目 录

无忧书库
http://pcbook.51soft.com

第1章 概述 …… (1)
1.1 Visual C++ 6.0特点 …… (1)
1.2 Visual C++ 6.0安装 …… (1)
1.3 Visual C++ 6.0的组成 …… (8)
1.3.1 可视化的工作平台(Workbench) …… (8)
1.3.2 AppWizard和ControlWizard …… (9)
1.3.3 ClassWizard和WizardBar …… (9)
1.3.4 ActiveX和Component Callery(组件廊) …… (9)
1.3.5 资源编辑器 …… (10)
1.3.6 调试器和诊断工具 …… (10)
1.3.7 Books Online(在线书籍) …… (10)
1.4 Windows编程模式 …… (10)
1.4.1 Windows SDK …… (11)
1.4.2 面象对象编程 …… (11)
1.4.3 MFC简介 …… (12)
1.5 简单的Visual C++ 6.0编程实例 …… (12)
1.5.1 Hello应用程序的说明 …… (12)
1.5.2 生成Hello工程文件 …… (13)
1.5.3 进行可视化编程 …… (18)
1.5.4 给应用程序添加代码 …… (20)
1.6 小结 …… (21)
第2章 Visual C++ 6.0集成环境 …… (23)
2.1 Visual C++ 6.0主窗口 …… (23)
2.2 AppWizard …… (24)
2.3 ClassWizard …… (29)
2.3.1 用Class Wizard创建新类 …… (30)
2.3.2 定义消息处理函数 …… (31)
2.3.3 删除消息处理函数 …… (31)
2.3.4 编辑消息处理函数 …… (32)
2.3.5 覆盖虚拟函数 …… (32)
2.4 可视化编程窗口 …… (32)
2.4.1 工作区窗口 …… (32)
2.4.2 常用控件 …… (33)
2.4.3 控件属性 …… (35)
2.5 集成调试(Debug) …… (36)
2.5.1 Debug菜单工具条 …… (37)
2.5.2 设置断点 …… (38)

2.6 “Edit”菜单 …… (39)
2.7 Box 例程序 …… (42)
2.8 Satcctl 例程序 …… (56)
2.9 小结 …… (65)
第 3 章 标准控件 …… (67)
3.1 静态文本控件 …… (67)
3.1.1 创建及初始化静态文本控件 …… (67)
3.2 列表框和组合框 …… (69)
3.2.1 创建和初始列表框和组合框 …… (69)
3.3 滚动条和滑杆 …… (69)
3.3.1 创建和初始化滚动条和滑块 …… (70)
3.4 公共对话框 …… (70)
3.4.1 颜色对话框 …… (70)
3.4.2 查找替换对话框 …… (71)
3.4.3 字体对话框 …… (72)
3.4.4 打开、保存文件对话框 …… (73)
3.4.5 页面设置对话框 …… (74)
3.4.6 打印对话框 …… (75)
3.5 实例分析 …… (76)
3.5.1 实例 Property …… (76)
3.5.2 Sliderspin 例程 …… (99)
3.6 小结 …… (111)
第 4 章 对话框 …… (112)
4.1 对话框的基础知识 …… (112)
4.1.1 对话框的组成 …… (112)
4.1.2 MFC 对对话框的支持 …… (112)
4.1.3 有模式和无模式对话框 …… (112)
4.1.4 对话框消息处理 …… (113)
4.1.5 对话数据交换和验证 …… (114)
4.2 创建模态对话框 …… (114)
4.2.1 消息对话框 …… (115)
4.3 创建无模式对话框 …… (116)
4.4 实例 …… (116)
4.5 小结 …… (126)
第 5 章 文本和字体 …… (127)
5.1 设备环境类 …… (127)
5.1.1 基类 CDC …… (127)
5.1.2 CPaintDC 类 …… (128)
5.1.3 CClientDC 和 CWindowDC …… (128)
5.1.4 CMetaFileDC 类和元文件 …… (129)

5.2 GDI 对象 …… (129)
5.2.1 GDI 对象的构造和有效期 …… (130)
5.2.2 选择库存 GDI 对象 …… (130)
5.2.3 生成用户化 GDI 对象 …… (131)
5.3 颜色映射 …… (133)
5.3.1 16 色和 256 色显卡 …… (133)
5.3.2 调色板 …… (133)
5.4 字体和正文 …… (134)
5.4.1 获取文字属性 …… (134)
5.4.2 创建字体 …… (135)
5.4.3 绘制正文 …… (135)
5.4.4 实例 Format …… (136)
5.5 小结 …… (147)
第 6 章 图形 …… (148)
6.1 GDI 位图和 DIB 位图 …… (148)
6.2 使用 GDI 位图 …… (149)
6.2.1 生成位图 …… (149)
6.2.2 显示和处理位图 …… (150)
6.3 使用 DIB 位图 …… (151)
6.3.1 BMP 文件的组成 …… (152)
6.4 使用 GDI DIB 位图的两个例子 …… (153)
6.4.1 Gdiball 实例 …… (153)
6.4.2 Opnbmp 实例 …… (163)
6.5 小结 …… (174)
第 7 章 菜单 …… (175)
7.1 创建菜单 …… (175)
7.1.1 动态创建菜单 …… (175)
7.2 快捷键的使用 …… (176)
7.2.1 建立快捷键 …… (176)
7.2.2 装入快捷键 …… (177)
7.3 命令处理 …… (177)
7.3.1 菜单消息的处理路径 …… (178)
7.3.2 处理两种菜单消息 …… (178)
7.3.3 处理一组菜单命令 …… (179)
7.4 菜单项的允许和禁止 …… (180)
7.5 实例 Menudemo 分析 …… (180)
7.6 小结 …… (199)
第 8 章 工具栏和状态栏 …… (200)
8.1 控件栏 …… (200)
8.2 工具栏的创建和控制 …… (200)

8.2.1 工具栏的创建 …… (201)
8.2.2 控制工具栏 …… (202)
8.3 状态栏的创建与控制 …… (204)
8.3.1 创建状态栏(CStatusBar) …… (204)
8.3.2 获得对状态栏的控制 …… (205)
8.3.3 管理消息行和状态指示器 …… (206)
8.3.4 tolstubar 例程 …… (206)
8.4 小结 …… (217)
第 9 章 文档和视 …… (218)
9.1 文档和视概述 …… (218)
9.1.1 使用文档/视窗结构的意义 …… (218)
9.1.2 SDI 和 MDI …… (218)
9.1.3 文档、视和框架 …… (219)
9.1.4 MFC 各对象的关系 …… (220)
9.2 文档和视的详述 …… (220)
9.2.1 文档模板 …… (220)
9.2.2 文档模板的创建 …… (221)
9.2.3 文档和视的创建 …… (221)
9.2.4 文档和视之间的相互作用函数 …… (221)
9.3 实例 Slg 分析 …… (223)
9.4 小结 …… (238)
第 10 章 单文档界面 …… (239)
10.1 序列化 …… (239)
10.1.1 创建可序列化的类 …… (239)
10.1.2 序列化对象 …… (241)
10.2 SDI 简介 …… (242)
10.2.1 创建空文档 …… (242)
10.2.2 打开文件 …… (243)
10.2.3 删除文档内容 …… (244)
10.2.4 保存文件 …… (244)
10.3 小结 …… (244)
第 11 章 多文档界面 …… (245)
11.1 MDI 简介 …… (245)
11.1.1 MDI 应用程序的组成 …… (245)
11.1.2 主框架和文档模板 …… (245)
11.1.3 初始化及清理文档和视 …… (246)
11.2 实例 mdi …… (247)
11.3 小结 …… (261)
第 12 章 切分窗口 …… (262)
12.1 切分窗口 …… (262)

12.2 多视选择 …… (262)
12.3 动态切分窗口 …… (263)
12.4 静态切分窗口 …… (264)
12.5 实例 Dsplitter …… (265)
12.6 实例 Ssplitter …… (281)
12.7 小结 …… (288)
第 13 章 打印 …… (289)
13.1 打印 …… (289)
13.1.1 缺省基本打印 …… (289)
13.1.2 实现高级打印 …… (290)
13.2 打印预览 …… (291)
13.2.1 打印预览过程 …… (292)
13.2.2 支持打印预览 …… (292)
13.2.3 高级打印预览 …… (292)
13.3 小结 …… (293)
第 14 章 上下文相关帮助 …… (294)
14.1 上下文相关帮助 …… (294)
14.1.1 帮助文档 …… (296)
14.1.2 App Wizard 对帮助的支持 …… (296)
14.1.3 常见帮助类型 …… (298)
14.2 帮助的相关文件 …… (300)
14.2.3 目录(Content)文件(.CNT) …… (304)
14.2.4 配置文件(.gid) …… (305)
14.3 在自己的应用程序中加入帮助功能 …… (305)
14.3.1 消息映射支持 …… (305)
14.3.2 F1 键和 Shift + F1 键的处理 …… (307)
14.3.3 调用 WinHelp 函数 …… (308)
14.4 小结 …… (309)
第 15 章 数据库 …… (310)
15.1 数据库管理 …… (310)
15.2 SQL …… (310)
15.2.1 SQL 的对象 …… (311)
15.2.2 SQL 的 Select 查询语句 …… (311)
15.3 ODBC …… (316)
15.3.1 ODBC 的部件 …… (316)
15.3.2 安装驱动程序和数据源 …… (318)
15.4 数据访问对象(DAO) …… (320)
15.5 DLE DB …… (322)
15.6 ActiveX 数据对象(ADO) …… (323)
15.7 MFC 数据库类 …… (323)

15.7.1 CDatabase 类 …… (323)
15.7.2 CRecordSet 类 …… (325)
15.7.3 刷新记录 …… (330)
15.7.4 关闭记录集 …… (330)
15.7.5 在记录集中移动 …… (330)
15.7.6 改变记录集中的数据 …… (331)
15.7.7 CRecordView 类 …… (334)
15.7.8 用 AppWizard 创建数据库应用程序 …… (335)
15.8 小结 …… (336)
第 16 章 ActiveX 控件 …… (337)
16.1 OLE 简介 …… (337)
16.1.1 组件对象模型 …… (338)
16.1.2 结构化存储 …… (340)
16.1.3 Monikers …… (341)
16.1.4 统一数据传输(UDT) …… (342)
16.1.5 OLE 文档 …… (343)
16.1.6 OLE 自动化 …… (343)
16.1.7 OLE 控件 …… (344)
16.2 ActiveX 文档 …… (344)
16.2.1 ActiveX 控件 …… (344)
16.3 COM …… (345)
16.3.1 Internet Monikers …… (346)
16.4 ActiveX 技术 …… (346)
16.4.1 ActiveX 超级链接 …… (346)
16.4.2 ActiveX 协议 …… (346)
16.4.3 ActiveX 服务器扩展 …… (347)
16.4.4 ActiveX Script …… (347)
16.4.5 代码签名 …… (347)
16.4.6 HTML 扩展 …… (347)
16.4.7 ActiveMovie …… (347)
16.4.8 ActiveX 控件的储备事件 …… (348)
16.4.9 ActiveX 控件的自定义事件 …… (350)
16.5 ActiveX 控件的方法 …… (352)
16.5.1 ActiveX 控件的方法 …… (352)
16.5.2 ActiveX 控件的储备方法 …… (353)
16.5.3 ActiveX 控件的自定义方法 …… (354)
16.6 ActiveX 控件的属性 …… (355)
16.6.1 ActiveX 控件的属性 …… (355)
16.6.2 ActiveX 控件的储备属性 …… (356)
16.6.3 ActiveX 控件的自定义属性 …… (358)

16.6.4 ActiveX 控件的高级属性 …… (360)
16.6.5 访问环境属性 …… (361)
16.7 通过 ActiveX ControlWizard 创建控件 …… (362)
16.7.1 使用 ActiveX ControlWizard 的步骤 …… (362)
16.7.2 在 ActiveX 控件文件中增加代码 …… (363)
16.8 ActiveX 控件的属性页 …… (363)
16.8.1 ActiveX 控件的属性页 …… (363)
16.8.2 增加用户属性页 …… (365)
16.8.3 使用储备属性页 …… (366)
16.9 增加正文敏感帮助 …… (367)
16.10 ActiveX 控件中的字体 …… (369)
16.10.1 字体储备属性 …… (369)
16.10.2 字体自定义属性 …… (370)
16.11 ActiveX 控件中的图片 …… (374)
16.11.1 自定义图片属性 …… (374)
16.11.2 实现自定义的图片属性 …… (374)
16.12 小结 …… (376)
第 17 章 动态链接库(DLL) …… (377)
17.1 动态库的基础知识 …… (377)
17.1.1 DLL 的概述 …… (377)
17.1.2 传统的 DLL …… (378)
17.1.3 MFC 库 …… (378)
17.2 MFC 库的动态链接库 …… (379)
17.2.1 MFC 库的动态链接库的命令 …… (379)
17.2.2 MFC 常规型 DLL …… (380)
17.2.3 MFC 扩展型 DLL …… (380)
17.3 创建动态链接库 …… (382)
17.3.1 DLL 的入口函数 DLLMain() …… (382)
17.3.2 创建 DLL …… (383)
17.4 加载 DLL …… (385)
17.5 为 DLL 创建接口 …… (387)
17.6 调试 DLL …… (388)
17.7 实例分析 …… (389)
17.7.1 Testdyn 例程 …… (389)
17.7.2 usemed 例程 …… (404)
17.8 小结 …… (413)
附录 1 MFC 消息映射函数 …… (415)
附录 2 C++ 教程 …… (420)

第1章 概 述

随着计算机多媒体技术及图形图像技术的蓬勃发展，可视化编程得到了广泛重视，越来越多的计算机专业人员和非专业人员都开始研究并应用可视化技术，要支持可视化编程，通常需要相应的可视化开发环境，Visual C++就是Microsoft公司推出的支持可视化编程的集成开发环境。

1.1 Visual C++ 6.0特点

Visual C++ 6.0是微软公司在多年使用、不断改进的基础上推出的用于支持Win95以上平台应用程序(Applications)、服务(Service)和控件(Control)的开放环境。它具有以下特点：

1)开放环境Visual Studio由一套集成工具组成，用于开发Win32(Win95/98或Windows NT)环境下运行的应用程序。

2)提供功能强大的向导工具(MFC AppWizard、ClassWizard)。

3)Developer Studio的项目工作区的形式组织元件、项目及项目等配置。

4)MFC类库支持多线程应用程序开发。

5)具有Windows Socket和Mapi支持，可以与网络及Email连接。

6)最快的集成数据库访问，允许用户适应强有力的数据库应用程序。

7)具有强有力的Internet支持。

8)对OLE提供强有力支持。

1.2 Visual C++ 6.0安装

使用Visual C++ 6.0开发程序之前，必须先将Visual C++ 6.0和Developer Studio安装到计算机上。安装Visual C++ 6.0和Developer Studio需要有IE4.01以上的浏览器，读者可以自己预先安装，这里我们省略安装IE4.01的过程。可以使用安装程序进行新的安装或更新已有的安装。如果是更新安装，安装程序在安装过程中仅仅复制新的或更新过的文件，此外安装程序还可以根据配置情况对安装进行裁剪。

在Windows 98中安装Visual C++ 6.0企业版的过程如下：

1)打开机器启动Win98。

2)将Visval C++ 6.0系统光盘插入CD-ROM中。

3)单击“开始”按钮，选择“运行”命令，弹出“运行”对话框(如图1.1所示)，在“打开”文本框键入以下命令“E:\setup.exe”。



图1.1

4)单击“确定”按钮，安装程序显示对话框如图1.2所示，单击“Next”按钮，弹出对话框(如图1.3所示)。

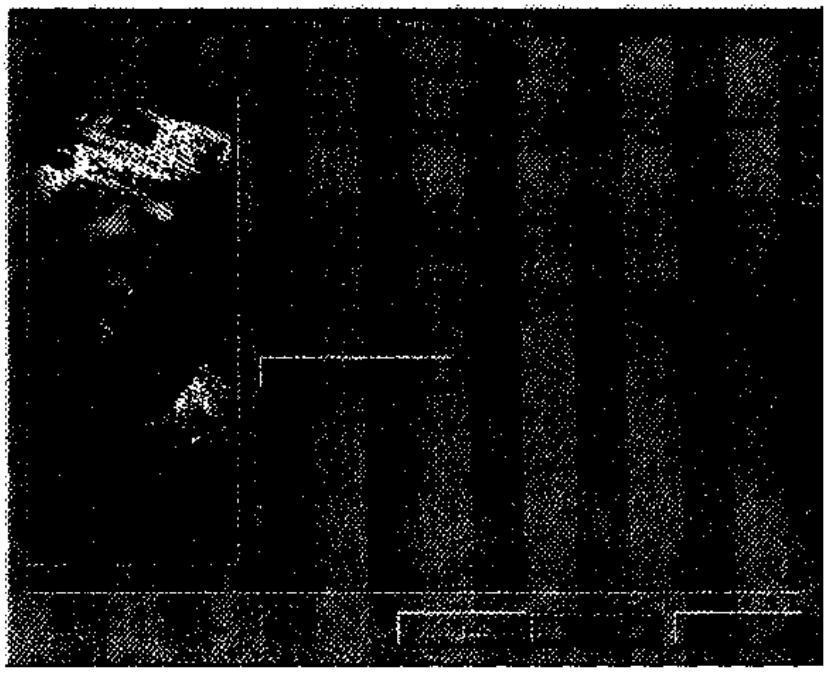


图 1.2　安装程序图示 2

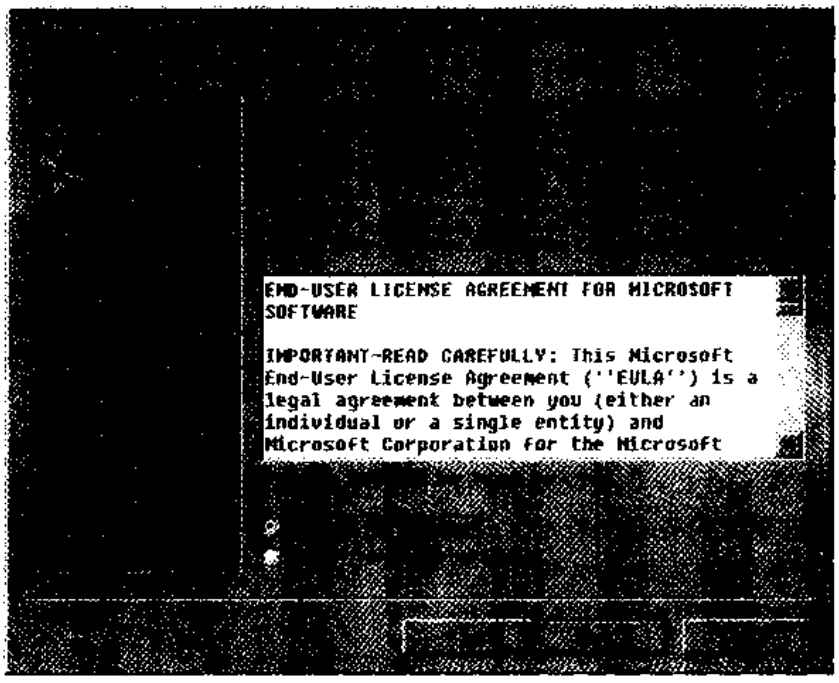


图 1.3　安装程序图示 3

5)选择"I accept the agreement"选项后,单击"Next"按钮,弹出对话框如图 1.4 所示,让你输入 CD 号和相关项目,你可以按图中输入。单击"Next"按钮,则弹出对话框如图 1.5 所示。

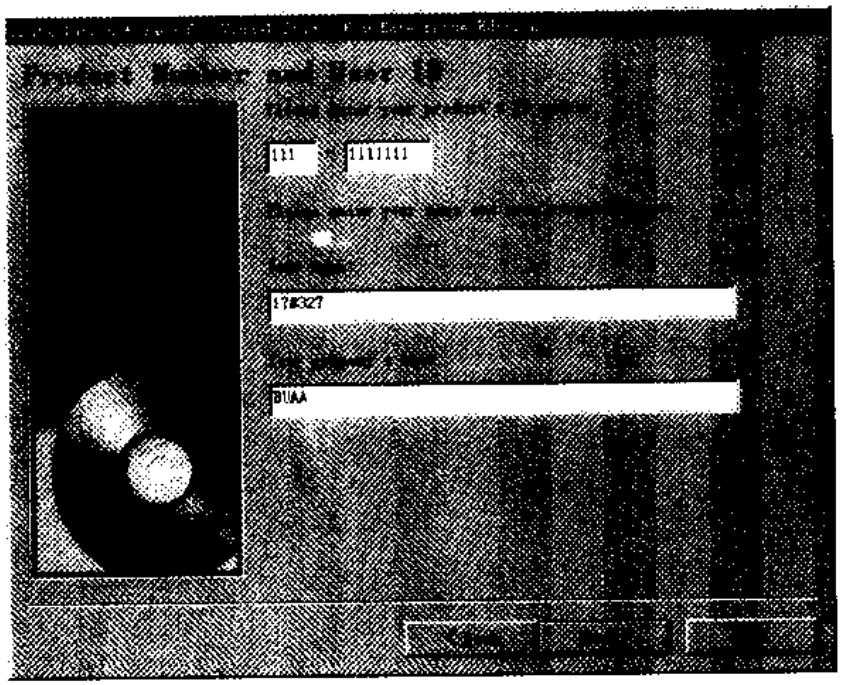


图 1.4　安装程序图示 4

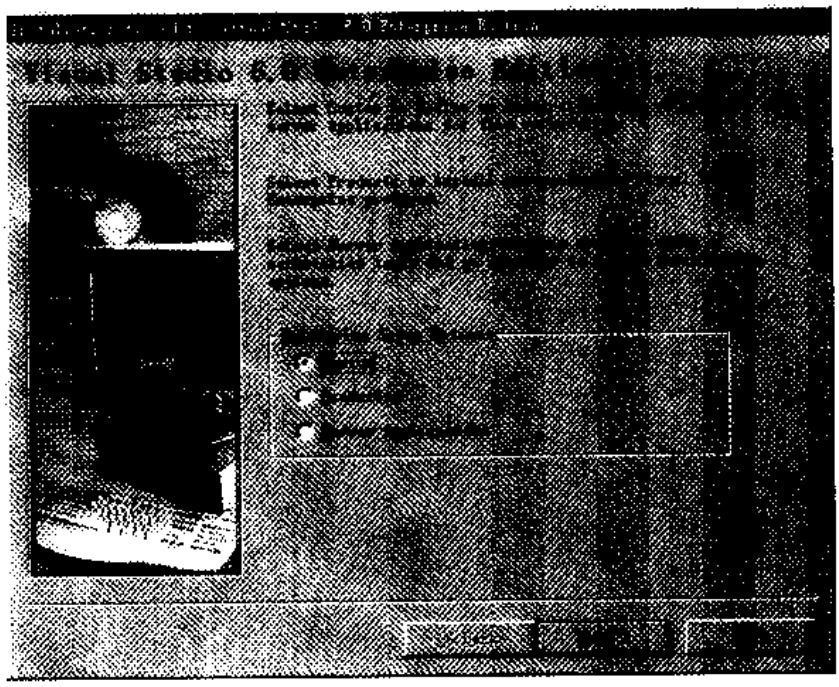


图 1.5　安装程序图示 5

6)单击"Next"按钮,弹出如图 1.6 所示的安装路径对话框。缺省时,Visual C++ 6.0 安装在 C:\Program Files\Dev Studio\vc 中,而 Visual Studio 安装在 C:\Program Files\DevStudio 中。如果要改变安装路径,单击"Browse"按钮,弹出"Change Directory"对话框,从中可以对安装路径进行更改,更改后单击"OK",返回原对话框。完成设置后单击"Next"按钮,弹出对话框如图 1.7 所示。

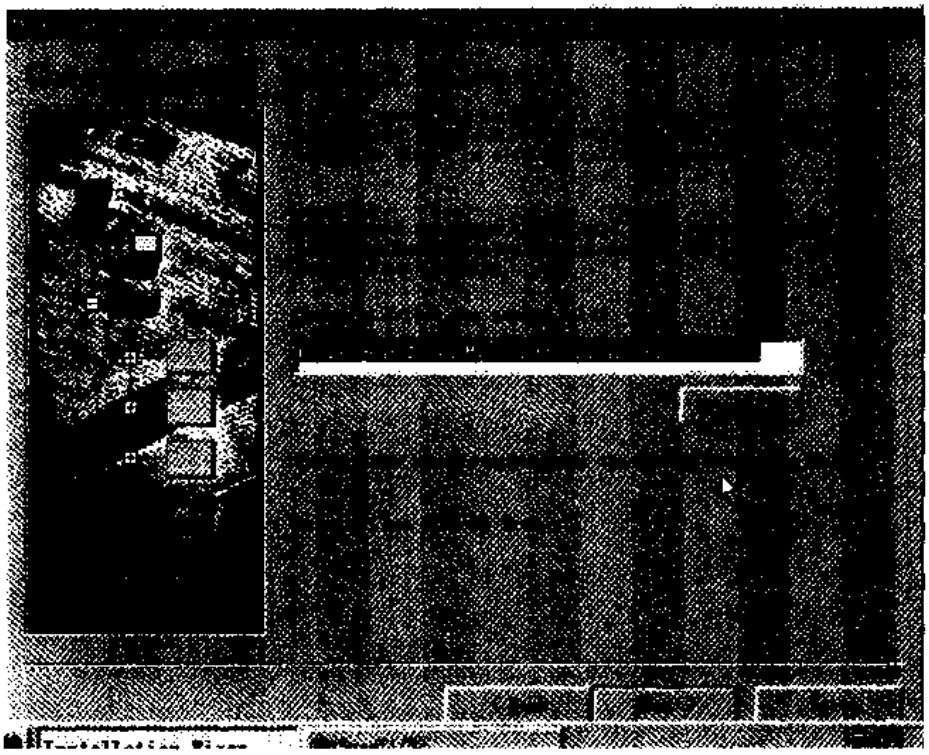


图 1.6　安装程序图示 6

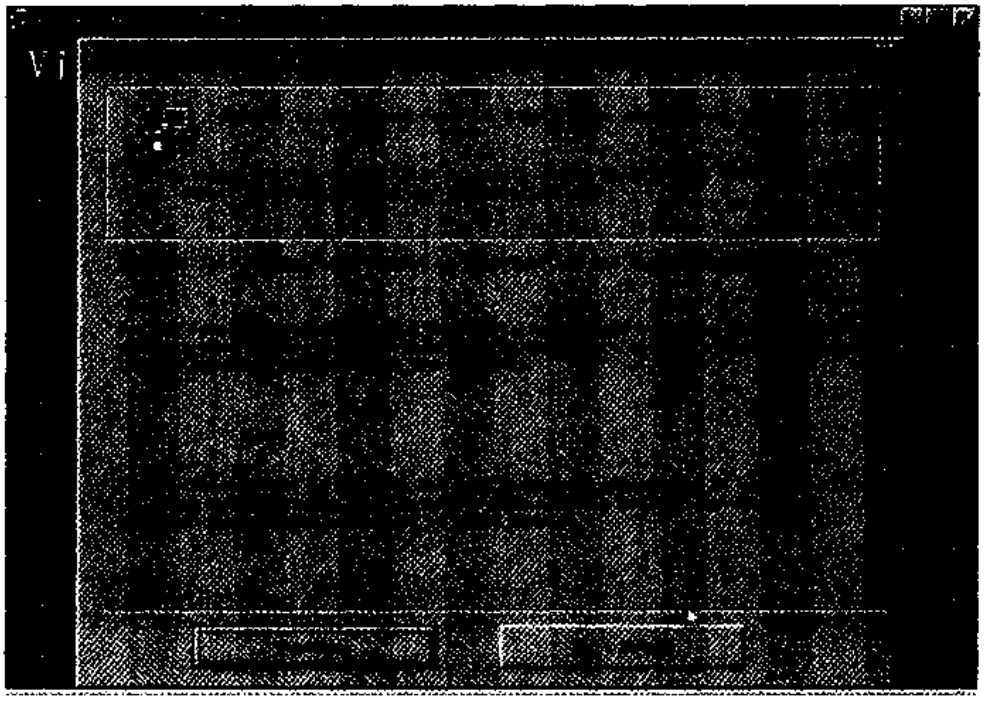


图 1.7　安装程序图示 7

单击“Continue”按钮，弹出对话框如图 1.8 所示，再单击“OK”，弹出对话框如图 1.9 所示，按提示输入后单击“OK”，弹出对话框如图 1.10 所示。



图 1.8　安装程序图示 8

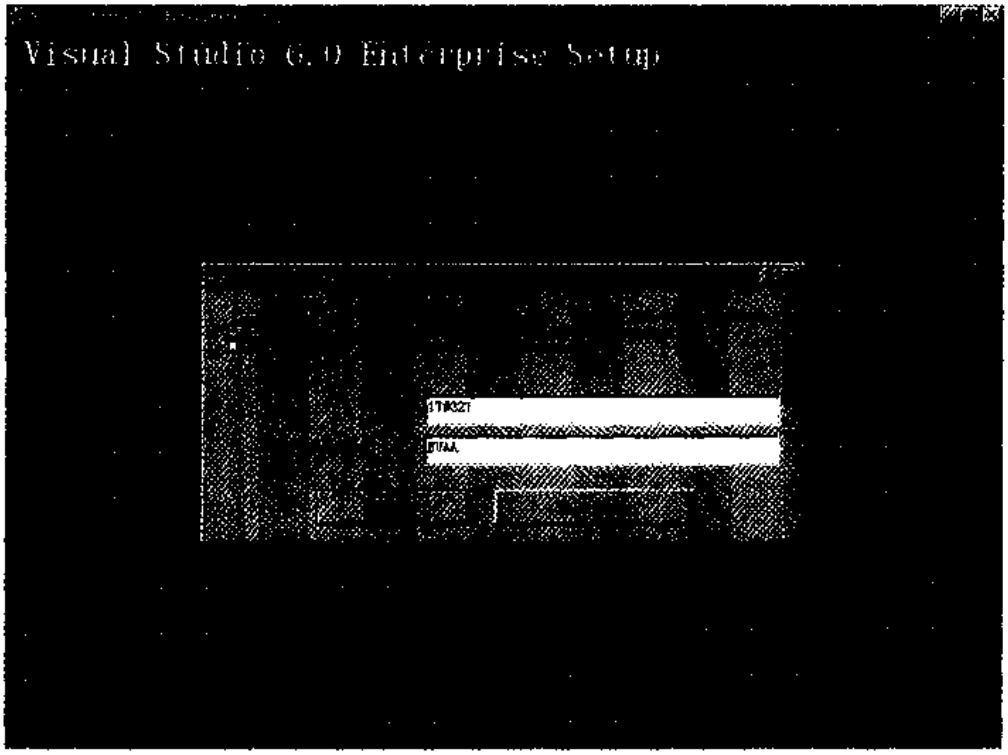


图 1.9　安装程序图示 9

确认无误后按“OK”,出现图 1.11 所示的对话框,在此对话框中读者可单击“Change folder”修改 folder 路径,修改后单击带有“?”的按钮,弹出对话框如图 1.12 所示。

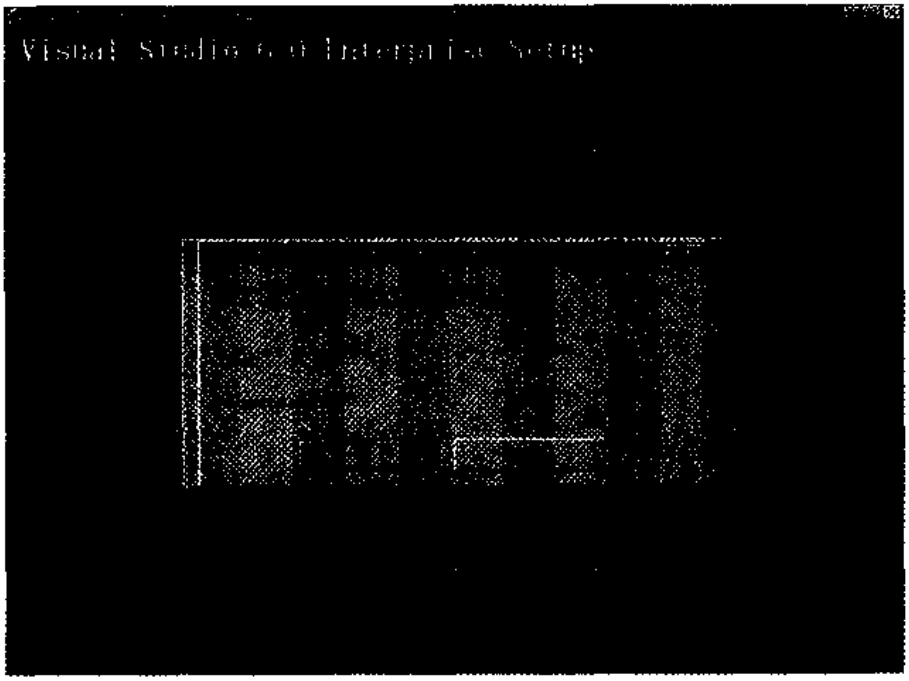


图 1.10　安装程序图示 10

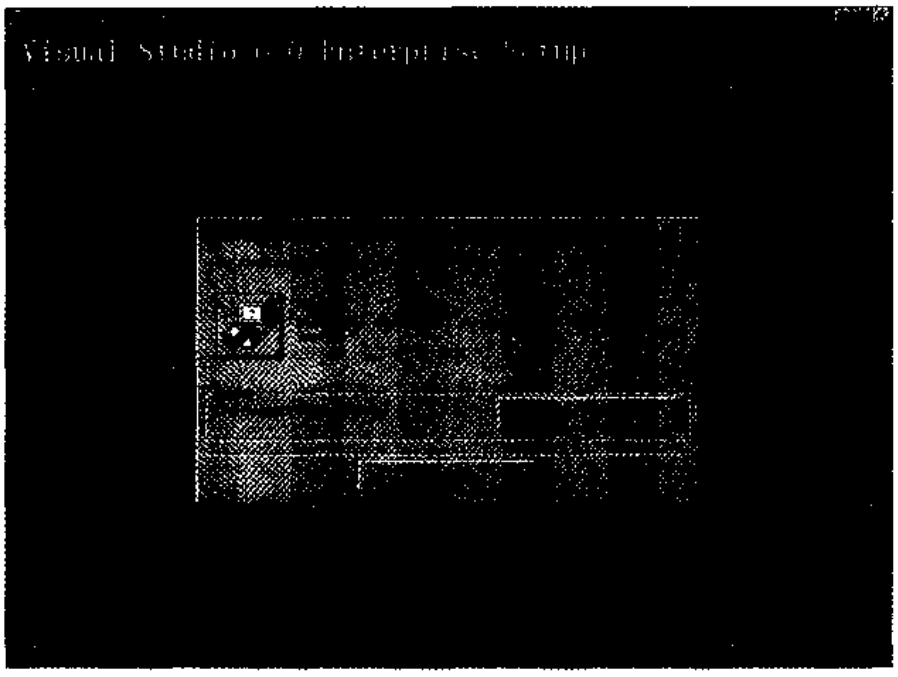


图 1.11　安装程序图示 11

7)该对话框为安装选择对话框,从中可以对需要安装部件进行设置。设置完毕单击"Continue",弹出如图 1.13 所示的对话框,显示安装进程。待安装完后弹出一如图 1.14 所示的对话框,显示系统正在更新。更新完成后弹出一如图 1.15 所示的对话框。至此安装顺利完成。

8)完成安装后,重新启动 Windows 98。

至此,Visual C++ 6.0 整个安装过程完成,需 15 分钟左右。

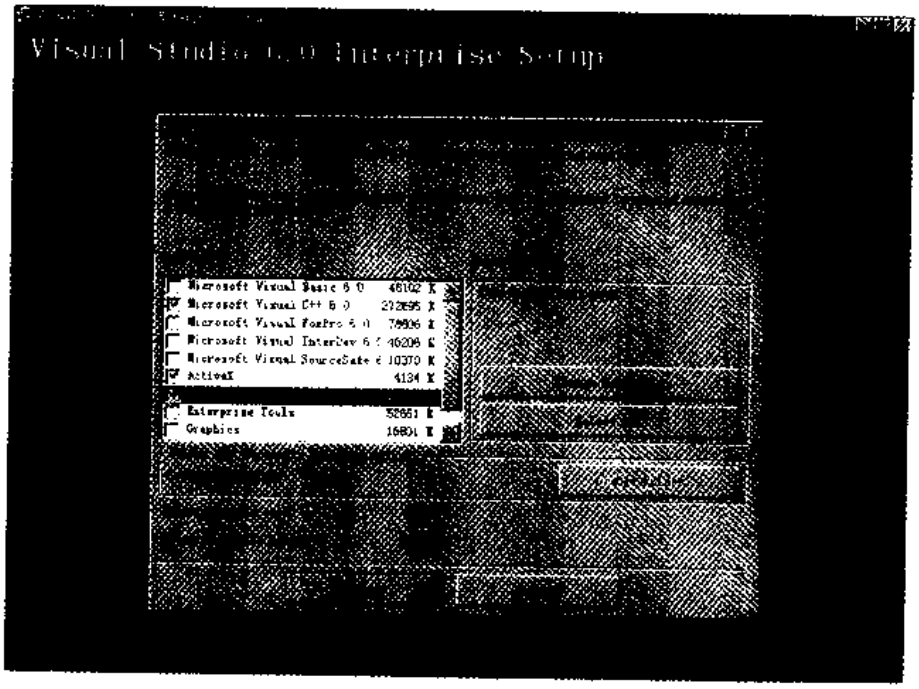


图 1.12　安装程序图示 12

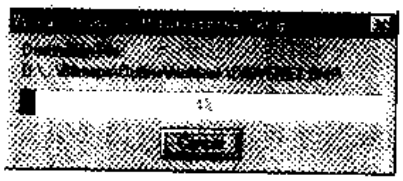


图 1.13　安装程序图示 13

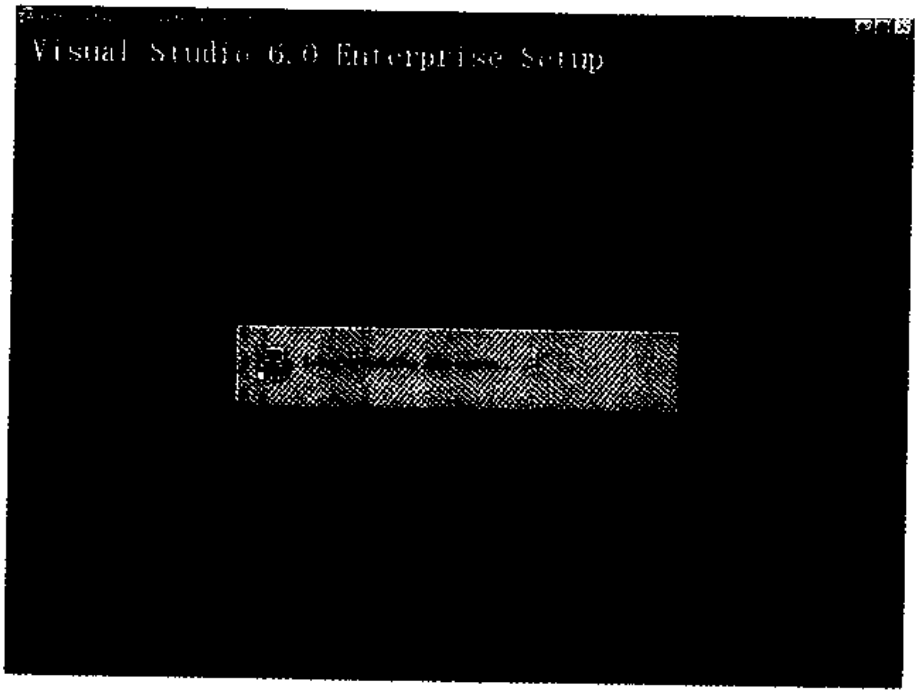


图 1.14　安装程序图示 14

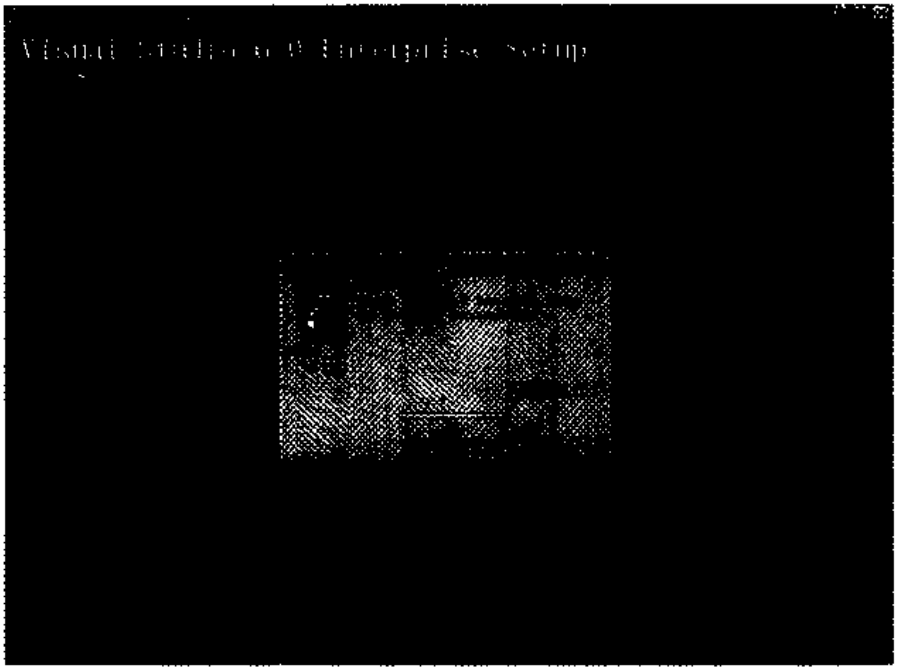


图 1.15　安装程序图 15

1.3　Visual C++ 6.0 的组成

在 Visual C++ 编程的过程中，虽然可以直接调用 Win32 函数，但程序员绝大部分的时间花在处理 MFC 上，如 ClassWizard 之间的关系以及程序调试上。如果在学习 Visual C++ 的过程中有一本类库参考之类的书，会给你提供许多帮助。初次步入 Visual C++ 6.0 的世界，的确觉得这是一个非常庞大的体系，在能够自己动手编写一些程序之前，很有必要弄明白它的各个组成部分的用途，虽然有时候并不需要用到每一部分。下面就 Visual C++ 6.0 的各个方面介绍其组成，使读者对它有个总体的印象。

1.3.1　可视化的工作平台(Workbench)

工作平台也就是开发环境。不同于有些编程工具，在编辑了一个源程序后，要通过键盘敲打有关的命令进行编译、调试。Visual C++ 所具有的可视化交互的集成开发环境是一个智能化程序的构造工厂，它会自动生成许多基本的支持文件，大大减轻程序员的重复性工作，而这些工作的确会花费我们很多精力。在生成一个程序(我们一直如此称呼，实际上是生成了一个包含许许多多文件的工程文件)之前，工作平台自动创建一个缺省的工作空间(Workspace)，只要不退出这个工作空间，以后所要操作的工程文件或别的文件都是放在这个特殊的空间里面。在 Visual C++ 4.0 之前的版本，记录工作空间所有信息的文件各后缀是 mak，在 Visual C++ 4.0 之后，后缀变为 mdp。因此只要双击带有 mdp 或 mak 后缀的文件名，就会打开 Visual C++ 环境，并装载构成该工作空间的所有文件。通常这些文件都在被 Visual C++ 自动生成之后放在以工程名命名的同一目录下。同时这个目录除包括一些主要的文件外，还有一个特殊的子目录，子目录根据编译环境的设置取名为 Debug 或 Release，里面放有编译之后生成的目标代码(obj)文件和可执行(exe)文件。值得一提的还有工作平台组成之一的文本编辑器，给人印象最深的是它会根据 C/C++ 的语法自动调整程序代码的格式，这在编写带有许多

循环语句程序的时候可以节省大量的时间。它还可通过有关菜单项的设置，用不同的颜色来加亮 C/C++ 的关键字，给人一目了然的感觉。

1.3.2 AppWizard 和 ControlWizard

为了节省制作人员的时间，Microsoft 在许多软件的模块中都添加了 Wizard(向导)这一功能组件。这些向导对那些高水平的人员可能并没多大的用处，但对一般的使用者，它的作用是显而易见的。向导正如其名，它会一步一步地带领你去选择它每一个页面所提供的选项。在一些关键的基本信息得到确认之后，它会根据这些信息构造出你所要制作的东西的基本框架。

AppWizard 用来创建基于 MFC 的 Windows 应用程序。它的功能非常强大，使用 AppWizard 可快速产生具有下面特征的应用程序：

- 单文档、多文档或基于对话框的应用程序。
- OLE 支持和数据库(ODBC 和 DAO)支持。
- 工具条、状态条、上下文敏感帮助和三维界面。
- 文件的内部功能，如打开、打印命令等。
- 窗口边框风格的控制。

当你按下 AppWizard 的最后一个确定键之后，它就构造好了 Windows 应用程序必要的开始文件，这些文件包括源文件、头文件、资源文件、工程文件等等。

ControlWizard 是一个定制的 OLE AppWizard，可用来创建 OLE 控件项目的框架。同 AppWizard 一样，它产生的许多初始文件都保持对接下来要介绍的 ClassWizard 的兼容性，也就是说它所产生的初始文件中的类、事件、属性和方法都及时记录在 ClassWizard 中，在以后的编程中，通过调整、增添 ClassWizard 中的项来对源程序代码做可视化更改。

1.3.3 ClassWizard 和 WizardBar

ClassWizard 是专门用来管理事件控制消息的工具，它提供了一种快捷的方式向工程中添加有关各界面元素的消息处理代码。如果不增加这样的代码，编译后的应用程序就会根据缺省设置来响应消息事件，而这些缺省信息都隐藏在 MFC 之中。ClassWizard 通过存取一个后缀为 CLW 的文件来保持与源代码的同步，随着版本的不断提高，Visual C++ 6.0 的跟踪能力已达到相当高的程度，这样可减少过去手工添加例行的数据交换和数据有效化的代码，非常省事。

WizardBar 又进一步简化了 ClassWizard 的一些经常性的任务，可以用它快速选择一个类组件，并修改相关虚拟函数、Windows 消息等。

1.3.4 ActiveX 和 Component Gallery(组件廊)

重用代码的思想在这里得到了极为明显的体现。随着 ActiveX 技术风靡全球，程序员只需用已有的控件来组装自己的程序，而不必耗费大量的精力来一句一句地编写源代码。提到 ActiveX 有必要多交待几句。ActiveX 是微软在 OLE 的基础上开发出来的与 Sun 公司的 JavaBean竞争的技术。ActiveX 的语言无关性体现在 ActiveX 控件可用不同的语言编写，如 Basic、C/C++、Java 等。在 Windows 程序设计的早期，微软采用相对简单的动态数据交换(DDE)技术来实现多个程序间的通信。这种技术随后发展到对象链接和嵌入(OLE)技术的 1.0 版本，它允许在程序之间链接和嵌入对象数据来创建混合文档。OLE 2.0 大大丰富了

OLE 1.0 的内容，其内容不再是简单的对象链接和嵌入，它建立了一个称为组件对象模型(Component object Model-COM)的新规范。ActiveX 也是用 COM 的一种技术，它使各种软件组件加入到运行在桌面或 Internet 上的应用程序之中。组件廊包含可重用代码，如 OLE 控件、具有相关资源的可重用C++类或第三方开发商创建的组件。组件廊由两部分组成，Developer Studio Components 和 Registered ActiveX Controls，可以很容易地创建或增加组件到组件廊中，也可以创建自己的组件类型，以及将组件从一个类型移动到另一个类型。

1.3.5 资源编辑器

Visual C++ 6.0 提供的资源编程器可以快速容易地创建或修改应用程序资源，使用资源编辑器可以编辑所有 Windows 资源，当在 Class View 或 Resource View 中双击资源的名字时，相应的资源编辑器就会出现。Visual C++ 6.0 的编辑器有多种，例如加速键编辑器编辑加速键表；二进制数据编辑器对二进制数据信息进行编辑；图形编辑器可编辑位图、光标、图标；对话框用对话框编辑器；字符串表有字符串编辑器；工具条编辑器和版本信息编辑器分别用来编辑工具条和版本信息。

1.3.6 调试器和诊断工具

要完成一个程序所花费的时间恐怕有一大半用在调试这一步上。执行调试之前，必须将最新工程编译成 obj 文件。调试是一个查错、改错的过程，调试器根据调试的结果在主窗口的 output 区域内显示有多少 Errors 和多少 Warnings，同时指出每一个错误和警告的所在地。在调试的过程中可根据不同的方式来设置各种断点，当程序运行到这些断点时就会停下来，在调试器的 Watch 窗口中可设置不同变量，以观察它们的变化，在堆栈窗口显示程序运行时调用的函数。Visual C++ 6.0 提供了许许多多的诊断工具，例如用于观察消息的 Seytt，进行内存检查的 HelpWalker，用于人工限制可使用内存的 Stress，用于观察消息的 DDESpy，还有一个查错的 ErrorLook。

1.3.7 Books Online(在线书籍)

Books Online 是 Visual C++ 6.0 所带的十分全面的联机帮助文档，它实际上包含了微软最新的软件编程技术，在主窗口的 Workspace 区域点击 Info view 页面卡就会出现以树状组织的整个内容，由三部分组成。而每个文档都是用 HTML 编写的，它们之间用链和谐地组织在一起。如果在硬盘上安装这部分内容要多占用 100M 的空间，足见其内容之多，它里面包含的 100 多个实例是学习 Visual C++ 的极好资料，而且还有 MFC 的全部内容，并对每个函数都做了充分详细的解释。

1.4 Windows 编程模式

虽然到目前为止，存在不少的程序的运行还离不开纯粹的 MS-DOS，但随着 Windows 98 操作系统彻底征服个人微机领域，Windows 编程也从过去的那种面向事务程序设计的模式转向到如今的面向对象的模式，普通人在学习了某种可视化编程工具之后，就可做过去仅能由电脑精英来完成的事情。

1.4.1 Windows SDK

在为开发 Windows 应用而设计的精美的应用程序框架问世之前，软件开发者不得不采用 SDK(Windows Software Development Kit)——软件开发工具包，它是随着 1985 年 11 月份发布的 Microsoft Windows 成为最为流行的用户界面环境而逐步被人们所掌握的。它的核心内容是 API(运用程序设计接口)，事件驱动和消息循环。API 是一个程序内(或一组相关程序内)的一套函数调用，程序员用它创建其他程序，Windows 正是利用 API 实现了它的惊人的图形用户界面(GUI)。事实上 Windows API 包括 16 位的 Windows API(Win16 API)和 32 位 Windows API (Win32 API)，它们都包含在 Windows 子系统的以 VSER32.DLL，GDI32.DLL，KERNELB2.DLL 为核心的动态链接库内。但由于 Win32 API 几乎包含了 Win16 API 所有的一切，并且还要多得多，Win16 API 被认为是 Win32 API 的子集。Win32 API 包含如下 Win16 API 没有实现的新特征：使用独立的寻址空间的抢先多任务进程(Preemptively Multitasked process)，抢先线程(Preemptively thread)，信号灯(Semaphore)，共享存储器(Shared memory)，命名管道(Named pipe)，邮箱槽(Mailslot)和存储器映射文件 I/O(Memory-maped file I/O)，图形设备接口(GDI)中改进了 Bezier 曲线(Bezier curve)，路径(Path)和多换(eransform)的概念，真正能够了解和运用上述概念对初学者有一定的难度，但要想编写出一些实用的应用程序，必须在它上面多花点时间。Windows 是消息驱动(或事件驱动)的操作系统(OS)，事件驱动意味着操作系统的每一部分和其他部分以及应用程序之间通过 Windows 消息进行通信。举一个极为普通的例子，当我们移动鼠标或按下鼠标某个键时，就向应用程序发送了一个消息，这些消息都存储到一个称之为消息队列(Message Queue)的保留区，Windows 应用程序能够向操作系统和其他的应用程序发送和接收几百条消息，并根据消息到达的先后时间和优先权等级进行处理，但有时并不全部处理。Windows 程序采用一个消息循环(Message loop)机制来组织整个程序的运行。Windows 程序的入口是 WinMain()函数，在该函数中完成下列任务：初始化应用程序；初始化创建应用程序窗口；进入应用程序的消息循环。进入整个程序的核心操作即处理消息循环时，调用 Win32()函数，这一窗口过程处理在消息队列中等待的消息或事件。

1.4.2 面向对象编程

面向对象编程(Object Oriented Programming-OOP)技术的出现是软件编程的一个里程碑，它把许多的程序员从无谓的重复性劳动中解放了出来，并由于它通过继承和多态大大简化了程序的开发过程，从而产生了更多的使用这一思想工具的程序员。过去的 Pascal 和 C 语言都是过程式语言，这种过程式编程(Procedural Programming)的致命缺点是随着程序规模的不断增大，隐藏在各个程序模块中的副作用越来越明显。经常出现的情况是当程序达到一定的规模后很难调试，不能发现错误所在。面向对象采用封装的技术很好地解决了这个在过程式语言中始终无法解决的问题。OOP 中必须清楚知道的三个概念是对象(Object)、方法(Method)和消息(Message)。在计算机中用被称为对象的新数据类型表示项，在代码内对象由叫做类的数据结构定义，通过获取消息与将方法作用于消息的途径，类知道如何执行指定操作。对象仅是类(Class)的运行实例，方法是包含在对象内的成员函数，消息是对方法的实际调用

1.4.3 MFC简介

MFC(Microsoft Foundation Class Library)类库是开发 Windows 的C++ 类集，是 Visual C++ 开发 Windows 程序的非常重要的工具。该类集以层次结构组织，封装了大部分 Windows API 函数，所包含的功能涉及到整个 Windows 操作系统。MFC 不仅为用户提供了 Windows 图形环境下应用程序的框架，而且提供了创建应用程序的组件，使用 MFC 类库和 Visual C++ 提供的高度可视的应用程序开发工具，可使程序的开发变得更简单，开发周期大大缩短，而且在代码的可靠性和可重用性上得到很大提高。

MFC 类库类似层次结构组织，几乎每个子层次结构都与一具体的 Windows 实体相对应，一些主要的接口类管理了难以掌握的 Windows 接口，这些子结构包括：窗口类、GDI 类、对象连接和嵌入类(OLE)、文件类、对象 I/O 类、异常处理类、集合类。

MFC 包括两部分，一部分从 Cobject 类派生，另一部分非 Cobject 类派生。

CObject 派生类包括：

- 数组、列表和映像类
- 调试的异常类
- 绘图和打印类
- 文件和数据库类
- Internet 和网络类
- MFC 应用程序结构类
- 窗口、对话框和控件类

非 CObject 派生类包括：

- 简单数据型类
- 支持类
- 典型模板类
- 结构类
- 同步类

1.5 简单的 Visual C++ 6.0 编程实例

Visual C++ 6.0 强大的操作界面也许令人眼花缭乱，但它同样强大的编程工具 AppWizard 和 ClassWizard 使得编程变得非常简单，可视化(Visual)真正体现了面向对象的特征，很多人视之为点击编程或彻底编程，程序员的主要精力不是放在美化界面的设计上，而是致力于源代码本身。这一章将指导您如何一步一步地编写一个简单的应用程序，并介绍怎样在对话框窗口中加一个命令按钮控制。

1.5.1 Hello应用程序的说明

我们要生成的这个应用程序的界面，如图 1.16 所示，标题是“Hello”，主窗口上有两个按钮，“欢迎”和“确定”，它实现下列一些简单的功能：

1)单击“欢迎”按钮，显示一个消息框，上面写着“欢迎进入 Visual C++ 6.0，欢迎使用，谢谢!”，如图 1.17 所示

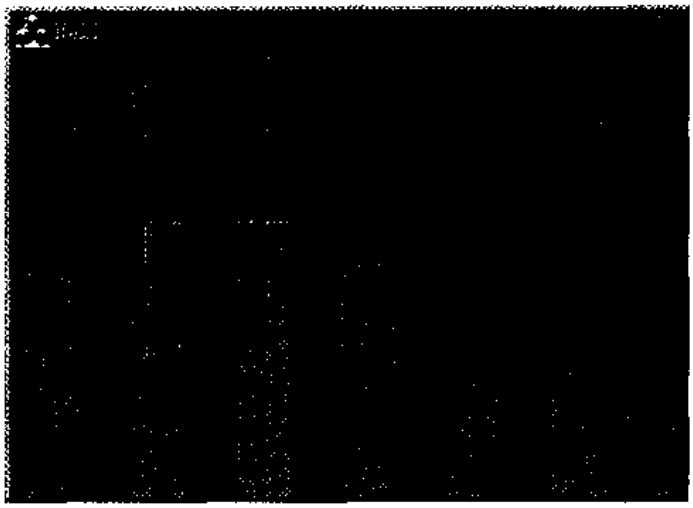


图 1.16　Hello 应用程序界面



图 1.17　消息框界面图

2)单击"确定"按钮关闭消息框。

3)单击应用程序左上角图标,弹出系统菜单,选择"关于"项时会出现"关于…"对话框,如图 1.18 所示。



图 1.18　关于对话框内容

4)单击"确定"按钮,关闭"关于…"对话框。

5)单击 Hello 对话框"确定"按钮,退出应用程序。

1.5.2　生成 Hello 工程文件

生成该工程文件按以下步骤进行:

1)进入 Visual C++ 6.0 的工作台。

注意:用户编写和维护 Visual C++ 程序是用 Developer Studio 来进行的,实际上名字 Developer Studio 和 Visual C++ 是可互换的,从现在起,本书用 Visual C++ 来代替 Developer Studio。

2)从顶部 File 菜单中选 New 选项,出现一对话框,如图 1.19 所示,该窗口中有四个页签,File、Projects、Workspace 和 Other Documents。

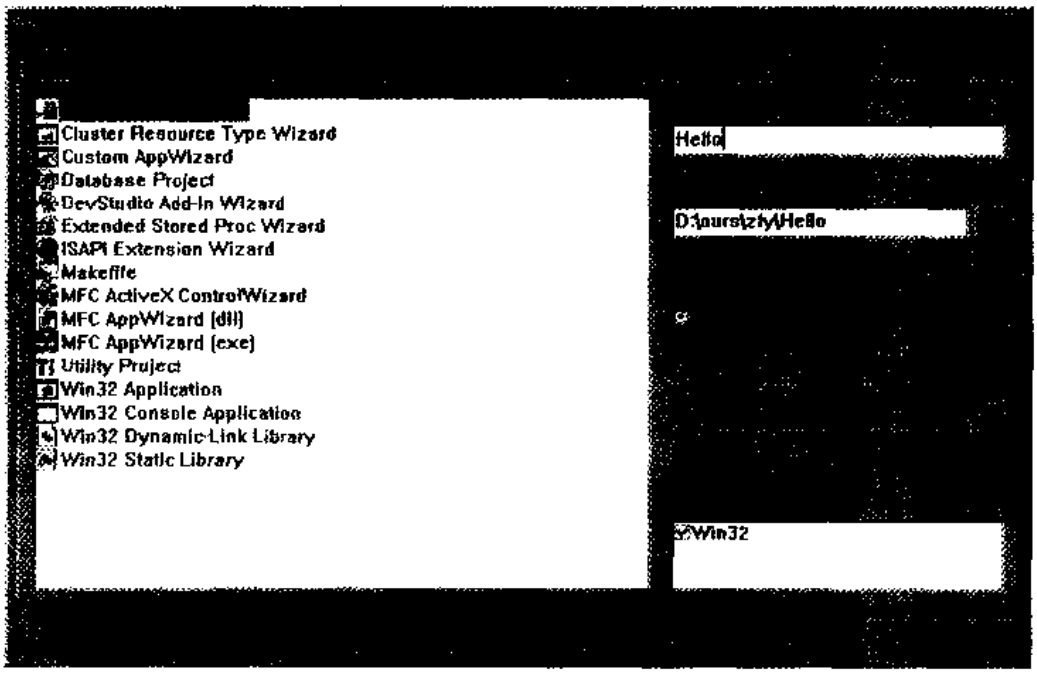


图 1.19 New 对话框

3)选择 Projects 页签,在该页左边的编辑框中选择 MFC AppWizard(exe)项,即要用 AppWizard 生成一可执行文件。

4)在 Projects Name 编辑框中输入要创建的工程文件的名字,在此命名为 Hello。

5)Location 编辑框中描述的是放该工程文件的路径,可根据用户需要,输入自己希望放置该文件的位置。

6)由于这是第一个工程文件,必须创建一个新的工作区,故 Create New Workspace 选项是必选的。

7)在 Platforms 编辑框中选择 Win32。我们使用的 Visual C++ 6.0 是在 32 位操作系统下运行的。

8)确定工程文件的类型、名称和路径选项后按 OK 按钮,就可以开始创建该工程文件了。

下面就是在 Visual C++ 6.0 MFC AppWizard 的可视化指导下,根据我们自己的设计,一步一步地来生成应用程序的工程文件和应用程序的框架文件。Visual C++ 显示出一个 MFC AppWizard-Step1 的窗口,如图 1.20 所示,这是生成工程文件的第一步,在这一步中,将让你选择创建什么类型的应用程序以及资源文件使用什么语言。

9)在 What type of application would you like to create? 提示下有三项选择:Single document(单文档),Multiple documents(多文档)和 Dialog based(对话框)。在本书中,我们要生成的应用程序都是基于对话框的应用程序,故选择 Dialog based 选项。

10)在 What language would you like your resources in? 中,选择"中文[中国]"项。

11)单击 Next 按钮,进入第二步。如图 1.21 所示。

12)在 What features would you like to include? 提示下有几个选项:

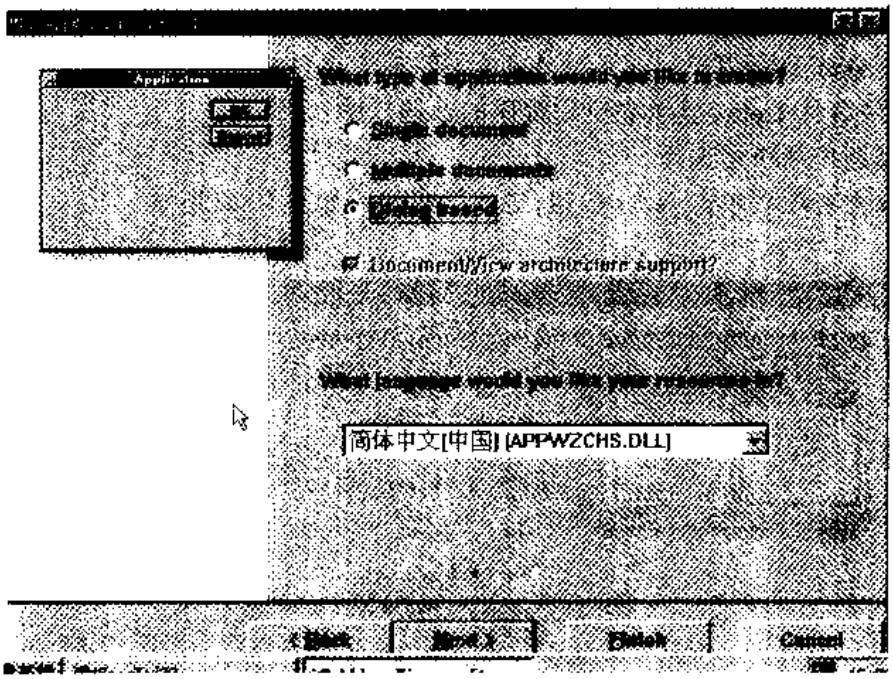


图 1.20　MFC AppWizard-Step 1 对话框

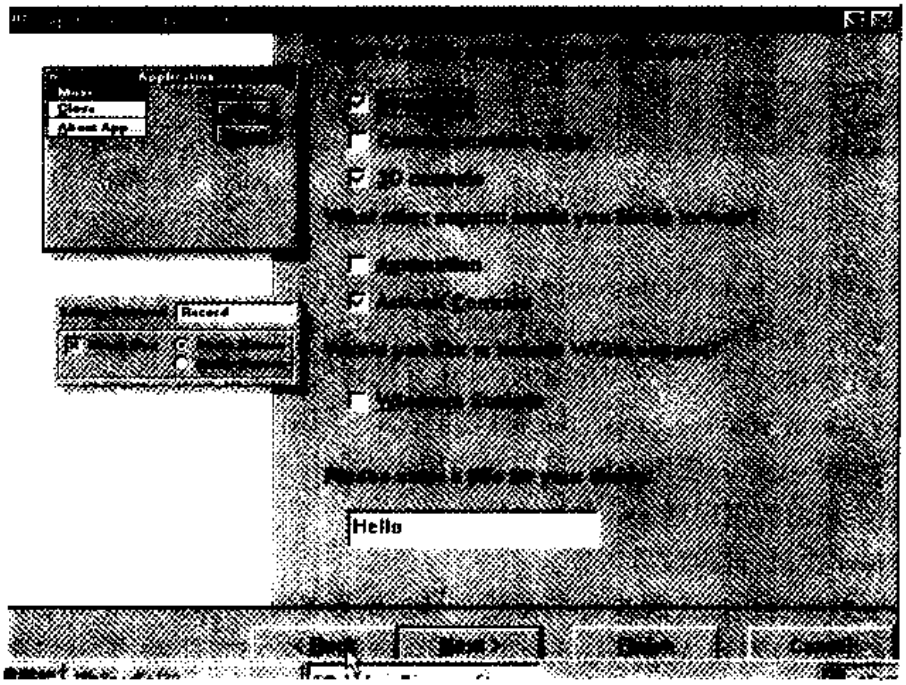


图 1.21　MFC AppWizard Step 2 of 4

- About 对话框
- 上下文相关(Context-Sensitive)帮助
- 三维图形控件

在前面显示的应用程序界面中可以看到有一个 About 对话框,图形控件也是三维的,至于上下文帮助则没有,所以按图 1.21 所示进行选择.

13)在 What other support would you like to include? 中,选择 ActiveX Controls 项

14)在 Please create a Title for your dialog 编辑框中可以给应用程序主窗口命名,这个例子中使用工程文件名称 Hello,显示在应用程序的标题栏中。

15)其他问题按图所示选择。

16)确定以上各项后单击 Next 按钮,进入第三步。如图 1.22 所示。

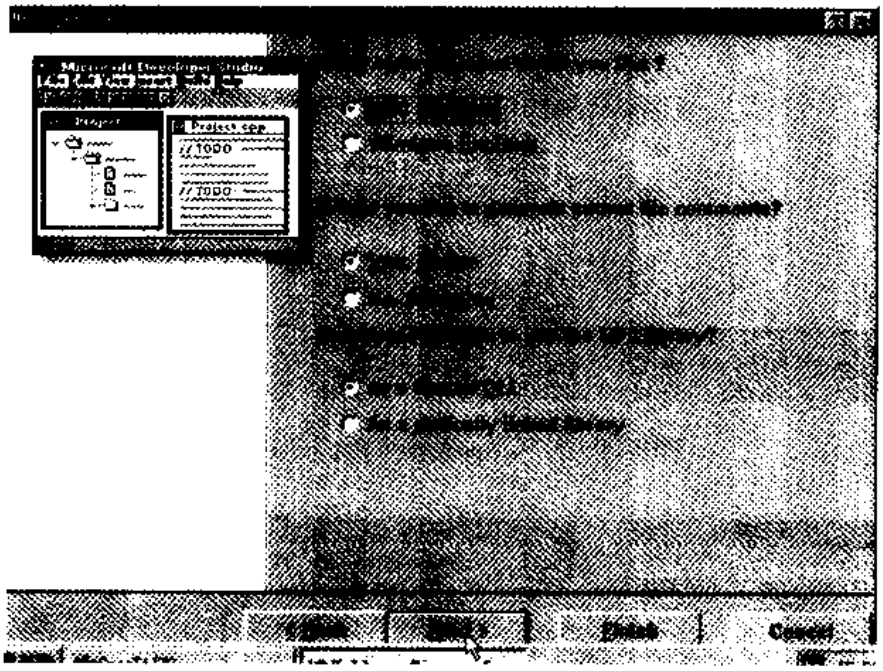


图 1.22 MFC AppWizard Step 5 of 6 对话框

17)Would you like to generate source file comments? 询问是否在生成的源代码中加注释,应选 Yes。因为在 App Wizard 生成的应用程序源代码文件.cpp 文件中加上代码的注释,有助于阅读和理解源代码的含义,毕竟 Visual C++ 还有代码编程部分。

18)在 How would you like to use the MFC library? 问题中,选择使用 MFC 的动态链接库 DLLS,而不是静态链接库。使用 DLL 可以在应用程序运行的时候链接起来,而不是在编译的时候就链接,使生成的应用程序 exe 文件比较小。

19)单击 Next 按钮,进入最后一步。

Visual C++ 显示出生成工程文件的最后一步,出现 MFC AppWizard-Step 6 of 6 的窗口(如图 1.23 所示),可以看到 AppWizard 给出的所创造的类名和文件名,如果你愿意可以改变类的名称,但一般不必去改变它,本例中对这个窗口不做任何改动。可以看到 AppWizard 将为应用程序生成两个派生类:CHelloApp 和 CHelloDlg。CHelloApp 类是 MFC 基本类库中的 CWinApp 类派生出来的,是应用程序的一个框架类,给应用程序提供了系统主窗口的一些特征,CHelloDlg 是从 MFC 基本类库 CDialog 类派生出来的。

20)单击 Finish 按钮显示 New Project Information 窗口,如图 1.24 所示。该窗口显示了前面几步所选择的全部设置,包括应用程序类型,新创建的类以及应用程序的全部特征,还显示了该应用程序将生成在哪个路径之中。通过这些,你可以最后检查一下应用程序的设置是否完全正确,如果出现不对的地方,可以按 Cancel 按钮取消前面操作,用 AppWizard 重新制作。

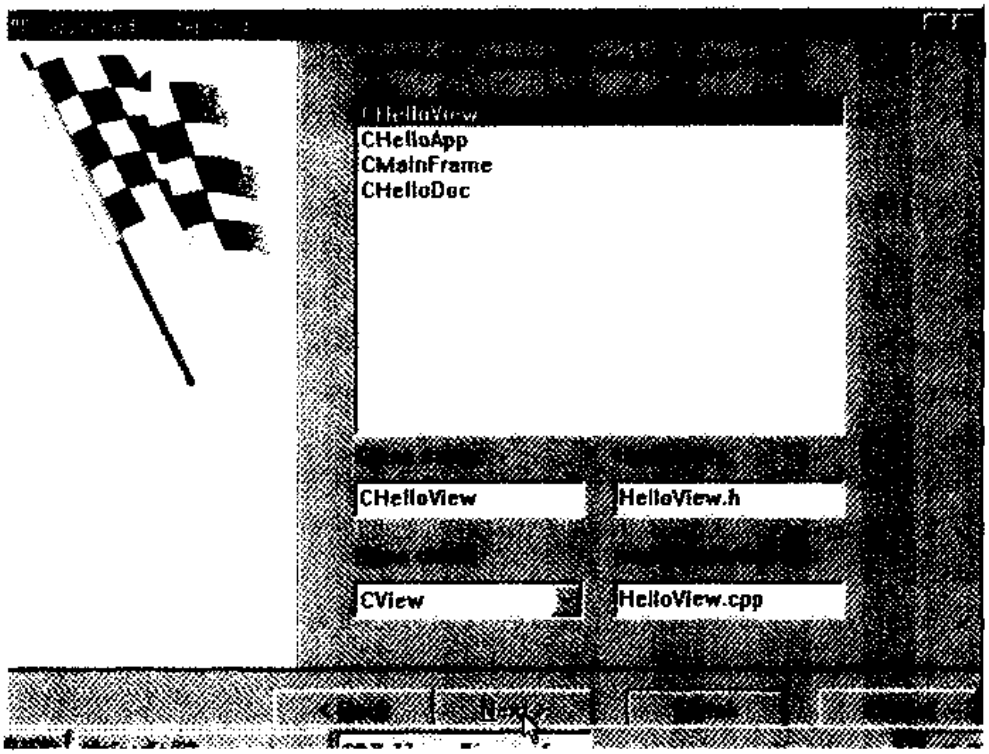


图 1.23　MFC AppWizard Step 6 of 6 对话框

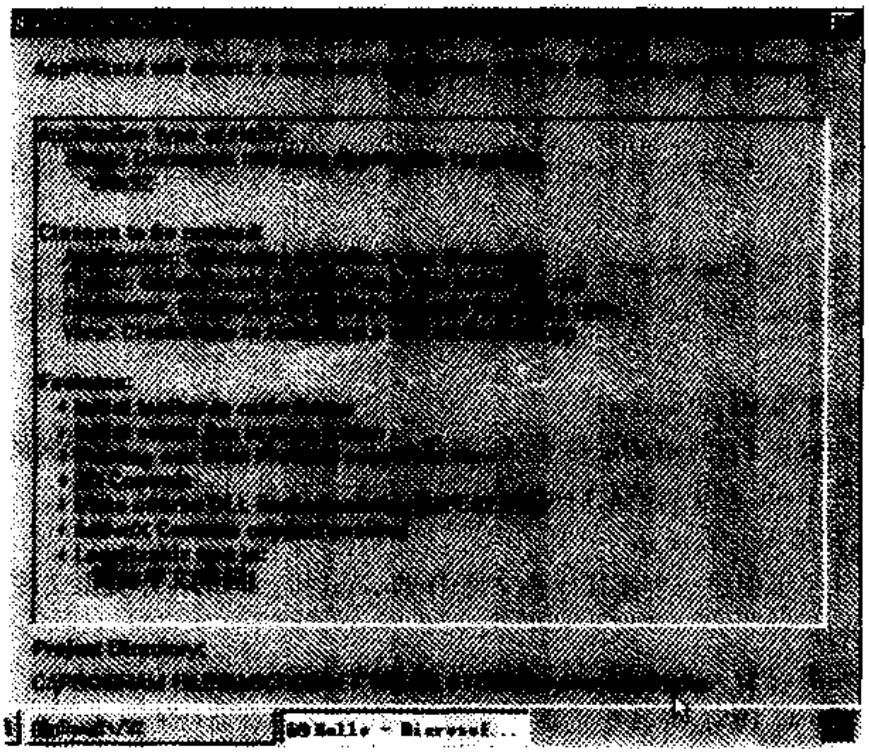


图 1.24　New Project Information 窗口

检查后若完全正确，则执行后面的步骤

21）单击 OK 按钮，AppWizard 完成应用程序的自动生成工作，在指定的目录下生成应用

程序框架所必须的全部文件。到现在 AppWizard 已完成了它全部工作，那么它到底做了些什么工作呢？它生成了一些什么文件，而每个文件又有什么用途呢？

Visual C++ 自动打开了 AppWizard 生成的工作区(Workspace)，在工作区上有四个页签：ClassView、ResourceView、FileView 和 InfoView。ClassView 页中显示的是 AppWizard 生成应用程序时，为应用程序新创建的类以及类的成员变量和成员函数，用鼠标双击任何一个成员变量或成员函数，就会打开含有该成员变量或成员函数的源代码文件，并且将光标放在源代码文件中的该成员变量或成员函数所在行。ResourceView 页中显示的是新建应用程序的各种资源，例如 About 对话框资源、子窗口图标、字符串、表示版本信息等。

FileView 页中显示的是 Visual C++ 的帮助信息，Visual C++ 6.0 的联机帮助很大，在硬盘中要占用一百兆空间，而且联机帮助非常详细且全面。

单击 FileView 制表页，显示该文件夹，再单击每个文件夹左边的"+"号，就显示出了 Hello 工程的所有文件。这些文件组成了 Hello 应用程序，它们的特征和用途分别是：

1)Hello.h

这个文件是应用程序的主文件，它包含有其他工程特有的头文件(如 Resource.h 文件)，并且声明了 CHelloApp 应用程序类。

2)Hello.cpp

这是一个主应用程序的源代码文件，其中定义了应用程序类 CHelloApp。

3)Hello.rc

这是一个该应用程序使用的所有 Microsoft Windows 资源的列表，显示出来的就是 ResourceView 页签。包括的资源有图片、位图以及存放在 RES 子目录下的光标，这个文件中的内容直接在 Visual C++ 6.0 的工作台中进行编辑。

4)Hello.ico

这是一个图片文件，是被用于应用程序的图案，这个图片放在主资源文件 Hello.h 中

5)Hello.clw

这个文件包含有 ClassWizard 在编辑现存的类或增加新的类时的有关信息，Class Wizard 在编辑 Message maps 和 Dialogdata maps 去创建原型成员函数时，必须使用这个文件存储信息。

6)HelloDlg.h 和 HelloDlg.cpp

这两个文件包含了 CHelloDlg 这个类，该类定义了应用程序的主对话框的特征。对话框的模板放在文件 Hello.h 中，能在 Visual C++ 6.0 的工作台中被直接编辑。

7)StdAfx.h 和 StdAfx.cpp

这两个文件用于创建一个新编译头文件 Hello.cpp 和一个预编译类型文件 StdAfx.abj。

8)Resource.h

这是一个标准的头文件，用于定义资源新的 IDS。

现在，我们对 AppWizard 这个工具已经有了初步认识，前面所有这些工作都是它完成的，到现在我们还没有编写一句代码，它就已经替我们生成了这么多文件。在下一节中，我们将会看到 AppWizard 到底做了些什么工作。

1.5.3 进行可视化编程

上一部分中，使用 AppWizard 生成了应用程序的工程文件，那么接下来我们要用 Visual

C++另一强大工具——ClassWizard来进行可视化程序设计。用AppWizard建立的应用程序除具有一些一般性的功能外，还应能完成某些特定的功能。因此，必须对AppWizard生成的工程文件进行加工。ClassWizard是Visual C++提供的对工程进行加工、完善和编辑的一个强大的可视化工具，它同图形化的界面一样，不需要程序员进行大量的代码操作，就能完成消息处理，函数的映射，图形界面的设计等功能。

现在，我们可以运行一下AppWizard生成的应用程序框架，看看还需要我们做些什么。

1)选择Build菜单中的Rebuild All项，Visual C++编译和连接Hello应用程序。

2)选择Build菜单中的Execute Hello.exe项，运行该应用程序。这时会出现一个对话框窗口，在它的中间有两个命令按钮："欢迎"和"确定"。

3)单击窗口左上角的小图标，显示窗口的系统菜单；选择该菜单的"关于Hello(A)..."项，弹出一个"关于"对话框。单击"确定"按钮，关闭"关于"对话框窗口。

4)单击"确定"按钮，就可以结束Hello应用程序。

通过运行AppWizard生成的应用程序，可以看到系统已经为应用程序做了一个框架。而在实际的应用程序中，不需要窗口中的"确定"按钮和那段文本，但要在窗口中重新增加一个命令按钮对象："欢迎"，并且单击它后要弹出一个对话框。

现在我们知道要做些什么了，那么开始用ClassWizard来实现这些功能，步骤如下：

1)在应用程序工作区中选择ResourceView项，该页有一个Helloresource的文件夹，打开这个文件夹，又出现四个不同的资源文件夹。

2)单击Dialog文件夹旁边的"+"号，将其打开，这时会出现两个资源对象，IDD_ABOUTBOX是应用程序的"关于"对话框，IDD_HELLO_DIALOG是作为应用程序主窗口的对话框。因为我们的任务是定做应用程序主窗口的对话框，所以应在IDD_HELLO_DIALOG中进行操作。

3)双击IDD_HELLO_DIALOG选项。

Visual C++ 6.0主窗口的对话框显示在编辑窗口中，并且显示一个工具窗口(列出各种控件)，如图1.25所示。



图1.25

4)单击IDD_HELLO_DIALOG对话框的"取消"按钮，按键盘上的Delete键，把这个对象从对话框中删除。

5)单击IDD_HELLO_DIALOG对话框的文本，按键盘上的Delete键，把这个对象从对话框中删除。

现在IDD_HELLO_DIALOG对话框是空白的，什么对象都没有。下一步是在这个对话框中加上一个控件："欢迎"命令按钮。

6)单击工具窗口中的命令按钮控件，然后在空白的对话框的任意位置单击一下鼠标，就会出现一个命令按钮。用鼠标将该按钮移动到你所需要的位置上。按钮的缺省标题是Button1，这里我们需要改变标题。

7)在Button1按钮上单击鼠标右键，在弹出的菜单中选择Properties项，就会显示一个按钮特性的对话框窗口。选择对话框中General页签。Caption编辑框的内容是按钮的标题，将原来缺省的文本Button1改为"欢迎"。

8)单击"欢迎"按钮，拖动按钮的边框，直到变成所需要的大小。用鼠标移动按钮的位置，可放在你想要放置的地方。做完以后窗口界面显示如图1.16所示。到现在为止，我们已经基本完成了应用程序的可视化工作，在Files菜单下选择

Save 项，保存我们已做的工作。下面观察一下可视化设计结果。

首先选择 Visual C++ 主菜单 Build 下的 Build Hello.exe 项，编译和连接应用程序。

其次在 Visual C++ 编译和连接完后，选择 Build 下的 Execute Hello.exe 项，运行应用程序。

显示一个如图 1.16 所示的应用程序主窗口，但此时按“欢迎”键和“确定”键时，应用程序没有任何反应，原因在于我们还没有给应用程序编写任何代码，也就是说，现在我们只是编写了一个没有任何用途的程序界面，下一部分将给这个界面添加代码。

1.5.4 给应用程序添加代码

首先，我们给“欢迎”按钮添加代码，步骤如下：

1)在“欢迎”按钮上单击鼠标右键，弹出一个菜单，选择菜单的 ClassWizard 选项。

Visual C++ 显示 ClassWizard 对话框，如图 1.26 所示，对话框有 5 个页签：

- Message maps
- Member Variables
- Automation
- ActiveX Events
- Class Info

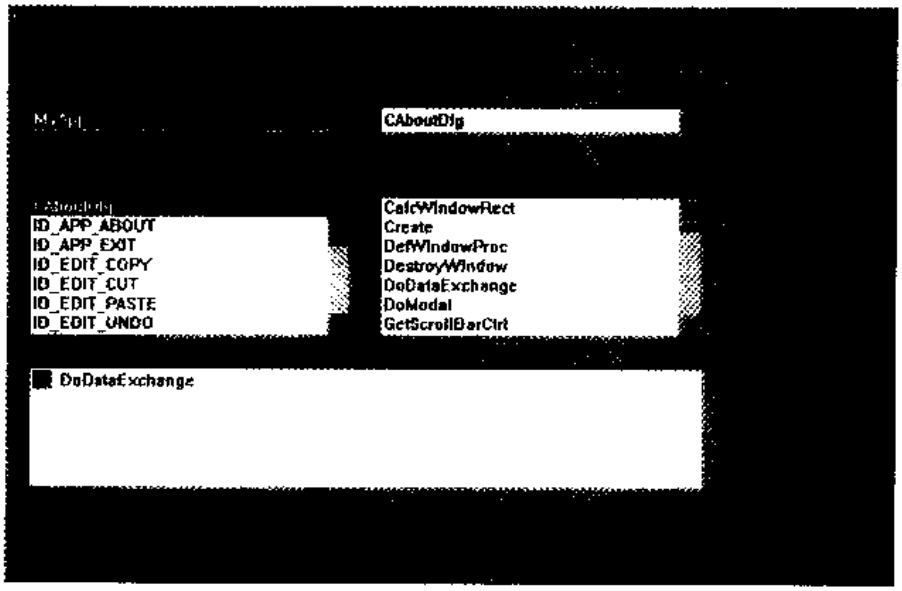


图 1.26 MFC ClassWizard 对话框

2)确定当前选择的是 Message maps 页签。

3)Project 下拉列表框中显示的是 Hello。因为 Class Wizard 是在工程文件 Hello 中扩展它的功能。按右边的箭头，弹出的菜单中只有这一项，这是因为在工程工作区只有这一个工程文件。

4)Class name 下拉列框表中显示的是 CHelloDlg，因为当前的任务是将“欢迎”按钮写入 CHelloDlg 类中。

5)在这两个对话框的下面显示了将要把连接代码放置在 HelloDlg.h 和 HelloDlg.cpp 两文

作中,以及这些文件的路径。

6)Object IDS 制表框用于选择代码连接在哪一对象上面,因为现在是给"欢迎"按钮连接代码,故在这一项中选择 IDC _ BUTTON 选项。

7)Messages 列表框中列出了与所选对象相关文件的消息,在 Object IDS 中选择了 IDC _ BUTTON 对象,与之相关的消息有 BN _ CLICKED(单击该按钮)和 BN _ DOUBLECLICKED(双击该按钮)。为了告诉 ClassWizard 现在要编写有关"单击该按钮"条件的代码,需要首先在 Object IDS 列表框中选择 IDC _ BUTTON。然后,在 Messages 列表框中选择 BN _ CLICKED。下一步要告诉 ClassWizard 添加对应于所选文件消息的处理函数,即当单击"欢迎"按钮后自动生成的函数。

8)单击 Add Function 按钮,显示一个增加成员函数的对话框,该窗口建议新增成员函数名为 OnButton。

9)单击"OK"按钮,接受建议的函数名。

这样函数 OnButton 加到了应用程序之中,每当用户单击"欢迎"按钮时就执行该成员函数,在 Message Maps 制表页 Member Functions 中增加了这样的一条语句:

OnButton ON _ IDC _ BUTTON:BN-CLICKED

这表示向"欢迎"按钮(IDC _ BUTTON)传送单击该按钮的消息(BN-CLOCKED)时,自动执行一个消息处理函数 ONButton。ClassWizard 仅仅写入消息处理的声明代码和程序框架,而编写该函数内容的任务则由程序员来完成。

编写程序代码按以下步骤完成。

1)单击 MFC ClassWizard 的 Edit Code 按钮,ClassWizard 打开 HelloDlg.cpp 源代码文件,并使光标停留在消息处理函数的源代码处。

2)在成员函数 OnButton()的定义中输入函数代码,如下所示:

```
void HelloDlg:OnButton()
messageBox("欢迎进入 Visual C++ 6.0,欢迎使用,谢谢!");
```

3)选择菜单 File 的 Save 项存盘。

现在,此应用程序全部完成可运行一下,步骤如下:

1) 选择主菜单 Build 中 Build Hello.exe 项。

2) 在编译和连接后,选择 Build 下的 Execute Hello.exe 项运行该程序,单击"欢迎"按钮,显示一消息框,上写"欢迎进入 Visual C++ 6.0,欢迎使用,谢谢!"。

3) 单击"确定"按钮,关闭消息框。

4) 单击应用程序左上角图标,显示一系统菜单,选择 About 项出现 About 对话框

5) 单击"确定"按钮,关闭"关于"对话框。

6) 单击"确定"按钮,退出该程序,确定创建了一个正确的应用程序。

如果您是按我们的步骤一步一步地来做的,那么你肯定会正确地完成这个应用程序的编写。

1.6 小 结

本章概要地介绍了微软的新作 Visual C++ 6.0 特点及组成,并引导读者一步一步地安装该最新版本的软件。对 Windows 编程模式作了比较详细的说明,力求读者理解这一重要的概

念，事件驱动和消息机制是整个 Windows 编程的基础，而 MFC 则是 Visual C++ 构造程序的框架，理解 OOP 的概念至关重要，但 Visual C++ 使程序员的工作简化到只处理一个个的单个事件。在“简单的 Visual C++ 6.0 编程实例”一节中，带领读者按基本的编程方式循序渐进地编写了本书的第一个示范程序。总之，这一章的目的是让读者对 Visual C++ 6.0 有一个概括的认识，接下来的几章将在各个方面进行探索。

第 2 章　Visual C++ 6.0 集成环境

Visual C++ 6.0 作为一个成熟的 32 位软件开发工具，它提供的开发环境具有良好的可视性，利用它可以开发出极其繁杂的标准 Windows 应用程序，在该环境下（以后称之为开发工作台），可以对 C 和C++ 应用进行各种可视化操作，包括创建、打开、浏览、编辑、保存、编译、链接和调试等。

2.1　Visual C++ 6.0 主窗口

安装完 Visual C++ 6.0 后，启动 Visual C++ 6.0 进入开放环境 Developer Studio（如图 2.1 所示），Developer Studio 由标题栏、菜单栏、工具栏、工作区窗口、源代码编辑窗口、输出窗口和状态栏组成。

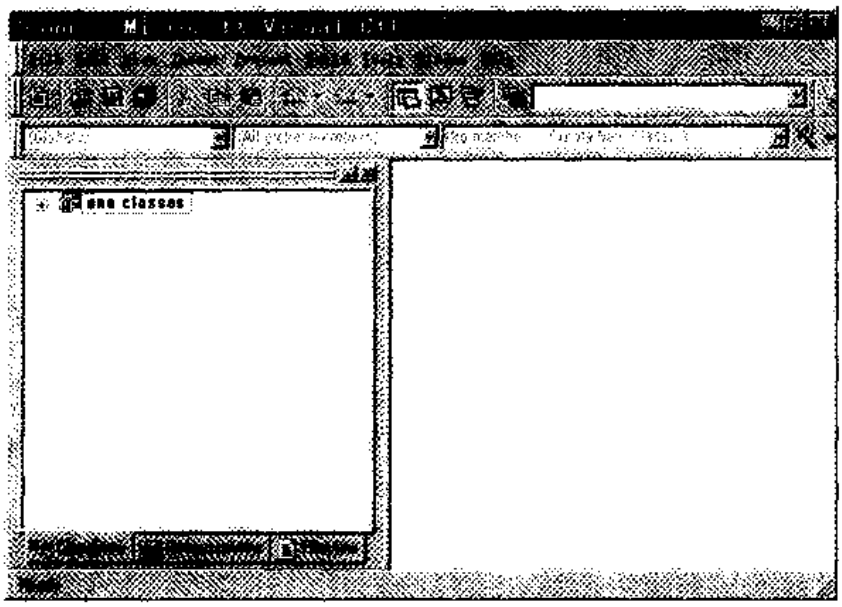


图 2.1　Developer Studio 环境

屏幕最上端为标题栏，用于显示应用程序名和所打开的文件名，标题栏的颜色用于指示对应窗口是否为激活的。标题栏左端是控制菜单框，用于打开窗口控制菜单的图标，用鼠标单击该图标或按 Alt + 空格键，将弹出窗口控制菜单。窗口控制菜单用于控制窗口的大小和位置，如还原、关闭、移动等，标题栏的右边有三个控制按钮，从左到右分别为最小化、还原、关闭按钮，这些按钮可以使用户快速设置窗口大小。

标题栏下面是菜单栏和工具栏，工具栏下面是两个窗口，一是工作区窗口，一是源代码编辑窗口。初始化时被激活的是源代码编辑窗的 Infoviewer 主题窗口。工作区窗口的下面是输出窗口，用于显示项目创建过程中所产生的错误信息。屏幕最底端是状态栏，给出当前操作或所选择命令的提示信息。

2.2 AppWizard

AppWizard,即 Application Wizard 的缩写形式俗称“应用程序创建向导”,V C++ 中提供了一系列的 Wizard 这里,我们将一一作介绍。

使用 MFC AppWizard 可以创建基于 MFC 类库的 Windows 应用程序(可执行文件或动态链接库)。MFC AppWizard 显示一系列对话框,用户可以从对话框中选择满足应用程序和所需要的选项。在定义完应用程序和项目选项后,MFC AppWizard 生成创建应用程序所需要的开始文件,使这些文件与 Class Wizard 兼容。

要用 MFC AppWizard 生成新的项目和应用程序需遵循以下步骤:

1)启动 Visual C++ 6.0。

2)从 File 菜单中选择 New 命令(或按 Ctl+N)弹出 New 对话框。

3)切换到 Project 选项卡,从项目类型列表框中单击要创建的项目类型,可以创建的项目类型有 ATLCOM AppWizard、Custom AppWizard、Database Project、Devstudio Addin Wizard、ISAPI、Extension Wizard、Makefile、MFC ActiveX CoufsolWizard、MFC AppWizard(exe)、MFC AppWizard(dll)、NewDatabase Wizard、Win32 Application、Win32 Dynamic-Link library、Win32condole Application 和 Win32 Static Library。这里选择“MFC AppWizard(exe)”。

4)在“Project name”文本框中键入项目的名字,如“MyApp”,MFC AppWizard 根据所键入的名字派生相应的开始文件和类的缺省名,并在根目录下创建以该名命名的子目录,子目录包含开始文件。

5)在 Location 文本框中键入用于存放项目的根目录。

6)如果要添加新的项目到已有的工作区中,请选择“Add to Current workspace”选项,否则将自动创建包含新项目的新工作区。若要使新项目成为已有项目的子项目,请选中“Dependency of”框并指定项目名。

7)从“Platforms”列表框选择可用的目标平台,缺省平台为 Win32。在创建项目时,系统缺省自动创建两种项目配置(即 Win32 Debug 和 Win32 Release)。Win32 Debug 配置包含若选择其他目标平台,则必须安装与该平台相关的 Visual C++ 6.0 版本。

8)单击 OK 按钮,弹出如图 2.2 所示对话框,从中确定应用程序的结构并为该源文本选择一种语言。

MFC AppWizard 可以创建以下语言类型的应用程序结构。

- Single documents(单文档):单文档界面(SDI)一次只允许打开一个文档边框窗口。
- Multiple documents(多文档):多文档界面(MDI)允许在应用程序的同一实例中打开多个文档边框窗口。
- Dialog-Based(基于对话框):基于对话框的应用程序将显示一个简单的对话框供用户输入。选择应用程序结构时,MFCAppWizard 对话框左上角会显示与所选择结构相对应的图表,例如,对于多文档界面结构,在名为“Application”的窗口中将显示两个文档边框窗口。资源文本可用的语言有中、英、法等,中文是资源文本的缺省语言,如果要选择其他语言,请确认系统已安装相应的动态链接库。

9)选择“Multiple documents”选项,表示将生成 MDI 应用程序。选择后单击 Next 按钮,弹出一对话框,从中可以选择数据库支持以决定如何支持,选项包括 ODBC 和 DAO 两种。如果

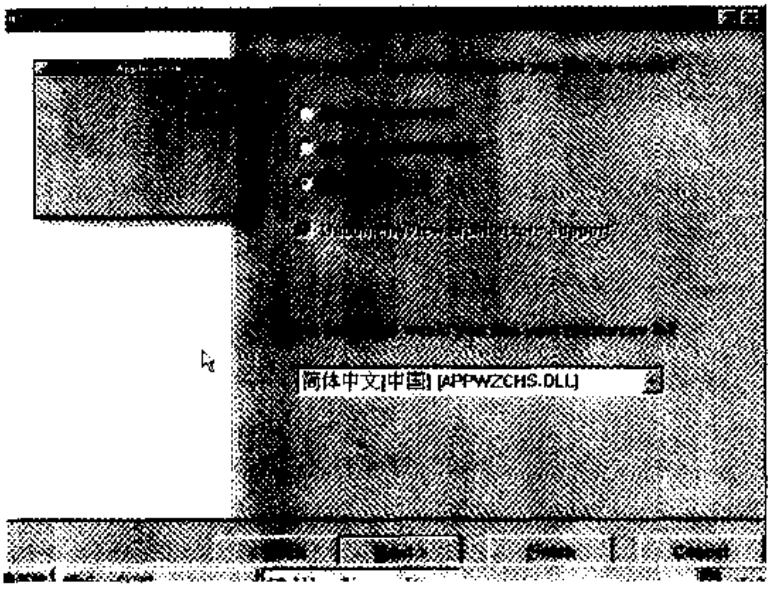


图 2.2　应用程序结构确定对话框

应用程序为 SDI 或 MDI 界面，则可以选择以下的数据库支持选项：

- None：该选项不支持任何 ODBC 库。如果应用程序不使用数据库，则选择该项将创建比较小的应用程序。
- Header files only：该选项提供最低限度的数据库支持，包含所有的数据库头文件和链接库，但 MFC AppWizard 不创建任何与数据库相关的类，用户需要自己创建
- Database View without file support：该项包含所有的数据库文件和链接库，并为用户创建记录视图(record view)和记录集(recordset)。使用该项，应用程序具有文档支持但不支持串行化(serialization)。
- Database View with file support：与上一选项唯一不同之处在于不仅有文档支持还有串行化支持。

若应用程序包含数据库，还必须选择数据源。

10)选择"None"选项，然后单击 Next 按钮，弹出一如图 2.3 所示的对话框，从中可以选择复合文档等选项。

MFC AppWizard 可以为不同的 ActiveX 控件很容易地生成相应的支持代码。选择任一选项都会使标准 ActiveX 资源有效，并将额外的自动化命令添加到菜单栏中。

若应用程序为 SDI 或 MDI 界面，就可选择以下复合文档支持选项：

- None：缺省设置，应用程序不带 ActiveX 支持。
- Container：应用程序包含被链接和嵌入的对象。
- Mini-Server：应用程序仅仅可以创建嵌入对象。
- Full-Server：应用程序既可独立运行又能支持 ActiveX，而且所创建的对象还可以包含在复合文档中。
- Both Container and server：应用程序既可作为容器又可作为服务器。
- Yes, please：用复合文档格式来串行化容器应用程序的文档。
- No thank you：不用复合文档格式来串行化容器应用程序文档。
- Automation：使应用程序支持自动化，这样应用程序就可以直接被其他自动化客户访问。

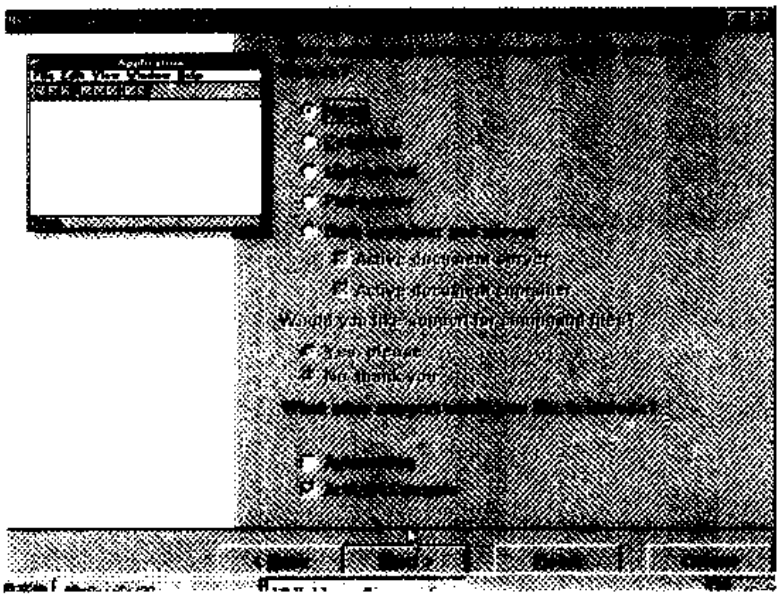


图 2.3　复合文档选择选项对话框

• ActiveX controls:使应用程序作为 ActiveX 控件。

11)选择 None 选项,然后,单击 Next 按钮,弹出如图 2.4 所示的对话框,从中可以选择各种不同的用户界面选项,并可选择 MAPI 和 Windows Sockets 支持,这里使用缺省设置。

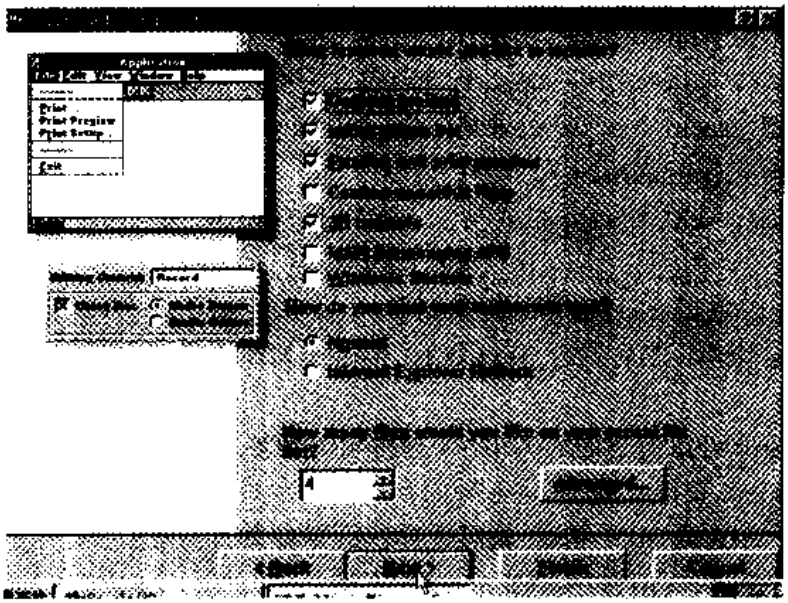


图 2.4　用户界面选项对话框

通过选择以下选项,可以指定应用程序的基本特征。

• Dockable toolbar:缺省设置,添加工具栏到应用程序中。
• Initial Status bar:缺省设置,添加状态栏至应用程序中。
• Printing and print preview:缺省设置,添加代码来调用(View)的成员函数,以便处理打印,打印设置和打印预览等命令。
• Context-sensitive Help:添加帮助按钮到应用程序中。

- 3D controls:缺省设置,为应用程序的用户界面添加三维外观。
- MAPI(Messaging API):使应用程序可以创建操作、传输和存储邮件的消息。
- Windows softs:使应用程序可以与 TCP/IP 网络通信。
- How many files would you like on your recent file list? 使应用程序可以跟踪最近使用过的文件,缺省时应用程序 file 菜单底部列出 4 个最近使用的文件。

