1;
    m_cy = -1;
}
BOOL CLifeDoc::OnNewDocument ()
{
    if (!CDocument::OnNewDocument ()) return FALSE;
    if ((m_cx == -1) && (m_cy == -1)) {
        m_cx = 24;
        m_cy = 24;
    }
    else { // dấu nhắc kích thước lưới
        CGridSizeDialog dlg;
        dlg.m_cx = m_cx;
        dlg.m_cy = m_cy;
        dlg.DoModal ();
        m_cx = dlg.m_cx;
        m_cy = dlg.m_cy;
    }
    int nByteCount = ((m_cx * m_cy) + 7) / 8;
    try { // cấp phát bộ nhớ cho lưới
        m_byGrid.SetSize (nByteCount);
        m_byBuffer.SetSize (nByteCount);
    }
}

```

```

catch (CMemoryException* e) {
    e->Delete ();
    return FALSE;
}
for (int i=0; i<nByteCount; i++) {
    m_byGrid[i] = 0;
    m_byBuffer[i] = 0;
}
return TRUE;
}

void CLifeDoc::DeleteContents ()
{
    m_byGrid.RemoveAll ();
    m_byBuffer.RemoveAll ();
}

void CLifeDoc::Serialize (CArchive& archive)
{
    if (archive.IsStoring ())    archive << (WORD) m_cx << (WORD) m_cy;
    else {
        WORD cx, cy;
        archive >> cx >> cy;
        m_cx = (int) cx;
        m_cy = (int) cy;
    }
    m_byGrid.Serialize (archive);
    m_byBuffer.SetSize (m_byGrid.GetSize ());
}

void CLifeDoc::OnOptionsStep ()
{
    Evolve ();
}

void CLifeDoc::OnOptionsOpenBoundary ()
{
    m_bOpenEdge = TRUE;
}

void CLifeDoc::OnOptionsClosedBoundary ()
{
    m_bOpenEdge = FALSE;
}

```

```

void CLifeDoc::OnUpdateOptionsOpenBoundaryUI (CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck (m_bOpenEdge ? 1 : 0);
}

void CLifeDoc::OnUpdateOptionsClosedBoundaryUI (CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck (m_bOpenEdge ? 0 : 1);
}

CSize CLifeDoc::GetGridSize ()
{
    return CSize (m_cx, m_cy);
}

BOOL CLifeDoc::GetCell (int nRow, int nCol)
{
    if ((nRow < 0) || (nRow >= m_cy) || (nCol < 0) || (nCol >= m_cx))
        return FALSE;
    int nBitNumber = ((nRow * m_cx) + nCol) % 8;
    int nByteNumber = ((nRow * m_cx) + nCol) / 8;
    return m_byGrid[nByteNumber] & (1 << nBitNumber);
}

BOOL CLifeDoc::ToggleCell (int nRow, int nCol)
{
    if ((nRow < 0) || (nRow >= m_cy) || (nCol < 0) || (nCol >= m_cx))
        return FALSE;
    int nBitNumber = ((nRow * m_cx) + nCol) % 8;
    int nByteNumber = ((nRow * m_cx) + nCol) / 8;
    m_byGrid[nByteNumber] ^= (1 << nBitNumber);
    SetModifiedFlag ();
    return m_byGrid[nByteNumber] & (1 << nBitNumber);
}

int CLifeDoc::Evolve ()
{
    int nPrevRow, nPrevCol;
    int nNextRow, nNextCol;
    int nNeighbors, i, j;
    int nLiveCells = 0;
    BOOL bAlive;
    for (i=0; i<m_cy; i++) { // hàng
        if (m_bOpenEdge) {

```

```

    nPrevRow = (i == 0) ? m_cy - 1 : i - 1;
    nNextRow = (i == (m_cy - 1)) ? 0 : i + 1;
}
else {
    nPrevRow = (i == 0) ? -1 : i - 1;
    nNextRow = (i == (m_cy - 1)) ? -1 : i + 1;
}
for (j=0; j<m_cx; j++) { // cột
    if (m_bOpenEdge) {
        nPrevCol = (j == 0) ? m_cx - 1 : j - 1;
        nNextCol = (j == (m_cx - 1)) ? 0 : j + 1;
    }
    else {
        nPrevCol = (j == 0) ? -1 : j - 1;
        nNextCol = (j == (m_cx - 1)) ? -1 : j + 1;
    }
    nNeighbors = 0;
    if (GetCell (nPrevRow, nPrevCol)) nNeighbors++;
    if (GetCell (nPrevRow, j)) nNeighbors++;
    if (GetCell (nPrevRow, nNextCol)) nNeighbors++;
    if (GetCell (i, nPrevCol)) nNeighbors++;
    if (GetCell (i, nNextCol)) nNeighbors++;
    if (GetCell (nNextRow, nPrevCol)) nNeighbors++;
    if (GetCell (nNextRow, j)) nNeighbors++;
    if (GetCell (nNextRow, nNextCol)) nNeighbors++;
    bAlive = GetCell (i, j);
    if ((bAlive && ((nNeighbors == 2) || (nNeighbors == 3))) || (!bAlive && (nNeighbors == 3))) {
        SetBackgroundCell (i, j, TRUE);
        nLiveCells++;
    }
    else SetBackgroundCell (i, j, FALSE);
}
}

int nByteCount = ((m_cx * m_cy) + 7) / 8;
for (i=0; i<nByteCount; i++)
    m_byGrid[i] = m_byBuffer[i];
SetModifiedFlag ();
UpdateAllViews (NULL, 1);
return nLiveCells;

```



```

}
void CLifeDoc::SetBackgroundCell (int nRow, int nCol, BOOL bState)
{
    int nBitNumber = ((nRow * m_cx) + nCol) % 8;
    int nByteNumber = ((nRow * m_cx) + nCol) / 8;
    if (bState)    m_byBuffer[nByteNumber] |= (1 << nBitNumber);
    else    m_byBuffer[nByteNumber] &= ~(1 << nBitNumber);
}

```

//-----

Tệp LIFEVIEW.H

```

class CLifeView : public CScrollView
{
    DECLARE_DYNCREATE (CLifeView)
private:
    int    m_nPrevRow;
    int    m_nPrevCol;
    CBitmap m_bmGrayCell;
    CBitmap m_bmBlueCell;
    CPoint m_ptOrg;
    void DrawCell (CDC*, int, int, BOOL);
    void SetGridOrigin (int, int);
    void SetScrollParms ();
public:
    CLifeView ();
    CLifeDoc* GetDocument ();
    virtual void OnInitialUpdate ();
protected:
    virtual void OnDraw (CDC*);
    virtual void OnUpdate (CView*, LPARAM, COobject*);
    afx_msg void OnSize (UINT, int, int);
    afx_msg BOOL OnEraseBkgn (CDC*);
    afx_msg void OnLButtonDown (UINT, CPoint);
    afx_msg void OnMouseMove (UINT, CPoint);
    DECLARE_MESSAGE_MAP ()
};
//-----

```

Tệp LIFEVIEW.CPP

```
#include <afxwin.h>
```

```

#include <afxext.h>
#include "Resource.h"
#include "MainFrame.h"
#include "LifeDoc.h"
#include "LifeView.h"
IMPLEMENT_DYNCREATE (CLifeView, CScrollView)
BEGIN_MESSAGE_MAP (CLifeView, CScrollView)
    ON_WM_SIZE ()
    ON_WM_ERASEBKGD ()
    ON_WM_LBUTTONDOWN ()
    ON_WM_MOUSEMOVE ()
END_MESSAGE_MAP ()
CLifeView::CLifeView ()
{
    m_bmGrayCell.LoadBitmap ("GrayCell");
    m_bmBlueCell.LoadBitmap ("BlueCell");
}
void CLifeView::OnInitialUpdate ()
{
    m_nPrevRow = -1;
    m_nPrevCol = -1;
    SetScrollParms ();
    CRect rect;
    GetClientRect (&rect);
    SetGridOrigin (rect.Width (), rect.Height ());
    CScrollView::OnInitialUpdate ();
}
void CLifeView::OnUpdate (CView* pSender, LPARAM lHint, CObject* pHint)
{
    Invalidate (lHint ? FALSE : TRUE);
}
void CLifeView::OnSize (UINT nType, int cx, int cy)
{
    CScrollView::OnSize (nType, cx, cy);
    SetGridOrigin (cx, cy);
}
BOOL CLifeView::OnEraseBkgnd (CDC* pDC)
{
    CRect rect;

```

```

    GetClientRect (&rect);
    CBrush brush ((COLORREF) ::GetSysColor (COLOR_APPWORKSPACE));
    pDC->FillRect (rect, &brush);
    return TRUE;
}

void CLifeView::OnDraw (CDC* pDC)
{
    CLifeDoc* pDoc = GetDocument ();
    CSize size = pDoc->GetGridSize ();
    CDC mdcGray, mdcBlue;
    mdcGray.CreateCompatibleDC (pDC);
    mdcBlue.CreateCompatibleDC (pDC);
    CBitmap* pOldGrayBitmap = mdcGray.SelectObject (&m_bmGrayCell);
    CBitmap* pOldBlueBitmap = mdcBlue.SelectObject (&m_bmBlueCell);
    CRect rect;
    rect = ((CPaintDC*) pDC)->m_ps.rcPaint;
    pDC->DPtoLP (&rect);
    int x1, y1, x2, y2;
    x1 = max (0, (rect.left - m_ptOrg.x) / CELLWIDTH);
    y1 = max (0, (rect.top - m_ptOrg.y) / CELLHEIGHT);
    x2 = min (size.cx, ((rect.right - m_ptOrg.x) / CELLWIDTH) + 1);
    y2 = min (size.cy, ((rect.bottom - m_ptOrg.y) / CELLHEIGHT) + 1);
    int i, j;
    for (i=y1; i<y2; i++)
        for (j=x1; j<x2; j++)
            pDC->BitBlt (m_ptOrg.x + (j * CELLWIDTH), m_ptOrg.y + (i * CELLHEIGHT), CELLWIDTH,
                        CELLHEIGHT, pDoc->GetCell (i, j) ? &mdcBlue : &mdcGray, 0, 0, SRCCOPY);
    mdcBlue.SelectObject (pOldBlueBitmap);
    mdcGray.SelectObject (pOldGrayBitmap);
}

void CLifeView::OnLButtonDown (UINT nFlags, CPoint point)
{
    CLifeDoc* pDoc = GetDocument ();
    CSize size = pDoc->GetGridSize ();
    if (((point.x < m_ptOrg.x) || (point.x > (m_ptOrg.x + (size.cx * CELLWIDTH)))) ||
        ((point.y < m_ptOrg.y) || (point.y > (m_ptOrg.y + (size.cy * CELLHEIGHT)))))
        return;
    point.x -= m_ptOrg.x;
    point.y -= m_ptOrg.y;

```

```

CClientDC dc (this);
OnPrepareDC (&dc);
dc.DPtoLP (&point);
int nRow = point.y / CELLHEIGHT;
int nCol = point.x / CELLWIDTH;
m_nPrevRow = nRow;
m_nPrevCol = nCol;
BOOL bState = pDoc->ToggleCell (nRow, nCol);
DrawCell (&dc, nRow, nCol, bState);
}

void CLifeView::OnMouseMove (UINT nFlags, CPoint point)
{
    if (!(nFlags & MK_LBUTTON))    return;
    CLifeDoc* pDoc = GetDocument ();
    CSize size = pDoc->GetGridSize ();
    if ((point.x < m_ptOrg.x) || (point.x > (m_ptOrg.x + (size.cx * CELLWIDTH))) ||
        (point.y < m_ptOrg.y) || (point.y > (m_ptOrg.y + (size.cy * CELLHEIGHT))))
        return;
    point.x -= m_ptOrg.x;
    point.y -= m_ptOrg.y;
    CClientDC dc (this);
    OnPrepareDC (&dc);
    dc.DPtoLP (&point);
    int nRow = point.y / CELLHEIGHT;
    int nCol = point.x / CELLWIDTH;
    if ((nRow == m_nPrevRow) && (nCol == m_nPrevCol))    return;
    m_nPrevRow = nRow;
    m_nPrevCol = nCol;
    BOOL bState = pDoc->ToggleCell (nRow, nCol);
    DrawCell (&dc, nRow, nCol, bState);
}

void CLifeView::DrawCell (CDC* pDC, int nRow, int nCol, BOOL bState)
{
    CDC mdc;
    mdc.CreateCompatibleDC (pDC);
    CBitmap* pOldBitmap = mdc.SelectObject (bState ? &m_bmBlueCell : &m_bmGrayCell);
    pDC->BitBlt (m_ptOrg.x + (nCol * CELLWIDTH), m_ptOrg.y + (nRow * CELLHEIGHT),
                CELLWIDTH, CELLHEIGHT, &mdc, 0, 0, SRCCOPY);
    mdc.SelectObject (pOldBitmap);
}

```

```

}

void CLifeView::SetGridOrigin (int cx, int cy)
{
    m_ptOrg.x = m_ptOrg.y = 0;
    CSize size = GetDocument ()->GetGridSize ();
    if (cx > (size.cx * CELLWIDTH)) m_ptOrg.x = (cx - (size.cx * CELLWIDTH)) / 2;
    if (cy > (size.cy * CELLHEIGHT)) m_ptOrg.y = (cy - (size.cy * CELLHEIGHT)) / 2;
}

void CLifeView::SetScrollParms ()
{
    CSize sizeTotal = GetDocument ()->GetGridSize ();
    sizeTotal.cx *= CELLWIDTH;
    sizeTotal.cy *= CELLHEIGHT;
    CSize sizePage (CELLWIDTH * 10, CELLHEIGHT * 10);
    CSize sizeLine (CELLWIDTH, CELLHEIGHT);
    SetScrollSizes (MM_TEXT, sizeTotal, sizePage, sizeLine);
}

CLifeDoc* CLifeView::GetDocument ()
{
    return (CLifeDoc*) m_pDocument;
}

//-----

```

Tệp SIZEDLG.H

```

class CGridSizeDialog : public CDialog
{
public:
    int m_cx;
    int m_cy;
    CGridSizeDialog (CWnd* pParentWnd = NULL) :
        CDialog ("GridSize", pParentWnd) {}
protected:
    virtual BOOL OnInitDialog ();
    virtual void OnCancel ();
    virtual void DoDataExchange (CDataExchange*);
};

//-----

```

Tệp SIZEDLG.CPP

```

#include <afxwin.h>

```

```

#include "Resource.h"
#include "SizeDlg.h"
BOOL CGridSizeDialog::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    CEdit* pEditHorz = (CEdit*) GetDlgItem (IDC_HORZ);
    CEdit* pEditVert = (CEdit*) GetDlgItem (IDC_VERT);
    pEditHorz->LimitText (4);
    pEditVert->LimitText (4);
    CenterWindow ();
    return TRUE;
}
void CGridSizeDialog::OnCancel ()
{
}
void CGridSizeDialog::DoDataExchange (CDataExchange* pDX)
{
    CDialog::DoDataExchange (pDX);
    DDX_Text (pDX, IDC_HORZ, m_cx);
    DDV_MinMaxInt (pDX, m_cx, 8, 512);
    DDX_Text (pDX, IDC_VERT, m_cy);
    DDV_MinMaxInt (pDX, m_cy, 8, 512);
}

```

//-----

Tệp ABOUTDLG.H

```

class CAboutDialog : public CDialog
{
private:
    CRect m_rect;
public:
    CAboutDialog (CWnd* pParentWnd = NULL) : CDialog ("About", pParentWnd) {}
protected:
    virtual BOOL OnInitDialog ();
    afx_msg void OnPaint ();
    DECLARE_MESSAGE_MAP ()
};

```

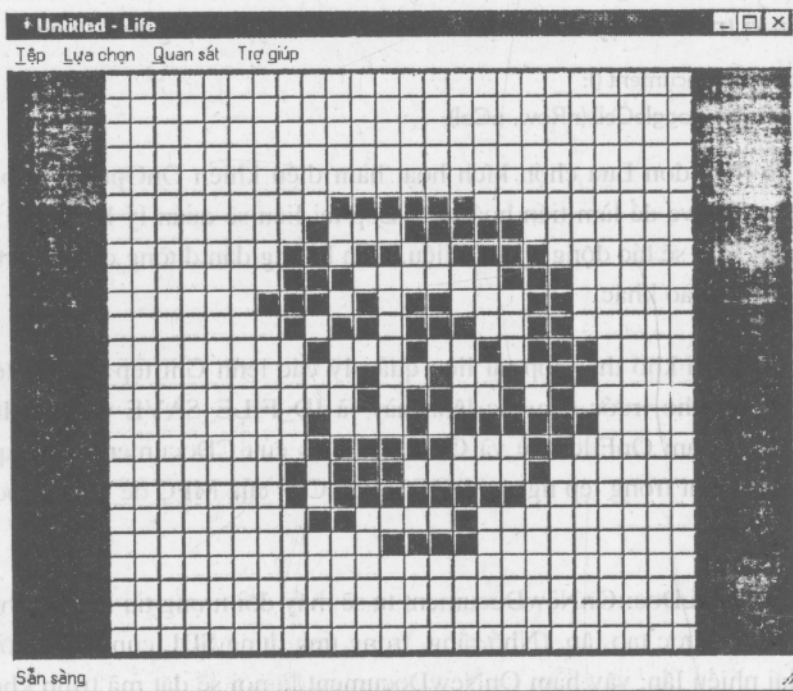
//-----

Tệp ABOUTDLG.CPP

```

#include <afxwin.h>
#include "Resource.h"
#include "AboutDlg.h"
BEGIN_MESSAGE_MAP (CAboutDialog, CDialog)
    ON_WM_PAINT ()
END_MESSAGE_MAP ()
BOOL CAboutDialog::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    CStatic* pStatic = (CStatic*) GetDlgItem (IDC_ICONRECT);
    pStatic->GetWindowRect (&m_rect);
    pStatic->DestroyWindow ();
    ScreenToClient (&m_rect);
    CenterWindow ();
    return TRUE;
}
void CAboutDialog::OnPaint ()
{
    CPaintDC dc (this);
    HICON hlcon = (HICON) ::GetClassLong (AfxGetMainWnd ()->m_hWnd, GCL_HICON);
    if (hlcon != NULL) {
        CDC mdc;
        mdc.CreateCompatibleDC (&dc);
        CBrush brush;
        brush.CreateStockObject (LTGRAY_BRUSH);
        CBitmap bitmap;
        bitmap.CreateCompatibleBitmap (&dc, 32, 32);
        CBitmap* pOldBitmap = mdc.SelectObject (&bitmap);
        CRect rclcon (0, 0, 32, 32);
        mdc.FillRect (&rclcon, &brush);
        mdc.DrawIcon (0, 0, hlcon);
        dc.StretchBlt (m_rect.left, m_rect.top, m_rect.Width(),
                      m_rect.Height (), &mdc, 0, 0, 32, 32, SRCCOPY);
        mdc.SelectObject (pOldBitmap);
    }
}
//-----

```



Hình 7.3 Kết quả thực hiện chương trình Life

Phần lớn các chức năng của chương trình *Life* là ở trong bốn lớp:

- Lớp ứng dụng *CLifeApp*, trong *LIFE.H* và *LIFE.CPP*
- Lớp cửa sổ *CMainFrame*, trong *MAINFRAME.H* và *MAINFRAME.CPP*
- Lớp tài liệu *CLifeDoc*, trong *LIFEDOC.H* và *LIFEDOC.CPP*
- Lớp quan sát *CLifeView*, trong *LIFEVIEW.H* và *LIFEVIEW.CPP*

Khi xây dựng ứng dụng *Life*, tác giả cố gắng gộp nhiều nhất có thể các đặc trưng. Tuy không có khả năng in, nhưng chương trình lại có các khả năng như nạp hoặc lưu lưới trò chơi. *Life* được thiết kế để cho biết thế nào là ứng dụng tài liệu/quan sát SDI. Người đọc có thể mở rộng thêm các chức năng khác.

Sau đây là một vài điểm thú vị khi đọc chương trình nguồn. Trước hết, lớp tài liệu *CLifeDoc* cung cấp khả năng gói dữ liệu ứng dụng bằng cách lưu các bit mô tả trạng thái on/off của từng tế bào trong đối tượng *CByteArray* với tên *m_byGrid*. *CByteArray* là thành phần dữ liệu riêng, do vậy nó không thể được xử lý từ bên ngoài lớp. Trạng thái của tế bào tại địa chỉ hàng cột cho trước có thể lấy ra bằng hàm thành phần chung *CLifeDoc::GetCell*, và tế bào có thể chuyển sang trạng thái khác nhờ hàm *CLifeDoc::ToggleCell*. Hàm *CLifeDoc::Evolve* làm tiến hoá lưới một thế hệ.

Nhấn chuột vào tế bào trong lưới sẽ kích hoạt hàm điều khiển `OnLButtonDown` của đối tượng quan sát để chuyển vị trí con chạy thành địa chỉ tế bào và chuyển trạng thái tế bào:

```
CLifeDoc* pDoc = GetDocument();
BOOL bState = pDoc->ToggleCell(nRow, nCol);
```

Chọn Bước đi từ thực đơn Lựa chọn kích hoạt hàm điều khiển `OnOptionsStep` của `CLifeDoc`, hàm này sẽ gọi hàm `Evolve` để làm tiến hoá lưới. Lớp tài liệu sẽ quản lý lệnh chứ không phải cửa sổ khung, bởi vì hàm `Step` sẽ tác động trên tài liệu. Nếu không dẫn đường cho lệnh thì hàm quản lý lệnh phải được đặt ở nơi nào khác.

Từ mã nguồn của trình khó thấy lớp tài liệu quản lý các lệnh Ghi tệp và Ghi tệp với tên khác trong thực đơn Tệp. ID cho trước của các lệnh này là `ID_FILE_SAVE` và `ID_FILE_SAVE_AS`, chúng sẽ kích hoạt các hàm `OnFileSave` và `OnFileSaveAs` của `CDocument`. Hãy quan sát ánh xạ thông điệp của `CDocument` trong tệp nguồn `DOCCORE.CPP` của MFC để thấy được việc thực hiện ánh xạ như thế nào.

Khi quan sát hàm `CLifeDoc::OnNewDocument` ta sẽ thấy đối tượng tài liệu tự khởi động lại như thế nào khi tài liệu mới được tạo lập. (Nhớ rằng, trong ứng dụng SDI, cùng một đối tượng tài liệu có thể được dựng lại nhiều lần, vậy hàm `OnNewDocument` là nơi sẽ đặt mã trình khởi động tài liệu mới). Nếu tài liệu được tạo lần đầu, kích thước của lưới mặc định là 24x24. Nếu không, người sử dụng có thể chọn kích thước lưới. Một khi kích thước lưới được đặt thì mảng `m_byGrid` và `m_byBuffer` được khởi động nhờ hàm `SetSize`, nó sẽ thuê vùng nhớ cho các mảng. (`m_byBuffer` là vùng đệm tạm thời để lưu bản sao của lưới trước khi làm tiến hoá chúng.) Hàm `CLifeDoc::DeleteContents` làm điều ngược lại bằng gọi hàm `CObArray::RemoveAll` để làm rỗng các mảng `m_byGrid` và `m_byBuffer` và giải phóng vùng nhớ đã cấp phát cho chúng. `CLifeDoc` không nạp chồng `CDocument::OnOpenDocument` bởi vì nạp tài liệu từ đĩa sẽ khởi động lại toàn bộ các thành phần dữ liệu cần thiết.

Lớp quan sát của Life có tên `CLifeView`, được suy diễn từ lớp `CScrollView`, bởi vậy quan sát có thể trượt nếu lưới quá lớn so với cả màn hình. Cấu tử của quan sát chỉ được gọi một lần, nạp hai *bitmap* để vẽ các tế bào. Hàm `OnInitialUpdate` được gọi mỗi khi tài liệu được tạo mới hay mở từ đĩa, sẽ đặt tham số trượt trên cơ sở kích thước lưới và kích thước quan sát. Các tham số trượt được tính lại mỗi khi kích thước quan sát thay đổi nhờ hàm `CLifeView::OnSize`.

Việc cài đặt hàm `OnUpdate` của `CLifeView` là không bình thường. Chúng chỉ chứa một lệnh:

```
Invalidate (fHint ? FALSE : TRUE);
```

trong đó, `fHint` là tham số do ứng dụng xác định khi một đối tượng gọi `UpdateAllViews` có thể sử dụng để chuyển thông tin đến quan sát. Mỗi khi cập nhật ban đầu cho quan sát, toàn bộ cửa sổ phải vẽ lại, kể cả nền, nhờ lệnh:

Invalidate (TRUE):

Mặt khác, nếu kích thước lưới thay đổi, lưới có thể không được vẽ lại hoàn toàn. Nếu không phải cập nhật ban đầu, thì việc vẽ lại được thực hiện bằng hàm:

Invalidate (FALSE):

để tránh nháy màn hình.

Cuối cùng là CLifeDoc::Evolve gọi hàm UpdateAllViews như sau:

UpdateAllViews (NULL, 1):

Việc gọi UpdateAllViews sẽ yêu cầu khung ứng dụng gọi OnUpdate cho từng quan sát nối với đối tượng CDocument. Vì chỉ có một quan sát trong thí dụ này, việc gọi UpdateAllViews dẫn tới gọi CLifeView::OnUpdate. Giá trị 1 chuyển cho UpdateAllViews để khởi động tham số IHint cho OnUpdate và gọi hàm Invalidate với tham số FALSE. OnUpdate chỉ được gọi một lần khác nữa khi CLifeView::OnInitialUpdate gọi cài đặt lớp cơ sở của OnInitialUpdate:

CScrollView::OnInitialUpdate ();

Khi điều này xảy ra, khung ứng dụng gọi hàm điều khiển OnUpdate với tham số IHint có giá trị 0 và CLifeView::OnUpdate gọi Invalidate với tham số TRUE để vẽ lại toàn bộ.

Khi người sử dụng chọn *Open*, *Save*, hay *Save As* từ thực đơn *File* của ứng dụng, khung ứng dụng sẽ vào/ra tài liệu hiện hành đến/từ đĩa nhờ gọi hàm CLifeDoc::Serialize. Vào/ra chỉ đòi hỏi vài dòng mã lệnh:

```
void CLifeDoc::Serialize (CArchive& archive)
{
    if (archive.IsStoring ())    archive << (WORD) m_cx << (WORD)m_cy;
    else
    {
        WORD cx, cy;
        archive >> cx >> cy;
        m_cx = (int) cx;
        m_cy = (int) cy;
    }
    m_byGrid.Serialize (archive);
    m_byBuffer.SetSize (m_byGrid.GetSize ());
}
```

Life vào/ra ba thành phần: độ rộng, độ cao của lưới, mảng m_byGrid. Khối if-else sẽ vào/ra các thành phần dữ liệu chứa kích thước lưới là m_cx và m_cy. Lệnh m_byGrid.Serialize vào/ra lưới chỉ bằng một lời gọi lệnh; chúng làm việc được bởi vì CByteArrays đủ thông minh để tự vào/ra. Lệnh cuối cùng đặt m_byBuffer cùng kích thước với m_byGrid. Kích thước của m_byGrid được đặt tự

động. Còn `m_byBuffer` thì không, cho nên kích thước của chúng phải đặt từ ngoài, nếu không sẽ không có vùng nhớ thuê cho chúng và khi thâm nhập sẽ bị phá hỏng.

7.12 TRIỂN VỌNG CỦA TÀI LIỆU/QUAN SÁT

Hiểu được mã nguồn của `Life` thì cũng sẽ hiểu được cơ chế của kiến trúc tài liệu/quan sát. Sử dụng mã nguồn trong chương trình 7.1 như là điểm bắt đầu, tương đối dễ thêm cửa sổ “bề” trên cơ sở lớp `C splitterWnd` của MFC, chuyển đổi `Life` sang ứng dụng MDI, thêm trợ giúp máy in, và các vấn đề khác.

Tóm lại, lập trình Windows không phải là dễ. Nhưng MFC làm cho việc lập trình dễ hơn nhờ dấu đi hàng ngàn chi tiết nhỏ trong khung ứng dụng. MFC hoàn hảo hơn thư viện lớp; đây là cách khác để viết ứng dụng, để giải phóng lập trình viên khỏi tập trung quá nhiều vào các dòng lệnh.

CHƯƠNG 8

IN VÀ QUAN SÁT TRƯỚC KHI IN

Chương trước mô tả kiến trúc tài liệu/quan sát làm đơn giản quá trình xây dựng các ứng dụng SDI và MDI bằng cách để cho khung ứng dụng thực hiện phần lớn các công việc của thao tác chương trình. Chương này sẽ mô tả kiến trúc tài liệu/quan sát làm đơn giản việc in và quan sát trước khi in như thế nào. Nhờ khung ứng dụng mà việc viết mã trình để chuyển tài liệu ra máy in và các thiết bị in khác trở nên đơn giản hơn.

8.1 TÀI LIỆU VÀ QUAN SÁT

Kiến trúc in của MFC được xây dựng xung quanh các hàm in API cơ bản và các thủ tục xây dựng trong GDI của Windows. Chương này sẽ:

- Trao đổi về mô hình in của Windows và xem xét các bước in tài liệu của ứng dụng xây dựng bằng công cụ phát triển phần mềm Windows (*Software Development Kit - SDK*).
- Mô tả quan hệ giữa kiến trúc in của MFC và kiến trúc in của Windows, và các cơ chế in của các ứng dụng MFC, bao gồm các hàm ảo cần nạp chồng trong các lớp của lập trình viên.
- Xây dựng chương trình xương sống mô tả việc sử dụng cùng mã trình để in và hiển thị trên màn hình.
- Xây dựng ứng dụng có khả năng in và quan sát trước khi in.

8.2 KIẾN TRÚC IN CỦA WINDOWS

Việc in tài liệu từ ứng dụng Windows mà không sử dụng khung ứng dụng sẽ phải thực hiện rất nhiều bước phức tạp. Ứng dụng có khả năng in thông thường bắt đầu bằng việc lấy ngữ cảnh thiết bị DC cho máy in. Giống như khi đưa dữ liệu ra màn hình, ta phải lấy DC màn hình, khi đưa dữ liệu ra máy in ta cũng phải lấy DC máy in. Nếu biết được tên máy in thì có thể tạo DC bằng hàm của Windows ::CreateDC hay hàm CDC::CreateDC của MFC.

```
CDC dc;  
dc.CreateDC(NULL, "HP LaserJet 4P", NULL, NULL);
```

Tên thiết bị chỉ ra trong tham số thứ hai là cùng tên xuất hiện trong tấm điều khiển của hộp thoại *Print Setup*. Cú pháp ở đây đơn giản hơn lập trình ứng dụng trên Windows 3.1, nơi yêu cầu có cả tên chương trình điều khiển máy in (*driver*) và các thông tin khác. Nếu ta không biết tên thiết bị mà vẫn muốn in ra máy in mặc định, thì có thể sử dụng hàm của MFC là

CPrintDialog::GetDefaults và CPrintDialog::GetPrinterDC.

```

CDC dc;
CPrintDialog dlg (FALSE);
dlg.GetDefaults ();
dc.Attach (dlg.GetPrinterDC ());

```

Nếu để cho người sử dụng chọn máy in, ta phải hiển thị hộp thoại *Print* (là một trong các hộp thoại chung do Windows cung cấp) nhờ hàm **CPrintDialog::DoModal**, sau đó gọi **CPrintDialog::GetPrinterDC** sau khi kết thúc công việc với hộp thoại.

```

CDC dc;
CPrintDialog dlg (FALSE);
if (dlg.DoModal () == IDOK)
    dc.Attach (dlg.GetPrinterDC ());

```

DC của máy in lấy được bằng một trong các cách trên đây phải được xoá khi không cần đến nữa. Nếu phải tạo đối tượng CDC trong ngăn xếp, thì việc xoá sẽ thực hiện tự động.

Một khi đã có DC máy in thì có thể bắt đầu in. Bước tiếp theo là gọi **::StartDoc** hay hàm tương đương của MFC là **CDC::StartDoc** để đánh dấu bắt đầu công việc in. **CDC::StartDoc** chỉ có một tham số là con trỏ đến cấu trúc **DOCINFO** lưu xâu ký tự với tên mô tả tài liệu sẽ in, tên của tệp sẽ sử dụng nếu thực hiện in ra tệp, và các thông tin khác về công việc in. Các lệnh:

```

DOCINFO di;
::ZeroMemory (&di, sizeof (DOCINFO));
di.cbSize = sizeof (DOCINFO);
di.lpszDocName = "Số tiền của năm tài chính";
dc.StartDoc (&di);

```

bắt đầu công việc in trên máy in liên quan đến đối tượng CDC. Nếu người sử dụng khảo sát hàng đợi khi tài liệu đang in, xâu "Số tiền của năm tài chính" sẽ có trong cột *Document Name*. Nếu **StartDoc** bị lỗi, chúng cho lại giá trị 0 hay nhỏ hơn 0. Nếu thực hiện tốt, chúng cho lại số nguyên dương, chỉ ra ID của công việc (*job*) in. ID này có thể được sử dụng cho các hàm điều khiển in của Win32 như **::GetJob** và **::SetJob**. Tiếp theo là gửi ra trang in. Văn bản và đồ họa được đưa ra máy in nhờ các hàm GDI. Nếu đối tượng dc là ngữ cảnh thiết bị của màn hình thì lệnh:

```
dc.Ellipse (0, 0, 100, 100);
```

sẽ vẽ elíp 100x100 (đơn vị logic) trên màn hình (ta có hình tròn). Nếu đối tượng dc là ngữ cảnh thiết bị của máy in, thì hình tròn được vẽ trên máy in. Các trang được đưa ra bằng các hàm giữa các lời gọi **CDC::StartPage** và **CDC::EndPage** để đánh dấu đầu, cuối trang. Nếu tài liệu có *nPageCount* trang thì sẽ in nó như sau:

```

dc.StartDoc (&di);
for (int i=1; i<=nPageCount; i++)

```

```

{
    dc.StartPage ();
    // In trang thứ i
    dc.EndPage ();
}

```

Việc gọi `EndPage` được hiểu đơn giản như gửi ký tự đẩy giấy *FF (form feed)* ra máy in. Giữa `StartPage` và `EndPage`, ta sẽ in các trang nhờ gọi hàm thành phần của `CDC`. Thậm chí, nếu tài liệu chỉ có một trang thì vẫn phải gọi `StartPage` và `EndPage`. Một lỗi mà lập trình viên hay gặp phải khi lần đầu viết mã trình để in là quên khởi động `DC` máy in cho mỗi trang. Trong Windows 95/98, thuộc tính mặc định của `DC` được phục hồi mỗi khi gọi `StartPage`. Ta không thể chọn co chữ hay đặt chế độ ánh xạ ngay sau khi `DC` được tạo và mong đợi các thuộc tính này sẽ có hiệu lực vô hạn như là `DC` màn hình. Thay vào đó, ta phải khởi động lại `DC` máy in cho mỗi trang in. (Trong Windows NT 3.5 và các phiên bản sau này, `DC` máy in nhớ các thông số khi gọi `StartPage` và `EndPage` – nhưng ngay cả trong ứng dụng Windows NT ta vẫn phải khởi động lại `DC` ở đầu mỗi trang in nếu muốn cùng mã trình ấy sẽ hoạt động trên Win95/98). Nếu sử dụng chế độ ánh xạ `MM_LOENGLISH` khi in, thì phải gọi `CDC::SetMapMode` ở đầu mỗi trang.

```

for (int i=1; i<=nPageCount; i++)
{
    dc.StartPage ();
    dc.SetMapMode (MM_LOENGLISH);
    // In trang i
    dc.EndPage ();
}

```

Nếu ta làm như sau:

```

// Không làm như thế này!
dc.SetMapMode (MM_LOENGLISH);
for (int i=1; i<=nPageCount; i++)
{
    dc.StartPage ();
    // In trang i
    dc.EndPage ();
}

```

thì chúng sẽ in ở chế độ ánh xạ `MM_TEXT`.

Sau khi xong trang cuối, phải kết thúc việc in bằng gọi hàm `CDC::EndDoc` – trừ khi lời gọi `EndPage` trước đó đã cho kết quả chỉ ra là `GDI` đã kết thúc công việc in. `EndPage` cho lại giá trị nguyên có dấu, nếu chúng lớn hơn 0 thì việc in là tốt. Nếu giá trị nhỏ hơn 0 hay bằng 0 có nghĩa gặp lỗi khi in hay người dùng tự xóa bỏ việc in khi trang đang in ra. Ý nghĩa của các giá trị cho lại được mô tả trong bảng 8.1.

Bảng 8.1 Mã cho lại khi in

<i>Giá trị cho lại</i>	<i>Mô tả</i>
SP_ERROR	Dừng in vì lý do không được xác định
SP_APPABORT	Dừng in do người dùng nhấn chuột vào phím Cancel
SP_USERABORT	Dừng in do người dùng thông qua shell của hệ điều hành
SP_OUTOFDISK	Đầy đĩa
SP_OUTOFMEMORY	Đầy bộ nhớ

Vòng lặp sau mô tả chính xác cách sử dụng EndDoc:

```
if (dc.StartDoc (&di) > 0)
{
    BOOL bContinue = TRUE;
    for (int i=1; i<=nPageCount && bContinue; i++)
    {
        dc.StartPage ();
        // Khởi động DC
        // In trang thứ i
        if (dc.EndPage () <= 0)
            bContinue = FALSE;
    }
    if (bContinue)
        dc.EndDoc ();
    else
        dc.AbortDoc ();
}
```

CDC::AbortDoc bãi bỏ công việc in như hàm EndDoc kết thúc công việc in khi in xong. Ta cũng có thể gọi AbortDoc vào thời điểm bất kỳ để dừng in.

8.3 THỦ TỤC Abort VÀ HỘP THOẠI Abort

Nếu tất cả đã in ra được thì không có vấn đề lớn để bàn. Nhưng còn có nhiều vấn đề khác xảy ra như việc in có thể chiếm nhiều thời gian, người dùng có thể kết thúc sớm việc in. Ứng dụng Windows thường để người sử dụng nhấn phím *Cancel* của hộp thoại để kết thúc sớm. Phím *Cancel* dừng in bằng cách yêu cầu hàm EndPage cho lại giá trị SP_APPABORT.

Cơ chế liên kết phím *Cancel* với việc in được mô tả trong ứng dụng ở hàm mà Windows gọi là thủ tục bãi bỏ in “*Abort procedure*”. Thủ tục bãi bỏ là hàm được lập trình viên xây dựng để Windows gọi nhiều lần như xử lý ra máy in. Nó được khai báo như sau:

```
BOOL CALLBACK AbortProc (HDC hDC, int nCode)
```

hDC chứa chỉ danh của ngử cảnh máy in. Đối số nCode có giá trị 0 nếu in tốt, hay giá trị SP_OUTOFDISK nếu đầy đĩa khi ghi tệp tạm thời phục vụ in. Thông thường bỏ qua nCode vì trình điều khiển in *spooler* của Windows trả lời điều kiện SP_OUTOFDISK bằng cách chờ khoảng trống của đĩa. Công việc của thủ tục bãi bỏ in bao gồm:

- Kiểm tra hàng đợi thông điệp bằng hàm `::PeekMessage` và lấy ra rồi phân phát nếu có thông điệp trong đó.
- Cho Windows biết việc in có được tiếp tục hay không bằng giá trị cho lại TRUE (tiếp tục in) hay FALSE (bãi bỏ in).

Một thủ tục bãi bỏ đơn giản có thể như sau:

```
BOOL CALLBACK AbortProc (HDC hDC, int nCode)
{
    MSG msg;
    while (::PeekMessage (&msg, NULL, 0, 0, PM_NOREMOVE))
        AfxGetThread ()->PumpMessage ();
    return TRUE;
}
```

Vòng lặp thông điệp bên trong hàm `AbortProc` cho phép thông điệp `WM_COMMAND` (sinh ra khi phím *Cancel* của hộp thoại bị nhấn) gửi đến thủ tục cửa sổ ngay cả khi ứng dụng đang bận in. Trong Windows 16 bit, vòng lặp thông điệp đóng vai trò quan trọng để thực hiện đa nhiệm sao cho *spooler* in và các tiến trình đang chạy khác đều có thể chiếm thời gian CPU. Trong Windows 95/98, thủ tục bãi bỏ làm tăng sức mạnh của đa nhiệm khi ứng dụng 32 bit sử dụng chương trình điều khiển in 16 bit.

Trước khi gọi `StartDoc`, ứng dụng gọi `SetAbortProc` để lập thủ tục bãi bỏ in, sau đó làm vô hiệu cửa sổ của chúng bằng gọi hàm `CWnd::EnableWindow(FALSE)`, và hiển thị trạng thái in hay hộp thoại “bãi bỏ”, đó là hộp thoại *modeless* chứa phím *Cancel* với một hay nhiều thông báo như hiển thị tên tài liệu, tổng số trang đã in. Việc vô hiệu cửa sổ chính đảm bảo rằng không còn ngắt nào sẽ ngắt tiến trình in. Cửa sổ sẽ được làm tích cực trở lại khi kết thúc in và hộp thoại bị phá hủy. Hộp thoại lập cờ `bUserAbort` từ FALSE sang TRUE nếu phím *Cancel* bị nhấn, và thủ tục bãi bỏ cho lại FALSE để bãi bỏ in nếu `bUserAbort` là TRUE.

```
BOOL CALLBACK AbortProc (HDC hDC, int nCode)
{
    MSG msg;
    while (!bUserAbort && ::PeekMessage (&msg, NULL, 0, 0, PM_NOREMOVE))
        AfxGetThread ()->PumpMessage ();
    return bUserAbort;
}
```


Việc in được thực hiện nếu không bị trở ngại gì (không bị người dùng nhấn *Cancel*). Khi phím *Cancel* bị nhấn thì `hUserAbort` đổi từ `FALSE` sang `TRUE`, lời gọi tiếp đến `AbortProc` cho lại giá trị 0. Windows kết thúc in. `EndPage` cho lại `SP_APPABORT` và gọi `EndDoc`.

8.4 CUỘN IN

Các vấn đề được đề cập đến thời điểm này đều thuộc “phần nổi” của tiến trình in, đó là phần mà ứng dụng có trách nhiệm. Còn Windows quản lý “phần nền”, đó là các vấn đề liên quan GDI, cuộn (*spooling*) in, trình điều khiển in (*driver*), và các thành phần khác của hệ thống in 32 bit. Nếu *metafile spool* tích cực, các lời gọi GDI được thực hiện thông qua DC máy in sẽ viết vào *metafile* trên đĩa cứng và lưu tại đó cho đến khi *spooler* của máy in lấy ra và “gửi chúng” đến chương trình điều khiển máy in. Nếu cuộn in bị vô hiệu lực thì các lệnh GDI được chuyển trực tiếp đến chương trình điều khiển in mỗi khi hàm `EndPage` được gọi. Cuộn in làm tăng tốc độ trở lại ứng dụng bằng cách để chương trình thực hiện toàn bộ việc in ra *metafile*, cho nên người dùng không cần phải chờ cho đến khi in xong mỗi trang. Hơn nữa các *metafile* làm tăng tốc độ in bằng cách tách nhiệm vụ của trình ứng dụng khỏi nhiệm vụ của trình điều khiển máy in. Đó là một lý do tại sao Windows 95/98 hình như in nhanh hơn Windows 3.1. Trong Windows 3.1, các lệnh GDI được trình điều khiển máy in xử lý trước khi chúng được “cuộn” vào đĩa. Trong Windows 95/98, đầu ra được “cuộn” trước sau đó mới đẩy ra trình điều khiển máy in sau, như vậy việc chuyển các lệnh GDI sang dữ liệu in phụ thuộc thiết bị được thực hiện hoàn toàn ở nền.

May thay, ta có thể bỏ qua những gì xảy ra ở phía sau của tiến trình in và chỉ tập trung vào in tài liệu. Vẫn còn nhiều chi tiết làm bản khoản trước khi có thể thực hiện hoàn hảo việc in, phân trang in và thực hiện các lời gọi GDI giữa `StartPage` và `EndPage` để in ra từng trang. Với hiểu biết cơ bản này, chúng ta hãy xem MFC thực hiện việc in dễ dàng hơn như thế nào.

8.5 KIẾN TRÚC IN CỦA MFC

Kiến trúc in đơn giản của MFC là một trong các lý do để lập trình viên dễ chuyển từ sử dụng SDK sang môi trường phát triển hướng đối tượng như Visual C++. Khi in từ ứng dụng tài liệu/quan sát, ta có thể quên đi rất nhiều đoạn mã trình như vừa mô tả. Khung ứng dụng sẽ sinh DC máy in cho ta và xoá chúng đi khi in xong. Khung ứng dụng còn gọi `StartDoc` và `EndDoc` để bắt đầu và kết thúc việc in, và gọi `StartPage` và `EndPage` để gộp các lời gọi GDI vào từng trang. Thậm chí chúng còn cung cấp hộp thoại *Cancel* và thủ tục bãi bỏ. Trong một số trường hợp, chúng còn cung cấp hàm rất giống `OnDraw` mà ta sử dụng để tải tài liệu trên màn hình ra máy in và xem qua trước khi in.

Chìa khoá của in trong kiến trúc tài liệu/quan sát là tập các hàm ảo của `CView` mà khung ứng dụng gọi ở các mức độ khác nhau trong quá trình in. Các hàm này được tập hợp trong bảng 8.2.

Hàm `CView` nào cần nạp chồng, và phải làm cái gì khi nạp chồng để có tính độc lập cao với nội dung muốn in ra. Ít nhất phải nạp chồng hàm `OnPreparePrinting` để gọi `DoPreparePrinting` để khung ứng dụng sẽ hiển thị hộp thoại `Print` và tạo DC in.

```
BOOL CMyView::OnPreparePrinting (CPrintInfo* pInfo)
{
    return DoPreparePrinting (pInfo);
}
```

Bảng 8.2 Nạp chồng `CView` cho in

Hàm	Mô tả
<code>CView::OnPreparePrinting</code>	Được gọi ngay khi bắt đầu in. Nạp chồng hàm <code>CView::DoPreparePrinting</code> cung cấp tổng số trang (nếu biết) cho khung ứng dụng và các thông tin khác về in.
<code>CView::OnBeginPrinting</code>	Gọi ngay trước khi in. Nạp chồng để xin cấp phát cơ chữ và các tài nguyên khác cho in.
<code>CView::OnPrepareDC</code>	Gọi trước mỗi trang in. Nạp chồng để đặt gốc của sổ hay vùng cắt (clip) để in trang hiện hành nếu <code>OnDraw</code> sẽ được sử dụng để sinh ra dữ liệu ra máy in.
<code>CView::OnPrint</code>	Gọi để in một trang tài liệu. Khung ứng dụng sẽ cung cấp tổng số trang và DC in. Nạp chồng để in <i>header</i> , <i>footer</i> , và các chi tiết khác của trang mà <code>OnDraw</code> không vẽ trên quan sát hay in trang không do <code>OnDraw</code> vẽ.
<code>CView::OnEndPrinting</code>	Gọi khi in xong. Nạp chồng để giải phóng tài nguyên được cấp phát ở hàm <code>OnBeginPrinting</code> .

Nếu `OnPreparePrinting` cho lại giá trị khác 0 thì bắt đầu tiến trình in, nếu cho lại giá trị 0 thì bãi bỏ việc in. Hàm `DoPreparePrinting` cho lại 0 nếu người dùng bãi bỏ in bằng nhấn chuột trên phím *Cancel* của hộp thoại *Print*, nếu không nối máy in hay nếu khung ứng dụng không tạo được DC in.

8.5.1 Chi tiết hơn về hàm `OnPreparePrinting`

Đối tượng `CPrintInfo` chuyển đến `OnPreparePrinting` có chứa thông tin mô tả các tham số của công việc in, bao gồm số trang cực tiểu, cực đại. Mặc định chúng có giá trị 1 và 0xFFFF, giá trị sau thông báo cho khung ứng dụng biết là không biết trước số trang cực đại. Nếu biết trước tài liệu có bao nhiêu trang định in thì gọi hàm `CPrintInfo::SetMaxPage` trước khi gọi `DoPreparePrinting`.

```
BOOL CMyView::OnPreparePrinting (CPrintInfo* pInfo)
{
    pInfo->SetMaxPage (nMaxPage);
    return DoPreparePrinting (pInfo);
}
```

Đến lượt khung ứng dụng sẽ hiển thị số trang cực đại trong hộp “to” của hộp thoại *Print*.

`SetMaxPage` là một trong số các hàm thành phần của `CPrintInfo` mà ta có thể gọi để đặt các tham số in hay tìm kiếm trong khung ứng dụng về các thông số do người dùng vào. `SetMinPage` cho phép chỉ ra số trang cực tiểu. `GetMinPage` và `GetMaxPage` cho biết tổng số trang cực tiểu và cực đại, còn `GetFromPage` và `GetToPage` cho biết số trang bắt đầu và kết thúc trong hộp “from”,

“to” của hộp thoại in. CPrintInfo còn chứa một số thành phần dữ liệu chung, bao gồm `m_ppd`, trả tới đối tượng khởi động CPrintDialog. Ta có thể sử dụng con trỏ này để thay đổi hộp thoại in trước khi hiển thị, hay lấy thông tin từ chúng. Chúng ta sẽ mô tả thí dụ về vấn đề này trong các trang sau.

8.5.2 OnBeginPrinting và OnEndPrinting

Thông thường số trang cực đại phụ thuộc vào kích thước của vùng in được của mỗi trang. Không may, ta chưa biết chúng cho đến khi người dùng chưa chọn loại máy in và khung ứng dụng chưa tạo DC in. Nếu không đặt tổng số trang cực đại trong OnPreparePrinting, ta phải đặt giá trị trong OnBeginPrinting nếu có thể. OnBeginPrinting nhận con trỏ đến cấu trúc CPrintInfo và con trỏ đến đối tượng CDC biểu diễn CD máy in. Ta có thể xác định kích thước của vùng in được của trang nhờ gọi hai lần hàm CDC::GetDeviceCaps - một lần với tham số HORZRES, lần khác với tham số VERTRES.

```
void CMyView::OnBeginPrinting (CDC* pDC, CPrintInfo* pInfo)
{
    int m_nPageHeight = pDC->GetDeviceCaps (VERTRES);
    int nDocLength = GetDocument ()->GetDocLength ();
    int nMaxPage = max (1, (nDocLength + (m_nPageHeight - 1)) / m_nPageHeight);
    pInfo->SetMaxPage (nMaxPage);
}
```

Trong thí dụ này, hàm GetDocLength là hàm tài liệu cho lại độ dài của tài liệu tính bằng *pixel*. Hàm CPrintInfo chứa thành phần dữ liệu `m_rectDraw`, mô tả vùng trang in được theo hệ tọa độ logic - nhưng đừng sử dụng chúng trong OnBeginPrinting bởi vì nó vẫn chưa được khởi động ngay trước khi gọi OnPrint.

Việc gọi hàm SetMaxPage trong OnPreparePrinting hay trong OnBeginPrinting cho khung ứng dụng biết phải gọi bao nhiêu lần OnPrint để in trang. Nếu có thể xác định độ dài tài liệu trước khi bắt đầu in, ta phải thực hiện phân trang khi in bằng nạp chồng hàm OnPrepareDC và đặt TRUE hay FALSE cho CPrintInfo::m_bContinuePrinting mỗi khi gọi OnPrepareDC. Giá trị FALSE sẽ kết thúc in. Nếu không gọi SetMaxPage, thì khung ứng dụng coi tài liệu dài một trang. Vì vậy, ta phải nạp chồng OnPrepareDC và đặt các trang sẽ in vào m_bContinuePrinting.

Hàm OnBeginPrinting là nơi tốt nhất để tạo co chữ và các tài nguyên GDI khác cần cho in. Giả sử hàm OnCreate của quan sát tạo co chữ mà OnDraw sử dụng để vẽ chữ trên màn hình, và độ cao co chữ trên cơ sở thước đo màn hình do GetDeviceCaps cho lại. Để in và có kết quả “ta thấy gì trên màn hình thì sẽ có trên máy in như vậy” (WYSIWYG), ta phải tạo co chữ độc lập theo kích thước máy in thay vì kích thước màn hình. Một giải pháp là để OnDraw tạo co chữ mỗi khi chúng được sử dụng cho loại thiết bị riêng - có CDC tương ứng - máy in hay màn hình. Giải pháp khác là tạo co chữ in trong OnBeginPrinting và xoá nó trong OnEndPrinting. Như vậy sẽ tránh việc tạo, xoá co

chữ nhiều lần mỗi khi gọi `OnDraw`.

Hàm `OnEndPrinting` là đối trọng của `OnBeginPrinting`. Đây là vị trí thuận lợi cho giải phóng cơ chế và các tài nguyên khác đã xin cấp phát trong `OnBeginPrinting`. Nếu không có tài nguyên để giải phóng, hay ta không nạp chồng hàm `OnBeginPrinting` thì cũng không cần nạp chồng hàm `OnEndPrinting`.

8.5.3 OnPrepareDC

MFC gọi hàm `OnPrepareDC` một lần cho mỗi trang của tài liệu. Một lý do để nạp chồng là thực hiện chia thời gian in trang như mô tả trong chương trước. Lý do khác là để tính tọa độ gốc vùng nhìn mới từ số trang hiện thời để `OnDraw` đẩy trang tương ứng ra máy in. Tương tự `OnBeginPrinting`, hàm `OnPrepareDC` nhận con trỏ dc và con trỏ đến đối tượng `CPrintInfo`. Không như `OnBeginPrinting`, `OnPrepareDC` được gọi trước khi màn hình vẽ lại cũng như khi chuẩn bị đầu ra một trang đến máy in. Khi ra màn hình, con trỏ CDC trở đến DC màn hình và con trỏ `CPrintInfo` có giá trị NULL. Để in, con trỏ CDC là DC máy in và con trỏ `CPrintInfo` có giá trị khác NULL. Trong trường hợp sau, số thứ tự trang sẽ in có thể lấy từ `CPrintInfo::m_nCurPage`. Ta có thể xác định hàm `OnPrepareDC` đã được gọi cho màn hình hay máy in nhờ hàm `CDC::IsPrinting`.

Cài đặt tiếp theo của `OnPrepareDC` sẽ chuyển gốc cửa quan sát theo hướng y sao cho điểm thiết bị (0,0) sẽ tương ứng với góc trên, trái của trang hiện hành. `m_nPageHeight` là thành phần dữ liệu của `CMyView`, chúng chứa độ cao của trang in được.

```
void CMyView::OnPrepareDC (CDC* pDC, CPrintInfo* pInfo)
{
    CView::OnPrepareDC (pDC, pInfo);
    if (pDC->IsPrinting ())
    {
        // Nếu đang in ...
        int y = (pInfo->m_nCurPage - 1) * m_nPageHeight;
        pDC->SetViewportOrg (0, -y);
    }
}
```

Đặt gốc cửa quan sát (*viewport*) theo cách này đảm bảo rằng hàm `OnDraw` định vẽ toàn bộ tài liệu thì chỉ vẽ một phần tương ứng với trang hiện hành. Đây là thí dụ đơn giản của `OnPrepareDC` với giả sử ta định in toàn bộ vùng in được của trang. Đôi khi cần phải đặt vùng cắt (*clip*) để giới hạn vùng in của tài liệu thành vùng nhỏ hơn trang in. Vùng hình chữ nhật được tạo nhờ hàm `CRgn::CreateRectRgn` và chọn nó cho DC để làm vùng clip nhờ hàm `CDC::SelectClipRgn`.

Theo quy tắc, ta chỉ cần nạp chồng `OnPrepareDC` để in tài liệu đa trang để `OnDraw` thực hiện in. Nếu ta thực hiện in toàn bộ nhờ `OnPrint`, thì không cần nạp chồng `OnPrepareDC` (như trong thí dụ ở đây). Khi nạp chồng `OnPrepareDC`, ta phải gọi lớp cơ sở trước khi làm bất cứ cái gì để cài đặt mặc định có cơ hội thực hiện các công việc của riêng nó. Điều này đặc biệt quan trọng khi lớp quan sát của ta được suy diễn từ `CScrollView`, bởi vì `CScrollView::OnPrepareDC` đặt gốc của quan sát cho DC màn hình để phù hợp với vị trí trượt hiện hành. Khi `CScrollView::OnPrepareDC` trở lại, chế độ ánh xạ của DC là chế độ ánh xạ được chỉ ra khi gọi `SetScrollSizes`. Nếu lớp quan sát của ta không suy diễn từ `CScrollView`, thì `OnPrepareDC` là nơi tốt nhất để gọi `SetMapMode`.

8.5.4 OnPrint

Sau khi gọi `OnPrepareDC`, nhưng trước khi in vật lý các trang, khung ứng dụng gọi `CView::OnPrint`. Cài đặt mặc định trong tệp `Viewcore.cpp` sẽ kiểm tra tính hợp lý của pDC và chuyển cho `OnDraw`.

```
void CView::OnPrint(CDC* pDC, CPrintInfo*)
{
    ASSERT_VALID(pDC);
    // Nạp chồng và đặt các biến in trên cơ sở trang in
    OnDraw(pDC); // Gọi Draw
}
```

Nếu đã chuẩn bị DC in cho việc in trong `OnPrepareDC`, hay nếu tài liệu chỉ chứa một trang, ta không cần nạp chồng `OnPrint`; ta có thể để cho cài đặt mặc định gọi `OnDraw` để in.

Thực tế, `OnPrint` thường được nạp chồng để thực hiện công việc in trang theo yêu cầu riêng. Hầu như lý do chính của nạp chồng `OnPrint` là để in *header*, *footer*, số trang và các thành phần trực quan khác chỉ thấy trên trang in chứ không thấy trên màn hình. Hàm `OnPrint` sau đây gọi hàm thành phần cục bộ `PrintHeader` và `PrintPageNumber` để in *header* và số trang trước khi gọi `OnDraw` để in trang:

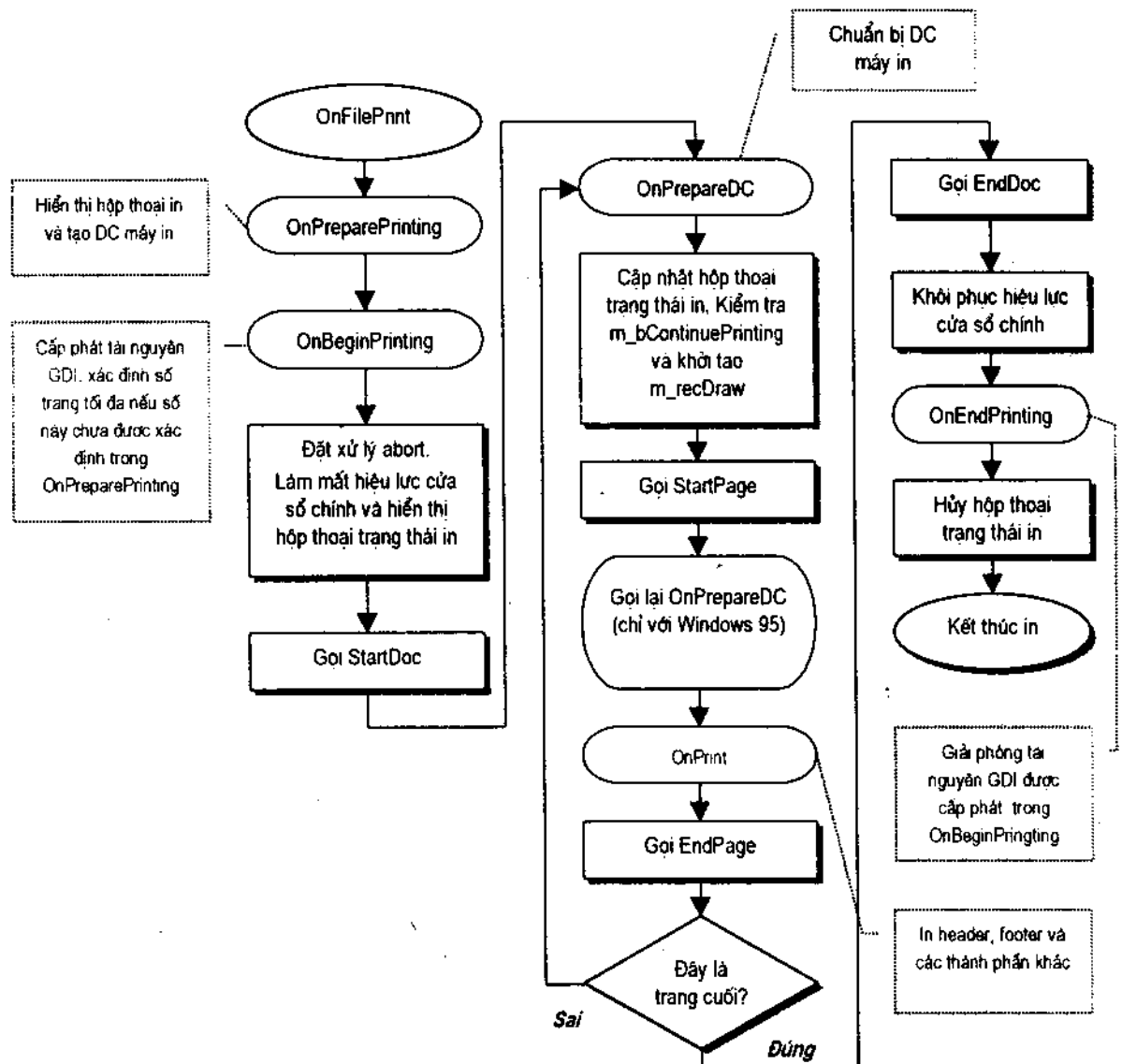
```
void CMyView::OnPrint (CDC* pDC, CPrintInfo* pInfo)
{
    PrintHeader (pDC);
    PrintPageNumber (pDC, pInfo->m_nCurPage);
    // Đặt gốc cổng quan sát và/hoặc vùng cắt trước khi gọi OnDraw...
    OnDraw (pDC);
}
```

Chú ý rằng bất cứ thay đổi nào của DC in bởi hàm `SetViewportOrg` hay `SelectClipRgn` phải làm trong `OnPrint` hơn là ở `OnPrepareDC` để *header* và số trang không bị tác động. Cách tiếp cận khác là thực hiện trước lời gọi `OnDraw` và thực hiện mọi việc in tại `OnPrint`. Nếu mã trình của vẽ và in

khác nhau, thì việc thực hiện in trong OnPrint sẽ chỉ ra ranh giới rõ ràng hơn giữa in tài liệu như thế nào và hiển thị ra màn hình như thế nào.

8.5.5 CView::OnFilePrint và các hàm điều khiển lệnh khác

Công việc in thường bắt đầu khi người dùng chọn lệnh *Print* từ thực đơn *File*. MFC cung cấp CView::OnFilePrint để ta có thể nối mục thực đơn ID_FILE_PRINT tới ánh xạ thông điệp quan sát.



Hình 8.1 Tổng quan về kiến trúc in của MFC

Hình 8.1 mô tả những gì xảy ra khi người dùng thực hiện in. Nó cũng chỉ ra kiến trúc in của MFC khớp với kiến trúc in của Windows như thế nào, bởi vì nếu bỏ đi chữ nhật biểu diễn các hàm CView ảo mà khung ứng dụng gọi, thì chỉ còn lại mô hình in của Windows. Chú ý rằng OnPrepareDC được gọi hai lần cho mỗi trang khi mã trình thực hiện dưới Windows 95/98. Lần gọi đầu để làm tương thích bản 16 bit của MFC, trong đó gọi OnPrepareDC trước StartPage (trong Windows 16 bit, chính EndPage, chứ không phải StartPage sẽ xoá dc). Lệnh gọi thứ hai được thực hiện, bởi vì trong Windows 95/98, thay đổi dc ở lần gọi thứ nhất bị vô hiệu khi StartDoc được gọi.

MFC còn định nghĩa trước chỉ danh ID lệnh và các hàm điều khiển lệnh mặc định cho các lệnh *Print Preview* và *Print Setup* của thực đơn *File*. Lệnh *Print Preview* (ID=ID_FILE_PRINT_PREVIEW) do CView::OnFilePrintPreview quản lý, và *Print Setup* (ID=ID_FILE_PRINT_SETUP) do CWinApp::OnFilePrintSetup quản lý. Tương tự OnFilePrint, các hàm điều khiển lệnh không móc trước với ánh xạ thông điệp của lớp của chúng. Để tác động, ta phải tự làm ánh xạ thông điệp. Dĩ nhiên, nếu sử dụng *AppWizard* để sinh lõi ứng dụng có khả năng in, thì chúng thực hiện ánh xạ thông điệp cho ta. *AppWizard* còn ánh xạ ID_FILE_PRINT_DIRECT đến CView::OnFilePrint để cho phép in trực tiếp, có nghĩa in thực hiện không phải vì chọn *Print* từ thực đơn *File* mà bằng cách di biểu tượng tài liệu đến máy in.

8.6 QUAN SÁT TRƯỚC KHI IN

Một khi ứng dụng tài liệu/quan sát có thể in, việc thêm “quan sát trước khi in” thật đơn giản. Hãy thêm lệnh *Print Preview* rồi móc nó vào CView::OnFilePrintPreview. Có rất nhiều mã trình trong thư viện lớp trợ giúp OnFilePrintPreview (xem chi tiết ở Viewprev.cpp), nhưng thật đơn giản. OnFilePrintPreview lấy cửa sổ khung và tô nó bằng quan sát tạo từ lớp đặc biệt suy diễn từ CScrollView với tên CPreviewView. Chúng thêm thanh công cụ với các phím để tiến hay lùi trang, chuyển qua lại giữa các quan sát một trang hay hai trang, phóng to, thu nhỏ, v.v.. CPreviewView::OnDraw vẽ chữ nhật trắng biểu diễn trang in (hay vẽ hai chữ nhật nếu chọn quan sát hai trang), đặt một số tham số khác để vùng in của chữ nhật trắng phù hợp với vùng in của trang thực, và gọi OnPrint để vẽ chữ nhật. Khi đề cập đến ứng dụng của ta thì đầu ra chuyển đến máy in bằng cùng hàm ảo được gọi để in. Nhưng trong thực tế, dữ liệu ra lại đi đến cửa sổ quan sát trước.

Một lý do để chức năng quan sát trước khi in hoạt động được là dc trong tham số pDC của các hàm in của CView hiện tại là hai dc trong một. Mỗi đối tượng CDC chứa hai hDC: một cho “output” DC (m_hDC) và một cho “attribute” DC (m_hAttribDC). MFC sử dụng DC ra để gọi hàm phát sinh dữ liệu ra vật lý và DC thuộc tính cho các lời gọi để hỏi các thông tin thuộc tính - thí dụ, màu, chế độ nền hiện hành. Phần lớn thời gian, m_hDC và m_hAttribDC cùng lưu trữ hDC. Nhưng trong khi quan sát trước khi in, m_hDC qui chiếu đến DC màn hình nơi các trang được quan sát trước và m_hAttribDC qui chiếu đến DC của máy in. Kết quả? Nếu ứng dụng của ta sử dụng GetDeviceCaps hay các hàm khác để hỏi, đáp GDI về khả năng của máy in hay các đặc tính của

trang in, thông tin nhận lại là chân thật vì nó đi từ DC in. Nhưng toàn bộ dữ liệu ra vật lý sẽ hướng tới DC màn hình.

Khi người dùng đóng cửa sổ quan sát trước khi in, khung ứng dụng gọi hàm ảo **OnEndPrintPreview** của **CView** để báo cho ứng dụng biết việc quan sát trước khi in kết thúc. Cài đặt mặc định gọi **OnEndPrinting** để làm tích cực lại quan sát ban đầu, và phá hủy cửa sổ xem trước khi in. Lập trình viên đôi khi nạp chồng **OnEndPrintPreview** để trượt đến trang cuối. Đoạn mã trình sau mô tả chúng:

```
void CMyView::OnEndPrintPreview (CDC* pDC, CPrintInfo* pInfo,
                                POINT point, CPreviewView* pView)
{
    nPage = pInfo->m_nCurPage;
    POINT pt;
    // Chuyển nPage vào vị trí trượt trong biến pt...
    ScrollToPosition (pt);
    CScrollView::OnEndPrintPreview (pDC, pInfo, point, pView);
}
```

Ta phải viết mã trình để chuyển số trang hiện hành vào vị trí trượt. Đừng dựa vào tham số *point* chuyển cho **OnEndPrintPreview**; ở bản MFC này chúng luôn có giá trị (0,0). Đừng quên gọi bản lớp cơ sở của **OnEndPrintPreview** để khung ứng dụng có thể ra khỏi chế độ xem trước khi in và phục hồi cửa sổ khung về trạng thái ban đầu.

Nếu cần phân biệt giữa in thực và quan sát trước khi in, ta phải kiểm tra **CPrintInfo::m_bPreview**. Tương tự, phải xem xét kỹ **CPrintInfo::m_nNumPreviewPages** để xác định xem một hay hai trang được hiển thị.

8.7 THÍ DỤ MẪU VỀ IN

Ứng dụng **EZPrint** (mã nguồn trong chương trình 8.1) biểu diễn một ứng dụng tài liệu/quan sát với rất ít mã trình nhưng trợ giúp in và xem trước khi in. Tài liệu của **EZPrint** chỉ đơn giản là đường tròn xanh, đường kính 10 cm (1000 đơn vị trong chế độ ánh xạ **MM_LOMETRIC**) bên trong là màu vàng. Thực đơn **Tệp** của ứng dụng có bốn mục: “In...”, “Quan sát trước trang in”, “Chọn máy in...”, và “Kết thúc”. Lệnh **In** và **Quan sát trước trang in** được ánh xạ đến **CView::OnFilePrint** và **CView::OnFilePrintPreview** trong ánh xạ thông điệp của **CPrintView**, lệnh **Cài đặt máy in** được ánh xạ đến **CWinApp::OnFilePrintSetup** trong ánh xạ thông điệp của **CPrintApp**. Lệnh **In** hiển thị hộp thoại *Print* để cho người dùng chỉ ra các thuộc tính in như loại máy in, trang in, tổng số bản sao. **Quan sát trước trang in** kích hoạt chế độ quan sát trước khi in (xem hình 8.2). **Cài đặt máy in** hiển thị hộp thoại *Print Setup* cho phép chọn máy in, cỡ giấy, hướng giấy (đứng hay nằm), ...v.v.

Chương trình 8.1 Chương trình EZPrint**Tệp RESOURCE.H**

```
#define IDR_MAINFRAME      100
//-----
```

Tệp RESRC1.H

```
#define _APS_NEXT_RESOURCE_VALUE  101
#define _APS_NEXT_COMMAND_VALUE  40001
#define _APS_NEXT_CONTROL_VALUE   1000
#define _APS_NEXT_SYMED_VALUE     101
//-----
```

Tệp EZPRINT.RC

```
#include "resrc1.h"
#define APSTUDIO_READONLY_SYMBOLS
#include "afxres.h"
#include "resource.h"
// Thực đơn
IDR_MAINFRAME MENU DISCARDABLE
BEGIN
    POPUP "&Tệp"
    BEGIN
        MENUITEM "&In... \tCtrl+P",      ID_FILE_PRINT
        MENUITEM "&Quan sát trước trang in",  ID_FILE_PRINT_PREVIEW
        MENUITEM "&Chọn máy in...",      ID_FILE_PRINT_SETUP
        MENUITEM SEPARATOR
        MENUITEM "&Kết thúc",            ID_APP_EXIT
    END
END
// Phím tăng tốc
IDR_MAINFRAME ACCELERATORS MOVEABLE PURE
BEGIN
    "P",      ID_FILE_PRINT,      VIRTKEY, CONTROL
END
// bảng chuỗi ký tự
STRINGTABLE DISCARDABLE
BEGIN
    IDR_MAINFRAME      "EZPrint"
END
```

```
STRINGTABLE DISCARDABLE
```

```
BEGIN
```

```
    ID_FILE_PRINT        "Thực hiện in"
```

```
END
```

```
//-----
```

Tệp EZPRINT.H

```
class CPrintApp : public CWinApp
```

```
{
```

```
public:
```

```
    virtual BOOL InitInstance ();
```

```
protected:
```

```
    DECLARE_MESSAGE_MAP ()
```

```
};
```

```
//-----
```

Tệp EZPRINT.CPP

```
#include <afxwin.h>
```

```
#include "Resource.h"
```

```
#include "EZPrint.h"
```

```
#include "EZPrintDoc.h"
```

```
#include "EZPrintView.h"
```

```
CPrintApp myApp;
```

```
BEGIN_MESSAGE_MAP (CPrintApp, CWinApp)
```

```
    ON_COMMAND (ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
```

```
END_MESSAGE_MAP ()
```

```
BOOL CPrintApp::InitInstance ()
```

```
{
```

```
    CSingleDocTemplate* pDocTemplate;
```

```
    pDocTemplate = new CSingleDocTemplate( IDR_MAINFRAME, RUNTIME_CLASS (CPrintDoc),
```

```
        RUNTIME_CLASS (CFrameWnd), RUNTIME_CLASS (CPrintView) );
```

```
    AddDocTemplate (pDocTemplate);
```

```
    CCommandLineInfo cmdInfo;
```

```
    ParseCommandLine (cmdInfo);
```

```
    if (!ProcessShellCommand (cmdInfo))
```

```
        return FALSE;
```

```
    return TRUE;
```

```
}
```

```
//-----
```

Tệp EZPrintDoc.H

```
class CPrintDoc : public CDocument
{
    DECLARE_DYNCREATE (CPrintDoc)
};
//-----
```

Tệp EZPrintDoc.CPP

```
#include <afxwin.h>
#include "EZPrintDoc.h"
IMPLEMENT_DYNCREATE (CPrintDoc, CDocument)
//-----
```

EZPrintView.H

```
class CPrintView : public CView
{
    DECLARE_DYNCREATE (CPrintView)
protected:
    virtual void OnDraw (CDC*);
    virtual BOOL OnPreparePrinting (CPrintInfo*);
    DECLARE_MESSAGE_MAP ()
};
//-----
```

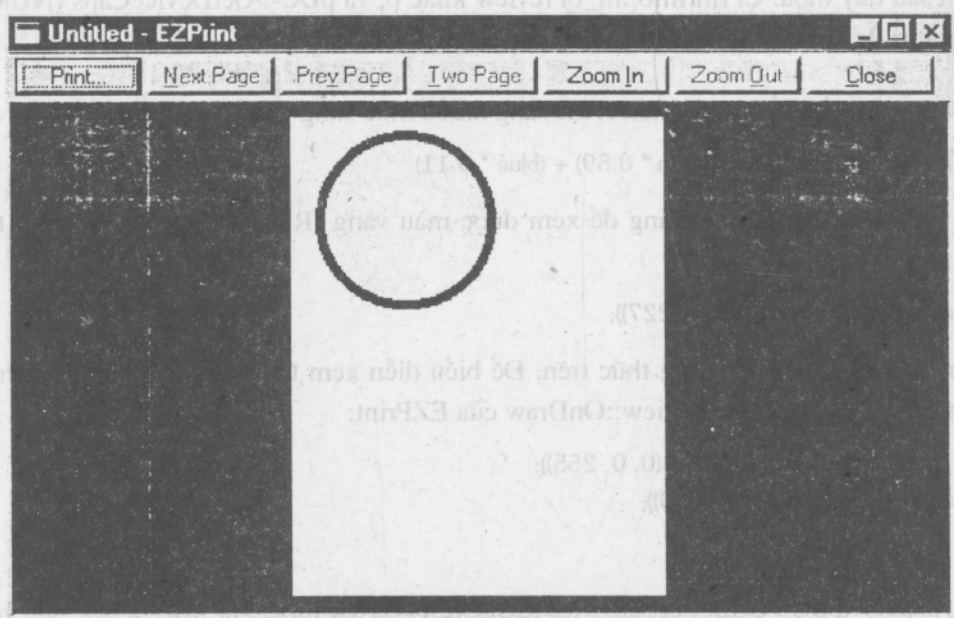
Tệp EZPrintView.CPP

```
#include <afxwin.h>
#include <afxext.h>
#include "EZPrintView.h"
IMPLEMENT_DYNCREATE (CPrintView, CView)
BEGIN_MESSAGE_MAP (CPrintView, CView)
    ON_COMMAND (ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND (ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP ()
void CPrintView::OnDraw (CDC* pDC)
{
    CPen pen (PS_SOLID, 50, RGB (0, 0, 255));
    CBrush brush (RGB (255, 255, 0));
    pDC->SetMapMode (MM_LOMETRIC);
    CPen* pOldPen = pDC->SelectObject (&pen);
    CBrush* pOldBrush = pDC->SelectObject (&brush);
```

```

pDC->Ellipse (100, -100, 1100, -1100);
pDC->SelectObject (pOldBrush);
pDC->SelectObject (pOldPen);
}
BOOL CPrintView::OnPreparePrinting (CPrintInfo* pInfo)
{
    pInfo->SetMaxPage (1);
    return DoPreparePrinting (pInfo);
}
//-----

```



Hình 8.2 Quan sát trước khi in của chương trình EZPrint

Không có gì nhiều để nói về khả năng in và quan sát trước khi in của EZPrint. Khung ứng dụng đã làm hầu hết mọi công việc. Lớp quan sát CPrintView chỉ nạp chồng một hàm in của CView là OnPreparePrinting để đặt đếm trang từ 1.