如果要设置高级选项,单击 Advance 按钮,弹出 Advanced Options 对话框(如图 2.5 所示),其中"Document Template Strings"选项卡用于设置应用程序的文档模板;"Windows styles"选项卡用于确定应用程序用户界面的风格。调整后单击 Close 按钮返回图 2.4 所示画面。

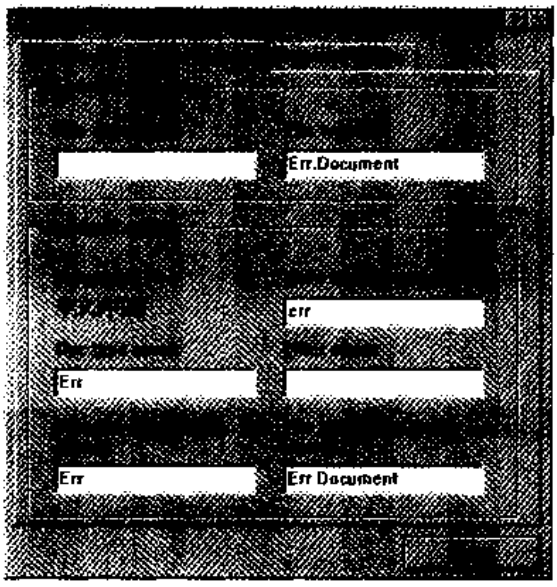


图 2.5 Advanced Aptions 对话框

12)单击 Next 按钮,弹出如图 2.6 所示对话框,从中可以选择以下项目的选项设置。

- would you like to generate source file comments? MFC AppWizard 在源文件中插入相应的注释以帮助用户编写程序。
- How would you like to use the MFC library? MFC 类库可以是静态链接库或动态链接库,可以从中确定自己所需要的状态。

13)单击 Next 按钮,弹出如图 2.7 所示的对话框,显示 AppWizard 将创建的类、头文件(.h)和实现文件(.cpp)的名称。

14)单击"Finish"按钮弹出"New Project Information"对话框(如图 2.8 所示),显示应用程序的规范说明,即应用程序所具有的特征。

15)单击 OK 按钮,确认前面所做的选择均为正确的,MFC AppWizard 根据这些选择生成应用程序的源文件。

完成以上步骤后,MFC AppWizard 为应用程序生成必要的开始文件并在项目工作区打开生成的项目。

用 MFC AppWizard 生成应用程序的开始文件后,就可以创建并运行应用程序。事实上,只是使用 MFC AppWizard 生成的开始文件就可以建立完整的可执行程序。建立应用程序后,

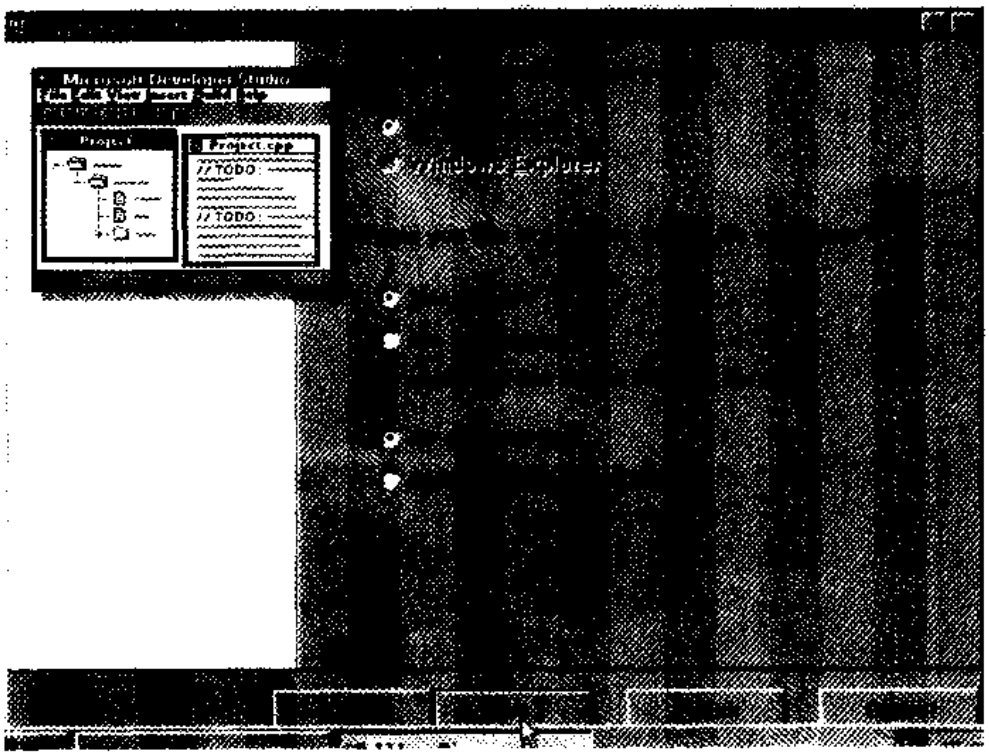


图 2.6

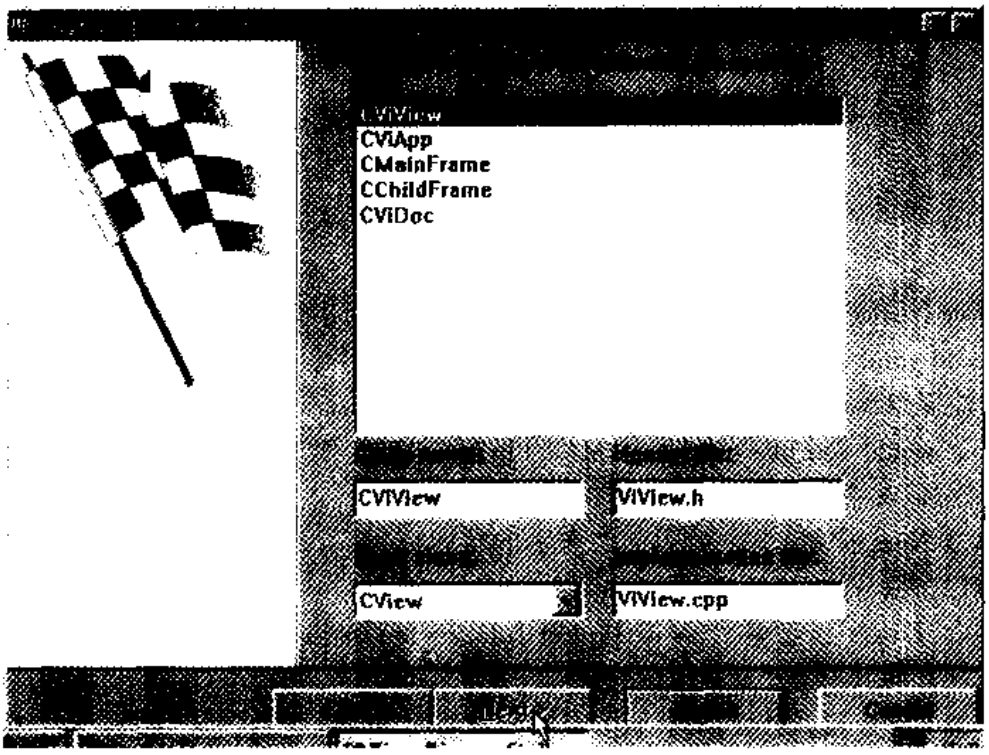


图 2.7　显示 AppWizard 创建的类、头文件实现文件名称对话框

可以从 Build 菜单选择"Execute MyApp.exe"命令来运行程序。

从图中可以看出，该程序包括标题栏、工具栏、状态栏和菜单栏，作为一个 MDI 应用程序，程序还包含一个打开的文档边框窗口，此外，实际操作一下该程序，还可发现有以下特点：

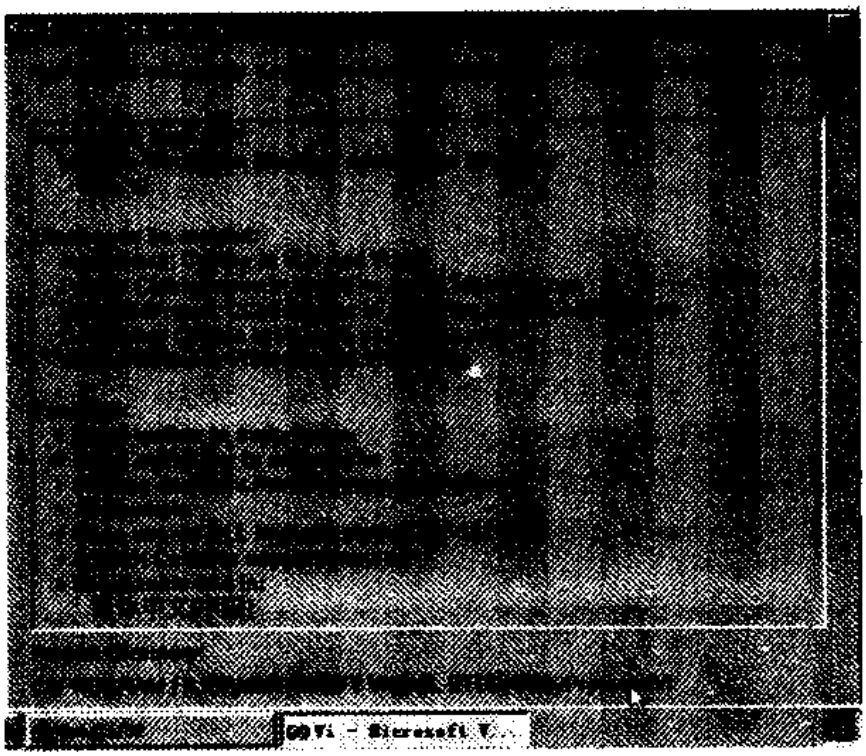


图 2.8 创建完成对话框

1)用"文件"菜单的"新建"命令打开新的文档窗口,如果已有多个窗口,可用"窗口"菜单上命令来更新排列窗口。

2)由于 AppWizard 未提供"编辑"菜单命令实现代码,因此该菜单上的命令不可执行

3)"文件"菜单上的"打开","保存","另存为"命令已经实现,读者不妨可以单击任一命令看看运行情况。

4)"文件"菜单中"关闭","打印","打印预览","打印设置"和"退出"命令已经实现

5)"文件"菜单中"最近文件"命令是位置句柄,用于列出最近使用过的文件。

6)"查看"菜单上的命令用于显示或隐藏工具栏或状态,这些命令已经实现。

7)"帮助"菜单的"关于"命令已经实现。用于显示关于对话框。

8)工具栏上的按钮已经实现,其功能类似于相应的菜单命令。

9)状态栏用于显示工具栏按钮的功能,是对菜单命令的功能描述。

2.3 ClassWizard

使用 ClassWizard 可以创建新类,定义消息处理函数,覆盖 MFC 虚拟函数,从对话框、表单视图(Form View)或者记录视图(Record View)的控件中获取数据。

ClassWizard 主要有以下功能特点:

- 创建新类:它是从处理 Windows 消息和记录集(Recordset)的主框架基类派生来的
- 映射消息给与窗口、对话框、控件、菜单选项和加速键有关的函数。
- 创建新的消息处理函数。
- 删除消息处理函数。
- 查看已经拥有处理函数的消息并将它加到相应的处理代码中去。

• 定义成员变量用于自动初始化,收集并验证输入到对话框或表单视图中的数据。

• 创建新类时,添加自动化(Automation)方法和属性。

从 View 菜单选择 ClassWizard 命令,将弹出 MFC ClassWizard 对话框。MFC ClassWizard 对话框包括五个选项卡:"Message Maps","Member Variables","Automation","ActiveX Events"和"Class Info"。

2.3.1 用 Class Wizard 创建新类

使用 ClassWizard,可以添加无错的类声明到项目中。多数可用的 ClassWizard 创建的类都是从 CmdTarget 类派生的用户界面类。要用 Class Wizard 创建新类,其方法为:

1)从 View 菜单选择 Class Wizard 命令,弹出 MFC ClassWizard 对话框。

2)单击"Addclass"按钮,从弹出菜单中选择 New 命令。

3)选择后弹出 New Class 对话框,如图 2.9 所示。

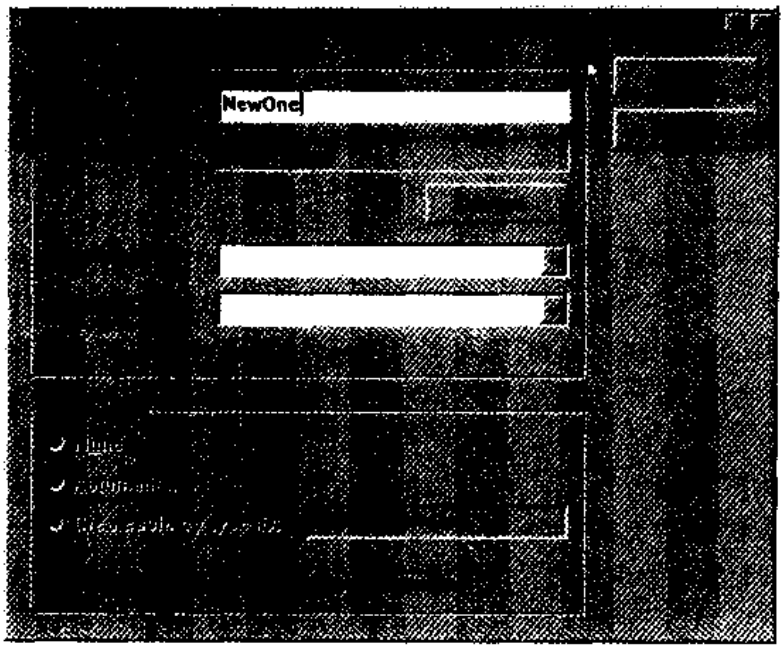


图 2.9 New Class 对话框

4)在 Name 文本框中输入新的类名称,如"NewOne"。从"Base Class"下拉列表框选择用于派生新类的基类,如"COne"。

5)若要改变头文件(.h)或者实现文件(.cpp)的缺省名字,请单击"Change"按钮,弹出"Change Files"对话框。在对应的文本框中输入相应的名字。更改后单击"OK"按钮返回。

6)从 Automation 框选择以下选项之一:

• 选择 None 选项,则不运行自动化。

• 选择 Automation 选项,则可通过自动化来扩展该类的功能,新创建的类可以用作自动化客户应用程序的可编程对象。

• 选择"Creatable by type ID"选项,则允许其他应用程序通过使用自动化来创建该类的对象。

7)单击 OK 按钮,Class Wizard 自动将该类的框架信息加到相应的头文件和实现文件中。

如果新创建的类实现 ActiveX 控制功能,则可使用"MFC ClassWizard"对话框的 ActiveX Events 选项卡来添加 ActiveX 文件,这些文件在某一特定动作派生时被初始化。

2.3.2 定义消息处理函数

创造新类或从其他项目导入类后,可用 ClassWizard 浏览有关的消息并快速定义相应的消息处理函数。其方法为:

1)从 View 菜单选择 ClassWizard 命令,弹出 MFC Class Wizard 对话框。

2)切换到"Message Maps"选项卡。

3)从"Class Name"下拉列表框中选择要处理的类名。

4)在 Object IDS 框中选择要定义消息处理函数的用户界面对象。

5)在 Message 框中选择要定义消息处理函数的消息,然后单击"Add Function"按钮(或双击消息名)。

6)若消息没有预定义处理函数名,则弹出"Add Member Function"对话框,键入消息处理函数名并按回车,或直接按 OK 按钮以接受 ClassWizard 建议的名字。

7)如果要添加多个消息处理函数,请重复 5)、6)两步操作,图 2.10 为添加多个消息处理的结果。

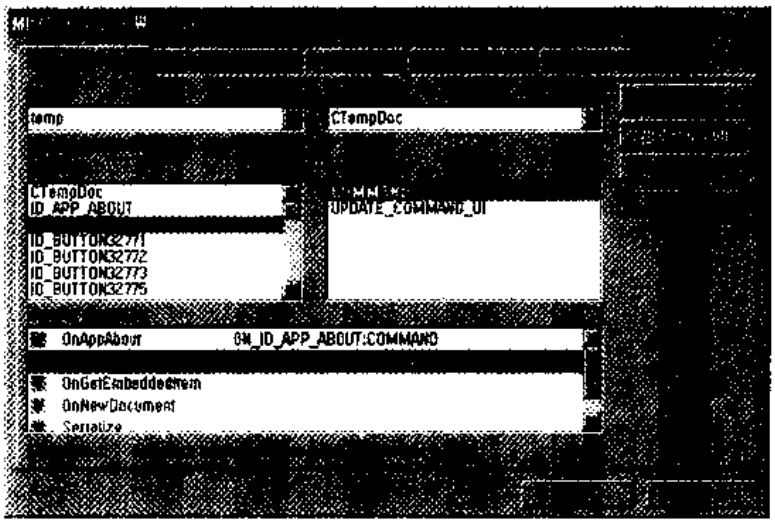


图 2.10 添加多个消息处理

8)单击 OK 按钮,Class Wizard 将往头文件中插入函数声明,往实现文件中插入完整正确的函数定义,并给出实现框架,更新的消息映射的包含新的消息处理函数。

2.3.3 删除消息处理函数

用 ClassWizard 删除消息处理函数时,首先必须从文件中删除函数定义及所有对函数的引用,具体步骤为:

1)在 MFC ClassWizard 对话框中选择 Message Maps 选项卡。

2)在 Class name 下拉列表框中选择被删除消息处理函数所在的类。

3)在 Member function 框中选择要删除的成员函数的名称。

4)单击"Edit Code"按钮打开成员函数所在的实现文件,删除函数定义的函数体。

5)返回 MFC ClassWizard 对话框并单击"Delete Function"按钮,系统自动从消息映射中删除有关成员函数的条目。

2.3.4 编辑消息处理函数

使用 ClassWizard 可以编辑消息处理函数,方法为:

1)在 MFC ClassWizard 对话框中选择"Message Maps"选项卡。

2)在 Class Name 下拉列表框中选择消息处理函数所在的类。

3)在 Member function 框中选消息处理函数的名称。

4)单击"Edit code"按钮或双击函数名跳转到成员函数的开始处,从中可以根据需要编辑修改消息处理函数的源代码。

2.3.5 覆盖虚拟函数

使用 ClassWizard 可以覆盖基类中定义的虚拟函数,具体方法为:

1)在 MFC ClassWizard 对话框中选择"Message maps"选项卡。

2)在 Class Name 下拉列表框中选择一个类号用作添加要覆盖的虚拟函数。

3)在 Object IDS 框中再次选择该类名,系统列出所有可供覆盖的虚拟函数及所有的标准的 Windows 消息。

4)在 Message 框中选择要覆盖的虚拟函数名。

5)单击"Add Function"按钮,这样系统将创建该虚拟函数并在"Member functions"框中列出相应的函数名。

6)单击"Edit Code"按钮跳转到相应的处理代码中去。

2.4 可视化编程窗口

AppWizard 在生成应用程序的框架和工程文件后,自动回到 Visual C++ 6.0 工作台,并可调入新的工程工作区。由此进入了编写可视化程序的主窗口—可视化编程窗口,本节将就该窗口的各个部分作一介绍。

2.4.1 工作区窗口

如前所述,工作区窗口有三个列表,下面分别描述。

1)Class View

显示工程中的所有类及其成员函数,单击"+"号,依次打开树形结构的每一项,显示出所有的成员函数和成员变量,双击其中一项,就会在右边的工作台文本的编辑窗口显示该成员的源代码。在 ClassView 中,快捷菜单(即按键弹出的菜单)根据所选的目标不同而动态地改变。当选择的是一个工程时可通过快捷菜单增加一新类;当选择的是一个派生类时,可通过快捷菜单增加成员函数或成员变量,当选择的是一个 COM 界面时,可通过快捷菜单给它增加属性和方法;对于 ActiveX Control 界面,你同样可定义新的文件。

2)File View

显示工程中所有文件及其相关联系，单击“+”号，依次打开树形结构的每一项，显示出所有资源文件、头文件和源代码文件。双击其中任一项，就会在右边工作台编辑窗口中打开这个文件，显示它的源代码。

3)Resource View

显示工程中的所有资源，单击“+”号依次打开树形结构每一项，直到所有的“+”号都变成“-”号，显示出了所有的资源，包括对话框图符、字符串表和版本信息。双击某项，就会显示该资源的图形编辑窗口，可直接在这个窗口上增加和修改资源特征。

2.4.2 常用控件

在进行程序设计时，我们不免会应用到可视化环境为我们设置的新颖的控件，在 Visual C++ 6.0 中，在其右边的工作台中显示一个可视化的对话框编辑窗口，并打开一控件工具窗口，(参见图 1.25)。

下面就这些控件逐一介绍。

1)静态文本框(Static Text)

用于提供一个只读性质的文本信息，不可编辑。但它与只读的正文编辑框不一样，在这里文本框中信息是不可选的。其可用于显示当前的目录路径、按钮状态或者时间和日期等。

2)正文编辑框(Edit Box)

这是一矩形的控件，用户可以在其中进行对正文的输入和编辑。但若将其定义成只读正文，就不能对编辑框中的内容进行编辑。通过属性设定，可把其定义成支持单行或多行文本。

对正文编辑框操作包含字符的插入和删除，没有提供对独立字模或者段落属性的支持，整个控件只支持一种特定的字模设置。

3)分组框(GroupBox)

此控件不对任何输入进行响应，它可用于对其他控件进行组织。可将一组属性相同的控件放在一个分组框中，通过分组框的标签来说明这一组控件，与其他类控件区分。

4)命令按钮(Command Button)

亦称为下推按钮，是最常用的控件之一。可以显示文本或图形。当给这个按钮发送一个消息时，就会去执行相关命令。

当按下此按钮的时候，其外观就会呈现被压下去的样子，此时若移开鼠标指针，按钮又会恢复原状。而只有按钮被压下去才会响应消息。

5)复选框(Check Box)

其支持的选项有两种状态：开/关。当方框中显示的是 X 时，表示开；空时，表示关。一个标准的复选框应包含对应标签，标签是以文本的形式显示出来的，一般为单行标签。

6)单选按钮(Radio Button)

其表示一个互相排斥的选项有限集合中的一个独立选项，任何一组单选按钮中只能对组件中的一个选项进行设置。故单选按钮通常是以两个或多个按钮形式出现。若被选择，其圆圈的中间就会出现一个小圆点，反之则没有。一个标准的单选按钮应包含对应的标签，标签是以文本的形式显示出来的，一般用单行标签。

7)组合框(Combo Box)

组合框把一个正文编辑框特性和一个下拉列表框特性组合在一起。这样，用户可以在列表中选择项或者输入一项。

8)列表框(List Box)

每一个下拉列表框中都有一个垂直滚动条,可通过滚动条中的箭头来选择列表中的某一项,也可直接单击某项来选择,被选的那一项被亮化。其用途是显示一系列选项的集合。支持对表中一个选项或多个选项进行选定。

9)水平滚动条(Horizontal Scroll Bar)和垂直滚动条(Vertical Scroll Bar)滚动条两端各有一箭头按钮,单击两端按钮时,槽中方块就按箭头所指方向移动。

10)选值框(SpinBox)

其作用是接收不连续输入值的一个有限集合,它是由一个编辑框和一个上下按钮组成的可把一个值输入到此框,也可利用按钮来增大或减小这个值。

11)进展指示条(Progress)

这个控件由一矩形条组成,在工作时矩形块呈从左到右逐渐填充矩形条。作用是显示一个执行时间比较长的过程完成了多少。

12)滑杆(Slider)

利用滑杆可以对一个连续的值(例如音量)进行设置或者调整。滑杆可以被定义很多属性,可把它的方向设置为水平或者垂直,对滑杆指示标志和滑杆组件的宽度和高度进行定义,对滑杆的改变条长进行定义或决定是否为滑杆显示它的步长标号等。

13)热键控件(Hot key)

其作用是为用户通过按键或按键组合来定义一个快捷菜单分配方案提供支持,用户若要为某个界面定制由应用程序所支持的快捷键就需要用到这个控件。

14)列表查看控件(List Control)

其作用是显示一列图标和标签组成的选项的集合,可通过不同的查看方法来显示其中的内容、图标、小图标、拟表、列表。

15)树形查看控件(Tree Control)

这是一特殊的列表框控件,它是根据对象的逻辑层次关系,采用一种"树"形轮廓图显示各个对象。

16)制表文控件(Tab Control)

这个控件类似于一个基体里的分隔物或文件框里的文件夹,使用它可同时显示多个信息页和控制页,页与页间是对等逻辑关系。

17)动画显示控件(Animate)

这个控件是一个显示 AVI 格式画面的窗口,AVI 格式是 Windows 标准的动画显示格式。通常用于在进行一个比较长的操作时,指示系统的活动情况。这种控件只能播放简单的 AVI 图画,不支持声音的播放。

18)多信息编辑框(Rich Edit)

该控件与正文编辑框类似,它还可以支持每个字符和段落格式的字模属性。而且还能支持对其中信息的打印及 OLE 对象的嵌入。

19)用户控件(Custom Control)

其功能是让你自己制作控件,而非使用 Windows 系统为你提供的那些控件。

以上介绍的每一控件在 Vect 的 MFC 类中都有相对应的基类,在实际应用中必须熟悉每个类库中的成员函数,不能调用其他类中函数,下表列出每个控件所对应的基类,以供参考。

表 2.1 部分控件所对应的 MFC 基类

控件	MFC 类	控件	MFC 类
Animation	CAnimateCtrl	Richedit	CRichEditCtrl
Button	CButton	Scroll Bar	CScroll Bar
Combo Box	CComboBox	Slider	CSliderCtrl
Edit Box	CEdit	Spin Button	CSpinButtonCtrl
Hot Key	CHotkeyctrl	Static Text	CStatic
List	CListctrl	Tab	CTabctrl
List Box	CList Box	Tree	CTreectrl
Progress	CProgressctrl		

2.4.3 控件属性

在设计应用程序时，大家肯定会用到某些控件，因而，了解它们的属性对很好地运用它们将会大有益处。

单击控件窗口上的命令按钮控件，选中并放置到编辑对话框的合适位置。在此按钮上单击右键，在弹出的下拉菜单中选择 Properties 项，就会显示一属性窗口，下面对控件的属性进行介绍。

选择 General 选项卡(如图 2.11 所示)。

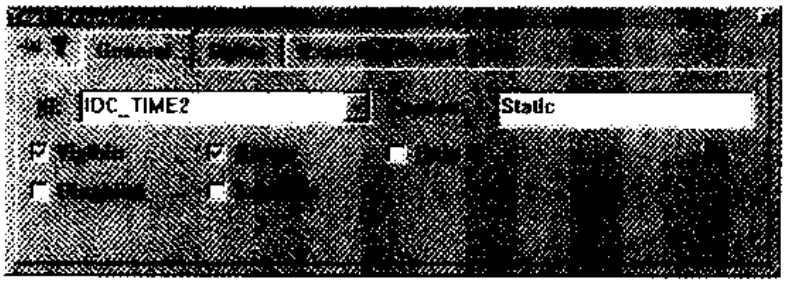


图 2.11 General 对话框

在该页中，各个选项含义如下(缺省值 True 为选中，False 为不选)：

1)ID：在头文件中定义的控件代号。

2)Captions：标记这个控件的事件，即显示在控件上的文本。

3)Visible：定义该控件在应用程序运行时是否可见。

类型 Bool，缺省值 True。

4)Disabled：在创始对话框时，定义该资源是否为不可选，即是否为无效。类型 Bool，缺省值 False。

5)Group：表示控件中的第一个控件，在这组控件中，用户可用键盘上的箭头按键在各个控件间移动。类型 Bool，缺省值 False。

6)Tab Stop：表示用户可通过 TAB 键移到该控件上。

7)Help ID：赋给控件一个帮助 ID，这个帮助 ID 是基于资源 ID 的。类型：Bool，缺省值 False。

在 Styles 选项卡中，各选项含义为：

1)Default Button:若为 True 时则该控件为这个对话框的缺省按钮。用户在对话框出现时只要按 Enter 键就会执行此按钮。Windows 只允许一个对话框拥有一个缺省按钮。

类型:Bool,缺省值 False。

2)Owner Draw:创建一个用户自画的控件。

3)Icon:表示按钮上显示的是一个图符,而非文本。类型 Bool,缺省值 False。

4)Bitmap:表示按钮上显示的是一个图标,而非文本。

5)Multiline:表示若文本中的字符太长不能在按钮矩形框的一行显示出来的话就使用多行显示。

类型 Bool,缺省值 False。

6)Notify:若命令按钮被单击或双击,就通知父窗口。

类型 Bool,缺省值 False。

7)Flat:使按钮看起来是平面的而不是三维的。类型 Bool,缺省值 False。

8)Horizontal alignment:定义控件的 Caption 文本显示在控件的左边、中间、右边还是缺省位置。

9)Vertical alignment:定义控件的 Caption 文本显示在上面、中间、下面还是缺省位置。

在 Extended Styles 选项卡中,各项含义为:

1)Client edge:创建边框下凹的效果。

类型 Bool,缺省值 False。

2)Static edge:创建一普通边框。

类型 Bool,缺省值 False。

3)Modal frame:提供一个三维效果的边框。

4)Transparent:使用此风格的窗口是透明的,它下面的其他窗口都不能被遮住。类型 Bool,缺省值 transparent。

5)Accept files:使用这种风格的对话框允许施拉文件。

类型 Bool。缺省值:False。

6)No Parent Notify:表示窗口不会发送一个 WH-PARENTNOTIFY 消息给父窗口。类型 Bool,缺省值 False。

7)Right aligned:表示对话框中的文本为右对齐。

类型 Bool,缺省值 False。

8)Right-to-left reading order:表示对话框中的文本是从左到右显示。类型:Bool,缺省值 False。

了解上述各选项卡中的选项含义后,在以后编程过程中应会灵活应用,按照自己的需要灵活选择。

2.5 集成调试(Debug)

编写程序总不可能一次性编写完运行即成功总会不免进行调试工作,熟练掌握编程工具提供的调试器对一个专门从事编程的人员至关重要,程序中错误的数量会随其规模的增大而增多。Visual C++ 6.0 的集成调试器,用于帮助定位可执行文件、动态链接库(DLL)、多线程、OLE 客户或服务器的错误,本节将介绍 Debug 菜单和工具条的使用及如何设置断点和使

用多个调试窗口来观察调试的过程。

2.5.1 Debug 菜单工具条

在主窗口 Build 菜单中有 Start Debug 项，当鼠标移到上面时会弹出级联菜单，有四个菜单项，Go、StepInto、Run to Cursor 和 Attach to Process…。其中，Go 选项从当前语句开始执行程序，直到遇到断点或遇到程序结束；Step Into 选项用于单步执行程序，并在遇到函数调用时进入函数内部，再从头单步执行；Run to Cursor 选项用于执行程序，使程序在运行到当前光标所在位置时停止，相当于设置一个临时断点；Attach to Process 选项在调试过程中直接进入到正在运行的过程中。

启动调试器后，Debug 菜单将代替 Build 菜单出现在菜单栏中。Debug 工具条也随即出现。Debug 菜单中包括调试过程经常需要用到的命令选项，下面对每一菜单项进行说明。

1. Go 选项

该选项用于在调试过程中从当前语句启动或继续运行程序（等价于 Build 工具栏的“Go”按钮）。执行时，程序会一直正常运行，直到运行到断点处为止。这样，在调试过程中，就可以越过某些已知正确或不感兴趣的程序段，从而提高调试速度。

2. Restart 选项

在调试选项中，我们经常进行这样一种循环操作，首先找到一条“错误”语句，接着对其做些修改，然后再从头开始对程序进行调试运行，从而确定刚刚修改的语句是否按预期的结果执行。这时，就可以使用 Restart 选项。选择该选项，系统重新装载程序到内存并放弃所有变量的当前值。

3. Stop Debugging 选项

该项用于中断当前的调试过程并返回正常的编辑状态。

4. Breaks 选项

在当前位置暂停程序运行。

5. Step Into 选项

在调试过程中单步执行程序，但当程序执行到某一函数调用语句时进入该函数中，从头单步执行。

6. Step Over 选项

在调试过程中单步执行程序，但当程序执行到某一函数调用的语句时，不进入函数内部，而是直接执行该调用语句，接着再执行调用语句后面的语句。

7. Step Out 选项

该选项与 Step Into 选项配合使用，当执行 Step Into 命令进入函数内部，并开始单步执行时，若发现并不需要对该函数的内部进行单步调试，那么就可使用“Step Out”选项使程序直接

向下进行，直到从该函数内部返回，在该函数调用语句后面的语句处停止。

8. Run to Cursor 选项

该选项用于在调试运行程序时，使程序运行到当前光标所在位置时停止。事实上，这相当于设置一临时断点。

9. Step Into Specific Function 选项

该选项用于单步执行选定的函数。

10. Exceptions 函数

选择该项将弹出 Exeptions 对话框，显示与当前程序有关的所有异常，可以控制调试器如何处理系统异常和用户自定义异常。

11. Threads 选项

选择该项将弹出 Threads 对话框，显示调试过程中可用的所有路程。可以挂起和恢复路程并设置焦点。

12. Show Next Statement 选项。

该选项用于显示正在执行的代码行。

13. Quick Watch 选项

选择该项将弹出 Quick Watch 对话框，可以查看及修改变量和表达式或将变量和表达式添加到 Watch 窗口。

Debug 工具条除包括 Debug 菜单的大部分菜单项之外，还含有下列几个选项：

- Watch 窗口，显示变量和表达式的名字和值。
- Variables，显示在当前执行语句中使用的变量的信息以及函数返回值（在 Auto 标签中），当前的本地变量（在 Locals 标签中），this 指针所指的对象（在 This 标签中）。
- Registers，显示通用目的和 CPU 状态寄存器的内容。
- Memory，显示当前内存内容。
- CallStack，显示没有返回的所有函数调用的堆栈。
- Disassembly，显示从编译程序的反汇编中获取的汇编语言代码。

2.5.2 设置断点

当程序运行到一个断点时就会停下来，这时可以检查程序的状态，运行代码，以及调式窗口估计表达式的值。调试器支持的断点类型有：

- 位置断点（无条件断点）：在一个特定的位置中断程序。
- 数据断点：当表达式为真或值改变时，中断程序。
- 消息断点：当在一个 WndProc 函数中接受一个消息时，中断程序。
- 条件断点：在指定位置，当表达式为真或值改变时，中断程序。

点击 Edit 菜单的 Breakpoints 项，会弹出 Breakpoints 对话框，该对话框有三个选项卡：Lo-

cation、Data、Messages，分别设置位置断点、数据断点和消息断点，条件断点的设置必须先设置位置断点，然后单击 Condition 按钮，从弹出的 Breakpoint Condition 对话框来指定程序执行的条件。也可通过在源代码的编辑窗口单击鼠标右键，弹出一快捷菜单，选 Insert/Remove Breakpoint 来快速插入或删除无条件断点。

设置断点时，有一定的断点位置格式，工作平台集成调试器支持的各种断点位置的定义格式说明如下。

表 2.2　集成调试器支持的断点位置

格式	例子	说明
·行号	·125	当前源文件的第 125 行
文件名! 行号	CTL.CPP! 90	CTL.CPP 文件的第 90 行
偏移地址	OX1A34	代码段中偏移地址为 OX1A34 行
偏移地址	IP	以指令指针寄存器内容为偏移地址
段地址:偏移地址	OXA100:OX1A34	段地址为 OXA100，偏移地址为 OX1A34
段地址:偏移地址	CS:OX1A34	段地址为 CS 内容，偏移地址为 OX1A34
段地址:偏移地址	CS:IP	分别以 CS 和 IP 作段地址和偏移地址

2.6　“Edit”菜单

在进行程序编写中使用频率最高的菜单栏莫非 Edit 菜单了，本节我们将对“Edit”菜单进行介绍以便更好地编辑自己的程序。

在“Edit”菜单中包含用于编辑或者搜索的命令选项(如图 2.12 所示)。

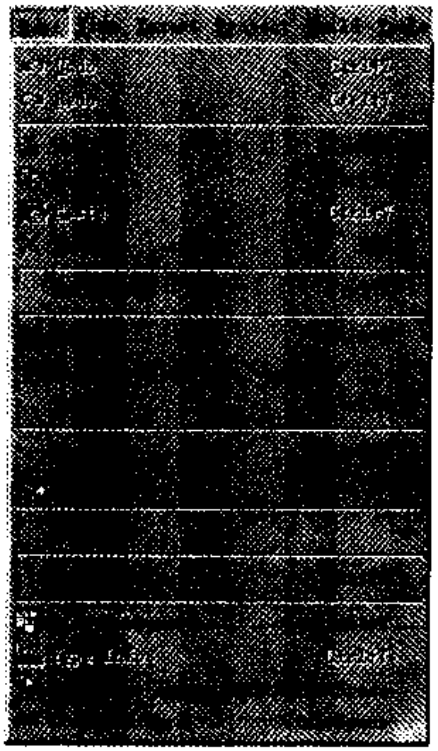


图 2.12　Edit 菜单

1."Undo"选项

该选项用于取消最近一次的编辑修改操作。

2."Redo"选项

该选项用于取消最近一次的"Undo"操作,可以恢复被"Undo"命令取消的修改操作。

3."Cut"选项

该选项将当前活动窗口中被选定的内容拷贝到剪贴板中,然后再将其从当前活动窗口中删除。可以用鼠标选定所要剪切的内容,方法为将光标定到要剪切内容的开始并按住鼠标左按钮,然后拖动鼠标移到要剪切内容的末尾,这时,整个被选内容就会反白显示。"Cut"选项通常与"Paste"选项联合使用来移动选定的内容。

4."Copy"选项

该选项将当前活动窗口中被选定的内容拷贝到剪贴板中,但并不将其从当前活动窗口中删除。"Copy"选项通常与"Paste"选项联合使用来复制选定的内容。

5."Paste"选项

该选项用于将剪贴板中的内容插入到当前光标所在的位置,必须先剪切或者复制选定内容到剪贴板后,才能进行粘贴。

6."Delete"选项

该选项用于删除被选定的内容。删除以后,还可以使用"Undo"命令来恢复删除操作。

7."Select All"选项

该选项用于选择当前活动窗口中的所有内容。

8."Find"选项

该选项用于查找指定的字符串。选择"Find"选项将弹出"Find"对话框。

在"Find"对话框中,可以在"FindWhat:"文本框中输入欲查找的字符串,并设置查找的方向(Up或Down)。此外,还可以根据需要进行区分大小写字符串查找(Match Case)或进行整词查找(Match Whole Word Only),这时就应选择相应的复选框。对话框中的"Regular Expression"选项特别有用,选择该项将按正则表达式来查找文件中匹配的文本。正则表达式是指用特殊的字符序列去匹配文件中的某个文本模式。表2.3列出了正则表达式的查找模式及其含义。

表2.3 正则表达式的查找模式

查找模式	含义	示例
*	匹配任意多个字符	Exam * 匹配 Exam,Example 和 Examl
.	匹配单个字符	Exam . 匹配 Exam2 和 Examp,不匹配 Example

续表

查找模式	含义	示例
^	匹配以指定字符串开头的每一行	^Exam 匹配以 Exam 开头的各行
+	匹配以指定字符串结束的字符串	+ Exa 匹配 is_ Exam 和 notExam
$	匹配以指定字符串结束的每一行	Exam $ 匹配以 Exam 结束的各行匹配指定字符集中的字符 Exam[1...9]匹配 Exam5,但不匹配 ExamS
\	匹配与指定字符一致的字符	Exam[A-Z n \ 0...9_可匹配 ExamSn5.但不匹配 ExamAm9
\ { \ }	匹配花括号间指定字符串的任意序列	Exam_ \ { # } \	* 可匹配 Exam _,Exam _ \ # 和 Exam _ # \ # }

9."Find in Files"选项

该选项用于在多个文件间搜索文本,而且搜索的对象可以是文本字符串,也可以是正则表达式。

10."Replace"选项

该选项用于替换指定的文本串。选择"Replace"选项将弹出"Replace"对话框。可以在"Find What:"文本框中输入被替换的文本串,再在"Replace With:"文本框中输入替换的文本串。与"Find"选项一样,也可以选择匹配的方式(如整字匹配、大小写区分或不区分、正则表达式等)。

11."Go To"选项

选择该项将弹出"Go To"对话框,可以指定如何将光标移到当前活动窗口的指定位置(如指定的行号、地址、书签、InfoViewer 书签、对象的定义位置、对象的引用位置等)。

12."Bookmarks"选项

选择该项将弹出"Bookmarks"对话框,可以设置或取消书签。书签用于在源文件中做标记。

13."Advanced"选项

选择该项将弹出级联菜单,其中包含有用于编辑或者修改的高级命令。例如,增量式搜索、将选定内容全部转换为大写或者小写、显示或者隐藏制表符等。

14."Breakpoints"选项

选择该项将弹出"Breakpoints"对话框,可以设置、删除和查看断点。

断点实际上是告诉调试器应该在何时何地中断程序的执行过程,以便检查程序代码、变量和寄存器值,必要的话可以修改、继续执行或中断执行。在 Visual C++ 6.0 中,断点分为位置断点、数据断点、消息断点和条件断点等类型。所有已设置的断点都出现在"Breakpoints"对话框底部的"Breakpoints"列表中。可以使用"Breakpoints"列表检查程序中的所有断点,也可以从列表中删除某一断点。

位置断点通常在源代码的指定行、函数的开始或指定的内存地址处设置，当程序执行到指定位置时，位置断点将中断程序的执行。数据断点是在某一变量或表达式上设置，当变量或表达式的值改变时，数据断点将中断程序的执行。消息断点是在窗口函数 WndProc 上设置，当接收到指定的消息时，消息断点中断程序的执行。条件断点是一种位置断点，仅当指定的条件为真时中断程序的执行。

"Breakpoints"对话框的"Location"、"Data"和"Messages"选项卡分别用于设置位置断点、数据断点和消息断点。条件断点的设置必须先设置位置断点，然后单击"Condition"按钮，从弹出的"Breakpoint Condition"对话框来指定中断程序执行的条件。

2.7 Box 例程序

Box 这个例子充分演示了平常普通使用的组合框和列表框控件的使用方法，如图 2.13 所示。这个小程序是基于对话框的，在对话框的左边是一组合框，其数据项的来源是通过在该组合框上面的编辑框输入字符串，再按其下面的"插入"按钮，如果编辑框内的字符串与组合框已存在的数据项不完全一样，组合框会增加一项包含编辑框内字符串的项。靠对话框右边是一可多选的列表框，它通过中间的四个位图按钮从左边的组合框来获取数据。这四个按钮可使数据项在组合框与列表框之间一项或多项来回移动。对话框底下中间的"删除"按钮可删除组合框或列表框内的项。"确定"按钮用于退出程序。

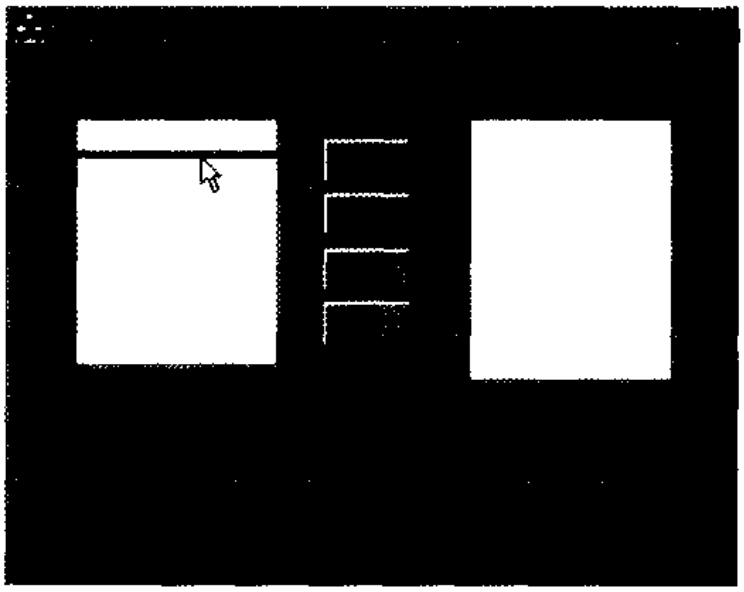


图 2.13 box 例程界面图

下面是创建该例子的步骤。

首先用 MFC AppWizard 创建基于对话框的应用程序，步骤如下：

1)启动 Microsoft Visual Studio。

2)选择菜单 File 的 New 菜单项，弹出 New 对话框。

3)在 New 列表中选择 Projects 选项卡，选择 MFC AppWizard(exe)项，在右边的 Project name 中填入 box。Location 中填入或选择该文件工程要放置的地方，若不填，则会在已出现的

Location 编辑框的目录下自动创建以 Project name 中填入的字符命名的目录。AppWizard 会把自动产生的文件全部自动放于该目录下。

4)点击 OK 按钮,New 对话框关闭,弹出 MFC AppWizard 向导的 MFC AppWizard-stepl 属性页。

5)在应用程序类型中选择 Dialog based。在下面的资源所用类型中选择中文(AppWZCHS.DLL)。

6)其他选项接受缺省设置,点击 Finish 按钮,最后出现下面的 New Project Information 对话框。

7)选项 OK 后,AppWizard 会在一开始我们选择的目录处产生许多相应的文件。若选择 Cancel 则表示放弃创建该工程。

接着编辑对话框资源,步骤如下:

1)点击 ResourceView,切换到资源视窗。

2)点击 Box Resource 节点,打开各资源。

3)点击 Dialog 节点,打开对话框资源。

4)删除对话框缺省的编辑控件"要做:在这里设置对话控件"。

5)在控件面板中拖动鼠标到 ComboBox 处,点击该控件并施放到对话框中,并放大到适当大小。

6)在对话框上的组合框上点击右键,打开组合框的属性对话框,点击 General 标签,切换到 General 属性页。

7)设置 ID 为 IDC_COMBO。

8)点击 Styles 标签,切换到 Styles 属性页。为 Type 属性选择 Simple。

9)在属性对话框外面任意处单击左键,关闭该属性对话框。

10)用类似方法,在控件面板上选取 listBox 控件并将其拖置对话框上,设置 ID 为 IDC_LIST。

11)点击 Styles 标签,在 Selection 中选择 Multiple。

12)删除"取消"按钮,方法是在取消按钮上点属标左键,之后按 Delete 键。

13)加入插入按钮,设置 ID 为 IDC_INSERT,Caption 为"插入"。

14)加入删除按钮,设置 ID 为 IDC_CUT,Caption 为"删除"。

15)添加位图按钮 BTN1,ID 为 IDC_BTN1,Caption 为 BTNI。并在 Styles 属性页中选 Ownerdraw 复选框。

16)重复前一步,添加按钮 BTN2,BTN3,BTN4,ID 分别为 IDC_BTN2,IDC_BTN3,IDC_BTN4。Caption 为 BTN2,BTN3,BTN4,不要忘了选各按钮的 Ownerdraw。

为什么要这样设置,在后面将做解释。

然后添加位图,步骤如下:

1)击点 Insert 菜单的 Resource 项,弹出 Insert Resource 对话框,选取 Bitmap,点击 New 按钮。

2)在右边的 Graphics 绘图面板上选取相应的工具绘制图形。

3)在 IDB_BITMAP1 上点击鼠标右键,选择属性对话框,设量 ID 为"BTN1D",不要丢了双引号。

4)按上述步骤按图,绘制图形"BTN1U"。

5)重复第 1 ~ 4 步,绘制位图"BTN2D","BTN2U","BTN3D","BTN3U","BTN4D","BTN4U"。

在该程序中,我们要使用 4 个位图按钮。MFC 使用 CBitmapButton 类来创建使用位图式图像而非文字标注的按钮控件。CBitmapButton 对象可包含四种位图,包括一个按钮可以假定的各种状态图像:上(或正常)、下(或被选)、拥有焦点和被禁止。只有第一种位图是需要的,其它可选。

位图按钮图像包括图像周围的边框和图像本身。边框通常起一个显示按钮状态的作用。例如,处于拥有焦点状态的位图一般类似于处于向上状态时的位图,但前者从边框嵌入一个虚线矩形或在边框画上粗实线。禁止状态的位图通常类似于向上状态,但颜色对比度较浅(类似于一个变淡或变灰的菜单项)。

这些位图可以是任意尺寸,但全都按向上状态位图的尺寸相同对待。

在创建一个位图按钮控件时,可设置 BS _ OWNERDRAW 风格来指定该按钮是自画式的,它便于 Windows 为此按钮发送 WM _ MEASUREITEM 和 WM _ DRAWITEM 消息,框架处理这些消息,并为应用程序管理此按钮的外观。

要在窗口的客户区中创建一个位图按钮控件,需遵循如下步骤:

1)为按钮创建一到四个位图图像。

2)构造 CBitmapButton 对象。

3)调用 Create 函数创建 Windows 按钮控件,并将它连接到步骤 2 中构造的 CBitmapButton 对象上。

4)当位图按钮构造好之后,调用 LoadBitmaps 成员函数装载这些位图资源。

要在对话框中包含位图按钮控件,需遵循以下步骤:

1)为按钮创建一至四个位图图像。

2)创建一个对话模板,在需要位图按钮处定位一个自画式按钮。模板中按钮的大小无关紧要。

3)设置按钮标题为类似"BTN"的值,并为按钮定义一个类似 IDC _ BTN 的标识符。

4)在应用程序的资源表中,为按钮创建的每个图像提供一个 ID,ID 是在步骤 3 中用作按钮标题的字符串后附加一个字母"U"、"D"、"F"或"X"(分别代表上、下、拥有焦点和禁止)形成的。例如,对于按钮标题"BTN",其 ID 应该是"BTNU","BTND","BTNF"和"BTNX"。

5)在程序的对话类(派生于 CDialog)中,增加一个 CBitmapButton 成员对象。

6)在 CDialog 对象的 OnInitDialog 过程中,调用 CBitmapButton 对象的 AutoLoad 函数,其所用参数为该按钮的控制 ID 和 CDialog 对象的 This 指针。

如果应用程序想要处理 Windows 通知消息。例如 BN _ CLICKED,它由位图按钮控制发送给它的父窗口(通常是一个 CDialog 派生类),则在 CDialog 派生对象中增加一个消息映射项,并为每条消息增加消息处理成员函数。由 CBitmapButton 对象发送的通知与 CButton 对象发送的通知相同。

之后用 ClassWizard 添加控件变量,步骤如下:

1)按 CTRL 键的同时双击 IDC _ COMBO 组合框控件,弹出 Add Member Variable 对话框。

2)在 Member Variable name 项中输入 m _ cmb。

3)接受 Category 项为缺省 Control。

4)接受 Variable Type 项为缺省 CComboBox。

5)点击 OK 按钮,关闭 Add Member Variable 对话框。

6)按 CTRL 键的同时双击列表控件 IDC _ LIST,弹出 Add Member Variable 对话框。

7)在 Member Variable name 项中输入 m _ list。

8)接受 Category 项为缺省 Control。

9)接受 Variable type 为缺省 CListBox。

10)点击 OK 按钮,关闭 Add Member Variable 对话框。

用 ClassView 添加成员变量 btn1、btn2、btn3、btn4,步骤如下:

1)在工程工作空间点击 ClassView,切换到 ClassView 视窗。

2)选择 CBoxDlg,点击鼠标右键,弹出快键菜单。

3)选择 Add Variable,弹出 Add Member Variable 对话框。

4)在 Variable Type 文本框中,输入 CBitmapButton。

5)在 Vaiable Declartion 文本框中,输入 btn1。

6)点击 OK 按钮,关闭 Add Member Variable 对话框。

7)重复 2~6 步,增加变量 btn2、btn3、btn4。

在 CBoxDlg::OnInitDialog()中添加初始化代码。

在前面的叙述中已说明创建位图按钮时需要把按钮的位图加到相应的控件之上。在 OnInitDiaLog 成员函数的注解行"//TODO:Add extra initialization here"后加入下列代码:

```
btn1.AutoLoad(IDC_BTN1,this);
btn2.AutoLoad(IDC_BIN2,this);
btn3.AutoLoad(IDC_BTN3,this);
btn4.AutoLoad(IDC_BTN4,this);
```

用 ClassWizard 处理按钮 IDC _ INSERT 的消息,步骤如下:

1)选择菜单 View 的 ClassWizard 菜单项,打开 ClassWizard。

2)点击 Message Maps 标签,切换到 Message Maps 属性页。

3)在 Class Name 列表框中选择 CBoxDlg。

4)在 Object IDS 一栏中选择 IDC _ INSERT。

5)在 Messages 一栏中选择 BN _ CLICKED。

6)选择 Add Function 按钮,弹出 Add Member Function 对话框,在 Member function name 中是建议使用的函数名 OnInsert,点击 OK 按钮接受它。

7)点击 Edit Code 关闭 ClassWizard。并将光标指向 OnInsert 函数的实现位置。

8)将语句"//TODO:Add your control notification handler code here"删去,并加上如下代码:

```
void CBoxDlg::OnInsert()
{
    CString str;
    m_cmb.GetWindowText(str);
    if(str.IsEmpty())
    {
```

```
            MessageBeep(MB_ICONEXCLAMATION);
            return;
        }
        if(m_cmb.FindStringExact(-1,str)<0)
        {
            m_cmb.InsertString(0,str);
            m_cmb.SetCurSel(0);
        }
    }
```

用 ClassWizard 按前面同样的方法处理按钮 IDC_CUT,IDC_BTN1,IDC_BTN2,IDC_BTN3 和 IDC_BTN4 的消息 BN_CLICKED,要添加的处理代码如下:

```
void CBoxDlg::OnCut()
{
    int count=m_list.GetSelCount();
    if(count>0)
    {
        int * rgIndex=new int[count];
        m_list.GetSelItems(count,rgIndex);
          int j=count-1;
        for(int i=0;i<count;i++)
            m_list.DeleteString(rgIndex[j--]);
         delete rgIndex;
    }
    int index=m_cmb.GetCurSel();
    if(index!=CB_ERR)
    {
        m_cmb.DeleteString(index);
        m_cmb.SetCurSel(0);
    }
}

void CBoxDlg::OnBtn1()
{
    CString str;
    int count=m_cmb.GetCount();
    for(int i=0;i<count;i++)
    {
        m_cmb.GetLBText(i,str);
        if(m_list.FindStringExact(-1,str)<0)
            m_list.InsertString(i,str);
    }
    m_list.SetCurSel(0);
    m_cmb.ResetContent();
```

```
}

void CBoxDlg::OnBtn2()
{

    CString str;
    int index=m_cmb.GetCurSel();
    if(index==CB_ERR)
        return;
    m_cmb.GetLBText(index,str);
    if(m_list.FindStringExact(-1,str)<0)
    {
        m_list.AddString(str);
        m_cmb.DeleteString(index);
    }
    m_cmb.SetCurSel(0);

}

void CBoxDlg::OnBtn3()
{
    CString str;

    int count=m_list.GetCount();
    for(int i=0;i<count;i++)
    {
        m_list.GetText(i,str);
        if(m_cmb.FindStringExact(-1,str)<0)
            m_cmb.InsertString(i,str);
    }
    m_list.ResetContent();
    m_cmb.SetCurSel(0);

}

void CBoxDlg::OnBtn4()
{
    int count=m_list.GetSelCount();
    int * rgIndex=new int[count];
    m_list.GetSelItems(count,rgIndex);
    for(int i=0;i<count;i++)
    {
        CString str;
```

```
            m_list.GetText(rgIndex[i],str);
            if(m_cmb.FindStringExact(-1,str)<0)
            {
                m_cmb.InsertString(0,str);
            }
        }
        for(i=0;i<count;i++)
            m_list.DeleteString(rgIndex[i]);
        m_list.SetCurSel(0);
        delete rgIndex;

}
// boxDlg.h : header file
//

#if !defined(AFX_BOXDLG_H__836AEAE8_25BB_11D2_BA07_444553540000__INCLUDED_)
#define AFX_BOXDLG_H__836AEAE8_25BB_11D2_BA07_444553540000__INCLUDED
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CBoxDlg dialog

class CBoxDlg : public CDialog
{
// Construction
public:
    CBoxDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CBoxDlg)
    enum { IDD = IDD_BOX_DIALOG };
    CListBox    m_list;
    CComboBox   m_cmb;
    CBitmapButton btn1;
    CBitmapButton btn2;
    CBitmapButton btn3;
    CBitmapButton btn4;
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CBoxDlg)
    protected:
```

```
        virtual void DoDataExchange(CDataExchange * pDX);      // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;

        // Generated message map functions
        //{{AFX_MSG(CBoxDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnInsert();
        afx_msg void OnCut();
        afx_msg void OnBtn1();
        afx_msg void OnBtn2();
        afx_msg void OnBtn3();
        afx_msg void OnBtn4();
        afx_msg void OnButton1();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_BOXDLG_H__836AEAE8_25BB_11D2_BA07_444553540000__INCLUDED_)

// boxDlg.cpp : implementation file
//

#include "stdafx.h"
#include "box.h"
#include "boxDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CBoxDlg dialog

CBoxDlg::CBoxDlg(CWnd* pParent /* =NULL */)
	: CDialog(CBoxDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CBoxDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CBoxDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CBoxDlg)
	DDX_Control(pDX, IDC_LIST, m_list);
	DDX_Control(pDX, IDC_COMBO, m_cmb);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CBoxDlg, CDialog)
	//{{AFX_MSG_MAP(CBoxDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_INSERT, OnInsert)
	ON_BN_CLICKED(IDC_CUT, OnCut)
	ON_BN_CLICKED(IDC_BTN1, OnBtn1)
	ON_BN_CLICKED(IDC_BTN2, OnBtn2)
	ON_BN_CLICKED(IDC_BTN3, OnBtn3)
	ON_BN_CLICKED(IDC_BTN4, OnBtn4)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBoxDlg message handlers

BOOL CBoxDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
```

```
    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon

    // TODO: Add extra initialization here
    btn1.AutoLoad(IDC_BTN1,this);
    btn2.AutoLoad(IDC_BTN2,this);
    btn3.AutoLoad(IDC_BTN3,this);
    btn4.AutoLoad(IDC_BTN4,this);

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CBoxDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}
```

```
// If you add a minimize button to your dialog, you will need the code below
//   to draw the icon.  For MFC applications using the document/view model,
//   this is automatically done for you by the framework.

void CBoxDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//   the minimized window.
HCURSOR CBoxDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}


void CBoxDlg::OnInsert()
{
    CString str;
    m_cmb.GetWindowText(str);
    if(str.IsEmpty())
    {
        MessageBeep(MB_ICONEXCLAMATION);
```

```
            return;
        }
        if(m_cmb.FindStringExact(-1,str)<0)
        {
            m_cmb.InsertString(0,str);
            m_cmb.SetCurSel(0);
        }
}

void CBoxDlg::OnCut()
{
        int count=m_list.GetSelCount();
        if(count>0)
        {
            int * rgIndex=new int[count];
            m_list.GetSelItems(count,rgIndex);
              int j=count-1;
            for(int i=0;i<count;i++)
                    m_list.DeleteString(rgIndex[j--]);
             delete rgIndex;
        }
        int index=m_cmb.GetCurSel();
        if(index!=CB_ERR)
        {
            m_cmb.DeleteString(index);
            m_cmb.SetCurSel(0);
        }
}

void CBoxDlg::OnBtn1()
{
        CString str;
        int count=m_cmb.GetCount();
        for(int i=0;i<count;i++)
        {
            m_cmb.GetLBText(i,str);
            if(m_list.FindStringExact(-1,str)<0)
                    m_list.InsertString(i,str);
        }
        m_list.SetCurSel(0);
        m_cmb.ResetContent();
}

void CBoxDlg::OnBtn2()
```

```
}

    CString str;
    int index=m_cmb.GetCurSel();
    if(index==CB_ERR)
        return;
    m_cmb.GetLBText(index,str);
    if(m_list.FindStringExact(-1,str)<0)
    {
        m_list.AddString(str);
        m_cmb.DeleteString(index);
    }
    m_cmb.SetCurSel(0);

}

void CBoxDlg::OnBtn3()
{
    CString str;

    int count=m_list.GetCount();
    for(int i=0;i<count;i++)
    {
        m_list.GetText(i,str);
        if(m_cmb.FindStringExact(-1,str)<0)
            m_cmb.InsertString(i,str);
    }
    m_list.ResetContent();
    m_cmb.SetCurSel(0);



}

void CBoxDlg::OnBtn4()
{
    int count=m_list.GetSelCount();
    int * rgIndex=new int[count];
    m_list.GetSelItems(count,rgIndex);
    for(int i=0;i<count;i++)
    {
        CString str;
        m_list.GetText(rgIndex[i],str);
        if(m_cmb.FindStringExact(-1,str)<0)
```

```
            {
                m_cmb.InsertString(0,str);
            }

        for(i=0;i<count;i++)
            m_list.DeleteString(rgIndex[i]);
        m_list.SetCurSel(0);
        delete rgIndex;


}

void CBoxDlg::OnButton1()
{
        // TODO: Add your control notification handler code here

}
```

2.8 Satcctl 例程序

该例子演示了静态控件的使用,时间控制以及如何设置对话框控件的颜色,如图 2.14 所示。对话框右边排列整齐的边框是具有各种风格的静态控件,各控件的左边标明了其风格。它们的底色为黑色,显示的字体为红色,各个控件显示了各种格式的时间值。

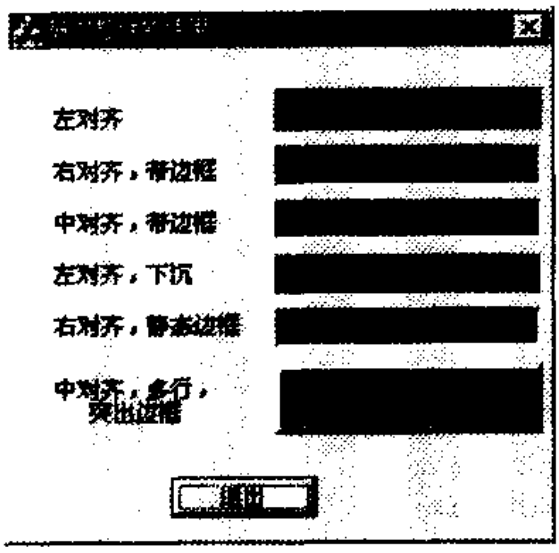


图 2.14 Satcctl 例程界面图

下面是创建该程序的步骤。

首先用 MFC AppWizard 创建基于对话框的应用程序,步骤如下:

1)启动 Microsoft Visual Studio。

2)选择菜单 File 的 New 菜单项,弹出 New 对话框。

3)在 New 列表中选择 Projects 选项卡,选择 MFC AppWizard(exe)项,在右边的 Project name 中填入 satectl。

4)点击 OK,New 对话框关闭弹出 MFC AppWizard 向导的 MFC AppWizard stepl 属性页。

5)在应用程序类型中选择 Dialog based。

6)其他选项接受缺省设置点击 Finish 按钮,随后出现下面的 New Project Information 对话框。

接着编辑对话框资源,步骤如下:

1)点击 Resource View,切换到资源视窗。

2)点击 satectl resources 节点,打开各资源,点击 Dialog 节点,打开对话框资源。

3)点击 IDD _ DIALOG,打开对话框编辑器。

4)点击鼠标右键,打开对话框的属性对话框,在 Caption 一栏输入"静态控件的使用"。

5)删除对话框缺省的编辑控件"要做,在这里设置对话控件"。

6)添加静态控件。

对话框上右边从上到下有六个静态控件,按图中所示,在各自的属性对话框的 Caption 一栏填入相对应的中文字符串。

右边第一个静态控件,ID 为 IDC _ TIME1,Styles 属性页中的 Aligntext 选择 Left 项。

第二个控件 ID 为 IDC _ TIME2,Styles 属性页中 Align text 为 Right,选择 Center Vertically和 Border。

第三个控件 ID 为 IDC _ TIME3,Styles 属性页中 Align text 选 Center,并选择 Border。

第四个控件 ID 为 IDC _ TIME4,Styles 属性页中的 Align text 为 Left 并选择 Sunken。

第五个控件 ID 为 IDC _ TIME5,Styles 属性页中选择 Client edge。

第六个控件 ID 为 IDC _ TIME6,Styles 属性页中 Align text 为 Center。

在 Sunken Extended styles 属性页中选择 Modal frame。

之后添加控件变量,步骤如下:

1)按 CTRL 键的同时双击静态控件 IDC _ TIME1,弹出 Add Member Variable 对话框。

2)在 Member Variable name 中输入 m _ staticl,category 项为 Control,Variable Type 为 CStatic。

3)点击 OK 按钮,关闭 Add Member Variable 对话框。

4)重复前三步分别为剩下的五个静态控件添加变量 m _ Static2、m _ Static3、m _ Static4、m _ Static5 和 m _ Static6。

用 ClassWizard 添加 OnCtlcolor 消息处理函数,步骤如下:

1)选择 View 菜单的 ClassWizard,打开 ClassWizard。

2)点击 Message Maps 标签,切换到 Message Maps 属性页。

3)在 Classname 列表框中选 CstaticlDlg。

4)在 Object IDS --栏中选择 CStaticlDlg。在 Message 一栏中选择 WM _ CTLCOLOR,点击 Add Function 按钮。

5)在下面的 Member functions 中,添加了消息处理函数 onctlcolor。

6)点击 Edit Code,关闭 Class Wizard,并将光标指向 onctlcolor 函数的实现位置。

7)在 CSatectlDlg::OnCtlcolor 函数中添加下列代码。

```
UINT nID;
nID = pWnd->GetDlgCtrlID();
if((nID <= IDC_TIME6)&&(nID >= IDC_TIME1))
{
        CBrush brush(RGB(0,0,0));
        pDC->SetBkColor(RGB(0,0,0));
        pDC->SetTextColor(RGB(255,0,0));
        return (HBRUSH)brush.m_hObject;
}
// TODO: Return a different brush if the default is not desired
return CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
```

在 CSatcctlDlg::OnInitDialog 中设置时间起动。

在 OnInitDialog 的提示语句"TODO:Add extra initia lization here"之中加入下一语句:SetTimer(1,1000,N ULL);

用 ClassWizard 处理 WM_TIMER 消息。

可参照前面用 ClassWizard 处理 WM_CTLCOLOR 消息一样,加入函数 OnTimer 来处理消息 WM_TIMER。

在 CSatcctlDlg::OnTimer 函数中加入下列代码:

```
CTime time = CTime::GetCurrentTime();
CString str1 = time.Format("%x");
m_static1.SetWindowText(str1);
CString str2 = time.Format("%S:%M:%H");
m_static2.SetWindowText(str2);
CString str3 = time.Format("%S:%M:%I%p");
m_static3.SetWindowText(str3);
CString str4 = time.Format("%A,%B,%Y");
m_static4.SetWindowText(str4);
CString str5 = time.Format("%a,%b,%y");
m_static5.SetWindowText(str5);
CString str6 = time.Format("%X");
str6 += "\n" + str6;
m_static6.SetWindowText(str6);
Invalidate();
CDialog::OnTimer(nIDEvent);
```

在这个例子中我们使用了计时器。在实际进行程序设计时,可以利用 Windows 的计时器,在与微处理器速度无关的普通时间间隔内,定时向程序发送消息。计时器的使用非常简单,我们只需用一个时间间隔参数调用 Cwnd 的 setTimer 函数,然后再利用 ClassWizard 为 WM_TIMER 消息提供一个消息控制函数即可。其函数原形为:

```
UINT SetTimer(INT nIDEvent,INT nElapse,
void(CALLBACK EXPORT * lpfnTimer)(HWND,UINT,UINT,DWORD));
```

使用时用 nIDEvent 指定一个非 0 的定时器标识符,nElapse 以毫秒数指出限时值,lpfn-

Timer 指出处理此 WM _ TIMER 消息的应用程序提供的 TimerProc 回调函数的地址，若此参数为 NULL，则此 WM _ TIMER 消息将被放到应用程序的消息队列中并由此 CWnd 对象来处理。

在这个程序中我们重载了 CWnd::OnCtlcolor 函数来控制六个静态控件的颜色，其原形如下：

HBRUSH OnCtlcolor(CDC, * PDC, CWnd, * pwnd, UINT, nCtlcocor);

参数 PDC 存放指向该子窗口的设备描述表子针。Pwnd 存放指向申请颜色的控制的指针。nctlcolor 存放指向控件类型的值，该值可以是下列值之一：

CTLCOLOR _ BTN 按钮控件

CTLCOLOR _ DLG 对话框

CTLCOLOR _ EDIT 编辑控件

OTLLOLOR _ LISTBOX 列表框控件

CTLLOLOR _ MSGBOX 消息框

CTLLOLOR _ SCROLLBAR 滚动条控件

CTLLOLOR _ STATIC 静态控件

每个控件在显示之前会立刻向其父窗口发送 WM _ CTLCOLOR 消息。如果用户在派生对话框中对 WM _ CTLCOLOR 消息进行了映射，就可以设定文本的前景色和背景色，同时还可以为控件或非文本区域选择一个刷子。如果想要设定个别控件的颜色，只需将 PWnd 具体置换成相应子窗口的 ID，并对它进行识别。

```
// satcctlDlg.h : header file
//

#if ! defined(AFX _ SATCCTLDLG _ H __ 872B3E28 _ 2582 _ 11D2 _ BA07 _ 444553540000 __ INCLUD-
ED _)
#define AFX _ SATCCTLDLG _ H __ 872B3E28 _ 2582 _ 11D2 _ BA07 _ 444553540000 __ INCLUDED _
#if _ MSC _ VER > = 1000
#pragma once
#endif // _ MSC _ VER > = 1000

/////////////////////////////////////////////////////////////////////////////
// CSatcctlDlg dialog

class CSatcctlDlg : public CDialog

// Construction
public:
    CSatcctlDlg(CWnd * pParent = NULL);  // standard constructor
// Dialog Data
    //{{AFX _ DATA(CSatcctlDlg)
    enum { IDD = IDD _ SATCCTL _ DIALOG };
    CStatic    m _ static5;
```

```
        CStatic     m_static6;
        CStatic     m_static1;
        CStatic     m_static4;
        CStatic     m_static3;
        CStatic     m_static2;
        //{{AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSatcctlDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);        // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;

        // Generated message map functions
        //{{AFX_MSG(CSatcctlDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg HBRUSH OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_SATCCTLDLG_H__872B3E28_2582_11D2_BA07_444553540000__
INCLUDED_)


// satcctlDlg.cpp : implementation file
//

#include "stdafx.h"
#include "satcctl.h"
#include "satcctlDlg.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSatcctlDlg dialog

CSatcctlDlg::CSatcctlDlg(CWnd* pParent /* =NULL */)
	: CDialog(CSatcctlDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSatcctlDlg)
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CSatcctlDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSatcctlDlg)
	DDX_Control(pDX, IDC_TIME5, m_static5);
	DDX_Control(pDX, IDC_TIME6, m_static6);
	DDX_Control(pDX, IDC_TIME1, m_static1);
	DDX_Control(pDX, IDC_TIME4, m_static4);
	DDX_Control(pDX, IDC_TIME3, m_static3);
	DDX_Control(pDX, IDC_TIME2, m_static2);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSatcctlDlg, CDialog)
	//{{AFX_MSG_MAP(CSatcctlDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_WM_TIMER()
	ON_WM_CTLCOLOR()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSatcctlDlg message handlers
```

```
BOOL CSatcctlDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}


	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
    SetTimer(1,1000,NULL);

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CSatcctlDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
```

```
    }

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSatcctlDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();

    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSatcctlDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CSatcctlDlg::OnTimer(UINT nIDEvent)
{

    CTime time = CTime::GetCurrentTime();
```

```
        CString str1 = time.Format("%x");
        m_static1.SetWindowText(str1);
        CString str2 = time.Format("%S:%M:%H");
        m_static2.SetWindowText(str2);
        CString str3 = time.Format("%S:%M:%I%p");
        m_static3.SetWindowText(str3);
        CString str4 = time.Format("%A,%B,%Y");
        m_static4.SetWindowText(str4);
        CString str5 = time.Format("%a,%b,%y");
        m_static5.SetWindowText(str5);
    CString str6 = time.Format("%X");
        str6 += "\n" + str6;
        m_static6.SetWindowText(str6);
        Invalidate();
        CDialog::OnTimer(nIDEvent);
}

HBRUSH CSatcctIDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{

        UINT nID;
        nID = pWnd->GetDlgCtrlID();
        if((nID <= IDC_TIME6) && (nID >= IDC_TIME1))
        {
                CBrush brush(RGB(0,0,0));
                pDC->SetBkColor(RGB(0,0,0));
                pDC->SetTextColor(RGB(255,0,0));
                return (HBRUSH)brush.m_hObject;
        }
        // TODO: Return a different brush if the default is not desired
        return CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
}
```

2.9 小　　结

本章对 Visual C++ 6.0 的集成环境作了大致介绍，通过本章读者可对这一 Visual C++ 的最新工作平台的以下方面有所了解。

- 如何用 AppWizard 创建单文档、多文档或基于对话框的应用程序。
- 在 AppWizard 中如何增加对数据库，OLE 的支持。
- 在 AppWizard 中如何控制工具条、状态条、上下文的敏感帮助和三维界面。
- 如果用 ClasslWizard 创建新类。
- 如何用 ClassWizard 处理消息处理函数。

- 如何定义类的成员变量。
- 如何用集成调试器调试典型的应用程序。
- 如何使用“Edit”菜单操作程序。

除此之外，希望读者对 Visual Studio 的主窗口有个初步的认识，并对可视化编程的具体内涵有一定了解。

第3章 标准控件

Windows是一个通过增加动态链接库来实现扩充的操作系统，随着Windows 98和WindowsNT 4.0一起发布的公用控件使得Windows应用程序的界面更加时髦，更加可人。如果在自己的应用程序中多增加些公用控件，会使你的作品更具有Windows的风格。通过Visual C++6.0可以创建或使用的控件有Windows标准控件、Windows公用控件、MFC附加的控件类(如位图按钮)和ActiveX控件等。其中标准控件和公用控件见表所示。这些控件的类都是从CWnd类派生出来的，都继承了CWnd类的大部分功能，故在创建和使用控件时可根据需要来使用CWnd类的消息方法，属性和风格，实际上这些控件本身就是一个特殊的窗口。

本章就静态文本、列表框、组合框、滚动条和滑块控件进行介绍，并用一节的篇幅来说明Windows公共对话框的使用。

3.1 静态文本控件

静态文本控件有时也称文本框，可用于显示文本串、框、矩形、图标、光标、位图或元文件。其通常不接收输入也不提供输出，但是当以SS_NOTIFY风格创建时将传送鼠标单击消息给父窗口。下面先介绍静态文本控件的创建方法，接着通过一个例子来详细说明该控件的使用方法。

3.1.1 创建及初始化静态文本控件

创建MFC支持的控件对象一般有两种方式：一种是从对话模板中创建，可用Class Wizard来进行管理；另一种控件对应类的Create方法来创建。

1. 使用对话框编辑器添加控件

用对话框资源编辑器创建对话模板资源时，可以从控件工具栏击点所需的控件，在这里是指静态文本控件(Static Text control)，然后在对话框的合适位置按住鼠标左键不放，并向右下方拉出合适的大小，当松开鼠标时，一个完整的控件就出现在对话框上面。当在控件上单击鼠标右键时，会弹出一快捷菜单，单击Properity项，弹出该控件的属性对话框(如图3.1所示)，可对控件的标识符(ID)、风格(Style)和控件的扩展风格(Extended Style)进行设置。先对控件上显示的字符等进行初始化的设置，之后可用ClassWizard生成该控件的对象变量，并响应控件在程序中必须响应的消息。ClassWizard除了在主菜单中可以找到外，在刚才按鼠标右键弹出的快捷菜单上也有。

2. 用Create方法创建控件

为创建一个静态文本控件需以下两个步骤：

1)通过使用C++的关键字New，调用构造函数CStatic::CStatic()为一个CStatic对象分

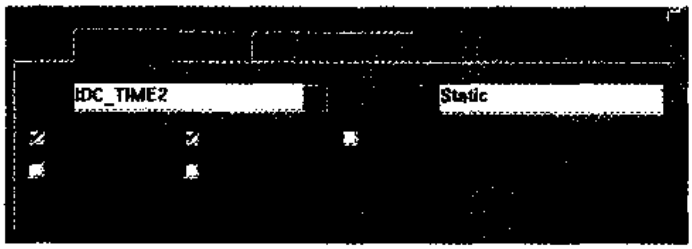


图 3.1

配一个实例。

2)初始化 CStatic 对象并为它附加一个 Windows 静态文本控件,这是通过 CStatic::Create()方法设置控件参数和风格来实现的,例如以下代码分配一个 CStatic 对象,并返回指针:CStatic * pMyStatic = new CStatic()。

指针 pMystatic 通过调用 CStatic::Create()方法来初始化,该方法声明如下:

```
BOOL Create(LPCSTR lp32Text,//指向控件上显示的文本字符串指针
DWORD dwStyle,//控件风格
Const RECT82 rect,//控件出现的矩形区域
CWnd * pParendWnd,//指向父窗口的指针
UINT nID;Oxffff);//控件的标识符
```

表 3.1 列出相关控件及它们相对应的类名:

表 3.1 控件及其相对应的类名

控件类	Windows 标准控件
CStatic	静态文本控件
CTabCtrl	标签控件
CStatusBarCtrl	状态条控件
CSpinButtonCtrl	滚动控件
CSliderCtrl	滑标控件
CRichEditCtrl	多信息编辑控件
CProgressCtrl	进度指示器
CImageCtrl	图象列表
CHotKeyCtrl	热键控件
CTreeCtrl	树视控件
CAnimateCtrl	动画控件
CToolTipCtrl	工具提示
CIleaderctrl	头控件
CToolBarCtrl	工具控件
CScrollBar	滚动条控件
CComboBox	组合框控件
CListBox	列表框控件
CButton	按钮控件

3.2 列表框和组合框

列表框是一种输入控制，它可以使用户从所列的表项中进行选择，通常列表框有边框，并且还包含一个垂直滚动条，当用户用鼠标点中了某项时，所选中的项就会被加亮显示。组合框将一个编辑框和一个列表框组合在一起，用户既可以从各个列表项中进行选择，也可以直接自己输入。组合框有三种类型：简单组合框，下拉组合框和下拉列表组合框。简单组合框包括一个编辑框以及一个总是可见的列表框，而下拉式组合框与简单组合框的不同之处在于只有当用户用鼠标点了向下的箭头时，列表框才会出现，下拉列表组合框同样也提供了一个下拉式列表框，只有当用户用鼠标点了向下的箭头时它才会出现，但下拉式列表组合框不包括编辑框。MFC 类库用类 CListBox 来实现列表框控件，用类 CComboBox 来实现组合框控件。

3.2.1 创建和初始列表框和组合框

同上节介绍静态文本控件一样，用两种方法来创建列表框和组合框。

1. 使用对话编辑器添加控件

可按上节介绍的静态文本控件在对话框模板上创建列表框和组合框。需要指出的是在创建组合框的同时可插入组合框的列表项。在组合框的属性对话框的 Data 选项卡的“EnterListBox items”编辑框内输入列表项，若要输出另一项，按 Ctrl + Enter。

2. 用 Create 方法创建控件

首先用构造函数分配实例，如：

```
CListBox * PMylistBox = new CListbox();
CComboBox * PMyComboBox = new CComboBox();
```

再调用 Create 方法初始化实例变量，CListBox 和 CComboBox 的 Create 方法声明如下：

```
BOOL Create(DWORD dwStyle,//控件的风格
Const RECT & rect,//控件位置及尺寸
CWnd * pParentWnd,//控件的父窗口指针
UINT ID);//控件的标识 ID
```

3.3 滚动条和滑杆

滚动条作为一种控件，可以在一整数范围内进行滚动，Windows 所提供的滚动条可以是单独的控件，也可作为窗口列表框和组合框的一个组成部分。单独的滚动条控件在外形及行为上同窗口中的滚动条非常相似，它包含了一个滚动框，该滚动框被用来表示当前的值。鼠标按键既可以使滚动框移动一行，也可使滚动框移动一页。滑杆也称滑块控件，与滚动条相似，是一种交互式的高度可视的控件，它包括一个滑动框和选择的刻度标志，可用鼠标或键盘来移动滑动框。MFC 类库用类 CScrollBar 来实现滚动条，用类 CSliderCtrl 类来实现滑块控件。

3.3.1 创建和初始化滚动条和滑块

同静态文本框一样，可用两种方法创建滚动条和滑块。

1. 使用对话编辑添加控件

（参照静态文本框控件一节）

2. 用 Create 方法创建控件

首先用构造函数分配实例，如：

```
CScrollBar * pMyScrollBar = new CScrollBar();
CSliderCtrl * pMySKiderCtrl = new CSliderCtrl();
```

再调用 Create 方法初始化实例对象，CScrollBar 的 Create 方法声明如下：

```
Bool Create(DWORD dwstyle,//控件风格
Const RECT &rect,//控件位置事件
CWnd * pParentwnd,//指向控件父窗口的指针
CINT ID);//控件的标识 ID
```

CSliderCtrl 的 Create 方法的使用同 CScrollBar 的 Create 方法一样。

3.4 公共对话框

Windows 提供了一组标准的用户接口对话框，用六个类来实现它们的功能，每个类都是从 CCommonDialog 派生的，如下列表所示。

表 3.2 用户接口对话框类

对话框类型	实现类	说明
Color	CColorDialog	选定一种颜色并创建这种颜色
Find/Replace	CFindReplaceDialog	查找一字符串或替换找到的字符串
Font	CFontDialog	选定并创建一种字体
Open/SaveAs	CFileDialog	打开或保存一个文件
PageSetup	CPageSetupDialog	创建页选项，实现首页面设置
Print	CPrint	设置打印参数

本节就这六个不同对话框类型进行讨论。

3.4.1 颜色对话框

类 CColorDialog 允许用户在自己的应用程序中插入颜色对话框，用户可自己选择一种颜色或混合出一种颜色（如图 3.2 所示），通过使用默认的构造函数可以创建颜色对话框，声明如下：

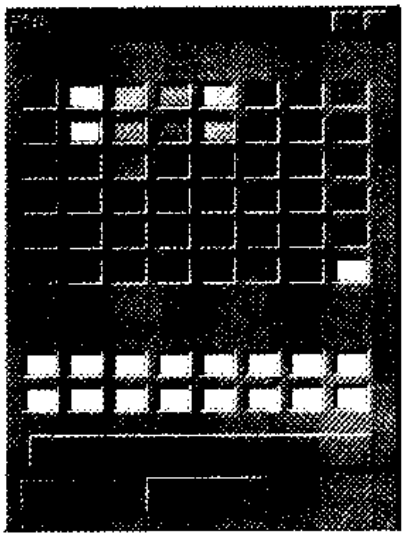


图 3.2　颜色对话框

CColorDialog (COLORREF ClrInit = 0, DWORD dwFlags = 0,
CWnd * pParentWnd = NULL);

上句中参数 ClrInit 指明当时对话框出现时，最初选择好颜色(缺省为黑色)；dwFlags 是一个标志集，自定义对话框的形式及作用；pParentWnd 是一个指向用户自己的窗口的对话框指针。

CColorDialog 类的使用过程如下：

1)构造 CColorDialog 类对象。

2)设置修改对象的数据成员 m _ cc 以便初始化对话框中控件的值，m _ cc 结构类型为 CHOOSECOLOR。

3)调试成员函数 DoModal 显示对话框并让用户从中选择颜色，如果选择"OK"按钮，DoModal 返回 IDOK；如果选择 Cancel 按钮 DoModal 返回 IDCANCEL，取消用户输入。

4)如果 DoModal 返回的是 IDOK，系统将用户输入信息保存在数据成员 m _ cc 中，使用 CColorDialog 的成员函数可获取用户输入的信息。下列的代码演示颜色对话框的使用

```
CColorDialog dlgcolor( m_cc, FULLOPENICC_RGBINIT, this);
if (dlgColor.DoModal())
{
//得到选择的颜色
COLORREF cr = dlgcolor.GetColor();
//接下来使用该颜色
}
```

注意：颜色对话框可通过设置在结构 CHOOSECOLOR 中的标识 CC _ FULLOPEN 来完整地打开和显示，若对话框不完全打开，它只显示左半部，其它的标识设置可参考相关的资料。

3.4.2　查找替换对话框

CFindReplaceDialog 类构造了 Find(如图 3.3 所示)和 Raplace(如图 3.4 所示)这两个无模

式对话框。用户可指明要寻找和替换的字符串，对话框接受并存贮这一输入。该类与其他公用对话框类的不同之处有以下两方面：

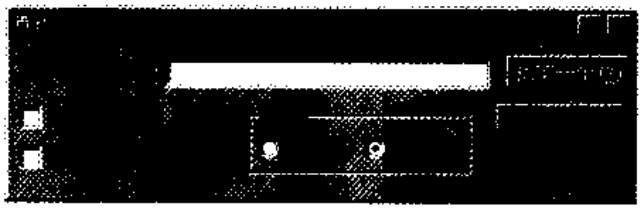


图 3.3　查找对话框



图 3.4　替换对话框

1)没有专门的应用程序代码来完成实际的寻找替换工作，而是必须通过注册一个回调消息处理函数，把用户的动作通知给父级窗口。

2)该对话框是无模式的，不同于其他对话框创建的方法。通过使用一个默认的不带参数的构造函数，创建对话框：

```
CFindReplaceDialog();
```

接下来用类方法 CFindReplaceDialog::Create()初始化并显示。

```
BOOL Create(BOOL bFindDialogOnly,
LPCTSTR lpszFindWhat,
LPCTSTR lpszReplaceWith = NULL,
DWORD dwFlags: FR_DOWN,
CWnd * pParentWnd = NULL);
```

当参数 bFindDialogOnly 为真时，显示查找对话框，为假时显示查找/替换对话框；lpszFindWhat 为要查寻的字符串；lpszReplaceWith 为替换被查寻的字符串的字符串；dwFlags 指示用来自定义对话框设置；pParentWnd 是指向对话框父窗口的指针。

3.4.3　字体对话框

CFontDialog 类允许用户在应用程序中嵌入字体对话框，如图 3.5 所示，在该对话框中，可以选择字体并设置属性。通过调用下面默认的构造函数来创建字体对话框。

```
CFontDialog(LPLOGFONT lplfInitial = NULL,
DWORD dwFlags = CF    EFFEOTSLCF-SCREENFONTS, CDC * pdcPrinter = NULL, CWnd * pParen-
```

tWnd = NULL);

参数 lplfIInitial 是指向结构 LOGFONT 的指针，初始化字体的特性。

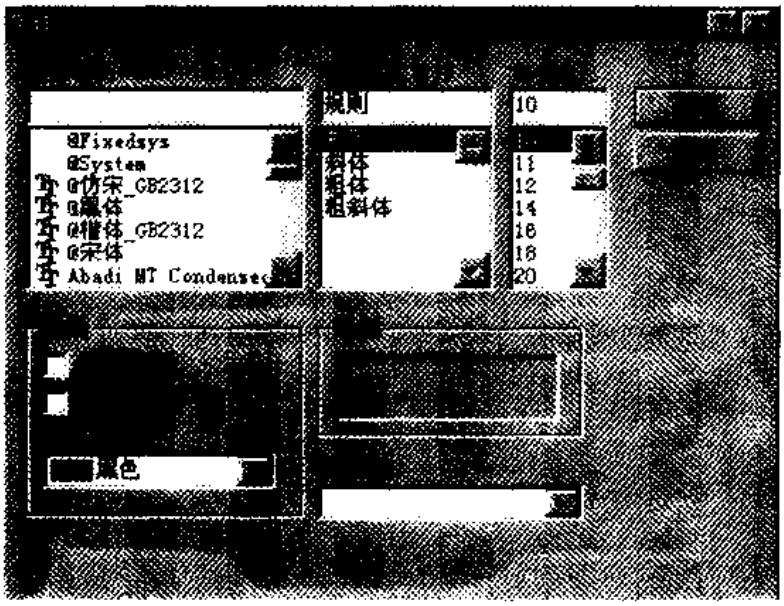


图 3.5 字体对话框

dwFlags 是字体选择标志集，可用位或操作符(|)来组合。pdcPrinter 是一个指向打印机设备描述表的参数，pParentWnd 为指向父窗口的指针。CFontDialog 的使用过程与 CColorDialog 类的使用过程极其相似，只不过把 CColorDialog 的数据成员 M _ CC 换成 CFontDialog 的 m _ cf，对应的结构 CHOOSECOLOR 改为 CHOOSEFONT 即可。

注意：如果直接利用这一章介绍的六个对话框类很容易达到目的，但在此基础上可以自己定义一个具有特色的公用对话框，例如在原来的公用对话框里面加上另外一些控件等，具体的方法是从相应的对话框类派生出一个新的子类，并且设计一个用户对话模板，给新类中加入消息映射项，从新的控件中处理通知消息，并且给基类传送未被处理的消息。

3.4.4 打开、保存文件对话框

CFileDialog 类允许用户在应用程序中插入通用文件对话框，使用文件对话框可以实现与 Windows 标准一致的 Open 和 Save as 对话框，如图 3.6 和 3.7 所示，新的 Windows 公用文件对话框也允许用户创建一个新的文件夹或在对话框内删除已存在的文件夹和文件

一个文件对话框通过使用默认的构造函数来创建：

```
CFileDialog(BOOL bOpenFileDialog, LPUTSTR, lp32RefExt = NULL,
LPCTSTR lp32FileName = NULL,
DWORD duFlagS: OFN-HIDERADONLYIOFN-ONERWRITEPROMPT, LPCTSTR lp32Filter = NULL,
CWnd * pParentWnd = NULL);
```

参数 OpenFileDialog 若为真则创建 FileOpen 对话框，反之则创建 FileSaveAs 对话框，lp32RefExt 为默认文件名的扩展名。lp32FileName 是最初出现在文件名编辑框中的文件名。

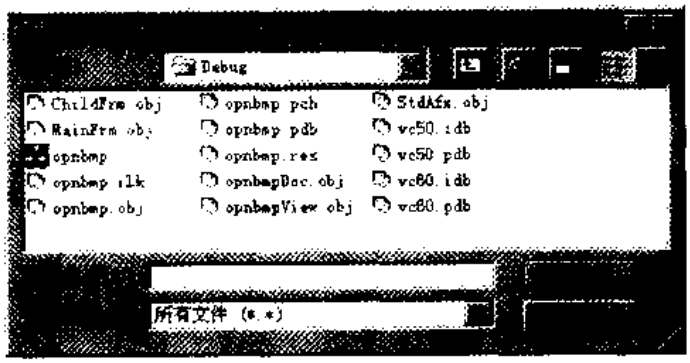


图 3.6 “打开”对话框

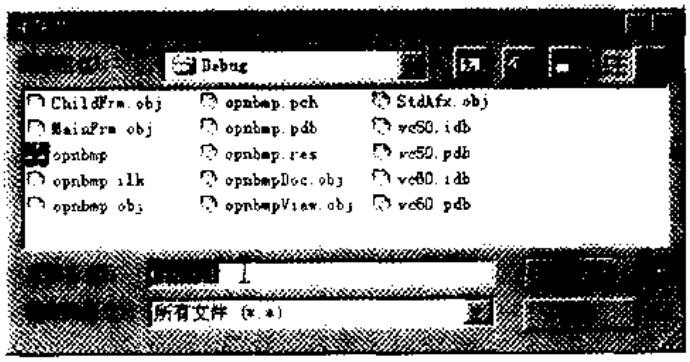


图 3.7 “保存为”对话框

dwFlags 是位标识的组合，lp32Filter 是用来指明文件扩展名过滤器的字符串组，指明文件列表框中应出现的文件类型。PParentWnd 是指向父窗口的指针。

出现在 FileOpen 和 FileSaveAs 对话框的文件类型组合框中的文件过滤字符串是成对定义的，字符串对中的第一个字符串出现在过滤组合框中，第二个字符串定义真正的过滤器，多个扩展符可用“;”作为分隔符指定。字符串以两个“|”字符结束，后跟一个 NULL 字符。例如 Microsoft Excel 允许用户按扩展符.XLC 或.XLS 打开文件，用于 Excel 的过滤器可写成：

```
CString strFilter = "Char Files(*.XLC)|*.XLC|
Worksheet Files(*.XLS)|*.XLS|DataFiles(*.XLC;*.XLS)|*.XLC;*.XLS|All Files(*.*)|
*.*|";
```

CFileDialog 类的使用过程与 CColorDialog 类相同，在这里把 CColorDialog 的数据成员 m_cc 改成 CFileDialog 的 m_ofn，相应的结构 CHOOSECOLOR 改为 OPENFILENAME。

3.4.5 页面设置对话框

类 CPageSetupDialog 提供 PageSetup 公用对话框，如图 3.8 所示。在该对话框中，用户可选择页大小、页源、页方向、边缘设置，用户也可从此对话框中直接调用 Print Setup 对话框。类 CPageSetupDialog 为在应用程序中加入完整的 PageSetup 对话框提供了简便的方法。

CPageSetupDialog 的缺省构造函数为：

```
CPageSetupDialog(DWORD dwFlags = PSD_MARGINS|PSD_INWININTLMEASURE, CWnd *
```

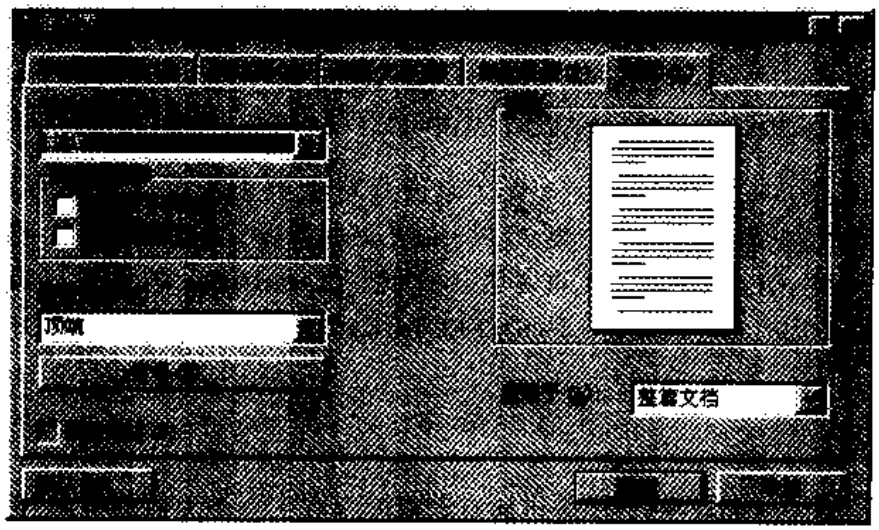


图 3.8　页面设置对话框

pParentWnd = NULL);

参数 dwFlags 指明一个或更多个可用来定制对话框设置的标识。pParentWnd 是指向父窗口的指针。有两个结构值得注意：一个是 PAGESETUPDLG，类 CPageSetupDialog 的数据成员 m_psd 利用该结构设置自定义对话框控件。另一个是 DEVMODE，它提供关于设置初始化的信息以及打印机环境的信息。

类 CPageSetupDialog 的使用方法可参照 CColorDialog 类的说明，同时要把 CcolorDialog 的数据成员 m_cc 换成 CPageSetupDialog 的 m_psd，相应的结构 CHOOSECOLOR 换为 PAGESETUPDIALOG。

3.4.6　打印对话框

CPrintDialog 类支持的公用对话框是为打印文件或设置打印机参数服务的，如图 3.9 所示。用户可选择打印机属性和选项，如需打印页数、打印文件份数等。

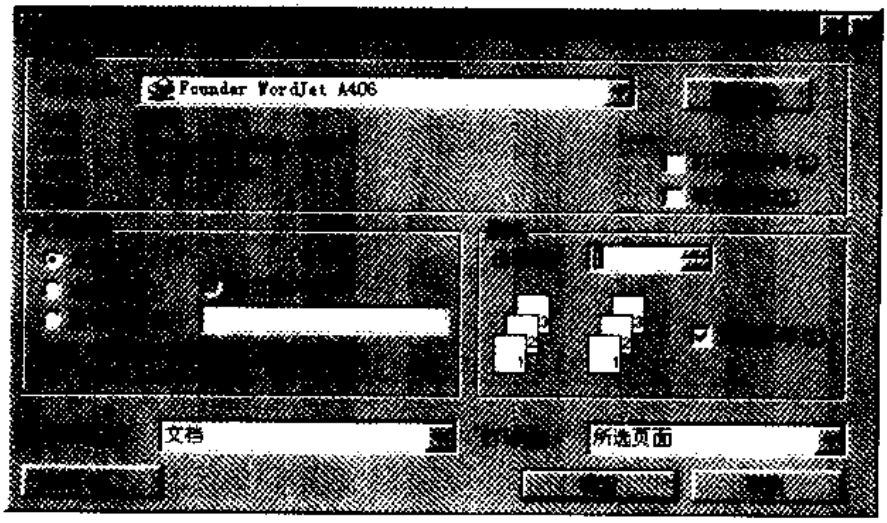


图 3.9　打印设置对话框

CPrintDialog 的缺省构造函数为：

CPrintDialog(BOOL bPrintSetupOnly, DWORD dwFlags = PD-ALLPAGES| PD-VSEDEVMODELOP-IES|PD-NOPAGENUMS|PD-HIDEPRINTTOFILE|PD-NOSELECTION, CWnd * pParentWnd = NULL);

参数 bPrintSetupOnly 指明显示一个打印设置对话框而不是一个打印对话框，参数 dwFlags 是定制对话框设置的位标志集，pParentWnd 是指向父窗口的指针。类 CPrintDialog 的使用方法可参照 CColorDialog 类的说明，同时把 CColorDialog 的数据成员 m _ cc 换成 CPrintDialog 的 m _ pd，相应的结构 CHOOSECOLOR 改为 PRINTDLG。

3.5 实例分析

在进行了一篇笼统的叙述后，读者可能对本章内容有了一个大概的了解，但可能较抽象，在本节我们将给读者演示两个实例，相信读者看完这两个例子后，一定会对本章内容有更深刻形象的了解。

3.5.1 实例 Property

该实例演示了下拉框、列表框、编辑框和按钮的使用方法，如图 3.10 所示。工程类型是基于对话框式的，对话框标题是“文件属性”，我们最终目的也是显示机器上任何价文件的文件属性。对话框上的控件分三列，下面以从左到右从上到下的次序依次介绍。第一个是“文件”静态文本控件，其下是一个编辑控件，用来显于文件的名字。其下是一列表框，作用是列出在当前目录下除目录外的所有文件。最下是“确定”按钮。第二列最上是“目录”静态文本控作，其下是一个 Simple 风格的下拉框(Combo Box)，用来显于当前驱动盘下的所有目录，再下是具有 Drop List 风格的下拉框，用来显于机口的磁盘符，最下为“取消”按钮。最后一列最上是“属性”静态文本控件，接着是显示文件属性的不可编辑的编辑框，最下是“显示属性”按钮。

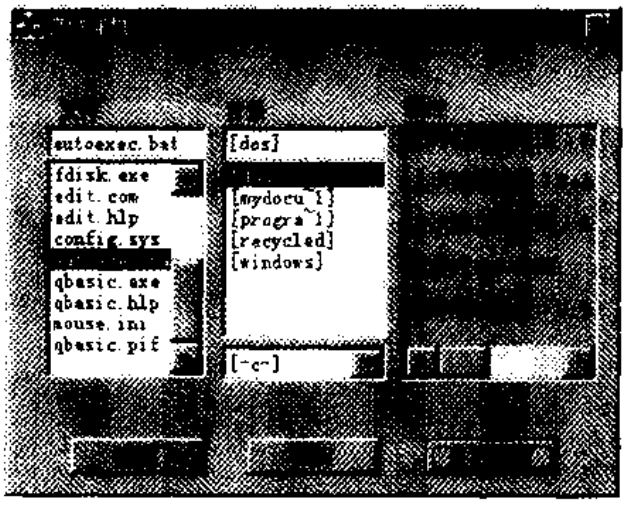


图 3.10 Property 例程界面

下面介绍创建该例程的步骤。

1)创建基于对话框的 Property 工程。

2)编辑对话框资源(IDD _ PROPERTY DIALOG)。按照上面介绍对话框上各控件的顺序并参照图示,在对话框上添加如下控件,它们的标识符 ID 分别为:IDC _ STATIC、IDC _ EDITFILE、IDC _ LIST、IDC _ OK、IDC _ STATIC、IDC _ COMBODIR、IDC _ COMBOPRI、IDC _ CANCEL、IDC _ STATIC、IDC _ EDITPRO 和 IDC _ BUTTON。

3)具体如何处理消息事件,可参照文件 property. h 和 propertyDlg.cpp,代码如下:

```
// propertyDlg.h : header file
//

#if ! defined(AFX_PROPERTYDLG_H__6A8813A8_18D1_11D2_BA07_444553540000__INCLUDED_)
#define AFX_PROPERTYDLG_H__6A8813A8_18D1_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CPropertyDlg dialog

class CPropertyDlg : public CDialog
{
// Construction
public:
	CPropertyDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
	//{{AFX_DATA(CPropertyDlg)
	enum { IDD = IDD_PROPERTY_DIALOG };
	CEdit m_editfile;
	CEdit m_editpro;
	CComboBox	m_combodri;
	CListBox	m_list;
	CComboBox	m_combodir;
	CString	m_strfile;
	CString	m_strpro;
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CPropertyDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL
```

```
// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CPropertyDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnButton();
	afx_msg void OnSelchangeList();
	afx_msg void OnSelchangeCombodri();
	afx_msg void OnDblclkCombodir();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_PROPERTYDLG_H__6A8813A8_18D1_11D2_BA07_444553540000__
INCLUDED_)

// propertyDlg.cpp : implementation file
//

#include "stdafx.h"
#include "property.h"
#include "propertyDlg.h"
#include <direct.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
```

```
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPropertyDlg dialog

CPropertyDlg::CPropertyDlg(CWnd* pParent /*=NULL*/)
```

```
    : CDialog(CPropertyDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CPropertyDlg)
    m_strfile = _T("");
    m_strpro = _T("");
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CPropertyDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPropertyDlg)
    DDX_Control(pDX, IDC_EDITFILE, m_editfile);
    DDX_Control(pDX, IDC_EDITPRO, m_editpro);
    DDX_Control(pDX, IDC_COMBODRI, m_combodri);
    DDX_Control(pDX, IDC_LIST, m_list);
    DDX_Control(pDX, IDC_COMBODIR, m_combodir);
    DDX_Text(pDX, IDC_EDITFILE, m_strfile);
    DDX_Text(pDX, IDC_EDITPRO, m_strpro);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPropertyDlg, CDialog)
    //{{AFX_MSG_MAP(CPropertyDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTON, OnButton)
    ON_LBN_SELCHANGE(IDC_LIST, OnSelchangeList)
    ON_CBN_SELCHANGE(IDC_COMBODRI, OnSelchangeCombodri)
    ON_CBN_DBLCLK(IDC_COMBODIR, OnDblclkCombodir)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPropertyDlg message handlers

BOOL CPropertyDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.
```

```
    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon

  UINT attrdri=DDL_DRIVES;
    m_combodri.Dir(attrdri,"[_*]");
    m_combodri.SetCurSel(0);
    CString strdri;
    m_combodri.GetLBText(0,strdri);
    strdri.MakeUpper();
    _chdrive(strdri[2]-'A'+1);
    UINT attrdir=DDL_DIRECTORY;
    attrdir |=~(DDL_READWRITE |DDL_DRIVES);
    m_combodir.Dir(attrdir,"*.*");
    m_combodir.SetCurSel(0);
    CString strdir;
    m_combodir.GetLBText(0,strdir);
    strdir=strdir.Mid(1,strdir.GetLength()-2);
    _chdir(strdir);
    m_list.Dir(DDL_READWRITE,"*.*");
    m_list.SetCurSel(0);

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CPropertyDlg::OnSysCommand(UINT nID, LPARAM lParam)
```

```
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CPropertyDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CPropertyDlg::OnQueryDragIcon()
```

```
{
    return (HCURSOR) m_hIcon;
}

void CPropertyDlg::OnButton()
{

    CFile file;
    CFileStatus status;
    if(file.Open(m_strfile,CFile::modeRead))
    {
        file.GetStatus(status);
        CString strc,strm,stra,strlen,strpath,edittext;
        strc = status.m_ctime.Format("%A,%B,%d,%Y");
        strm= status.m_mtime.Format("%A,%B,%d,%Y");
        stra = status.m_atime.Format("%A,%B,%d,%Y");
        strlen.Format("%ld",status.m_size);
        edittext ="文件创建的时间:" + strc +"\r\n\r\n上次修改时间:" + strm +"\r\n
\r\n" +
                "上次读文件时间:" + stra +"\r\n\r\n文件大小:" + strlen +"K\r\n\r
\n" +
                "文件所在途经:" + status.m_szFullName;
        m_strpro = edittext;
        UpdateData(false);
    }
    else return;


}

void CPropertyDlg::OnSelchangeList()
{

    CString str;
    m_list.GetText(m_list.GetCurSel(),str);
    m_strfile = str;
    UpdateData(false);
}

void CPropertyDlg::OnSelchangeCombodri()
{
    CString strdri;
    m_combodri.GetLBText(m_combodri.GetCurSel(),strdri);
    strdri.MakeUpper();
```

```
        _chdrive(strdri[2]-'A'+1);
        UINT attrdir=DDL_DIRECTORY;
        attrdir |=~(DDL_READWRITE |DDL_DRIVES);
        m_combodir.ResetContent();
        m_combodir.Dir(attrdir,"*.*");
        m_combodir.SetCurSel(0);
        CString strdir;
        m_combodir.GetLBText(0,strdir);
        strdir=strdir.Mid(1,strdir.GetLength()-2);
        _chdir(strdir);
        m_list.ResetContent();
        m_list.Dir(DDL_READWRITE,"*.*");
        m_list.SetCurSel(0);



}

void CPropertyDlg::OnDblclkCombodir()
{

        CString str;
        m_combodir.GetLBText(m_combodir.GetCurSel(),str);
        str=str.Mid(1,str.GetLength()-2);
        _chdir("..");
        _chdir(str);
        m_list.ResetContent();
        m_list.Dir(DDL_READWRITE,"*.*");
        m_list.SetCurSel(0);


}


#if !defined(AFX_EDITEX_H__81B87D34_2872_11D2_BA07_444553540000__INCLUDED_)
#define AFX_EDITEX_H__81B87D34_2872_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// editex.h : header file
//

/////////////////////////////////////////////////////////////////////////////
```

```
// CEditEx window

class CEditEx : public CEdit
{
// Construction
public:
	CEditEx();

// Attributes
public:

// Operations
private:
	LPCTSTR LockBuffer();
	void UnlockBuffer();
	UINT GetBufferLength();
	BOOL InitializeReplace(LPCTSTR lpszFind, BOOL bNext, BOOL bCase);
	BOOL SameAsSelected(LPCTSTR lpszCompare, BOOL bCase);
	BOOL FindText(LPCTSTR lpszFind, BOOL bNext, BOOL bCase);
	void OnTextNotFound(LPCTSTR);
public:
// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CEditEx)
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CEditEx();
	void GetSelectedText(CString& strResult);
	void OnEditFind();
	void OnEditReplace();
	void OnFindNext(LPCTSTR lpszFind, BOOL bNext, BOOL bCase);
	void OnReplaceSel(LPCTSTR lpszFind, BOOL bNext, BOOL bCase, LPCTSTR lpszReplace);
	void OnReplaceAll(LPCTSTR lpszFind, LPCTSTR lpszReplace, BOOL bCase);
	// Generated message map functions
protected:
	//{{AFX_MSG(CEditEx)
		// NOTE - the ClassWizard will add and remove member functions here.
	//}}AFX_MSG

	DECLARE_MESSAGE_MAP()
};
```

```
//////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_EDITEX_H__81B87D34_2872_11D2_BA07_444553540000__INCLUDED_)

// editex.cpp : implementation file
//

#include "stdafx.h"
#include "usetest.h"
#include "editex.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#define _UNICODE

//////////////////////////////////////////////////////////////////////////
// CEditEx

CEditEx::CEditEx()
{
}

CEditEx::~CEditEx()
{
}


BEGIN_MESSAGE_MAP(CEditEx, CEdit)
	//{{AFX_MSG_MAP(CEditEx)
		// NOTE - the ClassWizard will add and remove mapping macros here.
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
// CEditEx message handlers
static UINT  EndOfLine(LPCTSTR lpszText, UINT nLen, UINT nIndex)
```

```
{
        LPCTSTR lpsz = lpszText + nIndex;
        LPCTSTR lpszStop = lpszText + nLen;
        while (lpsz < lpszStop && *lpsz != '\n')
                ++lpsz;
        return lpsz - lpszText;
}

LPCTSTR CEditEx::LockBuffer()
{
//      HLOCAL hLocal = GetHandle();
//      ASSERT(hLocal != NULL);
//      LPCTSTR lpszText = (LPCTSTR)LocalLock(hLocal);
//      ASSERT(lpszText != NULL);
//      return lpszText;
        CString str;
        GetWindowText(str);
        return (LPCTSTR)str;
}

void CEditEx::UnlockBuffer()
{
        HLOCAL hLocal = GetHandle();
        ASSERT(hLocal != NULL);
        LocalUnlock(hLocal);
}

// this function returns the length in characters
UINT CEditEx::GetBufferLength()
{
//      LPCTSTR lpszText = LockBuffer();
//      UINT nLen = lstrlen(lpszText);
//      UnlockBuffer();
//      return nLen;
        CString str;
        GetWindowText(str);
        UINT nlen = str.GetLength();
        return nlen;
}

void CEditEx::GetSelectedText(CString& strResult)
{
        int nStartChar, nEndChar;
        GetSel(nStartChar, nEndChar);
```

```
        ASSERT((UINT)nEndChar <= GetBufferLength());
        LPCTSTR lpszText = LockBuffer();
        UINT nLen = EndOfLine(lpszText, nEndChar, nStartChar) - nStartChar;
        memcpy(strResult.GetBuffer(nLen), lpszText + nStartChar,
              nLen * sizeof(TCHAR));
        strResult.ReleaseBuffer(nLen);
//      UnlockBuffer();
}

// CEditView Find & Replace

void CEditEx::OnFindNext(LPCTSTR lpszFind, BOOL bNext, BOOL bCase)
{
        if (!FindText(lpszFind, bNext, bCase))
              OnTextNotFound(lpszFind);
}

void CEditEx::OnReplaceSel(LPCTSTR lpszFind, BOOL bNext, BOOL bCase, LPCTSTR lpszRe-
place)
{
        if (!InitializeReplace(lpszFind,bNext,bCase))
            return;
        ReplaceSel(lpszReplace);
        FindText(lpszFind, bNext, bCase);
}

void CEditEx::OnReplaceAll(LPCTSTR lpszFind, LPCTSTR lpszReplace, BOOL bCase)
{
        if (!InitializeReplace(lpszFind,1,bCase) || !SameAsSelected(lpszFind, bCase))
        {
              // initial find was not successful
              return;
        }

        do
        {
              ReplaceSel(lpszReplace);
        } while (FindText(lpszFind, 1, bCase));

}

BOOL CEditEx::InitializeReplace(LPCTSTR lpszFind,BOOL bNext,BOOL bCase)
        // helper to do find first if no selection
{
```

```
        // do find next if no selection
        int nStartChar, nEndChar;
        GetSel(nStartChar, nEndChar);
        if (nStartChar == nEndChar)
        {
                if (! FindText(lpszFind,bNext,bCase))
                {
                        // text not found
                        OnTextNotFound(lpszFind);
                }
                return FALSE;
        }

        if (! SameAsSelected(lpszFind, bCase))
        {
                if (! FindText(lpszFind,bNext,bCase))
                {
                        // text not found
                        OnTextNotFound(lpszFind);
                }
                return FALSE;
        }

        return TRUE;
}
/*
LRESULT CEditEx::OnFindReplaceCmd(WPARAM, LPARAM lParam)
{
        CFindReplaceDialog* pDialog = CFindReplaceDialog::GetNotifier(lParam);
        ASSERT(pDialog != NULL);
        if (pDialog->IsTerminating())
        {
                pDialog = NULL;
        }
        else if (pDialog->FindNext())
        {
                OnFindNext(pDialog->GetFindString(),
                        pDialog->SearchDown(), pDialog->MatchCase());
        }
        else if (pDialog->ReplaceCurrent())
        {
                OnReplaceSel(pDialog->GetFindString(),
                        pDialog->SearchDown(), pDialog->MatchCase(),
                        pDialog->GetReplaceString());
```

```
        }
        else if (pDialog->ReplaceAll())
        {
                OnReplaceAll(pDialog->GetFindString(), pDialog->GetReplaceString(),
                        pDialog->MatchCase());
        }
        return 0;
}
*/
typedef int (WINAPI* AFX_COMPARE_PROC)(LPCTSTR str1, LPCTSTR str2);

BOOL CEditEx::SameAsSelected(LPCTSTR lpszCompare, BOOL bCase)
{
        // check length first
        size_t nLen = lstrlen(lpszCompare);
        int nStartChar, nEndChar;
        GetSel(nStartChar, nEndChar);
        if (nLen != (size_t)(nEndChar - nStartChar))
                return FALSE;

        // length is the same, check contents
        CString strSelect;
        GetSelectedText(strSelect);
        return (bCase && lstrcmp(lpszCompare, strSelect) == 0) ||
                (!bCase && lstrcmpi(lpszCompare, strSelect) == 0);
}
BOOL CEditEx::FindText(LPCTSTR lpszFind, BOOL bNext, BOOL bCase)
{
        ASSERT(lpszFind != NULL);
        ASSERT(*lpszFind != '\0');

        UINT nLen = GetBufferLength();
        int nStartChar, nEndChar;
        GetSel(nStartChar, nEndChar);
        UINT nStart = nStartChar;
        int iDir = bNext ? +1 : -1;

        // can't find a match before the first character
        if (nStart == 0 && iDir < 0)
                return FALSE;

        CWaitCursor wait;
        LPTSTR lpszText;
        GetWindowText(lpszText,UINT_MAX);
```

```
        if (iDir < 0)
        {
                // always go back one for search backwards
                nStart -= (lpszText + nStart) -
                        _tcsdec(lpszText, lpszText + nStart);
        }
        else if (nStartChar != nEndChar && SameAsSelected(lpszFind, bCase))
        {
                // easy to go backward/forward with SBCS
                if (_istlead(lpszText[nStart]))
                        nStart++;
                nStart += iDir;
        }

        // handle search with nStart past end of buffer
        size_t nLenFind = lstrlen(lpszFind);
        if (nStart + nLenFind-1 >= nLen)
        {
                if (iDir < 0 && nLen >= nLenFind)
                {
                        // single-byte character set is easy and fast
                        nStart = nLen - nLenFind;
                        ASSERT(nStart + nLenFind-1 < = nLen);
                }
                else
                {
//                      UnlockBuffer();
                        return FALSE;
                }
        }

        // start the search at nStart
        LPCTSTR lpsz = lpszText + nStart;
        AFX_COMPARE_PROC pfnCompare = bCase ? lstrcmp : lstrcmpi;
                // single-byte string search
                UINT nCompare;
                if (iDir < 0)
                        nCompare = (UINT)(lpsz - lpszText) + 1;
                else
                        nCompare = nLen - (UINT)(lpsz - lpszText) - nLenFind + 1;

                while (nCompare > 0)
                {
```

```
                    ASSERT(lpsz >= lpszText);
                    ASSERT(lpsz+nLenFind-1 <= lpszText+nLen-1);

                    LPSTR lpch = (LPSTR)(lpsz + nLenFind);
                    char chSave = *lpch;
                    *lpch = '\0';
                    int nResult = (*pfnCompare)(lpsz, lpszFind);
                    *lpch = chSave;
                    if (nResult == 0)
                    {
//                      UnlockBuffer();
                        int n = (int)(lpsz - lpszText);
                        SetSel(n, n+nLenFind);
                        ASSERT_VALID(this);
                        return TRUE;
                    }

                    // restore character at end of search
                    *lpch = chSave;

                    // move on to next substring
                    nCompare--;
                    lpsz += iDir;
                }
//              UnlockBuffer();


        return FALSE;
}

void CEditEx::OnTextNotFound(LPCTSTR)
{
        MessageBeep(0);
}

// usetestDlg.h : header file
//
#include "editex.h"
#if !defined(AFX_USETESTDLG_H__81B87D2C_2872_11D2_BA07__444553540000__INCLUD-
ED_)
#define AFX_USETESTDLG_H__81B87D2C_2872_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
```

```
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CUsetestDlg dialog

class CUsetestDlg : public CDialog
{
// Construction
public:
	CUsetestDlg(CWnd* pParent = NULL);  // standard constructor
protected:
	void InitFindReplace(BOOL bFind);
// Dialog Data
	//{{AFX_DATA(CUsetestDlg)
	enum { IDD = IDD_USETEST_DIALOG };
	CEditEx m_edit;
	CFindReplaceDialog* pDlg;
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CUsetestDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CUsetestDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnFind();
	afx_msg void OnReplace();
	afx_msg LONG OnFindReplace(WPARAM wParam, LPARAM lParam);
	afx_msg void OnUndo();
	afx_msg void OnClear();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_USETESTDLG_H__81B87D2C_2872_11D2_BA07_444553540000__
INCLUDED_)

// usetestDlg.cpp : implementation file
//

#include "stdafx.h"
#include "usetest.h"
#include "usetestDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAboutDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
        //}}AFX_MSG
```

```
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUsetestDlg dialog

CUsetestDlg::CUsetestDlg(CWnd* pParent /* =NULL */)
	: CDialog(CUsetestDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CUsetestDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CUsetestDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CUsetestDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}
static UINT WM_FINDREPLACE = ::RegisterWindowMessage(FINDMSGSTRING);
BEGIN_MESSAGE_MAP(CUsetestDlg, CDialog)
```

```
    //{{AFX_MSG_MAP(CUsetestDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(ID_FIND, OnFind)
    ON_BN_CLICKED(ID_REPLACE, OnReplace)
    ON_REGISTERED_MESSAGE(WM_FINDREPLACE, OnFindReplace)
    ON_BN_CLICKED(IDC_UNDO, OnUndo)
    ON_BN_CLICKED(ID_CLEAR, OnClear)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CUsetestDlg message handlers

BOOL CUsetestDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon

    // TODO: Add extra initialization here
    m_edit.SubclassDlgItem(IDC_EDIT, this);
```

```
    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CUsetestDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CUsetestDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
```

```
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CUsetestDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CUsetestDlg::OnFind()
{

	InitFindReplace(TRUE);
}

void CUsetestDlg::OnReplace()
{

	InitFindReplace(FALSE);

}
void CUsetestDlg::InitFindReplace(BOOL bFind)
{

	pDlg = NULL;
	pDlg = new CFindReplaceDialog;
	ASSERT(pDlg! =NULL);
	if(! pDlg->Create(bFind,0,0,FR_DOWN,this))
	{
		pDlg = NULL;
		return;
	}
	ASSERT(pDlg! =NULL);
	pDlg->SetActiveWindow();
	pDlg->ShowWindow(SW_SHOW);

}
LONG CUsetestDlg::OnFindReplace(WPARAM wParam, LPARAM lParam)
{
    pDlg = CFindReplaceDialog::GetNotifier(lParam);
	ASSERT(pDlg! = NULL);
	if (pDlg->IsTerminating())
	{
		pDlg = NULL;
```

```
    }
    else if (pDlg->FindNext())
    {
        m_edit.OnFindNext(pDlg->GetFindString(),
            pDlg->SearchDown(), pDlg->MatchCase());
    }
    else if (pDlg->ReplaceCurrent())
    {
        m_edit.OnReplaceSel(pDlg->GetFindString(),
            pDlg->SearchDown(), pDlg->MatchCase(),
            pDlg->GetReplaceString());
    }
    else if (pDlg->ReplaceAll())
    {
        m_edit.OnReplaceAll(pDlg->GetFindString(), pDlg->GetReplaceString(),
            pDlg->MatchCase());
    }
    return 0;
}

void CUsetestDlg::OnUndo()
{
    // TODO: Add your control notification handler code here

}

void CUsetestDlg::OnClear()
{

    m_edit.SetSel(0,-1);
}
```

3.5.2 Sliderspin 例程

该例程说明滑块、滚动条、滚动控件以及颜色对话框的使用，如图 3.11 所示。调节代表红绿蓝三色的滚动控件可调整对话框上圆的颜色显示，也可在三个编辑控件中直接输入范围在 0～255 的颜色值，圆的颜色立刻改变。右下角“自定颜色”按钮的功能也是改变圆的颜色。点击“自定颜色”按钮后弹出颜色对话框，任选一颜色后，圆的颜色当即改变，同时在三个编辑控件内显示当前所选颜色的 RGB 值。对话框左下角的滑块用来改变圆的大小，每当用鼠标移动滑块的指针(thumber)时，圆随着或大或小，对话框右边的竖直滚动条也随着上下移动。

下面是创建该例程的步骤：

1)可参照前面的例子用 MFC AppWizard 创建名为 Sliderspin 的基于对话框的工程。

2)添加对话框资源。



图 3.11　Sliderspin 例程界面图

对照本例子的演示图，在对话框 IDD _ SLIDERSPIN _ DIALOG 上放置除圆之外的所有控件，并调整各控件的相对位置和大小，以便适合您的口味。把对话框的标题通过对话框的属性对话框改为“游标和滑块”。

编辑控件的 ID 值从上到下依次为 IDC _ EDIT _ RED，IDC _ EDIT _ BLUE 和 IDC _ EDIT _ GREEN，滑块的 ID 为 IDC _ SLIDER，滚动条的 ID 为 IDC _ SCROLL，自定颜色按钮的 ID 为 IDC _ COLOR，位于编辑控件右边的滚动控件从上到下其 ID 依次设为 IDC _ SPIN _ RED，IDC _ SPIN _ BLUE 和 IDC _ SPIN _ GREEN。

3）用 ClassWizard 增加控件变量，列表如下：

表 3.3　用 ClassWizard 增加的控件变量

Control IPS	Type	Member
IDC _ EDIT _ BLUE	int	m _ nBlue
IDC _ EDIT _ GREEN	int	m _ nGreen
IDC _ EDIT _ RED	int	m _ nRed
IDC _ COLOR	int	m _ nColor
IDC _ SCROLL	CScrollBar	m _ Scroll
IDC _ SLIDER	CSliderctrl	m _ Slider
IDC _ SPIN _ BLUE	CSpinButtonCtrl	m _ SpinBlue
IDC _ SPIN _ GREEN	CSpinButtonCtrl	m _ SpinGreen
IDC _ SPIN _ RED	CSpinButtonCtrl	m _ SpinRed

4）添加初始化代码。

在 Class CSliderSpinDlg：public Dialog 声明中加 入下列代码：

```
private CRect m_ Rect;
int m _ nwidth;
COLORREF m_ color;
```

在构造函数 CSlider SpinDlg::CsliderspinDlg 中添加以下代码：

```
m_nWidth:10;
m_kect:CRect(220,25,320,125);
```

滑块、滚动条和滚动控件需要在 CSliderspinDlg::OnInitdialog 中添加初始代码，在对话框打开时进行初始设置，代码如下所示：

```
BOOL CSliderSpinDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon


    m_spinRed.SetBuddy(GetDlgItem(IDC_EDIT_RED));
    m_spinGreen.SetBuddy(GetDlgItem(IDC_EDIT_GREEN));
    m_spinBlue.SetBuddy(GetDlgItem(IDC_EDIT_BLUE));

    m_spinRed.SetRange(0,255);
    m_spinGreen.SetRange(0,255);
    m_spinBlue.SetRange(0,255);
    m_Slider.SetRange(1,20);
    m_Slider.SetPos(10);
```

```
    m_Scroll.SetScrollRange(1,20);
    m_Scroll.SetScrollPos(10);
    return TRUE;  // return TRUE  unless you set the focus to a control
}
```

5)用 ClassWizard 处理消息。

在 ClassWizard 中选择 CSliderSpinDlg,然后在其中加入如下成员(表 3.4):

表 3.4 在 CSliderSpinDlg 中添加成员

函数对象 ID	消息	成员函数名
IDC_COLOR	BN_CLICKED	Oncolor
IDC_EDIT_BLUE	EN_CHANGE	OnChangeEditBlue
IDC_EDIT_GREEN	EN_CHANGE	OnChangeEditGreen
IDC_EDIT_RED	EN_CHANGE	OnChangeEditRed
IDC_SLIDER	WM_HSCROLL	OnScroll

在上述各函数中所要加入的代码如下所示:

```
void CSliderSpinDlg::OnChangeEditBlue()
{

    m_nBlue=m_spinBlue.GetPos();
    InvalidateRect(m_Rect);
}

void CSliderSpinDlg::OnChangeEditGreen()
{

    m_nGreen=m_spinGreen.GetPos();
    InvalidateRect(m_Rect);
}

void CSliderSpinDlg::OnChangeEditRed()
{

    m_nRed=m_spinRed.GetPos();
    InvalidateRect(m_Rect);
}

void CSliderSpinDlg::OnHScroll(UINT nSBCode, UINT nPos, CSliderCtrl * pSlider)
{

    m_nWidth=pSlider->GetPos();
```

```
        InvalidateRect(m_Rect);
        m_nScroll=m_nWidth;
        UpdateData(false);

}
void CSliderSpinDlg::OnColor()
{

        CColorDialog dlg(0,CC_RGBINIT,this);
        if(IDOK==dlg.DoModal())
        {
                m_color=dlg.GetColor();
                m_nGreen=GetGValue(m_color);
                m_nRed=GetRValue(m_color);
                m_nBlue=GetBValue(m_color);
                UpdateData(false);
                InvalidateRect(m_Rect);
        }
```

最后在 CSliderSpinDlg::OnPaint()函数的语句 CDialog::OnPaint();之后加上如下语句：

```
void CSliderSpinDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting

                SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
                CClientDC dc(this);
                CRect rect(270-m_nWidth*5/2,75-m_nWidth*5/2,
                        270+m_nWidth*5/2,75+m_nWidth*5/2);
```

```
        m_color = RGB(m_nRed,m_nGreen,m_nBlue);
            CBrush brushNew(m_color);
            CBrush* pOldBrush = dc.SelectObject(&brushNew);
            dc.Ellipse(rect);
            dc.SelectObject(pOldBrush);

        }
}
// SliderSpinDlg.h : header file
//

#if !defined(AFX_SLIDERSPINDLG_H__F10B49A8_181E_11D2_BA07_444553540000__INCLUDED_)
#define AFX_SLIDERSPINDLG_H__F10B49A8_181E_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CSliderSpinDlg dialog

class CSliderSpinDlg : public CDialog
{
// Construction
public:
    CSliderSpinDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    //{{AFX_DATA(CSliderSpinDlg)
    enum { IDD = IDD_SLIDERSPIN_DIALOG };
    CScrollBar  m_Scroll;
    CSpinButtonCtrl  m_spinRed;
    CSpinButtonCtrl  m_spinGreen;
    CSpinButtonCtrl  m_spinBlue;
    CSliderCtrl m_Slider;
    int         m_nBlue;
    int         m_nGreen;
    int         m_nRed;
    int         m_nScroll;
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSliderSpinDlg)
```

```
    protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;

        // Generated message map functions
        //{{AFX_MSG(CSliderSpinDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnChangeEditBlue();
        afx_msg void OnChangeEditGreen();
        afx_msg void OnChangeEditRed();
        afx_msg void OnHScroll(UINT nSBCode,UINT nPos,CSliderCtrl*   pSlider);
        afx_msg void OnColor();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        CRect m_Rect;
        int m_nWidth;
    COLORREF m_color;

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_SLIDERSPINDLG_H__F10B49A8_181E_11D2_BA07_444553540000__
INCLUDED_)

// SliderSpinDlg.cpp : implementation file
//

#include "stdafx.h"
#include "SliderSpin.h"
#include "SliderSpinDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
```

```
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
```

```
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSliderSpinDlg dialog

CSliderSpinDlg::CSliderSpinDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CSliderSpinDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CSliderSpinDlg)
    m_nBlue = 0;
    m_nGreen = 0;
    m_nRed = 0;
    m_nScroll = 0;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    m_nWidth=10;
    m_Rect=CRect(220,25,320,125);
}

void CSliderSpinDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CSliderSpinDlg)
    DDX_Control(pDX, IDC_SCROLL, m_Scroll);
    DDX_Control(pDX, IDC_SPIN_RED, m_spinRed);
    DDX_Control(pDX, IDC_SPIN_GREEN, m_spinGreen);
    DDX_Control(pDX, IDC_SPIN_BLUE, m_spinBlue);
    DDX_Control(pDX, IDC_SLIDER, m_Slider);
    DDX_Text(pDX, IDC_EDIT_BLUE, m_nBlue);
    DDX_Text(pDX, IDC_EDIT_GREEN, m_nGreen);
    DDX_Text(pDX, IDC_EDIT_RED, m_nRed);
    DDX_Scroll(pDX, IDC_SCROLL, m_nScroll);
    //}}AFX_DATA_MAP

BEGIN_MESSAGE_MAP(CSliderSpinDlg, CDialog)
    //{{AFX_MSG_MAP(CSliderSpinDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
```

```
    ON_EN_CHANGE(IDC_EDIT_BLUE, OnChangeEditBlue)
    ON_EN_CHANGE(IDC_EDIT_GREEN, OnChangeEditGreen)
    ON_EN_CHANGE(IDC_EDIT_RED, OnChangeEditRed)
    ON_WM_HSCROLL()
    ON_BN_CLICKED(IDC_COLOR, OnColor)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSliderSpinDlg message handlers

BOOL CSliderSpinDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon


    m_spinRed.SetBuddy(GetDlgItem(IDC_EDIT_RED));
    m_spinGreen.SetBuddy(GetDlgItem(IDC_EDIT_GREEN));
    m_spinBlue.SetBuddy(GetDlgItem(IDC_EDIT_BLUE));

    m_spinRed.SetRange(0,255);
```

```
    m_spinGreen.SetRange(0,255);
    m_spinBlue.SetRange(0,255);
    m_Slider.SetRange(1,20);
    m_Slider.SetPos(10);
    m_Scroll.SetScrollRange(1,20);
    m_Scroll.SetScrollPos(10);
    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CSliderSpinDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSliderSpinDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
```

```
        }
        else
        {
                CDialog::OnPaint();
                CClientDC dc(this);
                CRect rect(270-m_nWidth*5/2,75-m_nWidth*5/2,
                        270+m_nWidth*5/2,75+m_nWidth*5/2);
          m_color=RGB(m_nRed,m_nGreen,m_nBlue);
                CBrush brushNew(m_color);
                CBrush* pOldBrush=dc.SelectObject(&brushNew);
                dc.Ellipse(rect);
                dc.SelectObject(pOldBrush);

        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSliderSpinDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CSliderSpinDlg::OnChangeEditBlue()
{

        m_nBlue=m_spinBlue.GetPos();
        InvalidateRect(m_Rect);
}

void CSliderSpinDlg::OnChangeEditGreen()
{

        m_nGreen=m_spinGreen.GetPos();
        InvalidateRect(m_Rect);
}

void CSliderSpinDlg::OnChangeEditRed()
{

        m_nRed=m_spinRed.GetPos();
        InvalidateRect(m_Rect);
}
```

```
void CSliderSpinDlg::OnHScroll(UINT nSBCode, UINT nPos, CSliderCtrl * pSlider)
{

    m_nWidth = pSlider->GetPos();
    InvalidateRect(m_Rect);
    m_nScroll = m_nWidth;
    UpdateData(false);

}
void CSliderSpinDlg::OnColor()
{

    CColorDialog dlg(0,CC_RGBINIT,this);
    if(IDOK==dlg.DoModal())
    {
        m_color = dlg.GetColor();
        m_nGreen = GetGValue(m_color);
        m_nRed = GetRValue(m_color);
        m_nBlue = GetBValue(m_color);
        UpdateData(false);
        InvalidateRect(m_Rect);
    }

}
```

3.6 小　　结

本章讲述了静态文本控件、列表框、组合框、滚动条、滑块五个标准控件和六个公共对话框。一般的 Windows 应用通常要用到这些控件并和其它一些控件结合使用，来使得应用更加生动，并且提供更加规范的用户界面。通过本章的学习，要掌握以下内容。

- 如何创建静态文本控件，并在运行时改变它们。
- 如何创建列表框和组合框，怎样动态地删除、加入列表项。
- 如何创建滚动条和滑块，并熟练使用。
- 学会使用六个标准的公共对话框。

第4章 对 话 框

对话框在 Windows 应用中起着非常重要的作用，它是用户与应用程序的主要交互途径。对话框就像一个容器能够装载我们在上一章介绍的任何控件，因此对话框是一个真正的窗口，不但可以接收消息，而且还可以被移动和关闭，甚至可以在它的用户区中进行绘图操作。对话框分为"有模式"和"无模式"两大类，本章先介绍对话框的一般知识，然后分别介绍如何创建有模式和无模式对话框。

4.1 对话框的基础知识

Developer Studio 的对话框编辑器使得对话框的设计以及对话模板资源的创建更为容易，而 Class Wizard 则简化了对话框控件的初始化和验证以及获取用户输入的过程。在 MFC 类库中对话框由 CDialog 类管理。

4.1.1 对话框的组成

对话框主要由两部分组成。

1)对话模板资源：对话模板资源用于定义对话框控件及其分布。对话资源中保存有用于创建并显示对话窗口的对话模板，对话模板定义对话框的特性(如大小、风格等)以及对话框控件的类型和位置。除了直接使用作为资源保存的对话模板外，还可根据需要在内存中创建对话模板。

2)对话类：从 CDialog 类派生的对话类提供编程接口来管理对话框。每个对话框是一个窗口，对话窗口创建后，将自动根据对话模板资源来创建对话框控件。

4.1.2 MFC 对对话框的支持

当程序员创建自己的对话框时，最重要的 MFC 是 CDialog 类。它是 MFC 最基本的对话框类。当建立自己特定的对话框时，往往需要从这个基类派生出一个子类。建立对话框之前，程序员往往要首先查看一下预定义的对话框。MFC 用来创建对话框的类除 CDialog 外，还有 CPropertysheet(属性单)，Cpropertypage(属性页)，其中属性单(或称标签对话框)是一个复合对话框，它含有多个页，每页为单独的对话框，即称为属性页，而每个属性页拥有一组用来更新一些相关域的对话框控制。MFC 有两大类预定义的对话框类。一种是由 COLORDialog 派生的，它们用来建立 OLE 复合文档应用。另一种预定义的对话框类称为通用对话框(Common Dialog)，它们提供应用程序中经常需要的功能。最后一个有趣的对话框类是 CDialogBar——对话条类。它其实是派生于 CcontrolBar。CDialogBar 类建立一个连接在框架窗口边框上的窗口，其可以拥有任何类型的对话框控制。

4.1.3 有模式和无模式对话框

当程序建立一对话框时，就必须决定该对话框是有模式的还是无模式的。从编程的观点

看，意味着要在建立和撤消两个对话框的出入集合之间进行选择。从用户观点来看，对话框的模式影响到在对话框可见的情况下是否可进行其它工作的问题。模式是指对话框不同状态的一个术语。有模式对话框(Modal Dialog Box)是指具有排他属性的对话框，而无模式对话框(Modalless Dialog Box)是指不具有排他性质的对话框。有模式对话框工作在排他的状态，即使用户可以看到窗口其它的用户界面，如主窗口、菜单、工具条等，但用户此时不能使用它们。有模式对话框一旦进入排他状态，它就禁止了其它的用户界面对象。如果用户在启动了有模式对话框后又要选择菜单，只能听到嘟声，提醒用户除非撤消该模式对话框，否则对话框就控制了应用。为了保持完整性，值得指出的是，有模式对话框又分两种：应用模式和系统模式。在应用模式的对话框激活时，只是阻止了本应用的其它用户界面的使用，不阻止其它应用的用户界面的使用。而系统模式的对话框激活时，用户只能对该对话框进行响应，系统中其它部分被禁止。用户绝大多数情况下建立的都是应用模式的对话框。Windows 的系统模式对话框会中断多任务处理，所以平时极少使用。无模式对话框在非排斥状态工作。当这种对话框激活时，用户仍可和其它的用户界面进行交互，能够选择菜单，主窗口也能改变其大小，或缩小化为图标甚至关闭。事实上，除了含有对话框控制，无模式对话框和应用中的其它窗口是一样的。

从上面的介绍中，读者可能以为无模式对话框更为友好、更易于使用，因此会比不友好的有模式对话框使用得更多一些。事实上恰恰相反，用户遇到的大多数对话框都是有模式对话框。有模式对话框使用的更为普遍的原因是由于对话框的使用方法所致。作为菜单系统的扩展，对话框提供了应用程序需要完成命令的有关数据。通过有模式对话框，应用程序的其它用户界面被阻止，直到用户决定了是否继续执行命令(通过输入数据并点击 OK 按钮)或退出命令执行的请求(通过点击 Cancel 按钮或按 Esc 键)。尽管使用不多，无模式对话框还是有它的作用。一个普遍的例子是文本查找对话框，它能够让用户编辑找到的文本，且能继续查找。尽管对话条不属于对话框，但它可以被当作无模式对话框来看待。虽然存在差异，但有模式对话框和无模式对话框的基本建立步骤都是一样的。下一节，我们将讨论创建这两种对话框的五个基本步骤，同时也会指出两者在方法上的差异。

4.1.4 对话框消息处理

由于对话框亦为窗口，故同其它窗口一样，也需通过对消息的响应来操作。对话框和其它类型窗口的区别在于所处理的消息种类不同，其它类型的窗口往往要处理几十个消息，而对话框中常常只关心的是两类消息：WM_INITDIALOG 和 WM_COMMAND。顾名思义，消息 WM_INITDIALOG 通知我们进行对话框的初始化操作，而其它类型的窗口则在响应消息 WM_CREATE 时对窗口进行初始化。当对话框得到 WM_CREATE 消息时，其所拥有的控制仍是空的。当接收到消息 WM_INITDIALOG 时，对话框已处于以下状态：首先所有的控制已经建立完毕，且允许对它们进行初始化，例如，此时可以增加编辑控制的内容或列表框的字符串；其次，当这个消息出现时，对话框本身还没有显示出来，即程序员能够初始化对话框中的控制而不用担心用户会看到对话框中的内容。

对话框的另一重要消息 WM_COMMAND 是由对话框控制发出，用来通知对话框一些它可能感兴趣的文件。当对话框控制发出该消息时，它将自己的名字和通知码(Notification Codes)一起发送出去。通知码用于说明消息的原因，每种控制都有各自的通知码。如按钮发送 BN_CLICKED 通知来表明它被鼠标击中了，编辑控制发送 EN_CHANGE 通知来表明编辑内容被改变了。同菜单项一样，对话框控制也使用 ID 来使接收者知道发送消息的对象。

在一个拥有菜单和控制的窗口中，要注意不要将菜单命令 ID 和对话框控制的 ID 重合。来自对话框控制的 WM_COMMAND 消息和来自菜单项的消息处理方法不同，菜单项只有在用户选择时才发出命令消息，而对话框则在所有情况下都可能发出消息，故对对话框消息的处理要有更多的选择性。事实上，程序员只需关心这种消息中的很少一部分即可。

4.1.5 对话数据交换和验证

对话数据交换(Dialog Data Exchange 缩写为 DDX)用于初始化对话框中的控件并获取用户的数据输入。而对话数据验(Dialog Data Verify 缩写为 DDV)则用于自动验证对话框中的数据输入。要在对话框中使用 DDX 和 DDV 必须用 ClassWizard 创建数据成员，设置数据类型并指定验证规则。

1. 数据交换

如果使用 DDX 机制，通常在 OnInitDialog 函数或对话框对象的构造函数中设置对话框对象的数据成员的初值。在即将显示对话框之前，DDX 机制把成员变量的值传给对话框中的控件，当对话框在响应 DoModal 或 Create 而被显示时将在对话框控件中显示这些数据。CDialog 成员函数 OnInitDialog 缺省时调用 CWnd 类的成员函数 UpdateData 来初始化对话框控件。当用户单击"OK"按键或以 TRUE 为参数调用成员函数 UpdateData 时，DDX 机制将把值从对话框控件中传给数据成员。DDX 机制将对指定有验证规则的所有数据项进行验证。

函数 UpdateData 原型为：

Bool UpdateData(BOOL bsaveAndValidate = TRUE);

如果参数 bSaveAndValidate 为 FALSE，则初始化对话框；如果为 TRUE，则获取并验证对话数据。CDialog::OnInitDialog 的缺省实现是以 FALSE 为参数调用 UpdateData。而 CDialog::OnOK 的缺省实现是以 TRUE 为参数调用 UpdateData。执行交换任务时，UpdateData 建立 CDataExchange 对象并调用 CDialog 派生类的覆盖成员函数 DoDataExchange。函数 DoDataExchange 参数为指向 CDataExchange 的指定和覆盖成员函数 DoDataExchange 时，要为每个数据成员(控件)指定对 DDX 函数的一次调用。根据 UpdateData 传给 DoDataExchange 的 CDataExchange 参数，每个 DDX 函数都知道如何交换数据。

2. 数据验证

除了指定数据交换外，还可以调用 DDV 函数指定数据验证。如果验证失败，DDV 函数将显示消息框提示用户，并把焦点放在违反规则的控件上以使用户重新输入数据。给定控件的 DDV 函数必须紧跟在同一控件的 DDV 函数调用之后。必须使用 ClassWizard 编写数据映射中的所有 DDX 和 DDV 调用，不要人工编辑定界符之间的数据映射行。

4.2 创建模态对话框

创建对话框很容易，简单到用鼠标点几下，写一、二条语句，但要创建一个美观实用、毫无毛病的对话框也决非易事。一般建立模态对话框有五个步骤。

1)建立对话框模板，确定对话框中包括哪些控件以及放置这些控件的位置。

2)产生一个 CDialog 的派生类来为对话框处理消息，这一步较为简单，因为 ClassWizard

会替程序员编写这些代码。

3)需要程序编写建立对话框的代码,这将对后面的两步中的测试和代码优化提供基础。

4)增加 CDialog 派生类的对话框初始化代码,这是对 WM _ INITDIALOG 消息进行响应,设置所有控制的初始化状态。

5)编写处理控制发送通知的处理函数。

4.2.1 消息对话框

消息对话框是一种特殊的模态对话框,又称预定义对话框,它们的外观已预先定义好了,只能对他们做很小的修改,在屏幕上显示对话框的最快的方法就是使用 CWnd 的成员函数 Messagebox(),如下列语句所示:

Int MessageBox(lpText,lpCaption = NULL,WType = MB _ OK);

其中的 lpText 是指向消息框中将使用的文本的指针,lpCaption 是指向在消息框的标题栏中将使用的标题文本的指针,WType 指定了消息框中可拥有的控件。MessageBox()可用来设计具有一定长和宽的对话框,使用户可通过 OKCancel,Abort,Retry,Ignore,Yes 或 No 这些按钮与程序进行通信。通过设置 WType 参数(见表 4.1)来确定消息框的类型。消息对话框出现之后,当用户单击对话框中的按钮或关闭对话框时,函数 MessageBox()返回一整数值,对应于以下常量之一。

表 4.1 WType 参数设置

返回值	含义
IDABORT	单击 Abort 按钮
IDCANCEL	单击 Cancel 按钮
IDIGNORE	单击 Ignore 按钮
IDNO	单击 No 按钮
IDOK	单击 OK 按钮
IDRETRY	单击 Retry 按钮
IDYES	单击 Yes 按钮

表 4.2 列出有关消息框的类型及它们对应的 WType 含义。

表 4.2 消息框类型对应的 WType 含义

消息框的类型	WType 含义
MB _ ABORTRETRYIGNORE	Abort,Retry,Ignore 按钮
MB _ APPLMODAL	应用程序一级上的模态
MB _ DEFBUTTON1	第一个按钮是缺省的
MB _ DEFBUTTON2	第二个按钮是缺省的
MB _ DEFBUTTON3	第三个按钮是缺省的
MB _ YESNOCANCEL	Yes No 和 Cancel 按钮
MB _ SYSTEMMODAL	系统一级上的模态
MB _ RETRYCANCEL	Retry 和 Cancel 按钮

续表

返回值	含义
MB_OKCANCEL	OK 和 Cancel 按钮
MB_OK	OK 按钮
MB_ICONSTOP	包括停止符号图标
MB_ICONQUESTION	包括问号图标
MB_ICONINFORMATION	包括 i 圈图标
MB_ICONHAND	与 MB_ICONSTOP 相同
MB_ICONEXCLAMATION	包括声明指示图标
MB_ICONASTERISK	与 MB_ICONINFORMATION 相同

4.3 创建无模式对话框

上一节介绍了有模式对话框的创建以及控制。它是通过调用 DoModal 函数来直接显示一个有模式对话框的，当函数返回时，对话框将不再存在。而无模式对话框在它处于激活状态的同时，还允许用户做别的工作。创建无模式对话框要比创建有模式对话框复杂一些，我们首先需要构造一个对话框窗口，然后通过调用 CDialog 的 Create 成员函数来创建对话框窗口，这次调用的不再是 DoModal 函数，并且当对话框仍然留在屏幕上时，控制已经立刻返回了。

CDialog 类的 Create()函数原型有以下两种：

```
BOOL Create(LPOTSTR lpszTemplateName,CWnd pParentWnd=NULL);
BOOL Create(UINT nIDTemplate,CWnd pParentWnd=NULL);
```

参数 lpszTemplateName 是指向一个对话框模板资源名字符串的指针，pParentWnd 指向此对话对象所属的父窗口对象，如果该参数为 NULL，则对话对象的父窗口设置为主应用程序窗口，nIDTemplate 包含一个对话框模板资源的标识符。模态对话框与非模态对话框的另一个非常重要的差别在于：当在模态对话框中按下 OK 或 Cancel 按钮时(通过类方法 CDialog::OnOK()或 CDialog::OnCancal())，或者当调用 CDialog::EndDialoge 时，对话框自动关闭。而在非模态对话框中，调用基类 CDialog::OnCancel()仅仅调用 CDialog::EndDialog()，它隐藏对话框但并不删除它。解决的办法是覆盖 OnCancal()函数，并从中调用 DestroyWindow()。

4.4 实　　例

该例子演示了怎样生成一个无模态对话框，如图 4.1 所示。当在视窗内单击时会弹出一无模态对话框，该对话框有一标签控件，三页分别是 Tree、List 和 Edit，相应每页都有一对应的控件。下面具体介绍创建该工程的步骤。

1)创建单文档工程 md。

2)添加一对话框(IDC_DIALOG)，实现的类为 CMDialog，另类为 CDialog。在对话框上加上 Tab control(IDC_TAB)，并在这标签控件里面加三个一样大小的树视控件(IDC_TREE)。列视控件(IDC_LIST)和编辑控件(IDC_EDIT)。

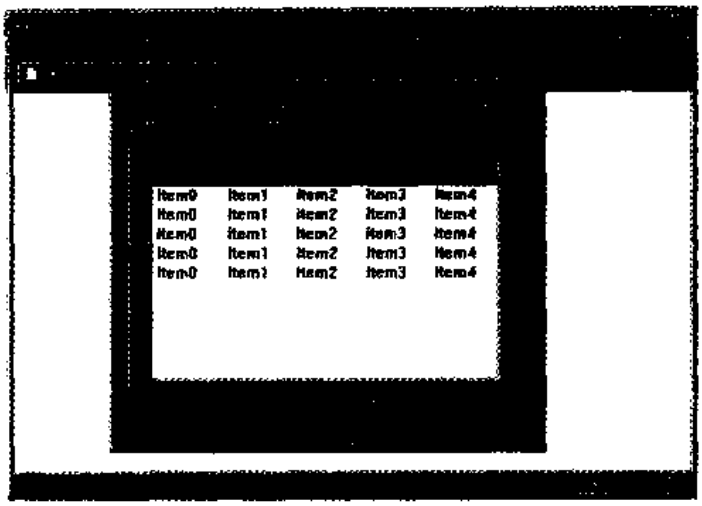


图 4.1　无模态对话框例程图示

3)编辑文件 Mdialog.h ,Mdialog.CPP,Mdview .h 和 Mdview. cpp 如下所示:

```
#if ! defined(AFX_MDIALOG_H__F04DACAB_1F51_11D2_BA07_444553540000__INCLUDED
_)
#define AFX_MDIALOG_H__F04DACAB_1F51_11D2_BA07_444553540000__INCLUDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// MDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CMDialog dialog

class CMDialog : public CDialog
{
// Construction
public:
	CMDialog(CWnd* pParent = NULL);
	CMDialog(CView* pView);
	BOOL Create();
public:
	CView* m_pView;
// Dialog Data
	//{{AFX_DATA(CMDialog)
	enum { IDD = IDD_DIALOG1 };
	CTreeCtrl	m_tree;
	CTabCtrl	m_tab;
	CListCtrl	m_list;
	CEdit m_edit;
```

```
        //{{AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMDialog)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CMDialog)
        virtual void OnCancel();
        virtual BOOL OnInitDialog();
      afx_msg void OnClickTab(NMHDR * pNotifyStruct, LRESULT* pResult);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_MDIALOG_H__F04DACAB_1F51_11D2_BA07_444553540000__IN-
CLUDED_)

// MDialog.cpp : implementation file
//

#include "stdafx.h"
#include "md.h"
#include "MDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMDialog dialog
```

```
CMDialog::CMDialog(CWnd* pParent /* =NULL */)
	: CDialog(CMDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CMDialog)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	m_pView=NULL;
}

CMDialog::CMDialog(CView* pView)
{
	m_pView=pView;
}

BOOL CMDialog::Create()
{
	return CDialog::Create(CMDialog::IDD);
}

void CMDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CMDialog)
	DDX_Control(pDX, IDC_TREE, m_tree);
	DDX_Control(pDX, IDC_TAB, m_tab);
	DDX_Control(pDX, IDC_LIST, m_list);
	DDX_Control(pDX, IDC_EDIT, m_edit);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CMDialog, CDialog)
	//{{AFX_MSG_MAP(CMDialog)
	ON_NOTIFY(NM_CLICK,IDC_TAB,OnClickTab)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMDialog message handlers

void CMDialog::OnCancel()
{

	DestroyWindow();
}
```

```
BOOL CMDialog::OnInitDialog()
{
	CDialog::OnInitDialog();
	TC_ITEM tci[3];
	CString str[3] = {"Tree","List","Edit"};
	for(int i=0;i<3;i++)
	{
		tci[i].mask= TCIF_TEXT;
		tci[i].pszText=(LPSTR)(LPCTSTR)str[i];
		tci[i].cchTextMax=str[i].GetLength();
		m_tab.InsertItem(i,&tci[i]);
	}
	CRect rect;
	m_list.GetWindowRect(&rect);
	char *colText[]={"col0","col1","col2","col3","col4"};
	char *itemText[]={"item0","item1","item2","item3","item4"};
	LV_COLUMN lvc;
	lvc.mask=LVCF_TEXT|LVCF_WIDTH|LVCF_SUBITEM|LVCF_FMT;
	lvc.fmt=LVCFMT_LEFT;
	lvc.cx=rect.Width()/5;
	for(i=0;i<5;i++)
	{
		lvc.iSubItem=i;
		lvc.pszText=colText[i];
		if(m_list.InsertColumn(i,&lvc)==-1)
			return NULL;
	}
	LV_ITEM lvi;
	lvi.mask=LVIF_TEXT;
	for(i=0;i<5;i++)
	{
		lvi.iItem=i;
		lvi.iSubItem=0;
		lvi.cchTextMax=sizeof(itemText)/sizeof(itemText[0]);
		lvi.pszText=itemText[0];
		if(m_list.InsertItem(&lvi)==-1)
			return NULL;
		for(int iSubItem=1;iSubItem<5;iSubItem++)
		{
			m_list.SetItemText(i,iSubItem,itemText[iSubItem]);
		}
	}
	TV_INSERTSTRUCT tvi;
```

```
        tvi.hParent = TVI_ROOT;
        tvi.hInsertAfter = TVI_LAST;
        tvi.item.mask = TVIF_TEXT;
        tvi.item.pszText = "father1";
        HTREEITEM hItem1 = m_tree.InsertItem(&tvi);

        tvi.hParent = hItem1;
        tvi.item.pszText = "child1";
        m_tree.InsertItem(&tvi);

        tvi.hParent = TVI_ROOT;
        tvi.item.pszText = "father2";
        HTREEITEM hItem2 = m_tree.InsertItem(&tvi);

        tvi.hParent = hItem2;
        tvi.item.pszText = "child2";
        m_tree.InsertItem(&tvi);

        tvi.hParent = TVI_ROOT;
        tvi.item.pszText = "father3";
        HTREEITEM hItem3 = m_tree.InsertItem(&tvi);

        tvi.hParent = hItem3;
        tvi.item.pszText = "child3";
        m_tree.InsertItem(&tvi);

        m_tree.Expand(hItem1,TVE_EXPAND);
        m_tree.Expand(hItem2,TVE_EXPAND);
        m_tree.Expand(hItem3,TVE_EXPAND);



        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
void CMDialog::OnClickTab(NMHDR* pNotifyStruct, LRESULT* pResult)
{
        int nCurtab=m_tab.GetCurSel();
        switch(nCurtab)
        {
        case 0:
                m_tree.ShowWindow(SW_SHOWNORMAL);
                m_edit.ShowWindow(SW_HIDE);
                m_list.ShowWindow(SW_HIDE);
```

```
            break;
        case 1:
            m_list.ShowWindow(SW_SHOWNORMAL);
            m_tree.ShowWindow(SW_HIDE);
            m_edit.ShowWindow(SW_HIDE);
            break;
        case 2:
            m_edit.ShowWindow(SW_SHOWNORMAL);
            m_tree.ShowWindow(SW_HIDE);
            m_list.ShowWindow(SW_HIDE);
            break;
        }
        *pResult = 0;
}
// mdView.h : interface of the CMdView class
//
/////////////////////////////////////////////////////////////////////////////
#include "MDialog.h"

#if !defined(AFX_MDVIEW_H__F04DACA3_1F51_11D2_BA07_444553540000__INCLUDED_)
#define AFX_MDVIEW_H__F04DACA3_1F51_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMdView : public CView
{
protected: // create from serialization only
    CMdView();
    DECLARE_DYNCREATE(CMdView)

// Attributes
public:
    CMdDoc* GetDocument();
    CMDialog* m_pDlg;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMdView)
    public:
```

```
        virtual void OnDraw(CDC * pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo * pInfo);
        virtual void OnBeginPrinting(CDC * pDC, CPrintInfo * pInfo);
        virtual void OnEndPrinting(CDC * pDC, CPrintInfo * pInfo);
        //{{AFX_VIRTUAL

// Implementation
public:
        virtual ~CMdView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CMdView)
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in mdView.cpp
inline CMdDoc * CMdView::GetDocument()
   { return (CMdDoc *)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_MDVIEW_H__F04DACA3_1F51_11D2_BA07_444553540000__INCLUDED_)

// mdView.cpp : implementation of the CMdView class
//

#include "stdafx.h"
```

```
# include "md.h"

# include "mdDoc.h"
# include "mdView.h"
# include "MDialog.h"

# ifdef _DEBUG
# define new DEBUG_NEW
# undef THIS_FILE
static char THIS_FILE[] = __FILE__;
# endif

/////////////////////////////////////////////////////////////////////////////
// CMdView

IMPLEMENT_DYNCREATE(CMdView, CView)

BEGIN_MESSAGE_MAP(CMdView, CView)
	//{{AFX_MSG_MAP(CMdView)
	ON_WM_LBUTTONDOWN()
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMdView construction/destruction

CMdView::CMdView()
{

	m_pDlg=new CMDialog();
}

CMdView::~CMdView()
{
	delete m_pDlg;
}

BOOL CMdView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
```

```
        //  the CREATESTRUCT cs

        return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMdView drawing

void CMdView::OnDraw(CDC* pDC)
{
        CMdDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        // TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CMdView printing

BOOL CMdView::OnPreparePrinting(CPrintInfo* pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}

void CMdView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        // TODO: add extra initialization before printing
}

void CMdView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)

        // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CMdView diagnostics

#ifdef _DEBUG
void CMdView::AssertValid() const
{
        CView::AssertValid();
}
```

```
void CMdView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CMdDoc* CMdView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMdDoc)));
    return (CMdDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMdView message handlers

void CMdView::OnLButtonDown(UINT nFlags, CPoint point)
{

    if(m_pDlg->GetSafeHwnd()==0)
        m_pDlg->Create();
    CView::OnLButtonDown(nFlags, point);
}
```

4.5 小　　结

本章讲解了对话框的基本知识，重点介绍了无模态对话框与有模态对话框的含义和区别，并以一实例具体介绍了创建无模态对话框的过程及其应用。通过本章学习，读者应能够做到以下几点：

- 区别有模态和无模态两种不同类型的对话框。
- 自己动手编写对话框例程。

相信只要您将本章内容弄熟，应该会实现此等目标。

第 5 章　文本和字体

图形设备接口(Graphics Device Interface 缩写为 GDI)是 Windows 用来进行绘图的重要工具,只要看一看身边的由计算机制作的多媒体世界就知道 GDI 的作用了。我们在编程时,经常需要在屏幕或打印机上绘图,都需要调用 GDI 函数,它绘制的对象不管最终多么复杂,其实都是由一些基本的元素如点、线、矩形、多边型、椭圆、位图以及文本等所组成。本章先介绍 Windows 图形的基础设备环境类,绘图时用的工具如笔、刷子等的 GDI 对象,然后说明如何使用颜色映射的方法,并用一节来介绍字体的使用。

5.1　设备环境类

Windows 设备环境是 GDI 的关键元素,它代表了物理设备,每一个 C++ 设备环境都有与之相对应的 Windows 设备环境,并且通过一个类似于 HDC 的句柄来标识。MFC 封装 Windows 提供的不同类型设备描述表进入不同的 DC 类,它封装了 C++ 类中的设备描述表句柄。此类有效地包含了有关设备的绘图属性信息,所有在 Windows 上制作绘图即是在设备描述表上制作,且全部绘图方法都是被封入的 DC 对象。

下列 DC 类是为设备描述定义的 MFC 基类,且给 MFC 应用程序提供绘图能力。

- CDC
- CPaintDC
- CClientDC
- CWindowDC
- CMetaFileDC

5.1.1　基类 CDC

CDC 是其他 DC 类的基类,CDC 基类定义设备描述表的对象,并且提供在显示器、打印机或 Windows 用户区上绘图的方法。利用 CDC 提供的类方法可再现全部图形输出。这些类方法提供下列服务:使用绘图工具,管理设备描述表,选择安全类型(type-safe)GDI 对象,控制着色和调色板,处理坐标映像和转换,加工多边形和区,绘制形状、绘制正文、使用不同字体、处理元文件等等,正如你所见,CDC 类是一庞然大物,封装了使用设备描述表的 GDI 函数。

为创建新的 CDC 设备描述表对象必须调用构造函数 CDC::CDC(),接着是三个初始化方法之一。三种方法如下:

- CreateDC()为指定设备,如打印机,创建设备描述表。
- CreateIC()为指定设备创建信息描述表以获取使用设备信息,而无需创建设备描述表。
- CreateCompatibleDC()创建新存储设备描述表,它与现有设备描述表兼容。通常用它准备在内存中的图形图像。

CDC::CreateDC()方法为指明的设备创建设备描述表。假如 DC 已创建,则 CreateDC()方法返回 True 值,否则返回 False。其原型如下所示:

```
Virtual BOOL CreateDC(LPCTSTR LPSZDriverName, LPCTSTR LPSZDeviceName, LPCTSTR LP-
SzOutput,
Const Void * lpInitData);
```

CDC::CreateIC()方法是为指定设备创建信息描述表(IC),它与设备描述表非常相似。假若 IC 已创建,则 CreatIC()方法返回 TRUE,否则返回 FALSE。原型如下:

```
Virtual BOOL CreatIC(LPCTSTR LpszDriverName, LPCTSTR LpszDeviceName,
LPCTSTR LPszOutput,
Const Void lpinitData);
```

上面两个方法取同样的四个参数,说明如下:

- lpszDriverName 指定原扩展的设备驱动程序文件名。
- lpszDeviceName 说明被支持的指定设备,一般为打印机,若驱动程序包含支持多于一个设备,则用此参数。
- lpszOutput 说明物理文件或输出端的文件或设备名。
- lpinitData 该结构包含设备驱动程序用的指定设备的初始化数据。

警告:在任一时刻,Windows 中仅有五个显示设备描述表可用,所有应用程序必须共享这些设备描述表。

最后的构造方法 CDC::CreateCompatibleDC()取一个参数,指向现有设备描述表指针,原型如下:

Virtual BOOL CreateCompatibleDC(CDC PDC);

此方法创建内部 DC,它与给定作为 DDC 参数的 DC 兼容。若此参数为 NULL,最终的内部 DC 与整个视频显示兼容。

5.1.2 CPaintDC 类

若我们需要重新编写新的 CPaint 函数,就需要使用 CPaintDC 类。尽管缺省的 OnPaint 函数会使用已经设置好的设备环境来调用 OnDraw,但有时我们还是需要编写操作都是针对 OnPaint 函数。我们一旦获得了一个 CDC 指针,就可将它当成任何设备环境指针来用。

在下面的例子中,OnPaint 函数创建了一个 CPaintDC 对象:

```
Void CMyView::OnPaint()
{CPaintDC dc(this);
OnPreporeDC(&dc);//expained afer
dc.Textout(0,0,"for the display,not the printer");
OnDraw(&dc);//stuff that's common to the display and printer
```

注意:CPaintDC 类的构造函数会自动调用 BeginPaint,而它的构造函数则会自动调用 EndPaint,因此,如果设备环境是在类中创建的话,那么 EndPaint 的调用完全是自动的。

5.1.3 CClientDC 和 CWindowDC

通常窗口的用户区域并不包括边框,标题条和菜单条。如果创建了 CClientDC 对象,该设

备环境的映射区域也仅限于用户区域，用户不可能在用户区域之外绘图。点(0,0)通常指的是用户区域的左上角。在以后的章节中，我们将会讲到，MFC 库中的 CView 对象总是和单个单独的主框架窗口的"子窗口"相关联，通常该子窗口还具有工具条、状态条和滚动条，作为视窗的用户区域，它当然并不包括主窗口用户区域的其它部分。如果窗口含有工具条的话，那么点(0,0)指的就是工具条下边界最左边的点。如果创建的是 WindowDC 对象，那么点(0,0)指的就是整个屏幕的左上角，这就允许用户在显示器的任何地方绘图，虽然这种机制工作很有用，但同时又很危险。通过调用 Cwnd 的成员函数 GetWindowRect，我们可以得到屏幕坐标系下的整个窗口的外边框矩形。

5.1.4 CMetaFileDC 类和元文件

Windows 应用程序使用两种图形工具存储图片：位图和元文件。一个元文件包含在设备描述表上创建图像或正文的 GDI 命令序列。在任何时刻都可将此图像从元文件放回到 DC，它的输出比位图要慢，但与设备无关。Windows 定义两种类型的元文件。一种是 Windows 元文件，它是为 Windows3.x 而设计的。另一种是增强型元文件，它是供 Windows 98 和 WindowsNT 使用的 Win32 元文件格式。MFC 库通过类 CMetaFileDC 提供对元文件的支持，为创建 Windows 元文件，必须首先在 CMetaFileDC 类基础上创建对象。通过调用 CMetaFileDC::Create()方法初始化新的 CMetaFileDC 对象，方法原型如下：

Bool Create(LPCTSTR lpszFilename = NULL);

若有文件名，此 lpszFilename 参数指向以 NULL 结尾的字符串，指明将创建的元文件文件名。若不提供文件名，则这方法创建新的 in-memory 元文件。CMetaFileDC::Create()方法创建实际 Windows 元文件 DC，并将它连接到 CMetaFileDC 对象上。为创建此元文件数据，发送系列 CDC GDI 命令到 CMetaFileDC 对象。当元文件绘图完成时，通过调用 CMetaFileDC::Close()方法关闭元文件的设备描述表，CMetaFileDC::Close()方法返回元文件句柄。通过调用以元文件句柄作为参数的 DC::PlayMetaFile()方法，可回放元文件。还可通过调用 Win32API 函数 CopyMetaFile()，将元文件存储为磁盘文件。当结束元文件时，用 Win32API 函数 DeleteMetaFile()从内存删除它。

注意：仅创建输出的 CDC GDI 命令，如 CDC::Ellapse()，CDC::MoveTo()和 CDC::Lineto()可用于写数据到元文件。

5.2 GDI 对象

Windows 的 GDI 对象类型是通过 MFC 库中的类来表示的。而 CGdiobject 是所有 GDI 对象类的抽象基类，即 Windows 的 GDI 对象是通过 CGdiobject 派生类的 C++对象来表示的。

下面给出 GDI 派生类列表。

- CBitmap——位图类，可以利用位图来表示图像，其中包括标像和光标，用来创建刷子。
- CBrush——刷子定义了一种位图形式的像素模板，利用它对区域内部的颜色填充。
- CFont——字体是一种具有某种风格和尺寸的所有字符的完整集合，它常常被当作资源存于磁盘中，其中有一些还依赖于某种设备。
- CPalette——调色板是一种颜色映射接口，它允许应用在不干扰其它应用的前提下可

以充分利用输出设备的颜色描绘能力。

- CPen——笔是一种用来画线及绘制有形边框的工具，我们可以指定它的颜色及厚度，并且可以指定它画实线、点线及虚线。
- CRgn——它是由多边形区域和椭圆区域组合而成的一种区域，可以利用它来进行增充、裁剪以及鼠标选中测试。

5.2.1 GDI 对象的构造和有效期

为使用 MFC 程序中图形对象，通常遵循下列三个步骤。

1)定义代码块内图形对象，用相应的 Create()方法初始化对象。例如，使用 CreatePalette()去创建 CPalette 对象。

2)一般通过使用 CDC::SelectObject()方法，选择新对象进入当前设备描述表。此方法返回指向被取代对象的指针(一个指向 CGdiobject 派生类的指针)。

3)当图形对象结束它的任务时，先前选中的图形对象返回设备描述表，保留事物的原来状态。

注意：当对象超出范围时，在设备环境上声明的对象自动删除。恢复 DC 的先前状态后，声明作为指针并与新操作符定位在一起的图形对象必须显式删除。对于显示设备环境来说，在每一个消息控制函数的入口处，设备环境都是未被初始化的，当函数退出之后，在该函数内部所进行的任何 GDI 选择都不再有效，因此，每次都必须从头开始设置设备环境。尽管 CView 类的虚成员函数 CPrepareDC 对于设置映射方式来说是非常有用的，但我们仍然需要小心谨慎地管理自己的 GDI 对象。对于其它设备环境来说，如打印机和内存缓冲区，用户的一些设置可以持续得长久一些。对于这些有效期长的设备环境来说，事情就变得有些复杂，这种复杂性是由 SelectObject 函数所返回的 GDI 的C++ 对象指针的临时性所导致。故我们不能简单地将这一指针保存在类的数据成员中，而应该借助于 GetSafeHdc 成员函数，将其转化为 Windows 句柄(唯一能够持久存在的 GDI 标识)。

5.2.2 选择库存 GDI 对象

Windows 提供了 2 种影响 CDC 绘图函数工作方式的工具：笔(pen)和刷(brush)。当前影响画线的方式，包括直线、曲线(如用 lineto 函数绘制的以及封闭图形周围的边框，如矩形和椭圆)。封闭图形包括二个分开的元素：边框和内部。当前刷影响封闭图形内部的绘制方式。首次生成设备环境时，它有缺省笔刷。缺省绘制宽为 1 个图素的实黑点(不管当前映射方式如何)，缺省刷将封闭图形内部增加统一的白色。为了改变当前的笔和刷，可选择笔和刷，然后选入设备环境对象。一旦选择笔或刷，自此以后的图形一直要明确选择不同的笔和刷。

为了选择惯用绘图工具，只有调用 CDC 成员函数 SelectStockobject:CGdiobject :: SelectStockObject(int nindex)；参数 nindex 是要选入设备情境对象的特定惯用对象的码。下表列出了选择了惯用笔和刷的 nindex 的值。

表 5.1 常用笔和刷的 nindex 值

数值	惯用对象
BLACK_BRUSH	黑刷
DKGRAY_BRUSH	深灰刷
GRAY_BRUSH	中灰刷
LTGRAY_BRUSH	淡灰刷
NULL_BRUSH	空刷(内部不填充)
NULL_PEN	空笔(不画笔或边框)
WHITE_PEN	白笔
WHITE_BRUSH	白刷(缺省刷)
BLACK_PEN	黑笔(缺省笔)

例如,下列码选择白笔和中灰刷:

```
Void Cview::Ondraw(CDC * PDC)
·PDC →SelectStockObject(WHITE_PEN);
PDC→SelectStockObject(GRAY_BRUSH);
//Call other graphing. funltions and drawgraphies
//CCines and borders will be white, interiors of closed
// figures will be medium-greay
```

说明:不管当前映射方式如何,画实线的惯用笔总是 1 图素宽的。

惯用刷画统一颜色(而不是图案)。

5.2.3 生成用户化 GDI 对象

按下列步骤可以生成用户化笔和刷。

1)对笔生成 Cpen 类的实例,对刷生成 CBrush 类实例。

2)调用相应的 CPen 类或 CBrush 类成员函数初始化笔和刷。

3)选择设备环境对象的笔和刷对象,存放原笔或刷对象的指针。

4)调用绘图函数生成图形输出。

5)选择设备环境对象的原有笔和刷。

为了生成临时笔或刷,可以在生成图形输出的函数内声明 CPen 或 CBrush 类的实例为本地对象。但若要在整个程序中重复使用某个笔或刷,将对象声明为视图类数据成员会更加有效。为了将笔初始化,可调用 Cpen 成员函数 Createpen,如下所示:

BOOL Createpen (int nPenstyle, int nwidth, colorREF CrColor);

nPenstyle 参数指定该笔要画的线类型。赋值 PS_NULL 样式使笔在样式 PS_DASH, PS_DDT, PS_DASHDOT 或 PS_DASHDOTDO 下只能用于笔宽为 1 时(笔宽大于 1 时这些样式产生实线)。

nwidth 参数指定笔所画出线的宽。如果线宽值为 0,则当前映象方式无论如何线宽总是 1 个图素(等于缺省和惯用笔生成的线宽)。Crcolor 参数指定颜色,最简单的方法是用

win32RGB 宏功能指定颜色，bRed，bGreen，bBlue 参数表示红绿蓝三色的相对密度，可根据三种颜色密度不同利用 RGB 宏功能指定其为主 VGA 图形方式中可用的 16 种纯颜色。

注意：笔只能指定纯颜色。纯颜色是指视频硬件生成的颜色，不必混合不同颜色的图素来模拟(称为配色)。如果赋值的颜色值不对应纯颜色，则改用最接近的纯颜色绘制。下列情况为规则的例外：如果笔是 PS_INSIDEFRAME 样式，笔宽大于 1，则 Windows 在所赋的颜色值不对应于纯颜色时采用配色。可以调用 CBrush 成员函数 CreateSolidBrush 初始化刷，使它用单个一致颜色增充图形内部。

BOOl. CreateSolidBrush(COLORREF crcolor)；其中参数 Crcolor 指定增充颜色。可以指定为任何颜色值，如果赋予的值不对应纯颜色，则 Windows 产生配色。也可以调用 CBRUSH 成员函数 CreateHatchBrush 初始化刷，用格子图案增充图形内部。

BOOL CreateHatchBrush(int nindex，colorREF crColor)；参数 nindex 指定所要图案，crcolor 参数指定格子线的颜色。最后也可调用 CBrush 成员函数 CreatePatternBrush 初始化刷子，填充有用户化图案的图形。

BOOL CreatePatternBrush(CBitmap * PBitmap)；PBitmep 参数是位图对象的指针。用这个刷子画图时，内部完全增上位图的备份，边对边铺成。位图大小应为 8×8 图素(如果位图是单色的，则 Windows 用当前文本和文本背景颜色绘制)。

说明：和对象一样，可以在生成时输入适当参数到 CBRUSH 构造器初始化刷对象。关于此方法，读者可参阅其他参考书，在此不多叙述。

一旦初始化笔或刷，要用 CDC 成员函数 SelectObject 将它先放进设备环境对象。为了选择笔，调用 SelectObject 下列版本：

Cpen * SelectObject(CPen * pPen)；

其中 pPen 是笔对象的指针。SelectObject 返回上次先进入设备环境对象的笔对象的指针(如果无上次选择的笔，则这时缺省笔是临时对象)。为了选择刷，可调用 Selectobject 下列版本：

CBrush * SelectObjectC(Brush * pBrush)；

其中 pBrush 是刷对象的指针。Selectobject 返回上次选择刷的指针(如果没有上次选择刷，则这时缺省刷是临时对象)。调用 SelectObject 选择笔或刷时，应有返回的指针。然后，完成调用图形函数显示用笔和刷的输出后，应当再次调用 SelectObject，将上次对象选回设备环境对象以删去设备环境中的笔和刷。

提示：应当删去设备环境对象中的笔和刷，以使笔和刷销毁后，设备环境对象中不会留下一个无效的句柄。初始化笔和刷时，Windows 提供同笔或刷对象一起有效的句柄，设备环境对象也存放这个句柄。笔和刷对象离开范围时，或因其他原因销毁时，对象销毁器销毁这个句柄。但这一步在设备环境对象比笔或刷对象更早被销毁时就不需要了。

5.3 颜色映射

Windows的GDI提供了一种"硬件独立"的颜色接口,应用程序只需提供"绝对的"颜色代码,GDI就会将该代码以合适的颜色或颜色组合映射到用户计算机的视屏显示器上,大多数Windows程序员都会针对一些常见的显示卡,尽量将自己所编写的应用的颜色显示进行优化。

5.3.1 16色和256色显卡

标准VGA显示卡使用的是18位颜色寄存器,因此它可表示出262,244种颜色。然而,由于视频内存的限制,标准VGA卡只好采用4位颜色代码,也就是说,它一次只能同时显示16种颜色,由于Windows对标题条、边框、滚动条等都采用固定的颜色,所以它们所采用的颜色也只好从16种标准纯色中进行选择了,要想得到显示卡能够显示的颜色就不那么容易了。每一种Windows颜色都是通过8位"红","绿","蓝"值组合起来来表示的。面向颜色的GDI函数可以接收32位COLOREF参数。这种类型的参数分别包含了8位红、绿和蓝的组合值。Windows的RGB宏可将8位红、绿、蓝值转化成COLOREF参数,下面语句如果在标准VGA卡的系统上运行时,就会创造一个具有模糊色的刷子:CBrush brush(RGB(128,128,192));

下列语句(见视图的OnDraw函数)将文本的背景置成亮红色。

CDC→SetBkColor(RGB(255,0,0));

和面向刷子的绘图函数不同,CDC的SetBkColor和SetTextColor函数并不是总是显示模糊色,若模糊色模板过度复杂的话,它们就会选择与其相近的纯色来显示。许多显示卡可以支持8位颜色码,即它们可以同时显示256种颜色,SVGA卡就是其中之一。如果用户手中拥有SVGA卡,那么为了激活256色方式,必须安装一个专门的Windows显示驱动程序,该驱动程序可由Microsoft提供,也可由显示卡的生产厂家提供。由于8位颜色需要的内存是4位的4倍,故在256方式下,Windows的显示更新速度明显要慢一些。若Windows配置的是256色显示卡,那么用户程序只限用20种标准色,除非激活Windows"颜色调色板"系统,该系统可由MFC库中的CPalette类或Windows的API支持。

若安装了256色显示驱动程序,那么除了前面我们所提的16种VGA标准色外,还可以多增加4种标准色,即一共有20种标准色。像标准VGA一样,此时我们仍可以使用RGB宏,如果用20种标准色来创建刷子,那么得到的是纯色刷子,否则的话,得到的将是模糊色(dithered color)刷子;如果使用的是PALETTERGB宏,那么得到的不是模糊色,而是和它最接近的标准纯色。

5.3.2 调色板

当Windows在标准VGA显卡上显示单色或16色DIB(或将它转换成GDI位图)时,它使用的是与指定颜色最接近的标准纯色,我们无法控制这一处理过程,对标准VGA显卡来说,调色板是不起作用的。然而,对于256色显示卡,我们就可以具有更多的灵活性。如果不使用颜色调色板,Windows将把颜色映射到驱动程序的20种标准颜色;若使用了颜色调色板,Windows将把DIB颜色尽可能准确地映射到显卡所支持的颜色,其中有20种调色板颜色总是被用作标准色,而其余256种调色板颜色可以被改变以便表示出具体的位图。

一旦调色板准备好,应用程序可请求显示颜色。若应用程序使用调色板,则此应用程序只

限于使用 20 种静态系统颜色。逻辑调色板允许应用程序选择显露于显示器上的颜色 Windows 中逻辑调色板由 LOGPALETTE 结构定义,此结构如下所示:

```
typedef struct tagLOGPALETTE
{Word palversion;
Word palNumEntries;
PALETTEENTRY PalPalEntry[1];}LOGPALETTE;
```

此结构数据成员如下:

- palversion 成员指定此结构的 Windows 版本号,它应是 OX300 值。
- palNumEntries 成员指定逻辑调色板中颜色项数目。
- palpalEntry 成员指定 PALETTEENTRY 结构的数目,它包含颜色值,并指定逻辑调色板中每个表项的使用标志。

提示:应将应用程序需要的最重要的颜色放在调色板低项索引上,这样安排使需要它们时有更好的机会放入系统调色板中。此结构最重要的部分是 palPalEntry 成员,它使用嵌入 PALETTEENTRY 结构定义在 LOGPALETTE 的 palPalEntry 颜色数组的单色项。

此 PALETTEENTRY 结构如下所示:

```
typedef struct tagPALETTEENTRY{BYTE peRed;
BYTE peGReen;BYTE PeBlue;BYTE peFlags;
} PALETTEENTRY;
```

此语法中,peRed、peGreen、peBlue 成员指定调色板表项的 RGB 颜色亮度值。peFlags 成员指定如何使用调色板面。

5.4 字体和正文

字体(Font)是一套具有统一大小和设计风格,用于表示字符数据的复杂图形图像。字体通常是用字体磅值和字体名表述,如"Arial"或"宋体",有时还带有诸如:加粗、倾斜等风格。一般地,我们用诸如 12 磅的 Arial 或三号加粗的宋体来指定字体。中文字号与西文磅值之间的转换关系请参看其他有关论述。

5.4.1 获取文字属性

为输出正文,Windows 使用边界矩形,并为显示像素,如位图,绘制当前字符的表示法。使用 Windows 的 TEXTMETRIC 的结构以获取有关显示器上正文字符信息的有 Win32API 函数的 MFC 方法。此 TEXTMETRIC 结构包含显示字体信息和给出以逻辑单位表示的尺寸信息,该逻辑单位与当前映像方式有关。

为取得当前选入设备描述表的有关字体的基本信息,可使用 CDC::GETTextMetries()方式,此方法原型如下:

```
BOOL GetTextMetries(LPTEXTMETRIC LPMetries);
```

在此语法中,lpMetries 是指向 TextMetries 结构的指针,此结构接受正文度量。例如,为了在给定设备描述表中找到当前字体的平均宽度,一般调用 GETTextMetries(),如以下代码

所示：

```
TEXTMETRIC tm;
lfcdc.GETTextMetric(&tm))
int nAueCharWidth = tM + mAumCharwidth
```

5.4.2 创建字体

创建字体必须向 GDI 发送字体请求，字体请求的表示方式是 LOGOUT(逻辑字体)数据结构向 GDI 发送字体请求，也就是增写该数据结构，并将其传递给 CFont 初始化函数 CFont::CreateFontIndirect()，该函数名中的 indirect 表明该函数需要的是一个指针参数。另一个初始化函数是 CFont::CreateFont()，它需一系列参数。在此不赘述。

创建非库存字体、第一步是增写 CFont 请求，在 CMainForm 的构造函数中，该数据结构用如下方式填写。

```
//初始化所期望的字体
memset(&lfCurrent,0,sizeof(LOGFONT));
//初始化字体名为 FontNew
Lstrcpy Cd-Ctcurrent,lfFaceName,"FontNew";//初始化字体大小为 12 磅
//获取 一个显示库 DC
CWindowDC dc(NULL);
int Cypixels = dc.GetDeviceCaps(LOGPIXELSY);
lfcurrent.lfrRight = MulDiv(12,CyPixels,72);
```

负的字体高度值请求一个没有内部引导高度的真正的磅值。一般地，应尽量避免在构造函数中调用可能会失败的函数，毕竟 C++ 没有简单的方法来报告这类问题。在字体初始化中可能会失败的步骤是 CFont 对象的分配，下面的语句是表示如何使用定义的逻辑字体结构，分配 CFont 对象并将字体请求传递给 GDI。

```
//创建新字体
dpFont = New CFont();
dpFont→CreateFontIndirect(&lfcurrent);
```

注意：上述语句中，GDI 除了复制一个请求以外没有做更多的事情，这是因为此时的字体请求还没有与任何指定的 GDI 设备联系起来，而这种联系在 GDI 准备考虑我们的请求之前将会建立起来。一个字体对象一旦被创建，便可通过调用 CDC::SelectObject()函数将其与 GDI 设备—显示屏幕，打印机，图元文件或位图联系起来。与库存字体一样，一旦一个用户字体对象被连入了 DC，那么它将影响所有的文字输出，直至另一个新字体被选中为止。

```
//获得"this"窗口的设备文本(DC)
CPaintDC dc(this);
//将当前字体选入 DC
pdc->SelectObject(&pFont);
```

5.4.3 绘制正文

使用设备描述表选中的字体、正文颜色和背景颜色绘制正文，调用 CDC::SetTextColor()

方法可设置正文颜色;调用 CDC::SetBkcolor()和 CDC::SetBkMode()方法可设置背景颜色,为把 DC 中正文颜色改变为蓝色,并使正文背景透明,可使用下列代码(假定 PDC 是指向 CDC 派生对象的指针):

```
pDC→SetTextColor(RGB(0,0,255);
pDC→SetBkMode(TRANSPARENT);
```

MFC 提供类 CDC 中的几种正文输出方法,其中最常用的是 DrawText(),TextOut()和 TabbedTextOut()。DrawText()方法在给定矩形内格式化正文,通过使用许多高级格式化特性做到这点,包括扩展制表符到空白空间,调整正文至矩形左、右或中央,自动将文本分成适合矩形内部的行,其原形为:int DrawText(const CString & str, LPRECT lpRect, UINT nFormat);参数 strr 是包含要绘制正文的 CString 对象,lpRect 参数指向 RECF 或 CRect 对象,该对象形成把正文格式化的边界框,nFormat 参数指定如何格式化正文。

TextOut()方法使用当前选中字体和颜色在指定位置上绘制正文字符串,其原型为:

```
BOOL TextOut(int x,int y,const CString & str);
```

参数 x 与 y 指定正文开始绘制的逻辑坐标,str 是包含将绘制正文的 CString 对象。

Tabbed TextOut()方法在指定位置上绘制正文字符串,并将制表符扩展到制表停止位置数组中的指定值,其原型为:CSize TabbedTextOut(int x,int y,//字符串起始点的逻辑坐标

const CString &str,//包含指定字符的 CString 对象

int nTabPositions,//在制表停止位置数组中的数值

LPINT lpnTabStopPositions,//指向制表停止位数值的指针

int nTabOrigin;//指定制表扩展起始位置的 x 坐标。

5.4.4 实例 Format

该实例中演示了如何显示不同种风格类型的字体。对话框最上面三个 Gray Box 分别对字体的粗、斜、下划线以及字体之间如何对齐与间距加以控制。"行距"之后的编辑框是确定显示在右边 Group Box 内字符行与行之间的距离,只能填入一位数,且范围是(1~4)。其下是"自选字体"按钮,点击之后,会弹出的字体对话框,该对话框可以更全面地选择字体的属性。如图 5.1 所示。

下面具体介绍创建该工程的步骤。

1)创建基于对话框的工程 format。

2)编辑对话框资源 IDD_FORMAT_DIALOG,在对话框上加 4 个组合框 Group Box,分别是"字体"、"对齐"、"间距"和"浏览"。在"字体"组合框内加三个 check Box,分别是"黑体"(IDC_BOLD)、"斜体"(IDC_ITALIC)和"下划线"(IDC_UNDERLINE)和"右对齐"(IDC_RIGHT)。在"对齐"组合框内加两个 Radio Button,分别是"左对齐"(IDC_LEFT)和"右对齐"(IDC_RIGHT)、"中对齐"(IDC_CENTER)。在"间距"组合框内加两个 Radio Button,是"可变"(IDC_VARIABLE)和"固定"(IDC_FIXED)。然后在对话框左下角加"行距"静态控件,编辑框控件(IDC_SPACING),按钮控件(IDC_FONT)。

3)具体如何实现可参见文件 formatDlg.h 和 formatDlg.cpp,代码如下所示:

```
// formatDlg.h : header file
```

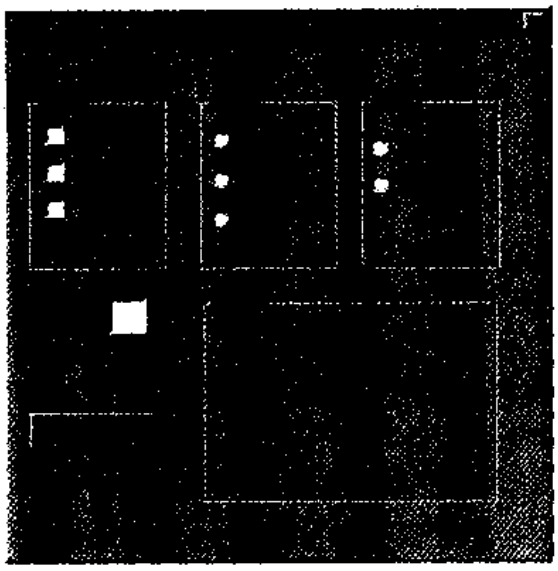


图 5.1 字体属性对话框

```
//

#if ! defined(AFX_FORMATDLG_H__A8600A68_1837_11D2_BA07_444553540000__INCLUDED_)
#define AFX_FORMATDLG_H__A8600A68_1837_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CFormatDlg dialog

enum{JUSTIFY_LEFT,JUSTIFY_CENTER,JUSTIFY_RIGHT};
enum{PITCH_VARIABLE,PITCH_FIXED};
class CFormatDlg : public CDialog
{
// Construction
public:
	CFormatDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CFormatDlg)
	enum { IDD = IDD_FORMAT_DIALOG };
	CEdit m_SpacingEdit;
	BOOL   m_Bold;
	BOOL   m_Italic;
```

```
        int          m_Justify;
        BOOL   m_Underline;
        int          m_Pitch;
        int          m_Spacing;
        //{{AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CFormatDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);        // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        HICON m_hIcon;
        RECT m_RectSample;
        CFont Font;

        // Generated message map functions
        //{{AFX_MSG(CFormatDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnBold();
        afx_msg void OnCenter();
        afx_msg void OnFixed();
        afx_msg void OnItalic();
        afx_msg void OnLeft();
        afx_msg void OnRight();
        afx_msg void OnUnderline();
        afx_msg void OnVariable();
        afx_msg void OnChangeSpacing();
        afx_msg void OnFont();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_FORMATDLG_H__A8600A68_1837_11D2_BA07_444553540000__IN-
CLUDED_)
```

```
// formatDlg.cpp : implementation file
//

#include "stdafx.h"
#include "format.h"
#include "formatDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
```

```
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFormatDlg dialog

CFormatDlg::CFormatDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CFormatDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CFormatDlg)
    m_Bold = FALSE;
    m_Italic = FALSE;
    m_Justify = -1;
    m_Underline = FALSE;
    m_Pitch = -1;
    m_Spacing = 1;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CFormatDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFormatDlg)
    DDX_Control(pDX, IDC_SPACING, m_SpacingEdit);
    DDX_Check(pDX, IDC_BOLD, m_Bold);
    DDX_Check(pDX, IDC_ITALIC, m_Italic);
    DDX_Radio(pDX, IDC_LEFT, m_Justify);
    DDX_Check(pDX, IDC_UNDERLINE, m_Underline);
    DDX_Radio(pDX, IDC_VARIABLE, m_Pitch);
    DDX_Text(pDX, IDC_SPACING, m_Spacing);
```

```
    //{{AFX_DATA_MAP

BEGIN_MESSAGE_MAP(CFormatDlg, CDialog)
    //{{AFX_MSG_MAP(CFormatDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BOLD, OnBold)
    ON_BN_CLICKED(IDC_CENTER, OnCenter)
    ON_BN_CLICKED(IDC_FIXED, OnFixed)
    ON_BN_CLICKED(IDC_ITALIC, OnItalic)
    ON_BN_CLICKED(IDC_LEFT, OnLeft)
    ON_BN_CLICKED(IDC_RIGHT, OnRight)
    ON_BN_CLICKED(IDC_UNDERLINE, OnUnderline)
    ON_BN_CLICKED(IDC_VARIABLE, OnVariable)
    ON_EN_CHANGE(IDC_SPACING, OnChangeSpacing)
    ON_BN_CLICKED(IDC_FONT, OnFont)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFormatDlg message handlers

BOOL CFormatDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
```

```
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon


    GetDlgItem(IDC_SAMPLE)->GetWindowRect(&m_RectSample);
    ScreenToClient(&m_RectSample);
    m_SpacingEdit.LimitText(1);

    return TRUE;  // return TRUE  unless you set the focus to a control

}

void CFormatDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CFormatDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
```

```
        GetClientRect( &rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CPaintDC dc(this);

        HFONT hFont = (HFONT)Font.GetSafeHandle();
        if(hFont! =NULL)
            Font.Detach();

        LOGFONT lf;
        int LineHeight;
        CFont * pOldFont;
        int x,y;

        CFont TempFont;
        if(m_Pitch= =PITCH_VARIABLE)
            TempFont.CreateStockObject(SYSTEM_FONT);
        else
            TempFont.CreateStockObject(SYSTEM_FIXED_FONT);
        TempFont.GetObject(sizeof(LOGFONT),&lf);

        if(m_Bold)
            lf.lfWeight = FW_BOLD;
        if(m_Italic)
            lf.lfItalic = 1;
        if(m_Underline)
            lf.lfUnderline=1;

        Font.CreateFontIndirect( &lf);
        pOldFont = dc.SelectObject(&Font);

        switch(m_Justify)
        {
        case JUSTIFY_LEFT:
            dc.SetTextAlign(TA_LEFT);
            x=m_RectSample.left+5;
            break;
        case JUSTIFY_CENTER:
```

```
                dc.SetTextAlign(TA_CENTER);
                x=(m_RectSample.left+m_RectSample.right)/2;
                break;
            case JUSTIFY_RIGHT:
                dc.SetTextAlign(TA_RIGHT);
                x=m_RectSample.right-5;
                break;
            default:
                dc.SetTextAlign(TA_LEFT);
                x=m_RectSample.left+5;
                break;

            }

            dc.SetBkMode(TRANSPARENT);
            LineHeight=lf.lfHeight*m_Spacing;
            y=m_RectSample.top+15;
            dc.TextOut(x,y,"AaBbCcDdEeFf");
            y+=LineHeight;
            dc.TextOut(x,y,"GgHhIiJjKkLl");
            y+=LineHeight;
            dc.TextOut(x,y,"MmNnOoPpQqRr");

            dc.SelectObject(pOldFont);


        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CFormatDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CFormatDlg::OnBold()
{

    m_Bold=!m_Bold;
    InvalidateRect(&m_RectSample);
    UpdateWindow();
}
```

```
void CFormatDlg::OnCenter()
{

    if(IsDlgButtonChecked(IDC_CENTER))
    {
        m_Justify=JUSTIFY_CENTER;
        InvalidateRect(&m_RectSample);
        UpdateWindow();
    }
}

void CFormatDlg::OnFixed()
{
    if(IsDlgButtonChecked(IDC_FIXED))
    {
        m_Pitch=PITCH_FIXED;
        InvalidateRect(&m_RectSample);
        UpdateWindow();
    }
}

void CFormatDlg::OnItalic()
{
    m_Italic=!m_Italic;
    InvalidateRect(&m_RectSample);
    UpdateWindow();

}

void CFormatDlg::OnLeft()
{
    if(IsDlgButtonChecked(IDC_LEFT))
    {
        m_Justify=JUSTIFY_LEFT;
        InvalidateRect(&m_RectSample);
        UpdateWindow();
    }
}

void CFormatDlg::OnRight()
{
    if(IsDlgButtonChecked(IDC_RIGHT))
    {
        m_Justify=JUSTIFY_RIGHT;
```

```
        InvalidateRect(&m_RectSample);
        UpdateWindow();
    }
}

void CFormatDlg::OnUnderline()
{
    m_Underline = ! m_Underline;
    InvalidateRect(&m_RectSample);
    UpdateWindow();

}

void CFormatDlg::OnVariable()
{
    if(IsDlgButtonChecked(IDC_VARIABLE))
    {
        m_Pitch = PITCH_VARIABLE;
        InvalidateRect(&m_RectSample);
        UpdateWindow();
    }
}

void CFormatDlg::OnChangeSpacing()
{

    int Temp = (int)GetDlgItemInt(IDC_SPACING);
    if(Temp > 0 && Temp < 4)
    {
        m_Spacing = Temp;
        InvalidateRect(&m_RectSample);
        UpdateWindow();
    }
}

void CFormatDlg::OnFont()
{

    CFontDialog dlg(NULL,CF_EFFECTS | CF_SCREENFONTS,NULL,this);
    if(IDOK == dlg.DoModal())
    {
        m_Bold = dlg.IsBold();
        m_Italic = dlg.IsItalic();
        m_Underline = dlg.IsUnderline();
```

```
    UpdateData(false);
    InvalidateRect(&m_RectSample);
    UpdateWindow();
}
```

5.5 小　　结

文本和字体在编程中几乎都要用到。本章向读者介绍了关于文本和字体的大部分内容，诸如，设备环境类，GDI 对象及文本和字体在编程中的具体应用。最后通过实例 Format，具体向读者展示了文本和字体在 VC++6.0 中编程中的用法。相信经过本章学习，读者对这部分内容会有一个全新认识。

第6章 图 形

Windows操作系统凭着图形化操作界面战胜不太友好的字符式操作系统，而被用户所喜欢，可见图形在Windows世界的地位和作用。

位图是形成某类图像的位的数组，视频显示器利用像素在屏幕上表示位图的位，位图以各种位深度（或颜色深度）和样式出现。图像的颜色深度与在位图图像中用以存储颜色数据的位数直接有关。Windows使用两类位图：设备相关位图（GDI）和设备无关位图（DIB），这两种类型位图可有不同的颜色深度。

6.1 GDI位图和DIB位图

GDI位图和DIB位图是两种不同的Windows位图，GDI位图是由MFC的CBitmap类表示的，在CBitmap类的对象中，包含了一种和Windows的GDI模块有关的Windows数据结构BITMAP，该结构是与设备有关的，应用程序可以得到GDI位图数据的一个备份，但其中位的安排完全依赖于显示设备。在同一台机器中，我们可以将GDI位图在不同的应用程序中任意地传送，但由于GDI位图对设备的依赖性，因此通过磁盘或调制解调器对它们进行传送没有太大的意义。

GDI位图是由保存位图数据的BITMAP结构定义的，BITMAP定义如下：

```
typedef struct tagBITMAP
{LONG bmType;//位图类型。
LONG bmWidth;//以像素表示的位图宽度。
LONG bmHeight;//以像素表示的位图高度。
LONG bmWidthBytes;//每条扫描线的字节数。
WORD bmPlanes;//在位图中的颜色面数。
WORD bmBitsPixel;//描述像素颜色所需位数。
LPVOID bmBits;//指向构成图像数据的字符值数组的指针。
}BITMAP;
```

DIB(Device-Independent Bitmap)位图是另外一种格式的位图，它完全解决了位图传送问题，任何运行Windows的机器都可以对DIB进行处理，它通常以后缀为BMP的文件形式保留在磁盘中，当BMP文件从磁盘文件中读出来以后，它通常被转成GDI位图，但必要的时候程序可直接利用DIB格式进行工作。

DIB位图区别于GDI位图的本质特性为：

(1)包含创建DIB图像设备上的颜色格式信息。

(2)包含创建DIB图像设备上的分辨率信息。

(3)包含创建DIB图像设备上的调色板信息。

(4)包含用以将调色板的红、绿、蓝(RGB)颜色元素映像到DIB图像中的像素的位数组。

(5)包含数据压缩标志，它指示哪种数据压缩方案(假若有)用于压缩磁盘文件。

6.2 使用 GDI 位图

使用 GDI 位图之前必须创建它，然后把它选进设备环境中，当使用完了以后，还得将它从设备环境中取出来，把它删掉，这是使用 GDI 位图的过程，本节将详细介绍 GDI 位图的使用。

6.2.1 生成位图

MFC 提供了管理位图的 CBitmap 类，因此生成位图的第一步就是声明这个类的实例，如 CBitmap m_Bitmap;

位图对象通常声明为一个主程序类(如视图类)的数据成员。一旦声明 CBitmap 对象，必须调用适当的 CBitmap 成员函数将对象初始化。下面介绍如何初始化 CBitmap 对象，一是调用 LoadBitmap 成员函数装入程序资源中的位图数据，一是调用成员函数 CreateCompatibleBitmap 生成空位图，运行程序时再画所要的图形。

说明：除了上面两种方法外，Windows 还提供了另外几个初始化位图对象的方法。例如，可以调用 LoadDEMBitmap 函数装入 Windows 提供的预定义位图，可调用 CreateBitmap 生成符合特定结构的位图，也可调用 Attach 用 Windows 位图句柄初始化位图对象，从资源装入位图。

将位图作为程序资源就可以利用交互式位图编辑程序来设计位图，如 Visual C++ 图形编辑器和 Windows 98 Paint 等其他绘图程序，因此这种方法特别适用于生成相对复杂的或非几何图形，它们很难用绘图函数生成。

为了用 Visual C++ 图形编辑器设计位图，选择 Insert 菜单的 Resource 命令或按 Ctrl + R 键，并在 Insert Resource 对话框中选择资源类型 Bitmap，然后用 Visual C++ 打开图形编辑器中的新位图窗口，之后可以设计所要的位图图形。

生成新位图时，Visual C++ 赋值缺省标识符，也可取别的标识符。方法是：双击位图窗口，在 Bitmap Properties 对话框中选择 General 标签，并在 ID 文本框中输入新的标识符。

提示：除设计新位图外，也可以从别的可以复制位图格式图形的程序中将图形复制到剪贴板，再选择 Edit 菜单的 Paste 命令，将其粘贴到 Visual C++ 图形编辑器中。也可以在别的绘图程序中设计位图，存成文件，可以使用任何能将文件存成位图格式的程序(文件扩展名为 .BMP 或 .DIB)。为了在程序中使用位图，选择 Insert 菜单的 Resource 命令，单击 Insert Resource 对话框 Import 键，然后选择其他程序中存盘的文件名。引入的位图出现在 Visual C++ 图形编辑器中，然后编辑，改变大小或赋予不同的标识符。

无论在 Visual C++ 图形编辑器中设计位图或是从位图文件引入，该位图都会加入程序资源。程序运行时，可以装入该位图并用它初始化位图对象。为此调用 CBitmap 成员函数 LoadBitmap，例如：

```
CBitmap m-Bitmap;
m-Bitmap.LoadBitmap(IDB-BITMAP1);
```

LoadBitmap 的参数是位图在图形编辑器中生成或从位图文件引入时赋予的标识符。

既然可以在编辑器中设计位图，那也可以在程序运行时初始化一个空位图并用 MFC 绘

图函数在位图中画出所要的图案。

步骤如下：

1)初始化空位图。

2)生成内存设备环境对象。

3)将位图选入内存设备环境对象。

4)在位图中对内存设备环境对象调用 CDC 绘图函数画出所要图形。

初始化空位图可调用 CreateCompatibleBitmap 函数，例如：

```
CBitmap m_Bitmap;
CClientDC dc(this);
m_Bitmap.CreateCompatibleBitmap(&dc,32,32);
```

函数 CreateCompatibleBitmap 的第一个参数是设备环境对象的地址，生成的位图将与这个对象相联系的设备兼容，第二、三个参数分别是位图的宽度和高度，单位是图素。

调用 CreateCompatibleBitmap 时，Windows 保留一块内存给位图。这个位图中存放的图素值最初是未定的，必须用绘图函数生成所要的图形。但在位图内画图前，要生成与位图相联系的设备环境对象，这就是内存设备环境对象。为了生成内存设备环境对象，先要声明 CDC 类的实例，再调用 CDC 的成员函数 CreateCompatibleDC，例如：

```
CClientDC dc(this);
CDC MemDC;
m_Bitmap.CreateCompatibleBitmap(&dc,32,32);
MemDC.CreatecompatibleDC(&dc);
```

函数 CreatecompatibleDC 的参数是设备环境对象的地址。接着调用 CDC 的成员函数 SelectObject将位图对象选入内存设备环境对象，例如：MemDC.SelectObject(&m_Bitmap)；现在可以对内存设备环境对象调用 CDC 绘图函数在位图内画所要的图形了。

6.2.2 显示和处理位图

生成所需的位图之后，把位图显示在窗口上或其他设备上才是最终目的。MFC 的 CDC 类提供了三个传送图形数据块的灵活而有效的函数 PatBlt，BitBlt 和 StretchBlt。生成图形时可用这些函数，复制图形数据块和以简单或复杂方法修改图形数据时(如传递颜色或翻折图形)时也用这些函数。这些函数可以用于将图形数据从同一设备显示表面的一个位置复制到另一位置，也可用于在不同设备间或设备与位图间复制图形数据。

PatBlt 可以通过 CDC 类的成员函数 PatBlt 用当前刷画矩形区。在位操作环境中，当前刷通常指当前图案。尽管也可通过调用 CDC::FillRect 用当前图案填充区域，但 PatBlt 函数更加灵活，其原形如下：BOOL PatBlt(int x,int y,int nWidth,int nHeight,DWORD dwRop)；前两个参数指定所画矩形区左上角的逻辑坐标，后两个参数按逻辑单位指定这个区的宽和高，最后一个参数是光栅操作码。光栅操作码提供了 PatBlt 的灵活性，它指定图案中的每个图素与目标地点的当前图素的组合方式，产生最后的目标图素颜色。下列函数 OnDraw 用当前图案画整个视图窗口，完全取代当前窗口内容，代码如下所示：

```
void CRetriveView::OnDraw(CDC * * PDC)
```

```
CRECT rect;
GetClientRect(*rect);
PDC->PatBlt(rect.left,rect.top,
rect.width(),rect.Height(),PATCOPY);
```

光栅操作码 PATCOPY 是将原位图的图素颜色直接 Copy 到目标位图。

BitBlt 函数可以把图形数据块从一个位置传送到另一位置，源位置和目标位置可以在同一设备(或位图)上，也可以在不同设备(或位图)上，其原形如下：

```
BOOL BitBlt(int x,int y,
int nWidth,int nHeight,
CDC *pSerDC,
int xScr,int yScr,
DWORD divRop);
```