```

BOOL CPrintView::OnPreparePrinting (CPrintInfo* pInfo)
{

```

```

    pInfo->SetMaxPage (1);

```

```

    return DoPreparePrinting (pInfo);
}

```

CPrintView::OnDraw thực hiện toàn bộ việc vẽ, đầu ra có thể là màn hình, máy in hay cửa sổ xem trước khi in. Để đảm bảo hình tròn cân đối, OnDraw sử dụng chế độ ánh xạ

MM_LOMETRIC. Điều này quan trọng bởi vì giá trị tính *pixellinch* của màn hình và máy in ít khi như nhau. Nếu vẽ trên màn hình và máy in ở chế độ MM_TEXT, hình tròn sẽ quá nhỏ trên máy in 600 DPI. Để có kết quả WYSIWYG, ta phải dân độ rộng và độ cao của hình tròn bằng tay khi in hay quan sát trước khi in bằng tỷ lệ suy diễn từ số lượng *pixellinch* của màn hình và máy in. Sử dụng chế độ ánh xạ với đơn vị logic thì GDI sẽ làm cho ta. Chức năng quan sát trước khi in của khung ứng dụng không thực hiện hoàn hảo việc này. Trang quan sát trước của thí dụ này chỉ ra hình tròn đầy màu ngay cả khi ta gắn vào máy tính một máy in đen trắng. Tự nhiên, hình tròn sẽ in màu nếu có máy in màu, nhưng các trang xem trước khi in lại không cho màu thực khi in đen trắng. Ta có thể thêm đoạn mã trình xem trước khi in bằng sử dụng mức sáng của ảnh nếu cả hai điều kiện sau đây thoả: CPrintInfo::m_bPreview khác 0, và pDC->GetDeviceCaps (NUMCOLORS) cho lại giá trị 2, cho biết rằng máy in là đen trắng.

Ta có thể chuyển đổi giá trị màu RGB sang nhiều mức sáng theo công thức:

$$\text{greylevel} = (\text{red} * 0.30) + (\text{green} * 0.59) + (\text{blue} * 0.11)$$

Lệnh này tạo chổi đa mức sáng để xem được màu vàng (RGB (255, 255, 0)) trên thiết bị đơn sắc:

```
CBrush brush (RGB (227, 227, 227));
```

Giá trị 227 được tính từ công thức trên. Để biểu diễn xem trước khi in đen trắng, hãy thay thế các dòng sau trong hàm CPrintView::OnDraw của EZPrint:

```
CPen pen (PS_SOLID, 50, RGB (0, 0, 255));
```

```
CBrush brush (RGB (255, 255, 0));
```

bằng:

```
BOOL bMono=(pDC->GetDeviceCaps (NUMCOLORS)==2) && (pDC->m_hDC != pDC->m_hAttribDC);
```

```
// Chỉ dành cho chế độ quan sát trước khi in
```

```
CPen pen (PS_SOLID, 50, bMono ? RGB (28, 28, 28) : RGB (0, 0, 255));
```

```
CBrush brush (bMono ? RGB (227, 227, 227) : RGB (255, 255, 0));
```

Dữ liệu ra của quan sát trước khi in được gửi ra là đa mức sáng khi máy in mặc định là mô hình đen-trắng.

8.8 THÍ DỤ IN HOÀN HẢO HƠN

EZPrint không phản ánh đúng ứng dụng trong thực tế. Chúng không đề cập đến xử lý trang vì tài liệu của chúng chỉ có một trang. Chúng tạo các tài nguyên GDI khi cần mỗi khi gọi hàm OnDraw, cho nên không có lý do nào để nạp chồng OnBeginPrinting và OnEndPrinting để xin cấp phát tài nguyên đặc thù cho máy in. Thậm chí OnPrepareDC và OnPrint cũng không cần thiết bởi vì trong EZPrint không có gì để phân biệt quan sát in và quan sát màn hình.

```

HexDump.cpp - HexDump
Iệp
00000000  2F 2F 20 20 48 65 78 44 - 75 6D 70 2E 63 70 70 OD    ....HexDump.cpp.
00000010  0A 23 69 6E 63 6C 75 64 - 65 20 3C 61 66 78 77 69    ...include...afxwi
00000020  6E 2E 68 3E 0D 0A 23 69 - 6E 63 6C 75 64 65 20 22    n.h....include..
00000030  52 65 73 6F 75 72 63 65 - 2E 68 22 0D 0A 23 69 6E    Resource.h....in
00000040  63 6C 75 64 65 20 22 48 - 65 78 44 75 6D 70 2E 68    clude..HexDump.h
00000050  22 0D 0A 23 69 6E 63 6C - 75 64 65 20 22 48 65 78    ....include..Hex
00000060  44 6F 63 2E 68 22 0D 0A - 23 69 6E 63 6C 75 64 65    Doc.h....include
00000070  20 22 48 65 78 56 69 65 - 77 2E 68 22 0D 0A 43 48    ..HexView.h...CH
00000080  65 78 44 75 6D 70 41 70 - 70 20 6D 79 41 70 70 3B    exDumpApp.myApp.
00000090  0D 0A 42 45 47 49 4E 5F - 4D 45 53 53 41 47 45 5F    ..BEGIN.MESSAGE.
000000A0  4D 41 50 20 28 43 48 65 - 78 44 75 6D 70 41 70 70    MAP..CHexDumpApp
000000B0  2C 20 43 57 69 6E 41 70 - 70 29 0D 0A 20 20 20 20    ..CWinApp.....
000000C0  4F 4E 5F 43 4F 4D 4D 41 - 4E 44 20 28 49 44 5F 46    ON.COMMAND..ID.F
000000D0  49 4C 45 5F 4F 50 45 4E - 2C 20 43 57 69 6E 41 70    ILE.OPEN..CWinAp
000000E0  70 3A 3A 4F 6E 46 69 6C - 65 4F 70 65 6E 29 0D 0A    p..OnFileOpen...
000000F0  20 20 20 20 4F 4E 5F 43 - 4F 4D 4D 41 4E 44 20 28    ....ON.COMMAND..
00000100  49 44 5F 46 49 4C 45 5F - 50 52 49 4E 54 5F 53 45    ID.FILE.PRINT.SE
00000110  54 55 50 2C 20 43 57 69 - 6E 41 70 70 3A 3A 4F 6E    TUP..CWinApp..On
00000120  46 69 6C 65 50 72 69 6E - 74 53 65 74 75 70 29 0D    FilePrintSetup..
00000130  0A 45 4E 44 5F 4D 45 53 - 53 41 47 45 5F 4D 41 50    .END.MESSAGE.MAP
00000140  20 28 29 0D 0A 42 4F 4F - 4C 20 43 48 65 78 44 75    ....BOOL.CHexDu
00000150  6D 70 41 70 70 3A 3A 49 - 6E 69 74 49 6E 73 74 61    mpApp..InitInsta
  
```

Hình 8.3 Tài liệu của HexDump

```

HexDump.cpp - HexDump
Print Next Page Prev Page Two Page Zoom In Zoom Out Close
E:\App_1\MFC_BOOK\HexDump\HexDump.cpp Page 1
00000000  2F 2F 20 20 48 65 78 44 - 75 6D 70 2E 63 70 70 OD    ....HexDump.cpp.
00000010  0A 23 69 6E 63 6C 75 64 - 65 20 3C 61 66 78 77 69    ...include...afxwi
00000020  6E 2E 68 3E 0D 0A 23 69 - 6E 63 6C 75 64 65 20 22    n.h....include..
00000030  52 65 73 6F 75 72 63 65 - 2E 68 22 0D 0A 23 69 6E    Resource.h....in
00000040  63 6C 75 64 65 20 22 48 - 65 78 44 75 6D 70 2E 68    clude..HexDump.h
00000050  22 0D 0A 23 69 6E 63 6C - 75 64 65 20 22 48 65 78    ....include..Hex
00000060  44 6F 63 2E 68 22 0D 0A - 23 69 6E 63 6C 75 64 65    Doc.h....include
00000070  20 22 48 65 78 56 69 65 - 77 2E 68 22 0D 0A 43 48    ..HexView.h...CH
00000080  65 78 44 75 6D 70 41 70 - 70 20 6D 79 41 70 70 3B    exDumpApp.myApp.
00000090  0D 0A 42 45 47 49 4E 5F - 4D 45 53 53 41 47 45 5F    ..BEGIN.MESSAGE.
000000A0  4D 41 50 20 28 43 48 65 - 78 44 75 6D 70 41 70 70    MAP..CHexDumpApp
000000B0  2C 20 43 57 69 6E 41 70 - 70 29 0D 0A 20 20 20 20    ..CWinApp.....
000000C0  4F 4E 5F 43 4F 4D 4D 41 - 4E 44 20 28 49 44 5F 46    ON.COMMAND..ID.F
000000D0  49 4C 45 5F 4F 50 45 4E - 2C 20 43 57 69 6E 41 70    ILE.OPEN..CWinAp
000000E0  70 3A 3A 4F 6E 46 69 6C - 65 4F 70 65 6E 29 0D 0A    p..OnFileOpen...
000000F0  20 20 20 20 4F 4E 5F 43 - 4F 4D 4D 41 4E 44 20 28    ....ON.COMMAND..
00000100  49 44 5F 46 49 4C 45 5F - 50 52 49 4E 54 5F 53 45    ID.FILE.PRINT.SE
00000110  54 55 50 2C 20 43 57 69 - 6E 41 70 70 3A 3A 4F 6E    TUP..CWinApp..On
  
```

Hình 8.4 Tiêu đề HexDump ở chế độ quan sát trước khi in

Thí dụ HexDump trong hình 8.3 thể hiện kiểu ứng dụng mà ta hay viết hơn. HexDump là chương trình quan sát mã 16, hiển thị nội dung của tệp dưới dạng nhị phân. Tài liệu in có tiêu đề trên đỉnh với tên tệp và đường dẫn nếu còn chỗ và số trang. Đầu tệp có gạch chân bằng đường vẽ mỏng. Đường thẳng được vẽ bằng CDC::MoveTo và CDC::LineTo; toàn bộ dữ liệu ra khác nhờ hàm CDC::TextOut.

Hình 8.4 chỉ ra đầu trang ở chế độ xem trước khi in. Khi tài liệu được in, HexDump hỏi CDC để lấy kích thước của miền trang in và hiệu chỉnh đầu ra cho tương ứng. Chiều cao của trang được sử dụng để tính số dòng in của một trang, độ rộng được sử dụng để chỉnh giữa số liệu ra mà không quan tâm đến kích thước trang. Mã nguồn của HexDump được mô tả trong chương trình 8.2.

Chương trình 8.2 HexDump

Tệp Resource.H

```
#define IDR_MAINFRAME      100
```

```
//-----
```

Tệp Resrc1.H

```
#ifdef APSTUDIO_INVOKED
```

```
#ifndef APSTUDIO_READONLY_SYMBOLS
```

```
#define _APS_NEXT_RESOURCE_VALUE    101
```

```
#define _APS_NEXT_COMMAND_VALUE     40001
```

```
#define _APS_NEXT_CONTROL_VALUE     1000
```

```
#define _APS_NEXT_SYMED_VALUE       101
```

```
#endif
```

```
#endif
```

```
//-----
```

Tệp HexDump.RC

```
#include "resrc1.h"
```

```
#define APSTUDIO_READONLY_SYMBOLS
```

```
#include "afxres.h"
```

```
#include "resource.h"
```

```
// Thực đơn
```

```
IDR_MAINFRAME MENU DISCARDABLE
```

```
BEGIN
```

```
    POPUP "&Tệp"
```

```
    BEGIN
```

```
        MENUITEM "&Mở tệp... \tCtrl+O",    ID_FILE_OPEN
```

```
        MENUITEM SEPARATOR
```

```

MENUITEM "&In... \tCtrl+P",      ID_FILE_PRINT
MENUITEM "&Quan sát trước trang in",  ID_FILE_PRINT_PREVIEW
MENUITEM "&Chọn máy in ...",      ID_FILE_PRINT_SETUP
MENUITEM SEPARATOR
MENUITEM "Tệp hiện thời",        ID_FILE_MRU_FILE1
MENUITEM SEPARATOR
MENUITEM "&Kết thúc",            ID_APP_EXIT
END
END
// Phim cấp tốc
IDR_MAINFRAME ACCELERATORS MOVEABLE PURE
BEGIN
    "O",      ID_FILE_OPEN,      VIRTKEY, CONTROL
    "P",      ID_FILE_PRINT,     VIRTKEY, CONTROL
END
// bảng xâu ký tự
STRINGTABLE DISCARDABLE
BEGIN
    IDR_MAINFRAME      "HexDump"
END
//-----
Tệp HexDump.H

class CHexDumpApp : public CWinApp
{
public:
    virtual BOOL InitInstance ();
protected:
    DECLARE_MESSAGE_MAP ()
};
//-----
Tệp HexDump.CPP

#include <afxwin.h>
#include "Resource.h"
#include "HexDump.h"
#include "HexDoc.h"
#include "HexView.h"
CHexDumpApp myApp;
BEGIN_MESSAGE_MAP (CHexDumpApp, CWinApp)

```



```

ON_COMMAND (ID_FILE_OPEN, CWinApp::OnFileOpen)
ON_COMMAND (ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP ()
BOOL CHexDumpApp::InitInstance ()
{
    SetRegistryKey ("Lập trình Windows với MFC");
    LoadStdProfileSettings ();
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate( IDR_MAINFRAME,  RUNTIME_CLASS (CHexDoc),
        RUNTIME_CLASS (CFrameWnd),  RUNTIME_CLASS (CHexView)
    );
    AddDocTemplate (pDocTemplate);
    CCommandLineInfo cmdInfo;
    ParseCommandLine (cmdInfo);
    if (!ProcessShellCommand (cmdInfo))    return FALSE;
    m_pMainWnd->DragAcceptFiles ();
    return TRUE;
}

```

//-----

Tệp HexDoc.H

```

class CHexDoc : public CDocument
{
    DECLARE_DYNCREATE (CHexDoc)
private:
    UINT m_nDocLength;
    BYTE* m_pFileData;
public:
    CHexDoc ();
    virtual void DeleteContents ();
    virtual void Serialize (CArchive&);
    UINT GetBytes (UINT, UINT, PVOID);
    UINT GetDocumentLength ();
};

```

//-----

Tệp HexDoc.CPP

```

#include <afxwin.h>
#include "HexDoc.h"
IMPLEMENT_DYNCREATE (CHexDoc, CDocument)

```

```

CHexDoc::CHexDoc ()
{
    m_nDocLength = 0;
    m_pFileData = NULL;
}

void CHexDoc::DeleteContents ()
{
    if (m_pFileData != NULL) {
        delete[] m_pFileData;
        m_pFileData = NULL;
        m_nDocLength = 0;
    }
}

void CHexDoc::Serialize (CArchive& ar)
{
    if (ar.IsLoading ()) {
        CFile* pFile = ar.GetFile ();
        m_nDocLength = (UINT) pFile->GetLength ();
        // Allocate a buffer for the file data
        try
            { m_pFileData = new BYTE[m_nDocLength]; }
        catch (CMemoryException* e) {
            m_nDocLength = 0;
            throw e;
        }
        // Đọc dữ liệu từ tệp vào vùng đệm
        try
            { pFile->Read (m_pFileData, m_nDocLength); }
        catch (CFileException* e) {
            delete[] m_pFileData;
            m_pFileData = NULL;
            m_nDocLength = 0;
            throw e;
        }
    }
}

UINT CHexDoc::GetBytes (UINT nIndex, UINT nCount, PVOID pBuffer)
{
    if (nIndex >= m_nDocLength)

```

```

        return 0;
    UINT nLength = nCount;
    if ((nIndex + nCount) > m_nDocLength)
        nLength = m_nDocLength - nIndex;
    ::CopyMemory (pBuffer, m_pFileData + nIndex, nLength);
    return nLength;
}

UINT CHexDoc::GetDocumentLength ()
{
    return m_nDocLength;
}

//-----

```

Tệp HexView.H

```

class CHexView : public CScrollView
{
    DECLARE_DYNCREATE (CHexView)
private:
    CFont m_screenFont;
    CFont m_printerFont;
    UINT m_cyScreen;
    UINT m_cyPrinter;
    UINT m_cxOffset;
    UINT m_cxWidth;
    UINT m_nLinesTotal;
    UINT m_nLinesPerPage;
    CHexDoc* GetDocument () { return (CHexDoc*) m_pDocument; }
    void FormatLine (UINT, CString&);
    void PrintPageHeader (CDC*, UINT);
    void PrintPage (CDC*, UINT);
public:
    virtual void OnInitialUpdate ();
protected:
    virtual void OnDraw (CDC*);
    virtual BOOL OnPreparePrinting (CPrintInfo*);
    virtual void OnBeginPrinting (CDC*, CPrintInfo*);
    virtual void OnPrint (CDC*, CPrintInfo*);
    virtual void OnEndPrinting (CDC*, CPrintInfo*);
    afx_msg int OnCreate (LPCREATESTRUCT);

```

```

DECLARE_MESSAGE_MAP ()
};
//-----
Tệp HexView.CPP

#include <afxwin.h>
#include <afxext.h>
#include "HexDoc.h"
#include "HexView.h"
#define PRINTMARGIN 2
IMPLEMENT_DYNCREATE (CHexView, CScrollView)
BEGIN_MESSAGE_MAP (CHexView, CScrollView)
    ON_WM_CREATE ()
    ON_COMMAND (ID_FILE_PRINT, CScrollView::OnFilePrint)
    ON_COMMAND (ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
    ON_COMMAND (ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP ()

int CHexView::OnCreate (LPCREATESTRUCT lpcs)
{
    if (CScrollView::OnCreate (lpcs) == -1)    return -1;
    CClientDC dc (this);
    int nHeight = ((dc.GetDeviceCaps (LOGPIXELSY) * 10) / 72);
    m_screenFont.CreateFont (nHeight, 0, 0, 0, FW_NORMAL, 0, 0, 0, DEFAULT_CHARSET,
        OUT_CHARACTER_PRECIS, CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH | FF_DONTCARE, "Courier New");
    TEXTMETRIC tm;
    CFont* pOldFont = dc.SelectObject (&m_screenFont);
    dc.GetTextMetrics (&tm);
    m_cyScreen = tm.tmHeight + tm.tmExternalLeading;
    dc.SelectObject (pOldFont);
    return 0;
}

void CHexView::OnInitialUpdate ()
{
    UINT nDocLength = GetDocument ()->GetDocumentLength ();
    m_nLinesTotal = (nDocLength + 15) / 16;
    SetScrollSizes (MM_TEXT, CSize (0, m_nLinesTotal * m_cyScreen),
        CSize (0, m_cyScreen * 10), CSize (0, m_cyScreen));
    ScrollToPosition (CPoint (0, 0));
}

```

```

CScrollView::OnInitialUpdate ();
}

void CHexView::OnDraw (CDC* pDC)
{
    if (m_nLinesTotal != 0) {
        CRect rect;
        pDC->GetClipBox (&rect);
        UINT nStart = rect.top / m_cyScreen;
        UINT nEnd = min (m_nLinesTotal - 1, (rect.bottom + m_cyScreen - 1) / m_cyScreen);
        CString string;
        CFont* pOldFont = pDC->SelectObject (&m_screenFont);
        for (UINT i=nStart; i<=nEnd; i++) {
            FormatLine (i, string);
            pDC->TextOut (2, (i * m_cyScreen) + 2, string);
        }
        pDC->SelectObject (pOldFont);
    }
}

BOOL CHexView::OnPreparePrinting (CPrintInfo* pInfo)
{
    return DoPreparePrinting (pInfo);
}

void CHexView::OnBeginPrinting (CDC* pDC, CPrintInfo* pInfo)
{
    int nHeight = ((pDC->GetDeviceCaps (LOGPIXELSY) * 10) / 72);
    m_printerFont.CreateFont (nHeight, 0, 0, 0, FW_NORMAL, 0, 0, 0, DEFAULT_CHARSET,
        OUT_CHARACTER_PRECIS, CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY,
        DEFAULT_PITCH | FF_DONTCARE, "Courier New");
    TEXTMETRIC tm;
    CFont* pOldFont = pDC->SelectObject (&m_printerFont);
    pDC->GetTextMetrics (&tm);
    m_cyPrinter = tm.tmHeight + tm.tmExternalLeading;
    CSize size = pDC->GetTextExtent ("-----1-----2-----" \
        "3-----4-----5-----6-----7-----8-", 81);
    pDC->SelectObject (pOldFont);
    m_nLinesPerPage = (pDC->GetDeviceCaps (VERTRES) -
        (m_cyPrinter * (3 + (2 * PRINTMARGIN)))) / m_cyPrinter;
    UINT nMaxPage = max (1, (m_nLinesTotal + (m_nLinesPerPage - 1)) / m_nLinesPerPage);
}

```

```

pInfo->SetMaxPage (nMaxPage);
m_cxOffset = (pDC->GetDeviceCaps (HORZRES) - size.cx) / 2;
m_cxWidth = size.cx;
}

void CHexView::OnPrint (CDC* pDC, CPrintInfo* pInfo)
{
    PrintPageHeader (pDC, pInfo->m_nCurPage);
    PrintPage (pDC, pInfo->m_nCurPage);
}

void CHexView::OnEndPrinting (CDC* pDC, CPrintInfo* pInfo)
{
    m_printerFont.DeleteObject ();
}

void CHexView::FormatLine (UINT nLine, CString& string)
{
    BYTE b[17];
    ::FillMemory (b, 16, 32);
    UINT nCount = GetDocument ()->GetBytes (nLine * 16, 16, b);
    string.Format ("%0.8X  %0.2X %0.2X %0.2X %0.2X %0.2X %0.2X %0.2X " \
        "%0.2X %0.2X - %0.2X %0.2X %0.2X %0.2X %0.2X %0.2X %0.2X " \
        "%0.2X  ", nLine * 16, \
        b[0], b[1], b[2], b[3], b[4], b[5], b[6], b[7], \
        b[8], b[9], b[10], b[11], b[12], b[13], b[14], b[15]);
    for (UINT i=0; i<nCount; i++) {
        if (!::IsCharAlphaNumeric (b[i]))
            b[i] = 0x2E;
    }
    b[nCount] = 0;
    string += b;
    if (nCount < 16) {
        UINT pos1 = 59;
        UINT pos2 = 60;
        UINT j = 16 - nCount;
        for (j=0; j<j; j++) {
            string.SetAt (pos1, ' ');
            string.SetAt (pos2, ' ');
            pos1 -= 3;
            pos2 -= 3;
            if (pos1 == 35) {

```

```

        string.SetAt (35, ' ');
        string.SetAt (36, ' ');
        pos1 = 33;
        pos2 = 34;
    }
}

void CHexView::PrintPageHeader (CDC* pDC, UINT nPageNumber)
{
    CString strHeader = GetDocument ()->GetPathName ();
    if (strHeader.GetLength () > 68)    strHeader = GetDocument ()->GetTitle ();
    CString strPageNumber;
    strPageNumber.Format ("Page %d", nPageNumber);
    UINT nSpaces = 81 - strPageNumber.GetLength () - strHeader.GetLength ();
    for (UINT i=0; i<nSpaces; i++)
        strHeader += ' ';
    strHeader += strPageNumber;
    UINT y = m_cyPrinter * PRINTMARGIN;
    CFont* pOldFont = pDC->SelectObject (&m_printerFont);
    pDC->TextOut (m_cxOffset, y, strHeader);
    y += (m_cyPrinter * 3) / 2;
    pDC->MoveTo (m_cxOffset, y);
    pDC->LineTo (m_cxOffset + m_cxWidth, y);
    pDC->SelectObject (pOldFont);
}

//-----
void CHexView::PrintPage (CDC* pDC, UINT nPageNumber)
{
    if (m_nLinesTotal != 0) {
        UINT nStart = (nPageNumber - 1) * m_nLinesPerPage;
        UINT nEnd = min (m_nLinesTotal - 1, nStart + m_nLinesPerPage - 1);
        CString string;
        CFont* pOldFont = pDC->SelectObject (&m_printerFont);
        UINT y;
        for (UINT i=nStart; i<=nEnd; i++) {
            FormatLine (i, string);
            y = ((i - nStart) + PRINTMARGIN + 3) * m_cyPrinter;
            pDC->TextOut (m_cxOffset, y, string);
        }
    }
}

```

```

    }
    pDC->SelectObject (pOldFont);
}
//-----

```

Lỗi của HexDump là mã trình trong lớp `CHexView`, chúng chuyển tài liệu ra màn hình hay máy in. `CHexView` thực hiện toàn bộ vẽ màn hình nhờ `OnDraw`. `CHexView::OnCreate` nạp giá trị ban đầu 10 điểm cho `CHexView::m_screenFont`, có chữ *Courier New* cho màn hình và khởi động `CHexView::m_cyScreen` bằng khoảng cách đứng (tính bằng *pixel*) để tách các đường liên nhau khỏi ký tự trên màn hình. `CHexView::OnDraw` vẽ lại quan sát bằng cách sử dụng `CDC::GetClipBox` để nhận ra chữ nhất cần vẽ lại, chuyển các tọa độ y của đỉnh và đáy chữ nhất thành số dòng bắt đầu và kết thúc trong tài liệu, và chỉ vẽ lại các đường cần vẽ lại. Vị trí trước được xác định tự động bởi vì `CHexView` được suy diễn từ `CScrollView`. Bởi vì `OnDraw` đủ thông minh để chỉ vẽ vài đường cần thiết. Khi không tối ưu `OnDraw` và nạp tài liệu rất lớn thì `CScrollView` sẽ trượt lỡ dở, thử viết lại `CHexView::OnDraw` để nó vẽ lại toàn bộ tài liệu mỗi khi được gọi.

```

void CHexView::OnDraw (CDC* pDC)
{
    for (int i=0; i<m_nLinesTotal; i++)
    {
        FormatLine (i, string);
        pDC->TextOut (2, (i * m_cyScreen) + 2, string);
    }
}

```

HexDump thực hiện toàn bộ công việc in trong `OnPrint`. `CHexView::OnPrint` gọi `CHexView::PrintPageHeader` để in tiêu đề ở đỉnh trang và `CHexView::PrintPage` để in thân trang. `OnBeginPrinting` cho khởi động `CHexView::m_printerFont` bằng có chữ 10 điểm, *Courier New* cho máy in, khởi động `CHexView::m_cyPrinter` bằng khoảng cách giữa hàng và `CHexView::m_nLinesPerPage` bằng tổng số dòng trên một trang sẽ in (trên cơ sở độ cao của vùng trang in được), `CHexView::m_exOffset` bằng giá trị x để chỉnh giữa dòng in trên trang, và `CHexView::m_exWidth` bằng độ rộng của mỗi dòng chữ. `CHexView::PrintPage` tính số dòng bắt đầu và kết thúc từ số trang hiện hành và tổng số dòng trong một trang. Vòng *for* để vẽ ở đây cũng như vòng *for* của `OnDraw`, chỉ khác cách chỉnh chữ trên trang và sử dụng `m_printerFont` để ra dữ liệu thay cho `m_screenFont`. Khi in (hay quan sát trước khi in) kết thúc, `OnEndPrinting` xóa có chữ in vừa tạo ra bởi `OnBeginPrinting`.

Có thể viết `CHexView::OnDraw` để chuyển dữ liệu ra cả màn hình và máy in. Nhưng mã trình ở đây rõ ràng hơn. Ta luôn có thể đặt các chức năng chung vào hàm riêng biệt để có thể gọi cả từ `OnDraw` và `OnPrint`.

8.9 CÁCH TIẾP CẬN DUY NHẤT ĐẾN VÀO/RA (Serialization)

HexDump quản lý vào/ra hàng cách ít thấy. Khi `CHexDoc::Serialize` đọc tài liệu từ đĩa, chúng không đọc các đối tượng từ *archive*. Thay vào đó, chúng thuê vùng đệm, rồi đọc tệp vào đó. Đoạn trình như sau:

```
if (ar.IsLoading ())
{
    CFile* pFile = ar.GetFile ();
    m_nDocLength = (UINT) pFile->GetLength ();
    m_pFileData = new BYTE[m_nDocLength];
    pFile->Read (m_pFileData, m_nDocLength);
}
```

`CHexDoc::DeleteContents` giải phóng vùng đệm khi tài liệu bị phá hủy hay sử dụng lại:

```
delete[] m_pFileData;
```

HexDump không ghi đĩa vì đây không phải chương trình soạn thảo, nhưng nếu muốn thì sử dụng `CFile::Write` để ghi lại nội dung lên đĩa như sử dụng `CFile::Read` để đọc vào bộ nhớ.

Xin cấp phát vùng đệm có độ lớn bằng kích thước tệp sẽ là cách làm không hiệu quả cho các tài liệu lớn bởi vì toàn bộ tài liệu phải đẩy vào bộ nhớ một lần. Tuy nhiên giới hạn do `CScrollView` còn chặt hơn cả vùng nhớ. Thí dụ HexDump nạp tệp lớn vài trăm kbyte. Không quan tâm đến độ lớn của tệp, HexDump sẽ không hiển thị nhiều hơn vài ngàn dòng. Cái gì đã xảy ra?

Vấn đề liên quan đến là việc kế thừa 16 bit của Windows 95/98, đặc biệt là dải trượt của thanh trượt là giá trị 16 bit. Khi `CHexView::OnInitialUpdate` đặt giới hạn quan sát trượt bằng `SetScrollSizes`, nó tính toán chiều cao ảo của quan sát bằng cách nhân tổng số dòng của tài liệu với tổng số *pixel* của một dòng. Nếu dòng cao 16 *pixel* và tài liệu chứa 1000 dòng, thì độ cao ảo của quan sát là 16000. Với tài liệu nhỏ thì chương trình chạy được, nhưng `CScrollView` không quản lý được độ cao hơn 32,767 (số dương lớn nhất có thể được biểu diễn bởi số nguyên có dấu 16 bit) bởi vì đây là giới hạn trên cùng của dải của thanh ghi trượt. Kết quả? Nếu nạp tài liệu chứa hơn 2000 dòng, thì `CScrollView` sẽ chỉ hiển thị một phần của tài liệu thậm chí ngay cả khi in và xem trước khi in hoạt động tốt. Để cho HexDump quản lý tài liệu lớn, ta sẽ phải tạo lấy `CView` có thanh trượt và tự xử lý các thông điệp của thanh trượt. (Cách đơn giản nhất là thêm thanh trượt vào `CView` để nạp chồng `CView::PreCreateWindow` và thêm `WS_VSCROLL` hay `WS_HSCROLL` vào kiểu của cửa sổ). `CScrollView` thuận tiện cho tài liệu nhỏ hay trung bình, nhưng với các tài liệu rất lớn thì phải tự cài đặt logic trượt.

Sau đây là một số thủ thuật và các câu trả lời các vấn đề hay gặp để giúp lập trình viên viết mã trình in tốt hơn và giải quyết các vấn đề chưa gặp trong các thí dụ trên.

8.9.1 Cho phép phím chọn hoạt động

Hộp thoại *Print* chuẩn bao gồm phím radio chọn hoạt động (*Selection*) để người sử dụng chọn in phần tài liệu hay toàn bộ tài liệu... Mặc định, phím này bị vô hiệu lực. Ta có thể làm hiệu lực chúng bằng cách thêm lệnh vào hàm nạp chồng `OnPreparePrinting` ngay trước khi gọi `DoPreparePrinting`:

```
pInfo->m_pPD->m_pd.Flags &= ~PD_NOSELECTION;
```

Để chọn phím *radio* sau khi chúng có hiệu lực, hãy thêm dòng:

```
pInfo->m_pPD->m_pd.Flags |= PD_SELECTION;
```

Trong đó, `m_pPD` trỏ đến `CPrintDialog` và `CPrintDialog::m_pd` giữ qui chiếu đến cấu trúc mà hộp thoại xây dựng trên đó, có tên là `PRINTDLG`. Mảng `Flags` của `PRINTDLG` giữ bit xác định các đặc tính của hộp thoại. Việc bỏ cờ `PD_NOSELECTION` do cấu tử `CPrintInfo` đã thêm trước đó làm hiệu lực phím *Selection*, và thêm cờ `PD_SELECTION` thì phím được chọn. Nếu `DoPreparePrinting` cho lại giá trị khác 0 (chỉ ra rằng hộp thoại đã kết thúc nhiệm vụ bằng phím OK), ta có thể biết được phím *Selection* có được chọn hay không nhờ hàm `CPrintDialog::PrintSelection`.

```
if (pInfo->m_pPD->PrintSelection()) {
    // In phần tài liệu đang được chọn
}
```

PrintSelection và các hàm khác của `CPrintDialog` cho lại thông tin về các thông số vào từ hộp thoại *Print* hay *Print Setup* có thể được gọi thông qua chuyển tham số `pInfo` tới `OnPreparePrinting` sau khi `DoPreparePrinting` trở lại. Chúng cũng có thể được gọi thông qua tham số `pInfo` chuyển cho `OnBeginPrinting` và các hàm nạp chồng của `CView`.

Ta có thể thay đổi thuộc tính mặc định và ứng xử của hộp thoại in chuẩn bằng cách thay đổi các giá trị trong cấu trúc `PRINTDLG`. Hãy xem thêm thông tin về `PRINTDLG` trong tài liệu đi kèm theo chương trình dịch hay Win32 SDK. Khi in hay gặp lỗi liên quan đến vùng in được trên trang. Ngay cả khi ta biết đang in ra trang giấy 8.5x11 inch, vùng trang có thể in là khác nhau với mỗi máy in. Vùng trang có thể in được thậm chí có thể khác nhau trên cùng máy in cùng cỡ giấy, phụ thuộc vào trình điều khiển máy in đã được cài đặt, và kích thước nằm, đứng của vùng trang in được có thể thay đổi do người dùng lựa chọn. Trình HexDump gọi `GetDeviceCaps` để xác định vùng in được mỗi lần in, hoặc sử dụng `CPrintInfo::m_rectDraw` trong hàm `OnPrint`. Điều chú ý này nhắc nhở lập trình viên phải cẩn thận khi viết chương trình cho máy in trong Windows để giảm thiểu số lỗi cho chương trình.

Bảng 8.3 liệt kê các giá trị khác mà ta có thể chuyển đến `GetDeviceCaps` để có các thông tin phụ về máy in hay các thiết bị ra khác. Ta đã xem xét cách sử dụng `NUMCOLORS` để nhận ra máy

in đen-trắng. Sử dụng `PHYSICALOFFSETX` và `PHYSICALOFFSETY` để đặt độ rộng giới hạn in trên cơ sở thông tin vào từ hộp thoại *Page Setup*. (Hàm `CWinApp::OnFilePrintSetup` của MFC hiển thị hộp thoại *Print Setup* thay cho hiển thị hộp thoại *Page Setup*, nhưng ta có thể tự hiển thị hộp thoại *Page Setup* nhờ hàm `CPageSetupDialog`). Nếu người dùng muốn giới hạn in một inch ở phía trái của trang thì phải trừ giá trị nhận được từ `GetDeviceCaps(PHYSICALOFFSETX)` đi giá trị cho lại của `GetDeviceCaps(LOGPIXELSX)` để có giá trị x từ phía trái của vùng trang in được khi bắt đầu in. Nếu trình điều khiển máy in cho lại thông tin chính xác, giới hạn sẽ ở vào khoảng lớn hơn một inch vài pixel. Các giá trị `HORZRES`, `VERTRES`, `LOGPIXELSX`, `LOGPIXELSY`, `PHYSICALWIDTH`, `PHYSICALHEIGHT`, `PHYSICALOFFSETX`, và `PHYSICALOFFSETY` có thể được tổ hợp để mô tả vùng in được của trang giấy.

Bảng 8.3 Các giá trị gửi cho `GetDeviceCaps` để lấy tham số thiết bị in.

Giá trị	Mô tả
<code>HORZRES</code>	Cho lại độ rộng tính bằng pixel của vùng in được
<code>VERTRES</code>	Cho lại độ cao tính bằng pixel của vùng in được
<code>HORZSIZE</code>	Cho lại độ rộng tính bằng mm của vùng in được
<code>VERTSIZE</code>	Cho lại độ cao tính bằng mm của vùng in được
<code>LOGPIXELSX</code>	Cho lại tổng số pixel/inch theo hướng ngang (máy in 300 dpi cho lại 300)
<code>LOGPIXELSY</code>	Cho lại tổng số pixel/inch theo hướng đứng (máy in 300 dpi cho lại 300)
<code>PHYSICALWIDTH</code>	Cho lại độ rộng trang giấy tính bằng pixel (máy in 300 dpi cho giá trị 2550 nếu chọn trang 8.5 x 11-inch)
<code>PHYSICALHEIGHT</code>	Cho lại độ cao của trang giấy tính bằng pixel (máy in 300 dpi cho giá trị 3350 nếu chọn trang 8.5 x 11-inch)
<code>PHYSICALOFFSETX</code>	Cho lại khoảng cách tính bằng pixel kể từ biên trái của trang đến giới hạn vùng in được của trang.
<code>PHYSICALOFFSETY</code>	Cho lại khoảng cách tính bằng pixel kể từ biên đỉnh của trang đến giới hạn trên của vùng in được của trang.
<code>TECHNOLOGY</code>	Cho lại giá trị chỉ ra kiểu của thiết bị ra mà DC có quan hệ (thường cho lại <code>DT_RASDISPLAY</code> đối với màn hình, <code>DT_RASPRINTER</code> đối với máy in, và <code>DT_PLOTTER</code> với máy vẽ)
<code>RASTERCAPS</code>	Cho lại đây các cờ bit chỉ ra mức của GDI mà máy in trợ giúp (thí dụ, cờ <code>RC_BITBLT</code> cho biết máy in trợ giúp <code>BitBlt</code> , và <code>RC_STRETCHBLT</code> cho biết máy in trợ giúp <code>StretchBlt</code>)
<code>NUMCOLORS</code>	Cho lại tổng số màu mà máy in trợ giúp (cho giá trị 2 nếu máy in đen-trắng)

Nếu ta phải làm việc với thiết bị ra không vẽ *bitmap* thì phải biết rằng nó có trợ giúp các hàm `CDC::BitBlt` và `CDC::StretchBlt` hay không bằng cách gọi hàm `GetDeviceCaps` với tham số `RASTERCAPS` rồi kiểm tra các cờ cho lại. Đây là trường hợp sử dụng máy vẽ, hay thiết bị véc tơ. Thí dụ, để biết thiết bị ra có phải là máy vẽ không thì phải gọi hàm `GetDeviceCaps(TECHNOLOGY)` và kiểm tra giá trị cho lại có phải là `DT_PLOTTER` hay không.

Một vấn đề có thể gặp phải là các thông tin do trình điều khiển máy in cho lại không phù hợp với dữ liệu ra do nó phát sinh. Thí dụ, vài trình điều khiển máy in cho cùng giá trị khi yêu cầu `PHYSICALWIDTH` / `PHYSICALHEIGHT` và `HORZRES` / `VERTRES`. Đôi khi hàm thông thường

như CDC::TextOut làm việc tốt với hàng trăm máy in nhưng lại không tốt với loại máy in nào đó do lỗi trong trình điều khiển của chúng ...

8.9.2 Trợ giúp đánh số trang mặc định

Trình HexDump gọi CPrintInfo::SetMaxPage từ OnBeginPrinting thay vì từ OnPreparePrinting vì hai lý do. Trước hết, tiến trình đánh số trang cần biết vùng trang in được. Thứ hai, OnBeginPrinting là hàm CView ảo đầu tiên mà MFC gọi với con trỏ đến DC máy in. Tuy nhiên, số trang cực đại chưa được đặt cho đến khi hàm OnPreparePrinting trở lại, số trang “*from*” trong hộp thoại *Print* được đặt 1 nhưng số trang “*to*” chưa được biết. Một số người sử dụng có thể cho đó là điều lạ vì ứng dụng không biết số trang cực đại. Hơn nữa để hiển thị chính xác số trang cực đại, rất nhiều ứng dụng hiển thị xâu “*Page nnn of nn*” trong thanh trạng thái khi xem trước khi in. Làm thế nào mà ứng dụng biết tài liệu được đánh số trang như thế nào trong khi chúng không biết loại máy in sẽ in tài liệu và hướng in trang ra sao?

Câu trả lời là chúng không biết chắc chắn, do đó chúng lấy số dự đoán trên cơ sở đặc tính của máy in mặc định. Đoạn mã trình sau đây khởi động đối tượng CSize bằng kích thước tính bằng *pixel* của vùng trang in được của máy in mặc định. Có thể gọi chúng từ OnPreparePrinting hay nơi khác để tính số trang cực đại.

```
CSize size;
CPrintDialog dlg (FALSE);
if (dlg.GetDefaults ())
{
    CDC dc;
    dc.Attach (dlg.GetPrinterDC ());
    size.cx = dc.GetDeviceCaps (HORZRES);
    size.cy = dc.GetDeviceCaps (VERTRES);
}
```

CPrintDialog::GetDefaults khởi động CPrintDialog::m_pd bằng giá trị của máy in mặc định mà không hiển thị hộp thoại. Nếu GetDefaults cho lại 0 thì có nghĩa là không có máy in nào nối với máy tính hay không máy in nào được gán là mặc định. Hàm CPrintDialog::GetPrinterDC cho lại hDC do GetDefaults lưu trong m_pd. Bởi vì đến lúc ứng dụng phải xoá dc, thậm chí là nó tạo qua Windows, hDC không bị tách ra trước khi dc ra khỏi vùng tác động vậy nó sẽ được phá hủy bởi hủy tử của đối tượng. Điều quan trọng là chuyển tham số FALSE chứ không phải TRUE cho cấu tử CPrintDialog, hoặc GetPrinterDC sẽ cho lại hDC với giá trị NULL.

8.10 LIỆT KÊ MÁY IN

Đôi khi cần phải xây dựng danh sách toàn bộ máy in mà người dùng có thể chọn được ngoài các

hộp thoại *Print* hay *Print Setup*. Trình trong chương trình 8.3 sử dụng `::EnumPrinters` để liệt kê các máy in nối cục bộ, và thêm các đầu vào trong hộp *combo*. Trong Windows 95/98, các máy in mạng cũng được đưa vào danh sách vì các máy in mạng, cục bộ đều được xử lý như nhau. Lỗi gọi thứ nhất tới `::EnumPrinters` lấy tổng số bytes cần thiết để lưu giữ mảng của các cấu trúc `PRINTER_INFO_5` để mô tả từng máy in. Lỗi gọi thứ hai để tạo thông tin. Khi trở lại, `dwPrinters` có tổng số máy in. Mỗi mảng `pPrinterName` của cấu trúc chứa con trỏ đến xâu ASCII có tên thiết bị của máy in liên quan. Việc liệt kê các máy in bằng các cấu trúc `PRINTER_INFO_5` được thực hiện nhanh vì không đòi hỏi các lời gọi xa; toàn bộ thông tin cần thiết cần làm đầy bộ đệm lấy được từ bản đăng ký trong OS. Để liệt kê nhanh các máy in trong Windows NT 3.51 và các phiên bản sau, thì phải sử dụng cấu trúc `PRINTER_INFO_4`.

Chương trình 8.3 Liệt kê các máy in

```
#include <winpool.h>
...
DWORD dwSize, dwPrinters;
::EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 5, NULL, 0, &dwSize, &dwPrinters);
BYTE* pBuffer = new BYTE[dwSize];
::EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 5, pBuffer, dwSize, &dwSize, &dwPrinters);
if (dwPrinters != 0) {
    PRINTER_INFO_5* pPmInfo = (PRINTER_INFO_5*) pBuffer;
    for (UINT i=0; i<dwPrinters; i++) {
        pComboBox->AddString (pPmInfo->pPrinterName);
        pPmInfo++;
    }
}
delete[] pBuffer;
```

Nếu máy in được chọn từ hộp *combo* và ta muốn tạo dc cho chúng thì phải chuyển tên thiết bị lưu trong cấu trúc `PRINTER_INFO_5` cho `CDC::CreateDC` như sau:

```
CString strPrinterName;
int nIndex = pComboBox->GetCurSel ();
pComboBox->GetLBText (nIndex, strPrinterName);
CDC dc;
dc.CreateDC (NULL, strPrinterName, NULL, NULL);
```

Kết quả là đối tượng `CDC` có thể được sử dụng đúng như chuyển địa chỉ của đối tượng `CDC` cho `OnBeginPrinting` và các hàm in `CView` khác.

CHƯƠNG 9

LIÊN KẾT CƠ SỞ DỮ LIỆU

Liên kết với cơ sở dữ liệu (CSDL) là một phần quan trọng của một chương trình ứng dụng. MFC cung cấp khả năng liên kết và khai thác CSDL theo mô hình dữ liệu quan hệ, gọi tắt là CSDL quan hệ. Trong chương này khi viết CSDL chúng ta cũng sẽ hiểu đây là CSDL quan hệ.

Trong mô hình dữ liệu quan hệ, dữ liệu được tổ chức thành một hoặc nhiều bảng. Mỗi hàng trong bảng chứa thông tin về một đối tượng độc lập, mỗi cột của bảng chứa thông tin về một đặc tính chung của các đối tượng. Mỗi hàng trong bảng được gọi là một bản ghi (*record*). Mỗi bản ghi chứa các thành phần dữ liệu tạo thành bản ghi. Các thành phần này được gọi là trường (*field*). Một trường là một ô trong bảng được chỉ danh bằng tiêu đề của cột. Tiêu đề của cột là tên trường.

CSDL quan hệ có thể được tạo và thao tác bằng nhiều phương pháp. Để khai thác một CSDL quan hệ ta có thể sử dụng ngôn ngữ SQL (*Structured Query Language*), được rất nhiều hệ quản trị CSDL và ngôn ngữ lập trình trợ giúp.

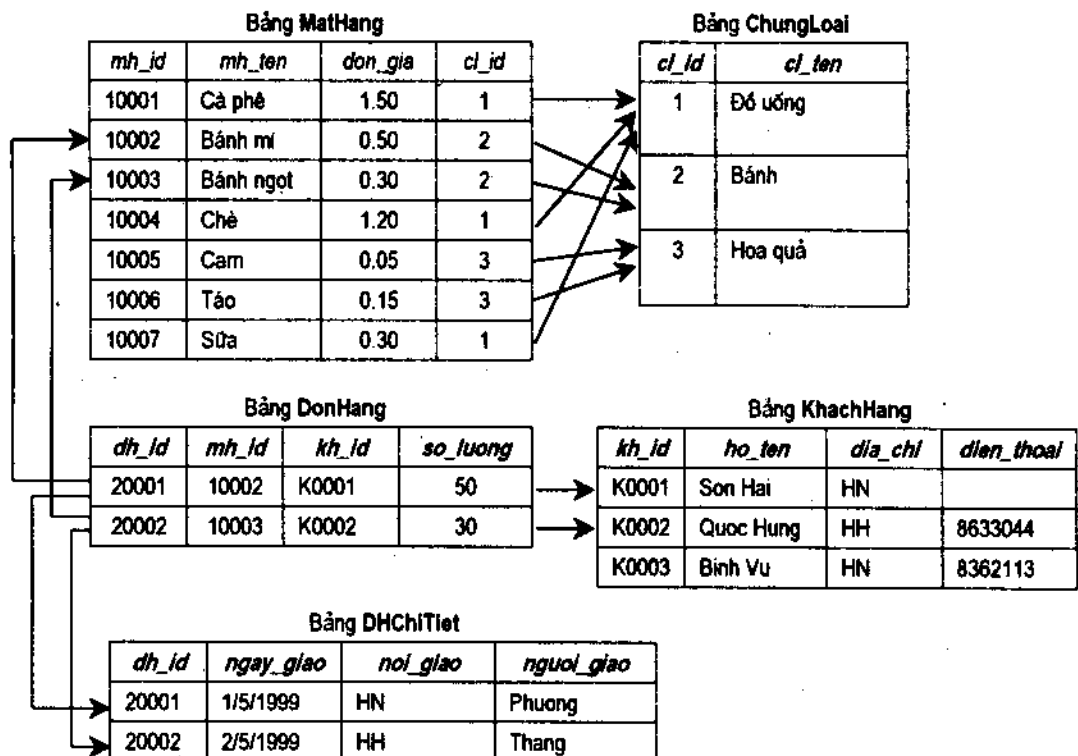
9.1. GIỚI THIỆU SQL

SQL là một ngôn ngữ quan hệ đơn giản, được thiết kế riêng cho việc thâm nhập CSDL quan hệ. MFC trợ giúp sử dụng SQL để hỏi và thực hiện các thao tác trên CSDL. SQL có các câu lệnh cho phép khai thác, sắp xếp, xóa và cập nhật các bản ghi; bổ sung hay loại bỏ các trường; tạo, kết nối và quản lý các bảng CSDL ...v.v.

Một chương trình MFC khi sử dụng SQL không cần phải viết đầy đủ tất cả các phần của câu lệnh SQL, khung làm việc sẽ hoàn thiện câu lệnh và gửi nó cho mô-đun CSDL (*database engine*) được lựa chọn sử dụng. Các câu lệnh SQL được kết thúc bằng dấu chấm phẩy (tương tự câu lệnh C++) và các từ khoá của ngôn ngữ được viết bằng chữ hoa. Chúng ta sẽ xét một số thí dụ về câu lệnh SQL và xem chúng làm việc như thế nào.

9.1.1 Khai thác dữ liệu bằng SQL

Để làm thí dụ chúng ta xây dựng một CSDL đơn giản để quản lý bán hàng gồm năm bảng như trong hình 9.1.



Hình 9.1 CSDL quản lý bán hàng

Bảng MatHang (mặt hàng) có các trường:

- mh_id**: mã mặt hàng,
- mh_ten**: tên mặt hàng,
- don_gia**: đơn giá,
- cl_id**: mã chủng loại.

Bảng ChungLoai (chủng loại mặt hàng) có các trường:

- cl_id**: mã chủng loại,
- cl_ten**: tên chủng loại.

Bảng DonHang (đơn hàng) có các trường:

- hd_id**: mã đơn hàng,

mh_id: mã mặt hàng,
kh_id: mã khách hàng,
so_luong: số lượng đặt mua.

Bảng **DHChiTiet** (đơn hàng chi tiết) có các trường:

dh_id: mã đơn hàng,
ngay_giao: ngày giao hàng,
noi_giao: nơi giao hàng,
nguoi_giao: người giao hàng.

Bảng **KhachHang** (khách hàng) có các trường:

mh_id: mã khách hàng,
ten_khach: họ và tên khách hàng,
dia_chi: địa chỉ khách hàng,
dien_thoai: số điện thoại liên hệ.

Hai bảng **MatHang** và **ChungLoai** liên kết với nhau qua trường **cl_id**.

Hai bảng **DonHang** và **MatHang** liên kết với nhau qua trường **mh_id**.

Hai bảng **DonHang** và **KhachHang** liên kết với nhau qua trường **kh_id**.

Hai bảng **DonHang** và **DHChiTiet** liên kết với nhau qua trường **dh_id**.

Để khai thác dữ liệu chúng ta sử dụng câu lệnh **SELECT**.

SELECT có thể thao tác trên một hoặc nhiều bảng trong CSDL. Kết quả thực hiện câu lệnh **SELECT** là một tập bản ghi (*recordset*) chứa thông tin đã được lọc từ các bảng dữ liệu được chỉ ra trong câu lệnh. Dữ liệu trong *recordset* được tổ chức theo dạng bảng, trong đó tên cột được đặt theo tên trường của bảng trong câu lệnh, các hàng hay các bản ghi được chọn ra từ các bảng và thoả mãn các điều kiện trong câu lệnh. Một *recordset* được tạo ra từ câu lệnh **SELECT** có thể chỉ có một bản ghi hoặc cũng có thể là rỗng (*empty*).

Thao tác khai thác đơn giản nhất là lấy ra tất cả các bản ghi từ một bảng. Thí dụ, lấy ra tất cả các bản ghi trong bảng **MatHang**:

```
SELECT * FROM MatHang;
```

Dấu sao (*) chỉ ra rằng chúng ta chọn tất cả các trường trong bảng được xác định bởi tham số theo sau từ khoá **FROM**. Thí dụ này chọn tất cả các bản ghi có trong bảng **MatHang**.

Nếu muốn lấy tất cả các bản ghi nhưng chỉ cần thông tin của một số trường trong mỗi bản ghi thì chúng ta thay dấu sao bằng tên các trường quan tâm. Tên các trường viết cách nhau bằng dấu

phẩy (,). Thí dụ:

```
SELECT mh_ten, don_gia FROM MatHang;
```

Câu lệnh trên chọn tất cả các bản ghi từ bảng MatHang nhưng chỉ quan tâm tới hai trường mh_ten và don_gia. Kết quả nhận được là một bảng có hai cột với tên là mh_ten và don_gia.

Trong câu lệnh trên, tên các trường không có dấu trắng (*space*). Trong câu lệnh SQL chuẩn, nếu tên trường có dấu trắng thì chúng phải được viết trong dấu nháy kép (""). Thí dụ:

```
SELECT "mh ten", "don gia" FROM MatHang;
```

Trong C++, câu lệnh SQL phải được viết dưới dạng xâu ký tự, tức là phải được đặt trong cặp dấu nháy kép, mà tên trường cũng đặt trong cặp dấu nháy kép cho nên sẽ dẫn đến sự nhầm lẫn. Để tránh điều này, trong C++ khi tên trường có dấu trắng, chúng ta sẽ viết tên trường trong cặp ngoặc vuông ([]), thí dụ: [mh ten] hay [don gia].

Chọn bản ghi

Không giống như trường, các bản ghi trong bảng không có tên. Cách duy nhất để chọn các bản ghi là áp dụng điều kiện chọn trên nội dung các trường trong bản ghi và khi đó chỉ các bản ghi thoả mãn điều kiện này mới được lựa chọn. Để đưa thêm điều kiện chọn, trong câu lệnh SELECT bổ sung thêm mệnh đề **WHERE**. Tham số theo sau từ khoá WHERE sẽ xác định điều kiện sử dụng để chọn bản ghi. Thí dụ, câu lệnh sau đây chọn tất cả các bản ghi thuộc chủng loại *Đồ uống* (cl_id có giá trị 1) trong bảng MatHang:

```
SELECT * FROM MatHang WHERE cl_id=1;
```

Chúng ta có thể sử dụng các toán tử so sánh <, >, <= và >= cũng như các phép toán logic AND, OR để xây dựng điều kiện chọn. Thí dụ:

```
SELECT * FROM MatHang WHERE cl_id=1 AND don_gia >= 0.50;
```

Câu lệnh này chọn ra các mặt hàng thuộc chủng loại *Đồ uống* có đơn giá lớn hơn hoặc bằng 0.50. Kết quả là chọn được hai bản ghi *Cà phê* và *Chè*.

9.1.2 Kết nối các bảng bằng SQL

Ta có thể sử dụng lệnh SELECT để kết nối các bảng. Thí dụ:

```
SELECT * FROM MatHang, DonHang;
```

Kết quả nhận được là một bảng có tám trường và mười bốn bản ghi như trong bảng 9.1. Các bản ghi được tạo bằng cách kết hợp mỗi bản ghi trong bảng MatHang với tất cả các bản ghi trong bảng DonHang. Nếu bảng MatHang có 100 mặt hàng và bảng DonHang có 500 đơn đặt hàng thì theo cách kết nối trên ta sẽ nhận được một bảng có 50000 bản ghi.

Bảng 9.1 Bảng kết quả kết nối hai bảng MatHang và DonHang

<i>MH_ID</i>	<i>MH_TEN</i>	<i>DON_GIA</i>	<i>CL_ID</i>	<i>DH_ID</i>	<i>MH_ID</i>	<i>KH_ID</i>	<i>SO_LUONG</i>
10001	caphe	1.50	1	20001	10002	K0001	50
10001	caphe	1.50	1	20002	10003	K0002	30
10002	banh mi	0.50	2	20001	10002	K0001	50
10002	banh mi	0.50	2	20002	10003	K0002	30
10003	banh ngot	0.30	2	20001	10002	K0001	50
10003	banh ngot	0.30	2	20002	10003	K0002	30
10004	che	1.20	1	20001	10002	K0001	50
10004	che	1.20	1	20002	10003	K0002	30
10005	cam	0.05	3	20001	10002	K0001	50
10005	cam	0.05	3	20002	10003	K0002	30
10006	tao	0.15	3	20001	10002	K0001	50
10006	tao	0.15	3	20002	10003	K0002	30
10007	sua	0.30	1	20001	10002	K0001	50
10007	sua	0.30	1	20002	10003	K0002	30

Để việc kết nối có hiệu quả, chúng ta cần bổ sung mệnh đề WHERE vào lệnh SELECT để chọn ra các bản ghi quan tâm. Thí dụ:

```
SELECT * FROM MatHang, DonHang WHERE MatHang.mh_id = DonHang.mh_id;
```

Kết quả nhận được là một bảng cho biết mỗi khách hàng sẽ mua mặt hàng nào như bảng 9.2.

Bảng 9.2 Kết quả kết nối có điều kiện: MatHang.mh_id = DonHang.mh_id

<i>MH_ID</i>	<i>MH_TEN</i>	<i>DON_GIA</i>	<i>CL_ID</i>	<i>DH_ID</i>	<i>MH_ID</i>	<i>KH_ID</i>	<i>SO_LUONG</i>
10002	banh mi	0.50	2	20001	10002	K0001	50
10003	banh ngot	0.30	2	20002	10003	K0002	30

Trong bảng kết quả kết nối ta thấy có nhiều trường không cần thiết, thí dụ có hai trường thể hiện mã của mặt hàng MH_ID (một của bảng MatHang, một của bảng DonHang). Để loại bỏ các trường không cần thiết hãy thay dấu * trong câu lệnh SELECT bằng danh sách các trường cần quan tâm. Thí dụ:

```
SELECT mh_ten, don_gia, kh_id, so_luong FROM MatHang, DonHang
WHERE MatHang.mh_id = DonHang.mh_id;
```

9.1.3 Sắp xếp các bản ghi

Khi khai thác dữ liệu trong CSDL bằng lệnh **SELECT** ta thường muốn kết quả sẽ được sắp xếp theo trật tự nhất định. Việc sắp xếp theo cách nào tùy thuộc vào từng yêu cầu cụ thể. Thí dụ danh sách các mặt hàng có khi cần được sắp xếp theo mã mặt hàng (*mh_id*), khi khác lại phải được sắp xếp theo chủng loại mặt hàng (*cl_id*).

Việc bổ sung mệnh đề **ORDER BY** vào lệnh **SELECT** giúp chúng ta có được kết quả sắp xếp theo yêu cầu. Thí dụ:

```
SELECT mh_ten, don_gia, kh_id, so_luong FROM MatHang, DonHang
WHERE MatHang.mh_id = DonHang.mh_id ORDER BY kh_id;
```

Kết quả thực hiện câu lệnh trên là một bảng có các bản ghi giống như kết quả thực hiện lệnh kết nối có điều kiện trong thí dụ cuối cùng nhưng với các bản ghi được sắp xếp theo thứ tự tăng dần của mã khách hàng.

Nếu chúng ta muốn sắp xếp theo hai trường *kh_id* và *mh_id* và các bản ghi sắp xếp theo thứ tự giảm dần thì câu lệnh **SELECT** sẽ như sau:

```
SELECT mh_ten, don_gia, kh_id, so_luong FROM MatHang, DonHang
WHERE MatHang.mh_id = DonHang.mh_id ORDER BY kh_id DESC, mh_id DESC;
```

Từ khoá **DESC** ở cuối mỗi trường trong mệnh đề **ORDER BY** chỉ ra việc sắp xếp sẽ được thực hiện theo thứ tự giảm dần. Từ khoá **ASC** xác định thứ tự sắp xếp tăng dần, tuy nhiên chúng ta thường thấy vắng từ khoá này trong câu lệnh vì việc sắp xếp mặc định là theo thứ tự tăng dần.

Trên đây là giới thiệu sơ bộ về SQL nhưng đủ để chúng ta có thể khai thác được CSDL.

9.2. TRỢ GIÚP CƠ SỞ DỮ LIỆU CỦA MFC

MFC cung cấp hai cách tiếp cận cho việc phát triển một ứng dụng CSDL. Mỗi cách tiếp cận sử dụng tập hợp các lớp riêng của mình.

Cách tiếp cận thứ nhất là sử dụng DAO (*Data Access Objects*). Các đối tượng này cung cấp giao diện cho mô-đen CSDL Jet (*Jet Database Engine*). Mô-đen CSDL Jet là một phần tách rời của chương trình, cung cấp khả năng cập nhật và khai thác dữ liệu trong các hệ quản trị CSDL. Jet là công cụ được hệ quản trị CSDL Access của Microsoft sử dụng. Jet đảm nhận các công việc khó khăn khi làm việc với CSDL trong Access. Jet được tối ưu cho việc thâm nhập trực tiếp các tệp CSDL Access (.mdb), nhưng nó cũng có khả năng gắn với các CSDL trợ giúp giao diện ODBC (*Open DataBase Connectivity*). Điều này cho phép chúng ta thao tác với các CSDL ở các khuôn dạng khác mà có trình điều khiển ODBC (*ODBC driver*) cho chúng. Những CSDL có thể thâm nhập bằng Jet, ngoài

Microsoft Access, bao gồm *Oracle*, *dBase 5*, *Btrieve 6.0* và *FoxPro*.

Cách tiếp cận thứ hai là sử dụng ODBC. Giao diện ODBC cung cấp trình điều khiển thâm nhập các tệp .mdb, đồng thời chúng cũng cung cấp các trình điều khiển để thâm nhập các tệp có khuôn dạng CSDL khác như cách tiếp cận trên.

Lựa chọn cách tiếp cận nào cho ứng dụng của mình là điều các lập trình viên cần cân nhắc. Nếu làm việc với CSDL trong môi trường khách/chủ (*Client/Server*) thì nên chọn cách tiếp cận thứ hai. Nếu làm việc với CSDL Access và không trong môi trường khách/chủ thì cách tiếp cận thứ nhất có hiệu quả hơn. Với các CSDL không sử dụng mô-đen *Jet* để điều khiển mà sử dụng cách tiếp cận DAO thì chúng ta phải làm việc thông qua giao diện ODBC được bổ sung trong DAO. Trong trường hợp này, cách tiếp cận DAO sẽ không hiệu quả bằng việc sử dụng trực tiếp cách tiếp cận ODBC.

Nếu chương trình chỉ cần duyệt CSDL, lập trình viên có thể quyết định trên cơ sở: nếu định sử dụng CSDL *Microsoft Access* và không làm việc trong môi trường khách/chủ thì nên sử dụng DAO, trong các trường hợp khác nên dùng ODBC.

9.2.1 DAO và ODBC

DAO sử dụng các đối tượng để thâm nhập và thao tác với CSDL. Chúng có các đối tượng để biểu diễn các bảng, các câu hỏi truy vấn và bản thân CSDL. Các đối tượng này giúp lập trình viên không phải quan tâm đến các thao tác chi tiết khi thao tác với một CSDL cụ thể, chúng cũng cung cấp các giao diện lập trình phù hợp với phương pháp lập trình hướng đối tượng.

ODBC là một hệ thống giao diện độc lập cho môi trường CSDL, môi trường này đòi hỏi có trình điều khiển ODBC cung cấp cho mỗi hệ thống CSDL cần khai thác. ODBC định nghĩa một tập các hàm cho các thao tác CSDL.

Mặc dù khái niệm ODBC hoàn toàn khác với DAO nhưng cách tiếp cận lập trình trong Visual C++ lại hoàn toàn giống nhau cho cả hai phương pháp. MFC gộp giao diện ODBC trong tập các lớp tương tự như các lớp cung cấp cho DAO.

Chúng ta không đi chi tiết vào tất cả các đặc điểm của DAO và ODBC mà chỉ xem xét cách sử dụng chúng thông qua việc xây dựng một ứng dụng cụ thể.

9.2.2 Các lớp trợ giúp cách tiếp cận DAO

Bảng 9.3 liệt kê tám lớp được sử dụng trong cách tiếp cận DAO. Các lớp cơ bản nhất của DAO trong MFC là *CDaoDatabase*, *CDaoRecordset* và *CDaoRecordView*. Lớp *CDaoDatabase* miêu tả cho CSDL. Một hoặc một vài lớp phái sinh từ lớp *CDaoRecordset* thực hiện lệnh *SELECT* trên

CSDL. Lớp `CDaoRecordView` hiển thị dữ liệu được tạo ra bởi các lớp phái sinh của `CDaoRecordset`.

Chúng ta sẽ thấy một ứng dụng ODBC cũng bao gồm tập các lớp cơ sở với các hàm tương tự trên. Các lớp `CDaoTableDef` và `CDaoQueryDef` cung cấp thêm khả năng mà ODBC không có.

Bảng 9.3 Các lớp sử dụng trong cách tiếp cận DAO

<i>Lớp</i>	<i>Chức năng</i>
<code>CDaoWorkspace</code>	Đối tượng của lớp này quản lý phiên giao dịch với CSDL từ lúc bắt đầu cho đến khi kết thúc. Đối tượng <code>CDaoDatabase</code> cần một đối tượng <code>CDaoWorkspace</code> đã sẵn sàng, nếu lập trình viên không tạo ra đối tượng <code>CDaoWorkspace</code> thì khung ứng dụng sẽ tự động tạo ra đối tượng này khi đối tượng <code>CDaoDatabase</code> được khởi tạo. Một đối tượng <i>workspace</i> có thể chứa vài đối tượng <i>database</i> .
<code>CDaoDatabase</code>	Đối tượng của lớp này thực hiện việc kết nối với một CSDL cụ thể. Đối tượng này luôn được tạo ra khi chúng ta thăm nhập CSDL, nhưng chúng ta cũng không cần phải tạo đối tượng này tường minh. Nó có thể được tạo ngầm khi chúng ta tạo đối tượng <code>CDaoRecordset</code> .
<code>CDaoRecordset</code>	Đối tượng phái sinh từ lớp này thể hiện kết quả của việc thực hiện câu lệnh <code>SELECT</code> của SQL, đây là một tập các bản ghi.
<code>CDaoRecordView</code>	Đối tượng phái sinh từ lớp này được sử dụng để hiển thị bản ghi hiện thời trong đối tượng <i>recordset</i> liên hệ với nó. Đối tượng này sử dụng một hộp thoại con để hiển thị dữ liệu. Có các cơ chế tự động cho việc cập nhật các điều khiển trong hộp thoại với dữ liệu hiện thời của đối tượng <i>recordset</i> .
<code>CDaoFieldExchange</code>	Lớp này cung cấp khả năng trao đổi dữ liệu giữa CSDL và đối tượng <i>recordset</i> . Lập trình viên có thể tự mình quản lý và sử dụng các đối tượng của lớp này, nhưng <i>AppWizard</i> và <i>ClassWizard</i> cũng tự động cài đặt và quản lý việc sử dụng chúng.
<code>CDaoQueryDef</code>	Đối tượng của lớp này định nghĩa câu hỏi trên CSDL, điều thường được định nghĩa trước trong CSDL. Đây là các câu hỏi chuẩn được sử dụng thường xuyên trong một CSDL đặc biệt. Một đối tượng <code>CDaoQueryDef</code> có thể được sử dụng để tạo ra đối tượng <code>CDaoRecordset</code> để miêu tả một câu lệnh <code>SELECT</code> chi tiết. Đối tượng của lớp này cũng được sử dụng để thực hiện tường minh một lệnh SQL bằng việc sử dụng hàm thành phần <code>Execute</code> .
<code>CDaoTableDef</code>	Đối tượng của lớp này định nghĩa một bảng trong CSDL. Nó có thể trình bày một bảng đã có hoặc có thể được sử dụng để tạo một bảng mới.
<code>CDaoException</code>	Đối tượng của lớp này được khởi tạo khi một điều kiện ngoại lệ xuất hiện từ các thao tác trên CSDL DAO. Các thành phần của lớp <code>CDaoException</code> cho phép lập trình viên khám phá nguyên nhân của trường hợp ngoại lệ.

9.2.2 Các lớp trợ giúp cách tiếp cận ODBC

MFC trợ giúp cách tiếp cận ODBC thông qua năm lớp được liệt kê trong bảng 9.4.

Các lớp của ODBC nhìn rất giống các lớp của DAO, giao diện do chúng cung cấp cũng tương tự như các lớp của DAO, nhưng việc xử lý bên trong cho việc thăm nhập CSDL thì có phần khác nhau.

Cách tốt nhất để hiểu các thao tác CSDL thực hiện như thế nào trong MFC là làm việc thông qua một thí dụ. Trong thí dụ dưới đây, ta sẽ sử dụng cách tiếp cận ODBC, áp dụng để thăm nhập CSDL *Microsoft Access*.

Bảng 9.4 Các lớp sử dụng trong cách tiếp cận ODBC

<i>Lớp</i>	<i>Chức năng</i>
CDatabase	Đối tượng của lớp này miêu tả kết nối với CSDL. Kết nối này phải tồn tại trước khi chúng ta thực hiện các thao tác trên CSDL. CSDL ODBC không cần sử dụng lớp workspace.
CRecordset	Đối tượng phái sinh từ lớp này được sử dụng để trình bày kết quả của việc thực hiện câu lệnh SELECT. Lớp này tương tự như lớp CDaoRecordset.
CRecordView	Đối tượng phái sinh từ lớp này được sử dụng để hiển thị thông tin hiện thời từ đối tượng recordset liên kết với nó. Lớp này tương tự như lớp CDaoRecordView.
CFieldExchange	Lớp này cung cấp cho việc trao đổi dữ liệu giữa CSDL và đối tượng recordset. Cách thức tương tự như đối tượng CDaoExchange.
CDBException	Đối tượng của lớp này miêu tả các trường hợp ngoại lệ xuất hiện trong các thao tác trên CSDL ODBC.

9.3. TẠO ỨNG DỤNG CƠ SỞ DỮ LIỆU

Chúng ta sẽ sử dụng CSDL như trong phần 9.1 để làm thí dụ. Các bảng sẽ đặt trong tệp **sampdata.mdb**. Tệp **sampdata.mdb** sẽ được tạo ra bằng chương trình *Microsoft Access*.

Trong bước thứ nhất một chương trình hiển thị các bản ghi trong bảng **MatHang** sẽ được tạo ra. Sau đó chúng ta sẽ bổ sung các mã trình để khai thác các đơn hàng. Trước khi bắt đầu, chúng ta phải đăng ký CSDL với hệ điều hành.

9.3.1 Đăng ký cơ sở dữ liệu ODBC

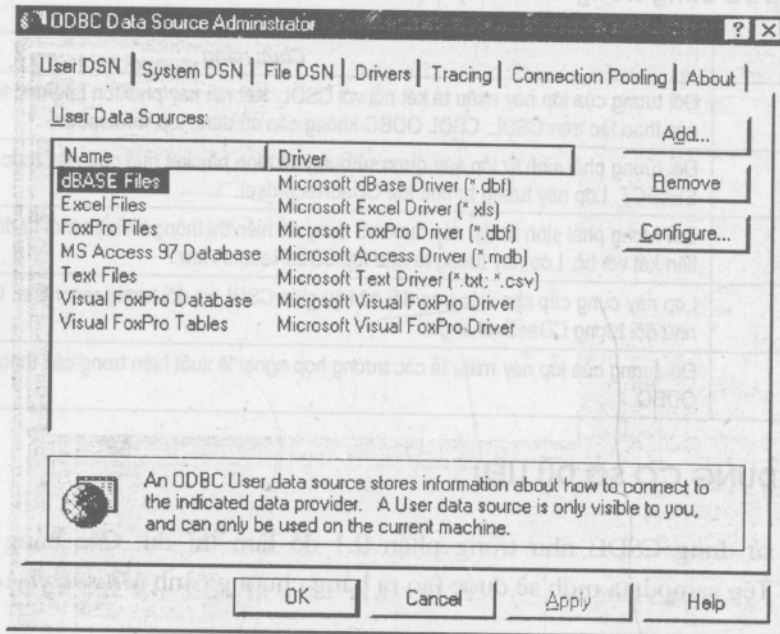
Trước khi có thể sử dụng ODBC cho CSDL thì nó cần phải được đăng ký với hệ điều hành. Việc đăng ký được thực hiện theo các bước sau:

Mở cửa sổ *Control Panel* (Start → Settings → Control Panel)

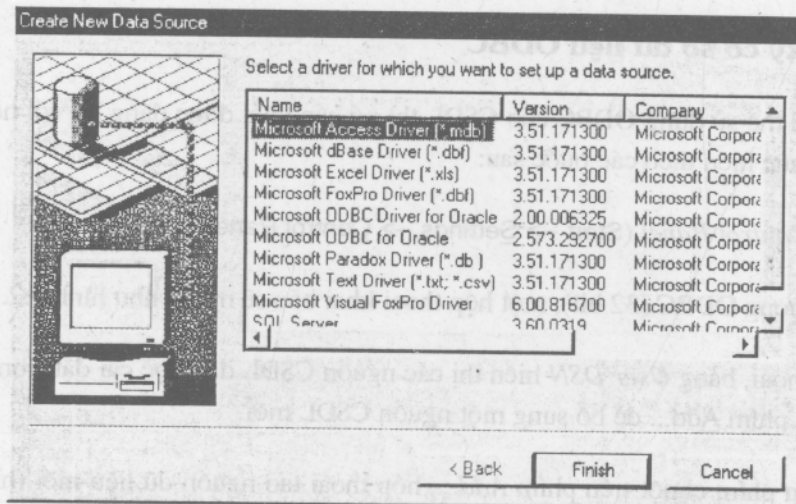
Chọn biểu tượng ODBC (32 bit), một hộp thoại khai báo sẽ mở ra như hình 9.2.

Trong hộp thoại, bảng *User DSN* hiển thị các nguồn CSDL đã được cài đặt trong hệ thống. Nhấn phím chuột vào phím **Add...** để bổ sung một nguồn CSDL mới.

Sau khi nhấn phím chuột trên phím **Add...**, hộp thoại tạo nguồn dữ liệu mới (hình 9.3) cho phép chọn trình điều khiển cho CSDL ta xây dựng. Ở đây chúng ta sẽ chọn *Microsoft Access Driver (*.mdb)*. Nếu chúng ta không nhìn thấy trình điều khiển này thì phải quay lại bảng đặc tính *Cài đặt Windows (Windows Setup)* để cài đặt chúng.

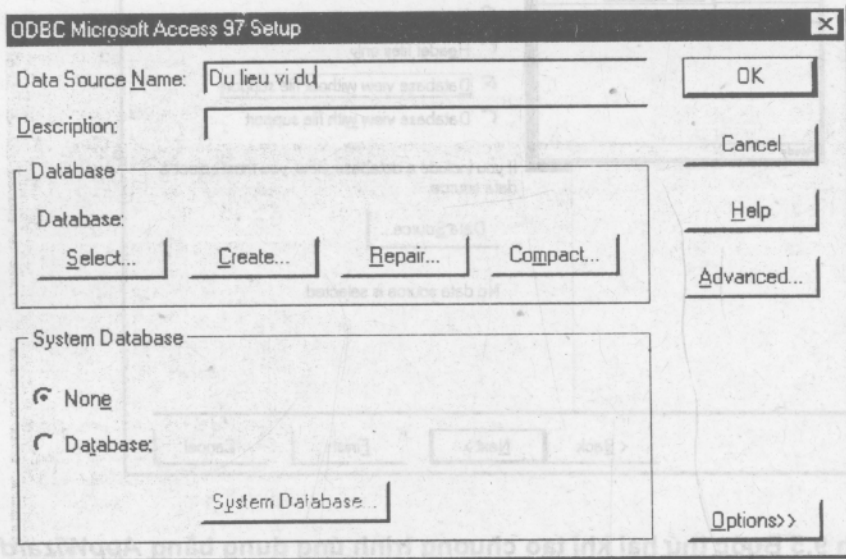


Hình 9.2 Hộp thoại quản lý nguồn CSDL của ODBC



Hình 9.3 Hộp thoại tạo nguồn dữ liệu mới

Sau khi chọn trình điều khiển hãy nhấn phím chuột trên phím Finish. Một hộp thoại mới mở ra để yêu cầu người sử dụng khai báo các thông số của CSDL như trên hình 9.4.



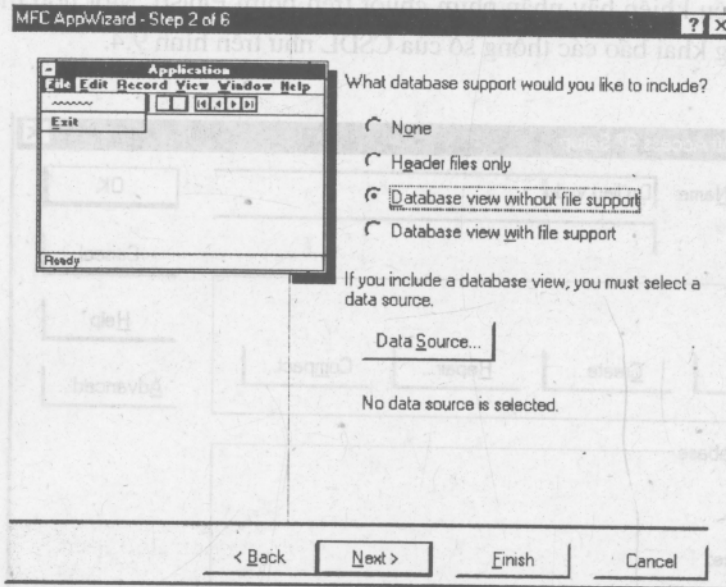
Hình 9.4 Hộp thoại cài đặt ODBC Microsoft Access

Tại hộp soạn thảo Data Source Name hãy nhập xâu **Du lieu vi du**, chúng sẽ được sử dụng để chỉ danh cho CSDL khi chúng ta tạo chương trình bằng công cụ *AppWizard*. Bây giờ chúng ta cần nhấn phím chuột trên phím Select... để đi đến bước cuối cùng. Bước cuối cùng hiển thị hộp thoại chọn tệp thông thường của Windows, tại đây ta chọn tệp **sampdata.mdb** và kết thúc bằng phím OK.

9.3.2 Sử dụng AppWizard để tạo chương trình ODBC

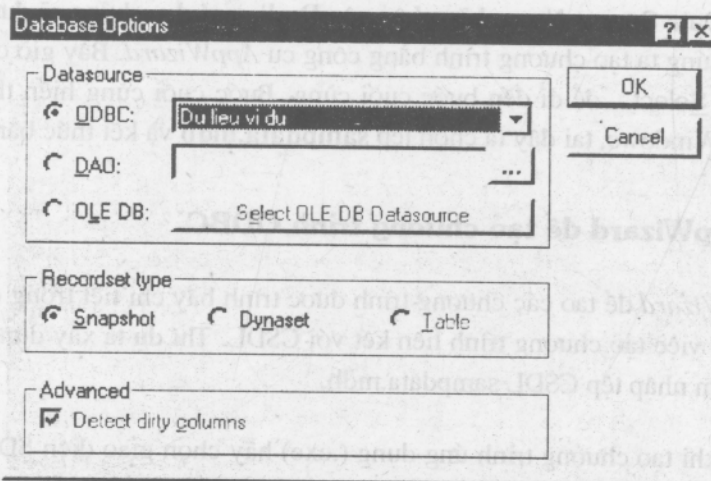
Việc sử dụng *AppWizard* để tạo các chương trình được trình bày chi tiết trong chương 13. Ở đây chúng ta tập trung vào việc tạo chương trình liên kết với CSDL. Thí dụ ta xây dựng chương trình có tên **DBSample** để thăm nhập tệp CSDL **sampdata.mdb**.

Tại bước thứ nhất khi tạo chương trình ứng dụng (.exe) hãy chọn giao diện SDI. Trong bước thứ hai ta chọn *Database view without file support* bằng việc đánh dấu vào lựa chọn này, lúc này phím Data Source... trở thành sẵn sàng như hình 9.5. Chúng ta không cần chọn *Database view file support* vì chương trình không cần mở các tệp bên ngoài. Chúng ta cần nhấn phím Data Source để chỉ ra CSDL mà chương trình sẽ sử dụng.



Hình 9.5 Bước thứ hai khi tạo chương trình ứng dụng bằng AppWizard

Hộp thoại xác định CSDL cho chương trình khi nhấn phím Data Source... xuất hiện trên màn hình như hình 9.6.



Hình 9.6 Xác định cơ sở dữ liệu

Tại đây ta chọn chức năng ODBC và chọn *Du lieu vi du* trong hộp *combo*. AppWizard tự động xen vào chương trình lớp CRecordset và CRecordView hoặc CDaoRecordset và CDaoRecordView nếu ta lựa chọn DAO. Chúng ta cũng cần lựa chọn kiểu cho đối tượng *recordset* là *Snapshot* hay

Dynaset bằng phím *radio* trong hộp thoại, nếu ta sử dụng DAO thì sẽ có thêm lựa chọn *Table*.

Kiểu *Snapshot* và *Dynaset*

Đối tượng *recordset* cung cấp kết quả thực hiện lệnh *SELECT* trên CSDL. Trong trường hợp sử dụng kiểu *Snapshot*, truy vấn sẽ được thực hiện một lần và kết quả được chứa trong bộ nhớ. Đối tượng *recordset* cho phép chúng ta lấy ra bất kỳ bản ghi nào trong bảng kết quả truy vấn. Nếu có bất kỳ sự thay đổi nào trên CSDL, thí dụ một người sử dụng khác cập nhật vào CSDL, đều không làm thay đổi dữ liệu đã có trong *recordset*. Nếu cần cập nhật dữ liệu cho *recordset*, chúng ta phải gọi thực hiện lại câu lệnh *SELECT*.

Đối tượng *recordset* với kiểu *snapshot* có một đặc trưng phụ thuộc vào việc sử dụng DAO hay ODBC:

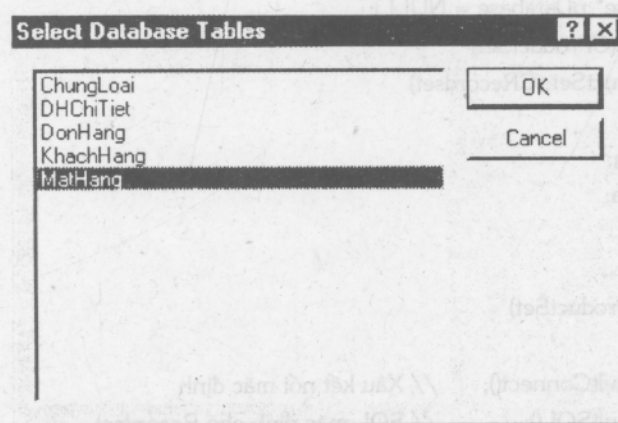
- Khi sử dụng DAO, đối tượng *recordset* chỉ có thể đọc CSDL.
- Khi sử dụng ODBC, đối tượng *recordset* có thể đọc/ghi CSDL.

Khi sử dụng kiểu *Dynaset*, đối tượng *recordset* sẽ tự động cập nhật bản ghi hiện thời trong CSDL khi chúng ta chuyển từ bản ghi này tới bản ghi khác trong bảng kết quả truy vấn cho *recordset*. Một *recordset* với kiểu *Dynaset* sẽ sử dụng một chỉ số (*index*) tới các bảng CSDL để phát sinh động nội dung cho mỗi bản ghi.

Recordset với kiểu *Snapshot* sẽ được chọn cho thí dụ của chúng ta.

Chọn bảng cơ sở dữ liệu

Sau khi chọn kiểu *Snapshot* và nhấn phím *OK*, ta đi đến hộp thoại chọn bảng cho *recordset* như hình 9.7.



Hình 9.7 Hộp thoại chọn bảng CSDL

Bảng được chọn là tham số cho lệnh SELECT sẽ cung cấp cho *recordset*. Trong hộp thoại liệt kê tất cả các bảng có trong CSDL *Du lieu vi du*, ta chọn bảng *MatHang* và kết thúc bằng phím OK. Dữ liệu cung cấp cho *recordset* là kết quả của lệnh:

```
SELECT * FROM MatHang;
```

Chúng ta chuyển qua các bước tiếp theo để đi đến bước cuối cùng của việc tạo chương trình bằng *AppWizard*. Tại đây, hãy đổi tên lớp *CDBSampleView* thành *CProductView* và *CDBSampleSet* thành *CProductSet*. Kết thúc việc tạo chương trình bằng phím *Finish*.

9.3.3 Tìm hiểu cấu trúc chương trình

Cấu trúc cơ bản của chương trình đã được trình bày trong các chương trước, chúng bao gồm lớp *CDBSampleApp*, lớp cửa sổ khung *CMainFrame*, lớp tài liệu *CDBSampleDoc* và lớp quan sát *CProductView*. Một đối tượng tài liệu mẫu chịu trách nhiệm tạo và liên kết cửa sổ khung, đối tượng tài liệu và đối tượng quan sát. Điều này được thực hiện trong hàm thành phần *InitInstance* của đối tượng ứng dụng.

Đối tượng tài liệu không có gì đặc biệt ngoại trừ có thêm một thành phần dữ liệu *m_producSet*, đây là đối tượng của lớp *CProductSet*. Đối tượng này được tự động tạo ra khi khởi tạo đối tượng tài liệu trong hàm thành phần *InitInstance* của đối tượng ứng dụng.

Tìm hiểu Recordset

Lớp *CProductSet* được suy diễn từ lớp *CRecordset* và khai báo của chúng như sau:

```
class CProductSet : public CRecordset
{
public:
    CProductSet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(CProductSet)
    //{{AFX_FIELD(CProductSet, CRecordset)
    CString    m_mh_id;
    CString    m_mh_ten;
    CString    m_don_gia;
    int m_cl_id;
    //}}AFX_FIELD
    //{{AFX_VIRTUAL(CProductSet)
public:
    virtual CString GetDefaultConnect();    // Xâu kết nối mặc định
    virtual CString GetDefaultSQL();        // SQL mặc định cho Recordset
    virtual void DoFieldExchange(CFieldExchange* pFX);    // trợ giúp RFX
    //}}AFX_VIRTUAL
};
```

Lớp `CRecordset` cung cấp các hàm để khai thác dữ liệu trong CSDL. Cấu tử của lớp `CRecordset` có tham số là một con trỏ tới đối tượng `CDatabase`, trong trường hợp mặc định con trỏ này nhận giá trị `NULL`. Xác định tham số cho cấu tử cho phép đối tượng `CProductSet` được tạo ra từ đối tượng có sẵn `CDatabase`, điều này cho phép một liên kết CSDL được sử dụng lại. Việc tạo liên kết mới với CSDL tốn nhiều công sức nên việc sử dụng lại một liên kết đã có sẽ rất tiện lợi.

Nếu không có con trỏ nào chuyển cho cấu tử thì khung làm việc sẽ tự động tạo một đối tượng `CDatabase` và gọi hàm thành phần `GetDefaultConnect` của lớp `CProductSet` để định nghĩa một liên kết. Hàm `GetDefaultConnect` do *AppWizard* tạo ra có nội dung như sau:

```
CString CProductSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=Du lieu vi du");
}
```

Đây là một hàm ảo thuần túy trong lớp cơ sở, do đó nó phải luôn được định nghĩa trong lớp phái sinh từ lớp `CRecordset`. Ở đây, hàm trả về một chuỗi ký tự, chuỗi này được gửi lại tới khung làm việc để làm chỉ danh cho CSDL và cho phép khung làm việc tạo một đối tượng `CDatabase` để tự động liên kết CSDL.

Trong thực tế, thông thường cần cung cấp thêm ID người sử dụng và mật khẩu (*password*) để được phép thâm nhập vào CSDL. Chúng ta có thể bổ sung thêm các thông tin này trong chuỗi trả về của hàm `GetDefaultConnect`. Thí dụ, thêm thông tin ID người sử dụng là *nshai* và mật khẩu là *popmap*:

```
return _T("ODBC;DSN=Du lieu vi du;UID=nshai;PWD=popmap");
```

Chúng ta cũng có thể để người sử dụng tự chọn tên CSDL từ các nguồn dữ liệu đã đăng ký với hệ thống thông qua hộp thoại của khung làm việc bằng cách viết lệnh *return* như sau:

```
return _T("ODBC;");
```

Lớp `CProductSet` có thêm các thành phần dữ liệu cho mỗi trường của bảng *MatHang*. Đó là các thành phần `m_mh_id`, `m_mh_ten`, `m_don_gia` và `m_cl_id` tương ứng với các trường `mh_id`, `mh_ten`, `don_gia` và `cl_id`. Kiểu của các thành phần này phụ thuộc vào kiểu dữ liệu của trường tương ứng. Nếu muốn loại bỏ bớt một số trường không quan tâm hãy dùng công cụ *ClassWizard* để xóa chúng. Chúng ta không được xóa các trường là trường khoá của bảng. Nếu các trường khoá bị xóa thì *recordset* sẽ không làm việc.

Câu lệnh SQL cung cấp cho *recordset* nơi chứa dữ liệu cho các thành phần nói trên được chỉ ra trong hàm `GetDefaultSQL` như sau:

```
CString CProductSet::GetDefaultSQL()
{
```

```

    return _T("[MatHang]");
}

```

Nếu ta chọn một vài bảng trong bước thứ hai khi tạo ứng dụng thì các bảng sẽ được khai báo thêm ở đây, tên các bảng sẽ nằm trong cặp ngoặc vuông ([]) và viết cách nhau bởi dấu phẩy. Thí dụ:

```

CString CProductSet::GetDefaultSQL()
{
    return _T("[MatHang].[ChungLoai]");
}

```

Hàm `GetDefaultSQL` được khung làm việc gọi khi khởi tạo câu lệnh SQL để cung cấp cho *recordset*. Khung làm việc đặt xâu trả về của hàm `GetDefaultSQL` vào trong câu lệnh `SELECT` theo mẫu sau:

```
SELECT * FROM <xâu trả về của hàm GetDefaultSQL>;
```

Phần sau chúng ta sẽ xem cách đưa thêm các mệnh đề `WHERE` và `GROUP BY` vào câu lệnh để lọc và sắp xếp các bản ghi.

Việc chuyển dữ liệu từ CSDL vào *recordset* và ngược lại được thực hiện bởi hàm thành phần `DoFieldExchange` của lớp `CProductSet`. Mã trình của hàm này do *AppWizard* tạo ra như sau:

```

void CProductSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(CProductSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[mh_id]"), m_mh_id);
    RFX_Text(pFX, _T("[mh_ten]"), m_mh_ten);
    RFX_Text(pFX, _T("[don_gia]"), m_don_gia);
    RFX_Int(pFX, _T("[cl_id]"), m_cl_id);
    //}}AFX_FIELD_MAP
}

```

Hàm này được gọi tự động bởi khung làm việc để cập nhật và khai thác dữ liệu trong CSDL. Nó làm việc tương tự như hàm `DoDataExchange` trong điều khiển hộp thoại. Tham số `pFX` xác định khi nào đọc/ghi dữ liệu. Mỗi khi hàm được gọi, nó chuyển một bản ghi tới/từ đối tượng *recordset*.

Trong thân hàm `DoFieldExchange`, hàm đầu tiên được gọi là `SetFieldType`, nó đặt cách thức cho các lời gọi hàm `RFX_` sau đó. Trong trường hợp này, cách thức được chỉ định như `outputColumn`, điều này chỉ ra rằng dữ liệu được trao đổi giữa trường và tham số tương ứng xác định trong mỗi lời gọi hàm `RFX_` sau đó.

Có một loạt các hàm `RFX_` cho các kiểu dữ liệu khác nhau của trường. Khi gọi hàm cho một trường, ta phải chọn hàm cùng kiểu với kiểu dữ liệu của chúng, thí dụ trường có kiểu xâu ký tự thì

gọi hàm **RFX_Text**, trường có kiểu *int* thì gọi hàm **RFX_Int...** Tham số thứ nhất cung cấp cho hàm **RFX_** là con trỏ đối tượng **pFX** xác định hướng chuyển động của dữ liệu (từ/đến *recordset*). Tham số thứ hai là tên bảng và tham số thứ ba là thành phần dữ liệu để chứa dữ liệu của trường trong bản ghi hiện thời.

Tim hiểu RecordView

Mục đích của lớp quan sát là để hiển thị thông tin từ đối tượng *recordset* trong cửa sổ ứng dụng, do đó chúng ta cần hiểu chúng làm việc như thế nào. Lớp **CProductView** do *AppWizard* tạo ra như sau:

```
class CProductView : public CRecordView
{
protected: // create from serialization only
    CProductView();
    DECLARE_DYNCREATE(CProductView)
public:
    //{{AFX_DATA(CProductView)
    enum{ IDD = IDD_DBSAMPLE_FORM };
    CProductSet* m_pSet;
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA
// Attributes
public:
    CDBSampleDoc* GetDocument();
// Operations
public:
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CProductView)
public:
    virtual CRecordset* OnGetRecordset();
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    virtual void OnInitialUpdate(); // called first time after construct
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL
// Implementation
public:
```

```

    virtual ~CProductView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Generated message map functions
protected:
    //{{AFX_MSG(CProductView)
    // NOTE - the ClassWizard will add and remove member functions here.
    // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

```

Lớp quan sát cho một *recordset* được suy diễn từ lớp CRecordView bởi vì chúng có nhiệm vụ hiển thị các trường mong muốn từ *recordset*. Lớp CRecordView bao gồm tất cả các hàm cần thiết để quản lý việc liên lạc với *recordset*. Chúng ta chỉ cần sử dụng *ClassWizard* để sửa đổi hoặc bổ sung thêm vào lớp quan sát cho phù hợp với ứng dụng.

Trong khối nằm giữa dòng giải thích AFX_DATA, kiểu *enum* bổ sung IDD_DBSAMPLE_FORM cho lớp. Đây là ID của hộp thoại rỗng mà *AppWizard* đã thêm vào trong chương trình. Chúng ta cần bổ sung thêm các điều khiển cho hộp thoại này để hiển thị dữ liệu trong các trường ta muốn xem của bảng MatHang. ID của hộp thoại được chuyển cho lớp cơ sở CRecordView trong danh sách đối của cấu tử:

```

CProductView::CProductView() : CRecordView(CProductView::IDD)
{ //{{AFX_DATA_INIT(CProductView)
    m_pSet = NULL;
    //}}AFX_DATA_INIT
}

```

Hành động này liên kết lớp quan sát với hộp thoại, điều đó là cần thiết để cơ chế chuyển dữ liệu giữa đối tượng *recordset* và đối tượng quan sát làm việc.

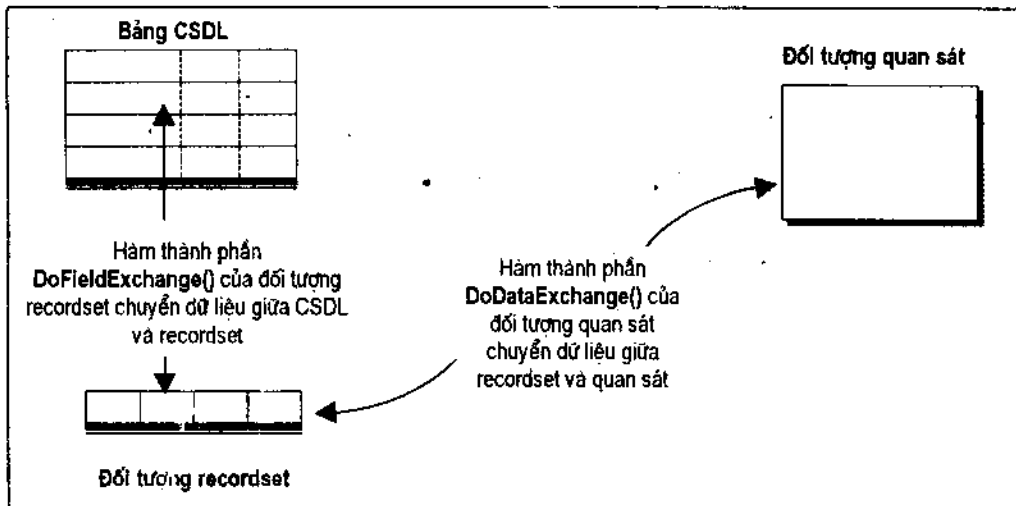
m_pSet là con trỏ tới đối tượng CProductSet, nó cũng được định nghĩa trong khối AFX_DATA. m_pSet được gán giá trị NULL khi khởi tạo trong cấu tử của lớp CProductView. Một giá trị khác hiệu quả hơn được gán cho m_pSet trong hàm thành phần OnInitialUpdate:

```

void CProductView::OnInitialUpdate()
{ m_pSet = &GetDocument()->m_productSet;
  CRecordView::OnInitialUpdate();
}

```

Hàm này được gọi khi đối tượng quan sát được khởi tạo. Chúng gán địa chỉ của thành phần `m_productSet` của tài liệu cho biến `m_pSet`.



Hình 9.8 Trao đổi thông tin giữa CSDL, recordset và đối tượng quan sát

Hình 9.8 là sơ đồ trao đổi dữ liệu giữa bảng CSDL, đối tượng `CProductSet` và đối tượng `CProductView`. Việc chuyển dữ liệu giữa các thành phần dữ liệu trong đối tượng `CProductSet` và các điều khiển trong hộp thoại quan hệ với đối tượng `CProductView` được quản lý bởi hàm thành phần `DoDataExchange` của lớp `CProductView`.

Tạo hộp thoại quan sát

Trong *Developer Studio*, hãy chọn cửa sổ *ResourceView*, mở rộng tài nguyên *Dialog* và nhấn đúp chuột vào `IDD_DBSAMPLE_FORM`. Hãy xóa đối tượng *static text* có thông báo `TODO` trong hộp thoại. Bổ sung vào hộp thoại các điều khiển văn bản tĩnh (*static text*) và điều khiển soạn thảo (*edit control*) để mô tả và hiển thị dữ liệu.

The screenshot shows a dialog box with the following controls:

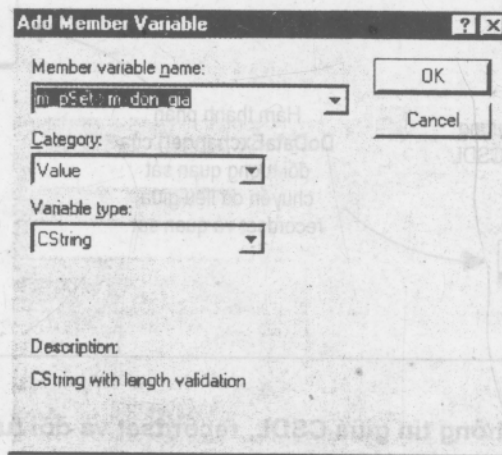
- Mã mặt hàng:** An edit control with a dropdown arrow.
- Tên mặt hàng:** An edit control.
- Đơn giá:** An edit control.
- Chủng loại:** An edit control.

Hình 9.9 Hộp thoại hiển thị dữ liệu mặt hàng

Thí dụ về bổ sung các điều khiển được trình bày trên hình 9.9. Theo thiết kế, các hộp soạn thảo theo thứ tự từ trên xuống được gán ID là IDC_MH_ID, IDC_MH_TEN, IDC_DON_GIA và IDC_CL_ID. Các ID này sẽ được sử dụng để liên kết điều khiển với dữ liệu trong *recordset*.

Liên kết các điều khiển với *recordset*

Việc liên kết điều khiển với các dữ liệu thành phần của *CProductSet* là rất đơn giản.



Hình 9.10 Hộp thoại liên kết điều khiển với dữ liệu

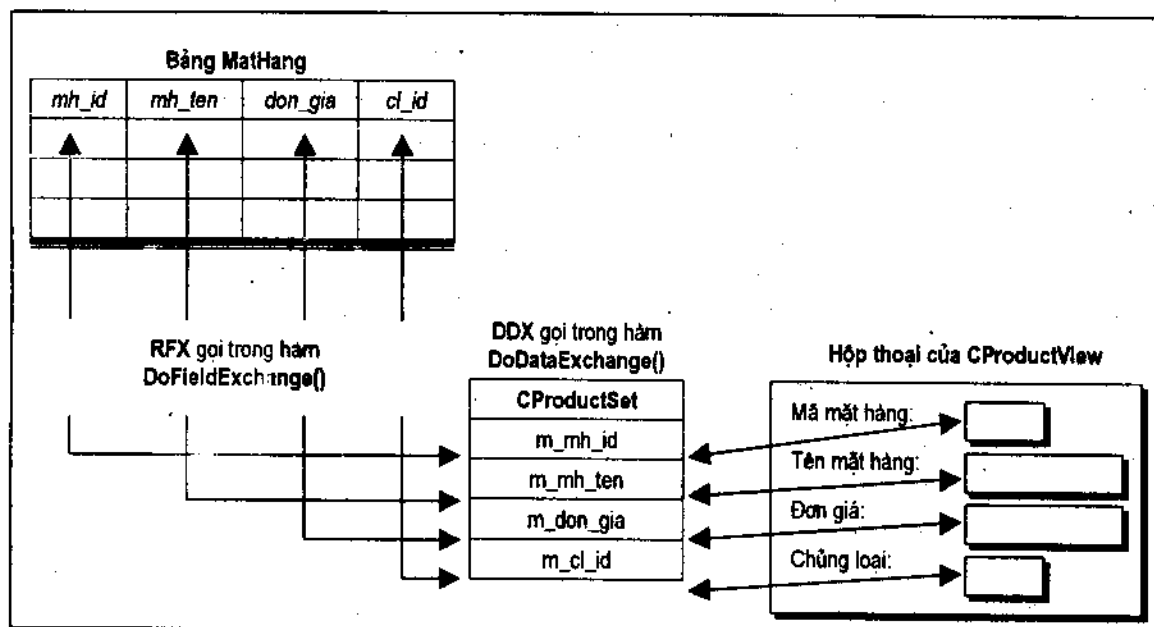
Hãy giữ phím *Ctrl* và nhấn đúp chuột vào điều khiển, khi đó một hộp thoại bổ sung thành phần mở ra như hình 9.10.

Tại đây ta sẽ chọn các thành phần dữ liệu sẽ liên kết với điều khiển. Thí dụ chúng ta chọn điều khiển IDC_MH_ID tương ứng với *m_pSet->m_mh_id*, IDC_MH_TEN tương ứng với *m_pSet->m_mh_ten*, IDC_DON_GIA tương ứng với *m_pSet->m_don_gia* và IDC_CL_ID tương ứng với *m_pSet->m_cl_id*.

Lúc này hàm *DoDataExchange* có nội dung như sau:

```
void CProductView::DoDataExchange(CDataExchange* pDX)
{
    CRecordView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CProductView)
    DDX_FieldText(pDX, IDC_MH_ID, m_pSet->m_mh_id, m_pSet);
    DDX_FieldText(pDX, IDC_MH_TEN, m_pSet->m_mh_ten, m_pSet);
    DDX_FieldText(pDX, IDC_DON_GIA, m_pSet->m_don_gia, m_pSet);
    DDX_FieldText(pDX, IDC_CL_ID, m_pSet->m_cl_id, m_pSet);
    //}}AFX_DATA_MAP
```

Đến đây chúng ta đã hoàn thành chương trình xem dữ liệu của bảng MatHang. Các bước còn lại là dịch và chạy chương trình. Hình 9.11 là sơ đồ trao đổi dữ liệu giữa CSDL và hộp thoại hiển thị dữ liệu của đối tượng CProductView.



Hình 9.11 Sơ đồ trao đổi dữ liệu giữa CSDL và hộp thoại hiển thị dữ liệu

9.3.4 Sắp xếp dữ liệu trong recordset

Như chúng ta đã biết trong phần trên, dữ liệu khai thác từ CSDL bằng *recordset* sử dụng lệnh SELECT của SQL. Lệnh này được tạo ra bởi khung làm việc khi gọi hàm thành phần GetDefaultSQL. Chúng ta có thể bổ sung thêm mệnh đề ORDER BY vào câu lệnh đã được tạo ra bằng cách gán giá trị cho thành phần dữ liệu *m_strSort* của đối tượng CProductSet. Thành phần dữ liệu *m_strSort* thừa hưởng từ CRecordSet. Đây là cách để sắp xếp dữ liệu kết quả truy vấn. Chúng ta chỉ cần gán xâu chứa tên trường cần sắp xếp cho *m_strSort*, khung làm việc sẽ bổ sung thêm từ khóa ORDER BY cho câu lệnh SELECT. Việc gán giá trị cho *m_strSort* sẽ được thực hiện ở đâu?

Việc chuyển dữ liệu giữa CSDL và *recordset* xuất hiện khi gọi hàm thành phần **Open** của đối tượng *recordset*. Trong chương trình của chúng ta, hàm thành phần **Open** của đối tượng *recordset* được gọi bởi hàm thành phần **OnInitUpdate** của lớp CProductView. Do đó chúng ta sẽ gán giá trị cho *m_strSort* trong hàm thành phần **OnInitUpdate** của lớp CProductView. Thử dụ như sau:

```
void CProductView::OnInitUpdate()
```

```

{
    m_pSet = &GetDocument()->m_productSet;
    m_pSet->m_strSort = "[mh_id]"; //xác định trường sắp xếp
    CRecordView::OnInitialUpdate();
}

```

Trong thí dụ này, chúng ta yêu cầu dữ liệu được sắp xếp theo trường *mã khách hàng*. Nếu muốn sắp xếp theo nhiều trường, bổ sung tên các trường vào xâu dữ liệu gán cho `m_strSort`, tên các trường được đặt trong cặp ngoặc vuông và cách nhau bằng dấu phẩy.

Thay đổi tiêu đề của sổ

Tiêu đề cho cửa sổ sẽ rõ ràng hơn nếu ta hiển thị tên của bảng đang được hiển thị. Chúng ta có thể làm điều này bằng việc bổ sung thêm vào đối tượng tài liệu đoạn mã trình sau:

```

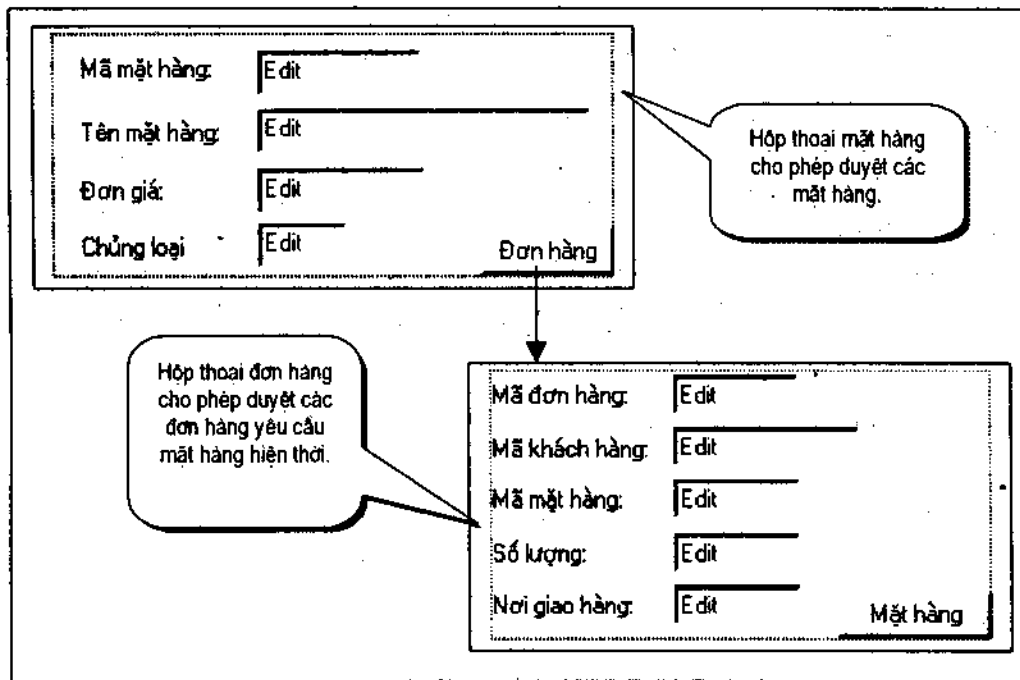
void CProductView::OnInitialUpdate()
{
    m_pSet = &GetDocument()->m_productSet;
    m_pSet->m_strSort = "[mh_id]"; //xác định trường sắp xếp
    CRecordView::OnInitialUpdate();
    //Đặt tiêu đề
    if(m_pSet->IsOpen()) //kiểm tra có phải recordset được mở
    {
        CString strTitle = _T("Tên bảng");
        CString strTable = m_pSet->GetTableName();
        if(!strTable.IsEmpty())
            strTitle += _T(": ") + strTable;
        GetDocument()->SetTitle(strTitle); //Đặt tiêu đề cho tài liệu
    }
}

```

9.3.5 Sử dụng đối tượng recordset thứ hai

Chúng ta đã hoàn thiện việc xem các mặt hàng, bây giờ sẽ mở rộng chương trình để xem các đơn đặt hàng yêu cầu mặt hàng đang hiển thị. Để làm điều này, chúng ta phải bổ sung thêm một lớp *recordset* để quản lý thông tin đơn hàng và một lớp quan sát mới để hiển thị thông tin trong *recordset*. Chúng ta sẽ bổ sung phím **Đơn hàng** vào hộp thoại mặt hàng để mở ra cửa sổ đơn hàng khi muốn xem các đơn hàng yêu cầu mặt hàng hiện thời.

Thí dụ, hộp thoại đơn hàng được xây dựng như trong hình 9.12.



Hình 9.12 Sơ đồ duyệt mặt hàng và đơn hàng

Hộp thoại mặt hàng sẽ xuất hiện trước, khi nhấn phím *Đơn hàng* sẽ chuyển sang cửa sổ đơn hàng và chúng ta có thể xem toàn bộ các đơn hàng yêu cầu mặt hàng hiện thời. Trong cửa sổ đơn hàng, chúng ta quay lại cửa sổ mặt hàng bằng phím *Mặt hàng*.

Bổ sung lớp *recordset*

Chúng ta sử dụng *ClassWizard* để bổ sung lớp *recordset*. Chọn thực đơn *View-->ClassWizard*, trong cửa sổ *ClassWizard* nhấn phím *Add Class...* và chọn *New...* từ thực đơn *pop up*. Trong hộp thoại tạo lớp hãy đưa vào tên của lớp là *COrderSet* và chọn lớp cơ sở *CRecordset*, kết thúc bằng phím *OK* để đi tới hộp thoại chọn *CSDL* như phần trước. Tại đây, ta chọn lại *Du lieu vi du* với kiểu *Snapshot*. Sau đó nhấn phím *OK* để chuyển đến hộp thoại chọn bảng. Chúng ta sẽ chọn hai bảng *DonHang* và *DHChiTiet* và kết thúc bằng phím *OK*.

ClassWizard tạo các thành phần dữ liệu cho tất cả các trường của mỗi bảng. Một điểm chú ý là trường *dh_id* có trong cả hai bảng do đó *ClassWizard* phân biệt chúng bằng phần tiền tố là tên bảng. Tên phần dữ liệu cho trường *dh_id* của bảng *DonHang* là *m_dh_id*, còn thành phần dữ liệu cho trường *dh_id* của bảng *DHChiTiet* là *m_dh_id2*.

Để móc (*hook*) *recordset* mới vào tài liệu, chúng ta cần bổ sung thêm thành phần dữ liệu định

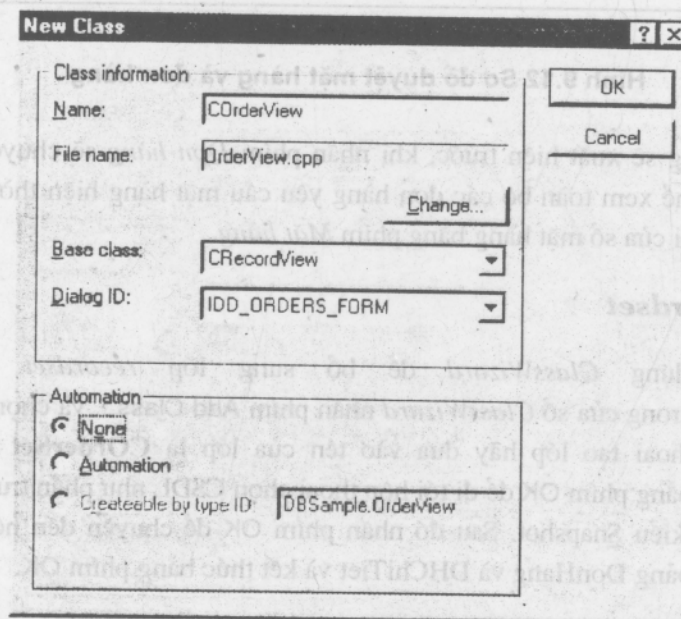
nghĩa chúng trong lớp CDBSampleDoc. Hãy nhấn phím phải của chuột trên lớp CDBSampleDoc trong cửa sổ *ClassView* và chọn **Add Member Variable...** trong thực đơn *pop up*. Cuối cùng là đặt tên biến là **m_OrderSet** với kiểu là **COrderSet** và chế độ thăm nhập là **public**.

Bổ sung lớp quan sát

Bây giờ chúng ta cần thiết kế hộp thoại để hiển thị đơn hàng như trong hình 9.12 bằng cách nhấn phím phải của chuột vào *Dialog* trong cửa sổ *ResourceView*, chọn **Insert Dialog** từ thực đơn *pop up* và sử dụng các thanh công cụ để thiết kế. Hộp thoại này có cùng kiểu với hộp thoại hiển thị mặt hàng và sẽ đặt chỉ danh là **IDD_ORDERS_FORM**. Phím **Mặt hàng** để quay trở lại hộp thoại mặt hàng sẽ được gán chỉ danh là **IDC_MAT_HANG**.

Tạo lớp quan sát cho recordset

* Để tạo lớp quan sát cho *recordset*, nhấn phím phải của chuột trên hộp thoại và chọn **ClassWizard...** từ thực đơn *pop up*. Một hộp thoại có hai lựa chọn **Create a new class** và **Select an existing class** được mở ra. Ta chọn **Create a new class** để đi tới hộp thoại tạo lớp mới như hình 9.13.



Hình 9.13 Hộp thoại tạo lớp mới

Ta đặt tên cho lớp mới này là **COrderView**, chọn lớp cơ sở là **CRecordView** và *Dialog ID* là **IDD_ORDERS_FORM**. Sau khi nhấn phím **OK**, *ClassWizard* sẽ tự động chọn **COrderSet** làm lớp

liên kết với lớp quan sát. Như vậy lớp COrderView là lớp phái sinh từ lớp CRecordView và được móc vào hộp thoại IDD_ORDERS_FORM.

ClassWizard tự động bổ sung thêm thành phần dữ liệu m_pSet vào lớp COrderView để giữ địa chỉ của đối tượng COrderSet.

Liên kết điều khiển với recordset

Sau khi tạo xong lớp COrderView, ta cần liên kết các điều khiển soạn thảo trong hộp thoại với các dữ liệu trong *recordset*. Cách tạo liên kết giống như cách tạo với hộp thoại mặt hàng: giữ *Ctrl* và nhấn đúp chuột vào điều khiển.

Tùy biến lớp quan sát

Câu lệnh SELECT cho đối tượng COrderSet sẽ tạo ra một bảng mới bằng phép kết nối hai bảng DonHang và DHChiTiet, như vậy sẽ có rất nhiều bản ghi như ta đã biết. Một vấn đề nữa là khi chuyển từ cửa sổ mặt hàng sang cửa sổ đơn hàng ta chỉ muốn hiển thị các đơn hàng có yêu cầu mặt hàng hiện thời. Do đó chúng ta phải đưa thêm mệnh đề WHERE vào câu lệnh SELECT để lọc các bản ghi cần thiết. Mệnh đề WHERE trong lệnh SELECT được gọi là bộ lọc.

Bổ sung bộ lọc cho recordset

Bổ sung bộ lọc cho câu lệnh SELECT được thực hiện bằng cách gán xâu điều kiện cho thành phần dữ liệu m_strFilter của đối tượng *recordset*. Thành phần dữ liệu m_strFilter được thừa hưởng từ lớp cơ sở CRecordSet. Cũng như mệnh đề ORDER BY được bổ sung bằng cách gán giá trị cho thành phần dữ liệu m_strSort của đối tượng *recordset*, việc gán giá trị cho m_strFilter cũng được thực hiện trong hàm thành phần OnInitUpdate của lớp quan sát.

Chúng ta cần đặt hai điều kiện trong bộ lọc, một là để hạn chế các bản ghi khi kết nối dữ liệu giữa hai bảng DonHang và DHChiTiet bằng điều kiện trường dh_id trong bảng DonHang phải bằng trường dh_id trong bảng DHChiTiet. Ta có thể viết điều kiện như sau:

```
[DonHang].[dh_id] = [DHChiTiet].[dh_id]
```

Điều kiện thứ hai là để chọn ra các đơn hàng có yêu cầu mặt hàng hiện thời bằng điều kiện trường mh_id trong *recordset* phải bằng trường mh_id của bản ghi hiện thời trong bảng MatHang. Điều này có nghĩa là chúng ta cần phải so sánh trường mh_id trong đối tượng COrderSet với một giá trị biến thiên. Giá trị biến thiên trong thao tác này được gọi là tham số và điều kiện trong bộ lọc có thể viết theo cách đặc biệt sau:

```
mh_id = ?
```

Giá trị thay thế cho dấu hỏi (?) sẽ được đặt trong hàm thành phần DoFieldExchange của đối

tượng *recordset*.

Chúng ta có thể định nghĩa xâu điều kiện cho bộ lọc bằng bổ sung câu lệnh sau vào hàm thành phần *OnInitUpdate* của lớp *COrderView*:

```
m_pSet->m_strFilter = "[mh_id]=? AND [DonHang].[dh_id]=[DHChiTiet].[dh_id]";
```

Định nghĩa tham số lọc

Chúng ta cần bổ sung thành phần dữ liệu cho lớp *COrderSet* để chứa giá trị hiện thời của trường *mh_id* trong đối tượng *CProductSet* và cũng sẽ là tham số thay thế cho dấu ? trong bộ lọc cho đối tượng *COrderSet*. Biến thành phần này sẽ được đặt tên *m_mh_idParam* với kiểu là *CString* (kiểu của trường *mh_id*) và chế độ thâm nhập là *public*. Cách thêm giống như cách thêm thành phần dữ liệu *m_OrderSet* của lớp *COrderSet*. Thêm vào cấu tử của lớp *COrderSet* đoạn trình để gán giá trị khởi tạo cho *m_mh_idParam* và xác định số lượng tham số của *recordset* như dưới đây:

```
COrderSet::COrderSet(CDatabase* pdb) : CRecordset(pdb)
{
    //{{AFX_FIELD_INIT(COrderSet)
    m_dh_id = _T("");          m_noi_giao = _T("");
    m_nguoi_giao = _T("");      m_dh_id2 = _T("");
    m_mh_id = _T("");          m_kh_id = _T("");
    m_so_luong = 0.0;          m_nFields = 8;
    //}}AFX_FIELD_INIT
    m_mh_idParam = _T(""); //Gán giá trị khởi tạo là xâu rỗng
    m_nParams = 1; //xác định số tham số của recordset
    m_nDefaultType = snapshot;
}
```

Biến *m_nParams* được thừa hưởng của lớp cơ sở *CRecordSet*. Chúng ta cần gán giá trị cho biến này bằng tổng số tham số (số dấu ?) sử dụng trong bộ lọc cho *recordset*, nếu không thì kết quả sẽ bị sai.

Để chỉ ra biến *m_mh_idParam* của lớp *COrderSet* là tham số thay thế cho dấu ? trong bộ lọc của đối tượng *COrderSet* thì cần bổ sung thêm đoạn mã trình vào hàm thành phần *DoFieldExchange* của lớp *COrderSet* như sau:

```
void COrderSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(COrderSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[DHChiTiet].[dh_id]"), m_dh_id);
    RFX_Date(pFX, _T("[ngay_giao]"), m_ngay_giao);
    RFX_Text(pFX, _T("[noi_giao]"), m_noi_giao);
}
```

```

RFX_Text(pFX, _T("[nguoi_giao]").m_nguoi_giao);
RFX_Text(pFX, _T("[DonHang].[dh_id]").m_dh_id2);
RFX_Text(pFX, _T("[mh_id]").m_mh_id);
RFX_Text(pFX, _T("[kh_id]").m_kh_id);
RFX_Double(pFX, _T("[so_luong]").m_so_luong);
//}}AFX_FIELD_MAP
// dat kieu truong nhu tham so
pFX->SetFieldType(CFieldExchange::param);
RFX_Text(pFX, _T("mh_idParam"), m_mh_idParam);
}

```

Dòng lệnh thứ nhất gọi tới hàm thành phần `SetFieldType` của đối tượng `pFX` để đặt kiểu gọi `param` cho hàm `RFX_` tiếp sau. Nếu trong bộ lọc có nhiều tham số, ta bổ sung thêm các hàm `RFX_` theo thứ tự các tham số. Với kiểu đặt `param`, tham số thứ hai trong hàm `RFX_` được bỏ qua, chúng ta có thể đưa vào giá trị `NULL` hoặc một xâu thông báo bất kỳ.

Khởi tạo quan sát

Chúng ta cần bổ sung thêm các mã trình vào hàm thành phần `OnInitialUpdate` của lớp `COrderView` để đặt chế độ sắp xếp và tiêu đề cho cửa sổ như sau:

```

void COrderView::OnInitialUpdate()
{
    BeginWaitCursor();
    CDBSampleDoc *pDoc=(CDBSampleDoc *)GetDocument(); //lấy con trỏ tài liệu
    m_pSet = &pDoc->m_OrderSet; //đặt con trỏ tới recordset
    // sử dụng CSDL trong CProductSet
    m_pSet->m_pDatabase = pDoc->m_productSet.m_pDatabase;
    // đặt mh_id hiện thời là tham số
    m_pSet->m_mh_idParam = pDoc->m_productSet.m_mh_id;
    // đặt bộ lọc
    m_pSet->m_strFilter = "[mh_id]=? AND [DonHang].[dh_id]=[DHChiTiet].[dh_id]";
    GetRecordset();
    // đặt tiêu đề
    if(m_pSet->IsOpen()) //kiểm tra có phải recordset được mở
    {
        CString strTitle = _T("Tên bảng");
        CString strTable = m_pSet->GetTableName();
        if(!strTable.IsEmpty()) strTitle += _T(":") + strTable;
        else strTitle += _T("Đơn hàng - nhiều bảng");
        GetDocument()->SetTitle(strTitle); //Đặt tiêu đề cho tài liệu
    }
}

CRecordView::OnInitialUpdate();

```



```
EndWaitCursor();
}
```

Chúng ta cần bổ sung thêm các dòng lệnh sau đây vào tệp định nghĩa hàm `OnInitUpdate`.

```
#include "ProductSet.h"
#include "OrderSet.h"
#include "DBSampleDoc.h"
```

Thêm nhập các bảng phức tạp

Để việc chuyển giữa hai cửa sổ mặt hàng và đơn hàng đúng theo mong muốn, chúng ta cần giữ được trạng thái của cửa sổ hiện thời, điều này có thể thực hiện bằng cách gán cho mỗi cửa sổ hay mỗi quan sát một chỉ danh duy nhất. Ta tạo thêm tệp mới có tên `OurConst.h` với nội dung sau để định nghĩa chỉ danh cho các cửa sổ:

```
#if !defined(OUR_CONST_H)
#define OUR_CONST_H
const UINT PRODUCT_VIEW = 1;
const UINT ORDER_VIEW = 2;
#endif // !defined(OUR_CONST_H)
```

Chúng ta sẽ sử dụng các hằng này để gán cho các quan sát và ghi lại chỉ danh của quan sát hiện thời trong đối tượng cửa sổ khung. Để làm điều này, cần bổ sung thêm thành phần dữ liệu `m_CurrentViewID` có kiểu `UINT` và chế độ thêm nhập *public* cho lớp `CMainFrame`. Đặt giá trị khởi tạo cho `m_CurrentViewID` trong cấu tử của lớp `CMainFrame`:

```
CMainFrame::CMainFrame()
{
    m_CurrentViewID=PRODUCT_VIEW;
}
```

Bổ sung câu lệnh

```
#include "OurConst.h"
```

vào đầu tệp chứa hàm `CMainFrame::CMainFrame`.

Chuyển các quan sát

Để cơ chế chuyển quan sát làm việc, chúng ta bổ sung hàm thành phần chung `SelectView` cho lớp `CMainFrame`. Hàm này có một tham số định nghĩa chỉ danh của quan sát và hàm sẽ chuyển từ quan sát hiện thời sang quan sát có chỉ danh xác định trong tham số.

Bổ sung hàm bằng cách nhấn phím phải của chuột trên `CMainFrame` trong cửa sổ *ViewClass* và chọn **Add Member Function...** từ thực đơn *pop up*. Xác định kiểu hàm là `void` và khai báo hàm là

SelectView(UINT ViewID), chế độ thâm nhập là *public*.

Khai báo nội dung hàm như sau:

```
void CMainFrame::SelectView(UINT ViewID)
{
    CView* pOldActiveView = GetActiveView(); //lấy quan sát hiện thời
    // lấy con trỏ tới quan sát mới nếu nó đã tồn tại, nếu không con trỏ là NULL
    CView* pNewActiveView = (CView*)GetDlgItem(ViewID);
    if(pNewActiveView==NULL) //nếu quan sát mới chưa tồn tại
    {
        switch(ViewID)
        {
            case ORDER_VIEW:
                pNewActiveView = (CView*)new COrderView; //tạo quan sát đơn hàng
                break;
            default:
                AfxMessageBox("ViewID không hợp lệ");
                return;
        }
    }
    // chuyển quan sát mới thành quan sát hiện thời
    CCreateContext context;
    context.m_pCurrentDoc = pOldActiveView->GetDocument();
    pNewActiveView->Create(NULL, NULL, OL, CFrameWnd::rectDefault, this, ViewID, &context);
    pNewActiveView->OnInitialUpdate();
    SetActiveView(pNewActiveView); //active quan sát mới
    pOldActiveView->ShowWindow(SW_HIDE); //giấu quan sát cũ
    pNewActiveView->ShowWindow(SW_SHOW); //hiển thị quan sát mới
    pOldActiveView->SetDlgCtrlID(m_CurrentViewID); //đặt chỉ danh quan sát cũ
    pNewActiveView->SetDlgCtrlID(AFX_IDW_PANE_FIRST);
    m_CurrentViewID = ViewID; //ghi lại chỉ danh quan sát mới
    RecalcLayout(); //vẽ lại quan sát
}
```

Nội dung của hàm có thể chia làm ba phần:

- Lấy con trỏ tới quan sát hiện thời và quan sát mới.
- Tạo quan sát mới nếu chưa tồn tại.
- Đổi quan sát mới thành quan sát hiện thời.

Ở đây có một vài điểm đáng chú ý:

- Bởi vì sẽ tạo một đối tượng COrderView do đó ta cần thâm nhập vào cấu trúc của lớp này.

Trong chế độ mặc định, cấu tử `COrderView` được đặt ở chế độ bảo vệ (*protected*) nên chúng ta phải đổi nó sang chế độ dùng chung (*public*).