前两个参数指定传送目标位置左上角的逻辑坐标，后两个参数指定传送的数据块逻辑尺寸。第五个参数 pSerDC 是源设备环境对象的指针，第六、七个参数是源设备中位图的左上角逻辑坐标，最后一个参数是光栅操作码。

BitBlt 将图形数据块从一个与第五个参数表示的环境对象相联系的设备复制到与调用函数的环境对象相联系的设备上。例如下列调用将图形数据块从位图(选入内存设备环境对象 MemDC)传送到视图窗口(与 *pDC 设备环境对象相联系)：

```
pDC->BitBlt(x,y,Width,Height,&MemDC,0,0,SRCCOPY);
```

BitBlt 画目标区时，每个图素的最后颜色取决于当前图素颜色，源设备中对应图素颜色和当前图案(即当前选入目标设备环境对象的刷)中对应图素的颜色。BitBlt 组合这些颜色的方式取决于赋予 divRop 参数的光栅操作码。由于传入 BitBlt 的光栅操作码影响来自三个不同图素的颜色组合方式，故可以赋值的码比 PatBlt 要多得多。

StretchBlt 是这三个位操作函数中最灵活的。它可用于完成 BitBlt 能完成的所有操作，还可在传送时改变数据块的大小和将块翻折(水平、垂直或双向)。其原形如下：

```
BOOL StretchBlt(int,x,int y,
int nWidth,int nHeight,
CDC *pSCCdc,
int xSrc,int Src,
int nSrcWidth,int nSrcHeight,
DWORD dwRop);
```

StretchBlt 除了比 BitBlt 增加两个指明源位图宽和高的参数外，其它同 BitBlt 一样。这样 StretchBlt 可用于指定源块和目标块的大小，而 BitBlt 只能指定一个块的大小。如果目标大小(nWidth,nHeight)小于源大小(nSrcWidth,nSrcHeight)，则图形压缩，如果目标大小大于源大小则图形扩大。如果 nWidth 和 nSrcWidth 的符号不同(一正一负)，则目标图形是原图的水平镜象。同样，如果 nHeight 与 nSrcHeight 符号相反，则目标图形是原图的垂直镜象。

6.3 使用 DIB 位图

上一节介绍的 GDI 位图都是与设备有关的，移值性很差。如果要将位图保存在文档中，

以便其他机器能够对它进行读取，则必须使用与设备无关的格式。几乎所有的绘图应用程序的工具都是以 DIB 格式来处理 BMP 磁盘文件的。DIB 格式可支持单色、16 色以及 256 色位图，也支持 24 位彩色位图。为了节省磁盘空间，DIB 格式还可对 16 色和 256 色位图进行压缩。

6.3.1 BMP 文件的组成

BMP 图像文件结构可分为三部分：表头、调色板和图像数据。只有全彩色 BMP 图像文件内没有调色板数据，其余不超过 256 种颜色的图像文件都必须设定调色板信息，即使是单色 BMP 图像文件也不例外。

1. BMP 表头数据

Windows 把 BMP 图像文件表头细分成两组数据结构：BITMAPFILEHEADER 和 BITMAPINFOHEADER，如果还需要调色板数据，则放在表头之后，两数据结构的内容如下：

```
typedef struct tagBITMAPFILEHEADER
{WORD bfType;//指定文件类型，必须是 BM。
DWORD bfSize;//双字节为单位指定文件的大小。
WORD bfReserved1;//保留，必须为 0。
WORD bfResreved2;//保留，必须为 0。
DWORD bfOffBits;//以字节为单位指出从 BITMAPFILEHEADER 到位图的位的偏移量。
}BITMAPFILEHEADER;
typedf stucct tagBITMAPINFOHEADER{
DWORD biSize;//指定该结构所需的字节数。
LONG biWidth;//以像素为单位指定位图的宽度。
LONG biHeight;//指定位图的高度，若为负值，该位图原点在左上角，高为 biheight 的绝对值。
WORD biPlanes;//为目标设备指定位面数，该值必须置为 1。
WORD biBitloisnt;//指定每个像素的位数，必须为 1,4,8,16,24 或 32 之一。
DWORD bicompression;//为一个压缩位图指定压缩类型。
DWORD biSireImage;//指定图像大小，若不压缩位图，可将该值置为 0。
LONG biXPelsPerMeter;//以像素点/米为单位指定位图的目标设备的水平分辨率。
LONG biYPelsPerMeter;//以像素点/米为单位指定位图的目标设备的垂直分辨率。
DWORD biClrUsed;//指定该位图实际使用的颜色表中的颜色索引数。
DWORD biClrImportant;//指定显示该位图所需的最重的颜色索引数目。
}BITMAPINFOHEADER;
```

2. 调色板数据

Windows 将 BMP 图像文件的调色板数据结构命名为 RGBQUAD，该结构描述组成一个颜色的红、绿、蓝相对强度值。

```
typedef struct tagRGBQUAD
{
BYTE rgbBlue;//指定颜色中蓝色的强度值。
BYTE rgbGreen;//指定颜色中绿色的强度值。
```

BYTE rgbRed;//指定颜色中红色的强度值。
BYTE rgbReserved;//保留,必须为0。
}RGBQUAD;

注意:此处调色板数据的排列顺序——蓝、绿、红与一般图像文件的颜色数据排列顺序正好相反。即使是单色的 BMP 图像文件,也必须存放调色板数据。

还有一个重要的数据结构 BITMAPINFO,该结构由前面介绍的 BITMAPINFOHEADER 和 RGBQUAD 结构组成,它提供了 WindowsDIB 的大小和颜色的完整定义。因此也可这样说,DIB 位图由两个不同的部分组成:描述位图大小和颜色的 BITMAPINFO 结构和定义位图像素的字节数组,其结构正说明了这一点。

typedef seruct tagBITMAPINFO
{BIMAPINFOHEADER bmiHeader;
RGBQUAD bmiColors[2];
}BITMAPINFO;

3.不压缩图像数据

BMP 图像文件对图像数据有三种处理方式:
(1)不压缩数据,任何 BMP 图像文件都能以这种方式处理。
(2)RLE4 压缩法,这是专用于 16 色图像数据的压缩方法。
(3)RLE8 压缩法,只用于压缩处理 256 色图像数据。

不压缩图像数据是 BMP 图像文件的通用处理方式。虽然这种作法会使得 BMP 图像文件的大小大于其它有压缩处理的图像文件,但是少了一道压缩和解压缩的手续,也让 BMP 图像文件的读或存取文件的速度超过压缩处理的图像文件。既然没有了压缩处理的问题,只要了解图像数据的排列及存储方式,必能正确地处理未压缩数据的 BMP 图像文件。

BMP 图像文件内的图像数据的排列顺序是以图像的左下角为起点,按照由左至右,由下而上的次序,将图像数据一点一点存入文件的。图像数据的存储方式是:单色图像是以一个字节记录八点;16 色图像是一个字节记录两点,左边四个 Bits 存第一点,右边四个 Bits 存第二点;256 色图像是一个字节记录一点;全彩色图像则是三个字节记录一点,而以 RGB,RGB,RGB…的次序排列下来。除此之外,BMP 文件规定文件内每行字节的个数必须是 4 的倍数,若未达到 4 的倍数,必须在每行的末端加上几个字节,以补足差额。

实际上 Windows32API 提供了许多读取、构造、存取 DIB 文件的函数。大大简化了有关 DIB 位图的操作,这在下节的例子里将要介绍。

6.4 使用 GDI DIB 位图的两个例子

在本节我们将给读者演示两个实例,分别就 GDI、DIB 位图使用进行实例介绍,相信读者看完这两个例子后,一定会对本章内容有更好的了解。

6.4.1 Gdiball 实例

该例子演示用 GDI 绘图的使用,如图 6.1 所示。在单文档视窗内显示了两个 Icon,左上

角的是 IDR _ MAINFRAME,靠中下的是 IDR _ GDIBALTYPE,均是缺省生成的。最引人注目的是一红一绿的两个小球在时钟控件的控制下从视窗的中间一开始相互对称地绕着 8 字型相向运动,当小球运动到与窗口"相碰"的时候,随即按光线的反射方向运动。这个例子生动地说明了静态显示 GDI 图形和动态显示的方法。

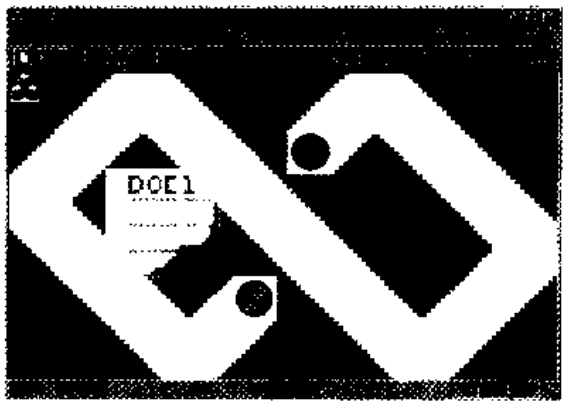


图 6.1 Gdiball 实例图示

先生成单文档工程 gdiball,再参考下面的文件 gdiballView.h 和 gdiballView.cpp 修改缺省产生的相应源文件。

```
// gdiballView.h : interface of the CGdiballView class
//
/////////////////////////////////////////////////////////////////////////////

#if ! defined(AFX_GDIBALLVIEW_H__D287212E_32E0_11D2_BA07_444553540000__INCLUDED_)
#define AFX_GDIBALLVIEW_H__D287212E_32E0_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CGdiballView : public CView
{
protected: // create from serialization only
    CGdiballView();
    DECLARE_DYNCREATE(CGdiballView)

// Attributes
public:
    CGdiballDoc* GetDocument();

    CPoint m_ptPixel;          // pixel size
```

```
        CSize   m_sizeTotala;            // total size for ball bitmap
    CSize   m_sizeTotalb;
        CSize   m_sizeRadiusa;           // radius of ball
    CSize   m_sizeRadiusb;
        CSize   m_sizeMovea;              // move speed
    CSize   m_sizeMoveb;
        CPoint m_ptCentera;              // current center for the ball
    CPoint m_ptCenterb;
        // for replicating bouncing ball
        CBitmap m_bmBalla;
        CBitmap m_bmBallb;

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CGdiballView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL

// Implementation
public:
        void makenewball();
        virtual ~CGdiballView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CGdiballView)
        afx_msg void OnSize(UINT nType, int cx, int cy);
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnTimer(UINT nIDEvent);
```

```
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in gdiballView.cpp
inline CGdiballDoc* CGdiballView::GetDocument()
   { return (CGdiballDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_GDIBALLVIEW_H__D287212E_32E0_11D2_BA07_444553540000__
INCLUDED_)

// gdiballView.cpp : implementation of the CGdiballView class
//

#include "stdafx.h"
#include "gdiball.h"

#include "gdiballDoc.h"
#include "gdiballView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CGdiballView

IMPLEMENT_DYNCREATE(CGdiballView, CView)

BEGIN_MESSAGE_MAP(CGdiballView, CView)
        //{{AFX_MSG_MAP(CGdiballView)
        ON_WM_SIZE()
        ON_WM_CREATE()
        ON_WM_TIMER()
        //}}AFX_MSG_MAP
```

```
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGdiballView construction/destruction
#define ABS(x) ((x) < 0 ? -(x) : (x) > 0 ? (x) : 0)

CGdiballView::CGdiballView()
{


}

CGdiballView::~CGdiballView()
{
}

BOOL CGdiballView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CGdiballView drawing

void CGdiballView::OnDraw(CDC* pDC)
{
        CGdiballDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        pDC->SaveDC();
        CRect Rect;
        GetClientRect(&Rect);
        CBrush brush(RGB(128,128,128));
        pDC->SelectObject(&brush);
        pDC->PatBlt(Rect.left,Rect.top,Rect.Width(),
                Rect.Height(),PATCOPY);
        HICON Hicon;
```

```
        int Iconheight,Iconwidth;
        HIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
        Iconwidth = ::GetSystemMetrics(SM_CXICON);
        Iconheight = ::GetSystemMetrics(SM_CYICON);
        pDC->DrawIcon(0,0,HIcon);
        BITMAP bm;
        CDC m_memDC;
        CBitmap * pBitmap = new CBitmap;
        pBitmap->LoadBitmap(IDB_BITMAP);
        m_memDC.CreateCompatibleDC(NULL);
        CBitmap * pOldBitmap = (CBitmap *)m_memDC.SelectObject(pBitmap);
        HBITMAP m_holdbitmap = (HBITMAP)pOldBitmap->GetSafeHandle();
        pBitmap->GetObject(sizeof(bm), &bm);
        int m_nSwidth = bm.bmWidth;
        int m_nSheight = bm.bmHeight;
        CRect rect(0,0,m_nSwidth,m_nSheight);
        pDC->DPtoLP(&rect);
        int m_nDwidth = rect.Width() * 4;
        int m_nDheight = rect.Height() * 4;
        pDC->StretchBlt(100,100,m_nDwidth,m_nDheight, &m_memDC,
              0,0,m_nSwidth,m_nSheight,SRCCOPY);
        m_memDC.SelectObject(pOldBitmap);
        pDC->RestoreDC(-1);

}

////////////////////////////////////////////////////////////////////////////
// CGdiballView printing

BOOL CGdiballView::OnPreparePrinting(CPrintInfo * pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}

void CGdiballView::OnBeginPrinting(CDC * /* pDC */, CPrintInfo * /* pInfo */)
{
        // TODO: add extra initialization before printing
}

void CGdiballView::OnEndPrinting(CDC * /* pDC */, CPrintInfo * /* pInfo */)
{
        // TODO: add cleanup after printing
}
```

```
/////////////////////////////////////////////////////////////////////////////
// CGdiballView diagnostics

#ifdef _DEBUG
void CGdiballView::AssertValid() const
{
	CView::AssertValid();
}

void CGdiballView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CGdiballDoc* CGdiballView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CGdiballDoc)));
	return (CGdiballDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CGdiballView message handlers

void CGdiballView::makenewball()
{
	m_sizeTotala.cx = (m_sizeRadiusa.cx + ABS(m_sizeMovea.cx)) << 1;
	m_sizeTotala.cy = (m_sizeRadiusa.cy + ABS(m_sizeMovea.cy)) << 1;

	m_sizeTotalb.cx = (m_sizeRadiusb.cx + ABS(m_sizeMoveb.cx)) << 1;
	m_sizeTotalb.cy = (m_sizeRadiusb.cy + ABS(m_sizeMoveb.cy)) << 1;

	if (m_bmBalla.m_hObject != NULL)
		m_bmBalla.DeleteObject();		// get rid of old bitmap

	if (m_bmBallb.m_hObject != NULL)
		m_bmBallb.DeleteObject();		// get rid of old bitmap

	CClientDC dc(this);
	CDC dcMema;
	CDC dcMemb;

	dcMema.CreateCompatibleDC(&dc);
```

```
dcMemb.CreateCompatibleDC(&dc);

m_bmBalla.CreateCompatibleBitmap(&dc, m_sizeTotala.cx, m_sizeTotala.cy);
ASSERT(m_bmBalla.m_hObject != NULL);
m_bmBallb.CreateCompatibleBitmap(&dc, m_sizeTotalb.cx, m_sizeTotalb.cy);
ASSERT(m_bmBallb.m_hObject != NULL);

CBitmap* pOldBitmapa = dcMema.SelectObject(&m_bmBalla);
CBitmap* pOldBitmapb = dcMemb.SelectObject(&m_bmBallb);

// draw a rectangle in the background (window) color
CRect recta(0, 0, m_sizeTotala.cx, m_sizeTotala.cy);
CBrush brBackgrounda(::GetSysColor(COLOR_WINDOW));
dcMema.FillRect(recta, &brBackgrounda);

CRect rectb(0, 0, m_sizeTotalb.cx, m_sizeTotalb.cy);
CBrush brBackgroundb(::GetSysColor(COLOR_WINDOW));
dcMemb.FillRect(rectb, &brBackgroundb);

CBrush brCrossa(HS_DIAGCROSS, 0L);
CBrush* pOldBrusha = dcMema.SelectObject(&brCrossa);

CBrush brCrossb(HS_DIAGCROSS, 0L);
CBrush* pOldBrushb = dcMemb.SelectObject(&brCrossb);

dcMema.SetBkColor(RGB(255,0,0));
dcMema.Ellipse(ABS(m_sizeMovea.cx), ABS(m_sizeMovea.cy),
      m_sizeTotala.cx - ABS(m_sizeMovea.cx),
      m_sizeTotala.cy - ABS(m_sizeMovea.cy));

dcMemb.SetBkColor(RGB(0,255,0));
dcMemb.Ellipse(ABS(m_sizeMoveb.cx), ABS(m_sizeMoveb.cy),
      m_sizeTotalb.cx - ABS(m_sizeMoveb.cx),
      m_sizeTotalb.cy - ABS(m_sizeMoveb.cy));

dcMema.SelectObject(pOldBrusha);
dcMema.SelectObject(pOldBitmapa);
dcMema.DeleteDC();

dcMemb.SelectObject(pOldBrushb);
dcMemb.SelectObject(pOldBitmapb);
dcMemb.DeleteDC();
```

```
void CGdiballView::OnSize(UINT nType, int cx, int cy)
{
	LONG lScale;

	m_ptCentera.x = cx >> 1;
	m_ptCentera.y = cy >> 1;

    m_ptCenterb.x = m_ptCentera.x;
	m_ptCenterb.y = m_ptCentera.y;

	lScale = min((LONG)cx * m_ptPixel.x,
		(LONG)cy * m_ptPixel.y) >> 4;
	m_sizeRadiusa.cx = (int)(lScale / m_ptPixel.x);
	m_sizeRadiusa.cy = (int)(lScale / m_ptPixel.y);
	m_sizeMovea.cx = max(1, m  sizeRadiusa.cy >> 2);
	m_sizeMovea.cy = max(1, m_sizeRadiusa.cy >> 2);

	m_sizeRadiusb.cx = m_sizeRadiusa.cx;
	m_sizeRadiusb.cy = m_sizeRadiusa.cy;
	m_sizeMoveb.cx = -m_sizeMovea.cx;
	m_sizeMoveb.cy = -m_sizeMovea.cy;


	makenewball();

	CView::OnSize(nType, cx, cy);
}

int CGdiballView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CView::OnCreate(lpCreateStruct) == -1)
		return -1;
	if (! SetTimer(1, 100 , NULL))
	{
		MessageBox(_T("Not enough timers available for this window."),
				_T("MDI:Bounce"), MB_ICONEXCLAMATION : MB_OK);

		// signal creation failure...
		return -1;
	}

	CDC* pDC = GetDC();
	m_ptPixel.x = pDC->GetDeviceCaps(ASPECTX);
```

```
        m_ptPixel.y = pDC->GetDeviceCaps(ASPECTY);
        ReleaseDC(pDC);



        return 0;
}

void CGdiballView::OnTimer(UINT nIDEvent)
{
        if (m_bmBalla.m_hObject == NULL)
                return;      // no bitmap for the ball
        if (m_bmBallb.m_hObject == NULL)
                return;      // no bitmap for the ball

        CRect rcClient;
        GetClientRect(rcClient);

        CClientDC dc(this);
        CBitmap* pbmOlda = NULL;
        CBitmap* pbmOldb = NULL;

        CDC dcMema;
        CDC dcMemb;

        dcMema.CreateCompatibleDC(&dc);
        pbmOlda = dcMema.SelectObject(&m_bmBalla);

        dcMemb.CreateCompatibleDC(&dc);
        pbmOldb = dcMemb.SelectObject(&m_bmBallb);

        dc.BitBlt(m_ptCentera.x - m_sizeTotala.cx / 2,
                  m_ptCentera.y - m_sizeTotala.cy / 2,
                  m_sizeTotala.cx, m_sizeTotala.cy,
                  &dcMema, 0, 0, SRCCOPY);
        dc.BitBlt(m_ptCenterb.x - m_sizeTotalb.cx / 2,
                  m_ptCenterb.y - m_sizeTotalb.cy / 2,
                  m_sizeTotalb.cx, m_sizeTotalb.cy,
                  &dcMemb, 0, 0, SRCCOPY);

        m_ptCentera += m_sizeMovea;
        m_ptCenterb += m_sizeMoveb;

        if ((m_ptCentera.x + m_sizeRadiusa.cx >= rcClient.righ) ) {
```

```
        (m_ptCentera.x - m_sizeRadiusa.cx < = 0))
    {
        m_sizeMovea.cx = -m_sizeMovea.cx;
    }

    if ((m_ptCentera.y + m_sizeRadiusa.cy > = rcClient.bottom) | |
        (m_ptCentera.y - m_sizeRadiusa.cy < = 0))
    {
        m_sizeMovea.cy = -m_sizeMovea.cy;
    }


    if ((m_ptCenterb.x + m_sizeRadiusb.cx > = rcClient.right) | |
        (m_ptCenterb.x - m_sizeRadiusb.cx < = 0))
    {
        m_sizeMoveb.cx = -m_sizeMoveb.cx;
    }

    if ((m_ptCenterb.y + m_sizeRadiusb.cy > = rcClient.bottom) | |
        (m_ptCenterb.y - m_sizeRadiusb.cy < = 0))
    {
        m_sizeMoveb.cy = -m_sizeMoveb.cy;
    }

    dcMema.SelectObject(pbmOlda);
    dcMema.DeleteDC();

    dcMemb.SelectObject(pbmOldb);
    dcMemb.DeleteDC();


    CView::OnTimer(nIDEvent);
}
```

6.4.2 Opnbmp 实例

该例程演示如何把 bmp 文件显示在视窗里，如图 6.2 所示。实现过程是打开 bmp 文件时把该文件的数据保存在相应的数据里面，接着由视窗类的 onDraw 函数，根据保存的数据，在视窗内画图，在这个过程中没有考虑调色板的使用，故显示出来的图形有些失真。但不失为显示 bmp 图形的一种思路。先生成多文档工程 opnbmp，再参考下面文件 opnbmpDoc.h，opnbmpDoc.cpp 和 opnbmpView.h，opnbmpView.cpp 修改或添加缺省产生的相应文件。

```
// opnbmpDoc.h : interface of the COpnbmpDoc class
//
```

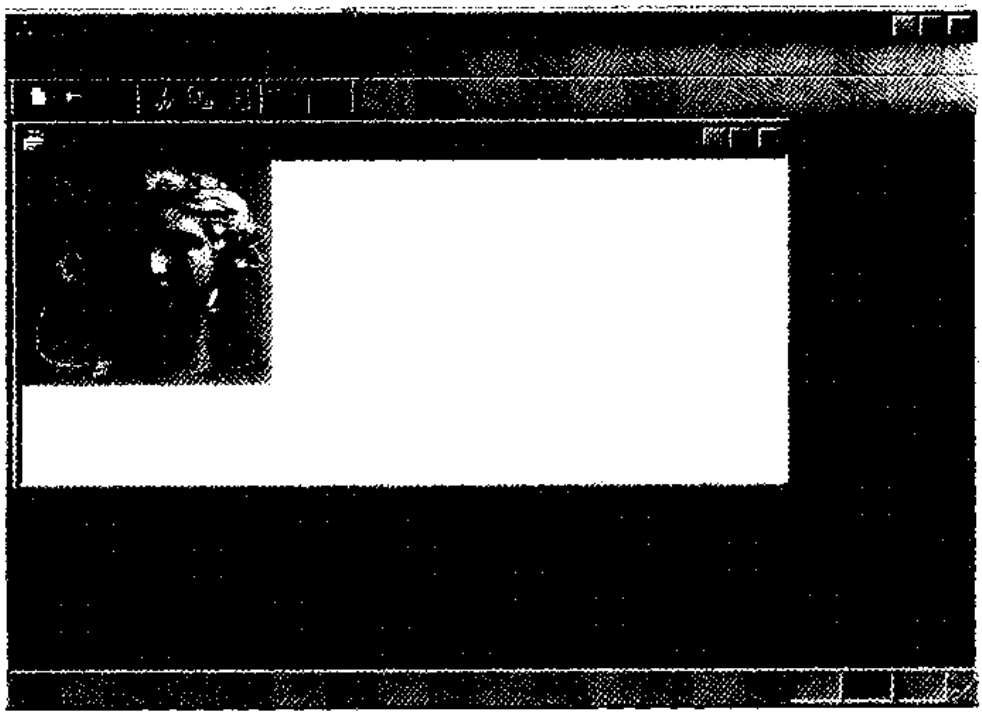


图 6.2　在视窗中显示 bmp 文件

```
////////////////////////////////////////////////////////////////////////////

#if ! defined(AFX_OPNBMPDOC_H__A6BCB94E_1CDA_11D2_BA07_444553540000__INCLUDED_)
#define AFX_OPNBMPDOC_H__A6BCB94E_1CDA_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


class COpnbmpDoc : public CDocument
{
protected: // create from serialization only
	COpnbmpDoc();
	DECLARE_DYNCREATE(COpnbmpDoc)

// Attributes
public:
	BOOL bOpen;
	char* m_lpbuf;
	DWORD m_dwLength;
	int m_nBits;
	LPBITMAPFILEHEADER m_lpBMFH;
```

```
        LPBITMAPINFOHEADER m_lpBMIH;
        LPBITMAPINFO m_lpBMI;
        LPSTR m_lpData;
protected:
        BOOL Read(CFile* pFile);
        BOOL Write(CFile* pFile);


// Operations
public:
        BOOL AllocateMemory(BOOL bReallocate=false);

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COpnbmpDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~COpnbmpDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(COpnbmpDoc)
                // 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()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.
```

```
#endif // ! defined(AFX_OPNBMPDOC_H__A6BCB94E_1CDA_11D2_BA07_444553540000
INCLUDED_)

// opnbmpDoc.cpp : implementation of the COpnbmpDoc class
//

#include "stdafx.h"
#include "opnbmp.h"

#include "opnbmpDoc.h"
#include <windowsx.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COpnbmpDoc

IMPLEMENT_DYNCREATE(COpnbmpDoc, CDocument)

BEGIN_MESSAGE_MAP(COpnbmpDoc, CDocument)
	//{{AFX_MSG_MAP(COpnbmpDoc)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COpnbmpDoc construction/destruction

COpnbmpDoc::COpnbmpDoc()


	m_dwLength=0;
	bOpen=false;
	m_lpbuf=NULL;


COpnbmpDoc::~COpnbmpDoc()
```

```
{
    if(m_lpbuf)
    {
        GlobalFreePtr(m_lpbuf);
    }
}

BOOL COpnbmpDoc::OnNewDocument()
{
    if (! CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}



/////////////////////////////////////////////////////////////////////////////
// COpnbmpDoc serialization

void COpnbmpDoc::Serialize(CArchive& ar)
{
    ar.Flush();
    if (ar.IsStoring())
    {
        Write(ar.GetFile());

    }
    else
    {
        bOpen = true;
        Read(ar.GetFile());
    }
}
BOOL COpnbmpDoc::AllocateMemory(BOOL bRealloc)
{
    if(bRealloc)
    {
        m_lpbuf = (char *)GlobalReAllocPtr(m_lpbuf,m_dwLength,GHND);
    }
    else{
```

```
        m_lpbuf = (char * )GlobalAllocPtr(GHND,m_dwLength);
    }
    m_lpBMFH = (LPBITMAPFILEHEADER)m_lpbuf;
    m_lpBMIH = (LPBITMAPINFOHEADER)(m_lpbuf + sizeof(BITMAPFILEHEADER));
    m_lpBMI = (LPBITMAPINFO)m_lpBMIH;
    return true;
}

BOOL COpnbmpDoc::Read(CFile * pFile)
{
    ASSERT(m_dwLength = = 0);
    m_dwLength = pFile->GetLength();
    if(! AllocateMemory())
        return false;
    DWORD dwCount = pFile->ReadHuge(m_lpbuf,m_dwLength);
    if(m_dwLength! = dwCount)
    {
        AfxMessageBox("Read Error");
        return false;
    }
    if(m_lpBMFH->bfType! = 0x4d42)
    {
        AfxMessageBox("Invalide Bitmap File");
        return false;
    }
    ASSERT((m_lpBMIH->biBitCount = = 1) || (m_lpBMIH->biBitCount = = 4) ||
        (m_lpBMIH->biBitCount = = 8) || (m_lpBMIH->biBitCount = = 24));
    m_lpData = (LPSTR)m_lpBMFH + m_lpBMFH->bfOffBits;
    m_nBits = m_lpBMIH->biBitCount;
    return true;

}

BOOL COpnbmpDoc::Write(CFile * pFile)
{
    TRY
    {
        pFile->WriteHuge(m_lpbuf,m_dwLength);
    }
    CATCH(CException,e)
    {
        AfxMessageBox("Write Error");
        return false;
    }
```

```
    END_CATCH
    return true;

}

/////////////////////////////////////////////////////////////////////////////
// COpnbmpDoc diagnostics

#ifdef _DEBUG
void COpnbmpDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void COpnbmpDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// COpnbmpDoc commands
// opnbmpView.h : interface of the COpnbmpView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_OPNBMPVIEW_H__A6BCB950_1CDA_11D2_BA07_444553540000__INCLUDED_)
#define AFX_OPNBMPVIEW_H__A6BCB950_1CDA_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class COpnbmpView : public CView
{
protected: // create from serialization only
    COpnbmpView();
    DECLARE_DYNCREATE(COpnbmpView)

// Attributes
private:
    LPBITMAPINFOHEADER m_lpBMIH;
```

```
        LPBITMAPINFO m_lpBMI;
        LPSTR m_lpData;

public:
        COpnbmpDoc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COpnbmpView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~COpnbmpView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(COpnbmpView)
                // 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()
};

#ifndef _DEBUG  // debug version in opnbmpView.cpp
inline COpnbmpDoc* COpnbmpView::GetDocument()
   { return (COpnbmpDoc*)m_pDocument; }
#endif
```

```
//////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_OPNBMPVIEW_H__A6BCB950_1CDA_11D2_BA07_444553540000__INCLUDED_)
// opnbmpView.cpp : implementation of the COpnbmpView class
//

#include "stdafx.h"
#include "opnbmp.h"

#include "opnbmpDoc.h"
#include "opnbmpView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////////////////////////////////////////
// COpnbmpView

IMPLEMENT_DYNCREATE(COpnbmpView, CView)

BEGIN_MESSAGE_MAP(COpnbmpView, CView)
	//{{AFX_MSG_MAP(COpnbmpView)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

//////////////////////////////////////////////////////////////////////////
// COpnbmpView construction/destruction

COpnbmpView::COpnbmpView()
```

```
:

:

COpnbmpView::~COpnbmpView()
{
:

BOOL COpnbmpView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// COpnbmpView drawing

void COpnbmpView::OnDraw(CDC* pDC)
{
        COpnbmpDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
      m_lpBMIH=pDoc->m_lpBMIH;
        m_lpBMI=pDoc->m_lpBMI;
        m_lpData=pDoc->m_lpData;

      CRect rect;
        GetClientRect(&rect);
        if(pDoc->bOpen)
        {
                if(! pDoc->m_lpbuf)
        {
                MessageBox("m_lpbuf null!");
                return ;
        }

        if(!::StretchDIBits(pDC->GetSafeHdc(),0,0,rect.Width(),rect.Height(),
                0,0,(WORD)m_lpBMIH->biWidth,(WORD)m_lpBMIH->biHeight,m_lpData,
                m_lpBMI,DIB_RGB_COLORS,SRCCOPY))
        {
                MessageBox("Stretch Error");
        }
        }
```

```
}

/////////////////////////////////////////////////////////////////////////////
// COpnbmpView printing

BOOL COpnbmpView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void COpnbmpView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void COpnbmpView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// COpnbmpView diagnostics

#ifdef _DEBUG
void COpnbmpView::AssertValid() const
{
    CView::AssertValid();
}

void COpnbmpView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

COpnbmpDoc* COpnbmpView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(COpnbmpDoc)));
    return (COpnbmpDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
```

```
// COpnbmpView message handlers
```

6.5 小　　结

本章通过两个生动的例子阐述了如何使用 GDI 位图和 DIB 位图。好的图片给人以美感，同样，好的程序界面是吸引用户不可忽视的一个方面。现代人追求动感、舒畅，通过这一章的学习，您定能满足用户的需要。

第7章 菜 单

菜单允许用户用一种标准方法与Windows应用程序进行交互，即一个样式一致的用户界面对象——菜单栏；Windows同时也支持一种更新的菜单使用方法，即允许用户在一个用户界面对象上单击鼠标右键来显示一个弹出或上下文菜单。菜单栏、子菜单和上下文菜单为Windows应用程序提供了样式一致的用户界面，这种应用程序的一致性使得用户可以轻松学会新的应用程序。

7.1 创建菜单

在Windows中通常是先设计菜单、图标和对话框，然后再把它们加到程序中。例如，在编写C++程序之前可以设计对话框的形状、位置、类型和大小，然后在扩展名为.rc的资源文件中加上这些规格说明。在Visual C++中可使用菜单编辑器完成这一功能。

7.1.1 动态创建菜单

一般地，在菜单编辑器中能对菜单进行的处理在程序运行时也同样能做，例如可以从零开始创建菜单、增加菜单项，或者删除菜单项，而且对于某些菜单属性，自绘制就像位图一样（或者是通过调用GDI绘制的几何图案），如自绘制菜单和定制的位图检查标记只有在程序运行时才能进行操作。尽管MFC提供了一个菜单类（Menu），但是AppWizard建立的代码并不使用它来创建菜单。MFC并不像WindowsAPI那样支持窗口和菜单的同时创建，而是在创建Windows的窗口时，由类CFrameWnd请求创建一个菜单。尽管CFrameWnd从逻辑上拥有菜单类的对象，它实际没有菜单类（CMenu）数据成员。虽然一般不会使用CMenu类来直接创建一个连接到框架窗口的菜单，但却可能使用它来对菜单对象进行动态操作，例如为了得到一个窗口菜单CMenu类指针，可以进行以下调用：

```
CMenu * PMenu = GetMenu();
```

这个函数返回一个从来没有创建过的对象指针。为什么能得到这种对象呢？简单地讲，是MFC在执行该函数时创建了一个CMenu类对象并将其放入临时对象中。

下面，我们将介绍菜单的动态创建方法，先从如何修改一个弹出式菜单的内容开始。

首先讨论如何从一个弹出式菜单中增加或删除菜单项。为此需要另一个CMenu指针来指向子菜单，在下面程序代码中，通过CMenu::GetSubMenu()函数来取得子菜单指针。

```
CMenu * pMenu = GetMenu();
CMenu * pSubmenu = pmenu->GetSubMenu(1);
```

这里CMenu::GetSubMenu()函数定义如下：

```
CMenu * GetSubMenu(int npos);
```

单一参数npos是一个以0为基的子菜单的索引。

我们可以使用 CMenu 类中基本菜单修改或查询函数，这些函数使用方法都很简单，读者可查阅有关参考书，例如，使用上面得到的子菜单指针(psub menu)，可以在子菜单的后面加上一个菜单项。

```
psubmenu→AppendMenu(MF _ STRING,//菜单标识
ID _ MENU _ ADDNEWITEM,//命令 ID
"Add New Item"//菜单字符串);
```

尽管 MFC 帮助数据库描述了参数信息，但最好还是仔细研究一下 CMenu::AppendMenu()的函数声明：

```
BOOL AppendMenu(UINT nFLag,
UINT nIDNewItem=0,
LPCTSTR lpSzNewItem=NULL);
```

其中参数 nFlag 是所要建立的菜单项的标识，当增加一个菜单项时，总是从几种菜单项类型中选择一种，经常使用的设置有三种：标识 MF _ STRING 用于建立一个以字符串标识的菜单项；标识 MF _ SEPARATOR 用于建立一个菜单项隔离符，以及标识 MF _ POPUP 用于建立一个新的弹出式菜单。每种标识类型中的选项是互斥的，标识类型之间用"|"连接。第二个参数 nIDNewItem 取决于是否插入一个新菜单项或一个新的弹出式菜单，如果插入菜单项，该参数含有一个整数命令 ID，如果插入一个新弹出式菜单，该参数表示插入菜单的句柄。最后一个参数 lpSZNewItem，依赖于插入项的类型，对 MF _ STRING，它是一个指向字符串的指针，对于 MF _ OWNERDRAW，它是一个唯一的标识。

7.2 快捷键的使用

在 Windows 的某些菜单和菜单项中有一个带下划线的字母，这个字母表示一个快捷键(shortcut key)。当打开一个带有快捷键的菜单时，用户只需同时按下 Alt 键和划线字母就可以选择某个特定的菜单和菜单项。例如，若要在执行程序时打开 File 菜单只需按下 Alt + F 键。

7.2.1 建立快捷键

在应用用程序中若用使用快捷键，则必须先对其作一定义。定义快捷键有两个基本步骤：首先建立快捷键表资源，该资源定义了键码和用户按下加速键后发出的命令 ID；然后，通知用户这个新的快捷键。对于那些和菜单项等价的快捷键，可以通过修改菜单项的字符来告诉用户。以下会向读者介绍每一步所做的具体工作。

1. 建立快捷键表资源

Visual C++ 集成环境内含有一个快捷键表编辑器。就像菜单编辑器一样，快捷键表编辑器提供了对每一项快捷键的编辑功能。在快捷键表编辑器中双击任何一个快捷键就可以打开如图 8.1 所示的加速键属性对话框，按下此处的图钉图标可以固定住该对话框。

点击快捷键属性对话框中的"?"号按钮，可获得每项控制的帮助信息。快捷键表本身有三栏：命令 ID、键和类型，这大致和快捷键属性对话框中的三个主要域相对应。下面我们讨论每个快捷键的属性，先从命令 ID 开始：

命令 ID 说明了当用户按下快捷键后，消息 WM _ COMMAND 要传送的命令码。除了 MFC 在文件 AFXRES.H 中的预定义的值之外，在应用程序的 RESOURCE.H 文件中也列出了有关命令 ID 的值。定义一个快捷键时，要确保它的命令 ID 与其对应菜单项的命令 ID 相同，这样，只要提供一个命令处理函数就可同时处理菜单和其对应快捷键的输入。Key 域指定了快捷键所用的键盘键。对于字母和数字字符，可以敲入所需的字母或数字，对于不可显示的字符，要使用虚键码。当浏览与 Key 域连接的组合框时，将看到以 VK 打头的虚键，如 VK _ F1 是 F1 功能键的虚键码，VK _ DOWN 为下箭头的虚键。

若不能确定使用哪个虚键码，最简单的方法是：点击 NextTypeKey 按钮；然后，按下自己想要的快捷键组合键。这样，快捷键编辑器就自动地填充这个键的快捷键属性对话框中的键，修改键以及类型域的值。

通过点击修改域中的检查框，能够为快捷键提供修改键，而且，应尽量避免使用 Alt 键组合的字母数字键来充当加速键，因为它有可能和菜单中的存取键冲突。即使发现一个 Alt 键组合的字母数字键并不和菜单的存取键相冲突，这样定义修改键不符合微软的习惯。类型域提供了在 ASCII 和虚键之间的选择。为了不依赖于某一具体种类的键盘，几乎所有的快捷键都采用虚键表示。可是，程序有时需要定义 ASCII 类型的快捷键来充当命令。在类型域中选择 ASCII 就可以定义字符而非键盘命令键来充当命令。ASCII 类型要区分大小写字母，而用户使用过程中一般都不区分大小写命令，故应避免将“A”和“a”定义为两个不同的命令。

2. 建立快捷键的菜单提示

要告诉用户所使用的快捷键，最好的方法是将其显示在菜单项中菜单名的后面。在菜单资源中，只要在菜单项名后面加入一个“&t”和快捷键的名字即可，例如标准的 Edit 菜单下的 Paste 菜单项对应的快捷键是 Ctrl + V。下面的菜单资源入口显示了在 Edit 菜单下的 Paste 菜单项中是如何加入快捷键提示的。

```
Paste&t Ctrl + V
```

在某些情况下，菜单项名后面没有对应的快捷键提示，这时，应该在帮助信息中告诉用户这个快捷键。

7.2.2 装入快捷键

为了给框架窗口装入一个快捷键资源，使用类方法 CFrameWnd::LoadAccedTable()。此方法原型如下：

```
BOOL LoadAccedTable(LPCTSTR lpszResourceName)
```

按照语法，参数 lpszResourceName 是加速键资源的名字(为 char *)或一个整数标识符(通过使用 Windows 宏 MAFEINTRESOURCE 将它转化为 Char *)。由于这是一个布尔型方法，当成功地装入快捷键资源的时候，返回真值，否则返回假值。在运行期间，调用 LoadAccedTable()来装入一个指定的快捷键表，每次只能装入一个快捷键表。当结束一个应用程序的执行时，所有从资源中装入的表都自动地被删除。

7.3 命 令 处 理

应用框架提供了一个非常复杂的命令消息传递系统，命令消息的传递路径决定着用应用

框架的哪一部分来处理命令比较合适。除了直接在资源的 ID 标识符上映射处理消息外，还可用 CWnd 的 SendMessage 或 PostMessage 函数进行传送。针对菜单，本节除说明菜单消息的处理路径外，还要阐述如何处理菜单消息。

7.3.1 菜单消息的处理路径

在 MFC 程序中，大多数要处理的消息都针对一种类型的对象：窗口。这符合 Windows-API 的模式，而且，这些消息的命令都是有实际意义的，如消息 WM_CREATE、WM_MOVE 和 WM_SIZE 就告诉我们一个窗口被创建，移动和改变其大小了。在 WindowsAPI 和 MFC 中，只有窗口对这类消息感兴趣，而且，非窗口类消息，例如键盘和鼠标消息，都直接针对某一具体的窗口，而不是被重新定向到其它对象上。MFC 和 WindowsAPI 在处理菜单消息时不一样。当应用创建菜单时，拥有菜单的框架窗口不必处理每一个菜单项的消息，相反，菜单消息可由拥有这个菜单命令资源的其它对象来处理。例如 CCmdTarget 派生类对象以及应用对话（由 CWinApp 派生），对于支持文档视窗架构的应用程序，文档对象和视窗对象也能处理菜单消息。

尽管任何由 CCmdTarget 派生的类对象都能接收菜单消息，但并不是所有的这类对象都可以同时接收到菜单消息，事实上，菜单消息是被按照一定的路径传送的。下表列出了当 MFC 发送命令消息给非文档视窗架构的应用，单文档的应用以及多文档应用时，查找消息映射表的次序。

表 7.1 不同应用类型中命令消息的处理途径

应用类型	基类	描述
非文档/视窗架构	CFrameWnd/CWndApp	框架窗口首先得到消息，应用第二个得到消息。
单文档界面(SDI)	/CSingleDocTemplate/CFrameWnd/CWndApp	激活视窗文档模板第二个得到消息，框架窗口第四个得到消息，应用最后得到消息。
单文档界面(SDI)	CView CDocument	激活的视窗首先得到消息，激活视窗的文档第二个得到消息。
多文档界面(MDI)	CMultiDocTemplate CMDIChildFrame CMDIFrameWnd CWinApp	激活视窗的文档模板第二个得到消息，子框架窗口第四个得到消息，父框架窗口第五个得到消息，应用最后得到消息。
多文档界面(MDI)	CView CDocument	激活的视窗首先得到消息，激活视窗的文档第二个得到消息。

若程序员要改变 MFC 的命令消息的发送路径，可以定义自己的优先级以把消息传给任一特定的对象。但是需要重载几个类中的 OnCmdMsg()这一命令路径查询函数。在进行此项工作之前，要研究一下 MFC 的源程序以便使自己的修改和其基类的处理相适应，尤其要注意下面的类：CView、CDocument、CFrameWnd 和 CMDIFrameWnd 之中缺省的 OnCmdMsg()成员函数。

7.3.2 处理两种菜单消息

对于菜单项，ClassWizard 允许程序员在 COMMAND 和 UPDATE_COMMAND_UI 中

进行选择，在 MFC 的源程序中，这两种消息间有细微的不同，它们分别是 WM _ COMMAND 和 ON _ UPDATE _ COMMAND _ UI。

WM _ COMMAND 消息意味着用户选择了一个菜单项（或快捷键，亦或工具条的按钮），这是用户的请求动作，要求得到一定的响应。程序员必须为菜单中的每一个菜单项提供一个 WM _ COMMAND 类的消息处理，否则，MFC 将找不到菜单项的 WM _ COMMAND 处理函数，该菜单项在显示时就会变灰，表明该菜单项失效，而用户是无法通过变灰的菜单项来发出命令的。下面是一个空的由 ClassWizard 创建的 WM _ COMMAND 处理函数：

```
Void DMainFrame::OnMenuEnable()
{ // TODO: Add your Command hardher cold here
}
```

另一方面，对 ON _ UPDATE _ COMMAND _ UI 消息的处理是完全可选择的。通过这个消息 MFC 对更新单个菜单项的显示外观提出特殊的要求，如程序员可能要给一个菜单项增加一个检查标记、失效一个菜单项或者激活一个菜单项。下面是一个完整的 ON _ UPDATE _ COMMAND _ UI 消息处理函数。

```
Void DMainFrame::OnUpdateCheckMenuItem(CCmdUI pl)
{ //如果 d _ bMenuChecked 为真则加上检查标志
// 否则，清楚检查标志
pl→Setcheck(d _ bMenuchecked);
//如果 d _ bMenuEnable 为真真则激活
// 否则，失效
pl→Enable(d _ bMenuEnable);
}
```

上面代码中，程序员得到一个指向 CCmdUI 对象的指针 Pl，通过调用 CCmdUI 的成员函数 SetCheck 和 Enable 来控制菜单项的状态。

7.3.3 处理一组菜单命令

若要动态创建菜单项，就需要提供菜单的处理函数，尽管可以使用 ClassWizard 来创建静态的处理函数，但是却无法知道程序运行时用户实际上需要动态创建多少个菜单项的命令处理函数。在这种情况下，MFC 允许使用一个命令处理器来处理一组命令，但需要程序员进行下面的动作。

程序员要做的第一件事是挑出一组和已被使用的命令 ID 不冲突的命令 ID。

第二件事，程序员需要在类的定义中增加一个成员函数。ClassWizard 为每个消息建立代码：包含文件声明、消息映射定义和成员函数定义。下面是处理一组命令 ID 的消息处理函数的例子。

```
Class CMainFrame::public CFrame
{
Void OnMyCommandRange(UINT id);
};
```

第三件事，建立消息映射入口，通过该入口，MFC 的消息处理机制能够找到相应的函数。

为了创建消息映射入口,需要在 BEGIN _ MESSAGE _ MAP()和 END _ MESSAGE MAP()之间增加一个映射入口宏 ON _ COMMAND _ RANGE()。该宏带三个参数:范围的起始、范围的结尾和函数。

```
ON_COMMAND_RANGE(IDMIN,IDMAX,ONnMyCOMMANDRANGE);
```

最后一件事是进行函数定义,该命令处理函数带有一个用户选择菜单项的命令 ID 参数。如下所示:

```
Void CMainFrame::OnMyCommandRange(UINT id)
{ SWitch(id)
case ID_COMMAND1:
case ID_COMMAND2:
};
```

7.4 菜单项的允许和禁止

如果某菜单项在当前命令传递路径中无法找到相应的命令消息控制函数,此时应用框架就可以禁止该菜单项,这样,我们也就用不着编写 ON _ UPDATE _ COMMAND _ UI 的消息控制函数了。如果我们将 CFrameWnd 的数据成员 m _ bAutoMenuEnable 置成了 FALSE(缺省值),那么我们就可以自动获得对菜单项进行禁止这一特性。假设某文档具有两个视,而只是第一个视类中包含了对 IDM _ IDOK 命令的消息控制,则只有当第一个视处于激活状态时,主框架菜单的 Zoom 菜单项才处于允许状态。让我们再来看看另外一个例子,假设应用框架提供了 Edit 中的 Cut、Copy 和 Paste 菜单项,如果我们没有在派生视类或文档类中给出相应的消息控制函数,那么这些菜单项将会被自动禁止。

7.5 实例 Menudemo 分析

该例子演示了常规菜单,动态菜单的创建和使用,如图 7.1 所示。这是个单元文档的例程。主菜单条上除了缺省的四项外,外加了一个"绘图工具(T)",其菜单项除了最上面的"选择工具"和 separator 分隔符外是八个用来绘图的形状,为了简单起见,在具体实现时,每种不同的图形形状对应一种颜色,每当点击这些菜单项,视窗的底色也随之改变,"选择工具"项被单击后,弹出一对话框,如图 7.2 所示。有 8 个复选框,使复选框成为未选状态,则"绘图工具"菜单项缺省对应菜单项,如图 7.3 所示。这反映了菜单的动态变化。本例程另外的一个功能说明动态菜单的创建与使用。如图 7.4 所示,当在视窗内点击鼠标右键时,弹出一个自己绘制的浮动菜单,每一菜单项同样也控制着一种颜色。

下面介绍该例程的创建过程。

1)生成单文档工程 menuDemo

2)添加资源

(1)修改主菜单 IDR _ MAINFRAME,在"编辑"与"查看"项之间加入"绘图工具(T)"一项,再在它下面加上以下几项:"选择工具"(ID _ CHOOSE),separator,"直线"(ID _ LINE),"水平线"(ID _ HLINE),矩形(ID _ RETANGLE),圆角矩形(ID _ RRETANGLE),"正方形"(ID _

SQUARE),圆角方形(ID_RSQUARE),椭圆(ID_ELLIPSE)和"圆"(ID_CIRCLE).

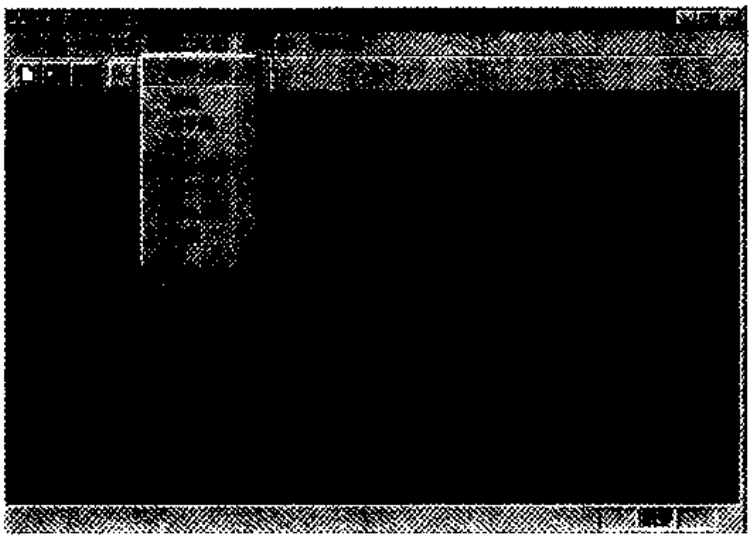


图 7.1　Menudemo 图例 1

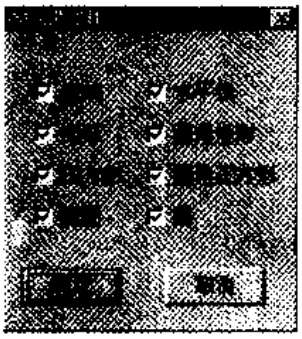


图 7.2　Menudemo 图例 2

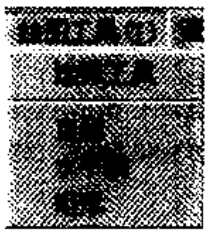


图 7.3　Menudemo 图例 3

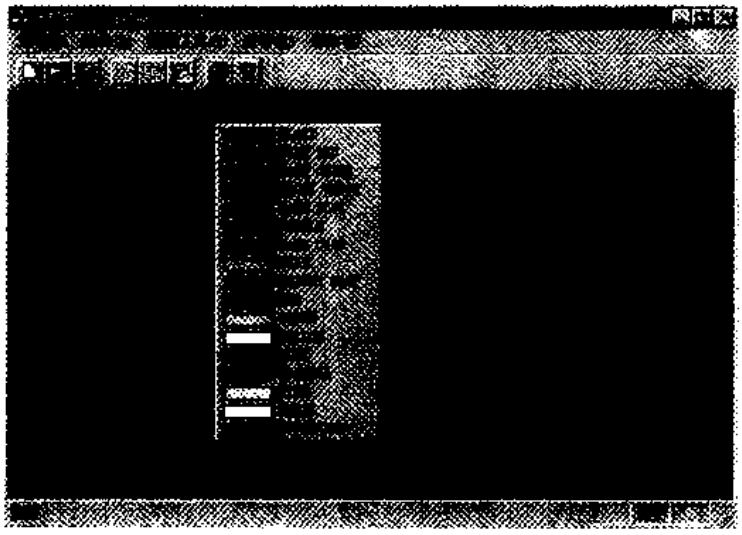


图 7.4　Menudemo 图例 4

(2) 新建一对话框 IDD_DIALOG1,实现类为 CToolDialog。它们的 ID 标识符依次从上到下、从左到右是 IDC_CHECK1 到 IDC_CHECK8。

(3) 添加 String Table 的标识提示符。

(4) 为生成浮动菜单,我们必须生成一新类 CcolorMenu,参见 menu.h 和 menu.cpp。

(5) 加入一些 ID 标识符的声明,参见 resource.h 文件。

3) 编辑文件 MainFrm.h, MainFrm.cpp, menuDemoView.h, menuDemoView.cpp 和 ToolDialog.h, ToolDialog.cpp。

```
//menu.h
#ifndef _MENU_H
#define _MENU_H

class CColorMenu : public CMenu
{
public:
        CColorMenu();
public:
        static int BASED_CODE indexMap[17];
        static COLORREF GetColor(UINT id);
public:
        HDC hDC;
public:
        virtual void DrawItem(LPDRAWITEMSTRUCT lpDIS);
        virtual void MeasureItem(LPMEASUREITEMSTRUCT lpMIS);
};
#endif



//MENU.CPP
#include "stdafx.h"
#include "menuDemo.h"
#include "menu.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[]=__FILE__;
#endif


int  BASED_CODE CColorMenu::indexMap[17]={
        0,1,2,3,4,5,6,7,12,13,14,15,16,17,18,19,0};
CColorMenu::CColorMenu()
{
```

```
        CreatePopupMenu();
        ASSERT(GetMenuItemCount() = = 0);
        for(int i = 0;i < = 16;i + + )
            AppendMenu(MF _ OWNERDRAW,ID _ COLOR0 + i,(LPCTSTR)(ID _ COLOR0 +
i));
}
COLORREF CColorMenu::GetColor(UINT id)
{
    if(id = = ID _ COLOR16)
        return ::GetSysColor(COLOR _ WINDOWTEXT);
    else
    {
        CPalette * pPal = CPalette::FromHandle((HPALETTE)GetStockObject(DEFAULT
PALETTE));
        ASSERT(pPal! = NULL);
        PALETTEENTRY pe;
        if(pPal->GetPaletteEntries(indexMap[id-ID _ COLOR0],1,&pe) = = 0)
            return ::GetSysColor(COLOR _ WINDOWTEXT);
        else
            return RGB(pe.peRed,pe.peGreen,pe.peBlue);
    }
}
void CColorMenu::DrawItem(LPDRAWITEMSTRUCT lpDIS)
{
    ASSERT(lpDIS->CtlType = = ODT _ MENU);
    UINT id = (UINT)(WORD)lpDIS->itemID;
    ASSERT(id = = lpDIS->itemData);
    ASSERT(id > = ID _ COLOR0);
    ASSERT(id < = ID _ COLOR16);
    CDC dc;
    hDC = lpDIS->hDC;
    dc.Attach(hDC);
    CRect rc(lpDIS->rcItem);
    ASSERT(rc.Width() < 500);
    if(lpDIS->itemState&ODS _ FOCUS)
        dc.DrawFocusRect(&rc);
    COLORREF cr = (lpDIS->itemState&ODS _ SELECTED)?
        ::GetSysColor(COLOR _ HIGHLIGHT):dc.GetBkColor();
    CBrush brushfill(cr);
    cr = dc.GetTextColor();
    if(lpDIS->itemState&ODS _ SELECTED)
        dc.SetTextColor(::GetSysColor(COLOR _ HIGHLIGHTTEXT));
    int nBkMode = dc.SetBkMode(TRANSPARENT);
    dc.FillRect(&rc,&brushfill);
```

```
        rc.left += 50;
        CString strColor;
        strColor.LoadString(id);
        dc.TextOut(rc.left,rc.top,strColor,strColor.GetLength());
        rc.left -= 45;
        rc.top += 2;
        rc.bottom -= 2;
        rc.right = rc.left + 40;
        CBrush brush(GetColor(id));
        CBrush * pOldBrush = dc.SelectObject(&brush);
        dc.Rectangle(rc);
        dc.SelectObject(pOldBrush);
        dc.SetTextColor(cr);
        dc.SetBkMode(nBkMode);
        dc.Detach();
}
void CColorMenu::MeasureItem(LPMEASUREITEMSTRUCT lpMIS)
{
        ASSERT(lpMIS->CtlType == ODT_MENU);
        UINT id = (UINT)(WORD)lpMIS->itemID;
        ASSERT(id == lpMIS->itemData);
        ASSERT(id >= ID_COLOR0);
        ASSERT(id <= ID_COLOR16);
        CString strColor;
        strColor.LoadString(id);
        lpMIS->itemWidth = 120;
        lpMIS->itemHeight = 15;
}


// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__3C9924CA_3244_11D2_BA07_444553540000__INCLUDED_)
#define AFX_MAINFRM_H__3C9924CA_3244_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

struct DynaMenuDef {
   char *      String;                          // color name (in string table)
   UINT        m_nID;                       // menu ID
```

```
};

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
        BOOL m_abAllowTool[NUM_TOOLS]; // flags whether colors may be used
      static DynaMenuDef m_aToolDef[NUM_TOOLS];

// Operations
public:


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnChoose();
                // 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()
```

```
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__3C9924CA_3244_11D2_BA07_444553540000__INCLUDED_)

// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "menuDemo.h"

#include "MainFrm.h"
#include "menu.h"
#include "ToolDialog.h"
#include "menuDemoDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame
DynaMenuDef CMainFrame::m_aToolDef[NUM_TOOLS] =
{
        {  T("直线"),           ID_LINE },
        { _T("水平线"),         ID_VHLINE },
        { _T("矩形"),           ID_RETANGLE },
        { _T("圆角矩形"),       ID_RRETANGLE },
        {  T("正方新形"),       ID_SQUARE },
        { _T("圆角方形"),       ID_RSQUARE },
        { _T("椭圆"),           ID_ELLIPSE },
        { _T("圆"),             ID_CIRCLE },

};

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
```

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
	ON_WM_CREATE()
	ON_COMMAND(ID_CHOOSE, OnChoose)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
	ID_SEPARATOR,           // status line indicator
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	for (int i = 0; i < NUM_TOOLS; i++)
	  m_abAllowToo[i] = TRUE;

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	if (!m_wndToolBar.Create(this) ||
		!m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
	{
		TRACE0("Failed to create toolbar\n");
		return -1;      // fail to create
	}

	if (!m_wndStatusBar.Create(this) ||
		!m_wndStatusBar.SetIndicators(indicators,
```

```
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar \n");
        return -1;      // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizeable toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
```

```
// CMainFrame message handlers

void CMainFrame::OnChoose()
{

        CToolDialog dlg(this);
        if(dlg.DoModal())
        {
            CMenu* pToolMenu = NULL;
            CMenu* pTopMenu = AfxGetMainWnd()->GetMenu();
            int iPos;
            for (iPos = pTopMenu->GetMenuItemCount()-1; iPos >= 0; iPos--)
              {
                    CMenu* pMenu = pTopMenu->GetSubMenu(iPos);
                    if (pMenu && pMenu->GetMenuItemID(0) == ID_CHOOSE)
                    {
                          pToolMenu = pMenu;
                          break;
                    }
              }

        // Update the color submenu to reflect the text colors available to
        // the active document

        // First, delete all items but ID_COLOR_OPTIONS at position 0
              for (iPos = pToolMenu->GetMenuItemCount()-1; iPos > 1; iPos--)
                    pToolMenu->DeleteMenu(iPos, MF_BYPOSITION);

        // Then, add a separator and an item for each available text color
              for (int i = 0; i < NUM_TOOLS; i++)
              {
                    if (m_abAllowTool[i] == TRUE)
                    {
                          pToolMenu->AppendMenu(MF_STRING,
                                m_aToolDef[i].m_nID, m_aToolDef[i].String);
                    }
              }
        }

}

// menuDemo.h : main header file for the MENUDEMO application
//
```

```
#if !defined(AFX_MENUDEMO_H__3C9924C6_3244_11D2_BA07_444553540000__INCLUDED_)
#define AFX_MENUDEMO_H__3C9924C6_3244_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CMenuDemoApp:
// See menuDemo.cpp for the implementation of this class
//

const NUM_TOOLS=8;
class CMenuDemoApp : public CWinApp
{
public:
	CMenuDemoApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMenuDemoApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CMenuDemoApp)
	afx_msg void OnAppAbout();
		// 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()
};


/////////////////////////////////////////////////////////////////////////////
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MENUDEMO_H__3C9924C6_3244_11D2_BA07_444553540000__INCLUDED_)
// menuDemoView.h : interface of the CMenuDemoView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MENUDEMOVIEW_H__3C9924CE_3244_11D2_BA07_444553540000__INCLUDED_)
#define AFX_MENUDEMOVIEW_H__3C9924CE_3244_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "menu.h"

class CMenuDemoView : public CView
{
protected: // create from serialization only
	CMenuDemoView();
	DECLARE_DYNCREATE(CMenuDemoView)

// Attributes
public:
	CMenuDemoDoc* GetDocument();
	CColorMenu menu;
	COLORREF col;


// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMenuDemoView)
	public:
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	protected:
```

```
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMenuDemoView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CMenuDemoView)
        afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
        //}}AFX_MSG
        afx_msg void OnColorSelected(UINT nID);

        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in menuDemoView.cpp
inline CMenuDemoDoc* CMenuDemoView::GetDocument()
   { return (CMenuDemoDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_MENUDEMOVIEW_H__3C9924CE_3244_11D2_BA07_444553540000__
_INCLUDED_)
// menuDemoView.cpp : implementation of the CMenuDemoView class
//

#include "stdafx.h"
#include "menuDemo.h"
```

```
# include "menuDemoDoc.h"
# include "menuDemoView.h"
# include "menu.h"

# ifdef _DEBUG
# define new DEBUG_NEW
# undef THIS_FILE
static char THIS_FILE[] = __FILE__;
# endif

/////////////////////////////////////////////////////////////////////////////
// CMenuDemoView

IMPLEMENT_DYNCREATE(CMenuDemoView, CView)

BEGIN_MESSAGE_MAP(CMenuDemoView, CView)
	//{{AFX_MSG_MAP(CMenuDemoView)
	ON_WM_CONTEXTMENU()
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
	ON_COMMAND_RANGE(ID_COLOR0, ID_COLOR16, OnColorSelected)

END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMenuDemoView construction/destruction

CMenuDemoView::CMenuDemoView()
{

	col = RGB(255,255,255);
}

CMenuDemoView::~CMenuDemoView()
{
}

BOOL CMenuDemoView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs
```

```
    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMenuDemoView drawing

void CMenuDemoView::OnDraw(CDC* pDC)
{
    CMenuDemoDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    pDC->SaveDC();
    CRect rect;
    GetClientRect(&rect);
    CBrush brush(col);
    pDC->SelectObject(&brush);
    pDC->PatBlt(0,0,rect.Width(),rect.Height(),PATCOPY);
  pDC->RestoreDC(-1);


}

/////////////////////////////////////////////////////////////////////////////
// CMenuDemoView printing

BOOL CMenuDemoView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CMenuDemoView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CMenuDemoView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CMenuDemoView diagnostics
```

```
#ifdef _DEBUG
void CMenuDemoView::AssertValid() const
{
	CView::AssertValid();
}

void CMenuDemoView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CMenuDemoDoc* CMenuDemoView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMenuDemoDoc)));
	return (CMenuDemoDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMenuDemoView message handlers

void CMenuDemoView::OnContextMenu(CWnd* pWnd, CPoint point)
{

	menu.TrackPopupMenu(TPM_LEFTALIGN | TPM_LEFTBUTTON,point.x,point.y,this);

}
void CMenuDemoView::OnColorSelected(UINT nID)
{
    col = menu.GetColor(nID);
	Invalidate();
}
#if !defined(AFX_TOOLDIALOG_H__1C416362_329C_11D2_BA07_444553540000__INCLUDED_)
#define AFX_TOOLDIALOG_H__1C416362_329C_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ToolDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
```

```
// CToolDialog dialog

class CToolDialog : public CDialog
{
// Construction
public:
	CToolDialog(CWnd * pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CToolDialog)
	enum { IDD = IDD_DIALOG1 };
		// NOTE: the ClassWizard will add data members here
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CToolDialog)
	protected:
	virtual void DoDataExchange(CDataExchange * pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CToolDialog)
		// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_TOOLDIALOG_H__1C416362_329C_11D2_BA07_444553540000__IN-
CLUDED_)
// ToolDialog.cpp : implementation file
//

#include "stdafx.h"
#include "menuDemo.h"
#include "ToolDialog.h"
```

```
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CToolDialog dialog

CToolDialog::CToolDialog(CWnd* pParent /*=NULL*/)
	: CDialog(CToolDialog::IDD, pParent)
{
	//{{AFX_DATA_INIT(CToolDialog)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}


void CToolDialog::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CToolDialog)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
    CMainFrame* m_pMainFrame=(CMainFrame*)GetParent();
	for (int i = 0; i < NUM_TOOLS; i++)
		DDX_Check(pDX, IDC_CHECK1 + i, m_pMainFrame->m_abAllowTool[i]);

}


BEGIN_MESSAGE_MAP(CToolDialog, CDialog)
	//{{AFX_MSG_MAP(CToolDialog)
		// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CToolDialog message handlers
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////
```

```
#if !defined(AFX_MAINFRM_H__3C9924CA_3244_11D2_BA07_444553540000__INCLUDED_)
#define AFX_MAINFRM_H__3C9924CA_3244_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

struct DynaMenuDef {
    char*       String;                         // color name (in string table)
    UINT        m_nID;                          // menu ID
};

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
        BOOL m_abAllowTool[NUM_TOOLS]; // flags whether colors may be used
      static DynaMenuDef m_aToolDef[NUM_TOOLS];

// Operations
public:


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
```

```
    CToolBar    m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnChoose();
        // 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()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__3C9924CA_3244_11D2_BA07_444553540000__INCLUDED_)
```

7.6 小　　结

本章我们与读者共同讨论了以下内容：

- 菜单的创建过程及 CMENU 类。
- 快捷键的加入使对菜单的使用更加方便快捷。
- 菜单项中相应的命令处理过程和应用。
- 菜单命令的允许/禁止的应用。

读者若真正地理解了上述内容，深信读者会在以后的学习和应用中能更加灵活地使用菜单项，使自己的程序更加实用。

第 8 章　工具栏和状态栏

Windows 在个人计算机上的霸主地位使得作为 Windows 标准窗口组成的元素工具栏和状态栏越来越受到无论是用户还是程序员的喜爱。工具栏包含位图按钮，这些按钮提供的热键功能极大地简化了用户的操作。状态栏可用来显示诸如页号、总页数、光标位置和当前时间之类的信息。本章先介绍工具栏和状态栏的基类对象控件栏，然后再分别概述工具栏和状态栏的使用。

8.1　控　件　栏

工具栏是 CToolBar 类的一个对象，而状态栏则是 CStatusBar 类的对象。CToolBar 类和 CStatusBar 类都是 CControlBar 类的派生类，而 CControlBar 类本身则是 CWnd 类派生出来的。CControlBar 类所支持的控制窗口位于主框架窗口内，并且这些控制栏窗口能够随着主框架窗口的尺寸的改变或移动自动调整自己的尺寸和位置。控制栏对象的构造、结构以及窗口的创建都是由应用框架来管理的，AppWizard 为派生框架类所产生的控制栏代码位于 MAINFRM.CPP 和 MAINFRM.H 文件中。

在典型的 SDI 应用中，CTOOLBar 对象占据了 CMainFrame 用户区域的上面部分，而 CStatusBar 对象则占据了 CMainFrame 用户区域的下面部分，视则占据了 CMainFrame 用户区域的其它部分。

假如 AppWizard 为应用产生了控制条代码，那么用户就可以从 View 菜单中选择命令分别对状态栏和工具栏进行允许和禁止。当控制栏被禁止时，它就会自动消失，并且视窗的尺寸会被重新计算。除了上述所讲的一些共同特点之外，工具栏和状态栏彼此是独立工作的，并且还具有许多不同的特性。

对象 CControlBar 通过检查其父窗口客户区域矩形放置它自己并相应地调整其大小。在考虑这个控件栏的大小后告诉其父窗口客户区域的新大小。

当 Windows 控件栏的公共控件被破坏时，CControlBar 类也使用公共数据成员 m-bAutoDelete 决定是否破坏 CControlBar 对象。

8.2　工具栏的创建和控制

工具栏(ToolBar)是包含一行按钮的控件栏，这些按钮有下压式按钮、复选框、单选按钮三种样式。工具栏通常在框架窗口的顶端排列，但 MFC 工具栏可执行一些有趣的任务。它们可以像一个独立的、可移动的、可改变大小的、可在顶端的窗口一样自由漂动，或者停放在框架窗口客户区的任何一边。工具栏目的是为菜单命令提供一个替补界面，工具栏也可以使用 ToolTips(工具提示)，ToolTips 是当鼠标移动到工具栏的按钮上时，在光标位置出现的小的弹出窗口中的文字标题。

8.2.1 工具栏的创建

AppWizard 自动为程序员创建工具栏，但是工具栏的一些特性（如显示和隐藏）被隐藏在 MFC 类中，为了给读者更为完整的叙述，我们将创建一个工具栏。

工具栏建立分五个步骤：

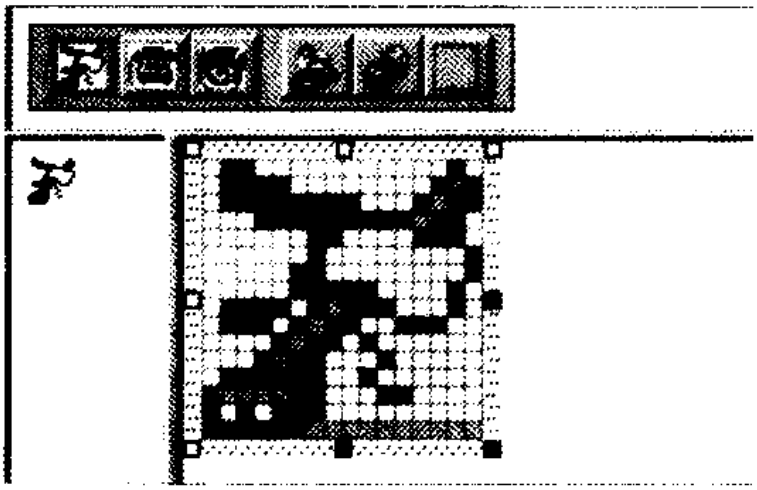
1）创建拥有按钮图像的位图。

2）创建一个映射应用按钮和程序的命令码的数组。

3）编写创建和初始化工具栏的程序代码。

4）第四和第五步是创建工具条的按钮图像和命令 ID 间的联系。

上面几个基本步骤完成后，才能细化工具条的其它属性。第一步是创建拥有所有按钮图像的位图。要将这个位图作为资源存储起来，首先要打开资源文件，使用位图编辑器，将所有按钮的位图图像排成一行，如图。缺省的单个按钮的图像大小为 16 像素宽和 15 像素高，当然也可以使用不同的尺寸，但是，必须通过调用 CToolBar::SetSize()来告诉工具对象其尺寸的大小。



第二步是建立一个映射按钮图像和命令 ID 的数组，这是一个 Unsigned int（UINT）类型的数组：

```
Static UINT BASED-CODE buttons[] =
{ //和位图 bitmapl.bmp 中次序一样。
ID_TOOLBAR_CREATE,
ID_SEPARATOR,
ID_TOOLBAR_SHOW};
```

这里 ID_TOOLBAR_CREATE 和 ID_TOOLBAR_SHOW 是两个命令码，另一项 ID_SEPARATOR 是分隔标志，在显示时使两个按钮之间隔开一定的距离。

第三步是创建和初始化工具条对象。和对其它窗口操作一样，首先要实例化该对象，然后调用初始化函数。请看下例：

```
d-pTOOLBar = new CTOOLBar();
d-PTOOLBar→Create(this,WS_CHILD | |CBRS_TOP,OX9100);
```

初始化函数 CTOOLBar::Create()重载了 CWnd::Create()。对于其它类型的窗口对象，风格域即中间的一个参数，控制对象的属性。这个例子中只有两个风格标志：WS_CHILD 和 CBRS_TOP，第一个风格 WS_CHILD 说明该窗口为一个子窗口；第二个风格 CBRS_TOP 是控制条特有的风格，它使得工具条位于框架窗口的顶端，下表列出了一些可能要用到的风格。

表 8.1 工具条窗口的风格

风格	描述
WS_VISIBLE	使控制条的初始状态为可见。
CBRS_BOTTOM	使控制条的初始位置位于可见窗口底部。
CBRS_FLYBY	当鼠标在按钮上停留时，在状态条窗口中显示命令描述
CBRS_NOALIGN	当控制条父窗口改变大小时，禁止控制条移动位置。
CBRS_TOOLTIPS	当鼠标在按钮上停留时，显示工具注释。
CBRS_TOP	使控制条的初始位置位于可见窗口的顶部。

创建好工具栏窗口后，就可以进行初始化工作了。

第四步，需要将资源文件中存储的按钮图像和工具栏连接起来，这可以通过调用 CTOOLBar::loadBitmap()来实现，如下所示。

```
pTOOLBar ->LoadBitmap(IDR_Cods);
```

第五步，也应该将命令 ID 和按钮连接起来，如下面的例子所示，这可以通过调用 CTOOLBar::SetButtons()来实现。

```
pToolBar->SetButtons(buttons,sizeof(buttons * sizeof(UINT));
```

现在，工具条的创建就算基本完成，当然可能还需要做一些优化工作。一般情况下，只有程序才能移动工具栏，但是程序员可以允许用户通过鼠标将工具栏移动，这可以通过调用 CTOOLBar::EnableDocking()和 CFrameWnd::EnableDocking()来实现：

```
pTOOLBar->EnableDocking(CBRS_ALIGN_ANY);EnableDocking(CBRS_ALIGN_ANY);
```

这样做了以后，用户就可以装载或卸下工具栏，或者在程序的控制下，通过调用 CFrameWnd::DockControlBar()来装载工具栏以及通过调用 CFrameWnd::FloatControlBar()来卸下工具条。

8.2.2 控制工具栏

1. 处理命令消息

当用户用鼠标按下某个工具栏按钮时，就会产生相应的命令消息，该命令消息可以像菜单命令消息一样传递。在大多数情况下，工具栏按钮都是和相应的菜单选项相对应的。例如，在标准应用框架工具栏中，磁盘按钮就和 FileSave 菜单项相对应，因为它们都产生了 ID-FILE-SAVE 消息，而接收到该命令消息的对象根本用不着关心该消息是通过按下工具栏中的按钮所产生的，还是通过菜单选择所产生的。

工具栏按钮并非一定非要和菜单项相对应，但如果某工具栏按钮没有与之对应的菜单项，

则建议最好为该按钮定义相应的快捷键，以便用户可以通过键盘或 Windows 的键盘宏产品来激活该命令。如果应用的某些工具条按钮没有与之相对应的菜单项，ClassWizard 就无法为它们定义相应的命令消息控制函数和更新命令消息控制函数。我们就必须自己为这些函数加入函数体、消息映射入口以及原型说明。在应用的上框架类中定义的静态 buttons 数据用来将命令同对应的按钮相联系。在正常情况下，AppWizard 在 MAINFRAM.CPP 中生成的 buttons 数组代码如下所示：

```
Static UINT BASED-CODE buss[]:
// Same order as in the bitmap toolbarbmp
ID_File_New,
ID_File_open,
ID_File_save,
ID_SEPARATOR,
ID_Edit_Cut,
ID_EDIT_COPY,
ID_EDIT_PASTE,
ID_SEPARATOR,
ID_FICE_PRINT
ID_APP_ABOUT
ID_CONTEXT_HELP,
};
```

其中 ID-SEPARATOR 用来对按钮进行分组，并在工具条中加入相应的间距，如果工具条所包含的位图片数目超过了 buttons 数组中所包含的元素数目(其中包括组分割符)，那么多余的按钮将不会被显示出来。

2. 更新 UI 消息

更新命令 UI 消息控制函数主要是用来对菜单项进行禁止和检查的，这些消息控制函数同样也可以用来作用于工具栏按钮，如果更新命令 UI 消息控制函数用 FALSE 参数来调用 CCmdUI 类的 Enable 成员函数，那么相应的按钮就会被置成禁止(变灰)状态，从而该按钮就不再会对作用于工具上的鼠标按键做出任何反应。

对于菜单项来说，CCmdDI 的 SetCbeck 函数会在菜单项的旁边显示一个检查标记，而对于工具栏来说，SetCheck 函数则把它所包含的按钮都当成检查框按钮来处理了。若更新命令 UI 消息控制函数以参数 I 来调用 SetCheck 函数，那么相应的按钮就会被置成 Down(检查)状态，而若更新命令 UI 消息控制函数以参数 D 来调用 SetCheck 函数，那么相应的按钮就会被置成 Up 状态(未被检查)。

说明：若 SetCheck 函数的参数值为 2，则相应的按钮就会置成“待决定”状态。该状态看起来有点像禁止状态，但实际上按钮仍然是活动的，并且它的颜色还是比较亮的。Microsoft 的 Word for windows 软件就对工具条中的黑体字按钮使用了 Up、Down 及待决定三种状态。如果用户选择的文本中所有字符的字体都是黑体，则黑体字按钮就处于 Down 状态，若被选中的文本中所有字符的字体都不是黑体，则该按钮就处于 Up 状态，而若被选中的文本中只有部分字符的字体为黑体，则该按钮就处于待决定状态。当某菜单项所在的弹出式菜单在屏幕上出

现时，对该菜单项进行处理的更新命令 UI 消息控制函数才会被调用，而工具条却总是处于显示状态，那么对它进行处理的更新命令 UI 消息控制函数就可连续地进行更新，若同一控制函数既要对某菜单项进行处理，又要对某工具栏按钮进行处理，那么在处理过程中及当该菜单项所在的弹出式菜单在屏幕上出现时，该控制函数都会被调用。

说明：即使工具栏按钮处于禁止状态，与之相关的键盘快捷键仍可以发送相应的命令消息，故我们自己设计的命令控制函数此时必须对快捷键命令消息及各种伪命令消息不予理会，换句话说，我们不能指望命令 UI 消息控制函数能够完全禁止相应命令的发送。

3. 显示和隐藏工具栏

在显示和隐藏工具栏的时候，需要注意的地方是：工具栏本身就是一个窗口。实际上，比起 CTOOLBar 的成员函数，这里用的更多的可能是 CWnd 的成员函数。

在能够显示或隐藏工具条之前，如果知道工具栏窗口是否已经可见对我们是很有帮助的。通过调用函数 CWnd::GetStyle()来得到 WS-VISIBLE，从而判断工具栏是否可见。下面的代码可以取得工具条窗口的可见状态标志。

```
BOOL bVisible = pTOOLBar->GetStyle() & WS_VISIBLE;
```

对函数 CWnd::Setstyle()的调用可以改变窗口的部分风格。然而，对 WS_VISIBLE 风格，这种调用却不起作用，相反，要通过调用 CWnd::ShowWindow()并且传送参数 WS_HIDE 使窗口消失。根据上面的代码得到的显示标志，显示或隐藏窗口的代码如下所示：

```
//显示或隐藏
int nShow = cbvisible? WS_HIDE:WS_SHOWNORMAL;
pTOOLBar->ShowWindow(nshow);
```

每当程序改变了工具栏状态以后，必须将这个变化通知给框架窗口，这只要通过重新计算工具栏的位置即可实现，即通过调用 CFrame Wnd::Recalclayont()来实现，如下所示：

```
RecalcLayout();
```

8.3 状态栏的创建与控制

状态栏包含两类窗片：消息行和状态指示器。左边的窗片是消息行，其余的都是状态指示器。AppWizard 创建的状态条带一个消息行和三个状态指示器。这三个指示器告诉你当前 NumLock 键、CapsLock 键和 ScrollLock 键是否按下了。消息行通常起到帮助用户的作用，例如对用户所选的菜单命令给出一定的解释。

进行状态条程序设计通常要包括下列步骤：

- 为状态栏的窗片和状态栏本身确定其各自的 ID 值。
- 创建一个定制的状态栏。
- 存取消息行的内容。
- 修改状态指示器。

8.3.1 创建状态栏(CStatusBar)

CStatusBar 类为你在 MFC 应用程序中使用状态栏提供了高级的 MFC 界面。CStatusBar

对象通常是从类 CFrameWnd 或 CMDIFrameWnd 派生的框架窗口对象的成员。Visual C++ AppWizard 为其生成的 MFC 应用程序创建了 CStatusBar 对象，此类使用起来相对比较容易。

使用 CStatusBar 对象相当简单，因为当应用程序框架产生时 Visual C++ AppWizard 通常会对你的框架窗口添加一个 CStatusBar 对象。一般使用预定义的 MFC 标识符来让 MFC 处理繁杂的工作。为此，设立 UINT 标识符数组并调用 CStatusBar::SetIndicators()来初始化状态栏窗格的数组。

例如，为了初始化一个典型的状态栏，在应用程序的主框架窗口中声明 CStatusBar 数据成员：

```
CStatusBar m_ctlStatusBar;
```

然而建立字符串数组或字符串资源 ID，这些是可以用于状态栏窗格的文本。典型地，可以用 MFC 预定义下述值：

```
Static UINT auIndicators[]:
{ ID_SEPARATOR,// Status line indicator
ID_INDICATOR_CAPS,
ID_INDICATOR_NUM,
ID_INDICATOR_SCRL,
};
```

通过调用 CStatusBar::Create()来初始化框架窗口中 WM-CREATE 消息处理器中的状态栏。

```
// create a status bar
if (m_ctlstatusBar=Create(this))
{TRACEO("Failed to create status bar/n");
return -1;
}
```

最后，调用 CStatus::SetIndicators()来设置各状态栏窗格的文本。

```
// Set the text in the status bar panes
if(!m_ctlstarus.SetIndicators(auIndicators,sizeof(auIndicators)/sizeof(UINT)))
{TRACEO("Failed to set status bar indicators \n");
return -1;
```

8.3.2 获得对状态栏的控制

在标准应用框架的运行过程中，状态栏实际上有一个子窗口，其 ID 为 AFX_IDW_STATUS_BAR，当应用框架想要显示菜单提示时，它就要寻找这一 ID。用于键盘状态指示器的更新命令 UI 消息控制函数被嵌入在主框架窗口类中，它们与一些字符串 ID 相连，这些字符串 ID 为 ID_INDICATOR_CAPS，ID_INDICATOR_NUM 和 ID_INICATOR_SCRL。为了获得对状态条的控制，我们必须使用不同的子窗口 ID，并且还必须使用不同的指示器 ID 常数。

状态条窗口的 ID 是由 CStatusBar 类的 Create 函数设置的，而该函数是由派生框架类的 onCreate 成员函数调用的。AppWizard 所产生的 CStatusBar 类的 Create 函数被包含在 MAIN-

FRM.CPP 文件中,该函数的第三个参数是窗口 ID,它被缺省设置为 AFX_IDW_STATUS_BAR。

为了能够使用我们定义的 ID,我们必须把如下函数调用:

```
m_WndStatusBar.Create(this);
```

换成:

```
m_WndStatus.Create(this,WS-CHILD | WS_VISIBLE | CBRS_BOTTOM,ID_MY_STATUS_BAR);
```

当然,我们还必须在 RESOURCE.H 文件中定义 ID_MY_STATUS_BAR 常量(利用 AppStudio)。

还有一点,标准应用框架的 View 菜单允许用户关闭或打开状态条,而这一逻辑是和 AFX_IDW_STATUS 这一窗口 ID 联系在一起的,因此,我们还必须对菜单逻辑进行修改。我们必须在自己的派生框架类中为 ID_VIEW_STATUS_BAR 命名 STATUS_BAR 为 StatusBar 菜单项的 ID。

8.3.3 管理消息行和状态指示器

要存取消息行,首先要获得指向该状态条的指针。因为状态条是用户窗口的兄弟窗口,所以只有调用 AfxGetApp 函数获取指向应用程序的指针,而通过该指针可以获得对该应用程序数据成员 mpMainWnd 的存取权。然后通过调用函数 GetDescendantWindow,m-pMainWnd 指针可使你获得状态条的地址。GetDescendantWindow 的参数是该状态条的 ID。下面语句是获取状态条地址的一个示意性例子。

```
CStatusBar * PSB;
(CStatusBar)AfxGetApp()→m_pMainWnd→Get DescendantWindow(ID_SB);
```

获得状态条地址后,调用 SetPaneText 函数就可以向消息行写入所需内容。该函数的参数有两个,第一个是该消息行窗片 ID 值的索引,第二个参数是消息行新内容的字符串,下面是一个例子:

```
int nIndex:1;
char szStr[81]="Hello There!";
PSB→setPaneText(nIndex,SzStr);
```

修改状态指示器要调用 OnUpdate 成员函数,该类函数负责响应指定 ID 的 ON-UPDATE_COMMAND_DI 消息。像工具条按钮一样,OnUpdate 函数向其参数 pCmdUI 发送 Enable 消息,该函数的参数是下述表达式:

GetKeyState(VK_Key)<0。

标识符 VK_Key 表示你想测试的虚拟键。

8.3.4 tolstubar 例程

这个例子充分演示了工具条和状态条的使用方法,如图 8.1 所示,它拥有的三个可浮动工具条,分别是缺省的标准工具条、将图标按钮排成三行的工具面板和一个很有特色的排列样式

工具条。它的特色之处在于由四个排列方式的按钮和一个用来选择右边排列方式按钮的组合框组成。每当用鼠标在组合框的不同项上移动时，右边四个按钮也随之选与不选。工具面板的按钮风格是只有一个按钮处于被选状态。当把鼠标移到这三个工具条按钮上的时候除了在底下的消息框内有解释说明外，还会在按钮旁边出现 Tooltip 提示。

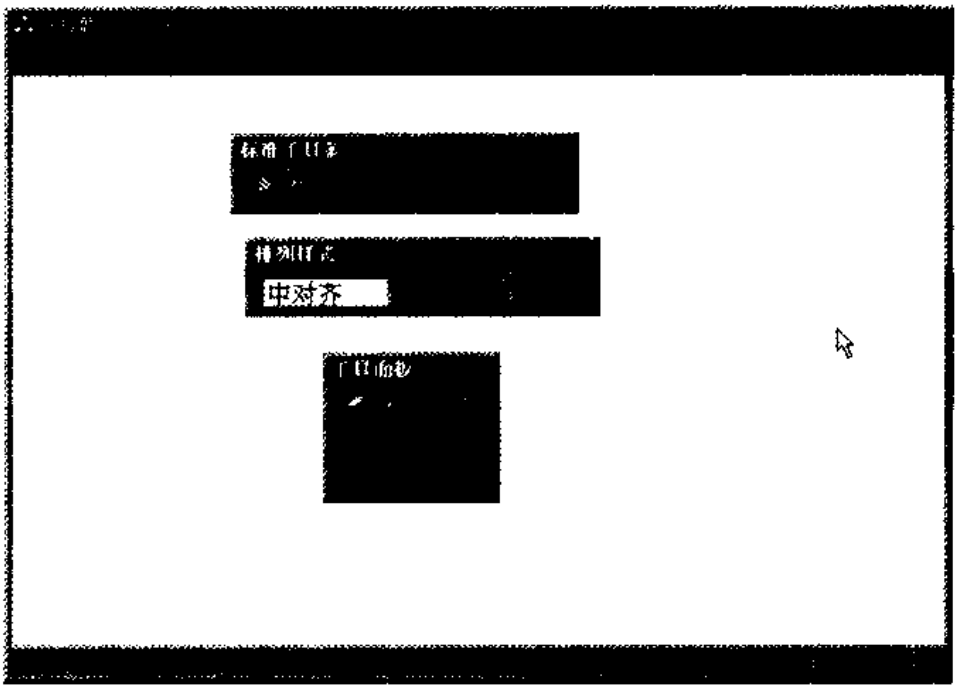


图 8.1　tolstubar 例程图示一

状态条在原来四个格的基础上，还增加了两个格，一个用来显示鼠标在窗口内移动时的 x、y 坐标值，另一个显示此时此刻的时间。

现在我们用 MFC AppWizard 创建基于单文档的应用程序，下面是创建该例子的步骤。

1)启动 Microsoft Developer Studio。

2)选择菜单 File 的 New 菜单项，弹出 New 对话框。

3)在 New 列表中选择 Projects 选项卡，选择 MFC AppWizard(exe)项，在右边的 Project name 中填入 tolstubar。

4)点击 OK，弹出 MFC AppWizard 向导的 MFCAppWizard-stepl 属性页。

5)在应用程序类型中选 Single document。

6)其他接受缺省设置，点击 Finish 按钮，最后用 New Project Information 编辑资源。

1.插入工具条位图

工具条中的每一个按钮似乎都应该有自己的位图，但实际上整个工具条只有一个位图，每一个按钮都在其中占有一个 15 像素高和 16 像素宽的位图片。应用框架为每一个按钮都提供了一个边框并且可以通过按钮的边框及位图片颜色的修改来反映当前按钮的状态。

1)点 Insert 菜单的 Resource 项，在弹出的 Insert Resource 对话框中选 Bitmap，点击 New 按钮。

先看一个我们要插入的两个工具条位图以及其中一个的风格图，如图 8.2 和 8.3 所示。



图 8.2　工具条位置图示



8.3　风格图示

为了最终产生符合要求的位图，必须用风格和每个按钮位图之间的分隔线来作参考。

2）为了做到这一点，只需从 Image 菜单中选择 Grid settings，然后对 Tile Grid 检查框置上检查标志，并接受 16 像素宽和 15 像素高的缺省值，最后生成的风格线会将按钮图像一个一个地分开。

这样产生的空白位图由 3×3 个大的风格位组成。

3）把鼠标指针放在位图最下层的中间的小黑方块上，待鼠标变成上、下箭头形时，按住左键，往上拖，到只存在一行 3 个方格为止。若往下拖则增加位图的行数。

4）把鼠标指针放在位图最右边的中间黑方块上，待鼠标变成左右箭头状时，按住左键，往右拖，直到产生您所需的方格数即可。若往左拖则位图格数减少。

5）照着前面的工具条按钮图，一个一个地把它们画上去。工具面板位图的 ID 取为 IDB_PALETTE，排列样式位图的 ID 为 IDB_STYLE。

提示：*若在别的什么地方能找到您所要的位图，拷贝过来，再粘贴到上面也可以*

2. 增加字符串资源

1）点击 Resource View 切换到资源视窗。

2）点击 tolstubar resources 下面的 String table 节点，双击其下面的 String table 节点，双击其下面的 String table，则在右边打开字符串资源列表。

3）在字符串资源表上任一处单击鼠标右键，在弹出的浮动菜单上，选择 New String 项，弹出字符串属性对话框。

4）在 ID 栏内填写每个位图按钮的 ID 符，Caption 栏填入相关的提示字符串。您可按照下图所示填入每个位图按钮的属性对话框，可能产生的 Value 值不一样，这没关系。增加 ID 标识符。

3. 编辑 MainFrm.h 文件

```
1.// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__6683C9AE_3160_11D2_BA07_444553540000__INCLUDED_)
#define AFX_MAINFRM_H__6683C9AE_3160_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
```

```
class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        BOOL CreateStyleBar();
        BOOL CreatePaletteBar();

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;
        CToolBar    m_wndStyleBar;
        CToolBar    m_wndPaletteBar;
        CComboBox   m_comboBox;
        CFont       m_font;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
```

```
        afx_msg void OnUpdateStyle(CCmdUI * pCmdUI);
        afx_msg void OnChangeStyle(UINT nID);
        afx_msg void OnPalette(UINT nID);
        afx_msg void OnUpdateTime(CCmdUI * pCmdUI);


};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_MAINFRM_H__6683C9AE_3160_11D2_BA07_444553540000__IN-
CLUDED_)
```

编辑 MainFrm.cpp 文件

```
1. // MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "tolstubar.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
        ON_COMMAND_RANGE(ID_STYLE_LEFT, ID_STYLE_JUSTIFIED, OnChangeStyle)
        ON_UPDATE_COMMAND_UI_RANGE(ID_STYLE_LEFT, ID_STYLE_JUSTIFIED, OnUp-
dateStyle)
```

```
    ON_COMMAND_RANGE(ID_PALETTEERASE, ID_PALETTEOVAL, OnPalette)
    ON_UPDATE_COMMAND_UI(ID_INDICATOR_TIME, OnUpdateTime)

END_MESSAGE_MAP()

static UINT BASED_CODE styles[] =
{
    // same order as in the bitmap 'styles.bmp'
    ID_SEPARATOR,           // for combo box (placeholder)
    ID_SEPARATOR,
    ID_STYLE_LEFT,
    ID_STYLE_CENTERED,
    ID_STYLE_RIGHT,
    ID_STYLE_JUSTIFIED,
};

static UINT BASED_CODE palette[] =
{
    // same order as in the bitmap 'palette.bmp'
    ID_PALETTEERASE,
    ID_PALETTEPEN,
    ID_PALETTESELECT,
    ID_PALETTEBRUSH,
    ID_PALETTESPRAY,
    ID_PALETTEPAINT,
    ID_PALETTELINE,
    ID_PALETTEEYEDROP,
    ID_PALETTEMAG,
    ID_PALETTERECT,
    ID_PALETTEROUND,
    ID_PALETTEOVAL
};

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_SEPARATOR,
    ID_SEPARATOR,
//  ID_INDICATOR_TIME,
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
```

```
/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	// TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	if (!m_wndToolBar.Create(this) ||
		!m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
	{
		TRACE0("Failed to create toolbar\n");
		return -1;      // fail to create
	}
	if (!CreatePaletteBar())
		return -1;

	if (!CreateStyleBar())
		return -1;


	if (!m_wndStatusBar.Create(this) ||
		!m_wndStatusBar.SetIndicators(indicators,
		  sizeof(indicators)/sizeof(UINT)))
	{
		TRACE0("Failed to create status bar\n");
		return -1;      // fail to create
	}

	// TODO: Remove this if you don't want tool tips or a resizeable toolbar
	m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
		CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

	m_wndToolBar.SetWindowText(_T("标准工具条"));
```

```
    UINT nID, nStyle;
    int cxWidth;

    m_wndStatusBar.GetPaneInfo( 0, nID, nStyle, cxWidth);
    m_wndStatusBar.SetPaneInfo( 0, nID, SBPS_STRETCH | SBPS_NORMAL, cxWidth);
    m_wndStatusBar.GetPaneInfo( 3, nID, nStyle, cxWidth);
    m_wndStatusBar.SetPaneInfo( 1, ID_INDICATOR_MOUSEPOS, nStyle, cxWidth);
    m_wndStatusBar.SetPaneInfo( 2, ID_INDICATOR_TIME, nStyle, cxWidth*2);
    m_wndStatusBar.SetTimer(1,1000,NULL);

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    EnableDocking(CBRS_ALIGN_ANY);
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar( &m_wndToolBar);
    m_wndPaletteBar.EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar( &m_wndPaletteBar);
    m_wndStyleBar.EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar( &m_wndStyleBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

BOOL CMainFrame::CreatePaletteBar()
{
    if (!m_wndPaletteBar.Create(this, WS_CHILD | WS_VISIBLE | CBRS_SIZE_FIXED
|
          CBRS_FLYBY |CBRS_TOP | CBRS_TOOLTIPS |TBSTYLE_WRAPABLE, IDC_
PALETTE) | |
          | m_wndPaletteBar.LoadBitmap(IDB_PALETTE) | |
          | m_wndPaletteBar.SetButtons(palette,
            sizeof(palette)/sizeof(UINT)))
    {
```

```
        MessageBox("创建错误!");
        return FALSE;        // fail to create
    }

    m_wndPaletteBar.SetWindowText(_T("工具面板"));
    // Create the Palette.  We are using hardcoded numbers for ease here
    // normally the location would be read in from an ini file.

//  m_wndPaletteBar.SetColumns(3);

    int count=m_wndPaletteBar.GetToolBarCtrl().GetButtonCount();
    DWORD dwStyle=m_wndPaletteBar.GetButtonStyle(0);
//  m_wndPaletteBar.SetButtonStyle(0,dwStyle |TBSTYLE_GROUP);
   for(int i=0;i<count;i++)
    {
        m_wndPaletteBar.SetButtonStyle(i,dwStyle |TBSTYLE_CHECK |TBSTYLE_
GROUP);
    }

    CRect rect;
    m_wndPaletteBar.GetToolBarCtrl().SetRows(3,false,&rect);
    RecalcLayout();


    return TRUE;
}

BOOL CMainFrame::CreateStyleBar()
{
    const int nDropHeight = 100;

    if (!m_wndStyleBar.Create(this, WS_CHILD |WS_VISIBLE |CBRS_TOP |
            CBRS_TOOLTIPS |CBRS_FLYBY, IDC_STYLE) ||
        !m_wndStyleBar.LoadBitmap(IDB_STYLE) ||
        !m_wndStyleBar.SetButtons(styles, sizeof(styles)/sizeof(UINT)))
    {
        MessageBox("创建错误!");
        return FALSE;        // fail to create
    }

    // Create the combo box
    m_wndStyleBar.SetButtonInfo(0, IDC_COMBO, TBBS_SEPARATOR, 100);

    // Design guide advises 12 pixel gap between combos and buttons
```

```
        m_wndStyleBar.SetButtonInfo(1, ID_SEPARATOR, TBBS_SEPARATOR, 12);
        CRect rect;
        m_wndStyleBar.GetItemRect(0, &rect);
        rect.top = 3;
        rect.bottom = rect.top + nDropHeight;
        if (! m_comboBox.Create(
                    CBS_DROPDOWNLIST |WS_VISIBLE |WS_TABSTOP,
                    rect, &m_wndStyleBar, IDC_COMBO))
        {
            TRACE0("Failed to create combo-box\n");
            return FALSE;
        }

        //  Fill the combo box
        m_comboBox.AddString(_T("左对齐"));
        m_comboBox.AddString(_T("中对齐"));
      m_comboBox.AddString(_T("右对齐"));
        m_comboBox.AddString(_T("左右对齐"));

        //  Create a font for the combobox

        m_font.Attach(::GetStockObject(SYSTEM_FONT));
        m_comboBox.SetFont(&m_font);

        m_wndStyleBar.SetWindowText(_T("排列样式"));


        return TRUE;
}
void CMainFrame::OnChangeStyle(UINT nID)
{
        m_comboBox.SetCurSel((int)(nID-ID_STYLE_LEFT));
}

void CMainFrame::OnUpdateStyle(CCmdUI* pCmdUI)
{
        UINT nIndex =m_comboBox.GetCurSel();
        pCmdUI->SetCheck((UINT)(ID_STYLE_LEFT+nIndex) == pCmdUI->m_nID);
}
void CMainFrame::OnPalette(UINT nID)
{

}
void CMainFrame::OnUpdateTime(CCmdUI* pCmdUI)
{
```

```
        CTime time = CTime::GetCurrentTime();
        CString str = time.Format("%H:%M:%S");
        pCmdUI->SetText(str);
        pCmdUI->Enable(true);
    }


    /////////////////////////////////////////////////////////////////////////////
    // CMainFrame diagnostics

    #ifdef _DEBUG
    void CMainFrame::AssertValid() const
    {
        CFrameWnd::AssertValid();
    }

    void CMainFrame::Dump(CDumpContext& dc) const
    {
        CFrameWnd::Dump(dc);
    }

    #endif //_DEBUG

    /////////////////////////////////////////////////////////////////////////////
    // CMainFrame message handlers
```

4.增加消息映射函数

打开 ClassWizard,在 Message Maps 页中的 Class Name 中选 CTolstubarView,

在 Object IDS 栏中选 CTolstubarView,Messages 栏中选 WM_MOUSEMOVE,点击 Add Function。再点击 Edit code。

下面是 OnMouseMove 成员函数的代码。

```
void CTolstubarView::OnMouseMove(UINT nFlags, CPoint point)
{
    CString str;
    str.Format("x:%d,y:%d",point.x,point.y);
    CClientDC dc(NULL);
    CSize size=dc.GetTextExtent(str);

    UINT nID, nStyle;
    int cxWidth;

    CStatusBar * status = (CStatusBar *) AfxGetApp()->m_pMainWnd->GetDescen-
```

```
dantWindow(AFX_IDW_STATUS_BAR);
        status->GetPaneInfo( 1, nID, nStyle, cxWidth);
        status->SetPaneInfo( 1, nID, nStyle, size.cx);
        status->SetPaneText(1,str);


        CView::OnMouseMove(nFlags, point);
}
```

8.4 小　　结

本章向读者讲述了在将来读者自己编程中最有可能用到的有关状态栏和工具栏方面的内容。可以通过本章学习,我们应能灵活应用此方面内容。

第9章 文档和视

什么是"视"？它可以理解为"显示"，也就是说，将数据显示出来，可以显示在屏幕上，也可以从打印机中输出，所以视用于连接文档和输出设备。

对于用户来说，它实际上是一个普通的窗口，可以被关闭、移动和改变它的尺寸，就像Windows的其他窗口一样。而对于程序员来说，它是从类库中的类CView派生出来的一个对象，视对象的行为由CView类及其派生类的成员函数和数据成员所决定。文档的概念容易理解，它就是存储数据的载体。可以在文档中对数据进行存盘和调出。

9.1 文档和视概述

MFC的文档/视窗(CDoc/View)架构管理一个单独的数据类型与该数据的多种应用之间的关系。文档即为数据源，其最常见类型是磁盘文件，但它不是唯一的数据源。视窗为用户提供了对数据的可视化显示(在窗口或打印纸上)。MFC的文档/视窗(Doc/View)架构为建立"管理磁盘数据"的应用程序提供了一个框架，应用这种框架的程序包括编辑器、文件浏览器、数据库查询前端等等。

9.1.1 使用文档/视窗结构的意义

文档和视窗最重要的特性是：数据操作与数据显示分离。由于数据操作和数据显示分别按标准方法封装于两个不同类型的对象中，因而，这两项任务的工作量都被大大简化了。各个对象单独存在，而不是将两种代码混在一起。一般地，对那些有大量数据需要管理，或数据显示非常复杂的应用，文档/视窗显得尤为重要。

文档/视窗的另一个重要特性是，许多的MFC类对公共的文档/视窗活动提供了大量的帮助。例如，对文件的操作——选择文件、开启文件和关闭文件——完全由类库框架来管理。用户所需做的仅仅是从数据流中读出所需要的字节。这种工作方式可以省去相同代码的重复编写，而将更多的时间用于编写应用程序的特殊代码上。文档/视窗还有一个特性就是打印预览功能。用户可以在打印之前，在屏幕上看到工作的结果。如果用户自己创建这种功能，那么可能需花几个月的时间编写数千条的语句，而使用MFC提供的功能，将会节约大量的时间和精力。

那么建立MFC应用时，是否一定需要文档/视窗支持呢？回答是否定的。若读者阅读过其他参考书，就会看到许多没有文档/视窗支持建立的MFC应用的例子。尽管这些例子大多数只是简单的程序，但它们表明在Windows用户界面中，并不一定都需要文档/视窗。

9.1.2 SDI和MDI

MFC不仅简化了单文档接口(SDI)，也简化了多文档接口(MDI)的应用程序。

SDI应用程序一次只允许打开一个文档边框窗口，而MDI应用程序允许在应用程序的同一个实例中打开多个文档边框窗口。MDI应用程序有一个主窗口，在其中可以打开多个MDI

子窗口，而这些子窗口本身就是边框窗口，每个窗口中都有一个独立的文档。有些应用程序中，子窗口可以是不同类型的，如视图窗口和数据窗口。在这种情况下，激活不同类型的 MDI 子窗口将改变菜单条。

9.1.3 文档、视和框架

框架的核心是文档和视的概念。一个文档是一个数据对象，用户编辑过程中与这个数据对象进行交互。它是由 File 菜单中的 New 或 Open 命令创建的，一般都保存在一个文件中。一个视是一个窗口对象，用户通过这个窗口对象与一个文档进行交互。

正在运行的应用程序的几个主要对象是：

• 文档

文档类(由 CDocument 派生而来)指定了应用程序的数据。

若想要应用程序中具有 OLE 功能，则文档类要从 COleDocument 或其派生类之中派生出来，这要依据所需功能的类型而定。

• 视

视类(从 CView 派生而来)就是用户的“数据窗口”。视类指定了用户以什么方式看到文档的数据，以及如何与其进行交互。在有些情况下，一个文档可能要有多个视。

若需要滚动，则视类要从 CScrollBar 派生。如果视中有一个以对话框模板资源的形式出现的用户接口，则视类要从 CFormView 派生。对于简单的文本数据，可以使用 CEditView 或从其派生类派生。对于基于格式的数据访问应用程序，如数据输入程序，视类要从 CRecordView 派生。

• 边框窗口

视是显示在“文档边框窗口”中的。在 SDI 应用程序中，文档边框窗口也是该应用程序的“主边框窗口”。在 MDI 应用程序中，文档窗口是显示在主边框窗口中的子窗口，所派生的主边框窗口类指定了包含视的边框窗口的格式及其他特性。要定制 SDI 应用程序的文档边框窗口，就要从 CFrameWnd 派生，而要定制 MDI 应用程序的主边框窗口，就要从 CMDIFrameWnd 派生。若要定制应用程序支持的不同类型的 MDI 文档边框窗口，则每种类型都要从 CMDIChildWnd 派生出一个类来。

• 文档模板

文档模板使得所创建的文档、视和边框窗口和谐地结合起来。一种特定的文档模板类创建并管理着已打开的同种类型的所有文档。支持多种类型文档的应用程序要有多个文档模板。SDI 应用程序要使用 CSingleDocTemplate 类，若是 MDI 应用程序则使用 CMultiDocTemplate 类。

• 应用程序对象

应用程序类(从 CWinApp 派生而来)控制着上述的所有对象，并指定了应用程序的活动，如初始化和清除工作。应用程序的一个且是唯一的一个应用程序对象将创建并管理该应用程序所支持的所有文档类型的文档模板。

• 线程对象

如果应用程序创建了独立的执行线程，例如在后台进行计算，则要使用从 CWinThread 派生的类，CWinApp 本身就是从 CWinThread 派生而来的，表示的是应用程序中的主执行线程(或主进程)。在辅线程也可以使用 MFC。

在运行着的应用程序中,这些对象用命令和其他消息结合在一起,共同对用户动作进行响应。单个应用程序对象管理着一个或多个文档(要看该应用程序是 SDI 还是 MDI)。用户通过包含在边框窗口中的一个视观察和处理文档。

9.1.4 MFC 各对象的关系

为了帮助读者在头脑中显现文档/视的创建过程,先想一想一个运行着的程序都包括什么:文档,用于包含视的边框窗口,以及与文档相关联的视。

- 文档中有该文档的视的列表,以及一个指针指向创建该文档的文档模板。
- 视中有一个指向其文档的指针,视是其父边框窗口的一个子窗口。
- 文档边框窗口有一个指向当前活动视的指针。
- 文档模板有其已打开文档的一个列表。
- 应用程序中有文档模板的一个列表。

Windows 跟踪所有已打开的窗口,以便能够给它们发送消息。

这些关系都是在文档/视的创建期间建立起来的。表 9.1 说明了运行着的程序中的对象怎样才能访问其他对象。通过调用 AfxGetApp 全局函数,任何对象都可获得指向应用程序对象的指针。

表 9.1 获得对应用程序中其他对象的访问权

源对象	如何访问其它对象
文档	用 GetFirstViewPosition 和 GetNextView 访问文档的视列表。调用 GetDocTemplate 获得文档模板。
视	调用 GetDocument 获取文档。 调用 GetParentFrame 获取边框窗口。 文档边框窗口,调用 GetActiveView 获得当前视。 调用 GetActiveDocument 获得当前视中的文档。
MDI 边框窗口	调用 MDIGetActive 获取当前活动的 CMDIChildWnd。

一个边框窗口一般有一个视,但有的时候,如在分割窗口中,同一个边框窗口中含有多个视。边框窗口有一个指向当前活动视的指针,无论什么时候激活了一个新视,该指针都会被更新。

9.2 文档和视的详述

上节对文档和视作了粗浅的介绍,本节将进一步从文档模板以及文档和视的创建和它们之间的相互作用等方面对文档和视作详细的说明。

9.2.1 文档模板

为了管理创建与视和边框相关的文档这一复杂过程,框架使用了两种文档模板类用于 SDI 应用程序的 CSingleDocTemplate 和利用于 MDI 应用程序的 CMultiDocTemplate。CSin-

gleDocTemplate一次可以创建和存储一种类型的文件。CMultiDocTemplate保存一种类型的多个打开文档的一个列表。

有些应用程序能支持多种文档类型。例如,应用程序可以支持文本文档和图形文档。在这样的应用程序中,当用户在File菜单中选择New命令时,对话框中将显示要打开的可能的新文件类型的一个列表。对于所支持的各种文档类型,应用程序将使用不同的文档模板对象。文档模板由应用程序创建和维护。在应用程序InitInstanceIn执行期间所完成的关键任务之一,就是构造一个或多个适当的文档模板。这将在下一小节"文档模板的创建"中讲述。应用程序对象在其模板列表中保存了一个指向其模板列表中各个文档模板的接口。如果要支持两种和多种文档类型,则对每种文档类型都要增加一次调用AddDocTemplate。

9.2.2 文档模板的创建

在响应File菜单中的New或Open命令创建一个新文档时,文档模板也要创建一个新的边框窗口,以便查看文档。

文档模板的构造函数指定了模板可以创建的文档、窗口和视等的类型。这是由传给文档模板构造函数的变量决定的。下面的应用程序代码说明的是创建CMultiDocTemplate的情况:

```
AddDocTemplate(new CMultiDocTemplate(IDR_SCRIBTYPE,
RUNTIME_CLASS(CScribDoc),
RUNTIME_CLASS(CMDIChildWnd),
RUNTIME_CLASS(CSoibView)));
```

指向新的CMultiDocTemplate对象的指针被用作AddDocTemplate的指针。传给CMultiDocTemplate构造函数的变量包括与文档类型的菜单和快捷键相关联的资源ID,以及RUNTIME_CLASS宏的三次引用。RUNTIME_CLASS返回的是以其变量为名称的C++类的RUNTIME_CLASS对象。传给文档模板构造函数的三个CRuntimeclass对象提供了在文档创建过程中创建指定的新对象所需的信息。上面的例子说明的是创建带有CScribview对象的CScribDoc对象所要求创建的文档模板。各个视都由标准的MDI子边框窗口分开。

9.2.3 文档和视的创建

框架提供了在File菜单中实现New和Open命令(还有其它命令)的方法。一个新文档及与其相关的视和边框窗口的创建工作,是应用程序对象、文档模板、新创建的文档和新创建的边框窗口等共同作用的结果。表9.2总结出了哪个对象创建了哪些内容。

表9.2 对象和所创建的内容

创建者	所创建的东西
应用程序对象	文档模板
文档模板	文档
文档模板	边框窗口
边框窗口	视

9.2.4 文档和视之间的相互作用函数

我们已经了解到,文档对象是用来保存数据的,而视对象则是用来显示数据的,并且允许

对数据进行编辑。SDI 应用的文档类是由 CDocument 类派生出来的，它可以有一个或多个视类，而这些视类最终都是由 CView 类派生的。在文档、视以及应用框架的其余部分之间包含了一系列非常复杂的相互作用过程，为了了解这一过程，我们必须首先学习有关文档类和视类的四个非常重要的成员函数，其中两个为虚函数，我们经常需要在派生类中对它们进行重新设计，而其余两个则不是虚函数，我们需要在派生类中对它们进行调用。下面让我们共同学习这些函数。

1. Cview 类的 GetDocument 函数

视对象只有一个与之相联系的文档对象，它所包含的 GetDocument 函数允许应用在视中得到与之相联系的文档。假如视对象接收到了一条消息，表示用户在编辑控制中输入了新的数据，此时，视就必须通知文档对象对其内部数据进行相应的更新。GetDocument 函数返回的是指向文档的指针，利用它我们就可以对文档成员函数公共数据成员进行访问。

CDocument 类的 GetNextView 函数是用来在文档中得到视，但由于文档可以有不止一个视，因此，有必要进行一个循环，对每个视都调用一次该函数。GetDocument 函数比 GetNewView 函数更常用一些。

当 AppWizard 产生 CView 的派生类时，它同时也创建一个保证类型的 GetDocument 函数，它返回的并不是指向 CDocument 类的指针，而是指向派生文档类的指针，该函数是一个内联函数，它类似于如下形式：

```
CMyDoc * GetDocument()
{return (CMyDoc * m_pDocment);}
```

当编译器在视类代码中遇到对 GetDocument 函数的调用时，它调用的实际上是派生视类中的该函数，而非 CView 类中的该函数，因此，我们没有必要将它所返回的指针强制转换为指向派生文档类的指针。然而，由于 CView 的 GetDocument 函数并不是虚函数，因此，下列的语句

```
pView→GetDocument();//pView is declared CView *
```

调用的是基类中的 GetDocument 函数，它所返回的是指向 CDocument 对象的指针

2. CDocument 类的 UpDateAllViews 函数

如果由于种种原因文档数据发生了改变，那么所有的视都必须被通知到，以使它们能够对所显示的数据进行相应的更新。若在派生文档类的成员函数中调用 UpDateAllView 函数，那么它的第一个参数 pSender 应为 NULL，如果 UpDateAllView 函数是在派生视类的成员函数中被调用的话，应该以如下方式将 pSender 参数设置为当前视。

```
GetDocument()→UpdateAllViews(this);
```

其中非空参数就使得应用框架不再通知当前的视，因为我们已假定当前视已经自己进行了更新。该函数还有一些提示参数，可以利用它们给视提供一些特殊的与应用有关的消息，以便视能够决定究竟哪些部分应该更新，这是对该函数的最高级的用法。

当 UpdateAllViews 函数被调用时，视具体是如何被通知的呢？请详细学习下面的 OnUpdate 函数。

3. CView 类的 OnUpdate 函数

这是一个虚函数，当应用调用了 CDocument 类的 UpdateAllView 函数时，应用框架就会相应地调用该函数，当然，我们也可以直接在派生类中调用它。通常派生视类的 OnUpdate 函数要对文档进行访问，读取文档的数据，然后再对视的数据成员或控制进行更新，以便反映出文档的变化。另外，还可以用 OnUpdate 函数体现的某部分无效以便能接收视的 OnDraw 函数，利用文档数据来重新对窗口进行绘制。OnUpdate 函数类似于下面的形式：

```
Void CanyView::OnOpdate(CView * psender, LPARAM LHint,
CObject * prlint)
{CMyDocument * pMyDoc = GetDocument();
CString last _ Name = pMyDoc→GetlastName();
m _ pNameStatic→SetWirdowText(last _ Name);//mpName Static
// is a CmyView date member
}
```

其中的提示信息是由 UpdateAllViews 函数直接传递过来的。缺省的 OnUpdate 函数使得整个窗口矩形无效，在重新设计该函数时，我们可以利用提示信息来定义一个较小的无效矩形。

当 CDocument 的 UpdateAllViews 函数被调用时，如果 Psender 参数指向了某个特定的视对象，那么除了该指定的视之外，文档的所有其他视的 OnUpdate 函数都会被调用。

4. CView 的 OnInitialUpdate 函数

当应用被启动时，或当用户从 File 菜单选择了 New 时，或当用户从 File 菜单中选择了 Open 时，CView 的该虚函数都会被自动调用。CView 基类中的 OnInitialUpdate 函数除了调用了 OnUpdate 函数之外，没做其他任何事情，若我们要在派生视类中重新设计 OnInitialUpdate 函数，一定要在其中调用基类的 OnInitialUpdate 函数，或用调用派生类的 OnUpdate 函数。

我们也可利用派生类的 OnInitialUpdate 函数对视对象进行初始化。当应用被启动时，应用框架在调用了 OnCreate 函数之后（如果我们对 OnCreate 函数进行了映射），会立即调用 OnInitialUpdate 函数。OnCreate 函数只能被调用一次。而 OnInitialUpdate 函数则可被调用许多次。

9.3 实例 Slg 分析

该例子演示了单文档多视的使用，如图 9.1 和 9.2 所示。菜单条上多加了一菜单项“改变视(C)”，当我们点击它时，视窗内的视由 Cslgview 切换到 Cform 视，再单击，视 Cform 又变为 cslgview，这样不断循环切换视类，以显示单文档多视的功能。

下面简要介绍如何创建该例程。

1)创建名为 slg 的单文档工程。

2)在主菜单(IDR _ MAINFRAME)最后位置，增加菜单项“改变视(C)”(IDC CHANGEVIEW)，Prompt(提示符)为“改变视 \ 切换视”

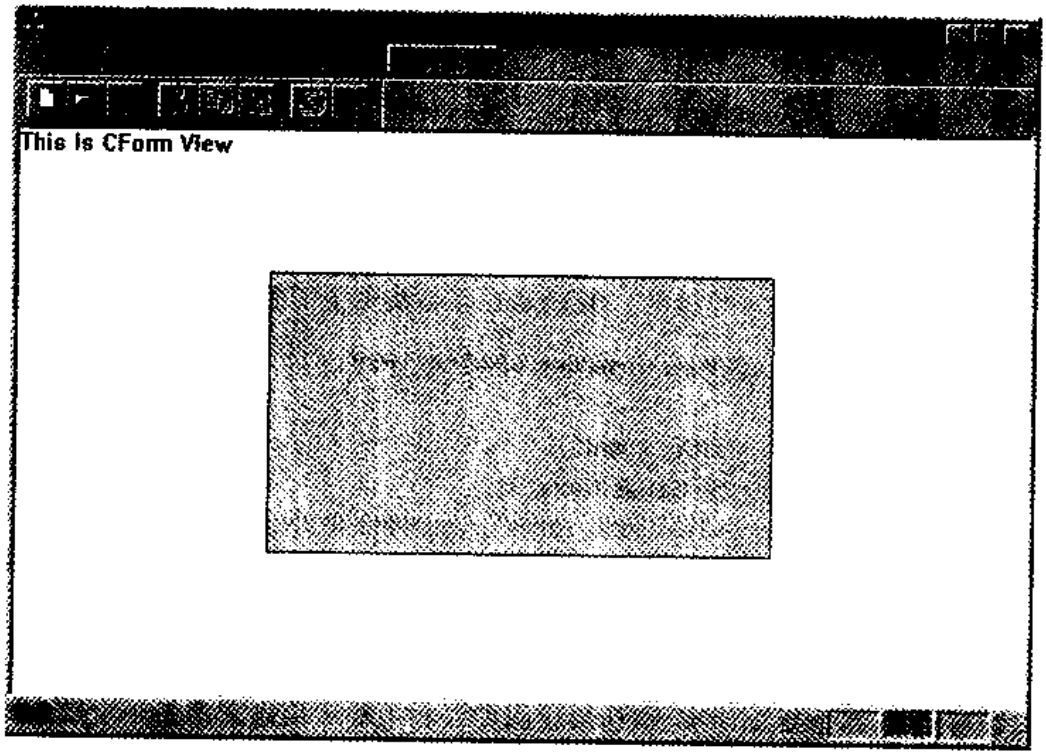


图 9.1　slg 图例 1

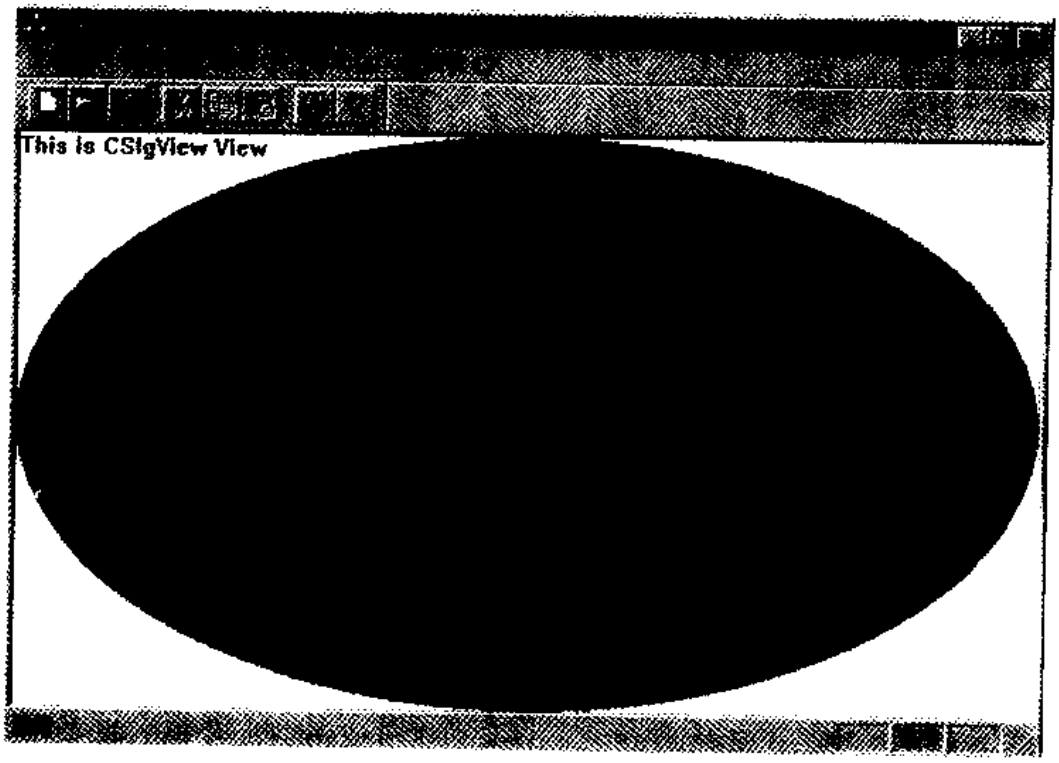


图 9.2　slg 图例 2

3)单独创建类 Cform,其父类为 Cview,并编辑文件 Form.h 和 Form.cpp。

4)编辑文件 MainFm.h,MainFm.cpp,slgDoc.h,slgDoc.cpp 和 slgView.h,slgView.cpp。

```
#include "sIgDoc.h"

#if !defined(AFX_FORM_H__9C969A2B_1DB4_11D2_BA07_444553540000__INCLUDED_)
#define AFX_FORM_H__9C969A2B_1DB4_11D2_BA07_444553540000__INCLUDED

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Form.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CForm view

class CForm : public CView
{
protected:
	DECLARE_DYNCREATE(CForm)

// Attributes
public:
	CForm();           //   constructor used by dynamic creation

// Operations
public:
	CSlgDoc* GetDocument();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CForm)
	protected:
	virtual void OnDraw(CDC* pDC);      // overridden to draw this view
	//}}AFX_VIRTUAL

// Implementation
protected:
	virtual ~CForm();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

	// Generated message map functions
protected:
```

```
    //{{AFX_MSG(CForm)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FORM_H__9C969A2B_1DB4_11D2_BA07_444553540000__INCLUDED_)

// Form.cpp : implementation file
//

#include "stdafx.h"
#include "sig.h"
#include "Form.h"
#include "sigDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CForm

IMPLEMENT_DYNCREATE(CForm, CView)

CForm::CForm()
{
}

CForm::~CForm()
{
}


BEGIN_MESSAGE_MAP(CForm, CView)
```

```
    //{{AFX_MSG_MAP(CForm)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CForm drawing

void CForm::OnDraw(CDC* pDC)
{
    CSlgDoc* pDoc = GetDocument();
    pDoc->str="This is CForm View";
    CRect rect;
    GetClientRect(&rect);
    CRect Rect(rect);
    rect.left=Rect.Width()/4;
    rect.top=Rect.Height()/4;
    rect.right=Rect.Width()*3/4;
    rect.bottom=Rect.Height()*3/4;
    CBrush brush(RGB(0,255,0));
    CBrush *pOldbrush;
    pOldbrush=pDC->SelectObject(&brush);
    pDC->Rectangle(rect);
    pDC->TextOut(0,0,pDoc->str);
    pDC->SelectObject(pOldbrush);
}

/////////////////////////////////////////////////////////////////////////////
// CForm diagnostics

#ifdef _DEBUG
void CForm::AssertValid() const
{
    CView::AssertValid();
}

void CForm::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}
CSlgDoc* CForm::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CSlgDoc)));
    return (CSlgDoc*)m_pDocument;
```

```
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CForm message handlers
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////
#include "Form.h"
#include "slgView.h"

#if !defined(AFX_MAINFRM_H__9C969A1F_1DB4_11D2_BA07_444553540000__INCLUDED
_)
#define AFX_MAINFRM_H__9C969A1F_1DB4_11D2_BA07_444553540000__INCLUDED

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
```

```
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;
        CForm* pFormView;
        CSlgView* pSlgView;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnChangeview();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_MAINFRM_H__9C969A1F_1DB4_11D2_BA07_444553540000__INCLUDED_)
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "slg.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
```

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
	ON_WM_CREATE()
	ON_COMMAND(IDC_CHANGEVIEW, OnChangeview)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
	ID_SEPARATOR,           // status line indicator
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	pFormView=NULL;
	pSlgView=NULL;


}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	if (!m_wndToolBar.Create(this) ||
		!m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
	{
		TRACE0("Failed to create toolbar\n");
		return -1;      // fail to create
	}

	if (!m_wndStatusBar.Create(this) ||
		!m_wndStatusBar.SetIndicators(indicators,
```

```
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar \n");
        return -1;      // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizeable toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
```

```
// CMainFrame message handlers

void CMainFrame::OnChangeview()
{
	CDocument * pDoc = GetActiveDocument();
	CView * pView;
	CView * pViewAdd;
	CView * pViewRemove;
	pView = GetActiveView();
	if(! pView->IsKindOf(RUNTIME_CLASS(CForm)))
	{
		if(pFormView == NULL)
		{
			pSlgView = (CSlgView * )GetActiveView();
			pFormView = new CForm;
			pFormView->Create(NULL,NULL,AFX_WS_DEFAULT_VIEW,
				rectDefault,this,ID_FORM_VIEW,NULL);
		}
		pViewAdd = pFormView;
		pViewRemove = pSlgView;
	}
	else
	{
		pViewAdd = pSlgView;
		pViewRemove = pFormView;
	}
	int nSwitchID = pViewAdd->GetDlgCtrlID();
	pViewAdd->SetDlgCtrlID(AFX_IDW_PANE_FIRST);
	pViewRemove->SetDlgCtrlID(nSwitchID);
	pViewAdd->ShowWindow(SW_SHOW);
	pViewRemove->ShowWindow(SW_HIDE);
	pDoc->AddView(pViewAdd);
	pDoc->RemoveView(pViewRemove);
	SetActiveView(pViewAdd);
	RecalcLayout();


}
// slgDoc.h : interface of the CSlgDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if ! defined(AFX_SLGDOC_H__9C969A21_1DB4_11D2_BA07_444553540000__INCLUDED_)
#define AFX_SLGDOC_H__9C969A21_1DB4_11D2_BA07_444553540000__INCLUDED
```

```
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


class CSlgDoc : public CDocument
{
protected: // create from serialization only
	CSlgDoc();
	DECLARE_DYNCREATE(CSlgDoc)

// Attributes
public:
	CString str;

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSlgDoc)
	public:
	virtual BOOL OnNewDocument();
	virtual void Serialize(CArchive& ar);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CSlgDoc();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
	//{{AFX_MSG(CSlgDoc)
		// 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()
```

```
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SLGDOC_H__9C969A21_1DB4_11D2_BA07_444553540000__INCLUDED_)
// slgView.h : interface of the CSlgView class
//
/////////////////////////////////////////////////////////////////////////////
#include "slgDoc.h"

#if !defined(AFX_SLGVIEW_H__9C969A23_1DB4_11D2_BA07_444553540000__INCLUDED_)
#define AFX_SLGVIEW_H__9C969A23_1DB4_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CSlgView : public CView
{
protected: // create from serialization only
	CSlgView();
	DECLARE_DYNCREATE(CSlgView)

// Attributes
public:
	CSlgDoc* GetDocument();

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSlgView)
	public:
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	protected:
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
```

```
        virtual void OnBeginPrinting(CDC * pDC, CPrintInfo * pInfo);
        virtual void OnEndPrinting(CDC * pDC, CPrintInfo * pInfo);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CSlgView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CSlgView)
                // 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()
};

#ifndef _DEBUG  // debug version in slgView.cpp
inline CSlgDoc * CSlgView::GetDocument()
   { return (CSlgDoc *)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_SLGVIEW_H__9C969A23_1DB4_11D2_BA07_444553540000__INCLUDED_)
// slgView.cpp : implementation of the CSlgView class
//

#include "stdafx.h"
#include "slg.h"

#include "slgDoc.h"
#include "slgView.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSlgView

IMPLEMENT_DYNCREATE(CSlgView, CView)

BEGIN_MESSAGE_MAP(CSlgView, CView)
    //{{AFX_MSG_MAP(CSlgView)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSlgView construction/destruction

CSlgView::CSlgView()
{

}

CSlgView::~CSlgView()
{
}

BOOL CSlgView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
```

```
// CSlgView drawing

void CSlgView::OnDraw(CDC * pDC)
{
        CSlgDoc * pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        pDoc->str = "This is CSlgView View";
        CBrush brush(RGB(255,0,0));
        CBrush * pOldbrush = pDC->SelectObject(&brush);
      CRect rect;
        GetClientRect(&rect);
        pDC->Ellipse(rect);
        pDC->TextOut(0,0,pDoc->str);
        pDC->SelectObject(pOldbrush);

}

/////////////////////////////////////////////////////////////////////////////
// CSlgView printing

BOOL CSlgView::OnPreparePrinting(CPrintInfo * pInfo)
{
        // default preparation
        return DoPreparePrinting(pInfo);
}

void CSlgView::OnBeginPrinting(CDC * /* pDC */, CPrintInfo * /* pInfo */)
{
        // TODO: add extra initialization before printing
}

void CSlgView::OnEndPrinting(CDC * /* pDC */, CPrintInfo * /* pInfo */)
{
        // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CSlgView diagnostics

#ifdef _DEBUG
void CSlgView::AssertValid() const
{
        CView::AssertValid();
}
```

```
void CSlgView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CSlgDoc* CSlgView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CSlgDoc)));
    return (CSlgDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CSlgView message handlers
```

9.4 小　　结

本章向读者介绍了视和文档的基本内容，通过最后一个例子，读者可以对这方面内容有一定了解。读者可以在以后的编程中使用这方面内容以便对此有更深刻的了解。

第 10 章　单文档界面

似乎把 SDI(Single Document Interface)译作单文档接口更适合我们的习惯。从文字意义上来看，单文档是指整个程序中只有一个文档，即只有一个 CDocument 类或 CDocument 的派生类。在前面已提到过，文档可有多个视与之联系在一起，故单文档程序有单视与多视之分，为 SDI 文档提供多视比较复杂，可以提供相应的菜单项供用户选择视，也可将多视安排在切分窗口中，切分窗口将在本书的第 12 章介绍。本章第一节介绍对文档的数据进行存取的机制——序列化，接下来的两节对 SDI 进行介绍并说明如何创建 SDI 应用程序。

10.1　序　列　化

"序列化"是从永久存储媒介中(如磁盘文件)写或读对象的一个过程。MFC 提供 Cobject 类中序列化的内在支持。这种所有派生自 CObject 的类都能充分利用 CObject 序列化协议。

MFC 使用 CArchive 类的对象作为要序列化对象和存储媒介之间的中介。此对象始终与 CFile 对象有关，从中它可以获得有关序列化的必要信息，如创建文件名和要求的操作是否读还是写。执行序列化操作的对象可以使用 CArchive 对象而不管存储媒介的本质。CArchive 对象使用重载的插入(>>)和抽取(<<)操作符执行写和读操作。

10.1.1　创建可序列化的类

使类可序列化需要五个步骤：

- 从 CObject(或从由 CObject 派生而来的一些类)派生用户的类。
- 在类声明中使用 DECLARE_SERIAL 宏。
- 加载 Serialize 成员函数。
- 定义不用变量的构造函数。
- 在实施类文件中使用 IMPLEMENT_SERIAL 宏。

如果直接调用 Serialize 而不是调用 CArchive 的">>"和"<<"操作符，那么后三步对序列化不要求。

1)从 Cobject 派生类并使用 DECLARE_SERIAL 宏。

基本的序列化协定和功能在 Cobject 类中定义。通过从 Cobject(或从由 CObject 派生的类)中派生，如下面 CPerson 类声明中所示，可以获得到 CObject 序列化协定和功能的访问。

DECLARE_SERIAL 宏在支持序列化的类声明中的使用如下所示：

```
Class CPerson:public cobject
{ DECLARE_SERIAL(CPerson)
// rest of declaration follows:
};
```

2)加载 Serialize 成员函数。

在 CObject 类中定义的 Serialize 成员函数对抓取对象当前日期必要的实际数据序列负责。Serialize 函数有一个它用来读和写对象数据的 CArchive 变量。CArchive 对象有一个成员函数 IsStoring 指示 Serialize 是在存储(写数据)还是加载(读数据)。使用 IsStoring 结果作为指导，要么用插入符(>>)在 Archive 对象中插入对象数据，要么用抽取符(<<)抽取数据。假设一个从 Cobject 派生来的类具有 CString 和 WORD 类型的两个新成份变量，下面的类声明中断显示了新的成份变量和已加载 Serialize 成员函数的声明。

```
class Cperson:public CObject
{ public:
DECLARE_SERIAL(CPerson);
// empty constructor is necessary
CPerson(){};
CString m_name;
WORD m_number;
void Serialize(CArchive & archive);
// rest of class deelaration
};
```

序列加载成员函数的方法如下所示：

(1)调用基础类版的 Serialize 以保证继承的对象部分序列。

(2)插入或抽取类特定的成份变量。

插入和抽取操作符与类相互作用以读和写数据。下面的例子显示实现上面声明的 Cperson 类的 Serialize。

```
void CPerson::Serialize(CAchive &archive)
{ // call base class function first
// base class is cobject in this cast
CObject::Serialize (archive);
// now do the stuff for our specific class
if(archive.IsStoring())
archive<<m_name<<m_number;
else
archive>>m_name>>m_number;
}
```

还可以使用 CArchive::Read 和 CArchive::Write 成员函数读和写大量的无类型数据。

3)定义无变量的构造函数。

MFC 在它重新创建对象(由于它们解除序列或即将从磁盘加载)时要求一些缺省的构造函数。解除序列过程将用重新建立对象所要求的数值填充所有的成员变量。此构造函数可以分开声明、保护或专有。如果使它成为保护或专有的，那么要保证它将只由序列函数使用。构造函数必须置对象于允许它在必要时能安全删除的状态。

4)IMPLEMENT_SERIAL 宏用于定义在从 CObject 派生可串行类时需要的各种函数。在实现文件(.cpp)中为用户的类使用此宏。宏的前两个变量是类名和其直接基础类名。

该宏的第三个变量是模式编号。模式编号实际上是类对象的版本号。模式编号使用大于

或等于 0 的整数。

下面的例子显示使用类的 Cperson(从 CObject 派生而来),IMDLEMENT _ SERIAL 的方法。

IMPLEMENT _ SERIAL(CPerson,CObject,1)

一旦具有可序列化的类,就能序列化该类的对象。

10.1.2 序列化对象

一旦具有可序列化的类时,就能通过 CArchive 对象访问从外部文件中串行那个类的对象。可以让框架为可序列化变量建立归档或自己建立 CArchive 对象。可通过使用"<<"和">>"操作符为 CArchive 在文件和可序列化对象之间传输数据,某些情况下还可通过调用 CObject 派生类的 Serialize 函数实现。

1.什么是 CArchive 对象

CArchive 对象为从或向 CFile 对象读或写可序列化对象提供类型安全的缓冲机制。通常 CFile 对象代表磁盘文件,但它还可以是内存文件(CMemFile 对象),亦可代表 Clipboard。虽然所给 CArchive 对象要么存储(写序列化)数据要么加载(读序列化)数据,但决不会两者同时进行。CArchive 对象的生命局限于向文件写对象或从文件读对象的一次经过。因此,两个连续建立的 CArchive 对象要求先文件序列化数据,然后从文件中解除序列化。

在数据序列化到归档时,归档累计数据直到其缓冲器满。然后归档把其缓冲器写到由 CArchive 对象指向的 CFile 对象。同样,当从归档读取数据时,它从文件读取数据到其缓冲器,然后从缓冲器到解除序列化的对象。此缓冲减少硬盘实际被读的次数,这样能改进应用程序的性能。

2.创建 CArchive 对象的两种途径

创建 CArchive 对象有两种方法:通过框架隐式地创建 CArchive 对象及显式地创建 CArchive 对象。

(1)通过框架隐式地创建 CArchive 对象

最通常也是最容易的方法是让框架为文档创建代表 File 菜单上 Save、SaveAs 和 Open 命令的 CArchive 对象。

下面是应用程序用户从 File 菜单执行 Save As 命令时框架进行的操作:

- 展现 File SaveAs 对话框,从用户取得文件名。
- 作为 CFile 对象打开由用户命名的文件。
- 创建指向此 CFile 对象的 CArchive 对象。在创建 CArchive 对象中,框架设置模式为"Store"(写,创建序列),与"load"(读,解除序列)相反。
- 调用 CDocument 派生类中定义的 Serialize 函数,按 CArchive 对象的参数传递它。

如前所述先调用文档的 Serialize 函数,然后把数据写到 CArchive 对象。Serialize 函数返回之际,框架销毁 CArchive 对象和 CFile 对象。因此,若框架为文档创建 CArchive 对象,所需要做的是实施文档的 Serialize 函数从旧文档中写和读数据。另外还得对文档中的的任何 CObject 派生对象实施 Serialize。

(2)显示创建 CArchive 对象

除了通过框架串行文档之外,还有可能需要 CArchive 对象的其他场合。如也许想要使用

用户界面保存不同于框架提供的文件。在这种情况下可以显式地创建 CArchive 对象，按与框架相同的方法执行，过程如下所示。

① 显式地创建 CArchive 对象

- 构造 CFile 对象中从 CFile 派生的对象。
- 为 CArchive 构造函数传递 CFile 对象，如下所示：

```
CFile theFile;
theFile.open(…,CFile::modewrite);
CArchive archive(&theFile,CArchive::store);
```

CArchive 构造函数的第三个参数是个枚举值，指明归档是否从或向文件加载还是保护数据。对象的 Serialize 函数通过调用归档对象的 IsStoring 函数检查此状况。

结束从或向 CArchive 对象存储或加载数据时，则关闭它。尽管 CArchive(和 CFile)对象会自动关闭归档(和文件)，但最好这样做，因为它可以更容易地从错误中恢复。要了解错误处理的更多信息，可参见有关资料。

② 关闭 CArchive 对象

下面的例子演示了关闭 CArchive 对象的方法。

```
archive.close();
theFile.Close();
```

使用 CArchive 的"<<"和">>"操作符

CArchive 不仅为写和读简单数据类型，也为从和向文件的 Cobject 提供">>"和"<<"操作符。

③ 通过归档存储对象到文件中

下面的例子显示了通过归档在文件中保存对象的方法。

```
CArchive archive(&theFile,CArchive::store);
WORD WEmployeeID;
…
archive << WEmployeeID;
```

④ 装入曾放在文件中的某个值的对象

下面的例子显示了从原先存储在文件中的值加载对象的方法。

```
CArchive archvie(& the File,CArchive::load);
WORD WEmployeeID;
archive << WEmployeeID;
```

10.2 SDI 简介

读者或许已经注意到了，AppWizard 所产生的每一个程序都包含有 File 菜单，其中包含了一些我们早已熟知的 New，Open，Save 和 Save As 命令。在本节，我们将向读者介绍如何使应用响应这些命令对文档进行读和写。

10.2.1 创建空文档

应用类的 InitInstance 函数在调用 AddDocTemplate 函数之外，会调用另一个非常重要的

CWinApp 的成员函数 OnFileNew，而它则会通过调用另外一个 CWinApp 函数 OpenDocumentFile，在相互连接的类中进行搜索，同时还要完成以下工作：

1）构造文档对象，但此时并不从磁盘中读数据。

2）构造主框架对象（即类 CMainFrame 的对象），并且还要创建主框架窗口，但并不是对它进行显示。主框架窗口中包括 IDR-MAINFRAME 菜单，工具条和状态条。

3）构造视对象，并且还创建视窗口，但并不对它进行显示。

4）创建文档、主框架和视对象之间的相互联系，不要把这些对象间的联系同通过调用 AddDocTemplate 所建立的类之间的联系混淆起来。

5）调用文档对象的 OnNewDocument 虚成员函数，它会调用 PeleteContents 虚函数。

6）调用视对象的 OnInitialUpdate 虚成员函数。

7）调用框架对象的 ActiveFrame 虚成员函数，以便显示出具有菜单视窗口和控制条的主框架窗口。

说明：上面所列出的一些函数并不直接被 OpenDocumentFile 函数所调用，而是通过应用框架被间接地进行了调用。

在 SDI 应用中，文档、主框架以及视对象都仅被创建一次，并且将存在于整个程序的运行过程中。CWinApp 的 OnFileNew 函数被 InitInstance 函数所调用，当用户选择了 File New 菜单项时，它也会被调用，然而在这种情况下，OnFileNew 函数必须采用不同的行动。它用不着再构造文档、框架及视对象了，因为它们都已经存在，但它要重新利用现存的文档对象完成上述的最后三个步骤。

注意：OnFileView 函数总是要调用 DeleteContents 函数，以便将文档关闭、清空。如果 SDI 应用不重新使用相同的文档对象，我们也就没有必要使用 OnNewDocument 函数，因为我们可以在文档类的构造函数中完成所有的文档初始化工作。现在，我们必须对 OnNewDocument 函数进行重新设计，以便每次当用户选择了 FileNew 或 FileOpen 时，都能利用它来初始化文档对象。AppWizard 会在它所产生的派生文档类中给出该函数的框架。

10.2.2 打开文件

当 AppWizard 创建应用时，它会自动将 FileOpen 菜单项映射到 CWinApp 的 OnFileOpen 成员函数上，它会通过进一步调用 CWinApp 的 OpenDocumentFile 函数来完成以下工作：

1）调用已经存在的文档对象的 OnOpenDocument 虚成员函数，该函数会提醒用户选择一个文件，然后再将其打开，再创建一个用于装入数据的 CArchive 对象，然后再调用 DeleteContents 函数。

2）调用文档的 Serialize 函数，它负责从旧文档中装入数据。

3）调用视的 OnInitialUpdate 函数。

说明：上述的一些函数并不是直接被 OpenDocumentFile 所调用，而是通过应用框架被间接进行了调用。

除了使用 FileOpen 菜单项外，我们还可以选择最近才使用的（Most Recently Used，简称 MRU）文件列表。应用框架会记录下四个最近才使用的文件，并将它们的名字显示在 File 菜单中，这些文件名是在程序的运行过程中被记录在应用的 INI 文件中的。

10.2.3 删除文档内容

当从磁盘文件中将数据装入已经存在的文档对象中时，无论如何，必须首先将文档对象中的现存内容清除掉，完成这一工作的最好的办法就是对派生文档类中的 CDocument 的 DeleteContents 虚成员函数进行重新设计。对文档类数据成员的一些必要的清除工作全都是在被重新设计的该函数中完成的。当 FileNew 或 FileOpen 菜单项被选中时，CDocument 的 OnFileNew 或 OnFileOpen 函数都要调用 DeleteContents 函数，也就是说，当文档对象第一次被创建之后，DeleteContents 函数立刻就会被调用，当文档被关闭时，它就被再次调用。如果希望文档类适用于 SDI 应用的话，最好将文档内容的清除工作放在 DeleteContents 成员函数中来完成，而不要将其放入析构函数中，析构函数仅被用来清除那些在对象存在期间都将存在的数据项。

10.2.4 保存文件

当 AppWizard 创建应用时，它会将 FileSave 菜单项映射到 CDocument 类的 OnFileSave 成员函数，OnFileSave 会调用 CDocument 的 OnSaveDocument 函数，它会用供存入使用的旧档对象来调用文档的 Serialize 函数。FileSaveAs 菜单项也是以类似的方式被控制的，它映射到 CDocument 类的 OnFileSave 函数，它会调用 OnSaveDocument 函数。在这里，所有将文档存盘的必要的文件管理工作都是由应用框架来完成的。

说明：的确，FileNew 和 FileOpen 菜单选择被映射到了应用类成员函数，而 File，Save 和 FileSaveAs 则被映射到了文档类成员函数。FileNew 被映射到了 OnFileNew，SDI 应用中的 InitInstance 函数调用的也是 OnFileNew。当应用框架调用 InitInstance 函数时，文档对象都还未被创建，因此，OnFileNew 不可能是 CDocument 的成员函数，而当文档被存盘时，文档对象则肯定已经存在。

Windows 的许多面向文档的应用都会跟踪用户对文档的修改，当用户试图关闭文档或者退出程序时，应用就会弹出一消息框，询问用户是否需要将文档保存。类库应用框架通过 CDocument 的数据成员 m-bModified 直接支持这种功能，若文档被修改了那么该布尔变量值为 TRUE，否则为 FALSE。

protected 型的 m_bModified 标记是通过 CDocument 的 SetModifiedFlag 和 IsModified 成员函数被访问的，当文档创建时，或者从磁盘中读出时，或者存盘时，文档的该标记就被置为 FALSE，而当文档数据被修改时，必须通过使用 SetModifiedFlag 函数将该标记置为 TRUE。

在 Visual 工作平台中，工具条中的磁盘按钮对应于 FileSave 菜单项，在当前被选中的文档未被修改时，它就会被禁止。

10.3 小　　结

单文档界面(SDI)在可视化编程中必不可少，读者应当掌握并能做到灵活应用，本章就这方面内容向读者论述了以下内容。

- 介绍了序列化的有关概念和应用
- 介绍了创建一个 SDI 应用程序的过程
- 有无实例辅助

第 11 章　多文档界面

前一章我们介绍了单文档界面的运用，这一章要介绍与之相对的多文档界面 MDI(Multiple Document Interface)。它是 Windows 的标准接口之一。现在使用的 Windows 应用工具很多是基于这种接口的，如 Visual Studio、Office 套件等。

11.1　MDI 简介

11.1.1　MDI 应用程序的组成

一个 MDI 应用程序由下列对象组成：

- 可见的 MDI 框架窗口，它控制着其他对象。MDI 框架窗口是 CMDIFrameWnd 或其派生类的一个实例。每个 MDI 应用程序只有一个 MDI 框架窗口。
- 不可见的 MDI 客户窗口，它支持对下属 MDI 子窗口的管理，这些子窗口是动态生成和消失的。每个 MDI 应用程序只能有一个 MDI 客户窗口。
- 可见的 MDI 子窗口，MDI 应用程序会动态地创建和删除 MDI 子窗口的多重实例。一个 MDI 子窗口是 CMDIChildWnd 类或其派生类的一个实例。在 MDI 框架窗口规定的区域内，MDI 应用程序可支持对子窗口的定位、重置大小(resizing)、移动、最小化和最大化。和非 MDI 窗口一样，任一时刻只有一个子窗口是活动的。

最大化后的子窗口拥有 MDI 客户窗口的空间，最小化后的子窗口是个图标(icon)，出现在 MDI 框架窗口的底部。

注意：MDI 框架窗口有一个菜单，通过它可对 MDI 子窗口和子窗口中的内容进行操作。MDI 子窗口不能有菜单，但可以有控件。当用户选择另一子窗口时，可切换至框架窗口，MFC 类库可以支持这一功能，该功能意味着可允许 MDI 子窗口有不同的使用(use)和数据。

11.1.2　主框架和文档模板

MDI 应用包含了两个不同的字符串和菜单资源，分别是由常量 IDR_MAINFRAME 和 IDR_MYDOCTYPE 来标识的。第一个资源设置是用于空主框架窗口的；第二个则是用于被占用的主框架窗口的。下面所示的就是这两种字符串，它们被分成了若干子字符串。

```
IDR_MAINFRAME
"MYAPP Windows Application" // application window caption
IDR_MYDOCTYPE
"Inn" // application window cpation
"MYApp/n" // root for default document name
"MYApp Document/n" // document type name"MYApp File(*.171)In" // document type description and filter
```

".17aln" // extention for documents of this type
"MYAppFileType ln" // registration database document ID
"MYAppFileType" // registration date base document description

说明:资源编译器并不接受以上的字符串连接方式,这些子字符串可被结合进一个大的字符串中。

应用窗口的标题来自 IDR _ MAINFRAME 字符串,当文档被打开时,文档的名字又被追加在标题的后面。IDR _ MYDOCTYPE 字符串中的最后两个子字符串是用来支持拖和放。

AppWizard 只为程序员创建单个文档类。然而,在有些情况下,程序员可能要支持多种文档类型。例如,应用程序可能需要数据表和条图文档,每种文档类型都由其自己的文档类表示,还可能由其自己的视类表示。当用户选择了 File New 命令时,框架将显示一个对话框,列出所支持的文档类型,然后再按用户选择的类型创建文档。每种文档类型都是由其自己的文档模板对象管理的。

要创建额外的文档类,可以使用 ClassWizard 对话框中的 AddClass 按钮。先选择 CDocument 作为要从其派生的 Class Type,并提供所需的文档信息,然后再实现这个新类的数据。

为了让框架知道额外的文档类,程序员必须在应用程序类的 InitInstance 覆盖函数中再增加调用一次 AddDocTemplate。

在 InitInstance 函数中,MDI 模板是通过以下类似的形式来构造的:

```
AddDocTemplate(new CMultiDocTemplate (IDR _ MYDOCTYPE,
RUNTIME _ CLASS(CMyDoc),
RUNTIME _ CLASS(MDIChildWnd),//Standard MDI Child frame
RUNTIME _ CLASS(Myview)));
```

上述对 AddDocTemplate 的调用使得 MDI 应用能够支持多个子窗口,每一个都和一个文档对象和视对象相连,若干个子窗口(及其相应的视对象)也可能与同一个文档对象相连。

11.1.3 初始化及清理文档和视

使用下述原则,对文档和视程序员要初始化和事后清理。

- 框架将初始化文档和视,程序员要初始化所增加的所有数据。
- 文档和视关闭时,框架将进行清理,对于程序员在堆栈中所分配的所有内存,程序员必须在那些文档和视的成员函数中进行释放。

注意:回顾一下,整个应用程序的初始化最好在 CWinApp 类的 InitInstance 成员函数的覆盖函数中进行,整个应用程序的清理最好在 CWinApp 的 ExitInstance 成员函数的覆盖函数中进行。

MDI 应用程序中一个文档的生命周期(以及其边框窗口和视)是这样的:

1)在动态创建过程中,文档的构造函数被调用。

2)对于每个新文档,该文档的 OnNewDocument 或 OnOpenDocument 被调用。

3)在文档的整个生命期中,用户与文档进行交互。

4)框架调用 DeleteContents 删除文档特定的数据。

5)文档的析构函数所调用。

文档的创建有两个不同的方法，因此，文档类必须两种方法都支持。首先用户可以用FileNew命令创建一个新的空文档。在这种情况下，要在CDocument类的OnNewDocument成员函数的覆盖函数中初始化文档。其次，用户可以用File菜单中的Open命令创建一个新文档，其内容将从一个文件中读取。在这种情况下，要在CDocument类的OnOpenDocument成员函数的覆盖中初始化文档。如果两种初始化是一样的，可以调用两个覆盖函数中的一个公共成员函数，或者，OnOpenDocument可以调用OnNewDocument来初始化一个空文档，然后完成打开操作。

视在其文档被创建之后再创建。初始化的最佳时间是在框架已经创建了文档、边框窗口和视之后。初始化视可以通过覆盖CView的OnInitialUpate成员函数来实现。如果需要在每次文档发生变化时重新进行初始化，或调整一些东西，可以覆盖OnUpdate。

文档要关闭时，框架将首先调用其DeleteContents成员函数。如果要释放在文档操作期间在堆栈中分配的内存，DeletContents就是释放内存的最佳位置。

注意：在文档的析构函数中不应该释放文档数据。如果是SDI应用程序的话，该文档对象可以被重新使用。

程序员可以覆盖一个视的析构函数，以释放在堆栈中分配的内存。

11.2 实例 mdi

该例子演示了如何生成子窗口以及如何排列它们。如图11.1所示。由于是多文档用户界面，打开最终生成的exe文件时，出现的第一个子窗口中画了一个椭圆，不过没有底色。把这个子窗口去掉，变成如图11.2所示的窗口。主菜单的两个菜单项“新框架(N)”和“窗口”是新增的，“窗口”下面还有三个菜单项：“层叠”，“平铺”和“排列”，还有原有的“查看”菜单下面的那个“查看文件”这一项。点击“新框架(N)”菜单，就会在视窗口产生一系列的子窗口，我们可用“窗口”菜单下面的三个菜单项的功能来控制这些子窗口的排列。“查看”菜单下面的“查看文件”菜单项的功能是在最后打开的子窗口内装入本例程的一个文件MainFm.cpp。下面介绍如何创建mdi例程。

1)生成多文档工程mdi。

2)编辑菜单IDR_MAINFRAME。在“查看(V)”菜单的最下面加菜单项“查看文件(V)”(IDC_VIEWFILE)，prompt为View File。在“查看(V)”菜单之后加菜单“新框架(N)”(IDC_NEWCHILD)，prompt为produce new childframe。再添加菜单“窗口(W)”。“窗口(W)”下面加三个菜单项“层叠(C)”(IDC_CASCADE)，prompt为Cascade windowd；“平铺(T)”(IDC_TITLE)，prompt为“Title window”；“排列(A)”(ILDC_ARRANGE)，prompt为Arange Window。

3)编辑下列文件：childFm.h，childFrm.cpp，MainFm.h，MainFm.cpp，mdiDoc.h，mdiDoc.cpp和mdiView.h，mdiview.cpp。

```
// ChildFrm.h : interface of the CChildFrame class
//
/////////////////////////////////////////////////////////////////////////////
```

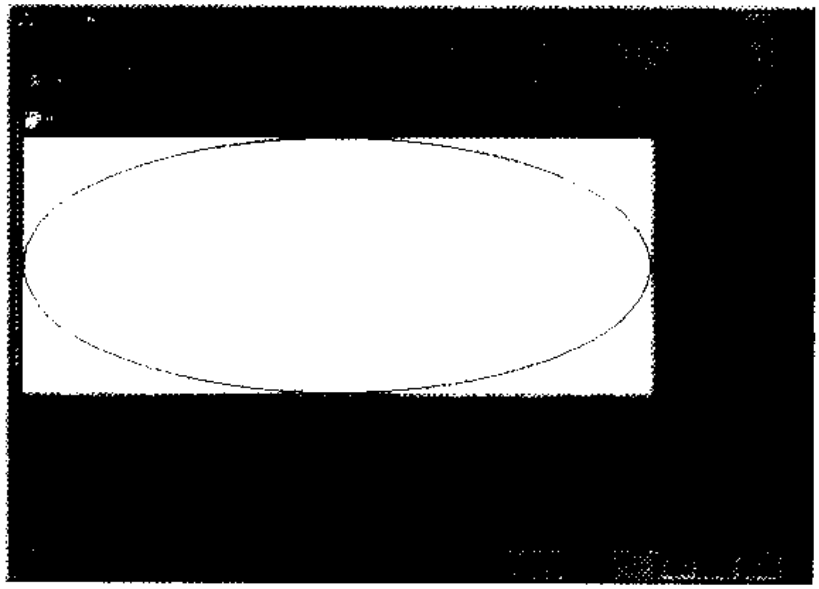


图 11.1　mdi 图例 1

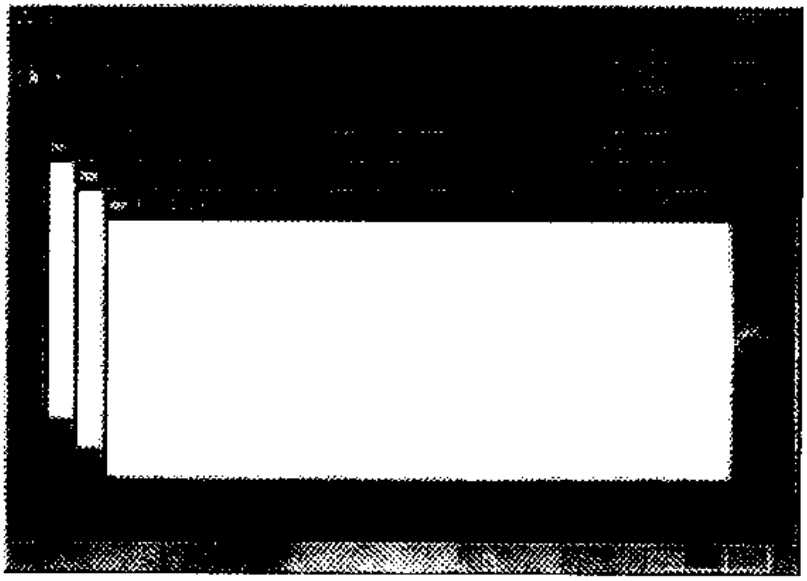


图 11.2　mdi 图例 2

```
#if ! defined(AFX_CHILDFRM_H__6E20050C_1E43_11D2_BA07_444553540000__INCLUDED_)
#define AFX_CHILDFRM_H__6E20050C_1E43_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CChildFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CChildFrame)
public:
    CChildFrame();
```

```
// Attributes
public:
	CEdit * edit;

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CChildFrame)
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CChildFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
	//{{AFX_MSG(CChildFrame)
	afx_msg void OnClose();
	afx_msg void OnSize(UINT nType, int cx, int cy);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line

#endif // !defined(AFX_CHILDFRM_H__6E20050C_1E43_11D2_BA07_444553540000__INCLUDED_)
// ChildFrm.cpp : implementation of the CChildFrame class
//

#include "stdafx.h"
#include "mdi.h"
```

```
#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CChildFrame

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
	//{{AFX_MSG_MAP(CChildFrame)
	ON_WM_CLOSE()
	ON_WM_SIZE()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChildFrame construction/destruction

CChildFrame::CChildFrame()
{

	edit=NULL;
}

CChildFrame::~CChildFrame()
{
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CMDIChildWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CChildFrame diagnostics
```

```
#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
	CMDIChildWnd::AssertValid();
}

void CChildFrame::Dump(CDumpContext& dc) const
{
	CMDIChildWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CChildFrame message handlers

void CChildFrame::OnClose()
{


	CMDIChildWnd::OnClose();
}

void CChildFrame::OnSize(UINT nType, int cx, int cy)
{
	CMDIChildWnd::OnSize(nType, cx, cy);

	if(edit)
		edit->SetWindowPos(0,0,0,cx,cy,SWP_SHOWWINDOW);


}
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__6E20050A_1E43_11D2_BA07_444553540000__INCLUDED_)
#define AFX_MAINFRM_H__6E20050A_1E43_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
```

```
class CMainFrame : public CMDIFrameWnd
{
	DECLARE_DYNAMIC(CMainFrame)
public:
	CMainFrame();

// Attributes
public:

private:
	int nchildnum;

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMainFrame)
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CMainFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
	CStatusBar  m_wndStatusBar;
	CToolBar    m_wndToolBar;

// Generated message map functions
protected:
	//{{AFX_MSG(CMainFrame)
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnNewchild();
	afx_msg void OnCascade();
	afx_msg void OnTile();
	afx_msg void OnArrange();
	afx_msg void OnViewfile();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
```

```
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__6E20050A_1E43_11D2_BA07_444553540000__INCLUDED_)
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "mdi.h"

#include "MainFrm.h"
#include "ChildFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
	ON_WM_CREATE()
	ON_COMMAND(IDC_NEWCHILD, OnNewchild)
	ON_COMMAND(IDC_CASCADE, OnCascade)
	ON_COMMAND(IDC_TILE, OnTile)
	ON_COMMAND(IDC_ARRANGE, OnArrange)
	ON_COMMAND(IDC_VEIWFILE, OnVeiwfile)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
	ID_SEPARATOR,           // status line indicator
```

```
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{

    nchildnum=0;
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizeable toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
```

```
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CMDIFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CMDIFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CMDIFrameWnd::Dump(dc);
}

#endif //  DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

void CMainFrame::OnNewchild()
{

    nchildnum++;
    CString str;
    str.Format("New Child%d",nchildnum);
    CChildFrame *  pchildframe=new CChildFrame();
    pchildframe->Create(NULL,str);
    pchildframe->ShowWindow(SW_SHOW);
```

```
}

void CMainFrame::OnCascade()
{
        MDIIconArrange();
}

void CMainFrame::OnTile()
{
        MDITile();
}

void CMainFrame::OnArrange()
{
        MDICascade();
}

void CMainFrame::OnVeiwfile()
{
        CChildFrame * pchild = (CChildFrame * )MDIGetActive();
      pchild->edit = new CEdit();
        CRect rect;
        pchild->GetClientRect(&rect);
        DWORD dwStyle = WS_CHILD | WS_VISIBLE | ES_MULTILINE | WS_HSCROLL | WS_
VSCROLL;
        pchild->edit->Create(dwStyle,rect,pchild,ID_EDIT);
        CFile file;
        char * filename = "d:\\chb\\mdi\\MainFrm.cpp";
        file.Open(filename,CFile::modeRead);
        DWORD len = file.GetLength();
        char * lpbuf = new char[len];
        file.Read(lpbuf,len);
        pchild->edit->SetWindowText(lpbuf);




// mdiView.h : interface of the CMdiView class
//
```

```
////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MDIVIEW_H__6E200510_1E43_11D2_BA07_444553540000__INCLUDED)
#define AFX_MDIVIEW_H__6E200510_1E43_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMdiView : public CView
{
protected: // create from serialization only
	CMdiView();
	DECLARE_DYNCREATE(CMdiView)

// Attributes
public:
	CMdiDoc* GetDocument();

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMdiView)
	public:
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	protected:
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CMdiView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:
```

```
// Generated message map functions
protected:
	//{{AFX_MSG(CMdiView)
		// 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()
};

#ifndef _DEBUG  // debug version in mdiView.cpp
inline CMdiDoc* CMdiView::GetDocument()
   { return (CMdiDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MDIVIEW_H__6E200510_1E43_11D2_BA07_444553540000__INCLUDED_)

// mdiView.cpp : implementation of the CMdiView class
//

#include "stdafx.h"
#include "mdi.h"

#include "mdiDoc.h"
#include "mdiView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMdiView

IMPLEMENT_DYNCREATE(CMdiView, CView)

BEGIN_MESSAGE_MAP(CMdiView, CView)
```

```
    //{{AFX_MSG_MAP(CMdiView)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMdiView construction/destruction

CMdiView::CMdiView()
{
    // TODO: add construction code here

}

CMdiView::~CMdiView()
{
}

BOOL CMdiView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMdiView drawing

void CMdiView::OnDraw(CDC* pDC)
{
    CMdiDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    CRect rect;
    GetClientRect(&rect);
    pDC->Ellipse(rect);

}
```

```
/////////////////////////////////////////////////////////////////////////////
// CMdiView printing

BOOL CMdiView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CMdiView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CMdiView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CMdiView diagnostics

#ifdef _DEBUG
void CMdiView::AssertValid() const
{
	CView::AssertValid();
}

void CMdiView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CMdiDoc* CMdiView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMdiDoc)));
	return (CMdiDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMdiView message handlers
```

11.3 小　　结

本章介绍了多文档的使用。多文档多视图是 Miorosoft 系列产品长期采用的风格。使用具有多文档风格的程序，可以带来很多方便。编制程序时可用它来处理比较复杂的文档和视，使界面显得内容多而不乱。

第 12 章　切 分 窗 口

Windows 98 的资源管理器是我们在 Windows 操作系统中见得最多的具有典型切分窗口特征的工具，它的左边是一个显示磁盘目标的树状视图，右边则是显示当前目录下的子目录和文件的列表视图。如果有哪一边显示的字符看不见被遮挡了，我们就可移动它们中间的分隔条来看清被遮住了的部分。本章先介绍切分窗口的有关概念，然后分别说明如何创建动态切分窗口和静态切分窗口。

12.1　切 分 窗 口

切分窗口看上去像是一种特殊的框架窗口，在各个窗片中包含若干个视。应用在创建的时候就可对窗口进行划分。用户也可以通过选择菜单命令或拖动切分窗口滚动条中的切分框对窗口进行划分。当窗口被切分之后，用户可通过鼠标移动窗棱或边框来调整窗片的相对尺寸。切分窗口既适用于 SDI 应用，也适用于 MDI 应用。

切分窗口是通过类 CSplitterWnd 来表示的，对 Windows 来说，CSplitterWnd 对象是一个真正的窗口，它完全占据了框架窗口（CFrameWnd 或 CMDIChildWnd）的用户区域，而视窗口则占据了切分窗口的窗片区域，切分窗口并不参与命令传递机制，窗片中活动着的视窗从逻辑上来看直接被连接到了它的框架窗口中。

切分窗口又分为动态和静态切分窗口两种。动态切分窗口允许用户在任何时候都可对窗口进行切分，既可通过选择菜单项来对窗口进行切分，也可通过拖动滚动条中的切分框对窗口进行切分。动态切分窗口中的图片通常使用的是同一个视类。当切分窗口被创建时，左上角窗片通常被初始化成一个特殊的视。在动态切分窗口中，滚动条在视中是被共用的，例如，在只进行了一次水平切分的窗口中，底滚动条同时控制着上下两个视。

对于静态切分窗口，当窗口第一次被创建时，窗片就已被切分好了，用户可以移动窗棱，但此时已不能对窗口进行合并或再划分。静态切分窗口允许使用多个视类，并且可在创建时对这些视类进行配置。在静态切分窗口中，每个窗片都有自己的滚动条。

12.2　多 视 选 择

当将多视表达方式与应用模式相结合时，可以有许多排列组合方式。下面列出的就是其中一些可能的组合方式。

- SDI 应用，带切分窗口，只有一个视类——其中每个切分窗口都可以被滚动到文档的不同部分。程序员可以决定所包含的水平和竖直窗片的最大数目，而用户则可以在运行中对窗口进行切分。
- SDI 应用，带切分窗口，具有多个视类——程序员可决定窗片的数目以及视的次序，而用户可在运行中改变窗片的尺寸。
- MDI 应用，不带切分窗口，只有一个视类——我们所见到的标准 MDI 应用都属于这种

情况，用户可以通过选择 New Window 菜单项为已经打开的文档再创建一个新的子窗口。

- MDI 应用，不带切分窗口，具有多个视类——只需对标准 MDI 应用做很小的改动就可以允许使用多视。

12.3 动态切分窗口

类 CSplitterWnd 提供一个分隔器窗口，分隔器窗口是含有多个窗格的窗口。窗格通常是应用程序特定的由 CView 派生的对象，但它也可以是具有适当子窗口 ID 的任何 CWnd 对象。

一个 CSplitterWnd 对象通常被嵌入 CFrameWnd 或 CMDIChildWnd 父对象。可按如下步骤创建一个 CSplitterWnd 对象：

1）在父框架中嵌入一个 CSplitterWnd 成员变量。

2）把父框架的 OnCreateClient 成员函数替换掉。

3）从替换 OnCreatClient 的代码中调用类 CSplitterWnd 的 Create 或 CreateStatic 成员函数。

调用 Create 成员函数可创建一个动态的分隔器窗口。动态的分隔器窗口通常用于创建和滚动同一文档的多个窗格或视图。框架将自动为该分隔器创建一个起始窗口，然后，当用户操纵该分隔器窗口的控制时，框架创建、修改尺寸并且排列附加的窗格。

当调用 Create 时，应指定一个最小行高度和最小列宽度，这两个最小值被用来确定窗格什么时候太小至不能被完全显示。程序员调用了 Create 之后，可通过调用 SetColumnInfo 和 SetRowInfo 来调整这些最小值。

还可使用 SetColumnInfo 和 SetRowInfo 成员函数来给一列设置一个"理想"宽度以及给行设置一个"理想"高度。当框架显示一个分隔器窗口时，首先显示父框架，然后才显示该分隔器窗口。然后，框架从分隔器窗口的客户区左上角至右下角，根据窗格的理想尺寸以行列排放各个窗格。

动态分隔器窗口中的所有窗格都必须是同一类的窗格。读者熟悉的支持动态分隔器窗口的应用程序包括 Microsoft word 和 Microsoft Excel 等。

本节介绍如何在一个已经存在的程序窗口中加入分隔框，使用户能将窗口分成二个分开的视图窗口（称为板面）。二个板面间的边框称为分隔条。二个视图窗口显示同一图形，但每个窗口可以单独滚动显示图形的不同部分。为了分隔窗口，用户可以双击分隔框（将窗口分成相同的二个视图）或拖动分隔框到所要的位置。窗口分隔时，单个垂直滚动条同时滚动一个视图窗口，但每个视图有自己的水平滚动条，所以视图在水平方面可单独滚动。

为了给程序加入分隔能力，必须修改主帧窗口类。首先在主窗口的 CMainFrame 类定义开头加入 m_SplitterWnd 的声明如下：

```
class CMainFrame:public CFrameWnd
{protected
CSpitterWnd m_SplitterWnd;
// remainder of CMainFrame definition
}
```

新成员对象 m_SplitterWnd 是 MFC 类 CSplitterWnd(从 CWnd 派生)的实例。这个对象用于生成和管理分隔器窗口,即可显示多个视图窗口的窗口。下一步是定义 OnCreateClient 虚拟函数的覆盖版作为 CMainFrame 类的成员。用类向导生成函数声明和最小函数定义。在类向导对话框中单击 MessageMaps 标签,在 Object IDS 列表框中选择 CMainFrame 类,在 Message 列表框中选择 OnCreateClient 虚拟函数。加入函数和单击 Edit Code 键后,删去 CFrameWnd::OnCreateClient 调用,使函数变成:

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs,CreateContext * pContext)
{ // TODO:Add your specialized code hare and/or call the base class
return m_SplitterWnd.Create
(this,// parent of splitter window
1,// maximum rows
2,// maximum columns
CSize(15,15),// minimum view window sizepContext);// pass on context information
}
```

主帧窗口首次生成时,MFC 调用 OnCreateClient 虚拟函数。这个函数的缺省版本(在 CFrameWnd 中定义)生成单个视图窗口,填满主帧窗口的客户区。刚刚生成的这个函数的覆盖版本调用 mSplitterWnd 对象的 CSplitterWnd::create 函数生成分隔器窗口而不是生成一个视图窗口。分隔器窗口本身起初生成一个视图窗口,后面用户双击分隔框时,分隔器窗口就生成另一个视图窗口,提供图形的第二个视图。传入 Create 的第一个参数指定分隔器窗口的双亲,传入 this 的分隔器窗口成为主帧窗口的孩子。

说明:分隔器窗口是主帧窗口的孩子,而每个视图窗口则是分隔器的"孩子"。

第二个参数指定垂直方向视图的最大数目,传入 1 表示用户不能用水平分隔条分隔窗口,因此窗口不出现水平分隔框。同样,第一个参数指定水平方向的最大视图数,传入 2 表示用户可将窗口分成左右二个视图。第二、三个参数只能赋值 1 或 2,如果两个参数都赋值 2,则可将窗口一分为四!

第四个参数设置视图窗口的最小水平和垂直尺寸,用户不能将分隔条移到生成更小的水平或垂直尺寸的视图的位置。第五个参数传入 OnCreateClient 函数的情境信息。

前面介绍了在现有 MFC 应用程序的程序窗口加入分隔能力。也可以用应用向导在生成新程序时加入分隔能力。为此,在应用向导的 Step4 对话框中使用下列步骤:

1)单击 Advance 键,打开 Advanced Options 对话框。

2)在 Advance Options 对话框中,单击 Window Styles 标签,可以设置影响主帧窗口的高级选项。

3)核选 Use Split Window(使用分隔窗口)选项。

这几步使应用向导自动生成上节介绍的码,可以人工调整传入 CSplitterWnd::Create 的参数(具体见第二,三,四个参数)以改变最大视图窗口数和最小视图窗口尺寸。

12.4 静态切分窗口

使用 CreateStatic 成员函数可创建一个静态分割器窗口。用户只能修改静态分隔器窗口

中的窗格的尺寸，但不能改变其中的窗格序号或次序。

在创建静态分隔器时必须专门创建该静态分隔器的所有窗格。必须在父框架的 OnCreateClient 成员函数返回之前确保创建了所有窗格，否则，框架将不能正确显示该窗口

CreateStatic 成员函数将使用最小行高度和最小列宽度来自动初始化一个静态分隔器。程序员在调用了 Create 之后，可通过调用 SetColumnInfo 和 SetRowInfo 成员函数来调整这两个最小值。在调用 CreateStatic 函数之后，还可以使用 SetColumnInfo 和 SetRowInfo 成员函数来指示自己希望的理想窗格尺寸。

静态分隔器中的各个窗格通常属于不同的类。关于静态分隔窗口的范例，读者可观察图形编辑器和 Windows File Manager，分隔器窗口支持特殊的滚动条(除窗格可能带有滚动框之外)。这些滚动条是 CSplitterWnd 对象的子对象并且被各窗体共享。

在创建分隔器窗口时创建这些特殊的滚动条。例如：若一个 CSplitterWnd 具有一行、两行，则 WS_VSCROLL 风格将显示被两个窗格共享的垂直滚动条。当用户移动该滚动条时，WM_VSCROLL 消息将被发送给两个窗格。当窗格设置滚动条位置时，此共享滚动条将被设置

12.5 实例 Dsplitter

该例子演示了动态切分窗口的使用，如图 12.1 所示，这是个多文档例程。静态切分窗口是基于子窗口的 2 * 2 窗口，在每个切分窗口的视内可用十字型的鼠标画直线。

下面分两步介绍该例程的创建过程

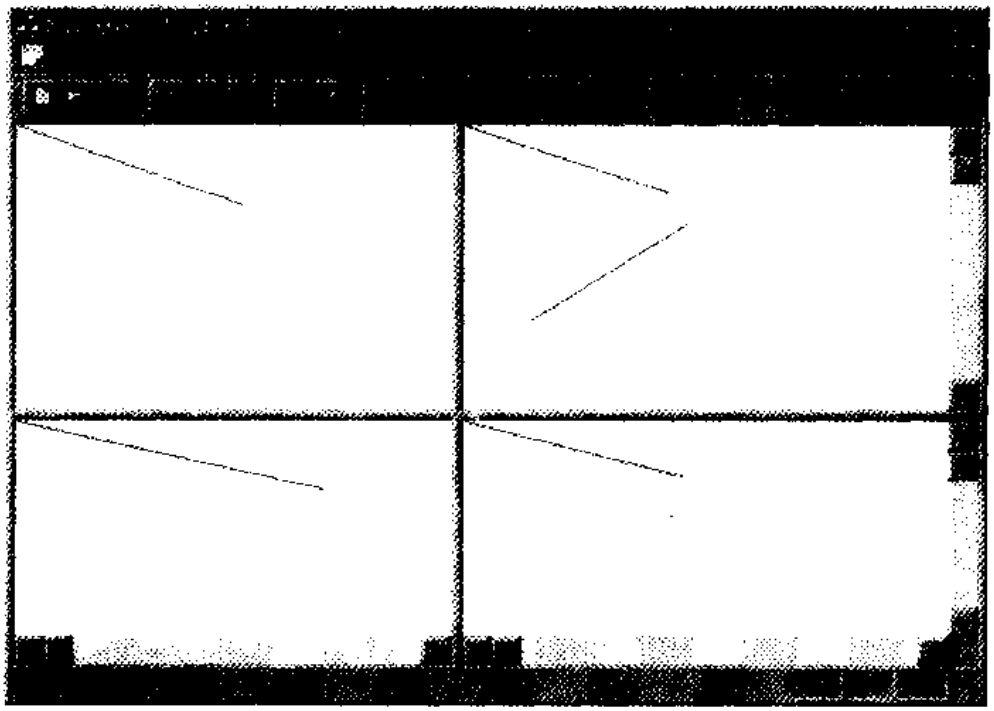


图 12.1 Dsplitter 图例

1)生成多文档工程 Dsplitter。

2)编辑文 ChildFrm.h, ChildFrm.cpp, DsplitterDoc.h, DsplitterDoc.cpp, DsplitterView.h, DsplitterView.cpp。

```
// DSplitterView.h : interface of the CDSplitterView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_DSPLITTERVIEW_H__4DC32647_3108_11D2_BA07_444553540000__INCLUDED_)
#define AFX_DSPLITTERVIEW_H__4DC32647_3108_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CDSplitterView : public CView
{
protected: // create from serialization only
	CDSplitterView();
	DECLARE_DYNCREATE(CDSplitterView)

// Attributes
public:
	CDSplitterDoc* GetDocument();
protected:
	int m_Dragging;
	HCURSOR m_hCross;
	CPoint m_PointOld,m_PointOrigin;

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CDSplitterView)
	public:
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	protected:
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	//}}AFX_VIRTUAL
```

```
// Implementation
public:
        virtual ~CDSplitterView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CDSplitterView)
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in DSplitterView.cpp
inline CDSplitterDoc* CDSplitterView::GetDocument()
   { return (CDSplitterDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_DSPLITTERVIEW_H__4DC32647_3108_11D2_BA07_444553540000__
INCLUDED_)
// DSplitterView.cpp : implementation of the CDSplitterView class
//

#include "stdafx.h"
#include "DSplitter.h"

#include "DSplitterDoc.h"
#include "DSplitterView.h"

#ifdef _DEBUG
```

```
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDSplitterView

IMPLEMENT_DYNCREATE(CDSplitterView, CView)

BEGIN_MESSAGE_MAP(CDSplitterView, CView)
	//{{AFX_MSG_MAP(CDSplitterView)
	ON_WM_LBUTTONDOWN()
	ON_WM_LBUTTONUP()
	ON_WM_MOUSEMOVE()
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDSplitterView construction/destruction

CDSplitterView::CDSplitterView()
{
	// TODO: add construction code here
	m_Dragging=0;
	m_hCross=AfxGetApp()->LoadStandardCursor(IDC_CROSS);

}

CDSplitterView::~CDSplitterView()
{
}

BOOL CDSplitterView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CView::PreCreateWindow(cs);
}
```

```
////////////////////////////////////////////////////////////////////////////
// CDSplitterView drawing

void CDSplitterView::OnDraw(CDC* pDC)
{
	CDSplitterDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	CPoint point = pDC->GetCurrentPosition();
	pDC->MoveTo(m_PointOrigin);
	pDC->LineTo(point);


	// TODO: add draw code for native data here
}

////////////////////////////////////////////////////////////////////////////
// CDSplitterView printing

BOOL CDSplitterView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CDSplitterView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CDSplitterView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

////////////////////////////////////////////////////////////////////////////
// CDSplitterView diagnostics

#ifdef _DEBUG
void CDSplitterView::AssertValid() const
{
	CView::AssertValid();
}
```

```
void CDSplitterView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CDSplitterDoc* CDSplitterView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CDSplitterDoc)));
	return (CDSplitterDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CDSplitterView message handlers

void CDSplitterView::OnLButtonDown(UINT nFlags, CPoint point)
{

	m_PointOld = point;
	m_PointOrigin = point;
	SetCapture();
	m_Dragging = 1;
	CRect rect;
	GetClientRect(&rect);
	ClientToScreen(&rect);
	::ClipCursor(&rect);

	CView::OnLButtonDown(nFlags, point);
}

void CDSplitterView::OnLButtonUp(UINT nFlags, CPoint point)
{

	if(m_Dragging)
	{
		m_Dragging = 0;
		::ReleaseCapture();
		::ClipCursor(NULL);
		CClientDC dc(this);
		dc.SetROP2(R2_NOT);
		dc.MoveTo(m_PointOrigin);
		dc.LineTo(m_PointOld);
		dc.SetROP2(R2_COPYPEN);
```

```
        dc.MoveTo(m_PointOrigin);
        dc.LineTo(point);
    }

    CDSplitterDoc* pDoc = (CDSplitterDoc*)GetDocument();
    pDoc->UpdateAllViews(this);


    CView::OnLButtonUp(nFlags, point);
}

void CDSplitterView::OnMouseMove(UINT nFlags, CPoint point)
{

    ::SetCursor(m_hCross);
    if(m_Dragging)
    {
        CClientDC dc(this);
        dc.SetROP2(R2_NOT);
        dc.MoveTo(m_PointOrigin);
        dc.LineTo(m_PointOld);
        dc.MoveTo(m_PointOrigin);
        dc.LineTo(point);
        m_PointOld = point;
    }


    CView::OnMouseMove(nFlags, point);
}




#if !defined(AFX_FVIEW_H__4DC32636_3108_11D2_BA07_444553540000__INCLUDED_)
#define AFX_FVIEW_H__4DC32636_3108_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// FView.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CFView form view
```

```
class CSSplitterDoc;
#ifndef __AFXEXT_H__
#include <afxext.h>
#endif

class CFView : public CFormView
{
protected:
        CFView();           // protected constructor used by dynamic creation
        DECLARE_DYNCREATE(CFView)

// Form Data
public:
        //{{AFX_DATA(CFView)
        enum { IDD = IDD_DIALOG1 };
        int          m_col;
        CString      m_editstr;
        //}}AFX_DATA

// Attributes
public:
        CSSplitterDoc* GetDocument();

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CFView)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);     // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        virtual ~CFView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

        // Generated message map functions
        //{{AFX_MSG(CFView)
        afx_msg void OnColor();
```

```
    afx_msg void OnSend();
    afx_msg void OnEmpty();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FVIEW_H__4DC32636_3108_11D2_BA07_444553540000__INCLUDED_)

// FView.cpp : implementation file
//

#include "stdafx.h"
#include "SSplitter.h"
#include "FView.h"
#include "SSplitterDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFView

IMPLEMENT_DYNCREATE(CFView, CFormView)

CFView::CFView()
    : CFormView(CFView::IDD)
{
    //{{AFX_DATA_INIT(CFView)
    m_col = -1;
    m_editstr = _T("");
    //}}AFX_DATA_INIT
}

CFView::~CFView()
```

```
{
}

void CFView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFView)
    DDX_Radio(pDX, IDC_RED, m_col);
    DDX_Text(pDX, IDC_EDIT, m_editstr);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CFView, CFormView)
    //{{AFX_MSG_MAP(CFView)
    ON_BN_CLICKED(IDC_BLUE, OnColor)
    ON_BN_CLICKED(IDC_SEND, OnSend)
    ON_BN_CLICKED(IDC_EMPTY, OnEmpty)
    ON_BN_CLICKED(IDC_RED, OnColor)
    ON_BN_CLICKED(IDC_GREEN, OnColor)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFView diagnostics

#ifdef _DEBUG
void CFView::AssertValid() const
{
    CFormView::AssertValid();
}

void CFView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}
#endif //_DEBUG

CSSplitterDoc* CFView::GetDocument()
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CSSplitterDoc)));
    return (CSSplitterDoc*)m_pDocument;
}
```

```
/////////////////////////////////////////////////////////////////////////////
// CFView message handlers

void CFView::OnColor()
{
	// TODO: Add your control notification handler code here
	UpdateData();
	COLORREF col;
	switch(m_col)
	{
	case 0:
		col = RGB(255,0,0);
		break;
	case 1:
		col = RGB(0,255,0);
		break;
	case 2:
		col = RGB(0,0,255);
		break;
	}
	CSSplitterDoc * m_pDoc = (CSSplitterDoc * )GetDocument();
	m_pDoc->col = col;
	m_pDoc->UpdateAllViews(NULL);

}

void CFView::OnSend()
{
	// TODO: Add your control notification handler code here
	UpdateData();
	CSSplitterDoc * m_pDoc = (CSSplitterDoc * )GetDocument();
	m_pDoc->drawstr = m_editstr;
	m_pDoc->UpdateAllViews(NULL);

}

void CFView::OnEmpty()
{
   // TODO: Add your control notification handler code here
	CSSplitterDoc * m_pDoc = (CSSplitterDoc * )GetDocument();
	m_pDoc->drawstr.Empty();
	m_pDoc->UpdateAllViews(NULL);
```

```
}

// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__4DC3262A_3108_11D2_BA07_444553540000__INCLUDED_)
#define AFX_MAINFRM_H__4DC3262A_3108_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
	CMainFrame();
	DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
protected:
	CSplitterWnd m_wndSplitter;


// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMainFrame)
	public:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	protected:
	virtual BOOL OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CMainFrame();
#ifdef _DEBUG
```

```
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // 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()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__4DC3262A_3108_11D2_BA07_444553540000__INCLUDED_)

// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "SSplitter.h"
#include "FView.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame
```

```
IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	ON_WM_CREATE()
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
	ID_SEPARATOR,           // status line indicator
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	// TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	if (!m_wndToolBar.Create(this) ||
		!m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
	{
		TRACE0("Failed to create toolbar\n");
		return -1;      // fail to create
	}
```

```
    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar \n");
        return -1;      // fail to create
    }

    // TODO: Remove this if you don't want tool tips or a resizeable toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG
```

```
////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
{
	BOOL result=m_wndSplitter.CreateStatic(this,1,2);
	result |= m_wndSplitter.CreateView(0,0,pContext->m_pNewViewClass,CSize(100,
100),pContext);
	result |= m_wndSplitter.CreateView(0,1,RUNTIME_CLASS(CFView),CSize(100,100),
pContext);
	return result;
}

// SSplitterDoc.h : interface of the CSSplitterDoc class
//
////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_SSPLITTERDOC_H__4DC3262C_3108_11D2_BA07_444553540000__IN-
CLUDED_)
#define AFX_SSPLITTERDOC_H__4DC3262C_3108_11D2_BA07_444553540000__INCLUDED

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


class CSSplitterDoc : public CDocument
{
protected: // create from serialization only
	CSSplitterDoc();
	DECLARE_DYNCREATE(CSSplitterDoc)

// Attributes
public:
	CString drawstr;
	COLORREF col;

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSSplitterDoc)
```

```
public:
virtual BOOL OnNewDocument();
virtual void Serialize(CArchive& ar);
//}}AFX_VIRTUAL

// Implementation
public:
virtual ~CSSplitterDoc();
#ifdef _DEBUG
virtual void AssertValid() const;
virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
//{{AFX_MSG(CSSplitterDoc)
// 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()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SSPLITTERDOC_H__4DC3262C_3108_11D2_BA07_444553540000__INCLUDED_)
```

12.6 实例 Ssplitter

该例子演示了静态切分窗口的使用，如图 12.2 所示。整个视窗内有两个视，左边是工程缺省产生的 Ssplitter view，右边是新添加的视 CFView，该视从 CFormView 类派生而来。它左边有一编辑框，右边是两个按钮和一个控制颜色变化的 Group Box。在编辑框内输入一些文字，再选择不同的颜色，单击“发送”按钮，编辑框内的文字将以所选颜色显示在左边的视 Ssplitter View 内。若点击“清空”按钮，左边视窗内的字文消失。

下面介绍该例程的创建过程。

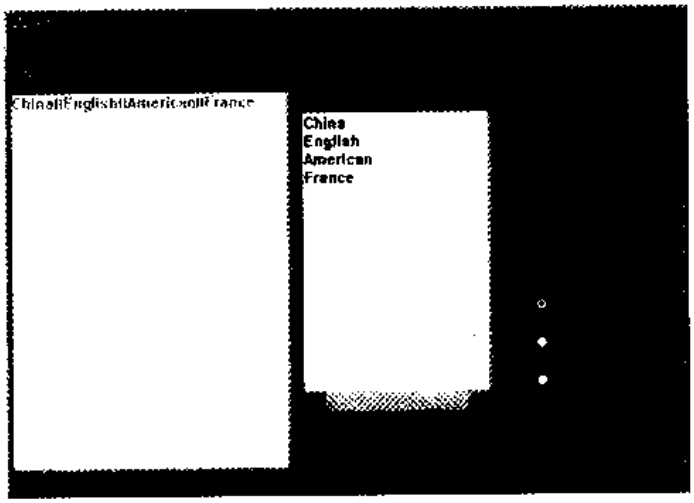


图 12.2 Ssplitter 图例

1)生成单文档工程 Ssplitter。

2)添加一对话框 IDD _ DIALOG1,"发送"按钮(IDC _ SEND),"清空"按钮(IDC EMPTY),"颜色"组合框(IDC _ STATIC)蓝色(IDC _ BLUE)。

3)编辑文件 Fview.h, Fview.cpp, MainFrm.h, MainFrm.cpp, SsplitterDoc.h, SsplitterDoc.cpp 和 SsplitterView.h, SsplitterView.cpp。

```
// SSplitterDoc.cpp : implementation of the CSSplitterDoc class
//

#include "stdafx.h"
#include "SSplitter.h"

#include "SSplitterDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSSplitterDoc

IMPLEMENT_DYNCREATE(CSSplitterDoc, CDocument)

BEGIN_MESSAGE_MAP(CSSplitterDoc, CDocument)
	//{{AFX_MSG_MAP(CSSplitterDoc)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
```

```
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSSplitterDoc construction/destruction

CSSplitterDoc::CSSplitterDoc()
{
    // TODO: add one-time construction code here

}

CSSplitterDoc::~CSSplitterDoc()
{
}

BOOL CSSplitterDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}



/////////////////////////////////////////////////////////////////////////////
// CSSplitterDoc serialization

void CSSplitterDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}
```

```
/////////////////////////////////////////////////////////////////////////////
// CSSplitterDoc diagnostics

#ifdef _DEBUG
void CSSplitterDoc::AssertValid() const
{
	CDocument::AssertValid();
}

void CSSplitterDoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CSSplitterDoc commands
// SSplitterView.h : interface of the CSSplitterView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_SSPLITTERVIEW_H__4DC3262E_3108_11D2_BA07_444553540000__INCLUDED_)
#define AFX_SSPLITTERVIEW_H__4DC3262E_3108_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CSSplitterView : public CView
{
protected: // create from serialization only
	CSSplitterView();
	DECLARE_DYNCREATE(CSSplitterView)

// Attributes
public:
	CSSplitterDoc* GetDocument();

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
```

```
        //{{AFX_VIRTUAL(CSSplitterView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CSSplitterView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CSSplitterView)
                // 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()
};

#ifndef _DEBUG  // debug version in SSplitterView.cpp
inline CSSplitterDoc* CSSplitterView::GetDocument()
   { return (CSSplitterDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_SSPLITTERVIEW_H__4DC3262E_3108_11D2_BA07_444553540000__
 INCLUDED_)
```

```
// SSplitterView.cpp : implementation of the CSSplitterView class
//

#include "stdafx.h"
#include "SSplitter.h"

#include "SSplitterDoc.h"
#include "SSplitterView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSSplitterView

IMPLEMENT_DYNCREATE(CSSplitterView, CView)

BEGIN_MESSAGE_MAP(CSSplitterView, CView)
    //{{AFX_MSG_MAP(CSSplitterView)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSSplitterView construction/destruction

CSSplitterView::CSSplitterView()
{
    // TODO: add construction code here

}

CSSplitterView::~CSSplitterView()
{
}
```

```
BOOL CSSplitterView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CSSplitterView drawing

void CSSplitterView::OnDraw(CDC* pDC)
{
	CSSplitterDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	pDC->SetTextColor(pDoc->col);
	pDC->TextOut(0,0,pDoc->drawstr);


	// TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CSSplitterView printing

BOOL CSSplitterView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CSSplitterView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CSSplitterView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CSSplitterView diagnostics
```

```
#ifdef _DEBUG
void CSSplitterView::AssertValid() const
{
    CView::AssertValid();
}

void CSSplitterView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CSSplitterDoc* CSSplitterView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CSSplitterDoc)));
    return (CSSplitterDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CSSplitterView message handlers
```

12.7 小　　结

切分窗口作为可视化编程的一个不可缺少的内容，有着广泛的应用，希望读者能熟练掌握和应用此方面技术。本书从以下几方面给读者简要介绍了此方面有关内容。

- 切分窗口的有关概念和分类
- 多视选择的创建和应用
- 动态切分窗口的创建和应用
- 静态切分窗口的创建和应用
- 实例辅助

第 13 章　打　　印

MFC 基本类库应用程序框架大大简化了打印的工作，而且还提供了打印预览的功能机制。如果要实现缺省的打印和打印预览功能根本用不着花力气，只要根据选项提供适当的接口即可。第 3 章介绍过页面设置和打印对话框，在此，框架会为我们缺省实现。当然本章除具体分析这些初步的过程外，还要进一步介绍如何实现更高级的打印和打印预览功能。

13.1　打　　印

正如显示有一个显示设备描述表一样，打印机也要有一个打印设备描述表。当使用图形函数时，MFC 使用同一代码在显示设备描述表上和打印机设备描述表上绘画。这种两重性是通过 CView::OnDraw()方法来实现的。传入 CView::OnDraw()的设备描述表指针可以来自表示两个不同设备的任一地方，这种安排为 MFC 的设备无关打印及打印预览打印提供了多种手段。

13.1.1　缺省基本打印

用应用向导生成新程序时，可以在应用向导 step4 对话框中选择 Printing and Print preview 选项包括基本打印和打印预览支持。选择这个选项就在程序 File 菜单中加入了 Print、Print Preview 和 Print Setup 命令。按照应用向导的实现，Print 命令可以打印一次所能放下的文档(多出部分略去)。同样，Print Preview 命令显示一个文档页面的打印样子。Print 和 Print Preview 命令都调用 OnDraw 函数生成实际文本和图形输出。Print Setup 命令显示 Print Setup 公共对话框，使用户可以选择打印机和指定打印机设备。

下面描述 Windows 中使用 MFC 框架的缺省打印过程。

在 MFC 应用程序中，视类有一个名为 OnDraw 的成员函数，它包含所有的拖曳码。OnDraw 采用 CDC 的指针作为参数。CDC 对象代表设备上下文接收由 OnDraw 产生的图象。当显示文档的窗口收到 WM_PAINT 消息时，框架调用 OnDraw 并给它传递屏幕的设备上下文(一个 PaintDC 对象，特定的)。相应地，OnDraw 输出到屏幕上。在 Windows 的编程中，给打印机发送输出只要通过使用恰当的设备上下文，非常类似于给屏幕发送输出。这是由于 Windows 图形设备界面(GDI)是独立于硬件的；可以为屏幕显示或打印使用相同的 GDI 函数。如果 OnDraw 接收的 CDC 对象代表打印机，那么 OnDraw 输出到打印机。这解释了 MFC 应用程序如何执行简单打印而不需要额外的努力。框架负责显示 Print 对话框并建立打印机的设备上下文。当用户从 File 菜单上选取 Print 命令时，窗口把此设备上下文传送给 OnDraw，它在打印机上画出文档。但在打印和屏幕显示之间有一些截然不同的地方。在打印时，必须把文档分成页，然后一次一页，而显示不管哪部分都在窗口中可见。作为必然结果，必须清楚纸张大小(是 A4 规格，法律规格还是信封)。也许要向不同方向打印，如横向或纵向。

13.1.2 实现高级打印

为使得 MFC 能响应打印命令，必须定义一个消息映射项，它使预定义的 MFC 值 ID-FILEPRINT 与控制打印过程的 CView::OnFilePrint()方法相关联。该方法调用其他 CView 帮助方法来使大量的打印过程自动化。可以直接用代码调用 CView::OnFilePrint()或者，MFC 调用它以响应具有 ID-FILE-PRINT 值的任何命令消息(如菜单上的打印命令)。

因为 OnFilePrint()是保护类成员，该项必须驻留在视图类的消息映像中，如下述：

```
BEGIN_MESSAGE_MAP(CMgView,CView)
// Regaired for printing
ON_COMMAND(ID_FILE_PRINT,CView::OnFilePrint)
END_MESSAGE_MAP()
```

CView::OnFilePrint()是类 CView 中较复杂的函数之一，此方法有 200 多行代码。CView::OnFilePrint()控制打印作业的整个路径。

由 CView::OnFilePrint()方法定义的 MFC 打印作业中基本步骤。框架以 CPrrintInfo 结构保存有关打印任务的大部分信息。CPrintInfo 中有几个值与页码标注有关，这些可访问值如表 13.1 中所示。

表 13.1 保存在 CPrintInfo 中的页码信息

成员变量或函数名	页码参考
GetMinPag/SetMinPage	文档第一页
GetMaxPage/SetMaxPage	文档最后一页
GetFormPage	要打印的第一页
GetTopage	要打印的最后一页
m_nCurPage	当前正在打印的页

页码从 1 开始，即第一页编为 1，不是 0。在打印过程开始时，框架调用视的 OnPreParePrinting 成员函数，传递给 CPrintInfo 结构指针。AppWizard 提供调用 DoPrparePrinting (CView 的另一个成员函数)的 OnPreparePrinting 实施。DoPreparePrinting 是显示 Print 对话框并创建打印机设备上下文的函数。

此时应用程序并不知道文档中有多少页，它使用缺省值 1 和 OxFFFF 作为文档第一和最后的页码。如果知道文档有多少页，则加载 OnPreparePrinting 并在向 DoPreparePrinting 发送之前调用 SetMaxPage 取得 CPrintInfo 结构，让用户能够指明文档的长度。

DoPreparePrinting 显示 Print 对话框，在返回时，CPrintInfo 结构包含由用户指明的数值。如果用户希望只打印有选择的部分页面，那么可以在 print 对话框中指定活动和结束码，框架使用 GetFormPage 和 GetToPage 函数获取这些值。如果用户不指明页面范围，那么框架调用 GetMinPage 和 GetMaxPage 并使用返回的值打印整个文档。

对要打印的各页文档而言，框架调用视类中的两个成员函数，OnPreparDC 和 Onprint，并给各个函数传递两个参数：CDC 对象的指针和 CPrintInfo 结构的指针。每次框架调用 OnPreParcDC 或 OnPrint 时，它都在 CPrintInfo 结构的 m-nCurPage 成份中传递不同的值。这样框架

告诉视应打印哪页。OnPrepareDC 成员函数也用于屏幕显示;它在绘画开始之前对设备上下文作调整。虽然 OnPrepareDC 与 OnPrint 的作用相似,但有一些差异;首先,CDC 对象代表打印机设备上下文而不是屏幕设备上下文。其次,CPrintInfo 对象作为第二个参数传递(该参数在 OnPrepareDC 为屏幕显示调用时是 NULL)。根据要打印的页面加载 OnPrepareDC 此对设备上下文进行调整,例如,可以移动视方向和裁剪区以保证打印文档需要的部分。OnPrint 成员函数执行实际的页面打印。"打印:缺省打印如何完成"显示框架如何用设备上下文调用 OnDraw 执行打印。更精确地说,框架用 CPrintInfo 结构和设备上下文调用 OnPrint,并且 OnPrint 把设备上下文传递给 OnDraw。加载 OnPrint 执行任何应在打印期间完成而且不为屏幕显示的复制(rendering),例如,打印页眉和脚注。然后从 OnDraws 的加载调用 OnDraw 以执行对屏幕显示和打印的复制。OnDraw 为屏幕显示和打印进行复制,这意味着应用程序是 WYSIWYG:"所见即所得"。然而,假设现在不是在编写 WYSIWYG 现用程序。例如,考虑一个文字编辑器使用黑体打印是通过显示控制码来在屏幕上指示黑体文字。在这种情况下,只能为屏幕显示使用 OnDraw。在加载 OnPrint 时,用对单独绘画函数的调用替代 OnDraw 的调用,该函数以书面方式绘出文档,使用不在屏蔽显示的属性。

打印机页面和文档页面在引用页码时,有时需要区分打印机页面的概念和文档的页面概念。从打印机视的观点出发,页面是一张纸。但一张纸并不一定等同于一页文档。例如,在打印新闻稿件时,纸张对折的地方,一张纸可能包含文档的第一和最后页(并排的)。同样,若打印电子表格,文档根本不由页面组成,而是一张纸可能包含第 1 到 20 行,第 6 到 10 列。

CPrintInfo 结构中的所有页均指打印机页。框架为通过打印机的每张纸调用一次 OnPrepareDC 和 OnPrint。当加载 OnPreparePrinting 函数指明文档长度时,必须使用打印机页。如果有一对一的吻合(即一张打印机页等于一张文档页),那么这是很容易的。另一方面如果文档页和打印机页不直接对应的话,必须在它们之间进行翻译转换。例如,打印电子表格。当加载 OnPreparePrinting 时,必须计算打印整个电子表格将需要有多少张纸,然后在调用 CPrintInfo 的 SetMaxPage 成员函数时使用该值。同样,在加载 OnPrepareDC 时,必须把 m _ nCurPage 翻译成将在特殊页面上出现的行和列范围,然后相应地调整视方向。

13.2 打印预览

框架还缺省实现了打印预览功能,并使得在应用程序中使用这一功能变得简单。打印预览将显示文档的一页或两页在打印时将出现的图象。这种实现方法还提供了一些控制,如用于打印被显示的页,移动到下一页或前一页,在显示一页和显示两页之间切换,缩放显示以便以不同的大小察看显示内容,以及关闭显示等等。如果框架知道文档的长度,它还会显示一滚动条,用于页与页之间的移动。要实现打印预览而不是直接在一个设备上绘制图象,框架必须用屏幕模拟打印机。为了做到一这点,Microsoft Foundation Class Library 实现了 CPreviewDC 类与 CPreviewView 类的联合使用。所有的 CDC 对象都有两个设备表。在 CPreviewDC 对象中,第一个设备描述表表示的是被模拟的打印机,第二个表示的是输出实际在其上显示的屏幕。

在响应 File 菜单的 Print Preview 命令时,框架将创建一个 CPreviewDC 对象。当应用程序执行设置打印机设备描述表特性的操作时,框架将对屏幕设备描述表执行类似的操作。例如,如果应用程序选择了打印的字体,框架将选择模拟打印机字体的屏幕显示字体;当应用程

序送出将要送到打印机的输出时，框架将把它送到屏幕上。

文档页的显示顺序和方式对于打印预览也不是同的。打印预览将一次显示一页或两页，在显示其它页之前将等待用户的提示，而不是从头到尾打印一个范围内的页。

除确保应用程序的 File 菜单中有 Print Preview 命令外，提供打印预览功能并不要求程序员做任何事。然而，我们可以用多种方法修改打印预览的特性，如果需要选择这样做的话。

13.2.1 打印预览过程

当用户从 File 菜单选取 Print Preview 命令时，框架建立一个 CPreviewDC 对象。每次应用程序执行设定打印机设备上下文特性的操作时，框架也在屏幕设备上下文上执行类似的操作。例如，如果应用程序选取一种字体，那么框架选取模拟打印机字体的屏幕显示字体。每次应用程序给打印机发送输出时，框架给屏幕发送输出。打印预览在各个绘画文档页面的顺序上也不同。在打印期间，框架继续打印直到某段页面完成。在打印预览期间，任何时候都显示一页或两页，然后应用程序等待，直到用户响应才显示其它页面。在打印预览期间，应用程序还必须应答 WM _ PRINT 消息，就像在原始屏幕上显示一样。

当预览模式激活时调用 On _ Prepare _ Printing 函数，就像它在打印任务开始时那样，传递给函数的 CPrintInfo 结构包含几个变量，其值可以假定以调整打印预置操作的某些特点。例如，可以设定 m _ nNumPreViewPages 变量以指明是否想要以一页或两页模式预览文档。

13.2.2 支持打印预览

CPreviewDC 类试图模仿显示屏上的打印机设备描述表，并出色的完成作业。CView::OnFilePrintPreview()方法提供 MFC 的打印预览功能，该方法必须被要求打印预览支持的打印预览命令处理程序引用。CView::OnFilePrintPreview()方法的视图类消息映像入口如下所示：

ON _ CDMMAND(ID _ FILE _ PRINT _ PREVIEW,CView::OnFilePrintPreview)。

由于预览窗口工具栏 Print 按钮没有对应的消息处理，映像中的 OnFilePrint()命令处理程序就不能工作，所以最好提供预览和打印入口，如下所示：