```
class COrderView : public CRecordView
{
public:
    COrderView();
protected:
    DECLARE_DYNCREATE(COrderView)
    .....
}
```

- Chúng ta đặt chỉ danh của quan sát mới là `AFX_IDW_PANE_FIRST` để chỉ định nó là cửa sổ đầu tiên cho chương trình.

Cần phải bổ sung thêm dòng lệnh sau:

```
#include "OrderView.h"
```

vào tệp định nghĩa hàm `CMainFrame::SelectView(UINT ViewID)`.

Bổ sung phím điều khiển chuyển quan sát

Bây giờ chúng ta bổ sung phím điều khiển **Đơn hàng** cho hộp thoại mật hàng để liên kết với hộp thoại đơn hàng và bổ sung mã điều khiển cho phím này và đối tác của nó trong hộp thoại đơn hàng để gọi tới hàm thành phần `SelectView` của lớp `CMainFrame`.

Mở hộp thoại `IDD_DBSAMPLE_FORM` trong cửa sổ *ResourceView* và bổ sung phím điều khiển **Đơn hàng** và đặt chỉ danh là `IDC_DON_HANG`.

Sau khi ghi lại *resource*, giữ phím *Ctrl* và nhấn đúp chuột vào phím này để bổ sung hàm `OnDonHang` cho lớp `CProductView`. Bổ sung thêm dòng lệnh sau vào hàm thành phần này:

```
void CProductView::OnDonHang()
{
    ((CMainFrame*)GetParentFrame())->SelectView(ORDER_VIEW);
}
```

Hàm thành phần `GetParentFrame` được thừa hưởng từ lớp `CWnd`, lớp cơ sở gián tiếp của lớp `CMainFrame`. Bổ sung hai dòng lệnh sau:

```
#include "OurConst.h"
#include "MainFrm.h"
```

vào tệp chứa hàm `CProductView::OnDonHang`.

Tiếp theo mở hộp thoại IDD_ORDERS_FORM trong cửa sổ *ResourceView*, giữ phím *Ctrl* và nhấn đúp chuột vào phím *Mặt hàng* để bổ sung hàm thành phần **OnMatHang** cho lớp **COrderView** và mã trình cho hàm.

```
void COrderView::OnMatHang()
{
    ((CMainFrame*)GetParentFrame())->SelectView(PRODUCT_VIEW);
}
```

Bổ sung hai dòng lệnh sau:

```
#include "OurConst.h"
#include "MainFrm.h"
```

vào tệp chứa hàm **COrderView::OnMatHang**.

Quản lý quan sát tích cực

Khi chuyển tới một quan sát đã tồn tại, chúng ta cần chắc chắn rằng *recordset* đã được cập nhật và hộp thoại được khởi tạo lại, như vậy dữ liệu hiển thị mới là chính xác. Khi một quan sát được trở thành tích cực hay không tích cực, khung làm việc gọi tới hàm thành phần **OnActivateView** của lớp. Chúng ta cần nạp chồng hàm này trong các lớp quan sát. Việc này có thể được thực hiện bằng cách sử dụng bảng ánh xạ thông điệp **Message Maps** trong hộp thoại *ClassWizard*. Hãy bổ sung đoạn mã trình sau vào hàm này:

```
void COrderView::OnActivateView(BOOL bActivate, CView* pActivateView, CView* pDeactivateView)
{
    if(bActivate)
    {
        //Lấy con trỏ tới đối tượng tài liệu
        CDBSampleDoc* pDoc = (CDBSampleDoc*)GetDocument();
        // Lấy con trỏ tới cửa sổ khung
        CMainFrame* pMFrame = (CMainFrame*)GetParentFrame();
        //Nếu quan sát cuối cùng là Mặt hàng, chúng ta phải thực hiện query lại
        if(pMFrame->m_CurrentViewID==PRODUCT_VIEW) {
            if(!m_pSet->IsOpen()) return;
            //đặt mh_id hiện thời làm tham số
            m_pSet->m_mh_idParam = pDoc->m_productSet.m_mh_id;
            m_pSet->Requery(); //Lấy lại dữ liệu từ CSDL
            if(m_pSet->IsEOF())
                AfxMessageBox("Không có đơn hàng nào cho mặt hàng này");
        }
        //Đặt tiêu đề cho cửa sổ
        CString strTitle = _T("Tên bảng:");
    }
}
```

```

CString strTable = m_pSet->GetTableName();
if(!strTable.IsEmpty()) strTitle += strTable;
else strTitle += _T("Đơn hàng - nhiều bảng");
GetDocument()->SetTitle(strTitle); //Đặt tiêu đề cho tài liệu
CRecordView::OnInitialUpdate(); //cập nhật các giá trị trong hộp thoại
}

CRecordView::OnActivateView(bActivate, pActivateView, pDeactivateView);
}

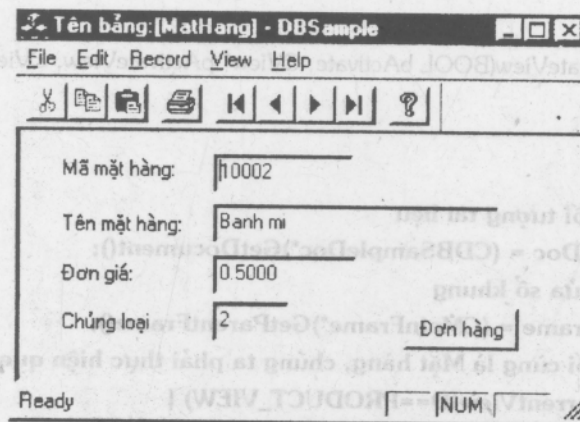
```

Tiếp theo hãy bổ sung hàm OnActivateView cho lớp CProductView:

```

void CProductView::OnActivateView(BOOL bActivate, CView* pActivateView, CView* pDeactivateView)
{
    if(bActivate)
    {
        //Cập nhật tiêu đề cho cửa sổ
        CString strTitle = _T("Tên bng:");
        CString strTable = m_pSet->GetTableName();
        strTitle += strTable;
        GetDocument()->SetTitle(strTitle);
    }
    CRecordView::OnActivateView(bActivate, pActivateView, pDeactivateView);
}

```



Hình 9.14 Một kết quả thực hiện chương trình

Bây giờ chúng ta có thể dịch và chạy chương trình vừa xây dựng. Hình 9.14 là màn hình kết quả khi thực hiện chương trình.

Ta có thể xây dựng thêm các quan sát mới như quan sát khách hàng, quan sát chủng loại hàng... theo cách xây dựng quan sát đơn hàng trên đây để hoàn thiện thêm chương trình.

NHÚNG VÀ LIÊN KẾT ĐỐI TƯỢNG

Nhúng và liên kết đối tượng (*Object Linking and Embedding - OLE*) là cơ chế cho phép lập trình viên viết những ứng dụng, thí dụ trình soạn thảo văn bản, cho phép ứng dụng khác soạn thảo dữ liệu riêng ngay trong chúng, thí dụ vẽ đồ họa. OLE còn cho khả năng ứng dụng do ta viết sẽ quản lý được dữ liệu đang ở trong ứng dụng khác.

Một khi lập trình viên cho phép ứng dụng của mình chứa các đối tượng dữ liệu thì họ có thể có bất cứ loại dữ liệu nào và có bao nhiêu đối tượng tùy ý. Cơ chế này cho phép kích hoạt các chương trình khác nhau liên quan đến mỗi loại đối tượng liên kết trong ứng dụng.

Một đối tượng của chương trình có thể xuất hiện trong chương trình khác theo hai cách khác nhau. Đối tượng của tài liệu (dữ liệu) ngoài có thể liên kết (*link*) với tài liệu (dữ liệu) của chương trình khác. Khi đó đối tượng ngoài sẽ không được lưu vào trong tài liệu của chương trình hiện hành mà chỉ là ảnh xạ để truy nhập sau này. Cách khác là tài liệu ngoài được nhúng (*embed*) vào tài liệu hiện hành. Khi đó, đối tượng của tài liệu ngoài sẽ được lưu trữ trong tài liệu hiện hành. Một tài liệu chứa đối tượng OLE liên kết hay nhúng được gọi là tài liệu đa hợp (*compound document*).

Lợi thế của đối tượng liên kết là chúng có thể sửa đổi độc lập với tài liệu đa hợp. Vậy, khi mở tài liệu đa hợp có chứa đối tượng liên kết để đọc thì bản sửa cuối cùng của đối tượng ngoài sẽ được cập nhật. Tuy nhiên, khi xóa tệp chứa đối tượng liên kết hay khi chuyển chúng sang danh mục khác trên đĩa thì tài liệu đa hợp sẽ không nhận biết điều này và không có khả năng tìm kiếm đối tượng liên kết đó. Với cơ chế nhúng, đối tượng chỉ tồn tại trong môi trường của tài liệu đa hợp. Tài liệu đa hợp cùng với các đối tượng nhúng được lưu trên một tệp.

Để cơ chế OLE làm việc được thì ứng dụng phải có đoạn mã trình đặc biệt trợ giúp các chức năng nói trên. Chương trình có khả năng quản lý các đối tượng nhúng được gọi là trình chứa OLE (*OLE container*) và chương trình tạo lập ra các đối tượng để nhúng trong trình chứa OLE được gọi là trình dịch vụ OLE (*OLE server*). Có hai loại trình dịch vụ OLE: trình dịch vụ đầy đủ (*full server*) và trình dịch vụ tối thiểu (*mini-server*). Trình dịch vụ đầy đủ có khả năng thao tác như ứng dụng độc lập, hay phục vụ các đối tượng nhúng trong tài liệu đa hợp. Trình dịch vụ tối thiểu không có khả năng hoạt động như ứng dụng độc lập. Chức năng duy nhất của chúng là trợ giúp các đối tượng trong tài liệu đa hợp. Một ứng dụng có thể là trình chứa OLE đồng thời là trình dịch vụ

OLE.

10.1 TÀI LIỆU ĐA HỢP

Khi làm việc với đối tượng nhúng trong chương trình, mã trình của ứng dụng phát sinh đối tượng nhúng được tự động kích hoạt để người dùng sửa đổi đối tượng trong cửa sổ trình (ứng dụng) chứa. Hoạt động này được gọi là kích hoạt tại chỗ. Ta có thể sửa đổi đối tượng trong ứng dụng chứa OLE nhờ nhấn đúp chuột trên nó khi dịch vụ OLE trợ giúp kích hoạt tại chỗ. Các thực đơn, thanh công cụ trong ứng dụng chứa sẽ thay đổi cho thích hợp với yêu cầu sửa đổi đối tượng của người sử dụng ứng dụng dịch vụ. Hơn thế nữa, nếu có vài đối tượng nhúng khác nhau, thực đơn và thanh công cụ trong ứng dụng chứa sẽ thay đổi cho phù hợp với từng loại đối tượng nhúng đang làm việc. Việc thay đổi hoàn toàn tự động.

Nếu trình dịch vụ OLE không trợ giúp kích hoạt tại chỗ thì khi nhấn đúp trên đối tượng nhúng, cửa sổ riêng biệt sẽ được mở để sửa đổi đối tượng nhúng. Để kết thúc sửa đổi đối tượng nhúng, ta chỉ việc đóng cửa sổ ứng dụng dịch vụ để tiếp tục công việc với ứng dụng chứa. Kích hoạt tại chỗ là cách hấp dẫn nhất để quản lý tài liệu đa hợp, chúng làm như là chỉ có một ứng dụng chạy. Phần lớn các ứng dụng chứa cho phép người dùng sửa đổi đối tượng nhúng trong cửa sổ ứng dụng dịch vụ, ngay cả khi dịch vụ không trợ giúp kích hoạt tại chỗ. Nhấn đúp đồng thời với nhấn phím Ctrl hay Alt là khởi đầu chế độ làm việc này của thao tác dịch vụ. Kích hoạt tại chỗ chỉ xảy ra với nhúng, không xảy ra với liên kết.

Một khi đối tượng được nhúng trong ứng dụng chứa, ứng dụng dịch vụ trợ giúp sẽ ở một trong hai trạng thái. Khi ứng dụng dịch vụ được kích hoạt tại chỗ để sửa đổi thì đối tượng được hiển thị với đường viền bóng trong vùng làm việc của cửa sổ ứng dụng chứa. Nếu nhấn phím chuột ngoài đối tượng, trình dịch vụ sẽ bị vô hiệu và đường viền bóng sẽ mất đi.

10.2 OLE LÀM VIỆC NHƯ THẾ NÀO?

Liên lạc giữa trình dịch vụ OLE có đối tượng nhúng với trình chứa OLE trước hết liên quan tới vùng cửa sổ mà đối tượng chiếm khi chúng cần được vẽ lại và liên quan đến các tài nguyên mà trình dịch vụ cần có trong trình chứa như thực đơn, thanh công cụ để sửa đổi đối tượng. Trình chứa sẽ không biết trình dịch vụ hiển thị gì. Chúng chỉ biết rằng một vùng làm việc của chúng được trình dịch vụ sử dụng còn việc hiển thị cái gì là trách nhiệm của trình dịch vụ. Trình chứa cũng không biết thực đơn hay thanh công cụ gì sẽ hiển thị, chúng chỉ biết dành chỗ để hiển thị chúng. Ta có thể dự đoán rằng liên lạc giữa trình chứa OLE và trình dịch vụ trợ giúp các đối tượng nhúng sẽ sử dụng hệ điều hành Windows làm trung gian. Mỗi trình OLE sẽ sử dụng thư viện liên kết động OLE DLL dùng chung. Chúng là một phần của Windows. Các hàm của DLL sẽ cho công cụ trao đổi thông tin giữa chúng. Vậy, chìa khoá của thao tác OLE là ghép nối chuẩn. Ghép nối chuẩn cho phép OLE

làm việc theo phong cách mô hình đối tượng thành phần (*Component Object Model - COM*). Đó là định nghĩa cách thể hiện đối tượng nhúng và cách thức liên lạc giữa chúng với trình chứa. COM là chủ điểm lớn, nhưng ở đây chỉ trình bày tóm tắt về ý tưởng.

COM có tập hàm chuẩn được sử dụng để liên lạc OLE, chúng được nhóm lại và có tên là ghép nối. Tương tự như lớp C++, chúng định nghĩa ghép nối thông qua các hàm thành phần chung (*public*). Với đối tượng COM như trình dịch vụ OLE, có ít nhất một ghép nối (hay nhóm hàm ghép nối) luôn được cài đặt với tên IUnknown. Ghép nối IUnknown có ba hàm chuẩn sau đây:

QueryInterface	Kiểm tra xem đối tượng có trợ giúp ghép nối hay không. Nếu trợ giúp, hàm cho tại con trỏ.
AddRef	Làm tăng số đếm số lượng trạm sử dụng ghép nối. Số đếm này cho đối tượng sử dụng ghép nối biết khi nào không cần đến nữa.
Release	Làm giảm số đếm tổng số trạm sử dụng ghép nối. Khi số đếm có giá trị 0 thì đối tượng có thể tự loại ra khỏi bộ nhớ.

Ta có thể làm bất kỳ cái gì với ba hàm trên. Vì hàm **QueryInterface** cho ta hỏi về ghép nối khác, có thể thăm nhập bất kỳ ghép nối nào mà đối tượng trợ giúp. OLE định nghĩa tập các thao tác chuẩn, mỗi thao tác được phân biệt bởi chỉ danh ghép nối (*Interface ID - IID*). Chúng được chuyển đến lời gọi **QueryInterface** như đối của hàm. Cũng có thể định nghĩa các ghép nối riêng của ta và cũng phải phân biệt chúng bằng IID. Vì MFC giúp ta sử dụng chúng một cách dễ dàng cho nên không cần phân tích sâu hơn các hàm ghép nối này.

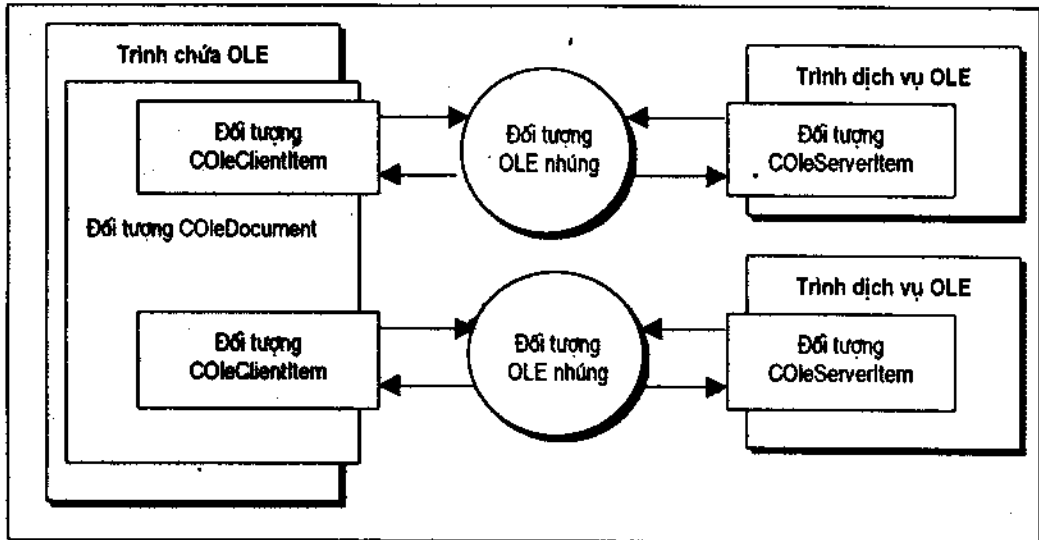
Để có thể sử dụng được trình dịch vụ OLE, phải có cách nhận dạng chúng. Khi chạy trình chứa OLE, ta không muốn phải duyệt qua toàn bộ đĩa cứng để xem có ứng dụng nào trên PC trợ giúp OLE hay không. Vậy, nhận dạng các chương trình OLE như thế nào?

Một đối tượng OLE có thể là chương trình, kiểu tài liệu hay bất kỳ loại đối tượng nào trợ giúp OLE. Mỗi đối tượng OLE trong hệ thống được nhận dạng bằng giá trị số 128-bit. Chúng được gọi là *class ID* hay *CLSID*. CLSID và IID là các kiểu đặc biệt của chỉ danh duy nhất toàn cầu (*Globally Unique ID - GUID*). Các chỉ danh này được gọi là duy nhất toàn cầu vì chúng được phát sinh bằng thuật toán đảm bảo chỉ danh là duy nhất trên toàn thế giới. Thông tin về đối tượng OLE trong hệ thống, bao gồm cả CLSID, được lưu trong cơ sở dữ liệu trên đĩa cứng. Cơ sở dữ liệu này được gọi là bộ đăng ký hệ thống (*system registry*). Ta có thể xem bộ đăng ký hệ thống bằng chương trình Regedit.exe. Trình dịch vụ OLE chưa được sử dụng chừng nào các thông tin để nhận dạng chúng chưa được nhập vào bộ đăng ký hệ thống.

10.3 CÁC LỚP MFC TRỢ GIÚP OLE

MFC có tập các lớp có khả năng biểu diễn các đối tượng OLE, cũng như các lớp có thể biểu diễn tài liệu chứa các đối tượng OLE. Thí dụ, lớp **COleServerDoc** thừa hưởng các chức năng của lớp cơ sở **COleLinkingDoc** và các lớp cơ sở gián tiếp **CDocument**, **COleDocument**.

Thí dụ ta có hai trình dịch vụ OLE và trình chứa OLE như hình 10.1.



Hình 10.1 Đối tượng OLE nhúng

Hai lớp đối tượng OLE được suy diễn từ các lớp CDocItem, COleClientItem và COleServerItem của MFC. Hình 10.1 cho thấy hai đối tượng OLE nhúng trong trình dịch vụ. Mỗi đối tượng nhúng có trình dịch vụ riêng, mỗi trình dịch vụ sẽ có đối tượng COleServerItem tương ứng với đối tượng trong trình dịch vụ. Lớp COleClientItem cung cấp ghép nối cho trình chứa để quản lý các đối tượng nhúng. Chúng bao gồm số lượng lớn các hàm cho phép truy nhập và xử lý đối tượng và các hàm cho phép liên lạc giữa trình chứa và trình dịch vụ. Các hàm quan trọng nhất cần thiết cho cài đặt là:

OnChange	Khung ứng dụng sẽ gọi hàm này khi trình dịch vụ phát tín hiệu thay đổi đối tượng nhúng. Thí dụ, yêu cầu đối tượng nhúng được vẽ lại trong trình chứa.
OnGetItemPosition	Khung ứng dụng gọi hàm này để có khung chữ nhật trong vùng làm việc của trình chứa khi hiển thị đối tượng OLE.
OnChangeItemPosition	Khung ứng dụng gọi hàm này để cho trình chứa biết kích thước của đối tượng nhúng đã thay đổi khi sửa đổi.
Serialize	Nếu thêm các thành phần vào đối tượng trong trình chứa, ta phải nhập chúng vào hàm này.

Việc vẽ lại hay sửa đổi đối tượng được thực hiện tại trình dịch vụ, nhưng việc hiển thị đối tượng trong vùng cửa sổ được trình chứa quản lý và thực hiện.

Đối tượng OLE nhúng trong trình dịch vụ được biểu diễn bởi đối tượng của lớp COleServerItem. Ghép nối trợ giúp trình chủ trong lớp COleServerItem cũng chứa rất nhiều hàm, trong đó có các hàm quan trọng sau:

OnDraw	Hàm này có nhiệm vụ vẽ đối tượng nhúng trong trình chứa khi chúng không bị sửa đổi. Khi đối tượng là kích hoạt tại chỗ, đối tượng được hàm này vẽ trong lớp quan sát của trình dịch vụ. Vẽ trong trình chứa được trình dịch vụ thực hiện bởi vì trình chứa không biết gì về bên trong đối tượng nhúng. Khi trình dịch vụ chạy ngừng rẽ, hàm này trong đối tượng quan sát có khả năng vẽ thông thường.
Serialize	Hàm này có trách nhiệm nhập đối tượng nhúng khi trình chứa có nhu cầu. Thông thường chúng được cài đặt bằng cách gọi hàm Serialize cho đối tượng tài liệu trong trình dịch vụ.
OnGetExtent	Khung ứng dụng gọi hàm này để lấy kích thước hiện hành của đối tượng nhúng. Chúng liên lạc với trình dịch vụ.
NotifyChange	Trình dịch vụ gọi hàm này khi chúng thay đổi đối tượng nhúng. Chúng thông báo thay đổi đến khung ứng dụng sẽ gọi hàm OnChange trong đối tượng COleClientItem tương ứng trong trình chứa.

Các lớp tài liệu đặc biệt là cần thiết cho các ứng dụng OLE vì các tài liệu phải có khả năng xử lý khi độ phức tạp của đối tượng OLE tăng lên. Có hai lớp tài liệu được sử dụng trong trình chứa OLE là: COleDocument và COleLinkingDoc. COleDocument trợ giúp các đối tượng nhúng được trình dịch vụ thực hiện sửa đổi tại chỗ. Chúng biểu diễn các đối tượng nhúng như hình thành đối tượng lớp suy diễn từ COleClientItem. Lớp COleLinkingDoc suy diễn từ COleDocument và thêm khả năng trợ giúp các đối tượng liên kết được lưu trữ riêng rẽ trong tài liệu trình chứa. Lớp tài liệu trong trình chứa thường được suy diễn từ COleDocument hay COleLinkingDoc. Tài liệu trong trình dịch vụ được suy diễn từ lớp COleServerDoc. Khi trình dịch vụ trợ giúp đối tượng nhúng thì đối tượng OLE được biểu diễn bởi lớp suy diễn từ lớp COleServerItem. Tất nhiên, tài liệu dịch vụ chỉ chứa một đối tượng của lớp đó để biểu diễn toàn bộ tài liệu nhúng trong trình chứa.

Lớp tài liệu cho trình dịch vụ OLE phải cài đặt hàm thành phần OnGetEmbeddedItem bởi vì đây là hàm ảo thuần túy trong lớp COleServerDoc. Nếu không cài đặt chúng thì chương trình không được dịch. Khung ứng dụng sẽ gọi hàm này để lấy con trỏ đến đối tượng OLE được trình dịch vụ trợ giúp. Đồng thời khung ứng dụng sử dụng chúng để gọi các hàm thành phần của đối tượng.

10.4 CÀI ĐẶT TRÌNH CHỨA OLE

Cách dễ dàng nhất để xây dựng trình chứa OLE là sử dụng AppWizard của trình dịch MS Visual C++. Hãy xem cách sử dụng trình dịch này ở chương 13. Thí dụ phát sinh *tệp project* cho ứng dụng với tên là WrxContainer. Hãy tuân thủ các bước của MFC AppWizard để phát sinh ra các tệp chương trình. Chú ý rằng trong bước 6, hãy sửa tên lớp CWrxContainerCntrlItem do trình dịch phát sinh ra thành CWrxContainerItem. Tiếp theo, ta có thể quan sát các tệp do AppWizard tạo ra để thấy được các công việc cần cài đặt trong trình chứa OLE.

10.4.1 Khởi động trình chứa OLE

Hàm thành phần InitInstance của lớp ứng dụng CWrxContainerApp sẽ thực hiện khởi động trình chứa OLE. Mã trình của chúng có thể như sau:

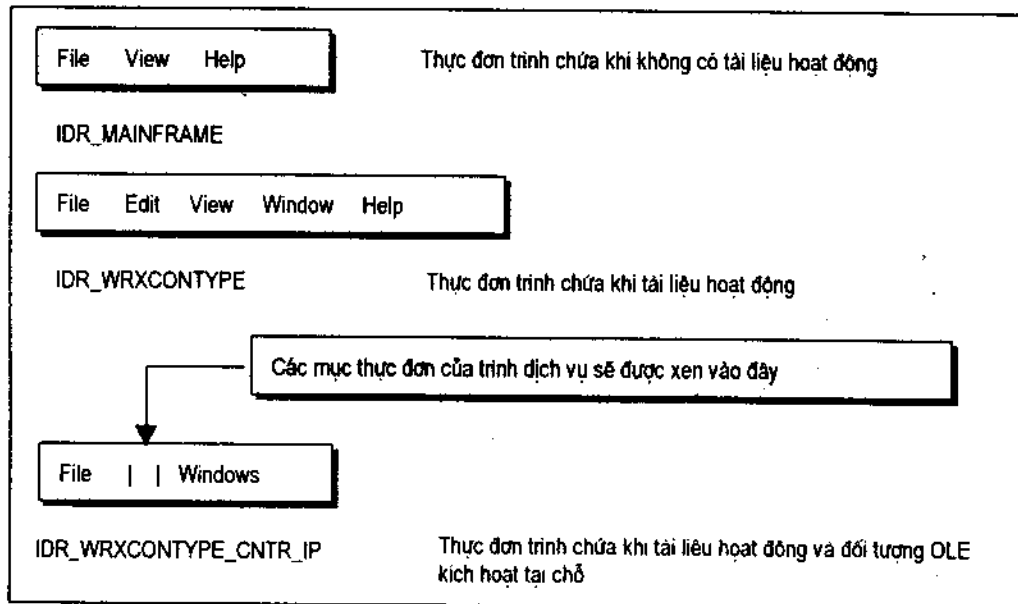
```

BOOL CWrxContainerApp::InitInstance()
{
    // Khởi động thư viện OLE
    if (!AfxOleInit())
    {
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }
    AfxEnableControlContainer();
    // Khởi động chuẩn
    // Nếu không sử dụng các đặc trưng này và muốn làm giảm kích thước tệp mã
    // trình thì lọc đi các hàm khởi động đặc biệt không cần thiết
#ifdef _AFXDLL
    Enable3dControls(); // Nếu sử dụng MFC trong DLL chia sẻ
#else
    Enable3dControlsStatic(); // Nếu liên kết tĩnh với MFC
#endif
    // Thay đổi khoá đăng ký. Có thể sửa đổi xâu dưới đây theo ý muốn
    SetRegistryKey(_T("GIS Department - Vietnam IOIT"));
    LoadStdProfileSettings(); // Nạp thuộc tính của tệp INI
    // Đăng ký mẫu tài liệu ứng dụng. Mẫu tài liệu ứng dụng được dùng kết nối
    // các tài liệu, cửa sổ khung và quan sát
    CMultiDocTemplate* pDocTemplate;
    pDocTemplate = new CMultiDocTemplate( IDR_WRXCONTTYPE,
        RUNTIME_CLASS(CWrxContainerDoc),
        RUNTIME_CLASS(CChildFrame), // Khung cửa sổ con MDI
        RUNTIME_CLASS(CWrxContainerView));
    pDocTemplate->SetContainerInfo(IDR_WRXCONTTYPE_CNTR_IP);
    AddDocTemplate(pDocTemplate);
    //Phần còn lại của hàm tương tự như các ứng dụng thông thường khác
}

```

Các dòng in đậm trên đây là sự khác nhau giữa trình chứa OLE với ứng dụng chuẩn. Đoạn trình trên gọi hàm `AfxOleInit` để khởi động DLL trợ giúp các thao tác OLE của hệ thống. Chúng thực hiện liên kết giữa ứng dụng với DLL. Nếu không khởi động được, sẽ hiển thị thông báo lỗi và trình chứa kết thúc. Gọi hàm `SetContainerInfo` của đối tượng mẫu tài liệu để tải ID thực đơn sẽ được sử dụng khi đối tượng OLE được nhúng vào kích hoạt tại chỗ. Trình chứa có ba tài nguyên thực đơn khác nhau như trong hình 10.2.

Thực đơn tương ứng với ID chuyển cho hàm `SetContainerInfo` có thanh tách thực đơn để chỉ ra đâu là mục thực đơn do trình dịch vụ cung cấp để xen vào. Các mục thực đơn được khung ứng dụng xen một cách tự động khi đối tượng nhúng tích cực.



Hình 10.2 Ba tài nguyên thực đơn

10.4.2 Lớp CWrxContainerItem

Đặc trưng mới của trình chứa OLE là có thêm lớp `CWrxContainerItem`, chúng được suy diễn từ lớp `COleClientItem`. Như đã biết, một đối tượng của lớp này là đối tượng OLE nhúng được trình dịch vụ OLE trợ giúp. Khi xen đối tượng vào trình chứa OLE, thì đối tượng `CWrxContainerItem` được lập và chúng chỉ mất đi khi tài liệu trình chứa OLE bị đóng lại hay mục nhúng tương ứng bị trình dịch vụ bãi bỏ. Khi `CWrxContainerItem` được lập, cấu tử đòi hỏi con trỏ đến đối tượng tài liệu của trình chứa OLE sao cho đối tượng được lập sẽ kết nối với tài liệu trình chứa. Lớp `CWrxContainerItem` có thể như sau:

```
class CWrxContainerItem : public COleClientItem
{
    DECLARE_SERIAL(CWrxContainerItem)
// Cấu tử
public:
    CWrxContainerItem(CWrxContainerDoc* pContainer = NULL);
// Chú ý: pContainer có thể là NULL để kích hoạt IMPLEMENT_SERIALIZE.
// IMPLEMENT_SERIALIZE yêu cầu lớp có cấu tử với đối số là 0.
// Thông thường đối tượng OLE được lập với con trỏ tài liệu khác NULL
// Các thuộc tính
public:
    CWrxContainerDoc* GetDocument()
    { return (CWrxContainerDoc*)COleClientItem::GetDocument(); }
```

```

CWrxCContainerView* GetActiveView()
{ return (CWrxCContainerView*)COleClientItem::GetActiveView(); }
// Các hàm ảo
public:
virtual void OnChange(OLE_NOTIFICATION wNotification, DWORD dwParam);
virtual void OnActivate();
protected:
virtual void OnGetItemPosition(CRect& rPosition);
virtual void OnDeactivateUI(BOOL bUndoable);
virtual BOOL OnChangeItemPosition(const CRect& rectPos);
// Cài đặt
public:
void GetServerSize();
void InvalidateItem();
CRect m_Rect;
~CWrxCContainerItem();
#ifdef _DEBUG
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif
virtual void Serialize(CArchive& ar);
};

```

Cấu tử của lớp trên đây được gọi một cách thông thường, lấy con trỏ đến tài liệu trình chứa là đối số. Tham số có giá trị mặc định NULL vì cơ chế nhập yêu cầu có cấu tử mặc định.

Hơn nữa, các thành phần dữ liệu của lớp CWrxCContainerItem có thể lưu trữ chữ nhật xác định vị trí của đối tượng nhúng. Hãy xen thêm khai báo sau đây vào vùng *public* của định nghĩa lớp:

```
CRect m_Rect; // Vị trí của mục trong đối tượng tài liệu
```

Vị trí của mỗi mục trong tài liệu được lưu trữ trong biến này. Ta có thể đặt giá trị ban đầu cho chúng trong cấu tử.

```

CWrxCContainerItem::CWrxCContainerItem(CWrxCContainerDoc* pContainer)
: COleClientItem(pContainer)
{
    m_Rect.SetRect(10, 10, 100, 100); // Đặt vị trí ban đầu
}

```

Chú ý rằng cấu tử sẽ gọi cấu tử lớp cơ sở trong danh sách khởi động và chuyển cho chúng con trỏ đến đối tượng tài liệu.

Khi trình dịch vụ thay đổi đối tượng nhúng, chúng gọi hàm thông báo cho khung ứng dụng biết

có sự thay đổi. Khung ứng dụng sẽ gọi hàm thành phần `OnChange` của đối tượng trong trình chứa OLE. Trình chứa quản lý cửa sổ nơi hiển thị đối tượng sẽ thực hiện một số công việc về sự thay đổi này.

Lý do để gọi hàm `OnChange` được thể hiện trong đối của hàm là `nCode` và `dwParam` với kiểu `OLE_NOTIFICATION` và `DWORD`. Nếu `nCode` có giá trị `OLE_CHANGED` thì có nghĩa rằng đối tượng đã thay đổi. Còn giá trị `OLE_CHANGED_STATE` cho biết đối tượng đã bị thay đổi theo cách khác. Mã trình của hàm thành phần `OnChange` có thể như sau:

```
void CWrxContainerItem::OnChange(OLE_NOTIFICATION nCode, DWORD dwParam)
{
    ASSERT_VALID(this);
    COleClientItem::OnChange(nCode, dwParam);
    // Khi một mục đang bị sửa đổi chúng sẽ gửi thông báo về sự thay đổi
    //trạng thái hay hiển thị nội dung của chúng.
    switch(nCode)
    {
        case OLE_CHANGED:
            InvalidateItem();
            GetServerSize(); // Cập nhật kích thước
            break;
        case OLE_CHANGED_STATE:
            InvalidateItem();
            break;
    }
}
```

Đoạn mã trình trên thay thế lời gọi `UpdateAllViews` trong các cài đặt thông thường. Ta sẽ cập nhật một cách có lựa chọn, phụ thuộc vào cái gì xảy ra với đối tượng nhúng. Khi giá trị `nCode` cho biết nội dung trình dịch vụ thay đổi thì ta phải vẽ lại đối tượng. Bắt đầu bằng việc gọi hàm `InvalidateItem` sẽ được cài vào lớp `CWrxContainerItem`. Ta cũng phải quan tâm đến việc trình dịch vụ có thể thay đổi kích thước của đối tượng, ta sẽ lưu trữ vùng cửa sổ nơi đối tượng chiếm trong thành phần `m_Rect` và hiệu chỉnh lại vùng nhìn của tài liệu trong trình chứa. Điều này được thực hiện trong hàm `GetServerSize` của lớp `CWrxContainerItem`.

Giá trị thứ hai của `nCode` phản ánh sự thay đổi trạng thái, chẳng hạn như đối tượng bị kích hoạt nhưng không bị sửa đổi, hay người dùng nhấn đúp chuột trên đối tượng trong vùng nhìn tài liệu để sửa đổi chúng. Trong trường hợp này, ta chỉ cần vẽ lại đối tượng bằng cách gọi hàm `InvalidateItem`. Hàm này được cài đặt như sau:

```
void CWrxContainerItem::InvalidateItem()
{
    // Gợi ý cập nhật toàn bộ vùng nhìn trong tài liệu
```

```
GetDocument()->UpdateAllViews(0, HINT_UPDATE_ITEM, this);
```

Nó gọi hàm thành phần `UpdateAllView` của đối tượng tài liệu để vẽ lại toàn bộ vùng nhìn. Giá trị tham số thứ hai là `HINT_UPDATE_ITEM` cho biết rằng tham số thứ ba là địa chỉ đối tượng hiện hành. Chúng được sử dụng khi hàm `OnUpdate` trong quan sát tài liệu trình chứa được gọi như là kết quả của lời gọi `UpdateAllView`.

Ta có thể định nghĩa giá trị của `HINT_UPDATE_ITEM` trong tệp định nghĩa lớp `CWrxContainerItem`. Xen dòng lệnh sau đây vào đầu tệp `CWrxContainerItem.h`:

```
#define HINT_UPDATE_ITEM 1
```

Hàm thành phần `GetServerSize` của lớp `CWrxContainerItem` sẽ được cài đặt như sau:

```
void CWrxContainerItem::GetServerSize()
{
    CSize aSize; // Lập đối tượng kích thước
    if (GetCachedExtent(&aSize)) // Nhận kích thước của mục hiện hành
    {
        // Kích thước được OLE xác định theo đơn vị HIMETRIC
        CClientDC aDC(0); // Nhận ngữ cảnh thiết bị
        aDC.HIMETRICtoDP(&aSize); // Chuyển đổi kích thước sang tọa độ thiết bị
        //Kiểm tra xem kích thước đã thay đổi và mục không bị kích hoạt tại chỗ?
        if (aSize != m_Rect.Size() && !IsInPlaceActive())
        {
            InvalidateItem();
            //Thay đổi kích thước mới cho mục
            m_Rect.right = m_Rect.left + aSize.cx;
            m_Rect.bottom = m_Rect.top + aSize.cy;
            InvalidateItem();
        }
    }
}
```

Kích thước của đối tượng OLE được lưu trữ trong đối tượng `aSize` nhờ hàm thành phần `GetCachedExtent` thừa hưởng từ lớp cơ sở `COleClientItem`. Nếu đối tượng rỗng, hàm này cho lại giá trị `FALSE` và sẽ không làm gì tiếp theo.

Kích thước của đối tượng OLE chuyển đến hay chuyển từ khung ứng dụng phải có đơn vị `HIMETRIC` để đảm bảo rằng chúng được thống nhất quản lý. Chúng được coi là đơn vị chuẩn cho kích thước đối tượng OLE vì có độ chính xác cao hơn các đơn vị khác như `LOMETRIC`, `HIENGLISH` hay `LOENGLISH`. Có nghĩa rằng mỗi khi chuyển kích thước cho khung ứng dụng thì phải chuyển đổi sang `HIMETRIC`. Mỗi khi nhận kích thước từ khung ứng dụng phải chuyển đổi từ

HIMETRIC sang đơn vị đang sử dụng. Trong trình chứa ta cần chữ nhật có đơn vị thiết bị (*pixel*), vậy ta sẽ sử dụng hàm chuyển đổi của đối tượng CClientDC.

Sau khi chuyển đổi *aSize* sang *pixel*, kiểm tra kích thước này có khác kích thước lưu trong *m_Rect* hay không? Ta không cần làm gì khác nếu kích thước không thay đổi. Nếu mục được kích hoạt tại chỗ và kích thước nói trên có thay đổi thì khung ứng dụng sẽ gọi hàm *OnChangeltemPosition*.

Hàm thành phần *Size* của lớp *CRect* cho lại kích thước chữ nhật lưu trong *m_Rect*. Hàm *IsInPlaceActive* thừa hưởng từ lớp *COleClientItem* sẽ cho lại giá trị *TRUE* nếu đối tượng đang được sửa đổi và giá trị *FALSE* trong trường hợp ngược lại.

Lớp *CWrxContainerItem* có hai hàm thành phần liên quan đến vị trí của đối tượng trong quan sát là *OnGetItemPosition* và *OnChangeltemPosition*. Khung ứng dụng sẽ gọi hàm *OnGetItemPosition* để biết nơi hiển thị đối tượng trong trình chứa mỗi khi đối tượng bị kích hoạt tại chỗ. Đối số của hàm sẽ chứa các thông tin cần thiết. Cài đặt của chúng có thể như sau:

```
void CWrxContainerItem::OnGetItemPosition(CRect& rPosition)
{
    ASSERT_VALID(this);
    rPosition = m_Rect;
}
```

Đoạn trình trên gán giá trị có trong thành phần *m_Rect* của đối tượng vào đối số của hàm. Vì chữ nhật trong *m_Rect* luôn được cập nhật mỗi khi có tín hiệu thay đổi từ trình dịch vụ, cho nên chúng luôn là chữ nhật hiện hành của đối tượng. Hàm thành phần *OnChangeltemPosition* được gọi mỗi khi dịch chuyển đối tượng những trong vùng quan sát, khi thay đổi kích thước khung đối tượng, hay khi trình dịch vụ yêu cầu thay đổi kích thước đối tượng. Cài đặt của hàm này có thể như sau:

```
BOOL CWrxContainerItem::OnChangeltemPosition(const CRect& rectPos)
{
    ASSERT_VALID(this);
    if (!COleClientItem::OnChangeltemPosition(rectPos))
        return FALSE;
    InvalidateItem();
    m_Rect = rectPos; // Đặt lại kích thước
    InvalidateItem();
    GetDocument()->SetModifiedFlag(); // Đánh dấu tài liệu thay đổi
    return TRUE;
}
```


10.4.3 Quản lý đa đối tượng nhúng

Để quản lý nhiều đối tượng nhúng trong trình chứa ta phải thêm chức năng cho lớp `CWrxContainerView` để người dùng có thể chuyển từ đối tượng nhúng này sang đối tượng nhúng khác. Có nghĩa là có khả năng theo dõi đối tượng tích cực hiện hành, xử lý nhấn phím chuột trong vùng nhìn để chuyển sang đối tượng tại vị trí con chuột và trả lời nhấn đúp chuột bằng kích hoạt đối tượng tại vị trí con chuột. Lớp quan sát đã có thành phần dữ liệu `m_pSelection` để lưu con trỏ đến đối tượng nhúng, vậy ta phải lưu đối tượng tích cực hiện hành vào biến này.

Chọn đối tượng

Ta cần xen thêm đoạn mã trình quản lý thông điệp `WM_LBUTTONDOWN` vào lớp `CWrxContainerView`. Đoạn mã trình đó có thể như sau:

```
void CWrxContainerView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // Lấy địa chỉ của mục bị nhấn
    CWrxContainerItem* pHitItem = HitTestItems(point);
    SelectItem(pHitItem); // Chọn mục bị nhấn
    if (pHitItem)
    {
        CRectTracker aTracker; //Lập chữ nhật
        SetupTracker(pHitItem, &aTracker); // Đặt chữ nhật vào mục đã chọn
        UpdateWindow(); // Vẽ lại cửa sổ
        // Cho phép chữ nhật thay đổi kích thước
        // Hàm Track() cho giá trị TRUE khi kích thước chữ nhật thay đổi
        if (aTracker.Track(this, point))
        {
            pHitItem->InvalidateItem();
            pHitItem->m_Rect = aTracker.m_rect;
            pHitItem->InvalidateItem();
            GetDocument()->SetModifiedFlag(); // đánh dấu sự thay đổi của tài liệu
        }
    }
    CView::OnLButtonDown(nFlags, point);
}
```

Đoạn trình trên gọi hàm trợ giúp `HitTestItems` để nạp giá trị ban đầu cho `pHitItem`. Hàm này duyệt toàn bộ các đối tượng OLE trong trình chứa cho đến khi tìm thấy đối tượng OLE có chữ nhật bao chứa đối tượng `point`. Đối tượng `point` được chuyển đến doi của hàm quản lý. Vậy đối tượng dưới con trỏ chuột sẽ là đối tượng bị nhấn đúp chuột. Địa chỉ của chúng được cho lại và lưu trong con trỏ cục bộ `pHitItem`. Nếu không đối tượng nào được chọn thì hàm `HitTestItems` cho lại con trỏ

giá trị NULL.

Nếu người sử dụng nhấn chuột trên đối tượng nhúng, ta sẽ tạo lập đối tượng của lớp CRectTracker. Đối tượng của lớp này là chữ nhật và được gọi là chữ nhật theo dõi. Ta có thể hiển thị chúng bằng các đường liên tục, chấm chấm hay gạch gạch. Có thể sử dụng đối tượng CRectTracker bất kỳ đâu. Đoạn trình trên đây khởi động đối tượng chữ nhật theo dõi aTracker nhờ hàm trợ giúp SetupTracker để đặt chữ nhật theo dõi có cùng kích thước với chữ nhật của đối tượng nhúng trả bởi pItem. Sau khi khởi động, đoạn trình trên gọi hàm UpdateWindow của lớp CWrxContainerView. Hàm này thừa hưởng gián tiếp từ lớp CWnd và yêu cầu vẽ ngay lại cửa sổ. Nếu thoả mãn điều kiện của lệnh if, hàm thành phần Track của đối tượng chữ nhật theo dõi được gọi. Đối của hàm là con trỏ đến cửa sổ hiện hành và vị trí của con chạy chuột hiện hành point. Hàm cho phép người sử dụng thay đổi kích thước chữ nhật theo dõi bằng di khung của chúng. Hàm Track lưu chữ nhật mới vào đối tượng chữ nhật theo dõi và cho lại giá trị TRUE khi thay đổi kích thước của chúng và cho lại giá trị FALSE trong trường hợp ngược lại. Nếu chữ nhật theo dõi thay đổi, đối tượng hiện hành được vẽ lại với kích thước cũ. Thành phần m_rect của đối tượng chữ nhật theo dõi sẽ chứa chữ nhật mới, chúng sẽ lưu trữ thành phần m_Rect của đối tượng nhúng. Cuối cùng đối tượng được vẽ lại với kích thước mới.

Hàm HitTestItems tìm mục mà người sử dụng nhấn chuột trên đó trong toàn bộ các mục nhúng của tài liệu. Hàm này có thể như sau:

```
CWrxContainerItem* CWrxContainerView::HitTestItems(CPoint aPoint)
{
    CWrxContainerDoc* pDoc = GetDocument();
    CWrxContainerItem* pItem = 0; // nơi lưu con trỏ đến mục
    // Lấy vị trí của mục thứ nhất
    POSITION aPosition = pDoc->GetStartPosition();
    while (aPosition) // Duyệt toàn bộ mục
    {
        pItem = (CWrxContainerItem*)pDoc->GetNextItem(aPosition);
        if (pItem->m_Rect.PtInRect(aPoint))
            return pItem; // Cho lại con trỏ đến mục được nhấn
    }
    return 0; // Không nhấn mục nào
}
```

Sau khi đã có con trỏ đến đối tượng tài liệu, ta tạo lập con trỏ pItem để lưu địa chỉ của đối tượng được chọn. Ta lấy vị trí của đối tượng thứ nhất trong tài liệu bằng hàm thành phần GetStartPosition của đối tượng tài liệu. Ta sẽ duyệt toàn bộ danh sách đối tượng nhúng bằng hàm thành phần GetNextItem của đối tượng tài liệu. Trong vòng lặp, thành phần m_Rect của từng đối tượng nhúng được kiểm tra nhờ hàm PtInRect của CRect để biết đối tượng aPoint có ở trong chữ

nhật hay không. Nếu có, hàm cho lại địa chỉ đối tượng nhúng. Nếu duyệt đến cuối danh sách mà không thoả mãn điều kiện trên, `aPosition` sẽ có giá trị 0 và vòng lặp kết thúc. Khi ta nhấn phím chuột vào điểm bên ngoài các đối tượng nhúng, chương trình phải bãi bỏ kích hoạt cho đối tượng hiện hành.

Khi người sử dụng nhấn chuột trên đối tượng, ta phải bãi bỏ kích hoạt cho đối tượng tích cực và kích hoạt đối tượng mới. Hàm trợ giúp `SelectItem` thực hiện việc này trong trình điều khiển `OnLButtonDown`. Hàm `SelectItem` có thể như sau:

```
void CWrxContainerView::SelectItem(CWrxContainerItem *pltem)
{
    if(m_pSelection!=0 && m_pSelection!=pltem)
        m_pSelection->Close(); // Bãi bỏ tích cực cho đối tượng
    if (m_pSelection != pltem)
    {
        if (m_pSelection) // Có lựa chọn cũ?
            OnUpdate(0, HINT_UPDATE_ITEM, m_pSelection); // cập nhật vùng
        m_pSelection = pltem; // Gán lựa chọn mới
        if (m_pSelection) // Có lựa chọn mới?
            OnUpdate(0, HINT_UPDATE_ITEM, m_pSelection);
    }
}
```

Lệnh `if` thứ nhất bãi bỏ kích hoạt của đối tượng đang được chọn (địa chỉ của chúng lưu trong `m_pSelection`) khi chúng không phải là đối tượng vừa mới chọn (địa chỉ lưu trong `pltem`). Lệnh `if` tiếp theo sẽ kiểm tra xem địa chỉ của đối tượng mới có khác với đối tượng cũ không. Nếu giống nhau, ta không phải làm gì. Nếu không ta kiểm tra địa chỉ của mục đang được chọn có khác 0 hay không để gọi hàm thành phần `OnUpdate` của quan sát. Cuối cùng ta lưu địa chỉ của đối tượng nhúng mới vào thành phần `m_pSelection`. Nếu chúng khác không ta lại gọi hàm thành phần `OnUpdate` lần nữa.

Hàm trợ giúp cuối cùng thực hiện đặt kiểu cho chữ nhật theo dõi. Mã trình của chúng như sau:

```
void CWrxContainerView::SetupTracker(CWrxContainerItem *pltem, CRectTracker *pTracker)
{
    pTracker->m_rect = pltem->m_Rect;
    if (pltem == m_pSelection)
        pTracker->m_nStyle |= CRectTracker::resizeInside;
    if (pltem->GetType() == OT_LINK)
        pTracker->m_nStyle |= CRectTracker::dottedLine;
    else
        pTracker->m_nStyle |= CRectTracker::solidLine;
    if (pltem->GetItemState() == COleClientItem::openState ||
        pltem->GetItemState() == COleClientItem::activeUIState)
```

```
pTracker->m_nStyle |= CRectTracker::hatchInside;
```

Thành phần `m_rect` của đối tượng chữ nhật theo dõi lưu trữ tọa độ của chữ nhật để biểu diễn chữ nhật theo dõi theo đơn vị thiết bị. Giá trị của chúng được gán từ thành phần `m_Rect` của đối tượng. Kiểu của đối tượng chữ nhật theo dõi được lưu trong thành phần `m_nStyle`. Các tên hằng của các thuộc tính kiểu chữ nhật theo dõi được lưu trong đối tượng `CRectTracker`. Giá trị của chúng có thể như sau:

<code>solidLine</code>	Khung chữ nhật là đường liên tục, áp dụng cho đối tượng thu đông
<code>dottedLine</code>	Khung chữ nhật là đường chấm chấm, sử dụng cho đối tượng liên kết.
<code>hatchedBorder</code>	Khung chữ nhật là đường gạch chéo, áp dụng cho đối tượng nhúng tích cực.
<code>resizeInside</code>	Điều khiển thay đổi kích thước xuất hiện trong chữ nhật.
<code>hatchInside</code>	Hiện thị gạch chéo bên trong chữ nhật, cho biết không sửa đổi được đối tượng.

Ta đã cài đặt khả năng co giãn đối tượng bằng di chuyển chữ nhật theo dõi, nhưng người sử dụng vẫn không nhận biết chúng trên màn hình. Giải quyết vấn đề này bằng thay đổi hình dạng con chạy chuột như trong các ứng dụng Windows thông thường. Vậy, ta phải xen thêm trình điều khiển xử lý thông điệp `WM_SETCURSOR`. Trình điều khiển phải có khả năng kiểm tra xem con chạy chuột có liên quan đến chữ nhật theo dõi của đối tượng đã được chọn hay không. Mã lệnh của chúng có thể như sau:

```
BOOL CWrxCContainerView::OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message)
{
    if (pWnd == this && m_pSelection)
    {
        CRectTracker aTracker; // Lập chữ nhật theo dõi
        SetupTracker(m_pSelection, &aTracker); // Đặt kiểu chữ nhật theo dõi
        //Thay đổi hình dạng con chạy chuột trên đối tượng đang chọn
        if (aTracker.SetCursor(this, nHitTest))
            return TRUE; // Nếu nhấn chuột trên chữ nhật tracker, cho lại TRUE
    }
    return CView::OnSetCursor(pWnd, nHitTest, message);
}
```

Tham số thứ nhất của hàm điều khiển là con trỏ đến cửa sổ chứa con chạy. Tham số thứ hai cho biết con chạy có trong cửa sổ quan sát hay không? Tham số thứ ba là thông điệp chuột, ta không xử lý chúng ở đây.

Kích hoạt đối tượng nhúng

Khi nhấn đúp phím chuột trên đối tượng thì đối tượng được chọn. Vậy ta phải xen thêm trình quản lý thông điệp `WM_LBUTTONDOWN` như dưới đây vào lớp `CWrxCContainerView`.

```

void CWrxContainerView::OnLButtonDbtClk(UINT nFlags, CPoint point)
{
    OnLButtonDown(nFlags, point);
    if (m_pSelection)
        m_pSelection->DoVerb((GetKeyState(VK_CONTROL) < 0) ? OLEIVERB_OPEN
        : OLEIVERB_PRIMARY, this);
    CView::OnLButtonDbtClk(nFlags, point);
}

```

Vì ta sử dụng phím trái chuột nên đoạn trình trên có gọi hàm `OnLButtonDown`. Nếu con trỏ `m_pSelection` khác không, trình sẽ gọi hàm thành phần `DoVerb` của đối tượng nhúng đang chọn. Động từ (*verb*) có ý nghĩa đặc biệt trong ngữ cảnh OLE. Chúng chỉ ra tác động của đối tượng nhúng để trả lời yêu cầu của người dùng. Hàm `GetKeyState` kiểm tra trạng thái của phím trên bàn phím. Nếu nhấn đúp phím chuột và nhấn phím `Ctrl` thì "động từ" `OLEIVERB_OPEN` được chọn, nếu không sẽ chọn "động từ" `OLEIVERB_PRIMARY`. Động từ `OLEIVERB_OPEN` sẽ mở đối tượng để sửa đổi trong cửa sổ riêng biệt của trình dịch vụ mặc dù đối tượng vẫn còn nhúng trong trình chứa. Đối tượng trong trình chứa sẽ có khung gạch chéo. Động từ `OLEIVERB_PRIMARY` kích hoạt trình dịch vụ và cho phép sửa đổi đối tượng ngay tại chỗ trong trình chứa.

Vẽ lại đa đối tượng nhúng

Ta phải mở rộng hàm `OnDraw` của lớp `CWrxContainerView` để vẽ lại các đối tượng trong tài liệu chứa. Hàm này có thể như sau:

```

void CWrxContainerView::OnDraw(CDC* pDC)
{
    CWrxContainerDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // Lấy vị trí của đối tượng nhúng thứ nhất
    POSITION aPosition = pDoc->GetStartPosition();
    while (aPosition) // thực hiện cho mỗi đối tượng trong danh sách
    {
        CWrxContainerItem* pItem = (CWrxContainerItem*)pDoc->GetNextItem(aPosition);
        pItem->Draw(pDC, pItem->m_Rect); // Vẽ đối tượng
        // Tạo chữ nhật theo dõi cho đối tượng tiếp theo
        CRectTracker aTracker; // Tạo chữ nhật chữ nhật theo dõi
        SetupTracker(pItem, &aTracker); // Đặt thuộc tính cho đối tượng hiện hành
        aTracker.Draw(pDC); // Vẽ chữ nhật chữ nhật theo dõi
    }
}

```

Đoạn trình trên duyệt toàn bộ đối tượng nhúng trong tài liệu nhờ hàm `GetNextItem` trong vòng lặp `while`. Hàm `Draw` thừa hưởng từ lớp cơ sở `COleClientItem` sẽ vẽ lại đối tượng. Ta không xây

dụng hàm này cho lớp `CWrxContainerView` vì chính trình dịch vụ vẽ đối tượng chứ không phải trình chứa. Hàm `OnDraw` của đối tượng OLE trong trình dịch vụ sẽ phát sinh ảnh đối tượng OLE trong khuôn mẫu metafile. Hàm `Draw` của `COleClientItem` có khả năng tham nhập chúng để hiển thị trong một ngữ cảnh thiết bị. Sau khi vẽ xong, đoạn trình trên tạo ra chữ nhật theo dõi có thuộc tính trên cơ sở trạng thái của đối tượng nhúng.

Xen đối tượng

Hàm `OnInsertObject` của lớp `CWrxContainerView` sau đây cho phép xen đối tượng mới vào trình chứa.

```
void CWrxContainerView::OnInsertObject()
{
    // Kích hoạt hộp thoại Xen đối tượng chuẩn để lấy thông tin cho
    // đối tượng mới CWrxContainerItem
    COleInsertDialog dlg;
    if (dlg.DoModal() != IDOK)    return;
    BeginWaitCursor();
    CWrxContainerItem* pItem = NULL;
    TRY
    {
        // Tạo đối tượng mới kết nối với tài liệu.
        CWrxContainerDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        pItem = new CWrxContainerItem(pDoc);
        ASSERT_VALID(pItem);
        // Khởi động đối tượng.
        if (!dlg.CreateItem(pItem))    AfxThrowMemoryException(); // xuất hiện lỗi?
        ASSERT_VALID(pItem);
        pItem->UpdateLink(); // Hiển thị lại đối tượng
        pItem->GetServerSize(); // cập nhật kích thước đối tượng
        if (dlg.GetSelectionType() == COleInsertDialog::createNewItem)
            pItem->DoVerb(OLEIVERB_SHOW, this);
        ASSERT_VALID(pItem);
        // Đối tượng xen cuối cùng ở trạng thái được chọn
        SelectItem(pItem); // Chọn đối tượng xen cuối cùng
        pItem->InvalidateItem();
        m_pSelection = pItem;
        pDoc->UpdateAllViews(NULL);
    }
    CATCH(CException, e)
    {
        if (pItem != NULL)
```

```

    {
        ASSERT_VALID(pItem);
        pItem->Delete();
    }
    AfxMessageBox(IDP_FAILED_TO_CREATE);
}
END_CATCH
EndWaitCursor();
}

```

10.5 CÀI ĐẶT TRÌNH DỊCH VỤ OLE

Tương tự như trình chứa OLE, hãy sử dụng *AppWizard* để xây dựng trình dịch vụ OLE. Chúng làm giảm thiểu đáng kể công việc cài đặt. Giả sử phải xây dựng trình dịch vụ đầy đủ có tên *Sketcher*, ta sẽ xây dựng *tệp project* có tên *Sketcher*. Trong quá trình phát sinh các tệp của ứng dụng hãy chọn thuộc tính *Full_server* tại bước 3. Chọn phím *Advanced* trong bước 4. Nhập "skd" vào hộp soạn thảo *File extension*. Nhấn phím *Finish* để phát sinh các tệp chương trình *SketcherDoc.cpp*, *SketcherView.cpp*,

10.5.1 Tăng cường chức năng vẽ cho chương trình

Ta hãy tăng khả năng vẽ bằng phím chuột cho trình dịch vụ vừa phát sinh trên đây. Hãy nhập các dòng mã trình vào các tệp như sau:

Tệp *Elements.CPP*

```

// Cài đặt lớp đường vẽ
#include "stdafx.h"
#include <math.h>
#include "OurConstants.h"
#include "Elements.h"
IMPLEMENT_SERIAL(CElement, CObject, VERSION_NUMBER)
IMPLEMENT_SERIAL(CLine, CElement, VERSION_NUMBER)
// Cấu tử CLine
CLine::CLine(const CPoint& Start, const CPoint& End)
{
    m_StartPoint = Start;    // Điểm đầu của đường
    m_EndPoint = End;        // Điểm cuối đường
    // Định nghĩa chữ nhật
    m_EnclosingRect = CRect(Start, End);
    m_EnclosingRect.NormalizeRect();
}

```

```

//-----
void CLine::Draw(CDC* pDC, const CElement* pElement) const
{
    // Vẽ đường
    pDC->MoveTo(m_StartPoint);
    pDC->LineTo(m_EndPoint);
}
//-----
// Lấy chữ nhật bao đoạn thẳng
CRect CElement::GetBoundRect() const
{
    CRect BoundingRect;          // Đối tượng lưu chữ nhật bao
    BoundingRect = m_EnclosingRect;
    // Tăng độ rộng chữ nhật
    int Offset = 1;
    BoundingRect.InflateRect(Offset, Offset);
    return BoundingRect;
}
//-----
void CElement::Serialize(CArchive& ar)
{
    CObject::Serialize(ar);      // Gọi hàm lớp cơ sở
    if (ar.IsStoring())
        ar << m_EnclosingRect ;
    else
        ar >> m_EnclosingRect ;
}
//-----
void CLine::Serialize(CArchive& ar)
{
    CElement::Serialize(ar);     // Gọi hàm của lớp cơ sở
    if (ar.IsStoring())
        ar << m_StartPoint << m_EndPoint;
    else
        ar >> m_StartPoint >> m_EndPoint;
}
//-----

```

Tập Elements.H

```

#if !defined(Elements_h)
#define Elements_h
class CElement : public CObject
{

```



```

DECLARE_SERIAL(CElement)
protected:
    CRect m_EnclosingRect; // Chữ nhật bao quanh đường
public:
    virtual ~CElement(){} // Hủy tử ảo
    // Hàm vẽ ảo
    virtual void Draw(CDC* pDC, const CElement* pElement = 0) const {}
    CRect GetBoundRect() const; // Lấy chữ nhật bao đoạn thẳng
    virtual void Serialize(CArchive& ar);
protected:
    CElement(){} // Cấu tử mặc định
};
// Lớp định nghĩa đối tượng đường
class CLine : public CElement
{
    DECLARE_SERIAL(CLine)
public:
    // Hiển thị đường
    virtual void Draw(CDC* pDC, const CElement* pElement = 0) const;
    // Cấu tử
    CLine(const CPoint& Start, const CPoint& End);
    virtual void Serialize(CArchive& ar);
protected:
    CPoint m_StartPoint;
    CPoint m_EndPoint;
    CLine(){}
};
#endif //!defined(Elements_h)
//-----

```

Tệp OurConstants.H

```

const WORD LINE = 101U;
const UINT VERSION_NUMBER = 1;
//-----

```

Bước tiếp theo là sửa đổi các tệp của chương trình, bao gồm:

Cài đặt dữ liệu tài liệu và các hàm ghép nối

Xen các thành phần dữ liệu protected sau vào lớp CSketcherDoc.

```

WORD m_Element;
CSize m_DocSize;
CtypedPtrList <CObList, CElement*> m_ElementList; //Danh sách đoạn thẳng

```

Mở tệp StdAfx.h, xen dòng lệnh `#include <afxtempl.h>` vào sau dòng lệnh `#include <afxole.h>`. Sửa đổi cấu tử CSketcherDoc như sau:

```
CSketcherDoc::CSketcherDoc()
{
    EnableCompoundFile();
    m_Element = LINE;          // Khởi động kiểu đường
    m_DocSize = CSize(200,200); // Kích thước tài liệu
}
```

Vì có sử dụng tên hằng LINE cho nên xen thêm dòng lệnh `#include "OurConstants.h"` vào đầu tệp SketcherDoc.cpp. Hủy tử của lớp phải bãi bỏ đối tượng danh sách đường thẳng.

```
CSketcherDoc::~CSketcherDoc()
{
    // Lấy vị trí đầu danh sách
    POSITION aPosition = m_ElementList.GetHeadPosition();
    // xoá từng thành phần
    while (aPosition)
        delete m_ElementList.GetNext(aPosition);
    m_ElementList.RemoveAll(); // Xoá toàn bộ con trỏ
}
```

Tăng cường các hàm ghép nối vào vùng public cho lớp tài liệu như sau:

```
WORD GetElementType() const
{ return m_Element; }
void AddElement(CElement* pElement)
{ m_ElementList.AddTail(pElement); }
POSITION GetListHeadPosition() const
{ return m_ElementList.GetHeadPosition(); }
CElement* GetNext(POSITION& aPos) const
{ return m_ElementList.GetNext(aPos); }
CSize GetDocSize() const
{ return m_DocSize; }
```

Vì có sử dụng đến lớp CElement, vậy xen thêm dòng `#include <Elements.h>` vào tệp SketcherDoc.h. Tiếp theo phải cài đặt hàm vào-ra để khung ứng dụng sử dụng lưu tài liệu khi đối tượng nhúng không còn ở trạng thái tích cực. Khi kích hoạt lại đối tượng, tài liệu sẽ được khôi phục.

```
void CSketcherDoc::Serialize(CArchive& ar)
{
    m_ElementList.Serialize(ar); // Vào ra danh sách đoạn thẳng
    if (ar.IsStoring())
```

```

        ar << m_Element << m_DocSize;
    else
        ar >> m_Element >> m_DocSize;
}

```

Cài đặt thực đơn

Ta sẽ xen thêm thực đơn Element, mục thực đơn Line vào IDR_SKETCHTYPE của trình dịch vụ này. Tiếp theo ta xen thêm trình điều khiển COMMAND và UPDATE_COMMAND_UI vào lớp CSketcherDoc cho mục thực đơn Line. Hãy sử dụng *ClassWizard* để làm nhẹ công việc này. Trình điều khiển chọn thực đơn có thể như sau:

```

void CSketcherDoc::OnElementLine()
{
    m_Element = LINE;    // Kiểu vẽ là đường
}

```

Còn trình điều khiển cập nhật chọn mục thực đơn là:

```

void CSketcherDoc::OnUpdateElementLine(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_Element==LINE);
}

```

Ngoài thực đơn IDR_SKETCHTYPE, ta còn thấy hai thực đơn khác trong trình dịch vụ. Thực đơn IDR_SKETCHTYPE_SRVR_EMB được sử dụng khi trình dịch vụ mở hoàn toàn để sửa đổi đối tượng nhúng trong trình chứa. Còn thực đơn IDR_SKETCHTYPE_SRVR_IP được sử dụng khi sửa đổi đối tượng tại trình chứa. Hãy sao chép mục thực đơn Element của thực đơn IDR_SKETCHTYPE vào hai thực đơn này.

File	Edit	View	Element	Windows	Help
------	------	------	---------	---------	------

IDR_SKETCHTYPE_SRVR_EMB

Edit	View	Element		Help
------	------	---------	--	------

IDR_SKETCHTYPE_SRVR_IP

Cài đặt các chức năng quan sát

Xen các dòng khai báo thành phần dữ liệu sau đây vào vùng *protected* của lớp CSketcherView.