```
BEGIN-MESSAGE-MAP(MYView,CView)
// Required for printing
ON _ COMMAND(ID _ FILE _ PRINT,CView::OnFilePrint)
// Required for print preview
ON _ COMMAND(ID _ FILE _ PRINT _ PREVIEW,CView::OnFilePrintPreview)
END _ MESSAGE _ MAP()
```

以上是实现 MFC 的默认打印预览的全过程，它拥有某些令人叹服的功能。为了显示打印预览的方法，可以把 PrintPreview 命令与框架窗口菜单捆在一起，或者调用 CView::OnFilePrintPreview()方法。

13.2.3 高级打印预览

可以用几种很方便的方法修改打印预览的行为和表现。例如，可以：

- 使打印预览窗口显示滚动条，方便任何文档页的访问。
- 通过从当前页开始显示，使打印预览维持用户在文档中位置。

- 使打印预览和打印的不同初始化执行。
- 使打印预览以自己的格式显示页码。

若知道文档有多长并用适当的值调用 SetMaxPage，框架不仅在预置模式中而且也在打印期间使用此信息。一旦框架知道文档的长度，它就可以提供带滚动条的预览窗口，允许用户在预览模式中通贯文档，整页前和整页后移动。如还没有设定文档的长度，框架不能定位滚动框以指示当前位置，也就不增加滚动条。在这种情况下，用户必须使用预览窗口控制栏上的 NextPage 和 Previews Page 按钮来整页移动。

对于打印预览而言，给 CPrintInfo 的 m_nCurPage 变量赋一个值是很有用的，尽管从不为日常打印这样做。在日常打印期间，此变量载送来自框架的信息给视类，这就是框架如何告诉视要打印哪页的方式。

在打印预览模式启动时，m_nCurPage 变量以相反方向运载信息，即从视到框架。框架使用此变量的值确定应首先预览哪页。此变量的确定值是 1，因此开始显示文档的第一页。可以加载 OnPreparePrinting，设定此变量为在 Print Preview 命令激活时正在视察的页码。这样，应用程序在从正常显示模式转向打印预览模式时维持用户的当前位置。

根据是实际打印还是打印预览的不同，调用时 OnPreparePrinting 要执行不同的初始化。通过检查 CPrintInfo 结构中的 m_bPreview 变量可以确定。在打印预览开始时此变量设定成 TRUE。

CPrintInfo 结构不包括名为 m_StrPageDesc 的变量，它用于格式以单页和多页框显示在屏幕底部的字符串。虽然缺省情况下这些字符是“Page n”和“Pages n-m”形式，但可以从 OnPreparePrinting 内部修改 m_strPageDesc 并把字符串设成更完美的。

13.3 小　　结

打印是每一个可视化编程中不可缺少的内容和环节，读者对这方面内容是必须掌握的。本章简述了打印的基本概念和有关细节内容以及打印预览的应用创建。相信读者经过本章学习将能掌握如何更好地利用打印的相关技术。

第 14 章　上下文相关帮助

好的 Windows 应用程序的重要特性之一就是有个良好的帮助系统。你应考虑到，许多用户在使用你的应用程序之前，并没有花时间来阅读手册，这是相当重要的。

只要实现合理，你的帮助系统能迅速而容易地告诉用户他想知道的东西。幸运的是，Windows 已标准化了 Windows 帮助的用户界面。在本章，将看到在应用程序中如何实现 Windows 帮助，包括如何增加环境敏感的帮助系统，用来给用户更容易地访问相关的帮助主题。

14.1　上下文相关帮助

当 Microsoft 为应用程序增加帮助时，它已完成了大部分工作。它们已谨慎处理了显示给用户的用户界面，你需为你的项目增加特殊内容，并增加代码至你的应用程序中。在本章后面，你将看到任何添加你自己的帮助，本章我们将以 Windows 帮助如何工作作为基础开始为您介绍。

Windows 帮助由单独的、程序名为 WinHelp.exe 的程序来处理（简称 WinHelp）。此程序给用户展示了标准帮助对话框，显示你的帮助项目内容，如图 14.1 所示。

默认情况下，WinHelp 对话框提供 3 个标签页面，尽管你也可尽情地增加你自己的定制标签页面。目录标签页面将显示帮助文件的目录表，如图 14.1 所示。

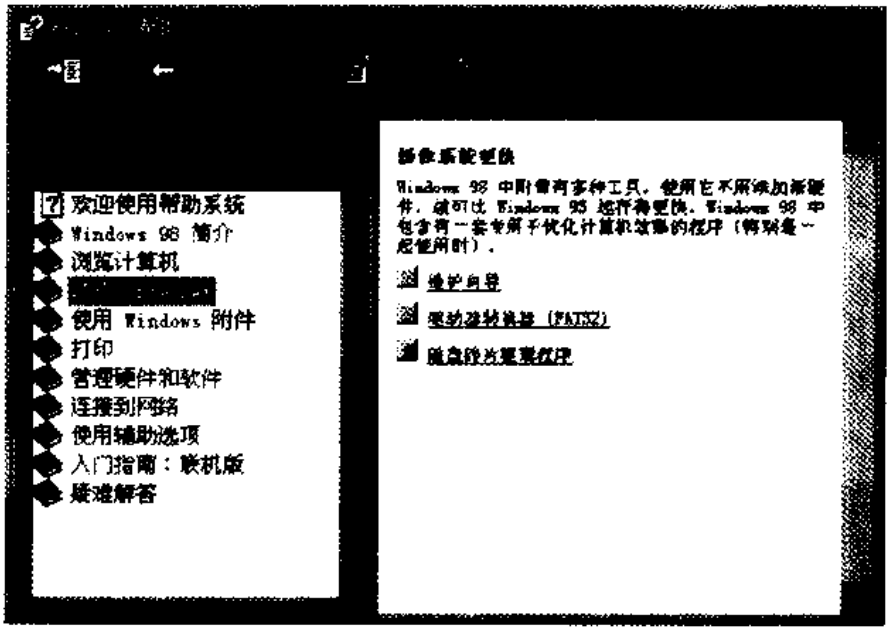


图 14.1　Windows 帮助一

索引标签页面显示帮助文件的主题列表，并允许用户在搜索列表中选择他感兴趣的主题，

如图 14.2 所示。

另外，WinHelp 允许用户通过用搜索标签页面(如图 14.3)来搜索帮助文件。

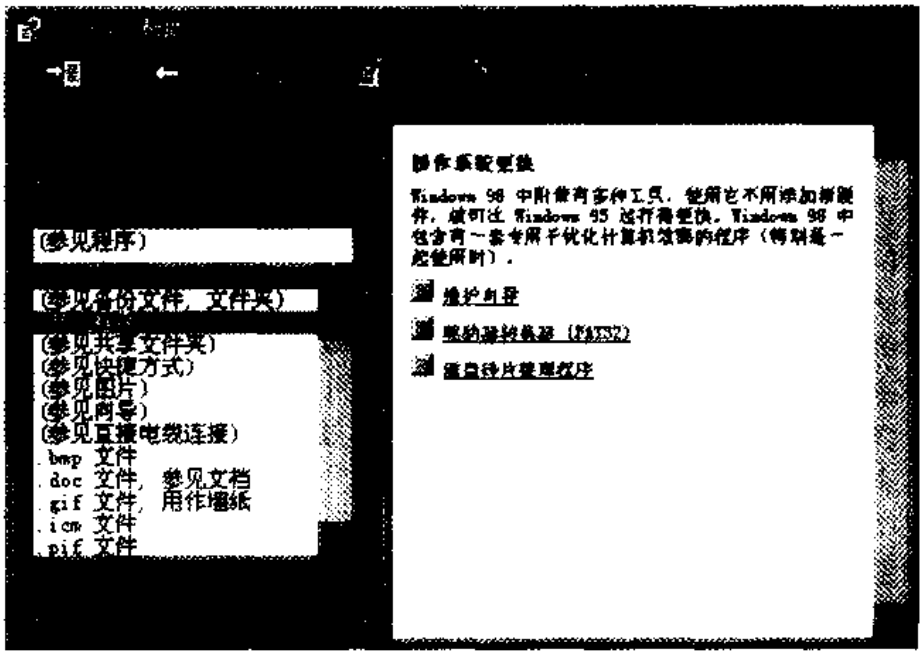


图 14.2 Windows 帮助二

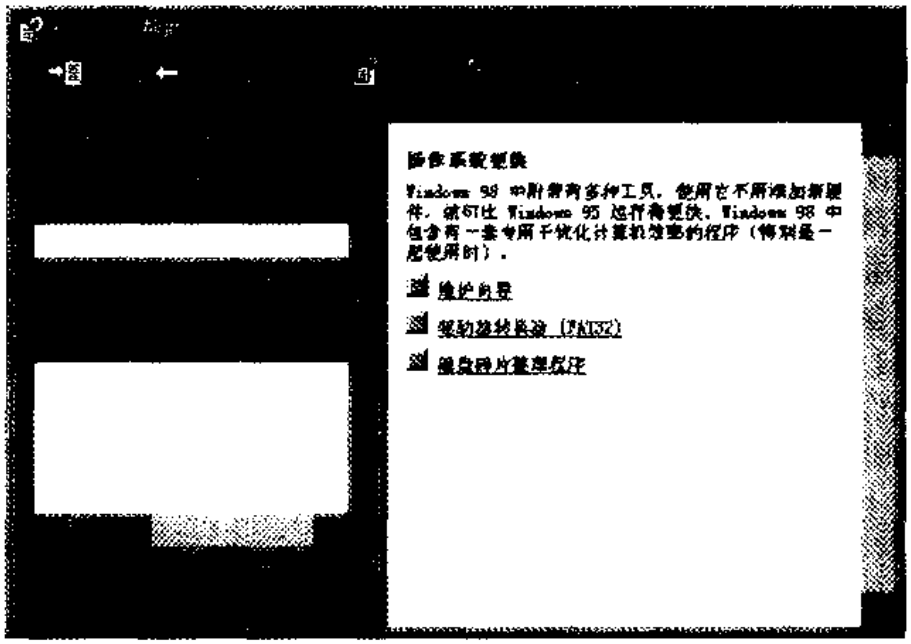


图 14.3 Windows 帮助三

此搜索标签页面允许用户搜索帮助文件中的单词列表，或在顶层的编辑器中键入关键字或短语。只要选择了一个单词或短语，用户将在底端列表中看到包含此单词的主题列表。击点显示按钮，将产生选定主题的帮助文本。

14.1.1 帮助文档

一个 Windows Help 文件是二进制文件，包含一个交互的超文档，它由一系列的主题组成。Windows Help 引擎(WINDOWS.EXE)显示这个二进制文件。Help 文件使用超媒体概念，包括超文本和超图形，使用户可以容易地从一个主题转到另一个题，并很快地找到弹出一个信息框。超图形是其内部定义有热点区的图象，热点执行与该文本键相同的任务，使用户跳转到一个不同的主题或弹出一个消息框。

Windows Help 系统的用户一般通过一个应用程序的 Help 菜单进入 Help 文件。支持上下文敏感的 Help 应用程序通过把 Help 文件找到的主题映像到应用程序的用户接口对象，如对话框、菜单、工具栏按钮、命令按钮和其他控件上，完成帮助功能。MFC 使用其内含的方法可以很容易地添加上下文敏感的 Help 到应用程序内。通过在应用程序的源文件和头文件创建消息映像项和消息处理程序方法来处理 WinHelp 的调用，从而将 Help 文件链接到你的 MFC 应用程序中。WinHelp 把 Help 主题和你的应用程序通过 Help 上下文 ID 联系起来，在 .HPL 文件和 .RTF 主题文件中说明这些上下文 ID。

用户可以按以下方法访问一个 Help 文件：

- 从应用程序的 Help 菜单中选择一项。
- 从活动窗口、对话框、消息框、工具栏按钮，菜单项中按 F1 功能键，按下 F1 将显示与当前活动窗口或控件(假设在 Help 文件中存在一个)相关的帮助主题。
- 按下 Shift + F1 组合键，鼠标单击 Help 工具栏按钮，或单击某个对话框标题栏的问号(?)按钮。任何一个动作都可能把应用程序放入上下文感感的 Help 模式。在这种模式下，鼠标光标变成带问号的箭头，表明 Help 模式已激活，当在 Help 模式下，用户可以单击任何窗口或控件来调用相关 Help 主题(如果存在)。

WindowsHelp 经过多年的发展，已成为一个非常复杂的超文档系统。通过 WinHelp API 中的有效宏和调用 Windows API 函数或自定义 DLL 可扩充 Help 文件功能。按以下步骤创建一个 Windows 帮助文件：

- 一个 Help 项目(.HPJ)定义所有的文件、位图、DLL 样式、应用程序上下文 ID，及其他所需元素。
- 一个或多个多格式文本文件(.PTF)包括特定代码一起的主题文本，这些代码用来格式化文本和支持不同的 Windows Help 特征。
- 分段的超图形文件(.SHG)，用于把交互的艺术工作和其他图形文件(例如位图或文件)在 Help 文件中显示。
- 内容文件(.CNT)定义有关 Help 文件的主题。
- Windows 98 Help 编辑器(HCW.EXE)。(如图 14.4 所示)

本章讨论以上全部内容，并介绍一些技术和工具，用来为你的应用程序生成专业水平的 Help 系统。通过本章学习，我们将举一个简单的应用程来说明 MFC 应用程序创建一个典型的 Help 系统的全部步骤。

14.1.2 AppWizard 对帮助的支持

为 Windows 编写的应用程序通常提供上下文敏感的 Help，允许用户对某个指定窗口、对话框、命令或工具条按钮取得 Help。MFC 使得为应用程序增加上下文敏感 Help 变得简单

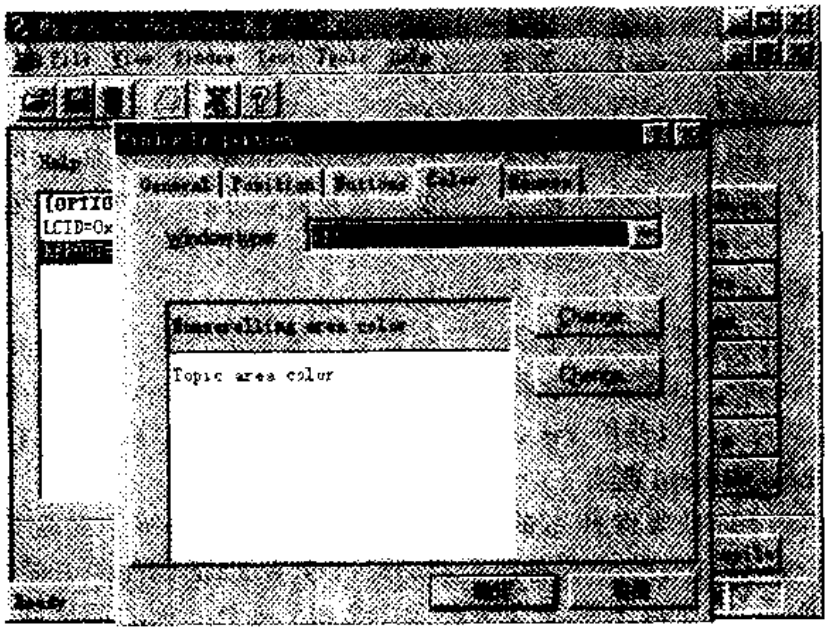


图 14.4 HCW.EXE 帮助编辑界面

用户可以有三种方式来访问 Help。

- 从菜单 Help 项上取得 Help。
- 通过按 F1 键实际取得任务的 Help,这种帮助称为"F1 Help"。
- 通过用 Shift + F1 激活"帮助模式"然后选取要取得帮助的用户界面对象取得帮助。这种帮助称为"Shift + F1 Help"。

MFC 框架中的 Help 子系统组成如下,在选择了 Context—Sensitive Help 选项时大多都被 AppWizard 支持。

- 带有几个命令的 Help 下拉菜单。对于新的 MDI 应用程序来说,这个菜单有两个拷贝:一个是没有打开任何文档的应用程序,一个是使用自己的菜单结构的各类文档的菜单。AppWizard 都支持这些菜单。
- WinApp 派生应用程序类中的几个消息映射入口。这些入口把 F1 和 Shift F1 命令结合到它们各自的命令处理器中。AppWizard 支援这些消息映射入口。
- F1 和 Shift - F1 的消息处理器。类 WinApp 支持这些处理器,并且 AppWizard 支持它们的消息映像入口。
- CWinApp::WinApp 成员函数,它调用 Windows 帮助程序。
- Help 的其它 AppWizard 支持,包括几个 Help 相关文件。包括含有 Windows 用户按 Help 入口的.RTF 文件,如 File 和 Edit 菜单。可以编辑这些文件以修改已提供的正文并添加自己应用程序特定的 Help 信息。
- 把应用程序中资源和命令 ID 映射到 Windows Help 中"帮助上下文"的机械和工具。MAKEHM 工具在稍后的"使用 MAKEHM"和 MAKEHELP.BAT 中介绍。

1)Help 菜单支持

框架实现下面的 Help 菜单命令。

2)Help 索引

打开有关应用程序内容屏幕上的帮助。

3)Help 使用

打开有关如何使用 Windows Help 解释的帮助。

4)更精确的上下敏感性

F1 Help 和 Shift + F1 Help 激活基于帮助上下文的 ID。框架中标准 Help 可以从窗口、对话框、消息框、菜单或工具条上获得的 Help 上下文。如果需要对这个机制更精确控制，可以覆盖部分机制。

5)Help 支持工具

将使用三个主要工具来开发应用程序的 Help 系统。它们分别是：AppWizard，MAKEHM 和 Windows Help Compiler。还需要一个编辑器，如 Windows 的 Microsoft Word，能够编辑丰富的文字格式(.RTF)文件。可以使用 Visual C++ 图形编辑器创建包括在 Help 文件中的位图。

6)选择 AppWizard 中 Help 选项

AppWizard 是实现上下文敏感 Help 的第一个工具。在 AppWizard 中设定 Context _ Sensitive Help 选项。于是 AppWizard 在链接整个帮助机制的(WinApp 派生类中提供消息映射入口。AppWizard 还创建一组开始文件，如下表所示。这些文件放在你的项目目录 AppWizard 建立的 HLP 子目录中。

表 14.2 AppWizard 提供的 Help 文件

文件	描述
_项目.HLP	Windows Help Compiler 用来编译 Help 的 Windows Help 项目文件
MAKEHELP.BAT	管理 Help ID 映像并调用 Help Compiler 的批处理文件
HLP.BMP	提供 Help 文件使用的各种位图文件
HLP*.RTF	以.RTF 格式包含电框架提供应用程序组成启动 Help 的基于 Help 文件帮助。项目文件(.HPJ)和 MAKEHELP.BAT 在项目目录中。其它文件在项目目录的 HLP 子目录中

14.1.3 常见帮助类型

一般将 Windows 98 下的帮助分为标准帮助和性能支持帮助。

1)标准帮助是帮助模型中最流行的形式，主要用作参考功能，而且像打印文档一样，包含目录、索引和跳转(与交叉参考相同)，"这是什么?"也属于标准帮助。标准主题主要有以下几种：

(1) 过程主题

标准帮助是帮助中最重要的，也是最流行的形式，而过程主题是标准主题最重要，也最流行的主题。它们在帮助文档中起着中心的作用，用来解释任务的目的，如何执行这些任务，以及什么是需要特别注意的地方。

下表列出了过程主题的通常组成部分。

注意：并不是每个过程主题都必须包含所有的组成部分。

表 14.3 主题的组成部分

组成部分	描述
操作	需要执行的操作
提示	操作中需要注意的地方
步骤	包含一个行为陈述，通常带有一个修饰符，有时包含一个反馈陈述。
技巧	快捷操作之类的东西
相关主题	与本操作相关的操作、概念等

(2) "这是什么?"主题

"这是什么?"主题在效果上是 WinHelp 4.0 的特征之一，尽管其能力在 WinHelp 3.x 中已经存在。该主题以弹出窗口形式出现，而且它从软件产品的界面中直接显示。如图 14.5 所示。

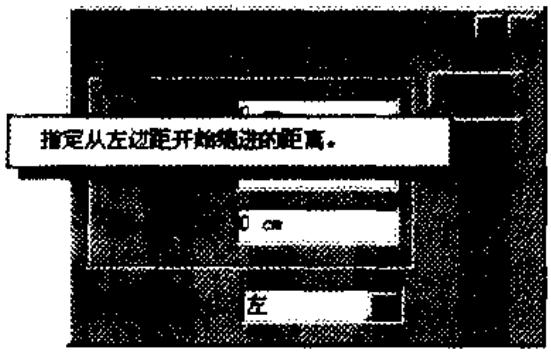


图 14.5 "这是什么?"帮助

"这是什么?"帮助主题显示方式依赖于该特征的实现方法，主要有：

- 右键点出感兴趣的元素(如果依据标准编程，用户点击鼠标右键，然后左键点击"这是什么?"命令以显示主题)。
- 点击在对话框中的标题条的帮助按钮，然后点击感兴趣的元素。
- 点击感兴趣的元素，然后按 F1 键。
- 按 Shift + F1 键，然后点击菜单命令或按钮。

(3) 定义性主题

当编写过程主题时，作者感觉到有必要解释对于某些读者来说不太熟悉的用词。在帮助中，这些词通常用定义性主题来解释。定义性主题与"这是什么?"一样，用弹出窗口来显示(如图 14.6 所示)。

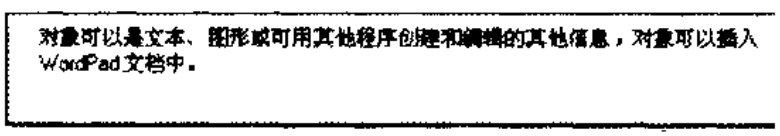


图 14.6 定义性主题帮助

(4) 疑难解答主题

用户进入过程主题是为了获取如何做某些事情的信息。如果产生一个问题,一个错误消息可以描述适当的动作。但是,通常情况下,用户可能遇到错误信息,但没有给出恰当的原因,这样,疑难解答主题就具有很高的价值。

广义来说,疑答解答主题在帮助系统中有两种类型:诊断主题和推荐主题。

诊断主题仅是对用户可能会遇到问题或问题的症状列表,用户只需点击跟所遇到的问题最相似的问题即可。如果问题列表短,一个诊断主题就够了。通常,需要一个诊断主题的层次结构,它引导用户到一个更加详细的陈述,最后到推荐主题。从用户观点来看,推荐主题是最重要的,它们使得用户能够解决他们的问题。推荐主题很像过程主题,通常有一个标准头,用来陈述问题或陈述正确的动作,以及描述更正问题的步骤,如果没有明确的方法,则给另一些提示或建议。

推荐主题包含更多的疑难解答逻辑,如果主题没有解决问题,用户可以跳转到更深层的推荐主题,这样的跳转可以将用户带到诊断文题,从而可此继续进行疑难解答。

2)性能支持帮助

性能支持帮助是与标准帮助完全不同的一种模型。标准帮助用途在于参考,而性能支持帮助提供给用户的是对话框。性能支持帮助分为"向导"和"指导"两种类型。根据提示执行一定的步骤来完成某种操作,这种类型的帮助叫做"向导"。"指导"是通过一系列的帮助主题,用此解释应用软件中涉及的相关概念和原理,并辅助以实况演示,用来说明执行某个操作的动态细节。

14.2 帮助的相关文件

1.多格式文本文件(RTF)

多格式文本文件是包含特殊代码的文本文件,这些代码描述文本格式和文档外观的其他方面,多格式文本文件是一个相对复杂的控制文档外观的符号和代码系统。因为 Windows Help 编译器只识别全部 RTF 规范的一个集,所以编排一个有多格式文本代码标识的主题文件相对容易。这些工作可由一些 Windows Help 实用程序和字处理软件完成。创作 Windows Help 文件时生成具体 RTF 代码的传统工具是 Microsoft Word。使用 Windows 下的 Word。

程序员往往不喜欢花费时间去完成可以被自己完成的枯躁的工作。如果你使用Microsoft Word,会很容易地创建一个主题文件,其中包含部分 Windows Help 编译器期望发现的嵌入的多格式文件代码。另外,Word 能够把一标准二进制 Word 文档(.DOC)透明地转换为 ASCII 多格式文本格式(.RTF)

提示:大的文档被转换成为 RTF 格式需花费较多的时间,你可以在创建主题文件期望使用标准的 .DDC 格式来节省时间,只有当准备编译 Help 文件时才将它转换为 RTF 格式文件。

在 Word 中,Help 主题的结束由分页符标明。如果一个分页符不存在,主题则变成嵌套在另一主题中的,这种安排对创建文档层次是非常重要的。Help 主题作为脚注通过在文本中嵌入专用符号而创建。在.RTF 格式的 Help 主题编码时最常使用的四个符号如表 14.4 中所示,用以下术语表示:

- 上下文串，定义 Help 主题的开始。
- 标题，单独的 Help 主题时使用。
- 关键字，查找 Help 主题。
- 浏览顺序，指明用户如何在 Help 主题之间转换。

表 14.4 在 RTF 文题定义中最常用的四个专用符号

符号	意义
#	后面跟随主题的上下文串
$	后面跟随主题的标题
K	后面跟随与主题相关的关键字表
+	后面跟随浏览顺序级别和索引

这四个符号对于 Help 文件的编码非常重要，它们被用于以下方面：

- 在.RTF 中定义的主题使用 Word 的脚注特性，用'#'符号标识主题的开始。例如：用 Help 上下文串 Sample_Topic 定义一个 Help 主题的开始，脚注如下：# Sample_Topic。
- 主题标题用做查找时在 Help 文件的 Find 到表框中显示的串。主题的标题用'$'符号定义。设置主题为 Sample_Topic，脚注如下：

  $ sample Topic
- 关键字用于在查找期间发现一个主题。关键字列表可以通过使用关键字符号(K)标识关键字列表的开始来定义。脚注如下：K.Jet;heli copter;Star Shlp;shuttlecraft
- 浏览顺序允许在你的 Help 主题中定义一个层次结构。例如，应用程序的主窗口可能拥有自己的 Help 文件子段，仅仅处理用户接口元素(我们将随意称该层为 main)，任何被包括在 Main 浏览顺序中的主题按数值索引排序。浏览顺序的定义使用符号"+"和随后的浏览顺序名称，一个冒号和浏览索引，如下所示：

  # Main:01

当用户单击 Help 窗口按钮栏的 Browse 按钮时，这些索引按照数值顺序查找。用户能够使用这种方法向前或向后查询主题。

提示：Help 文件的全部主题层次能够被浏览顺序的不同编组定义，以允许用户对 Help 系统方便地导航。

2.帮助项目文件(.HPJ)

定义一个有指定 Help 项目文件的 Help 文件，它是以.HPJ 为扩展名的 ASCII 文本文件。HPJ 文件包含几个段，每个段用于专用目的。适于各个段的选项使编译后的 Help 文件具有很广泛的能力。.HPJ 通过定义将被编译到 Help 文件中的资源文件中，包括窗口样式、大小、颜色等将用于最终 Help 文件的资源，给出 Help 编辑器为创建 Help 文件所需的信息。

注意：你必须认真选择 Windows 98 和 Windows NT 下的 Windows Help 编译器的.HPJ 文件名。虽然允许长文件名，但其中一定不能包括空格，如果文件名有空格，会使 Help 编译器产

生混淆，使 Help 文件不能正确地被编译。一个 Help 项目文件类似一个应用程序初始化文件(.INI)，其中每段之间由方括号分隔(例如[Config]段或[Files]段)。

3.定义 Help 项目文件段

.HPJ 文件的每个段都具有专门目的，并可能包括多个子段和选项，使用文本编辑程序的工具，修改这些段非常容易。

在以下的内容中将探讨在 Help 项目文件中最常用的段：

相关段请见表 14.5：

表 14.5 常用的段

段	描述
[OPTIONS]	列出构建一个 Help 文件时所用的 Help 编译器选项
[FILES]	列出用于 Help 文件的源文件
[WINDOWS]	定义在 Help 文件中使用的全部窗口
[MAP]	把应用程序主题映射到用户接口控件
[ALIAS]	为 Help 主题提供别名
[CONFIG]	列出在 Help 文件中使用的 WinHelp 宏或 DLL 宏函的调用
[BAGGAGE]	列出一些将被键入 Help 文件的外部文件，这些外部文件类似被定制 Help DLL 读出的资源

• [OPTIONS]段

[OPTIONS]段本身就是可选的，但你应当在 Help 项目文件中包含一个，用于定义一些可用的选项。如果你使用这个段，应把它列在 Help 项目文件的最前面，因为它告诉 Help 编译器构造一个 Help 文件时将使用哪些选项。除了设置编译选项，该段还允许你设置多个选项用于获得编译时构造信息。

• [FILES]段

Help 项目文件的[FILES]段包含用于构造 Help 文件的全部.RTF 主题文件表。作为一个通用例子，示例 Help 文件的[FILES]段如下所示：

```
[FILES]
C:\MYAPP\HELP\Topic.RTF;the main topicfile
C:\MYAPP\HELP\GLOSSARY.RTF;a glossary topic file
C:\MYAPP\HELP\COSTSVPT.RTF;a customer support topic file
```

• [WINDOWS]段

Help 项目文件的[WINDOWS]段是由一个窗口定义表组成的，这些窗口定义运用于数据显示的 Help 文件使用。Help 文件中的主 Help 窗口和所有辅 Help 窗口都在此列出，这个段中必须至少有一个窗口定义，每个窗口属性被 7 个项定义，第一个是窗口的名字，其余 6 个是由逗号分界的值，为创建一个窗口结出具体值。

注意：如果 6 个项的一个或多个值没有给出，逗号分隔的 6 个部分仍然必须存在！

窗口定义使用如下语法(虽然在这里写成两行,但窗口说明应在.HPJ 文件中仅写为一行)。

Windows Name = "Caption",(left,top,width,height),

state & buttons,(Scrolling_RGB)(Nonscrolling_RGB).state

HCW 允许你设置 Help 窗口各个方面,窗口定义语法的细节在 HCW Help 文件中有详细文档说明。图 14.4 显示 HCW 对话框,允许你修改 Help 窗口属性。

Help 窗口的窗口状态和用于 Help 窗口的按钮条中的预定义按钮——被样式位决定。样式位可能被联合起来作为窗口定义的 State & button 入口使用。

提示:为了具有更大的灵活性,你可以在一个 Help 项目中定义多于 256 个不同的窗口,其中之一是主 Help 窗口,其余都是辅窗口。

• 〔MAP〕段

Help 文件项目的〔MAP〕段是 .RTF 主题文件中使用的所有主题上下文串(主题 ID)与应用程序中统一上下文 ID 产生联系的地方。这就是当用户要求上下的敏感的 Help 时,应用程序如何决定显示哪个主题。一个映像语句的基本语法说明如下:

Topic Title Help ID

在这个语法中,TopicTitle 是在 .RTF 主题文件里定义的主题名称,Help ID 是数值,用于应用程序中的窗口,控制或菜单,标识 Help 标题。上下文联系可以直接放入 Help 项目文件或包含于一个外部文件内。

若你使用 # include 语句把文件包含在一个〔MAP〕段里,可以用风格或C ++ 注释,也可使用 # define 语句,但在与 HCW 一起使用 # define 语句时要注意,因为 HCW 无法识别任何算术类型,只能是常量值 1。下面是几个例子:

```
# define HELP_TOPIC1 5 //this is okey
# define HELP_TOPIC2 HELP_TOPIC1 + 5 //this doesn't work
# define HELP_TOPIC3 HELP_TOPIC1 + 10 //this doesn't work
```

• 〔ALIAS〕段

一个唯一的串标识符称为上下文串或主题 ID,它标识 Help 文件中每个主题。Help 文件中使用的上下文串对每个主题必须是唯一的。使用多个上下文串会使 Help 编译器产生混淆并在 Help 系统中产生错误。Help 项目文件的〔ALIAS〕段新的上下文串与已有的上下文串产生联系,新映像它们做为别名串。别名串对应分配给 .RTF 主题文件中的上题上下文串。

提示:〔ALIAS〕段是可选的。如果使用它,必须在 Help 说明文件中放入〔MAP〕段之前。如果〔ALIAS〕段不在〔MAP〕段之前,Help 编辑器就无法识别别名和上下文串的相互关系。

假设应用程序为一个对话框中每个控制定义一个上下文 ID,但 Help 文件仅包含一个用于全部对话框的主题,若一个用户打算获得对于对话框中的某个控件的上下文敏感的帮助,WinHelp 引擎在 Help 文件中将找不到上下文串,而是显示一个错误信息。这样可能会令用户恼怒,如果你定义别名,把所有的对话框的上下文 ID 映射到一个 Help 主题上下文中,当用户请求对话框中任何一个控制的上下文敏感的 Help 时,WinHelp 将会显示对话框主题。列表清单(下)显示了 Help 文件上下文 ID 的别名,这些 ID 对应于单独的 Help 文件主题 Sample_Di-

alog 的一个实例对话框中的两个编辑控件和两个单选按钮控件，它防止在 Help 主题没有被找到时出现令人恼怒的错误信息。列表清单的作用是映像几个上下文 ID 别名到唯一的 Help 主题上下文串。举例如下：

```
[ALIAS]
HID_EDIT_FIRSTNAME = Sample_Dialog
HID_EDIT_LASTNAME_Sample_Dialog
HID_RADIO_AGE = Sample_Dialog
HID_RADIO_BIRTHDATE = Sample_Dialog
```

• [CONFIG]段

Help 项目文件的[CONFIG]段列出 Help 文件被打开时 WinHelp 执行的宏。除了标准的 WinHelp 宏，你可以从 DLL(包括 Win32API)中注册子程序当作宏使用。

下列例子注册了虚构的 DLL 子程序 ShowMessage()，一个字符串作为参数与 WinHelp 一起使用。同时也创建一个按钮用于显示这些主题信息，并增加用于主题导航的 BrowseNext 和 BrowsePrevious 按钮。

```
[CONFIG]
RegisterRoutine(MyDll,ShowMessage,S)
CreateButton(btn_showmsg,Message,showMessage(Howdy1))
BrowseButtons()
```

注意：WinHelp API 在 Win32 SDK 中有完整文档。

• [BAGGAGE]段

Help 项目文件的[BAGGAGE]段列出 HCW 有储在最终二进制 Help 文件内的文件，非常像被链接到一个可执行的资源。这些文件一般被自定义 WinHelp Plls 和 MCI 命令使用。

提示：Help 仅使用一个 baggage 文件的文件名，不包含路径，但文件名是大小写敏感的；为了使用它，你必须像定义.HPJ 文件中的[BAGGAGE]段一样准确地定义这个文件名。

WinHelp 为 DLLS 提供回调函数，因此 DLL 能够从一个 Help 文件的[BAGGAGE]段中使用文件，从 Help 文件内部文件系统中访问数据。

14.2.3 目录(Content)文件(.CNT)

一个目录文件(.CNT)被 Windows 98 的 WinHelp 实现所使用，用来表现 Help 文件的常用接口，这个常用接口是 Help Topics 对话框。

这个对话框允许用户从目录页上的层次树中选择一个主题，这个对话框同时也提供 Find 页用于全文查找和 Index 页用于 Help 文件索引。目录页定义在对话框的目录页内显示的层次，它们通过 HCW 或一个文件编辑器被容易地创建。

全部 HELP1 目录文件(HELP1.CNT)如下：

```
:Base HELP1.HLP
1.Help Topics for HELP1
2.Help OverView = Help  Overview
```

2. The HELP1 User Interface = User_Interface2. Why Make a Help File? = Why_Make_A_Help File

2. A Jump Topic = Jump_Topic

这个代码告诉我们基本 Help 文件是 HELP1.HLP,"Help Topic for HELP1"是第一层主题。其他的是第二层主题而且被嵌套在主层,即第一层主题内。表中第一个入口仅是一个标题,没有有关一个 Help 主题的内容,仅有一个标题的行在 WinHelp 的新思想中被认为是一本书,那些包含 Help 主题内容的行是书中的页。一本书可以被嵌套在另一本中,这反映了交互式文档层次。

14.2.4 配置文件(.gid)

配置(.gid)文件是隐含文件,在帮助文件第一次打开由 WinHelp 产生,或当帮助文件更新时产生。GID 文件包含有关帮助文件的信息,主要有以下几方面内容:

- 在处理目录.CNT 文件后,主题的二进制表示,包含跳转和命令,及发现的主题存在者。
- 在目录文件中包含的所有帮助文件的文件名和标题。
- 来自其他帮助文件的关键词(如果目录文件使用了 Index 语句)。
- 列出具有全文搜索索引文件的文件列表(中文 Windows 98 无)。
- 帮助窗口和对话框的大小和位置。

从技术上讲,目录文件仅在用户第一次启动帮助文件时被 WinHelp 文件读取。如果您的计算机运行速度不是特别快,当帮助文件第一次启动时,将出现消息"正在安装帮助文件"。此时,一个具有.gid 扩展名的文件被创建,其作为显示帮助主题浏览器时,WinHelp 用来参考的信息。

若 WinHelp 不能在与目录文件相同的目录下创建.gid 文件,则它试图在"\Windows\Help"目录中创建一个.gid 文件。如果没有上述目录,则在 Windows 目录下创建。如果多个用户同时试图在网络服务器上构造一个.gid 文件,仅第一个用户创建了该文件,其他用户将看到一个动画窗口,直到第一个用户创建.gid 文件成功。

14.3 在自己的应用程序中加入帮助功能

本节将给读者讲述如何在自己的应用程序中加入帮助功能,以使自己的应用程序更加实用。

14.3.1 消息映射支持

要支持 Help 菜单命令、F1 帮助和 Shift + F1 帮助,AppWizard 给(FramWnd 派生的或 CMDIFrameWnd 派生的类消息映射添五个入口。此消息映射在 MAINPRM.CPP 文件中。运行 AppWizard 之后,选择 Context)_SenSitive Help 选项,对 MDI 应用程序的 MainFrame 类消息映射出现的形式如下:

```
//CMain Frome
BEGIN_MESSAGE_MAP(CMain Frame,CMDIFrameWnd)
```

```
//{{AFX_MSG_MAP(CMainFrame)
//NOTE_ the Classwizard.will.add and remove mapping macros here.
//Do NOT EDIT.what you see in these blocks of generated code!
ON_WM_CREATE()
//}}AFX  MSG_MAP
//GLobal help commands
ON_COMMAND(ID_HELP_INDEX,CMDIFrameWnd::On_Help_Index)
ON_COMMAND(ID_HELP_USING,(MDIFrameWnd::On_Help_Vsing)
ON_COMMAND(ID_HELP;CMDIFrameWnd::On_Help)ON_COMMAND(ID_CONTEXT_HELP.
(MPIFrameWnd::On_Contert_Help)
ON_COMMAND(ID_DEFAULT_HELP.(MDIFrameWnd::On_Help_Index)
END_MESSAGE_MAP()
```

对 SDI 应用程序来说,这段代码中的 CMDIFrameWnd 用 CFrameWnd 代替。

五个帮助相关的消息映射入口跟在备注//Global help 命令之后。表 14.6 解释这些入口中各个命令 ID 用途。

表 14.6 Help 相关命令 ID

命令 ID	用途
ID_HELP_INDEX	通过显示 Windows Help 索引对 Help 菜单上的 Index 项作应答
ID_HELP_USING	通过显示有关使用 Windows Help 的信息对 Help 菜单上的 Using Help 项作应答
ID_HELP	通过显示 Windows Help 中指定主题对 F1 作应答
ID_CONTEXT_HELP	通过把应用程序放入帮助模式对 Shift + F1 作应答
ID_DEFAULT HELP	在指定帮助上下文不能找到时使用

注意:所有这些命令都映射到 CMDIFrameWnd(在 MDI 应用程序情况下);在 SDI 应用程序情况映射下到 CFrameWnd 的成员函数。不同于放入消息映射的大多数其它命令,这些是具有类库预定义的处理函数,消息映射入口支持该命令。

应用程序的快捷键定义 ID_CONTEXT_HELP 的 Shift + F1 和 ID_HELP 的 F1。通过使用 Visual C++ 改变加速键表中的键值可以改变用于这些帮助函数的键。

当用户选择 Help 菜单命令(或使用文章"Help: F1 和空格 HIFT + F1SHELP"中"F1_Help"支持描述的上下文敏感 Help 技术之一)时,调用 WinApp 的 WinHelp 成员函数。于是,这个行动开始 Windows 帮助程序并把上下文信息传递给它。一旦用 ClassWizard 激活 Help 菜单命令后,就可以完成帮助,通过:

1)运行 MAKE HELP.BAT。

一旦使用 Visual C++ 创建应用程序的对话框、菜单和其他资源后,就可运行由 AppWizard 创建的 MAKEHELP.BAT 文件产生一个帮助(.HM)文件。帮助映射文件对应对话框、菜单命令和应用程序中其他资源的功能定义帮助上下文 ID。

2)编辑 Help 主题。

从你的项目 HLP 子目录中由 AppWizard 提供的 RTF 文件开始,删除不必要的主题,编辑

剩余的材料等等，并将有关菜单命令，对话框，工具条按钮等的主题加到应用程序上。每个帮助主题都要求一个帮助上下文 ID。使用在建立资源时提供的菜单命令对话框符号。这就是定义在应用程序 RESOURCE.H 文件中的符号：如 IDD_MY_DIALOG。在.RTF 文件中，上下文 ID 是字符串，不是数字，但应用程序和框架按数字，而不是字符串引用上下文 ID。由 MAKEHM 工具创建.HM 文件建立起上下文 ID 数字和上下文 ID 主题屏幕之间的联系。在 129 条"Help:编辑 Help 主题"描述了把这些 ID 映射到 Help 主题的过程和在.RTF 文件主题的编写。

3)编译 Help。

当 Help 主题完成编辑时，可以运行 MAKEHELP.BAT 创建 Help 图文件，然后进行 Windows Help Compiler。

14.3.2 F1 键和 Shift+F1 键的处理

F1 键通常是通过 AppWizard 在 RC 文件中所插入的键盘快捷键入口来控制的。与 F1 键相连的快捷键所传递的命令为 ID_HELP，它被映射到了 CWinApp 的 OnHelp 成员函数。

说明：在活动着的有模式对话框或菜单选择过程中，F1 键是由 Windows 内部来处理的，它同样也可以激发对同等 OnHelp 函数的调用。

CWinApp::OnHelp 函数会向最外面的框架窗口发送 WM_COMMAND HELP 消息，在 SDI 应用中，该消息是由主框架基类 CFrameWnd 的 OnCommandHelp 函数来控制的，它会用帮助上下文 HIDR_MAINFRAME 来调用 WINHELP。主框架窗口为最高层的窗口，它为 WM_COMMANDHELP 消息处理过程链的入口点，如果想要针对视窗或其他在派框架生类中我们自己的 OnCommandHelp 函数，应该向下直接将 WM_COMMANDHELP 消息处理到活动着的视窗中，当视窗接收到该消息时，它就可以用相应的帮助上下文 ID 来调用 WINHELP 了。

在 MDI 应用中，WM_COMMAND_HELP 消息在 CMDIFrameWnd 类和 CMDIChildWnd 类中都进行了映射。如果不存在 MDI 子窗口，框架窗口就将上下文设为 HIDR_MAINFRAME；如果有一个或多个子窗口，框架窗口就会向活动着的子窗口收送消息，活动着的子窗口再设置文档的帮助上下文。如果需要针对视显示帮助信息，就必须从 CMDIChildWnd 中派生出一个类，在该派生类中对 OnCommandHelp 函数进行映射，该函数再将 WM_COMMAND HELP 消息传送给视。

注意：对 F1，WM_COMMAND_HELP 的处理过程总是自顶向下的，应用首先将该消息收送给最上层的窗口，由它来选择是否将此消息传送给子窗口，这一路径同通常的命令处理路径不同。

Shift+F1 键当被用户按下时，或者用鼠标点了 Context Help 工具栏按钮时，给过映射的菜单命令消息就会传递给 CWinApp 的 OnContextHelp 函数，当用户对鼠标光标定位之后再按下鼠标键时，WM_HELPHITTEST；消息就会传递给最里层的接到鼠标接键消息的窗口，如果该消息在这一窗口中没有经过映射，那么下一个外层窗口就有机会得到该消息。

在 SDI 应用中，WM_HELPHITTEST 消息被映射到了 CFrome Wnd 类的 On_Help_Hit_Test 成员函数，它将帮助上下文设置为 HIDR_MAINFRAME，而在 MDI 应用中，该消息在

CMDIChildWnd 类和 CMDIFrameWnd 类中都有映射。如果鼠标光标落在了窗口中，相应的文档帮助上下文就会被设置，否则，上下文就会被设置成 HIDR _ MAINFRAME。

如果希望得到针对视的帮助上下文，只需在视类中对 WM _ HELPHITEST 消息进行映射而不要再把该消息传递给框架。On _ Help _ Hit _ Test 函数中的 Param 参数包含了鼠标的设备单位坐标值，改坐标值相对于窗口用户区域的左上角，其中 Y 值在高位部分，而 X 值则在低位部分。可以利用这些坐标值来为视窗中的一些项设置帮助上下文。

请记住对 Shift + F1 的处理过程总是自底而上的。消息首先被传送给最底层窗口，如果该窗口没有对该消息进行映射的话，那么它就接着被传递给了父窗口。

14.3.3 调用 WinHelp 函数

在你生成了一个项目帮助文件后，你需要能在你的应用程序中访问帮助事件。这可用::WinHelp()函数来完成，尽管你也会看到一些简单调用::WinHelp()的 MFC 函数。

1.::WinHelp()

Win32API 通过一个简单函数调用::WinHelp()来访问 WinHelp 系统的许多不同特征。::WinHelp()的原型如下：

```
BOOL WinHelp(HWND HWndMain, LPCTSTR lpszHelp,
UNIT UComand, DWORD dwData);
```

HWndMain 参数应传递请求帮助的窗口句柄，lpszHelp 参数应包含将要使用的帮助文件的路径。

UCommand 参数用于指定将要执行的操作，dwData 根据 UCommand 的值用于不同的方式。

UCommand 的常用值如下：

- HELP _ FINDER 显示 Help Topics 对话框，包含帮助文件的目录表。
- HELP _ HELPONHELP 显示有关使用 WinHelp 的帮助信息。
- HELP _ CONTEXT 根据环境 ID 显示主题标志，该环境 ID 显现在 hpj 文件的〔MAP〕部分。dwData 应包括该环境 ID。
- HELP _ CONEXTPOPUP 这与 HELP _ CONTEXT 相似，但帮助主题作为弹出式窗口被显示。dwData 应包括该环境 ID。
- HELP _ QUIT：通知 WinHelp 关闭全部的 WinHelp 窗口。dwData 被忽略。要获得关于使用::WinHelp 的其他命令，请参看 Visual Studio 的相关帮助。

2. 通过 MFC 使用帮助

Microsoft 基本类库对 WinHelp 系统提供了简易的访问，尤其是作用 AppWizard 已经创建了你的应用程序。

(1)CWinApp::WinHelp()

CWinApp 类支持 WinHelp()方法，它可简化显示帮助过程。CWinApp::WinHelp 原型如下：

```
Virtual void WinHelp(DWORD dwData, UNIT nCmd);
```

由此可见,此函数包括了 Win32API::WinHelp 函数,当然 MFC 为你保持了窗口句柄的踪迹和帮助文件路径。其中参数 dwData 和 nCmd 的值应与前面所示 dwData 和 uCommand 的值相同。

(2)MFC 帮助处理程序

若你已经用 Wizard 创建了你的 MFC 应用程序,并选择了环境敏感帮助选项,你将注意到几条已被加到你的应用程序相应的消息映像中的语句。它们被用于处理由 Help 菜单上的命令产生的消息,该菜单是由 AppWizard 产生的。此消息映像项目的例子如下所示。

```
ON_COMMAND(ID_HELP_FINDER,CFrameWnd::On_Help_Finder)
ON_COMMAND(ID_HELP,CFrameWnd::On_Help)
ON_COMMAND(ID_CONTEXT_HELP,CFrameWnd::On_Context_Help)
ON_COMMAND(ID_DEFAULT_HELP,CFrameWnd::On_Help_Finder)
```

ID_HELP_FINDER 消息由 Help 菜单上的 Help Topics 命令传送。

处理函数 On_Help_Finder()仅需调用 CWinApp::WinHelp(),如下所示:

```
AfxGetApp()→WinHelp(DL,HELP_FINDER);
```

ID_HELP 消息在用户按 F1 键时被发送。On_Help 函数将设法显示与当前窗口相关的帮助主题。若没有,则显示默认帮助主题。

ID_CONTEXT_HELP 消息在用户按下 SHIFT + F1 或去帮助模式工具输入帮助模式时被收送。OnContextHelp()处理程序将应用程序置于帮助模式。一旦在帮助模式中,用户输入进行不同的处理不同,MFE 将试着对任何被击点的控件显示帮助,而不为控件发送命令信息。

对特殊环境,若发现没有帮助主题,发送 ID_DEFAULT_help 消息。MFC 通过显示帮助文件的目录表来处理的消息。

另外,CWinApp 类支持 OnHelpUsing()方法,该方法将显示关于使用 WinApp 界面的帮助。你可为此容易地给应用程序增加支持,只要增加带有 ID_HELP_USING 的 ID 菜单项,并增加下列消息映像入口:ONCOMMAND(ID_HELP_USING,OnHelpUsing)。

在大多数情况下,为请求帮助使用预定义的命令 ID,且简单地使用消息映像入口来调用在 CWinApp 类中的合适处理函数,这是相当容易的。然而,在某些情况下,你可能从你的应用程序直接调用 CWinApp::WinHelp()。

14.4 小　　结

ContextHelp 类为读者提供了一个强大的上下文相关帮助内容。上下文相关帮助对于编写一个优质的应用程序来说是很必要的,望读者能通过本章学习对其有关内容有所了解。本章讨论了如下内容:

- 上下文帮助的有关内容和概念
- 帮助命名的处理与应用
- 头文件的要求和设置
- 创造相关帮助的资源要求
- 创造相关帮助的帮助文件要求

第 15 章　数　据　库

15.1　数据库管理

计算机技术已使世界范围的商务工作发生了永久的变化。过去堆满在仓库中、橱柜里的信息资料，如今用鼠标点一下就可以立即取出来，国外顾客的订货也可以立即在制造车间的地板上加工处理。虽然在 20 年前，很多这样的信息就已经被传输到公司的主机数据库上了，但直到今天，主机数据库仍运行在批处理方式下。如果要执行查询，就要通知管理信息系统(Management Information System：MIS)部门，由该部门尽快交付查询的数据。显然这并不是足够快的。

在关系数据库发展的同时，另外两种技术使得基于客户机/服务器(Client/Server)结构的数据系统迅猛发展起来。第一种重要的技术是个人计算机，它价格低廉，并像 SQL Server 和 Oracle 等容易地建立和管理数据。用户们对于不断升级系统已经习惯了，因为变化是如此之快，甚至极为先进的系统都在不停地降价。第二种重要的技术是局域网(Local Area Networks：LANS)的开发，它们正在世界范围内集成到办公室里。虽然用户本地终端连接到公司主机这种结构，但现在文件可以存在办公室的微机里，并且连接到网络的所有计算机都可以得到该文件。苹果公司 1984 年开发出的 Macintosh 微机引进了崭新的图形用户界面技术。这样，计算机变得又便宜，功能又强大，并且很容易使用，而且它们可以从远地访问，大量数据可以下载到部门的服务器中。更不用说微软公司的 Windows 95/98、Windows NT 了。在这两种计算机技术快速变化和发展的时候，一种新型的系统出现了，这就是客户机服务器系统。在这种系统中，处理过程被分割成客户机和数据库服务器两部分，原来基于主机系统的应用编程也急剧地转向这种模式。

客户机服务器是一个处理模型。在这个模型中，单个应用被分割成多个处理过程(前端和后端)。并且这些处理过程可互相协作(对终端用户是透明的)，作为一个统一的任务完成处理工作。客户机/服务器体系将处理器结合在一起，提供一个单一系统的形象(初象)。可共享的资源作为客户机放置，得到允许得到的服务。这种结构是无限递增的。反之，服务器也可以成为要求得到网络上其他服务器服务的客户机，等等。

15.2　SQL

结构化查询语言 SQL(Structured Query Language)是专门用于处理关系型数据库的基于文本的语言。由 IBM 最早提出，但后来为绝大多数主要的客户/服务器(Client/Server)数据库提供商所采用，它是 ODBC 最基本的部分。

要了解结构化查询语言 SQL，首先必须了解关系模型中关系对象的概念和一些常用的 SQL 语句。

15.2.1 SQL的对象

SQL对具有关系模型的各种对象进行操作,这些对象主要有:

1)SQL表、行和列。

表是关系数据库的核心概念,像报表一样,由行与列组成。行就是记录,是表中的数据项。列是域。

2)SQL数据库。

数据库是具有关系的一组表。

3)SQL语句。

语句是符合SQL语法的字符串,一般以分号结尾。它可以处理关系对象。SQL不区分字母的大小写,并根据SQL语句功能分为数据定义、数据处理、数据控制和数据查询四类语句。

15.2.2 SQL的Select查询语句

Select语句是SQL的灵魂所在,当你想从数据库中取出数据时,就会用到SELECT。它的基本语法十分简单,虽然可以有许多附加的内容,这在以后可以看到。SELECT语句最简单的形式为:

SELECT select_list FROM table name

Select-list可以有不同的形式,可以是最简单的,也可以是最通用的(*)。Select-list将从选择表中选择所有列。下面的例子将对表中的每一行检索MyTable中所有的列:

SELECT * FROM MyTable

还可以从表中选择指定的一个字段。例如:

SELECT Empld EmpName FROM MyTable

另外,可以在Select-List中定义固定的文本,这在C++应用程序中非常重要,而且在报告生成效用方面(加入固定文本后的格式效果)是很常用的。例如,下面的语法将返回'……',仿佛它是另外一列的值。

SELECT EmpId,'……',Emp Name FROM MyTable

虽然你在C++应用程序中可以找到较好的格式数据,但这在向用户提供结果时是非常有用的。

1.WHERE从句

在上面的例子中,我们已检索到表中所有的行,你可以用WHERE从句限制这些返回行。为了操作返回的行,WHERE可以规定许多不同情况的表达式。最简单的表达式有下面的比较操作符<、>、<=、>=、=或<>。例如,可以用下述语句检索所有的职员号为123的行:

SELECT * FROM Employee WHERE Empld = 123

可以用AND或OR把几种情况合起来,例如:

SELECT * FROM Employee

WHERE Salary > 50000 AND Salary < SELECT * FROM Employee WHERE Rept = 'MIS'OR Rept = 'Salas'

另外,可用 NOT 操作符否定某种状态,如下列语句将选出不在 Human Resources 的工作者的行:

SELECT * FROM Employee WHERE Not Dept = 'HR'

这个简单的例子相当于下述的使用不等式操作符的语句:

SELECT * FROM Employee WHERE Dept < > 'HR'

2.表达式

在单列名字的位置上,可以使用多列的表达式,多列用 + 、- 、/、* 连接。表达式可用在 Select _ List 中,例如:

SELECT PartNum,PartCost + ExtraCost FROM Parts

WHERE 也可包含表达式,例如:

SECECT Part NumFROM Parts WHERE (Ertralost/Part cost) > 25

3.Like 谓词

除比较操作符以外,SQL 为字符串提供一种专用的比较操作符。LIKE 允许你选择与某种形式相匹配的字符串的行。在这种形式中,你可以包括任何的字符串,包括"%"和"_"。例如,选择 Tile 中以"Data"开头的字符串的行:

SELECT Chapter Name FROM cgapters WHERE Title LIKE 'ata%'

4.IN 谓词

可用 IN 谓词简化一些 WHERE 从句。该从句用某一组值中的一个值来选择行。例如:

SELECT EmpNum FROM Employee

WHERE Rept = 'MIS'OR Rept = 'HR'OR Dept = 'Satas'

可用 IN 谓词简化为下列的语句形式:

SELECT EmpNum FROM Employee WHERE Dept NDT IN('MIS','HR')

注释:IN 谓词是在主存储水平一致的情况下被描述的,并不被所有驱动程序支持的。

5.BETWEEN 谓词

在许多情况下,需要选择到值在某范围内的行,例如:

SELECT EmpNum FROM Employee WHERE Salary > 5oo AND Salary in < 30000 2000 AND 3000

这个查询可用 BETWEEN 简化为下列语句形式:

SELECT EmpNum FROM Employee WHERE Salary BETWEEN 500 and 3000

同其他比较操作符一样,也可以把 BETWEEN 用于非数字列中,如:SELECT EmpNum FROM Employee WHERE Name BETWEEN 'Andersen'AND'Baker'。为了选择不在给定范围内的行,你可以加入 NOT,如:NOT BETWEEN。

注释:BETWEEN 谓词是在同一级主存储器情况下被定义的,它不允许被其他驱动程序支持。

6. DISTINCT 关键字

在许多情况下，你想找到一个列在一组结果中的不同的值，可用 DISTINCT 关键字做到这点，它会删除结果中重复的行值，如：

SELECT DISTINCT Dept FROM Employee

如果没有 DISTINCT 关键字，你就会得到 Employee 表中每行的 dept 值，而这些值有很多可能是重复的。

7. ORDER BY 从句

缺省时，通过查询返回的记录是任意顺序的，如果你想让行按特定的顺序返回，需要用 ORDER BY 从句，它允许你指定一个或多个列用来排序返回的记录。对每个列，可用 ASC 或 DESC 指定升幂或降幂顺序。

下面例子将返回按 Employee ID 升幂排列的记录：

SELECT EmpId FROM Employee ORDER BY EmpId ASC

如果你接 ORDER BY 以句中指定了多个列，首先从许多记录中挑选出来，组成第一列；然后，在第一列中选出相等值的记录组成第二列。例如，下面的语句，将对记录依据 salary 按升幂顺序排序，然后，具有相同 salary 的记录依据 EmpId 按降幂顺序进一步排序。

SELECT Empld, salany FROM Emplyee ORDER BY Salary ASC Empld DESC

8. 集合

在许多情况下，你可能要在由结果集中返回的所有记录的基础上计算数值，例如，计录所有返回记录中的某一个列的总数。这种类型的计算可以用集合函数实现，集合函数如下所示。

AVG：一个列值的平均数；

MAX：在结果集中列的最大值；

COUNT：返回行的数目；

MIN：在结果集中列的最小值；

SVM：在结果集中这些列值的总和；

也许最常用的集合是得到返回行的数目，下面的例子将返回 Employee 表中行的数目：

SELECT COUNT(*)FROM Employee

如果你用了 COUNT(*)，查询中所有的行都被计算。但是，如果指定了一个列的名字，只有列值不为空的行被计算。例如，在一个或多个行中 Dept 列的值为空，那么下面的查询语句仅返回有值的 Dept 行的数目：

SELECT COUNT (Dept) FROM Employee.

也可把 DISTINCT 关键字与集合功能结合在一起使用，如在下面的查询中，将返回曾用过的不同的 department 代码的数目：

SELECT COUNT (DISTINCT Dept)FROM Emloyee

在许多情况下，要在结果的某组行中使用集合功能，这可通过加入 GROUP BY 从句做到，它可以指定用来组成行的列，而这些行包括在集合计算中。下面的例子将产生一个 department 代码表，随后是每个department的 salaries 总数。

SELECT Dept,SUM(Salary)FROM Employee GROUP BY Dept

前面你看到了 WHERE 从句是如何控制查询的返回行的,与之相似,加入一个 HAVING 从句,可以限制用集合功能返回行,下面的例子将只返回部门内的总工资数少于 $200000 的部门:

SELECT Dept SU M(Salary)FROM Employee

GROUP BY Dept

HAVING SUM(Salary) < 200000

在更复杂的查询中,弄懂 WHERE 和 HANING 从句如何一起使用是比较困难的。当对集合进行操作时,按下述步骤进行:

1)选择符合 WHERE 从句的所有的行。

2)计算集合的值。

3)用 HAVING 从句限制了由集合得出的行。

9. 联合

对关系数据库最重要的一种操作是联合,它返回的是从几个不同的表中集合起来的数据的行,这些不同的表是在查询过程中相互联合的。如果你用的数据库是标准化的,那么联合更重要。

下面看一个例子,假如你有一个关于 parts 信息的 parts 表,包含 descriptions,还有一个关于 parts 价格信息的 prices 表。下面的查询将对两个表进行联合操作:

SECECT Description,price FROM parts,prices

然而,你可能对结果感到惊奇,上面的例子返回两个表的 Cartesion 结果,这个结果是 parts 表和 price 表的所有行可能结合的名字,在很多情况下,这个结果无用。

为使结果更有用,可以加入 WHERE 从句。最常见的情况是两个表中有共同的列。例如,假设 price 表和 parts 表都包括 partNum 列,可以用下面的查询产生 parts in descpription 和 price 匹配的行:

SELECT Description,Price FROM Parts prices WHERE parts partNum = prices,partNum

注释:在下面的例子中,我们用 parts、partNum 和 prices,partNum 而不是 PartNum,这在联合中很必要,它区分一个表的 PartNum 与另外一个表中有相同名字的列。

10. 相关的名称

为了简化一些语句,SQL 允许使用相关的名称,可以省略冗长的语言,下面你会看到这对于复杂的语句是必要的。如:

SELECT MyFirst Table Name,MyFirstTable.Num,MySecond Table

My Second Table Tim FROM My First Table,My Second Table Date,

WHERE MyFirstTable.id = My second Table.Id

我们使用相关的名称,用 f 代替 My FirstTable,用 s 代替 Mysecond Table.则生成等价语句如下:SELECT f.Name,f.Num,S.Date,S.Time.

FROM MyFirstTable f,MySecondTable s WHERE f.id = s.id

这时,相关名称不影响语句的意思。在更复杂的语句中,你可能用了很多次同一个表,这

时，相关名称可简化你用到的表。

11. 外部联合

到目前为止你看到的仅是内部联合，它可能不能产生联合表的 Cartesian 结果。如果用下面的语句产生一个联合：

SELECT * FROM Employee, Department

WHERE Employee.DeptNum = Department, DeptNum.

数据库会收集 Employee 表中的所有的行与 Department 表中的行相匹配。因为没有对应的入口在 Employee 表中，所以不会有结果。为了产生这样的一个查询：返回的行不包括在一个表中匹配的行，应使用外部联合。

例如设想你要为每个 employee 选择行使用外部查询：SELECT * FROM

| Employee LEFT OUTER JOIN Department ON

Employee DeptNum = Department DeptNum|

在这个例子中，我们用到了 LEFT OUTER JOIN 以确保左边 Empcoyee 中的所有行都在结果中。还可用 RIGHT OUTER JOIN 以确保右边的表，或用 FULL OUTER JOIN 确保两个表中的所有行都在结构中，不管是否有匹配的行出现在另外的一个表中。

12. 子查询

在我们看到的 WHERE 从句的例子中，已比较了文本值或比较了两个列。其实，也可以在 WHERE 从句中用另一种查询得到信息，这就是子查询。使用子查询的第一种情况是用 EXISTS(或 NOT EXISTS)谓项。设想挑选一些没有 Employee in department：

NOT EXISTS(SELECT * FROM Employee)

这将返回所有 department 的名字，因为在 Employee 表的记录中没有找到 department number。

还可以挑选 department 的表至少有一个高工薪的 employee 的 department，这个查询语句如下：

```
SELECT DeptName FROM DEPARTMENT d
WHERE EXISTS
(SELECT * FROM Employee e
WHERE e.salary> 100000 AND d.DeptNum= e.DeptNum)
```

使用子查询的第二种情况是用 IN 或 NOT IN 谓项，例如可以选择目前有 employee 分配去的 department：

SELECT DeptName FROM Department

WHERE DeptNum IN(SELECT DeptNum FROM Employee)

第三种使用子查询的情况是使用 ANY 或 ALL 进行比较。例如，有独立的 employees 表和 executives 表，想选择一些比普通 employee 报酬高的 executives，可用下面的查询：

```
SELECT Ex.Name FROM EXECUTIVE EX
WHERE Ex, Salary> ALL
(select Emp. Salary FROM EmployeeEmp)
```

同样,也可用 ANY 修饰语选择一些 executives,至少有一个普通 employee 的报酬比他高。查询语句如下:

```
SELECT Ex.Name FROM Executives EX
WHERE Ex.Salary < ANY
(SELECT Emp.Salary FROM Employee Emp)
```

13.联合查询

SQL 允许从两个独立的查询中通过 UNION 关键字得出一个结果。缺省时,双重的行可从结果中移走,虽然可用 UNION ALL 而不是 UNION 来中止。

还有,你可以用指定一个 ORDER BY 从句来选择最终的结果,在联合中,在最后查询中使用 ORDER BY 从句。例如,可用下面的查询得到一个在 MIS 和 Sales department 中 employee 的记录:

```
SELECT * FROM Employee WHERE Dept = 'MIS'
UNION
SELECT * FROM Employee WHERE Dept = 'SALES'
```

当然,可以有几种方法产生上述查询的结果,这是一个 UNION 使用的简单例子。

15.3 ODBC

ODBC 是开放式数据库互连(Open DataBase Connectivity)的缩写。它是 Microsoft 公司对数据库进行访问的标准应用程序编程接口(API),是 Windows 开放式服务体系结构 OSA(Open Server Architectwre)的一部分,它用结构化查询语言 SQL(Structured Query Language)来存取数据。程序开发者在 ODBC 层面上开发的应用程序,只要配上为某数据库管理系统(DBMS)专门编定的驱动程序,即可实现对该数据库的访问,无须再修改应用程序。由于 ODBC 是基于 SQL 的,所以它能向数据库提供完善而一致的接口。同时由于大多数 DBMS 商为了占有 Windows 用户市场,都开发了 ODBC 驱动程序,因此基于 ODBC 开发的数据库应用程序对各种数据库具有广泛的适用性。

15.3.1 ODBC 的部件

ODBC 由下列几个部件组成:

- 应用程序
- ODBC 管理程序
- 驱动程序管理器
- ODBC 光标库
- ODBC 数据库驱动程序
- ODBC API

只有熟悉了这些部件,才能正确有效地理解和编写应用程序。

下面分别介绍各个部件。

1. 应用程序

应用程序是程序开发者为实现对数据库的操作而编制的程序，建立在 ODBC 上的应用程序在 DBMS 改变时，无须作修改。

应用程序在使用前须加入头文件和库。若使用的是 Visual C++ 和 MFC 的 ODBC 类，需让 AppWizard 把这些类添加进应用程序。

2. ODBC 管理程序

ODBC 管理程序是一个控制面板(Control Panel)程序，对使用 ODBC 很重要。它管理已安装的驱动程序，协助管理数据源，还提供 ODBC 函数调用的跟踪功能。

安装 Visual C++ 6.0 时，ODBC 的图标应加入控制面板窗口。若未出现，需重装 Visual C++ 6.0。ODBC 管理程序同 ODBC 之所有动态链接库部件一样，是一个可再发行的文件。图 odbc 显示了 ODBC 管理程序的主窗口，它由一个数据源列表组成，表中每项有一个用户定义的数据源名和那个数据源所使用的驱动程序名。该窗口是 ODBC 注册段的一个接口，数据源的配置存于此处。

3. 驱动程序管理器

包含在 ODBC32.DLL 中的驱动程序管理器是 ODBC 最重要的部件，它负责处理 ODBC 的初始化管理 ODBC 注册段，同时还提供指向 ODBC 函数项和检验参数有效性的驱动程序。连接数据源的操作实际上是通过驱动程序管理器完成的。

4. ODBC 光标库

它是动态链接库(ODBC32.DLL)，存在于 ODBC 驱动程序管理器和驱动程序之间，用于处理数据的滚动。

5. 驱动程序

ODBC 驱动程序也是动态链接库，提供应用程序到指定数据库内数据的一个接口，这一接口对于支持的所有数据是几乎一致的。Visual C++ 6.0 提供了 SQL Server、MS Access、MS-Foxpro、MS-Excel、DBASE、Paradox、Text File 和 Oracle in ODBC 驱动程序。

6. ODBC API

这是一个由调用函数、出错信息和标准 SQL 语法组成的库，在 DBMS 上操作数据。

Visual C++ 融合了许多关键的 ODBC 组成部分，包括所需的头文件库和 DLL。它还提供了一些工具，包括 ODBC Drive Manager、ODBC carsorLibrary，以及 ODBC Administration 控制面板，还可用于配置数据源。Visual C++ 还用 Microsoft SQL Server 提供了一个 32 位的 ODBC 驱动程序。

ODBC 是一个工业标准，是 Microsoft Windows Open Services Architceture(WOSA)的一个组成部分。下图说明了应用程序，数据库类，ODBC 和 DBMS 之间的关系。

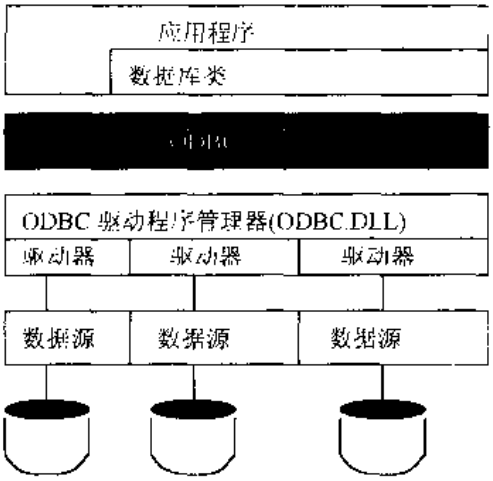


图 15.1 应用程序、数据库类、ODBC 和 DBMS 之间的关系

15.3.2 安装驱动程序和数据源

ODBC Administrator 用于注册和配置在网络上局部或全部可用的数据源。ClassWizard 使用由 ODBC Administrator 提供的信息建立应用程序中的代码,用户与数据源链接起来。

必须使用 ODBC Administrator 增加和消除数据源。根据 ODBC 驱动程序的不同,还可以创建新的数据源。

在设置期间,选择要安装的 ODBC 驱动程序,然后安装使用 ODBC Administrator 的其它驱动程序。

注意:ODBC Administrator 在 Setup 期间安装。如果选择 Installation 而不选择"Database (ODBC)files"复选框,则需要再次运行 Setup 以安装必要的文件。

1.安装和删掉 ODBC 驱动程序

1)通过在 Control Panel 中双击其图标打开 ODBC Administrator。显示 Data Source 对话框。

2)选取 Drivers 按钮。

要增加驱动程序,选择 Add,然后指定到 ODBC 驱动程序的路径。选取一个或多个要安装的 ODBC 驱动程序,然后选择 OK。ODBC Administrator 把文件复制到磁盘上。要消除驱动程序,选取要删除的驱动程序,然后选择 Delete。如果对随后的消息框回答 Yes,那么 ODBC Administrator 删除驱动程序和从 Data Source 对话框中删除与其有关的数据源。ODBC Administrator 并不从磁盘中清除这些文件。

3)选择 Close 返回到 Data Sowrles 对话框,再一次选择 Close 退出 ODBC Administrator。

提示:要了解有关驱动程序的信息,加亮该驱动程序并选择 About。假定我们用的是 Microsoft Windows 95 环境,并假定已经有一个用 MS-Access 建立好的数据库 Patient.mdb。我们要用 visual C++ 6.0 来编制一个应用程序,对该数据库进行输入,查询,增加和删除等操作。

在使用 MFC ODBC 数据库类编程之前,必须先进行一些准备工作,使我们将要编制的数据库应用程序能对指定数据库中的数据进行操作。

2. 安装数据源

要想使用 ODBC 数据库类来实现对数据库 patient 操作编程之前我们首先必须将数据库 patient 注册到 ODBC 数据源。数据源是通过 ODBC 管理程序注册的。注册步骤如下：

1)先在 Windows 95 的控制面板上用鼠标双击“32bit ODBC”，打开 ODBC 管理程序，屏幕上将显示 ODBC 管理程序的主窗口(见图 15.2)。主窗口是一个数据源的列表，列出了已注册的数据源。

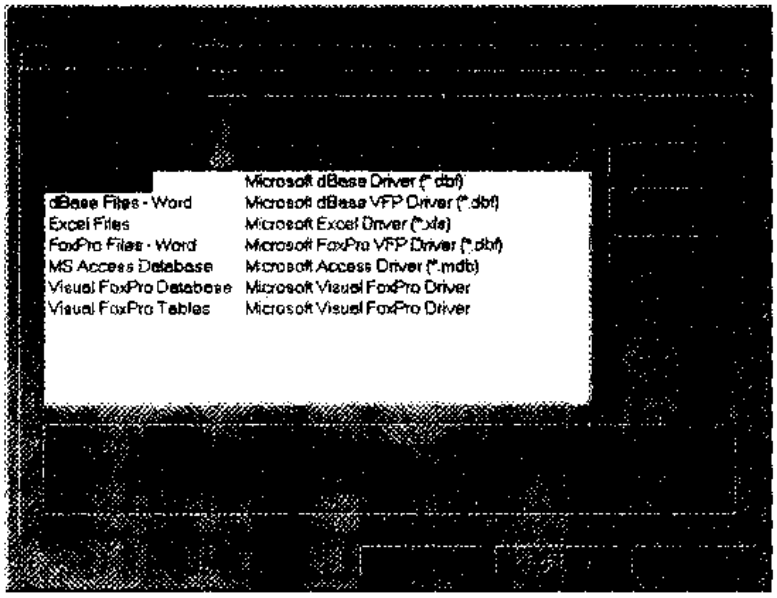


图 15.2 ODBC 管理程序主窗口

2)若 Patient 不在此列，单击 Add 按钮，屏幕上将出现 Create New Data Source 对话框(见图 15.3)。

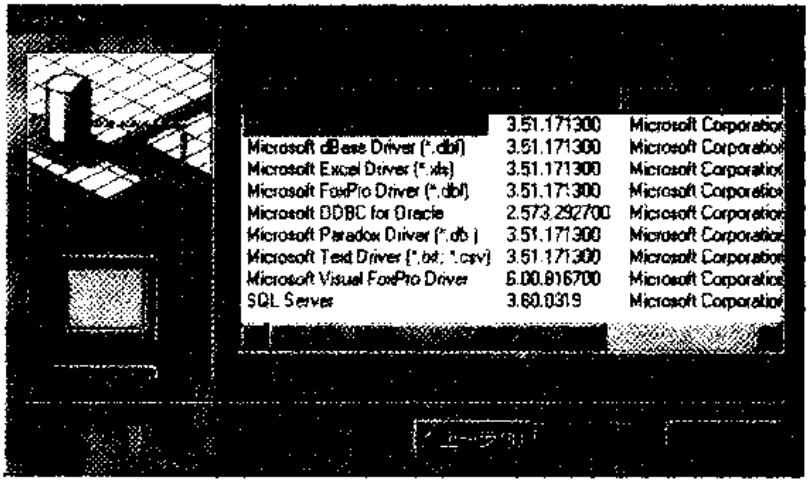


图 15.3 Create New Data Source 对话框

3)选择 Microsoft Access Driver(*.mdb)，该驱动程序的配置对话框将在屏幕上显示出来(如图 15.4 所示)。

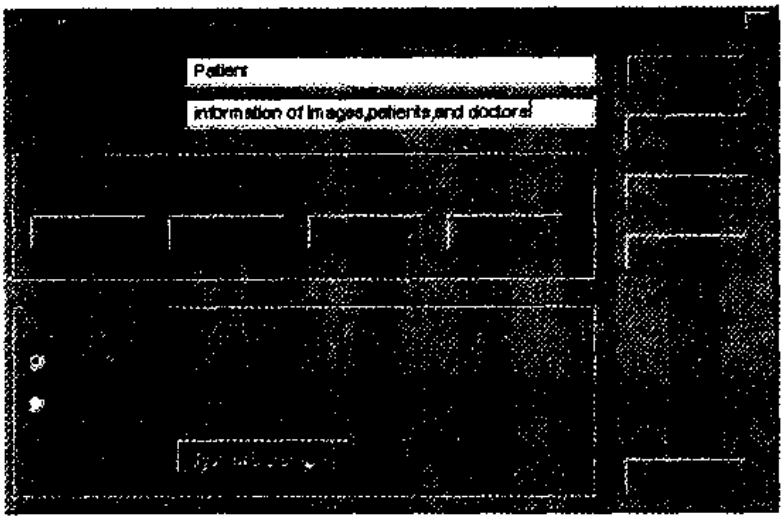


图 15.4　Microsoft Access Driver 配置对话框

4)在 Data Source Name 框中输入"Patient";Description 框内容可以忽略,也可输入所需的信息,我们输入"information of images, patients, and doctors"。

5)单击 select 按钮,在打开的对话框中选择数据库所在的目录和数据文件名。

6)单击 OK 按钮,即可完成 patient 数据源的安装。

15.4　数据访问对象(DAO)

DAO(Data Access Object)是一套简化数据库编程的 OLE 对象。一个代表着数据库的 Database 对象包含一系列 Tabledef 对象,其中每个依次都包含一系列 Field 对象。每个对象都有其属性及实现适当功能的方法。图 15.5 表明了这个对象体系。

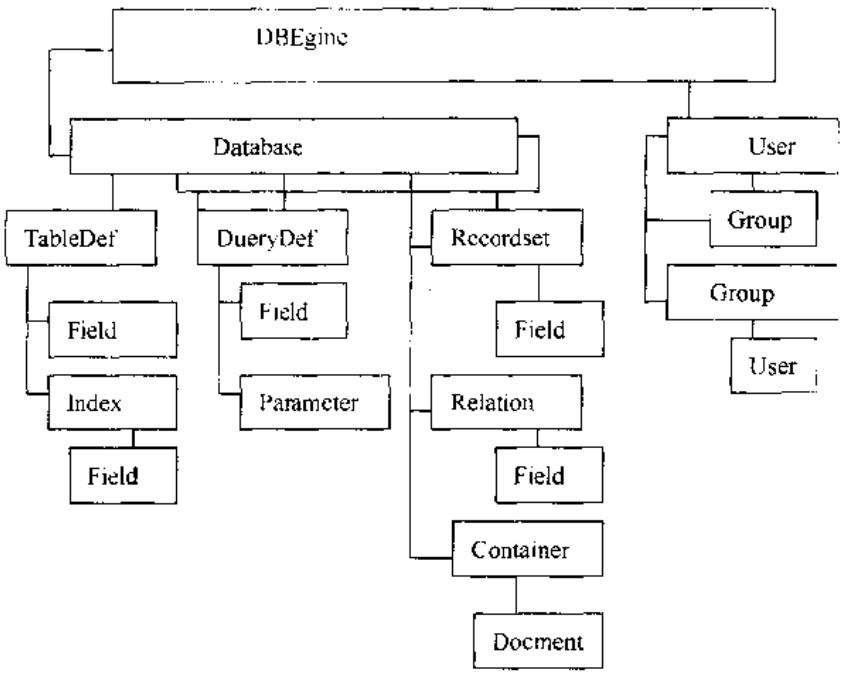


图 15.5　DAO 对象体系

DAO 使用一套称为 Jet 引擎的 DLL 图 15.6。所有这些 DLL 不仅对 MDB 文件提供访问，也面向其他的数据库格式，包括 Dbase 格式(如 dBase 和 Foxpro)、ParadorDB 格式，Microsoft Excel 及 Lotus 1_2_3 的电子表格数据，还有普通的文本文件格式，例如固定宽度并以逗号分隔的文本。Jet 甚至提供通向 ODBC 源的路径。

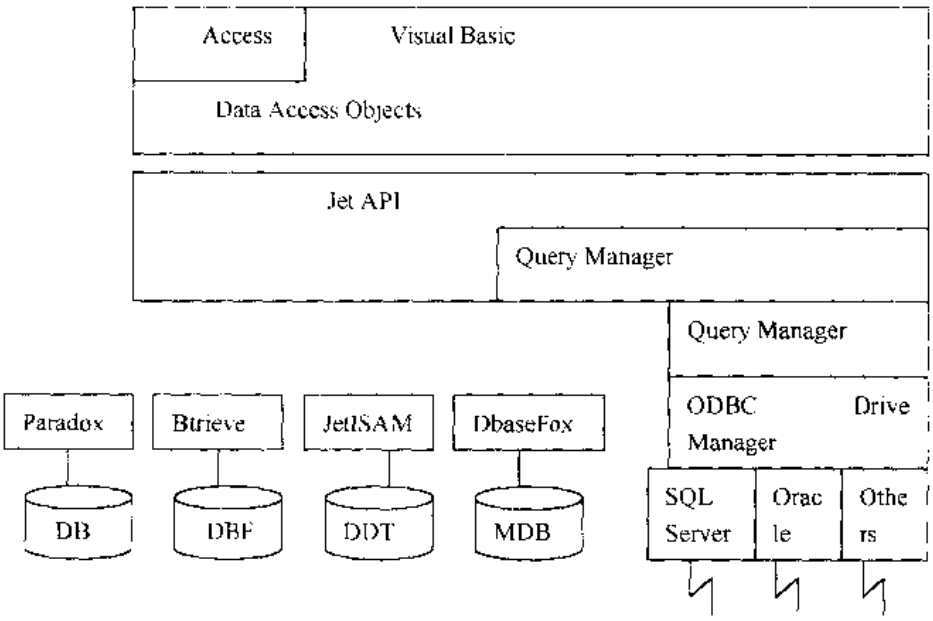


图 15.6 DAO 与 Jet 浏览

Jet 内含一个强有力的查询处理器，这使得复杂的操作变得简单。例如，可以查询连接到 SQL Server 表的 MDB 表，并更新查询结果视图。创建能依次引用其他查询的已有贮查询(等同于 SQL 视图)，这些被引用的查询仍然是可更新的，能力非常强大。它减轻了开发应用程序的负担。只需修改查询即可更改应用程序。

注意：这样的视图通常代表继续关系，而在这里代表包含关系。

1. DAO 与 OLE 自动操作

DAO(Data Access Objects)是以 OLE 自动操作双接口实现的。DAO DLL 和 DAO 3032，DLL 都执行一个 32 位内部过程 OLD 自动的服务器，这个服务器按序调用在 Jet 引擎和 MSJT 3032.DLL 中找到 DLL 入口点。这些与 Visual C++一同提供的 DLL 同样被用来在 Microsoft Access 95，Visual Basic 及 Microsoft Excel 95 中提供编程访问数据的能力。DAO DLL 包括一个类型库，此库描述了内含的所有接口，提供了 ANSI 和 Uicode 版本的接口，而不再需要类似 MFCAN 32.DLL 的外部支持。DAO OLE 自动操作服务器只是复杂的 OLE 自动操作组件的首例之一。事实上最早在 Windows NT 3.51 中以及现在在 Windows 95/98 出现的大多数 OLE 自动操作功能得以开放的部分原因便是与之并行发展的 DAO 的推动。

该类型库中展现了 DAO 接口在头文件 DBDAOINT.H 中定义。图 15.7 是部分最高级别 OBEgin 对象的定义。DBEngine 有 5 个属性：UeBion IniPath、DefaultUser、DefaultPass word 和 LoginTimeout.IniPath 和 LoginTimeout 既可检索也可设置。还有 4 种方法：CompactDatabase，

OpenDatabase, CreateWorkspace 和 CreateDatabase。图 15.7 的声明足够明显，几乎可以猜出其中大多数功能。但要使用它们仍需要许多幕后管道。例如，因为 DAODBEngine 是 OLE 自动操作双接口，它像任何其它的 OLE 接口一样必须对其进行"AddRef"和"Release"。而且 DAO 使用的是 OLEBSTRS 而不是原先的 C/C++ 字符指针或 MFC (String CBSTR 是一种字符串，它的长度存在第一个 DWORD 中并且以 NULL 结尾)，而且许多参数尤其是可选参数，都是以 VARIANTS 传递的。幸运的是，MFC 4.0 提供了一些新的类来简化其使用方法。

```
DECLARE_INTERFACE_(_DAOenginge,_DAO)
.
  STDMETHOD(get_Veisim)(THIS_BSTR FAR* pbstr) PURE;
  STDMETHOD(get_Inipath)(THIS_BSTR FAR* pbstr) PURE;
  STDMETHOD(put_Inipath)(THIS_path) PURE;
  STDMETHOD(put_DefauttUser)(THIS_BSTR vser) PURE;
  STDMETHOD(get_DefauctPassuord)(THIS_BSTR pw) PURE;
  STDMETHOD(get_LoginTimeont)(THIS_Short FAR * ps) PURE;
  STDMETHOD(get_Error)(THIS_PACErrors FAR * ppers) PURE;
  STDMETHOD(CompautDatabase)(THIS_BSTR SrName,BSTR DstName,VARIANT DstConnect,VARIANT Options,
VARIANT SrConnect) PURE;
  STDMETAOD(CrenteDatabase)(THIS_BSTR Name,BSTR Connect,VARIANT Option,DAODatabase FAR * FAR *
ppdb)PURE;
.
```

图 15.7 DBDAOINT.H

2.在C++ 中编制 DAO 程序

用C++ 对 DAO 进行编程有三种方法：

- 直接调用 OLE 自动操作双接口。
- 使用 MFC CDao 类。
- 使用 DAO SDK 类。

以上的每一种方法都有优点和缺点。因为直接通过自动操作编程相当冗长，多数人选择 MFC 或 DAO SDK 类。MFC 类(CDaoXxx)设计感觉上与 MFC 的 ODBC 类(CDatabase)相同，但能力有显著增强，比如它可以在运行中指定字段名。如果已非常熟悉 CData base，CRecordset 等等，MFC DAO 类将是学习 DAO 的捷径，因为它与 ODBC 类在命名与操作上都很相像。

15.5 DLE DB

OLEDB 是 Microsoft 新的数据库接口，它基于通用对象模型 Common Object Modle (COM)。这种接口提供很大的灵活性，允许你访问许多不同的数据源，包括任何一个可以在表中表达的数据。从 Excel 电子表格到特殊数据获取硬件。OLEDB 也提供了一些技巧，允许你产生 OLEDB 数据资源以满足特殊应用。

然而，OLE DB 的这种灵活性并不是任意的。接口用起来可能比较复杂，在使用 OLE DB

接口时你必须加入许多代码。对许多应用程序来说,使用 ActiveX Data Object 会更简单,它会为 OLE DB 数据资源提供更为简单的接口,并且不减弱灵活性。

15.6 ActiveX 数据对象(ADO)

ActiveX 数据对象(ADO)是 Microsoft 数据库接口的最新组成部分,定义了一组 COM 对象,以操作不同数据源的数据。Microsoft 提供了 OLE DB 接口实现的 ADO 对象,尽管你也许会选择使用其他方式实现的 ADO 对象,这些方式提供同样的接口来访问其他数据。你甚至想实现你自己的 ADO 提供程序。

ADO 趋于代替你非常熟悉的数据访问对象(DAO)和远程数据对象(RDO)。ADO 为不同类型的数据提供了一个非常通用的接口,不管这些接口在单机或是分布在网络中。

因为 ADO 中使用的对象是 COM 对象,它们很容易在众多编程环境中使用,包括 VB,VBA,VBScript 和 Java Script,甚至 V C++,虽然在C++ 中使用 ADO 对象比较困难,但新的 VC++ 6.0 使得在C++ 中用 ADO 和在 VB 中使用一样简单。

15.7 MFC 数据库类

MFC 提供了一些类用于使用 ODBC API 与 ODBC 数据源接口,也提供了处理数据访问对象(DAO)的类,来处理台式数据库。在本节中,我们将逐一看到 ODBC 类,它与 DAO 类非常相似,可以在联机文献中找到更多的关于 DAO 类的内容。

在本节中,将看到 MFC 提供给数据库访问的三种主要类,其有 CDatabase 类,用来与一个数据源相连;CRecordSet 类,用来处理从数据库返回的一组记录;CRecordView 类,简化从 CRecordSet 对象中得到数据的显示。

只要看到了组成 MFC 类库应用程序构成块的这些类,就会了解如何使用 AppWizard 来帮助创建使用 MFC ODBC 类的应用程序。

15.7.1 CDatabase 类

MFC CDatabase 类用来封装应用程序对一个数库连接的处理。这个连接可能是通过网络与一个数据库服务器的连接,也可能是在本地机上用来跟踪对一个台式数据库的设置。在许多情况下,与 CDatabase 有关的方法直接与 ODBC CAPI 的函数相对应,而该 ODBC API 处理连接句柄。

1.打开数据库连接

打开一个数据库连接,为了创建一个新的 CDatabase,且将它与一个数据库连接,可以建立一个新的 CDatabase 对象,且调用它的 OpenEx 成员:

Virtual BOOL OpenEx (LPCTSTR lpszConnect String,DWORD dwOptions = 0);

当设置 OpenEx()时,可以使用 lpszConnectString 以指定一个完整的连接字符串:

"DSN = MyData Source; UID = sa; PWD = Dont Tell"这个字符串定义了要连接的数据源的名字和用来连接它的用户的 ID 和 PWD。另外,这个字符串可以包含额外的由 ODBC 驱动程序提供的关键字。

为使事情更加容易，可以为 lpszConnectString 传送 NULL，这时 MFC 允许用户从对话框中选择数据源。虽然这个选项对你来说很容易，但应记住，从对话框选择数据源对用户来说并非总是给出简单选择接口的最佳选择。当 lpszConnectString 的值缺省时，调用 OpenEx()时数据库被打开，同时可对该数据库进行写访问，但没有装入光标库。通过修改 dwOptions 的值，可以修改数据库打开方式，dwOptions 的值可为下列标志符：

- CDatabase::OpenExclusive 以独占方式打开。
- CDatabase::OpenReadOnly 以只读方式打开开。
- CDatabase::useCursorLib 装载 ODBC 光标库，它可为原来不支持静态 snapsho 的驱动程序提供支持，但是如果光标库被装入后就不能使用 dynashot，即使驱动程序支持他们(后面将介绍关于光标和 dynasets 的知识)
- CDatabase::noOdbcDialog 禁止显示任何连接对话框。
- CDatabase::forceOdbcDialog 指明 ODBC 连接对话框总是对用户打开。如果 OpenEx()成功地与数据源相连，将返回 TRUE，只有用户放弃了对话框才会返回 FALSE。如果连接中发生错误，例如连接超时或口令失效，就会出现异常情况

除了 OpenEx()函数，CDatabase 提供了 Open()函数，如下所示，它用同样的方式打开了数据库，但使用不同的参数。

```
Virtual BOOL Open (LPCTSTR lpszDSN,BODL bExclusive = FALSE
BOOL bReadOnly = FALSE,LPCTSTR lpszConnect = "ODBC";,
BOOL bUseCursorLib = TRUE);
```

lpszDSN 字符串包括数据源的名字，而 lpszConnect 字符串包括 OpenEx()中 lpszConnect String 所具有的选项。注意调用 Open()时，lpszConnect 必须包含"ODBC"，用它可允许将来对其他数据库的操作。

bExClusive，bReadOnly，bUseCursorLib 参数允许你对 OpenEx()中传给 bw Option 参数的相同选项使用布尔值。

不管用 OpenEx()还是 Open()，结束时应调用 Cdatabase::Cloe()关闭对数据源的连接。

2. 连接信息

在任何时刻，可调用 Cdatabase 的 IsOpen()以查看它目前是否连接。另外，Cdatabase 的 Get Connect()返回用来连接数据库的连接字符串，GetDatabaseName()成员将返回用到的数据源名字。

CDatabase Class 还提供了几种查询当前连接能力的成员函数(这是由 ODBC 驱动程序支持的操作)包括 CanUpdate()，CanTransaet()，CetBookmark，Persistence()，GetCursor，Conmit，Beharior()和 GetCursor，Rollback，Behavior()。在后面将会看到这些函数的作用。

3. 用 CDatabase 执行 SQL 语句

在后面将会看到，对于返回结果集的一般查询来说，使用 CRecordset 类很容易。然而，可以不用 CRecordset 执行不返回结果集的 SQL 语句。这是通过调用 CDatabase::ExecuteSQL()实现，该函数原型如下：

```
Void ExecuteSQL(LPCSTR lpszSQL);
```

这个函数只简单地带了一个传给 lpszSQL 的 SQL 字符串。并对数据源执行该函数。注意 ExecuteSQL()并没有返回值。如果某些东西出错了,例如 SQL 语句失败,就会出现 CDBException 类型异常。应用程序应当注意这些异常情况,以判断语句是否正确的执行。

OnSetOptions():调用 CreateSQL()时,在 MFC 框架把 SQL 语句送给数据库前,会调用 CDatabase 的 OnSetOptions()成员。这允许 CDatabase 对象在执行 SQL 语句之前建立所要求的选项。

缺省执行时,这个函数为语句句柄(它将用于操作)简单地把 SQL_QUERY_TIMEOUT 选项设置为调用 CDatabase::Set QueryTimeOut()所规定的值。通过从 CDatabase 派生出你自己的类和重载 OnSetOptions()函数,可以加入你所需要的其他选项,OnSetOptions()是用来传送 MFC 执行语句的语句句柄。

提示:CDatabase Class 还提供了一个额外的隐含功能。它使你的类可以把参数捆绑利用 ExecuteSQL()执行的语句。为了使用此功能,从 CDatabase 派生出你自己的类而重载 BindParameters()函数:

VOID CDatabase::BindParameters(HSTMT hsmt)

为了有效地捆绑参数,如上一章看到的,使用 ODBC API。在做这类操作时,对于 MFC 数据库类考虑源码是很有帮助的,包括在 ODBCORE.CPP 中使用 Visual C++。

4. CDatabase 进行事务处理

CDatabase 类也可以处理数据库连接中的事务。事务允许把一些 SQL 语句作为一个单独操作执行,如果在事务中一个语句出错,那么其他语句也会停止执行。

当需要对一个数据库进行一些不同但相关的操作时,这种做法最有用。例如:如果你想进行销售管理,可能要修改发货单据和帐单。如果修改了其中一个而另一个修改失败,可以希望额外的顾客服务调用。

注释:并非所有 ODBC 驱动程序支持专务,可以调用 CDatabase::CanTransactex 查看当前驱动程序是否支持事务。

为了开始一个使用 MFC ODBC 类的事务,应调用 CDatabase::BeginTrans()。然后可调用 CDatabase::ExcuteSQL(),执行组成这个事务的操作,或者使用从这个 CDatabase 中派生出的 CRecordest 对象执行这些操作。

注意:对于一些 ODBC 驱动程序,当 CRecordset 对象已经打开时,调用 BeginTrans 会随着 RollBack 处理而导致错误。为了避免这种错误,最好先调用 Begintrans(),然后再打开用在事务中的记录集。一个事务可用两种不同的方法结束。如果所用操作成功且想结束事务,可调用 CDatabase::Committrans()。如果发生一个错误,而想放弃事务,应调用 CDatabase::RollBack(),它会中止由 CDatabase 或任何派生的记录集,执行事务中所有的操作时建议调用 BeginTrans()。

15.7.2 CRecordset 类

虽然 CRecordSet 类允许对一个数据库执行 SQL 语句,但却用 CRecordset 类提供了应用

程序与数据交互的实质。CRecordset 封装对数据库的查询包括加入、修改、删除行。

一般很少从 CDatabase 派生出自己的类，但如果没有自己的类，就几乎不可能使用 CRecordset 对象，因为 CRecordset 类为派生出的类提供了一个框架，派生出的类使用成员变量接收数据库的行返回的数据，下面将会看到。

1. 从 CRecordset 中派生类

CRecordset 类的目的是让应用程序访问从数据库中返回的结果集。MFC 允许从 CDatabase 派生类，该 CDatabase 使用成员变量来处理结果集中的数据。

首先，需要为列的每一项专用明一个 CRecordset 成员，MFC 使用记录字段交换(CRFX)自动更新这些变量，很快你将会发现这一点。下面的例子说明了由 ClassWizard 产生的一个记录集类的定义，该定义说明了用于操作列数据的成员函数的定义方法。

```
class CMyRecordset:Public CRecordset
{public:
CMy Recordset(CDatabase * pDatabase=NULL;
DECLARE _ DYNAMIC(/MyRecordset)
//Field/Param Data
//{{AFX _ FIELD (MyRecordset,CRecordset)
long m _ Num;CString m _ Title;
long m _ pages;long m _ mykey;
CString m _ name.
//}}AFX _ FIELD
//Overrides
//Classwizard generated virtual functionoverrides
//{{AFX  VIRTUAL(Clny Recordset)
Pubic:
Virtual CString GetDefault Connect();//Default Connection string
Virtual CString GetDefault SQL();//Default SQL for Recordset
Vitual void DoFieldExchange(CFieldExchange * pFx);//RFX Support
//}}AFX _ VIRTUAL
//:Implementation
#ifdef _ DEBUG
virtual void Assett Valid() Const;
Virtual void Dump (CDump Context & de)const;
#endif
:
```

下面，应向类中加入代码，把 m _ nField 成员设为类所使用的域的数目。还应该初始化成员变量的值为一个合适的缺省值，而这些成员变量是为列数据所声明的。下面的例子为前面定义的类展示出了由 ClassWizard 创建的结构：

```
CMy Recordset::CMyRecordset(CDatabase pdb)
CReardset:(pdb)
```

```
//  AFX_FIELD_INIT(CMy Recordset)
m_Num=0;m_Title=_T("");
m_pages=0;m_mykey=0;
m_name=m_TC("");m_nFields=5;
//} AFX_FIELD_INIT
m_nRefaultType=dynaset;
```

然后为类将建立 RFX。在取回任何数据前，都要打开 Crecordset。在下面章节中将会看到。

2. 记录字段交换(RFX)

通过使用 RFX，MFC 框架可以在数据库和 Crecordset 变量间来回移动数据，这与"对话框数据交换原理"很相似。

对于 CRecordset 类，交换是通过执行 DoFieldExchange()函数而建立的。象 CDialog::DoDataExchange()函数一样，大量的执行将使用一组由 MFC 提供用来定义 RFX 的宏。MFC 为不同的数据类型提供了不同的 RFX 宏，下表列出了可用的 RFX 宏和它们用到的成员变量的数据类型。

表 15.1 RFX 宏

RFX—宏	成员变量类型
RFX_Text	CString
RFX_Single	float
RFX_LongBinary()	CLongBinary
RFX_Long()	LONG
RFX_Int()	int
RFX_Double()	double
RFX_Binary()	CByteArray
RFX_Bool()	BOOL
RFX_Byte()	int
RFX_Date()	CTime

表中列出的每个宏有三个参数：使向 DoExChange()的 CFieldExChange 指针；数据库字段的名称；字段中存放数据的成员变量。下面的例子说明它们的使用。

```
ODid CMRecordset::DOFieldExchange
(CFieldExchange * PFX)
//{AFX_FIECD_MAP(CMYRecordset)
PFX  SetFieldType(CFFeldExchange::OutputColumn);
RFX_Long(pFx,_T("[nUM]"),m  Num);
RFX_Text(pFx,_T("Title"),m_Title);
RFX_Long(pFx,_T("page"),m  pages);
RFX_Long(pFx,_T("mykey "),m_mykey);
```

```
RFX_Text(pFx,_T("[name]").m_name);
//!AFX_FIELD_MAP
}
```

一旦设立了 DoExchange()函数,记录集类的成员变量将被自动修改,当获取一个新记录时,为当前记录保存这个值。当加入新的记录或修改现有的记录时,成员变量中的数据将自动地被使用,后面将会看到。

GetFieldValue()

除了使用 RFX,也可调用 CRecordset::GetFieldValue(),它允许取回当前行中任何字段的值,即使没有为列定义成员变量或者没有设立 RFX。在下列的索引或列的名字基础上,这个函数取回这个列的值。可把取回的值作为 Cstring 或 CBD_Variant 对象,它允许用单一的数据类型处理许多不同类型的数据

3. 打开记录集

在开始处理一个记录集之前,需要构建一个新的 CRecordse 对象,然后调用该对象的 Open()成员打开记录集。

CRecordset 类的构造函数以指向一个 CDatase 对象的指针参数,该对象用来处理数据库。如果已经产生一个 CDatabase,可以简单地传送一个指向对象的指针。

另一方面,可把 NULL 传给 CRecordset 结构,这时,MFC 使用被 CRecordset::GetDefaultConnect()返回的连接字符串构建一个新的 CDatabase 对象并且打开它。如果没有执行 GetDefaultConnect(),则简单地返回 ODBC。如果用 ClassWizard 创建你的 CRecordset 类,在选用的数据源基础上,它就会重载 Get DefaultConnect()。如果没有用 Classwizard,那么在 CRecordset 结构中,要传递一个指向 CDatabase 类的有效指针,或为 GetDefaultCorect()提供你自己的重载。

接着调用 CRecordset::Open()打开记录集。

```
Virtual BOOL Open(UNIT nOpenType = AFX_DB_USE_DEFAULT_TYPE,
LPCTSTR lpszSlee = NULL,DWORD dwOptions = none);
```

(1)记录集类型

Open()的第一个参数指明需要打开的记录集类型。这个值取回记录集的 ODBC 光标类型,并且决定了应用程序在记录集和数据库变化类型间访问的能力,nOpenType 的参数有下列值:

• CRecordset::dynaset:这种类型使用一个键集驱动光标,光标不能前后移动。每当检出一个新记录时,当记录集打开时,应用程序应能发现初始返回的行的任何变化,但不知道其他应用程序可能加入的其他行。dynaset 不会反映记录集中行的顺序变化。

注释:ODBC 光标库并不支持 dynaset 类型,如果使用这种类型,而且驱动程序支持键集驱动光标,应确认产生 CDatabase 时没有装入光标库。

• CRecordset::Snapshot:这种类型允许光标向前后滚动,但不能查到其他应用程序产生的任何变化(除非关闭记录集再重新打开,或者调用 Requery())

• CRecordset::forwardOnly:这种类型的记录集使用只能向前滚动的光标,不能反映数据库的任何变化。

• CRecordset::dynamic:这种类型使用动态光标,允许在两个方向滚动。每次从记录集取回一个新行时,能看到数据值的变化,也能看到新的记录或顺序的改变。

注释:许多 ODBC 应用程序不支持上面几种记录集类型所要求的光标。如果要求一个不支持的类型,Open()会出现异常情况。

(2)定义记录集的内容

CRecordset::Open()中 lpszSQL 参数定义了为记录集而取回数据的类型。lpszSQL 应被传进一个以 NULL 结尾的字符串,这个字符串包括一个 SQL SELECT 语句,或包含从一个过程中返回结果集的 call 语句。传进 lpszSQL 的字符串也可能只是表的名字 MFC 依此选择表中的所有行。

另外,也可把 NULL 传给 lpszSQL,这时 MFC 要调用 CRecordset::Get DefaultSQL()为记录集返回一个包含缺省 SQL 字符串的字符串。如果用 ClassWizard 产生 CRecordset 类,那么执行 GetDefaultSQL()。在传给 lpszSQL 的 SQL 字符串中,或从 Get Default SQL()返回的字符串中,可能包含 WHERE GROUP BY,ORDER BY 从句,可以把各种 WHERE 从句放在 CRecordset 类的 m_STRfILTER 成员中,以使记录集更加灵活。同样,也可把 GROUP BY,ORDER BY 从句放进 m_strSort 中。这样做,可以更容易地整理你的记录集,没有必要改变基本 SQL 字符串,或修改 GetDefault SQL()的返回值。

(3)记录集选项

第三个也是最后一个 CRecordset::Open 的参数允许指定几个不同的记录集选项。dwOptions 是一个可能包含许多不同常量联合的位置。想知道更多可能的选项,可查阅 CRecordset::Open()的联机帮助文献。

在记录集取回数据前,如果想指定选项,可以重载 CDatabase 的 OnsetOption()。这个函数是作为 CRecordset::Open()缺省执行的一部分而调用的。它可以在取回数据的语句句柄中设置不同的选项。

下面的例子说明如何产生一个记录集且调用 Open()取回数据,包括一些简单的异常处理。

```
BOOL bRC;
//Allocate new recordset
CMy Recordset * pRecordset = new CMyRecordset(pDb);
//Open Recordsef,catching exeptions
try
 //Open recordset with dynamic arsor
bRc = pRecordsef-> Open(CRcordset::dynamic,"SELECT"Num, Title,pages FROM MyChapters.
CRecordset::none);
if(bRc)
TRACE ("Recordset Opened OK/n")
else TRACE("Rcecordset Not Opened/n")
:
Catch(CMemoryExeption * pEr)
 pEx-> Report Error();
·Catch(CDBExceptionn * pEx)
```

```
pEX->ReportError();
TRACE("RETCODE:%D STReRROR[%s]strState:[%s]/n",
pEx->m_nRetCode,pEx->m_strError;
pEx->m_strStateNativeorigin);
```

(4)其他的信息

调用 Open()后,通过调用 Crecordset::GetSQL()可以取回用来选择记录的全部 SQL 字符串。如果不能确定要取回的表的名字可以调用 CRecordset::GetTableName()获得。另外,通过调用 CRecordset::GetODBCFieldCount()可获得返回的域的数目,调用 CRecordset::GetODBCFieldInfo()可取回每个域的信息。

15.7.3 刷新记录

如果想刷新记录集,可调用 CRecordset::Requery()。它可以再次从数据库取回数据,包括打开记录集后产生的变化。如果使用静态光标记录集,则这是唯一一种了解其他用户对数据进行任何变换的方法。象其他函数一样,并非所有的 ODBC 驱动程序都支持Requery(),可以调用 CRecordset::CanRestart()查看当前驱动程序是否支持。

15.7.4 关闭记录集

当完成了对一个记录集的操作以后应调用 CRecordset::Close()。调用 Close()后,如果想再次打开记录集,可以调用 Open()。如果用某些 ODBC 驱动程序进行事务处理,这是很必要的。

15.7.5 在记录集中移动

在 CRecordset 中所定义的成员变量只是用来一次存放一个数据。为了访问记录集中所有记录则需要用到移动方法,我们从 CRecordset::MoveNext()方法开始讲述。

1.向前移动

MoveNext()没有参数,只是简单的移动到记录集中的下一行,在这个过程中修改记录集中的变量。需要注意的是当滚动到最后一个记录时,调用 Move()将会出现异常情况。为了防止这一现象,在调用 Move()前,应调用 IsEOF()查看是否在记录集的末尾。

注释:当记录集打开时会自动选择第一个行,因此没有调用 MoveNext()时,记录集的成员变量反映与第一个行。

下面的例子说明使用 MoveNext()以跟踪在记录集中的行。但要注意的是为了简化例子已消除了异常处理,但应用程序应准备捕捉可能出现的异常。

```
do{
TRACE ("Num:%d Title:[%s pages:%d/n",pRecordset->
m_Num,pReaocrdset->m_Title,pRecordset->m_pages);
pRecordset->MoveNext();
while(! pRecordset->IsEOF());
```

2.滚动

如果记录集使用了仅能向前移动的光标,则必须调用 MoveNext(),然而,如果驱动程序支持滚动光标,就有几种选择。可以通过 CRecorset::CanScroll()查看记录集是否支持滚动

MoveFirst()和 MoveLast()方法会移向记录集中的第一个或最后一个记录。通过调用 SetAbsolutePosition(),可以移到任意指定的行,它把一个以零为基础的指示方法带进记录集中。

另外,MovePrev()成员将移向前一个行,当处于记录集开始时,调用 MovePrev()会出现异常。因此,调用 MovePrev()之前先调用 IsEOF(),查看是否处于记录集的开始。

3.使用书签

如果 ODBC 驱动程序支持书签(可以调用 CanBookmark()查看),可以用书签导航到指定的行。调用 GetBookmark()成员可为记录集中的当前行取回一个书签。然后可以把这个书签值传给 SetBookmark(),就可返回那个指定的行(也就是调用 GetBookmark()时的当前行)。

4.CRecordset::Move()

以前的每个方向函数都可被通用的 Move()函数支持,它是从当前行向前或向后移动若干行。典型例子如:

Virtual void Move (long nRows, WORD wFetchType = SQL_FETCH_RELAIVE);

nRows 参数表示移动的行数,正值向前移动,负值向后移动。

可以忽略 WFetchType 参数而简单地使用缺省值 SQL_FETCH_RELATIVE,或者用下列值中的任一个:

- SQL_FETCH_RELATIVE:从当前行中开始取出 nRows 中行。
- SQL_FETCH_NEXT:取出下一行。
- SQL_FETCH_PRIOR:取出前一行。
- SQL_FETCH_FIRST:取出记录集中的第一行。
- SQL_FETCH_LAST:取出记录集中最后一行。
- SQL_FETCH_ABSOLUTE:取出记录集中的第 nRows 行。
- SQL_FETCH_BOOKMARK:用一个传给 nRows 的书签值指定取出的行。

如果 Move()操作中发生错误,就会出现异常情况,否则,当前行被设置为新行,记录集中的成员变量被修改以存放新行的数据。

15.7.6 改变记录集中的数据

很多情况下,如果应用程序需要做一些实际工作,它将最终修改数据库中的数据。这可能包括向表加入新的记录,删除记录,或者修改现有记录的数据。因为删除一个记录最简单,我们从它开始介绍。

1.删除一个行

为了从数据源中删除一个行,应打开记录集调用 Move()或 MoveNext(),直到想删除的记录,然后调用 CRecordset::Delete()。这会把当前行的数据库打开,且把记录的成员变量设为 NULL。可调用 Move 或 MoveNext()移动到其他有效行。

可以调用 CRecordset 中的 Is Deleted(),查看当前记录是否已被删除。

2. 加入一个新行

加入一个新记录之前,应确定没有把记录集设为只读模式,这可调用 CRecordset::CanAppend()简单查看是否可以加入使用这个记录集的记录。

加入一个新记录有三个步骤:首先,应调用 CRecordset::AddNew()创建一个新的空记录;然后,把记录集的成员变量设为想加到新记录中的值。最后,用 CRecordset::Updata()向数据库加入新记录。如下列例子所示:

```
if(pRecordset->Can Append())
{pRecordset->AddNew();
pRecordset->n_Num=222;
pRecordset->m_Title="New MFC Record";
pRecordset->m_page=321;
if(pRecordset->Updata())
TRACE("Row Added OK/n")I
else TRACE("Row Not Added/n");
//end if can Append();
```

其中如果使用 dynaset,新记录将作为记录集的最后一行出现。如果作用 Snapshot,调用 Requery()修改包含新记录的记录集。

3. 编辑现有的记录

与加入一个新记录一样,编辑也有三个步骤:首先,调用 CRecordset::Edit()开始对当前记录的修改过程;其次,可以改变记录集中成员变量的值;最后,可调用 CRecordset::Opdata(),这些变化就会被传给数据库。如果调用 Edit()后,不选择修改行,可调用 CanCeldata()放弃该修改。

注释:修改一个记录之前,应调用 CRecordset::CanUpdata(),查看记录集是否允许编辑现有的行。

下面的例子显示出如何用 Edit()和 Updata()此修改数据库中现有的当前行。

```
if (pRecordset->CanUpdata())
{pRecordset->Edit();
pRecordset->m_Num=222;
pRecordset->m_Title="New MFC Record";
pRecordset->m_pages=21;
if(pRecordset->Updata())
TRACE ("ROW Added OK/n");
else TRACE("ROW Not Added/n";
//end if canAppend()
```

(1)处理 NULL 值:如果想把记录中的域设为 NULL 值,首先应查看 NULL 值是否可以通过调用 IsFieldNullable()而为那个字段所接收。然后调用 SetFieldNull()把这个字段设为

NULL 值。也可调用 IsFieldNull()查看这个字段是否被设为 NULL 值。

(2)行锁定:修改行时,可以选择框架如何锁定正在修改的行。最有利的锁定是仅在调用 Update()过程期间锁定,最不好的锁定是调用 Edit()时锁定行。直到调用 Update()后才释放这个行。在已打开记录集后调用 Edit()之前,通过调用 CRecordset::SetLocking. Mode()可设定锁定模式。

4.事务处理

应用程序可能想在操作中对数据库进行多次改变,可能包括很多记录集,为了做到这点,可调用 CDatabase 的 BeginTrans()成员函数,该函数将用于一个事务,然后使用从连接中派生的记录集对数据库进行改变。如果一切正常,通过通用 CDatabase 的 CommitTrans()成员可以对数据库作改变。如果出错或想回到最后一次调用 BeginTrans()时的状况,可调用 CDatabase::RollBack()。

注释:并非所有记录集都允许事务操作,可以通过调用 CRecordset::Transact()查看是否支持事务。

5.使用语句参数

在许多情况下,可能用几个变量值执行许多非常相似的查询。例如:假如想派生出一个 CRecordset 类,该类可获得在某工资范围内雇员的一系列记录,除了为某个范围派生出一个独立的类这种方法外,还可以构建一个使用语句参数的类,语句参数在执行语句是修改查询。

为了使用参数,可把参数位置标志符加入组成记录集查询的 SQL 字符串中。最常用的是,把 m_strFilter 参数化。虽然也可以在 m_strSort 字符串或在传给 Open()的 SQL 字符串中使用参数。对于本章所例举的工资范围的例子中,可以这样使用 m_strFilter 的值:

m_strFilter = "WHERE Salary > = ? AND Salary < = ?";

这个语句使用了两个参数,无论何时记录集调用 OPen()和 Requery(),MFC 用实际的值代替 SQL 中的"?"。为了使用这些参数,对每个参数来说,把成员变量加进你的记录集类中,且把 CRecordsetinm_nParams 成员设为要正使用的参数的数。

下面,修改记录集的 DoFieldExChange()成员,在参数成员变量和送到数据库的 SQL 语句之间移动数据。加入一个调用 CFieldExchange::SetFieldType()语句,以把字段类型设为 CField Exchange::param,且为每个参数设为 RFX—宏。参数的 RFX—宏必须按 SQL 语句中的位置标志符的顺序出现,RFX—宏中参数的名字并非用来与参数匹配,因此可以选择你喜欢的任何随机的名字。下面例子帮助说明向 DoFieldExchange()中添加参数。

```
Ooid CMyRecordset::DoFieldExchange(CField Exchange pFx)
//{{AFX_FIELD_MAP(CMyRecordset)
PFX->SetField Type(CFieldExchange)
::Outputcolumn);
RFx_Long(pFx,_T("[Num]"),m_Num);
RFX_TEXT(pFx,_T("[TITLE]");m_Title);
    RFX_Long(pFx,_T("[Pages]"),m_pages);
RFX_Long(pFx,_T("[mykey]"),m_mykey);
```

```
RFX_Text(pFx,_T("[name]"),m_name);
//}}AFX_FIELD_MAP
pFx->SetFieldType(CField ExChange::param);pFx->Text(pFx,"myFirst Param",m_strParamOne);
pFx->Long(pFx,_"my Second Param",m_nParam Two);}
```

6.记录集错误处理

在大多数情况下,如果与记录集有关的 ODBC 处理发生错误,MFC 会出现 CDBException 类型的异常情况。应用程序应跟踪它们以对错误做出反应。

另外,你可能想重载 CRecordset::Check(),它是由框架调用以查看从 ODBC 操作的返回码。

15.7.7 CRecordView 类

你将看到的最后一个 MFC 数据库类是 CRecordView。这个类基本类似于一个窗体视,但具有几个增强功能,使得从记录集中显示数据更为容易。CRecordView 允许使用对话框数据交换方式直接从记录集显现数据。它也为在记录集中向第一个,最后一个、下一个、前一个记录移动以提供缺省操作。

1.创建一个 CRecordView

像其他对话框类型一样,CRecordView 使用一个对话模板以定义视窗的布局。这样,在为视创建一个有效的类工具之前,要用资源编译器为记录视创建一个对话框模板。

接着,为视从 CRecordView 中派生出一个新的类。虽然可以用手工操作,但用 ClassWizard 更容易。如果加入一个新对话框模板后启用 ClassWizard,就会提醒为新资源创建一个类。然后可以选择 Create 产生一个新的类来着手定义新视类。

在 New Class 对话框中,命名类且设置 Base 类为 CRecordView。也可以在这个对话框中选择为 OLE 自动化处理添加支持,当在该对话框中敲击 OK 时,会收到一个对话框提示作用于这个视的记录集类。可以从下拉菜单中选择一个现有的记录集类,或者可以敲击 New,让 ClassWizard 引导你构建一个新的用于这个视的记录集类。

2.用 CRecordView 进行对话框数据交换

CRecordView 类支持略加修改的对话框数据交换机制的版本,不同的是不用在这个视类的成员变量与控件之间移动数据,而可在视控件与视类有关的 CRecordset 的列数据成员变量之间移动。

这是在执行 DoDataExchange()中用特殊 DDX_Field 函数完成的。也可以用 ClassWizard 很简单地创建。如果选择一个控件且敲击 Add Variable,会看到 Add Member variable 对话框。在 Member 变量名域中选择下拉制表,可以为相关的 CRecordset 增加一个变量。这将对 DoDataExChange()函数自动加入合适的码。每当视被修改时,都会从记录集中显现数据。下例从一简单的 CRecord View 来说明了 DoData Exchange()函数。

```
Void CMfcExView::DoDataExChange(pDx);
//{{AFX_DATA_MAP(CMfcExView)DDX_FieldText(pDx,IDC_EDIT1,m_pset->m_ID,m
```

```
pSet);
DDX_FieldText(pDx,IDC_EDIT2,m_pset->m_Title,m_pSet);
DDX_FieldText(pDx,IDC_EDIT3,m_pset->m_Number,m_pSet);
//}}AFX_DATA_MAP
```

3. OnGetRecordest()

CRecordView的OnGetRecord()成员是由主框架调用，取回一个指向与视有关的CRecordset指针。主框架在最终修改视时将调用OnGetRecordset()，连接修改时也调用。由Class Wizard提供的缺省执行返回一个存在CRecord View::m_pSet的指针，并且如果必要的话将创建一个与记录集类相关的新对象。如果想对创建记录集加入任何特殊处理，可以自己执行这个函数。

4. CRecordView::OnMove()

CRecordView的OnMove()函数提供一种简便的方法，以把视移向记录集中的一个新行，它调用相关的CRecordset的Move()，且修改显示以显示出这个新的行。如果在记录视中当前行被修改，OnMove()会自动修改数据库。OnMove()仅有一个参数，指明移向何处，这些数可能是下列常数之一：

- ID_RECORD_FIRST
- ID_RECORD_LAST
- ID_RECORD_NEXT
- ID_RECORD_PREV

当在记录集中移动记录，会发现CRecordView的IsOnFirstRecord()和IsOnLastRecord()很有用，如果视当前定位在记录集的第一行或最后一个行，它们将返回TRUE。

15.7.8 用AppWizard创建数据库应用程序

现在已看到了如何进入MFC的数据库框架，让我们看一下AppWizard如何帮我们把各部分集成到一起，创建一个应用程序。

当用MFC的应用程序AppWizard创建一个应用程序时，其第二步中允许为应用程序选择几个不同层次的数据库支持。

注释：AppWizard的数据库选项不能用于对话框的应用程序。

Header文件的可选项仅包括MFC数据库类的头文件和增加构建的设置，以与MFC数据库相连，但它不定义任何类。

Database视的两个选项都产生一个记录集和记录视类。必须敲击Data Source按钮，选择一个用于这些类的数据资源，敲击后出现允许选择数据资源和设置所用记录集类型的对话框。你也会被提醒为记录集和记录视类使用特殊的表、视和数据资源中存放的过程。

带有文件支持选项的Database视为文件连续化和与文件有关的File菜单选项提供支持，而没有文件支持选项的Database视不支持连续化，且没有把Open、Close、Save、Save As命令加到File菜单中。

15.8 小　　结

数据库技术作为当今社会计算机技术软件开发中一个不可缺少的内容，越来越引起人们的重视，本章就此方面进行了如下内容的讨论和介绍：

- 数据库基本概念和管理
- SQL 语言的基本内容
- ODBC 简介和应用
- 表记录简介和处理
- 数据库中对记录的处理、增、删
- 数据访问对象(DAO)的简介和应用

第 16 章　ActiveX 控件

ActiveX 控件这一十分时髦的名词，一定已被大家所熟知，本章我们将详述以下几方面内容：

- OLE 简介
- ActiveX 文档
- COM
- ActiveX 技术
- ActiveX 控件的方法
- 通过 ActiveX ControlWizard 创建控件
- ActiveX 控件中的图片
- 增加正文敏感帮助
- ActiveX 控件的属性页

下面就每一部分内容给大家详细介绍。

16.1　OLE 简 介

1991 年，微软发布一种叫 OLE 1.0 的规范。它代表了对象的链接和嵌入技术。OLE 1.0 是处理复合文档的一种方法。复合文档就是在一个文档中同时保存有多种格式的数据，如文本、图象、声音。那时，面向对象的编程方法是一种新的方法，而 OLE 1.0 规范则向面向对象技术迈进了一步。进一步来说，OLE 1.0 是以文档为中心而不是以应用程序为中心发展的结果。不幸的是 OLE 1.0 受到软件开发商的冷遇。只有很少的独立软件开发商(ISA)使用 OLE 这一技术。OLE 1.0 非常复杂并很难理解。它必须用复杂的 API C 来编程，其中包含 OLE 编程概念的雏形。这些新的概念对大多数程序开发者都是不适用的。

幸运的是微软公司继续发展了 OLE 技术。1993 年，微软发布了 OLE 2.0 规范，该规范含有的内容比正在使用的复合文档多，它是基于对象服务的一整套的体系结构，而且能够被扩展、定制和增强。这项服务体系结构的基础是组件对象模型(COM)。它包括以下服务：

- 组件对象模型 COM
- 剪贴板
- 拖放
- 嵌入
- 在位激活
- 链接
- Monikers(永久命名)
- OLE 自动化
- OLE 控件
- OLE 文档

- 结构化存储
- 统一数据传输(UDT)

OLE2.0 版的发布对标准的计算方法产生了影响,它得到计算机界的各大殊荣:PC Magazine 的优秀技术奖,PC/Computering 的软件革新 MVP 奖。另外 OLE2.0 还成功推出了它的新编程接口。开发者现在可以轻松地用 OLE 技术开发软件。OLE2.0 服务包含了许多概括在面向对象的编程方法,如:封装、多态和基于对象的结构。1993 年 2 月发布的 Visual C++1.0 和微软基本类(MFC)库 2.0 版使 OLE2.0 获得了更大的成功。MFC 一个C++类库封装了 OLE API,这样使开发者更容易使用 OLE 服务结构。

注释:OLE 编程非常复杂,不要因为能轻松地使用 MFC 库而对此不以为然。然而我还建议使用 MFC 编程,因为 MFC 能为我们提供编程框架从而使编程变得轻松些。一开始就在 API 级编程可能会导致失败。如果你不懂 OLE,我建议你用 V C++6.0 和 MFC 来学习服务、概念以及它的标准,然后回过头来理解低级的 C API。理解服务和它的接口以及什么时候使用它们是 OLE 和 ActiveX 编程的关键。

现在 OLE 不再是对象链接和嵌入的缩略语。这个名字已经绝对化了。微软所指的是简单的 OLE,读作"O-lay" 注意 OLE 不再有版本号了,因为它是可扩展的结构,它可以在不修改基础的情况下扩展和增强。OLE 控件就是证明。OLE 控件不是原始 OLE 发布版的一部分。它直到原始版本发布一年后才出现。实际上,用 OLE1.0 创建的对象现在仍在工作,并与现代的 OLE 应用程序一起使用。只是它们的功能被限制在 OLE 1.0 规范内,所以从现在开始,不再区分是 OLE 1.0 还是 OLE2.0,除非特别指出。

16.1.1 组件对象模型

当微软设计 OLE 时,他已考虑到用面向对象编程。COM 对象很像具体化了的C++类或 Ada 包。实际上 COM 设计是考虑到了C++编程人员。它支持封装、多态和可重用 然而 COM 被设计成兼容二进制,因而又不同于一个C++对象。作为一个编程人员,你可能清楚编译型的语言,如 C、C++、Pascal 和 Ada 都是依赖于机器的。作为一个二进制对象,COM 对象关心的是它与其他对象的接口。当不在原开发环境中使用时,一个 COM 接口是暴露的而且是可见的。说它可见是因为它是一个二进制对象,所以它是不依赖于机器的。这并不要求主环境或交互式对象知道关于 COM 对象的任何东西。当对象在其母应用程序中被创建时,COM 并不关心该对象是如何交互的。这种交互是在母应用程序和子对象间进行的。当对象与外部交互时,COM 关心的是如何与该对象接口。应该注意的是 COM 不是一种编程语言,它是一种能使各软件组件作为对象进行相互作用的二进制标准。COM 不特指一种编程语言,因而它能与任何一种支持 COM 对象二进制形式的语言一起工作,它是方便这种编程标准的编程模型。

COM 中包括两种成员:属性和方法。属性是数据成员,方法是函数成员。COM 对象有一个统一的接口,无论它完成什么,COM 对象必须实现一个 IUnknown 接口。这个接口是其他对象的主接口,也是其他 COM 派生类的基类。IUnknown 接口有下列成员函数:

- ULONG AddRef (void)
- ULONG Release (viod)
- HRESULT QueryInterface (REFIID id, void * * ipv)

每个对象实现一个 vtable，vtable 就是一个指向成员函数的指针数组(如图 16.1 所示)，所有对象的通用接口 Vtable 中保存了指向类成员函数的指针表对象实现(Common Access To Object)，vtable 被所有对象的实例共享，并保存每个对象的专有数据。客户应用程序引发一个接口实例，并得到一个指向 vtable 的指针。每次生成一个接口，对象中的引用计数就用 AddRef()加一。相反，每次销毁了一个引用，引用计数器就用 Release ()来减一。一旦引用计数等于零，这个对象就被销毁。为了了解对象支持的接口，可以使用 QueryInterface()。

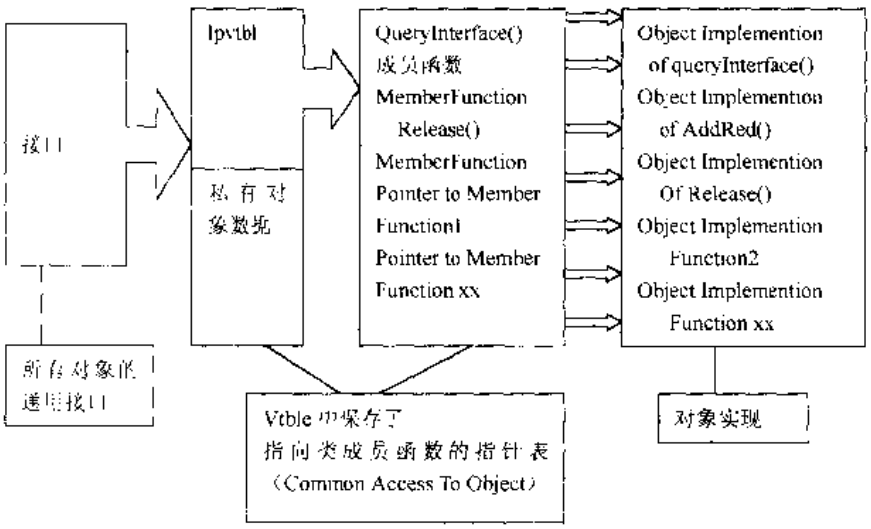


图 16.1

COM 接口不能被直接访问。它们总是通过一个指针被访问。QueryInterface (REFIID id viod * * ipv)函数带有一个引用 id 和一个 void 指针。(注意指针前面的双星号：* * ipv)这个 id 号是唯一的 128 位 ID，它表示你想要获取的接口。ipv 是存放要取回接口的指针的地方。

请看以下代码段：

```
IAnyInterface * pAny = NULL;
if (pUnknown->QueryInterface (IID-IAnyInterface, (void * * ) &pAny) = =NOERROR)
.
PAny->DoAnyObjectWork ();
pAny->Release ();
:
```

pUnknown 是指向对象 Iunknown 接口的指针。DoAnyObjectWork ()是要求工作类的成员函数。通过对象接口指针 pAny 访问它的成员函数。Visual C++ 和 MFC 通过使用接口映射封装了 Iunknown 的实现。其映射与用来映射窗口消息的消息映射很相似。这是个比较容易理解的实现。Visual C++ 和 MFC 向导和在 MFC Class Library 中实现的 OLE 类封装了这其中的大部分工作。

到目前为止，我们对 COM 有了大概的认识。要真正理解 COM 需要对它有更深的了解。实际上可以用整个一章的篇幅来讨论这个问题。如果有兴趣想进一步了解，请参考附录 A“附加资源”，它为更深入地研究这一主题提供了极好的参考。

16.1.2 结构化存储

不幸的是，今天大多数操作平台有不同的文件系统，这使共享数据很困难。另外，这些文件系统是在大型机年代发展起来的。那时在一个时间内只能有一个应用程序可以修改或访问文件的数据。构建 COM 是为了使不同的应用程序能相互操作与集成。为了做到这一点，COM 需要在底层的文件系统中有多个应用程序向同一文件写入数据。OLE 的结构化存储就是为满足这个要求而设计的。

结构存储是一个文件内的文件系统。可以把它看成由存储和流构成的层次(如图 16.2 所示)，在这个层次树中，每个结点只有一个父结点，但可有多个子结点。就像 Windows 98 的文件管理器。文件夹是存储结点，文件是流结点。结构存储提供了一种在一个文件内组织数据的方法。另外，这种组织数据的方法不仅限于文件，它同样适用于内存管理和数据库

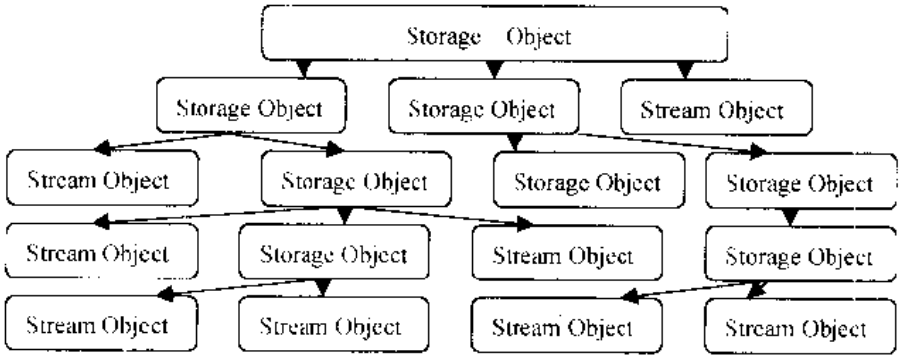


图 16.2 结构存储示意图

流对象包含数据。这数据可以是本地数据，也可以是来自其他外部对象的数据。存储对象是在二进制级兼容的，因而理论上，它们兼容于所有符合 OLE 规范的平台。然而各平台间略有差异。

结构化存储对象由下列 OLE 接口来操纵：

- IPersistStorage
- IStorage
- IStream

正如名字所示，Istroage 操纵存储对象，Istream 操纵流。你可能很少单独操纵一个存储或流对象，更常见的是，你会使用 IpersistStorage()来操纵永久存储对象。永久存储对象就是既使对象被撤消它里面的数据还能保存。例如，想允许用户定义对象如文本标签的颜色，就应该永久的存储对象的前景和背景颜色。下次创建对象时，就可以从永久存储中读出上一次用户选择的颜色。然后可以将这些属性应用于创建的对象，这样可以保持用户喜欢的颜色。通过完成以下操作，IpersistStorage 能做到这一点。

- InitNew
- Load
- Save
- SaveCompleted
- HandOffStorage

MFC 用一个容易使用的类 COleDocument 封装了 IPersistStorage 和 ISorage。另外，为支持存储和流的操作，MFC 提供了 COleStremFile。OLE 文档和它的扩展 ActiveX 文档将在 12 章的"ActiveX 文档"中介绍。应注意到 MFC 中的复合文档不仅指 OLE 文档和 ActiveX 文档，而且还指任何结构化存储数据的文件类型。

16.1.3 Monikers

Moniker 是在基于对象系统中(如 OLE)引用一段数据或引用一个对象的方法。当链接一个对象时，它的 moniker 也就建立起来。比如，在 Word 文档中链接了一个声音文件，这个 WAV 文件并不存储到 Word 文档中，而是自动建立一个 moniker 来指出 WAV 文件的位置。

为用 moniker 来定位并捆绑到一个对象，必须使用 IMoniker 接口并调用其成员 Imoniker: BindToObject ()。通过使用对象智能永久名称，IMoniker 接口将定位链接对象的位置并返回它的接口指针。这时，moniker 自身就消亡了。想一想这和在 C 或C++ 中定位一段数据时撤消指针的引用相似。IMoniker 是从 IPersistStream 中派生出来，因而它能将自己序列化成数据流。这里有 5 种基本类型的 moniker:

- File monikers
- Item monikers
- Anti-monikers
- Pointer monikers
- Composite monikers

1) File Monikers

File monikers 永久存储一个文件名。为了将文本文件名与文件对象相连，一个指向文件对象接口的指针将被返回，这样就可以操纵这个文件对象。

2) Item Monikers

Item monikers 指向文件中一个具体的位置，如一张图片或一段嵌入的视频图像。

3) Anti-Monikers

Anti-Monikers 在一组 moniker 或一个 moniker 链中，它删除最后一个链接的 moniker。

4) Pointer Monikers

它只简单地指向另一个 monikers，把所有的 moniker 组成一个链。应注意它不是永久的。

5) Composite Monikers

它是一个 moniker 的有序集合。在它的根部是一个引用文档路径名的 File moniker。之后是一系列的 Item Monikers。当在一个对象中需要一个 moniker 的集合时，就需要使用 Composite Moniker。

6) MFC 封装的 Moniker

在前面介绍的结构化存储中，我们知道了 MFC 有一个叫作 ColeStreamFile 的类。这个类的目的是封装 ISream 的功能使你能够访问复合文档中的数据流。COleStreamFile 的派生类是 CMonikerFile。这个类封装了 IMoniker 接口提供的 moniker 的功能。这个类能够让你访问由 MIoniker 命名的 IStream。应该注意该类并未封装整个 IMoniker 接口。该类提供了处理复合文档中流式数据的能力。所以如果想捆绑到一个存储或对象，必须直接实现 IMoniker 接口。也就是说，不能直接使用 MFC 来实现上面所说的所有的 moniker 类型。

16.1.4 统一数据传输(UDT)

在OLE里,可以用结构化存储来存储对象,并用Moniker来找出对象的位置。但是,这里还需要一种机制可以将数据从其存储(链接或嵌入)的地方移到客户可以处理的地方。统一数据传输(Uniform Data Transfer)就能做到这一点,并且它能向数据对象和客户通知数据的改变。UDT通过IDataObject接口提供该服务,并且用于以下三个应用方面:

- 剪贴板
- OLE拖放
- 链接与嵌入

1. 剪贴板

系统剪贴板是用于内部进程通信的系统级服务。因为它是系统级服务,所以所有的应用程序都可以访问它。OLE可用剪贴板完成进程间对象的统一数据传输。在IDataObject指针的帮助下,可以用OleSetClipboard()函数将一个对象剪切或拷贝到剪贴板中,并且通过剪贴板将对象显露给所有进程。同样,想从剪贴板粘贴回数据时,可以使用IDataObject指针和OleGetClipboard()来做到这一点。因为它保持了对象的完整性,所以它是一个非常强大的工具。使用它你可以在应用程序间移动复杂对象的数据类型。

通过使用CWnd和COleClientItem类的成员函数,Visual C++和MFC可以让你访问剪贴板。

2. 拖放

拖放是用户在应用程序里或应用程序间选择移动对象的方法。UDT被用来实现拖拉操作。一旦选择了对象,源应用程序就打包对象并用IDataObject指针调用DoDragDrop(),源应用程序使用IDropSource接口,它生成一个指向它的指针。这个指针被传递给DoDrapDrop()源应用程序控制鼠标光标并在鼠标释放时处理对象。

一旦用户把对象拖拉到新的客户区或目标,客户应用程序启动一个IDropTarget接口。在一个指向IDropTarget指针的帮助下,客户应用程序用IDropTarget接口中的函数跟踪与它联系的对象。当对象在目标区放下,IDropTarget::Drop()函数就会被调用。Drop()函数给目标传递一个源对象的IDataObject指针,现在客户应用程序拥有了IDataObject指针,它可以自由地操纵对象了。

- COleDropSource
- COleDropTarget
- COleClientItem
- COleServerItem
- COleDataSource

3. 嵌入与链接

一个被链接的对象其实并没有存储在OLE文档中去。在文档中,一个Moniker被存储用来引用链接的对象。OLE文档使用UDT可以把数据从数据对象源移动到容器应用程序,结果这些数据可以被正确地传递。链接对象是通过IOleLink接口来被操纵的。通过链入而是

嵌入一个对象，可以减少复合文档的大小。另外，可以暴露该链接对象，这样它可以被多个人使用。

嵌入对象是通过结构化方式存储的对象。虽然这会使复合文档尺寸变大，但它提供了一种在一个文件中存储多种数据类型的方法。

16.1.5 OLE 文档

OLE 文档就是用结构存储方式保存对象的复合文档。这些对象可以是本地数据或者是通过使用 moniker 而链接于文档外的数据。此外，一个 OLE 文档可以包含其他进程建立的对象，对象被嵌入文档好象它就是文档的一部分。就象其他 OLE 对象，OLE 文档通过接口来控制。OLE 是几个 OLE 服务器的混合体。下面列出了一些用于实现 OLE 文档接口的接口：

- IOleItemContainer
- IPersistFile
- IClassFactory
- IOleInPlaceActiveFrame
- IOleInPlaceSite

ColeDocument 在 MFC 中封装了 OLE 文档和 ActiveX 文档的功能。但是应注意的是一系列 MFC 类完成了这些功能。后面的章节将详细讨论这些类。

在位激活(In-Place Activation)

OLE 文档支持在位激活，或者一般所说的可视化编辑，这使我们可以在容器应用中编辑一个嵌入对象。而且嵌入对象看起来就象是容器的本地对象。当我们在容器中激活可视化编辑，容器的用户接口支持选择的、创建该对象的服务器应用程序的用户接口函数。有一系列完整的接口可完成和支持在位激活功能。这些接口都以 IoleInPlace 开头，下面的接口可用来实现并支持在位激活功能：

- IOleInPlaceObject
- IOleInPlaceActiveObject
- IOleInPlaceSite
- IOleInPlaceActiveSite
- IOleInPlaceActiveFrame

16.1.6 OLE 自动化

OLE 自动化允许通过使用高级的宏语言或脚本语言如 VBScript 和 JavaScript 在一个应用程序内部操作另一个应用程序的属性和方法。这样可以来定制对象并提供应用程序间的互操作性。

在 OLE 自动化的世界里，有 OLE 自动化组件和 OLE 自动化控制程序。一个 OLE 自动化组件就是一项由一个应用程序提供而被另一个应用程序使用的服务。微软的 Excel 就是这方面很好的例子，因为它提供创建和操作表页、单元格、行列等服务。通过 OLE 自动化组件得到的服务被存储在类型库中。这个库是以 .TLB 为扩展名的二进制文件。OLE 自动化组件是用对象描述语言定义的，这种语言所组成的文件扩展名是 DDL。DDL 文件被编译进入一个类型库中。OLE 自动化客户是指使用 OLE 自动化提供服务的控制应用程序。OLE 自动化客户通过 IDispatch 接口工作。这个接口向控制应用程序开放可用服务。

16.1.7 OLE 控件

如前面所讨论的，OLE 控件是一个被嵌入应用程序的自包含可重用组件。OLE 控件也可以被认为是可以在编译或运行时刻设置属性的 OLE 自动化对象。OLE 控件还包含完成特定操作的方法。OLE 控件和自动化对象的区别是 OLE 控件是一个容器对象。

ActiveX 使用了 OLE 技术并使它超越了本地机的范围，进入了一般的企业网和 Internet。特别的是，OLE 技术的使用已经扩大到以下的 ActiveX 服务中。

- ActiveX 文档
- ActiveX 控件
- COM
- Internet monikers

这些还不是全部。OLE 的基本要素还将在新的 ActiveX 技术中体现。现在开始，让我们注意 OLE 向 Active 演化进程。

16.2 ActiveX 文档

ActiveX 是 OLE 文档在网络上的发展应用。ActiveX 通过 Web 浏览器可以激活现存的 OLE 文档，如 Microsoft Project 和 Microsoft Powerpoint。这样就有了一个带多种数据的复合文档，其中包含了通过 WWW (World Wide Web)可以访问的链接和嵌入对象。使用 ActiveX 的超级链接技术，你可以在 Web 上扩展 OLE 文档。

异步存储

在 WWW 上使用 ActiveX 文档的能力引起另一项 ActiveX 技术的诞生，即异步存储。基本上，它是结构化存储在 Web 上的扩展，它允许存储能异步的进行。很明显，由于 Internet 的带宽影响，如果允许存储同步地进行，那么在数据到来之前或没有发生永久存储，无论客户还是服务器都不能做任何事。使用 ActiveX 超级链接和异步 monikers 技术，异步存储就可以实现。

16.2.1 ActiveX 控件

ActiveX 控件就是 OLE 控件和 OCX 在 Internet 环境上的扩展。微软已经用 ActiveX 控件替代 OLE 控件。记住，OLE 是一个可扩展的结构。因而那些可重用的组件不仅可以嵌入到 Web 页，而且还可以用在非网络应用程序中。

ActiveX 控件可以用不同的编程语言创建，包括 C，C++，Java 和下一个版本的 Visual Basic。它们也可以通过 VBScript，JavaScript 被控制，因此非编程人员也能使用它们。

ActiveX 控件是一个很大的组件，因为可以使用大量的含有预制功能的小组件。ActiveX 的能力很无尽的。目前，ActiveX 控件从一个日历控件到图片控件都有。图片控件可以显示静态图象。

COM 是 ActiveX 的基础。ActiveX 控件是建立在一系列以 COM 为基础的 OLE 服务基础之上的。下面描述的是 ActiveX 控件的技术。

- 组件对象模型
- 可链接对象

- 统一数据传输
- OLE 文档
- 属性页
- 永久存储
- OLE 自动化

1. ActiveX 控件：COM

象前面讨论的 OLE 控件一样，ActiveX 控件也是 COM 对象。它们是进程中 OLE 自动服务器，并可以由外部被激活。像其他 COM 对象一样，它们对外有 IUnknown 接口，这样容器应用程序就可以通过接口返回的指针访问它们的属性和方法。

2. ActiveX 控件：可链接对象

ActiveX 支持两种从控件到客户应用程序间的通讯方式。这种通讯方式称作可链接对象。它能使控件向客户通知事件或激发一个方法或事件。它也可以使客户和控件之间直接进行通讯。

3. ActiveX 控件：统一数据传输

控件可以进行拖放操作只要客户应用程序支持这项功能。

4. ActiveX 控件：复合文档

在本章开始，你就看到了一个对象如何从另一个应用程序嵌入主应用程序。并且，这个对象可以被在位激活进行可视化编辑。同样 OLE 控件是建立在 OLE 文档基础之上的，所以也能进行可视化编辑。

5. ActiveX 控件：属性页

像他们原有的 OLE 控件一样，ActiveX 控件也有属性页，它把自己的属性和方法对用户开放。在属性页中，属性可以被自由设置。

6. ActiveX 控件：OLE 自动化

ActiveX 控件也是一个自动服务器。它的属性和方法可以通过属性页在编译时被设置，也可以通过 VBScript 和 JavaScript 在运行时设置。

7. ActiveX 控件：永久存储

COM 对象可以用不同种方法进行永久存储。ActiveX 用永久存储保存它的状态。这能把控件的状态初始化到最后使用时的状态。

16.3 COM

正如以前所学的，COM 是对象的一个二进制标准，在 ActiveX 产生之前 COM 已经有了。要想在 Internet 上使用和交换对象，OLE 则需要被扩展。这就导致了分布式 COM 的形成。

分布式 COM (DCOM)

分布式 COM 又称 DCOM,通常被认为是网络 OLE。它是在二进制 COM 对象的基础上,在 LAN、WAN 和 Internet 上的延伸。现在就可以通过网络实例化,并捆绑一个对象了。

16.3.1 Internet Monikers

随着 ActiveX 的出现和 COM 在网络上的扩展,moniker 也进行了必要的扩展。出现了两种新的类型:

- URL moniker
- 异步 moniker

1. URL moniker

URL (uniform resource locator)也就是统一资源定位器。它用于对象的 Web 基的寻址。moniker 是一个智能命名系统,可以通过使用 IMoniker 接口对 moniker 对象和智能化名字来定位一个对象。这种能力也同样被扩展到 URL,因为它有通过 DCOM 在网络上传递对象的能力。

2. 异步 moniker

以前,moniker 是同步将它们捆绑到一个对象上的。在捆绑完成之前任何事都做不了。在低速、长等待时间的 Internet 上等待传输完成,将是无法忍受的。使用异步 moniker,对象的接口可以协商捆绑进程的传输来异步完成捆绑。目前,异步 moniker 的实现仅为 URL moniker。

16.4 ActiveX 技术

ActiveX 还有一些与 OLE 无关的技术。这些技术使在 WWW 上创建交互式的应用程序变得方便。它们是:

- ActiveX 超级链接
- ActiveX 会议
- ActiveX 服务器扩展
- 代码签名
- HTML 扩展
- ActiveMovie

16.4.1 ActiveX 超级链接

ActiveX 超级链接是允许在 HTML 文件中在位激活一个非 HTML 文件。使用 ActiveX 文档容器,可以用超级链接在 HTML 文档中访问 Microsoft Word、Microsoft Excel、Microsoft PowerPoint、Visio 和 CorelDRAW 等文档。

16.4.2 ActiveX 协议

ActiveX 会议服务是一套可以在 Internet 上实时地多路、多媒体通信的技术。这很像电视

会议，只不过可以在 PC 机上做到这一点。想一想，这对编程人员意味着什么，我们可以在家里做所有的工作并进行远程通信。这是一个编程的、开放的、有无尽革新能力的接口体系。

16.4.3 ActiveX 服务器扩展

ActiveX 服务器扩展以前称为 ISAPI (Intetnet Server API)是用来增强 Internet 服务器的功能的。以前，做这方面的工作只能用 CGI (Common Gateway Interface)代码。ActiveX 服务器扩展提供了一种可选择的方法。通常，它使用动态连接库来实现并提供一些 HTTP 服务器没有提供的功能。比如，链接到一个数据库。

16.4.4 ActiveX Script

ActiveX Script 把 OLE 自动化带进了 Internet，自动化控件现在可以使用 DCOM 和支持脚本的 ActiveX，通过 Internet 来访问自动化组件服务了。可以用不同的脚本语言，如 VBScript、JavaScript、Visual Basic for Application、Lisp 和 Scheme。

16.4.5 代码签名

代码签名是一种为代码确定电子标签的新技术。它能为网络上交互作用的应用程序提供安全保证。基本上，软件商都在自己的软件中提供符合一定规范的数字签名。在客户一边，当应用程序或组件从网络下载时，它将调用 Win32 API WinVerifyT ()这个函数检查并核实这个数字签名。

16.4.6 HTML 扩展

HTML (Hypertext Markup Language)是一种所有基于 Web 文档产品的语言。为了支持 ActiveX 控件和 ActiveX 脚本，必须对 HTML 语言进行扩展。此外，Web 浏览器必须修改以提供新的语言扩展。它还可以修改以适用新的语言。可以用 HTML 的 < OBJECT > 标签把一个 ActiveX 控件加到 Web 页中。

16.4.7 ActiveMovie

ActiveMovie 是一种用来替换老式媒体控制接口和 Windows 视频的新技术。它是一个声音流视频流框架标准。使用它，可以播放 MPEG、AVI 和 Apple QuickTime 电影。

ActiveX 控件通过事件告诉容器对该控件进行了操作。常用的事件包括：单击控件、键盘输入数据以及改变在进行控件的方面变化、状态。当发生这些行动时，控件会将对应的事件报告给容器。

MFC 支持两种事件：储备事件和自定义事件。储备事件是那些由 COleControl 类自动处理的事件。自定义事件是当一个特殊的行动发生在控件上时，允许控件通知容器。

控件要正确驱动事件的话，控件类必须映射每个控件到一个成员函数，当相应的事件发生时，该成员函数被调用。这种映射机制(称为事件映射表)集中了事件的信息，并且允许 ClassWizard 方便地访问和操作控件事件。这个事件映射表声明在控件类声明的头文件中，使用下面这个宏声明：

DECLARE _ EVENT _ MAP

图 16.3 是 ClassWizard 中的 ActiveX Event 选项卡，使用该选项卡可增加自定义事件和储

备事件。

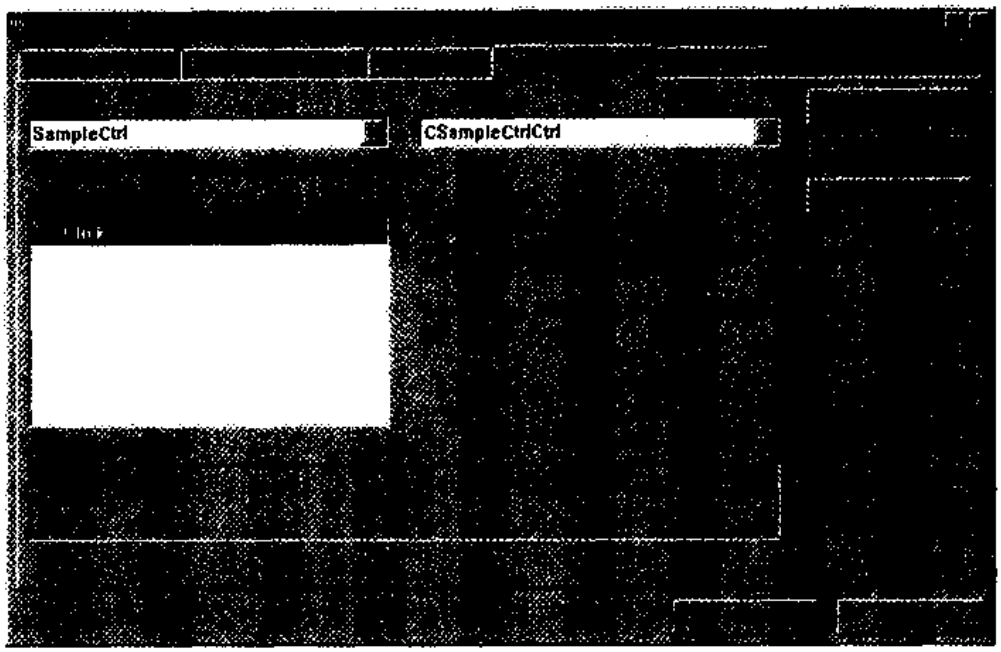


图 16.3　MFC ClassWizard 中的 ActiveX Event 选项卡

一旦声明了事件映射表,就必须在控件的实现文件(.CPP 文件)中进行定义。下面几行代码定义是定义事件映射表:

```
BEGIN_EVENT_MAP(CSampleCtrl, COleControl)
//{{AFX_EVENT_MAP(CSampleCtrl)
...
//}}AFX_EVENT_MAP
END_EVENT_MAP()
```

如果使用 ControlWizard 创建工程文件,将自动地增加这几行代码。如果不使用 ControlWizard,就必须人工增加这几行代码。

通过 ClassWizard,可以增加 COleControl 类支持的储备事件,或者自己定义的自定义事件。对于每一个新的事件,ClassWizard 自动地增加适当的入口至控件事件映射表和控件的 ODL 文件中。

16.4.8　ActiveX 控件的储备事件

储备事件不同于自定义事件,它们由 COleControl 类自动地驱动。COleControl 类包含常用动作的驱动事件的预定义成员函数,该类可实现的一些常用的动作包括单击和双击控件,键盘事件、按钮状态的改变。储备事件的事件映射入口总是以 EVENT_STOCK 为前缀。

1. ClassWizard 支持的储备事件

下面列出了 COleControl 类提供的九个储备事件,可以在 ClassWizard 的 ActiveX Events 制表页中指定与控件相应的事件。

(1) 事件 1:Click

驱动函数:void FireClick()

说明:当控件捕获到鼠标,并收到任何一个 BUTTONUP 消息(左键、中键或右键),而且在该控件之上释放鼠标,则发生该事件。储备事件 MouseDown 和 MouseUp 发生在该事件之前。

事件映射入口:EVENT _ STOCK _ CLICK()

(2) 事件 2:DblClick

驱动函数:void FireDblClick()

说明:类似于 Click 事件,不同的是当接收到 BUTTONDBLCLK 消息时发生该事件。

事件映射入口:DVENT _ STOCK _ DBLCLICK()

(3) 事件 3:Error

驱动函数:void FireError(SCODE scode,LPCSTR lpszDescription,UINT nHelpID = 0)

说明:当你的 ActiveX 控件内部出错时(不包括方法调用和属性访问)发生。

事件映射入口:EVENT _ STOCK _ ERROR()

(4) 事件 4:KeyDown

驱动函数:void FireKeyDown (short nChar,short nShiftState)

说明:当接收到 WM _ SYSKEYDOWN 或者 WM _ KEYDOWN 消息时发生。

事件映射入口:EVENT _ STOCK _ KEYDOWN()

(5) 事件 5:KeyPress

驱动函数:void FireKeyPress(short * pnChar)

说明:当接收到 WM _ CHAR 消息时发生。

事件映射入口:EVENT _ STOCK _ KEYPRESS()

(6) 事件 6:KeyUp

驱动函数:void FireKeyUp(short nChar,short nShiftState)

说明:当接收到 WM _ SYSKEYUP 或者 WM _ KEYUP 消息时发生。

事件映射入口:EVENT _ STOCK _ KEYUP()

(7) 事件 7:MouseDown

驱动函数:void FireMouseDown(short nButton,short,nShiftState,float x,float y)

说明:接收到任何 BUTTONDOWN 消息(鼠标左键、中键或右键)时发生。

注意:鼠标捕获在事件发生之前。

事件映射入口:EVENT _ STOCK _ MOUSEDOWN()

(8) 事件 8:MouseMove

驱动函数:void FireMouseMove(short nButton,short nShiftState,float x,float y)

说明:当接收到 WM _ MOUSEMOVE 消息时发生。

事件驱动入口:EVENT _ STOCK _ MOUSEMOVE()

(9) 事件 9:MouseUp

驱动函数:void FireMouseUp(short nButton,short nShiftState,float x,float y)

说明:接收到任何 BUTTONUP 消息(鼠标左键、中键或右键)时发生。

注意:鼠标捕获在事件发生之前。

事件驱动入口:EVENT _ STOCK _ MOUSEUP()

2. 使用 ClassWizard 增加储备事件

增加储备事件所要做的工作比增加自定义事件要少一些,因为实际事件的触发已经由基类 COleControl 自动完成了。下面介绍如何给一个用 ControlWizard 创建的控件增加储备事件。该事件称为 KeyPress,当一个键被按下,同时该控件处于活动状态时,触发该事件。这个过程也能用来增加其他储备事件,只是改变所要求的储备事件名字。

(1) 调入你的控件的工程文件。

(2) 在 View 菜单中单击 Class Wizard。

(3) 选择 ActiveX Event 制表页。

(4) 从 Class Name 框中选择控件类名。

(5) 单击 add Event。

(6) 在 External Name 框中单击 KeyPress。

(7) 单击 OK。

(8) 再次单击 OK,确定你的选择并退出 ClassWizard。

3. ClassWizard 对储备事件的改变

因为储备事件是由控件的基类处理的,ClassWizard 不会改变类的声明,它只是增加事件至控件映射表,并且在 ODL 文件中建立一个入口。下面这行语句是在控件类实现文件(.CPP)的控件映射表中增加的一行语句:

EVENT _ STOCK _ KEYPRESS()

增加的这行代码表示,当控件处于活动状态,并接收到 WM _ CHAR 消息时,发生 KeyPress 事件。

ClassWizard 还在控件的 ODL 文件中增加下面这行代码:

[id(DISPID _ KEYPRESS)] void KeyPress(short * KeyAscii);

这行语句把 KeyPress 事件和标准调度 ID 联系起来,使容器预先知道 KeyPress 事件。

16.4.9 ActiveX 控件的自定义事件

自定义事件与储备事件不同,它不是由 COleControl 类自动触发的。自定义事件是把控件开发者定义的一个行为作为一个事件,事件映射入口由宏 EVENT _ CUSTOM 来表示。下面使用 ClassWizard 实现 ActiveX 控件工程中的自定义事件。

1. 使用 ClassWizard 增加自定义事件

下面的过程是增加一个自定义事件 ClickIn。可以使用这个过程增加其他自定义事件,只是简单地更改自定义事件的名字和参数。

(1) 调入控件的工程文件;

(2) 在 View 菜单中单击 ClassWizard;

(3) 选择 ActiveX Event 制表页;

(4) 从 Class Name 框中选择的控件类名;

(5) 单击 Add Event;

(6) 在 External Name 框中输入 ClickIn;

(7) 在 Internal name 框中事件驱动函数的名称。在这个例子中,使用 ClassWizard 提供的缺省名称 FireClickIn;

(8) 增加一个称为 xCood 的参数(类型为 OLE _ XPOS _ PIXELS);

(9) 增加一个称为 yCood 的参数(类型为 OLE _ XPOS _ PIXELS);

(10) 单击 OK;

(11) 再次单击 OK,确定你的选择并退出 ClassWizard。

2. ClassWizard 对自定义事件的改变

当增加自定义事件时,ClassWizard 改变了控件类的.H 文件、.CPP 文件和.ODL 文件。

下列代码被增加到控件类的头文件(.H)中:

```
void FireClickIn(OLE _ XPOS _ PIXELS xCoord,OLE _ YPOS _ PIXELS yCoord)
{FireEvent(eventidClickIn,EVENT _ PARAM
(VTS _ XPOS _ PIXELS,VTS _ YPOS _ PIXELS),xCoord,yCoord);}
```

这些代码声明一个称为 FireClickIn 的内联函数,调用一个带有 ClickIn 事件和参数的函数 COleControl::ClickIn,这个函数是使用 ClassWizard 定义的。

此外,在控件类的实现文件(.CPP)中还增加了控件的事件映射项,如下:

EVENT _ CUSTOM("ClickIn", FireClickIn, VTS _ XPOS _ PIXELS, VTS _ YPOS _ PIXELS)

这行代码映射事件 ClickIn 到内联函数 FireClickIn,并传递所定义的参数。

最后,在控件的.ODL 文件中还增加了下面这行语句:

```
[id(1)] void ClickIn(OLE _ XPOS _ PIXELS xCoord,OLE _ YPOS _ PIXELS yCoord);
```

这条语句分配给 ClickIn 事件一个指定的 ID 号,这是 ClassWizard 事件列表中的事件位置。事件列表中的这个入口,允许容器预先知道该事件,例如,当事件触发时,可以提供可执行的处理代码。

3. 调用 FireClickIn

现在,已经使用 ClassWizard 增加了自定义事件 ClickIn,必须确定该事件何时触发。当相关的行为发生时,你必须调用函数 FireClickIn 来实现该事件。对于这个例子,当用户在一个圆形或矩形区域单击鼠标时,该控件利用 WM _ LBUTTONDOWN 消息处理中的 InCircle 函数触发 ClickIn 事件。下面是增加 WM _ LBUTTONDOWN 处理过程:

(1) 调入控件的工程文件;

(2) 在 View 菜单中单击 ClassWizard。

(3) 选择 Message Maps 制表页;

(4) 从 Object IDs 框中选择控件类名,本例中为 CSampleCtrl;

(5) 在 Message 框中选择要处理的消息,本例中选择 WM _ LBUTTONDOWN;

(6) 单击 Add Function,在应用程序中增加一个消息处理函数;

(7) 单击 Edit Code,跳到文件中该消息处理函数的位置,或者单击 OK,确定你的选择。

下面这些代码表示每次在控件窗口中单击鼠标左键时,调用 InCircle 函数:

```
void CSampleCtrl::OnLButtonDown(UINT nFlags,CPoint point)
{
```

```
if (InCircl(point))
FireClickIn(point.x,point.y);
COleControl::OnLButtonDown(nFlags,point);
}
```

注意：当 ClassWizard 创建关于鼠标动作的消息处理函数时，自动增加一个对基类的同一个消息处理函数的调用。不要删除这个调用。

在下例中，事件仅当发生在控件的圆形或矩形区域时触发，要实现这个功能，在控件的实现文件(.CPP)中增加下面的 InCircle 函数：

```
BOOL CSampleCtrl::InCircle(CPoint& point)
{
CRect rc;
GetClientRect(rc);
//Determine radii
double a=(rc.right-rc.left)/2;
double b=(rc.bottom-rc.top)/2;

//Determine x,y
double x=point.x-(rc.left+rc.right)/2;
double y=point.x-(rc.left+rc.right)/2;

//Apply ellipse formula
return (x * x)/(a * a)+(y * y)/(b * b)<=1);
}
```

还需要增加 InCircle 函数的声明到控件的头文件(.H 文件)中：

```
BOOL InCircle(CPoint& point);
```

16.5 ActiveX 控件的方法

16.5.1 ActiveX 控件的方法

ActiveX 控件触发事件与它的控件容器进行通信，容器也通过方法和属性与控件进行通信。方法和属性提供了其他应用程序(如自动化客户和 ActiveX 控件容器等)所使用的输出接口。

方法的使用和作用类似于C++类中的成员函数。控件能实现两种类型的方法：储备方法和自定义方法。与储备事件一样，储备方法是那些由类 COleControl 提供使用的方法；自定义方法由开发者定义，允许附加地定做控件。

Microsoft 基本类库实现一种机制，使控件支持储备方法和自定义方法。该机制的第一个部分是 CWnd 类的派生类 COleControl，该类的成员函数支持所有 ActiveX 控件的储备方法。第二个部分是调度映射，类似于消息映射，但不是将函数映射到窗口消息 ID 上，而是将虚拟成员函数映射到 IDispatch ID。

对于一个支持各种方法的控件，它的类必须声明调度映射，通过在控件类的头文件(.H 文件)中加上下面这行语句实现：

DECLARE_DISPATCH_MAP()

调度映射的主要目的是在被外部调用者(如容器)使用的方法名称与实现该方法的控件类的成员函数之间建立一个关系。在声音一个调度映射时，必须在控件的实现文件(.CPP 文件)中定义该调度映射，如下面几行代码所示：

```
BEGIN_DISPATCH_MAP(CSampleCtrl,COleControl)
//{{AFX_DISPATCH_MAP(CSampleCtrl)
...
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

如果使用 ControlWizard 建立工程，这几行代码将自动加入，如果没使用 ControlWizard，就必须手工加入这几行代码。

16.5.2 ActiveX 控件的储备方法

储备方法与自定义方法的不同之处在于，储备方法已经在类 COleControl 中定义了实现。例如，COleControl 包含预先定义的成员函数，支持控件的 Refresh 方法，这个储备方法的调度映射入口是 DISP_STOCKFUNC_REFRESH。

COleControl 类支持两个储备方法：DoClick 和 Refresh。Refresh 方法在用户立即更改控件显示的时候被调用，DoClick 方法是触发控件的 Click 事件时被调用。下面介绍了这两个方法：

• 方法：DoClick

调度映射入口：DISP_STOCKPROP_DOCLICK()

说明：触发一个 Click 事件。

• 方法：Refresh

调度映射入口：DISP_STOCKPROP_REFRESH()

说明：立即更新控件的显示

1. 使用 ClassWizard 增加储备方法

增加储备方法只需要单使用 ClassWizard，下面的过程介绍了给 ControlWizard 创建的控件增加一个 Refresh 方法。

(1) 调入控件的工程文件。

(2) 在 View 菜单中单击 ClassWizard 选项。

(3) 选择 Automation 制表页。

(4) 在 Class Name 框中单击这个控件类。

(5) 单击 Add Method(见图 16.4)。

(6) 在 External name 框中，单击 Refresh。

(7) 单击 OK。

(8) 再次单击 OK，确定你的选择并关闭 ClassWizard。

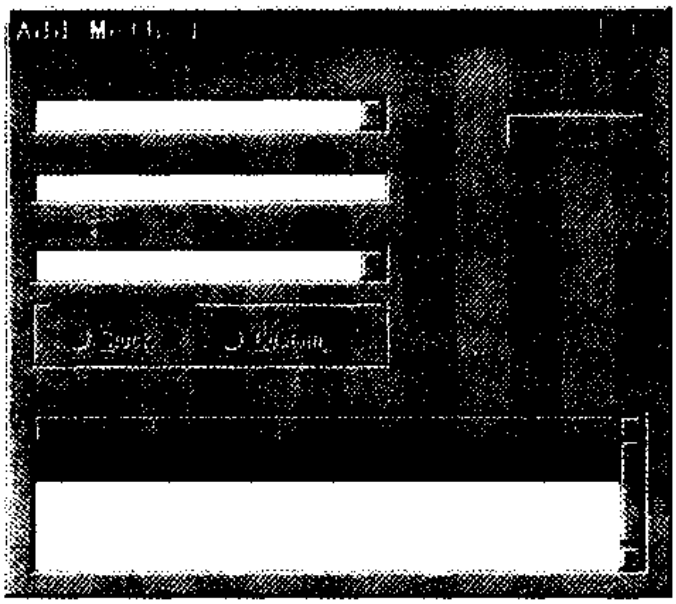


图 16.4　显示了 ClassWizard 中的 Add Method 对话框。

2. ClassWizard 对储备方法的改变

因为储备方法 Refresh 是控件基类支持的，所以 ClassWizard 不改变控件的类声明，它只是把方法的入口增加到控件调度映射表和 ODL 文件中。下面这行语句将增加到 CPP 文件的控件调度映射表中：

DISP_STOCKFUNC_REFRESH()

这使得用户能使用到该方法。

下面这行语句增加到控件的 ODL 文件中：

[id(DISPID_REFRESH)]void Refresh();

16.5.3　ActiveX 控件的自定义方法

自定义方法与储备方法的不同之处在于，COleControl 类中并没有定义它的实现，必须为每一个增加到控件中的自定义方法定义它的实现。

ActiveX 控件的用户可以在任何时候调用自定义方法以完成控件的特定动作，自定义方法的调度映射入口的格式为 DISP_FUNCTION。

1. 用 ClassWizard 增加自定义方法

下面介绍如何增加自定义方法 PtInCircle 至 ActiveX 控件，PtInCircle 确定鼠标是否在控件所定义的范围之内。其他的自定义方法也可使用同样的过程，只要改一改自定义方法的名称和参数就行了。

(1) 调入控件的工程文件

(2) 在 View 菜单中单击 ClassWizard 选项。

(3) 选择 Atuomation 制表页。

(4) 在 Class Name 框中选择这个控件类。

(5) 单击 Add Method。

(6) 在 External name 框中,输入 PtInCircle。

(7) 在 Internalname 框中,输入方法的内部函数的名称,或者使用 ClassWizard 提供的缺省名(在这个例子中为 PtInCircle)。

(8) 在 Return Type 框中,单击 BOOL,作为方法的返回值。

(9) 增加参数 xCoord(类型为 OLE_XPOS_PIXELS)

(10) 增加参数 yCoord(类型为 OLE_YPOS_PIXELS)

(11) 单击 OK,关闭 Add Method 对话框。

(12) 单击 OK,确定选择并关闭 ClassWizard。

2. ClassWizard 对自定义方法的改变

当增加一个自定义方法时,ClassWizard 在控件类的头文件(.H 文件)和实现文件(.CPP 文件)中做了一些改变。下面这行语句增加到头文件中的调度映射表声明中:

afx_msg BOOL PtInCircle(long xCoord,long yCoord);

这行语句声明了一个方法处理函数,称为 PtinCircle,用户使用外部函数名 PtInCircle 就可以调用该函数。

下面这行语句将增加到控件的 ODL 文件中:

[id(1)]boolean PtInCircle(OLE_XPOS_PIXELS xCoord, OLE_YPOS_PIXELS yCoords);

这条语句为 PtInCircle 方法分配了一个 ID 号,是从 ClassWizard 的方法和属性列表中的方法位置中指定的。因为使用了 ClassWizard 来增加自定义方法,所以方法入口被自动地添加到工程的 ODL 文件中。

此外,下面这条语句被增加到控件类的 CPP 文件中,增加在控件的调度映射表中:

DISP_FUNCTION(CSampleCtrl,"PtInCircle",PtInCircle,VT_BOOL,VTS_I4 VTS_I4)

宏 DISP_FUNCTION 映射方法 PtInCircle 到控件的处理函数 PtInCircle 中,并声明返回值类型为 BOOL。

最后,ClassWizard 增加函数 CSampleCtrl::PtinCircle 的定义到该控件类的 CPP 文件的后部:

```
BOOL CSampleCtrl::PtInCircle(short xCoord,short yCoord)

return InCircle(CPoint(xCoord,yCoord));
```

16.6 ActiveX 控件的属性

16.6.1 ActiveX 控件的属性

ActiveX 控件通过触发事件传送信息给控件容器,反过来,容器利用方法和属性来传送信息给控件。方法和属性的作用分别类似于 C++ 类中的成员函数和成员变量,属性是 ActiveX

控件的数据成员，为应用程序提供接口，这些应用程序包含 ActiveX 控件，诸如自动化客户和 ActiveX 控件容器等。

ActiveX 控件能执行储备和自定义的方法和属性。类 COleControl 为储备属性提供了实现代码，而自定义属性则是由开发者自己定义，以增加特殊的能力至 ActiveX 控件。

16.6.2 ActiveX 控件的储备属性

储备属性与自定义属性的不同之处在于，储备属性已经在类 COleControl 中定义并实现。COleControl 类中包含预先定义的支持控件的通用属性的成员函数，这些通用的函数包含控件的标题，以及前景和背景颜色。储备属性的调度映射入口总是加上前缀 DISP _ STOCKPROP。

本节描述如何给 ClassWizard 创建的 ActiveX 控件增加一个储存属性(所举例子中的属性是标题)，并解释改过后的代码。

本小节的主要内容包括：

- 使用 ClassWizard 增加一个储备属性；
- ClassWizard 为增加储备属性所做的更改；
- ClassWizard 支持的储备属性；
- 储备属性和通知；
- 颜色属性。

1. 使用 ClassWizard 增加一个储备属性

增加储备属性所要做的工作比增加自定义属性要少一些，因为属性的支持已经由基类 COleControl 自动完成了。下面步骤介绍如何给一个 ActiveX 控件增加储备的标题属性，这个过程也能用来增加其他储备属性，只是需改变所要求的储备属性名字。

(1) 调入控件的工程文件。

(2) 在 View 菜单中单击 ClassWizard。

(3) 单击 Automation 制表页。

(4) 从 Class Name 框中选择控件类名。

(5) 单击 Add Property。

(6) 在 External Name 框中单击 Caption。

注意：在 Implementation group 中，自动选择 Stock。

(7) 单击 OK，关闭 Add Property 对话框。

(8) 再次单击 OK，确定选择并退出 ClassWizard。

2. ClassWizard 为增加储备属性所做的更改

因为 COleControl 类支持储备属性，所以 ClassWizard 不会改变类的声明，它只是增加属性至调度映射表。下面这行语句是在该控件类实现文件(.CPP)的调度映射中增加的一行语句：

```
DISP_STOCKPROP_CAPTION()
```

ClassWizard 还在控件的对象描述文件(ODL 文件)中增加下面这行代码：

```
[id(DISPID_CAPTION),bindable,requestedit]BSTR Caption;
```

这行语句给 Caption 属性分配一个指定的 ID 号。

这使得控件的用户可以使用 Caption 属性，具体使用该属性的值，可以访问 COleControl 基类的成员变量和成员函数。

3. ClassWizard 支持的储备属性

COleControl 类提供了九个储备属性，可以在 ClassWizard 对话框的 Automation 制表页中指定控件所要的属性。

这九个储备属性是：

(1) Appearance

调度映射入口：DISP_STOCKPROP_APPEARANCE()

可使用的值：m_sAppearance

(2) BackColor

调度映射入口：DISP_STOCKPROP_BACKCOLOR()

可使用的值：调用 GetBackColor

(3) BorderStyle

调度映射入口：DISP_STOCKPROP_BORDERSTYLE()

可使用的值：mBorderStyle

(4) Caption

调度映射入口：DISP_STOCKPROP_CAPTION()

可使用的值：调用 InternalGetText

(5) Enabled

调度映射入口：DISP_STOCKPROP_ENABLED()

可使用的值：m_bEnabled

(6) Font

调度映射入口：DISP_STOCKPROP_FONT()

“AvtiveX 控件的字体的使用”一节。

(7) ForeColor

调度映射入口：DISP_STOCKPROP_FORECOLOR()

可使用的值：调用 GetForeColor

(8) hWnd

调度映射入口；DISP_STOCKPROP_HWND()

可使用的值：m_hWnd

(9) Text

调度映射入 口：DISP_STOCKPROP_TEXT()

可使用的值：调用 InternalGetText，这个属性和 Caption 属性除了属性名不同以外，其他都一样。

4. 储备属性和通知

大多数储备属性都有可重载的通知函数。例如，只要 BackColor 属性发生改变，就会调用 OnBackColorChanged 函数(该控件类的成员函数)，在 COleControl 类中缺省的执行函数是 In-

validateControl，如果想给这个响应增加一些功能，可以重载该函数。

5. 颜色属性

当绘制一个控件，可以使用储备的 ForeColor 和 BackColor 属性，或者自己自定义颜色属性。要使用颜色属性，调用 COleControl::TranslateColor 成员函数。这个函数的参数是颜色属性的值和任意一个调色板句柄，返回值为 COLORREF，能传递给 GDI 函数，类似函数 SetTextColor 和 CreateSolidBrush。

储备属性 ForeColor 和 BackColor 的值可分别调用函数 GetForeColor 和 GetBackColor 来访问。

下面举一个绘制控件时使用这两个颜色属性的例子。先调用函数 TranslateColor 初始化一个临时的 COLORREF 变量和 CBrush 对象，一个使用 ForeColor 属性，另一个使用 BackColor 属性。然后，使用 CBrush 对象描绘控件的矩形框，而使用 ForeColor 属性设置正文的颜色。

```
CBrush bkBrush(TranslateColor(GetBackColor()));
COLORREF clrFore = TranslateColor(GetForeColor());
pdc->FillRect(rcBounds, &bkbrush);
pdc->SetTextColor(clrFore);
pdc->DrawText(InternalGetText(),-1,rcBounds,
DT_SINGLELINE|DT_CENTER|DT_VCENTER);
```

16.6.3 ActiveX 控件的自定义属性

自定义属性与储备属性的不同之处在于，COleControl 类中没有包含它的实现。

这一节将描述如何使用 ClassWizard 增加自定义属性至 ActiveX 控件上，并讲解因此而产生的代码的改变，包括下面两个内容：

- 使用 ClassWizard 增加自定义属性。
- ClassWizard 对增加自定义属性的代码改变。

自定义属性使用四种不同的实现方式：成员变量、带通知的成员变量、Get/Set 方法以及参数方式。

- 成员变量的实现方式

这种实现方式把属性的状态作为控件类的一个成员变量。如果属性值何时改变并不很重要时，可使用这种成员变量的实现方式。与其他三种方式比起来，这种实现方式生成的代码最少。

设置调度映射入口的宏为 DISP_PROPERTY。

- 带通知的成员变量的实现方式

这种实现方式包括成员变量和 ClassWizard 建立的通知函数，当属性值改变时，自动调用通知函数。当属性值发生改变时，希望通知用户时，使用这种带通知的成员变量方式。因为需要调用函数，所以这种方式的速度比较慢。

设置调度映射的宏为 DISP_PROPERTY_NOTIFY。

- Get/Set 方法的实现方式

这种实现方式由控件类中的一对成员函数组成，当用户需要属性的当前值时，自动调用 Get 成员函数，而当用户需要改变属性时，自动调用 Set 成员函数。这种实现方式用于需要在

运行时对属性值进行操作，赋一个有效的值改变实际属性，或者实现一个只读或只写的属性类型。

设置调度映射入口的宏为 DISP_PROPERTY_EX。在后面将通过 ClassWizard 增加一个 CircleOffset 自定义属性来说明这种实现方式。

• 参数实现方式

ClassWizard 支持参数实现方式。带参数的属性（有时称为属性数组）用于访问控件的单一属性的值。这种方式的调度映射入口是 DISP_PROPERTY_PARAM。

1. 使用 ClassWizard 来增加自定义属性

在这里演示一个增加自定义属性 CircleOffset 的例子，使用 Get/Set 方法的实现方式。使用其他几种增加自定义属性的方式也很简单。

（1）调入控件的工程文件。

（2）在 View 菜单中单击 ClassWizard。

（3）单击 Automation 制表页。

（4）从 Class Name 框中选择控件类名。

（5）单击 Add Property。

（6）在 External Name 框中输入 CircleOffset。

（7）在 Implementation 框中单击 Get/Set Methods。

（8）在 Type 框中选择属性类型为 short。

（9）为 Get 和 Set 函数输入一个唯一的函数名，或者接受缺省的函数名。

（10）单击 OK，关闭 Add Property 对话框。

（11）再次单击 OK，确定选择并退出 ClassWizard。

2. ClassWizard 对增加自定义属性的代码改变

当增加 CircleOffset 自定义属性时，ClassWizard 在头文件（.H 文件）和实现文件（.CPP）中更改了一些代码。

在头文件中声明了两个函数 GetCircleOffset 和 SetCircleOffset，如下所示：

```
afx_msg short GetCircleOffset();
afx_msg void SetCircleOffset(short nNewValue);
```

在控件的 ODL 文件中增加了下面这行语句：

```
[id(1)] short CircleOffset;
```

这条语句给 CircleOffset 属性分配了一个 ClassWizard 方法和属性列表中的 ID 号。

此外，还在控件的 CPP 文件中的调度映射表中增加了下面这条语句，给 CircleOffset 属性映射了控件的两个处理函数。

```
DISP_PROPERTY_EX(CSampleCtrl,"CircleOffset",
GetCircleOffset,SetCircleOffset,VT_I2)
```

最后，在控件的 CPP 文件的末尾增加函数 GetCircleOffset 与 SetCircleOffset 的定义。通常，你要使 Get 函数返回一个属性值，而在属性改变之前和之后执行 Set 函数中包含的代码。

```
void CFooCtrl::SetCircleOffset(short nNewValue)
{
```

```
//TODO: Add your property handler here
SetModifiedFlag();
}
```

我们注意到，ClassWizard 在 Set 函数的定义中自动增加对函数 SetModifiedFlag 的调用，这个函数是当属性改变时设置一个标志。

16.6.4 ActiveX 控件的高级属性

这一节介绍有关 ActiveX 控件的高级的属性的实现：

- 只读和只写属性；
- 返回属性的错误代码。

1. 只读和只写属性

ClassWizard 提供了一个快速而简单的方法增加控件的只读和只写属性，具体步骤如下：

(1) 调入控件的工程文件。

(2) 在 View 菜单中单击 ClassWizard。

(3) 单击 Automation 制表页。

(4) 从 Class Name 框中选择控件类名。

(5) 单击 Add Property。

(6) 在 External Name 框中输入属性的名称。

(7) 在 Implementation 框中单击 Get/Set Methods。

(8) 在 Type 框中选择属性类型为 proper。

(9) 如果要设置只读属性，删除 ClassWizard 指定的 Set 函数名；如果要设置只写属性，删除 ClassWizard 指定的 Get 函数名。

(10) 单击 OK，关闭 Add Property 对话框。

(11) 再次单击 OK，确定选择并退出 ClassWizard。

当完成上面的步骤以后，ClassWizard 在调度映射表入口插入函数 SetNotSupported 或者函数 GetNotSupported，替代通常情况下的 Set 和 Get 函数。

如果想要改变现有的属性为只读或只写，可以编辑调度映射入口，从控件类中删除没有用的 Set 函数或 Get 函数。

如果需要设置属性为有条件的只读或只写(例如，仅当控件在某个特殊的模式时是可操作的)，可以提供正常的 Set 函数或 Get 函数，并且在适当的地方调用函数 SetNotSupported 或者函数 GetNotSupported。例如：

```
void CSampleCtrl::SetMyProperty(short propVal)
{
if (m_bReadOnlyMode)        // some control-specific state
SetNotSupported();
else
m_ipropVal = propVal;       // set property as normal
}
```

在这个例子中，当 m_bReadOnlyMode 为 TRUE 时，调用 SetNotSupported 函数，否则给这个属性设置一个新的值。

2. 返回属性的错误代码

为了指示当获取和设置属性值时发生了错误，可以使用函数 COleControl::ThrowError，它使用一个 SCODE 类型(状态码)的参数，可以使用预先定义的 SCODE 参数，或者自己定义一个参数。

Helper 函数中有大部分常用的预定义的 SCODE 值，如 COleControl::SetNotSupported，COleControl::GetNotSupported，以及 COleControl::SetNotPermitted 等。

16.6.5 访问环境属性

控件通过访问容器的环境属性可以得到关于容器的一些信息，这些属性显示了容器的一些可视的特征，如容器的背景颜色和使用的字体，以及操作特征，如容器当前是处于用户模式还是设计模式。

要访问一个环境属性，可以调用函数 COleControl::GetAmbientProperty，这个函数把环境属性的调度 ID 值作为第一个参数(文件 OLECTL.H 定义了标准的环境属性的调度 ID 值)。

函数 GetAmbientProperty 的参数包括一个调度 ID、一个表示所要的属性类型的标记符和一个指向返回值地址的指针。指针所指的数据类型随着标记符的改变而变化，如果容器支持该属性，函数返回 TRUE，否则返回 FALSE。

下面举一个例子，获取一个称为 UseMode 的环境属性，如果容器不支持该属性，假设缺省值为 TRUE。

```
BOOL bUserMode;
if (! GetAmbientProperty(DISPID_AMBIENT_USERMODE,VT_BOOL,&bUserMode))
bUserMode = TRUE;
```

COleControl 类还提供了帮助函数，可以访问许多常用的环境属性，并且当得不到这些属性时返回合适的缺省值，这些帮助函数是：

COleControl::AmbientBackColor	AmbientDisplayName
AmbientFont	AmbientForeColor
AmbietnLocaleID	AmbientScaleUnits
AmbientTextAling	AmbientUserMode
AmbientUIDead	AmbientShowHatching
AmbientShowGrabHandles	

如果一个环境属性的值改变(通过对容器的操作)，将调用控件的成员函数 OnAmbientPropertyChanged。该函数的参数是被影响的环境属性的调度 ID。这个调度 ID 的值也许为 DISPID_UNKNOWN，这表示一个或多个环境属性被改变，但无法得到被影响的属性的信息。

在前面我们介绍了 ActiveX 控件的属性、方法和事件，这里将介绍如何创建 ActiveX 控件。

16.7 通过 ActiveX ControlWizard 创建控件

当第一次创建一个 ActiveX(即以前的 OLE)控件应用程序时,可以利用 MFC ActiveX ControlWizard 作为向导,通过一些对话框,在这些对话框中选择程序的特性与功能。

在使用 ActiveX ControlWizard 时,可以回到前一步或进入后一步,更改已经选择的选项。对于 ActiveX ControlWizard 的每个选项都可获得帮助信息,在该控件上单击鼠标右键就可以获得该选项更多的信息。

下面,介绍创建一个 MFC 的 ActiveX 控件程序的步骤。

1) 在 File 菜单上,单击 New,再选择 Projects 制表页;

2) 指定工程名、路径、工作区名和操作平台选项,然后双击 MFC ActiveX ControlWizard 图标

3) 选择关于你的程序的适当选项,完成 MFC ActiveX ControlWizard 的步骤;

4) 准备使用你的 ActiveX 控件。

16.7.1 使用 ActiveX ControlWizard 的步骤

使用 MFC ActiveX ControlWizard 有两个步骤,能前进和后退每一个步骤,并且更改曾经选择的选项。

对于每一步的每个选项都可获得帮助信息,在该控件上单击鼠标右键就可以获得该选项更多的信息

1. 步骤一的选项

(1) 选择的工程中所有的控件的数目。ControlWizard 能为每个工程生成多达 99 个单独控件,每个控件有一个控件类和属性页类。

(2) 选择是否想使工程中的控件有运行许可口令。

(3) 选择是否让 MFC ActiveX ControlWizard 在源文件中产生注释,帮助阅读和编写程序。

(4) 选择是否让 MFC ActiveX ControlWizard 产生帮助文件,提供正文敏感帮助。该帮助支持需要使用帮助编译程序,这是 Visual C++ 提供的。

2. 步骤二的选项

(1) 如果要在工程中增加多个控件,或者要编辑使用的类和文件的名称,则从列表中选择想要编辑的控件,单击 Edit Name。对于想要编辑的每个控件重复这个过程。

(2) 选择希望每个控件都拥有的属性。

(3) 对于窗口类控件,选择使用哪个窗口类。

(4) 单击 Advanced 按钮,给控件增加非窗口活动性、鼠标指针通知、优化编码,或者异步调用属性

(5) 单击 Finish 完成。

16.7.2 在 ActiveX 控件文件中增加代码

在使用 MFC ActiveX ControlWizard 创建一个 ActiveX 控件后，得到一组启动程序。这组启动程序包括所有为建立控件所必要的文件，包括源文件(.CPP)和头文件(.H)、资源文件(.RC)、模板定义文件(.DEF)、工程文件(.DSP)、对象描述语言文件(.ODL)等，这些文件与 ClassWizard 创建的文件是可兼容的，可以利用 ClassWizard 定义控件的事件、属性和方法。

工程文件包含大量的已内置的功能，这些功能包括拖动控件、串行化数据、定义分配表、事件和消息映射等。

最后一步是提供指定的应用程序源代码，并且定义启动文件响应哪些消息和命令，连接控件和它的父程序。如果没有父程序，可以通过 AppWizard 创建 ActiveX 控件容器。

如果在创建工程时，选择了 comments 选项，启动文件中就会显示绿色字的注释，这些注释告诉你在何处应增加自己编写的代码，以及帮助指定的控件定义类的行为。

在源代码中，"//TODO"后面的代码是注释，告诉你在此处增加什么样的代码。

下面的例子告诉你在何处增加编码：

```
CTestControl::CTestControl()
{
InitializeIIDs(&IID_DTest, &IID_DTestEvents);
//TODO:在此处初试化控件的实例数据
}
CTestControl::~CtestControl()
{
//TODO:在此处释放控件的实例数据
}
```

16.8 ActiveX 控件的属性页

16.8.1 ActiveX 控件的属性页

属性页允许用户浏览和更改 ActiveX 控件的属性，用户使用控件的属性对话框访问这些属性，属性对话框中包含一个或多个属性页，提供给用户一个浏览和编辑控件属性的图形化界面。

ActiveX 控件的属性页通过两种方式显示：

1) 当引用了控件的属性动词(ELEIVERB_PROPERTIES)时，控件打开一个模式属性对话框窗口，包含控件的属性页。

2) 容器可以显示控件属性页的非模式对话框。

一个属性对话框包含显示当前属性页的页面、在不同属性页之间进行切换的标签，以及实现一些常用任务的按钮，如关闭属性页对话框、取消所做的更改、立即对 ActiveX 控件应用所做的更改。

如果用 ControlWizard 创建控件的工程文件，那么 ControlWizard 会提供一个缺省的属性页类，它是从 COlePropertyPage 类中派生出来的。一开始，这个属性页是空的，但是可以在属

性页上增加对话框或者控件的设置,因为 ControlWizard 仅建立一个缺省的属性页类,所以要使用 ControlWizard 创建其他的属性页类(也是从 COlePropertyPage 类派生)。

实现一个缺省属性页要分三个步骤:

1) 增加 COlePropertyPage 类的派生类到控件的工程文件中。如果是通过 ControlWizard 建立的工程,那么已经存在了一个缺省的属性页类。

2) 使用对话框编辑器增加一些控件到属性页模板中。

3) 定义控件的 DoDataExchange 函数,在属性页控件和 AvtiveX 控件之间交换数据。

下面举一个非常简单的例子,只有一个控件(命名为"Sample"),由 ClassWizard 创建,并且只有一个储备的 Caption 属性。

1) 在属性页上增加控件

(1) 打开控件的工程,在工程工作区窗口单击 ResourceView 制表页。

(2) 双击 Dialog directory 图符。

(3) 打开 IDD _ PROPPAGE _ SAMPLE 对话框。

在这个例子中,ClassWizard 在对话框 ID 的末尾添加了该工程的名称。

(4) 在控件工具窗口单击所要的控件,用鼠标将其拖到对话框页面中。

在这个例子中只需要一个静态文本控件"Caption"和一个编辑框控件,标识号为 IDC _ CAPTION。

(5) 单击工具栏中的 Save 按钮,保存你的工作。

现在,用户界面已经修改,你应该连接编辑框和 Caption 属性,下面的编辑 CSamplePropPage::DoData'ange 函数的工作将完成这个功能。

2) 自定义 DoDataExchange 函数

属性页的 DoDataExchange 函数允许连接属性页值和控件的实际属性值,要建立这个连接,必须映射属性页中的相应项到各自的控件属性上。

使用属性页类的 DDP _函数实现这个映射,DDP _函数类似于标准 MEC 对话框的 DDP _函数,除了引用了一个成员变量外,DDP _函数还引用了控件属性的名称。下面是属性页类的 DoDataExchange 函数的一个典型的映射入口。

```
DDP_ Text(pDX,IDC _ CAPTION,m _ caption,_ T("Caption"));
```

这样,函数在属性页的成员变量 m _ caption 和控件的 Caption 属性之间建立了连接关系。

DDP _函数还可用于其他的对话框控件类型,如检查框、单选按钮和列表框等,下面介绍所有属性页的 DDP _函数和它们的用处。

(1) DDP _ CBIndex:连接组合框中所选的字符串索引和一个控件的属性。

(2) DDP _ CBString:连接组合框中所选的字符串和一个控件的属性,所选的字符串只要前面几个字符与属性的值匹配就行了,不必完全匹配。

(3) DDP _ CBStringExact:连接组合框中所选的字符串和一个控件的属性,所选的字符串和属性的字符串值必须完全匹配。

(4) DDP _ Check:连接检查框和一个控件的属性。

(5) DDP _ LBIndex:连接列表框中所选的字符串索引和一个控件的属性。

(6) DDP _ LBString:连接列表框所选的字符串和一个控件的属性,所选的字符串只要前面几个字符与属性的值匹配即可,不必完全匹配。

(7) DDP _ LBStringExact:连接列表框中所选的字符串和一个控件的属性,所选的字符串

和属性的字符串值必须完全匹配。

(8) DDP_Radio:连接单选按钮和一个控件的属性。

(9) DDP_Text:连接文本框和一个控件的属性。

16.8.2 增加用户属性页

有的时候,ActiveX 控件可能会有很多属性,而不能合理地放在一个属性页中,在这种情况下,应该增加 ActiveX 的属性页来显示所有这些属性。

下面举例说明,这个例子是由 ControlWizard 创建的 ActiveX 控件,所以类名和标识号都是唯一的。

建议新的属性页按照 ActiveX 控件的标准属性页的尺寸创建,储备的图形和颜色属性页的尺寸是 250×62 对话框单位(Dialog Unit——DLU),标准字体属性页的尺寸是 250×110 DLU,ControlWizard 创建的缺省属性页使用 250×62DLU 的标准。

具体步骤是:

1) 打开控件的工程文件,单击工程工作区窗口的 ResourceView 制表页。

2) 选择 Insert 菜单中的 Resource 项。

3) 双击 dialog resource type,创建一个新的对话框资源。

4) 删除 OK 和 Cancel 按钮控件。

5) 在对话框模板上单击鼠标右键,打开 Dialog Properties 对话框窗口,输入资源 ID 号。

在这个例子中输入 IDD_PROPPAGE_NEWPAGE。

6) 单击 Styles 制表页,在 Style 框中选择 Child,在 Border 框中选择 None。

确保 Styles 制表页中,Titlebar 项没有选中,More Styles 制表页的 Visible 项没有选中。

7) 单击工具栏的 Save。

8) 双击对话框模板,打开 ClassWizard。

9) 在 adding a Class 对话框中,单击 OK。

10) 在 New Class 对话框的 Class Name 框中,输入新的对话框类的名称(在这个例子中,输入 CAddtlPropPage)。

11) 如果想改变文件名,单击 Change,输入你的实现文件和头文件的名称,否则的话,接受缺省的名称。

12) 在 Class Type 框中,选择 COlePropertyPage。

13) 在 Dialog ID 框中,选择 IDD_PROPPAGE_NEWPAGE。

14) 单击 OK,创建这个类。

15) 单击 OK,关闭 ClassWizard。

要允许控件的用户访问这个新的属性页的话,在控件实现文件的控件属性页宏中做如下更改:

```
BEGIN_PROPPAGEIDS(CSampleCtrl,2)
PROPPAGEID(CMyPropPage::guid)
PROPPAGEID(CAddtlPropPage::guid)
...
END_PROPPAGEIDS(CSampleCtrl)
```

注意：必须将宏 BEGIN_PROPPAGEIDS(属性页的数目)的第二个参数由 1 改为 2。此外，还必须在控件的实现文件中包括属性页类的头文件。

下一步是创建两个新的字符串资源，为这个新的属性页提供一个类型名和一个标题。

1）打开你的控件的工程文件，在工程工作区窗口单击 ResourceView。

2）双击 String Table 文件夹，然后双击你要增加字符串的字符串表资源。

此时，在窗口中打开一个字符串表。

3）在字符串表最后的控柑中输入标题的字符串，例如“Additional Property Page”。

这将打开一个 String 属性页，显示 Captin 框和 ID 框，在 Caption 框中是刚才输入的字符串。

4）在 ID 框中选择或输入一个 ID 字符串，完成后按 Enter 键。

在这里，我们使用 IDS_SAMPLE_ADDPAGE 作为新的属性页的类型名。

5）重复第三步和第四步，在 ID 框输入 IDS_SAMPLE_ADDPPG_CAPTION，在 Caption 框输入“Additional Property Page”。

6）在新的属性页类(这个例子中为 CAddtlPropPage 类)的.CPP 文件中更改函数 CAddtlPropPage::CAddtlPropPageFactory::UpdateRegistry，使得 IDS_SAMPLE_ADDPAGE 通过 AfxOleRegisterPropertyPageClass 函数传递。代码如下所示：

```
BOOL CAddtlPropPage::CAddtlPropPageFactory::UpdateRegistry(BOOL bRegister)
{
if bRegister)
return AfxOleRegisterPropertyPageClass(AfxGetInstanceHandle(),
m_clsid,IDS_SAMPLE_ADDPAGE);
else
return AfxOleUnregisterClass(m_clsid,NULL);
}
```

更改构造函数 CAddtlPropPage，使 IDS_SAMPLE_ADDPPG_CAPTION 传递给 COlePropertyPage 类的构造函数，如下所示：

```
CAddtlPropPage::CAddtlPropPage():
//* * * * * Add your code below this line * * * * * * //
COlePropertyPage(IDD,IDS_SAMPLE_ADDPPG_CAPTION)
//* * * * * * Add your code above this line * * * * * * //
{
//{{AFX_DATA_INIT(CAddtlPropPage)
//NOTE:ClassWizard will add member initialization here
// DO NOT EDIT what you see in these blocks of gererated code!
// }}AFX_DATA_INIT
```

完成这些更改后，重新编译工程文件，使用测试容器测试这个新的属性页。

16.8.3 使用储备属性页

MFC 提供三个储备属性页用于 ActiveX 控件：CLSID_CColorPropPage，CLSID_CFont-

PropPage 和 CLSID _ CPicturePropPage。这些属性页分别显示用户界面的储备颜色、字体和图形。

要合并这些属性页到一个控件中，必须在初始化控件属性页 ID 的数组代码中增加它们的 ID 号，如下所示：

```
BEGIN_PROPPAGEIDS(CSampleCtrl,4)
PROPPAGEID(CMyPropPage::guid)
PROPPAGEID(CLSID_CFontPropPage)
PROPPAGEID(CLSID_CColorPropPage)
PROPPAGEID(CLSID_CPicturepropPage)
END_PROPPAGEIDS(CSampleCtrl)
```

我们注意到宏 BEGIN _ PROPPAGEIDS 中的属性页数目为 4，这表示该控件支持的属性页的数目。

作了这些更改以后，重新编译你的工程，这时控件就具有字体、图形和颜色的属性页了。

16.9 增加正文敏感帮助

通过修改 .ODL 文件，可以使 ActiveX 控件对于当前选择的控件、属性或事件提供正文敏感帮助，在这一部分中，将介绍一个修改帮助文件 GENERIC 的例子。

在 .ODL 文件的库声明的 helpfile 属性中提供了控件的帮助文件，下面代码中，GENERIC.HLP 文件就是控件的帮助文件：

```
[uuid(5CE3E2E3-8D26-101B-B883-444553540000), version(1.0),
helpstring("GENERIC ActiveX Control module"),
helpfile("GENERIC.HLP")]
library GENERICLib
```

与之相关的 GENERIC.HPJ 文件列出了每一个帮助主题的正文敏感帮助标识符，如下所示：

```
[MAP]
GENERIC          10
GenericShape     50
FlashColor       60
ClickIn          70
ClickOut         80
```

要指定一个控件的帮助的主题 ID，可以在 GENERIC.ODL 文件的 coclass 声明中增加正文敏感帮助标识符 10，如下所示：

```
[uuid(5CE3E2E0-8D26-101B-B883-444553540000),
helpstring("GENERIC Control"),helpcontext(10)]
coclass CGENERIC
```

要指定一个控件属性的帮助主题 ID，可以在属性列表的属性声明中增加正文敏感帮助标

识符,下面的代码是增加 GenericShape 属性和 FlashColor 属性的正文敏感帮助标识符 50 和 60:

```
[uuid(5CE3E2E1-8D26-101B-B883-444553540000),
helpstring("Dispatch interface for GENERIC Control")]
dispinterface _DGENERIC
{
properties:
//NOTE-ClassWizard will maintain property information here.
// Use extreme caution when editing this section.
//{{AFX_ODL_PROP(CGENERICCtrl)
[id(3),helpcontext(50)]GenericShapeConstants CenericShape;
[id(1),helpcontext(60)]OLE_COLOR FlashColor;
[id(4)]BSTR FontName;
[id(5)]boolean FontBold;
[id(6)]boolean FontItalic;
[id(7)]boolean FontStrikethru;
[id(8)]boolean FontUnderline;
[id(9)]float FontSize;
[id(DISPID_BACKCOLOR),bindable,requestedit]OLE_COLOR BackColor;
[id(DISPID_CAPTION),bindable,requestedit]BSTR Caption;
[id(DISPID_FONT),bindable]IFontDisp * Font;
[id(DISPID_BORDERSTYLE),bindable,requestedit]
        BorderStyleConstants BorderStyle;
[id(DISPID_ENABLED),bindable,requestedit]boolean enabled;
[id(2)]MousePointerConstants MousePointer;
[id(10)]IPictureDisp * MouseIcon;
//}}AFX_ODL_PROP
};
```

要指定控件的事件的帮助主题 ID,可以在事件列表的事件声明中增加正文敏感帮助标识符,下面的代码就是给 ClickIn 事件和 ClickOut 事件增加正文敏感帮助标识符 70 和 80:

```
[uuid(5CE3E2E2-8D26-101B-B883-444553540000),
helpstring("Event interface for GENERIC Control")]
dispinterface _DGENERICEvents
{
properties:
// Event interface has no properties
};
properties:
// Event interface has no properties

methods:
// NOTE-ClassWizard will maintain event information here.
```

```
// Use extreme caution when editing this section.
//{{AFX_ODL_EVENT(CGENERICCtrl)
[id(1),helpcontext(70)]void ClickIn(float * X,float * Y,BSTR * Caption);
[id(2),helpcontext(80)]void ClickOut();
//}}AFX_ODL_EVENT
};
```

16.10 ActiveX 控件中的字体

如果要在 ActiveX 控件中显示文本，可以允许用户通过改变字体属性来改变文本的显示。字体属性通过字体对象实现，具有两个类型：储备属性和自定义属性。字体储备属性在使用 ClassWizard 实现时已经预先定义了，而字体自定义属性没有预定义，需要控件的开发人员定义属性的行为和用法。

16.10.1 字体储备属性

字体储备属性是由 COleControl 类预定义了，此外，还可以得到一个字体属性页，允许用户在属性页中改变字体对象的各个属性页，如字体名、字体大小以及字体风格。

访问字体对象可通过 COleControl 类的成员函数 GetFont，SetFont 和 InternalGetFont 来实现。用户可以象使用其他的属性的 Get/Set 函数一样，使用 GetFont 和 SetFont 函数访问字体对象。当在一个对象内部要求访问字体属性时，使用 InternalGetFont 函数。

使用 ClassWizard 的 Automation 制表页增加字体储备属性是非常简单的，你只要选择 Font 属性，ClassWizard 自动在调度映射表中加入储备字体入口。

1. 增加字体储备属性

增加字体储备属性的具体步骤是：

1) 打开控件的工程文件，选择 View 菜单的 ClassWizard 项，打开 ClassWizard；

2) 单击 Automation 制表页；

3) 在 Class Name 框，选择控件类名；

4) 单击 Add Property；

5) 在 External name 框，单击 Font；

6) 单击 OK；

7) 单击 OK，确定选择并关闭 ClassWizard。

ClassWizard 在控件类的实现文件中增加下面这行语句到控件的调度映射表中：

```
DISP_STOCKPROP_FONT()
```

此外，ClassWizard 在控件的.ODL 文件中增加下面这行语句：

```
[id(DISPID_FONT),bindable]IFontDisp * Font;
```

2. 更改 OnDraw 函数

在缺省情况下，OnDraw 函数对所有控件中的文本使用 Windows 系统字体，也就是说，要改变函数 OnDraw 代码，必须选择一个字体对象到设备上下文中。要做到这一点，调用函数

COleControl::SelectStockFont，并传给控件的设备上下文，如下所示：

```
void CSampleCtrl::OnDraw(CDC * pdc,const CRect& rcBounds,const CRect& rcInvalid)

CFont * pOldFont;
TEXTMETRIC tm;
const CString& strCaption = InternalGetText();

pOldFont = SelectStockFont( pdc );
pdc->FillRect(rcBounds,CBrush::FromHandle(
(HBRUSH)GetStockObject(WHITE_BRUSH)));
pdc->Ellipse(rcBounds);
pdc->GetTextMetrics(&tm);
pdc->SetTextAlign(TA_CENTER/TA_TOP);
pdc->ExtTextOut(rcBounds.left + rcBounds.right)/2,
        (rcBounds.top + rcBounds.bottom-tm.tmHeight)/2,
ETO_CLIPPED,rcBounds,strCaption,strCaption.GetLength(),NULL);
pdc->SelectObject(pOldFont);
```