```

CPoint m_FirstPoint;    // Điểm đầu của đoạn thẳng

```

```
CPoint m_SecondPoint;    // Điểm thứ hai của đoạn thẳng
CElement* m_pTempElement;
```

Vì có sử dụng đến lớp CElement, vậy xen thêm dòng #include <Elements.h> vào tệp SketcherView.h. Các thành phần dữ liệu này được nạp giá trị ban đầu tại cấu tử của lớp.

```
CSketcherView::CSketcherView()
{
    m_FirstPoint = CPoint(0,0);
    m_SecondPoint = CPoint(0,0);
    m_pTempElement = 0;
}
```

Các hàm quản lý thông điệp cần xen vào lớp quan sát tại thời điểm này là WM_LBUTTONDOWN, WM_LBUTTONUP và WM_MOUSEMOVE. Hãy sử dụng *ClassWizard* để làm nhẹ công việc này. Các hàm có thể như sau:

```
void CSketcherView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CClientDC aDC(this);
    OnPrepareDC(&aDC);
    aDC.DPtoLP(&point);    // Chuyển đổi đơn vị thiết bị sang đơn vị logic
    m_FirstPoint = point;
    SetCapture();
}

void CSketcherView::OnLButtonUp(UINT nFlags, CPoint point)
{
    CSketcherDoc* pDoc = GetDocument();
    if (this == GetCapture())    ReleaseCapture();
    // Nếu có đoạn thẳng, thêm chúng vào tài liệu
    if (m_pTempElement)
    {
        pDoc->AddElement(m_pTempElement);
        pDoc->UpdateAllViews(0, 0, m_pTempElement);
        m_pTempElement = 0;
        pDoc->NotifyChanged();    // Thông báo cho trình chứa...
        pDoc->SetModifiedFlag();    // ...và trình dịch vụ
    }
}

void CSketcherView::OnMouseMove(UINT nFlags, CPoint point)
{
    CClientDC aDC(this);
    OnPrepareDC(&aDC);
    if ((nFlags & MK_LBUTTON) && (this == GetCapture()))
```

```

{
    aDC.DPtoLP(&point);
    m_SecondPoint = point;
    if(m_pTempElement)
    {
        aDC.SetROP2(R2_NOTXORPEN);    // Chọn chế độ vẽ
        // Vẽ lại đoạn thẳng cũ để xoá chúng
        m_pTempElement->Draw(&aDC);
        delete m_pTempElement;    // xoá đoạn thẳng cũ
        m_pTempElement = 0;
    }
    m_pTempElement = CreateElement(); // Tạo đoạn thẳng mới
    m_pTempElement->Draw(&aDC);    // Vẽ đoạn thẳng
}
}

```

Hàm trên đây có gọi hàm `CreateElement`, ta hãy cài đặt chúng:

```

CElement* CSketcherView::CreateElement()
{
    CSketcherDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);    // Kiểm tra xem con trỏ có đúng?
    if(pDoc->GetElementType() == LINE)
        return new CLine(m_FirstPoint, m_SecondPoint);
    else {
        AfxMessageBox("Bad Element code", MB_OK);
        AfxAbort();
        return NULL;
    }
}

```

Cài đặt hiển thị dữ liệu

Việc vẽ các đoạn thẳng được thực hiện trong hàm `OnDraw` của lớp `CSketcherView`. Cài đặt chúng như sau:

```

void CSketcherView::OnDraw(CDC* pDC)
{
    CSketcherDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    POSITION aPos = pDoc->GetListHeadPosition();
    CElement* pElement = 0;    // Lưu con trỏ đoạn thẳng
    while (aPos)
    {

```

```
pElement = pDoc->GetNext(aPos); // Lấy con trỏ đoạn thẳng hiện hành
```

```
// Nếu đoạn thẳng quan sát được...
```

```
if (pDC->RectVisible(pElement->GetBoundRect()))
```

```
    pElement->Draw(pDC);        // ...vẽ chúng
```

Hàm OnUpdate cũng phải được cài đặt cho lời gọi hàm UpdateAllViews khi xen thêm một đoạn thẳng mới vào tài liệu. Đoạn mã trình của hàm này được xen vào lớp CSketcherView.

```
void CSketcherView::OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint)
```

```
{
```

```
    if (pHint) {
```

```
        CClientDC aDC(this);
```

```
        OnPrepareDC(&aDC);
```

```
        // Lấy kích thước chữ nhật và chuyển đổi hệ tọa độ
```

```
        CRect aRect = static_cast<CElement*>(pHint)->GetBoundRect();
```

```
        aDC.LPtoDP(aRect);
```

```
        aRect.NormalizeRect();
```

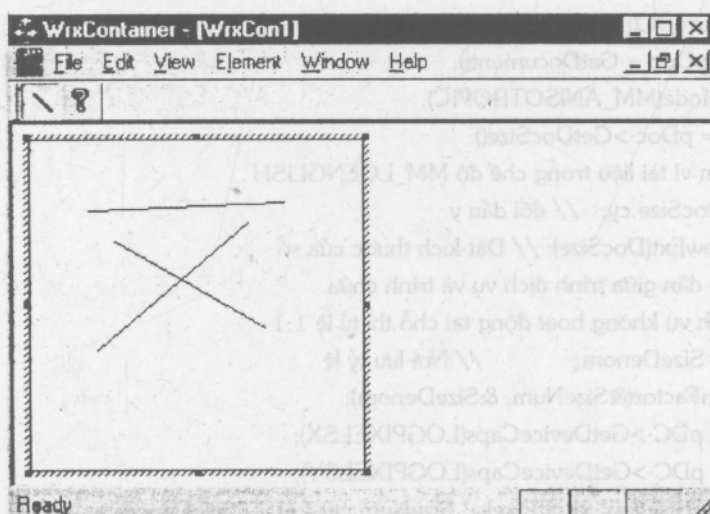
```
        InvalidateRect(aRect);
```

```
    }
```

```
    else    InvalidateRect(0);
```

```
}
```

Chú ý rằng phải xen thêm lệnh #include <OurConstants.h> dưới dòng lệnh #include <Sketcher.h> trong tệp SketcherView.cpp.



Hình 10.3 Trình chứa OLE và trình dịch vụ OLE

Đến đây ta có thể dịch, chạy chương trình Sketcher có cơ chế OLE. Hãy khởi động trình chứa WrxContainer vừa xây dựng trong chương này. Chọn mục thực đơn *Edit/Insert New Object ...* để xuất hiện danh sách các dịch vụ OLE, trong đó có *Sketcher Document*. Nếu ta chọn chúng thì đối tượng Sketcher được nạp để vẽ các đoạn thẳng.

10.5.2 Cài đặt các chức năng dịch vụ

Đối tượng dịch vụ là bản của lớp COleServerItem trong trình dịch vụ. Chúng có trách nhiệm vẽ đối tượng nhúng khi không còn tích cực. Trong thí dụ Sketcher, AppWizard tự động phát sinh lớp CSketcherSrvItem suy diễn từ COleServerItem để biểu diễn đối tượng nhúng trong Sketcher. Mỗi khi đối tượng nhúng được sửa đổi, hàm OnDraw trong lớp CSketcherView được chuyển tới trình chứa để hiển thị. Khi đối tượng nhúng không tích cực, trình chứa yêu cầu đối tượng CSketcherSrvItem vẽ chúng. Vậy, ta phải cài đặt chức năng này.

Cài đặt đối tượng nhúng

Đối tượng CSketcherSrvItem có hai nhiệm vụ chính: vẽ đối tượng nhúng khi không sửa đổi tại chỗ và cung cấp kích thước tài liệu khi khung ứng dụng yêu cầu. Hàm thành phần OnDraw của lớp CSketcherSrvItem thực hiện vẽ còn hàm thành phần OnGetExtent cung cấp kích thước tài liệu.

Cách tốt nhất để vẽ đối tượng nhúng là sử dụng chế độ ánh xạ MM_ANISOTROPIC. Hàm thành phần OnPrepareDC của lớp CSketcherView thực hiện chuyển đổi chế độ ánh xạ.

```
void CSketcherView::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    CView::OnPrepareDC(pDC, pInfo);
    CSketcherDoc* pDoc = GetDocument();
    pDC->SetMapMode(MM_ANISOTROPIC);
    CSize DocSize = pDoc->GetDocSize();
    // Chiều y là âm vì tài liệu trong chế độ MM_LOENGLISH
    DocSize.cy = -DocSize.cy; // đổi dấu y
    pDC->SetWindowExt(DocSize); // Đặt kích thước cửa sổ
    // Tính tỷ lệ co giãn giữa trình dịch vụ và trình chứa
    // Nếu trình dịch vụ không hoạt động tại chỗ thì tỷ lệ 1:1
    CSize SizeNum, SizeDenom; // Nơi lưu tỷ lệ
    pDoc->GetZoomFactor(&SizeNum, &SizeDenom);
    int xLogPixels = pDC->GetDeviceCaps(LOGPIXELSX);
    int yLogPixels = pDC->GetDeviceCaps(LOGPIXELSY);
    int xExtent = (DocSize.cx * xLogPixels * SizeNum.cx) / (100*SizeDenom.cx);
    int yExtent = (DocSize.cy * yLogPixels * SizeNum.cy) / (100*SizeDenom.cy);
    pDC->SetViewportExt(xExtent, -yExtent); // Đặt kích thước vùng nhìn
}
```

Hiển thị đối tượng nhúng

Hàm thành phần OnDraw của lớp CSketcherSrvItem hiển thị đối tượng nhúng.

```

BOOL CSketcherSrvItem::OnDraw(CDC* pDC, CSize& rSize)
{
    UNREFERENCED_PARAMETER(rSize);
    CSketcherDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // Đặt chế độ ánh xạ và kích thước
    pDC->SetMapMode(MM_ANISOTROPIC);
    CSize DocSize = pDoc->GetDocSize(); // Lấy kích thước tài liệu
    DocSize.cy = -DocSize.cy;
    pDC->SetWindowOrg(0,0);
    pDC->SetWindowExt(DocSize);
    // Mã trình vẽ đối tượng ở đây
    POSITION aPos = pDoc->GetListHeadPosition();
    CElement* pElement = 0;
    while (aPos) {
        pElement = pDoc->GetNext(aPos);
        if (pDC->RectVisible(pElement->GetBoundRect()))
            pElement->Draw(pDC);
    }
    return TRUE;
}

```

Sau khi chọn chế độ ánh xạ, hàm OnDraw lấy kích thước tài liệu để đặt lại kích thước cửa sổ. Giả sử mã trình của Sketcher có chế độ ánh xạ là MM_LOENGLISH, gốc tọa độ ở góc trên-trái của vùng làm việc. Hàm trên có hàm chuyển đổi sang chế độ MM_ANISOTROPIC để phù hợp với qui định chuẩn khi vẽ đối tượng dịch vụ nhúng. Việc vẽ lại đối tượng được thực hiện trong hàm OnDraw. Hàm không vẽ trực tiếp trên màn hình, mà các hàm GDI sẽ tạo ra ảnh tài liệu trong metafile độc lập thiết bị. Sau đó, kích thước vùng nhìn được hiệu chỉnh trước khi hiển thị trong chữ nhật của cửa sổ trình chứa. Kết quả là ảnh sẽ bị biến dạng. Nếu muốn giữ nguyên hình dạng thì sử dụng chế độ ánh xạ MM_ISOTROPIC. Trong chế độ ánh xạ này ta không đặt giá trị cho tham số rSize của hàm OnDraw.

Lấy kích thước của đối tượng nhúng

Khung ứng dụng gọi hàm thành phần OnGetExtent của lớp đối tượng nhúng trong trình dịch vụ để lấy kích thước tài liệu. Mã trình của hàm này như sau:

```

BOOL CSketcherSrvItem::OnGetExtent(DVASPECT dwDrawAspect, CSize& rSize)

```



```

{
    if (dwDrawAspect != DVASPECT_CONTENT)
        return COleServerItem::OnGetExtent(dwDrawAspect, rSize);
    // Gọi hàm CSketcherSrvrItem::OnGetExtent để lấy kích thước theo đơn vị
    // HIMETRIC.
    CSketcherDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    rSize = pDoc->GetDocSize();
    CClientDC aDC(0);
    aDC.SetMapMode(MM_ANISOTROPIC);
    // Đặt giá trị kích thước của sổ là 1 inch trong chế độ MM_LOENGLISH
    aDC.SetWindowExt(100, -100);    // Kích thước chiều y là âm
    // Đặt giá trị kích thước vùng nhìn pixels/inch
    aDC.SetViewportExt(aDC.GetDeviceCaps(LOGPIXELSX), aDC.GetDeviceCaps(LOGPIXELSY));
    aDC.LPtoHIMETRIC(&rSize);    // chuyển đổi kích thước tài liệu thành //HIMETRIC
    return TRUE;
}

```

Để chương trình có thể dịch được, hãy xen thêm lệnh `#include <Elements.h>` vào đầu tệp `SrvrItem.cpp`.

Chương trình Sketcher trên đây vừa chạy độc lập vừa có khả năng chạy với trình chứa. Việc đăng ký với Windows được tự động thực hiện.

CHƯƠNG 11

ĐIỀU KHIỂN ActiveX

Chương này sẽ trình bày các quan niệm về ActiveX và các hoạt động của chúng. Sau đó sẽ trình bày các bước tạo lập ActiveX đơn giản bằng Visual C++.

11.1 ActiveX VÀ OLE

OLE và ActiveX chỉ là những thuật ngữ thương mại của Microsoft. Thuật ngữ OLE thảo luận trong chương trước là tên tiên thân của ActiveX. Chúng liên quan đến khả năng nhúng tài liệu được tạo lập từ một ứng dụng vào tài liệu được tạo lập từ một ứng dụng khác. Quan niệm ban đầu về OLE đã thay đổi cơ bản theo thời gian để ngày nay hình thành khái niệm mô hình đối tượng thành phần (*Component Object Model - COM*). Về cơ bản thì ý tưởng của COM và OLE đều là đặc tả ghép nối tổng quan để xây dựng các thành phần phần mềm cho phép liên kết ảo với nhau trong một ngữ cảnh.

ActiveX là khái niệm do Microsoft đặt ra cho công nghệ có thể áp dụng cho Internet. Vì vậy, như OLE, ActiveX hình thành trên cơ sở COM. Giữa OLE và ActiveX có nhiều điểm giống nhau và có thể sử dụng thay thế cho nhau trong nhiều trường hợp. Trước hết ta sẽ khảo sát khái niệm OLE và ActiveX, sau đó là áp dụng cho các điều khiển.

11.1.1 Điều khiển OLE là gì?

Điều khiển OLE là tiện ích cho phép lập trình viên sử dụng mã trình của người khác. Thí dụ, lập trình viên Visual Basic có thể sử dụng các điều khiển tạo ra từ C++ trong mã trình của họ. Điều khiển OLE còn được gọi là OCX vì phần mở rộng tên tệp của chúng là .ocx. Điều khiển OLE cho khả năng cài đặt phần mềm trên cơ sở thành phần có sử dụng COM để liên lạc. Thực tế đòi hỏi các ứng dụng ngày càng phức tạp. Chúng yêu cầu tích hợp các ứng dụng từ tập các thành phần mặc dù các thành phần đó được xây dựng độc lập. Khi phát triển ứng dụng tuân thủ chuẩn trình chứa OLE thì có thể sử dụng trình dịch vụ OLE. Hạn chế của trình OLE là trình chứa OLE không biết trình dịch vụ làm cái gì, và không có cơ chế để lập trình viên liên lạc với chúng. Điều khiển OLE lại khác, chúng có khả năng liên lạc thường xuyên với trình chứa.

11.1.2 Điều khiển ActiveX là gì?

Điều khiển ActiveX là điều khiển thoả mãn hai điều kiện sau: có khả năng liên lạc với trình

chứa của nó thông qua ghép nối COM có tên **IUnknown** và có khả năng tạo lập đầu vào riêng trong bộ đăng ký hệ thống (*System Registry*). Như trình bày trong chương trước, mỗi đối tượng COM phải cài đặt ghép nối có tên **IUnknown**. Tuy nhiên có nhiều đối tượng COM (thí dụ trình chứa và trình dịch vụ OLE) có cài đặt ghép nối khác. Vậy, một điều khiển COM sẽ là điều khiển ActiveX chừng nào chúng có khả năng tạo ra các đầu vào trong đăng ký hệ thống.

Thí dụ trong chương trước đã cài đặt trình dịch vụ OLE và trình chứa OLE bằng MFC. Mặc dù ta đã không quan sát chi tiết các ghép nối trong mã trình MFC để cài đặt trình chứa và trình dịch vụ OLE, nhưng chắc chắn rằng mỗi trình dịch vụ hay trình chứa OLE đòi hỏi cài đặt tập ghép nối để tương tác với nhau theo cách riêng. Tương tự, điều khiển OLE phải cài đặt tập giao diện riêng. Vì trình dịch vụ và điều khiển OLE là đối tượng COM cho nên chúng phải cài đặt **IUnknown**. Đồng thời chúng đều có khả năng tạo ra đầu vào đăng ký riêng, cho nên chúng là điều khiển ActiveX. Ta có thể nói rằng tất cả các điều khiển OLE là điều khiển ActiveX, nhưng không phải tất cả các điều khiển ActiveX đều là điều khiển OLE.

Trong tài liệu Visual C++ đã sử dụng tên điều khiển OLE và điều khiển ActiveX thay cho nhau. Rất nhiều lớp của MFC vẫn còn sử dụng tên "điều khiển OLE".

11.1.3 Điều khiển OLE làm việc như thế nào?

Trước hết, điều khiển OLE liên lạc với môi trường sử dụng chúng thông qua tập ghép nối OLE chuẩn, riêng biệt cho điều khiển OLE. Chuẩn cho điều khiển OLE là phần mở rộng của chuẩn liên quan đến các tài liệu đa hợp OLE. Ta có thể dễ dàng sử dụng lại điều khiển OLE trong các ngữ cảnh khác nhau vì trình sử dụng điều khiển OLE sẽ sử dụng cùng các ghép nối mà không phải quan tâm đến điều khiển làm cái gì. Chương trình sử dụng điều khiển OLE được gọi là trình chứa điều khiển OLE (*OLE Control Container*). Chúng trợ giúp ghép nối chuẩn để liên lạc với điều khiển OLE. Trình chứa OLE là trình ứng dụng thông thường nhưng có sử dụng một hay nhiều điều khiển OLE khi cài đặt. Vì điều khiển OLE sử dụng ghép nối OLE cho nên chúng có thể được sử dụng cho bất kỳ chương trình nào được thiết kế để hoạt động như trình chứa điều khiển OLE. Khi cài đặt điều khiển OLE bằng Visual C++ thì chúng cũng có thể được sử dụng cho bất kỳ ứng dụng nào được viết bằng ngôn ngữ khác nếu chúng trợ giúp ghép nối điều khiển OLE chuẩn.

Lợi thế lớn nhất của điều khiển OLE so với trình dịch vụ OLE là khả năng tích hợp với trình chứa. Có ba cách tương tác giữa điều khiển OLE và trình chứa chúng: khả năng truyền tải dữ liệu đến/từ trình chứa OLE; điều khiển OLE trợ giúp ghép nối để trình chứa chúng có thể ứng xử; cuối cùng là điều khiển có thể gửi thông điệp tới trình chứa chúng. Tên gọi tương ứng với các cơ chế liên quan với ba khả năng này là: thuộc tính (*property*), phương pháp (*method*) và sự kiện (*event*).

Thuộc tính (*property*)

Thuộc tính là các biến chỉ ra các vấn đề về điều khiển OLE. Mặc dù chúng có tên, nhưng thuộc tính được nhận dạng bởi các giá trị nguyên gọi là DispID, viết tắt của Dispatch ID. Trong trường hợp thuộc tính chuẩn thì DispID là các số âm. Có ba loại thuộc tính được sử dụng để liên lạc giữa điều khiển OLE và trình chứa chúng: thuộc tính bao quanh, thuộc tính điều khiển và thuộc tính mở rộng.

Thuộc tính bao quanh (*Ambient Property*)

Thuộc tính này chỉ ra các thông tin về môi trường do trình chứa cung cấp. Điều khiển không thể thay đổi thuộc tính này nhưng có thể sử dụng giá trị để tích hợp tốt hơn với trình chứa. MFC định nghĩa tên hằng cho 18 thuộc tính bao quanh chuẩn trong tệp Olectl.h. Chúng đều có giá trị DispID âm. Một vài thuộc tính bao quanh hay sử dụng là:

BackColor	Chỉ ra màu nền, giá trị RGB do trình chứa sử dụng
DisplayName	Chỉ ra tên của điều khiển sử dụng cho thông báo lỗi
Font	Cơ chữ được trình chứa sử dụng
ForeColor	Màu nền (giá trị RGB) để trình chứa hiển thị đồ họa
ScaleUnits	Tên của đơn vị tọa độ được trình chứa sử dụng
TextAlign	Cho biết trình chứa hiển thị văn bản trong điều khiển như thế nào.

Mỗi DispID của thuộc tính bao quanh trên đây đều có tên hằng bằng cách gắn theo xâu DISPID_AMBIENT_. Thí dụ, DISPID_AMBIENT_FORECOLOR là DispID của thuộc tính ForeColor. Khi cài đặt điều khiển OLE và trình chứa OLE không bắt buộc phải sử dụng các thuộc tính bao quanh này.

Thuộc tính điều khiển (*Control Property*)

Thuộc tính điều khiển là các thuộc tính do điều khiển gán giá trị, đồng thời chúng cho thông tin về điều khiển. Chúng có thể là thuộc tính bất kỳ liên quan đến điều khiển, tuy nhiên chúng cũng có tập thuộc tính chuẩn. Nếu cài đặt điều khiển bằng *MFC ActiveX Control Wizard* thì lớp cơ sở của điều khiển là COleControl. Lớp này cài đặt chín đặc tính điều khiển chuẩn (xem bảng 11.1), chúng được gọi là thuộc tính có sẵn (*stock*).

DispID của các thuộc tính này có thể được xác định bằng tên hằng, tạo bởi gắn thêm xâu DISPID_ vào tên thuộc tính. Thí dụ, tên hằng cho thuộc tính Font là DISPID_FONT. Các tên hằng này được định nghĩa trong Olectl.h.

Thông thường ta phải cài đặt thêm thuộc tính không chuẩn cho điều khiển. Chúng có thể là bất cứ cái gì ta cần cung cấp để làm phương tiện thích nghi với ứng xử của điều khiển.

Bảng 11.1 Các đặc tính điều khiển chuẩn của lớp COleControl

<i>Đặc tính</i>	<i>Chức năng</i>
BackColor	Màu nền cho điều khiển.
Appearance	Nếu có giá trị FALSE thì điều khiển hiển thị 2 chiều, nếu có giá trị TRUE hiển thị ba chiều.
BorderStyle	Điều khiển có hiển thị khung hay không?
Font	Font hiện hành cho điều khiển
ForeColor	Điều khiển sử dụng màu này để hiển thị nền đồ họa
Enabled	Nếu có giá trị TRUE, điều khiển tích cực
HWND	Chỉ danh cửa sổ điều khiển
Text	Giá trị cho hộp ký tự, hộp danh sách, hộp combo trong điều khiển.
Caption	Tiêu đề của điều khiển.

Thuộc tính mở rộng (Extended Property)

Thuộc tính mở rộng là thuộc tính áp dụng cho điều khiển nhưng do trình chứa gán giá trị. Điều khiển có khả năng tham nhập thuộc tính mở rộng do trình chứa xác định. Có bốn thuộc tính mở rộng với các tên *Visible*, *Parent*, *Cancel* và *Default*.

Trang thuộc tính (property page)

Trang thuộc tính là hộp thoại để hiển thị ghép nối cho việc sửa đổi các thuộc tính sao cho người lập trình có thể thay đổi giá trị của chúng. Với các điều khiển phức tạp, trang thuộc tính (*property page*) được sử dụng. Mỗi trang có một nhóm thuộc tính liên quan. Một vài trang được tổ chức thành một tấm (*property sheet*). MFC có lớp CPropertySheet để định nghĩa tấm thuộc tính, và lớp CPropertyPage định nghĩa trang. Ta có thể sử dụng điều khiển trên mỗi trang thuộc tính để thay đổi giá trị thuộc tính.

Phương pháp (method)

Ở đây, phương pháp là hàm kích hoạt được trong điều khiển để thực hiện một số công việc khi trả lời các yêu cầu từ ngoài. Bảng 11.2 liệt kê các phương pháp được định nghĩa sẵn trong lớp COleControl.

Bảng 11.2 Các phương pháp định nghĩa sẵn trong lớp COleControl

<i>Phương pháp</i>	<i>Chức năng</i>
Refresh	Yêu cầu điều khiển hiển thị lại
DoClick	Mô phỏng điều khiển khi nhấn phím trái chuột

Ta có thể cài đặt thêm các phương pháp để điều khiển thực hiện khi có tác động. Khả năng điều

khuyến OLE tác động trở lại các thuộc tính bao quanh, khả năng trình chứa gọi phương pháp của điều khiển để tác động trên các thao tác của điều khiển được gọi là Automation. Ta có thể cài đặt *Automation* trong trình dịch vụ OLE để cung cấp cơ chế lập trình để trình chứa tương tác với trình dịch vụ.

Sự kiện (event)

Sự kiện là tín hiệu gửi từ điều khiển OLE đến trình chứa như kết quả của tác động trên điều khiển, hay khi điều khiển thu nhận thông điệp Windows. Sự kiện điều khiển có tham số kèm theo để cung cấp thêm thông tin về chúng. Phải cài đặt các hàm cho trình chứa để phục vụ các sự kiện này. Bảng 11.3 mô tả các sự kiện điều khiển OLE chuẩn hay được sử dụng.

Bảng 11.3 Các sự kiện điều khiển OLE chuẩn hay được sử dụng

<i>Sự kiện</i>	<i>Xuất hiện khi</i>
Click	Xuất hiện khi phím chuột bị nhấn và nhả trên điều khiển
DbClick	Xảy ra khi điều khiển bị nhấn đúp
KeyDown	Xảy ra khi phím bị nhấn
KeyPress	Xảy ra khi nhận thông điệp WM_CHAR
KeyUp	Xảy ra khi phím bị nhả
MouseDown	Xảy ra khi phím chuột bị nhấn và con chạy trên điều khiển
MouseMove	Xảy ra khi con chạy chuột chạy trên điều khiển
MouseUp	Xảy ra khi phím chuột bị nhả trên điều khiển
Error	Ra hiệu cho trình chứa khi có lỗi xảy ra

Các sự kiện nói trên đều được lớp COleControl trợ giúp.

Ghép nối với điều khiển OLE

Để trình chứa có thể sử dụng đặc tính, sự kiện và phương pháp của điều khiển, ta phải mô tả chúng từ ngoài. Các điều khiển được cài đặt bằng *MFC ActiveXControlWizard* sẽ được mô tả trong tệp thư viện kiểu, chúng có phần mở rộng là .tlb. Tệp này được phát sinh khi định nghĩa thành phần ghép nối bằng ngôn ngữ mô tả đối tượng (*Object Description Language - ODL*). ODL là công cụ định nghĩa ghép nối cho OLE. Một cách khác để định nghĩa ghép nối là sử dụng ngôn ngữ định nghĩa ghép nối (*Interface Definition Language - IDL*). Microsoft đã tích hợp ODL và IDL thành MIDL (*Microsoft IDL*).

Khi sử dụng *ClassWizard*, ta không còn bận tâm nhiều đến chi tiết của MIDL khi cài đặt đặc tính hay thành phần ghép nối cho điều khiển. Trong tệp .odl ta thấy các lệnh liên quan đến DispID cho đặc tính điều khiển và các biến trong mã trình. Việc làm tương tự như trên khi áp dụng cho

DispID của phương pháp, sự kiện. Thư viện kiểu của điều khiển được phát sinh động khi dịch chương trình.

11.2 CÀI ĐẶT ĐIỀU KHIỂN ActiveX

Giả sử ta phải cài đặt mô hình đèn tín hiệu giao thông như điều khiển ActiveX. Tín hiệu giao thông tự động đổi màu sau mỗi chu kỳ. Cách dễ nhất để cài đặt điều khiển ActiveX (OLE) cơ sở là sử dụng *ControlWizard* của Visual C++.

11.2.1 Phát sinh điều khiển ActiveX cơ sở

Sử dụng *ControlWizard* để phát sinh các tệp cho ứng dụng mô hình tín hiệu giao thông.

Cấu trúc chương trình

Hãy quan sát *ClassView* của Visual C++ ta thấy biểu tượng mới cho *_DTrafficSignal* và *_DTrafficSignalEvents*. Đó là hai thành phần ghép nối COM. Biểu tượng màu xanh cho biết *AboutBox* là phương pháp ghép nối. Toàn bộ đặc tả ghép nối là ở trong tệp *TrafficSignal.odl*. *ControlWizard* đã phát sinh ba lớp: lớp ứng dụng *CTrafficSignalApp*, lớp điều khiển *CTrafficSignalCtrl* và lớp trang thuộc tính *CTrafficSignalPropPage*.

Lớp ứng dụng

Lớp ứng dụng *CTrafficSignalApp* rất đơn giản, chúng chỉ chứa hai thành phần: hàm *InitInstance* gộp mã trình khởi động của lập trình viên, hàm *ExitInstance* chứa các công việc dọn dẹp khi kết thúc điều khiển. Ở đây đã dấu đi các phức tạp bên trong chương trình. Lớp cơ sở của lớp ứng dụng này là lớp *COleControlModule* suy diễn từ *CWinApp*. Chúng cung cấp toàn bộ các chức năng của ứng dụng Windows thông thường. Bản mặc định của hàm *InitInstance* gọi bản trong *COleControlModule* để khởi động điều khiển.

Lớp điều khiển

Lớp điều khiển *CTrafficSignalCtrl* suy diễn từ lớp *COleControl* của MFC và cung cấp ghép nối cho trình chứa điều khiển. Ta có thể xen thêm vào lớp này các hàm thành phần hay dữ liệu riêng cho ứng dụng để có được điều khiển như mong muốn.

Khi gửi thông điệp *WM_PAINT* đến điều khiển, hàm *OnDraw* được gọi, vậy ta sẽ xen thêm các thao tác vẽ cho điều khiển. Hàm thành phần *DoPropExchange* quản lý vào/ra các thuộc tính của điều khiển. Khi xen thuộc tính mới cho điều khiển ta phải tự xen mã trình để thực hiện vào/ra. Hàm thành phần *OnResetState* được khung ứng dụng gọi khi cần đặt lại giá trị mặc định cho các thuộc tính điều khiển. Cài đặt mặc định của chúng gọi *DoPropExchange* để làm việc này. Nếu điều khiển

cần khởi động đặc biệt thì phải xen chúng vào hàm `OnResetState`. *ControlWizard* còn phát sinh bốn *macro*

```
DECLARE_OLECREATE_EX,
DECLARE_OLETYPELIB,
DECLARE_PROPPAGEIDS,
DECLARE_OLECTLTYPE
```

để thiết lập cơ chế mà các thao tác của điều khiển ActiveX yêu cầu.

Lớp điều khiển bao gồm mã trình trợ giúp đặc trưng của ghép nối với trình chứa. *ControlWizard* đã phát sinh trong lớp này ba nhóm mã trình liên quan đến các định nghĩa ánh xạ thông điệp (*message map*), ánh xạ phân phối (*dispatch map*) và ánh xạ sự kiện (*event map*). Ta đã quen biết với ánh xạ thông điệp trong các trình ứng dụng Windows thông thường. Ánh xạ phân phối xác định kết nối giữa tên trong và tên ngoài của thuộc tính và phương pháp để trình chứa tham nhập. Ánh xạ sự kiện gồm đặc tả của hàm lớp để đốt cháy sự kiện khi điều khiển gửi chúng đến trình chứa.

Cài đặt mặc định của lớp điều khiển được đặt trong tệp `TrafficSignalCtrl.cpp`. Trừ danh sách trang thuộc tính, toàn bộ chúng được *ClassWizard* quản lý, cho nên ta có thể bỏ qua nội dung chi tiết của chúng. Quan sát tệp này ta thấy, sau cài đặt cho bốn ánh xạ trên là khối chứa danh sách các trang thuộc tính của điều khiển. Thí dụ này chỉ có một trang cho nên ta chỉ có một *macro* `PROPPAGEID` trong chương trình. *Macro* tiếp theo trong mã trình là:

```
IMPLEMENT_OLECREATE_EX(CTrafficSignalCtrl, "TRAFFICSIGNAL.TrafficSignalCtrl.1",
    0xa833b92a, 0x78ff, 0x11d0, 0x92, 0x57, 0, 0x20, 0x18, 0x34, 0xe2, 0xa3)
```

Mục đích của *macro* này là lập ra "xưởng sản xuất lớp" (*class factory*) cho điều khiển. *Class factory* là đối tượng có khả năng tạo ra các đối tượng COM. Các đối tượng mà nó có thể phát sinh là các bản của điều khiển. Các bản của điều khiển được nhận dạng bằng `CLSID`, được chỉ ra trong 11 tham số cuối của *macro* trên đây. Đây là số duy nhất được *ControlWizard* phát sinh tự động. Đối tượng *class factory* cài đặt ghép nối COM chuẩn với tên ghép nối `IClassFactory`. Tuy nhiên ta không phải quan tâm chi tiết hơn về cơ chế này vì khung ứng dụng quản lý giúp ta. *Macro* tiếp theo trong mã trình là `IMPLEMENT_OLETYPELIB()`. Chúng tạo ra hàm thành phần của lớp điều khiển dùng để khai thác thông tin về ghép nối với trình chứa. Ta cũng không cần quan sát sâu hơn vì khung ứng dụng quản lý giúp ta. Tiếp theo là định nghĩa hai hằng tổng thể. Chúng là cấu trúc chứa chỉ danh duy nhất cho ghép nối với trình chứa. Hàm `UpdateRegistry` nạp chồng cài đặt lớp cơ sở. Mục đích của hàm này là yêu cầu điều khiển đăng ký vào bộ đăng ký hệ thống. Nếu chưa đăng ký thì điều khiển chưa được sử dụng. Hàm này là thành phần của lớp `CTrafficSignalCtrlFactory` chứ không phải của `COleControl`. Việc phát sinh định nghĩa cho `CTrafficSignalCtrlFactory` được thực hiện trong chương trình dịch cho nên ta không thấy được mã trình cho chúng.

Phần cài đặt còn lại của `CTrafficSignalCtrl` là cài đặt mặc định đơn giản của các thành phần lớp.

Lớp trang thuộc tính

Lớp `CTrafficSignalPropPage` cài đặt khả năng thiết đặt thuộc tính điều khiển thông qua trang thuộc tính. Mỗi bản của lớp này quản lý một trang thuộc tính. *ControlWizard* đã phát sinh chúng cho ta. Nhiệm vụ chính của lớp này là chuyển tải dữ liệu từ trang thuộc tính vào các biến của cài đặt điều khiển ActiveX. Dữ liệu được nhập thông qua các điều khiển như phím, hộp danh sách trên trang thuộc tính. Hàm `DoDataExchange` quản lý việc trao đổi dữ liệu này.

Khi quan sát lớp trang thuộc tính trong tệp `TrafficSignalPpg.cpp` ta thấy chúng chứa mã trình định nghĩa CLSID và cài đặt hàm thành phần `UpdateRegistry`. Cần phải có chúng vì mỗi trang thuộc tính là một đối tượng COM. Chúng có class factory riêng và đầu vào riêng trong bộ đăng ký hệ thống. Hiện tại cấu tử không chứa mã trình. *ClassWizard* sẽ xen thêm mã trình khi khởi động thuộc tính của điều khiển. Tương tự, hàm `DoDataExchange` sẽ được mở rộng bằng việc thêm các biến lưu trữ giá trị thuộc tính.

11.2.2 Định nghĩa đối tượng tín hiệu giao thông

Ta có thể biểu diễn tín hiệu giao thông trong lớp. Hãy xen lớp với tên `CTrafficSignal` vào chương trình. Hãy đổi tên hai tệp do Visual C++ tự động phát sinh thành `OurTrafficSignal.h` và `OurTrafficSignal.cpp`.

Đối tượng tín hiệu giao thông sẽ biểu diễn tín hiệu theo trạng thái riêng. Việc thay đổi trạng thái được tác động từ ngoài. Nhưng chương trình phải có khả năng khởi động trạng thái tín hiệu và chuyển tín hiệu từ trạng thái này sang trạng thái khác. Tín hiệu có ba trạng thái: dừng, đi, chuẩn bị dừng.

Ta sẽ cài đặt khả năng tín hiệu tự vẽ. Với giả sử về hoạt động của tín hiệu giao thông như trên, ta có thể định nghĩa lớp tín hiệu như sau:

```
class CTrafficSignal
{
public:
    CTrafficSignal();
    virtual ~CTrafficSignal();
    // Ghép nối của lớp
    void SetPosition(CPoint ptPosition)
        {m_ptPosition = ptPosition;}
    void SetHeight(int nHeight)
        {m_nHeight = nHeight;}
    void Draw(CDC* pDC);           // Vẽ tín hiệu giao thông
    void SetSignalState(int nState)
```

```

    {m_nSignalState = nState;}
    int NextState();           // Chuyển sang trạng thái tiếp
private:
    CPoint m_ptPosition;      // Đáy tín hiệu
    int m_nHeight;           // Độ cao tín hiệu
    int m_nSignalState;       // Trạng thái tín hiệu
};

```

Lớp trên có hàm thành phần Draw chưa được định nghĩa. Hàm này vẽ tín hiệu từ các giá trị m_ptPosition và m_nHeight. Hàm chưa được định nghĩa khác là NextState, làm thay đổi tín hiệu đến trạng thái tiếp theo bằng cách gán giá trị tương ứng cho m_nSignalState.

Cài đặt hàm NextState

Hàm sẽ định nghĩa ba trạng thái của tín hiệu bằng biến *const* trong tệp mới có tên OurConstants.h như sau:

```

// Định nghĩa hằng
#ifndef __OURCONSTANTS_H__
#define __OURCONSTANTS_H__
const int STOP          = 101;
const int GO            = 103;
const int READY_TO_STOP = 104;
#endif

```

Tiếp theo ta xen dòng lệnh #include "OurConstants.h" vào tệp OurTrafficSignal.cpp vì hàm NextState sẽ sử dụng các hằng này. Hàm NextState được cài đặt trong tệp OurTrafficSignal.cpp như sau:

```

// Đổi tín hiệu đến trạng thái tiếp theo theo trình tự
int CTrafficSignal::NextState()
{
    switch (m_nSignalState)
    {
        case STOP:
            m_nSignalState = GO;
            break;
        case GO:
            m_nSignalState = READY_TO_STOP;
            break;
        case READY_TO_STOP:
            m_nSignalState = STOP;
            break;
        default:

```

```

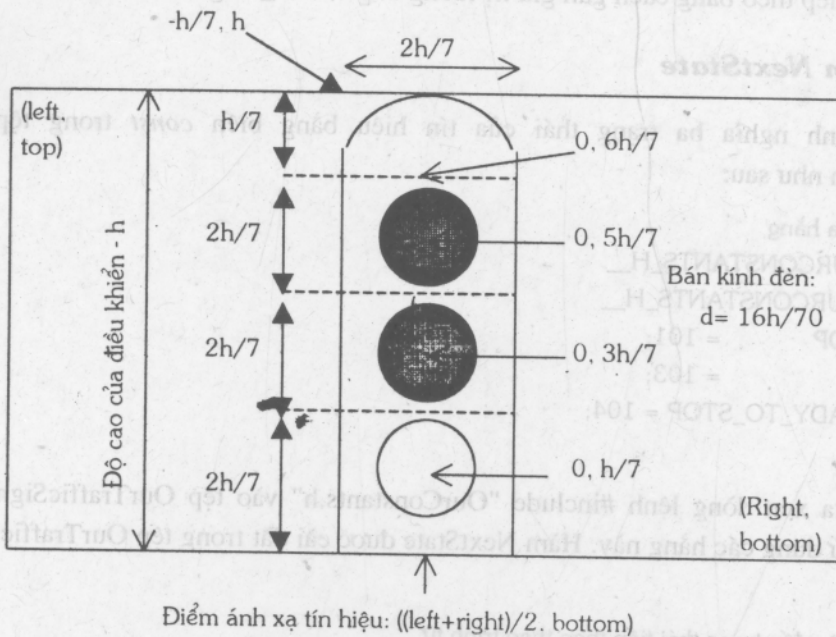
m_nSignalState = STOP;
AfxMessageBox((LPCTSTR)"Invalid signal state");
}

return m_nSignalState;
}

```

Cài đặt hàm Draw

Để vẽ tín hiệu ta phải biết tính độ rộng từ độ cao, và vị trí của đèn từ điểm ánh xạ `m_ptPosition`. Kích thước được xác định như trên hình 11.1.



Hình 11.1 Đèn tín hiệu giao thông

Độ cao của tín hiệu bằng độ cao của điều khiển. Gốc tọa độ của các kích thước đèn được đặt tại điểm ánh xạ tín hiệu. Hàm vẽ tín hiệu lấy giá trị trong thành phần dữ liệu `m_ptPosition` làm gốc tọa độ. Đoạn mã trình của hàm vẽ được xen vào tệp có thể như sau:

```

// Vẽ đèn tín hiệu
void CTrafficSignal::Draw(CDC* pDC)
{
    // Chọn bút và màu tô cho đèn
    CBrush* pOldBrush = (CBrush*)pDC->SelectStockObject(GRAY_BRUSH);
    CPen* pOldPen = (CPen*)pDC->SelectStockObject(BLACK_PEN);
    // Đặt giá trị vẽ khung đèn
}

```

```

int nLeft = m_ptPosition.x - m_nHeight/7;
int nTop = m_ptPosition.y - (long)m_nHeight*6L/7L;
int nRight = m_ptPosition.x + m_nHeight/7;
int nBottom = m_ptPosition.y;
pDC->Rectangle(nLeft, nTop, nRight, nBottom); // Vẽ khung
// Tính tọa độ để vẽ nửa hình tròn
CRect rect(nLeft, nTop - m_nHeight/7, nRight, nTop + m_nHeight/7);
CPoint ptStart(nRight, nTop);
CPoint ptEnd(nLeft, nTop);
pDC->Chord(rect, ptStart, ptEnd);
//nơi sẽ vẽ đèn tín hiệu
pDC->SelectObject(pOldBrush);
pDC->SelectObject(pOldPen);
}

```

Đoạn mã trình trên đã chọn màu vẽ, màu tô để vẽ khung đèn nhờ hàm `SelectStockObject`. Tiếp theo là tính tọa độ của chữ nhật bao quanh đèn. Chế độ ánh xạ hiện hành là `MM_TEXT`. Khung đèn được vẽ nhờ hàm `Rectangle`. Hàm `Chord` được sử dụng để vẽ nửa hình tròn trên đỉnh đèn.

Để vẽ ánh sáng ta định nghĩa màu sẽ sử dụng trong tệp `OurConstants.h`.

```

const COLORREF RED    = RGB(255, 0, 0);
const COLORREF ORANGE = RGB(225, 150, 0);
const COLORREF GREEN  = RGB(0, 255, 0);
const COLORREF GRAY   = RGB(100, 100, 100);

```

Màu đỏ, vàng, xanh dành cho đèn sáng còn màu xám cho đèn tắt. Để vẽ màu cho đèn ta phải tạo màu tô phụ thuộc theo trạng thái đèn lưu trong `m_nSignalState`. Các dòng lệnh sau đây được xen thêm vào hàm `Draw` để thực hiện công việc này.

```

// Tạo màu tô cho ánh sáng
CBrush brStop;           // Màu tô cho dừng
CBrush brReady;          // Màu tô cho đèn chuẩn bị đi
CBrush brGo;             // Màu tô cho đèn đi
switch (m_nSignalState)
{
case STOP:                // Màu đỏ
    brStop.CreateSolidBrush(RED);
    brReady.CreateSolidBrush(GRAY);
    brGo.CreateSolidBrush(GRAY);
    break;
case GO:                  // Màu xanh
    brStop.CreateSolidBrush(GRAY);
    brReady.CreateSolidBrush(GRAY);

```

```

        brGo.CreateSolidBrush(GREEN);
        break;
    case READY_TO_STOP:           // màu vàng
        brStop.CreateSolidBrush(GRAY);
        brReady.CreateSolidBrush(ORANGE);
        brGo.CreateSolidBrush(GRAY);
        break;
    default:
        brStop.CreateSolidBrush(GRAY);
        brReady.CreateSolidBrush(GRAY);
        brGo.CreateSolidBrush(GRAY);
}

```

Giá trị của `m_nSignalState` quyết định màu đèn. Khi đã có màu tô đèn, ta có thể vẽ chúng. Đoạn mã trình sau được xen vào hàm `Draw` trong tệp `OurTrafficSignal.CPP`.

```

.....
// Xác định chữ nhật bao quanh một đèn
int nMargin = (long)m_nHeight * 2L/70L; // 10% độ rộng
nLeft += nMargin;
nTop += nMargin;
nRight -= nMargin;
int nStep = (long)m_nHeight * 2L/7L; // Khoảng cách giữa hai đèn
nBottom = nTop + nStep - 2 * nMargin;
// Vẽ đèn dừng
pDC->SelectObject(&brStop);
pDC->Ellipse(nLeft, nTop, nRight, nBottom);
// Vị trí đèn sẵn sàng
nTop += nStep;
nBottom += nStep;
// Vẽ đèn sẵn sàng
pDC->SelectObject(&brReady);
pDC->Ellipse(nLeft, nTop, nRight, nBottom);
// Vị trí đèn đi
nTop += nStep;
nBottom += nStep;
// Vẽ đèn đi
pDC->SelectObject(&brGo);
pDC->Ellipse(nLeft, nTop, nRight, nBottom);
.....

```

Cài đặt cấu tử

Ta phải xen mã trình cài đặt cấu tử cho tệp `OurTrafficSignal.CPP`. Đoạn mã trình đó như sau:

```
CTrafficSignal::CTrafficSignal()
{
    m_ptPosition = CPoint(0, 0); // Vị trí tùy chọn
    m_nHeight = 1000;           // Độ cao tùy chọn
    m_nSignalState = STOP;      // Trạng thái ban đầu là STOP
}
```

11.2.3 Sử dụng đối tượng CTrafficSignal

Để xen đối tượng đèn tín hiệu giao thông vào điều khiển, ta phải xen thành phần *protected* vào lớp CTrafficSignalCtrl. Trong tệp CTrafficSignalCtrl.h sẽ có đoạn mã trình sau:

```
protected:
    CTrafficSignal* m_pSignal; // Con trỏ đến đối tượng đèn tín hiệu giao thông
```

Xen thêm dòng khai báo lớp CTrafficSignal như sau vào trước phần định nghĩa lớp CTrafficSignalCtrl trong tệp CTrafficSignalCtrl.H:

```
class CTrafficSignal;
```

Trong cấu tử của lớp CTrafficSignalCtrl phải tạo đối tượng của lớp CTrafficSignal. Cấu tử sẽ như sau:

```
CTrafficSignalCtrl::CTrafficSignalCtrl()
{
    InitializeIDs(&IID_DTrafficSignal, &IID_DTrafficSignalEvents);
    m_pSignal = new CTrafficSignal;
}
```

Vì ta tạo đối tượng trong cấu tử cho nên phải phá hủy chúng trong hủy tử:

```
CTrafficSignalCtrl::~CTrafficSignalCtrl()
{
    delete m_pSignal;
}
```

Mặc định, hàm Draw trong điều khiển ActiveX sẽ vẽ elíp. Ta phải xoá đoạn mã trình này để thay thế bằng đoạn mã trình vẽ tín hiệu giao thông như sau:

```
void CTrafficSignalCtrl::OnDraw(
    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    pdc->FillRect(rcBounds,
        CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH))); // Tô nền
    // Chọn độ cao đèn tín hiệu
    m_pSignal->SetHeight(abs(rcBounds.Height()));
    // Điểm tham chiếu
```

```

CPoint ptPosition(((long)rcBounds.right + rcBounds.left)/2L, rcBounds.bottom);
m_pSignal->SetPosition(ptPosition);
m_pSignal->Draw(pdc);          // Vẽ đèn tín hiệu
)

```

Tham số rcBounds của hàm OnDraw xác định chữ nhật mà điều khiển chiếm. Từ đó chúng ta tính điểm tham chiếu của đối tượng đèn tín hiệu.

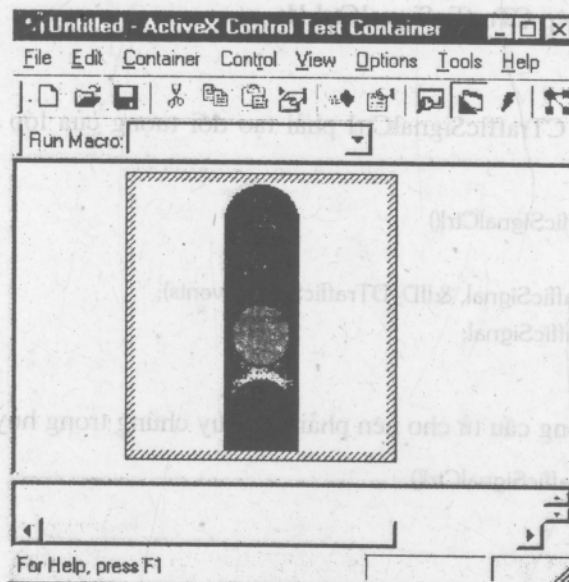
Cuối cùng ta phải xen các lệnh sau vào tệp C:\TrafficSignalCtrl.CPP:

```

#include "OurTrafficSignal.h"
#include "OurConstants.h"

```

Sau khi dịch chương trình ta có tệp TrafficSignal.OCX. Hãy sử dụng công cụ *ActiveX Control Test Container* của *Developer Studio* để kiểm tra chúng. Màn hình kết quả như trên hình 11.2.



Hình 11.2 Điều khiển ActiveX TrafficSignal

11.2.4 Sử dụng các đặc tính có sẵn và bao quanh

Trong thí dụ này ta thực hiện cài đặt đặc tính có sẵn (*stock property*) BackColor vào điều khiển ActiveX. Hãy sử dụng *ClassWizard* để xen các đặc tính có sẵn vào điều khiển. Chọn thực đơn View | ClassWizard của *Developer Studio*, chọn bảng Automation. Chọn CTrafficSignalCtrl cho hộp danh sách Class name:. Nhấn phím Add Property. Chọn BackColor cho hộp danh sách External name. Phím radio Stock trong nhóm Implementation được chọn tự động. Nhấn phím OK để trở về hộp

thoại *ClassWizard*. Hộp danh sách *External name* của hộp thoại này có chứa tên **BackColor**. Tiền tố S cho biết đây là đặc tính có sẵn (*stock*). Đặc tính lập trình viên tạo ra (*custom*) sẽ có tiền tố C. Nếu quan sát cài đặt lớp **CTrafficSignalCtrl** ta thấy ánh xạ phân phối (*dispatch*) đã được cập nhật.

```
BEGIN_DISPATCH_MAP(CTrafficSignalCtrl, COleControl)
    //({AFX_DISPATCH_MAP(CTrafficSignalCtrl)
    DISP_STOCKPROP_BACKCOLOR()
    //})AFX_DISPATCH_MAP
    DISP_FUNCTION_ID(CTrafficSignalCtrl, "AboutBox", DISPID_ABOUTBOX, AboutBox, VT_EMPTY,
    VTS_NONE)
END_DISPATCH_MAP()
```

Dòng in đậm trong mã trình đảm bảo rằng từ ngoài điều khiển có thể sử dụng đặc tính **BackColor**. Hãy xen thêm đoạn mã trình sau vào hàm **OnDraw** để điều khiển ActiveX sử dụng giá trị của đặc tính **BackColor** làm màu nền.

```
// Sử dụng đặc tính BackColor của điều khiển để đặt màu nền
CBrush brBack(TranslateColor(GetBackColor()));
// Sử dụng đặc tính BackColor của trình chứa để đặt màu nền
// CBrush brBack(TranslateColor(AmbientBackColor()));
pdc->FillRect(rcBounds, &brBack); // Tô nền
```

Hàm **GetBackColor** thừa hưởng từ **COleControl**. Chúng cho lại giá trị màu kiểu **OLE_COLOR** lưu trong đặc tính có sẵn của điều khiển. Kiểu **OLE_COLOR** là định nghĩa chuẩn cho giá trị màu khi trao đổi giữa các đối tượng COM. Hàm **TranslateColor** chuyển đổi giá trị **OLE_COLOR** thành **COLORREF** (giá trị RGB). Một vài hàm khác định nghĩa trong **COleControl** hay được sử dụng là **GetForeColor** và **GetScaleUnits**.

Đoạn mã trình trên đây đã sử dụng đặc tính có sẵn của trình chứa để khởi động đặc tính **BackColor** của điều khiển. Vậy, lần đầu tiên khi xen điều khiển vào trình chứa, màu nền của điều khiển và màu nền của trình chứa là như nhau. Nhưng khi thay đổi màu nền trình chứa thì đặc tính có sẵn của chúng vẫn không được cập nhật. Vậy phải sử dụng hàm thành phần **AmbientBackColor** thừa hưởng từ **COleControl** để xác định màu nền hiện hành, sau đó mới khởi động đặc tính **BackColor** của điều khiển để có màu nền điều khiển và màu nền trình chứa là như nhau. Tuy nhiên, khi thay đổi màu nền bao quanh của trình chứa thì phải yêu cầu điều khiển vẽ lại để có được màu mới.

11.2.5 Cài đặt thuộc tính mới cho điều khiển (*custom property*)

Ta có thể định nghĩa bốn loại thuộc tính custom cho điều khiển ActiveX. Chúng phản ánh cách thao tác khác nhau của thuộc tính. Loại đơn giản nhất của thuộc tính custom là **DISP_PROPERTY**. Thành phần dữ liệu của lớp điều khiển sẽ biểu diễn chúng. Thuộc tính **DISP_PROPERTY** cho phép

thâm nhập tự do.

- Kiểu thuộc tính `DISP_PROPERTY_NOTIFY` được biểu diễn bởi thành phần dữ liệu của lớp điều khiển và hàm của chúng trong lớp được gọi khi thâm nhập giá trị đặc tính. Điều này cho phép điều khiển thích nghi tức thì thao tác của chúng với giá trị mới cho điều khiển.
- Kiểu `DISP_PROPERTY_EX` có nhiều hàm của trình chứa thâm nhập trực tiếp dữ liệu của chúng. Các hàm này được gọi là `Get/Set`.
- Kiểu `DISP_PROPERTY_PARAM` tương tự kiểu `DISP_PROPERTY_EX`. Chúng có các hàm `Get/Set` để xử lý, đồng thời chúng có thể kích hoạt nhiều tham số lưu trong một trường.

Thí dụ sau đây cài đặt thuộc tính *custom*. Thuộc tính thứ nhất cho phép xen thời gian khác nhau để điều khiển xe dừng và xe đi trong hệ thống đèn tín hiệu giao thông. Thuộc tính thứ hai cho khả năng khởi động từ đèn đỏ hay đèn xanh.

Hãy sử dụng *ClassWizard* để cài đặt các thuộc tính *custom*. Trước hết ta xen thuộc tính định nghĩa đèn nào sáng khi hệ thống bắt đầu chạy. Ta sử dụng biến logic, nếu chúng có giá trị `TRUE` thì khởi động từ đèn đỏ. Sau khi khởi động *ClassWizard*, hãy chọn bảng *Automation*. Chọn lớp `CTrafficSignalCtrl` cho hộp danh sách *Class name* rồi nhấn phím *Add Property...* để hiển thị hộp thoại. Nhập `StartRed` cho tên ngoài (*External name*), `m_bStartRed` cho tên biến (*Variable name*), chọn `BOOL` cho kiểu (*Type*). *ClassWizard* sẽ phát sinh biến và hàm thông báo với tên `OnStartRedChanged`. Kiểu thuộc tính lần này là `DISP_PROPERTY_NOTIFY`. Phím *radio Member variable* được chọn mặc định. Nhấn phím *OK* để trở về bảng *Automation*. Công việc được lặp lại để xen thuộc tính *custom* xác định thời gian cho dừng hay đi của đèn tín hiệu. Ta cài đặt thuộc tính kiểu `DISP_PROPERTY_EX` bằng cách chọn phím *radio Get/Set methods*. Hãy nhập tên ngoài là `StopOrGoTime`, chọn *long* cho kiểu. Chọn *OK* để về bảng *Automation*. Như vậy ta đã thêm hai thuộc tính *custom* vào thuộc tính có sẵn, hai hàm vào lớp `CTrafficSignalCtrl`. Nhấn phím *OK* để kết thúc hộp thoại *ClassWizard*.

Nếu quan sát ánh xạ phân phối của lớp `CTrafficSignalCtrl`, ta thấy mã trình của chúng như sau:

```
BEGIN_DISPATCH_MAP(CTrafficSignalCtrl, COleControl)
    //{{AFX_DISPATCH_MAP(CTrafficSignalCtrl)
    DISP_PROPERTY_NOTIFY(CTrafficSignalCtrl, "StartRed", m_bStartRed,
        OnStartRedChanged, VT_BOOL)
    DISP_PROPERTY_EX(CTrafficSignalCtrl, "StopOrGoTime", GetStopOrGoTime,
        SetStopOrGoTime, VT_I4)
    DISP_STOCKPROP_BACKCOLOR()
    //}}AFX_DISPATCH_MAP
    DISP_FUNCTION_ID(CTrafficSignalCtrl, "AboutBox", DISPID_ABOUTBOX,
```

```
AboutBox, VT_EMPTY, VTS_NONE)
END_DISPATCH_MAP()
```

Khởi động đặc tính của người sử dụng (custom)

Ta phải nạp giá trị ban đầu cho các thuộc tính *custom*. Vì chưa định nghĩa biến cho thuộc tính `StopOrGoTime`, ta sẽ xen trực tiếp thành phần dữ liệu vào vùng *protected* của lớp `CTrafficSignalCtrl`.

```
long m_lStopOrGoTime; // Thời gian dừng hay đi
```

Hãy mở rộng mã trình của hàm `DoPropExchange` của lớp điều khiển để khởi động thuộc tính này và `m_bStartRed`.

```
void CTrafficSignalCtrl::DoPropExchange(CPropExchange* pPX)
{
    ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
    COleControl::DoPropExchange(pPX);
    PX_Bool(pPX, _T("StartRed"), m_bStartRed, TRUE);
    PX_Long(pPX, _T("StopOrGoTime"), m_lStopOrGoTime, 5000);
    // Đặt trạng thái tín hiệu theo thuộc tính StartRed
    if(m_bStartRed) m_pSignal->SetSignalState(STOP);
    else m_pSignal->SetSignalState(GO);
}
```

Đoạn mã trình trên đây sử dụng hàm tổng thể `PX_`. Tham số của chúng có sử dụng *macro* `_T` với tên ngoài của thuộc tính.

11.2.6 Làm đèn tín hiệu hoạt động

Để đèn tín hiệu hoạt động được ta cần xen thêm các dòng lệnh sau vào vùng *protected* của lớp `CTrafficSignalCtrl`.

```
UINT m_nChangeTime; // Thời gian của đèn vàng
BOOL m_bSignalGo; // Giá trị TRUE là đang đèn xanh
UINT m_nTimerID; // Chỉ danh đồng hồ
```

Biến thứ nhất là khoảng thời gian của đèn vàng. Biến thứ hai có giá trị `TRUE` nếu đèn tín hiệu hoạt động, ngược lại chúng có giá trị `FALSE`. Biến thứ ba là chỉ danh đồng hồ để xác định thời gian chuyển từ trạng thái này sang trạng thái khác. Giá trị của chúng được khởi động trong cấu tử lớp như sau:

```
CTrafficSignalCtrl::CTrafficSignalCtrl()
{
    InitializeIDs(&IID_DTrafficSignal, &IID_DTrafficSignalEvents);
}
```

```

m_pSignal = new CTrafficSignal; // Tạo đèn tín hiệu
m_bSignalGo = FALSE;           // Khởi động trạng thái dừng cho đèn
m_nChangeTime = 1500U;         // Thay đổi đèn, đơn vị ms
m_nTimerID = 10;                // Chỉ danh đồng hồ
}

```

Khởi động và dừng đèn tín hiệu

Giả sử ta sử dụng phím chuột để dừng hay khởi động đèn tín hiệu. Nếu nhấn phím chuột trên điều khiển thì đèn đang dừng sẽ chạy hay ngược lại. Hãy sử dụng *ClassWizard* để xen thêm hàm quản lý thông điệp WM_LBUTTONDOWN vào lớp CTrafficSignalCtrl như sau:

```

void CTrafficSignalCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{
    // Nếu đèn dừng thì khởi động chúng và ngược lại
    m_bSignalGo = !m_bSignalGo;
    if (m_bSignalGo)    StartSignal();
    else                StopSignal();
    COleControl::OnLButtonDown(nFlags, point);
}

```

Sau đó ta xen thêm hàm thành phần StartSignal vào vùng *private* của lớp như sau:

```

void CTrafficSignalCtrl::StartSignal()
{
    // Đặt thời gian cho đồng hồ
    m_nTimerID = SetTimer(m_nTimerID, (UINT)m_lStopOrGoTime, NULL);
    if (!m_nTimerID) {
        AfxMessageBox((LPCTSTR) "No Timer!");
        exit(1);
    }
    InvalidateControl(); // Yêu cầu vẽ lại điều khiển
}

```

Hàm thành phần SetTimer thừa hưởng từ lớp CWnd. Tham số thứ hai của hàm sẽ lưu khoảng thời gian tính bằng mili giây (ms). Vì số lượng đồng hồ là có hạn cho nên khi hàm này cho lại giá trị FALSE thì phải kết thúc chương trình. Nếu khởi động được đồng hồ thì đoạn trình trên yêu cầu vẽ lại để đèn tín hiệu khởi động từ đặc tính StartRed.

Cài đặt hàm StopSignal để dừng đèn tín hiệu như sau:

```

void CTrafficSignalCtrl::StopSignal()
{
    KillTimer(m_nTimerID); // Phá hủy đồng hồ
    InvalidateControl();    // Yêu cầu vẽ lại điều khiển
}

```

Sau khi phá hủy đồng hồ, đoạn mã trình yêu cầu vẽ lại điều khiển. Sử dụng *ClassWizard* để cài đặt hàm quản lý thông điệp WM_TIMER. Hãy xen các dòng lệnh sau vào hàm:

```
void CTrafficSignalCtrl::OnTimer(UINT nIDEvent)
{
    UINT nInterval = 0; // Tính bằng ms
    // Chuyển sang trạng thái mới, đặt thời gian cho đồng hồ trên cơ sở
    // trạng thái mới
    switch (m_pSignal->NextState())
    {
        case STOP:
        case GO:
            nInterval = (UINT)m_lStopOrGoTime; // Thời gian dừng hoặc đi
            break;
        default:
            nInterval = m_nChangeTime; // Thời gian chuyển tiếp
    }
    InvalidateControl();
    // Đảm bảo đã phá hủy đồng hồ cũ
    KillTimer(m_nTimerID);
    // Đặt sự kiện đồng hồ mới
    m_nTimerID = SetTimer(m_nTimerID, nInterval, NULL);
    if (!m_nTimerID)
    {
        AfxMessageBox((LPCTSTR)"No Timer!");
        exit(1);
    }
}
```

Tín hiệu chuyển sang trạng thái tiếp theo khi trình gọi hàm thành phần NextState của đối tượng CTrafficSignalCtrl. Trạng thái mới sẽ xác định khoảng thời gian để lưu vào nInterval. Gọi hàm InvalidateControl để yêu cầu đèn tín hiệu vẽ lại theo trạng thái mới và khởi động chu kỳ mới.

Cài đặt hàm thông báo cho điều khiển

Hàm thông báo OnStartRedChanged được gọi khi đặc tính StartRed bị sửa đổi từ ngoài. Cài đặt của hàm này như sau:

```
void CTrafficSignalCtrl::OnStartRedChanged()
{
    // Dừng đèn tín hiệu khi cần
    if (m_bSignalGo)
    {
        m_bSignalGo = FALSE; // Dừng đèn tín hiệu
    }
}
```

```

    StopSignal();
}
// Đặt đối tượng tín hiệu vào trạng thái phù hợp
if (m_bStartRed)
    m_pSignal->SetSignalState(STOP);
else
    m_pSignal->SetSignalState(GO);
InvalidateControl();
SetModifiedFlag();
}

```

Đoạn trình trên sử dụng hàm thành phần `SetSignalState` của đối tượng `CTrafficSignal` để đặt lại trạng thái tín hiệu.

Cài đặt hàm đặc tính *Get/Set*

Hàm `Get` cho biết đặc tính `StopOrGoTime`. Chúng cho lại khoảng thời gian hiện hành.

```

long CTrafficSignalCtrl::GetStopOrGoTime()
{
    return m_lStopOrGoTime;
}

```

Hàm `Set` đòi hỏi mã trình phức tạp hơn.

```

void CTrafficSignalCtrl::SetStopOrGoTime(long nNewValue)
{
    // Only alter the control if the value is different
    if (m_lStopOrGoTime != nNewValue) {
        m_lStopOrGoTime = nNewValue; // Đặt lại thời gian cho dừng hay đi
        OnStartRedChanged();         // Đặt trạng thái ban đầu
        SetModifiedFlag();
    }
}

```

Giá trị mà đối hàm sẽ nhận là giá trị mới của đặc tính. Nếu chúng là mới thì được lưu vào thành phần dữ liệu `m_lStopOrGoTime`.

11.2.7 Cài đặt sự kiện cho điều khiển ActiveX

Nhớ lại rằng sự kiện được sử dụng để thông báo cho trình chứa biết cái gì đã xảy ra trong điều khiển ActiveX. Trong thí dụ này, trình chứa có thể biết khi nào thay đổi trạng thái đèn tín hiệu và trạng thái tiếp theo của chúng là gì. Ta sử dụng *ClassWizard* để cài đặt các sự kiện vào điều khiển. Chọn bảng `ActiveX Events`. Chọn `CTrafficSignalCtrl` cho tên lớp, nhấn phím `Add Event...` Nhập xâu

SignalChanged cho tên ngoài. Xen tham số *INewState* có kiểu *long* vào *Parameter list*. *ClassWizard* tự động phát sinh tên cho tên nội bộ (*Internal name*). Đó là tên hàm sẽ được gọi khi ta muốn đốt cháy sự kiện. Nhấn phím OK để phát sinh sự kiện. Bảng ActiveX Events sẽ chỉ ra các sự kiện mới. Chúng sẽ được nhập vào ánh xạ sự kiện của lớp *CTrafficSignalCtrl* và định nghĩa hàm *FireSignalChanged*.

Vị trí tốt nhất để đặt đốt cháy sự kiện này là trình quản lý thông điệp *WM_TIMER* vì tại đây ta làm thay đổi trạng thái đèn tín hiệu. Đoạn mã trình mở rộng trong hàm *OnTimer* như sau:

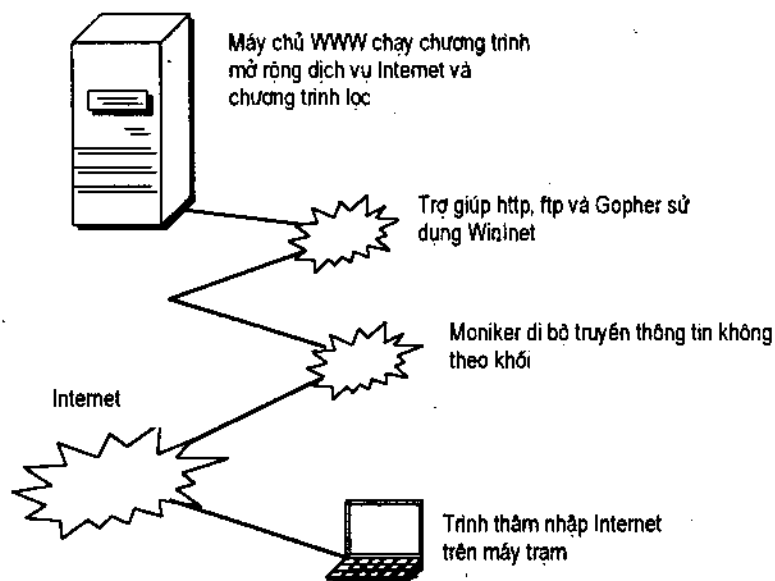
```
void CTrafficSignalCtrl::OnTimer(UINT nIDEvent)
{
    UINT nInterval = 0;
    int nNewState = m_pSignal->NextState();
    switch (nNewState)
    {
        case STOP: case GO:
            nInterval = (UINT)m_lStopOrGoTime;
            break;
        default:
            nInterval = m_nChangeTime;
    }
    FireSignalChanged(nNewState);
    InvalidateControl();
    KillTimer(m_nTimerID);
    m_nTimerID = SetTimer(m_nTimerID, nInterval, NULL);
    if (!m_nTimerID) {
        AfxMessageBox((LPCTSTR)"No Timer!");
        exit(1);
    }
}
```

Trên đây ta đã cài đặt điều khiển ActiveX đèn tín hiệu giao thông. Điều khiển này có thể được sử dụng cho các trình chứa được phát triển bằng *Visual C++* hay *Visual Basic*. Ta có thể làm tăng tính dễ sử dụng cho điều khiển bằng cách thay đổi một vài mã lệnh trong tệp *.odl*. Nhớ lại rằng tệp *.odl* được gán theo thư viện kiểu (*type library*) để trình chứa có thể sử dụng vào việc tìm kiếm các thông tin về điều khiển ActiveX. Thí dụ, ta có thể thay đổi tệp *.odl* sao cho lập trình viên có thể sử dụng tên hằng cho dữ liệu chuyển tới sự kiện *SignalChanged*.

CHƯƠNG 12

LẬP TRÌNH INTERNET

Internet càng ngày càng phát triển rộng rãi và đem lại cho người dùng những lợi ích to lớn. Những lợi ích này có được nhờ sự đóng góp rất lớn của các kho dữ liệu cũng như những chương trình duyệt và khai thác dữ liệu trên Internet. Càng ngày trên thị trường càng xuất hiện nhiều công nghệ, công cụ và ngôn ngữ lập trình giúp các nhà phát triển, các lập trình viên xây dựng các kho dữ liệu và các chương trình ứng dụng một cách nhanh chóng, hiệu quả trong các môi trường mạng.



Hình 12.1 Tương tác máy trạm và máy chủ trong Internet

Microsoft cung cấp rất nhiều giao diện lập trình ứng dụng (API) để xây dựng các ứng dụng trên Internet cho cả máy chủ và máy trạm. Các trình duyệt chạy trên máy trạm có thể tham nhập WWW (World Wide Web) và hiển thị các trang HTML (Hypertext Markup Language) chứa văn bản, hình ảnh, điều khiển ActiveX. Máy chủ cung cấp các dịch vụ FTP (File Transfer Protocol), HTTP (Hypertext Transfer Protocol), Gopher và chạy các chương trình mở rộng dịch vụ Internet sử dụng CGI (Common Gateway Interface) và giao diện lập trình ứng dụng dịch vụ Internet (Internet Server Application Programming Interface - ISAPI). Nhờ các chương trình ứng dụng trên máy trạm mà người sử dụng có thể khai thác thông tin và cung cấp các dữ liệu trên Internet.

MFC cung cấp nhiều lớp hỗ trợ xây dựng các chương trình trên Internet. Chúng ta có thể sử dụng lớp **COleControl**, **CDocObjectServer** và các lớp MFC liên quan để xây dựng các điều khiển ActiveX và tài liệu ActiveX. Chúng ta có thể sử dụng các lớp của *MFC WinInet* như **CInternetSession**, **CFTPConnection** và **CAsyncMonikerFile** để khai thác thông tin sử dụng các giao thức trên Internet như FTP, HTTP và Gopher. Chúng ta cũng có thể sử dụng lớp **CHttpServer**, **CHttpFilter** và các lớp liên quan trong ISAPI của MFC để xây dựng các *chương trình mở rộng dịch vụ Internet (Internet Server Extension DLL)* và *chương trình lọc (Filter)* trên máy chủ. Bảng 12.1 mô tả ngắn gọn một số loại công việc có thể làm được với các công nghệ trên.

Bảng 12.1. Một số loại công việc làm được với công nghệ Internet của Microsoft

<i>Nếu Có</i>	<i>Công việc muốn làm</i>	<i>Nên sử dụng</i>
Dịch vụ Web	Theo dõi việc đăng nhập và biết chi tiết thông tin về yêu cầu URL	Viết một chương trình lọc sử dụng ISAPI yêu cầu thông báo các sự kiện đăng nhập và sự sắp xếp trong câu lệnh URL
Dịch vụ Web	Viết một hệ thống nhập yêu cầu khách hàng	Viết một chương mở rộng dịch vụ internet sử dụng ISAPI để xử lý trên máy chủ một biểu mẫu HTML đã được người sử dụng điền đầy đủ thông tin từ máy trạm. Cần nhắc việc sử dụng bộ kết nối cơ sở dữ liệu (một phần của IIS), đối với việc xử lý dữ liệu đơn giản hay sử dụng hoàn toàn ISAPI cho việc thăm nhập cơ sở dữ liệu.
Trình duyệt Web	Cung cấp nội dung động	Tạo các điều khiển ActiveX và tài liệu ActiveX.
Các máy trạm với các trình duyệt khác nhau nối với máy chủ	Thay đổi các dữ liệu trong các trang HTML khi gửi từ máy chủ cho mỗi máy trạm, như vậy những người sử dụng có một trình duyệt mới hơn sẽ nhận được những điểm đặc trưng mới nhất còn những người sử dụng có một trình duyệt cũ vẫn có thể đọc được thông tin bình thường	Viết một chương trình lọc sử dụng ISAPI
Ứng dụng trợ giúp Internet trên máy trạm	Tăng cường khả năng truyền tệp	Sử dụng WinInet

Chương này sẽ tập trung vào việc xây dựng các chương trình mở rộng dịch vụ Web bên phía máy chủ và sử dụng các lớp của WinInet để khai thác thông tin trên Internet thông qua các giao thức FTP, HTTP và Gopher bên phía máy trạm.

12.1. MỞ RỘNG DỊCH VỤ WEB BẰNG ISAPI

Lập trình viên có thể sử dụng ISAPI để viết các ứng dụng cơ sở dữ liệu, thí dụ như một hệ thống nhận đơn đặt hàng hoặc một danh mục khách hàng. Máy chủ có thể sưu tập thông tin từ người sử dụng thông qua các biểu mẫu HTML và gửi lại thông tin cho người sử dụng thông qua các trang

HTML tối ưu khác. Các chương trình ứng dụng trên có thể được viết dưới dạng chương trình mở rộng dịch vụ Internet. Một chương trình mở rộng dịch vụ Internet sử dụng ISAPI, được hiểu như một *chương trình dịch vụ Internet (Internet Server Application - ISA)*, là một tệp DLL được cài đặt trên thư mục thực hiện dùng chung (thư mục *share*) của máy chủ. Để chạy chương trình này, người sử dụng gõ vào dòng lệnh URL từ trình duyệt trên máy trạm, thí dụ: `http://myserver/register.dll?`. Chương trình dịch vụ Internet yêu cầu sẽ được chạy trên máy chủ và gửi lại dữ liệu cho máy trạm dưới dạng trang HTML. Tuy nhiên chúng ta hoàn toàn có thể điều khiển được khuôn dạng dữ liệu gửi lại cho máy trạm, thí dụ có thể gửi lại dữ liệu theo khuôn dạng tệp ảnh nhĩ phân hay bất kỳ khuôn dạng nào khác.

Bên cạnh việc mở rộng dịch vụ Web bằng chương trình dịch vụ Internet, chúng ta cũng có thể mở rộng dịch vụ Web bằng chương trình lọc sử dụng ISAPI.

Đặc trưng tiện lợi của WWW là để chạy một ứng dụng trên máy trạm ta không cần phải cài đặt mã nguồn lên đó. Điều này giúp việc nâng cấp ứng dụng rất dễ dàng. Để nâng cấp chương trình mở rộng dịch vụ Web, ta chỉ cần dừng dịch vụ Web, thay thế tệp chương trình cũ bằng tệp chương trình mới, sau đó chạy lại dịch vụ Web. Yêu cầu tiếp theo của máy trạm sẽ gọi chương trình dịch vụ Internet hay chương trình lọc mới nhất. Như vậy việc cài đặt chương trình nâng cấp chỉ phải thực hiện trên máy tính chủ.

Để tạo một chương trình lọc hay chương trình dịch vụ Internet bằng ISAPI, Visual C++ cung cấp công cụ *ISAPI Extension Wizard* giúp lập trình viên rất dễ dàng tạo lập chương trình. Sau khi *ISAPI Extension Wizard* tạo xong *project* cho chương trình, đối với chương trình dịch vụ Internet, lập trình viên chỉ cần viết thêm các hàm xử lý chức năng mới và tạo ánh xạ phân tích cú pháp cho các chức năng đó; đối với chương trình lọc, lập trình viên viết thêm các xử lý cho từng sự kiện.

12.1.1 Các lớp MFC của ISAPI

Nếu phần mềm dịch vụ Internet tuân theo chuẩn ISAPI thì chúng ta có thể tạo các chương trình dịch vụ Internet và các bộ lọc bằng MFC. MFC cung cấp cho lập trình viên năm lớp gói trong ISAPI để tạo và quản lý các chương trình mở rộng dịch vụ Internet và chương trình lọc, bao gồm:

- **CHttpServer** là đối tượng tạo lập và quản lý chương trình mở rộng dịch vụ Internet DLL (*Dynamic Linking Library*).
- **CHttpServerContext** là đối tượng tạo ra từ đối tượng CHttpServer để quản lý mỗi yêu cầu riêng biệt của máy tính trạm.
- **CHtmlStream** là một đối tượng tạo ra từ đối tượng CHttpServer để quản lý vùng đệm cho việc trả lời máy tính trạm. CHttpStream có thể chứa nội dung theo định dạng của tệp HTML để trả lời máy trạm.
- **CHttpFilter** là một đối tượng tạo ra để quản lý việc lọc đối với các thông tin đến và gửi

đi.

- **CHttpFilterContext** là một đối tượng tạo ra từ đối tượng **CHttpFilter** để quản lý lọc theo yêu cầu của máy trạm.

12.1.2. Chương trình mở rộng dịch vụ Internet

Chương trình mở rộng dịch vụ Internet được biết như một chương trình dịch vụ Internet, tuy nhiên chúng ở dạng chương trình của thư viện liên kết động DLL chứ không phải ở dạng chương trình thực hiện EXE, chúng được viết ra để mở rộng thêm các chức năng của dịch vụ Web. *Microsoft Developer Studio* cung cấp công cụ *Wizard* giúp lập trình viên dễ dàng tạo ra một chương trình dịch vụ Internet bằng các thao tác hỏi đáp và lập trình viên chỉ cần viết thêm các hàm chức năng của riêng mình. Cách tạo một chương trình dịch vụ Internet bằng công cụ *Wizard* được giới thiệu trong chương 13 "Hướng dẫn sử dụng *Developer Studio*".

Để dễ dàng hiểu được chương trình mở rộng dịch vụ Internet chúng ta xây dựng một thí dụ đơn giản.

12.1.2.1 Thí dụ Counter

COUNTER là một thí dụ về chương trình mở rộng dịch vụ Internet. Chương trình này cho chúng ta biết cách sử dụng ISAPI, mở rộng dịch vụ Internet gửi lại dữ liệu ảnh (thay vì gửi lại kết quả dưới dạng trang HTML) cho trình duyệt Web trên máy trạm như thế nào. Chương trình này đòi hỏi máy trạm có cài đặt dịch vụ Web trợ giúp ISAPI. Máy trạm yêu cầu COUNTER lấy giờ hệ thống trên máy chủ, hoặc báo cho máy trạm biết số lần thăm nhập vào một trang HTML nào đó của *Web site* trên máy chủ, kết quả được chuyển thành ảnh *xbitmap* và gửi về cho máy trạm. Việc gửi lại kết quả cho máy trạm dưới dạng ảnh *xbitmap* chỉ đơn thuần cho thấy chúng ta hoàn toàn có thể điều khiển được khuôn dạng dữ liệu kết quả gửi lại cho máy trạm.

Chương trình 12.1 COUNTER

Tệp *Charset.H*

```
// Chứa ảnh các ký tự dưới dạng bitmap. Các ký tự có kích thước 8x16
#define char_width 8
#define char_height 16
static unsigned char char_bits[11][char_height] =
{
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3C, 0x66, 0xC3, 0xC3,
      0xDB, 0xDB, 0xC3, 0xC3, 0x66, 0x3C }, // 0
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x1C, 0x1E, 0x18,
      0x18, 0x18, 0x18, 0x18, 0x18, 0x7E }, // 1
```

```
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3E, 0x63, 0x60, 0x30,
  0x18, 0x0C, 0x06, 0x03, 0x63, 0x7F }, // 2
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3E, 0x63, 0x60, 0x60,
  0x3C, 0x60, 0x60, 0x60, 0x63, 0x3E }, // 3
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x38, 0x3C, 0x36,
  0x33, 0x7F, 0x30, 0x30, 0x30, 0x78 }, // 4
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x03, 0x03, 0x03,
  0x3F, 0x60, 0x60, 0x60, 0x63, 0x3E }, // 5
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1C, 0x06, 0x03, 0x03,
  0x3F, 0x63, 0x63, 0x63, 0x63, 0x3E }, // 6
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x63, 0x60, 0x60,
  0x30, 0x18, 0x0C, 0x0C, 0x0C, 0x0C }, // 7
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3E, 0x63, 0x63, 0x63,
  0x3E, 0x63, 0x63, 0x63, 0x63, 0x3E }, // 8
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3E, 0x63, 0x63, 0x63,
  0x7E, 0x60, 0x60, 0x60, 0x30, 0x1E }, // 9
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18,
  0x00, 0x00, 0x00, 0x18, 0x18, 0x00 }, // :
};
//-----
```

Tệp charset.H2

```
// Các ký tự đặt ở chế độ định dạng xbitmap (XBM)
// Các chữ số và dấu hai chấm ':' được thể hiện ở đây.
#define char_width 8
#define char_height 16
// các ký tự từ 0 đến 9 dấu hai chấm ':' trong định dạng XBM
static unsigned char char_bits[11][char_height] = {
  { 0x00, 0x38, 0x44, 0xA2, 0xA2, 0x91, 0x91, 0x89, 0x89, 0x85,
    0x45, 0x22, 0x1C, 0x00, 0x00, 0x00 },
  { 0x00, 0x10, 0x18, 0x14, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10,
    0x10, 0x70, 0x0C, 0x00, 0x00, 0x00 },
  { 0x00, 0x3C, 0x42, 0x81, 0x80, 0x40, 0x20, 0x10, 0x08, 0x04,
    0x02, 0xE1, 0x1F, 0x00, 0x00, 0x00 },
  { 0x00, 0x3C, 0x43, 0x80, 0x80, 0x60, 0x30, 0x4C, 0x80, 0x80,
    0x40, 0x30, 0x0F, 0x00, 0x00, 0x00 },
  { 0x00, 0x40, 0x60, 0x50, 0x48, 0x44, 0xE2, 0x5F, 0x40, 0x40,
    0x40, 0x40, 0x40, 0x00, 0x00, 0x00 },
  { 0x00, 0xF0, 0x0F, 0x01, 0x01, 0x7D, 0x83, 0x80, 0x80, 0x40,
    0x30, 0x0C, 0x03, 0x00, 0x00, 0x00 },
```

```

{ 0x00, 0x10, 0x08, 0x04, 0x02, 0x02, 0x39, 0x45, 0x83, 0x81,
  0x81, 0x42, 0x3C, 0x00, 0x00, 0x00 },
{ 0x00, 0xFF, 0x80, 0x40, 0x20, 0x10, 0x08, 0x08, 0x04, 0x04,
  0x02, 0x02, 0x02, 0x00, 0x00, 0x00 },
{ 0x00, 0x3C, 0x42, 0x81, 0x81, 0x42, 0x3C, 0x42, 0x81, 0x81,
  0x81, 0x42, 0x3C, 0x00, 0x00, 0x00 },
{ 0x00, 0x3C, 0x42, 0x81, 0x81, 0xC1, 0xA2, 0x9C, 0x40, 0x40,
  0x20, 0x10, 0x08, 0x00, 0x00, 0x00 },
{ 0x00, 0x00, 0x00, 0x00, 0x08, 0x08, 0x00, 0x00, 0x00, 0x08,
  0x08, 0x00, 0x00, 0x00, 0x00, 0x00 },
};

```

```
//-----
```

Tệp Counter.H

// Định nghĩa lớp CCounterExtension - lớp chính của chương trình.

```

class CCounterExtension : public CHttpServer
{
public:
    CCounterExtension();
    ~CCounterExtension();
    BOOL GetExtensionVersion(HSE_VERSION_INFO* pVer);
    void Default(CHttpServerContext* pCtxt);
    void Clock(CHttpServerContext* pCtxt);
    DECLARE_PARSE_MAP()

private:
    CMapStringToString m_Paths;
    int GetPageCount(CString& szPage);
    void OutputXBM(CHttpServerContext* pCtxt, CString& szDigits);
};

```

```
//-----
```

Tệp Resource.H

```

// sử dụng cho counter.rc
#define IDS_SERVER            102
#define IDS_RETRYERR         103
#define IDS_OPENERR          104
#define IDS_SERIALIZEERR      105
// Tiếp theo là các giá trị mặc định cho các đối tượng mới
#define _APS_NEXT_RESOURCE_VALUE 103
#define _APS_NEXT_COMMAND_VALUE 32768

```

```

#define _APS_NEXT_CONTROL_VALUE    103
#define _APS_NEXT_SYMED_VALUE     103
//-----
Tệp Counter.RC

#include "resource.h"
#include "afxres.h"
// Phiên bản
VS_VERSION_INFO VERSIONINFO
FILEVERSION 4.2.0.1
PRODUCTVERSION 4.2.0.1
FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
FILEFLAGS 0x1L
#else
FILEFLAGS 0x0L
#endif
FILEOS 0x4L
FILETYPE 0x2L
FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "CompanyName", "GIS Department - VN IOIT - July/1999\0"
            VALUE "FileDescription", "Chương trình dịch vụ Internet COUNTER\0"
            VALUE "FileVersion", "4. 2. 0. 1\0"
            VALUE "InternalName", "COUNTER\0"
            VALUE "LegalCopyright", "Copyright ©1999 GIS Department\0"
            VALUE "OriginalFilename", "COUNTER.DLL\0"
            VALUE "ProductName", "Chương trình mở rộng dịch vụ Internet COUNTER \0"
            VALUE "ProductVersion", "4, 2, 0, 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END

```

```
// Bảng xâu ký tự
STRINGTABLE DISCARDABLE
BEGIN
    IDS_SERVER        "Chương trình mở rộng dịch vụ Internet COUNTER"
    IDS_RETRYERR      "Có quá nhiều yêu cầu mở tệp nhật ký của chương trình COUNTER!\n"
    IDS_OPENERR       "Lỗi khi mở tệp nhật ký\nNguyên nhân: %s\n"
    IDS_SERIALIZEERR   "Không thể vào/ra từ tệp COUNTER \nNguyên nhân: %s\n"
END

#include "afxres.rc"    // Các thành phần chuẩn
#include "afxisapi.rc"  // Tài nguyên trợ giúp Internet
//-----

Tệp Counter.DEF

:Khai báo các tham số môđun chương trình
LIBRARY        "COUNTER"
EXPORTS
    HttpExtensionProc
    GetExtensionVersion
//-----

Tệp Counter.CPP

// Định nghĩa và khởi tạo các thủ tục cho chương trình.
// Chương trình này có thể thực hiện theo ba phương pháp:
// <IMG SRC="/scripts/counter.dll?clock">
// sẽ trả lại ảnh xbitmap có chứa thời gian hiện hành của hệ thống
// <IMG SRC="/scripts/counter.dll/mydir/mypage.htm">
// sẽ trả lại ảnh x-bitmap chứa
// số lần thăm nhập tệp /mydir/mypage.htm
// <IMG SRC="/scripts/counter.dll">
// sẽ trả lại ảnh xbitmap chứa số lần một trang gọi tới chương trình counter.dll
#include <afx.h>
#include <afxdb.h>
#include <afxisapi.h>
#include "resource.h"
#include "counter.h"
#include "charset.h"

// Chương trình gửi lại ảnh xbitmap chứ không phải trang HTML.
static const TCHAR szContentType[] = _T("Content-Type: image/x-xbitmap\r\n");
// Các dòng sau định nghĩa nơi lưu trữ và tên tệp nhật ký (log file).
// Tệp này nên đặt ở thư mục đọc ghi được.
```

```

static const TCHAR szLogFile[] = _T("C:\\counter.log");
// Hai dòng sau sẽ thông báo cho máy trạm không dùng cache cho thông tin này
// bởi vì nó là thông tin động.
static const TCHAR szExpires[] = _T("Expires: Thu, 01 July 2000 01:00:00 GMT\\r\\n");
static const TCHAR szNoCache[] = _T("Pragma: no-cache\\r\\n");
// ánh xạ phân tích cú pháp câu lệnh
BEGIN_PARSE_MAP(CCounterExtension, CHttpServer)
    ON_PARSE_COMMAND(Clock, CCounterExtension, ITS_EMPTY)
    ON_PARSE_COMMAND(Default, CCounterExtension, ITS_EMPTY)
    DEFAULT_PARSE_COMMAND(Default, CCounterExtension)
END_PARSE_MAP(CCounterExtension)
// Chỉ có duy nhất một đối tượng CCounterExtension
CCounterExtension theExtension;
// Viết bổ sung các hàm của CCounterExtension
CCounterExtension::CCounterExtension()
{
}
CCounterExtension::~CCounterExtension()
{
}
BOOL CCounterExtension::GetExtensionVersion(HSE_VERSION_INFO* pVer)
{
    // Gọi giá trị mặc định để khởi tạo
    CHttpServer::GetExtensionVersion(pVer);
    // Nạp xâu mô tả
    TCHAR sz[HSE_MAX_EXT_DLL_NAME_LEN+1];
    ISAPIVERIFY(::LoadString(AfxGetResourceHandle(), IDS_SERVER, sz, HSE_MAX_EXT_DLL_NAME_LEN));
    _tcsncpy(pVer->lpszExtensionDesc, sz);
    return TRUE;
}
// Các lệnh của đối tượng CCounterExtension
void CCounterExtension::Default(CHttpServerContext* pCtxt)
{
    // Không cần gọi hàm StartContent() hoặc hàm WriteTitle() ở đây vì
    // các hàm này gửi lại các thẻ chỉ dành cho trang HTML. Chúng ta sẽ gửi lại một ảnh.
    // Không cho phép các trang này bị cache
    AddHeader(pCtxt, szExpires);
    AddHeader(pCtxt, szNoCache);
    TCHAR pstrBuffer[1024];

```

```

// Kiểm tra đường dẫn đi sau tên của chương trình DLL.
// thí dụ như <IMG SRC="/scripts/counter.dll/mydir/mypage.htm">
// Nếu có, sử dụng đường dẫn này. Nếu không, sử dụng URL của
// trang gọi tới chương trình DLL này.
if (_tcslen(pCtxt->m_pECB->lpszPathInfo) != 0) {
    _tcsncpy(pstrBuffer, pCtxt->m_pECB->lpszPathInfo);
}
else
{
    // HTTP_REFERER chứa URL đầy đủ của trang đã gọi chương trình DLL này
    DWORD dwSize = 1024;
    pCtxt->GetServerVariable(_T("HTTP_REFERER"), pstrBuffer, &dwSize);
}
// Gọi hàm thành phần để xem trang đã được thăm nhập bao nhiêu lần và tăng số đếm.
// GetPageCount sẽ trả lại giá trị -1 nếu có lỗi khi đọc tệp nhật ký của counter
CString szPath(pstrBuffer);
int nCount = GetPageCount(szPath);
if (nCount != -1)
{
    // gọi hàm thành phần để đưa ra ảnh xbitmap chứa các số
    CString szCount;
    szCount.Format(_T("%d"), nCount);
    OutputXBM(pCtxt, szCount);
}
// EndContent() chỉ thích hợp cho tệp HTML, nên chúng ta không cần gọi nó ở đây.
}

void CCounterExtension::Clock(CHttpServerContext* pCtxt)
{
    // Hàm Clock được gọi khi chương trình được thăm nhập với khuôn dạng
    // <IMG SRC="/scripts/counter.dll?Clock">
    // Không cần gọi hàm StartContent() hoặc hàm WriteTitle() ở đây vì
    // các hàm này gửi lại các thẻ chỉ dành cho trang HTML. Chúng ta sẽ gửi lại một ảnh.
    // Không cho phép các trang này bị cache
    AddHeader(pCtxt, szExpires);
    AddHeader(pCtxt, szNoCache);
    // Lấy giờ hệ thống hiện thời và biến đổi về xâu ký tự
    CTime time = CTime::GetCurrentTime();
    CString szTime = time.Format(_T("%H%%M"));
    // gọi hàm thành phần để tạo ra ảnh xbitmap chứa các chữ số
    OutputXBM(pCtxt, szTime);
}

```



```

// EndContent() chỉ thích hợp cho tệp HTML, nên chúng ta không cần gọi nó ở đây.
}

void CCounterExtension::OutputXBM(CHttpContext* pCtxt, CString& szDigits)
{
    // Hàm đọc một xâu ký tự bao gồm các chữ số từ 0..9 và dấu : .
    // tạo ra ảnh xbitmap cho xâu ký tự và gửi ra vùng đệm. Bắt đầu ghi kiểu thích hợp cho máy trạm
    AddHeader(pCtxt, szContentType);
    // Ghi nhớ: đoạn chương trình này chỉ làm việc đúng với ảnh dữ liệu có kích thước 8x8
    int nFinalWidth = char_width * szDigits.GetLength();
    int nFinalHeight = char_height;
    // Ghi ra phần đầu ảnh XBM. Chúng ta biến đổi về kiểu long int bởi vì
    // toán tử tách (<<) không được viết lại để thâm nhập một số kiểu int.
    *pCtxt << _T("#define counter_width ") << (long int)nFinalWidth << _T("\r\n");
    *pCtxt << _T("#define counter_height ") << (long int)nFinalHeight << _T("\r\n");
    *pCtxt << _T("static unsigned char counter_bits[] = {\r\n");
    // Tạo ảnh theo từng hàng
    for (int nLine=0; nLine<nFinalHeight; nLine++)
    {
        for (int nChar=0; nChar<szDigits.GetLength(); nChar++)
        {
            int nDigitOffset;
            if (szDigits[nChar] >= _TEXT('0') && szDigits[nChar] <= _TEXT('9'))
                nDigitOffset = szDigits[nChar] - _TEXT('0');
            else
                nDigitOffset = 10; // dấu hai chấm có chỉ số 10 trong bảng bitmap
            CString szHex;
            szHex.Format(_T("0x%02X, "), char_bits[nDigitOffset][nLine]);
            *pCtxt << szHex;
        }
        *pCtxt << _T(");\r\n");
    }
}

int CCounterExtension::GetPageCount(CString& szPage)
{
    // Đưa vào trang được xác định duy nhất (szPage), kiểm tra trong "CSDL"
    // xem trang này đã được thâm nhập bao nhiêu lần và hàm trả lại
    // số lần thâm nhập. Nếu việc thực hiện có lỗi, hàm sẽ trả lại giá trị -1
    CFile file;
    CFileException e;
    int nOpenAttempts = 0;

```

```

// để chắc chắn chỉ có duy nhất một thread thao tác trên tệp nhật ký tại một thời điểm
while (TRUE)
{
    if (file.Open(szLogFile, CFile::modeRead | CFile::shareExclusive
        | CFile::modeCreate | CFile::modeNoTruncate, &e))
    { // Mở tệp thành công ,
        break;
    }
    else
    { // Không mở được tệp... xem nguyên nhân tại sao
        if (e.m_cause == CFileException::sharingViolation)
        { // Đụng độ, một thread khác đang mở tệp, đợi để thực hiện lại
            ::Sleep(100);
            nOpenAttempts++; // tăng số đếm thử
            if (nOpenAttempts == 30)
            { // quá nhiều lần làm lại, có một lỗi nào đó
                CString szErrText;
                szErrText.LoadString(IDS_RETRYERR);
                TRACE0(szErrText);
                return -1; // Trả về giá trị -1, không thành công
            }
        }
        else
        {
            // không phải là lỗi đụng độ. Có thể là một lỗi nghiêm trọng
            TCHAR szCause[255];
            e.GetErrorMessage(szCause, 255);
            CString szErrText;
            szErrText.LoadString(IDS_OPENERR);
            TRACE1(szErrText, szCause);
            return -1; // trả về giá trị -1, không thành công
        }
    }
}

// Mở tệp nhật ký thành công, bắt đầu đọc tệp
CArchive archive(&file, CArchive::load);
// đọc tuần tự dữ liệu
try {
    m_Paths.Serialize(archive);
}

```

```

}
catch (CArchiveException* e)
{ // nếu đọc đến cuối tệp, có thể số đếm chưa tồn tại, tuy nhiên đây không phải là lỗi
    if (e->m_cause != CArchiveException::endOfFile) {
        TCHAR szCause[255];
        e->GetErrorMessage(szCause, 255);
        CString szErrText;
        szErrText.LoadString(IDS_SERIALIZEERR);
        TRACE1(szErrText, szCause);
        m_Paths.RemoveAll();
        archive.Close();
        file.Close();
        e->Delete();
        return -1;
    }
    e->Delete();
}
archive.Close();
file.Close();
// Thử tìm trang yêu cầu trong tệp nhật ký
int nVal = 0;
CString szCount;
if (m_Paths.Lookup(szPage, szCount))
    nVal = atoi(szCount);
nVal++; // Tăng số lần truy nhập.
// Đặt giá trị mới.
szCount.Format(_T("%d"), nVal);
m_Paths.SetAt(szPage, szCount);
// Ghi lại giá trị mới vào tệp nhật ký
if (!file.Open(szLogFile, CFile::modeWrite | CFile::shareExclusive | CFile::modeCreate, &e))
{
    TCHAR szCause[255];
    e.GetErrorMessage(szCause, 255);
    CString szErrText;
    szErrText.LoadString(IDS_OPENERR);
    TRACE1(szErrText, szCause);
    return -1; // Không mở được tệp để ghi, trả về giá trị -1
}
CArchive archiveStore(&file, CArchive::store);

```

```

try {
    m_Paths.Serialize(archiveStore);
}
catch (CArchiveException* e)
{
    TCHAR szCause[255];
    e->GetErrorMessage(szCause, 255);
    CString szErrText;
    szErrText.LoadString(IDS_SERIALIZEERR);
    TRACE1(szErrText, szCause);
    archiveStore.Close();
    file.Close();
    e->Delete();
    return -1; // trả về giá trị -1, lỗi.
}
archiveStore.Close();
file.Close();
return nVal; // Trả về số lần truy nhập
}
//-----

```

Sau khi dịch xong chương trình, sao chép tệp COUNTER.DLL vào thư mục thực hiện của dịch vụ Web trên máy chủ, sao chép các tệp MFCxx.DLL của MFC (các tệp này phải cùng phiên bản với chương trình MFC đã dịch chương trình COUNTER) vào thư mục SYSTEM của WINDOWS trên máy chủ.

Tạo trang HTML để thăm nhập chương trình COUNTER. Trang HTML sau giới thiệu ba cách gọi chương trình COUNTER. Trang HTML giả định chương trình COUNTER.DLL được cài đặt trong thư mục ảo /SCRIPTS trên máy chủ.

```

<HTML>
<HEAD>
<TITLE>Giới thiệu chương trình COUNTER</TITLE>
</HEAD>
<BODY>
Giờ hiện thời của hệ thống:
<IMG SRC="/scripts/counter.dll?clock"><P>
Số lần ghé thăm trang /mydir/mypage.htm:
<IMG SRC="/scripts/counter.dll/mydir/mypage.htm"><P>
Số lần mở trang này:
<IMG SRC="/scripts/counter.dll"><P>

```

```
</BODY>
</HTML>
```

Nếu thư mục /SCRIPTS được phân quyền READ và EXECUTE, thay đổi các dòng như sau:

```
...
Số lần ghé thăm trang /mydir/mypage.htm:
<IMG SRC="/scripts/counter.dll/mydir/mypage.htm?"><P>
Số lần mở trang này:
<IMG SRC="/scripts/counter.dll?"><P>
...
```

Lớp chính của chương trình là lớp CCounterExtension, suy diễn từ lớp CHttpServer.

12.1.2.2 Lớp CHttpServer

Một chương trình mở rộng dịch vụ Internet bao giờ cũng có một lớp chính suy diễn từ lớp CHttpServer, thí dụ lớp CCounterExtension trong chương trình COUNTER, các hàm chức năng của chương trình mở rộng dịch vụ Internet được khai báo trong lớp này. Lớp CHttpServer cùng với lớp CHttpServerContext cung cấp một phương pháp mở rộng các hàm tuân thủ ISAPI của dịch vụ HTTP. Các lớp này được khai báo trong tệp `afxisapi.h` của MFC. Lớp CHttpServer chứa các hàm ISAPI và có thể xử lý tất cả các kiểu yêu cầu của máy trạm, nó được dùng cho cả chương trình CGI và chương trình dịch vụ Internet.

Khi dịch vụ HTTP tiếp nhận một yêu cầu gửi từ trình duyệt trên máy trạm, một đối tượng CHttpServer được khởi tạo và một đối tượng CHttpServerContext cũng được tạo ra để quản lý và phục vụ yêu cầu máy trạm. Mỗi chương trình mở rộng dịch vụ Internet chỉ tồn tại một đối tượng CHttpServer, tuy nhiên nó có thể có nhiều đối tượng CHttpServerContext vì đối tượng CHttpServerContext được tạo ra cho mỗi yêu cầu của máy trạm.

Một đối tượng CHttpServer sử dụng nhiều đối tượng CHttpServerContext thực hiện trong các *thread* riêng biệt. Thiết kế này cho phép xử lý cùng một lúc nhiều yêu cầu của các máy trạm gọi tới đối tượng CHttpServer. Đối tượng CHttpServer liên lạc với máy trạm hay chính bản thân máy chủ thông qua đối tượng CHttpServerContext.

Sau khi khởi động rồi nạp chương trình dịch vụ Internet, máy chủ thực hiện ngay hàm `GetExtensionVersion` để lấy số hiệu phiên bản làm cơ sở cho chương trình. Hàm thành phần `HttpExtensionProc` được gọi mỗi khi có yêu cầu của máy trạm. Trong trường hợp mặc định, `HttpExtensionProc` sẽ đọc dữ liệu của máy trạm và quyết định sẽ thực hiện hành động nào. Lập trình viên có thể viết lại hàm này để tối ưu cho công việc của mình, tuy nhiên chúng ta nên dùng chế độ xử lý mặc định.

Các hàm thành phần khác của lớp CHttpServer dành cho việc xử lý các yêu cầu của máy trạm,

định dạng dữ liệu trả lời và trao đổi với máy trạm. Các hàm thành phần của lớp `CHttpServer` được mô tả dưới đây:

Cấu tử

`CHttpServer` Tạo lập một đối tượng của lớp `CHttpServer`.

Các hàm nạp chồng

`CallFunction` Tìm và thực hiện hàm tương thích được liên kết với câu lệnh trong URL

`OnParseError` Kiến tạo mô tả các lỗi trả về cho máy trạm

`HttpExtensionProc` Sử dụng hàm *callback* để đọc dữ liệu từ máy trạm và quyết định lựa chọn hành động nào

`GetExtensionVersion` Lấy số liệu phiên bản làm cơ sở cho chương trình DLL

`ConstructStream` Kiến tạo đối tượng `CHtmlStream`

Các hàm thuộc tính

`StartContent` Chèn các thẻ bắt đầu trang HTML vào đối tượng `CHtmlStream` để gửi lại cho máy trạm. Có thể viết lại hàm này để thay đổi hoặc bỏ qua các thẻ mặc định

`EndContent` Chèn các thẻ kết thúc trang HTML vào đối tượng `CHtmlStream` để gửi lại cho máy trạm. Có thể viết lại hàm này để thay đổi hoặc bỏ qua các thẻ mặc định

`WriteTitle` Chèn tiêu đề trang HTML vào đối tượng `CHtmlStream` để gửi lại cho máy trạm. Có thể viết lại hàm này để tạo tiêu đề mong muốn

`GetTitle` Lấy tiêu đề của tài liệu HTML được gửi cho máy trạm

`AddHeader` Công thêm phần đầu cho việc trả lời trước khi nó được gửi cho máy chủ

`InitInstance` Đặt các giá trị khởi tạo cho đối tượng `CHttpServer`. Có thể viết lại hàm này để đặt các giá trị khởi tạo mong muốn cho đối tượng `CHttpServer`

Để tạo trang HTML gửi lại cho máy trạm, ta xây dựng hàm xử lý theo mẫu sau:

```
void CCounterExtension::ClockHTML(CHttpServerContext* pCbxt)
{
    StartContent(pCbxt); //chèn thẻ bắt đầu trang HTML
    WriteTitle(pCbxt); //chèn tiêu đề của trang HTML
    //xử lý dữ liệu và tạo nội dung trang HTML
    // Lấy giờ hệ thống hiện thời của máy chủ và biến đổi kết quả thành chuỗi ký tự
    CTime time = CTime::GetCurrentTime();
    CString szTime = time.Format(_T("%H:%M"));
    // Các dòng dữ liệu HTML
    *pCbxt << _T("<b><i>Giờ hiện thời của máy chủ:</b></i>");
    *pCbxt << _T("<h3>");
    *pCbxt << _T(szTime);
    *pCbxt << _T("</h3>");
    //...
    EndContent(pCbxt); //chèn thẻ kết thúc trang HTML
}
```

Trở lại với thí dụ COUNTER. Thông thường một chương trình dịch vụ Internet gửi lại kết quả cho máy trạm dưới khuôn dạng trang HTML, một chương trình dịch vụ Internet muốn gửi dữ liệu cho máy trạm với khuôn dạng khác cần có một số thay đổi nhỏ. Thứ nhất, không cần gọi các hàm chức năng **StartContent** và **WriteTitle** bởi vì các hàm này tạo nên phần đầu dữ liệu chỉ áp dụng cho các trang HTML. Thứ hai, hàm **AddHeader** cần được gọi với nội dung thích hợp với kiểu xâu ký tự. Cuối cùng, không cần gọi hàm **EndContent**, bởi vì hàm này cũng chỉ áp dụng cho các trang HTML. Chúng ta có thể lấy hàm **Clock** làm thí dụ:

```
static const TCHAR szExpires[] = _T("Expires: Thu, 01 July 2000 01:00:00 GMT\r\n");
static const TCHAR szNoCache[] = _T("Pragma: no-cache\r\n");
...
void CCounterExtension::Clock(CHttpServerContext* pCtxt)
{
    // Hàm Clock được gọi khi chương trình được thăm nhập với khuôn dạng
    // <IMG SRC="/scripts/counter.dll?Clock">
    // Chúng ta không cần gọi hàm StartContent hoặc hàm WriteTitle ở đây vì
    // các hàm này gửi lại các thẻ chỉ dành cho trang HTML.
    // Chúng ta sẽ gửi lại một ảnh. Không cho phép các trang này bị cache
    AddHeader(pCtxt, szExpires);
    AddHeader(pCtxt, szNoCache);
    // Lấy giờ hệ thống hiện thời, và biến đổi về xâu ký tự
    CTime time = CTime::GetCurrentTime();
    CString szTime = time.Format(_T("%H%%M"));
    // gọi hàm thành phần để tạo ra ảnh xbitmap chứa các chữ số
    OutputXBM(pCtxt, szTime);
    // EndContent chỉ thích hợp cho tệp HTML, nên chúng ta không cần gọi nó ở đây.
}
```

Hàm **Clock** được máy trạm gọi để lấy giờ hệ thống của máy chủ. Thay vì trả lại kết quả dưới dạng trang HTML thì hàm **Clock** gửi lại kết quả dưới dạng dữ liệu ảnh *xbitmap*, do đó để hiển thị được kết quả này trong trang HTML ta phải đưa vào thẻ hiển thị ảnh như sau:

```
<IMG SRC="/scripts/counter.dll?Clock">.
```

Nếu chúng ta thay nội dung hàm **Clock** bằng nội dung hàm **ClockHTML** như trên thì việc lấy giờ hệ thống có thể được gọi từ trình duyệt như sau:

```
http://myserver/scripts/counter.dll?Clock.
```

Trong hàm **Clock**, việc gọi hàm **OutputXBM** sẽ biến đổi xâu dữ liệu **szTime** thành ảnh *xbitmap* và gửi ảnh này cho đối tượng **CHttpServerContext**. Trong thí dụ này, kiểu ảnh *xbitmap* được lựa chọn bởi vì dữ liệu kết quả gửi về cho máy trạm hoàn toàn là các ký tự, như vậy chọn kiểu ảnh này để dễ viết chương trình. Dữ liệu ảnh được tạo trong hàm **OutputXBM** thông qua các bit ảnh lưu trong tệp **CHARSET.H**. Kiểu ảnh *xbitmap* đòi hỏi các bit ảnh được gửi theo thứ tự đảo ngược. Do

đó các bit trong tệp CHARSET.H đã được đảo ngược. Thí dụ số "3" sẽ như sau:

```
0x3E 00111110
0x63 00110011
0x60 01100000
0x60 01100000
0x3C 00111101
0x60 01100000
0x60 01100000
0x60 01100000
0x63 01100011
0x3E 00111110
```

Ảnh *xbitmap* ở dạng đen trắng. Nếu muốn có ảnh màu, ta cần tạo ảnh ở các dạng ảnh nhị phân khác, thí dụ như GIF. Để gửi lại dữ liệu cho máy trạm dưới dạng nhị phân, chúng ta cần bổ sung một số bước. Bởi vì lớp CHtmlStream không cho phép gửi một chuỗi các byte riêng lẻ, cho nên ta cần tạo ra một lớp phái sinh từ lớp CHtmlStream, sau đó sử dụng lớp này để gửi dữ liệu về cho máy trạm. Thí dụ:

```
class CBinaryHtmlStream : public CHtmlStream
{
public:
    // Nạp chồng toán tử tách (<<) để gửi từng byte
    CBinaryHtmlStream& operator<<(BYTE b)
    {
        Write(&b, 1);
        return *this;
    }
}
```

Để gửi các dữ liệu của loại nói trên, hãy thay thế nội dung hàm OutputXBM bằng đoạn mã trình như sau:

```
static const TCHAR szContentType[] = _T("Content-Type: image/gif\r\n");
void CCounterExtension::OutputXBM(CHttpServerContext* pCtxt, CString& szDigits)
{
    // đoạn mã trình dưới đây sẽ lấy dữ liệu chứa trong szDigits để tạo một ảnh mới
    // ở dạng ảnh GIF. ảnh này sẽ được chứa trong bộ nhớ mảng m_ImageByteArray[].
    // Số lượng byte của ảnh được lưu trong biến m_nNumberBytes.
    // Xác định kiểu dữ liệu gửi về cho máy trạm
    AddHeader(pCtxt, szContentType);
    CBinaryHtmlStream* pStream = new CBinaryHtmlStream;
    ISAPIVERIFY(pStream != NULL);
    int nCount;
    for (nCount = 0; nCount < m_nNumberBytes; nCount++)
        *pStream << m_ImageByteArray[nCount];
    *pCtxt << *pStream;
```

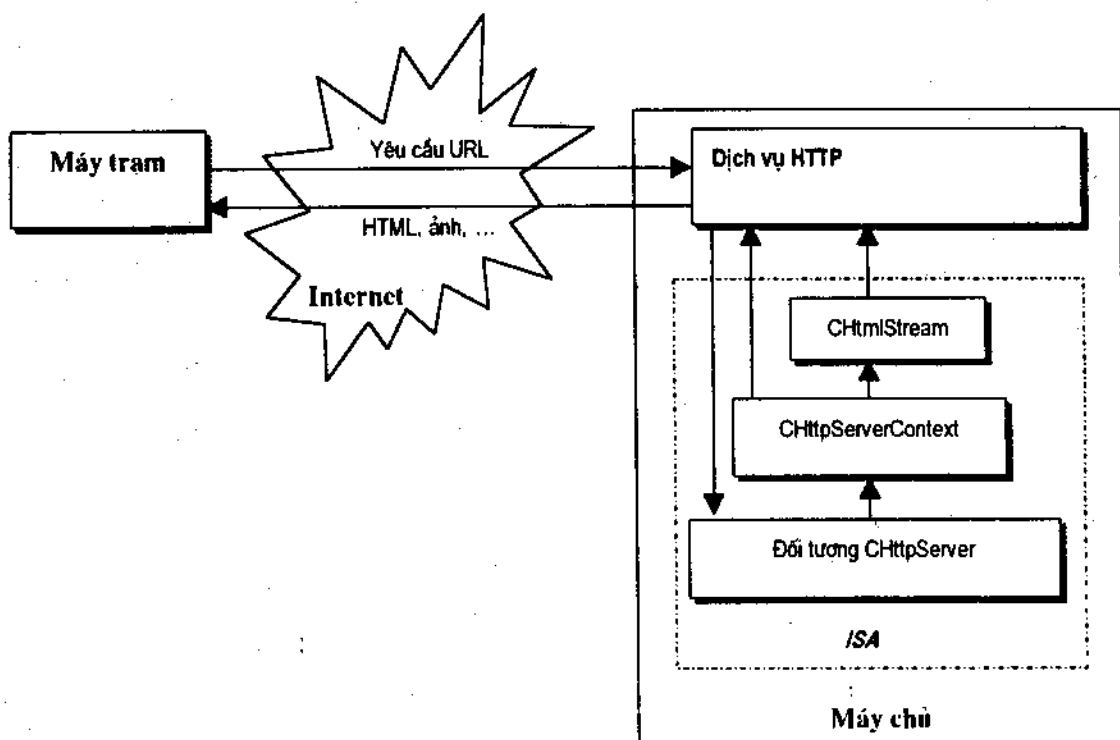


```

delete pStream;
// ... các mã trình ...
}

```

Trong thí dụ COUNTER ta thấy các hàm thành phần của lớp CHttpServer thường có một tham số là con trỏ trỏ tới đối tượng CHttpServerContext. Đối tượng CHttpServerContext cung cấp các công cụ mà đối tượng CHttpServer cần có để xử lý dữ liệu mà máy trạm gửi về HTTP server. Khi chương trình dịch vụ Internet xử lý các yêu cầu, nó sử dụng các hàm thành phần của lớp CHttpServerContext để thi hành các nhiệm vụ.



Hình 12.2 Sơ đồ trao đổi thông tin giữa máy trạm với chương trình ISA

12.1.2.3 Lớp CHttpServerContext

Các thành phần của lớp CHttpServerContext được mô tả như dưới đây:

Các thành phần dữ liệu

m_pECB Con trỏ tới cấu trúc EXTENSION_CONTROL_BLOCK
m_pStream Con trỏ tới đối tượng CHtmlStream

Cấu tử

CHttpServerContext Kiến tạo đối tượng CHttpServerContext

Các thao tác

GetServerVariable	Sao chép thông tin liên quan tới giao tiếp HTML hoặc với bản thân máy chủ vào trong một vùng đệm
WriteClient	Gửi ngay tức thì thông tin cho máy trạm có yêu cầu HTTP
ReadClient	Đọc thông tin từ phần thân của một yêu cầu HTTP của máy trạm vào một vùng đệm
ServerSupportFunction	Cung cấp cho ISA các hàm có mục đích tổng thể cũng như các hàm cụ thể để bổ sung cho dịch vụ HTTP

Toán tử

<<	Ghi dữ liệu vào dòng dữ liệu
----	------------------------------

Khi chương trình dịch vụ Internet cần phải trao đổi thông tin với máy trạm, thí dụ như muốn gửi ngay thông báo lỗi cho máy trạm gọi hàm **WriteClient** để thực hiện. Trong các trường hợp khác, máy chủ nên gửi các thông báo cho máy trạm thông qua thành phần dữ liệu **m_pStream**. **m_pStream** là con trỏ trỏ tới đối tượng **CHtmlStream**, đối tượng để quản lý tệp (trang) HTML trong bộ nhớ. Tệp bộ nhớ HTML rất có hiệu quả trong việc lưu trữ tạm thời các byte dữ liệu thô hoặc các đối tượng nối tiếp trước khi truyền đi. Mặc dù lớp **CHtmlStream** không phải là lớp thừa hưởng từ lớp **CFile**, nhưng nó được đối xử như lớp **CMemFile** trừ khi **CHtmlStream** được sử dụng để chứa dữ liệu trong vùng đệm trước khi gửi đi. Dữ liệu chứa trong tệp bộ nhớ **CHtmlStream** không thể đọc được. Thông thường đối tượng **CHtmlStream** được tự động tạo ra và do **CHttpServer::ConstructStream** quản lý; tuy nhiên chúng ta có thể nạp chồng lại hàm **CHttpServer::ConstructStream** để thực hiện các yêu cầu riêng của mình.

Chương trình dịch vụ Internet trao đổi thông tin với máy chủ thông qua cấu trúc **EXTENSION_CONTROL_BLOCK**, cấu trúc để quản lý các câu lệnh của máy tính trạm. Mỗi đối tượng **CHttpServer** chứa một cấu trúc **EXTENSION_CONTROL_BLOCK** do máy chủ cung cấp.

Khi đối tượng **CHttpServer** tiếp nhận một yêu cầu của máy trạm, ánh xạ phân tích cú pháp sẽ phân tích lệnh, các tham số và gán lệnh cho hàm thành phần tương ứng, các tham số với các tham số của hàm.

12.1.2.4 Ánh xạ phân tích cú pháp

Các biểu mẫu HTML cơ bản được sử dụng để sưu tập thông tin từ người sử dụng. Các thông tin này được gửi về máy chủ thông qua yêu cầu URL, yêu cầu URL này bao gồm tên của chương trình mở rộng dịch vụ Internet, tên hàm chức năng và danh sách các tham số; các tham số dựa trên cơ sở các thông tin mà người sử dụng nhập vào các biểu mẫu HTML. Khi nhận được yêu cầu URL, máy chủ sẽ phân tích các tham số bằng việc gửi nó qua một loạt các phân tích *macro* trong ánh xạ phân tích cú pháp, ánh xạ phân tích cú pháp sẽ gọi thực hiện hàm tương ứng được khai báo trong lớp chính của chương trình dịch vụ Internet và gửi lại kết quả xử lý cho máy trạm. MFC cho phép tạo phân tích các tham số rất dễ: lập trình viên viết các hàm, khai báo các tham số của chúng, và sử dụng *macro parse map* để "cài" các tham số dòng lệnh vào các tham số của hàm, bao gồm cả các

lựa chọn và các tham số mặc định. Chỉ có một ánh xạ phân tích cú pháp duy nhất được tạo cho một đối tượng CHttpServer. Bảng 12.2 liệt kê các *macro* ánh xạ phân tích cú pháp của ISAPI.

Bảng 12.2 Ánh xạ lệnh của ISAPI

<i>Macro</i>	<i>Hành động</i>
BEGIN_PARSE_MAP	Bắt đầu định nghĩa một ánh xạ phân tích
ON_PARSE_COMMAND	Phân tích câu lệnh gửi từ máy trạm
ON_PARSE_COMMAND_PARAM	Định nghĩa câu lệnh cho đối tượng CHttpServer từ máy trạm
DEFAULT_PARSE_COMMAND	Gọi tới hàm mặc định
END_PARSE_MAP	Kết thúc định nghĩa của ánh xạ phân tích

Khai báo ánh xạ phân tích cú pháp

Trong thí dụ COUNTER, ánh xạ phân tích cú pháp được khai báo như sau:

```
BEGIN_PARSE_MAP(CCounterExtension, CHttpServer)
    ON_PARSE_COMMAND(Clock, CCounterExtension, ITS_EMPTY)
    ON_PARSE_COMMAND(Default, CCounterExtension, ITS_EMPTY)
    DEFAULT_PARSE_COMMAND(Default, CCounterExtension)
END_PARSE_MAP(CCounterExtension)
```

Ánh xạ phân tích cú pháp trên khai báo hai hàm Clock và Default là hai chức năng mà COUNTER thực hiện được. Hàm Default được khai báo thêm trong DEFAULT_PARSE_COMMAND, điều này chỉ ra rằng hàm sẽ được thực hiện mặc định khi máy trạm gọi COUNTER.DLL mà không chỉ rõ phải thực hiện chức năng nào. Thí dụ, máy trạm gửi yêu cầu:

`http://myserver/scripts/counter.dll`

sẽ tương đương với gửi yêu cầu:

`http://myserver/scripts/counter.dll?default.`

Sau đây ta sẽ xem kỹ hơn về các lệnh *macro* trong ánh xạ phân tích cú pháp.

BEGIN_PARSE_MAP

Cú pháp: `BEGIN_PARSE_MAP(theClass, baseClass)`

Trong đó, tham số *theClass* chỉ ra tên của lớp làm chủ ánh xạ phân tích cú pháp này, tham số *baseClass* chỉ tên lớp cơ sở của lớp *theClass*; *baseClass* phải là lớp phái sinh từ lớp CHttpServer.

Thí dụ sử dụng: `BEGIN_PARSE_MAP(CCounterExtension, CHttpServer)`

Macro `BEGIN_PARSE_MAP` được sử dụng để bắt đầu định nghĩa ánh xạ cú pháp. Khi đối tượng `CHttpServer` tiếp nhận câu lệnh máy trạm, ánh xạ phân tích cú pháp liên kết câu lệnh với hàm thành phần và các tham số tương ứng trong lớp. Trong tệp chương trình (.CPP) định nghĩa các hàm thành phần của lớp *theClass*, bắt đầu ánh xạ phân tích với *macro* `BEGIN_PARSE_MAP`, thêm các *macro* đầu vào cho mỗi phân tích cú pháp của hàm và các thuộc tính, kết thúc ánh xạ phân tích cú pháp bằng *macro* `END_PARSE_MAP`.

ON_PARSE_COMMAND

Cú pháp: `ON_PARSE_COMMAND(FnName, mapClass, Agrs)`

Trong đó, tham số `FnName` là tên của hàm thành phần, đây cũng là tên của lệnh, `mapClass` là tên của lớp mà hàm sẽ liên kết, `Agrs` là các tham số sẽ liên kết với các tham số của hàm `FnName`.

Thí dụ: `ON_PARSE_COMMAND(Clock, CCounterExtension, ITS_EMPTY)`

Macro `ON_PARSE_COMMAND` được sử dụng trong ánh xạ phân tích cú pháp để định nghĩa câu lệnh cho máy trạm gọi tới đối tượng `CHttpServer`. Hàm thành phần có tên xác định bởi `FnName` phải có tham số đầu tiên là một con trỏ tới đối tượng `CHttpServerContext`. Các tham số *Agrs* của `ON_PARSE_COMMAND` có thể nhận một trong các giá trị ở bảng 12.3.

Bảng 12.3 Các giá trị cho tham số của *macro* `ON_PARSE_COMMAND`

<i>Ký hiệu</i>	<i>Kiểu hoặc chú thích</i>
<code>ITS_EMPTY</code>	<i>Agrs</i> không thể là rỗng. Sử dụng <code>ITS_EMPTY</code> khi hàm không có tham số
<code>ITS_PSTR</code>	Con trỏ tới chuỗi ký tự
<code>ITS_I2</code>	Kiểu short
<code>ITS_I4</code>	Kiểu long
<code>ITS_R4</code>	Kiểu float
<code>ITS_R8</code>	Kiểu double

Thí dụ:

```
BEGIN_PARSE_MAP(CDerivedClass, CHttpServer)
    DEFAULT_PARSE_COMMAND(Myfunc, CDerivedClass)
    ON_PARSE_COMMAND(Myfunc, CDerivedClass, ITS_PSTR ITS_I2)
    ON_PARSE_COMMAND_PARAMS("string integer=42")
    ON_PARSE_COMMAND(Myfunc2, CDerivedClass, ITS_PSTR ITS_I2 ITS_PSTR)
    ON_PARSE_COMMAND_PARAMS("string integer string2='Default value'")
END_PARSE_MAP(CDerivedClass)
```

Trong đó các hàm `Myfunc` và `Myfunc2` khai báo như sau:

```
void Myfunc(CHttpServerContext* pCtxt, LPTSTR pszName, int nNumber);
```

```
void Myfunc2(CHttpServerContext* pCtxt, LPTSTR pszName, int nNumber, LPTSTR pszTitle);
```

Các vấn đề cần chú ý khi phát triển chương trình:

- Sử dụng dấu nhảy đơn nếu muốn đưa một chuỗi ký tự có chứa ký tự trống (*space*) làm giá trị mặc định của tham số ITS_PSTR.
- Tham số đầu tiên của hàm FnName phải là con trỏ tới đối tượng CHttpRequestContext, các tham số tiếp theo phải được khai báo theo đúng thứ tự đã định nghĩa trong *macro* ON_PARSE_COMMAND.

ON_PARSE_COMMAND_PARAMS

Cú pháp: ON_PARSE_COMMAND_PARAMS(Params)

Trong đó, **Params** là các tham số liên kết với các tham số Args trong *macro* ON_PARSE_COMMAND đứng ngay trước nó.

Macro ON_PARSE_COMMAND_PARAMS nhận biết và chỉ rõ giá trị mặc định cho các tham số của hàm FnName được khai báo trong *macro* ON_PARSE_COMMAND đứng ngay trước nó.

Thí dụ:

```
ON_PARSE_COMMAND(Myfunc, CDerivedClass, ITS_PSTR ITS_I2)
ON_PARSE_COMMAND_PARAMS("string integer=42")
```

Nếu một tham số được đặt tên, máy trạm phải cung cấp tên tham số khi yêu cầu. Thí dụ, nếu các tham số khai báo như sau:

```
ON_PARSE_COMMAND_PARAMS("string integer=42")
```

thì tham số *string* phải được máy trạm cung cấp, nếu không yêu cầu sẽ bị sai.

Nếu tham số là lựa chọn, máy trạm không cần thiết phải cung cấp nó, nếu máy trạm không cung cấp nó sẽ nhận giá trị mặc định. Thí dụ, nếu các tham số được khai báo như sau:

```
ON_PARSE_COMMAND_PARAMS("string=default integer=42")
```

thì không tham số nào bị buộc phải định nghĩa trong câu lệnh máy trạm, và tham số *string* trong trường hợp mặc định sẽ nhận giá trị là một chuỗi rỗng.

DEFAULT_PARSE_COMMAND

Cú pháp: DEFAULT_PARSE_COMMAND(FnName, mapClass)

Trong đó, **FnName** là tên của hàm thành phần, cũng là tên của câu lệnh, **mapClass** là tên lớp liên kết với hàm thành phần.

Nếu một yêu cầu từ máy trạm gửi cho đối tượng CHttpRequest không chỉ ra lệnh cụ thể, thì hàm FnName được khai báo trong *macro* DEFAULT_PARSE_COMMAND sẽ được gọi thực hiện.

Macro `DEFAULT_PARSE_COMMAND` có thể được đặt ở bất kỳ đâu trong ánh xạ phân tích cú pháp.

END_PARSE_MAP

Cú pháp: `END_PARSE_MAP(theClass)`

Trong đó, tham số `theClass` là tên lớp chủ của ánh xạ phân tích cú pháp.

Sử dụng macro `END_PARSE_MAP` để kết thúc định nghĩa ánh xạ phân tích cú pháp. Nó luôn đi kèm với macro `BEGIN_PARSE_MAP`.

Khi lập trình viên muốn bổ sung thêm một chức năng cho ISA thì sẽ phải khai báo thêm hàm thành phần xử lý trong lớp chính của chương trình; viết mã trình xử lý của hàm; khai báo hàm trong ánh xạ phân tích cú pháp. Thí dụ trong chương trình `COUNTER` để bổ sung thêm chức năng lấy thời gian hệ thống trên máy chủ và gửi kết quả lại dưới dạng trang HTML ta làm như sau:

- Khai báo thêm hàm thành phần `ClockHTML(CHttpRequest* pClt)` trong lớp `CCounterExtension`.
- Thêm nội dung hàm `ClockHTML` vào tệp `COUNTER.CPP`.
- Thêm dòng khai báo sau vào ánh xạ phân tích cú pháp:

`ON_PARSE_COMMAND(ClockHTML, CCounterExtension, ITS_EMPTY)`

12.1.2.5 Quản lý lỗi

MFC cung cấp khả năng thông báo lỗi khi chạy chương trình. MFC quản lý và gửi trả về các thông báo lỗi HTTP thích hợp cho máy trạm (thí dụ như yêu cầu sai). Bảng 12.4 liệt kê một số mã thông báo thường gặp.

Bảng 12.4 Một số thông báo thường gặp.

<i>Mã thông báo</i>	<i>Giải thích</i>
200	URL đã xác định, sự truyền tin tiếp sau đây
400	Không thể hiểu được yêu cầu
404	Không tìm thấy yêu cầu URL
405	Máy chủ không cung cấp phương pháp yêu cầu
500	Lỗi dịch vụ không xác định
503	Dịch vụ chưa sẵn sàng để dùng

Các mã thông báo trả về có thể tìm thấy trong tệp `AFXISAPI.H`. Các thông báo trả về của HTTP có thể chia thành các nhóm như trong bảng 12.5.

Bảng 12.5 Các nhóm thông báo của HTTP

Nhóm	Giải thích
200299	Thành công
300399	Thông tin
400499	Yêu cầu lỗi
500599	Lỗi dịch vụ

MFC cung cấp một phương pháp quản lý lỗi thông qua hàm chức năng `OnParseError` của lớp `CHttpServer`. Khung ứng dụng gọi hàm này để phân tích các lỗi. Khi lỗi được xác định, thông báo về nguyên nhân của lỗi được gửi về cho máy trạm của nó trong một trang HTML hay trong một thông báo lỗi do hàm `CHttpServerContext::WriteClient` gửi. Lập trình viên có thể nạp chồng hàm này theo yêu cầu của chương trình. Cú pháp của hàm `CHttpServer::OnParseError` như sau:

```
virtual BOOL OnParseError( CHttpServerContext* pCtxt, int nCause );
```

Trong đó, tham số `pCtxt` là con trỏ tới đối tượng `CHttpServerContext`, đối tượng chứa mã trạng thái của việc truyền tin hiện thời trong thành phần `dwHttpStatusCode` của cấu trúc `EXTENSION_CONTROL_BLOCK`. Những mã trạng thái bao gồm:

```
HTTP_STATUS_BAD_REQUEST
HTTP_STATUS_AUTH_REQUIRED
HTTP_STATUS_FORBIDDEN
HTTP_STATUS_NOT_FOUND
HTTP_STATUS_SERVER_ERROR
HTTP_STATUS_NOT_IMPLEMENTED
```

`nCause` chỉ ra nguyên nhân của lỗi, nó có thể là một trong các giá trị liệt kê trong bảng 12.6.

Bảng 12.6. Các nguyên nhân lỗi của trình mở rộng dịch vụ Internet

Giá trị kiểu enum	Mô tả
<code>callOK</code>	<code>OnParseError</code> đã quản lý lỗi
<code>callParamRequired</code>	Thiếu một tham số yêu cầu
<code>callBadParamCount</code>	Quá thiếu hoặc quá nhiều tham số
<code>callBadCommand</code>	Không tìm thấy tên lệnh
<code>callNoStackSpace</code>	Không cấp được bộ nhớ cho stack
<code>callNoStream</code>	Không đối tượng <code>CHtmlStream</code> nào sử dụng được
<code>callMissingQuote</code>	Tham số thiếu dấu nháy (")
<code>callMissingParams</code>	Không tham số nào sử dụng được
<code>callBadParam</code>	Tham số sai khuôn dạng

Kết quả trả về của hàm là một số khác 0 nếu phân tích thành công, ngược lại hàm trả lại giá trị 0.

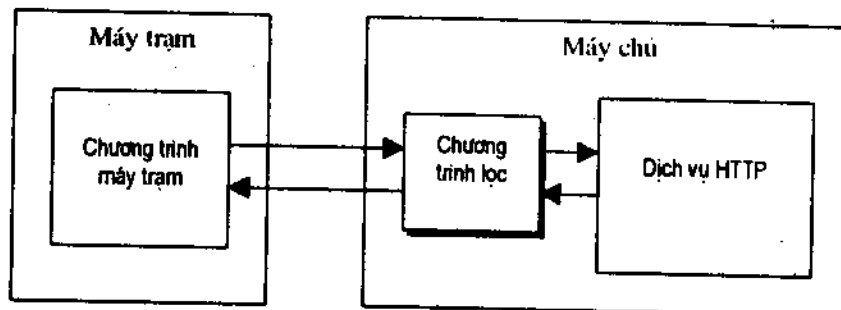
12.1.3 Chương trình lọc sử dụng ISAPI

Chương trình lọc là một chương trình DLL chạy trên máy chủ có dịch vụ HTTP trợ giúp ISAPI. Khi máy trạm gửi yêu cầu cho máy chủ, trên máy chủ sẽ xuất hiện các sự kiện thí dụ như: sự kiện đăng nhập, sự kiện ánh xạ URL, sự kiện đọc dữ liệu, sự kiện gửi dữ liệu v.v. Mỗi khi xuất hiện sự kiện, máy chủ sẽ gửi thông báo *xuất hiện sự kiện* và gọi các chương trình tương ứng để xử lý sự kiện đó. Một chương trình lọc sẽ đăng ký gắn với một số sự kiện nào đó, khi chúng xuất hiện thì chương trình lọc được gọi và lập trình viên có thể giám sát và thay đổi dữ liệu đến từ máy trạm hoặc dữ liệu gửi về cho máy trạm. Chương trình lọc ISAPI có thể được sử dụng để mở rộng dịch vụ Internet như: cung cấp mã khoá khách hàng, thực hiện nén dữ liệu, các phương pháp xác nhận bổ sung, ...v.v.

MS Developer Studio cung cấp công cụ *ISAPI Extension Wizard* giúp lập trình viên dễ dàng tạo ra một chương trình lọc. Chương 13 sẽ giới thiệu chi tiết về cách dùng công cụ *ISAPI Extension Wizard* để tạo ra chương trình lọc.

12.1.3.1 Chương trình lọc mở rộng máy chủ như thế nào?

Chương trình lọc được đặt giữa phần kết nối mạng với máy trạm và dịch vụ HTTP như trên hình 12.3. Chương trình lọc có thể tác động tới một số hành động phục vụ tùy thuộc vào việc đăng ký chương trình lọc gắn với các sự kiện nào. Chương trình lọc có thể được đặt để đọc dữ liệu thô từ máy trạm, xử lý các phần đầu của thông tin, quản lý việc truyền tin trên các cổng được bảo vệ sử dụng công nghệ truyền tin cá nhân (*Personal Communications Technology - PCT*) hoặc lớp socket an toàn (*Secure Sockets Layer - SSL*), hoặc quản lý các mức khác trong việc xử lý các yêu cầu HTTP.



Hình 12.3 Vị trí của chương trình lọc

Để dễ hiểu hơn về chương trình lọc chúng ta cùng nghiên cứu một chương trình đơn giản, chương trình lọc MFCUCASE.

12.1.3.2 Chương trình lọc MFCUCASE

Chương trình lọc MFCUCASE xem địa chỉ URL có chứa thư mục có tên “/UC/” (*Upper Case*). Nếu chương trình thấy dòng ký tự này trong địa chỉ URL, nó sẽ loại bỏ và tạo ra địa chỉ URL mới và chuyển đổi dữ liệu từ *URL site* thành chữ hoa trước khi gửi về cho máy trạm. Thí dụ, máy chủ có tên POPMAP và trên đó chứa một tệp có tên GIOITHIEU.HTM, từ trình duyệt Web trên máy trạm ta có thể gửi yêu cầu “http://popmap/gioithieu.htm” để đọc tệp GIOITHIEU.HTM. Khi chương trình MFCUCASE đã được cài đặt trên máy chủ, ta có thể gửi yêu cầu như sau “http://popmap/uc/gioithieu.htm” để đọc tệp GIOITHIEU.HTM, tuy nhiên lúc này tất cả các dòng dữ liệu của tệp GIOITHIEU.HTM đã được biến đổi thành chữ hoa. Chương trình lọc MFCUCASE được gắn với sự kiện ánh xạ URL và đọc dữ liệu thô, tức là chương trình sẽ được gọi khi xuất hiện các sự kiện này.

Chương trình 12.2 MFCUCASE

Tệp *mfcucase.def*