当 OnDraw 函数使用字体对象改变以后，控件中的文本的显示特性将按照控件的字体储备属性设置。

16.10.2 字体自定义属性

除了字体储备属性外，ActiveX 空还有字体自定义属性，要增加一个字体自定义属性，需要：

- 用 ClassWizard 实现一个字体自定义属性；
- 处理标准字体改变通知；
- 实现一个新的字体通知接口。

1. 实现一个字体自定义属性

要实现一个字体的自定义属性，可以使用 ClassWizard 增加该属性，然后修改一些代码，下面的步骤说明如何增加一个自定义的 HeadingFont 属性到一个控件中，该控件命名为 Sample

1）打开控件的工程文件，单击 View 菜单的 ClassWizard 项，就打开了 ClassWizard；

2）单击 Automation 制表页；

3）单击 Add Property；

4）在 External name 框，输入属性的名称：Headingont；

5）在 Implementation 框，单击 Get/Set Methods；

6）在 Return Type 框，选择 LPFONTDISP 为属性的类型；

7）单击 OK；

8）单击 OK，确定选择并关闭 ClassWizard。

ClassWizard 会创建代码增加 HeadingFont 自定义属性到 CSampleCtrl 类和 SAMPLE.ODL 文件中，因为 HeadingFont 属性是 Get/Set 属性类型，所以 ClassWizard 会修改 CSampleC-

trl 类的调度映射表，增加一个 DISP_PROPERTY_EX 宏。

```
BEGIN_DISPATCH_MAP(CSampleCtrl,COleControl)
//{{AFX_DISPATCH_MAP(CSampleCtrl)
DISP  PROPERTY_EX(CSampleCtrl,"HeadingFont",GetHeadingFont,
      SetHeadingFont,VT_DISPATCH)
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

DISP_PROPERTY_EX 宏连接 HeadingFont 属性名和 CSampleCtrl 类的 Get/Set 方法：GetHeadignFont 和 SetHeadingFont，同时指定了该属性值的类型为 VT_DISPATCH。

ClassWizard 还在控件的头文件(.H 文件)中声明函数 GetHeadingFont 和 SetHeadingFont，并在控件的实现文件(.CPP 文件)中增加了函数模板。

```
LPDISPATCH CSampleCtrl::GetHeadingFont()
{
//TODO:Add your property handler here
return NULL;
}

void CSampleCtrl::SetHeadingFont(LPDISPATCH newValue)
{
//TODO:Ddd your property handler here
SetModidfiedFlag()
}
```

最后，ClassWizard 修改控件的.ODL 文件，增加 HeadingFont 属性的入口：

```
[id(1)]IDispatch * HeadingFont;
```

现在，已经给控件增加了 HeadingFont 属性，下面需要更改控件的头文件和实现文件，使之完全支持新的属性。

在控件的头文件中，增加下面这几行保护成员变量的声明：

```
protected:
CFontHolder m_fontHeading;
```

在控件的实现文件中，要做下面一些工作。

在控件的构造函数中初始化 m_fontHeading，代码如下所示：

```
CSampleCtrl::CSampleCtrl():m_fontHeading(&m_xFontNotification)
{
//[...body of constructor...]
}
```

声明静态的 FONTDESC 结构，包含缺省的字体属性，代码如下所示：

```
static const FONTDESC fontdescHeading=
{sizeof(FONTDESC),OLESTR("MS Sans Serif"),FONTSIZE(12), FW_BOLD, ANSI_CHARSET,
FALSE, FALSE, FALSE};
```

在控件的 DoPropExchange 成员函数中，增加对 PX _ Font 函数的调用，这是初始化字体自定义属性：

```
void CSampleCtrl::DoPropExchange(CPropExchange * pPX)
{
COleControl::DoPropExchange(pPX)
//[...other PX_ function calls..]
PX_ Font(pPX, _ T("HeadingFont"), m_ fontHeading, & _ fontdescHeading);
}
```

完成控件的 GetHeadingFont 成员函数，代码如下所示：

```
LPFONTDISP CSampleCtrl::GetHeadingFont()
{
return m_ fontHeading.GetFontDispatch();
}
```

完成控件的 SetHeadingFont 成员函数，代码如下所示：

```
void CSampleControl::SetHeadingFont(LPFONTDISP newValue)
{
m_ fontHeading.InitializeFont(& _ fontdescHeading,newValue);
OnFontChanged();     //notify any changes
SetModifiedFlag();
}
```

修改控件的 OnDraw 成员函数，定义一个变量保存原先选择的字体，语句如下所示：

```
CFont * pOldHeadingFont;
```

在控件的 OnDraw 函数中，增加下面这行代码来选择自定义的字体到设备上下文，语句如下所示：

```
pOldHeadingFont = SelectFontObject(pdc,m _ fontHeading);
```

在 OnDraw 成员函数，选择原先的字体回到设备上下文，如下面这条语句：

```
pdc->SelectObject(pOldHeadingFont);
```

实现了字体自定义属性后，就需要实现一个标准的字体属性页，允许控件的用户改变当前属性，在 BEGIN _ PROPPAGEIDS 宏的后面插入下面这行代码，增加标准属性页的 ID 号：

```
PROPPAGEID (CLSID_ CFontPropPage)
```

还必须在 BEGIN _ PROPPAGEIDS 宏中增加计数参数，如下：

```
BEGIN_ PROPPAGEIDS(CSampleCtrl,2)
```

做完所有这些修改后，重新编译整个工程文件，使修改的代码有效。

2. 处理字体通知

在大多数情况下，控件需要知道字体对象的属性何时被改变，每个字体对象当被改变时，应该能调用 COleControl 类的 IFontNotification 接口成员函数，通知该控件。

如果控件使用字体储备属性，通知的处理通过 COleControl 类的 OnFontChanged 成员函数完成，当增加了一个字体自定义属性，也使用相同的处理方式。

3. 多个字体对象接口的实现

在图 16.5 中,我们可以看到,两个对象使用同一个 IFontNotification 接口,如果你想要区分哪一个字体对象发生了改变的话,就会出现一些问题。

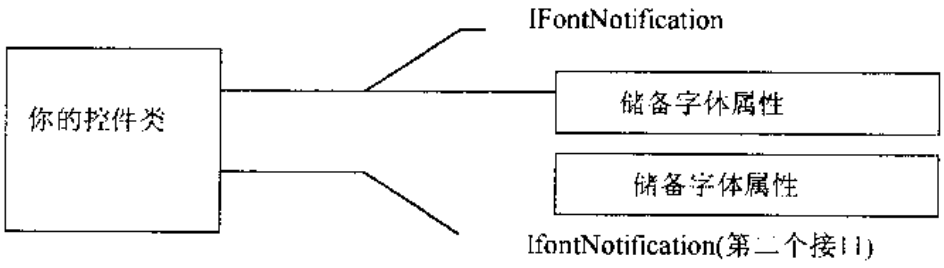


图 16.5 多个字体对象接口

要区分控件的字体对象的通知,就是为控件中的每一个字体对象创建一个独立的 IFontNotification 接口的实现。下面将介绍为第二个字体属性实现单独的通知接口的步骤,假设第二个字体属性为 HeadingFont 属性,并且已经增加到控件中。

在控件的头文件中的类声明中,增加下面这些代码:

```
protected:
BEGIN_INTERFACE_PART(HeadingFontNotify,IPropertyNotifySink)
        INIT_INTERFACE_PART(CSampleCtrl,HeadingFontNotify)
STDMETHOD(OnRequestEdit)(DISPID);
STDMETHOD(OnChanged)(DISPID);
END_INTERFACE_PART(HeadingFontNotify)
```

上面语句实现了一个称为 HeadingFontNotify 的 IPropertyNotifySink 接口,这个新的接口包含一个称为 OnChanged 的方法。

在控件的实现文件的构造函数中,改变通知 &m_xFont 为 &m_xHeadingFontNotify,然后增加下面代码:

```
STDMETHODIMP_(ULONG) CSampleCtrl:XHeadingFontNotify::AddRef()
{
METHOD_MANAGE_STATE(CSampleCtrl,HeadingFontNotify)
return 1;
}
STDMETHODIMP_(ULONG) CSampleCtrl::XHeadingFontNotify::Release()
{
METHOD_MANAGE_STATE(CSampleCtrl,HeadingFontNotify)
return 1;
}
STDMETHODIMP CSampleCtrl::XHeadingFontNotify::QueryInterface(
  REFIID iid, LPVOID FAR * ppvObj)
{
```

```
METHOD_MANAGE_STATE(CSampleCtrl,HeadingFontNotify)
if (IsEqualIID(iid,IID_IUnknown)||
IsEqualIID(iid,IID_IPropertyNotifySink))
{
*ppvObj=this;
AddRef();
return NOERROR;
}
return ResultFromScode(E_NOINTERFACE);
}

STDMETHODIMP CSampleCtrl::XHeadingFontNotify::OnChanged(DISPID)

METHOD_MANAGE_STATE(CSampleCtrl,HeadingFontNotify)
pThis->InvalidateControl();
return NOERROR;
}

STDMETHODIMP CSampleCtrl::XHeadingFontNotify::OnRequestedit(DISPID)
{
return NOERROR;
}
```

做完所有的修改以后，重新编译工程文件，并使用测试容器来测试该接口。

16.11 ActiveX 控件中的图片

16.11.1 自定义图片属性

图片类型是所有 ActiveX 控件的通用类型之一，图片类型处理图元文件、位图或者图符，允许用户指定一个图片显示在一个 ActiveX 控件上。自定义图片属性使用图片对象和 Get/Set 函数来实现，控件用户通过图片属性页访问自定义图片属性。

ActiveX 控件类提供若干组件实现控件的图片属性，这些组件包括：

1）CPictureHolder 类。

该类可方便地访问图片图片对象和自定义图片属性的显示项功能。

2）LPPICTUREDISP 类型的属性支持，使用 Get/Set 函数实现。

使用 ClassWizard 可快速增加支持图片类型的自定义属性。

3）管理控件属性和各属性项的属性页。

16.11.2 实现自定义的图片属性

当完成这一部分简要介绍的步骤后，用户就可以在控件上显示选择的图片，而且还可以使用属性页改变显示的图片，该属性页上显示当前的图片以及能让用户选择不同图片的 Browse

按钮。

要实现一个自定义图片属性，必须做下面的工作：

1）在控件工程文件中增加代码。

必须增加一个标准图片属性页 ID、一个 CPictureHolder 类型的数据成员以及自定义 LPPICTUREDISP 类型的属性。

2）在控件类中修改几个函数，这几个函数是与 ActiveX 控件的绘图相关的几个函数。

在控件实现文件中，宏 BEGIN _ PROPPAGEIDS 的后面增加下面这行语句，就可以增加标准图片属性页：

PROPPAGEID(CLSID _ CPicturePropPage)

还要增加 BEGIN _ PROPPAGEIDS 宏的引用计数，如下所示：

BEGIN _ PROPPAGEIDS(CSampleCtrl,2)

在控件头文件的控件声明中加入下面这行，增加 CPictureHolder 数据成员到控件类。

CPictureHolder m _ pic;

下面，增加一个支持图片类型的自定义属性，步骤如下：

1）打开控件的工程文件，单击 View 菜单的 ClassWizard 项，就打开了 ClassWizard；

2）单击 Automation 制表页；

3）单击 Add Property；

4）在 External name 框，输入属性的名称。在这个例子中是输入 ControlPicture；

5）在 Implementation 框，单击 Get/Set Methods；

6）在 Return Type 框，选择 LPFONTDISP 为属性的类型；

7）输入 Get/Set 函数的名称，或者接受缺省的名称（在这个例子中，缺省的函数名为 GetControlPicture 和 SetControlPicture）；

8）单击 OK，关闭 Add Property 对话框；

9）单击 OK，确定选择并关闭 ClassWizard。

ClassWizard 在控件的头文件中增加下面代码到调度映射表声明中：

```
afx _ msg LPPICTUREDISP GetControlPicture();
afx _ msg void SetControlPicture(LPPICTUREDISP new Value);
```

此外，插入下面代码到控件实现文件的调度映射表中：

```
DISP _ PROPERTY _ EX(CSampleCtrl,"ControlPicture",GetControlPicture,
SetControlPicture,VT _ PICTURE);
```

ClassWizard 还增加下面两个函数到控件实现文件中：

```
LPPICTUREDISP CSampleCtrl::GetControlPicture()

//TODO:Add your property handler here
return NULL;

void CSampleCtrl::SetControlPicture(LPPICTUREDISP newValue)

//TODO:Add your property handler here
SetModifiedFlag();
```

除了在控件的工程中增加一些内容外，还需要修改几个函数来改变对控件的描述，这几个函数是：OnResetState，OnDraw，以及自定义图片属性的 Get/Set 函数（在这个例子中，控件类为 CSampleCtrl，CPictureHolder 类的数据成员为 m _ pic，自定义图片属性名为 ControlPicture）。

在控件的 OnResetState 函数中，增加下面的语句：

```
m _ pic.CreateEmpty();
```

这是设置控件的图片为一个空白的图片。

要正确调用图片属性，需在控件的 OnDraw 函数中调用 CPictureHolder::Render 函数，如下所示：

```
void CSampleCtrl::OnDraw(
CDC * pdc,const CRect& rcBounds,const CRect& rcInvalid)
{
// * * * * * * 在下面增加你的代码 * * * * * * //
m _ pic.Render(pdc,rcBounds,rcBounds);
}
```

在控件的自定义图片属性的 Get 函数中，增加下面代码：

```
return m _ pic.GetPictureDispatch();
```

在控件的自定义图片属性的 Set 函数中，增加下面代码：

```
m _ pic.SetPictureDispatch(newValue);
    InvalidateControl();
```

增加下面代码到 COleControl 派生类的 DoPropExchange 函数中：

```
PX _ Picture(pPX,"ControlPicture",m _ pic);
```

完成修改的工作以后，重新编译工程文件，并使用测试容器测试新的属性。

16.12 小　　结

本章详细介绍了当今社会上经常用到的一项技术——ActiveX 控件，读者需细心阅读本章内容，相信一定会大有收获。

第 17 章　动态链接库(DLL)

对 Windows 应用程序而言,创建一个没有使用 DLL 的应用程序几乎是不可能的。你可能没有感觉到前面提供的例子中全部使用了 DLL,但的确所有的 Win32 API 函数和无数的 Win32 操作系统的函数都包含在 DLL 中。

一般情况下,DLL 就是各个库中的函数的集合。但是,它又不同于静态库 LIB 中的函数。因为它不能用连接器直接连入可执行文件,而是通过包含在可执行文件中的信息进行检索而后使用。库中的代码在程序运行时才加载。这样就允许几个不同的进程在内存中共享库,从而使得在运行几个共用许多同样的库的应用程序时对内存容量的需求大大减少,同时,使 EXE 文件易于管理。

然而,如果库只被单个应用程序使用,那还是用一个简单的静态链接库更有效。当然,如果函数只在一个程序中使用,那只需要将它直接连入文件。

在大多数情况下,在连接时,工程可以静态地,或隐含地与 DLL 连接。在运行时操作系统会替你管理加载 DLL 的过程,当然,也可以在运行时显式地或动态地加载 DLL,正如在后面将看到的。通过本章学习将使你对 DLL(动态链接库)有一个初步的了解并学会初步使用 DLL(动态链接库)。

17.1　动态库的基础知识

17.1.1　DLL 的概述

动态链接库是个包含函数的库文件,我们可以很容易地分配新的函数和资源。动态链接库和其他C++库不一样,它们是在运行时和应用程序连接,而不是地编译、连接过程中被应用。

动态链接库的英语拼写是:Dynamic Link Library,简称 DLL,与之相应的是静态链接库(Static Link Library),它们之间的区别在于:静态链接库是在应用程序的编译过程中与应用程序相连接的,而动态链接库是在应用程序的执行过程中与应用程序相连接的。

如果与应用程序连接的是静态链接库,每一个应用程序在编译过程中都必须拷贝一份库的代码,这样造成了资源的浪费,而且使应用程序本身的代码开销很大。而动态链接库在编译过程中并不连接应用程序,而是当应用程序运行时连接,可以和其他的应用程序共享库中的函数和资源,减少了因重复拷贝而造成的应用程序的冗长以及计算机资源的占用。当用户创建使用动态链接库的应用程序时,必须将动态链接库文件和应用程序的可执行文件一同分发。

动态链接库(DLL)是由函数、对象以及资源等组成的二进制的库文件,此库文件能同时被多个进程共享使用。用户可能在应用程序中基于如下要求创建和使用 DLL。它们是:

- 在不同的可执行文件之间共享公共的程序。
- 在设计应用程序时,将其拆分成各个相互独立的功能部件,为以后对这些功能部件各自升级提供较方便的途径。

- 使资源数据独立于可执行的应用程序之外，但又能较方便快速地访问它。此种应用的一个典型的例子是为一个应用程序设计各种语言的 DLL，每个 DLL 中都包含有指定语言的字符串表。

17.1.2 传统的 DLL

读者即使没有编写过自己的 Windows DLL，或许也使用过 DLL，这些 DLL 或者是随 Windows 一起提供的，或者是由其它软件销售商提供的。例如，文件对话框以及对拖放的支持就被包含在 Windows 的 COMMDLG.DLL 和 SHELL.DLG 中。使用 C 语言 APIDLL 并不是什么难事，只要保证 make 文件中包含了对 DLL 引入库的引用，并且保证在运行时可以在磁盘中找到相应的 DLL 文件即可。我们可以像调用自己工程中的函数一样来调用 DLL 中的函数。

传统的 C 语言 API.DLL 有一个非常重要的特性，它们可以被任何 C 编译器使用，也可以被其他语言和程序设计环境使用。例如，用 Borland C 编写的数据库 DLL，可以被 Microsoft.C 程序调用，也可以被 Microsoft Visual Basic、Delphi 程序调用。

我们可以将传统的 DLL 看成是独立的程序，它有它自己的事例句柄，自己的全局内存，自己的堆栈以及自己的资源。DLL 中并不包含自己的 Windows 消息循环，它是在程序启动时或当需要的时候被装入的，当最后一个客户应用被终止后，它才被从内存中卸掉。如果传统的 DLL 申请了内存，那么当 DLL 被终止时，所申请到的内存就会被释放，而并不会等到客户被终止时才释放。DLL API 常要求客户进行内存申请，并将申请到的缓冲区地址传递给 DLL，然后 DLL 就可在该缓冲区上进行操作，最后该内存的释放由客户来完成的。例如，对于 ODBC，客户需要为列数据申请一块缓冲区，而 DLL 负责往缓冲区中写数据。

关于 DLL 有若干个连接选择，对于需要经过编译的 C 程序来说，引入库是常见的一种连接方式。客户使用符号化的函数名来调用 DLL 中的函数，当 DLL 被装入时，Windows 会将函数调用同 DLL 函数地址相匹配。函数既可以通过符号化的名字来匹配，也可以通过有序数来匹配。在有序数方式下，DLL 中的每一个函数都被赋予了一个唯一的整数，这种方式的效率会很高。解释语言 Windows 环境常常需要以更加动态的方式来使用 DLL，它们可以在运行时刻选择和装入 DLL，也可直接调用 DLL 函数，而无需事先对地址进行分辨。

如果已有的函数原型没有发生变化，那么 DLL 可以被更新，客户应用也用不着被重新进行编译，但如果改变了 DLL 中的函数名，返回数据类型或者是参数类型，那么就应该重新编译客户应用。

Visual C++ 支持用 MFC 库中的类编写的传统的 DLL 创建，但要使用 C 语言外部接口。

17.1.3 MFC 库

MFC 库可以容纳整个C++类，我们可以像使用静态连入的类一样来使用 DLL 中所包含的类，我们可以创建 DLL 所包含类的对象，也可将它们作为基类。然而传统的 DLL 与 MFC 库 DLL 之间有一些主要的差别。

MFC 库 DLL 的使用权仅限于 Microsoft C++ 编译器。

与传统的 DLL 不同，MFC 库的 DLL 只能被使用于 Microsoft 类库应用，为什么会有这种限制呢？Microsoft C++ 编译器使用 5 种特殊的内部函数名，它包含了类名、函数名、返回数据类型、参数数据类型以及 public/protected/private 访问权限。例如，对于 Microsoft C++ 编译器，函数

CWnd * GetDescendantWindow(int nID) Const

被赋予的是分割名(mangled name)或者修饰名(decorated name)。

? GetDescendant. Window@CWnd@@RFCPEV1@H@Z

而其他编译器使用是与之不同的名字分割算法,因此,如果某个 Borland C++ 客户程序包含对 Get DescendantWindow 的函数调用,Borland 分割名将无法与 Microsoft 分割名相匹配,或许在将来会存在一个统一的名字分割算法。

17.2 MFC 库的动态链接库

使用 MFC 有两种形式。一种是普通的 MFC DLL,它可以使用 MFC 但不能在 DLL 和应用程序间传递 MFC 对象指针;另一种 MFC 支持的 DLL 叫做 MFC 扩展 DLL,这种 DLL 需要一些附加的操作,但它允许你在 DLL 的应用程序间自由地传递 MFC 对象指针。

17.2.1 MFC 库的动态链接库的命令

包括在 MFC 新版中的 DLL 和库遵循一条结构化命名约定。使用 DLL 或库的使用更方便。库的名字约定在"库版本"中描述。本文描述 MFC 3.0 版中要求建立和使用 DLL 的文件命名约定。

MFC AFXDLL 版本有不同形式。这些形式根据下表详述的 MFC[01D]30[O][D],DLL 约定命名。需要建立这些 DLL 的应用程序或扩展 DLL 的输入库与 DLL 有相同的基本名,但其扩展为.LIB。

表 17.1 AFXDLL 命名约定

DLL	描述
MFC 30. DL	无 Unicode 支持发行版本
MFC30D. DLL	无 Unicode 支持调试版本
MFC030. DLL	OLE 支持的发行版本
MFCD30D. DLL	DLE 支持的调试版本
MFCD30. DLL	数据库支持的发行版本
MFC30V. DLL	数据库支持的调试版本
MFC30U. DLL	有 Unicode 支持发行版本
MFC30UD. DLL	有 Unicode 支持调试版本
MFC030UD. DLL	有 Unicode 支持发行 OLE 的版本
MFC030UD. DLL	有 Unicode 支持调试 OLE 的版本

如果动态地链间 MFC AFXDLL 版,无论它来自应用程序还是扩展 DLL,都必须在产品中包括 MFC 30. DLL。如果应用程序有 OLE 或数据库支持,还必须分别包括 MFC030DLL 或 MFCD30、DLL。如果在应用程序中要求 Unicode 支持,那么在它们的名字中还要包括"U"的 DLL 版。

注意:对数据库类来说 Unicode 不支持。

如果静态地把 DLL 链向 MFC,那么必须用 MFC USRDLL 版之一与之相链。这些版本根据表 17.2 中所列[N|U]AFXDW[D].LIB 约定命令。

表 17.2 USRDLL 命名约定

DLL	描述
NAFXDW.LIB	发行 VSRDLL 版本
NAFXDWD.LIB	调试 VSRDLL 版本
UAFXDW.LIB	需 Unicode 支持发行 USRDLL 版本
UAFXDW.LIB	需 Unicode 支持调试 USRDLL 版本

17.2.2 MFC 常规型 DLL

在 MFC4.0 以下版,USRDLL 这一术语的含义是 DLL 内部使用 MFC,但引出函数采用链接方式,USRDLLS 可以用于 MFC 和非 MFC 的应用程序中。一个值得高兴的事是 Microsoft 从 MFC 4.0 版开始,将 USRDLL DLL 命名为常规 DLL,因而 USRDLL 这一术语现在已经过时了,一个常规 DLL 可以同 MFC 以静态或动态方式相链接,同时与 USRDLL 一样,它也适用于 MFC 开发环境下采用支持 DLL 调用的程序设计语言开发应用程序中。

常规 DLL 使用 CWinApp 类封装 Win32 API 函数 DLLMain()并为 DLL 设置 MFC 外壳程序,经这样做后,对调用程序来说是透明的,因而使得常规 DLL 在外表上看起来像采用 C 语言编写的 SDK 式的 DLL。为达到这种效果,所有的引出函数必须像如下方式所示那样在接口和实现时采用表达式 extern"C"声明。

```
extern"C"DLL Export FLOAT Multiply2Floats(FLOAT f1,f2){return f1 * f2;
}
```

注意:所有要从 DLL 中实现引入的应用程序函数也必须象在 DLL 中引出函数那样采用相同的链接说明。

例如 DLL 采用 extern"C"说明一个引出函数,调用者也必须用 extern"C"对它加以引入说明。

在常规型 DLL 中使用资源

也能与在应用程序中使用资源相同的方式在常规 DLL 中使用资源,在常规 DLL 中存放的资源能被各个应用程序访问。如果在 DLL 中使用资源(如图标等),则必须首先采用 AFX-AFX _ MANAGE _ STATE(AfxGetStaticModuleSTATE1))

17.2.3 MFC 扩展型 DLL

用 MFC 可以创建一种表面上看不像一组函数而像应用程序的 DLL,这种类型的 DLL 可被用来建立 MFC 派生类。

为了使 DLL 可以自由地在应用程序和 DLL 之间传递指向 MFC 对象的指针,需要创建扩展型 DLL。这种 DLL 必须与 MFC 动态链接库连接。以前,这种 DLL 叫做 AFXDLL。

创建 MFC 扩展型 DLL 的最简单的方法是从 MFC AppWizard(.dll)着手,第一步选择

MFC扩展型DLL选项,它将以正确的设置来创建一个MFC扩展DLL项目,由它产生推荐台包含了DllMain()函数,它完成了一些基本的初始化工作。还应注意到这种类型的DLL不应包含CWinApp派生类的声明。

1. 初始化扩展DLL

MFC扩展DLL需要一些特殊的初始化来把它装入到MFC框架之中。为说明这些,让我们看一个由ApWizard生成的DllMain()函数。

```
Static AFX_EXTENSION_MODULE MyExtDLL={NULL,NULL};
extern"C" int APIENTRY
    DllMain(HINSTANCE,hinstance,DWORD dwReason,Lpvoid LPReserved)
{if (dwReason==DLL_PROCESS_ATTACH)
{TRACEO("MYEXT DLL inctializing! In");
// ExtemSion DLL one-time initializationAfXinitExtensionModule(MyExtDLL,hinstance);
//insert this DLL into the resource Chain
new CDyLinkLibrary (MyExtDLL);
}
else if (dwReason==DLL_PROCESS_PETACH)
TRACE("MYEXT.DLL Terminating! \n");

return 1;//OK
}
```

这个函数最重要的部分是调用AfxInitExtensionModule();它初始化在DLL使它在MFC框架中正确地工作。它需要传递给MFCMain()的DLL实例的句柄和AFX-EXTENSION-MODULE结构,但必须保证声明了一个,因为CDynLinkLibrary的构造函数将做初始化工作。必须在你的DLL中创建一个CDynLinkLibrary,它的构造函数初始化AFXEXTENSIONMODULE结构,而且它还把DLL加到MFC能处理的DLL表中。

2. 动态加载扩展DLL

从MFC 4.0开始,框架就支持动态加载和卸除MFC DLL和你自己的扩展应用程序。为使DLL正常工作,当进程分离时,需要在DllMain()函数中调用AfxTeminate,它可以通过向DllMain()加入前面所提到的代码来实现。

```
if(dwReason==DLL_PROCESS_DETACH)
{AfxTermExtensionModule(MyExtDll);
}
```

另外,还要记住的是现在的DLL是扩展型DLL,应该用AfxLoadLibrary(),AfxFreeLibrary()动态装载。而不是一般的LoadLibrary()和FreeLibrary()。

3. 输出扩展型DLL

在使用扩展型DLL中的函数或类之前,需要声明它们,虽然需要手工在.def文件中加入许多相同的名字。可以用MFC用于从扩展DLL中输出的语句,这些语句包括AFX_EXT_

CLASS 和 AFX _ EXT _ API,参见如下语句:

```
Class AFX _ EXT _ CLASS CMyClass:public CObject
{ Your Class declaration
}
Void AFX _ EXT _ API MyFun(C);
```

17.3 创建动态链接库

17.3.1 DLL 的入口函数 DLLMain()

C/C++ 运行时库代码启动一个 DLL,每次程序挂接到或脱离 DLL 时,此代码都要使用 CK++语言中的库函数 DLLMain(),它是 DLL 程序的入口函数。另外,32 位 Windows 程序可以是多线程的(Multithreaded),每当一个线程挂接到或脱离 DLL 时,都要调用 DLLMain()函数。在如下四种情况下,需要调用 DLLMain()函数。

- 进程挂接:进程将挂接到 DLL 时。
- 进程脱离:进程将从 DLL 中脱离开时。
- 线程挂接:线程将挂接到 DLL 时。
- 线程脱离:线程将从 DLL 中脱离开时。

注意:如果应用程序的 DLL 不需要分配额外附加的内存或其他资源时,完全可以从 DLL 源程序中除掉 DLLMain()函数,编译器将会采用一个默认的 DLLMain()函数而自动链接到 DLL 程序中。

正如如下所示的函数原型那样,DLLMain()函数用 APIENTRY 调用约定,并带有三个形参:

BOOL APIENTRY DLLMain (HANDLE hinst, UINT Wreasonforcall, LPVOID lpReseWed)

参数 hinst 是应用程序或 thread 的实例句柄;Wreasonforcall 指明要调用 DLLMain()函数的原因,即应用程序或 thread 当前是进行初始化还是结束工作,lpReserved 保留给 Windows 系统使用。表 17.3 列出 DLLMain()函数中参数 Wreasonforcall 的四种可能值及含义。

表 17.3 DLLMain()函数中参数 Wreasonforcall 四种可能值及含义

值	描述
DLL _ PROCESS _ ATTACH	一个新的进程试图访问 DLL
DLL _ THREAD _ ATTACH	当前进程的一个新的线程试图访问 DLL
DLL _ PROCESS _ DETACH	进程的最终线程在脱离 DLL
DLL _ THREAD _ DETACH	进程的线程之一脱离 DLL

下列表清单显示 DLLMain()函数的实现框架。

列表清单:DLLMain()函数实现框架。

```
BOOL APIENTRY DLLMain(HANDLE hModule,DWORD Wreasonforcall
(PVOID lpReserved)
{Switch (Wreasonforcall)
{Case DLL _ PROCESS _ ATTACH;
//…
Case DLL _ THREAD _ ATTACH;
//…
Case DLL _ THREAD _ DETACH;
//…
Case DLL _ PROCESS _ DETACH;
//…
}
return TRUE;}
```

17.3.2 创建 DLL

建立 DLL 项目的最简单方式是用 AppWizard 来创建一个新的项目，对简单的 DLL 你可用 DLL 项目类型。AppWizard 将创建一个包含基本要素的 DLL 框架，下面需要手工加入代码来满足自己的需求。

如果打算编写一个高水平的 MFC 函数，像文档和视或 OLE 自动服务器，MFC 的 AppWizard 将帮助你完成一些附加的工作。这些工作包括加入对 MFC 库的引用，以及加入了由 CWinApp 派生的对象的声明文件。

提示：比较方便的做法是先创建一个顶层的 DLL 项目作为测试项目，然后创建此 DLL 项目作为一个子项目，可以创建一个测试程序，如果需要的话，DLL 会自动创建。DLLMain()

绝大多数 DLL 只是一些被外部程序调用的函数集合。这些函数各式各样，但每个 DLL 都包含一个 DLLMain()函数，它的作用是初始化 DLL 和卸载时清理 DLL。早期版本的对应函数是 libMain()和 WEP()。下面的例子说明 DllMain()的框架。

```
BOOL WINAPI DllMain(HANDLE hInst,
Dword DwReason,lpvoic lpVOID lpReserved){switch(dwReason)
{case DLL _ PROCESS _ ATTACH;
// DO per-process initialization
break;
Case DLL _ THREAD _ ATTACH;
//DO per-thread initialization
break;
Case DLL _ THREAD _ DETACH;
// Clean up any per-thread structures
break;
case DLL _ PROCESS _ DETACH;
// Clean up any per-process structures
break;}// end switch
if (bAllWentWell)
```

```
return TRUE;
else return FALSE;
} //end DLLMain()
```

DllMain()函数可能被调用多次,参数 dwReason 说明 DllMain()的调用原因,它包含以下几种值:

当进程第一个加载 DLL 并调用 DLLMain()时,DwReason 等于 DLL _ PROCESS _ ATTACH。每当进程创建一个新线程时,DwReason 等于 DLL _ THREAD _ ATTACH。

当进程结束,DwReason 等于 DLL _ PROCESS _ DETACH。当线程结束,DwReason 等于 DLL _ THREAD _ DETACH。

根据 DwReason 的值,应该作一些线程或进程开始前的初始化工作。一般地,进程的初始化工作是建立多线程共享的资源,例如共享文件或初始化库。线程的初始化要完成独立于线程的工作,像初始化线程的局部存储区。

DLL 中包含了独立与调用程序的资源,如果 DLL 中的函数将用到这些资源,那得把 hInst 句柄保存到一个安全地方,这个句柄将在加载资源时用到。

lpReserved 是保留参数,不需要关心它,但可以检测这个指针值,如果 DLL 被动态加载,它等于 NULL,如果是静态加载,它等于非 NULL。

如果 DLLMain()的操作一切正常,它将返回 TRUE;否则返回 FALSE。

注释:假如源程序中没有 DLLMain(),编译器会自动增加一个 DLLMain(),但它只简单地返回 TRUE。

1. 从 DLL 中输出函数

为使用 DLL 库中的函数,每个函数在 DLL 的索引表中必须有入口地址。为让编译器增加到输出表,有两种选择。

在每个函数声明的开始处增加一 deslspee(dllexport);或者使用 MFC 提供的与-declspee(dllexport)相关联的宏,这些宏有 AFX _ CLASS _ EXPORT,AFX _ DATA _ EXPORT,AFX _ API _ EXPORT。目前版本的 Visual C++ 中,它们都一样。它们有可能用来支持未来的某些增强功能。

-declspec 方法不如第二种方法用得多,第二种方法涉及模块定义文件 1.def),这样可以操作更多的控制。

(1)模块定义文件

Visual C++ 中的.def 文件相当明了,特别是许多在以前版本中使用的复杂操作的 Win32 不再使用。正如你在下面例子中看到的,.def 文件给每一个库以一个名字和一个描述,然后是函数列表。

```
LIBRARY "MYDLL"
DESCRRIPTION MYDLL Example    namic Link Library
EXPORTS
MyInitialize
MyCreate
MyConnect @3NONAME
```

```
MySend @4
MyReceive @5
MyDisconnect
```

@号的含义是给函数指定一个数字序号,这个数字可以被 GetProcAddress()使用。实际上,编译器将给每一个函数指定一个数字,但是这样每个函数对应的数字不明确,不如自己指定数字。

另外,你可能已注意到例子中的 NONAME,它告诉编译器不要在外部函数表中包含这个函数。有时,这可以节省许多 DLL 空间。这种方法对用隐式方法与 DLL 连接的应用程序没有影响,因为隐式连接只使用内部序号。但是,动态加载 DLL 的程序需要传递一般数字给 GetPorcAddress,而不是传递函数名。

(2)输出类

在 DLL 中定义输出点可能有点小窍门。可以显式的定义每个输出函数,包括还没有定义的。

如果看到由类的实现部分的代码所生成的映象文件,会惊奇地发现许多函数,它们包括隐式的构造函数,析构函数、MFC 在 DECLARE-MESSAGE-MAP 宏中声明的函数及自己定义的函数。

虽然可以自己输出这些函数,但有更简单的方法。如果在你的类声明中使用 AFX_CLASS_EXPORT 宏,编译器就更仔细地处理每个输出函数来允许应用程序使用包含在 DLL 中的类。

2. DLL 的建立

如果用 AppWizard 创建一个项目,它已正确地更新.def 文件,所以不必担心。但是如果正在创建自己的项目文件,或者其他不用 AppWizard 的项目,就应该自己给连接器指定 DLL 开关。这样连接器将产生 DLL,而不是普通的可执行文件。

注释:如果.def 文件中有 LIBRARY 行,就不需要指明连接开关 DLL。

17.4 加载 DLL

当应用程序开始运行时,它就开始寻找与它连接的 DLL 库,并把它映射到进程的内存空间。为找到 DLL,操作系统需要查找如下设置:

- EXE 文件运行的目录
- 进程的当前目录
- Windows 的系统目录
- Windows 目录
- PATH 环境变量所指目录

注释:对 WindowsNT 来说,带 3 的目录表示 32 位系统,如 SYSTEM32,它们将首先被查找,然后,再在不带 3 的目录中寻找。

如果 DLL 没被找到,应用程序就会显示一个对话框说明没有找到 DLL,并显示寻找的路径,最后进程结束。

如果找到 DLL,它就被映射到进程的内存空间,直到进程结束,应用程序现在可以任意调用 DLL 中包含的函数,而不必做更多的事情。如果你想动态加载或卸出 DLL,下面将介绍它的实现方法。

注释:和 Windows 3.1 不同,全局内存并不是真正的全局,每个进程都有自己独立的拷贝。

1.动态加载 DLL

有时,应用程序自己加载 DLL,可能更好些,例如,希望指定用户可以使用哪一个 DLL,或通过用户的选择来使用不同的 DLL,动态加载过程允许使用哪一个 DLL 来达到不同的目的。比如,开发了一个通讯模块,它可以在程序运行时让用户选择是 TCP/IP 还是 NetBIOS 网,程序由此决定加载哪个 DLL。

(1)::LoadLibrary()

动态加载 DLL 的第一件事就是把 DLL 模块映射到进程的内存空间。这件事是::LoadLibrary()完成,它需要一个参数,即模块名。代码如下所示:

```
hMODULE hMyDLL
hMyDH = LoadLibrary("Mylib");
if (hMyDll = NULL)
//Could not Load Dll handle the error…
```

如果不加指定,Windows 系统假定缺省的扩展名为".dll",在例子中,系统将寻找"Mylib Dll"文件。除非指定路径,否则,Windows 像寻找隐式 DLL 那样寻找用户 DLL(如前所述,从进程加载文件的地方开始,沿 PATH 指定的路径去寻找)。

一旦找到文件,Windows 将用已在内存的 DLL 全路径名与找到文件的全路径名比较,如果匹配,返回库句柄;而不是再加载库的拷贝。

注释: LoadLibrary()也可以加载有关可执行文件到内存。可执行文件的句柄在调用 FindResoure()和 LoadResource()时用到。

如果找到 DLL 并加载成功,LoadLibrary()将返回一个句柄。记住要保存这个句柄,你很快看到这个句柄的用途。如果失败,LoadLibrary()返回 NULL。

注释:如果没有找到 DLL,系统一般显式有关对话框,说明文件未找到。然而,如果用的是 MFC,它将调用::SetErrorMode(),让系统简单地把错误返回给应用程序,而不是在 LoadLibrary()函数返回之前告诉用户。这就允许自己处理错误,如果没有用 MFC,也可以自己调用::SetErrorMode)。

(2)::GetProcAddress()

只要 DLL 被成功加载,下一步的任务就是找到所使用的函数的地址。::GetProcAddress()可以做这件事情。它需要一个由 LoadLibrary()返回的句柄和函数名,这个名字应该是 DLL 中的输出函数名。举例子说明如下:

```
HMODULE hMyDll;
UINT(pfnMyFunc)(char * strMyName);
hMyDll = ::LoadLibrary("MyLib");
ASSERT(hMyDll) = NULL);
pfnMyFunc = (UINT( * )(Char))::GetProAddress(hMyDll,"MyFunc");
ASSERT(pfnMyFunc! =NULL);
UINT nRC = (pfnMyFunc)("Save");
pfnMyFunc = (UINT * )(Char * )::GetProcAddress(hMyDll,MAKEINTRESOURCE(42));
```

如果没有找到函数,这个函数将返回 NULL;如果找到函数,它返回函数的类属指针。应用程序应该保证使用与 DLL 函数相同的返回值和参数外表。在前面的例子中,MyFunc()接收一个字符串指针,返回一个 UINT。如果在参数列表中有匹配的参数,当通过指针调用时,有可能破坏堆栈。

(3)FreeLibrary()

使用 DLL 的程序结束时,调用 FreeLibrary()从内存中卸载它。

2. 加载 MFC 扩展 DLL

可以用 AfxLoadLibary 和 AfreeLibrary()来加载或卸出 MFC 扩展 DLL。这些函数与 Win32API 很相似,但它们保证被扩展 DLL 初始化的 MFC 结构不会被多线程破坏。本章中将看到更多的关于扩展 DLL 的内容。

3. 资源 DLL

也可以用动态装入的方式来装入资源 DLL,通过这种方式,MFC 将为应用程序装入缺省资源。为此,你首先要调用 LoadLibrary(),将 DLL 映射到内存,然后调用 AfxRecource(),让框架知道已经加载了一个新资源,而不是与可执行文件相连的资源。如果想用几种不同资源,例如不同语言的本地化,这种方法很有用。

17.5 为 DLL 创建接口

应用程序使用的 DLL 中所定义的各种符号称为符号引入当采用隐式链接方式来使用 DLL 时,调用的应用程序必须同 DLL 的输入库相链接,同时还必须使用 dllimport 属性项加以说明。

在编写 DLL 程序时,应创建一个头文件并用它来包含应用程序所需的各种引入说明。这样,应用程序只需简单地用 # include 直接访问 Dll 的 Public 属性的符号和对象。对此,Microsoft 公司为程序员推荐下述方法,即为-deslspec(dllimport)定义一个宏,然后采用此宏来说明一个引入符号。例如:

```
# define DllImport-deslspec(sdllimport)
DllImport int SomeFunction1();
DllImport void SomeFunction2();
```

注意:dllimport 属性说明项必须只能用于说明中,而不能用于定义中。

1.引出符号

宏也常用于对引出符号的说明,以使程序更具可读性并且容易修改。

define DllExport-declspec(dllexport)

使用 dllexport 属性说明项并配以-declspec 关键字告诉编译器的输出符号,-declspec 关键字为给定符号扩充存储类属性,并规定编译器如何访问它。

注意:为允许应用程序正确地声明每个 DLL 的符号,DLL 的头文件应对各个符号采用 dllimport 方式声明。

2.指定 C 语言方式链接以保持兼窗口性

当用C++语言编写 Visual C++程序时,为使它在 C 语言编写的程序中可用,则必须规定为 C 链接方式。如果未采用 C 链接方式,编译器将采用C++链接方式,在函数名前面增加一个附加字符,此字符告诉链接器所使用的参数的数据类型从而可以区分出许多重载函数,如果不这样做C++程序则不识别它。为指定 C 链接方式,应在引出函数中采用 extern"C"表达式:

extern"C"-delspec(dllexport)DWORD SomeFunction(DWORD dwParam);

17.6 调试 DLL

可以用两种方法调试一个动态链接库(DLL)。如果有 DLL 和调用程序的源代码,就可以打开调用程序的工程,并从此处调试 DLL。如果动态装入 DLL,则可以在 Project Settings 属性表的 Additional DLLS 属性项中指定它。如果只有 DLL 的源程序,可以打开构造 DLL 的工程,用 Project Settings 属性表中的 Debug 属性项指定调用 DLL 的可执行文件。

1.用可执行文件的工程调用 DLL

1)从 Build 菜单中选择 Settings,弹出 Project Settings 属性表。

2)选择 Debug 属性页。

3)在 Category 下拉列表框中选择 General。

4)在 Program Arguments 文本框中,输入可执行文件需要的命令行参数。5)在 Local Name 栏输入调试的 DLLS 名。

6)在 Preload 栏中,如果想在调试开始时装入模块,选择该栏选框。

7)选择 OK,存储在工程中的信息。

8)从 Build 菜单中选择 GO,开始调试。

在 DLL 或调用中可以设置断点,可以打开 DLL 的一个源文件,并在该文件中设置断点,即使它不是可执行工程的一部分。

2.用工程 DLL 调用 DLL

1)从 Build 菜单中选择 Settings,弹出 Project Settings 属性表。

2)选择 Debug 属性表。

3)在 Category 下拉列表框中选择 General。

4)在 Executable For Dcbug Session 文本框中,输入调用 DLL 的可执行文件名。

5)在 Category 列表框中,选择 Additional DLLS。

6)在 Local Module Name 栏中,输入希望调用的 DLLS 名。

7)选择 OK,存储工程中的信息。

8)在 DLL 源文件或函数符号处设置断点。

9)从 Build 菜单中选择 GO,开始调试。

17.7 实 例 分 析

在本节我们将给读者演示两个实例,相信读者看完这两个例子后,一定会对本章内容有一更好的了解。

17.7.1 Testdyn 例程

该实例演示动态链接库的使用方法。如图 17.1 和 17.2 所示,我们知道动态链接库(注意,我们所说的动态链接库指 dll 文件)有几种类型 win32,dynmmic _ Link ,Library 和 MFC Appwizard (dll)支持的三种:Regnlar DLL with statically cinkecl,regnlar ,DLL (using shared MFC DLL)和 MFC tension DLL(using shared MFC DLL)。在下一个例子里面演示如何生成和使用 MFC Extension DLL(using shared MFC DLL)。本例子着重介绍 win32 dynamic _ Link Library 和 Regnlar DLL vsing shared MFC DLL 型动态链接库的生成和使用。

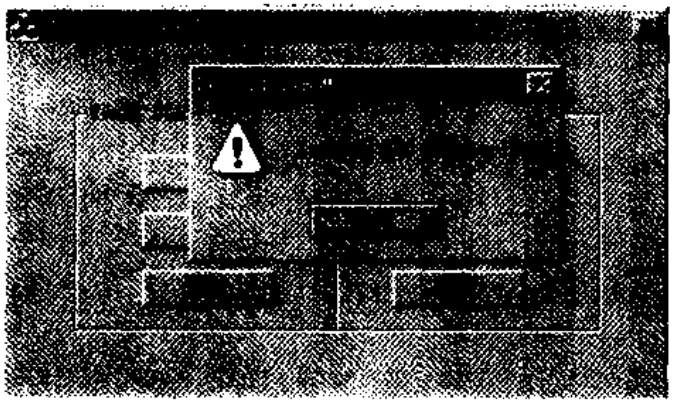


图 17.1 测试动态 DLL 图一

两个 GroupBox 分居对话框的两边,每个 Group Box 里面都有三个用来测试动态链接库的按钮,分别是"装入""卸下"和"测试"。Win32 DLL Group Box 演示一个 Dynamic _ Link Library 的使用,操作顺序是先装入,再测试,若卸下之后要测试,则必须先装入,若择作顺序不对,则有提示信息产生,如图 17.1 所示,MFCRegular DLL Group Box 演示一个 Regular DLL (using shared MFC DLL)的使用,使用方法同左边一样。当先"装入"dll 文件后,点击"测试"按钮,弹出如图所示对话框。则弹出 Windows98 的计算器,如图 17.2 所示。

我们先介绍产生调用 dll 文件的两个动态链接库的工程 DllDemoM 和 DllDemoW。DllDemoM 工程是 Regular DLL (using shared MFC DLL)型的,在这个工程里面我们要生成一个可

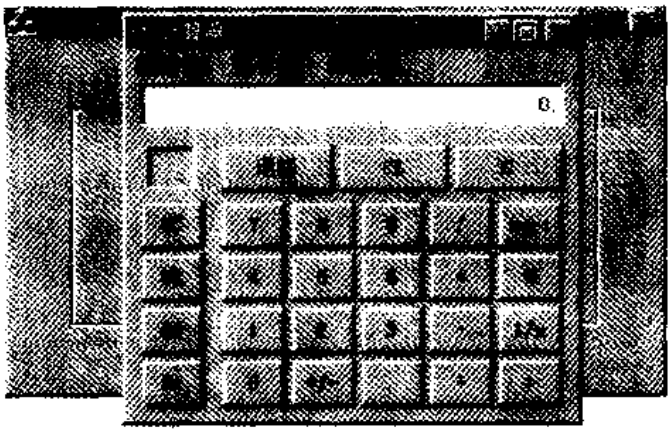


图 17.2 测试动态 DLL 图二

在 testdyn 工程调用的出数 Message，它的作用是弹出一对话框。参见下面两个文件 DllDemoM.cpp 和 DllDemoM.h。DllDemoW 是 Win32 dynamic _ Link Library 型的动态链接库工程，在这工程内我们要新建一函数 Execute，调出 Windows 随带的计算器。参见文件 DllDemoW.cpp。

接下来介绍 testdyn 的创建过程。

1)创建对话框类型的 tesedyn 工程，

2)把对话框上所有的控件都删掉，把标题换成测试动态连接库的使用，加上两个 Group Box，Win32 DLL 和 MFC Regular DLL。在 Win32 DLLGroup Box 内加上三个 Button："装入"(IDC _ LOADW)、"卸下"和测试(IDC _ TESTW)。在 MFC Regular DLL Group Box 内加上三个 Button：装入(IDC _ LOADM)、卸下(IDC _ VLOADM)和测试(IDC _ TGSTM)

3)具体的实现参见下面的文件 tdstdynDLg.h 和 testdynDLg.cpp。

```
// DllDemoM.h : main header file for the DLLDEMOM DLL
//

#if ! defined(AFX_DLLDEMOM_H__59533FC9_30A9_11D2_BA07_444553540000__INCLUDED
_)
#define AFX_DLLDEMOM_H__59533FC9_30A9_11D2_BA07_444553540000__INCLUDED

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

////////////////////////////////////////////////////////////////////////////
```

```
// CDllDemoMApp
// See DllDemoM.cpp for the implementation of this class
//

class CDllDemoMApp : public CWinApp
{
public:
	CDllDemoMApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CDllDemoMApp)
	//}}AFX_VIRTUAL

	//{{AFX_MSG(CDllDemoMApp)
		// 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()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DLLDEMOM_H__59533FC9_30A9_11D2_BA07_444553540000__INCLUDED_)

// DllDemoM.cpp : Defines the initialization routines for the DLL.
//

#include "stdafx.h"
#include "DllDemoM.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

UINT Message();
```

```
//
//    Note!
//
//        If this DLL is dynamically linked against the MFC
//        DLLs, any functions exported from this DLL which
//        call into MFC must have the AFX_MANAGE_STATE macro
//        added at the very beginning of the function.
//
//        For example:
//
//        extern "C" BOOL PASCAL EXPORT ExportedFunction()
//        {
//            AFX_MANAGE_STATE(AfxGetStaticModuleState());
//            // normal function body here
//        }
//
//        It is very important that this macro appear in each
//        function, prior to any calls into MFC. This means that
//        it must appear as the first statement within the
//        function, even before any object variable declarations
//        as their constructors may generate calls into the MFC
//        DLL.
//
//        Please see MFC Technical Notes 33 and 58 for additional
//        details.
//

/////////////////////////////////////////////////////////////////////////////
// CDllDemoMApp

BEGIN_MESSAGE_MAP(CDllDemoMApp, CWinApp)
    //{{AFX_MSG_MAP(CDllDemoMApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDllDemoMApp construction

CDllDemoMApp::CDllDemoMApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
```

```
}

//////////////////////////////////////////////////////////////////////////
// The one and only CDllDemoMApp object

CDllDemoMApp theApp;

UINT Message()
{
	return MessageBox(NULL,"Testing MFC Regular DLL","From DllDemoM",MB_ICONEXCLA-
MATION);
}


#include <windows.h>

void Execuate();


BOOL WINAPI APIENTRY DllMain(HANDLE hModule,DWORD reason,LPVOID lpReserved)
{
	switch(reason)
	{
	case DLL_PROCESS_ATTACH:
		break;
	case DLL_THREAD_ATTACH:
		break;
	case DLL_THREAD_DETACH:
		break;
	case DLL_PROCESS_DETACH:
		break;
	}
	return true;
}

void Execuate()
{
	STARTUPINFO StartupInfo;
	memset(&StartupInfo,0,sizeof(STARTUPINFO));
	StartupInfo.cb = sizeof(STARTUPINFO);
	PROCESS_INFORMATION ProcessInfo;
	BOOL suc = ::CreateProcess("c:\\Windows\\Calc.exe",
		NULL,
		NULL,
```

```
            NULL,
            FALSE,
            0,
            NULL,
            NULL,
            &StartupInfo,
            &ProcessInfo);
    if(!suc)
            MessageBox(NULL,"Error",NULL,MB_ICONEXCLAMATION);

}



// testdynDlg.h : header file
//

#if !defined(AFX_TESTDYNDLG_H__A4A28D0A_2E0B_11D2_BA07_444553540000__INCLUDED_)
#define AFX_TESTDYNDLG_H__A4A28D0A_2E0B_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CTestdynDlg dialog

class CTestdynDlg : public CDialog
{
// Construction
public:
    CTestdynDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(CTestdynDlg)
    enum { IDD = IDD_TESTDYN_DIALOG };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTestdynDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL
```

```
// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CTestdynDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnLoadm();
	afx_msg void OnLoadw();
	afx_msg void OnTestm();
	afx_msg void OnTestw();
	afx_msg void OnUloadm();
	afx_msg void OnUloadw();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_TESTDYNDLG_H__A4A28D0A_2E0B_11D2_BA07_444553540000__INCLUDED_)// testdynDlg.h : header file
//

#if !defined(AFX_TESTDYNDLG_H__A4A28D0A_2E0B_11D2_BA07_444553540000__INCLUDED_)
#define AFX_TESTDYNDLG_H__A4A28D0A_2E0B_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CTestdynDlg dialog

class CTestdynDlg : public CDialog
{
// Construction
public:
```

```
        CTestdynDlg(CWnd * pParent = NULL); // standard constructor

    // Dialog Data
        //{{AFX_DATA(CTestdynDlg)
        enum { IDD = IDD_TESTDYN_DIALOG };
            // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA

        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CTestdynDlg)
        protected:
        virtual void DoDataExchange(CDataExchange * pDX);       // DDX/DDV support
        //}}AFX_VIRTUAL

    // Implementation
    protected:
        HICON m_hIcon;

        // Generated message map functions
        //{{AFX_MSG(CTestdynDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        afx_msg void OnLoadm();
        afx_msg void OnLoadw();
        afx_msg void OnTestm();
        afx_msg void OnTestw();
        afx_msg void OnUloadm();
        afx_msg void OnUloadw();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
    };

    //{{AFX_INSERT_LOCATION}}
    // Microsoft Visual Studio will insert additional declarations immediately before the previous
    line.

    #endif // !defined(AFX_TESTDYNDLG_H__A4A28D0A_2E0B_11D2_BA07_444553540000__
    INCLUDED_)



    // testdynDlg.cpp : implementation file
```

```
//

#include "stdafx.h"
#include "testdyn.h"
#include "testdynDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

HINSTANCE hdllw=NULL;
HINSTANCE hdllm=NULL;
typedef void (* EXECUATE)(void);
typedef UINT (* MESSAGE)(void);
EXECUATE execuate;
MESSAGE message;

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestdynDlg dialog

CTestdynDlg::CTestdynDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CTestdynDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CTestdynDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CTestdynDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CTestdynDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTestdynDlg, CDialog)
	//{{AFX_MSG_MAP(CTestdynDlg)
	ON_WM_SYSCOMMAND()
```

```
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_LOADM, OnLoadm)
    ON_BN_CLICKED(IDC_LOADW, OnLoadw)
    ON_BN_CLICKED(IDC_TESTM, OnTestm)
    ON_BN_CLICKED(IDC_TESTW, OnTestw)
    ON_BN_CLICKED(IDC_ULOADM, OnUloadm)
    ON_BN_CLICKED(IDC_ULOADW, OnUloadw)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestdynDlg message handlers

BOOL CTestdynDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);           // Set small icon

    // TODO: Add extra initialization here

    return TRUE;  // return TRUE unless you set the focus to a control
```

```
}

void CTestdynDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

void CTestdynDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}
```

```
// The system calls this to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CTestdynDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_ hIcon;
}


void CTestdynDlg::OnLoadm()
{
	// TODO: Add your control notification handler code here
	if(hdllm! =NULL)
	{
		MessageBox("该动态连接库已被装入!");
		return;
	}
	hdllm=LoadLibrary("e:\\chb\\projects\\DllDemoM\\debug\\DllDemoM.dll");
	if(hdllm==NULL)
		MessageBox("不能装入该动态连接库!");
	message=(MESSAGE)GetProcAddress(hdllm,"Message");



}

void CTestdynDlg::OnLoadw()
{
	// TODO: Add your control notification handler code here
	if(hdllw! =NULL)
	{
		MessageBox("该动态连接库已被装入!");
		return;
	}
	hdllw=LoadLibrary("e:\\chb\\projects\\DllDemoW\\debug\\DllDemoW.dll");
	if(hdllw==NULL)
		MessageBox("不能装入该动态连接库!");
	execuate=(EXECUATE)GetProcAddress(hdllw,"Execuate");


}

void CTestdynDlg::OnTestm()
{
	// TODO: Add your control notification handler code here
```

```
    if(hdllm==NULL)
    {
        MessageBox("请先装入动态连接库!");
        return;
    }
    message();

}

void CTestdynDlg::OnTestw()
{
    // TODO: Add your control notification handler code here
    if(hdllw==NULL)
    {
        MessageBox("请先装入动态连接库!");
        return;
    }
    execuate();


}

void CTestdynDlg::OnUloadm()
{
    // TODO: Add your control notification handler code here
    if(hdllm==NULL)
    {
        MessageBox("动态连接库已卸下或者没有装入!");
        return;
    }
    FreeLibrary(hdllm);
    hdllm=NULL;

}

void CTestdynDlg::OnUloadw()
{
    // TODO: Add your control notification handler code here
    if(hdllw==NULL)
    {
        MessageBox("动态连接库已卸下或者没有装入!");
        return;
    }
    FreeLibrary(hdllw);
```

```
        hdllw = NULL;

}



/////////////////////////////////////////////////////////////////////////////
// CDllDialog dialog
# include "resource.h"

extern "C" AFX_EXT_API void WINAPI InitMfcexndDLL();

class AFX_EXT_CLASS CDllDialog : public CDialog
{
        DECLARE_DYNCREATE(CDllDialog)

// Construction
public:
        CDllDialog(CWnd* pParent = NULL); // standard constructor

// Dialog Data
        //{{AFX_DATA(CDllDialog)
        enum { IDD = IDD_DIALOG1 };
        int             m_Color;
        //}}AFX_DATA

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDllDialog)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);      // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:
        void PositionCtl();

        // Generated message map functions
        //{{AFX_MSG(CDllDialog)
        afx_msg void OnColor();
        afx_msg void OnPaint();
        afx_msg void OnDimension();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
```

```
;
#undef AFX_DATD
#define AFX_DATA
```

17.7.2 usemed 例程

该例演示了动态链接库的使用方法。如图 17.3 所示，当在视窗内单击鼠标左键时弹出一对话框。对话框上的 Dimen 按钮可控制对话框的大小，点击它时对话框的宽度将加倍。其上的颜色 Group Box 用来控制对话框底色，可在红、绿、蓝之间切换。点击 OK 或 Cancel 按钮，对话框关闭。

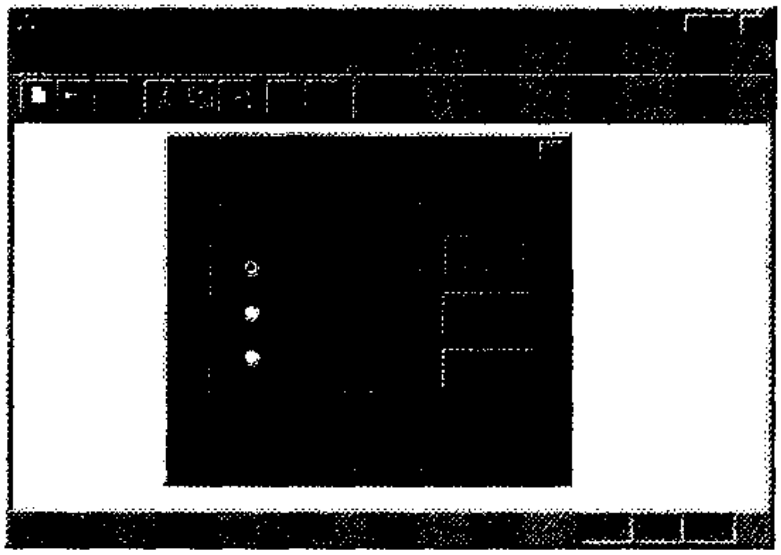


图 17.3 usmed 例程图

为了说明动态链接库的使用方法，我们先生成一动态链接类型的工程 mfcexnd 来实现上面对话框的功能。然后生成 usemed，单击时调用 mfcexnd 产生的 dll 文件，来产生已在 mfcexnd.dll 文件内实现了的对话框。

首先介绍怎样生成 mfcexnd 工程。

1)在开始创建之前选择 MFC AppWizard(dll)类型，接下来的一步，提示 what type of DLL (using shared MFC DLL)，接着点 Finish 即可。

2)生成基本框架之后，打开 Resource View 标签框，在其中加入一对话框(IDD _ DIALOG1)实现类为 CDDiaLog，内加入控制对话框颜色的 Group Box Color (IDC _ COLOR)，其中有三个选项 Red (IDC _ RED)，Green(IDC _ GREEN)和 Blue(IDC _ BLUE)。再加上一按钮 Dimen(IDC _ DIMENSION)用来改变对话框的大小。具体实现的代码可见下面的 mfcexnd.h 和 mfcexnd.cpp。

接下来介绍 usemed 工程如何调用 mfcexnd 中的函数。

先简单地生成 MFC AppWizard(exe)型的单文档(single dounment)usemed 工程。由于我们要实现当单击视窗，便出现 mfcexnd 中实现的对话框，所以首先必须在文件 usemedView.h

的开头加上语句“CDDialog”，在文件 usemedView.cpp 开头加上 # Include“mfcexnd.h”。具体如何调用见下面两个文件 usemedView.h 和 usemedView.cpp。不过需要注意的是最后编译 usemed 工程时，需设置如下环境，打开 Tools 的 options 窗口，在 Directories 对话框内的 show directoryies for：栏内选 Include files，在下面的 Directories 内添加一次上面已说明的包括文件 mfcexnd.h 的目录，再把由 mfcexnd 工程生成的文件 mfcexnd.dll 文件拷贝到机器系统 system 目录下。

```
// mfcexnd.cpp : Defines the initialization routines for the DLL.
//

#include "stdafx.h"
#include "mfcexnd.h"
#include "resource.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#include <afxdllx.h>



static AFX_EXTENSION_MODULE MfcexndDLL = { NULL, NULL };

extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
	// Remove this if you use lpReserved
	UNREFERENCED_PARAMETER(lpReserved);

	if (dwReason == DLL_PROCESS_ATTACH)
	{
		TRACE0("MFCEXND.DLL Initializing!\n");

		// Extension DLL one-time initialization
		if (!AfxInitExtensionModule(MfcexndDLL, hInstance))
			return 0;

		// Insert this DLL into the resource chain
		// NOTE: If this Extension DLL is being implicitly linked to by
		// an MFC Regular DLL (such as an ActiveX Control)
		// instead of an MFC application, then you will want to
```

```
        // remove this line from DllMain and put it in a separate
        // function exported from this Extension DLL. The Regular DLL
        // that uses this Extension DLL should then explicitly call that
        // function to initialize this Extension DLL. Otherwise,
        // the CDynLinkLibrary object will not be attached to the
        // Regular DLL's resource chain, and serious problems will
        // result.

    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
        TRACE0("MFCEXND.DLL Terminating!\n");
        // Terminate the library before destructors are called
        AfxTermExtensionModule(MfcexndDLL);
    }
    return 1; // ok
}
extern "C" void WINAPI InitMfcexndDLL()
{
    // create a new CDynLinkLibrary for this app
    new CDynLinkLibrary(MfcexndDLL);
    // nothing more to do
}

/////////////////////////////////////////////////////////////////////////////
// CDllDialog dialog
IMPLEMENT_DYNCREATE(CDllDialog, CDialog)


CDllDialog::CDllDialog(CWnd* pParent /*=NULL*/)
    : CDialog(CDllDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(CDllDialog)
    m_Color = 0;
    //}}AFX_DATA_INIT
}


void CDllDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDllDialog)
    DDX_Radio(pDX, IDC_RED, m_Color);
    //}}AFX_DATA_MAP
```

```
}


BEGIN_MESSAGE_MAP(CDllDialog, CDialog)
	//{{AFX_MSG_MAP(CDllDialog)
	ON_BN_CLICKED(IDC_BLUE, OnColor)
	ON_WM_PAINT()
	ON_BN_CLICKED(IDC_GREEN, OnColor)
	ON_BN_CLICKED(IDC_RED, OnColor)
	ON_BN_CLICKED(IDC_DIMENSION, OnDimension)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CDllDialog message handlers

void CDllDialog::OnColor()
{
    UpdateData();
	Invalidate();
}

void CDllDialog::OnPaint()
{
	CPaintDC dc(this); // device context for painting


	CRect rect;
	GetWindowRect(&rect);
	ScreenToClient(&rect);
	COLORREF col;
	switch(m_Color)
	{
	case 0:
		col= RGB(255,0,0);
		break;
	case 1:
		col= RGB(0,255,0);
		break;
	case 2:
		col= RGB(0,0,255);
		break;
	}
```

```
        CBrush brush(col);
        CBrush* pOldBrush = dc.SelectObject(&brush);
        dc.PatBlt(rect.left,rect.top,rect.Width(),rect.Height(),PATCOPY);
        dc.SelectObject(pOldBrush);

        // Do not call CDialog::OnPaint() for painting messages
}

void CDllDialog::OnDimension()
{

        CRect rect;
        GetWindowRect(&rect);
        int x = rect.left;
        int y = rect.top;
        int cx = rect.Width() * 2;
        int cy = rect.Height();
        SetWindowPos(NULL,x-cx/2,y,cx,cy,SWP_SHOWWINDOW|SWP_NOZORDER);

        PositionCtl();



}

void CDllDialog::PositionCtl()
{
        CRect rectwin,rectcol,rectred,rectgreen,rectblue,
                rectok,rectcal,rectdim;
        CStatic* m_col;
        CButton *m_red,*m_blue,*m_green,*m_ok,*m_cal,*m_dim;
        m_col=(CStatic*)GetDlgItem(IDC_COLOR);
        m_col->GetWindowRect(&rectcol);
        m_red=(CButton*)GetDlgItem(IDC_RED);
        m_red->GetWindowRect(&rectred);
        m_blue=(CButton*)GetDlgItem(IDC_BLUE);
        m_blue->GetWindowRect(&rectblue);
        m_green=(CButton*)GetDlgItem(IDC_GREEN);
        m_green->GetWindowRect(&rectgreen);
        m_ok=(CButton*)GetDlgItem(IDOK);
        m_ok->GetWindowRect(&rectok);
        m_cal=(CButton*)GetDlgItem(IDCANCEL);
        m_cal->GetWindowRect(&rectcal);
        m_dim=(CButton*)GetDlgItem(IDC_DIMENSION);
```

```
    m_dim->GetWindowRect(&rectdim);

    GetWindowRect(&rectwin);
    ScreenToClient(&rectcol);
    ScreenToClient(&rectred);
    ScreenToClient(&rectblue);
    ScreenToClient(&rectgreen);
    ScreenToClient(&rectok);
    ScreenToClient(&rectcal);
    ScreenToClient(&rectdim);

    int offset = rectwin.Width()/2;
    m_col->SetWindowPos(NULL,rectcol.left + offset,rectcol.top,0,0,SWP_NOSIZE|SWP
_NOZORDER);
    m_red->SetWindowPos(NULL,rectred.left + offset,rectred.top,0,0,SWP_NOSIZE|
SWP_NOZORDER);
    m_blue->SetWindowPos(NULL,rectblue.left + offset,rectblue.top,0,0,SWP_NOSIZE|
SWP_NOZORDER);
    m_green->SetWindowPos(NULL,rectgreen.left + offset,rectgreen.top,0,0,SWP_NO-
SIZE|SWP_NOZORDER);
    m_ok->SetWindowPos(NULL,rectok.left + offset,rectok.top,0,0,SWP_NOSIZE|SWP_
NOZORDER);
    m_cal->SetWindowPos(NULL,rectcal.left + offset,rectcal.top,0,0,SWP_NOSIZE|SWP
_NOZORDER);
    m_dim->SetWindowPos(NULL,rectdim.left + offset,rectdim.top,0,0,SWP_NOSIZE|
SWP_NOZORDER);



}

// usemedView.h : interface of the CUsemedView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_USEMEDVIEW_H__FF2BDEDA_2A6E_11D2_BA07_444553540000__IN-
CLUDED_)
#define AFX_USEMEDVIEW_H__FF2BDEDA_2A6E_11D2_BA07_444553540000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CDllDialog;
```

```
class CUsemedView : public CView
{
protected: // create from serialization only
	CUsemedView();
	DECLARE_DYNCREATE(CUsemedView)

// Attributes
public:
	CUsemedDoc* GetDocument();
	CDllDialog* pdlg;

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CUsemedView)
	public:
	virtual void OnDraw(CDC* pDC); // overridden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	protected:
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CUsemedView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
	//{{AFX_MSG(CUsemedView)
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
#ifndef _DEBUG // debug version in usemedView.cpp
inline CUsemedDoc* CUsemedView::GetDocument()
{ return (CUsemedDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_USEMEDVIEW_H__FF2BDEDA_2A6E_11D2_BA07_444553540000__
INCLUDED_)

// usemedView.cpp : implementation of the CUsemedView class
//

#include "stdafx.h"
#include "usemed.h"
#include "mfcexnd.h"

#include "usemedDoc.h"
#include "usemedView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CUsemedView

IMPLEMENT_DYNCREATE(CUsemedView, CView)

BEGIN_MESSAGE_MAP(CUsemedView, CView)
    //{{AFX_MSG_MAP(CUsemedView)
    ON_WM_LBUTTONDOWN()
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CUsemedView construction/destruction

CUsemedView::CUsemedView()
{
	// TODO: add construction code here

}

CUsemedView::~CUsemedView()
{
}

BOOL CUsemedView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	// the CREATESTRUCT cs

	return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CUsemedView drawing

void CUsemedView::OnDraw(CDC* pDC)
{
	CUsemedDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	// TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CUsemedView printing

BOOL CUsemedView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CUsemedView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
```

```
    // TODO: add extra initialization before printing
}

void CUsemedView::OnEndPrinting(CDC * /*pDC*/, CPrintInfo * /*pInfo*/)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CUsemedView diagnostics

#ifdef _DEBUG
void CUsemedView::AssertValid() const
{
    CView::AssertValid();
}

void CUsemedView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CUsemedDoc * CUsemedView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CUsemedDoc)));
    return (CUsemedDoc *)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CUsemedView message handlers

void CUsemedView::OnLButtonDown(UINT nFlags, CPoint point)
{

   CDllDialog dlg;
    dlg.DoModal();
    CView::OnLButtonDown(nFlags, point);
}
```

17.8 小　　结

DLL作为提高资源利用效率的一项重要技术,对编写一个好的应用程序是不可缺少的。

本章只是对这方面内容做了解一般性介绍，读者应能通过阅读相关书籍，更好地掌握这方面内容。本章向读者介绍了 DLL 的基础知识及 MFC 的 DLL，最后向读者介绍了 DLL 的创建及应用，希望读者在阅读完之后对 DLL 有初步认识和应用。

附录 1　MFC 消息映射函数

WM_COMMAND 消息控制函数

映射入口　　函数原型

ON_COMMAND(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_COMMAND-Ex(〈id〉,〈memberFxn〉) afx_msg BOOLmemberFxn(UINT);

子窗口通知消息控制函数

映射入口　　函数原型

一般的控制通知代码

ON_CONTROL(〈WNotifyCode〉,〈id〉,〈memberFxn〉 afx_msg void memberFxn();

ON_NOTIFY(〈WNotifyCode〉,〈id〉,〈memberFxn〉 afx_msg void memberFxn();

用户按钮通知代码

ON_BN_CLICKED(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_BN_disable(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_BN_DOUBLECLICKED (〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_BN_HILITE (〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_BN_PAINT(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_BN_ONHILITE (〈id〉,〈memberFxn〉) afx_msg void memberFxn();

组合框通知代码

ON_CBN_CLOSEUP (〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_CBN_DBLCLK(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_CBN_DROPDOWN(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_CBN_EDITUPDATE(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_CBN_ERRSPACE(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_CBN_KILLFOCUS(〈id〉,〈memberFxn〉) afx_msg void memberFxn();ON_CBN_SELEHANGE (〈id〉,〈memberFxn〉) afx_msg void memberFxn();ON_CBN_SELENDCANCEL(〈id〉,〈member-Fxn〉) afx_msg void memberFxn();ON_CBN_SELENDOK(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_CBN_SETFOCUS(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

编控制通知代码

ON_EN_CHANGE(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_EN_ERRSPACE(〈id〉,〈memberFxn〉) afx_msg void memberFxn();

ON_EN_HSCROLL〈id〉,〈memberFxn〉 afx_msg void memberFxn();

ON_EN_KILLFOCUS〈id〉,〈memberFxn〉 afx_msg void memberFxn();

ON_EN_MAXTEXT〈id〉,〈memberFxn〉 afx_msg void memberFxn();

ON_EN_SETFOCUS(id),(memberFxn) afx_msg void memberFxn();
ON_EN_UPPATE((id),(memberFxn)) afx_msg void memberFxn();
ON_EN_USCROLL(id),(memberFxn, afx_msg void memberFxn();

外表框通知代码

ON_LBN_DBLCLK(id),(memberFxn) afx_msg void memberFxn();
ON_LBN_ERRSPACE(id),(memberFxn) afx_msg void memberFxn();
ON_LBN_KILLFOCUS(id),(memberFxn) afx_msg void memberFxn();
ON_LBN_SELCHANGE(id),(memberFxn) afx_msg void memberFxn();
ON_LBN_SETFOCUS(id),(memberFxn) afx_msg void memberFxn();

滑块控制通知代码

ON_WM_HSCROLL(id),(memberFxn) afx_msg void memberFxn();
ON_WM_VSCROLL(id),(membe·Fxn) afx_msg void memberFxn();
ON_EN_UPDATE(id),(memberFxn) afx_msg void memberFxn();

滚动条控制通知代码

ON_WM_HSCROLL(id),(memberFxn) afx_msg void memberFxn();
ON_WM_VSCROLL(id),(memberFxn) afx_msg void memberFxn();

列表视图通知代码

LVN_BEGINDRAG(id),(memberFxn, afx_msg void memberFxn();
LVN_BEGINLABELEDIT(id),(memberFxn) afx_msg void memberFxn();
LVN_BEGINRDRAG(id),(memberFxn) afx_msg void memberFxn();
LVN_COLUMNCLICK(id),(memberFxn) afx_msg void memberFxn();
LVN_DELETEALLITEMS(id),(memberFxn)afx_msg void memberFxn();
LVN_DELETEITEM(id),(memberFxn) afx_msg void memberFxn();
LVN_ENDLABELEDIT(id),(memberFxn) afx_msg void memberFxn();LVN_GETDISPINFO(id),
(memberFxn) afx_msg void memberFxn();
LVN_INSERTITEM(id),(memberFxn, afx_msg void memberFxn();
LVN_ITEMCHANGED(id),(memberFxn) afx_msg void memberFxn();
LVN_KEYDOWN(id,),(memberFxn) afx_msg void memberFxn();
LVN_PEN(id),(memberFxn, afx_msg void memberFxn();
LVN_SETDISPINEO(id),(memberFxn) afx_msg void memberFxn();

树视图通知代码

TVN_BEGINDRAG(id),(memberFxn) afx_msg void memberFxn();
TVN_BEGINLABELEDIT(id),(memberFxn) afx_msg void memberFxn();
TVN_BEGINRDRAG(id),(memberFxn) afx_msg void memberFxn();
TVN_DELETEITEM(id),(memberFxn, afx_msg void memberFxn();
TVN_ENDLABELEDIT(id),(memberFxn) afx_msg void memberFxn();
TVN_GETDISPINFO(id),(memberFxn) afx_msg void memberFxn();
TVN_ITEMEXPANDED(id),(memberFxn) afx_msg void memberFxn();

TVN_KEYDOWN(id,(memberFxn) afx_msg void memberFxn();
TVN_SELCHANGED(id,(memberFxn, afx_msg void memberFxn();
TVN_SELCHANGING(id),(memberFxn) afx_msg void memberFxn();TVN_SETDISPINFO(id),
(memberFxn) afx_msg void memberFxn();

列表控制通知代码

TCN_KEYDOWN(id),(memberFxn) afx_msg void memberFxn();
TCN_SELCHANGE(id),(memberFxn) afx_msg void memberFxn();
TCN_SELCHANGING(id),(memberFxn) afx_msg void memberFxn();

WINDOWS通知消息控制函数

映射入口　　函数原型

ON_WM_ACTIVATE() afx_msg void OnActivate (UINT CWnd * BOOL);
ON_WM_ACTIVATEAPP() afx_msg void OnActiviteApp(BOOL,HANDLB);
ON_WM_ASKCBFORMATNAME() afx_msg void OnAskCbFormatName(UINT,LPSTR);
ON_WM_CANCELMODE() afx_msg void OnCancelmode();
ON_WM_CHANGECBCHAIN() afx_msg void OnChangeCbChain(HWND,HWND);
ON_WM_CHAR() afx_msg void OnChar(UINT,UINT,UINT);
ON_WM_CHARTOITEM() afx_msg int OnCharToIten(UINT CWnd * UINT);
ON_WM_CHILPACTIVATE() afx_msg void OnChildActwate();
ON_WM_CLOSE() afx_msg void OnClose();
ON_WM_COMPACTING() afx_msg void OnCompacting(UINT)
ON_WM_COMPAREITEM() afx_msg int OnCompareItem(LPCOMPAREITCMSTRVCT);
ON_WM_CREATE() afx_msg int OnCreate(LPCREATESTRVCT);
ON_WM_CTLCOLOR() afx_msg HBRUSH OnCtlcolor(CDC * Cwnd * UINT);
ON_WM_DEADCHAR() afx_msg void OnDeadChar(UINT UINT UINT);
ON_WM_DELETEITEM() afx_msg voidOnDeleteItem(LPDELETEITEMSTRUCT);
ON_WM_DESTROY() afx_msg void OnPestroy();
ON_WM_DESTROYCLIPBORD() afx_msg void OnPestoryClipboard();
ON_WM_DEVMODE-CHANGE() afx_msg void OnDevModeChange(LPSTR);
ON_WM_DRAWCLIPBOARD() afx_msg void OnDrawClipBoard();
ON_WM_DRAWITEM() afx_msg voidOnDrawItem(LPDRAWITEMSTRVCT);
ON_WM_DROPFILES() afx_msg void OnDropFiles(HANDLE);
ON_WM_ENABLE() afx_msg void OnEnable(BOOL);ON_WM_ENDSESSION() afx_msg void On
EndSession(BOOL);
ON_WM_ENTERIDLE() afx_msg void On EnterIdle(UINT,CWnd *);
ON_WM_ERASEBKGND() afx_msg BOOL OnErase Bkgnd(CDC *);
ON_WM_FONTCHANGE() afx_msg void OnFontChange();

WINDOWS通知消息控制函数

映射入口　　函数原型

ON_WM_GETDLGCODE() afx_msg UINT OngETdLGmODEES;
ON_WM_GETMINMAXINFO() afx_msg void OnGetMinMarInfo(LPPOINT);

ON_WM_HSCROLL() afx_-msg void OnHscroll(UINT,UINT,CWnd*);
ON_WM_HSCROLLCLIPBOARD() afx_msg void OnHscrollClipboard(CWnd*UINT,UINT);
ON_WM_ICONERASEBKGND() afx_msg void OnIconEraseBkgnd(CDC*);
ON_WM_INITMENO() afx_msg void OnInitMenu(CMenu*);
ON-WM_INITMENUPOPUP() afx_msg void OnInit MenuPopup(CMenu*,UINT,UINT);
ON_WM_KEYDOWN() afx msg void OnkeyDown(UINT,UINT,UINT);
ON-WM_KEYUP() afx_msg void OnKeyUp(UINT,UINT,UINT);
ON_WM_KILLFOCUS() afx_msg void OnKillFocus(CWnd*);
ON_WM_LBUTTONDBLCLK() afx_msg void OnLButtonDblclk(UINT,CPoint);
ON_WM_LBUTTDNUP() afx_msg void OnLButtonUp(UINT,CPoint);
ON_WM_MBUTTONDBLCLK() afx_msg void OnMButton DBCLK(UINT,CPoint);
ON_WM_MBUTTONDOWN() afx_msg void OnMButtonDown(UINT,CPoint);
ON_WM_MBUTTONUP() afx_msg void OnMButtonUp(UINT,CPoint);
ON_WM_MDIACTNATE() afx_msg void OnMDIActivate CBOOL,(Wnd*,CWnd*);
ON_WM_MEASUREITEM() afx_msg void OnMeasureItem(LPMEASUREITEMSTRUCT);
ON_WM_MENCHAR() afx_msg LONG ONMenuChar(UINT,UINT,CMENU);
ON_WM_MENUSELECT() afx_msg void OnMenuSelect(UINT,UINT,HMENO);
ON_WM_MOUSEACTIVATE() afx_msg void OnMouseActivate(CMnd*UINT,UINT);
ON_WM_MOUSEMOVE() afx_msg void OnMouseMove(UINT,CPoint);
ON_WM_MOVE() afx_msg void OnMove(int,int);ON_WM_NCACTIVATE() afx_msg BOOL On-
NcActinate(BOOL);
ON_WM_NCCALCSITE() afx_msg void OnNcCalcSize(LPRECT);
ON_WM_NCCREATE() afx_msg BOOL ONNcCreate(LPCREATESTRUCT);
ON_WM_NCDESTROY() afx_msg void OnNcDestroy();
ON_WM_NCHITTEST() afx_msg UINT OnNcHitTest(CPoint);
ON_WM_NCLBUTTONDBLCLK() afx_msg void OnNcLButtonDblclk(UINT,CPoint);
ON_WM_NCLBOTTONDOWN() afx_msg void OnNcLButtonDown(UINT,CPoint);
ON_WM_NCLBUTTONUP() afx_msg void OnNcLButtonUp(UINT,CPaint);
ON_WM_NCMBUTTONDBLCLK() afx_msg void OnNcMButtonDBlclk(UINT,CPoint);
ON_WM_NCMBUTTONDOWN() afx_msg void OnNcMButtonDown(UINT,CPoint);
ON_WM_NCMBUTTONUP() afx_msg void OnNcMButtonUp(UINT,CPoint);
ON_WM_NCMOUSEMOVE() afx_msg void OnNcMouseMove(UINT,CPoint);
ON_WM_NCPAINT() afx_msg void OnNcPaint();
ON_WM_NCRBUTTONDBLCLK()afx_-msg void OnNcRButtonDblclk(UINT,CPoint);
ON_WM_NCRBUTTONDOWN()afx_msg void OnNcRButtonDown(UINT,CPoint);
ON_WM_NCRBUTTONUP()afx_msg void OnNcRButtonUp(UINT,CPoint);
ON_WM_PAINT()afx_msg void OnPaint();

WINDOWS 通知消息控制函数

映射入口　　函数原型

ON_WM_PAINTCLIPBOARD() afx_msg void OnPaintClipboard(CWnd*,HANDLE);
ON_WM_PALETTECHANGED() afx_msg void OnPaletteChanged(CWnd*);
ON_WM_PALETTECHANGING() afx msg void OnPaletteIsChanging(CWnd*);
ON_WM_PARENTNOTIFY() afx_msg void OnParentNotify(UINT,LONG);

ON_WM_QUERYDRAGICON() afx_msg HCURSOR OnQueryDragIcon();
ON_WM_QUERYNEWPALETTE()afx_msg BOOL OnQueryNewPalette();
ON_WM_QUERYENDSESSION() afx_msg BOOL OnQueryEndSession();
ON_WM_QUERYDPEN() afx_msg BOOL OnQuery Open();
ON_WM_RBOTTONDBLCLK() afx_msg void OnRButtonDblclk(UINT,CPoint);
ON_WM_RBOTTONDOWN() afx_msg void OnRButton Down(UINT,CPoint);
ON_WM_RBUTTONUP() afx_msg void OnRButto Up(UINT,CPoint);
ON_WM_RENDERALLFDRMATS() afx_msg void OnRenderAclFormats();
ON_WM_RENDERFORMAT() afx_msg void OnRenderFormat(UINT);
ON_WM_SETCURSOR() afx_msg void OnSetFocus(CWnd*);
ON_WM_SHOWWINDOW() afx_msg void OnShowWindow(BOOL,UINT);
ON_WM_SIZE() afx_msg void OnSize(UINT,int,int);
ON_WM_SIZECLIPBDARD() afx_msg void OnSizeClipboard(CWnd*HANDLE);
ON_WM_SPDOLERSTATUS() afx_msg void OnSpoolerStatus(UINT,UINT);
ON_WM_SYSCHAR() afx_msg void OnSysChar(UINT,UINT,UINT);
ON_WM_SYSCOLORCHANGE() afx_msg void OnSysColorChange();
ON_WM_SYSCOMMAND() afx_msg voic OnSysDeadChar(UINT,UINT,UINT);
ON_WM_SYSKEYDOWN() afx_msg void OnSyskeyDown(UINT,UINT,UINT);
ON_WM_SYSKEYUP() afx_msg void OnsyskeyUp(UINT,UINT,UINT);
ON_WM_TIMECHANGE() afx_msg void OnTimechang();
ON_WM_VKEYTOITEM() afx_msg void OnVkeyToITEM(UINT,CWnd*,UINT);
ON_WM_TIMER() afx_msg void OnTimer(UINT);
ON-WM_VSCROLL() afx-msg void OnVScoll(UINT,UINT,CWnd*);
ON-WM_VSCROLLCLIPBOARD() afx-msg void OnVscrollclipboard(CWnd*,UINT,UINT);
ON-WM_WINDOWPOSCHANGED() afx-msg void OnWindowPosChanged(WINDOWPOS FAR*);
ON-WM_WINDOWPOSCHANGING() afx-msg void OnWindowPosChanging(WINDOWPOS FAR*);
ON-WM_WINTNICHANGE() afx-msg void OnWinInichange(LPSTR);

用户自定义消息代码

映射入口　　函数原型

ON-MESSAGE(〈message〉,〈memberFxn〉) afx msg LONGimemberFxn(UINT, LONG); ON-REGISTERED-MESSAGE(〈nMessageVariable〉,〈memberFxn〉) afx-msg LONGmemberFxn(UINT, LONG);

附录 2　C++ 教程

为了更好地学好 Visual C++ 6.0，读者必须懂得C++ 语言和面向对象编程方法。为此，我们在这里给出了C++ 语言的基本内容，当然，要想精通C++ 语言，读者还必须自己多下功夫学习，这对更好地利用本书是很有必要的。用 C 语言和其他类似的传统语言所写的程序必然给一组数据结构用来处理这些数据结构的例程，C 没有提供数据抽象的功能，故写的程序通常对哪些例程处理了存储数据不太清楚。C++ 语言正是为了弥补此方面的缺陷，提供对数据结构的封装和抽象及相关操作等功能使程序整体复杂性大大降低，且容易理解与修改。此外，此类设计会使程序的出错几率大大降低，为了实现此种功能，C++ 提出了一个新的概念：类和对象，及对其的操作。下面就这方面内容向读者做一介绍。

一、类的构成和对象

C++ 中，用关键字 Class 声明来构造一类，如下：

```
Class Point{
int Vx,Vy;
public:
void Setpt (int,int);
void offsetpt(int,int);
}p1,p2;
```

例中声明了 Point 类，其中有两个私有成员(Vx，Vy)。关键字 private 为缺省，外加两个公有成员 setpt()，offsetpt()，由关键字 public 提出，其中类的私有成员只能被类的函数成员访问，而公有成员可以被类的使用者访问。

另外，此类中还定义了两个类的对象 P1 和 P2。(Point P1 P2)对象作为类的一个实例，其调用类的函数成员的语句可写成 P1.setpt(10,20)；p2.offsetpt(2,2)；而 V_x，V_y 作为私有成员不能被类外使用者使用。如写成 P1.V_x = 10，则为错。

二、内置函数

为了节省类中函数调用的开销，对一些小而常用的操作可将函数定义为内置函数来解决。这将使该函数的代码被插入在函数的每个调用处，故可避免函数调用机制所带来的开销。在上一节 Point 类中，我们若将函数 setpt()定义为内置，语句如下：

```
inline void point::setpt(intx,inty)
{xval = x;
yval = y;
}
```

三、构造函数与析构函数

1. 构造函数

在C语言中,内部类型变量可以一次定义和初始化,C++也提供了相似的方法进行类的对象定义和初始化。这是通过一个称为构造函数成员来实现的,构造函数通过它和类本身同名来识别。如:

```
Class point{
int xval,yval;
public
point(int x,int y){xval=x;yval=y;}
void offsetpt(int,int){};
```

现在我们可以一次完成定义 point 类型的对象和初始化。如下所示:

```
point pt(10,20);
```

一般来说,一个类可有多个构造,其区分主要通过函数参数类型和函数个数来区别,如上,我们还可写出以下构造函数:

```
point(float x,float y);
```

其中可通过传递参数的类型的不同来区别所调用的构造函数。

2.析构函数

在运用一个类的过程中,不完全对其开辟空间,而这些空间应该在函数执行完后进行释放,有时程序员可以会忘记调用这个释放函数,C++中提供了一个析构函数,其在对象名超出定义域时释放该对象,析构函数也是与类具有相同的名字,只是在其前面加上一个"~"符号,如下所示:

```
Class string{
int len;
Char *s;
public
String (Char *Str);
String (int i);
~String(){delete s}
//定义了一个析构函数};
```

上类中定义了两个构造函数,一个析构函数,其用法读者通过注释就会明白,在此将不细述。

3.new 和 free

C++中提供了两个动态存储空间的分配和释放的运算符 new 和 delete,其用法与C语言

中 malloc 与 free 相似，代码如下：

```
Char * Str = malloc(20);
Char * Str = new Char[20];
```

以上两者相互等价，只是形式不同而已。为释放空间，在C++中使用 delete，如释放 Str 空间，C 和C++写法分别为 free(Str)和 delete str。

四、友元函数

由于类的私有成员不能被类外函数使用，为了使独立于类外的函数也有权使用类的私有成员，可通过将该函数定义为类的友元函数来实现，其可通过关键字“friend”来实现。如下所示：

```
Class point{
int Vx,Vy;
public
point(intx,inty);
void friend get (point p1);
//友元函数};
```

此类中将 get()函数定义为 point 类的友元函数，通过关键字 friend 实现，此后 get 函数可利用 xVal，yVal，也可将一个类定义为另一类的友元，使得该类中的函数能利用另一个类的私有成员。

1.友元类

为了增强两个类间的互访性，可将一个类定义为另一个类的友元类，即使两个类都可以使用对方的私有成员。构造如下：

```
Class A {
friend Class B {....}
;
或 Class B {
friend Class A{
.....
}
;
```

经过以上声明后，现在 Class B 和 Class A 中的函数均能调用另一个类的私有成员，从而增强了互访性能。这种设计的典型应用程序为数组类。此不细述。

五、重载

重载包括函数重载与运算符重载两种，是C++语言提供的重要特性。

1.函数重载

函数重载即在一个程序中声明多个名称相同的函数,这些函数可完成不同的功能,可带不同类型数目的参数及返回值,使用它可减轻用户的记忆负担,并使程序结构简单易懂。下面举例说明。

```
# include〈stdioch〉
# include〈conio.h〉
unsigned int square (uasigned Chari);
unsigned long square (unsigned int i);
double square (double i);
Void main()
{…
Csq = Square( .char C);
isq = Square (int i);
fsq = squar (float f);
…
}
```

上面的程序定义了函数 square()共有三个,但其带有的参数类型各有不同,当在主程序中调用此函数时,我们可根据其参数类型来区别调用的是哪一个 square()函数,另外也可根据其参数个数来区别。如下所示:

```
unsigned square (int i,in^ j);
unsigned square (int x);
```

读者可具体体会。

2.运算符重载

读者或许在利用 MFC 库进行程序设计的早期阶段还没有编写过重载运算符,但一定使用过 MFC 库提供的重载运算符。只有读者对此懂得较多,应用起来才会得心应手。

重载运算符之所以有用完全是由于它可以使得C++应用代码易写且易读,但也确定有一些程序员不太喜欢用它。只有当重载运算符的含义比较直观且自然时才可使用重载运算符,故尽管重载运算符只是用来代替成员函数调用的,但有时直接使用成员函数可能会更好些。

C++中可被重载的运算符如下:

+ , - ,/,%,^,| ~ ! ,= 〈〉 = 〉 = + + - - <= = ~ = || + = - + /+ % = ^= + = |= << = >> = → new delete

六、引用

引用与指针非常相似,它用作引入对象的别名,有两个用处:给函数传递大型的对象以及从函数中返回值。其使用运算符 & 实现。例如:

int &j = i,即 j 为 i 的一个引用,即为 i 的一个副本。其也可用于类的对象,如在 point 类,

我们可定义成 Point &(p+2) = p+1,即 p+2 为 p+1 的一个引用。

如果 pt1.offset(2,2)执行后则 p+1,p+2 都具有(2,2)的值。引用是引入对象的一个同义词,就像一个单一的对象可以一次拥有多个引用,为了避免大型对象函数间传递时的复制,实现重载运算符特别有用。另外返回引用值的函数调用可以被赋值,这可以被用作有价值的程序设计技巧。

1.引用参数 Const 型与非 const 型

为了判断程序员使用引用参数的真实目的(其一可能是要利用该参数在调用程序中改变变量的值,另一可以是为了避免在函数中调用大的对象),我们可用 Const 型参数和非 Const 型参数来实现如下:

```
Class XY{
Public:
double x,y;
XY();XY(double x,dowble y);
double get x()const return x}
double gety() const {return y};
```

如下一函数所示:

```
Void show (Const x &8+ xy){
printf ("x= %f,y= %f/n",xy.ge*x(),xy.gety()){};
```

从上面语句可看出,show 函数现在无法在调用程序内部改变 XY 对象值了,如有 xy.x = 500 则会使编译出错,调用 xy.get x 时也将如此,此时将 getx,gety 定义为 Const 类,则可调用,而非 Const 型参数则无此限制。

2.返回引用

函数返回的可以是一引用,即为一指针,这些引用既可以是 Const 型也可以是非 Const 型,下面给出上面 getx 函数的新版本,让其返回一引用,语句如下:

```
Const.Double &XY::get Const refx()const {return y};
```

这样定义后由于其返回引用为静态,则语句 xy.get const refx() = 3.0 将会出错,若不定义为 Const 型,则可在主函数中对引用返回值进行赋值,此即为返回引用的用途。引用的运用很广,读者可查询其他资料得到更多的知识。或者为新的类重新定义函数,如下所示:

```
Class A{
int Vx,Vy;
public:
A(int,int);
Void friend add (&a);
```

下面我们定义一个类 B,其为类 A 的派生。代码如下:

```
Class B:A;
int a,b;
public
B (int,int);
friend sub (B &b);
.
```

经过派生后B继承了A中的公有部分，而私有部分则不可访问，由于省略了派生类型，上面语句为私有派生即(private)。若为公有(public)派生，可写为：

```
Class B:public A;
.;
```

其中区别在于A中的公有成员在B中发生了变化，若为private派生则变为B中的私有成员；公有派生则保持原成员的性质。

七、类的派生

在C++ 中，具有很多公共内容的类并不少见。分别定义这些类将导致代码的大量复制以及在存储和开销方面上的浪费。克服这种困难的最自然方法是使语言能具有表达这种公共性的功能，这样就可以使公共部分只被书写一次，并可随后产生所需多种变体，C++中具有从现有类(称为基类 base Class)派生出一个新类的功能。程序员可具有对类派生的多种控制方法：新的类可在基类所提供的基础上创建新的成员；在新的类隐藏基类的任何函数成员。

1.虚函数

下面看两个类

```
Class List ;
Element * elems;
public
list();
wlist();
virtual Bool Insert (int);
Bool Contains (int);
.
Class Set:List;
int card;
public;
11
Bool Insert (int);
;
```

可以看出，Set类为List类的私有派生，两者中均有 Insert()函数，在此我们可以将 Insert()在基类中定义为虚函数(用关键字 virtual 实现)，它告诉编译程序，Insert 在从 list 派生出的类中可能有不同的实现。

八、多继承

一个派生类可以有多个基类，这就引出了本书主题——一对多继承，其会使一个类具有多个基类的属性，从而来集中地实现某个要求。定义形式如下：

```
Class A{…}
Class B{…};
Class C{…}
Class D:A,B{…};或 Class D:A public B{…};
Class F:B,C{…};Class F:public B public{…};
```

可看出两基类的逗号为区分，并且各自都可以拥有一继承限定符。

利用多继承的特性，使我们不必修改任何的数据就可以集成许多应用的数据结构。使得使用者更容易地进一步开发及扩充应用程序。

1.纯虚函数

Virtual Void Display(....) 定义了一个纯虚函数现在 Orbiter 基类已正式成为一个“抽象的基类”，并且编译器会强迫各个派生类(它们被用来创建具体的对象)来提供 Display 成员函数的具体实现过程。

九、静态类成员

如果需要知道所有在空间中沿轨道运行的物体的数目，该怎么办？我们当然可以定义一个全程变量，但那将与封装相违背。C++ 提供了静态数据成员和静态成员函数，如此以来和类相联系而并非和某一特定对象相联系。

1.静态数据成员

下面给出 Orbiter 类的说明，其中包含一个静态整型数据成员 nCount。

```
Class Orbiter{
protected:
double m-mass;
XY m_current,m_prior,m_thrust;
public:
Static int n_count;
Orbiter (XY &m_Current,XY &m_prior,double mass);
Const.XY &GePosition() Cont;
};
```

其中 XY 类声明如下：

```
Class XY{
Public
```

```
double x,y;
XY|x=0.0;y=0;|//default coustnctor
XY(double xary,double yary|//explicit constrnctor
·x=xary;y=xary;|
XY(Const xy &xy)| // copy coustrutor
x=xy.x;y=xy.y;|
Const XY & operator=(const XY & XY)|//assignment operator
x=xy,x;y=xy,y;
return this;|
;;
```

不管有多少个 Orbiter 对象存在，都将只存在一个 nCount 备份。上述的类说明虽然对类进行了定义，但它并没有为 ncount 留出存储空间，我们编写自己的和如下类的全程定义代码，并且在连接的过程中只能出现一次：

int Orbiter ::nCount=0;// initialization is optional

由于该变量被说明成 public 型，因此，我们在程序中可按如下方式使用它：Orbiter :: ncount + + ;

当然，如在 Orbiter 的成员函数内部对 nCount 进行访问的话。比如说在构造函数中，那么可以将 Orbiter 省去。

可以按下述方式来对其进行全程初始化：

Const int Orbiter.nMaxCount=256

2.枚举类型——实现静态数据成员的捷径

如果类需要有一个常量静态整型变量，我们可以采用一个简单的方法来避免使用单独的语言对其进行初始化，我们可按如下方式在类说明中放入一个 enum 语句：

enum |ninarCount=256|;

3.静态成员函数

如果静态 Orbiter 数据成员 Ncount 为 private 型的，就需要一个静态成员函数来对其进行访问。如果定义了一个 public 型 Count 函数，它返回的是一个对整型变量的引用，那么可以将该函数用在赋值语句的两边，语句如下所示：

```
public:
Static int & Count ()|return nCount|
```

下面给出的是使用新的 Count 函数的一些代码：

```
public:
Static int & Count |return nCount;|
//here'is Some Code that uses the new Count function;
int nOldCount=Orbiter::Count()++
int nNewCount=Orbiter::Count();
assert (nNewCount==nOld Count+1);
```

静态类成员函数的一个更有趣的应用是被用来构造对象。假设我们需要构造一个新的沿轨道运动的物体的对象，但直到运行时刻才会知道应该使用 Orbiter 类的哪个派生类，下面给出的静态构造函数使用了 Case 语句来选择对象类，代码如下所示：

```
Static Orbiter * Orbiter::MalceNaw (in: selection,
XY & Current,XY 8T Prior;double mass,XY&m_ thrust,double fuel;XY & Orientation.plant * #
planet)
{Switch (Selection)(
Case 0:
return (new planet (current,prior mass));
Case 1:
return (new Spaleship (current,prior, thrust,mass, fuel,orientation));
Case # :
return (new Moon (current prior,mass,pPlanet));
default: return Null;
}
}
```

现在我们给 Orbiter 指针类组赋值，语句如下所示：

```
Orbiter * pOrbiterArray [MAX]
XY Current i prior, thrust ovientation;
clouble mass, fuel;
planet * pPlanet;
for (int i=0,i CMAX;i+ + ){
pOrbiterArray[i] = Orbiter::MakeNew (i%),
current,prior,mass, theust, fual.orientation,pPlanet);
}.
```

经过以上的介绍，读者应能够对C++有一个基本的概念，同时也希望读者能够自己在业余的时间多看些相关资料以对本书有更好的应用。

无忧书库书籍版权申明

无忧书库提供的所有书籍、资料版权归原作者和出版商所有。

本站提供下载的书籍仅供个人学习、参考之用，任何集体、个人不得用于商业用途。

请您在下载书籍 **24** 小时之后删除书籍，购买正版书籍！

本站书籍如有侵犯您的版权，请与我们联系，我们将尽快删除。联系信箱：**pcbook@51soft.com**。

无忧书库
http://pcbook.51soft.com
2000 年 **4** 月 **13** 日