```
// Khai báo các hàm xuất (export) cho MFCUCASE.DLL
LIBRARY      mfcucase
DESCRIPTION  'Chương trình lọc sử dụng ISAPI của MFC'
EXPORTS
    HttpFilterProc
    GetFilterVersion
//-----
```

Tệp *resource.h*

```
// Tệp MFCUCASE.rc sẽ sử dụng resource.h
#define IDS_MFCUCASE_FILTER      101
#ifdef APSTUDIO_INVOKED
#ifdef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE  103
#define _APS_NEXT_COMMAND_VALUE  32768
#define _APS_NEXT_CONTROL_VALUE   103
#define _APS_NEXT_SYMED_VALUE    103
#endif
#endif
//-----
```

Tệp *resource.rc*

```
#include "resource.h"
#include "afxres.h"
// Phiên bản
```

```

VS_VERSION_INFO VERSIONINFO
FILEVERSION 1,0,0,1
PRODUCTVERSION 1.0.0.1
FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
FILEFLAGS 0x1L
#else
FILEFLAGS 0x0L
#endif
FILEOS 0x4L
FILETYPE 0x2L
FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "Chương trình mở rộng dịch vụ Internet MFCUCASE\0"
            VALUE "FileVersion", "1. 0. 0. 1\0"
            VALUE "InternalName", "MFCUCASE\0"
            VALUE "LegalCopyright", "Copyright © 1999\0"
            VALUE "OriginalFilename", "MFCUCASE.DLL\0"
            VALUE "ProductName", "Chương trình mở rộng dịch vụ Internet MFCUCASE\0"
            VALUE "ProductVersion", "1. 0. 0. 1\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END
// Bảng xâu ký tự
STRINGTABLE DISCARDABLE
BEGIN
    IDS_MFCUCASE_FILTER "Chương trình lọc MFCUCASE"
END
//-----

```

Tệp mfcucase.h

```
// Định nghĩa lớp lọc
class CUpCaseFilter : public CHttpFilter
{
public:
    CUpCaseFilter();
    ~CUpCaseFilter();

public:
    virtual BOOL GetFilterVersion(PHTTP_FILTER_VERSION pVer);
    virtual DWORD OnSendRawData(CHttpFilterContext* pCtxt,
        PHTTP_FILTER_RAW_DATA pRawData);
    virtual DWORD OnUrlMap(CHttpFilterContext* pCtxt,
        PHTTP_FILTER_URL_MAP pUrlMap);
};
//-----
```

Tệp mfcucase.cpp

```
// phát triển lớp lọc
// Chương trình có thể không cần dòng khai báo #ifdef _AFXDLL như dưới đây.
// Project cho phép xây dựng ứng dụng như chương trình MFC DLL thông thường
// được liên kết với thư viện tĩnh (sử dụng định nghĩa _AFXDLL)
// hoặc chương trình DLL không sử dụng các lớp MFC ngoài các lớp do ISAPI
// cung cấp (không cần định nghĩa _AFXDLL).
#ifdef _AFXDLL
#include <afx.h>
#include <afxwin.h>
#else
#include <afxisapi.h>
#include "mfcucase.h"
#include "resource.h"
// chỉ có duy nhất một đối tượng CUpCaseFilter
CUpCaseFilter theFilter;
// Cài đặt CUpCaseFilter
CUpCaseFilter::CUpCaseFilter()
{
}
CUpCaseFilter::~CUpCaseFilter()
{
}
```

```

// CUpCaseFilter object
BOOL CUpCaseFilter::GetFilterVersion(PHTTP_FILTER_VERSION pVer)
{
    // Gọi cách xử lý mặc định để khởi tạo
    CHttpFilter::GetFilterVersion(pVer);
    // Đặt các cờ chúng ta quan tâm
    pVer->dwFlags |= (SF_NOTIFY_NONSECURE_PORT |
                     SF_NOTIFY_URL_MAP | SF_NOTIFY_SEND_RAW_DATA);
    // Nạp xâu mô tả
    TCHAR sz[SF_MAX_FILTER_DESC_LEN+1];
    ISAPIVERIFY(!::LoadString(AfxGetResourceHandle(),
                             IDS_MFC!CASE_FILTER, sz, SF_MAX_FILTER_DESC_LEN));
    _tcscpy(pVer->lpzFilterDesc, sz);
    return TRUE;
}

DWORD CUpCaseFilter::OnSendRawData(CHttpFilterContext* pCtxt,
PHTTP_FILTER_RAW_DATA pRawData)
{
    LPTSTR    pstrIn;
    DWORD     cbBuffer;
    DWORD     cbTemp;
    // Nếu đã có dữ liệu, phân tích chúng
    if (pCtxt->m_pFC->pFilterContext != NULL)
    {
        // đặt con trỏ tới vùng dữ liệu
        pstrIn = (LPTSTR) pRawData->pvInData;
        cbBuffer = 0;
        cbTemp = 0;
        //đang đợi cho phần đầu?
        if (pCtxt->m_pFC->pFilterContext == (VOID *) 1)
        {
            // duyệt từ đầu đến cuối
            while (cbBuffer < pRawData->cbInData)
            {
                if (pstrIn[cbBuffer] == '\n' && pstrIn[cbBuffer+2] == '\n') {
                    cbBuffer += 3;
                    break;
                }
                cbBuffer++;
            }
        }
    }
}

```

```

// phần đầu xác định nội dung là trang HTML?
while (cbTemp < cbBuffer)
{
    if (pstrIn[cbTemp] == '/' && pstrIn[cbTemp+1] == 'h' &&
        pstrIn[cbTemp+2] == 't' && pstrIn[cbTemp+3] == 'm')
    {
        // đúng, đây là trang HTML, đặt cờ.
        pCtxt->m_pFC->pFilterContext = (VOID*) 2;
        break;
    }
    cbTemp++;
}
// nếu không phải là trang HTML - xóa cờ
if (cbTemp == cbBuffer)    pCtxt->m_pFC->pFilterContext = NULL;
}
// nếu có dữ liệu trong vùng đệm, đổi dữ liệu thành chữ hoa!
if (pCtxt->m_pFC->pFilterContext != NULL)    {
    while (cbBuffer < pRawData->cbInData) {
        if (isleadbyte(pstrIn[cbBuffer]))    cbBuffer += 2;
        else {
            pstrIn[cbBuffer] = _totupper(pstrIn[cbBuffer]);
            cbBuffer++;
        }
    }
}
}
return SF_STATUS_REQ_NEXT_NOTIFICATION;
}

DWORD CUpCaseFilter::OnUrlMap(CHttpFilterContext* pCtxt, PHTTP_FILTER_URL_MAP pUrlMap)
{
    LPTSTR pstrPhysPath = pUrlMap->pszPhysicalPath;
    pCtxt->m_pFC->pFilterContext = NULL;
    // nếu tiếp nhận URL, kiểm tra xem có chứa xâu "\UC\"
    while (*pstrPhysPath != '\0')
    {
        if (pstrPhysPath[0] == '\\' && (pstrPhysPath[1] == 'u' || pstrPhysPath[1] == 'U') &&
            (pstrPhysPath[2] == 'c' || pstrPhysPath[2] == 'C') && pstrPhysPath[3] == '\\')
        {
            // tìm thấy xâu "\UC", loại bỏ xâu "\UC\" để tìm ra đường dẫn đúng
            while (pstrPhysPath[3] != '\0') {

```

```

    pstrPhysPath[0] = pstrPhysPath[3];
    pstrPhysPath = _tcsinc(pstrPhysPath);
}
pstrPhysPath[0] = '\0';
// đặt cờ nhắc kiểm tra dữ liệu và đổi thành chữ hoa
pCtxt->m_pFC->pFilterContext = (VOID*) 1;
break;
}
pstrPhysPath = _tcsinc(pstrPhysPath);
}
return SF_STATUS_REQ_NEXT_NOTIFICATION;
}
// Nếu muốn xây dựng chương trình lọc không sử dụng các lớp MFC.
// cần phát triển phiên bản riêng AfxGetResourceHandle() để các lớp trong
// ISAPI của MFC có thể tìm thấy các tài nguyên mà nó cần.
// Nếu sử dụng các lớp MFC cùng với các lớp của ISAPI, không cần đoạn trình dưới đây
static HINSTANCE g_hInstance;
HINSTANCE AFXISAPI AfxGetResourceHandle()
{
    return g_hInstance;
}
BOOL WINAPI DllMain(HINSTANCE hInst, ULONG ulReason, LPVOID lpReserved)
{
    if (ulReason == DLL_PROCESS_ATTACH) g_hInstance = hInst;
    return TRUE;
}
//-----

```

Sau khi soạn thảo, dịch và liên kết chương trình trên ta có tệp MFCUCASE.DLL. Để chạy được chúng ta phải thực hiện các bước sau đây:

- Đăng ký với máy chủ thông qua chương trình REGEDIT.EXE: chèn tên chương trình MFCUCASE.DLL (bao gồm cả đường dẫn) vào dòng khai báo trong HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/W3SVC/Parameters/Filter DLLs. Nếu có nhiều chương trình lọc được khai báo, các khai báo phân cách nhau bằng dấu phẩy ','.
- Khởi động lại dịch vụ Web (bằng cách dừng dịch vụ Web và khởi động lại hoặc khởi động lại máy chủ) để nạp chương trình MFCUCASE.DLL.

Cũng giống như chương trình dịch vụ Internet, một chương trình lọc bao giờ cũng có một lớp chính là lớp phái sinh từ lớp CHttpFilter. Lớp chính của chương trình MFCUCASE là lớp

CUpCaseFilter. Trong lớp chính sẽ mô tả các hàm thành phần nạp chồng của lớp CHttpFilter, chúng được viết lại để khai báo chương trình lọc gắn với sự kiện nào và xử lý như thế nào khi sự kiện đó xuất hiện. Trong chương trình MFCUCASE các hàm thành phần đó là: GetFilterVersion, OnSendRawData và OnUrlMap. Trước hết chúng ta tìm hiểu các lớp trong ISAPI của MFC được sử dụng như thế nào để xây dựng chương trình lọc - lớp CHttpFilter và CHttpFilterContext.

12.1.3.3 Lớp CHttpFilter

Lớp CHttpFilter và lớp CHttpFilterContext tạo và quản lý việc lọc. Đối tượng CHttpFilter được tạo lập khi chương trình lọc được nạp vào máy, đối tượng CHttpFilterContext được tạo lập mỗi khi chương trình lọc tiếp nhận yêu cầu. Mỗi khi chương trình lọc được nạp, nó thông báo cho máy chủ về các sự kiện mà nó sẽ liên kết. Sau đó, mỗi khi các sự kiện được chọn trước xuất hiện, chương trình lọc được gọi để xử lý sự kiện đó. Chú ý rằng khi chương trình lọc bắt đầu xử lý yêu cầu thì nó sẽ nhận dữ liệu mà không quan tâm đến các yêu cầu từ tệp, ứng dụng CGI hay ứng dụng ISAPI. Các thành phần của lớp CHttpFilter được mô tả như sau đây:

Cấu tử

CHttpFilter Tạo lập đối tượng CHttpFilter

Thuộc tính

GetFilterVersion Lấy số hiệu phiên bản của chương trình lọc sau khi đối tượng CHttpFilter được tạo lập

Các hàm nạp chồng

OnReadRawData Cho phép chương trình đọc dữ liệu thô. Dữ liệu trả về bao gồm hai phần: khai báo và nội dung

OnPreprocHeaders Thông báo cho máy trạm biết máy chủ đã thực hiện tiến xử lý phần đầu thông tin

OnAuthentication Xác nhận máy trạm

OnUrlMap Thông báo cho máy trạm khi máy chủ đã liên kết URL logic với đường dẫn vật lý

OnSendRawData Gửi dữ liệu thô từ máy chủ cho máy trạm

OnLog Đăng nhập thông tin vào tệp dịch vụ

OnEndOfNetSession Thông báo cho máy trạm đã kết thúc phiên giao dịch

Hàm xử lý

HttpFilterProc Trả lại thông báo chỉ ra sự kiện nào đi qua chương trình lọc và đã được xử lý

Hai hàm thành phần GetFilterVersion và HttpFilterProc của lớp CHttpFilter được coi như các điểm vào của chương trình lọc. Khi máy chủ nạp hoặc gọi chương trình lọc, nó gọi ngay hai hàm thành phần này để xác định chương trình lọc sẽ gắn với các sự kiện nào và xử lý như thế nào.

Hàm GetFilterVersion

Trước khi sử dụng, chương trình lọc phải được đăng ký với máy chủ thông qua chương trình REGEDIT.EXE bằng việc chèn chương trình lọc (bao gồm cả đường dẫn) vào dòng khai báo trong HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/W3SVC/Parameters/Filter DLLs. Khi máy chủ bắt đầu khởi động, nó đọc các giá trị này và nạp các chương trình lọc (các tệp DLL)

đã được khai báo. Sau đó nó gọi hàm `CHttpFilter::GetFilterVersion` để thi hành ba nhiệm vụ:

- Trao đổi thông tin phiên bản: hàm `GetFilterVersion` thực hiện so sánh số hiệu phiên bản của chương trình lọc với các số liệu phiên bản của máy chủ.
- Xác định các sự kiện yêu cầu: đăng ký chương trình lọc gắn với các sự kiện nào. Nếu chúng ta đăng ký chương trình lọc gắn với các sự kiện không liên quan, chương trình lọc có thể không thực hiện như mong muốn.
- Xác định mức ưu tiên: mức ưu tiên của chương trình lọc được xác định bởi thành phần `dwFlags` của cấu trúc `HTTP_FILTER_VERSION` (cấu trúc chứa các thông tin về chương trình lọc) trong hàm `GetFilterVersion`. Sau khi hàm `GetFilterVersion` thực hiện, mỗi khi máy chủ xử lý một sự kiện, nó sẽ gọi các chương trình lọc đã đăng ký gắn với sự kiện này. Thứ tự gọi các chương trình lọc phụ thuộc vào mức ưu tiên.

Hàm `GetFilterProc`

Khi một sự kiện xuất hiện, máy chủ thông báo cho các chương trình lọc gắn với sự kiện này bằng việc gọi hàm `GetFilterProc`. Xử lý mặc định của hàm `GetFilterProc` là xác định máy chủ thông báo cho mình sự kiện nào và gọi hàm thành phần của lớp `CHttpFilter` dành cho việc xử lý sự kiện này. Lập trình viên có thể viết lại hàm này để xác định việc xử lý cụ thể cho từng sự kiện.

Chương trình lọc thực hiện xử lý dữ liệu đến/đi của máy trạm bằng việc gọi các hàm nạp chồng liên quan tới các sự kiện. Viết lại các hàm nạp chồng để thực hiện việc xử lý dữ liệu theo mong muốn. Thí dụ, chúng ta có thể tạo chương trình lọc để mã hoá hay nén dữ liệu bằng việc viết lại hàm nạp chồng `OnSendRawData`.

12.1.3.4 Lớp `CHttpFilterContext`

Lớp `CHttpFilterContext` cung cấp các công cụ cho đối tượng `CHttpFilter` để xử lý dữ liệu gửi cho chương trình lọc. Khi chương trình lọc tiếp nhận một yêu cầu, nó tạo lập một đối tượng `CHttpFilterContext`. Khi chương trình lọc xử lý các yêu cầu, nó sử dụng các hàm thành phần của lớp `CHttpFilterContext` để thực hiện các nhiệm vụ như đọc, ghi dữ liệu, v.v. Cũng giống như chương trình mở rộng dịch vụ Internet, chỉ có một đối tượng `CHttpFilter` tồn tại trong một chương trình lọc, nhưng đối tượng này có thể tạo và quản lý nhiều đối tượng `CHttpFilterContext`, mỗi đối tượng `CHttpFilterContext` phục vụ cho một yêu cầu HTTP. Thiết kế này cho phép một chương trình lọc cùng một lúc có thể xử lý nhiều yêu cầu của các máy trạm.

Chương trình lọc nhiều khi cần phải trao đổi thông tin với máy trạm, thí dụ gửi lại ngay một thông báo lỗi cho máy trạm, khi đó nó sẽ gọi hàm thành phần `WriteClient` để thực hiện công việc này. Dưới đây là các thành phần của lớp `CHttpFilterContext`:

Các thành phần dữ liệu

m_pFC Con trỏ tới cấu trúc HTTP_FILTER_CONTEXT - cấu trúc chứa thông tin về yêu cầu hiện thời

Cấu trúc

CHttpFilterContext Tạo lập đối tượng CHttpFilterContext

Các hàm thuộc tính

GetServerVariable Sao chép các thông tin liên quan tới việc liên lạc HTTP hoặc với bản thân máy chủ vào một vùng đệm

AddResponseHeaders Bổ sung phần đầu cho trả lời HTTP

WriteClient Gửi ngay lại dữ liệu cho máy trạm

AllocMem Cấp bộ nhớ

ServerSupportFunction Cung cấp thông tin ISA tổng thể cho máy trạm

12.1.3.5 Các bước xây dựng một chương trình lọc sử dụng ISAPI

Một chương trình lọc sử dụng ISAPI có thể được xây dựng dễ dàng nhờ các công cụ của Visual C++. Hãy xem chi tiết cách sử dụng Visual C++ trong chương 13, tuy nhiên dưới đây là các bước chính để xây dựng chương trình lọc:

- Tạo project cho chương trình lọc: sử dụng công cụ *ISAPI Extension Wizard*.
- Bổ sung các xử lý:
 - ♦ Xác định chương trình lọc sẽ gắn với những sự kiện nào và mức ưu tiên của chương trình thông qua việc gán các giá trị thích hợp cho biến dwFlags của cấu trúc HTTP_FILTER_VERSION trong hàm GetFilterVersion của lớp CHttpFilter. Các giá trị này có thể đã được xác định bởi ISAPI Extension Wizard. Thí dụ:

```
pVer->dwFlags |= (SF_NOTIFY_NONSECURE_PORT |
```

```
SF_NOTIFY_URL_MAP | SF_NOTIFY_SEND_RAW_DATA);
```

Bảng 12.7 liệt kê các giá trị có thể sử dụng để khai báo mức ưu tiên của chương trình lọc, chương trình lọc gắn với sự kiện nào.

Bảng 12.7 Các giá trị sử dụng để khai báo cho chương trình lọc

<i>Giá trị</i>	<i>Ý nghĩa</i>
SF_NOTIFY_ORDER_DEFAULT	Chương trình lọc có mức ưu tiên mặc định.
SF_NOTIFY_ORDER_LOW	Chương trình lọc có mức ưu tiên thấp.
SF_NOTIFY_ORDER_MEDIUM	Chương trình lọc có mức ưu tiên trung bình
SF_NOTIFY_ORDER_HIGH	Chương trình lọc có mức ưu tiên cao.
SF_NOTIFY_SECURE_PORT	Thông báo chương trình lọc chuyển dữ liệu qua cổng được bảo vệ
SF_NOTIFY_NONSECURE_PORT	Thông báo chương trình lọc chuyển dữ liệu qua cổng không được bảo vệ
SF_NOTIFY_READ_RAW_DATA	Cho phép chương trình lọc đọc dữ liệu thô.
SF_NOTIFY_PREPROC_HEADERS	Máy chủ thực hiện tiến xử lý phần đầu dữ liệu
SF_NOTIFY_AUTHENTICATION	Máy chủ xác nhận máy trạm

SF_NOTIFY_URL_MAP	Máy chủ ánh xạ địa chỉ URL logic vào đường dẫn vật lý
SF_NOTIFY_SEND_RAW_DATA	Máy chủ gửi lại dữ liệu thô cho máy trạm
SF_NOTIFY_LOG	Máy chủ ghi thông tin vào dịch vụ đăng nhập
SF_NOTIFY_END_OF_NET_SESSION	Đang kết thúc phiên giao dịch với máy trạm

- ♦ Lập trình viên có thể bổ sung thêm sự kiện mới:
 - Bổ sung khai báo sự kiện mới vào biến dwFlags
 - Bổ sung khai báo hàm lọc tương đương với sự kiện vào tệp .h
 - Bổ sung định nghĩa hàm lọc vào tệp .cpp

Thí dụ: Muốn chương trình MFCUCASE gắn thêm với sự kiện xác nhận máy trạm khi đó:

```
pVer->dwFlags |= (SF_NOTIFY_NONSECURE_PORT | SF_NOTIFY_URL_MAP |
SF_NOTIFY_SEND_RAW_DATA | SF_NOTIFY_AUTHENTICATION);
```

Trong định nghĩa lớp CUpCaseFilter (MFCUCASE.H) bổ sung khai báo:

```
virtual DWORD OnAuthentication(CHttpFilterContext* pCtxt,
PHTTP_FILTER_AUTHENT pAuthent);
```

Trong tệp MFCUCASE.CPP bổ sung thêm hàm:

```
DWORD CFilter1Filter::OnAuthentication(CHttpFilterContext* pCtxt,
PHTTP_FILTER_AUTHENT pAuthent)
{
    // bổ sung các lệnh xử lý ở đây
    .....
    // trả về mã trạng thái thích hợp
    return SF_STATUS_REQ_NEXT_NOTIFICATION;
}
```

Chúng ta cũng có thể loại bớt các sự kiện và hàm không cần thiết.

- ♦ Viết các xử lý riêng cho từng hàm nạp chồng. Thí dụ trong chương trình MFCUCASE hàm OnUrlMap được viết lại như sau để kiểm tra xem yêu cầu URL có chứa xâu "/uc/" hoặc xâu "/UC/" không, nếu có thì loại bớt đi:

```
DWORD CUpCaseFilter::OnUrlMap(CHttpFilterContext* pCtxt,
PHTTP_FILTER_URL_MAP pUrlMap)
{
    LPTSTR pstrPhysPath = pUrlMap->pszPhysicalPath;
    pCtxt->m_pFC->pFilterContext = NULL;
    // Khi nhận được xâu URL.
    // kiểm tra xem xâu này có chứa xâu "\UC\" không.
    while (*pstrPhysPath != '\0') {
```

```

if (pstrPhysPath[0] == '\\' &&
    (pstrPhysPath[1] == 'u' || pstrPhysPath[1] == 'U') &&
    (pstrPhysPath[2] == 'c' || pstrPhysPath[2] == 'C') &&
    pstrPhysPath[3] == '\\')
{
    // tìm thấy xâu "\\UC\\"; loại bỏ xâu này
    while (pstrPhysPath[3] != '\\0')
        pstrPhysPath[0] = pstrPhysPath[3];
    pstrPhysPath = _tcsinc(pstrPhysPath);
}
pstrPhysPath[0] = '\\0';
// đặt lại cờ xác định sẽ chuyển đổi
// các xâu dữ liệu về chữ hoa
pCtxt->m_pFC->pFilterContext = (VOID*) 1;
break;
}
pstrPhysPath = _tcsinc(pstrPhysPath);
}
return SF_STATUS_REQ_NEXT_NOTIFICATION;
}

```

- Bằng chương trình REGEDIT.EXE hãy đăng ký chương trình lọc với máy chủ thông qua việc bổ sung đường dẫn và tên tệp chương trình lọc:

HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/W3SVC/Parameters/Filter DLL

Bảng 12.8 là một vài khuyến cáo cách sử dụng sự kiện hay hành động cho các chức năng của chương trình lọc.

Bảng 12.8 Chương trình lọc và sự kiện

<i>Chương trình lọc</i>	<i>Sự kiện/hành động</i>
Đếm số lượng người thăm nhập máy chủ	SF_NOTIFY_LOG
Thực hiện mã hoá khách hàng	SF_NOTIFY_READ_RAW_DATA, SF_NOTIFY_WRITE_RAW_DATA
Thực hiện nén dữ liệu khách hàng	SF_NOTIFY_READ_RAW_DATA, SF_NOTIFY_WRITE_RAW_DATA
Đọc dữ liệu thô	SF_NOTIFY_READ_RAW_DATA
Xử lý thông tin header	Gọi hàm GetServerVariable
Xác nhận người sử dụng	Tạo bộ lọc với yêu cầu khai báo mức ưu tiên cao

12.2. ỨNG DỤNG INTERNET SỬ DỤNG WININET

Ứng dụng Internet trên máy trạm, hay còn gọi là ứng dụng Internet, là chương trình có khả năng thăm nhập dữ liệu tài nguyên mạng trên máy chủ sử dụng giao thức FTP, HTTP hay Gopher. Một

ứng dụng Internet có khả năng thăm nhập máy chủ thông qua mạng Internet hay Intranet để khai thác dữ liệu như bản đồ thời tiết, giá cổ phiếu, ...v.v.

Từ phiên bản 4.2 trở đi, MFC bổ sung mở rộng Internet *Win32 Internet Extension* hay *WinInet* để lập trình viên có thể tạo ứng dụng Internet. Chúng ta có thể sử dụng WinInet để xây dựng chương trình Internet ở mức lập trình cao mà không cần phải làm việc trực tiếp với *WinSock* hay TCP/IP. WinInet cung cấp các hàm cho cả ba giao thức FTP, HTTP và Gopher với các giao diện lập trình Win32 quen thuộc. Lập trình viên có thể sử dụng WinInet viết một chương trình Internet bằng cách gọi trực tiếp các hàm của Win32 hay sử dụng các lớp của WinInet. Trong chương này chúng ta sẽ xem xét các lớp của WinInet trợ giúp xây dựng ứng dụng Internet. Với WinInet chúng ta có thể thực hiện các công việc như sau:

- Lấy về các trang HTML. HTTP là giao thức được sử dụng để truyền các trang HTML từ máy chủ về trình duyệt trên máy trạm.
- Gửi yêu cầu FTP để nạp lên hay lấy về các tệp hay đọc danh sách các thư mục.
- Sử dụng thực đơn hệ thống gopher để thăm nhập các tài nguyên trên Internet

Chương trình Internet sử dụng các lớp của WinInet có thể đọc các thông tin từ dịch vụ HTTP, FTP và Gopher như đọc các tệp trên đĩa cứng.

Bảng 12.9 Các lớp của WinInet

Lớp	Chức năng
CInternetSession	Khởi tạo phiên giao dịch Internet: có thể dùng để mô tả liên lạc với máy chủ uỷ quyền (proxy server)
CInternetConnection	Quản lý liên lạc của máy trạm với máy chủ Internet. Là lớp cơ sở của các lớp CFtpConnection, CGopherConnection và CHttpConnection
CFtpConnection	Quản lý liên lạc với dịch vụ FTP trên máy chủ và cho phép thao tác trực tiếp với tệp và thư mục trên máy chủ.
CGopherConnection	Quản lý liên lạc với máy chủ Gopher
CHttpConnection	Quản lý liên lạc với dịch vụ HTTP trên máy chủ
CInternetFile	Là lớp cơ sở của lớp CGopherFile và CHttpFile. Lớp CInternetFile cùng với các lớp dẫn xuất cho phép thăm nhập các tệp trên hệ thống từ xa sử dụng các giao thức Internet.
CGopherFile	Cung cấp các hàm để tìm và đọc tệp trên máy chủ Gopher
CHttpFile	Cung cấp các hàm để gửi yêu cầu và đọc các tệp trên máy chủ HTTP
CFileFind	Thực hiện tìm kiếm các tệp ngay trong máy. Là lớp cơ sở của các lớp CFtpFileFind và CGopherFileFind.
CFtpFileFind	Trợ giúp việc tìm kiếm các tệp trên máy chủ FTP
CGopherFileFind	Trợ giúp việc tìm kiếm các tệp trên máy chủ Gopher
CGopherLocator	Nhận "locator" của Gopher trên máy chủ Gopher, xác định kiểu locator và đưa nó vào trạng thái sẵn sàng cho CGopherFileFind
CInternetException	Là đối tượng để mô tả điều kiện ngoại lệ liên quan tới thao tác Internet. Lớp này chứa hai thành phần dữ liệu chung: thành phần thứ nhất để nắm giữ các lỗi liên quan với trường hợp ngoại lệ; thành phần thứ hai để nắm giữ ngữ cảnh của chương trình Internet liên quan đến lỗi.

Bảng 12.9 liệt kê các lớp của *WinInet* để xây dựng chương trình Internet. Các lớp này được khai báo trong tệp *AFXINET.H*, riêng lớp *CFileFind* được khai báo trong *AFX.H*.

12.2.1 Xây dựng chương trình Internet bằng MFC WinInet

Điểm cơ bản của tất cả các chương trình Internet là giao dịch Internet (*Internet session*). MFC xây dựng giao dịch Internet như là một đối tượng của lớp *CInternetSession*. Sử dụng lớp này, lập trình viên có thể tạo ra một hoặc một vài giao dịch Internet đồng thời.

Để trao đổi thông tin với máy chủ, ta phải tạo lập các đối tượng *CInternetConnection* và *CInternetSession*. Lập trình viên có thể tạo một *CInternetConnection* bằng việc gọi hàm *CInternetSession::GetFtpConnection*, *CInternetSession::GetHttpConnection* hay *CInternetSession::GetGopherConnection*. Mỗi lời gọi này chỉ rõ kiểu giao thức sử dụng. Những lời gọi này không phải là để mở một tệp trên máy chủ để đọc hoặc ghi. Nếu lập trình viên muốn đọc hoặc ghi dữ liệu thì phải mở tệp ở một bước riêng biệt.

Đối với phần lớn các giao dịch Internet, đối tượng *CInternetSession* cùng làm việc với đối tượng *CInternetFile* theo các qui định sau:

- Để có một giao dịch Internet, lập trình viên phải tạo một phiên bản của đối tượng *CInternetSession*.
- Nếu giao dịch Internet để đọc hoặc ghi dữ liệu, lập trình viên phải tạo một phiên bản của đối tượng *CInternetFile* (hoặc lớp phái sinh từ nó là *CHttpFile* hay *CGopherFile*). Phương pháp dễ nhất để đọc dữ liệu là gọi hàm *CInternetSession::OpenURL*. Hàm này phân tích cú pháp xâu URL do lập trình viên cung cấp, mở liên lạc với máy chủ (xác định trong xâu URL) và trả về một đối tượng *CInternetFile* chỉ đọc. *CInternetSession::OpenURL* không đòi hỏi xác định rõ một kiểu giao thức nào, cách gọi trên làm việc với cả xâu URL của FTP, HTTP hay Gopher. *CInternetSession::OpenURL* thậm chí làm việc với cả các tệp ngay trên máy trạm (khi đó nó trả về đối tượng *CStdioFile* thay cho đối tượng *CInternetFile*).
- Nếu giao dịch Internet không phải để đọc hay ghi dữ liệu mà để thực hiện các nhiệm vụ khác, thí dụ như xóa tệp trong một thư mục FTP, thì lập trình viên không cần phải tạo một phiên bản *CInternetFile*.

Có hai phương pháp để tạo đối tượng *CInternetFile*, đó là:

- Nếu sử dụng *CInternetSession::OpenURL* để tạo kết nối với máy chủ, thì kết quả hàm *OpenURL* sẽ trả lại một đối tượng *CInternetFile*.
- Nếu sử dụng các hàm thành phần *CInternetSession::GetFtpConnection* (cho giao thức FTP), *GetGopherConnection* (cho giao thức Gopher) hay *GetHttpConnection* (cho giao

thức HTTP) để tạo lập kết nối với máy chủ thì lập trình viên phải gọi các hàm tương ứng sau đây:

CFTPConnection::OpenFile,

CGopherConnection::OpenFile,

CHttpConnection::OpenRequest,

kết quả trả về là đối tượng CInternetFile, CGopherFile hay CHttpFile tương ứng.

Các bước phát triển tiếp theo của chương trình Internet phụ thuộc vào việc lập trình viên tạo chương trình dựa trên cơ sở OpenURL hay sử dụng giao thức FTP, HTTP hay Gopher.

12.2.2 Các bước xây dựng một chương trình Internet

Các bước xây dựng một chương trình Internet cơ bản

Bảng 12.10 thể hiện các bước để xây dựng một chương trình Internet cơ bản.

Bảng 12.10 Các bước xây dựng chương trình Internet

Mục đích	Hành động thực hiện	Kết quả
Bắt đầu phiên giao dịch Internet	Tạo một đối tượng CInternetSession	Khởi tạo WinInet và kết nối với máy chủ
Đọc hay tạo lựa chọn yêu cầu Internet. Thí dụ: đặt giới hạn hết thời gian (time-out) hay số lượng lần làm lại.	Sử dụng CInternetSession::QueryOption hay CInternetSession::SetOption	Trả về giá trị FALSE nếu không thực hiện thành công
Tạo một hàm callback để giám sát trạng thái của giao dịch	Sử dụng CInternetSession::EnableStatusCallback	Tạo lập callback cho CInternetSession::EnableStatusCallback. Viết lại hàm OnStatusCallback để tạo thủ tục callback theo mong muốn.
Nói với máy chủ Internet, máy chủ intranet hay các tệp ngay trong máy trạm	Sử dụng CInternetSession::OpenURL	Phân tích xâu URL và mở liên lạc với máy chủ chỉ định. Trả lại một đối tượng CStdioFile hoặc CInternetFile. Thông qua đối tượng này để thăm nhập dữ liệu trên máy chủ hoặc tệp.
Đọc dữ liệu từ tệp	Sử dụng CInternetFile::Read	Đọc dữ liệu (theo số lượng byte chỉ định) vào một vùng đệm
Quản lý các trường hợp ngoại lệ	Sử dụng lớp CInternetException	Quản lý tất cả các trường hợp ngoại lệ của Internet
Kết thúc phiên giao dịch Internet	Loại bỏ đối tượng CInternetSession	Tự động xóa các quản lý tệp và các kết nối

Đoạn chương trình dưới đây giới thiệu cách tạo ra một trình duyệt Web đơn giản theo các bước trình bày trên:

```
#include <afxinet.h>
//giả sử các tên URL đã được khởi tạo
//Bắt đầu giao dịch internet
```

```

CInternetSession session("My Session");
CStdioFile* pFile = NULL;
//sử dụng một URL và hiển thị một trang Web
while (lpzURL = DisplayPage(...))
{
    pFile = session.OpenURL(lpzURL); //mở tệp
    while (pFile->Read(szBuff,1024) > 0)
    {
        //đọc tệp...
    }
    delete pFile; //đóng tệp
}
//kết thúc giao dịch Internet
session.Close();

```

Các bước xây dựng một chương trình FTP cơ bản

Bảng 12.11 thể hiện các bước để xây dựng một chương trình FTP trên máy trạm.

Bảng 12.11 Các bước xây dựng chương trình FTP

<i>Mục đích</i>	<i>Hành động thực hiện</i>	<i>Kết quả</i>
Bắt đầu phiên giao dịch FTP	Tạo đối tượng CInternetSession	Khởi tạo WinInet và kết nối với máy chủ
Kết nối với máy chủ FTP	Sử dụng CInternetSession::GetFtpConnection	Trả về đối tượng CFTPConnection
Chuyển tới thư mục FTP mới trên máy chủ	Sử dụng CInternetSession::SetCurrentDirectory	Thay đổi thư mục hiện đang được kết nối trên máy chủ
Tìm tệp đầu tiên trong thư mục FTP	Sử dụng CFTPFileFind::FindFile	Tìm tệp đầu tiên. Trả về giá trị FALSE nếu không tìm thấy tệp
Tìm tệp tiếp theo trong thư mục FTP	Sử dụng CFTPFileFind::FindNextFile	Tìm tệp tiếp theo. Trả về giá trị FALSE nếu không tìm thấy tệp
Nếu tìm thấy tệp cần thiết, mở tệp để đọc hay ghi dữ liệu	Sử dụng CFTPConnection::OpenFile, sử dụng tên tệp do FindFile hay FindNextFile trả về	Mở tệp trên máy chủ để đọc hay ghi dữ liệu. Kết quả trả về là đối tượng CInternetFile
Đọc dữ liệu từ tệp hoặc ghi dữ liệu vào tệp	Sử dụng CInternetFile::Read hoặc CInternetFile::Write	Đọc dữ liệu từ tệp vào vùng đệm hoặc ghi dữ liệu từ vùng đệm vào tệp
Quản lý các trường hợp ngoại lệ	Sử dụng lớp CInternetException	Quản lý tất cả các kiểu ngoại lệ của Internet
Kết thúc phiên giao dịch FTP	Loại bỏ đối tượng CInternetSession	Tự động loại bỏ các quản lý tệp và các kết nối

Các bước xây dựng một chương trình FTP cơ bản để xóa tệp

Bảng 12.12 thể hiện các bước để xây dựng một chương trình FTP cơ bản để xóa tệp.

Bảng 12.12 Các bước xây dựng chương trình FTP để xóa tệp

<i>Mục đích</i>	<i>Hành động thực hiện</i>	<i>Kết quả</i>
Bắt đầu phiên giao dịch FTP	Tạo đối tượng CInternetSession	Khởi tạo Wininet và kết nối với máy chủ
Kết nối với máy chủ FTP	Sử dụng CInternetSession::GetFtpConnection	Trả về đối tượng CFtpConnection
Kiểm tra và chắc chắn đã vào đúng thư mục trên máy chủ FTP	Sử dụng CFtpConnection::GetCurrentDirectory hoặc CFtpConnection::GetCurrentDirectoryAsURL	Trả về tên hoặc URL của thư mục hiện đang liên kết trên máy chủ
Chuyển tới thư mục FTP mới trên máy chủ	Sử dụng CInternetSession::SetCurentDirectory	Thay đổi thư mục hiện đang được kết nối trên máy chủ
Tìm tệp đầu tiên trong thư mục FTP	Sử dụng CFtpFileFind::FindFile	Tìm tệp đầu tiên. Trả về giá trị FALSE nếu không tìm thấy tệp
Tìm tệp tiếp theo trong thư mục FTP	Sử dụng CFtpFileFind::FindNextFile	Tìm tệp tiếp theo. Trả về giá trị FALSE nếu không tìm thấy tệp
Nếu tìm thấy tệp cần thiết, xóa tệp đó	Sử dụng CFtpConnection::OpenFile, sử dụng tên tệp do FindFile hay FindNextFile trả về	Xóa tệp trên máy chủ
Quản lý các trường hợp ngoại lệ	Sử dụng lớp CInternetException	Quản lý tất cả các kiểu ngoại lệ của Internet
Kết thúc phiên giao dịch FTP	Loại bỏ đối tượng CInternetSession	Tự động loại bỏ các quản lý tệp và các kết nối

Các bước xây dựng một chương trình Gopher cơ bản

Bảng 12.13 thể hiện các bước để xây dựng một chương trình Gopher cơ bản.

Bảng 12.13 Các bước để xây dựng một chương trình Gopher

<i>Mục đích</i>	<i>Hành động thực hiện</i>	<i>Kết quả</i>
Bắt đầu phiên giao dịch Gopher	Tạo đối tượng CInternetSession	Khởi tạo Wininet và kết nối với máy chủ
Kết nối với máy chủ gopher	Sử dụng CInternetSession::GetGopherConnection	Trả về đối tượng CGopherConnection
Tìm tài nguyên đầu tiên trong gopher	Sử dụng CGopherFileFind::FindFile	Tìm tệp đầu tiên. Trả về giá trị FALSE nếu không tìm thấy tệp
Tìm tài nguyên tiếp theo trong thư mục gopher	Sử dụng CGopherFileFind::FindNextFile	Tìm tệp tiếp theo. Trả về giá trị FALSE nếu không tìm thấy tệp
Mở tệp tìm thấy bởi FindFile hay FindNextFile để đọc dữ liệu	Lấy gopher locator sử dụng CGopherFileFind::GetLocator. Sử dụng CGopherConnection::OpenFile	Mở tệp xác định bởi locator. Kết quả trả về là một đối tượng CGopherFile
Mở tệp sử dụng gopher locators do lập trình viên cung cấp	Sử dụng CGopherConnection::CreateLocator. Sử dụng CGopherConnection::OpenFile	Mở tệp xác định bởi locator. Kết quả trả về là một đối tượng CGopherFile
Đọc tệp	Sử dụng CGopherFile	Đọc dữ liệu từ tệp vào buffer
Quản lý các trường hợp ngoại lệ	Sử dụng lớp CInternetException	Quản lý tất cả các kiểu ngoại lệ của Internet
Kết thúc phiên giao dịch Gopher	Loại bỏ đối tượng CInternetSession	Tự động loại bỏ các quản lý tệp và các kết nối

Đoạn chương trình dưới đây giới thiệu cách khai thác thư mục Gopher.

```
#include <afxinet.h>
//Bắt đầu phiên giao dịch
CInternetSession session("My Gopher Session");
CGopherConnection* pConn = NULL;
CGopherFileFind* pFile;
//Kết nối với máy chủ
pConn = session.GetGopherConnection("gopher.yoursite.com");
pFile = new CGopherFileFind(pConn);
BOOL bFound = pFile->FindFile(lpszFileToFind);
while (bFound)
{
    bFound = pFile->FindNextFile();
    //khai thác các thuộc tính của tệp
}
delete pFile; //giải phóng pFile
delete pConn; //Kết thúc kết nối
session.Close(); //Kết thúc phiên giao dịch
```

Các bước xây dựng một chương trình HTTP cơ bản

Bảng 12.14 thể hiện các bước để xây dựng một chương trình HTTP cơ bản.

Bảng 12.14 Các bước xây dựng một chương trình HTTP

Mục đích	Hành động thực hiện	Kết quả
Bắt đầu phiên giao dịch HTTP	Tạo đối tượng CInternetSession	Khởi tạo Wininet và kết nối với máy chủ
Kết nối với máy chủ HTTP	Sử dụng CInternetSession::GetHttpConnection	Trả về đối tượng CHttpConnection
Mở một yêu cầu HTTP	Sử dụng CHttpConnection::OpenRequest	Kết quả trả về là một đối tượng CHttpFile
Gửi một yêu cầu HTTP	Sử dụng CHttpFile::AddRequestHeaders và CHttpFile::SendRequest	Tìm tệp tiếp theo. Trả về giá trị FALSE nếu không tìm thấy tệp
Mở tệp tìm thấy bởi FindFile hay FindNextFile để đọc dữ liệu	Lấy gopher locator sử dụng CGopherFileFind::GetLocator. Sử dụng CGopherConnection::OpenFile	Mở tệp xác định bởi locator. Kết quả trả về là một đối tượng CGopherFile
Đọc tệp	Sử dụng CHttpFile	Đọc dữ liệu dữ liệu từ tệp vào buffer
Quản lý các trường hợp ngoại lệ	Sử dụng lớp CInternetException	Quản lý tất cả các kiểu ngoại lệ của Internet
Kết thúc phiên giao dịch Gopher	Loại bỏ đối tượng CInternetSession	Tự động loại bỏ các quản lý tệp và các kết nối

Đoạn chương trình dưới đây giới thiệu cách khai nạp về một trang Web.

```

#include <afxinet.h>
// giả sử máy chủ, cổng và các tên URL đã được khởi tạo
// Bắt đầu giao dịch
CInternetSession session("My Session");
CHttpConnection* pServer = NULL;
CHttpFile* pFile = NULL;
try
{
    CString strServerName;
    INTERNET_PORT nPort;
    pServer = session.GetHttpConnection(strServerName, nPort);
    pFile = pServer->OpenRequest(CHttpConnection::HTTP_VERB_GET, strObject);
    pFile->AddRequestHeaders(szHeaders);
    pFile->SendRequest();
    pFile->QueryInfoStatusCode(dwRet);
    if (dwRet == HTTP_STATUS_OK)
    {
        UINT nRead = pFile->Read(szBuff, 1023);
        while (nRead > 0)
        {
            //read file...
        }
    }
    delete pFile;
    delete pServer;
}
catch (CInternetException* pEx)
{
    //nắm bắt các lỗi của Wininet
}
session.Close();

```

Bây giờ chúng ta sẽ xây dựng một chương trình đơn giản chạy trong môi trường DOS để thâm nhập Internet - chương trình TEAR.

12.2.3 Chương trình TEAR

Chương trình TEAR giới thiệu cách xây dựng một chương trình DOS bằng MFC sử dụng Wininet để thâm nhập Internet. Chương trình giới thiệu làm cách nào để tạo ra yêu cầu HTTP sử dụng các đối tượng CHttpFile, CHttpConnection và CInternetSession.

Chương trình 12.3 Chương trình TEAR**Tệp tear.h**

```
// Định nghĩa các lớp CTearSession và CTearException
class CTearSession : public CInternetSession
{
public:
    CTearSession(LPCTSTR pszAppName, int nMethod);
    virtual void OnStatusCallback(DWORD dwContext, DWORD dwInternetStatus,
        LPVOID lpvStatusInformation, DWORD dwStatusInformationLen);
};

class CTearException : public CException
{
    DECLARE_DYNCREATE(CTearException)
public:
    CTearException(int nCode = 0);
    ~CTearException() {}
    int m_nErrorCode;
};
//-----
```

Tệp tear.cpp

```
// Chương trình DOS thâm nhập Internet
#include <afx.h>
#include <afxwin.h>
#include <afxinet.h>
#include "tear.h"
#include <iostream.h>
#include <stdlib.h>
// Các biến tổng thể
LPCTSTR pszURL = NULL;
BOOL bStripMode = FALSE;
BOOL bProgressMode = FALSE;
DWORD dwAccessType = PRE_CONFIG_INTERNET_ACCESS;
DWORD dwHttpRequestFlags =
    INTERNET_FLAG_EXISTING_CONNECT | INTERNET_FLAG_NO_AUTO_REDIRECT;
const TCHAR szHeaders[] =
    _T("Accept: text/*\r\nUser-Agent: MFC_Tear_Sample\r\n");
// Đối tượng CTearSession
// TEAR muốn sử dụng lớp CTearSession - dẫn xuất của CInternetSession
```

```

// để có thể viết lại hàm OnStatusCallback().
CTearSession::CTearSession(LPCTSTR pszAppName, int nMethod)
    : CInternetSession(pszAppName, 1, nMethod)
{
}

void CTearSession::OnStatusCallback(DWORD /* dwContext */, DWORD dwInternetStatus,
    LPVOID /* lpvStatusInformation */, DWORD /* dwStatusInformationLen */)
{
    if (!bProgressMode)    return;
    if (dwInternetStatus == INTERNET_STATUS_CONNECTED_TO_SERVER)
        cerr << _T("Đã kết nối!") << endl;
}

// CTearException - được sử dụng để kiểm tra lỗi
// TEAR thông qua các kiểu exception để quản lý các vấn đề có thể gặp phải khi thi hành
// yêu cầu của người sử dụng.
IMPLEMENT_DYNCREATE(CTearException, CException)
CTearException::CTearException(int nCode) : m_nErrorCode(nCode)
{
}

void ThrowTearException(int nCode)
{
    CTearException* pEx = new CTearException(nCode);
    throw pEx;
}

// Các hàm xử lý
void ShowBanner()
{
    cerr << _T("TEAR - Lọc web trang HTML trên Internet!") << endl;
    cerr << _T("Phiên bản 4.2 - Phòng GIS - Viện CNTT & Microsoft - 1999") << endl;
    cerr << endl;
}

void ShowUsage()
{
    cerr << _T("Sử dụng: TEAR [luachon] <URL>") << endl << endl;
    cerr << _T("\t<URL> con trỏ tới tài nguyên HTTP") << endl;
    cerr << _T("\t[luachon]:") << endl;
    cerr << _T("\t\t/F yêu cầu nạp lại trang đã yêu cầu") << endl;
    cerr << _T("\t\t/P hiển thị chi tiết các thông tin xử lý") << endl;
    cerr << _T("\t\t/S loại bỏ các thẻ HTML") << endl << endl;
}

```

```

cerr << _T("\\t\\t/L su dung tham nhap Internet cuc bo") << endl;
cerr << _T("\\t\\t/D su dung tham nhap Internet theo cau hinh da dat (mac dinh)") << endl;
cerr << _T("\\t\\t/G su dung tham nhap Internet qua gateway") << endl;
cerr << endl;
exit(1);
}

// ParseOptions() kiểm tra các lựa chọn trong dòng lệnh và đặt cờ tổng thể
// để cho chương trình biết.
// ParseOptions() đặt pszURL trở tới xâu URL của người sử dụng.
BOOL ParseOptions(int argc, char* argv[])
{
    int nIndex;
    for (nIndex = 1; nIndex < argc; nIndex++)
    {
        // lựa chọn hay URL?
        if (*argv[nIndex] == '-' || *argv[nIndex] == '/')
        {
            if (argv[nIndex][1] == 'D' || argv[nIndex][1] == 'd')
                dwAccessType = PRE_CONFIG_INTERNET_ACCESS;
            else if (argv[nIndex][1] == 'L' || argv[nIndex][1] == 'l')
                dwAccessType = LOCAL_INTERNET_ACCESS;
            else if (argv[nIndex][1] == 'G' || argv[nIndex][1] == 'g')
                dwAccessType = GATEWAY_INTERNET_ACCESS;
            else if (argv[nIndex][1] == 'S' || argv[nIndex][1] == 's')
                bStripMode = TRUE;
            else if (argv[nIndex][1] == 'P' || argv[nIndex][1] == 'p')
                bProgressMode = TRUE;
            else if (argv[nIndex][1] == 'F' || argv[nIndex][1] == 'f')
                dwHttpRequestFlags |= INTERNET_FLAG_RELOAD;
            else
            {
                cerr << _T("Lỗi: không nhận biết được lựa chọn: ") << argv[nIndex] << endl;
                return FALSE;
            }
        }
        else
        {
            // không thể có nhiều URL
            if (pszURL != NULL)

```

```

    {
        cerr << _T("Loi: chi chap nhan mot URL!") << endl;
        return FALSE;
    }
    else
        pszURL = argv[nIndex];
    }
}
return TRUE;
}

// StripTags() đọc vùng đệm và loại bỏ các thẻ HTML trong nó.
// Hàm sử dụng biến tĩnh để nhớ trạng thái của nó trong trường hợp
// một thẻ HTML không nằm trọn trong vùng đệm.
void StripTags(LPTSTR pszBuffer)
{
    static BOOL bInTag = FALSE;
    LPTSTR pszSource = pszBuffer;
    LPTSTR pszDest = pszBuffer;
    while (*pszSource != '\0')
    {
        if (bInTag)
        {
            if (*pszSource == '>')
                bInTag = FALSE;
            pszSource++;
        }
        else
        {
            if (*pszSource == '<')
                bInTag = TRUE;
            else
            {
                *pszDest = *pszSource;
                pszDest++;
            }
            pszSource++;
        }
    }
    *pszDest = '\0';
}

// Hàm main()

```

```

int main(int argc, char* argv[])
{
    ShowBanner();
    if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
    {
        cerr << _T("MFC loi khi khoi tao.\n");
        return 1;
    }
    if (argc < 2 || !ParseOptions(argc, argv) || pszURL == NULL)
        ShowUsage();
    int nRetCode = 0;
    CTearSession session(_T("TEAR - chuong trinh vi du MFC"), dwAccessType);
    CHttpConnection* pServer = NULL;
    CHttpFile* pFile = NULL;
    try
    {
        // Kiểm tra xâu URL
        CString strServerName;
        CString strObject;
        INTERNET_PORT nPort;
        DWORD dwServiceType;
        if (!AfxParseURL(pszURL, dwServiceType, strServerName, strObject, nPort) ||
            dwServiceType != INTERNET_SERVICE_HTTP)
        {
            cerr << _T("Loi: chi su dung URL bat dau bang http://") << endl;
            ThrowTearException(1);
        }
        if (bProgressMode)
        {
            cerr << _T("Dang ket noi Internet...");
            VERIFY(session.EnableStatusCallback(TRUE));
        }
        pServer = session.GetHttpConnection(strServerName, nPort);
        pFile = pServer->OpenRequest(CHttpConnection::HTTP_VERB_GET,
            strObject, NULL, 1, NULL, NULL, dwHttpRequestFlags);
        pFile->AddRequestHeaders(szHeaders);
        pFile->SendRequest();
        DWORD dwRet;
        pFile->QueryInfoStatusCode(dwRet);
    }
}

```

```

// Nếu kết nối bị từ chối, nhắc người sử dụng mật khẩu
if (dwRet == HTTP_STATUS_DENIED)
{
    DWORD dwPrompt;
    dwPrompt = pFile->ErrorDlg(NULL, ERROR_INTERNET_INCORRECT_PASSWORD,
        FLAGS_ERROR_UI_FLAGS_GENERATE_DATA |
        FLAGS_ERROR_UI_FLAGS_CHANGE_OPTIONS, NULL);
    if (dwPrompt != ERROR_INTERNET_FORCE_RETRY)
    {
        cerr << _T("Ket noi bi tu choi: Sai mat khau\n");
        ThrowTearException(1);
    }
    pFile->SendRequest();
    pFile->QueryInfoStatusCode(dwRet);
}
CString strNewLocation;
pFile->QueryInfo(HTTP_QUERY_RAW_HEADERS_CRLF, strNewLocation);
// bạn đã gửi lại?
// các mã trạng thái đáp lại có trong WININET.H
if (dwRet == HTTP_STATUS_MOVED ||
    dwRet == HTTP_STATUS_REDIRECT ||
    dwRet == HTTP_STATUS_REDIRECT_METHOD)
{
    CString strNewLocation;
    pFile->QueryInfo(HTTP_QUERY_RAW_HEADERS_CRLF, strNewLocation);
    int nPlace = strNewLocation.Find(_T("Location: "));
    if (nPlace == -1)
    {
        cerr << _T("Loi: dia chi gui lai khong co gi moi") << endl;
        ThrowTearException(2);
    }
    strNewLocation = strNewLocation.Mid(nPlace + 10);
    nPlace = strNewLocation.Find("\n");
    if (nPlace > 0)
        strNewLocation = strNewLocation.Left(nPlace);
    pFile->Close();
    delete pFile;
    pServer->Close();
    delete pServer;
}

```



```

if (bProgressMode)
{
    cerr << _T("Thao trong: gui lai cho ");
    cerr << (LPCTSTR) strNewLocation << endl;
}
// xác định địa điểm cũ
if (!AfxParseURL(strNewLocation, dwServiceType, strServerName, strObject, nPort))
{
    cerr << _T("Lỗi: không phân tích được xau URL.") << endl;
    ThrowTearException(2);
}
if (dwServiceType != INTERNET_SERVICE_HTTP)
{
    cerr << _T("Lỗi: URL không xác định tại nguyên HTTP.") << endl;
    ThrowTearException(2);
}
// thử với địa chỉ mới
pServer = session.GetHttpConnection(strServerName, nPort);
pFile = pServer->OpenRequest(CHttpConnection::HTTP_VERB_GET,
    strObject, NULL, 1, NULL, NULL, dwHttpRequestFlags);
pFile->AddRequestHeaders(szHeaders);
pFile->SendRequest();
pFile->QueryInfoStatusCode(dwRet);
if (dwRet != HTTP_STATUS_OK)
{
    cerr << _T("Lỗi: Nhận mã trạng thái ") << dwRet << endl;
    ThrowTearException(2);
}
}
cerr << _T("Mã trạng thái là ") << dwRet << endl;
TCHAR sz[1024];
while (pFile->ReadString(sz, 1023))
{
    if (bStripMode)
        StripTags(sz);
    cout << sz;
}
// Chú ý: bởi vì thông thường dịch vụ HTTP gửi lại văn bản thông thường,
// nên đoạn mã trình trên (đọc từng dòng) thực hiện tốt.

```

```

// Tuy nhiên, các dữ liệu tài nguyên khác (thí dụ dịch vụ FTP) có thể gửi lại dữ liệu dưới
// dạng nhị phân, khi đó cần quản lý vùng đệm như sau:
#if 0
    while (nRead > 0)
    {
        sz[nRead] = '\0';
        if (bStripMode)
            StripTags(sz);
        cout << sz;
        nRead = pFile->Read(sz, 1023);
    }
#endif
    pFile->Close();
    pServer->Close();
}
catch (CInternetException* pEx)
{
    // bắt lỗi từ WinInet
    TCHAR szErr[1024];
    pEx->GetErrorMessage(szErr, 1024);
    cerr << _T("Lỗi: (") << pEx->m_dwError << _T(")");
    cerr << szErr << endl;
    nRetCode = 2;
    pEx->Delete();
}
catch (CTearException* pEx)
{
    // bắt được lỗi
    nRetCode = pEx->m_nErrorCode;
    TRACE1("Lỗi: Thoát ra với CTearException(%d)\n", nRetCode);
    pEx->Delete();
}
if (pFile != NULL)    delete pFile;
if (pServer != NULL) delete pServer;
session.Close();
return nRetCode;
}

```

Để chạy chương trình TEAR hãy mở cửa sổ DOS, gõ vào dòng lệnh:

TEAR xâu_URL, thí dụ:

TEAR `http://www.microsoft.com/ie/support/default.htm` <enter>

Chương trình TEAR cps khả năng thâm nhập dữ liệu qua Internet, thực hiện sao chép trang `http://www.microsoft.com/ie/support/default.htm`, đưa kết quả ra thiết bị chuẩn (*stdout*). Chương trình TEAR chấp nhận một số lựa chọn. Để hiển thị các lựa chọn của chương trình tại cửa sổ DOS hãy nhập lệnh TEAR <enter>.

Người sử dụng có thể sử dụng lựa chọn /D (mặc định) để yêu cầu chương trình sử dụng các tham số thâm nhập Internet đã được xác định khi cài đặt kết nối mạng trong chương trình *Control Panel* hay *Internet Access Setup Wizard*. Sử dụng lựa chọn /L nếu máy tính trong mạng cục bộ nối thẳng với Internet. Sử dụng lựa chọn /G nếu máy tính nối với Internet qua *gateway*. Chương trình đáp ứng các lựa chọn này bằng việc thay đổi cờ giao dịch chuyển cho việc tạo lập *CInternetSession*. -

Lựa chọn /S loại bỏ các thẻ định dạng của các xâu văn bản nhận được. Lựa chọn /P yêu cầu chương trình sử dụng *CInternetSession::EnableStatusCallback* để lấy thông tin trong quá trình xử lý kết nối.

Một kết quả thực hiện chương trình TEAR có thể như sau:

```
C:\>tear http://popmap-2/scripts/mypage.htm
TEAR - Lay ve trang HTML tren Internet!
Phien ban 1.0 - Phong GIS - Vien CNTT - 1999
```

Ma trang thai la 200

```
<HTML>
<HEAD>
<TITLE>Counter Demonstration</TITLE>
</HEAD>
<BODY>
The current system time is:
<IMG SRC="http://203.162.0.3/scripts/counter.dll?clock"><P>
Visits to the page referred to by mypage.htm:
<IMG SRC="http://203.162.0.3/scripts/counter.dll/mypage.htm"><P>
Visits to this page:
<IMG SRC="http://203.162.0.3/scripts/counter.dll"><P>
</BODY>
</HTML>
```

```
C:\>
```

12.2.4 Quản lý thông tin trạng thái Internet trong quá trình thực hiện

Khi sử dụng các lớp của WinInet, lập trình viên có thể sử dụng hàm thành phần **OnStatusCallback** của đối tượng **CInternetSession** để khai thác thông tin trạng thái. Nếu trong chương trình, chúng ta xây dựng lớp phái sinh từ lớp **CInternetSession**, nạp chồng hàm **OnStatusCallback** và cho phép OS gọi chúng, MFC sẽ gọi hàm **OnStatusCallback** của chúng ta trong quá trình xử lý thông tin trong toàn bộ phiên giao dịch Internet.

Bởi vì một phiên giao dịch Internet có thể trợ giúp một vài liên kết, hàm **OnStatusCallback** cần một kỹ thuật để nhận biết mỗi thay đổi trạng thái của một kết nối hay giao dịch. Kỹ thuật được sử dụng là cung cấp tham số định danh ngữ cảnh cho hàm thành phần của các lớp trong WinInet. Tham số này luôn có kiểu **DWORD** và luôn có tên là **dwContext**. Ngữ cảnh gán cho mỗi đối tượng Internet riêng biệt được sử dụng chỉ để định danh hành động đối tượng tạo ra trong hàm thành phần **OnStatusCallback** của đối tượng **CInternetSession**. Khi hàm **OnStatusCallback** được gọi, nó nhận một vài tham số, các tham số này làm việc cùng nhau để thông báo cho chương trình biết việc xử lý nào đã được thực hiện cho giao dịch và kết nối nào.

Chúng ta có thể chỉ rõ tham số **dwContext** khi tạo ra một đối tượng **CInternetSession**. Bản thân đối tượng **CInternetSession** không sử dụng định danh ngữ cảnh mà chuyển giá trị này cho các đối tượng phái sinh từ **CInternetConnection** không nhận được định danh ngữ cảnh rõ ràng của bản thân mình. Đến lượt mình, các đối tượng **CInternetConnection** này lại chuyển định danh ngữ cảnh cho các đối tượng **CInternetFile** tạo ra nếu nó không được chỉ rõ một giá trị khác. Mặt khác, nếu chúng ta chỉ rõ một giá trị cụ thể thì đối tượng và các công việc nó làm sẽ sử dụng giá trị này. Ta có thể sử dụng các giá trị định danh để chỉ định thông tin trạng thái nào sẽ được chuyển trả lại trong hàm **OnStatusCallback**.

Thí dụ, khi xây dựng một chương trình tạo kết nối với một dịch vụ FTP để đọc tệp, đồng thời cũng kết nối với một dịch vụ HTTP để đọc trang Web, chúng ta sẽ có một đối tượng **CInternetSession**, hai đối tượng **CInternetConnection** (một đối tượng là **CFTPSession**, một đối tượng là **CHttpSession**) và hai đối tượng **CInternetFile**. Nếu sử dụng giá trị mặc định cho các tham số **dwContext** thì ta không thể phân biệt giữa việc yêu cầu hàm **OnStatusCallback** cho xử lý liên kết FTP hay cho liên kết HTTP. Nếu chỉ rõ giá trị cho **dwContext**, thì ta có thể kiểm tra nó trong hàm **OnStatusCallback** để biết thao tác nào tạo ra *callback*.

CHƯƠNG 13

HƯỚNG DẪN SỬ DỤNG DEVELOPER STUDIO

Trong các chương trước chúng ta đã biết cách xây dựng và viết các chương trình ứng dụng trong Window bằng MFC. Chương này sẽ giới thiệu nhanh cách sử dụng chương trình *Developer Studio* của Microsoft để soạn thảo chương trình nguồn, dịch, liên kết và chạy kiểm tra các chương trình.

13.1 CÁC THÀNH PHẦN CỦA HỆ THỐNG

13.1.1 Chương trình soạn thảo

Đây là môi trường để lập trình viên tạo và soạn thảo các văn bản chương trình nguồn. Giống như các chương trình soạn thảo văn bản khác, chương trình soạn thảo này cũng có đầy đủ các chức năng như cắt, dán, sửa đổi, xen... Ngoài ra chương trình soạn thảo này còn tự động nhận biết các từ khoá của ngôn ngữ C++ và cho phép lập trình viên phân biệt các kiểu từ khoá bằng màu khác nhau. Điều này không chỉ giúp cho việc tạo nên các chương trình nguồn dễ đọc mà còn giúp lập trình viên dễ dàng phát hiện các lỗi cú pháp khi soạn thảo.

13.1.2 Chương trình dịch

Chương trình dịch có chức năng chuyển chương trình nguồn ra ngôn ngữ máy, phát hiện và thông báo các lỗi khi dịch chương trình. Chương trình dịch có thể phát hiện các lỗi về cú pháp, các lỗi về cấu trúc hay các phần chương trình không bao giờ được thực hiện. Đầu ra của chương trình dịch được hiểu như mã đối tượng và được chứa trong một tệp đối tượng có cùng tên với tên tệp chương trình nguồn và có phần mở rộng là OBJ.

13.1.3 Chương trình liên kết

Chương trình liên kết thực hiện việc liên kết các modul được tạo ra từ chương trình dịch cộng thêm các modul thư viện cần thiết của C++ được yêu cầu và hợp nhất lại để tạo nên một chương trình thực hiện hoàn thiện. Chương trình dịch cũng có khả năng phát hiện và thông báo các lỗi khi liên kết như chương trình thiếu các đoạn mã cho các hàm hay không có các hàm thư viện .v.v.

13.1.4 Các thư viện của C++

Các chương trình thư viện là phần mở rộng của C++, chúng cung cấp các hàm xử lý thông dụng.

Thí dụ các thư viện có thể chứa các hàm tính căn, so sánh xâu ký tự, lấy các thông tin thời gian .v.v. Visual C++ cung cấp hai loại thư viện sau đây: .

- Loại thứ nhất là thư viện chuẩn, chúng chứa các hàm thông dụng nhất cho tất cả các chương trình dịch của C++.
- Loại thứ hai là các thư viện lớp được gọi là thư viện MFC. Thư viện MFC cung cấp các thành phần cơ bản cho tất cả các chương trình Windows. Thư viện MFC cũng được xem như một chương trình khung bởi vì nó cung cấp một loạt các cấu trúc thành phần tạo nên nền tảng cho hầu hết các chương trình Windows.

13.1.5 Các công cụ khác

Developer Studio còn bao gồm các công cụ quan trọng trợ giúp lập trình như *AppWizard*, *ClassWizard* và *Wizard Bar*.

- *AppWizard*: là công cụ tự động tạo ra khung cơ bản cho chương trình Windows; lập trình viên chỉ cần bổ sung thêm các hàm xử lý riêng của mình.
- *ClassWizard*: Các lớp là nét đặc trưng quan trọng nhất của C++ và là nền tảng cho lập trình Windows với Visual C++. *ClassWizard* cung cấp một cách đơn giản cho việc mở rộng các lớp được tạo ra từ *AppWizard*. Nó cũng giúp lập trình viên dễ dàng tạo thêm các lớp mới trên cơ sở các lớp trong MFC.
- *Wizard Bar*: cung cấp một khả năng khác cho việc quản lý, sửa đổi và mở rộng mã chương trình bằng việc lựa chọn hiển thị trong vùng thanh công cụ của cửa sổ *Developer Studio*.

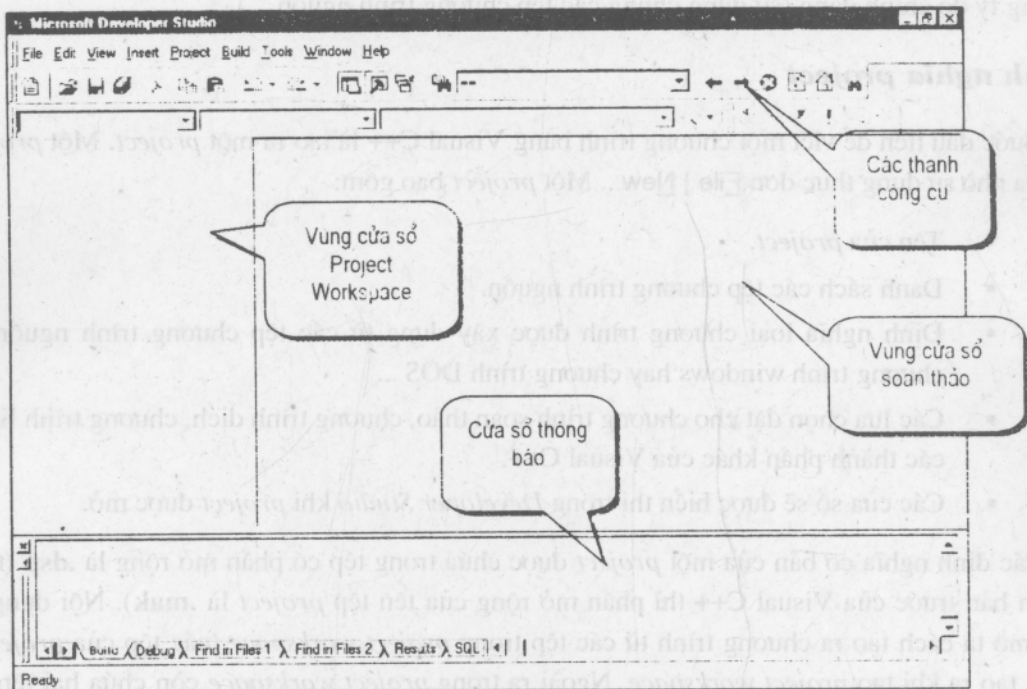
13.2 SỬ DỤNG MÔI TRƯỜNG PHÁT TRIỂN CỦA VISUAL C++

Tất cả các chương trình được mô tả trong các chương trước của tài liệu này đều có thể phát triển trong môi trường của *Developer Studio*. Người sử dụng cũng có thể sử dụng môi trường này xây dựng các ứng dụng riêng, phức tạp hơn. Màn hình giao diện sau khi khởi động chương trình Microsoft Visual C++ 6 được thể hiện trên hình 13.1.

Màn hình giao diện của *Developer Studio* bao gồm các phần chính như sau:

- Cửa sổ *Project Workspace*: Hiển thị thông tin của *project* như các lớp định nghĩa, các tệp liên quan....
- Cửa sổ soạn thảo: Soạn thảo các văn bản chương trình nguồn bao gồm các tệp *.CPP, *.H, *.RC và các tệp tài nguyên khác.
- Cửa sổ thông báo: Hiển thị các thông báo khi dịch và liên kết chương trình.
- Các thanh công cụ: Cho phép thực hiện các chức năng trong thực đơn bằng cách nhấn

vào các phím hình tương ứng.



Hình 13.1 Màn hình giao diện của Developer Studio

Lựa chọn hiển thị các thanh công cụ

Để lựa chọn việc hiển thị các thanh công cụ mong muốn, ta đưa con trỏ của chuột vào vùng thanh công cụ và ấn phím phải của chuột, một bảng lựa chọn sẽ cho phép ta chọn/không chọn các thanh công cụ cần thiết. Ta có thể đặt các thanh công cụ vào vị trí mong muốn bằng việc sử dụng chuột để di-kéo-nhả (*drag around*).

13.3 PROJECT VÀ PROJECT WORKSPACE

Một *project* đơn giản là một loại chương trình, nó có thể là một chương trình trong môi trường DOS (*console program*), một chương trình Windows hay một vài loại chương trình khác. Một *project workspace* là một thư mục để lưu tất cả thông tin của *project* và các tệp dữ liệu liên quan. Khi ta tạo một *project* thì một *project workspace* cũng được tự động tạo ra và *Developer Studio* sẽ quản lý tất cả các tệp chương trình nguồn và các tệp liên quan trong thư mục *project workspace*. Thư mục này cũng chứa các thư mục con (*Debug*, *Release*) để chứa các tệp kết quả (.obj, .exe...), chúng được tự động tạo ra khi thực hiện dịch và liên kết chương trình.

Trong cùng một *project workspace*, lập trình viên cũng có thể tạo thêm các *project* khác. Tuy nhiên chúng ta không nên làm điều này trừ khi đã hiểu rõ cách sử dụng chương trình dịch và có những lý do chính đáng (sử dụng chung các tệp chương trình nguồn...).

Định nghĩa project

Bước đầu tiên để viết một chương trình bằng Visual C++ là tạo ra một *project*. Một *project* được tạo ra nhờ sử dụng thực đơn **File | New...** Một *project* bao gồm:

- Tên của *project*.
- Danh sách các tệp chương trình nguồn.
- Định nghĩa loại chương trình được xây dựng từ các tệp chương trình nguồn, thí dụ: chương trình windows hay chương trình DOS ...
- Các lựa chọn đặt cho chương trình soạn thảo, chương trình dịch, chương trình liên kết và các thành phần khác của Visual C++.
- Các cửa sổ sẽ được hiển thị trong *Developer Studio* khi *project* được mở.

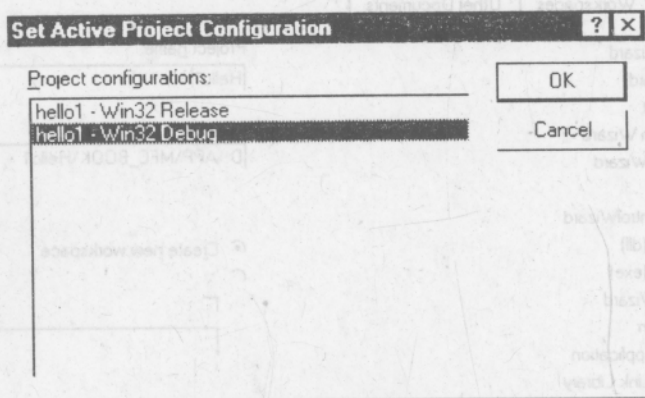
Các định nghĩa cơ bản của một *project* được chứa trong tệp có phần mở rộng là **.dsp** (trong các phiên bản trước của Visual C++ thì phần mở rộng của tên tệp *project* là **.mak**). Nội dung của tệp này mô tả cách tạo ra chương trình từ các tệp trong *project workspace* (các tệp của *project*) và nó được tạo ra khi tạo *project workspace*. Ngoài ra trong *project workspace* còn chứa hai tệp có phần mở rộng là **.opt** và **.dsw**. Tệp có phần mở rộng là **.opt** chứa thông tin về các thiết lập môi trường (*settings*) cho *project workspace*. Tệp này cũng chứa các thông tin về trạng thái của *project* để có thể khôi phục lại trạng thái trước đó của *project* khi được mở. Tệp có phần mở rộng là **.dsw** chứa các thông tin xa hơn về *workspace*, thí dụ, *workspace* chứa các *project* nào.

Tất cả các tệp trên được tự động tạo ra và được Visual C++ và Developer Studio cập nhật cho nên lập trình viên không nên sửa đổi chúng. Các thay đổi cần thiết nên được thực hiện thông qua các chức năng trong thực đơn của *Developer Studio*.

Phiên bản Debug hay Release của chương trình

Sau khi tạo ra một *project*, lập trình viên có thể thay đổi các thông số, các lựa chọn để mô tả cách xử lý các chương trình nguồn khi dịch và liên kết (thay đổi cấu hình) thông qua chức năng **Project | Settings...** trên thực đơn. Khi tạo ra một *project* mới, chương trình *Developer Studio* tự động sinh ra cấu hình để tạo ra hai phiên bản của chương trình. Một phiên bản bao gồm các thông tin cho phép lập trình viên chạy chương trình theo từng lệnh và theo dõi việc thực hiện từng lệnh, kiểm tra giá trị của các biến dữ liệu, phát hiện lỗi khi thực hiện, phiên bản này được gọi là phiên bản gỡ rối (*Debug*). Phiên bản thứ hai không chứa các thông tin gỡ rối và chương trình có mã tối ưu, phiên bản này được gọi là phiên bản phân phát (*Release*). Việc thay đổi cấu hình của hai phiên bản này được thực hiện thông qua chức năng **Build | Configurations...** trên thực đơn.

Lập trình viên có thể lựa chọn việc tạo ra phiên bản chương trình nào khi làm việc thông qua chức năng Build | Set Active Configuration...



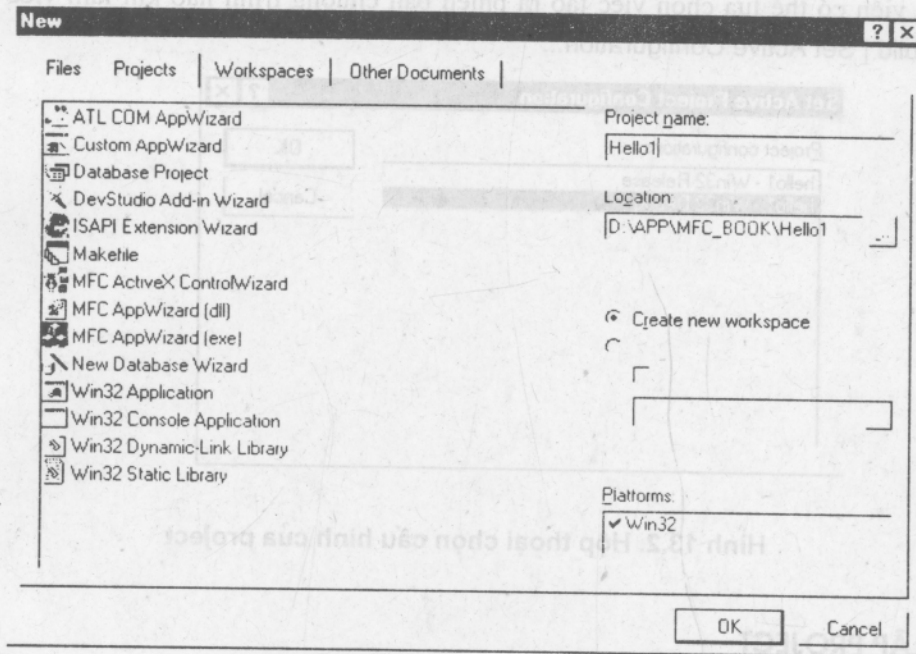
Hình 13.2. Hộp thoại chọn cấu hình của project

13.4 TẠO LẬP PROJECT

Để biết cách tạo ra một *project* bằng *Developer Studio* chúng ta bắt đầu bằng việc tạo ra chương trình Hello1 mô tả trong chương 1. Sau đây là các bước thực hiện:

- Khởi động chương trình *Developer Studio*.
- Chọn chức năng **File | New...** chọn bằng *Projects*, hộp thoại *New* được hiển thị như trên hình 13.3.
- Chọn Win32 Application trong danh sách kiểu *project*.
- Hãy nhập xâu ký tự **Hello1** vào cửa sổ soạn thảo *Project name*.
Tên thư mục chứa *project Hello1* được tự động tạo ra (trùng với tên *project*) như thông báo ở cửa sổ soạn thảo *Location*:. Lập trình viên có thể thay đổi thư mục chứa *project Hello1* bằng cách gõ vào tên thư mục mong muốn hoặc chọn thông qua phím **...** (*browse*).
- Kết thúc công việc bằng nhấn phím OK.

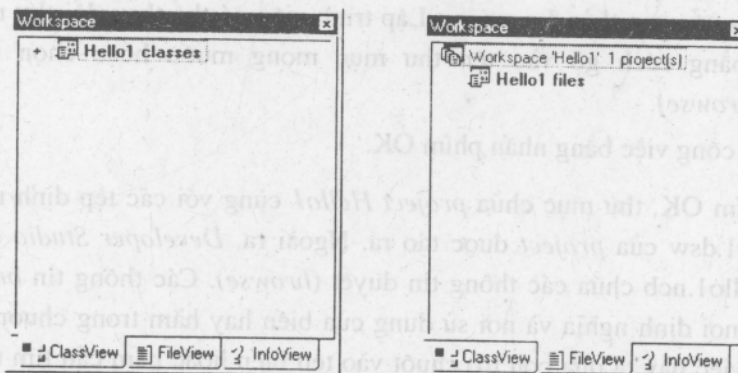
Sau khi nhấn phím OK, thư mục chứa *project Hello1* cùng với các tệp định nghĩa Hello1.dsp, Hello1.opt và Hello1.dsw của *project* được tạo ra. Ngoài ra, *Developer Studio* còn tạo thêm tệp Hello1.ncb. Tệp Hello1.ncb chứa các thông tin duyệt (*browse*). Các thông tin *browse* sẽ giúp lập trình viên xác định nơi định nghĩa và nơi sử dụng của biến hay hàm trong chương trình trong khi soạn. Để thực hiện việc này ta đưa con trỏ chuột vào tên biến hoặc hàm cần tìm trong cửa sổ soạn thảo, nhấn phím phải chuột và chọn chức năng *Go To Definition* hay *Go To Reference* trong thực đơn *pop-up*.



Hình 13.3 Hộp thoại tạo project

Một không gian làm việc mới sẽ được mở trong *Developer Studio*. Hai bảng mới sẽ được mở trong cửa sổ *Project Workspace* là *ClassView* và *FileView*.

- Bảng *ClassView* cho phép lập trình viên quan sát tổng thể cũng như nội dung của các lớp định nghĩa trong *project*.
- Bảng *FileView* cho phép lập trình viên quan sát các tệp nguồn của *project*.



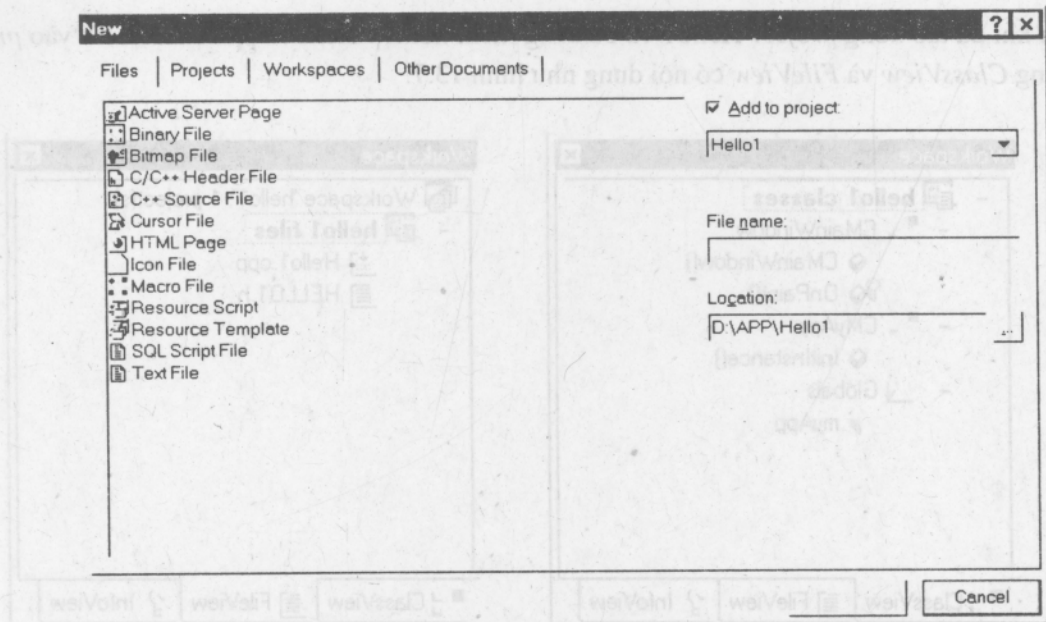
Hình 13.4 Nội dung bảng ClassView và FileView

Khi lập trình viên tạo các tài nguyên như thực đơn, hộp thoại, ảnh bitmap, biểu tượng,... thì *Developer Studio* tạo thêm một bảng *ResourceView*. Các bảng *ClassView* và *FileView* được thể hiện trong hình 13.4.

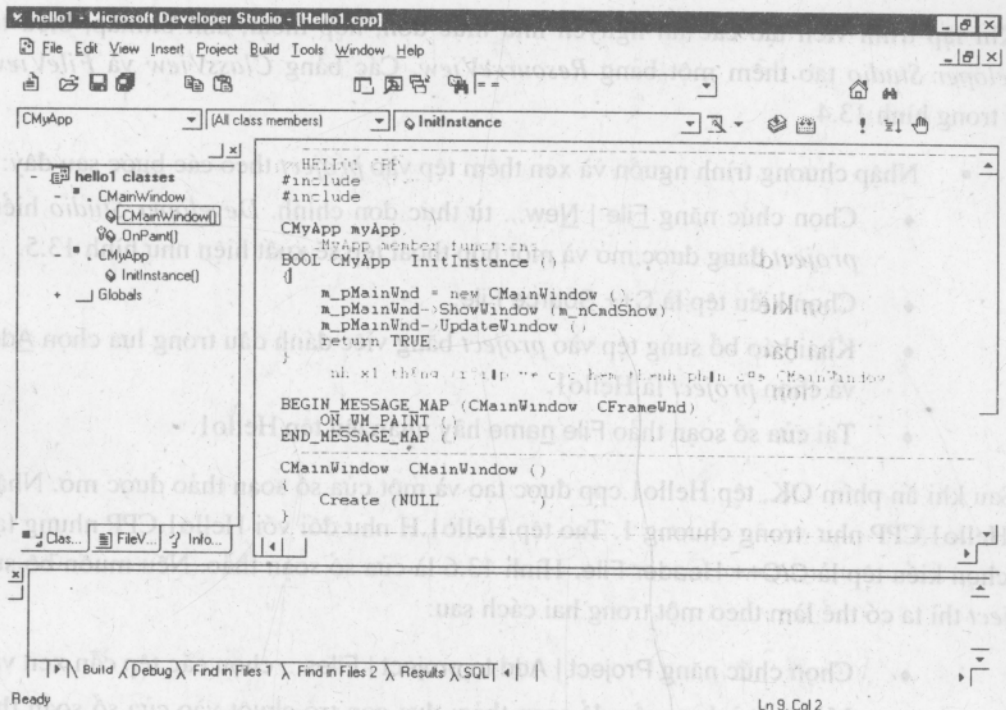
- Nhập chương trình nguồn và xen thêm tệp vào *project* theo các bước sau đây:
 - ♦ Chọn chức năng **File | New...** từ thực đơn chính. *Developer Studio* hiểu rằng một *project* đang được mở và một hộp thoại tệp sẽ xuất hiện như hình 13.5.
 - ♦ Chọn kiểu tệp là **C++ Source File**.
 - ♦ Khai báo bổ sung tệp vào *project* bằng việc đánh dấu trong lựa chọn **Add to project** và chọn *project* là **Hello1**.
 - ♦ Tại cửa sổ soạn thảo **File name** hãy nhập tên tệp **Hello1**.

Sau khi ấn phím **OK**, tệp **Hello1.cpp** được tạo và một cửa sổ soạn thảo được mở. Nhập nội dung tệp **Hello1.CPP** như trong chương 1. Tạo tệp **Hello1.H** như đối với **Hello1.CPP** nhưng tại hộp thoại tệp chọn kiểu tệp là **C/C++ Header File**. Hình 13.6 là cửa sổ soạn thảo. Nếu muốn bổ sung tệp vào *project* thì ta có thể làm theo một trong hai cách sau:

- ♦ Chọn chức năng **Project | Add to project | Files...** chọn các tệp cần xen vào *project*.
- ♦ Mở tệp trình nguồn để soạn thảo; đưa con trỏ chuột vào cửa sổ soạn thảo và nhấn phím phải chuột; chọn chức năng **Insert File Into Project** trong thực đơn pop-up.

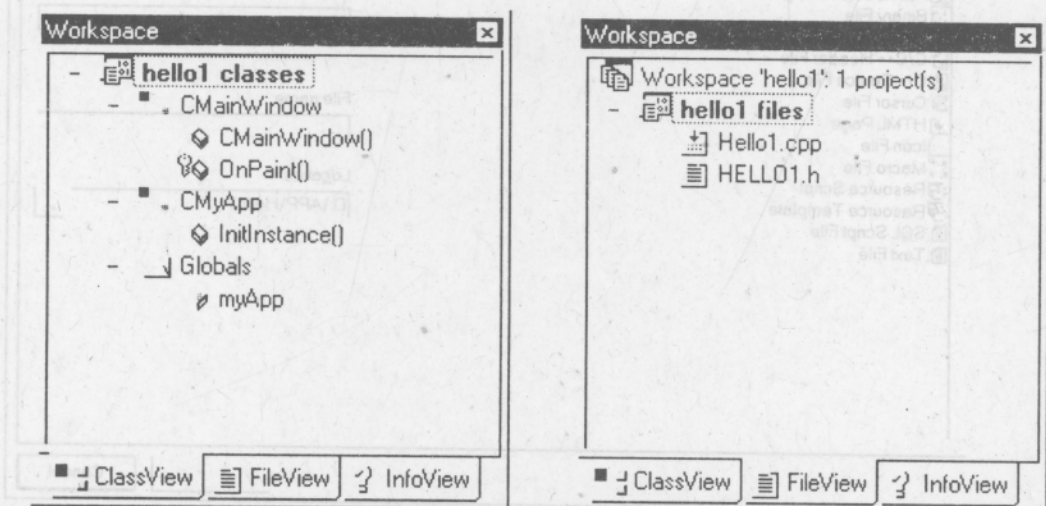


Hình 13.5 Hộp thoại tạo tệp



Hình 13.6 Cửa sổ soạn thảo chương trình

Sau khi đã tạo xong *project* Hello1 và bổ sung thêm hai tệp Hello1.cpp và Hello1.h vào *project*, các bảng *ClassView* và *FileView* có nội dung như hình 13.7.



Hình 13.7 Nội dung bảng ClassView và FileView

13.5 XÂY DỰNG CHƯƠNG TRÌNH

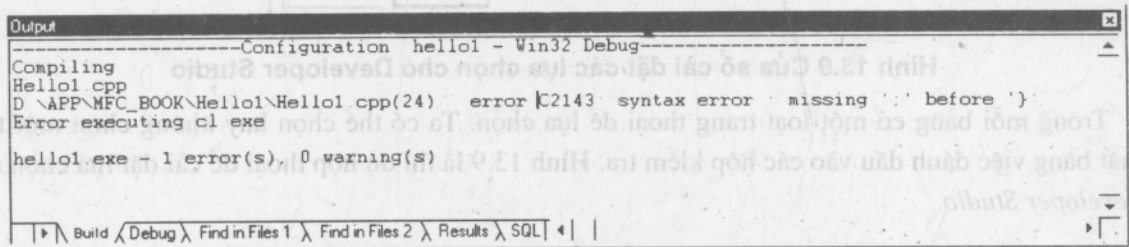
Việc dịch các tệp chương trình nguồn tạo ra các tệp đối tượng .OBJ và liên kết các tệp đối tượng để tạo ra một chương trình được gọi là xây dựng chương trình (*building a project*). Các cách để xây dựng một chương trình được mô tả dưới đây:

Cách 1: Nhấn vào phím  trên thanh công cụ xây dựng (*Build toolbar*).

Cách 2: Chọn chức năng **Build** | **Build Hello1.exe** trên thực đơn chính.

Cách 3: Chọn chức năng **Build** trong thực đơn xuất hiện khi nhấn phím phải của chuột trên **Hello1 files** trong bảng *FileView* trên cửa sổ *Project workspace*.

Chú ý rằng các tệp nguồn sẽ tự động được ghi lại trước khi thực hiện việc xây dựng chương trình. Khi xây dựng chương trình cửa sổ thông báo sẽ cung cấp cho lập trình viên trạng thái và các lỗi trong quá trình xử lý. Hình 13.8 là thí dụ cửa sổ thông báo lỗi khi dịch chương trình Hello1.



Hình 13.8 Cửa sổ thông báo khi dịch

Đối với các lỗi cú pháp, lập trình viên đặt lựa chọn vào lỗi này và nhấn phím *Enter*, *Developer Studio* sẽ chuyển con trỏ tới dòng lệnh có lỗi trong cửa sổ soạn thảo để sửa. Trong nhiều trường hợp, lập trình viên không biết rõ nguyên nhân của một số lỗi và cần biết thêm thông tin về chúng thì có thể chuyển lựa chọn vào lỗi quan tâm và nhấn phím *F1*, một cửa sổ trợ giúp sẽ được mở để mô tả chi tiết về lỗi.

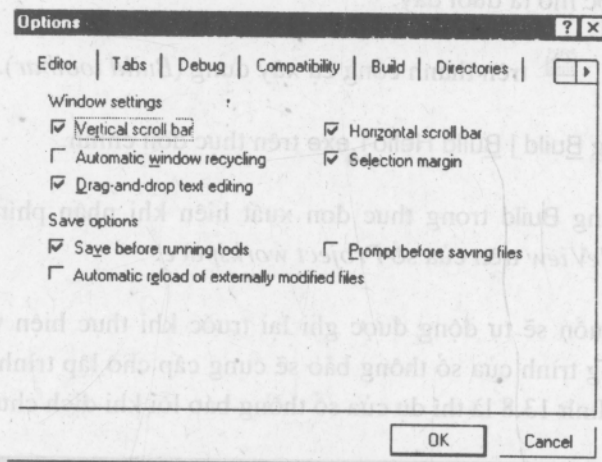
Sau khi sửa xong tất cả các lỗi, lập trình viên cần phải sử dụng chức năng *Build* để xây dựng lại chương trình.

13.6 CÀI ĐẶT CÁC LỰA CHỌN TRONG VISUAL C++

Có hai loại lựa chọn trong Visual C++. Loại thứ nhất là các lựa chọn các công cụ của *Developer Studio*, các lựa chọn này có tác dụng đối với tất cả các *project*. Loại thứ hai là các lựa chọn riêng cho từng *project*, các lựa chọn này khai báo cách thức xử lý khi dịch và liên kết chương trình.

13.6.1 Cài đặt các lựa chọn cho Developer Studio

Việc cài đặt các lựa chọn cho *Developer Studio* được thực hiện thông qua hộp thoại khi lập trình viên chọn chức năng **Tools | Options...** từ thực đơn chính.



Hình 13.9 Cửa sổ cài đặt các lựa chọn cho Developer Studio

Trong mỗi bảng có một loạt trang thoại để lựa chọn. Ta có thể chọn hay không chọn một tính chất bằng việc đánh dấu vào các hộp kiểm tra. Hình 13.9 là thí dụ hộp thoại để cài đặt lựa chọn cho *Developer Studio*.

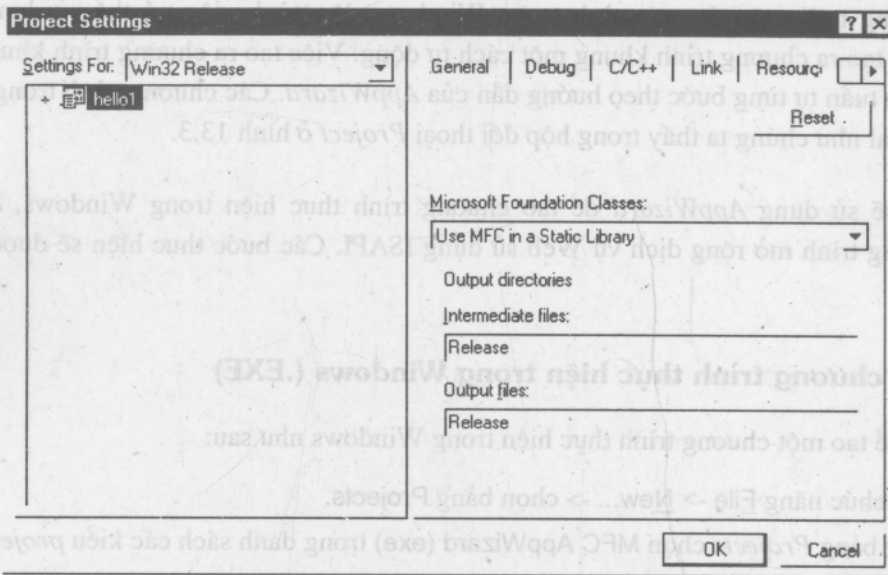
13.6.2 Cài đặt các lựa chọn cho project

Để cài đặt các lựa chọn cho *project*, hãy chọn chức năng **Project | Settings...** từ thực đơn chính. Một hộp thoại được mở ra như hình 13.10. Có thể bỏ qua hầu hết các lựa chọn này nhưng có hai lựa chọn cần quan tâm là bảng **General** và bảng **Browse Info**.

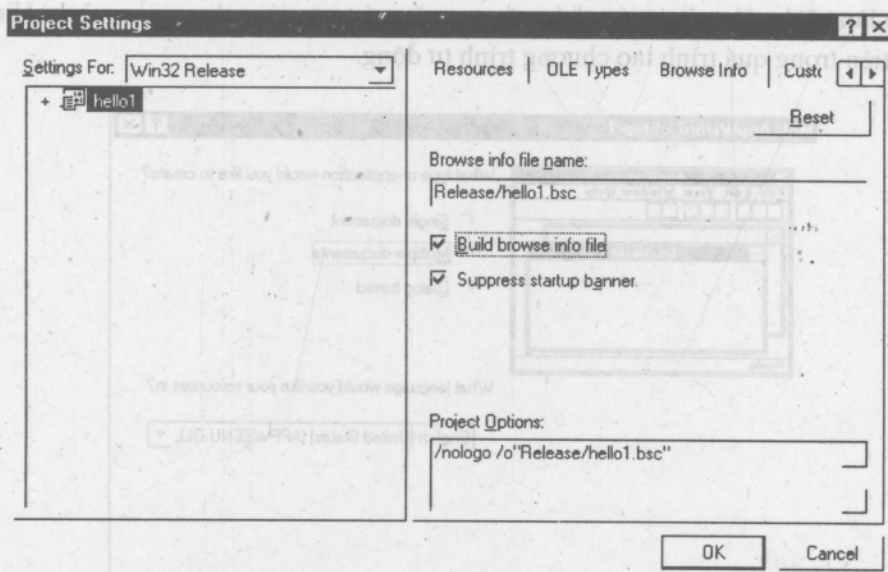
Trong bảng **General** đối với các ứng dụng DOS, hãy chọn chức năng **Not Using MFC** trong hộp combo **Microsoft Foundation Classes**. Đối với các ứng dụng Windows 32 bit lập trình viên có thể chọn chức năng **Use MFC in a Static Library** hoặc **Use MFC in a Shared DLL**. Trong cuốn sách này đối với tất cả các thí dụ trừ *Life* và *EzPrint* lập trình viên có thể chọn chức năng **Use MFC in a Static Library** hoặc **Use MFC in a Shared DLL**. Riêng đối với thí dụ *Life* và *EzPrint* lập trình viên phải chọn chức năng **Use MFC in a Shared DLL** thì chương trình mới thực hiện đúng.

Lập trình viên sẽ nhận thấy sự tiện lợi của việc tạo lập các thông tin *browse* cho *project* thông qua bảng **Browse Info**. Các thông tin *browse* cho phép lập trình viên có thể tìm thấy trong văn bản chương trình nguồn nơi định nghĩa (khai báo) hay sử dụng của bất kỳ biến dữ liệu hay hàm nào bằng việc nhấn phím chuột phải và chọn từ thực đơn *pop-up* như đã hướng dẫn ở phần trên.

Để *Developer Studio* tạo ra các thông tin *browse*, lập trình viên cần chọn chức năng **Build** *browse info* file trong bảng *Browse Info* như hình 13.11.



Hình 13.10 Cửa sổ cài đặt các lựa chọn cho project



Hình 13.11 Cửa sổ cài đặt các lựa chọn xây dựng Browse Info

13.7 TẠO CHƯƠNG TRÌNH WINDOWS BẰNG CÔNG CỤ AppWizard

Để xây dựng mới một chương trình trong Windows, lập trình viên có thể sử dụng công cụ *AppWizard* để tạo ra chương trình khung một cách tự động. Việc tạo ra chương trình khung tự động được thực hiện tuần tự từng bước theo hướng dẫn của *AppWizard*. Các chương trình trong Windows có rất nhiều loại như chúng ta thấy trong hộp đối thoại *Project* ở hình 13.3.

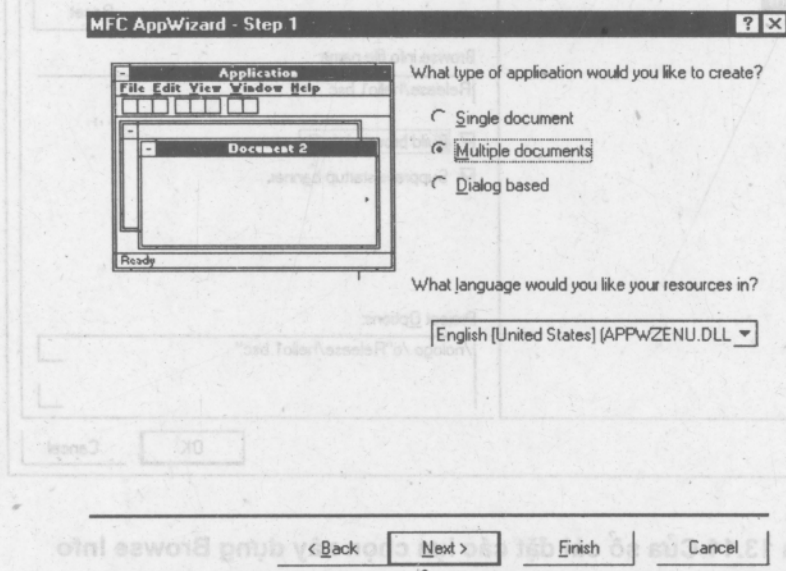
Chúng ta sẽ sử dụng *AppWizard* để tạo chương trình thực hiện trong Windows, điều khiển *ActiveX*, chương trình mở rộng dịch vụ Web sử dụng *ISAPI*. Các bước thực hiện sẽ được trình bày dưới đây.

13.7.1 Tạo chương trình thực hiện trong Windows (.EXE)

Các bước để tạo một chương trình thực hiện trong Windows như sau:

1. Chọn chức năng **File -> New... ->** chọn bảng **Projects**.
2. Trong bảng **Projects** chọn **MFC AppWizard (exe)** trong danh sách các kiểu *project*.
3. Đặt tên cho *project* trong cửa sổ soạn thảo *project name*, chọn **Create new workspace** và chọn **OK**.

Sau khi nhấn phím **OK**, chương trình *AppWizard* sẽ hiển thị từng bước các hộp thoại với các lựa chọn cho phép lập trình viên chọn các chức năng muốn có trong ứng dụng của mình. Hình 13.12 là hộp thoại đầu tiên trong quá trình tạo chương trình tự động.



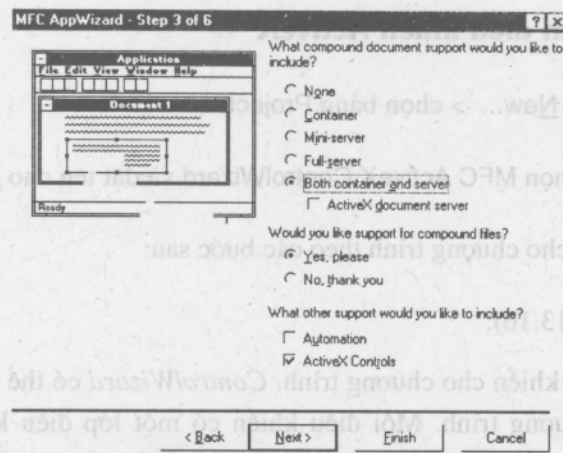
Hình 13.12 Hộp thoại tạo chương trình bằng *AppWizard*

Lập trình viên có thể bỏ qua tất cả các lựa chọn và chấp nhận lựa chọn mặc định bằng việc nhấn phím **Finish**. Khi đó một cửa sổ sẽ được hiển thị như hình 13.13. Cửa sổ này cung cấp cho lập trình viên các thông tin về *project* sẽ được tạo ra như các tên tệp chương trình nguồn, các chức năng của chương trình, thư mục chứa *project*,... Nếu không đồng ý với cách tạo chương trình trên ta có thể hủy bỏ bằng phím **Cancel**, khi đó *AppWizard* sẽ quay lại màn hình trước đó để lập trình viên có thể chọn lại các chức năng mong muốn.



Hình 13.13 Cửa sổ thông báo project tạo bằng *AppWizard*

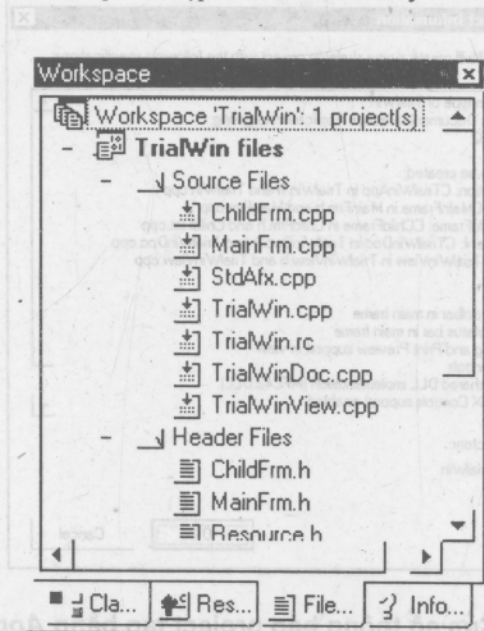
Trong trường hợp ta muốn xây dựng chương trình là trình chứa OLE hay trình dịch vụ OLE thì trong bước thứ nhất hãy chọn *Next* để đi tới bước thứ ba. Hình 13.14 là hộp thoại của bước này.



Hình 13.14 Hộp thoại lựa chọn khả năng của chương trình

Tại đây lập trình viên có thể đặt lựa chọn cho chương trình có thể là chương trình chứa OLE, trình dịch vụ OLE tối thiểu, trình dịch vụ OLE đầy đủ hay vừa là trình chứa OLE vừa là trình dịch vụ OLE.

Khi đồng ý với cách tạo chương trình trên (nhấn phím OK), chương trình *AppWizard* tạo ra các tệp chương trình nguồn cần thiết và sau đó quay trở lại cửa sổ chính. Bây giờ nếu khai triển bảng *FileView* trong cửa sổ *project workspace*, lập trình viên sẽ thấy như trong hình 13.15.



Hình 13.15 Cửa sổ FileView

13.7.2 Tạo chương trình điều khiển ActiveX

1. Chọn chức năng **File** -> **New...** -> chọn bảng **Projects**.
2. Trong bảng **Projects**, chọn **MFC ActiveX ControlWizard** và đặt tên cho *project*.
3. Xác định các lựa chọn cho chương trình theo các bước sau:

Bước thứ nhất (xem hình 13.16):

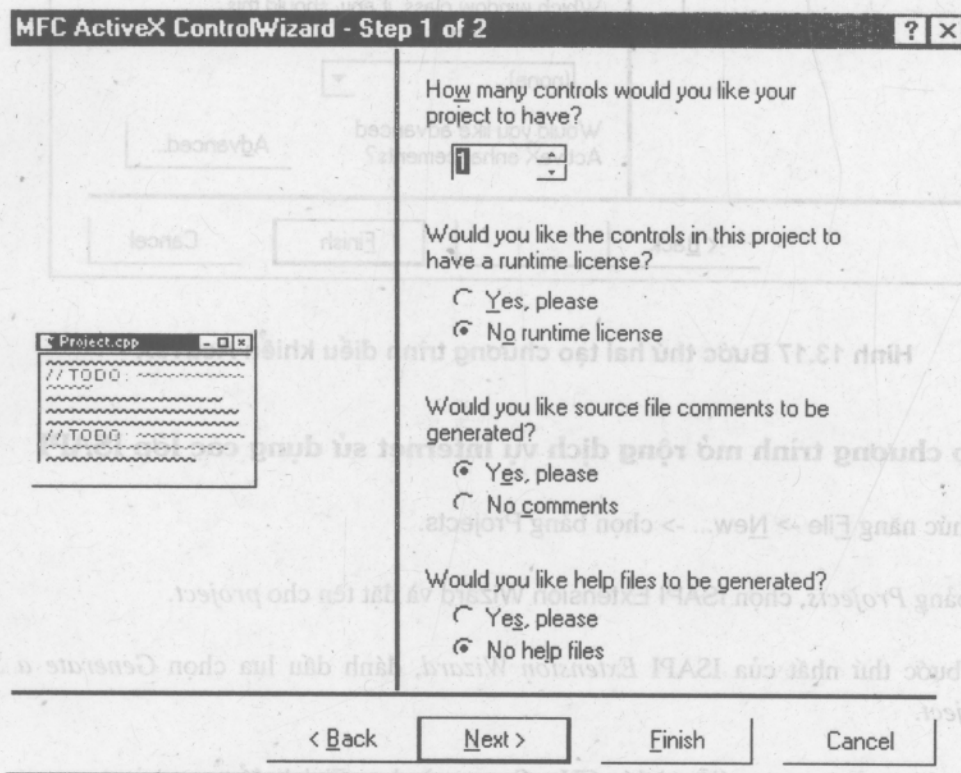
- Chọn số các điều khiển cho chương trình. *ControlWizard* có thể tạo được tối đa 99 điều khiển cho một chương trình. Mỗi điều khiển có một lớp điều khiển và một lớp trang thuộc tính.
- Xác định muốn *ControlWizard* tạo *run-time license* cho các điều khiển không.

- Xác định muốn *ControlWizard* tạo các dòng chú thích hướng dẫn lập trình trong chương trình không.
- Xác định muốn *ControlWizard* tạo các tệp trợ giúp không. Các tệp trợ giúp đòi hỏi phải dùng chương trình *Help compiler* (đi kèm với Visual C++) để dịch.

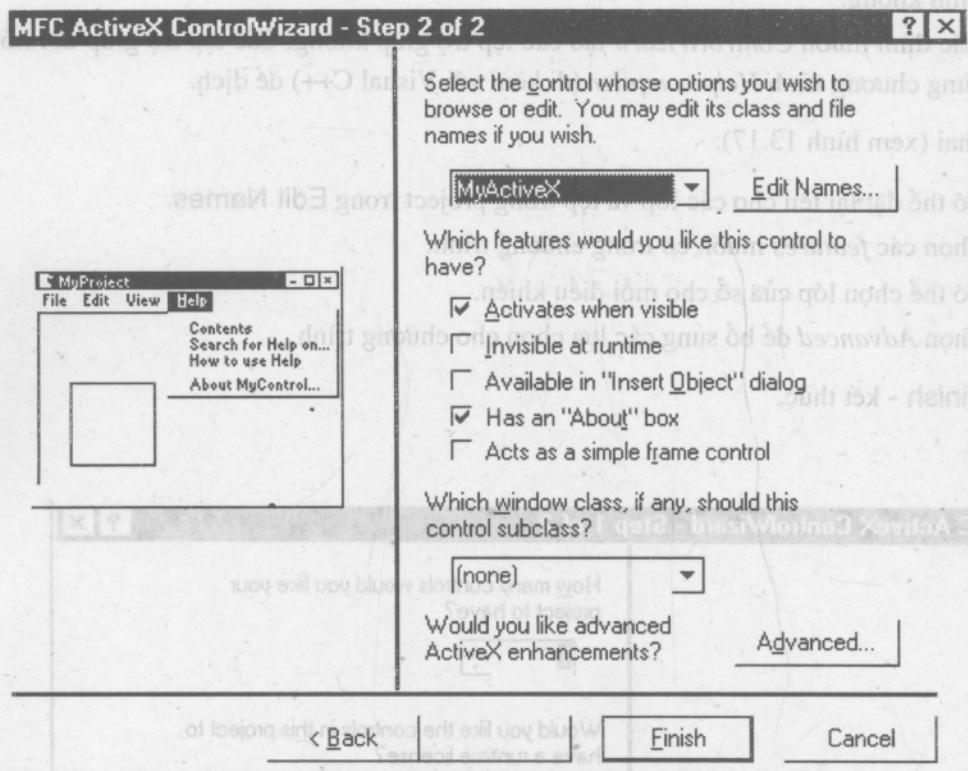
Bước thứ hai (xem hình 13.17):

- Có thể đặt lại tên cho các lớp và tệp trong project trong Edit Names.
- Chọn các *features* muốn có trong chương trình.
- Có thể chọn lớp cửa sổ cho mỗi điều khiển.
- Chọn *Advanced* để bổ sung các lựa chọn cho chương trình.

4. Chọn Finish - kết thúc.



Hình 13.16 Bước thứ nhất tạo chương trình điều khiển ActiveX

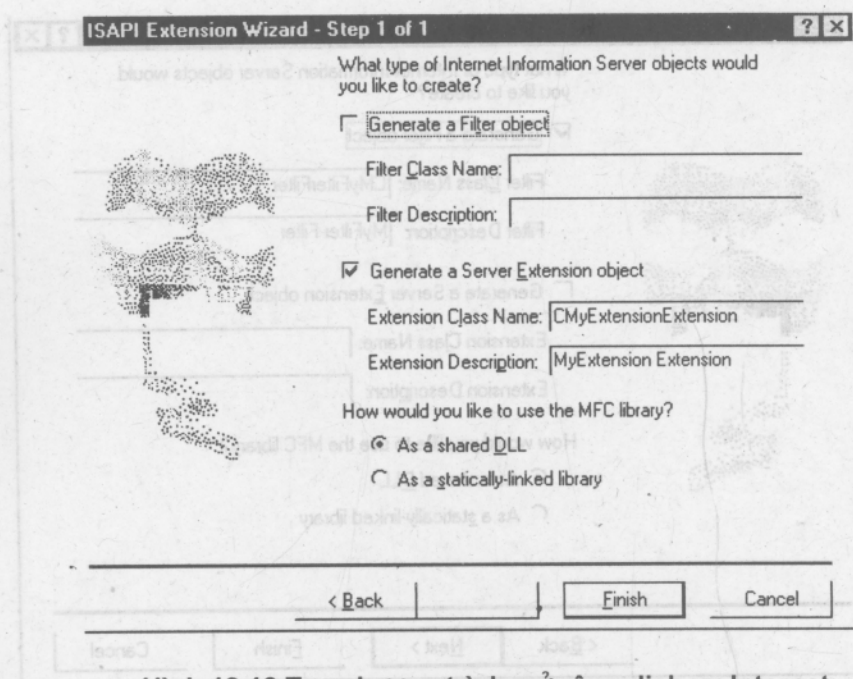


Hình 13.17 Bước thứ hai tạo chương trình điều khiển ActiveX

13.7.3 Tạo chương trình mở rộng dịch vụ Internet sử dụng các lớp ISAPI

1. Chọn chức năng **File -> New...** -> chọn bảng Projects.
2. Trong bảng *Projects*, chọn *ISAPI Extension Wizard* và đặt tên cho *project*.
3. Trong bước thứ nhất của *ISAPI Extension Wizard*, đánh dấu lựa chọn *Generate a Server Extension object*.

Đặt tên cho lớp mở rộng suy diễn từ lớp *CHttpServer* và chọn **Finish** để tạo *project*.



Hình 13.18 Tạo chương trình mở rộng dịch vụ Internet

13.7.4 Tạo chương trình lọc sử dụng các lớp ISAPI

1. Chọn chức năng **File -> New... ->** chọn bảng **Projects**.
2. Trong bảng **Projects**, chọn **ISAPI Extension Wizard** và đặt tên cho *project*.
3. Trong bước thứ nhất của **ISAPI Extension Wizard**, đánh dấu lựa chọn **Generate a Filter object**.
4. Đặt tên cho đối tượng **CHttpFilter**, chọn cách liên kết với các thư viện DLL của MFC theo kiểu tĩnh hay động.
5. Trong bước thứ hai của **ISAPI Extension Wizard**, chọn mức ưu tiên cho chương trình lọc. Mức ưu tiên mặc định (nên dùng) là **Low**.
6. Chọn kiểu liên kết nào tạo ra thông báo cho chương trình lọc. Chương trình lọc có thể tiếp nhận thông báo thông từ cổng được bảo vệ hay cổng tự do hoặc cả hai (mặc định).
7. Chọn thông báo nào sẽ được chương trình lọc xử lý (chương trình lọc gắn với sự kiện nào). Chỉ chọn những thông báo mà chương trình lọc muốn quản lý.
8. Chọn **Finish** để kết thúc.

ISAPI Extension Wizard - Step 1 of 2

What type of Internet Information Server objects would you like to create?

☒ Generate a Filter object

Filter Class Name:

Filter Description:

☐ Generate a Server Extension object

Extension Class Name:

Extension Description:

How would you like to use the MFC library?

☒ As a shared DLL

☐ As a statically-linked library

< Back Next > Finish Cancel

Hình 13.19 Bước thứ nhất tạo chương trình lọc

ISAPI Extension Wizard - Step 2 of 2

What notification priority will your filter have?

☐ High

☐ Medium

☒ Low (Default)

What connection types interest your filter?

☒ Secured port sessions

☒ Nonsecured port sessions

Which notifications will your filter process?

☐ Incoming raw data and headers

☐ Outgoing raw data and headers

☐ Post-preprocessing of the request headers

☐ Client authentication requests

☐ URL mapping requests

☐ Server log writes

☒ End of connection

< Back Finish Cancel

Hình 13.20 Bước thứ hai tạo chương trình lọc

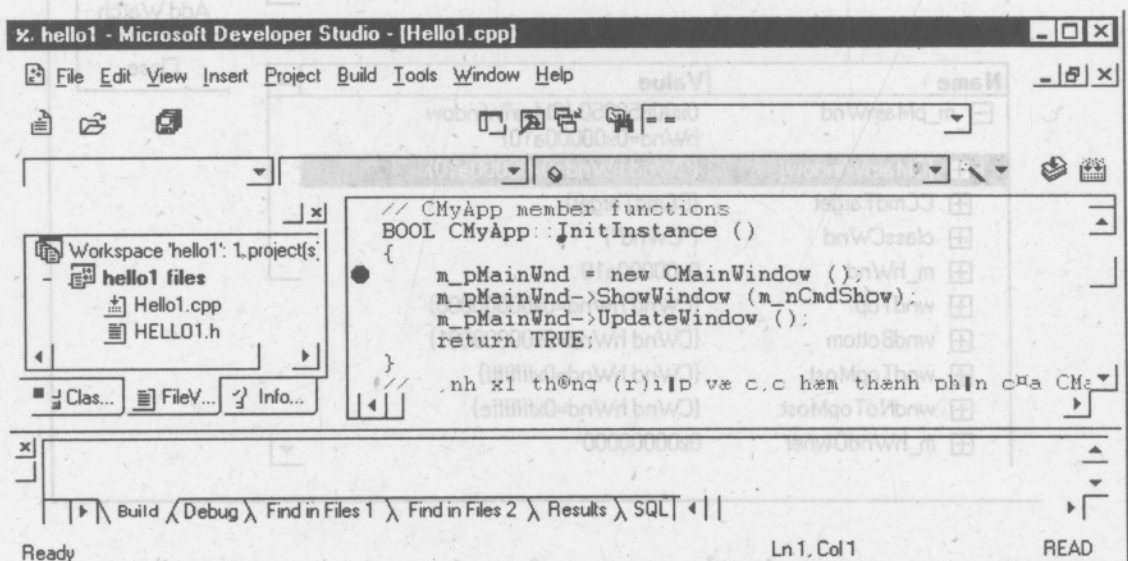
13.8 GỠ RỐI (DEBUG) CHƯƠNG TRÌNH TRONG DEVELOPER STUDIO

Khi xây dựng một chương trình bằng Visual C++ lập trình viên khó có thể tạo ra một chương trình hoàn hảo ngay từ đầu. Chương trình mới tạo ra thường gặp những lỗi khó lường trước, trong trường hợp này lập trình viên thường phải thực hiện chương trình theo từng bước để theo dõi quá trình thực hiện với mục đích tìm ra lỗi. Việc thực hiện chương trình theo từng bước được gọi là quá trình gỡ rối. Để gỡ rối được một chương trình trong *Developer Studio*, lập trình viên phải xây dựng nó ở chế độ gỡ rối như đã hướng dẫn ở phần trên.

Khi xây dựng chương trình ở chế độ gỡ rối, các thông tin gỡ rối sẽ được xen thêm vào chương trình để các tiện ích gỡ rối sử dụng được. Có rất nhiều các tiện ích gỡ rối, chúng ta sẽ xem xét một số tiện ích thông dụng nhất.

13.8.1 Đặt điểm dừng

Việc đầu tiên trước khi lập trình viên thực hiện debug chương trình đó là đặt các điểm dừng (*breakpoints*) trong chương trình. Các điểm dừng cho phép lập trình viên tạm dừng quá trình thực hiện chương trình để kiểm tra giá trị các biến và có thể thay đổi các giá trị này nếu thấy chúng không đúng như mong muốn. Đặt điểm dừng đơn giản chỉ là đặt con trỏ tại câu lệnh nơi muốn tạm dừng trong quá trình thực hiện chương trình và nhấn chuột vào phím *Insert/Remove Breakpoint* hoặc nhấn phím F9.



Hình 13.21 Điểm dừng trong chương trình

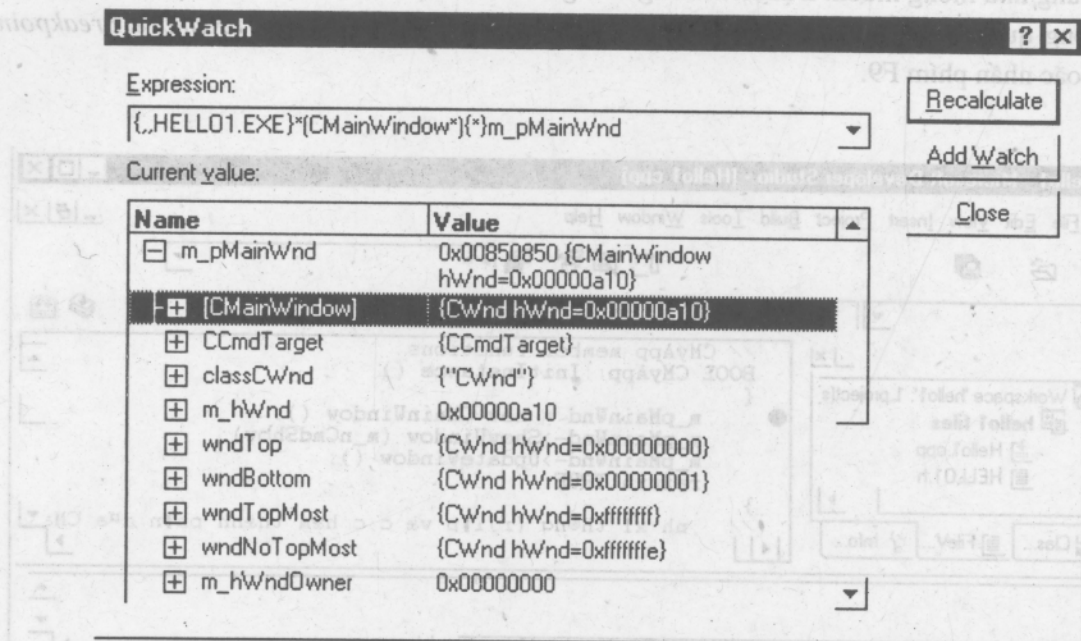
13.8.2 Thực hiện gỡ rối chương trình

Để thực hiện chương trình đến điểm dừng đã đặt ta chọn chức năng **Build | Start Debug | Go** trên thực đơn hoặc ấn phím F5. Sau khi bắt đầu thực hiện chương trình ở chế độ gỡ rối, chức năng **Build** trên thực đơn chính được thay thế bằng chức năng **Debug**.

Để thực hiện chương trình đến điểm dừng tiếp theo ta chọn chức năng **Debug | Go** trên thực đơn hoặc ấn phím F5.

Để thực hiện từng lệnh trong chương trình, ta chọn chức năng **Debug | Step Into** trên thực đơn hoặc ấn phím F11.

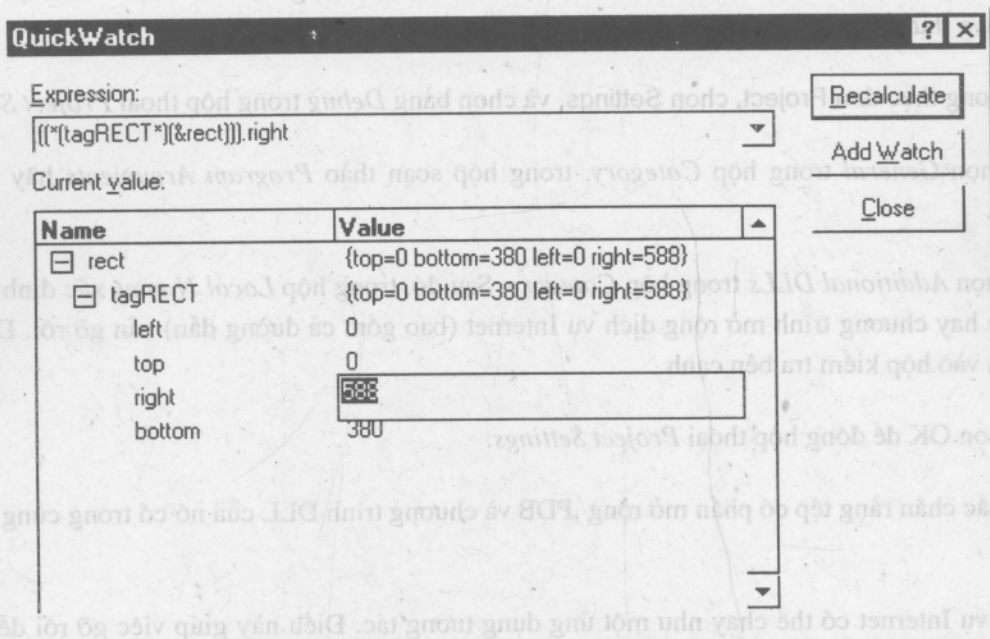
Trong quá trình gỡ rối, để xem nhanh giá trị của một biến, tại điểm dừng ta đưa con trỏ chuột vào biến cần xem trong cửa sổ soạn thảo, một cửa sổ con màu vàng sẽ hiển thị ngay phía dưới cho biết giá trị hiện thời của biến đó. Cách xem này không cho lập trình viên biết giá trị từng thành phần của biến nếu biến đó là biến cấu trúc. Để xem giá trị của một biến một cách chi tiết, hãy đưa con trỏ soạn thảo vào trong tên biến cần xem và sử dụng chức năng **Debug | QuickWatch** hoặc ấn phím Shift+F9, khi đó một cửa sổ thông báo sẽ được hiển thị như hình 13.22.



Hình 13.22 Cửa sổ xem giá trị của biến trong quá trình gỡ rối

Cửa sổ này cũng cho phép lập trình viên thay đổi giá trị của biến hoặc giá trị của một biến thành

phân bằng cách nhấn đúp vào vùng giá trị (Value) của biến cần thay đổi và sửa lại giá trị này như hình 13.23.



Hình 13.23 Thay đổi giá trị của biến trong quá trình gỡ rối

13.8.3 Kết thúc quá trình gỡ rối

Có hai cách để kết thúc quá trình gỡ rối. Cách thứ nhất là cho chương trình kết thúc bình thường bằng các lệnh kết thúc chương trình. Cách thứ hai là kết thúc bằng việc hủy bỏ quá trình gỡ rối thông qua chức năng **Debug | Stop Debugging** trên thực đơn hoặc ấn phím **Shift+F5**.

13.8.4 Gỡ rối cho chương trình lọc và trình mở rộng dịch vụ Internet

Để chạy chương trình lọc hay chương trình mở rộng dịch vụ Internet ở chế độ gỡ rối, chúng ta cần máy chủ tạm dừng các dịch vụ. Chúng ta có thể bắt đầu gỡ rối theo các bước được mô tả sau đây:

1. Dừng các dịch vụ Internet Information Server publishing (WWW, FTP và Gopher) bằng chương trình Internet Information Server Manager hay bằng chương trình Service (biểu tượng Service) trong Windows NT Advanced Server Control Panel.

2. Chạy *Microsoft Developer Studio*, đóng *workspace* đang làm việc bằng lệnh **Close**

Workspace trong thực đơn File.

3. Trong thực đơn File, chọn Open Workspace để mở chương trình INETINFO.EXE. Tập này chứa trong thư mục khi cài đặt *Microsoft Internet Information Server*.

4. Trong thực đơn Project, chọn Settings, và chọn bảng *Debug* trong hộp thoại *Project Settings*.

5. Chọn *General* trong hộp *Category*, trong hộp soạn thảo *Program Arguments* hãy nhập: -e W3Svc.

6. Chọn *Additional DLLs* trong hộp *Category*. Sau đó, trong hộp *Local Names* xác định chương trình lọc hay chương trình mở rộng dịch vụ Internet (bao gồm cả đường dẫn) cần gỡ rối. Đánh dấu lựa chọn vào hộp kiểm tra bên cạnh.

7. Chọn OK để đóng hộp thoại *Project Settings*.

8. Chắc chắn rằng tập có phần mở rộng .PDB và chương trình DLL của nó có trong cùng một thư mục.

Dịch vụ Internet có thể chạy như một ứng dụng tương tác. Điều này giúp việc gỡ rối dễ hơn rất nhiều. Để làm được điều này cần có một thay đổi nhỏ trong hệ thống. Đối với khoản mục của người sử dụng mà máy chủ sẽ chạy với nó, cần bổ sung thêm một số đặc quyền.

Để bổ sung đặc quyền, chạy chương trình Quản lý (User Manager - MUSRMGR.EXE), và chọn *User Rights* từ thực đơn *Policies*. Đánh dấu vào hộp lựa chọn *Show Advanced User Rights*. Sau đó chọn "Act as part of the operating system" từ danh sách *drop-down Right*, và bổ sung khoản mục người sử dụng.

Lập lại quá trình xử lý với "Generate Security Audits". Chắc chắn rằng đã dừng các dịch vụ Internet và chưa chạy chương trình INETINFO.EXE. Log off và log on lại. Sau đó ta có thể nạp dịch vụ Internet với dòng lệnh :

```
INetInfo.exe -e W3Svc
```

Bây giờ đã sẵn sàng để chạy. Trong *Developer Studio*, chọn Build -> Start Debug -> Go. Hành động này sẽ khởi động dịch vụ WWW và việc gỡ rối sẽ được thực hiện. Chúng ta có thể nhận được cảnh báo "Tập không chứa các thông tin gỡ rối", nhưng chúng có thể được bỏ qua.

Khi kết thúc *Microsoft Developer Studio*, chúng ta có thể ghi lại *workspace* cho INETINFO.EXE (thông thường ghi dưới tên tập INETINFO.MDS) để lần sau có thể mở lại để gỡ rối mà không phải đặt lại các lựa chọn.

MỘT SỐ THUẬT NGỮ

ánh xạ thông điệp (message map)

là cơ chế để MFC gán các thông điệp mà ứng dụng nhận được vào các hàm thành phần trong MFC và các lớp phái sinh từ MFC.

cấu tử (constructor)

là hàm thành phần đặc biệt của lớp, có cùng tên với tên của lớp. Cấu tử có nhiệm vụ tạo dựng đối tượng. Khi khai báo đối tượng của lớp thì chúng được gọi để khởi động ban đầu cho các thành phần của đối tượng như cấp phát bộ nhớ, gán giá trị ban đầu cho các thành phần...

chỉ danh (handle)

là biến phân biệt các đối tượng với nhau hay là tham chiếu gián tiếp tới một tài nguyên của hệ điều hành.

đa luồng chạy chương trình (multithread)

là khả năng chạy nhiều *thread* đồng thời của hệ điều hành.

đối tượng (object)

1. Tổng quát: là một *instance* của thực thể chứa dữ liệu và các hàm thao tác trên dữ liệu đó.
2. Trong lập trình hướng đối tượng: đối tượng là thực thể có trạng thái, ứng xử và chỉ danh. Trạng thái của đối tượng bao gồm thuộc tính và giá trị hiện hành của thuộc tính. Ứng xử của đối tượng bao gồm các thao tác trên dữ liệu và chúng có khả năng làm thay đổi trạng thái. Chúng ta sử dụng chỉ danh của đối tượng để phân biệt giữa các đối tượng.

hàm callback (callback function)

là hàm của chương trình ứng dụng nhưng được gọi bởi hệ thống (thí dụ như Windows) khi xuất hiện một sự kiện hay khi “cửa sổ” hoặc “phông chữ” được liệt kê... Thí dụ như các hàm *callback* trong các thủ tục của sổ, các thủ tục hộp thoại, vv. Hàm *callback* cũng được sử dụng để xử lý các giao dịch trao đổi động các dữ liệu.

hàm ảo (virtual function)

là hàm thành phần của lớp cơ sở được khai báo với từ khóa *virtual*. Nếu lớp cơ sở chứa một hàm ảo và lớp phái sinh của nó cũng định nghĩa một hàm tương tự thì hàm của lớp phái sinh sẽ được gọi cho các đối tượng của lớp phái sinh ngay cả khi hàm được gọi bởi con trỏ hay quy chiếu đến lớp cơ sở.

hủy tử (destructor)

là hàm thành phần đặc biệt của lớp, có cùng tên với lớp và có thêm dấu ngã (~). Khi đối tượng bị xóa bỏ thì hàm này tự động được gọi để thực hiện các dọn dẹp như giải phóng vùng nhớ đã được cấp phát ...

giao diện lập trình ứng dụng (Application Programming Interface - API)

là tập các hàm trong thư viện chương trình mà chương trình ứng dụng có thể gọi để thực hiện các dịch vụ mức thấp trong hệ điều hành. Thí dụ, với các máy tính có giao diện đồ họa cho người sử dụng thì một API của chúng có thể thực hiện quản lý các cửa sổ ứng dụng, biểu tượng, thực đơn, hộp thoại ...

giao diện lập trình ứng dụng dịch vụ Internet (Internet Server API - ISAPI)

là tập các hàm được sử dụng để tạo ra trình mở rộng hay trình lọc của Dịch vụ thông tin Internet của Microsoft® (IIS).

giao diện lập trình ứng dụng Windows (Windows API)

là một phần của hệ điều hành Windows. Chúng bao gồm các hàm để chương trình ứng dụng liên lạc với Windows và các hàm dịch vụ khác để tạo ra các chương trình ứng dụng chạy trong môi trường Windows.

instance

Khi hệ điều hành Windows nạp ứng dụng hay thư viện động (DLL) để thực hiện, chúng tạo ra "ngữ cảnh thực hiện". Mỗi ngữ cảnh thực hiện này được gọi là *instance*. Đoạn mã trình ứng dụng hay thư viện động do *instance* thứ nhất nạp xuống sẽ được chia sẻ (đồng thời sử dụng) cho các yêu cầu thực hiện *instance* sau này.

lớp (class)

là định nghĩa cho một kiểu đối tượng. Một lớp mô tả các biến thuộc tính và các phương thức của một đối tượng. Biến thuộc tính được gọi là thành phần dữ liệu, phương thức được gọi là hàm thành phần.

lớp cơ sở (base class)

là lớp từ đó người ta xây dựng các lớp phái sinh kế thừa các thành phần dữ liệu và các hàm thành phần. Lớp cơ sở còn được gọi là lớp cha.

lớp thừa kế

xem lớp phái sinh

lớp phái sinh (derived class)

là lớp kế thừa hàm thành phần hay thành phần dữ liệu từ một lớp khác (lớp cơ sở). Lớp phái sinh còn được gọi là lớp con.

LẬP TRÌNH WINDOWS BẰNG VISUAL C++

In 4.000 bản, khổ 19 × 27cm tại Xưởng in Chi nhánh Công ti Phát triển Công nghệ và Truyền hình – Tp. Hồ Chí Minh. Giấy phép xuất bản số: 62-2005/CXB/6-1684/XBGD/GD do Cục Xuất bản cấp ngày 21 tháng 10 năm 2005. In xong và nộp lưu chiểu quý IV năm 2005.



Giá: 40.600 đ